

# I.GIỚI THIỆU

## 1.LÝ DO CHỌN ĐỀ TÀI

Trong thời đại chuyển đổi số, các hệ thống phần mềm đang ngày càng trở thành một phần thiết yếu trong mọi lĩnh vực, từ tài chính, y tế, giáo dục đến thương mại điện tử. Yêu cầu từ phía người dùng không chỉ dừng lại ở việc cung cấp tính năng mà còn bao gồm:

- Tốc độ phát triển: Các sản phẩm mới cần được ra mắt nhanh chóng để đáp ứng nhu cầu thị trường.
- Chất lượng cao: Giảm thiểu lỗi và cải thiện trải nghiệm người dùng.
- Khả năng duy trì: Hệ thống phải hoạt động ổn định, kể cả trong điều kiện tải lớn hoặc gấp sự cố bất ngờ.

Theo một khảo sát của State of DevOps Report, các tổ chức áp dụng CD/CI thường đạt được:

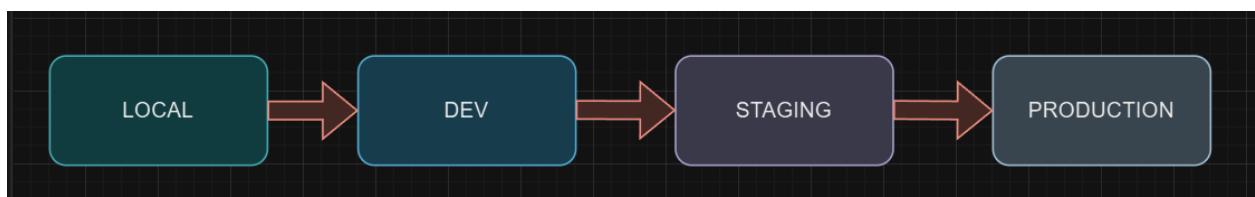
- Thời gian triển khai nhanh hơn 208 lần so với phương pháp truyền thống.  
[\(Announcing the 2023 State of DevOps Report | Google Cloud Blog\)](#)
- Tỷ lệ thất bại triển khai thấp hơn 7 lần.

Trong bối cảnh đó, các phương pháp truyền thống như kiểm thử và triển khai thủ công không còn phù hợp, do:

- Dễ mắc lỗi khi xử lý các thay đổi lớn trong mã nguồn.
- Mất nhiều thời gian để phát hiện và khắc phục lỗi trong hệ thống.
- Khó kiểm soát và đảm bảo tính ổn định của ứng dụng khi phát hành.

Ví dụ minh họa: Trong một hệ thống lớn triển khai dịch vụ web hoặc ứng dụng, thường có nhiều môi trường như:

- Local (phát triển): Lập trình viên kiểm tra tính năng ban đầu.
- Dev (tích hợp): Nhóm phát triển tích hợp mã nguồn.
- Staging (kiểm thử): Kiểm tra toàn bộ hệ thống.
- Production (sản xuất): Người dùng thực tế sử dụng.



Hình 1. Các môi trường trong triển khai phần mềm, ứng dụng

=> Với phương pháp truyền thống, việc triển khai qua từng môi trường phải làm thủ công, gây tốn thời gian, dễ mắc lỗi thao tác và tăng chi phí nhân lực. Khi áp dụng CI/CD, quy trình này được tự động hóa, mã nguồn được kiểm tra, tích hợp

và triển khai đồng nhất qua các môi trường mà không cần can thiệp thủ công. Điều này giúp:

- Giảm thời gian triển khai từ vài ngày xuống vài giờ.
- Tối ưu hóa chi phí và giảm lỗi thao tác.
- Đảm bảo chất lượng và tính ổn định của hệ thống.

Bên cạnh đó, khi các hệ thống lớn có số lượng người dùng cao, tính ổn định trở thành yêu cầu cấp thiết. Monitoring đóng vai trò quan trọng trong việc:

- Theo dõi tình trạng và hiệu suất hệ thống theo thời gian thực.
- Hỗ trợ phát hiện sớm các vấn đề và gửi cảnh báo khi có sự cố.
- Đảm bảo hệ thống luôn hoạt động ổn định, đáp ứng yêu cầu của người dùng.

## 2. MỤC TIÊU NGHIÊN CỨU

### 2.1. Mục tiêu lý thuyết

Hiểu rõ các khái niệm liên quan:

- Tìm hiểu khái niệm về CI (Continuous Integration (Tích hợp liên tục)), CD (Continuous Deployment/Delivery (Triển khai/ vận chuyển liên tục)) và Monitoring (Giám sát), cũng như vai trò của chúng trong quy trình phát triển phần mềm.
- Phân tích các đặc điểm và yêu cầu của DevOps pipeline (quy trình DevOps).

Lợi ích và hạn chế của CD/CI và Monitoring:

- Xác định những lợi ích rõ ràng khi áp dụng, như cải thiện năng suất, giảm lỗi phần mềm, tăng tốc độ triển khai.
- Đánh giá những thách thức và hạn chế, chẳng hạn như chi phí triển khai hoặc khó khăn khi tích hợp các công cụ khác nhau.

Tổng quan về công cụ:

- CI/CD: Jenkins, GitHub Actions, GitLab CI/CD.
- Monitoring: Prometheus, Grafana.
- Các công cụ khác sử dụng trong quá trình triển khai: GitHub, GitLab, DockerHub, ...

Phân tích chi tiết các công cụ phổ biến hiện nay, so sánh tính năng, ưu nhược điểm, và trường hợp sử dụng thực tế của các công cụ này.

### 2.2. Mục tiêu thực nghiệm

Xây dựng hệ thống mẫu:

- Thiết kế pipeline CI/CD từ giai đoạn commit mã nguồn, kiểm thử tự động, đến triển khai tự động trên môi trường staging hoặc production.
- Tích hợp hệ thống monitoring để giám sát hiệu suất và phát hiện lỗi trong thời gian thực.
- Tích hợp các công cụ:
  - + Sử dụng lần lượt các công cụ GitHub Action, GitLab CI/CD, Jenkins để tự động hóa quy trình CI/CD.
  - + Triển khai Prometheus và Grafana để giám sát hiệu năng ứng dụng mẫu (CPU, RAM, throughput).

Ứng dụng thực tế:

- Xây dựng một ứng dụng mẫu (REST API hoặc ứng dụng web).
- Thử nghiệm với các tình huống thực tế: Deploy (Triển khai) thành công và thất bại, giả lập lỗi hệ thống để kiểm tra khả năng phát hiện và cảnh báo từ monitoring.

### 2.3. Mục tiêu đánh giá

Đo lường hiệu quả, sử dụng các chỉ số:

- Thời gian triển khai: Từ lúc commit đến khi ứng dụng được deploy.
- Số lỗi phát hiện trước khi đến production: Đánh giá hiệu quả của CI/CD pipeline (quy trình CI/CD).
- Thời gian phát hiện và khắc phục lỗi hệ thống: Đo lường hiệu suất của monitoring (giám sát).

So sánh và phân tích:

- So sánh hiệu quả của pipeline (quy trình) khi có và không có monitoring (giám sát) tích hợp.
- Phân tích mức độ phù hợp của từng công cụ được sử dụng.

Bài học kinh nghiệm:

- Rút ra bài học về cách thiết kế và vận hành pipeline (quy trình) hiệu quả.
- Đề xuất những cải tiến hoặc giải pháp thay thế phù hợp hơn trong tương lai.

## 3. PHẠM VI NGHIÊN CỨU

### 3.1. Phạm vi lý thuyết

Khái niệm cơ bản:

- Nghiên cứu các định nghĩa và vai trò của Continuous Integration (Tích hợp liên tục), Continuous Deployment/Delivery (Triển khai/ Vận chuyển liên tục), và Monitoring (Giám sát) trong phát triển phần mềm.

- Mô tả các thành phần của một pipeline CI/CD, từ commit mã nguồn, kiểm thử tự động, đến triển khai.
- Hiểu các khái niệm liên quan đến monitoring như:
  - + Observability (tính quan sát được): Metrics, logs, traces.
  - + Alerting (cảnh báo): Thiết lập cảnh báo khi hệ thống có lỗi.

Lợi ích và thách thức, phân tích lợi ích của việc tích hợp CI/CD và monitoring:

- Tăng tốc độ phát triển, giảm lỗi, đảm bảo độ ổn định của hệ thống.

Các thách thức thường gặp:

- Đồng bộ hóa giữa đội ngũ phát triển và vận hành.
- Chi phí triển khai hệ thống phức tạp.
- Độ khó trong việc thiết kế pipeline cho các hệ thống lớn.

### 3.2. Phạm vi công cụ

Continuous Integration/Continuous Deployment:

- Jenkins: Công cụ mã nguồn mở, hỗ trợ linh hoạt trong việc tích hợp và triển khai liên tục.
- GitHub Actions: Giải pháp tích hợp CI/CD trực tiếp trong GitHub, dễ sử dụng, phù hợp với các dự án nhỏ đến trung bình.
- GitLab CI/CD: Giải pháp tích hợp CI/CD trực tiếp trong GitLab, dễ sử dụng, phù hợp với các dự án trung bình đến lớn.

Monitoring:

- Prometheus: Công cụ thu thập và lưu trữ metrics (CPU, bộ nhớ, thời gian phản hồi,...) với ngôn ngữ truy vấn PromQL.
- Grafana: Nền tảng trực quan hóa dữ liệu, tạo bảng biểu và dashboard từ Prometheus hoặc các nguồn khác.

Hệ sinh thái phụ trợ:

- Docker: Được sử dụng để đóng gói và triển khai các thành phần hệ thống trong môi trường container hóa. Giúp đảm bảo tính đồng nhất giữa các môi trường (development, staging, production). Tăng cường tính linh hoạt và khả năng mở rộng của hệ thống.
- Git: Công cụ quản lý mã nguồn phân tán, cho phép lưu trữ mã nguồn trên môi trường cục bộ. Hỗ trợ quản lý phiên bản, hợp nhất mã nguồn và theo dõi lịch sử thay đổi.
- GitHub: Nền tảng lưu trữ mã nguồn trên môi trường server, tích hợp với Git để quản lý phiên bản. Hỗ trợ các tính năng CI/CD thông qua GitHub Actions. Cho phép cộng tác và quản lý dự án dễ dàng trong nhóm phát triển.

- Docker Hub: Nền tảng lưu trữ container image phổ biến, giúp chia sẻ và phân phối các image Docker. Cho phép các nhóm phát triển dễ dàng tải lên, lưu trữ và truy xuất các image.  
Tích hợp tốt với Docker CLI để quản lý image.

### 3.3. Phạm vi thực nghiệm

Ứng dụng mẫu:

- REST API hoặc một ứng dụng api nhỏ (Node.js).
- Chức năng cơ bản: API xử lý CRUD hoặc hiển thị giao diện đơn giản.
- Quy trình pipeline:

Bước 1: Tích hợp kiểm thử mã nguồn tự động (Unit Test, Integration Test).

Bước 2: Triển khai ứng dụng mẫu lên môi trường staging hoặc production (sử dụng Docker).

Bước 3: Triển khai các công cụ monitoring để theo dõi tài nguyên và hiệu suất ứng dụng.

Mô phỏng tình huống thực tế:

- Deploy lỗi: Kiểm tra pipeline tự động rollback.
- Tăng tải hệ thống: Theo dõi CPU, RAM, và phản hồi từ monitoring.
- Sự cố: Phát hiện lỗi qua cảnh báo từ Prometheus/Grafana.
- Thu thập dữ liệu đánh giá: Tổng hợp logs, metrics từ hệ thống monitoring. Đo lường thời gian triển khai, phát hiện và khắc phục lỗi.

Tích hợp thực tiễn:

- Kết hợp các công cụ đã nghiên cứu để xây dựng một hệ thống CI/CD pipeline hoàn chỉnh.
- Cung cấp tài liệu chi tiết cho việc tái sử dụng và cải thiện pipeline.

Giới hạn lý thuyết:

- Tập trung vào CD/CI và monitoring trong bối cảnh DevOps, không đi sâu vào các lĩnh vực liên quan như bảo mật (DevSecOps) hay AI-driven monitoring.

Giới hạn công cụ:

- Chỉ sử dụng các công cụ mã nguồn mở hoặc có phiên bản miễn phí (Jenkins, GitHub Actions, GitLab CI/CD, Prometheus, Grafana).

Giới hạn ứng dụng mẫu:

- Ứng dụng thử nghiệm sẽ có độ phức tạp trung bình, đủ để minh họa các khái niệm mà không tập trung quá nhiều vào logic nghiệp vụ.

## II. TỔNG QUAN LÝ THUYẾT

### 1. Khái niệm về CD/CI

Continuous Integration (CI) và Continuous Delivery/Continuous Deployment (CD) là hai khái niệm cốt lõi trong phát triển phần mềm hiện đại.

Continuous Integration (CI): Tích hợp liên tục

- Là quá trình tự động tích hợp mã nguồn của các lập trình viên vào một nhánh chung (thường là nhánh chính - main branch).

Quy trình CI bao gồm:

- Kiểm tra mã nguồn (linting, kiểm tra cú pháp).
- Chạy các bộ kiểm thử tự động (unit test, integration test).
- Tạo bản dựng (build) nếu không phát sinh lỗi.
- Mục tiêu: Phát hiện lỗi sớm, giảm thời gian tích hợp và đảm bảo mã nguồn luôn ở trạng thái ổn định.

Continuous Delivery (CD): Vận chuyển liên tục

- Là quá trình tự động hóa việc đóng gói, kiểm thử, và chuẩn bị triển khai ứng dụng sau giai đoạn CI.

Mục tiêu:

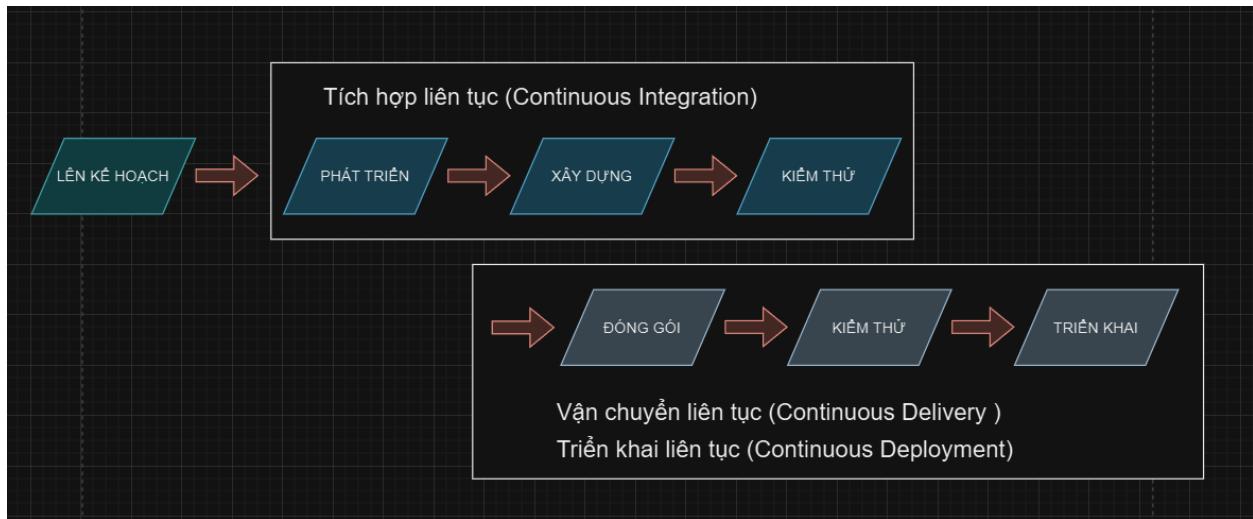
- Rút ngắn thời gian đưa sản phẩm ra thị trường.

Continuous Deployment (CD): Triển khai liên tục

- Là bước tiếp theo của Continuous Delivery, trong đó ứng dụng được triển khai tự động vào môi trường production mà không cần sự can thiệp thủ công.

Mục tiêu:

- Tăng tốc độ phát hành ứng dụng và cải thiện trải nghiệm người dùng.



Hình 2. Quy trình CI/CD

## 2. Monitoring trong phát triển phần mềm

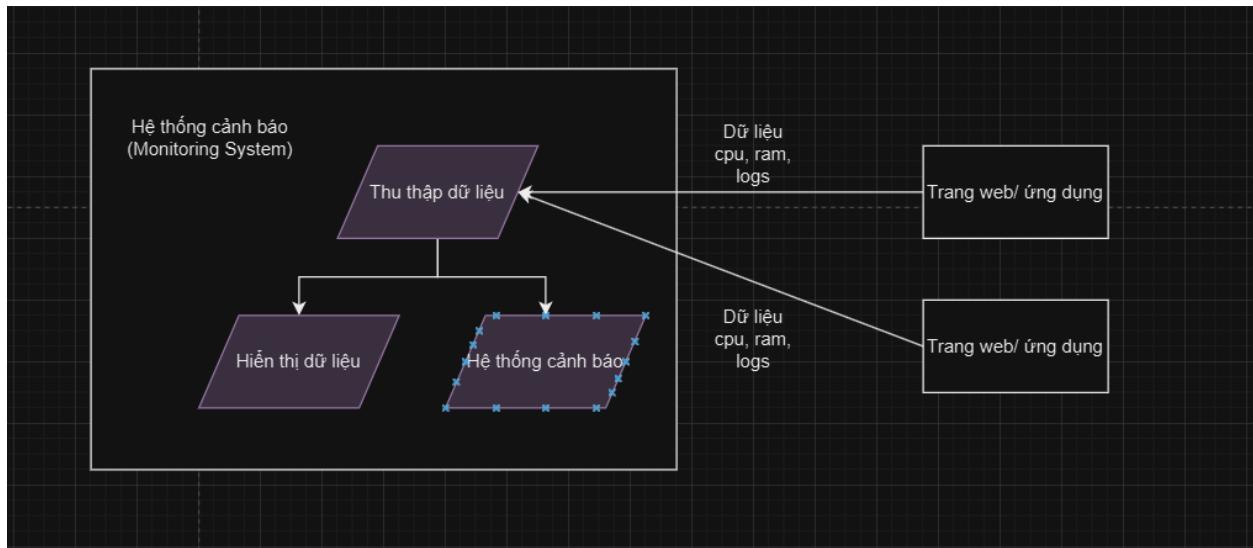
Monitoring (giám sát hệ thống) là quá trình thu thập, phân tích, và hiển thị dữ liệu hiệu suất, hoạt động của hệ thống phần mềm theo thời gian thực.

Mục tiêu:

- Đảm bảo hệ thống hoạt động ổn định.
- Phát hiện các vấn đề sớm và giảm thời gian khắc phục sự cố (MTTR - Mean Time to Resolve).
- Tối ưu hóa hiệu suất của hệ thống.

Thành phần chính của Monitoring:

- Thu thập dữ liệu (Data Collection): Thu thập số liệu từ các nguồn như CPU, RAM, lưu lượng mạng, logs ứng dụng.
- Lưu trữ dữ liệu (Data Storage): Lưu trữ dữ liệu hiệu suất để phân tích lâu dài.
- Hiển thị dữ liệu (Visualization): Cung cấp dashboard để theo dõi dữ liệu trực quan.
- Cảnh báo (Alerting): Gửi thông báo qua email, Slack, hoặc SMS khi có sự cố xảy ra.



Hình 3. Hệ thống giám sát

### 3. Tổng quan về các công cụ phụ trợ cho CI/CD và Monitoring

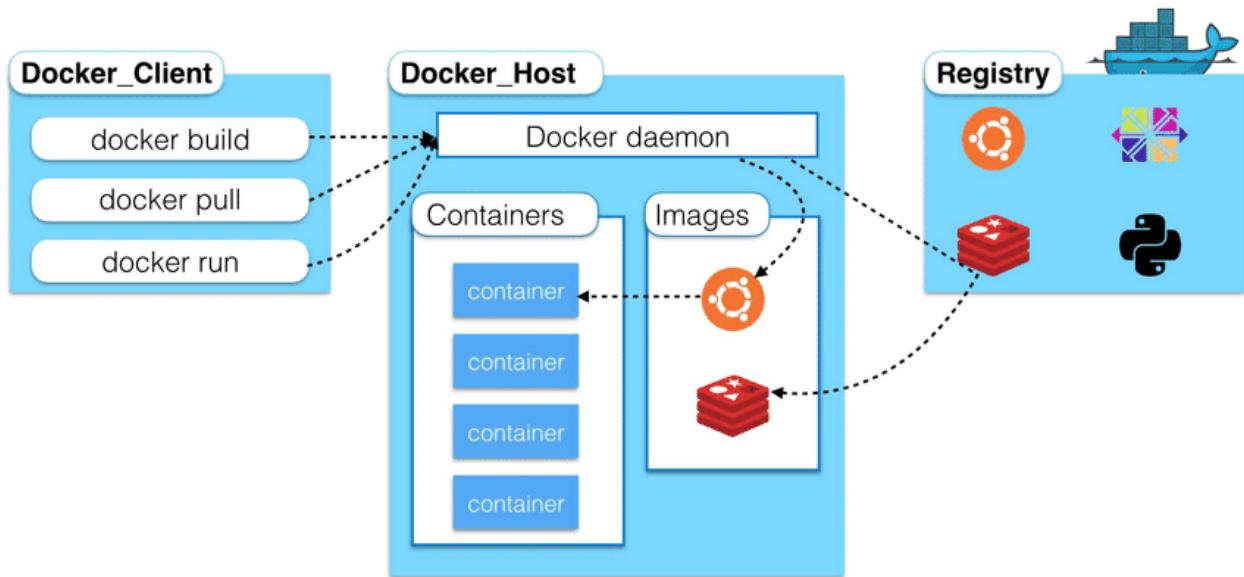
#### 3.1 Docker

Docker là một nền tảng container (nơi chứa) hóa mã nguồn mở, cung cấp giải pháp đóng gói ứng dụng và các phụ thuộc của nó thành một đơn vị duy nhất gọi là container. Mục tiêu chính của Docker là đảm bảo tính nhất quán giữa các môi trường phát triển, thử nghiệm và sản xuất. Điều này giúp giảm thiểu lỗi xảy ra do sự khác biệt giữa các môi trường.

Docker bao gồm các thành phần chính:

- Docker Client: Giao diện dòng lệnh (CLI) để tương tác với Docker.
- Docker Host:
  - + Docker Engine: Thành phần cốt lõi giúp xây dựng và quản lý container.
  - + Images: Các mẫu template dùng để tạo container.
  - + Containers: Phiên bản thực thi của một image, hoạt động như một môi trường ứng dụng.
  - + Storage: Lưu trữ dữ liệu cho container (persistent storage).
  - + Docker Daemon (dockerd): Quản lý tất cả các container, images và kết nối với Docker Client.

- Docker Registry: Kho lưu trữ Docker Images, ví dụ: Docker Hub (public) hoặc các registry riêng (như Harbor).



Hình 4. Kiến trúc Docker

Docker trong hệ thống CI/CD: Docker đóng vai trò là một thành phần cốt lõi trong quy trình CI/CD, giúp chuẩn hóa môi trường triển khai và tự động hóa quá trình từ xây dựng, kiểm thử đến triển khai ứng dụng. Dưới đây là các bước cụ thể:

- Build:

- + Mục đích: Tạo Docker Image chứa mã nguồn và các phụ thuộc cần thiết để ứng dụng hoạt động.

- + Quy trình:

- B1: Mã nguồn được commit/push lên hệ thống quản lý phiên bản (GitHub/GitLab).

- B2: CI pipeline được kích hoạt để kiểm tra mã nguồn (code linting, static analysis).

- B3: Docker sử dụng Dockerfile để tạo Docker Image:

- Test:

- + Mục đích: Đảm bảo mã nguồn không bị lỗi trước khi triển khai.

- + Quy trình:

B1: CI pipeline khởi tạo một container từ image đã build.

B2: Container được sử dụng để thực hiện các kiểm thử tự động.

- Deploy:

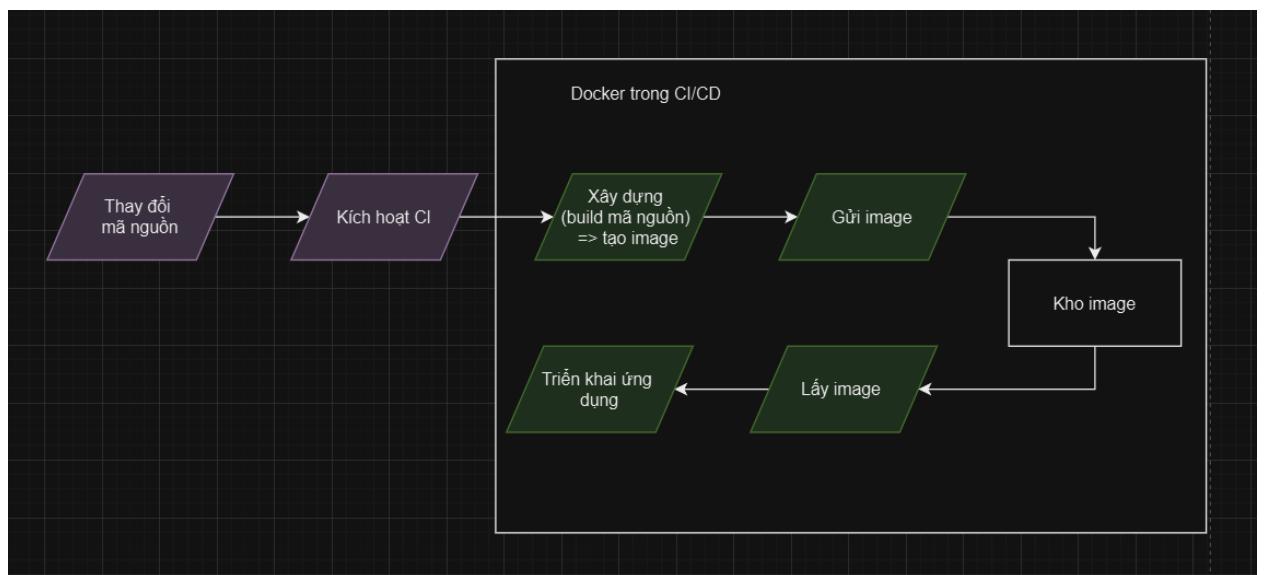
- + Mục đích: Tự động triển khai ứng dụng lên các môi trường khác nhau (staging, production).

- + Quy trình:

B1: Docker Image sau khi kiểm thử thành công được đẩy lên Docker Registry (Docker Hub).

B2: CD pipeline kéo image từ registry và triển khai container lên môi trường đích.

B3: Các môi trường (staging, production) được cấu hình đồng nhất, đảm bảo container chạy giống hệt nhau.



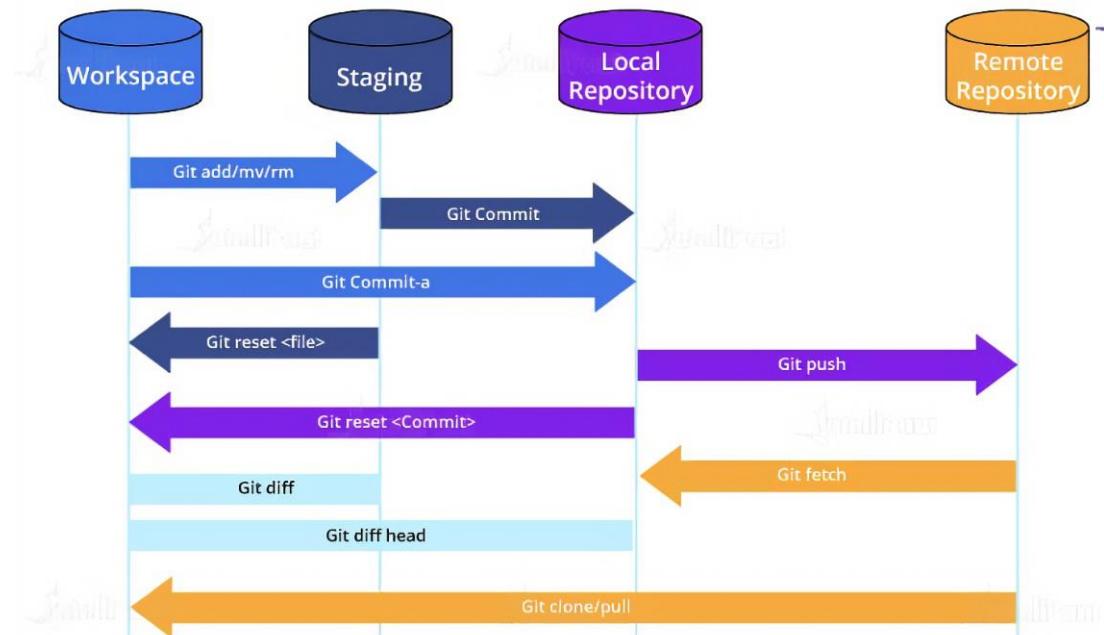
Hình 5. Docker trong quy trình CI/CD

## 3.2 Git

Git là một hệ thống quản lý phiên bản phân tán (Distributed Version Control System) được phát triển bởi Linus Torvalds vào năm 2005. Công cụ này được thiết kế để giúp theo dõi lịch sử thay đổi của mã nguồn, hỗ trợ cộng tác nhóm và duy trì chất lượng dự án một cách hiệu quả.

## Cấu trúc và thành phần chính của Git:

- Kho lưu trữ (Local Repository): Lưu trữ toàn bộ lịch sử thay đổi của dự án, bao gồm commit, branch, và tag.
- Commit: Lưu lại một thay đổi cụ thể trong mã nguồn. Mỗi commit có một mã hash duy nhất.
- Branch: Tạo nhánh để phát triển tính năng mới hoặc sửa lỗi mà không ảnh hưởng đến nhánh chính (main/master).
- Merge: Kết hợp các thay đổi từ một nhánh vào nhánh khác.
- Remote Repository: Kho lưu trữ trên server, ví dụ: GitHub, GitLab, hoặc Bitbucket, để cộng tác và chia sẻ mã nguồn.



Hình 6. Kiến trúc Git

Git trong hệ thống CI/CD: Git đóng vai trò quan trọng trong CI/CD vì nó quản lý mã nguồn, kích hoạt pipeline, và hỗ trợ làm việc nhóm hiệu quả:

- Build:
  - + Mục đích: Đảm bảo mã nguồn hợp lệ và sẵn sàng để chuyển sang các bước tiếp theo.
  - + Quy trình:

B1: Mã nguồn được commit/push/merge/.... lên Git repository.

B2: Git thông báo (webhook, ...) cho hệ thống CI/CD để khởi động pipeline.

B3: Hệ thống kiểm tra mã nguồn thông qua các bước như code linting hoặc static analysis.

- Deploy:

- + Mục đích: Tự động triển khai ứng dụng lên các môi trường khác nhau như staging hoặc production, sử dụng GitHub Action

- + Quy trình:

B1: Mã nguồn đã kiểm thử thành công được triển khai từ nhánh cụ thể.

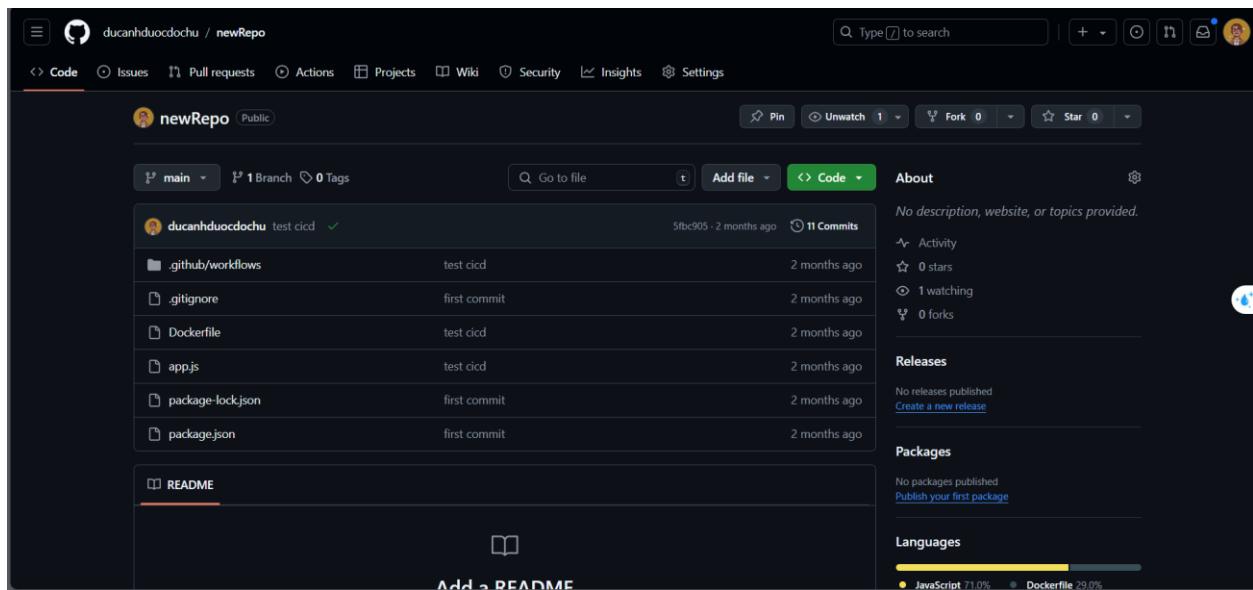
B2: Hệ thống CI/CD thực hiện triển khai tự động thông qua script.

### 3.3 GitHub

GitHub là một nền tảng lưu trữ và quản lý mã nguồn dựa trên Git, được sử dụng để theo dõi lịch sử phát triển của phần mềm và hỗ trợ làm việc nhóm. Được ra mắt vào năm 2008, GitHub đã trở thành một trong những công cụ quan trọng nhất trong ngành công nghệ phần mềm, với hàng triệu lập trình viên và tổ chức trên toàn thế giới sử dụng.

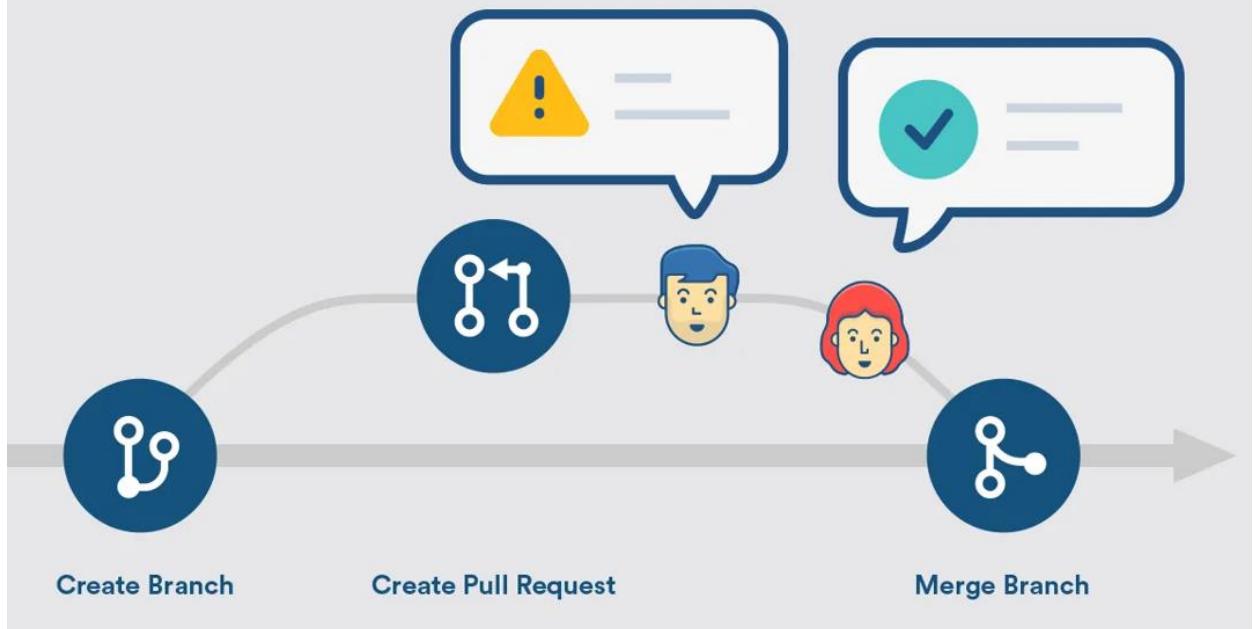
Các tính năng của Github:

- Quản lý mã nguồn: GitHub cho phép người dùng lưu trữ, truy xuất và quản lý phiên bản mã nguồn bằng Git. Mọi thay đổi trong mã nguồn được ghi lại, giúp theo dõi lịch sử phát triển và phục hồi khi cần thiết.



Hình 7. Giao diện quản lý mã nguồn GitHub

- Cộng tác nhóm:
- + Pull Requests: Tính năng này giúp các thành viên gửi thay đổi mã nguồn để đánh giá và hợp nhất vào dự án chính.
- + Issues: Theo dõi các vấn đề, tính năng mới, hoặc nhiệm vụ cần thực hiện.



