TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**SỬ THỊ YẾN LINH – 52300218**

**LÊ THỊ MỸ PHỤNG – 52200264**

**BÁO CÁO CUỐI KỲ**

**ĐIỆN TOÁN ĐÁM MÂY**

**XÂY DỰNG MÔ PHỎNG HỆ THỐNG MINICLOUD CƠ BẢN**

Người hướng dẫn

**TS. Trần Chí Thiện**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2025**

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**SỬ THỊ YẾN LINH – 52300218**

**LÊ THỊ MỸ PHỤNG – 52200264**

**BÁO CÁO CUỐI KỲ**

**ĐIỆN TOÁN ĐÁM MÂY**

**XÂY DỰNG MÔ PHỎNG HỆ THỐNG MINICLOUD CƠ BẢN**

Người hướng dẫn

**TS. Trần Chí Thiện**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2025**

**LỜI CẢM ƠN**

*Kính gửi thầy Trần Chí Thiện,*

*Em xin gửi tới thầy lời tri ân sâu sắc và chân thành nhất vì sự tận tâm, nhiệt huyết và kiến thức quý báu mà thầy đã truyền đạt cho chúng em trong suốt học phần* ***Cloud Computing.***

*Trong quá trình học, em không chỉ được tiếp cận với những khái niệm, mô hình và công nghệ cốt lõi của điện toán đám mây, mà còn hiểu rõ hơn về cách tư duy thiết kế và vận hành một hệ thống thực tế. Em đặc biệt ấn tượng với cách thầy truyền đạt kiến thức luôn rõ ràng, mạch lạc, đi từ nền tảng đến nâng cao, kết hợp giữa lý thuyết và các tình huống ứng dụng cụ thể.*

*Em thật sự trân trọng khoảng thời gian được học với thầy, trân trọng những góp ý chi tiết trên bài làm, những lưu ý nhỏ trong quá trình triển khai hệ thống, cũng như sự nghiêm túc nhưng luôn tạo điều kiện để sinh viên có thể chủ động tìm tòi và phát triển. Những kinh nghiệm và kiến thức mà thầy chia sẻ chắc chắn sẽ là hành trang quý giá cho em trong các học phần tiếp theo cũng như trên con đường nghề nghiệp sau này.*

*Một lần nữa, em xin chân thành cảm ơn thầy vì tất cả sự tận tình, kiên nhẫn và tâm huyết dành cho chúng em trong suốt học kỳ vừa qua. Kính chúc thầy thật nhiều sức khỏe, luôn giữ vững nhiệt huyết với nghề và gặt hái thêm nhiều thành công trong công tác giảng dạy và nghiên cứu.*

*TP. Hồ Chí Minh, ngày 2 tháng 12 năm 2025*

*Tác giả*

*Tác giả 1 Tác giả 2*

*Sử Thị Yến Linh Lê Thị Mỹ Phụng*

**CÔNG TRÌNH ĐƯỢC HOÀN THÀNH**

**TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi và được sự hướng dẫn khoa học của TS. Trần Chí Thiện. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong Dự án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

**Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung Dự án của mình**. Trường Đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

*TP. Hồ Chí Minh, ngày 2 tháng 12 năm 2025*

*Tác giả 1 Tác giả 2*

*Sử Thị Yến Linh Lê Thị Mỹ Phụng*

**MỤC LỤC**

**[DANH MỤC HÌNH ẢNH 4](#_Toc23355)**

**[CHƯƠNG 1. GIỚI THIỆU VÀ MỤC TIÊU 1](#_Toc21657)**

**[1.1 Bối cảnh và tổng quan về điện toán đám mây 1](#_Toc30380)**

**[1.2 Mô hình MiniCloud và bài toán MyMiniCloud 1](#_Toc28380)**

**[1.3 Nền tảng kỹ thuật sử dụng 3](#_Toc4165)**

[1.3.1 Container và Docker 3](#_Toc24206)

[1.3.2 Docker Compose và Docker Network 4](#_Toc10968)

[1.3.3 Các thành phần nền tảng 4](#_Toc29820)

**[1.4 Mục tiêu học tập và phạm vi thực hiện 4](#_Toc20533)**

[1. Hiểu và mô phỏng kiến trúc cloud 5](#_Toc1672)

[2. Thực hành container hóa và orchestration 5](#_Toc27847)

[3. Thực hành các dịch vụ nền tảng cloud 5](#_Toc11604)

[4. Thực hành reverse proxy và API gateway 5](#_Toc9468)

[5. Kỹ năng thiết kế, triển khai và đánh giá hệ thống 5](#_Toc6280)