Hình 8. Làm việc nhóm với GitHub, pull request

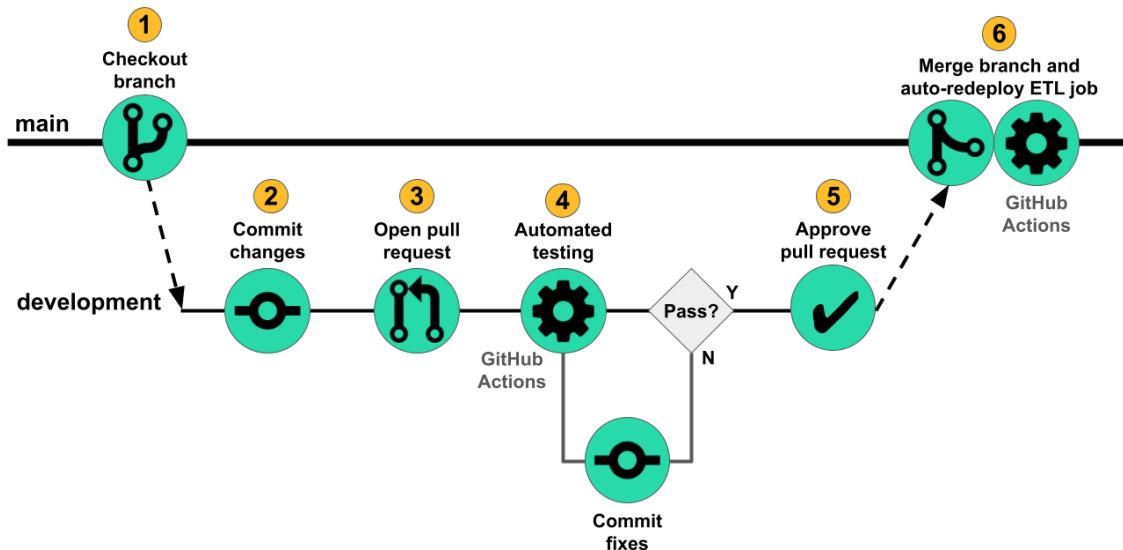
- + Lập trình viên phát triển một chức năng mới, tạo nhánh mới để phát triển code.

+ Lập trình viên đưa mã nguồn mới nhất được phát triển lên GitHub qua Git, sử dụng Push. Tạo Pull Request, người quản lý dự án sẽ kiểm tra Pull Request, kiểm tra và đưa mã nguồn mới nhất từ nhánh của lập trình viên vào nhánh chính qua Merge.

- Tích hợp công cụ CI/CD:

+ GitHub Actions cung cấp khả năng tự động hóa quy trình tích hợp liên tục (CI) và triển khai liên tục (CD), giúp giảm thiểu công việc thủ công và tăng hiệu quả.

+ Kết hợp với Jenkins cung cấp khả năng tự động hóa quy trình tích hợp liên tục (CI) và triển khai liên tục (CD).



Hình 9. Tích hợp GitHub với GitHub Action

- Quản lý dự án:

+ Hỗ trợ lập kế hoạch và theo dõi tiến độ dự án thông qua GitHub Projects (dựa trên bảng Kanban).

+ Cho phép tích hợp với các công cụ quản lý như Jira, Trello.

- Lưu trữ mã nguồn mở:

GitHub là nền tảng hàng đầu để chia sẻ mã nguồn mở, nơi các dự án công khai có thể nhận được sự đóng góp từ cộng đồng toàn cầu.

### 3.4 Gitlab

GitLab là một nền tảng DevOps tích hợp toàn diện cung cấp các công cụ cho quản lý mã nguồn, kiểm thử, tích hợp liên tục (CI), triển khai liên tục (CD), và giám sát. Với khả năng kết hợp các quy trình từ phát triển đến triển khai trong một hệ thống duy nhất, GitLab trở thành lựa chọn phổ biến trong các dự án phần mềm hiện đại.

Các thành phần chính của GitLab:

- Quản lý mã nguồn (Source Code Management - SCM):
  - + Hỗ trợ Git để lưu trữ và quản lý mã nguồn.
  - + Các tính năng chính: Pull Request (Merge Request), Code Review, và Branch Management.
  - + Tích hợp quyền kiểm soát chặt chẽ thông qua chính sách nhánh (Branch Policies).

Name	Last commit	Last update
src	feat(project): create base project	2 weeks ago
.gitlab-ci.yml	Update .gitlab-ci.yml	2 weeks ago

Hình 10. Giao diện quản lý mã nguồn của GitLab

- Tích hợp liên tục (Continuous Integration - CI):
  - + GitLab CI/CD (Jenkins) tích hợp sẵn trong hệ thống, không cần công cụ bên thứ ba.
  - + Sử dụng GitLab Runner để thực thi các pipeline CI trên các môi trường khác nhau.
- Triển khai liên tục (Continuous Deployment - CD):

- + Tự động hóa quá trình triển khai lên các môi trường staging và production.
- + Hỗ trợ triển khai trên Docker, Kubernetes, hoặc các nền tảng cloud như AWS, Azure, GCP.
  - DevOps Analytics:
- + Cung cấp bảng điều khiển (Dashboard) hiển thị hiệu suất, trạng thái pipeline, và báo cáo chi tiết.
- + Giúp các nhóm phát triển theo dõi và cải thiện quy trình DevOps.

The screenshot shows the GitLab interface for the 'shoeshop' project. The left sidebar is open, showing various project management sections like Project information, Repository, Issues, Merge requests, CI/CD, Pipelines, Security & Compliance, Deployments, Monitor, Infrastructure, Packages & Registries, and Analytics. The Pipelines section is currently selected. The main area displays the 'Pipelines' page for the 'shoeshop' project. It shows a table with the following data:

Status	Pipeline ID	Triggerer	Commit	Stages	Duration	Action Buttons
failed	#11 latest	git	develop -> c49668bc Update Jenkinsfile	✖	1 week ago	<button>C</button> <button>⋮</button>
failed	#10	git	develop -> 14f73d3a Update Jenkinsfile	✖	1 week ago	<button>C</button> <button>⋮</button>
failed	#9	git	develop -> 0fa3bc31 Update README.md	✖	00:00:01 1 week ago	<button>C</button> <button>⋮</button>
failed	#8	git	develop -> 6262e9fb Update Jenkinsfile	✖	00:00:01 1 week ago	<button>C</button> <button>⋮</button>
failed	#7	git	develop -> 174c3da5 Update Jenkinsfile 1	✖	00:00:01 1 week ago	<button>C</button> <button>⋮</button>
			development -> 41529975	-	00:00:00	<button>C</button> <button>⋮</button>

Hình 11. Màn hình hiển thị trạng thái pipeline

- Quản lý dự án:
  - + Issue Tracking: Tích hợp hệ thống quản lý nhiệm vụ (tasks) và lỗi (bugs).
  - + Roadmap: Lập kế hoạch và theo dõi tiến độ dự án.
  - + Wiki: Tài liệu hóa trực tiếp trên GitLab.
- Bảo mật và giám sát:
  - + Code Scanning: Phát hiện các lỗ hổng bảo mật trong mã nguồn.
  - + Secret Detection: Kiểm tra và cảnh báo khi phát hiện thông tin nhạy cảm trong repository.
  - + Giám sát hiệu suất: Tích hợp với Prometheus để theo dõi hệ thống và ứng dụng.

GitLab trong quy trình CI/CD:

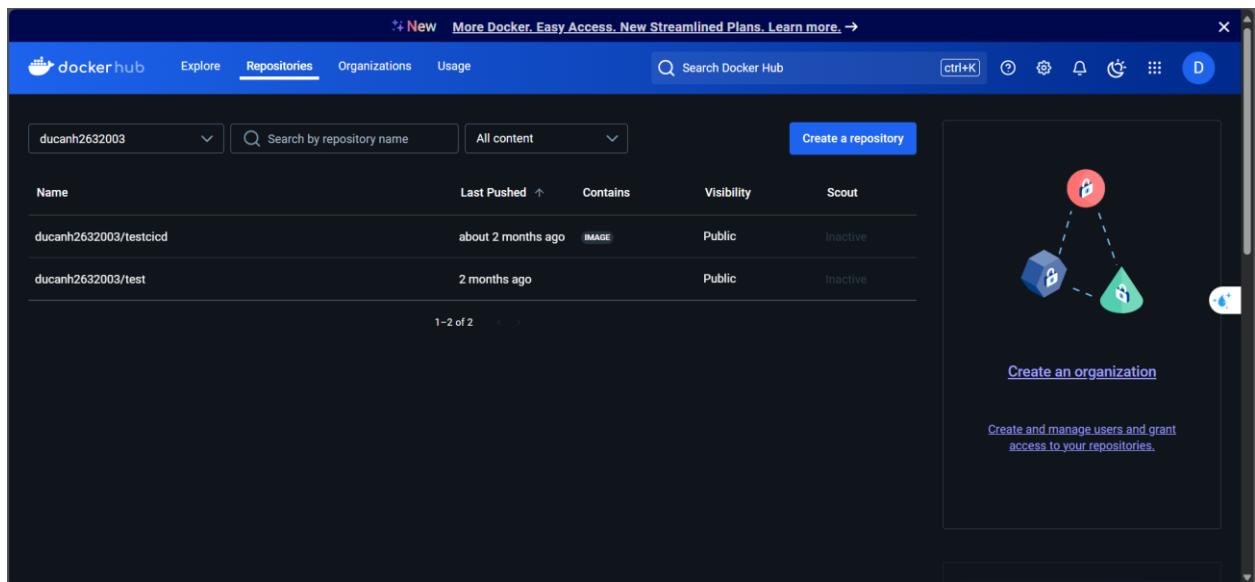
- GitLab quản lý mã nguồn, kiểm tra các trạng thái push, merge,... Kích hoạt CI/CD pipeline
- GitLab cung cấp công cụ GitLab Runner giúp triển khai CICD pipeline qua script
- GitLab có thể phối hợp với Jenkins để thực thi CICD pipeline trên Jenkins server.

### 3.5 DockerHub

DockerHub là một nền tảng lưu trữ Docker Image phổ biến nhất, cung cấp cả registry public và private. Đây là nơi lưu trữ, chia sẻ và phân phối Docker Image, giúp triển khai container dễ dàng trên các hệ thống khác nhau.

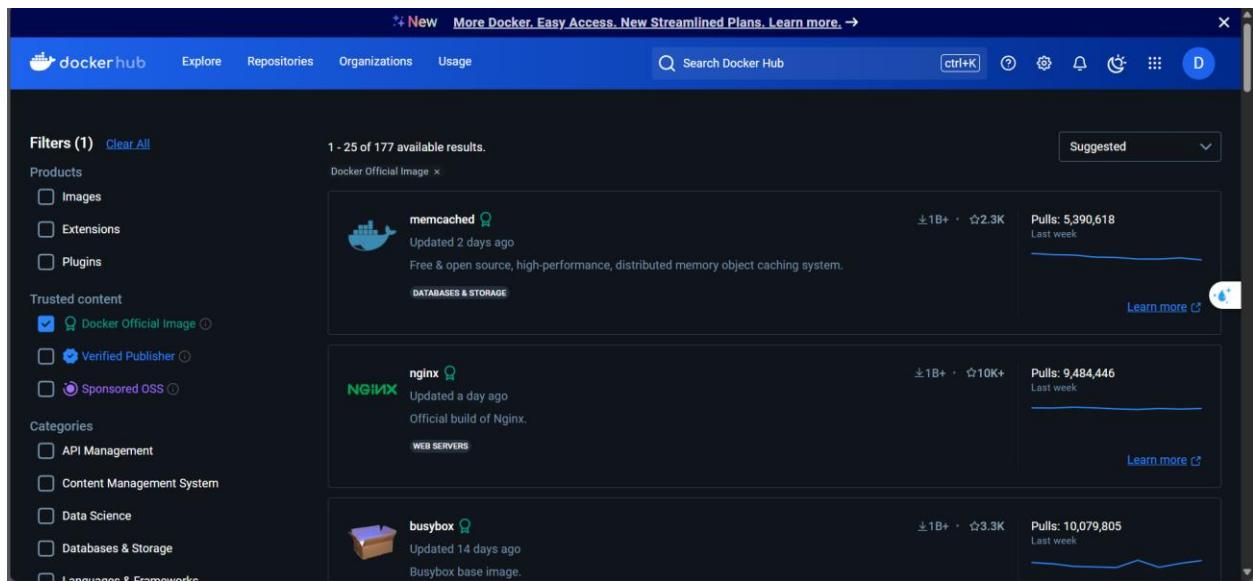
#### Chức năng chính của DockerHub

- Public Repository: Lưu trữ và chia sẻ Docker Image miễn phí.
- Private Repository: Lưu trữ Docker Image riêng tư với kiểm soát quyền truy cập.



Hình 12. Giao diện quản lý image trên DockerHub

- Docker Official Images: Cung cấp các image chính thức từ nhà phát triển (như Node.js, Nginx, Ubuntu).



Hình 13. Giao diện tìm kiếm các image từ các nhà phát triển

- Webhooks: Kích hoạt tự động các pipeline CI/CD khi image được cập nhật.

DockerHub trong quy trình CICD:

- Quản lý, phân phối các Docker Images, quản lý phiên bản của Docker Images
- Sau khi xây dựng, đưa Docker Images lên DockerHub để lưu trữ phiên bản.
- Sử dụng webhook, khi Docker Images thay đổi trên DockerHub, lấy Docker Images về server để triển khai thành các container.

## 4. Tổng quan về các công cụ CI/CD

Các công cụ CI/CD đóng vai trò quan trọng trong việc tự động hóa quy trình kiểm thử, tích hợp và triển khai phần mềm. Dưới đây là tổng quan về ba công cụ phổ biến: GitHub Actions, GitLab CI/CD, và Jenkins.

### 4.1 GitHub Actions

GitHub Actions là một nền tảng tự động hóa workflow được tích hợp trực tiếp vào GitHub, hỗ trợ thiết lập các quy trình CI/CD ngay trong repository mã nguồn.

**Đặc điểm chính:**

- Tích hợp trực tiếp: Không cần cài đặt hoặc tích hợp thêm công cụ bên ngoài.
- Workflow dựa trên YAML: Các quy trình được định nghĩa trong file .github/workflows/\*.yml.

- Event-driven: Kích hoạt tự động dựa trên các sự kiện như push, pull\_request, hoặc schedule.

Các thành phần chính:

- Workflow: Quy trình tự động gồm nhiều job và step.
- Job: Một tập hợp các step được thực hiện tuần tự.
- Runner: Môi trường thực thi workflow (có thể là hosted hoặc self-hosted).

Ưu điểm:

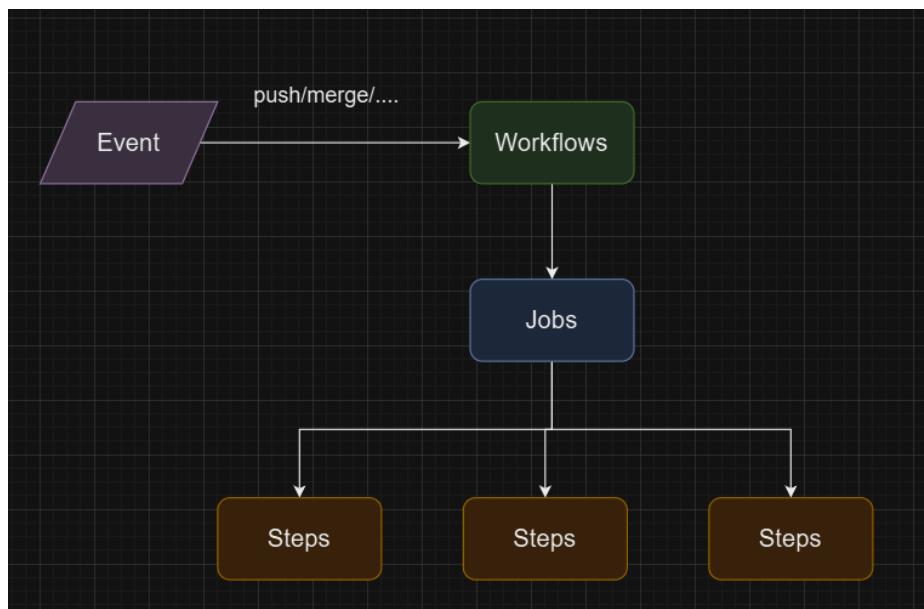
- Dễ sử dụng, không yêu cầu nhiều cấu hình phức tạp.
- Kho thư viện action phong phú trên GitHub Marketplace.
- Hỗ trợ mạnh mẽ cho các dự án nguồn mở (free runner cho public repositories).

Hạn chế:

- Chi phí cao với các dự án private khi sử dụng nhiều runner.
- Phụ thuộc nhiều vào hệ sinh thái GitHub.

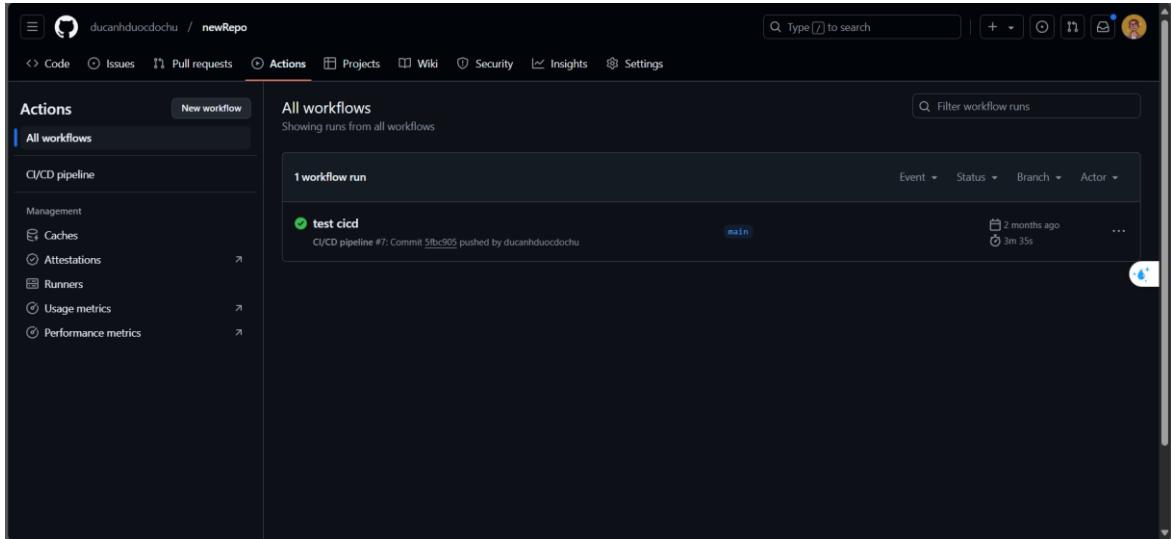
Sử dụng GitHub Actions cho CI/CD:

- CI: Tự động kiểm tra mã nguồn khi có pull request, chạy các kiểm thử tự động.
- CD: Triển khai ứng dụng lên các nền tảng như Docker Hub, AWS, hoặc Kubernetes.



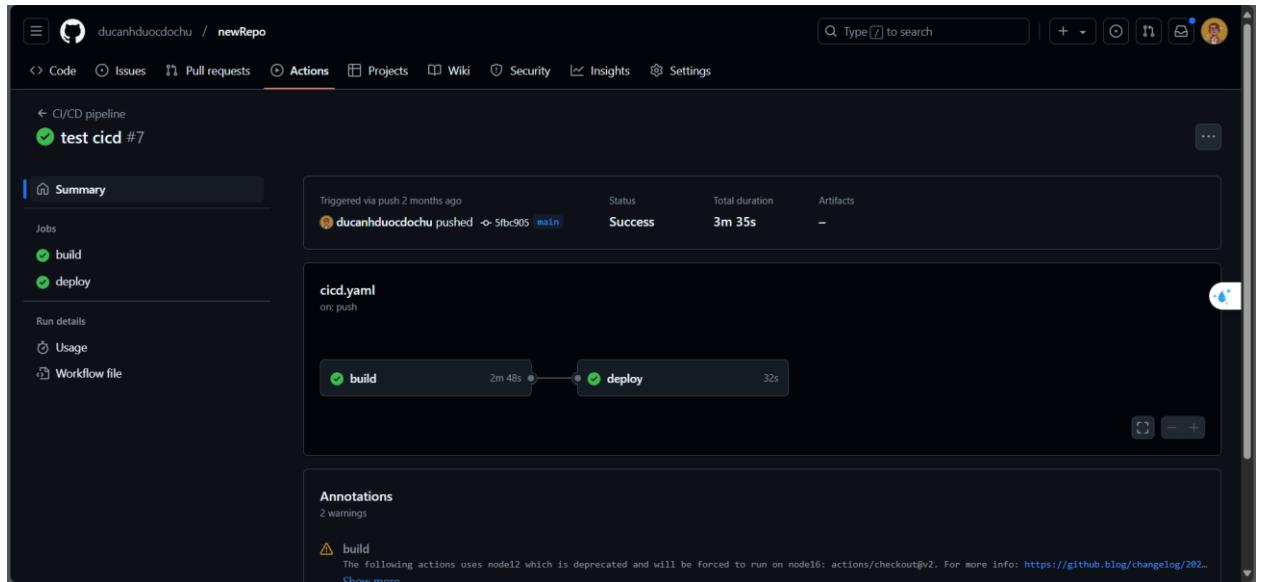
Hình 14. Cấu trúc workflows GitHub Action

- Giao diện GitHub Action trên GitHub, màn hình hiển thị các workflows, các workflows sẽ được chạy khi có event. Ví dụ về giao diện với workflows test cicd



Hình 15. Giao diện hiển thị các workflows của GitHub Action

- Trong workflows là những jobs, build và deploy được khai báo trong script của file .github/workflows/\*.yml trong dự án.



Hình 16. Giao diện các jobs trong các workflows

- Các ví dụ về file .github/workflows/\*.yml trong dự án, được cấu hình theo template của GitHub Action
- + Cấu trúc workflows đơn giản:

```

name: Basic Workflow
on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v3
      - name: Print Hello
        run: echo "Hello, GitHub Actions!"

```

Hình 17. Cấu trúc file workflows đơn giản

name: Tên workflow (hiển thị trên GitHub UI).

on: Sự kiện kích hoạt workflow (push, pull\_request, schedule, etc.).

jobs: Các công việc (job) cần thực hiện.

runs-on: Hệ điều hành runner sẽ sử dụng.

steps: Các bước thực hiện trong job.

+ Workflows sử dụng biến môi trường:

```

jobs:
  build:
    runs-on: ubuntu-latest
    env:
      NODE_ENV: production
    steps:
      - name: Print environment variable
        run: echo "Environment: $NODE_ENV"

```

Hình 18. Cấu trúc file workflows với biến môi trường

Biến môi trường được sử dụng là NODE\_ENV, sử dụng biến môi trường giúp tạo ra các phiên bản khác nhau khi thay đổi biến môi trường.

+ Workflows có nhiều job phụ thuộc:

```

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Build application
        run: echo "Building app"

  test:
    runs-on: ubuntu-latest
    needs: build
    steps:
      - name: Run tests
        run: echo "Testing after build"

```

Hình 19. Cấu trúc file workflows với các job phụ thuộc

Khi job build hoàn thành thì job test khởi chạy.

+ workflows sử dụng cache

```

build:
  runs-on: ubuntu-latest
  steps:
    - name: Checkout code
      uses: actions/checkout@v3

    - name: Cache dependencies
      uses: actions/cache@v3
      with:
        path: ~/.npm
        key: ${{ runner.os }}-node-${{ hashFiles('**/package-lock.json') }}
        restore-keys:
          ${{ runner.os }}-node-

    - name: Install dependencies
      run: npm install

    - name: Build project
      run: npm run build

```

Hình 20. Cấu trúc file workflows với cache

Khi sử dụng cache (bộ nhớ đệm) trường hợp:

+ Dependency management (các phụ thuộc): Lưu trữ các package cài đặt (npm, pip, Maven, Gradle...).

- + Build artifacts: Lưu các file build để sử dụng trong job tiếp theo.
- + Công cụ biên dịch: Lưu các file object được tạo trong quá trình biên dịch.
- + Dữ liệu tạm: Bất kỳ dữ liệu nào cần được tái sử dụng giữa các lần chạy workflow.

Tốc độ pipeline sẽ tăng tốc độ, tuy nhiên GitHub cho phép số lượng dung lượng không quá lớn cho cache, không phù hợp với dự án lớn.

#### 4.2 GitLab CI/CD

GitLab CI/CD là một hệ thống tự động hóa mạnh mẽ được tích hợp sẵn trong GitLab, cho phép người dùng thực hiện toàn bộ quy trình DevOps.

**Đặc điểm chính:**

- Tích hợp hoàn chỉnh trong GitLab.
- Cấu hình pipeline qua file .gitlab-ci.yml.
- Hỗ trợ nhiều runner cho các môi trường khác nhau (Docker, Kubernetes, cloud).

Các thành phần chính:

- Pipeline: Tập hợp các job thực thi theo từng stage.
- Stage: Một giai đoạn trong pipeline (ví dụ: build, test, deploy).
- GitLab Runner: Công cụ thực thi job, có thể tự cài đặt hoặc sử dụng shared runner.

**Ưu điểm:**

- Tích hợp trực tiếp, không cần công cụ bên ngoài.
- Khả năng tùy chỉnh cao với các job phức tạp.
- Hỗ trợ triển khai đa dạng: từ container (Docker) đến cloud (AWS, Azure).

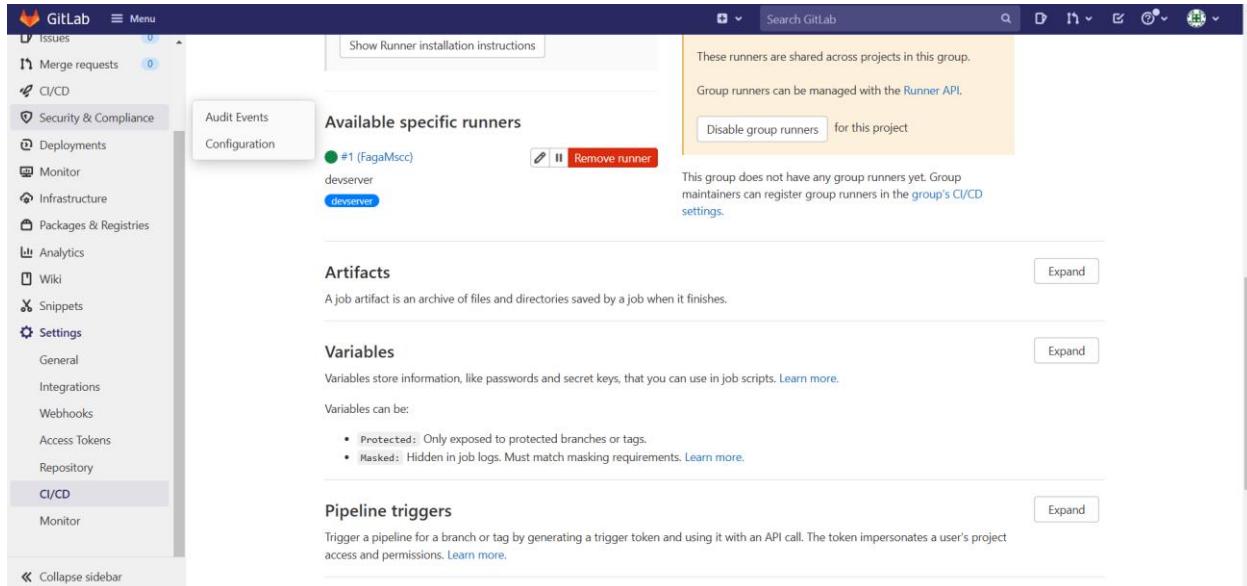
**Hạn chế:**

- Yêu cầu cấu hình nhiều hơn so với GitHub Actions.
- Dễ gặp giới hạn tài nguyên với shared runner miễn phí.

Cách hoạt động của GitLab CI/CD, GitLab CI/CD dựa trên:

- Pipeline: Tập hợp các job chạy theo trình tự hoặc song song để hoàn thành một tác vụ (build, test, deploy).
- Jobs: Các tác vụ riêng lẻ, được định nghĩa trong file .gitlab-ci.yml.

- Stages: Giai đoạn của pipeline, ví dụ: build, test, deploy.
- Runners: Các tác nhân chịu trách nhiệm thực thi job. Runner có thể là shared (của GitLab) hoặc specific (cài đặt riêng), các runner sẽ chạy trên các server được triển khai và được kết nối với GitLab



Hình 21. Giao diện thêm runner trên GitLab

Quy trình CI/CD dựa trên file .gitlab-ci.yml, dựa vào thao tác push code, merge code trên GitLab để kích hoạt runner chạy file CI/CD.

Quy tắc viết file .gitlab-ci.yml:

```

stages:
  - build
  - test
  - deploy

build_job:
  stage: build
  script:
    - echo "Building the application"

test_job:
  stage: test
  script:
    - echo "Running tests"

deploy_job:
  stage: deploy
  script:
    - echo "Deploying to production"
  
```

Hình 22. Cấu trúc cơ bản của file .gitlab-ci.yml

- + Stages: Các giai đoạn của pipeline (build, test, deploy).
- + Jobs: Các tác vụ cụ thể được thực thi trong từng stage.
- + Scripts: Các lệnh chạy trong mỗi job.

```
variables:
  NODE_ENV: production
  DEPLOY_PATH: /var/www/app
```

Hình 23. Biến môi trường trong GitLab CI/CD

- + variables: Các biến môi trường được sử dụng

```
deploy_job:
  stage: deploy
  script:
    - echo "Deploying to $DEPLOY_PATH"
```

Hình 24. Sử dụng biến môi trường trong GitLab CI/CD

```
deploy_job:
  stage: deploy
  script:
    - echo "Deploying to production"
  only:
    - main
```

Hình 25. Sử dụng điều kiện trong GitLab CI/CD

- only: điều kiện chọn job

```
deploy_job:
  stage: deploy
  script:
    - echo "Deploying to production"
  rules:
    - if: '$CI_COMMIT_BRANCH == "main"'
      when: always
    - if: '$CI_COMMIT_BRANCH == "staging"'
      when: manual
```

Hình 26. Sử dụng điều kiện nâng cao trong GitLab CI/CD

- rules: sử dụng rules để linh hoạt hơn với điều kiện

```

cache:
paths:
- node_modules/

```

Hình 27. Sử dụng Cache trong GitLab CI/CD

- cache: sử dụng cache (bộ nhớ đệm) lưu trữ để tăng tốc độ pipeline

### 4.3 Jenkins

Jenkins là một công cụ mã nguồn mở lâu đời và phổ biến trong hệ sinh thái CI/CD. Với khả năng tùy chỉnh cao, Jenkins phù hợp với các tổ chức cần xây dựng pipeline phức tạp.

**Đặc điểm chính:**

- Mã nguồn mở: Hoàn toàn miễn phí và có cộng đồng hỗ trợ lớn.
- Pipeline as Code: Cấu hình qua file Jenkinsfile sử dụng Groovy.
- Plugin phong phú: Hơn 1,800 plugin hỗ trợ tích hợp với các công cụ khác.

Các thành phần chính:

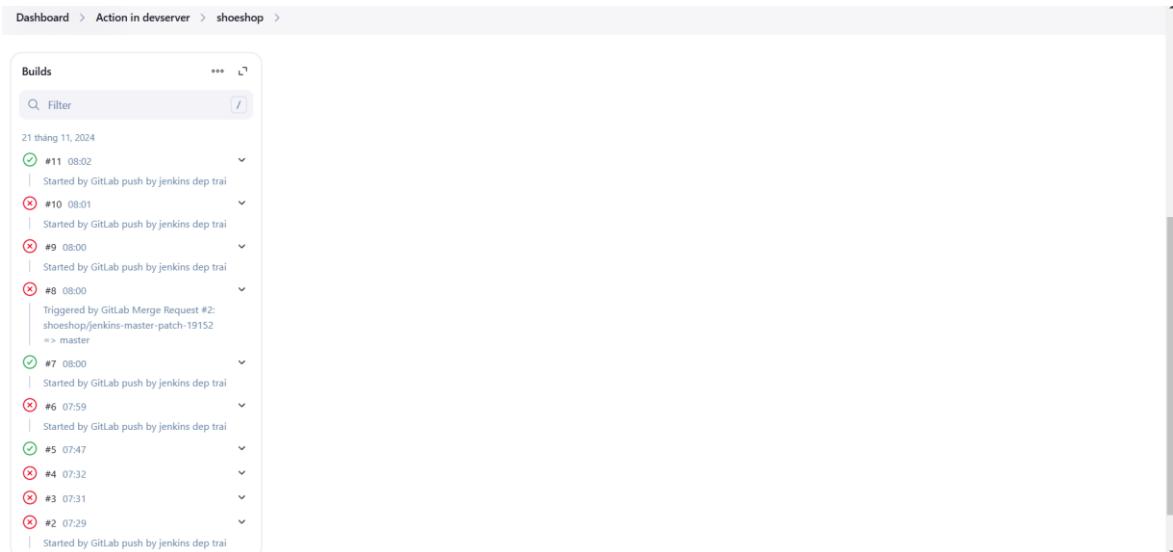
- Jenkins Master: Điều phối và quản lý pipeline.
- Jenkins Agent: Thực thi các job của pipeline.
- Jenkinsfile: File định nghĩa pipeline với các bước chi tiết.

The screenshot shows the Jenkins dashboard with the following components:

- Left Sidebar:** Includes links for "New Item", "Build History", "Project Relationship", "Check File Fingerprint", "Manage Jenkins", "My Views", and "Open Blue Ocean".
- Top Bar:** Shows the Jenkins logo, search bar ("Search (CTRL+K)"), user info ("admin"), and log out button.
- Project List:** A table showing two projects:
 

S	W	Name	Last Success	Last Failure	Last Duration	F
Folder icon	Sun icon	Action in devserver	N/A	N/A	N/A	Star icon
Folder icon	Sun icon	test	N/A	N/A	N/A	Star icon
- Build Queue:** Shows "No builds in the queue."
- Build Executor Status:** Shows "Built-In Node" (0/2) and "dev-server" (offline).
- Bottom Right:** REST API and Jenkins 2.485 links.

Hình 28. Giao diện chính của Jenkins



Hình 29. Giao diện pipeline

**Ưu điểm:**

- Tính linh hoạt cao, phù hợp với nhiều loại dự án.
- Hỗ trợ triển khai trên các môi trường tự quản lý.
- Dễ tích hợp với các công cụ DevOps khác (Docker, Kubernetes, Git).

**Hạn chế:**

- Yêu cầu cấu hình phức tạp, đặc biệt đối với người mới.
- Giao diện không thân thiện bằng các công cụ hiện đại như GitHub Actions hoặc GitLab CI/CD.

Sử dụng Jenkins cho CI/CD:

- CI: Tích hợp với Git để tự động build và kiểm tra mã nguồn.
- CD: Tự động triển khai ứng dụng trên các môi trường staging/production.

## 5. Tổng quan về các công cụ Monitoring

Monitoring là một phần không thể thiếu trong việc quản lý và vận hành hệ thống phần mềm, giúp phát hiện sự cố và cải thiện hiệu suất. Dưới đây là tổng quan về hai công cụ phổ biến: Grafana và Prometheus.

### 5.1 Grafana

Grafana là một nền tảng phân tích và trực quan hóa dữ liệu mã nguồn mở, được sử dụng để theo dõi và hiển thị các chỉ số từ hệ thống.

### Đặc điểm chính:

- Khả năng kết nối đa dạng: Grafana hỗ trợ tích hợp với nhiều nguồn dữ liệu khác nhau như Prometheus, InfluxDB, Elasticsearch, MySQL.
- Dashboard động: Người dùng có thể tạo và tùy chỉnh dashboard theo nhu cầu để hiển thị dữ liệu thời gian thực.
- Cảnh báo (Alerting): Cung cấp tính năng gửi cảnh báo dựa trên các quy tắc được định nghĩa trước.

### Chức năng chính:

- Trực quan hóa: Hiển thị dữ liệu qua biểu đồ, bảng, heatmap, và các widget khác.
- Tùy chỉnh dashboard: Hỗ trợ tạo các dashboard cho từng dịch vụ hoặc bộ phận.
- Cảnh báo: Tích hợp các kênh cảnh báo như email, Slack, PagerDuty.

### Ưu điểm:

- Giao diện thân thiện và trực quan.
- Dễ dàng mở rộng nhờ hệ thống plugin phong phú.
- Hỗ trợ dữ liệu thời gian thực, giúp phát hiện vấn đề nhanh chóng.

### Hạn chế:

- Phụ thuộc vào nguồn dữ liệu bên ngoài như Prometheus.
- Quản lý quyền hạn phức tạp đối với hệ thống lớn.

### Hình minh họa:

- Dashboard Grafana mẫu: Hiển thị CPU, RAM, và trạng thái ứng dụng.

## 5.2 Prometheus

Prometheus là một hệ thống giám sát mã nguồn mở tập trung vào việc thu thập, lưu trữ, và truy vấn dữ liệu dạng time-series (theo thời gian).