**[1.5 Cấu trúc báo cáo 5](#_Toc23308)**

**[CHƯƠNG 2. KIẾN TRÚC VÀ SƠ ĐỒ HỆ THỐNG 7](#_Toc30206)**

[2.1 Cơ sở lý thuyết về kiến trúc cloud và microservices 7](#_Toc4058)

**[2.1.1 Kiến trúc nhiều lớp (multi‑tier) 7](#_Toc2971)**

**[2.1.2 Microservices và reverse proxy / API gateway 7](#_Toc24585)**

**[2.1.3 Monitoring và observability 7](#_Toc24734)**

[2.2 Kiến trúc logic của MyMiniCloud 8](#_Toc9844)

**[2.2.1 Tầng Client 8](#_Toc31396)**

**[2.2.2 Tầng Reverse Proxy / API Gateway 9](#_Toc22442)**

**[2.2.3 Tầng dịch vụ ứng dụng 9](#_Toc5060)**

**[2.2.4 Tầng dữ liệu và lưu trữ 10](#_Toc1574)**

**[2.2.5 Tầng monitoring và visualization 10](#_Toc6749)**

**[2.2.6 Tầng DNS nội bộ 10](#_Toc15699)**

[2.3 Kiến trúc triển khai trên Docker và sơ đồ network 11](#_Toc4244)

**[2.3.1 File docker-compose và dịch vụ 11](#_Toc4506)**

[2.4 Sơ đồ flow request giữa các server 13](#_Toc20633)

[2.5 Đánh giá kiến trúc MyMiniCloud theo yêu cầu đề 20](#_Toc30230)

**[CHƯƠNG 3. CẤU HÌNH & DOCKER FILE 21](#_Toc9252)**

**[3.1 Cấu trúc thư mục dự án 21](#_Toc11418)**

**[3.2 File docker-compose và định nghĩa service 21](#_Toc20684)**

**[3.3 Dockerfile của từng nhóm server 23](#_Toc2858)**

**[3.4 Các file cấu hình chính 25](#_Toc2085)**

**[CHƯƠNG 4. DEMO & KIỂM THỬ 26](#_Toc27170)**

**[4.1 Môi trường chạy và cách khởi động hệ thống 26](#_Toc25036)**

**[4.2 Kiểm thử chức năng từng server 26](#_Toc26776)**

**[CHƯƠNG 5. ĐÁNH GIÁ VÀ PHÂN TÍCH 28](#_Toc30406)**

**[5.1 Mức độ đáp ứng yêu cầu đề tài 28](#_Toc24762)**

**[5.2 Ưu điểm của hệ thống 31](#_Toc12505)**

[5.2.1 Kiến trúc mô phỏng sát thực tế 31](#_Toc29752)

[5.2.2 Môi trường triển khai thống nhất, dễ tái lập 31](#_Toc13781)

[5.2.3 Tích hợp đầy đủ monitoring và visualization 32](#_Toc6402)

[5.2.4 Gắn kết tốt giữa các thành phần 32](#_Toc21980)

**[5.3 Hạn chế của hệ thống 32](#_Toc4316)**

**[5.4 Khó khăn gặp phải trong quá trình thực hiện 33](#_Toc23847)**

[5.4.1 Khối lượng công nghệ lớn trong thời gian ngắn 33](#_Toc19087)

[5.4.2 Cấu hình và debug network Docker 33](#_Toc27884)

[5.4.3 Thiết kế sơ đồ và trình bày báo cáo 34](#_Toc25035)

**[5.5 Hướng phát triển và mở rộng 34](#_Toc23611)**

[1. Triển khai trên môi trường cloud thực tế 34](#_Toc23285)

[2. Nâng cấp kiến trúc HA và scale-out 34](#_Toc26902)

[3. Bổ sung logging và tracing 34](#_Toc25201)

[4. Tích hợp Auth sâu hơn vào ứng dụng 35](#_Toc794)

[5. Hoàn thiện quy trình DevOps 35](#_Toc21285)

**[PHỤ LỤC 36](#_Toc1118)**

# DANH MỤC HÌNH ẢNH

[Hình 1: Kiến trúc logic dịch vụ ứng dụng MyMiniCloud 8](#_Toc215579856)

[Hình 2: Kiến trúc Internal DNS và hệ thống giám sát MyMiniCloud 8](#_Toc215579857)

[Hình 3: Sơ đồ network Docker cloud-net 13](#_Toc215579858)

[Hình 4. Flow request truy cập trang chủ và blog qua API Gateway. 14](#_Toc215579859)

[Hình 5: Flow request API /student qua API Gateway tới Application Server và nguồn dữ liệu. 15](#_Toc215579860)

[Hình 6:Flow đăng nhập người dùng vào hệ thống thông qua API Gateway và Keycloak 17](#_Toc215579861)

[Hình 7: Flow truy cập file tĩnh trên MinIO qua API Gateway. 18](#_Toc215579862)

[Hình 8: Flow thu thập và trực quan hóa metrics giữa Node Exporter, Prometheus và Grafana 20](#_Toc215579863)

# GIỚI THIỆU VÀ MỤC TIÊU

## Bối cảnh và tổng quan về điện toán đám mây

Điện toán đám mây (Cloud Computing) hiện là nền tảng hạ tầng chủ đạo cho hầu hết các hệ thống thông tin hiện đại. Theo định nghĩa kinh điển của NIST (National Institute of Standards and Technology), cloud computing là “mô hình cho phép truy cập thuận tiện, theo yêu cầu (on‑demand) qua mạng tới một tập hợp tài nguyên tính toán có thể cấu hình (ví dụ: mạng, máy chủ, lưu trữ, ứng dụng, dịch vụ) được cung cấp và thu hồi nhanh chóng với nỗ lực quản trị tối thiểu từ phía nhà cung cấp”.

NIST chỉ ra 5 đặc tính cốt lõi của cloud:

* **On‑demand self‑service** – người dùng tự cấp phát tài nguyên khi cần.
* **Broad network access** – truy cập qua mạng bằng nhiều loại thiết bị.
* **Resource pooling** – tài nguyên được dùng chung và trừu tượng hóa.
* **Rapid elasticity** – mở rộng/thu hẹp linh hoạt.
* **Measured service** – sử dụng được đo lường và giám sát.

Song song đó, các nhà cung cấp cloud (AWS, Azure, GCP) thường cung cấp dịch vụ theo ba mô hình: IaaS (hạ tầng), PaaS (nền tảng), SaaS (phần mềm), triển khai trên nhiều vùng (region, availability zone) với các thành phần quen thuộc như web server, application server, database, hệ thống xác thực, lưu trữ đối tượng, DNS nội bộ, giám sát và reverse proxy / load balancer.

Tuy nhiên, việc tiếp cận trực tiếp hạ tầng cloud thật có một số hạn chế: chi phí sử dụng tài nguyên, độ phức tạp của kiến trúc và mức trừu tượng cao khiến người học khó quan sát chi tiết cấu hình từng dịch vụ.

## Mô hình MiniCloud và bài toán MyMiniCloud

Để khắc phục các hạn chế trên, đề tài môn học yêu cầu sinh viên xây dựng một hệ thống **MiniCloud cơ bản** – gọi là **MyMiniCloud** – chạy hoàn toàn trên môi trường cục bộ, nhưng mô phỏng tương đối đầy đủ các thành phần của một Cloud Platform thực tế.

* Xây dựng hệ thống Cloud thu nhỏ gồm 9 loại máy chủ bắt buộc, mỗi loại thể hiện một vai trò điển hình trong kiến trúc cloud:

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Loại server** | **Vai trò chính** | **Ví dụ công nghệ / image** |
| 1 | Web Server | Cung cấp web tĩnh / blog | Nginx, Apache |
| 2 | Application Server | Xử lý nghiệp vụ, trả JSON/API | Python Flask, Node.js |
| 3 | Database Server | Lưu trữ dữ liệu quan hệ / NoSQL | MySQL, MariaDB, Postgres |
| 4 | Authentication / Identity Server | Xác thực, single sign‑on | Keycloak, Flask Auth |
| 5 | File / Object Storage Server | Lưu trữ đối tượng, file, tài nguyên tĩnh | MinIO, Nextcloud |
| 6 | DNS / Name Service | Quản lý ánh xạ tên miền nội bộ | BIND9, CoreDNS |
| 7 | Monitoring Server | Theo dõi CPU, RAM, network, container | Prometheus + Node Exporter |
| 8 | Logging / Visualization Server | Lưu và hiển thị log / metric | Grafana, Loki, ELK |
| 9 | Reverse Proxy / Load Balancer | Điều phối truy cập, cân bằng tải | Nginx, HAProxy, Traefik |

* Mỗi loại server được triển khai bằng một container riêng, kết nối với nhau thông qua mạng Docker nội bộ cloud-net, khởi động đồng bộ bằng một file docker-compose.yml duy nhất.
* (Khuyến khích) Triển khai toàn bộ cụm lên máy ảo AWS EC2 để có trải nghiệm gần với cloud thật.