### Đặc điểm chính:

- Thu thập dữ liệu dạng time-series: Các chỉ số được gắn nhãn (label) và lưu trữ kèm theo timestamp.

- Ngôn ngữ truy vấn PromQL: Cho phép người dùng viết các truy vấn mạnh mẽ để phân tích dữ liệu.
- Hệ thống cảnh báo tích hợp: Prometheus Alertmanager quản lý và gửi cảnh báo.

Chức năng chính:

- Scraping: Thu thập dữ liệu từ các endpoint được cấu hình.
- Lưu trữ: Lưu trữ dữ liệu time-series trong cơ sở dữ liệu của Prometheus.
- Truy vấn: Sử dụng PromQL để phân tích dữ liệu hoặc gửi cảnh báo.

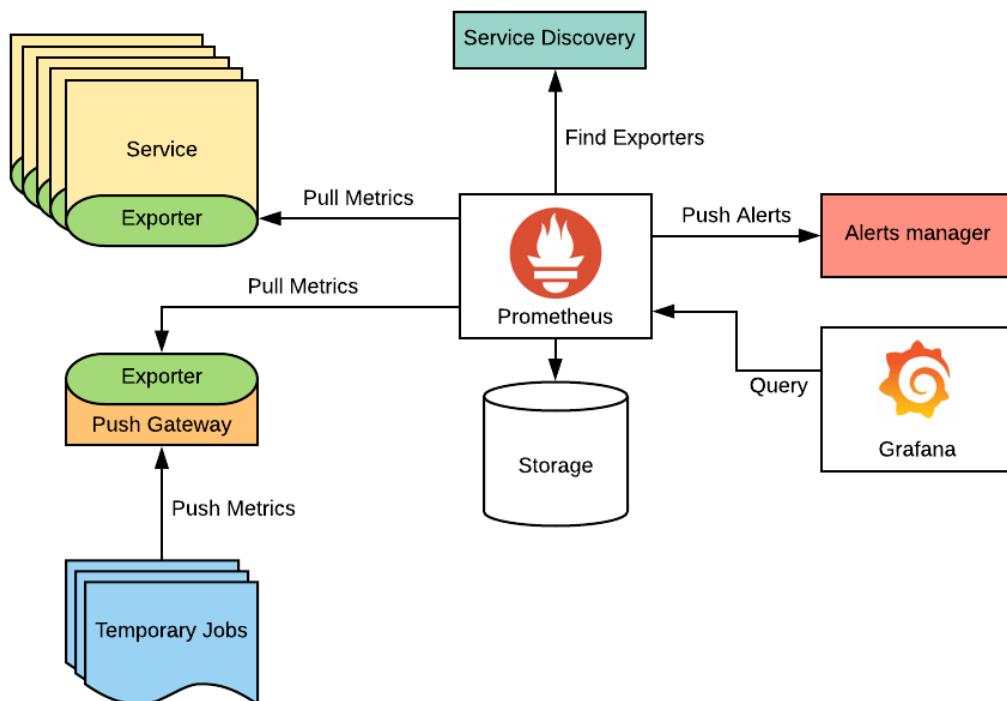
Ưu điểm:

- Hiệu suất cao trong việc thu thập và lưu trữ dữ liệu time-series.
- Hỗ trợ nhãn (label) giúp phân loại và lọc dữ liệu dễ dàng.
- Dễ dàng tích hợp với nhiều công cụ như Grafana.

Hạn chế:

- Không hỗ trợ hệ thống phân tán (cluster) nguyên bản.
- Khó sử dụng cho người mới bắt đầu do yêu cầu cấu hình chi tiết.

Hình minh họa:



Hình 30. Kiến trúc Prometheus

## Tích hợp Grafana và Prometheus trong hệ thống Monitoring

Prometheus thu thập dữ liệu và cung cấp API cho Grafana để hiển thị trực quan. Hai công cụ này thường được sử dụng song song để xây dựng một hệ thống giám sát mạnh mẽ.

Quy trình:

- Prometheus:
  - + Thu thập dữ liệu từ các ứng dụng và dịch vụ qua exporter (ví dụ: Node Exporter, Blackbox Exporter).
  - + Lưu trữ và xử lý dữ liệu.
    - Grafana:
      - + Kết nối với Prometheus qua API.
      - + Tạo các dashboard hiển thị các chỉ số quan trọng.

### III. PHÂN TÍCH VÀ NGHIÊN CỨU

#### 1. Phân tích các mô hình CD/CI

##### 1.1. Mô hình CI/CD đơn giản với GitHub, GitHub Actions.

###### a. Các công cụ sử dụng:

- GitHub: Quản lý mã nguồn và theo dõi lịch sử thay đổi.
- GitHub Actions: Công cụ tích hợp CI/CD để tự động hóa các workflow.
- Docker: Đóng gói ứng dụng thành container để triển khai dễ dàng và thống nhất trên nhiều môi trường.
- Docker Hub: Lưu trữ Docker image và chia sẻ chúng giữa các môi trường.

###### b. Quy trình chi tiết:

Push mã lên GitHub:

- Mỗi khi có thay đổi mã, nhà phát triển push code lên GitHub repository.

Thiết lập GitHub Actions:

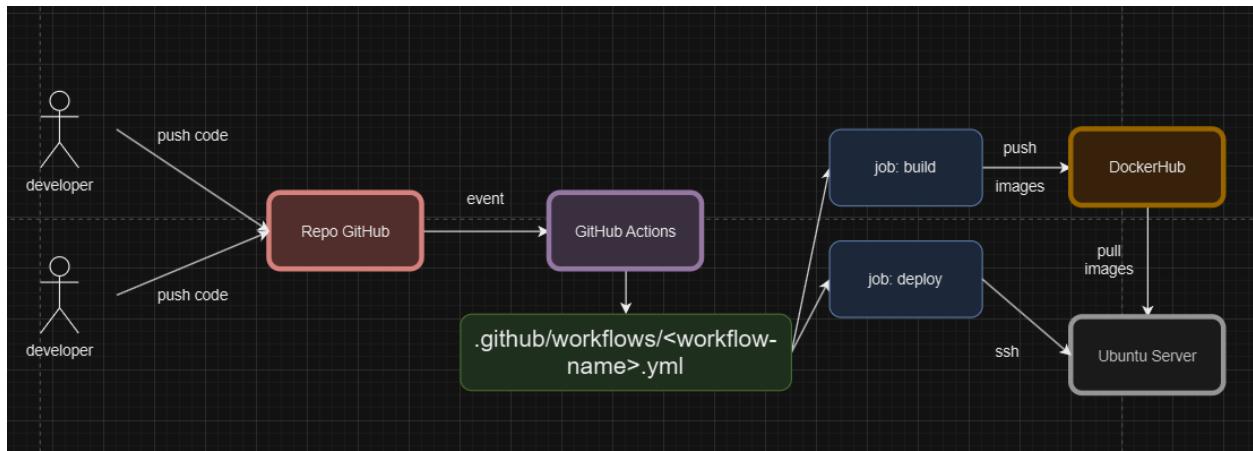
- Tạo workflow trong file .github/workflows/<workflow-name>.yml.
- Các bước trong .github/workflows/<workflow-name>.yml (có thể có) bao gồm:

B1: Kiểm tra mã nguồn (linting, testing): Đảm bảo mã nguồn đạt tiêu chuẩn.

B2: Build Docker image: Đóng gói ứng dụng vào container.

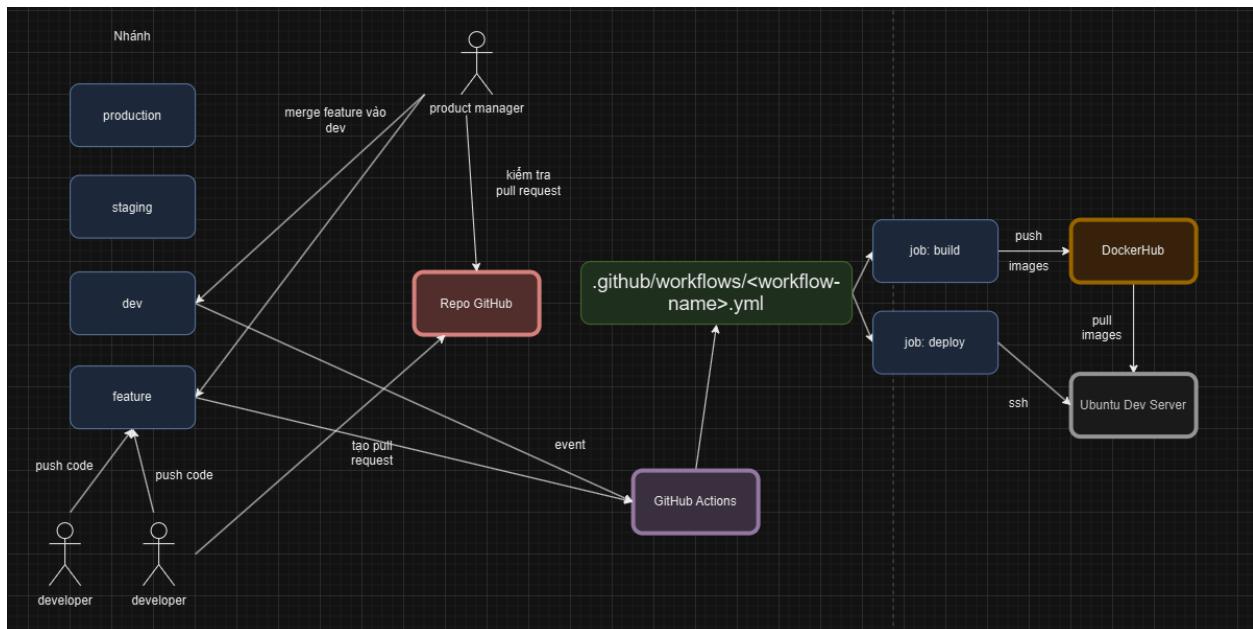
B3: Đăng Docker image lên Docker Hub

B4: Pull Docker image về Server, run container để triển khai



Hình 31. Mô hình CI/CD đơn giản

Sơ đồ với một mô hình CI/CD đơn giản, từ sơ đồ trên, có thể phát triển với thao tác kích hoạt sự kiện GitHub Actions phức tạp hơn, các môi trường đa dạng hơn:



Hình 32. Mô hình CI/CD mở rộng với nhiều môi trường

Thay vì chỉ triển khai phần mềm với một môi trường, CI/CD sẽ được triển khai trên nhiều môi trường, việc triển khai trên nhiều môi trường giúp việc triển khai, kiểm thử, tích hợp và người dùng cuối sử dụng, có thể sử dụng song song.

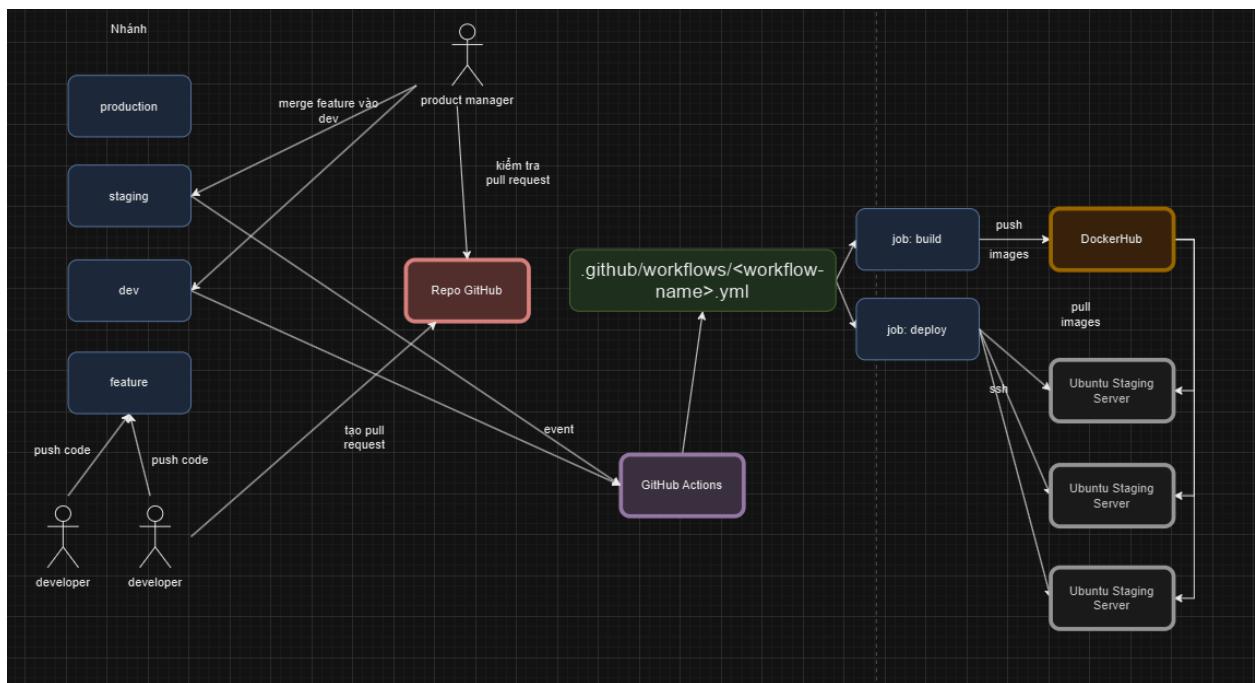
- + Dev: môi trường cho lập trình viên tích hợp
- + Staging: môi trường kiểm thử
- + Production: môi trường cho người dùng cuối

Vì vậy, môi trường triển khai khác nhau, sử dụng các quy trình giống nhau nhưng môi trường triển khai sẽ khác nhau. VD: Ubuntu Dev Server, Ubuntu Staging Server, Ubuntu Production Server

Với thao tác sử dụng push code đơn giản không thể triển khai đa môi trường, phối hợp nhóm trên GitHub.

Thay vì những thao tác tiếp nhận push code, khi lập trình viên gộp code từ những nhánh phụ, nhánh chức năng vào các nhánh của môi trường dev tích hợp, CI/CD sẽ được kích hoạt và triển khai trên môi trường dev.

Tương tự, sau khi tích hợp trên môi trường dev, khi lập trình viên gộp code từ môi trường dev tích hợp vào môi trường staging kiểm thử, CI/CD sẽ được kích hoạt và triển khai trên môi trường staging.



Hình 33. Mô hình CI/CD mở rộng với phương pháp làm việc nhóm

Môi trường staging gần giống với môi trường production, nhưng số lượng server cần triển khai thường nhiều hơn để đảm bảo khả năng chịu tải cao, hỗ trợ tính đồng thời lớn và duy trì sự ổn định của hệ thống.

c. Phù hợp với:

Dự án nhỏ và vừa, yêu cầu CI/CD cơ bản.

Các đội phát triển nhỏ muốn triển khai nhanh mà không cần đầu tư nhiều vào hạ tầng CI/CD.

Dự án sử dụng Docker để đóng gói ứng dụng.

d. Ưu điểm:

Tích hợp chặt chẽ với GitHub: Không cần công cụ bên thứ ba.

Đơn giản hóa quy trình CI/CD: GitHub Actions dễ thiết lập và quản lý.

Hỗ trợ Docker mạnh mẽ: Thuận tiện trong việc triển khai container hóa.

Miễn phí với các dự án open-source: Chi phí thấp cho các dự án cá nhân và đội nhóm nhỏ.

Khả năng tùy chỉnh cao: Workflow có thể mở rộng theo nhu cầu.

e. Nhược điểm:

Giới hạn tài nguyên miễn phí: Với repository private, GitHub chỉ cung cấp một số lượng chạy workflows miễn phí nhất định.

Không phù hợp cho quy mô lớn: Khi dự án phức tạp hơn, việc mở rộng hệ thống CI/CD chỉ với GitHub Actions có thể trở nên khó khăn.

Quản lý secret phức tạp hơn: Nếu không cẩn thận, có thể để lộ các thông tin nhạy cảm trong workflow.

Tốc độ hạn chế: Các máy chủ runner miễn phí của GitHub có thể chậm hơn so với các giải pháp tự host.

## 1.2. Mô hình CI/CD với GitLab

Việc chuyển từ GitHub Actions sang GitLab CI/CD phù hợp khi dự án cần một nền tảng toàn diện để quản lý mã nguồn, CI/CD, và môi trường triển khai. GitLab CI/CD nổi bật với khả năng tích hợp sẵn Container Registry, quản lý secrets bảo mật hơn, và hỗ trợ self-hosted runner giúp tối ưu chi phí khi mở rộng quy mô. Trong khi GitHub Actions phù hợp với các dự án nhỏ và ngắn hạn nhờ sự đơn giản và chi phí thấp, GitLab CI/CD lại vượt trội trong các dự án lớn, đa môi trường, với quy trình phức tạp đòi hỏi sự ổn định và hiệu quả. Nếu đội ngũ phát triển cần tự quản lý hạ tầng hoặc muốn giảm phụ thuộc vào các công cụ riêng lẻ, GitLab CI/CD là lựa chọn chiến lược và bền vững hơn.

a. Các công cụ sử dụng

GitLab: Nền tảng quản lý mã nguồn và tích hợp CI/CD trực tiếp.

GitLab CI/CD: Công cụ tích hợp sẵn trong GitLab để tự động hóa các quy trình CI/CD.

GitLab Runner: Tác nhân (agent) chịu trách nhiệm thực thi các pipeline/job của GitLab CI/CD.

Docker: Đóng gói ứng dụng thành container để triển khai nhất quán trên nhiều môi trường.

Container Registry (GitLab): Lưu trữ Docker image trực tiếp trên GitLab, giúp quản lý và tích hợp chặt chẽ với pipeline.

### b. Quy trình chi tiết

#### Bước 1: Push mã lên GitLab

Nhà phát triển push mã nguồn lên GitLab repository (trên các nhánh như dev, staging, main,...).

Nhà phát triển có thể sử dụng merge mã nguồn giữa các môi trường,

#### Bước 2: Thiết lập pipeline trong GitLab

Tạo file .gitlab-ci.yml trong repository để định nghĩa workflow CI/CD. File này chứa các stages và jobs tương ứng với từng bước triển khai.

#### Bước 3: Các bước trong pipeline (có thể có)

##### + Bước 3.1: Kiểm tra mã nguồn (Linting, Testing)

Đảm bảo mã nguồn đạt tiêu chuẩn và không có lỗi.

##### + Bước 3.2: Build Docker Image

Sử dụng Docker để đóng gói ứng dụng thành container.

##### + Bước 3.3: Đẩy Docker Image lên Container Registry (GitLab hoặc Docker Hub)

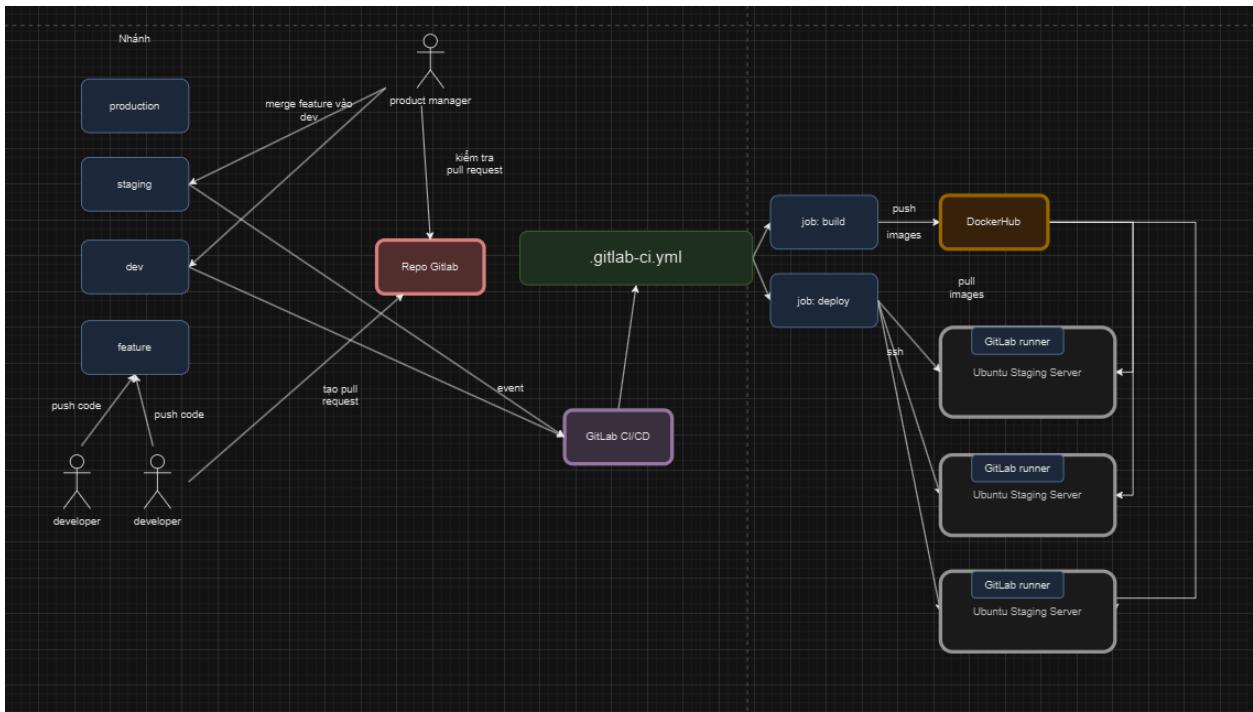
Lưu trữ image đã build để sử dụng cho các môi trường tiếp theo.

##### + Bước 3.4: Deploy Docker Container lên server

Pull Docker image từ Container Registry.

Chạy container để triển khai ứng dụng.

#### Sơ đồ mô hình CI/CD:



Hình 34. Mô hình CI/Cd với GitLab

- Một quy trình triển khai GitLab CICD trên môi trường staging
- GitLab sử dụng phương pháp làm việc nhóm của GitHub, vì vậy, tương tự các quy trình thao tác từ push code, merge code,... giống như GitHub.
- File .gitlab-ci.yml trong mã nguồn sẽ giúp khởi chạy quá trình CI/CD trong file.
- Trên các máy chủ khởi chạy cần cài đặt cascc gitlab runner và kết nối với gitlab.

Phát triển mô hình CI/CD đa môi trường với GitLab:

- Quy trình triển khai trên nhiều môi trường:
  - + Dev (Development): Môi trường tích hợp của lập trình viên. CI/CD kích hoạt khi có thay đổi trên nhánh dev.
  - + Staging: Môi trường kiểm thử tích hợp. CI/CD kích hoạt khi có thay đổi từ nhánh dev merge vào nhánh staging.
  - + Production: Môi trường triển khai cho người dùng cuối. CI/CD kích hoạt khi nhánh staging merge vào main. Cách thiết lập pipeline cho từng môi trường.

Ưu điểm:

- Tích hợp mạnh mẽ với GitLab: Không cần công cụ bên ngoài, mọi thứ quản lý trong một nền tảng.
- Hỗ trợ Container Registry: Tích hợp lưu trữ Docker image dễ dàng.
- Tính năng mạnh mẽ như Review Apps, Environments: Hỗ trợ kiểm thử và triển khai linh hoạt.
- Quản lý pipelines chuyên sâu: Dễ dàng giám sát trạng thái pipeline và môi trường.
- Shared và Specific Runners: Tối ưu chi phí bằng shared runner hoặc tối đa hóa hiệu năng với specific runner.

Nhược điểm:

- Yêu cầu cấu hình phức tạp hơn ban đầu: Đặc biệt với dự án lớn.
- Hạn chế tài nguyên với GitLab.com (phiên bản miễn phí): Số lượng minute sử dụng CI/CD giới hạn.
- Chi phí cao hơn cho Self-hosted Runner: Nếu tự triển khai hạ tầng riêng.

### 1.3. Mô hình CI/CD với GitLab, Jenkins

Việc chuyển từ GitHub Actions sang GitLab CI/CD phù hợp khi dự án cần một nền tảng toàn diện để quản lý mã nguồn, CI/CD, và môi trường triển khai. GitLab CI/CD nổi bật với khả năng tích hợp sẵn Container Registry, quản lý secrets bảo mật hơn, và hỗ trợ self-hosted runner giúp tối ưu chi phí khi mở rộng quy mô. Trong khi GitHub Actions phù hợp với các dự án nhỏ và ngắn hạn nhờ sự đơn giản và chi phí thấp, GitLab CI/CD lại vượt trội trong các dự án lớn, đa môi trường, với quy trình phức tạp đòi hỏi sự ổn định và hiệu quả. Nếu đội ngũ phát triển cần tự quản lý hạ tầng hoặc muốn giảm phụ thuộc vào các công cụ riêng lẻ, GitLab CI/CD là lựa chọn chiến lược và bền vững hơn.

#### a. Các công cụ sử dụng

GitLab: Lưu trữ mã nguồn và tích hợp GitLab CI/CD.

GitLab Runner: Kích hoạt các pipeline cơ bản.

Jenkins: Công cụ CI/CD mạnh mẽ, phù hợp với quy trình phức tạp.

Jenkins Agent: Tác nhân xử lý job, đảm bảo khả năng mở rộng.

Docker: Đóng gói và triển khai ứng dụng.

Container Registry (Docker Hub hoặc GitLab): Lưu trữ Docker image.b. Quy trình chi tiết

### b. Quy trình chi tiết

B1: Push mã nguồn lên GitLab: Nhà phát triển đẩy mã lên repository GitLab (dev, staging, main).

B2: Thiết lập GitLab CI/CD với Jenkins: Trong GitLab, file .gitlab-ci.yml sẽ được cấu hình để kích hoạt job trong Jenkins.

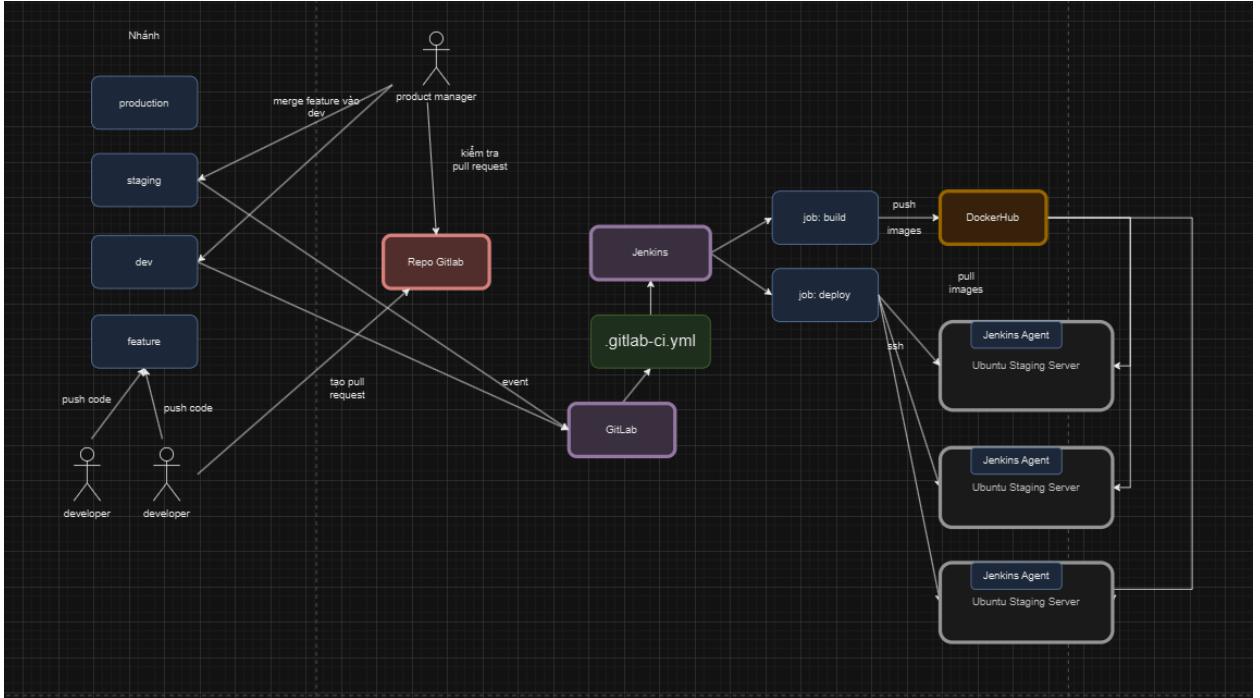
B3: Tạo Jenkins Pipeline: Pipeline trong Jenkins sẽ định nghĩa các bước:

- Kiểm tra mã nguồn: Linting, Unit Test.
- Build Docker Image: Đóng gói ứng dụng.
- Push Docker Image: Lưu trữ trên Container Registry.
- Triển khai: Pull và chạy container.

Khi kết hợp Jenkins với GitLab, Jenkins có thể tự động thực thi các job xây dựng, kiểm tra và triển khai ứng dụng mỗi khi có sự thay đổi trong kho mã nguồn GitLab. Quá trình này thường dựa trên webhooks, cho phép GitLab thông báo cho Jenkins về các sự kiện như commit mới, pull request hoặc merge request. Dưới đây là cách hoạt động của Jenkins khi kết hợp với GitLab:

- Cài đặt webhook trên GitLab: GitLab có thể gửi các sự kiện như push, merge request hoặc tag đến Jenkins thông qua webhook. Bạn sẽ cấu hình webhook trong GitLab để trả đến URL của Jenkins server, thường là một Jenkins job hoặc một pipeline cụ thể.
- Jenkins nhận thông báo: Khi một sự kiện xảy ra trong GitLab (ví dụ, push code mới lên kho GitLab), webhook sẽ kích hoạt Jenkins để bắt đầu một job. Jenkins Master nhận được thông báo và xử lý yêu cầu.
- Jenkins chạy job: Jenkins có thể thực thi các công việc trên một hoặc nhiều Agent (nếu có) để build, test, deploy hoặc thực hiện các tác vụ khác liên quan đến mã nguồn được cập nhật từ GitLab. Tùy vào cấu hình, Jenkins có thể chạy trực tiếp trên Master hoặc phân tán công việc cho các Agent.
- Cấu hình CI/CD pipeline trong Jenkins: Trong Jenkins, bạn có thể cấu hình các pipeline để xây dựng, kiểm tra và triển khai mã nguồn từ GitLab tự động. Jenkins sử dụng các plugin như GitLab Plugin, Git Plugin để lấy mã nguồn từ GitLab, rồi thực hiện các bước như compile, test, deploy.

- Kết quả và thông báo: Sau khi job hoàn thành, Jenkins sẽ gửi thông báo về kết quả thực thi (thành công hoặc thất bại) trở lại GitLab. Điều này có thể được thể hiện qua các comment trong merge request hoặc push status, giúp các developer biết được trạng thái build của mã nguồn mới.



Hình 35. Kiến trúc CICD với GitLab, Jenkins

### Ưu điểm:

- Khả năng mở rộng cao: Jenkins Agent phân tách tài nguyên, xử lý đồng thời nhiều job.
- Tích hợp chặt chẽ: GitLab quản lý mã nguồn, Jenkins đảm nhiệm CI/CD.
- Tính linh hoạt: Jenkins Pipeline có thể tùy chỉnh cho quy trình phức tạp.

### Nhược điểm:

- Cấu hình phức tạp: Yêu cầu thiết lập kết nối giữa GitLab và Jenkins, cũng như quản lý Jenkins Agent.
- Chi phí vận hành cao: Nếu sử dụng nhiều Agent hoặc self-hosted infrastructure.

## 2. Các yêu cầu về monitoring trong DevOps

Monitoring là một phần không thể thiếu trong DevOps để đảm bảo hệ thống hoạt động ổn định, phát hiện lỗi kịp thời và tối ưu hóa hiệu suất. Các yêu cầu chính gồm:

### 2.1. Yêu cầu cơ bản

Hiệu năng hệ thống (Performance Monitoring):

- Giám sát CPU, RAM, ổ cứng, băng thông mạng, và các chỉ số cơ bản khác.
- Ví dụ: Sử dụng Prometheus + Grafana để hiển thị dashboard.

Trạng thái dịch vụ (Service Health Monitoring):

- Đảm bảo các dịch vụ và ứng dụng đang hoạt động bình thường (HTTP, database, API).
- Sử dụng công cụ như Nagios, Zabbix, hoặc UptimeRobot.

Log Monitoring:

- Theo dõi log hệ thống và ứng dụng để phát hiện lỗi.
- Sử dụng ELK Stack (Elasticsearch, Logstash, Kibana) hoặc Splunk.

Cảnh báo (Alerting):

- Gửi cảnh báo qua email, Slack, hoặc SMS khi xảy ra sự cố.
- Công cụ: AlertManager (Prometheus), PagerDuty.

### 2.2. Yêu cầu nâng cao

Application Performance Monitoring (APM):

- Theo dõi hiệu suất ứng dụng ở mức chi tiết (response time, error rate, transaction tracing).
- Công cụ: New Relic, Dynatrace, AppDynamics.

Distributed Tracing:

- Theo dõi yêu cầu qua nhiều dịch vụ microservices.
- Công cụ: Jaeger, Zipkin.

User Experience Monitoring:

- Giám sát hành vi người dùng và trải nghiệm trên giao diện.
- Công cụ: Google Analytics, Datadog RUM.

## Capacity Planning:

- Dự đoán tài nguyên cần thiết dựa trên lịch sử sử dụng.
- Công cụ: AWS CloudWatch, Azure Monitor.

## 2.3. Thực tiễn tốt nhất

Tự động hóa: Thiết lập cảnh báo và hành động tự động khi xảy ra lỗi.

Quan sát tập trung (Centralized Monitoring): Thu thập log và metric từ tất cả các dịch vụ về một nơi.

CI/CD Monitoring: Giám sát pipeline để phát hiện lỗi trong build/test/deploy.

## 3. So sánh các mô hình CI/CD

Yếu tố	GitHub Action	GitLab CI/CD	Jenkins CI/CD
1. Chi phí	<ul style="list-style-type: none"> <li>- Miễn phí cho public repo.</li> <li>- Có giới hạn cho private repo (2,000 phút/tháng gói Free, thêm phí nếu vượt quá).</li> <li>- Gói trả phí: Bắt đầu từ \$4/user/tháng.</li> </ul>	<ul style="list-style-type: none"> <li>- Miễn phí với GitLab Community Edition (CE).</li> <li>- GitLab SaaS Premium: \$19/user/tháng.</li> <li>- Tính phí theo số lượng pipeline nếu dùng GitLab Runner trên GitLab SaaS.</li> </ul>	<ul style="list-style-type: none"> <li>- Open-source (miễn phí).</li> <li>- Tốn chi phí cài đặt, quản lý server và các plugin trả phí nếu cần.</li> </ul>
2. Nhân lực	<ul style="list-style-type: none"> <li>- Tự động hóa cao.</li> <li>- Ít yêu cầu về quản trị (do chạy trên cloud của GitHub).</li> </ul>	<ul style="list-style-type: none"> <li>- Đơn giản nếu sử dụng GitLab SaaS.</li> <li>- Nếu dùng GitLab tự quản lý, cần quản trị hệ thống và runner.</li> </ul>	<ul style="list-style-type: none"> <li>- Cần đội ngũ quản lý server Jenkins.</li> <li>- Cần cấu hình và cập nhật plugin thủ công.</li> </ul>
3. Quy mô dự án	<ul style="list-style-type: none"> <li>- Phù hợp với nhóm vừa và nhỏ hoặc dự án open-source.</li> <li>- Tích hợp tốt với các dự án trên GitHub.</li> </ul>	<ul style="list-style-type: none"> <li>- Phù hợp với mọi quy mô, từ nhỏ đến rất lớn.</li> <li>- Tốt cho các tổ chức sử dụng GitLab làm nền tảng chính.</li> </ul>	<ul style="list-style-type: none"> <li>- Phù hợp với quy mô lớn và phức tạp.</li> <li>- Hỗ trợ tốt cho các dự án lớn, đa dịch vụ.</li> </ul>

4. Khả năng mở rộng	<ul style="list-style-type: none"> <li>- Giới hạn vào GitHub ecosystem.</li> <li>- Hỗ trợ workflow đa nền tảng nhưng không mở rộng như Jenkins.</li> </ul>	<ul style="list-style-type: none"> <li>- Hỗ trợ tốt qua GitLab Runner.</li> <li>- Tích hợp đa nền tảng, nhưng phụ thuộc vào GitLab ecosystem.</li> </ul>	<ul style="list-style-type: none"> <li>- Khả năng mở rộng cao nhất.</li> <li>- Hỗ trợ plugin đa dạng, có thể tích hợp với mọi công cụ và môi trường.</li> </ul>
5. Dễ sử dụng	<ul style="list-style-type: none"> <li>- Cực kỳ dễ sử dụng với giao diện trực quan.</li> <li>- Workflow được định nghĩa bằng YAML.</li> </ul>	<ul style="list-style-type: none"> <li>- Giao diện thân thiện, dễ cấu hình pipeline qua YAML.</li> <li>- Cung cấp template cho các pipeline phổ biến.</li> </ul>	<ul style="list-style-type: none"> <li>- Khá phức tạp đối với người mới bắt đầu.</li> <li>- Pipeline thường cần viết script chi tiết.</li> </ul>
6. Tích hợp hệ sinh thái	<ul style="list-style-type: none"> <li>- Tích hợp tốt với GitHub, Azure, và các công cụ cloud phổ biến như AWS, GCP.</li> </ul>	<ul style="list-style-type: none"> <li>- Tích hợp tốt với GitLab, Kubernetes, và các công cụ CI/CD khác.</li> </ul>	<ul style="list-style-type: none"> <li>- Hỗ trợ tích hợp đa dạng với bất kỳ công cụ nào qua plugin (Docker, Kubernetes, AWS, v.v.).</li> </ul>
7. Tính linh hoạt	<ul style="list-style-type: none"> <li>- Hạn chế nếu cần workflow phức tạp ngoài phạm vi GitHub.</li> </ul>	<ul style="list-style-type: none"> <li>- Tốt nếu bạn sử dụng GitLab làm trung tâm quản lý.</li> </ul>	<ul style="list-style-type: none"> <li>- Linh hoạt cao, tùy chỉnh theo nhu cầu cụ thể của doanh nghiệp.</li> </ul>
8. Cộng đồng và hỗ trợ	<ul style="list-style-type: none"> <li>- Cộng đồng lớn và hỗ trợ mạnh mẽ qua GitHub Docs.</li> <li>- Không có hỗ trợ chính thức cho gói miễn phí.</li> </ul>	<ul style="list-style-type: none"> <li>- Cộng đồng GitLab phát triển nhanh.</li> <li>- Hỗ trợ tốt hơn nếu dùng GitLab Premium hoặc Ultimate.</li> </ul>	<ul style="list-style-type: none"> <li>- Cộng đồng rất lớn (mã nguồn mở lâu đời).</li> <li>- Hỗ trợ chậm nếu không thuê dịch vụ của bên thứ ba.</li> </ul>
9. Tính bảo mật	<ul style="list-style-type: none"> <li>- Hỗ trợ bảo mật mạnh mẽ (GitHub Secret, mã hóa pipeline).</li> </ul>	<ul style="list-style-type: none"> <li>- Bảo mật tốt, đặc biệt trong GitLab Premium (DLP, kiểm soát truy cập).</li> </ul>	<ul style="list-style-type: none"> <li>- Phụ thuộc vào cấu hình và plugin, dễ có lỗ hổng nếu không quản lý chặt.</li> </ul>
10. Tính triển khai	<ul style="list-style-type: none"> <li>- Triển khai đơn giản trên cloud.</li> </ul>	<ul style="list-style-type: none"> <li>- Hỗ trợ triển khai tốt với Kubernetes</li> </ul>	<ul style="list-style-type: none"> <li>- Hỗ trợ triển khai phức tạp, đa nền tảng, phù hợp cho</li> </ul>

		và cloud (AWS, GCP, Azure).	các hệ thống on-premise hoặc hybrid.
--	--	-----------------------------	--------------------------------------

Nên chọn GitHub Actions khi:

- Dự án đang được lưu trữ trên GitHub.
- Nhóm nhỏ hoặc vừa, ưu tiên tự động hóa mà không cần nhiều quản trị.
- Muốn sử dụng một công cụ CI/CD gọn nhẹ, chi phí thấp, dễ bắt đầu.