Hệ thống **MyMiniCloud** – hiện thực 9 loại server nói trên bằng các dịch vụ cụ thể:

* web-frontend-server (Nginx tĩnh) – Web Server.
* application-backend-server (Flask) – App Server.
* relational-database-server (MariaDB) – Database Server.
* authentication-identity-server (Keycloak) – Auth Server.
* object-storage-server (MinIO) – Object Storage.
* internal-dns-server (Bind9) – DNS nội bộ.
* monitoring-node-exporter-server & monitoring-prometheus-server Monitoring.
* monitoring-grafana-dashboard-server – Visualization.
* api-gateway-proxy-server (Nginx) – Reverse Proxy / API Gateway.

## Nền tảng kỹ thuật sử dụng

### Container và Docker

**Container** là đơn vị phần mềm tiêu chuẩn đóng gói mã nguồn và toàn bộ phụ thuộc (runtime, thư viện, công cụ hệ thống, cấu hình) để ứng dụng có thể chạy nhất quán trên nhiều môi trường khác nhau. Docker là nền tảng container phổ biến nhất hiện nay, cho phép xây dựng image, khởi tạo container và quản lý vòng đời ứng dụng một cách tự động, reproducible.

Trong MyMiniCloud, mỗi server (web, app, db,…) được đóng gói thành một Docker image riêng; khi khởi động, image sinh ra container chạy trên Docker Engine. Docker giúp sinh viên:

* Tách biệt môi trường giữa các service, tránh xung đột thư viện.
* Khởi tạo/dừng toàn bộ hệ thống bằng một lệnh docker compose up -d.

### Docker Compose và Docker Network

Docker Compose cho phép định nghĩa nhiều service trong một file YAML, nêu rõ image, port, volume, biến môi trường, cùng các network mà service tham gia.

MyMiniCloud sử dụng một network bridge chung cloud-net, để mọi container có thể gọi nhau qua tên service (DNS nội bộ của Docker) thay vì IP tĩnh. Đây là thực hành quan trọng giúp hiểu cơ chế service discovery trong môi trường microservices.

### Các thành phần nền tảng

Một số công cụ chính được sử dụng:

* Nginx: web server hiệu năng cao, đồng thời là reverse proxy và load balancer. Nginx thường được cấu hình làm reverse proxy đứng trước các ứng dụng backend, nhận request và phân phối tới các upstream server phù hợp.
* Prometheus: bộ công cụ monitoring và alerting mã nguồn mở, thu thập số liệu dạng time‑series bằng cách “scrape” HTTP endpoint từ các target (Node Exporter, ứng dụng) và lưu vào cơ sở dữ liệu time‑series riêng.
* Grafana: nền tảng trực quan hóa dữ liệu, cho phép truy vấn, vẽ biểu đồ, dashboard và thiết lập cảnh báo dựa trên các nguồn dữ liệu như Prometheus, ElasticSearch, Loki,…
* MinIO: hệ thống object storage mã nguồn mở, tương thích Amazon S3, tối ưu cho workload cloud‑native và có thể triển khai cục bộ ở quy mô nhỏ đến rất lớn.
* Keycloak: giải pháp Identity and Access Management (IAM) mã nguồn mở, hỗ trợ chuẩn OpenID Connect, OAuth 2.0, SAML 2.0, cung cấp SSO, quản lý user, client và token cho ứng dụng hiện đại.
* BIND9: phần mềm DNS server phổ biến, cho phép quản lý zone, bản ghi A, CNAME…, dùng rộng rãi trong môi trường Internet và nội bộ.

Việc lựa chọn bộ công cụ này bám sát gợi ý trong đề, đồng thời giúp sinh viên tiếp xúc với các giải pháp đang được áp dụng rất nhiều trong thực tế DevOps / SRE.

## Mục tiêu học tập và phạm vi thực hiện

Dựa trên đề bài và triển khai thực tế, các mục tiêu chính của đề tài có thể tóm tắt như sau:

1. Hiểu và mô phỏng kiến trúc cloud
   * Nhận diện được vai trò của 9 thành phần cơ bản trong một nền tảng cloud.
   * Nắm được luồng giao tiếp giữa client, reverse proxy, web/app, database, storage, DNS, monitoring.
2. Thực hành container hóa và orchestration
   * Biết cách viết Dockerfile cho web/app.
   * Biết cách sử dụng docker-compose.yml để khởi động cụm dịch vụ, cấu hình network, volume, restart policy.
3. Thực hành các dịch vụ nền tảng cloud
   * Xây dựng website tĩnh và blog.
   * Tạo API backend, đọc dữ liệu JSON và SQL.
   * Cài đặt Keycloak, tạo realm, user, client.
   * Sử dụng MinIO để tạo bucket, upload object và chia sẻ qua URL public.
   * Cấu hình Bind9 cho zone nội bộ cloud.local.
   * Cấu hình Prometheus + Node Exporter + Grafana để thu thập và trực quan hóa metrics CPU, RAM, Network.
4. Thực hành reverse proxy và API gateway
   * Cấu hình Nginx làm entry point, route /, /api/, /auth/, /storage/, /student/ tới backend tương ứng.
   * (Tùy chọn) Cấu hình load balancing round robin giữa hai web server.
5. Kỹ năng thiết kế, triển khai và đánh giá hệ thống
   * Thiết kế kiến trúc, sơ đồ network, sơ đồ flow request.
   * Viết báo cáo kỹ thuật chuẩn mực, có cấu trúc 5 chương, thể hiện rõ phần core và phần mở rộng đã thực hiện.