Nên chọn GitLab CI/CD khi:

- Sử dụng GitLab làm nền tảng quản lý mã nguồn chính.
- Cần tích hợp CI/CD toàn diện, đặc biệt với Kubernetes hoặc multi-cloud.
- Quy mô tổ chức lớn hơn, cần tính năng cao cấp như bảo mật và quản lý pipeline phức tạp.

Nên chọn Jenkins khi:

- Dự án lớn, đa nền tảng, hoặc yêu cầu tùy chỉnh phức tạp.
- Muốn kiểm soát hoàn toàn hệ thống CI/CD và không muốn phụ thuộc vào một nhà cung cấp cụ thể.
- Có đội ngũ DevOps hoặc nhân lực quản trị để duy trì hệ thống.

# IV. XÂY DỰNG VÀ TRIỂN KHAI HỆ THỐNG MẪU

## 1. Phân tích, thiết kế ứng dụng API

### 1.1. Use Case Diagram (Biểu đồ ca sử dụng)

Biểu đồ này thể hiện các chức năng chính của hệ thống và mối quan hệ giữa người dùng với các chức năng.

Các thành phần chính:

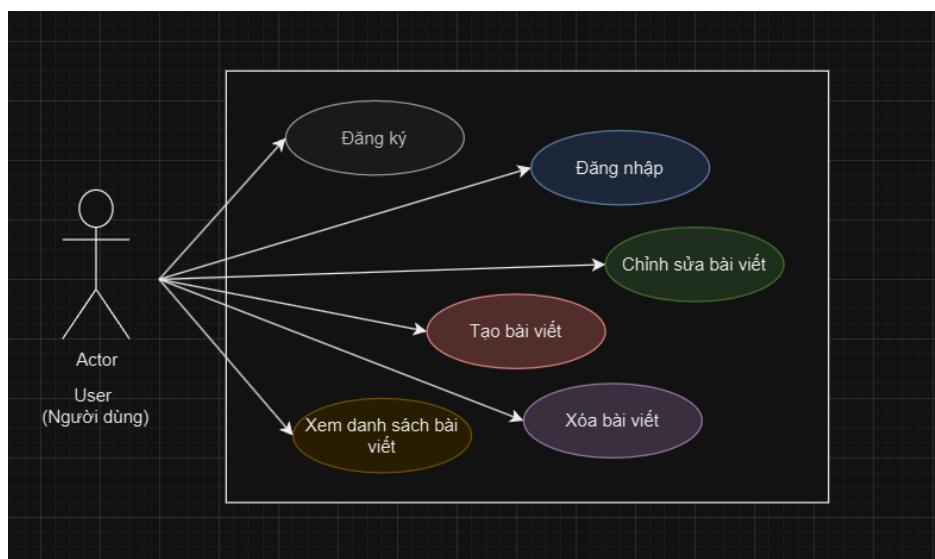
Actor:

- User: Người dùng hệ thống.

Use Cases:

- Đăng ký (Register).
- Đăng nhập (Login).
- Tạo bài viết (Create Post).
- Xem danh sách bài viết (View Posts).
- Chính sửa bài viết (Edit Post).
- Xóa bài viết (Delete Post).

Biểu đồ:



### 1.2. Actor

User:

- Đăng ký, đăng nhập và thực hiện các thao tác CRUD trên bài học.
- Làm việc với hệ thống thông qua giao diện frontend hoặc công cụ gửi yêu cầu (Postman).

### 1.3 Phân tích chi tiết

#### a. Usecase Đăng ký

Tên Use Case: Đăng ký tài khoản

Mô tả sơ lược: Người dùng mới tạo tài khoản trên hệ thống để có thể truy cập và sử dụng các chức năng khác.

Actor chính: Người dùng (User).

Actor phụ: Không có.

Tiền điều kiện: Người dùng chưa có tài khoản trong hệ thống.

Hậu điều kiện: Tài khoản mới được tạo thành công, thông tin được lưu vào cơ sở dữ liệu.

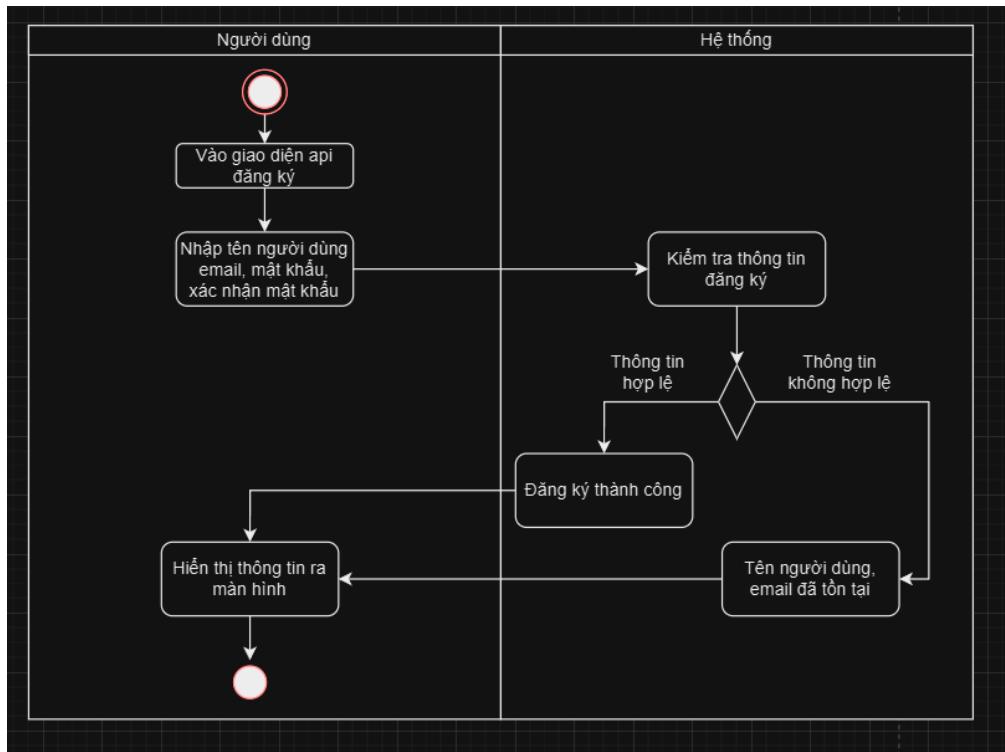
Luồng sự kiện chính:

- Người dùng nhập thông tin đăng ký (username, email, password).
- Hệ thống kiểm tra các trường thông tin đã nhập: Username chưa tồn tại. Email hợp lệ.
- Hệ thống lưu thông tin người dùng vào cơ sở dữ liệu.
- Hệ thống gửi phản hồi "Đăng ký thành công" cho người dùng.

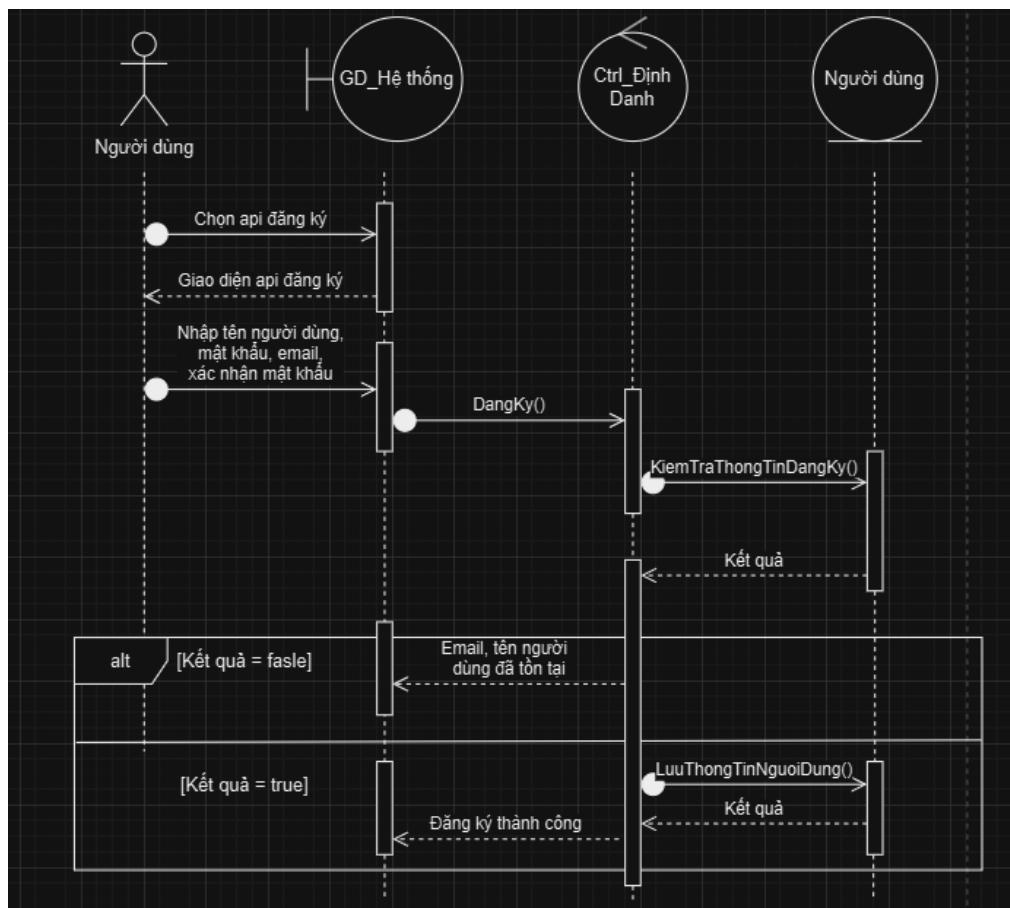
Luồng sự kiện thay thế:

- Nếu username đã tồn tại hoặc email không hợp lệ, hệ thống trả về thông báo lỗi và yêu cầu người dùng nhập lại thông tin.

Sequence diagram:



Activity diagram:



## b. Usecase Đăng nhập

Tên Use Case: Đăng nhập

Mô tả sơ lược: Người dùng đã có tài khoản đăng nhập để sử dụng các tính năng của hệ thống.

Actor chính: Người dùng (User).

Actor phụ: Không có.

Tiền điều kiện: Người dùng đã có tài khoản hợp lệ.

Hậu điều kiện: Người dùng được xác thực và nhận token truy cập.

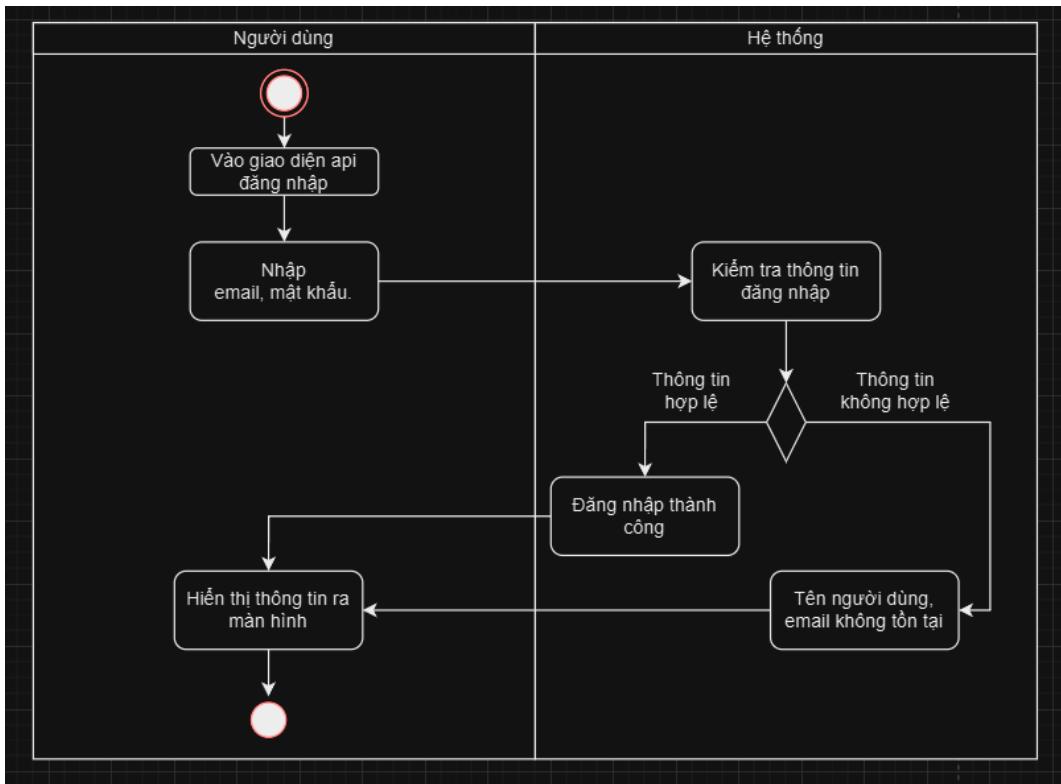
Luồng sự kiện chính:

- Người dùng nhập thông tin đăng nhập (username, password).
- Hệ thống kiểm tra thông tin đăng nhập: Username tồn tại. Password khớp với username đã lưu trong cơ sở dữ liệu.
- Hệ thống tạo token truy cập (JWT).
- Hệ thống trả về token cho người dùng.

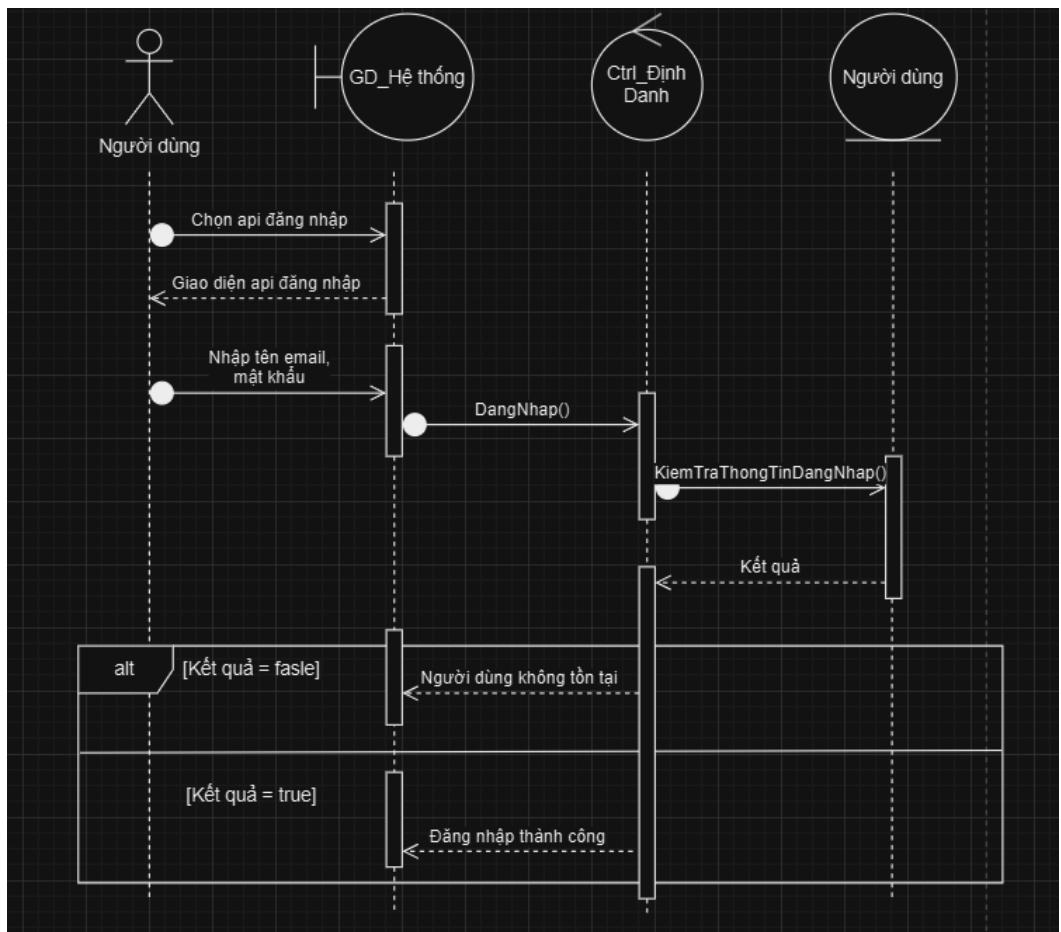
Luồng sự kiện thay thế:

- Nếu username không tồn tại hoặc password sai, hệ thống trả về thông báo lỗi "Đăng nhập thất bại" và yêu cầu nhập lại thông tin.

Sequence diagram:



Activity diagram:



### c. Usecase Tạo bài viết

Tên Use Case: Tạo bài viết

Mô tả sơ lược: Người dùng tạo một bài viết mới trên hệ thống.

Actor chính: Người dùng (User).

Actor phụ: Không có.

Tiền điều kiện: Người dùng đã đăng nhập và có quyền tạo bài viết.

Hậu điều kiện: Bài viết mới được lưu vào cơ sở dữ liệu.

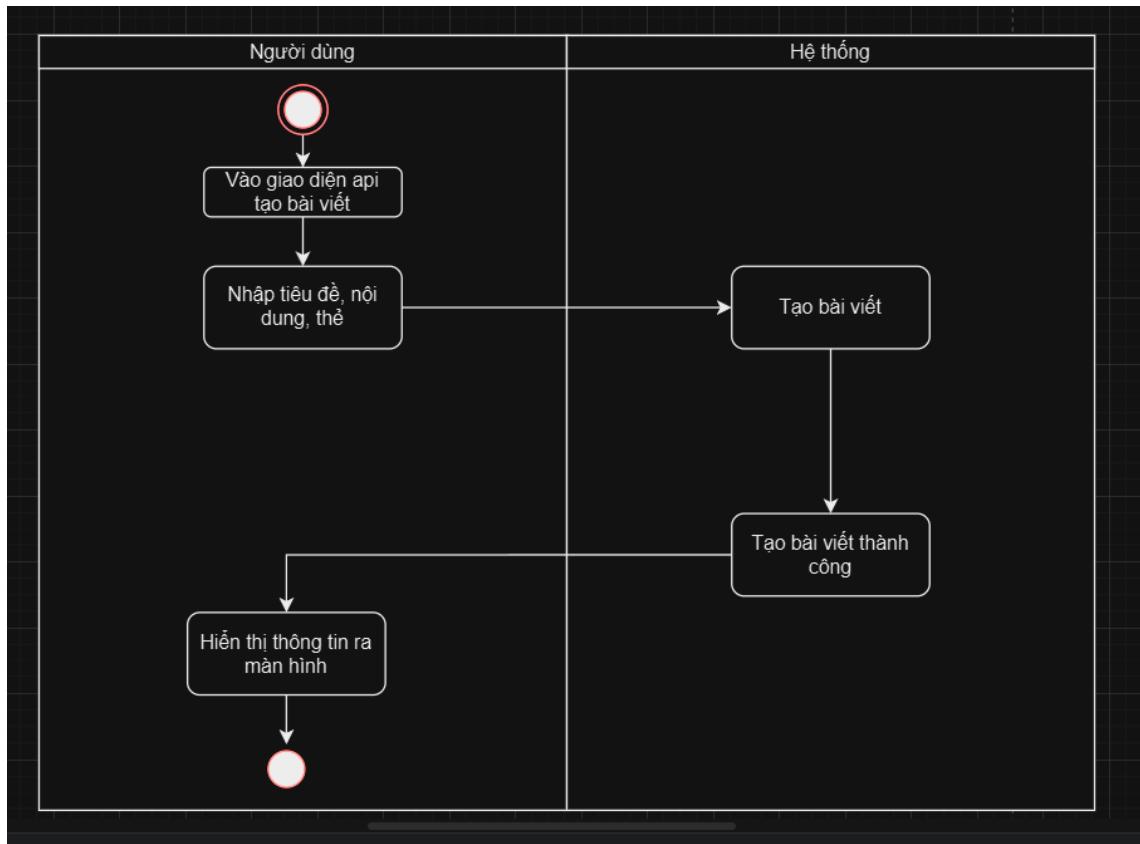
Luồng sự kiện chính:

- Người dùng nhập thông tin bài viết (title, content, tags nếu có).
- Hệ thống kiểm tra thông tin đầu vào: Title và content không rỗng. Hệ thống lưu bài viết vào cơ sở dữ liệu kèm thông tin người tạo. Hệ thống trả về thông báo "Tạo bài viết thành công" cùng thông tin bài viết.

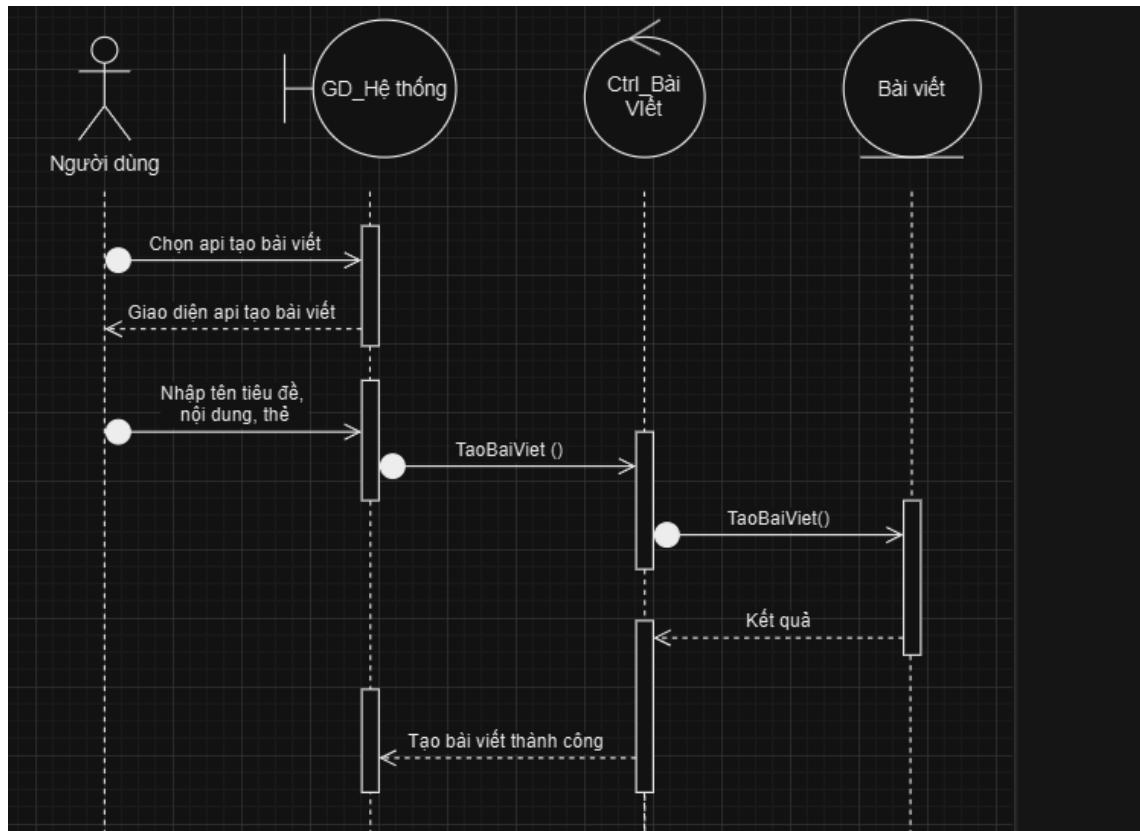
Luồng sự kiện thay thế:

- Nếu title hoặc content bị thiếu, hệ thống trả về thông báo lỗi và yêu cầu người dùng nhập lại.

Sequence diagram:



Activity diagram:



#### d. Usecase Xem danh sách bài viết

Tên Use Case: Xem danh sách bài viết

Mô tả sơ lược: Người dùng xem danh sách các bài viết hiện có trong hệ thống.

Actor chính: Người dùng (User).

Actor phụ: Không có.

Tiền điều kiện: Không có.

Hậu điều kiện: Danh sách bài viết được hiển thị cho người dùng.

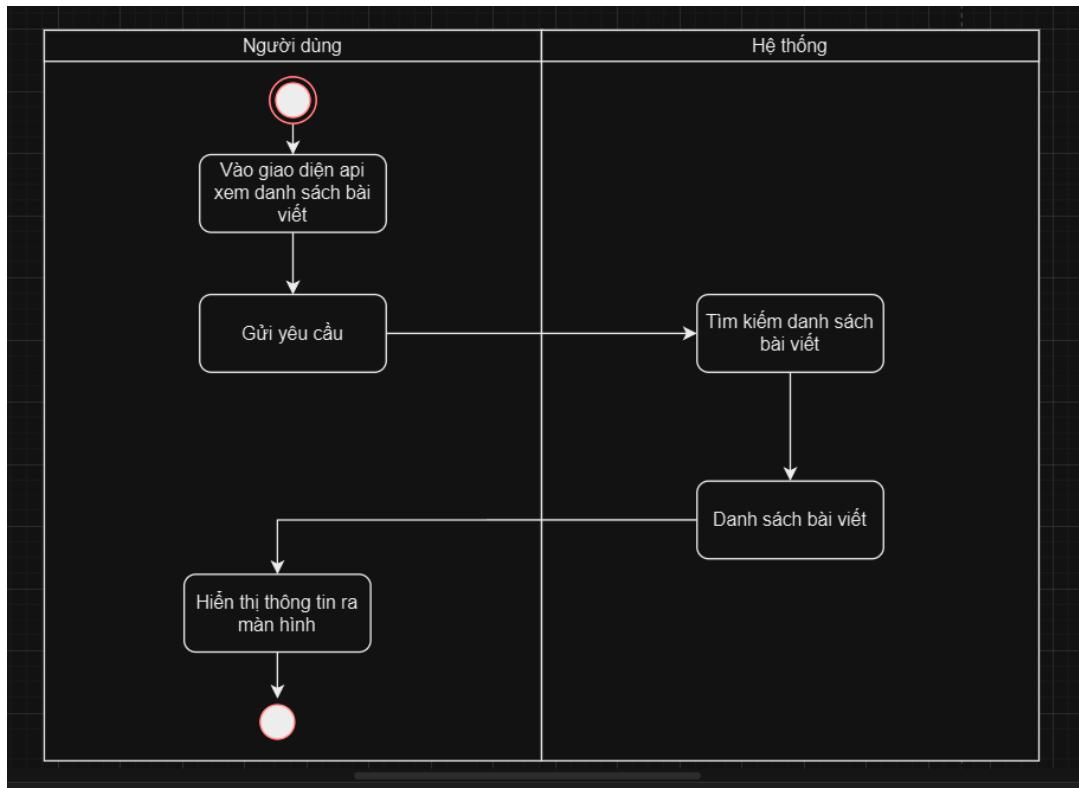
Luồng sự kiện chính:

- Người dùng gửi yêu cầu lấy danh sách bài viết.
- Hệ thống truy vấn cơ sở dữ liệu để lấy danh sách bài viết.
- Hệ thống trả về danh sách bài viết bao gồm các thông tin cơ bản (title, author, createdAt).

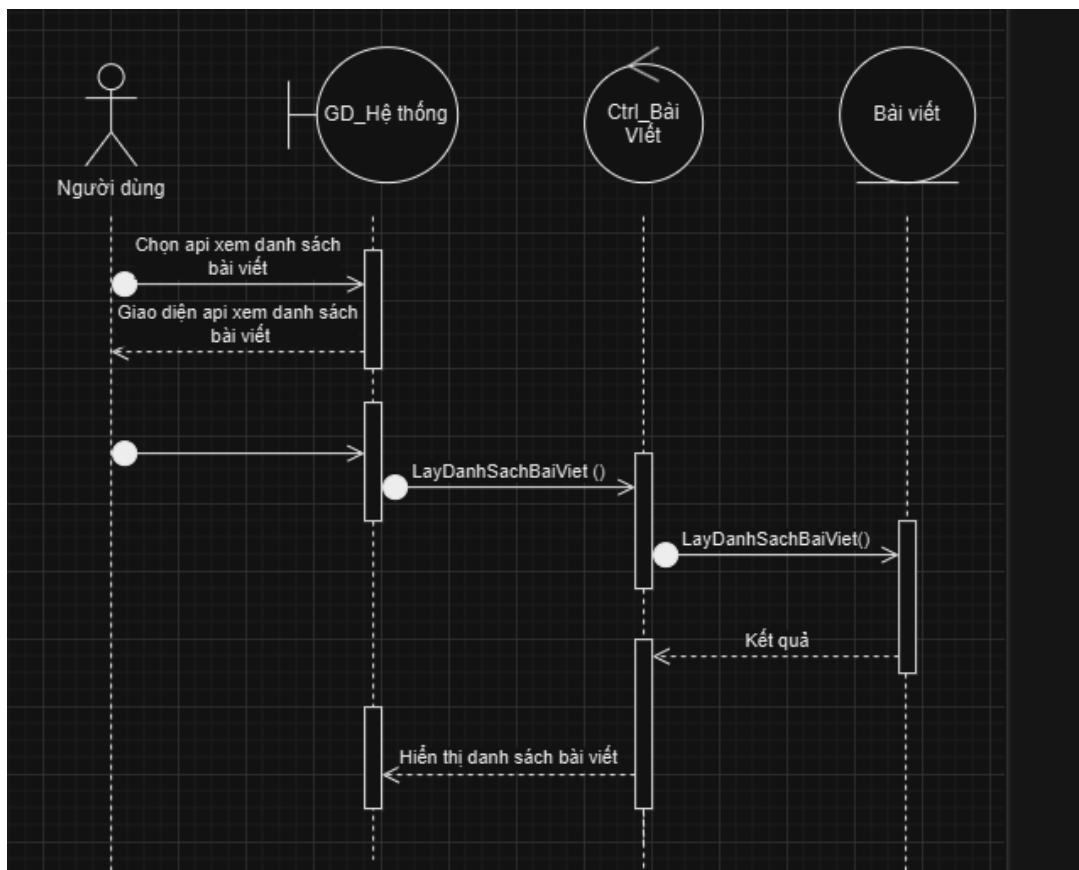
Luồng sự kiện thay thế:

- Không có luồng thay thế đáng kể trong trường hợp này.

Sequence diagram:



Activity diagram:



### e. Usecase Chính sửa bài viết

Tên Use Case: Chính sửa bài viết

Mô tả sơ lược: Người dùng chỉnh sửa nội dung bài viết do chính mình tạo ra.

Actor chính: Người dùng (User).

Actor phụ: Không có.

Tiền điều kiện: Người dùng đã đăng nhập và có quyền chỉnh sửa bài viết.

Hậu điều kiện: Nội dung bài viết được cập nhật trong cơ sở dữ liệu.

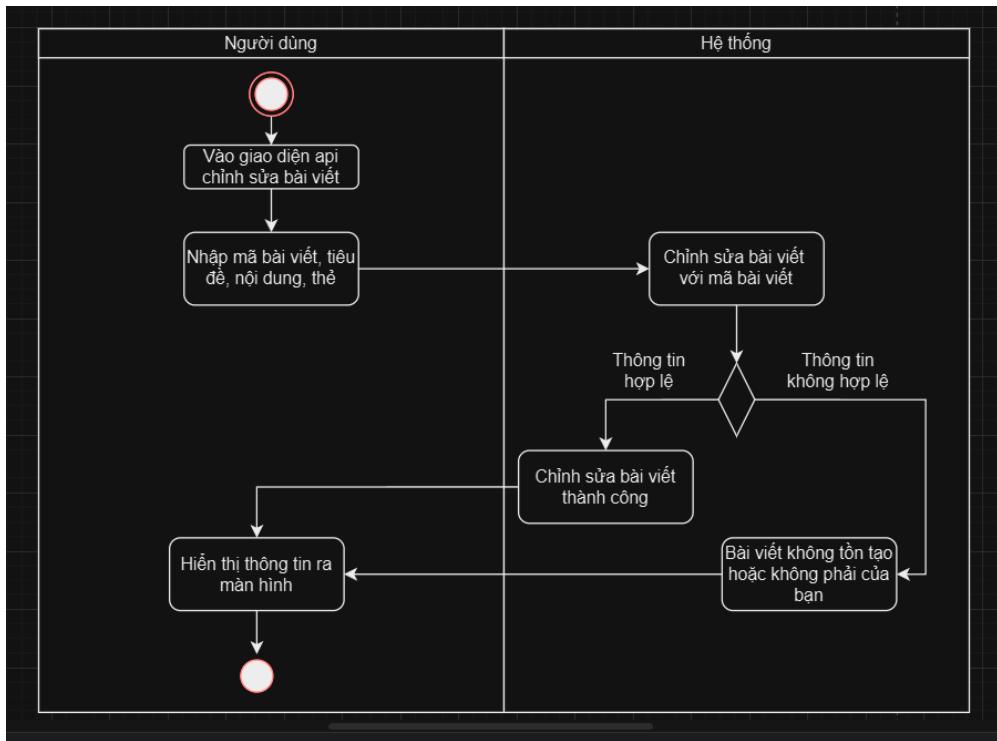
Luồng sự kiện chính:

- Người dùng chọn bài viết cần chỉnh sửa và nhập thông tin mới (title, content).
- Hệ thống kiểm tra quyền chỉnh sửa (người dùng là tác giả bài viết).
- Hệ thống cập nhật bài viết trong cơ sở dữ liệu.
- Hệ thống trả về thông báo "Chỉnh sửa thành công".