Báo cáo tập trung mô tả đầy đủ 9 server trong môi trường cục bộ; phần triển khai trên AWS EC2.

## Cấu trúc báo cáo

Báo cáo được tổ chức theo yêu cầu:

* Chương 1: Giới thiệu và mục tiêu: Bối cảnh điện toán đám mây, mô hình MyMiniCloud, mục tiêu học tập và phạm vi.
* Chương 2: Kiến trúc & sơ đồ: Kiến trúc logic, kiến trúc triển khai, sơ đồ network và flow request.
* Chương 3: Cấu hình & Dockerfile: Trình bày chi tiết các tệp cấu hình cho từng server.
* Chương 4: Demo & kiểm thử: Ảnh chụp, kết quả lệnh kiểm thử tương ứng 9 server.
* Chương 5: Đánh giá & phân tích: Nhận xét về ưu điểm, khó khăn, bài học và hướng phát triển.

# KIẾN TRÚC VÀ SƠ ĐỒ HỆ THỐNG

## Cơ sở lý thuyết về kiến trúc cloud và microservices

### Kiến trúc nhiều lớp (multi‑tier)

Các hệ thống web hiện đại thường được xây dựng theo kiến trúc nhiều lớp:

* Lớp trình diễn (presentation layer): giao diện người dùng (browser, mobile app).
* Lớp logic nghiệp vụ (application layer): nơi đặt các API và business logic.
* Lớp dữ liệu (data layer): hệ quản trị CSDL hoặc dịch vụ lưu trữ.

Mô hình này giúp tách biệt mối quan tâm (separation of concerns): mỗi lớp đảm nhiệm một vai trò rõ ràng, dễ mở rộng và bảo trì. Khi chuyển sang môi trường cloud, mỗi lớp thường được triển khai thành nhiều service tách biệt (microservices), giao tiếp qua HTTP/REST hoặc gRPC.

### Microservices và reverse proxy / API gateway

Trong microservices, ứng dụng được chia nhỏ thành các dịch vụ độc lập, mỗi dịch vụ triển khai một tập chức năng cụ thể và có thể được phát triển, triển khai, mở rộng riêng biệt. Để client không phải biết chi tiết từng địa chỉ dịch vụ, hệ thống thường sử dụng reverse proxy hoặc API gateway làm điểm vào chung.

Reverse proxy là máy chủ đặt giữa client và một hoặc nhiều server phía sau; nó nhận request từ client và chuyển tiếp đến server thích hợp, đồng thời có thể thực hiện caching, nén, cân bằng tải, SSL termination, chỉnh sửa header. Nginx là một trong những reverse proxy phổ biến nhất hiện nay. Trong môi trường cloud, reverse proxy thường được xem như “cổng” (gateway) của toàn bộ hệ thống.

### Monitoring và observability

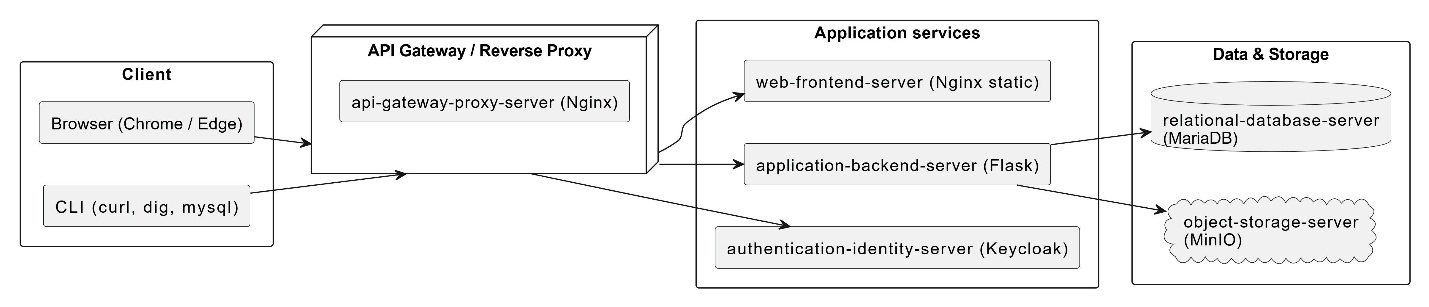
Giám sát (monitoring) là thành phần không thể thiếu của hệ thống cloud. Prometheus là toolkit monitoring mã nguồn mở, lưu trữ dữ liệu theo dạng chuỗi thời gian (time‑series) và thu thập dữ liệu bằng cách định kỳ gọi HTTP endpoint /metrics của các target (exporter hoặc ứng dụng).