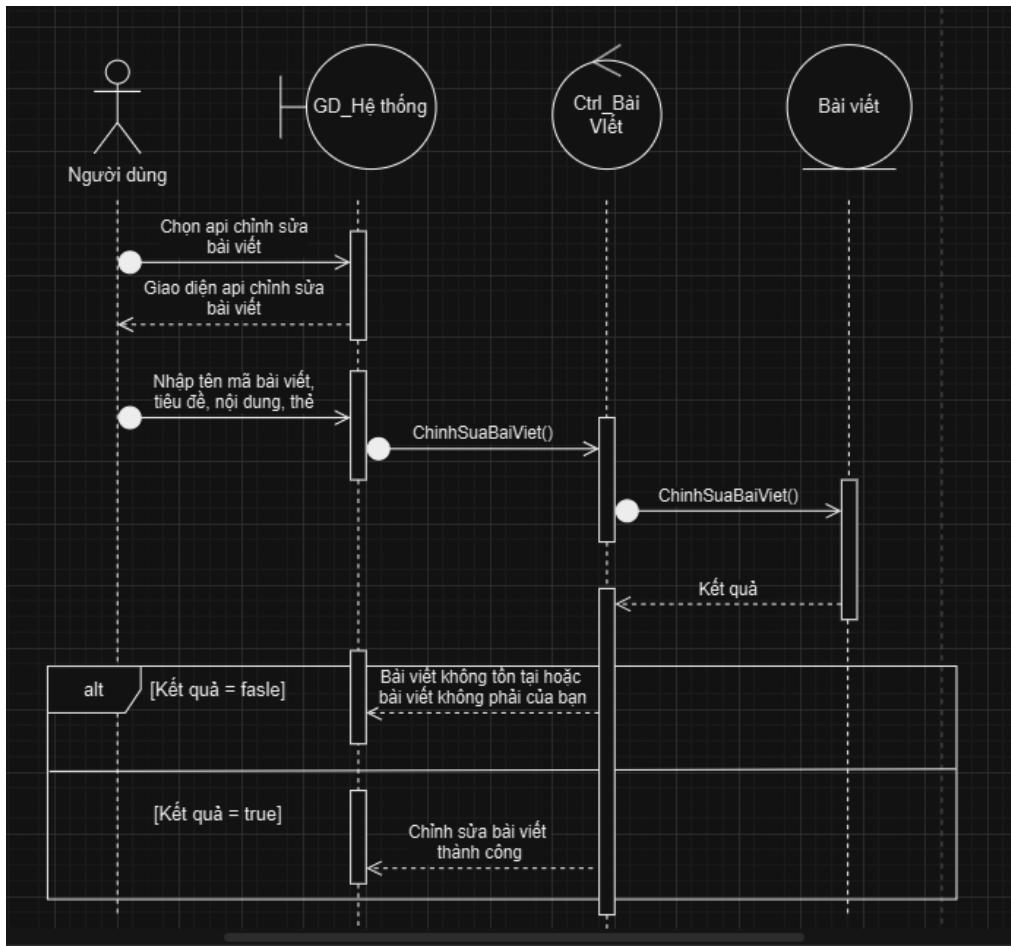
Luồng sự kiện thay thế:

- Nếu người dùng không có quyền chỉnh sửa, hệ thống trả về thông báo lỗi "Bạn không có quyền chỉnh sửa bài viết này".

Sequence diagram:



Activity diagram:



## f. Usecase Xóa bài viết

Tên Use Case: Xóa bài viết

Mô tả sơ lược: Người dùng xóa bài viết do chính mình tạo ra.

Actor chính: Người dùng (User).

Actor phụ: Không có.

Tiền điều kiện: Người dùng đã đăng nhập và có quyền xóa bài viết.

Hậu điều kiện: Bài viết bị xóa khỏi cơ sở dữ liệu.

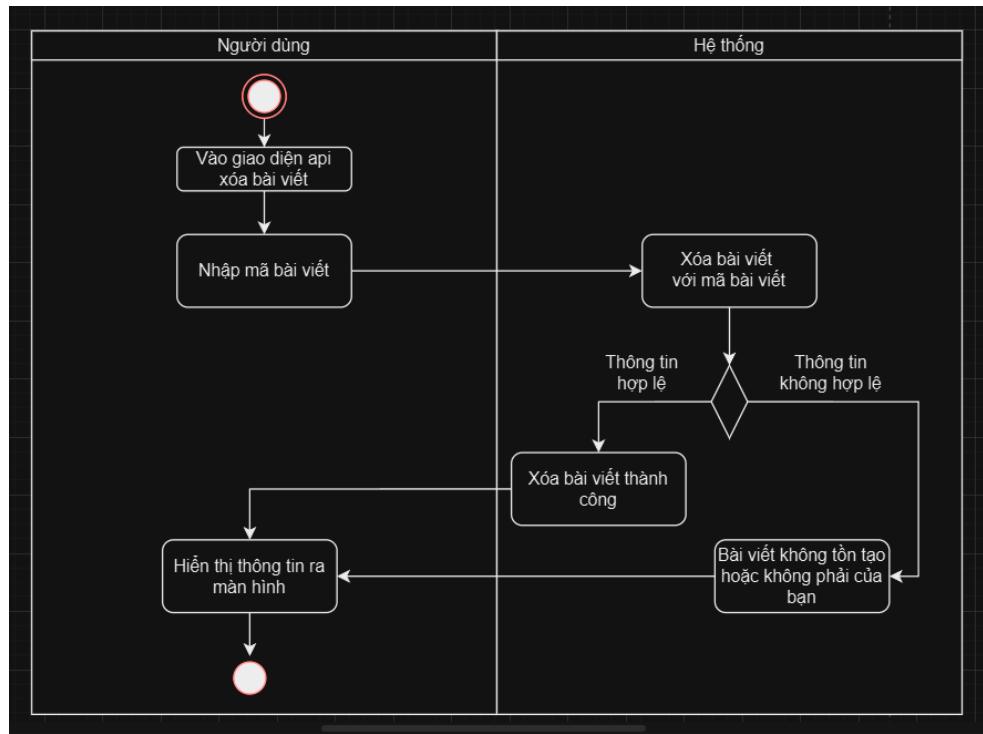
Luồng sự kiện chính:

- Người dùng chọn bài viết cần xóa.
- Hệ thống kiểm tra quyền xóa (người dùng là tác giả bài viết).
- Hệ thống xóa bài viết khỏi cơ sở dữ liệu.
- Hệ thống trả về thông báo "Xóa bài viết thành công".

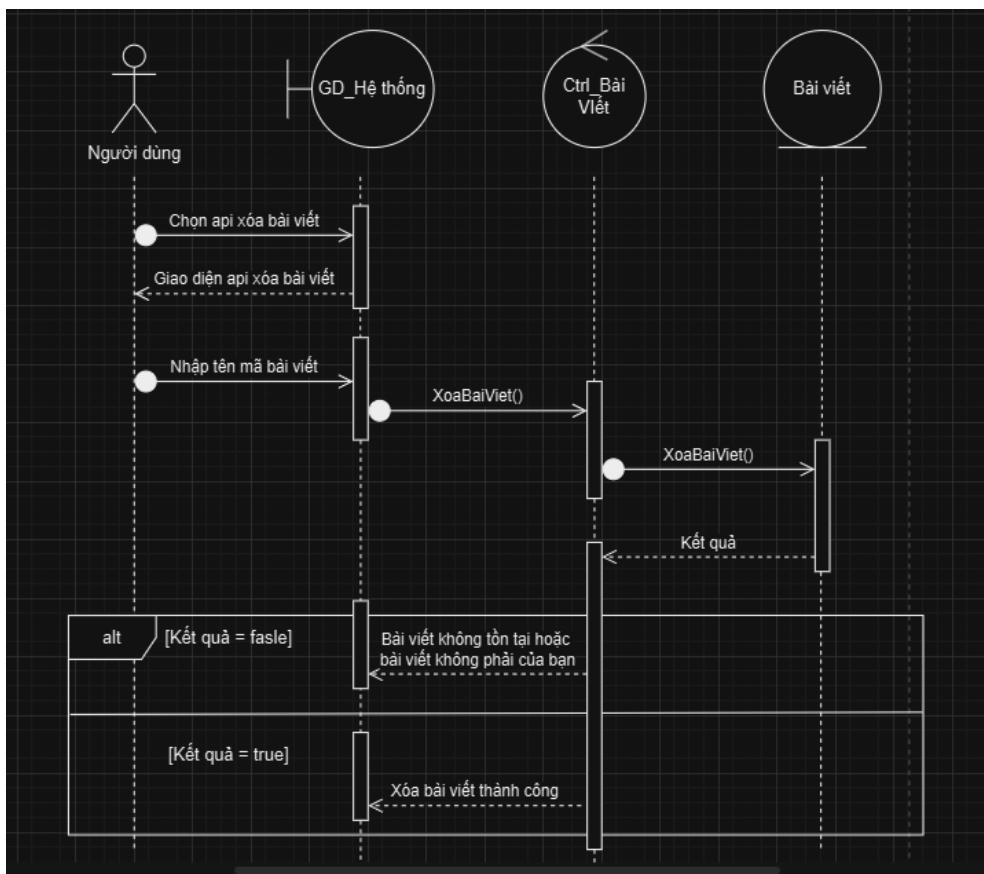
Luồng sự kiện thay thế:

- Nếu người dùng không có quyền xóa, hệ thống trả về thông báo lỗi "Bạn không có quyền xóa bài viết này".

Sequence diagram:



Activity diagram:



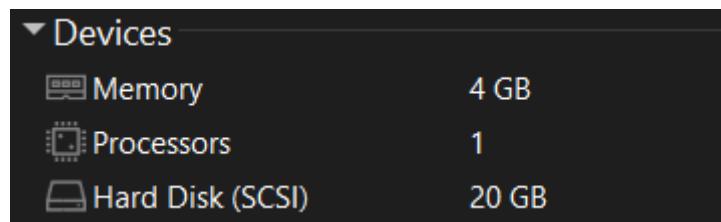
## 2. Triển khai CI/CD

Phần này sẽ xây dựng hệ thống CICD từ đầu, với các công cụ để triển khai trên một máy chủ duy nhất, mở rộng ra nhiều máy chủ sau này. Các công cụ được sử dụng GitLab, Jenkins, Docker,....

### 2.1 Chuẩn bị máy chủ để triển khai dự án

Máy chủ cho dự án là máy chủ Ubuntu 20.04, Ubuntu là một hệ điều hành mã nguồn mở dựa trên Linux, phổ biến và được tối ưu hóa cho các môi trường máy chủ. Máy chủ Ubuntu (Ubuntu Server) được thiết kế để cung cấp hiệu năng cao, ổn định và bảo mật, phù hợp cho các ứng dụng doanh nghiệp, trang web, cơ sở dữ liệu, và điện toán đám mây.

Một số thông số về Memory, Processors, Hard Disk của máy chủ:



SSH máy tính cá nhân vào máy chủ:

```
PS D:\Code> ssh ducanh@192.168.91.110
ducanh@192.168.91.110's password:
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.4.0-200-generic x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/advantage

 System information as of Sun 08 Dec 2024 04:47:38 AM UTC

 System load: 0.49      Processes:           246
 Usage of /: 50.9% of 17.77GB   Users logged in:     0
 Memory usage: 9%          IPv4 address for docker0: 172.17.0.1
 Swap usage:  0%          IPv4 address for ens33: 192.168.91.110

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
 just raised the bar for easy, resilient and secure K8s cluster deployment.
 https://ubuntu.com/engage/secure-kubernetes-at-the-edge

 * Introducing Expanded Security Maintenance for Applications.
 Receive updates to over 25,000 software packages with your
 Ubuntu Pro subscription. Free for personal use.
 https://ubuntu.com/pro

 Expanded Security Maintenance for Applications is not enabled.

 0 updates can be applied immediately.

 Enable ESM Apps to receive additional future security updates.
 See https://ubuntu.com/esm or run: sudo pro status

 The list of available updates is more than a week old.
 To check for new updates run: sudo apt update
 New release '22.04.5 LTS' available.
 Run 'do-release-upgrade' to upgrade to it.

 Last login: Sun Dec  8 03:14:25 2024 from 192.168.91.1
 ducanh@devserver:~$
```

Kiểm tra các thông số qua các câu lệnh top, htop, free -m:



```
ducanh@devserver:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:       3888         299       3049           1        540       3352
Swap:      3900          0       3900
```

Cài đặt Docker trên máy chủ triển khai dữ án:

```
sudo apt update
sudo apt install -y docker.io
sudo systemctl start docker
sudo systemctl enable docker
```

```
ducanh@devserver:~$ docker --version
Docker version 27.3.1, build ce12230
```

Tạo thư mục triển khai dự án posts và kiểm tra:

```
root@devserver:~# sudo mkdir -p /home/posts
root@devserver:~# cd /home
root@devserver:/home# ls
docker-app  ducanh  gitlab-runner  jenkins  posts
```

Phân quyền thư mục, đảm bảo người dùng chỉ định được quản lý thư mục:

```
root@devserver:/home# sudo chown -R $USER:$USER /home/posts
root@devserver:/home# sudo chmod -R 755 /home/posts
root@devserver:/home# |
```

Tạo mới người dùng cho dự án trên máy chủ, phân quyền người dùng cho thư mục đó, đảm bảo tính bảo mật, mỗi dữ án một thư mục, một người dùng:

```

root@devserver:/home# adduser posts
Adding user `posts' ...
Adding new group `posts' (1002) ...
Adding new user `posts' (1002) with group `posts' ...
The home directory `/home/posts' already exists. Not copying from `/etc/skel'.
adduser: Warning: The home directory `/home/posts' does not belong to the user you are currently creating.
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for posts
Enter the new value, or press ENTER for the default
    Full Name []: posts
    Room Number []:
    Work Phone []:
    Home Phone []:
    Other []:
chfn: name with non-ASCII characters: 'pests'
Is the information correct? [Y/n] Y
root@devserver:/home# sudo usermod -aG www-data posts
root@devserver:/home# sudo chown -R ten_nguoi_dung:www-data /home/posts
chown: invalid user: 'ten_nguoi_dung:www-data'
root@devserver:/home# sudo chown -R posts:www-data /home/posts
root@devserver:/home# |

```

Kiểm tra người dùng đó đã được phân quyền cho thư mục chua:

```

root@devserver:/home# ls -ld /home/posts
drwxr-xr-x 2 posts www-data 4096 Dec  8 04:56 /home/posts
root@devserver:/home# |

```

=> Vậy là các bước cơ bản để chuẩn bị máy chủ Ubuntu nhằm triển khai một dự án đã hoàn tất. Từ việc cài đặt hệ điều hành, cập nhật gói phần mềm, thiết lập môi trường mạng, bảo mật hệ thống, đến quản lý người dùng và phân quyền cho dự án tất cả đều được thiết lập để đảm bảo rằng máy chủ sẵn sàng hoạt động ổn định và an toàn.

Bước tiếp theo trong triển khai: Trong thực tế, đối với các dự án lớn hoặc hệ thống cần mở rộng, việc sử dụng một máy chủ duy nhất có thể không đáp ứng được các yêu cầu về hiệu suất, khả năng chịu tải, hoặc tính sẵn sàng cao. Các giải pháp nâng cao như quản lý cụm máy chủ (cluster) hoặc triển khai trên nền tảng Kubernetes (K8s) thường được áp dụng. Tuy nhiên, trong phạm vi này, chúng ta tập trung vào các bước cơ bản để khởi tạo một máy chủ duy nhất.

Các vấn đề nâng cao không được đề cập: Dưới đây là một số khái niệm nâng cao không được đề cập nhưng rất quan trọng trong triển khai thực tế:

- Cụm máy chủ (Server Clusters):

- + Dùng để kết hợp nhiều máy chủ vật lý hoặc ảo lại với nhau nhằm đảm bảo khả năng mở rộng và tính sẵn sàng cao.

- + Các mô hình cụm phổ biến: Load Balancing Cluster: Phân tải truy cập giữa nhiều máy chủ. High-Availability Cluster (HA): Đảm bảo dịch vụ luôn sẵn sàng kể cả khi một máy chủ gặp sự cố.
- + Công nghệ hỗ trợ: HAProxy, Nginx Load Balancer, Elastic Load Balancer (AWS).
  - Triển khai Kubernetes (K8s): Một nền tảng mạnh mẽ để quản lý các container (Docker) trong môi trường cụm.
- + Tự động mở rộng tài nguyên (Auto-scaling).
- + Quản lý trạng thái ứng dụng (StatefulSet, Deployments).
- + Cân bằng tải (Load Balancing) và khám phá dịch vụ (Service Discovery).
- + Ứng dụng thường triển khai trong môi trường Kubernetes Cluster trên các dịch vụ như Google Kubernetes Engine (GKE), Amazon EKS, hoặc Azure AKS.
  - CI/CD và Quản lý cấu hình: Sử dụng các công cụ như Jenkins, GitLab CI/CD, hoặc GitHub Actions để tự động hóa quy trình triển khai. Kết hợp quản lý cấu hình bằng Ansible, Terraform, hoặc Chef để tự động thiết lập và duy trì hệ thống.
  - Giám sát và Logging: Để theo dõi hiệu suất, ghi lại nhật ký lỗi và tối ưu hóa hệ thống, các công cụ như Prometheus, Grafana, hoặc ELK Stack (Elasticsearch, Logstash, Kibana) rất quan trọng.

=> Tóm lại, trong khi một máy chủ đơn giản có thể đáp ứng tốt cho các dự án nhỏ hoặc giai đoạn khởi đầu, thì các hệ thống lớn hơn cần các giải pháp nâng cao để đảm bảo hiệu suất, khả năng mở rộng, và tính ổn định. Những khía cạnh này, tuy không được đề cập ở đây, sẽ rất hữu ích khi bạn tiến sâu vào việc xây dựng hệ thống phân tán hoặc triển khai ở quy mô lớn hơn.

## 2.2 Chuẩn bị máy chủ GitLab lưu trữ mã nguồn của dự án

Một số thông số về Memory, Processors, Hard Disk của máy chủ, máy chủ GitLab với quy mô doanh nghiệp thường từ 8/16 GB, Hard Disk có thể từ 80GB đến 200GB, ở ví dụ này với quy mô nhỏ:

▼ Devices	
Memory	4 GB
Processors	1
Hard Disk (SCSI)	20 GB

SSH máy tính cá nhân vào máy chủ:

```
PS D:\Code> ssh ducanh@192.168.91.100
ducanh@192.168.91.100's password:
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.4.0-200-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

 System information as of Sun 08 Dec 2024 07:37:47 AM UTC

 System load:  0.57           Processes:          284
 Usage of /:   67.1% of 17.77GB  Users logged in:   1
 Memory usage: 83%            IPv4 address for ens33: 192.168.91.100
 Swap usage:   6%

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
 just raised the bar for easy, resilient and secure K8s cluster deployment.

 https://ubuntu.com/engage/secure-kubernetes-at-the-edge

 * Introducing Expanded Security Maintenance for Applications.
 Receive updates to over 25,000 software packages with your
 Ubuntu Pro subscription. Free for personal use.

 https://ubuntu.com/pro

 Expanded Security Maintenance for Applications is not enabled.

 0 updates can be applied immediately.

 Enable ESM Apps to receive additional future security updates.
 See https://ubuntu.com/esm or run: sudo pro status

 The list of available updates is more than a week old.
 To check for new updates run: sudo apt update
 New release '22.04.5 LTS' available.
 Run 'do-release-upgrade' to upgrade to it.

Last login: Sun Dec  8 03:14:34 2024 from 192.168.91.1
ducanh@gitlabserver:~$ |
```

Tiến hành cài đặt GitLab và các tệp hỗ trợ cần thiết

```
curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-ee/script.deb.sh
```

```
sudo apt install gitlab-ee -y
```

Sau khi cài đặt thành công GitLab, tiến hành thêm host cho máy chủ để có thể tiến hành truy cập với mạng Lan. Vì sao phải thêm host ? Việc thêm hostname vào GitLab giúp dễ dàng truy cập qua tên miền thay vì địa chỉ IP, hỗ trợ cấu hình SSL/TLS để sử dụng HTTPS, và giúp quản lý các dịch vụ trong môi trường mạng lớn hoặc phân tán. Hostname cũng tạo điều kiện thuận lợi cho việc mở rộng hệ thống, bảo trì dễ dàng, và đảm bảo sự tương thích với các công cụ như reverse proxy, load balancer, và GitLab CI/CD. Nó cũng giúp tăng cường bảo mật và khả

năng quản lý khi triển khai GitLab trong môi trường sản xuất hoặc container hóa.  
Chạy câu lệnh

```
sudo nano /etc/gitlab/gitlab.rb
```

Sửa file:

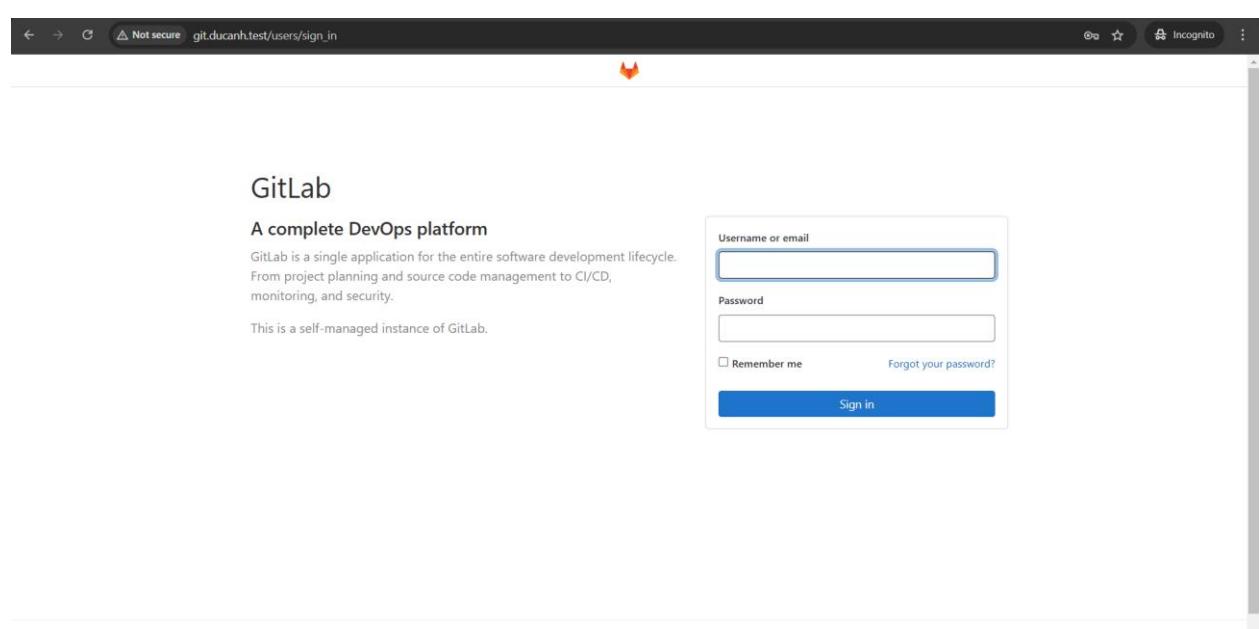
```
##! Note: During installation/upgrades, the value of the environment variable
##! EXTERNAL_URL will be used to populate/replace this value.
##! On AWS EC2 instances, we also attempt to fetch the public hostname/IP
##! address from AWS. For more details, see:
##! https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instancedata-data-retrieval.html
external_url 'http://git.ducanh.test'
puma['worker_processes'] = 2 # Số worker
puma['worker_timeout'] = 60 # Thời gian timeout
puma['memory_limit'] = 3000 # Giới hạn bộ nhớ (MB), nếu có
## Roles for multi-instance GitLab
##! The default is to have no roles enabled, which results in GitLab running as an all-in-one instance.
##! Options:
##!   redis_sentinel_role redis_master_role redis_replica_role geo_primary_role geo_secondary_role
##!   postgres_role consul_role application_role monitoring_role
##! For more details on each role, see:
##! https://docs.gitlab.com/omnibus/roles/README.html#roles
##!
# roles ['redis_sentinel_role', 'redis_master_role']
```

Cập nhật file hosts của máy chủ, thêm một dòng gồm địa chỉ ip và tên hostname

```
sudo nano /etc/hosts
```

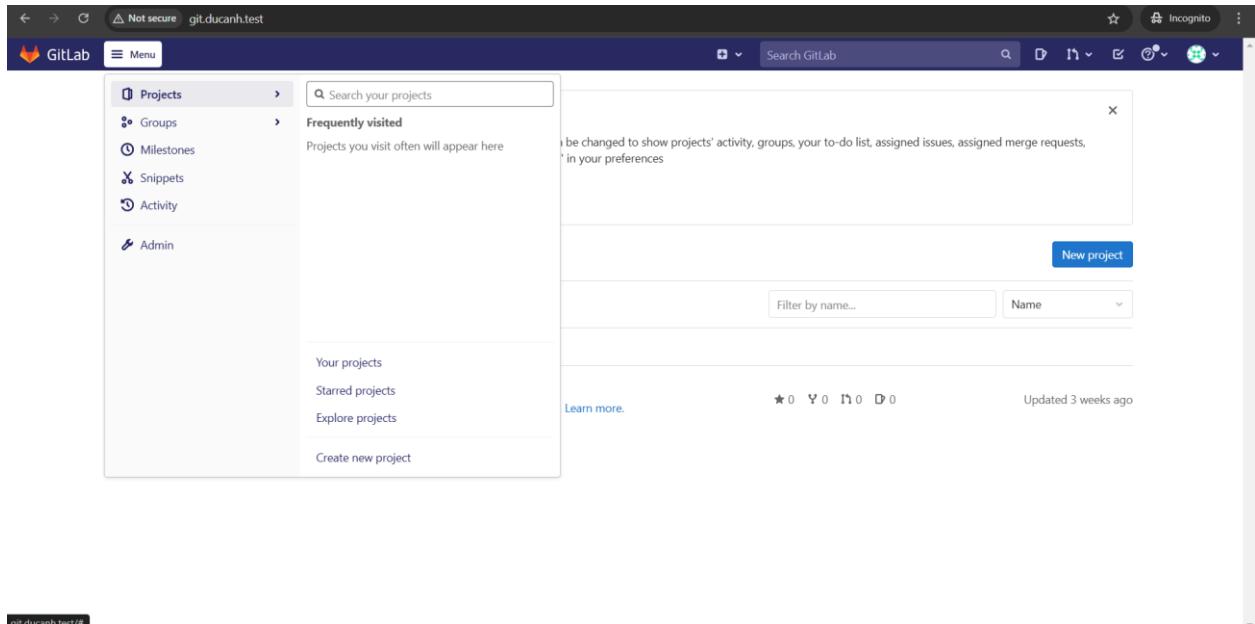
```
<IP-ADDRESS> gitlab.yourdomain.com
```

Với url, truy cập trang chủ quản lý của GitLab:



Đăng nhập người dùng root của dự án, đây cũng là tài khoản admin, dành cho quản lý dự án cấp cao nhất.

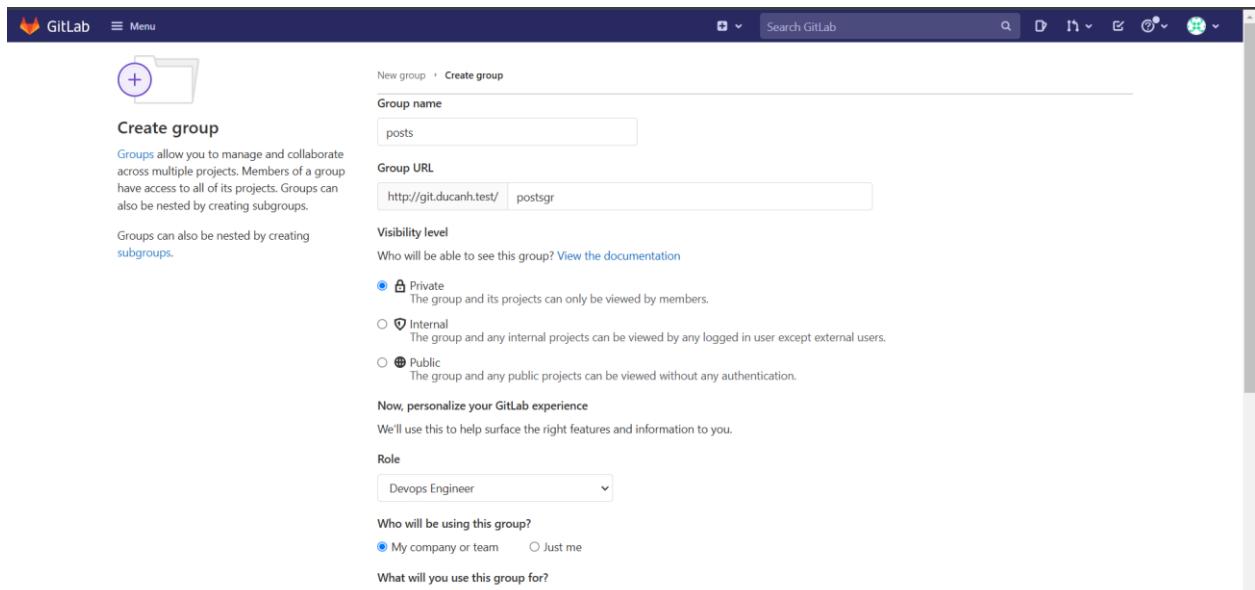
Ta sẽ tiến hành đăng nhập với vị trí devops engineer (kỹ sư phát triển và vận hành):



The screenshot shows the GitLab homepage at `git.ducanh.test`. The top navigation bar includes links for Groups, Milestones, Snippets, Activity, Admin, and a search bar. A sidebar on the left provides access to 'Your projects', 'Starred projects', 'Explore projects', and 'Create new project'. The main content area displays a 'Frequently visited' section with a note about changing preferences for project activity, a 'New project' button, and filters for 'Name'. At the bottom, there are statistics for stars (0), forks (0), issues (0), and merge requests (0), along with an 'Updated 3 weeks ago' message.

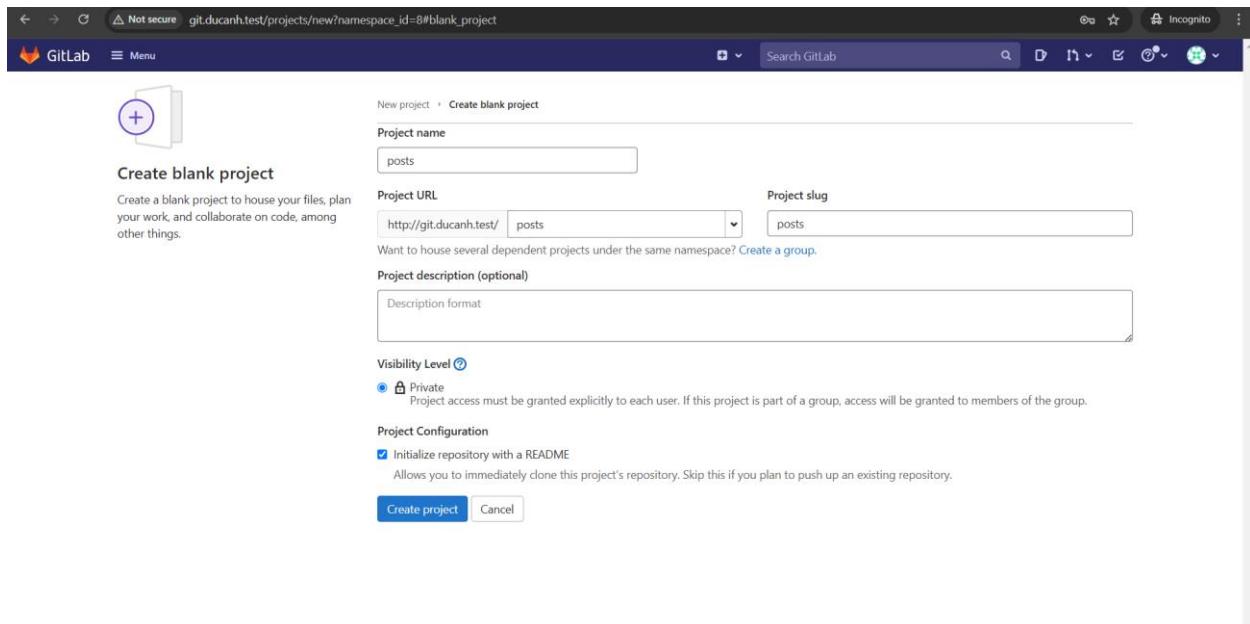
Trong màn hình chính của GitLab, bây giờ, ta tiến hành tạo ra nhóm phát triển dự án, ta quy ước mỗi nhóm đảm nhiệm một nghiệp vụ, một service hay một dữ án.

Với các dự án có quy mô lớn như microservice, trong một dự án lớn, gồm nhiều nhóm, mỗi nhóm sẽ đảm nhiệm một dự án riêng.



The screenshot shows the 'Create group' form. It includes fields for 'Group name' (set to 'posts'), 'Group URL' (set to `http://git.ducanh.test/postsgroup`), and 'Visibility level' (set to 'Private'). Below these, there's a section for personalizing the experience ('Now, personalize your GitLab experience') and a 'Role' dropdown set to 'Devops Engineer'. At the bottom, there's a question 'Who will be using this group?' with options 'My company or team' (selected) and 'Just me'.

Sau khi tạo xong nhóm, ta tiếp hành tạo project:



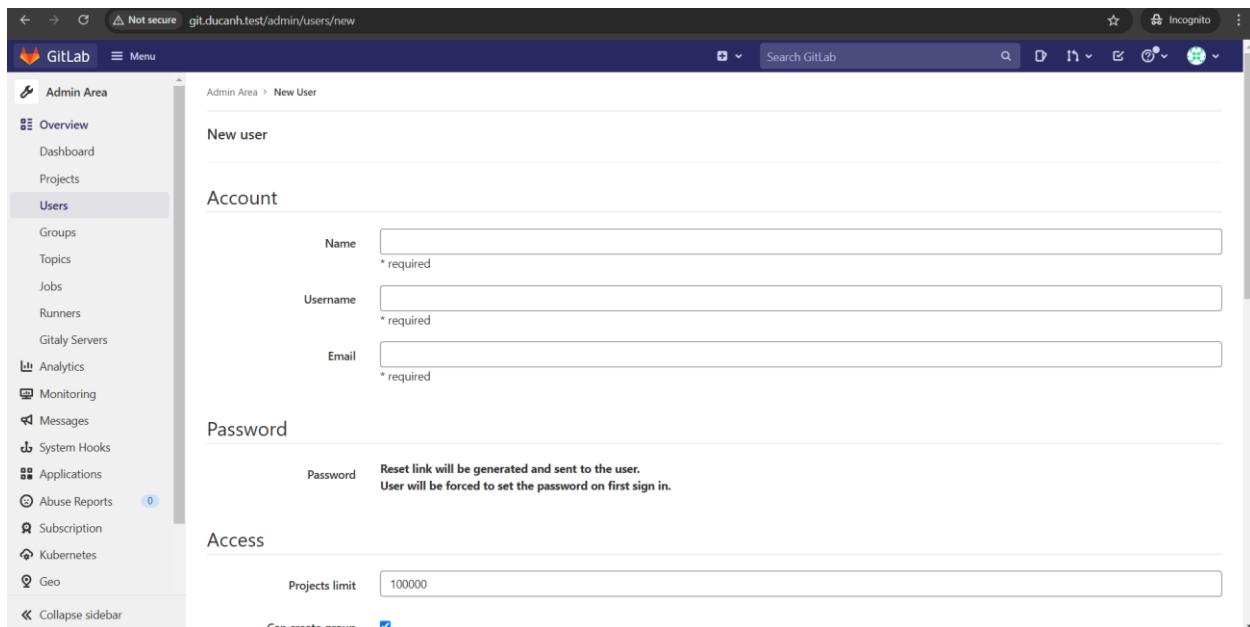
The screenshot shows the 'Create blank project' page on GitLab. The URL is `git.ducanh.test/projects/new?namespace_id=8#blank_project`. The form fields include:

- Project name:** posts
- Project URL:** `http://git.ducanh.test/` (with 'posts' selected)
- Project slug:** posts
- Description (optional):** (empty)
- Visibility Level:** Private (selected)
- Project Configuration:** Initialize repository with a README (checked)

At the bottom are 'Create project' and 'Cancel' buttons.

Sau khi tạo xong project posts, ta tiếp hành thêm các developer (lập trình viên), project manager (quản lý dự án).

Với quy mô dự án nhỏ, ta sẽ thêm 2 lập trình viên xây dựng 2 chức năng, 1 quản lý dự án.



The screenshot shows the 'New user' form in the GitLab Admin Area. The URL is `git.ducanh.test/admin/users/new`. The left sidebar shows 'Admin Area' with 'Users' selected. The form fields include:

- Account:**
  - Name: [empty]
  - Username: [empty]
  - Email: [empty]
- Password:** Password [empty] Reset link will be generated and sent to the user. User will be forced to set the password on first sign in.
- Access:**
  - Projects limit: 100000
  - Can create group: checked

Với màn hình tạo người dùng, ta tạo lần lượt các người dùng trên và cho vào dự án:

Admin Area > Users

Users Cohorts

Active 8 Admins 3 2FA Enabled 0 2FA Disabled 8 External 0 Blocked 0 Pending approval 0 Deactivated 0 Without projects 5 New user

Name	Projects	Groups	Created on	Last activity
posts-develop-devops posts-develop-devops@git.ducanh.test	0	0	8 Dec, 2024	Never
posts-develop-admin posts-develop-admin@git.ducanh.test	0	0	8 Dec, 2024	Never
posts-develop-post posts-develop-post@git.ducanh.test	0	0	8 Dec, 2024	Never
posts-develop-user posts-develop-user@git.ducanh.test	0	0	8 Dec, 2024	Never

P posts

Project information

Activity Labels Members Repository Issues Merge requests CI/CD Security & Compliance Deployments Monitor Infrastructure Packages & Registries Analytics Wiki Snippets Settings

Members

Administrator @root

Members >

Filter members Account

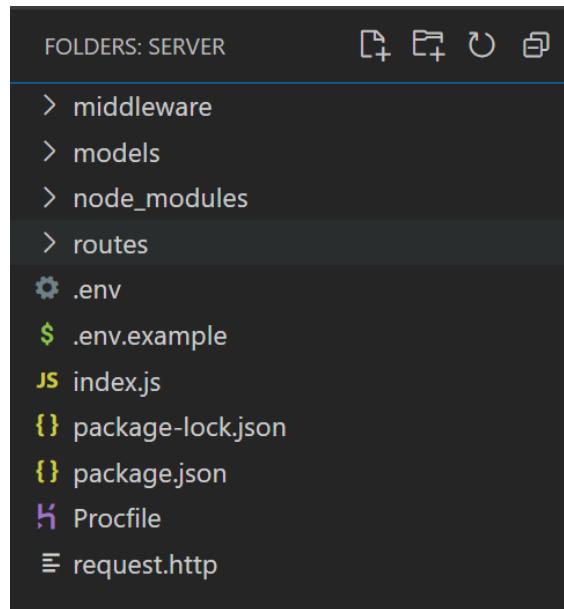
Account	Source	Access granted	Max role	Expiration
Administrator @root	posts	26 minutes ago	Owner	Expiration date
posts-develop-admin @posts-develop-admin	Direct member	27 seconds ago by Administrator	Maintainer	Expiration date
posts-develop-devops @posts-develop-devops	Direct member	just now by Administrator	Maintainer	Expiration date
posts-develop-post @posts-develop-post	Direct member	12 seconds ago by Administrator	Developer	Expiration date
posts-develop-user @posts-develop-user	Direct member	12 seconds ago by Administrator	Developer	Expiration date

Tiếp theo, admin (người quản trị của dự án) sẽ phải tạo ra mã nguồn cơ bản của dự án, các thiết lập ban đầu.

Ta tiếp tục công việc với vị trí người quản trị dự án.

### 2.3 Thiết lập mã nguồn ban đầu của dự án và các môi trường phát triển

Dự án được xây dựng với nền tảng NodeJs, ứng dụng api sử dụng cơ sở dữ liệu MongoDB.



Tại máy của người quản trị, tiến hành sử dụng các câu lệnh để kết nối với GitLab:

```
● PS D:\Code\TaskManagement\server> git config --global user.name "posts-develop-admin"
● PS D:\Code\TaskManagement\server> git config --global user.email "posts-develop-admin@git.ducanh.test"
✖ PS D:\Code\TaskManagement\server> git remote add origin http://git.ducanh.test/posts/posts.git
error: remote origin already exists.
```

Sử dụng các thao tác git add, git push, git commit để có thể đưa mã nguồn lên GitLab:

Name	Last commit	Last update
middleware	initial core	3 minutes ago
models	initial core	3 minutes ago
routes	initial core	3 minutes ago
.env.example	initial core	3 minutes ago
.gitignore	initial core	3 minutes ago
Procfile	initial core	51 minutes ago
index.js	initial core	3 minutes ago
package-lock.json	initial core	3 minutes ago

Tiếp theo, ta tạo ra các nhánh phát triển cho dự án: Các nhánh phát triển mong muốn:

- + dev (dành cho các lập trình viên tích hợp)
- + staging (kiểm thử phần mềm)
- + main (nhánh dành cho môi trường người dùng)
- + feature- (các nhánh chức năng của dự án)

Ở dự án này, ta sẽ xây dựng các nhánh bao gồm main, staging, dev và feature-user (chức năng người dùng), feature-post (chức năng bài viết):

The screenshot shows the GitLab interface for a repository named 'posts'. The sidebar on the left is collapsed. The main area displays the 'Branches' page with the following details:

- Active branches:**
  - dev**: Initial core commit by '4a8b36fb' 7 minutes ago. Buttons: Merge request, Compare, Delete.
  - feature-post**: Initial core commit by '4a8b36fb' 7 minutes ago. Buttons: Merge request, Compare, Delete.
  - feature-user**: Initial core commit by '4a8b36fb' 7 minutes ago. Buttons: Merge request, Compare, Delete.
  - main (default)**: Initial core commit by '4a8b36fb' 7 minutes ago. Buttons: Merge request, Compare, Delete.
  - staging**: Initial core commit by '4a8b36fb' 7 minutes ago. Buttons: Merge request, Compare, Delete.
- Protected branches can be managed in project settings.**

Sau khi tạo xong các nhánh ta tiến hành bảo vệ các nhánh, đây là bước quan trọng, giúp phân quyền giữa lập trình viên và quản trị trong dự án.

Trên nhánh dev, chỉ có người quản trị được merge hoặc push mã nguồn.

Branch	Allowed to merge	Allowed to push	Allowed to force push
dev (default)	Maintainers	Maintainers	<input checked="" type="checkbox"/> Unprotect
feature-post	Developers + Mai...	Developers + Mai...	<input checked="" type="checkbox"/> Unprotect
feature-user	Developers + Mai...	Developers + Mai...	<input checked="" type="checkbox"/> Unprotect
main	Maintainers	No one	<input checked="" type="checkbox"/> Unprotect
staging	Maintainers	No one	<input checked="" type="checkbox"/> Unprotect

Phần tiếp theo, ta sẽ chuẩn bị máy chủ Jenkins, quay trở lại với vị trí devops engineer (kỹ sư phát triển và vận hành)

## 2.4 Chuẩn bị máy chủ Jenkins để thực thi quy trình CI/CD

Một số thông số về Memory, Processors, Hard Disk của máy chủ, máy chủ Jenkins với quy mô doanh nghiệp thường từ 4/8 GB, Hard Disk có thể từ 20GB đến 50GB, ở ví dụ này với quy mô nhỏ:

▼ Devices	
Memory	4 GB
Processors	1
Hard Disk (SCSI)	20 GB

SSH máy tính cá nhân vào máy chủ:

```
PS D:\Code> ssh ducanh@192.168.91.130
ducanh@192.168.91.130's password:
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.4.0-200-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

System information as of Sun 08 Dec 2024 10:39:11 AM UTC

System load: 0.12           Processes:          215
Usage of /:   46.4% of 17.77GB  Users logged in:    1
Memory usage: 41%            IPv4 address for ens33: 192.168.91.130
Swap usage:   0%

* Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
just raised the bar for easy, resilient and secure K8s cluster deployment.

https://ubuntu.com/engage/secure-kubernetes-at-the-edge

Expanded Security Maintenance for Applications is not enabled.

2 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

New release '22.04.5 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Sun Dec  8 03:14:47 2024 from 192.168.91.1
ducanh@jenkinsserver:~$ |
```

Cài đặt java 11 và Jenkins:

```
sudo apt install openjdk-11-jdk -y
```

```
sudo apt install wget
wget -q -O - https://pkg.jenkins.io/keys/jenkins.io.key | sudo tee /etc/apt/trusted.gpg
sudo sh -c 'wget -q -O /etc/apt/sources.list.d/jenkins.list https://pkg.jenkins.io/deb/stable/jenkins.list'
```

```
sudo apt update
sudo apt install jenkins -y
```

Các bước thêm host và tạo người dùng tương tự, ta có một máy chủ Jenkins, dưới đây là giao diện chính của Jenkins:

The screenshot shows the Jenkins dashboard with the following details:

- Jobs:** Action in devserver, test
- Status:** Last Success: N/A, Last Failure: N/A, Last Duration: N/A
- Build Queue:** No builds in the queue.
- Build Executor Status:**
  - Built-In Node: 0/2
  - dev-server: offline

Đầu tiên, ta cần kết nối Jenkins với các server triển khai dự án. Và để kết nối, ta sẽ sử dụng Jenkins Agent, Jenkins Agent (trước đây gọi là Slave) là một máy chủ phụ trợ giúp mở rộng khả năng xử lý của Jenkins Master. Trong một môi trường có nhiều job hoặc yêu cầu tài nguyên cao, việc sử dụng Agents giúp phân tán các công việc và tăng tốc quá trình build và deploy. Jenkins Master quản lý cấu hình và điều phối các job, trong khi các Agent chịu trách nhiệm thực thi các công việc đó. Các Agent có thể chạy trên máy chủ hoặc container riêng biệt và giúp giảm tải cho Master, đồng thời cung cấp khả năng chạy nhiều job đồng thời. Chúng cũng hỗ trợ nhiều nền tảng khác nhau, từ đó giúp Jenkins đáp ứng các yêu cầu build đa dạng. Kết nối giữa Master và Agent có thể thực hiện qua các giao thức như SSH, Jenkins remoting, Swarm hoặc Kubernetes. Việc sử dụng Jenkins Agent

không chỉ giúp cải thiện hiệu suất mà còn mở rộng khả năng của Jenkins trong việc xử lý các pipeline phức tạp.

Và để cài đặt Jenkins Agent trên các máy chủ triển khai dự án, ta cần cài đặt phiên bản java bằng với phiên bản java trên máy chủ Jenkins.

```
# Cập nhật danh sách các gói
sudo apt update

# Cài đặt Java 17
sudo apt install openjdk-17-jdk -y

# Kiểm tra phiên bản Java đã cài đặt
java -version
```

Trước khi kết nối với máy chủ Jenkins, tại các máy chủ triển khai, ta cần thêm người dùng Jenkins với mong muốn: các thao tác liên quan đến CI/CD chỉ có người dùng Jenkins sử dụng, thao tác nhằm tăng tính bảo mật cho dự án.

Quay lại với giao diện máy chủ Jenkins, trong phần Manage Jenkins, Nodes. Đây là giao diện hiển thị các kết nối với các máy chủ sẽ triển khai dự án, bao nhiêu máy chủ từng đó kết nối. Các kết nối này sẽ được Jenkins quản lý để kích hoạt các thao tác CI/CD.

The screenshot shows the Jenkins 'Nodes' management page. At the top, there's a navigation bar with links for 'Dashboard', 'Manage Jenkins', and 'Nodes'. Below the navigation, there are sections for 'Build Queue' (empty) and 'Build Executor Status' (one 'Built-In Node' online, one 'dev-server' offline). The main 'Nodes' section has a table with the following data:

S	Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
1	Built-In Node	Linux (amd64)	In sync	8.58 GiB	3.81 GiB	8.58 GiB	0ms
2	dev-server		N/A	N/A	N/A	N/A	N/A

At the bottom of the table, it says 'Data obtained' and lists times: 1 min 6 sec for all columns. There are also icons for 'S' (Server), 'M' (Master), and 'L' (Label). On the right side of the table, there are buttons for '+ New Node' and 'Configure Monitors'.

Lựa chọn New Node để tạo một Node tương ứng một kết nối từ máy chủ Jenkins đến các máy chủ triển khai:

New node

Node name

Type

- Permanent Agent
 

Adds a plain, permanent agent to Jenkins. This is called "permanent" because Jenkins doesn't provide higher level of integration with these agents, such as dynamic provisioning. Select this type if no other agent types apply — for example such as when you are adding a physical computer, virtual machines managed outside Jenkins, etc.
- Copy Existing Node

Create

REST API Jenkins 2.485

Sau khi kết nối:

Nodes

S	Name ↓	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	Built-In Node	Linux (amd64)	In sync	8.58 GiB	3.81 GiB	8.58 GiB	0ms
	dev-server		N/A	N/A	N/A	N/A	N/A
	Data obtained	6 min 1 sec	6 min 1 sec	6 min 1 sec	6 min 1 sec	6 min 1 sec	6 min 1 sec

Build Queue: No builds in the queue.

Build Executor Status: Built-In Node (0/2), dev-server (offline).

Icon: S M L

Legend

REST API Jenkins 2.485

Bây giờ ta cần kết nối máy chủ triển khai đến Node này, nhấn vào Node, Jenkins đã có hướng dẫn cụ thể để có thể kết nối:

The screenshot shows the Jenkins interface for managing a slave node named 'dev-server'. On the left, there's a sidebar with options like Status, Delete Agent, Configure, Build History, Load Statistics, Log, and Open Blue Ocean. The main area is titled 'Agent dev-server' and contains sections for 'Run from agent command line: (Unix)' and 'Run from agent command line: (Windows)'. Below these are command examples and a note about running with a secret file. A table at the bottom shows build executor status: 0/1 available, with 'dev-server' listed as offline.

Trên máy chủ triển khai dự án, chuyển qua người dùng Jenkins và chạy các câu lệnh trên dưới nèn:

```
root@devserver:/var/lib/jenkins# java -jar agent.jar -url http://jenkins.ducanh.test/ -secret @secret-file -name "dev-server" -workDir "/var/lib/jenkins"
Dec 08, 2024 11:30:16 AM org.jenkinsci.remoting.engine.WorkDirManager initializeWorkDir
INFO: Using /var/lib/jenkins/remoting as a remoting work directory
Dec 08, 2024 11:30:17 AM org.jenkinsci.remoting.engine.WorkDirManager setupLogging
INFO: Both error and output logs will be printed to /var/lib/jenkins/remoting
Dec 08, 2024 11:30:17 AM hudson.remoting.Launcher createEngine
INFO: Setting up agent: dev-server
Dec 08, 2024 11:30:17 AM hudson.remoting.Engine startEngine
INFO: Using Remoting version: 3283.v92c105e0f819
Dec 08, 2024 11:30:17 AM org.jenkinsci.remoting.engine.WorkDirManager initializeWorkDir
INFO: Using /var/lib/jenkins/remoting as a remoting work directory
Dec 08, 2024 11:30:17 AM hudson.remoting.Launcher$CuiListener status
INFO: Locating server among [http://jenkins.ducanh.test/]
Dec 08, 2024 11:30:17 AM org.jenkinsci.remoting.engine.JnlpAgentEndpointResolver resolve
INFO: Remoting server accepts the following protocols: [JNLP4-connect, Ping]
Dec 08, 2024 11:30:17 AM hudson.remoting.Launcher$CuiListener status
INFO: Agent discovery successful
  Agent address: jenkins.ducanh.test
  Agent port: 8999
  Identity: 69:82:a8:69:bf:77:d3:41:f9:01:ec:6e:f0:9a:8f:a4
Dec 08, 2024 11:30:17 AM hudson.remoting.Launcher$CuiListener status
INFO: Handshaking
Dec 08, 2024 11:30:17 AM hudson.remoting.Launcher$CuiListener status
INFO: Connecting to jenkins.ducanh.test:8999
Dec 08, 2024 11:30:17 AM hudson.remoting.Launcher$CuiListener status
INFO: Server reports protocol JNLP4-connect-proxy not supported, skipping
Dec 08, 2024 11:30:17 AM hudson.remoting.Launcher$CuiListener status
INFO: Trying protocol: JNLP4-connect
Dec 08, 2024 11:30:17 AM org.jenkinsci.remoting.protocol.impl.BIONetworkLayer$Reader run
INFO: Waiting for ProtocolStack to start.
Dec 08, 2024 11:30:18 AM hudson.remoting.Launcher$CuiListener status
INFO: Remote identity confirmed: 69:82:a8:69:bf:77:d3:41:f9:01:ec:6e:f0:9a:8f:a4
Dec 08, 2024 11:30:18 AM hudson.remoting.Launcher$CuiListener status
INFO: Connected
```

Sau khi kết nối thành công máy chủ Jenkins đến máy chủ triển khai dự án:

The screenshot shows the Jenkins 'Nodes' page. On the left, there are sections for 'Build Queue' (empty) and 'Build Executor Status' (two entries: 'Built-In Node' and 'dev-server'). The main area displays a table of nodes:

S	Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	Built-In Node	Linux (amd64)	In sync	8.58 GiB	3.81 GiB	8.58 GiB	0ms
	dev-server	Linux (amd64)	In sync	7.79 GiB	3.81 GiB	7.79 GiB	127ms

Icons for sorting (S), medium (M), and large (L) are at the bottom of the table. A legend is located on the right.

Tiếp theo, ta cần kết nối máy chủ Jenkins đến máy chủ GitLab, và để kết nối với máy chủ Gitlab, Jenkins hỗ trợ nhiều công cụ, trong đó có công cụ kết nối với máy chủ GitLab, tương tự trong phần Manage Jenkins, ta chọn Plugins, ta thêm Plugin Gitlab:

The screenshot shows the Jenkins 'Plugins' page. The 'Installed plugins' section is selected. A search bar at the top right shows 'Git lab'. A single result is listed:

Name	Enabled
GitLab Plugin 1.9.6	<input checked="" type="checkbox"/> <span style="color: red;">X</span>

A note below the plugin says: 'This plugin allows GitLab to trigger Jenkins builds and display their results in the GitLab UI. Report an issue with this plugin.'

Tiếp theo, ta cần kết nối máy chủ Jenkins và máy chủ GitLab, trong phần Manage Jenkins, System, ta cần điền một số thông số liên quan đến server GitLab:

The screenshot shows the Jenkins configuration interface for managing external connections. Under the 'GitLab' section, there is a checked checkbox for 'Enable authentication for '/project' end-point'. Below it, there's a 'GitLab connections' section with a form to add a new connection. The form fields include:

- Connection name:** A dropdown menu containing 'gitlab server'.
- GitLab host URL:** A text input field containing 'http://git.ducanh.test'.
- Credentials:** A dropdown menu containing 'GitLab API token (jenkins git lab user)'.

At the bottom of the form are 'Save' and 'Apply' buttons.

Tiếp theo tại máy chủ GitLab, chúng ta cần tạo người dùng mới là Jenkins với quyền admin, để có thể sử dụng các dự án trên GitLab và triển khai:

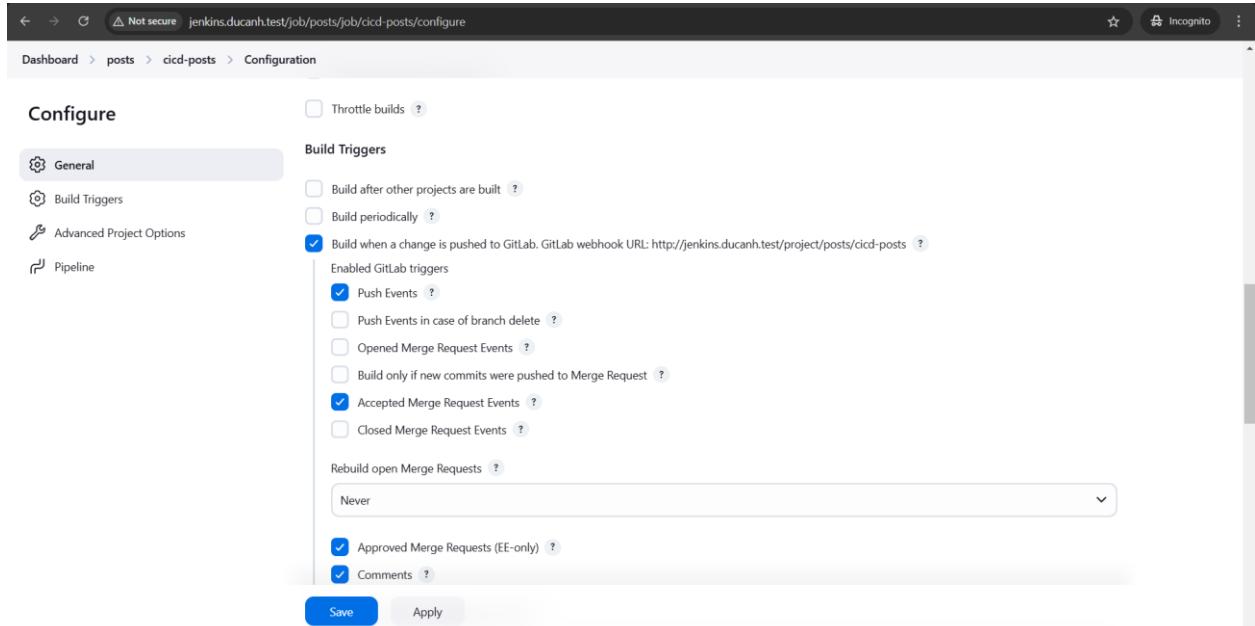
The screenshot shows the GitLab Admin Area under the 'Users' tab. The table lists the following users:

Name	Projects	Groups	Created on	Last activity
posts-develop-admin	1	1	8 Dec, 2024	8 Dec, 2024
posts-develop-post	1	1	8 Dec, 2024	Never
posts-develop-user	1	1	8 Dec, 2024	Never
jenkins dep trai	0	0	21 Nov, 2024	21 Nov, 2024
Dai dep trai	1	1	16 Nov, 2024	16 Nov, 2024
Duc Anh dep trai	1	1	16 Nov, 2024	8 Dec, 2024
Administrator	2	3	16 Nov, 2024	8 Dec, 2024

Qua các thao tác sử dụng token, ta sẽ kết nối thành công máy chủ Jenkins đến GitLab.

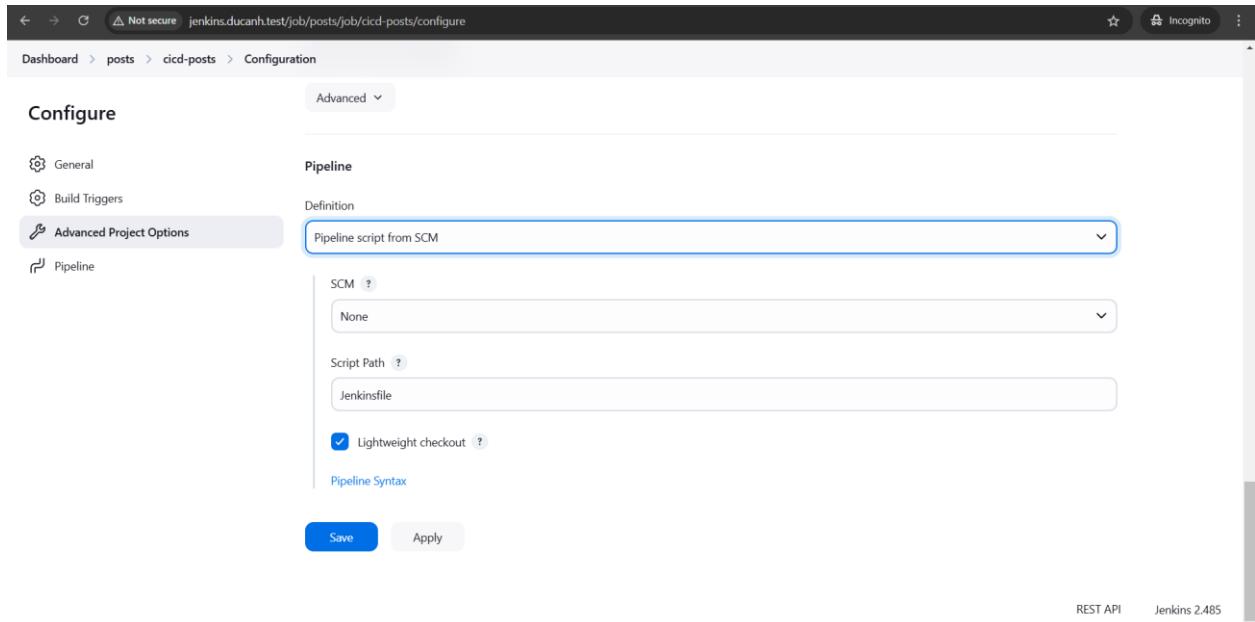
Tiếp theo, ta sẽ tạo folder dự án trên máy chủ Jenkins, trong folder tạo một pipeline

Một số chú ý trong phần tạo pipeline, thứ nhất, ta cần thiết lập các điều kiện để chạy pipe, ví dụ: pipeline chạy khi có đẩy mã nguồn, hợp nhất mã nguồn từ các nhánh:



The screenshot shows the Jenkins configuration page for a job named 'cicd-posts'. Under the 'General' tab, there is a 'Build Triggers' section. It includes options like 'Throttle builds', 'Build after other projects are built', 'Build periodically', and 'Build when a change is pushed to GitLab'. The 'Build when a change is pushed to GitLab' option is checked, and its sub-options show 'Push Events' (checked), 'Push Events in case of branch delete', 'Opened Merge Request Events', 'Build only if new commits were pushed to Merge Request' (unchecked), 'Accepted Merge Request Events' (checked), and 'Closed Merge Request Events'. Below this, there is a dropdown for 'Rebuild open Merge Requests' set to 'Never'. At the bottom are 'Save' and 'Apply' buttons.

Ngoài ra, ta cần chọn tùy chọn mặc định, sử dụng Jenkinsfile để chạy pipeline, lựa chọn SCM với tùy chọn Git, điền các thông tin cần thiết từ GitLab



The screenshot shows the Jenkins configuration page for a job named 'cicd-posts'. Under the 'Advanced' tab, there is a 'Pipeline' section. It includes a 'Definition' dropdown set to 'Pipeline script from SCM', an 'SCM' dropdown set to 'None', a 'Script Path' input field containing 'Jenkinsfile', and a checked 'Lightweight checkout' checkbox. At the bottom are 'Save' and 'Apply' buttons.

Sau khi đã cài đặt thành công, tiếp đến ta sẽ cấu hình webhook của GitLab để có thể thao dõi các sự kiện trên GitLab và kích hoạt quy trình tự động trên Jenkins:

Sau khi đã thiết lập thành công các kết nối từ máy chủ Jenkins với máy chủ GitLab,

máy chủ Jenkins với máy chủ triển khai dự án. Ta sẽ tiến hành viết Jenkinsfile để thực thi quy trình triển khai ứng dụng. Trước đó, ta sẽ tiến hành viết dockerFile cho dự án.

## 2.5 Viết dockerFile cho dự án

Các câu lệnh gồm môi trường sử dụng, đường dẫn thư mục dữ án, sao chép các thư viện, phụ thuộc Docker Image, chạy trong DockerImage với PORT 3000

```
FROM node:18

WORKDIR /

COPY package*.json ./

RUN npm install

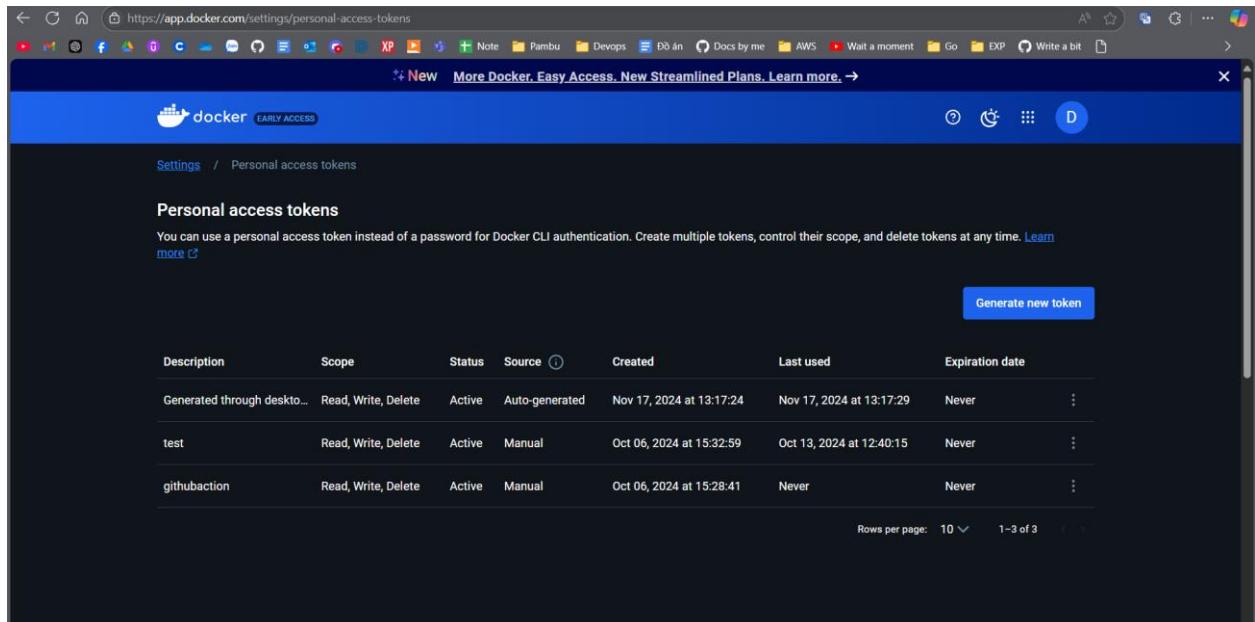
COPY . .

EXPOSE 3000

CMD ["npm", "start"]
```

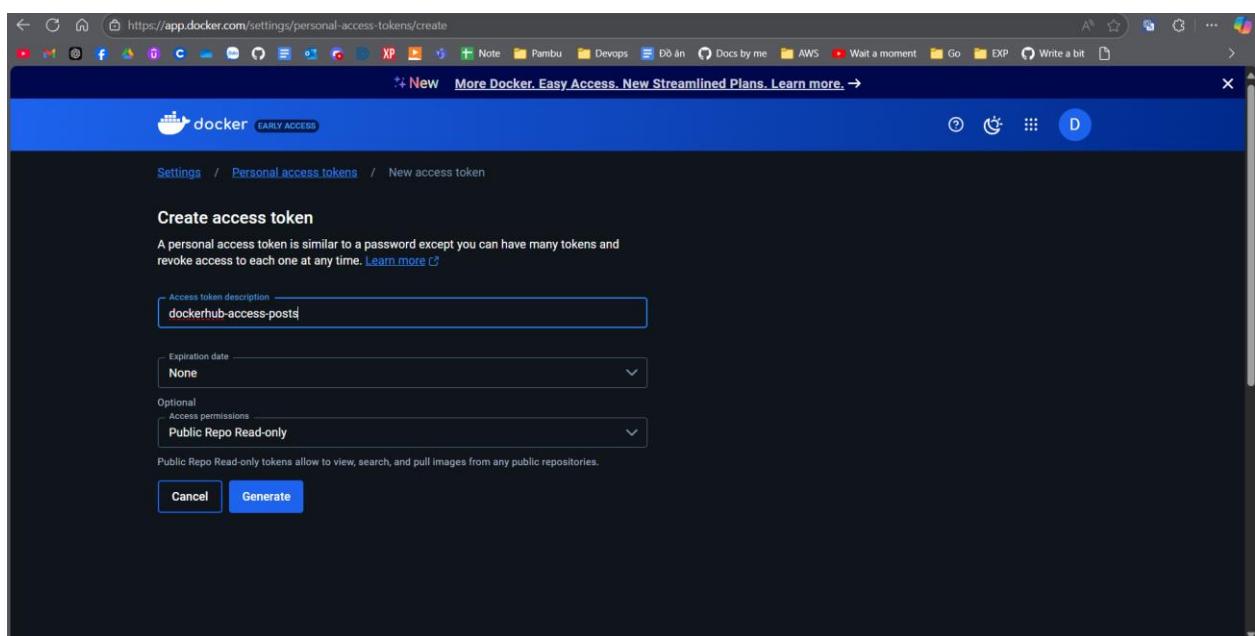
## 2.6 Kết nối với DockerHub

Để có thể kết nối Jenkins với DockerHub cần có access token của DockerHub, trong giao diện của DockerHub, ta tìm access token và tạo ra key mới, lưu lại sử dụng trong pipeline Jenkins.

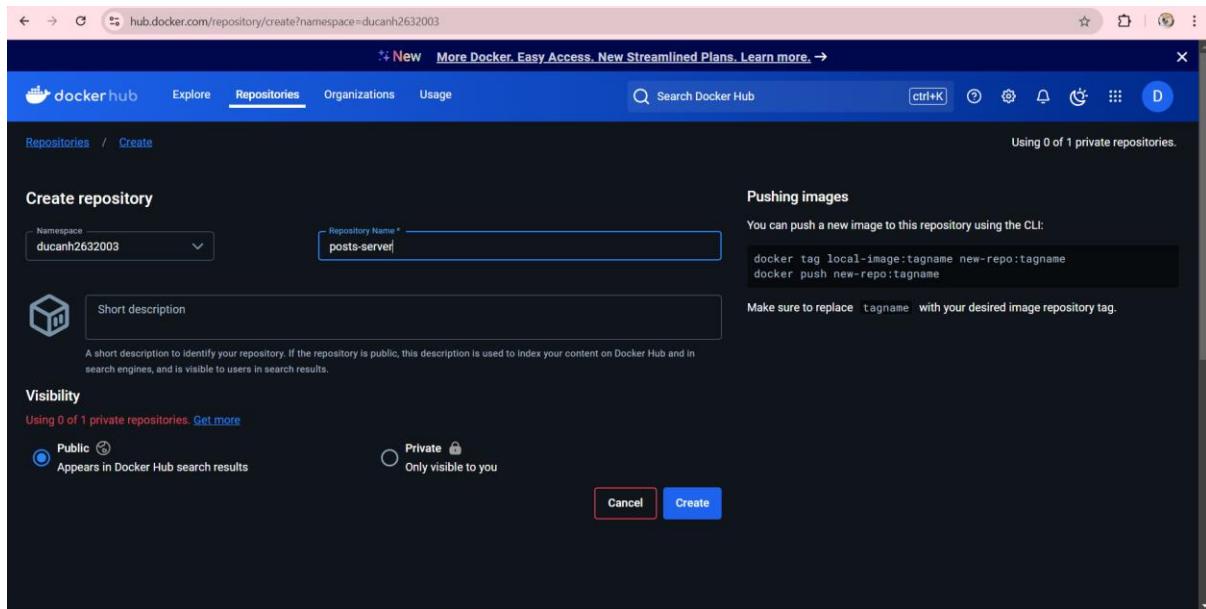


Description	Scope	Status	Source	Created	Last used	Expiration date
Generated through desktop...	Read, Write, Delete	Active	Auto-generated	Nov 17, 2024 at 13:17:24	Nov 17, 2024 at 13:17:29	Never
test	Read, Write, Delete	Active	Manual	Oct 06, 2024 at 15:32:59	Oct 13, 2024 at 12:40:15	Never
githubaction	Read, Write, Delete	Active	Manual	Oct 06, 2024 at 15:28:41	Never	Never

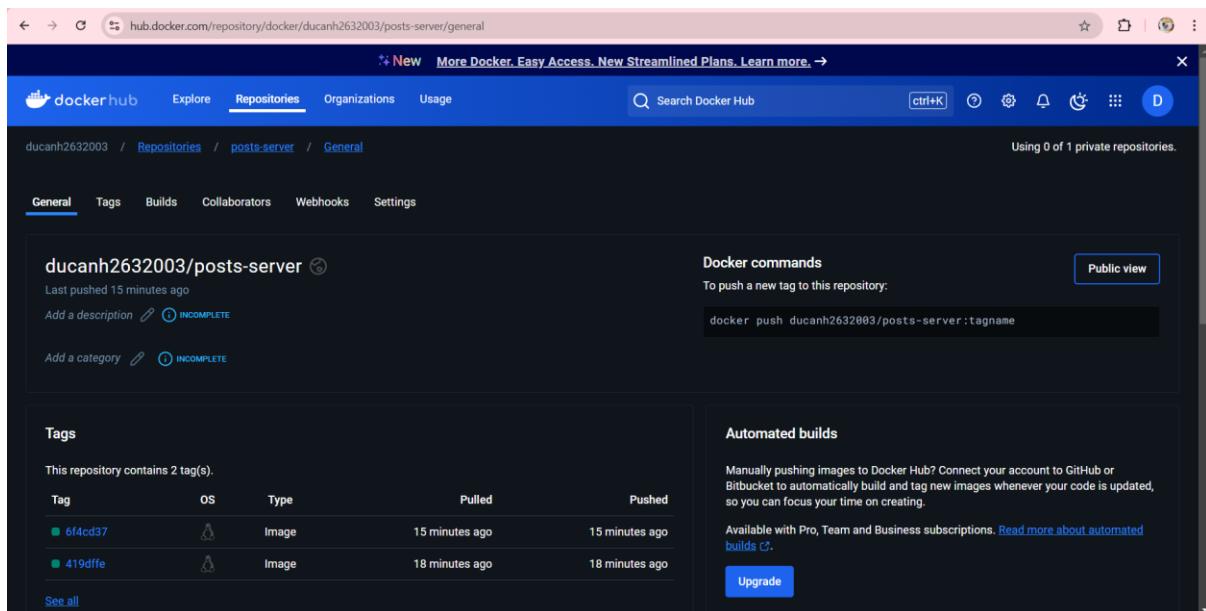
Sau khi chọn “generate new token”:



Tiếp theo, ta sẽ tạo một kho chứa các docker image của dự án, cụ thể trong trường hợp nào ta sẽ tạo một repository có tên “posts-server”:



Sau khi tạo, ta có kho chứa là các phiên bản của image:



## 2.7 Viết Jenkinsfile kiểm tra

Với vị trí là devops engineer, ta sẽ tiến hành viết Jenkinsfile ở nhánh dev (nhánh tích hợp). Viết 1 đoạn Jenkinsfile để kiểm tra kết nối, bao gồm sử dụng agent “dev-server” (sử dụng node chạy file Jenkinsfile được kết nối với máy chủ triển khai được thiết lập trước đó), kèm theo chạy mã trên máy chủ triển khai các câu lệnh whoami, pwd, ls -la:

The screenshot shows a Jenkinsfile editor interface. At the top, there are tabs for 'dev' and 'Jenkinsfile'. To the right of the tabs is a dropdown menu labeled 'Select a template type'. The main area contains the Jenkinsfile code:

```
1 pipeline[  
2     agent{  
3         label 'dev-server'  
4     }  
5     stages{  
6         stage('info'){  
7             steps{  
8                 sh(script: """ whoami;pwd; ls -la""", label: "first stage")  
9             }  
10        }  
11    }  
12 ]
```

Sau khi đẩy mã nguồn trên nhánh dev, kiểm tra pipeline trên máy chủ Jenkins:

The screenshot shows the Jenkins 'Builds' page. At the top, there is a 'Builds' header, a 'Filter' input field, and a search icon. Below the header, it says 'Today'. There are three entries listed:

- #3 13:34 (Green checkmark) - Started by GitLab push by Administrator
- #2 13:33 (Red X) - Started by GitLab push by Administrator
- #1 13:19 (Red X) - Started by GitLab push by posts-develop-admin

Vậy là pipeline được viết bằng Jenkinsfile đã hoạt động thành công. Ta có thể xem log quá trình chạy:

The screenshot shows the Jenkins interface for a build named 'cicd-posts' (build #3). The left sidebar has a 'Console Output' section selected. The main area displays the Jenkinsfile's git configuration and the execution of the 'git' command to fetch changes from the remote repository.

```

Started by GitLab push by Administrator
Lightweight checkout support not available, falling back to full checkout.
Checking out git http://git.ducanh.test/posts/posts.git into /var/lib/jenkins/workspace/posts/cicd-posts@script/7f827c383d820c81d4d5f567d15b5ad0b74497b9e44c832379b3137a812fb020 to read Jenkinsfile
The recommended git tool is: NONE
using credential 3
> git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/posts/cicd-posts@script/7f827c383d820c81d4d5f567d15b5ad0b74497b9e44c832379b3137a812fb020/.git # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url http://git.ducanh.test/posts/posts.git # timeout=10
Fetching upstream changes from http://git.ducanh.test/posts/posts.git
> git --version # timeout=10
> git --version # 'git' version 2.25.1'
using GIT_ASKPASS to set credentials jenkins git lab user
> git fetch --tags --force --progress -- http://git.ducanh.test/posts/posts.git +refs/heads/*:refs/remotes/origin/* # timeout=10
skipping resolution of commit remotes/origin/dev, since it originates from another repository
Seen branch in repository origin/dev
Seen branch in repository origin/feature-post
Seen branch in repository origin/feature-user
Seen branch in repository origin/main
Seen branch in repository origin/staging
Seen 5 remote branches

```

Trước khi xây dựng các stage với JenkinsFile, ta sẽ thiết lập các biến môi trường được lưu trữ bảo mật với Jenkins, trong giao diện Jenkins. Ta vào phần Manage Jenkins, Credentials:

The screenshot shows the 'New credentials' form under the 'Manage Jenkins > Credentials' section. The 'Kind' field is set to 'Secret text'. The 'Scope' field is set to 'Global (Jenkins, nodes, items, all child items, etc)'. The 'Secret' field contains the value 'db-username'. The 'ID' field is set to 'db-username'. The 'Description' field is empty. A 'Create' button is at the bottom.