Grafana đóng vai trò nền tảng visualization, cung cấp dashboard, biểu đồ, cảnh báo giúp người vận hành quan sát trạng thái hệ thống theo thời gian thực.

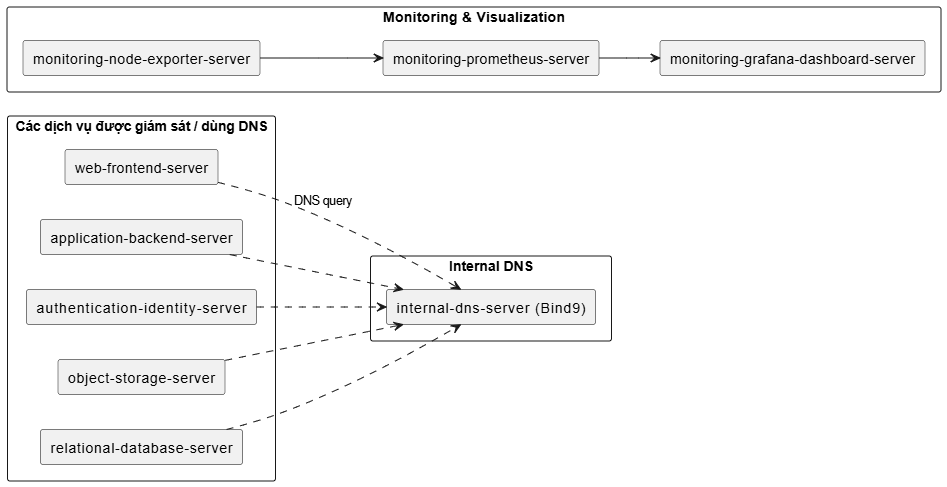
Trong MyMiniCloud, Prometheus và Grafana được sử dụng để đo CPU, RAM, traffic mạng của server thông qua Node Exporter, mô phỏng thực tế các hệ thống giám sát trong doanh nghiệp.

## Kiến trúc logic của MyMiniCloud

Dựa trên yêu cầu của đề và các nguyên lý lý thuyết, MyMiniCloud được thiết kế theo kiến trúc nhiều lớp như sau:



Hình 1: Kiến trúc logic dịch vụ ứng dụng MyMiniCloud



Hình 2: Kiến trúc Internal DNS và hệ thống giám sát MyMiniCloud

### Tầng Client

Tầng client bao gồm:

* Trình duyệt web (Chrome/Edge) truy cập các giao diện web (Home, Blog, Keycloak, MinIO, Grafana).
* Công cụ dòng lệnh (curl, dig, mysql) dùng để kiểm thử API, DNS, database.

Client không truy cập trực tiếp vào từng container nội bộ, mà chủ yếu truy cập qua reverse proxy (cổng 80) hoặc một số port demo đã được expose.

### Tầng Reverse Proxy / API Gateway

Tầng này được hiện thực bởi container api-gateway-proxy-server chạy Nginx. Trong file nginx.conf, server block lắng nghe cổng 80 và định nghĩa các route chính:

* location / → proxy tới web-frontend-server:80 (Web Server).
* location /api/ → proxy tới application-backend-server:8081/.
* location /auth/ → proxy tới authentication-identity-server:8080/ (Keycloak).
* location /storage/ → proxy tới object-storage-server:9000/ (MinIO).
* location /student/ → proxy tới application-backend-server:8081/student (mở rộng).

Nhờ API Gateway, client chỉ cần gọi http://localhost/... thay vì phải nhớ từng port riêng lẻ. Đây là mô hình thường gặp ở các API Gateway trong kiến trúc cloud (ngang với AWS API Gateway hoặc ingress controller trong Kubernetes).