Ta tiến hành tạo biến môi trường như db\_username: sử dụng để truy cập database. Tương tự, ta cung cấp db\_password và access\_token\_secret:

ID	Name	Kind	Description
jenkinsgitlabuser	GitLab API token (jenkins git lab user)	GitLab API token	jenkins git lab user
1	GitLab API token (jenkins git lab user)	GitLab API token	jenkins git lab user
2	jenkins***** (jenkins git lab user)	Username with password	jenkins git lab user
3	jenkins***** (jenkins git lab user)	Username with password	jenkins git lab user
docker-hub-token	docker-hub-token	Secret text	
db_username	db_username	Secret text	
db_password	db_password	Secret text	
access_token_secret	access_token_secret	Secret text	

Ta bắt đầu xây dựng lần lượt các stages trong JenkinsFile. Bắt đầu với câu lệnh mở đầu pipeline với node Jenkins Agent nào được chạy và các biến môi trường sẽ sử dụng:

```
pipeline {
    agent {
        label 'dev-server'
    }
    environment {
        DOCKER_IMAGE = "ducanh2632003/posts-server"
    }
}
```

Việc sử dụng biến môi trường giúp lưu trữ các thông tin quan trọng, tái sử dụng.

Ta xây dựng stage prepare (chuẩn bị), ở đây chúng ta sử dụng commitId có được mỗi khi commit từ GitLab với các thao tác push hay merge mã nguồn. Ta sẽ sử dụng commitId để làm tag cho Docker Image hiện tại. Tag là đoạn mã thẻ hiện phiên bản của Docker Image:

```
stage('Prepare') {
    steps {
        script {
            def commitHash = sh(script: "git rev-parse --short HEAD", returnStdout: true).trim()

            // Lấy Docker Hub token từ Jenkins Credentials
            withCredentials([string(credentialsId: 'docker-hub-token', variable: 'DOCKER_HUB_TOKEN'),
                string(credentialsId: 'db-username', variable: 'DB_USERNAME'),
                string(credentialsId: 'db-password', variable: 'DB_PASSWORD'),
                string(credentialsId: 'access-token-secret', variable: 'ACCESS_TOKEN_SECRET')])

            // Gán các thông tin vào môi trường
            env.DOCKER_HUB_TOKEN = DOCKER_HUB_TOKEN
            env.DB_USERNAME = DB_USERNAME
            env.DB_PASSWORD = DB_PASSWORD
            env.ACCESS_TOKEN_SECRET = ACCESS_TOKEN_SECRET

            echo "Docker Hub Token: ${env.DOCKER_HUB_TOKEN}"
            echo "DB Username: ${env.DB_USERNAME}" // Cảnh thận với việc in ra thông tin nhạy cảm
            env.DOCKER_IMAGE_TAG = commitHash
        }
    }
}
```

Stage build Docker Image, đoạn mã này giúp xây dựng Docker Image cho mã nguồn, câu lệnh docker build sẽ tìm kiếm file DockerFile trong mã nguồn được đẩy lên GitLab. Ở trường hợp này, docker sử dụng file DockerFile được ta chuẩn bị ở trên để xây dựng Docker Image:

```
stage('Build Docker Image') {
    steps {
        script {
            echo "Building Docker image..."
            sh """
                docker build -t ${DOCKER_IMAGE}:latest .
            """
        }
    }
}
```

Stage push Docker Image lên DockerHub, nơi lưu trữ các phiên bản image, lần lượt các thao tác đăng nhập và push Docker Image lên:

```
stage('Push to Docker Hub') {
    steps {
        script {
            echo "Logging in to Docker Hub using Access Token..."
            sh """
                echo $DOCKER_HUB_TOKEN | docker login -u ducanh2632003 --password-stdin
                docker tag ${DOCKER_IMAGE}:latest ${DOCKER_IMAGE}:${DOCKER_IMAGE_TAG}
                docker push ${DOCKER_IMAGE}:${DOCKER_IMAGE_TAG}
            """
        }
    }
}
```

Stage deploy Docker Image, ở bước này, ta sẽ tiến hành sử dụng các câu lệnh với server triển khai. Ta sẽ chuyển sang người dùng Jenkins trên server triển khai, thực hiện thao tác xóa các Docker Image, Docker Container cũ và lấy từ Docker Hub các Docker Image, Docker Container mới và tiến hành chạy ứng dụng với Docker.

```
stage('Deploy Docker Image') {
    steps {
        script {
            echo "Deploying Docker container..."

            sh """
                # Chuyển sang người dùng jenkins
                sudo su - jenkins -c '
                    # Dừng tất cả container đang chạy
                    docker stop \$(docker ps -q)

                    # Xóa tất cả container
                    docker rm \$(docker ps -aq)

                    # Xóa tất cả image
                    docker rmi \$(docker images -q)

                    # Kéo Docker image mới nhất
                    docker pull ${DOCKER_IMAGE}:${DOCKER_IMAGE_TAG}
                '
            '
        }
    }
}
```

```

# Chạy container mới
docker run -d --name posts-server -p 3000:3000 \
    -e DB_USERNAME=${DB_USERNAME} \
    -e DB_PASSWORD=${DB_PASSWORD} \
    -e ACCESS_TOKEN_SECRET=${ACCESS_TOKEN_SECRET} \
    ${DOCKER_IMAGE}:${DOCKER_IMAGE_TAG}

# Kiểm tra trạng thái container
if [ `$(docker ps -q -f name=posts-server)` ]; then
    echo "Container started successfully!"
else
    echo "Failed to start container. Checking logs..."
    docker logs posts-server || true
    exit 1
fi
"""

}

```

Tiến hành đây mã nguồn và xem thành quả:

Tất cả các stage đã được hoàn thành.

Ứng dụng có thể truy cập:

### 3. Kịch bản phát triển phần mềm sử dụng quy trình CI/CD

Hệ thống đã được thiết lập xong, bắt đầu triển khai triển xây dựng dự án:

Dự án sẽ được 2 lập trình viên với 2 chức năng User (người dùng) và Post (bài viết).

Vì vậy ta sẽ xây dựng 2 nhánh feature từ nhánh dev vừa được triển khai bên trên:

The screenshot shows a GitLab repository named 'posts' under the 'posts' project. The sidebar on the left lists various project management sections like Issues, Merge requests, CI/CD, etc. The main area displays the repository structure with a 'dev' branch selected. A message at the top indicates a push to the 'feature-user' branch. Below the message is a 'Create merge request' button. A 'Switch branch/tag' dropdown is open, showing the current selection 'dev'. A search bar for branches and tags is also present. The right side of the screen lists recent commits, all of which were pushed 2 weeks ago. The commits include updates to Dockerfile, Jenkinsfile, and Profile.

Commit	Message	Last update
.gitignore	initial core	2 weeks ago
Dockerfile	Update Dockerfile	4 hours ago
Jenkinsfile	Update Jenkinsfile	3 hours ago
Profile	initial core	2 weeks ago

Mỗi lập trình viên được thêm vào dự án sẽ thực hiện một chức năng của nhánh.

Đầu tiên, ta đóng vai là lập trình viên trên nhánh feature-user, bắt đầu xây dựng dự án với chức năng user.

Truy cập với tài khoản dành cho lập trình viên phụ trách chức năng user

The screenshot shows a GitLab project interface for a repository named 'posts'. The sidebar on the left contains links for Project information, Repository, Issues, Merge requests, CI/CD, Security & Compliance, Deployments, Monitor, Infrastructure, Packages & Registries, Analytics, Wiki, and Snippets. The main area displays the project details: 48 Commits, 5 Branches, 0 Tags, 266 KB Files, and 266 KB Storage. A commit titled 'Update index.js' by 'Administrator' is shown, dated 3 hours ago. Below the commit is a note about no license. A table lists files with their last commit and update times:

Name	Last commit	Last update
middleware	initial core	2 weeks ago
models	initial core	2 weeks ago
routes	initial core	2 weeks ago

Lập trình viên phụ trách chức năng user sẽ lấy source code về:

```
root@devserver:~# git config --global user.name "posts-develop-user"
root@devserver:~# git config --global user.email "posts-develop-user@git.ducanh.test"
root@devserver:~# git clone http://git.ducanh.test/posts/posts.git
Cloning into 'posts'...
Username for 'http://git.ducanh.test': posts-develop-user
Password for 'http://posts-develop-user@git.ducanh.test':
remote: Enumerating objects: 165, done.
remote: Counting objects: 100% (16/16), done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 165 (delta 7), reused 16 (delta 7), pack-reused 149
Receiving objects: 100% (165/165), 56.83 KiB | 2.10 MiB/s, done.
Resolving deltas: 100% (88/88), done.
```

Sau đó, lập trình viên tiến hành lập trình các chức năng theo nghiệp vụ, với chức năng user, ta sẽ chỉnh sửa 2 file:

The screenshot shows a code editor with the file 'auth.js' open. The code is a middleware function for an Express router. The changes were made by 'Henry Nguyen, 4 years ago' and are described as 'Done'. The code includes imports for express, Router, argon2, jsonwebtoken, and User models. It defines a route for '/api/auth' that checks if the user is logged in and returns a JWT token.

```
JS auth.js middleware JS auth.js routes X
routes > JS auth.js > router.get('/').callback
Henry Nguyen, 4 years ago | author (Henry Nguyen)
1 const express = require('express')
2 const router = express.Router()
3 const argon2 = require('argon2')
4 const jwt = require('jsonwebtoken')
5 const verifyToken = require('../middleware/auth')
6
7 const User = require('../models/User')
8
9 // @route GET api/auth
10 // @desc Check if user is logged in
11 // @access Public
12 router.get('/', verifyToken, async (req, res) => {
    try {
        const user = await User.findById(req.userId).select('-password')
        if (!user)
            return res.status(400).json({ success: false, message: 'User not found' })
        res.json({ success: true, user })
    } catch (error) {
        console.log(error)
        res.status(500).json({ success: false, message: 'Internal server error' })
    }
})
```

```

JS auth.js middleware > JS auth.js routes
middleware > JS auth.js > [e] verifyToken
Henry Nguyen, 4 years ago | 1 author (Henry Nguyen)
1 const jwt = require('jsonwebtoken')
2
3 const verifyToken = (req, res, next) => {
4   const authHeader = req.header('Authorization')
5   const token = authHeader && authHeader.split(' ')[1]
6
7   if (!token) return res
8     .status(401)
9     .json({ success: false, message: 'Access token not found' })
10
11   try {
12     const decoded = jwt.verify(token, process.env.ACCESS_TOKEN_SECRET)
13
14     req.userId = decoded.userId
15     next()
16   } catch (error) {
17     console.log(error)
18     return res.status(403).json({ success: false, message: 'Invalid token' })
19   }
20 }
21
22 module.exports = verifyToken

```

Sau đó, ta tiến hành đẩy mã nguồn mới lên nơi lưu trữ tạm thời ở local, với lệnh git checkout feature-user: chuyển sang nhánh feature-user

git add .: thêm các file code đã thay đổi

git status: kiểm tra các thay đổi

```

root@devserver:~/posts/middleware# git checkout feature-user
M      middleware/auth.js
M      routes/auth.js
Branch 'feature-user' set up to track remote branch 'feature-user' from 'origin'.
Switched to a new branch 'feature-user'
root@devserver:~/posts/middleware# git add .
root@devserver:~/posts/middleware# git status
On branch feature-user
Your branch is up to date with 'origin/feature-user'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   auth.js

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   ./routes/auth.js

```

Ta tiến hành commit, để có thể đưa mã nguồn lên GitLab lưu trữ.

```

root@devserver:~/posts# git commit -m "feature(user): add route, controller, middleware"
[feature-user 7fb5613] feature(user): add route, controller, middleware
 2 files changed, 135 insertions(+), 63 deletions(-)
 rewrite middleware/auth.js (72%)
 rewrite routes/auth.js (80%)

```

```

root@devserver:~/posts# git push
Username for 'http://git.ducanh.test': posts-develop-user
Password for 'http://posts-develop-user@git.ducanh.test':
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 1.53 KiB | 521.00 KiB/s, done.
Total 6 (delta 1), reused 0 (delta 0)
remote:
remote: To create a merge request for feature-user, visit:
remote:   http://git.ducanh.test/posts/posts/-/merge_requests/new?merge_request%5Bsource_branch%5D=fea
ture-user
remote:
To http://git.ducanh.test/posts/posts.git
  8c0aef1..7fb5613  feature-user -> feature-user

```

Lúc này trên GitLab, nhánh feature-user đã có code mới nhất của chúng ta:

Name	Last commit	Last update
middleware	feature(user): add route, controller, middleware...	2 minutes ago
models	initial core	2 weeks ago
routes	feature(user): add route, controller, middleware...	2 minutes ago
.env.example	initial core	2 weeks ago
.gitignore	initial core	2 weeks ago
Dockerfile	Update Dockerfile	5 hours ago
Jenkinsfile	Update Jenkinsfile	3 hours ago
Profile	initial core	3 weeks ago

Sau khi có mã nguồn mới nhất vừa được đẩy lên từ local, CICD sẽ không chạy, lập trình viên sẽ tiến hành tạo pull request để admin (quản lý dự án) có thể kiểm tra và tích hợp mã nguồn vào nhánh dev. Tiến hành tạo pull request:

Lúc này admin (quản lý) sẽ vào kiểm tra code và cho phép merge vào nhánh dev):

Kiểm tra các file mã nguồn đã thay đổi:

Tiến hành merge request, CICD sẽ chạy và triển khai dự án tự động lên môi trường dev:

Description Triggered by GitLab Merge Request #1: posts/feature-user => dev

Start Prepare Build Docker Image Push to Docker Hub Deploy Docker Image Clean Docker Cache End

Clean Docker Cache - <1s

- > Cleaning up Docker cache... — Print Message <1s
- > docker builder prune -force docker image prune --all -force docker volume prune -force docker network prune --force — Shell Script <1s
- > Pipeline completed. — Print Message <1s
- > Build, push, and deploy completed successfully! — Print Message <1s

CICD đã khởi chạy xong, kiểm tra chức năng user đã có thể sử dụng:

Chức năng đăng ký:

Chức năng đăng nhập:

Tương tự, phía lập trình viên phụ trách chức năng post cũng thực hiện các thao tác tương tự:

```
root@devserver:~/posts# git config --global user.name "posts-develop-post"
root@devserver:~/posts# git config --global user.email "posts-develop-post@git.ducanh.test"
```

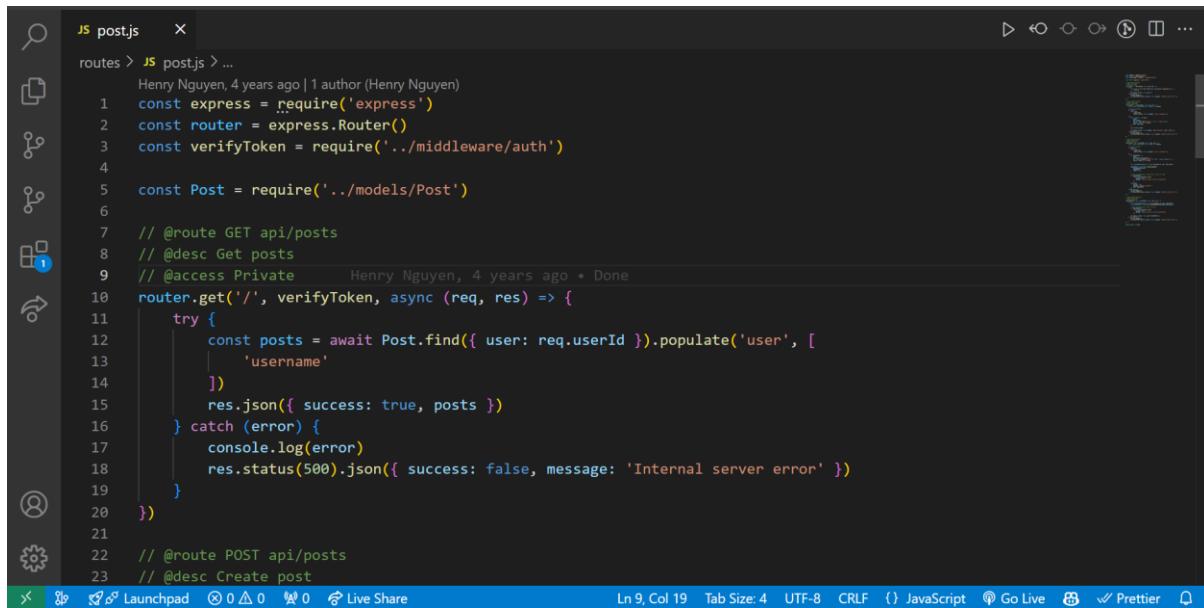
Chuyển sang nhánh feature post:

```
root@devserver:~/posts# git checkout dev
Switched to branch 'dev'.
Your branch is up to date with 'origin/dev'.
root@devserver:~/posts# git checkout feature-post
Switched to branch 'feature-post'.
Your branch is up to date with 'origin/feature-post'.
```

Ta lấy code mới nhất từ nhánh dev về:

```
root@devserver:~/posts/middleware# git pull origin dev
Username for 'http://git.ducanh.test': posts-develop-post
Password for 'http://posts-develop-post@git.ducanh.test':
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), 244 bytes | 244.00 KiB/s, done.
From http://git.ducanh.test/posts/posts
 * branch      dev      -> FETCH_HEAD
   8c0aef1..b6ba33a dev      -> origin/dev
Updating 8c0aef1..b6ba33a
Fast-forward
 middleware/auth.js |  27 ++++++-----+
 routes/auth.js    | 133 ++++++++++++++++++++++++++++++++
 2 files changed, 116 insertions(+), 44 deletions(-)
```

Chỉnh sửa mã nguồn:



The screenshot shows a code editor interface with the file 'post.js' open. The code is a Node.js module for a POST endpoint. It imports express and a Router, and uses a verifyToken middleware. It defines a Post model and handles a GET request to retrieve posts for a user. The code is annotated with comments and JSDoc-style documentation. The editor has a dark theme and various toolbars and status bars at the bottom.

```
JS post.js
routes > JS post.js > ...
  Henry Nguyen, 4 years ago | 1 author (Henry Nguyen)
1  const express = require('express')
2  const router = express.Router()
3  const verifyToken = require('../middleware/auth')
4
5  const Post = require('../models/Post')
6
7  // @route GET api/posts
8  // @desc Get posts
9  // @access Private
10 router.get('/', verifyToken, async (req, res) => {
11     try {
12         const posts = await Post.find({ user: req.userId }).populate('user', [
13             'username'
14         ])
15         res.json({ success: true, posts })
16     } catch (error) {
17         console.log(error)
18         res.status(500).json({ success: false, message: 'Internal server error' })
19     }
20 })
21
22 // @route POST api/posts
23 // @desc Create post
```

Đẩy mã nguồn lên kho lưu trữ tạm thời trên local và commit:

```

root@devserver:~/posts# git add .
root@devserver:~/posts# git status
On branch feature-post
Your branch is ahead of 'origin/feature-post' by 3 commits.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   routes/post.js

root@devserver:~/posts# git commit -m "feature(post): add route, controller"

```

Commit và đẩy mã nguồn lên GitLab:

```

root@devserver:~/posts# git push
Username for 'http://git.ducanh.test': posts-develop-post
Password for 'http://posts-develop-post@git.ducanh.test':
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 1.16 KiB | 395.00 KiB/s, done.
Total 4 (delta 1), reused 0 (delta 0)

remote: To create a merge request for feature-post, visit:
remote: http://git.ducanh.test/posts/posts/-/merge_requests/new?merge_request%5Bsource_branch%5D=feature-post
remote:
To http://git.ducanh.test/posts/posts.git
  8c0aef1..cbe6da7  feature-post -> feature-post

```

Kiểm tra trên GitLab:

The screenshot shows a GitLab repository named 'posts'. A success message at the top says 'You pushed to feature-post just now'. Below it, there's a 'Create merge request' button. The file list shows several files with their last commit details:

Name	Last commit	Last update
middleware	feature(user): add route, controller, middlew...	36 minutes ago
models	initial core	2 weeks ago
routes	feature(post): add route, controller	3 minutes ago
.env.example	initial core	2 weeks ago
.gitignore	initial core	2 weeks ago
Dockerfile	Update Dockerfile	5 hours ago
Jenkinsfile	Update Jenkinsfile	4 hours ago
Profile	initial core	3 weeks ago

Tạo Pull Request cho admin kiểm tra:

The screenshot shows a GitLab merge request page for a project named 'posts'. The merge request, titled 'feature(post): add route, controller', has been created just now by a developer. It is currently in the 'Request to merge' state, targeting the 'dev' branch from the 'feature-post' branch. A prominent message indicates that no pipeline has been added, suggesting the addition of a .gitlab-ci.yml file. Below this, a note asks if technical debt or code vulnerabilities are being added. The merge request has been approved by a participant. At the bottom, there is a 'Merge' button and a note stating that one commit and one merge commit will be added to the 'dev' branch. The right side of the screen displays various settings and metadata for the merge request.

Lúc nào admin sẽ vào dự án, merge mã nguồn và dự án sẽ kích hoạt CICD triển khai tự động:

This screenshot is identical to the one above, showing the same merge request details and approval status. The key difference is the presence of a 'Merge' button at the bottom of the main content area, indicating that the merge has been initiated manually by an administrator.

CICD sẽ được chạy:

Jenkins Pipeline Status:

- Branch: -
- Commit: -
- Duration: 34s (a few seconds ago)
- Last Triggered by GitLab Merge Request #2: posts/feature-post => dev

Description: Triggered by GitLab Merge Request #2: posts/feature-post => dev

Pipeline Stages:

- Start
- Prepare
- Build Docker Image
- Push to Docker Hub
- Deploy Docker Image
- Clean Docker Cache
- End

Clean Docker Cache - <1s

Action	Description	Time
> Cleaning up Docker cache... - Print Message		<1s
> docker builder prune --force docker image prune --all --force docker volume prune --force docker network prune --force	- Shell Script	<1s
> Pipeline completed.	- Print Message	<1s
> Build, push, and deploy completed successfully!	- Print Message	<1s

Chạy thành công, kiểm tra các chức năng của ứng dụng đã được sử dụng:

Sử dụng api create post:

Postman Collection: Posts-For-CICD / create post

POST http://192.168.91.110:3000/api/posts

Params: Authorization, Headers (10), Body (raw), Scripts, Settings

Body: raw, JSON

```
{
  "title": "post 1",
  "description": "description of post 1",
  "url": "url for post",
  "status": "TO LEARN"
}
```

Response: 200 OK, 71 ms, 500 B

Body (Pretty):

```
{
  "success": true,
  "message": "happy learning!",
  "post": {
    "_id": "67712545f76b30012a77df5",
    "title": "post 1",
    "description": "description of post 1",
    "url": "https://url for post",
    "status": "TO LEARN",
    "user": "67711c97338d2a0013870e38",
    "__v": 0
  }
}
```

Sử dụng api update post:

The screenshot shows the Postman interface with a collection named 'Posts-For-CICD'. A PUT request is selected for the 'update post' endpoint. The request URL is `http://192.168.91.110:3000/api/posts/67712354f576b30012a77df5`. The request body is raw JSON:

```

1  {
2     "title": "post 1",
3     "description": "description of post 1",
4     "url": "url for post",
5     "status": "LEARNED"
6   }

```

The response status is 200 OK, with a response time of 49 ms and a size of 503 B. The response body is:

```

1  {
2     "success": true,
3     "message": "Excellent progress!",
4     "post": {
5         "_id": "67712354f576b30012a77df5",
6         "title": "post 1",
7         "description": "description of post 1",
8         "url": "https://url for post",
9         "status": "LEARNED",
10        "user": "67711c07338d2a0013879e38",
11        "_v": 0
12    }
13 }

```

Sử dụng api get list post:

The screenshot shows the Postman interface with the same collection. A GET request is selected for the 'get post' endpoint. The request URL is `http://192.168.91.110:3000/api/posts`. The request includes an Authorization header with the value `Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJtJc2V...`.

The response status is 200 OK, with a response time of 75 ms and a size of 504 B. The response body is:

```

1  {
2     "success": true,
3     "posts": [
4         {
5             "_id": "67712354f576b30012a77df5",
6             "title": "post 1",
7             "description": "description of post 1",
8             "url": "https://url for post",
9             "status": "LEARNED",
10            "user": {
11                "_id": "67711c07338d2a0013879e38",
12                "username": "joawaaa"
13            },
14            "_v": 0
15        }
16    ]
17 }

```

Vậy là dự án đã được triển khai đủ các chức năng.

## 4. Tích hợp Monitoring

### 4.1 Tích hợp Prometheus

Trước hết, ta sẽ tích hợp Prometheus với các chức năng như xây dựng log và event, giám sát API và cơ sở dữ liệu, theo dõi tài nguyên hệ thống (CPU, RAM, Disk, Network), và tích hợp visualization.

Ta sẽ chuẩn bị một máy chủ Window Server triển khai Prometheus, máy chủ Prometheus sẽ lắng nghe các máy chủ triển khai dự án trong cùng mạng.

Tải về Prometheus phù hợp với máy chủ [Download | Prometheus](#)

The screenshot shows the Prometheus download page. In the top right corner, there is a sidebar with a list of exporters: prometheus, alertmanager, blackbox\_exporter, consul\_exporter, graphite\_exporter, memcached\_exporter, mysqld\_exporter, node\_exporter, promlens, pushgateway, and statsd\_exporter. Below this, there are dropdown menus for 'Operating system' (set to 'popular') and 'Architecture' (set to 'popular'). The main content area is titled 'prometheus' and describes it as 'The Prometheus monitoring system and time series database'. It links to the GitHub repository. A table below shows file details for version 3.1.0-rc.0 / 2024-12-18 (Pre-release). The table has columns: File name, OS, Arch, Size, and SHA256 Checksum. The 'prometheus.exe' file is highlighted.

Cấu hình file Prometheus.yml trong tệp cấu hình tải về trên máy chủ:

Name	Date modified	Type	Size
data	12/30/2024 4:00 PM	File folder	
LICENSE	12/30/2024 3:43 PM	File	12 KB
NOTICE	12/30/2024 3:43 PM	File	4 KB
<b>prometheus.exe</b>	12/30/2024 3:43 PM	Application	146,590 KB
<b>prometheus.yml</b>	12/30/2024 3:46 PM	Yaml Source File	2 KB
promtool.exe	12/30/2024 3:43 PM	Application	139,743 KB

Cấu hình file yml, một số lưu ý:

```
global:  
  scrape_interval: 15s  
  evaluation_interval: 15s
```

**scrape\_interval:** Định nghĩa khoảng thời gian mà Prometheus sẽ lấy dữ liệu từ các endpoint. Ở đây, Prometheus lấy dữ liệu mỗi 15 giây. Mặc định là 1 phút, nhưng cấu hình này giảm xuống còn 15 giây để thu thập dữ liệu nhanh hơn.

**evaluation\_interval:** Khoảng thời gian để Prometheus đánh giá các rules (nếu được định nghĩa).

Ví dụ: Nếu bạn có các cảnh báo hoặc biểu thức, chúng sẽ được tính toán mỗi 15 giây.

```
alerting:  
  alertmanagers:  
    - static_configs:  
      - targets:  
        # - alertmanager:9093
```

**alertmanagers:** Khai báo vị trí của Alertmanager - công cụ gửi thông báo khi có cảnh báo.

**targets:** Định nghĩa địa chỉ nơi Alertmanager đang chạy.

Trong ví dụ này, dòng - alertmanager:9093 đã bị comment. Bạn có thể kích hoạt nó nếu cần gửi cảnh báo đến Alertmanager.

```
rule_files:  
  # - "first_rules.yml"  
  # - "second_rules.yml"
```

**rule\_files:** Định nghĩa các file chứa rules (quy tắc) để Prometheus đánh giá và kích hoạt cảnh báo.

Ví dụ: first\_rules.yml hoặc second\_rules.yml có thể chứa các quy tắc như "cảnh báo nếu CPU vượt quá 90%".

Hiện tại các dòng này đang bị comment, cần bỏ comment nếu bạn muốn sử dụng rules.

**scrape\_configs:** Phần cấu hình chính để Prometheus thu thập dữ liệu từ các nguồn khác nhau.

Mỗi nguồn dữ liệu (job) được định nghĩa bằng một khối - job\_name.

```
- job_name: "prometheus"
  static_configs:
    - targets: ["localhost:9090"]
```

**job\_name:** Tên của job này, ở đây là "prometheus".

Prometheus sẽ tự giám sát chính nó (Prometheus server cũng expose /metrics tại cổng 9090).

**static\_configs:** Danh sách các endpoint cố định để Prometheus thu thập dữ liệu.

**targets:** Danh sách địa chỉ endpoint, ở đây là localhost:9090 (Prometheus server chạy trên máy cục bộ).

```
- job_name: "node_api"
  metrics_path: "/metrics"
  static_configs:
    - targets: ["localhost:5000"]
```

**job\_name:** Tên của job này, ở đây là "node\_api". Tên này sẽ được gắn làm label job=<job\_name> trong dữ liệu thu thập.

**metrics\_path:** Đường dẫn mà Prometheus sẽ truy cập để lấy dữ liệu metric từ API.

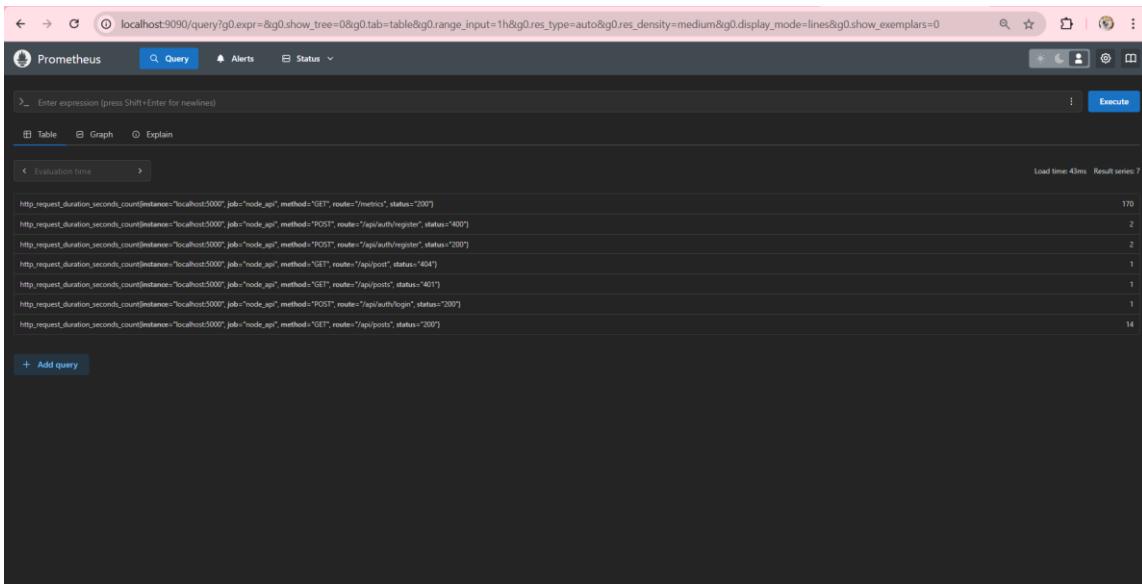
Mặc định là /metrics. Ở đây, nó được giữ nguyên.

**static\_configs:**

**targets:** Endpoint của API Node.js đang chạy. Ở đây, Prometheus sẽ thu thập metric từ localhost:5000/metrics.

Chạy file prometheus.exe để chạy Prometheus trên PORT 9090

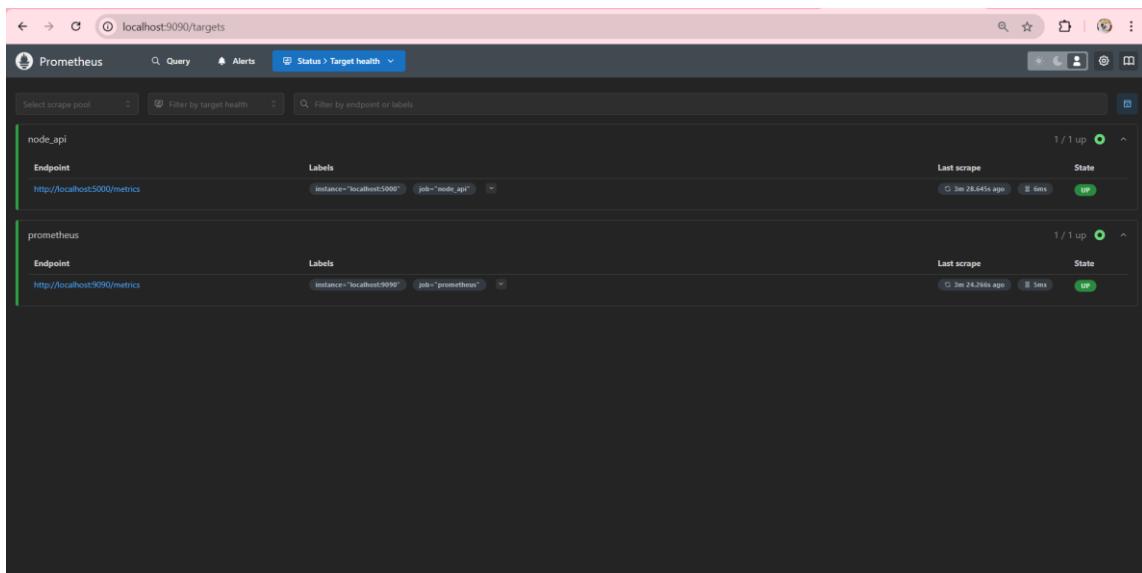
Kiểm tra PORT 9090 trên Window Server, Prometheus được chạy:



Để hệ thống API expose dữ liệu metric, bạn cần cài đặt các thư viện như prom-cl (với NodeJS). Một vài thiết lập trên hệ thống API hướng đến:

- Xây dựng hệ thống ghi nhận nhật ký và sự kiện.
- Theo dõi hiệu suất và trạng thái của API, cơ sở dữ liệu.
- Giám sát toàn diện các tài nguyên hệ thống như CPU, RAM, ổ đĩa và mạng.
- Tích hợp khả năng trực quan hóa dữ liệu giúp dễ dàng phân tích và đánh giá.

Truy cập PORT 9090 trên Window Server kiểm tra phần status > target health, kiểm tra ứng dụng hệ thống API đã được kết nối:



Hệ thống API với tên node\_api đã được kết nối với máy chủ Prometheus, ngoài ra Prometheus còn có chức năng tự kiểm tra trạng thái hoạt động của chính mình.

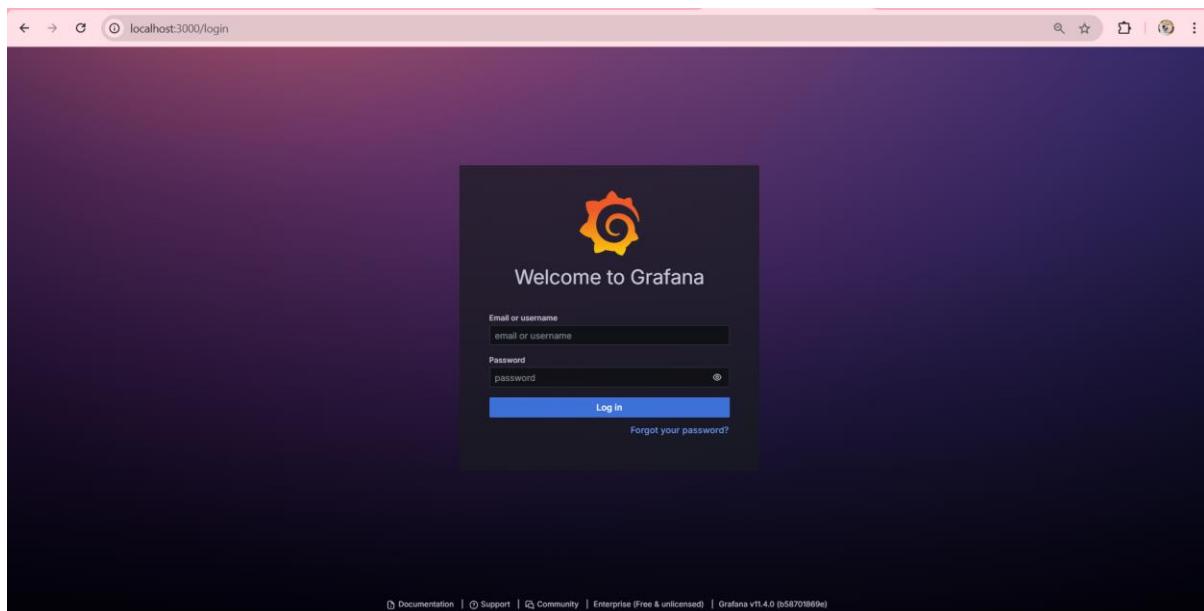
## 4.2 Tích hợp Grafana

Tương tự với Prometheus, cài đặt Grafana trên Window Server:

Tải về Grafana phù hợp với máy chủ: [Download Grafana | Grafana Labs](https://grafana.com/grafana/download)

The screenshot shows the Grafana Labs download page. At the top, there's a navigation bar with links like Products, Open Source, Solutions, Learn, Docs, Company, Downloads, Contact us, and Sign in. Below the navigation is a yellow header bar with the Grafana logo and a 'Download' button. The main content area is titled 'Download Grafana'. It features a 'Grafana Cloud' section with a note about using Grafana Cloud for installation and scaling. Below this, there's a dropdown for 'Version' set to '11.4.0' and a dropdown for 'Edition' set to 'Enterprise'. A note states that the Enterprise Edition is the default and recommended edition. Underneath are details: License is 'Grafana Labs License', Release Date is 'December 6, 2024', and Release Info is 'What's New In Grafana 11.4.0'. At the bottom, there are download links for 'Linux', 'Windows', 'Mac', 'Docker', and 'Linux on ARM64'. A note at the very bottom says 'Ubuntu and Debian (64 Bit) SHA256: 0d0fd961b7a86d00b5d28344242e89352c40f73e7c05329852d76d896ea52'.

Grafana được chạy trên Port 3000 của Window Server với tài khoản mặc định admin:



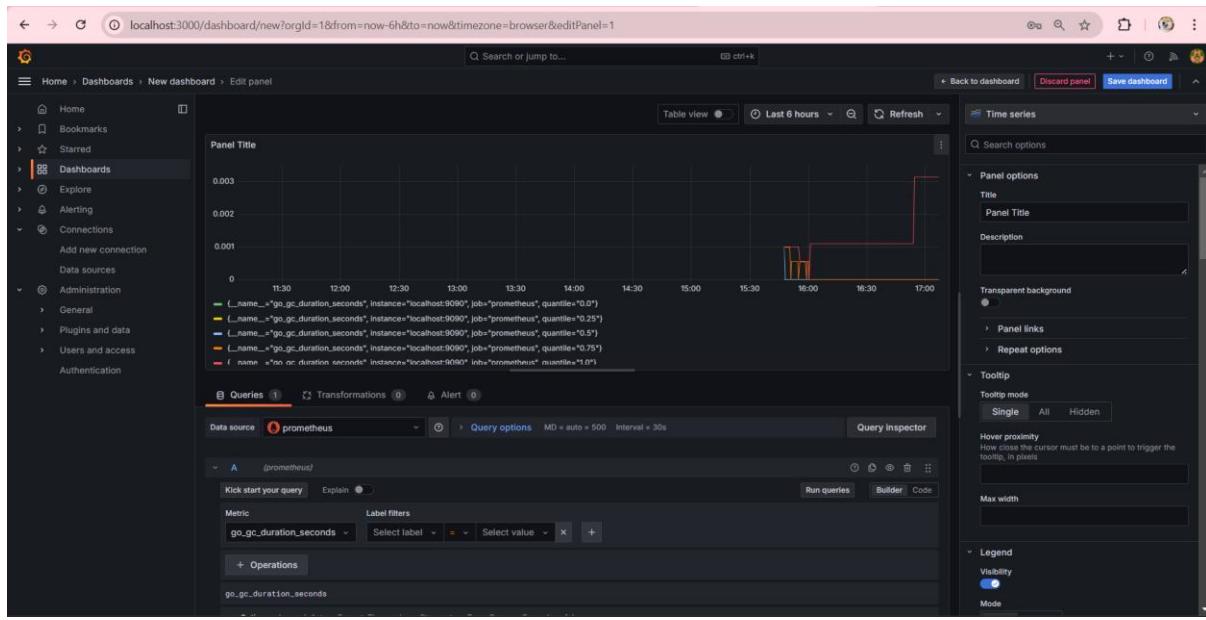
## Giao diện chính của Grafana:

The screenshot shows the Grafana home page at [localhost:3000](http://localhost:3000/). The left sidebar includes sections for Home, Bookmarks, Starred, Dashboards, Explore, Alerting, Connections (selected), Data sources, Administration, General, Plugins and data, Users and access, and Authentication. The main content area features a "Welcome to Grafana" banner with links to "Basic" setup steps (Tutorial, Data Source and Dashboards, Grafana fundamentals), a "COMPLETE" step for adding a data source, and a "DASHBOARDS" step for creating a first dashboard. Below this is a "Dashboards" section with links to "Starred dashboards" and "Recently viewed dashboards". On the right, there's a "Latest from the blog" section with two posts: one about Grafana's holistic approach to observability and another about monitoring local weather.

Kết nối Grafana với Prometheus trong Connections > Data Source, thiết lập Prometheus với Port 9090 và url của Window Server:

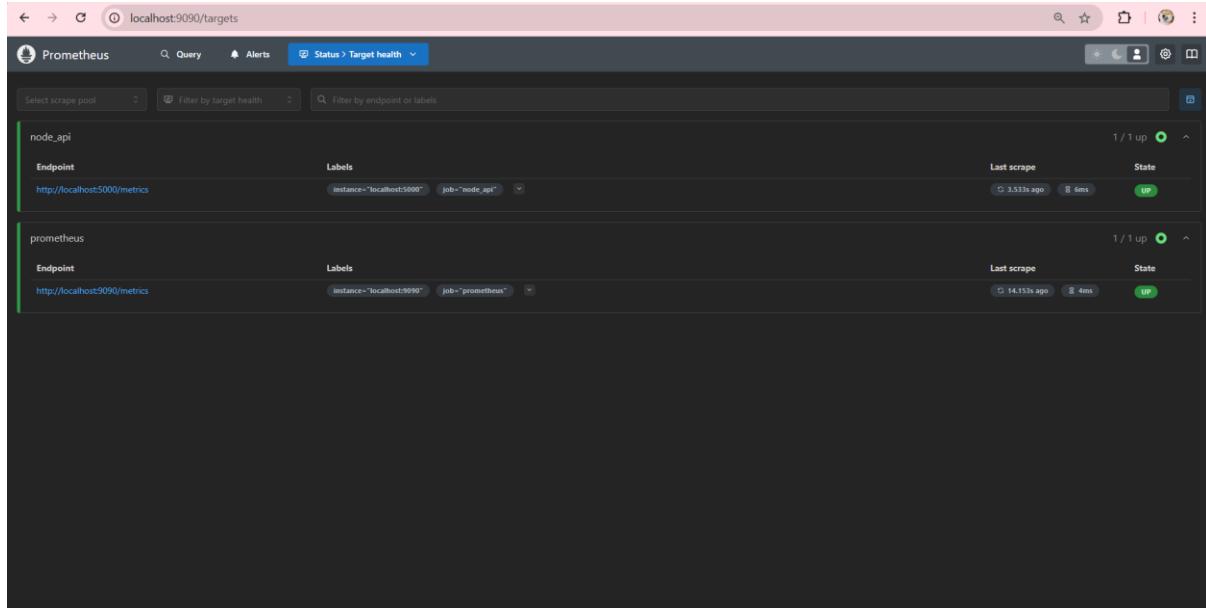
The screenshot shows the "Connections > Data sources" page for Prometheus. The left sidebar is identical to the home page. The main area shows a "prometheus" data source entry with the following details: Type: Prometheus, Status: Supported. Under "Connection", the "Prometheus server URL" field is set to "http://localhost:9090". Under "Authentication", the dropdown is set to "No Authentication". A note at the bottom states: "Before you can use the Prometheus data source, you must configure it below or in the config file. For detailed instructions, [view the documentation](#)". Fields marked with \* are required.

Test thử kết nối với truy vấn hiển thị Dashboard CPU:



## 5. Một số kịch bản với Monitoring

Khi hệ thống đã được triển khai, ta có thể check hệ thống API có đang chạy ổn định:



Hệ thống node\_api đang có trạng thái hoạt động (state = up)

Khi ta ngắt hệ thống:

The screenshot shows the Prometheus interface under the 'Status' tab, specifically the 'Target health' section. There are two entries:

- node\_api**:
  - Endpoint: http://localhost:5000/metrics
  - Labels: instance=“localhost:5000”, job=“node\_api”
  - Last scrape: 8.895s ago
  - State: DOWN (red circle)
  - Error message: "Error scraping target: Get “http://localhost:5000/metrics”: dial tcp [::]:5000: connecte: No connection could be made because the target machine actively refused it."
- prometheus**:
  - Endpoint: http://localhost:9090/metrics
  - Labels: instance=“localhost:9090”, job=“prometheus”
  - Last scrape: 4.516s ago
  - State: UP (green circle)

Hệ thống node\_api đang có trạng thái tạm dừng (state = down)

Ở phần query của Prometheus, ta có thể dùng các câu truy vấn:

The screenshot shows the Prometheus interface under the 'Query' tab. A single query is entered in the text area:

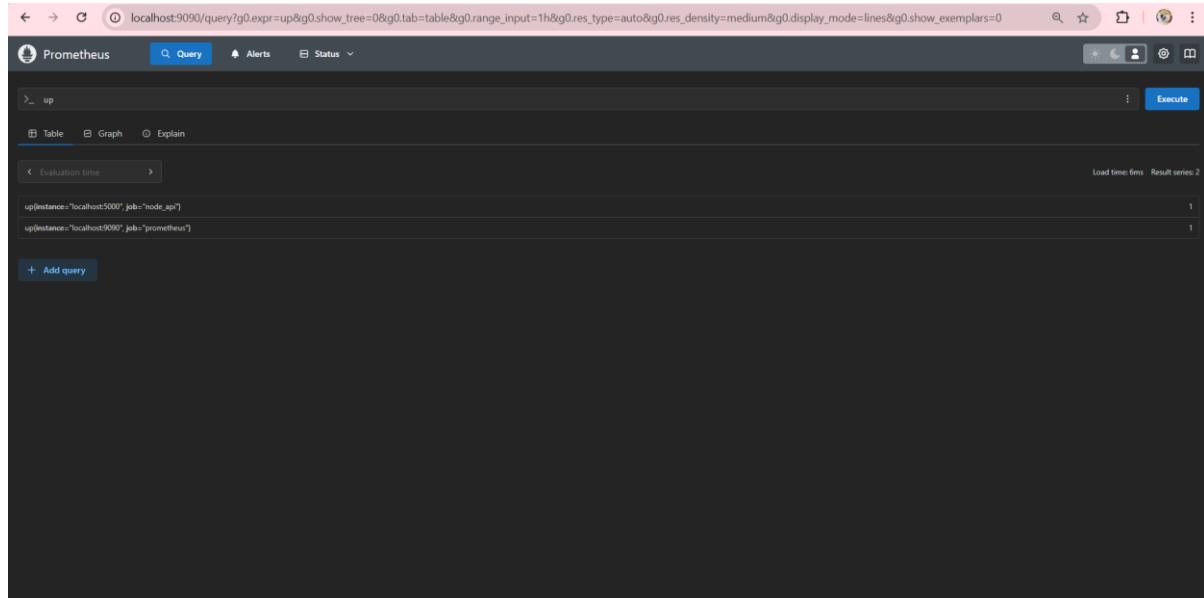
```
localhost:9090/query?g0.expr=&g0.show_tree=0&g0.tab=table&g0.range_input=1h&g0.res_type=auto&g0.res_density=medium&g0.display_mode=lines&g0.show_exemplars=0
```

The results are displayed in a table:

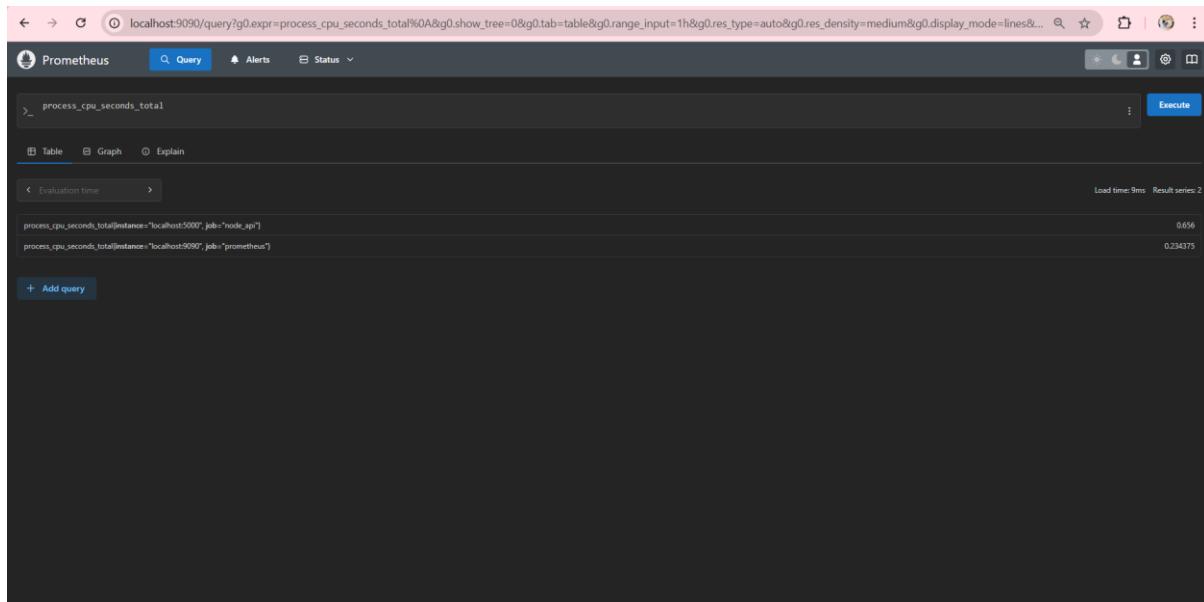
Instance	job	method	route	status
localhost:5000	node_api	GET	metrics	200

Load time: 9ms Result series: 1

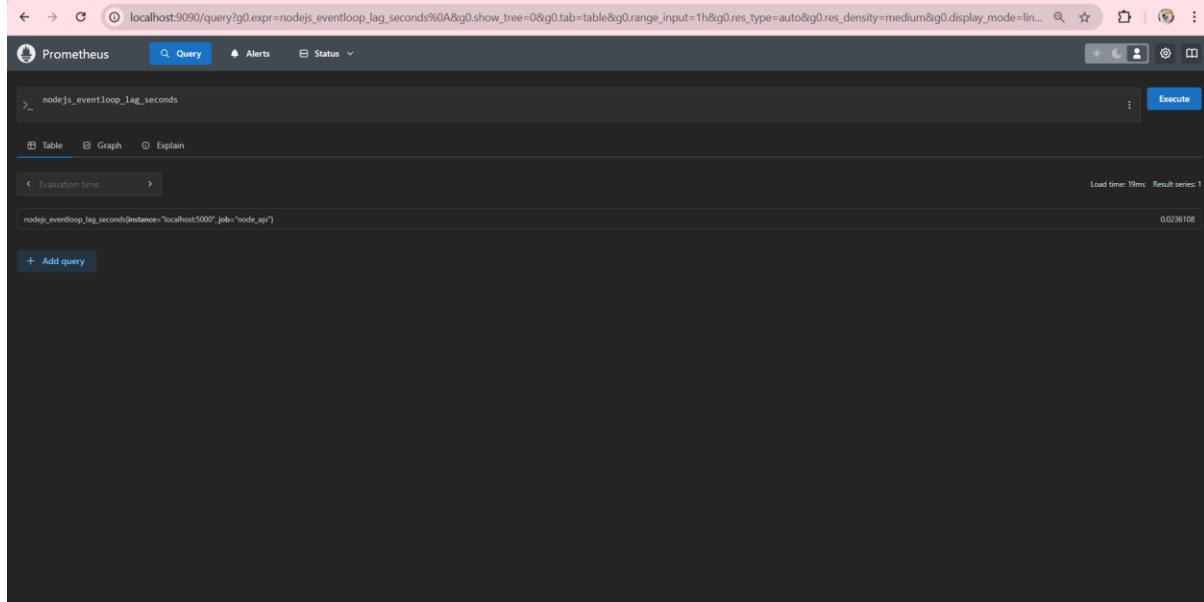
Kiểm tra các hệ thống được thu thập log đang có trạng thái bật, dưới đây, Prometheus đang theo dõi hệ thống api node\_api và theo dõi chính nó



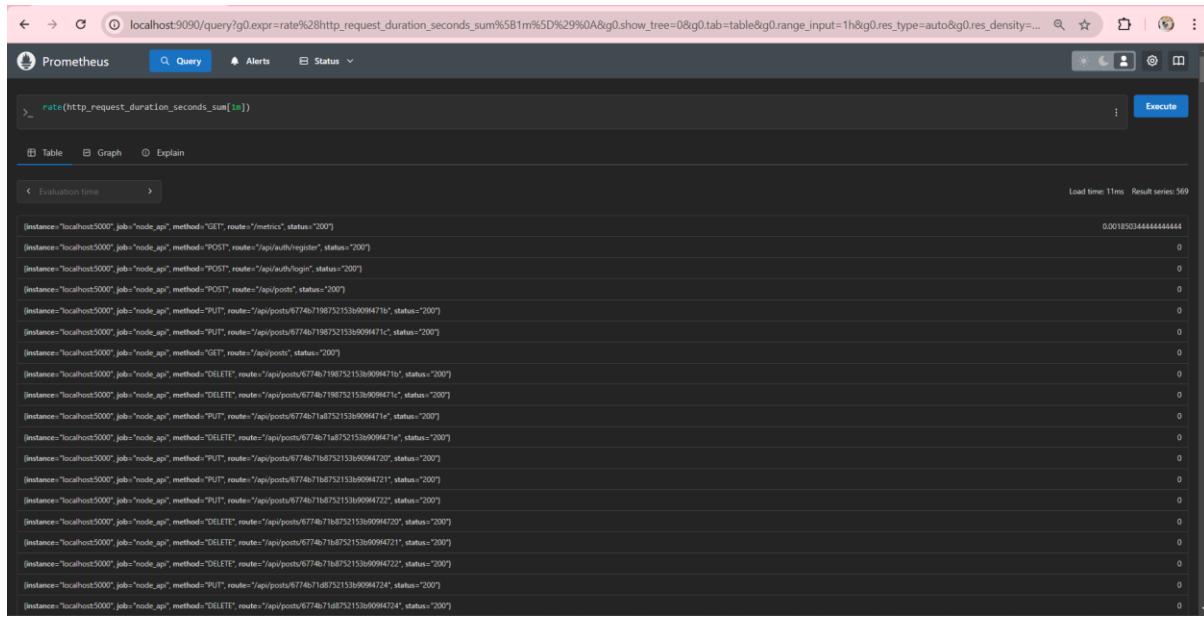
Kiểm tra cpu của các hệ thống, node\_api đang dùng 60% CPU:



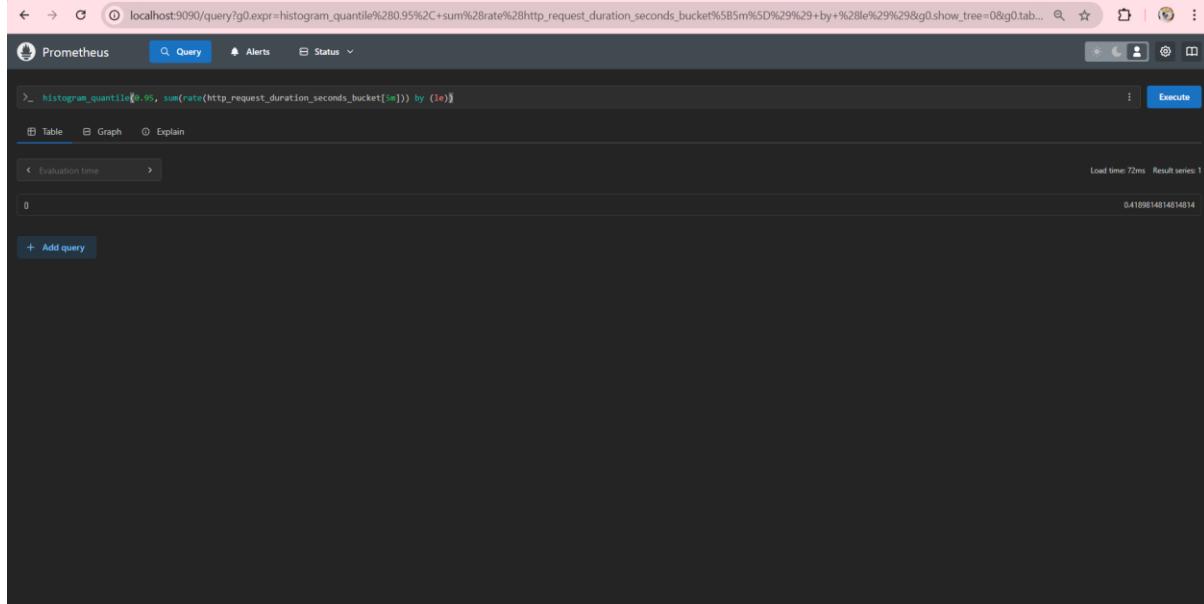
Đối với ứng dụng nodejs được cài đặt bên trên, kiểm tra độ trễ giữa các vòng lặp của ứng dụng, thời gian 0.02s:



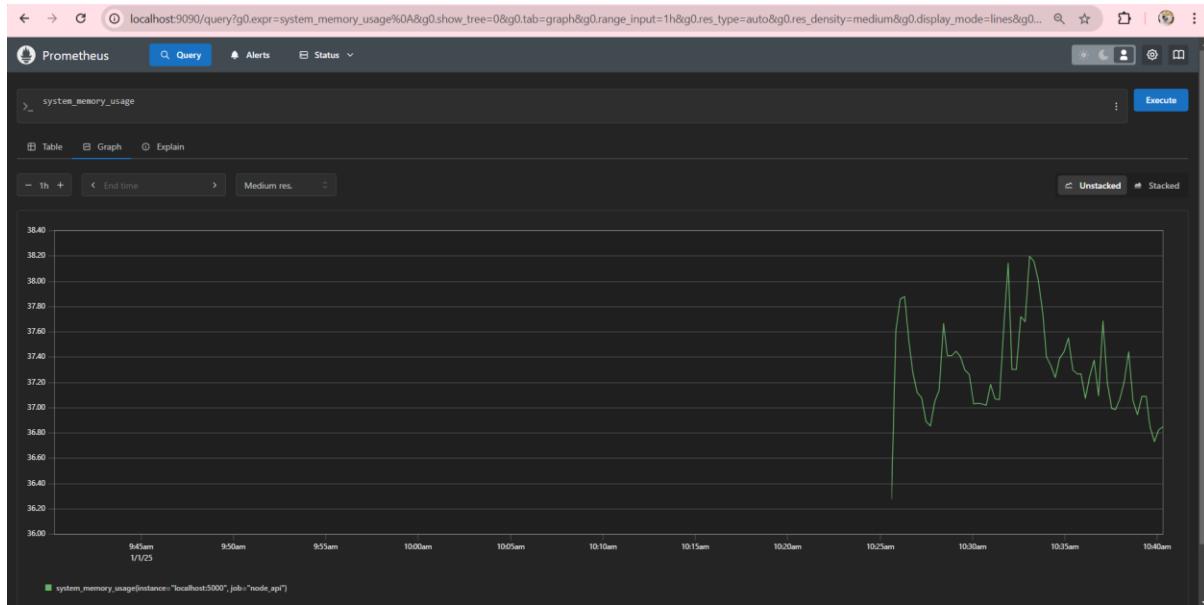
Tốc độ xử lý yêu cầu http trong vòng 1 phút, api lấy danh sách được dùng nhiều và thời gian xử lý trung bình thường là 0.2s:



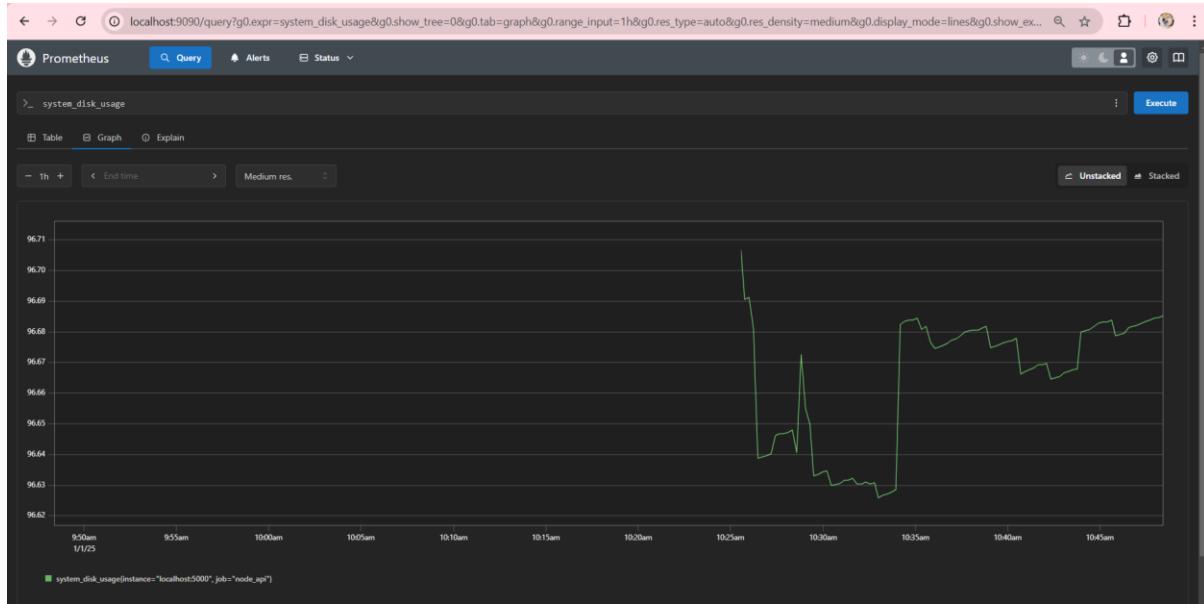
Thời gian xử lý yêu cầu http trong 95% trường hợp, trung bình thời gian xử lý là 0.4s:



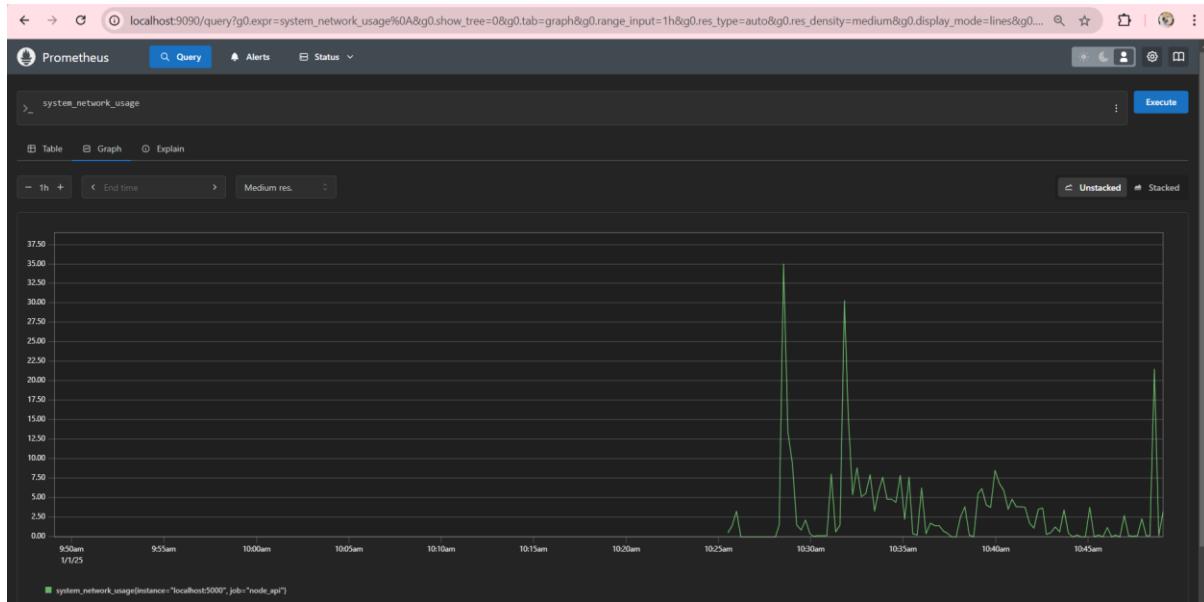
Tỉ lệ sử dụng ram hiện tại, ram luôn dưới 40%, hệ thống đang có tính ổn định:



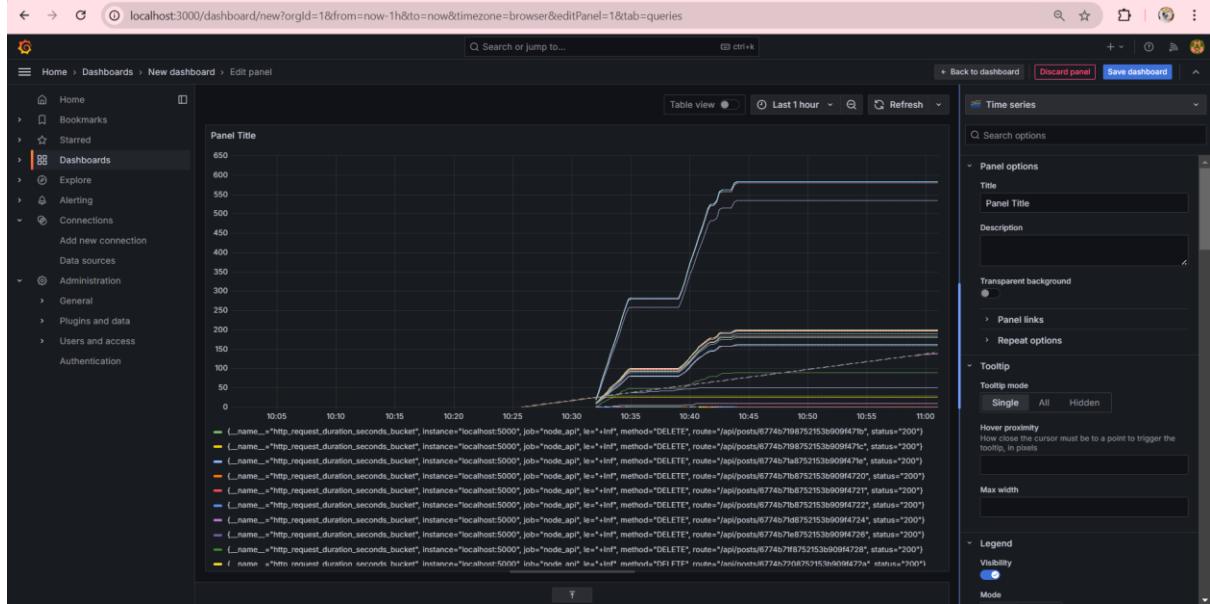
Tỉ lệ sử dụng disk:



Tốc độ mạng trên hệ thống, tốc độ mạng không có sự ổn định, chạm mốc 2mb/s là mốc khá thấp và cần được cải thiện:



Ngoài ra, ta có thể sử dụng Grafana để hiển thị các biểu đồ, trực quan hóa dữ liệu, dưới đây là biểu đồ thể hiện số lượng yêu cầu theo thời gian. Có thể thấy số lượng API lấy danh sách bài viết hay lấy chi tiết bài viết luôn vượt trội so với các API đăng ký, đăng nhập,....



# V. KẾT QUẢ VÀ ĐÁNH GIÁ

## 1. Kết Quả

Sau quá trình nghiên cứu và triển khai, các kết quả chính đã đạt bao gồm:

Hệ thống CI/CD được triển khai thành công:

- Quy trình CI/CD được cài đặt và tích hợp thành công trên các công cụ GitLab, GitHub Actions, và Jenkins, DockerHub
- Mã nguồn được tự động build, test, và deploy đến môi trường production mỗi khi có thay đổi trong repository.

Hệ thống monitoring được tích hợp:

- Sử dụng Grafana và Prometheus, các dashboard giúp theo dõi hiệu năng hệ thống được thiết lập.
- Các metric như CPU, RAM, Disk Usage, và Network Activity được thu thập và hiển thị theo thời gian thực.

Quy trình DevOps được chuẩn hóa:

- Môi trường Dev, Staging, và Production được thiết lập một cách khoa học.
- Quy trình CI/CD giúp giảm thời gian triển khai đồng thời đảm bảo chất lượng phân phát.

Tính khả thi và hiệu quả cao:

- Hệ thống mẫu hoạt động đồng bộ, đáp ứng tốt nhu cầu của quy trình DevOps.
- Kết quả thu được gợi ý cho nhiều dự án thực tế.

## 2. Đánh Giá

Hiệu quả và đóng góp của quy trình CI/CD:

- Quy trình CI/CD đã giúp tự động hóa toàn bộ các khâu build, test, và deploy, giảm tối đa khả năng sai sót nhân tạo trong các quy trình lặp lại.
- Giúp giảm thời gian triển khai từ vài ngày xuống chỉ còn vài giờ, đảm bảo tính đồng bộ của các môi trường.

- Sử dụng Docker và DockerHub giúp đồng bộ hóa và tái sử dụng các container đã được build.

Lợi ích của monitoring:

- Monitoring giúp phát hiện sớm các vấn đề như tự dung CPU, RAM, hoặc sự gia tăng bất thường trong tài nguyên.
- Dữ liệu từ Prometheus được đánh giá cao về khả năng lưu trữ và truy vấn nhanh chóng.
- Grafana có khả năng tùy biến dashboard, giúp nhóm phát triển có cái nhìn tổng quan và chi tiết về hệ thống.

Hạn chế của hệ thống hiện tại:

- Việc thiết lập monitoring và CI/CD ban đầu đòi hỏi nhiều công sức và kiến thức chuyên môn.
- Chưa tích hợp sâu các công cụ AI/ML để tự động dự báo vấn đề.
- Cần nâng cao khả năng giám sát log chi tiết hơn bằng các công cụ như ELK Stack (Elasticsearch, Logstash, Kibana).

Hướng phát triển trong tương lai:

- Triển khai các công cụ machine learning để phân tích xu hướng tài nguyên và tự động phát hiện nguy cơ.
- Đồng bộ hóa nhiều hơn với các công nghệ orchestration như Kubernetes để quản lý container hiệu quả.
- Tích hợp thêm nhiều đầu vào monitoring như log chi tiết, sự kiện runtime, và tracing trong microservices.