### Tầng dịch vụ ứng dụng

Tầng này gồm ba container chính:

1. Web Frontend – web-frontend-server
   * Sử dụng Nginx để phục vụ static site.
   * Cấu trúc thư mục gồm index.html (trang Home) và thư mục blog/ với index.html và tối thiểu 3 bài blog con, phục vụ yêu cầu mở rộng “Blog cá nhân 3 bài”.
   * Có thể nhúng ảnh avatar từ MinIO qua route /storage/….
2. Application Backend – application-backend-server
   * Sử dụng Flask (Python) lắng nghe cổng 8081 trong container.
   * Cài đặt các endpoint:
     + /hello: trả về JSON { "message": "Hello from App Server!" }.
     + /student: đọc dữ liệu từ students.json hoặc từ bảng students trong studentdb và trả JSON danh sách sinh viên (mở rộng Backend API).
3. Authentication / Identity – authentication-identity-server (Keycloak)
   * Sinh viên triển khai Keycloak với admin admin/admin.
   * Mở rộng: tạo realm mới theo MSSV, tạo user sv01, sv02, client flask-app (public) theo hướng dẫn của đề; mục tiêu là làm quen với mô hình Identity Provider, realm, client, user, token trong OIDC.

### Tầng dữ liệu và lưu trữ

Tầng này bao gồm:

* Relational Database – relational-database-server
  + Dùng image MariaDB, khởi tạo database minicloud và bảng notes chứa dữ liệu “Hello from MariaDB!”.
  + Mở rộng: tạo database studentdb và bảng students(id, student\_id, fullname, dob, major) với ≥ 3 bản ghi để phục vụ API /student.
* Object Storage – object-storage-server (MinIO)
  + Cung cấp dịch vụ lưu trữ kiểu S3 với endpoint 9000 (API) và 9001 (console).
  + Yêu cầu core: tạo bucket demo và upload file index.html.
  + Mở rộng: tạo thêm bucket profile-pics (upload avatar) và documents (upload file PDF báo cáo), cấp quyền public và kiểm thử truy cập qua URL.

### Tầng monitoring và visualization

* Monitoring – monitoring-node-exporter-server + monitoring-prometheus-server
  + Node Exporter expose metrics hệ thống tại :9100.
  + Prometheus được cấu hình trong prometheus.yml với hai scrape\_configs:
    - job\_name: 'node' → target monitoring-node-exporter-server:9100.
    - job\_name: 'web' (mở rộng) → target web-frontend-server:80.
* Visualization – monitoring-grafana-dashboard-server
  + Kết nối đến Prometheus qua datasource http://monitoring-prometheus-server:9090.
  + Tạo dashboard “System Health of 52300218” với 3 panel: CPU Usage (%), Memory Usage (%), Network Receive (bytes/s), sử dụng các query PromQL theo gợi ý trong nháp.

### Tầng DNS nội bộ

Internal DNS – internal-dns-server (Bind9):

* Quản lý zone cloud.local trong file db.cloud.local, với bản ghi:

web-frontend-server IN A 10.10.10.10

app-backend IN A 10.10.10.20

minio IN A 10.10.10.30

keycloak IN A 10.10.10.40

* Mở rộng: thêm các bản ghi app-backend.cloud.local, minio.cloud.local, keycloak.cloud.local và kiểm thử bằng lệnh dig trên host.

DNS nội bộ giúp hiểu cách các service trong cloud sử dụng tên miền nội bộ để gọi nhau thay vì IP cố định, đồng thời kết nối với kiến thức DNS tổng quát.

## Kiến trúc triển khai trên Docker và sơ đồ network

### File docker-compose và dịch vụ

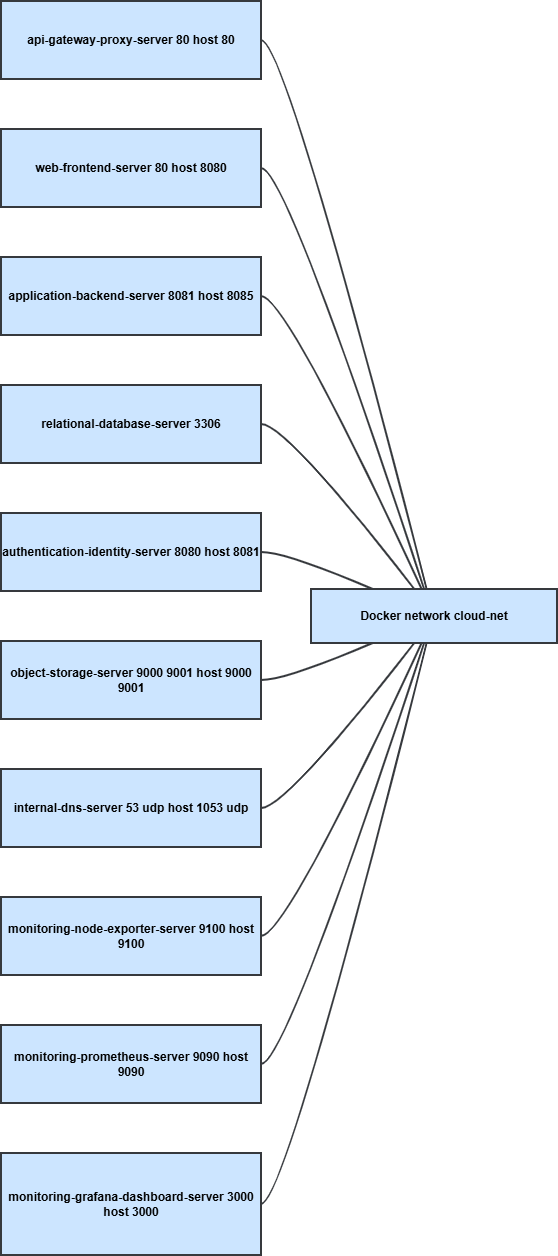
Toàn bộ 9 server (cùng Node Exporter) được khai báo trong file docker-compose.yml trong thư mục gốc dự án hotenSVminiclouddemo. Mỗi service có:

* container\_name riêng (ví dụ web-frontend-server, object-storage-server).
* ports ánh xạ port nội bộ ra host theo bảng “Cổng dùng trong lab” (8080, 8085, 3306, 8081, 9000/9001, 1053/udp, 9090, 9100, 3000, 80).
* volumes để lưu dữ liệu bền (MinIO data, file cấu hình, SQL init).
* restart: unless-stopped theo yêu cầu kỹ thuật chung.

Đoạn compose dành cho một số service tiêu biểu được trích trong đề và nháp (MariaDB, Keycloak, MinIO, Bind9, Prometheus, Grafana, Nginx Proxy).

### Network cloud-net

Network cloud-net được khai báo ở cuối file compose, sau đó mỗi service chỉ định networks: [cloud-net]. Khi khởi động, Docker tạo một bridge network và tự động gán mỗi container một địa chỉ IP; đồng thời Docker DNS cho phép sử dụng container\_name làm hostname.



Hình 3: Sơ đồ network Docker cloud-net

Một node trung tâm "Docker network: cloud-net".

Bên ngoài Docker host, client truy cập vào các port được map theo bảng:

80 (proxy), 8080 (web), 8085 (app), 8081 (Keycloak), 9000/9001 (MinIO), 1053/udp (DNS), 9090 (Prometheus), 9100 (Node Exporter), 3000 (Grafana).

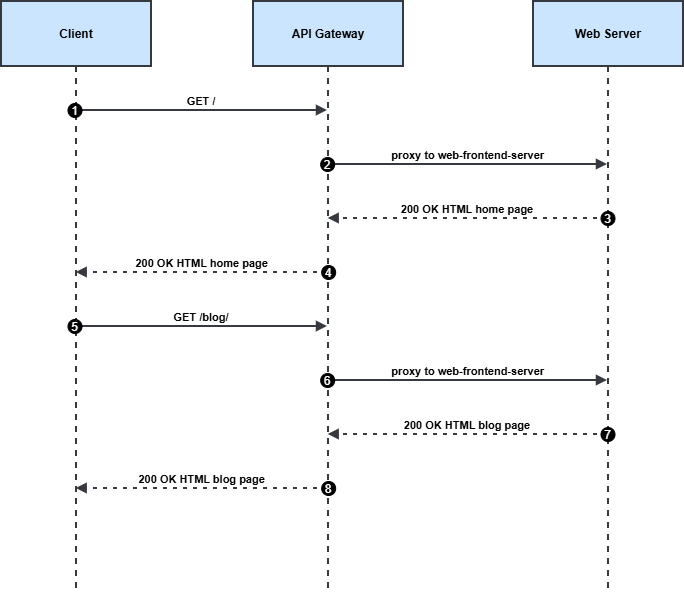
Sơ đồ network minh họa rõ yêu cầu đề: “Tất cả container dùng chung mạng nội bộ cloud-net; sử dụng docker-compose.yml để khởi động toàn hệ thống.”

## Sơ đồ flow request giữa các server

Để đáp ứng yêu cầu “flow request giữa server”, báo cáo lựa chọn hai luồng tiêu biểu: luồng truy cập web và luồng API /student. Các luồng này được mô tả bằng sơ đồ sequence trong

### Flow truy cập Web Frontend

Mục tiêu: kiểm tra hoạt động của Web Server và route / trên API Gateway.



Hình 4. Flow request truy cập trang chủ và blog qua API Gateway.

Các bước:

1. Client gửi GET http://localhost/.
2. Request tới host port 80, được api-gateway-proxy-server nhận.
3. Nginx trong proxy áp dụng rule:

location / {

proxy\_pass <http://web-frontend-server:80>;

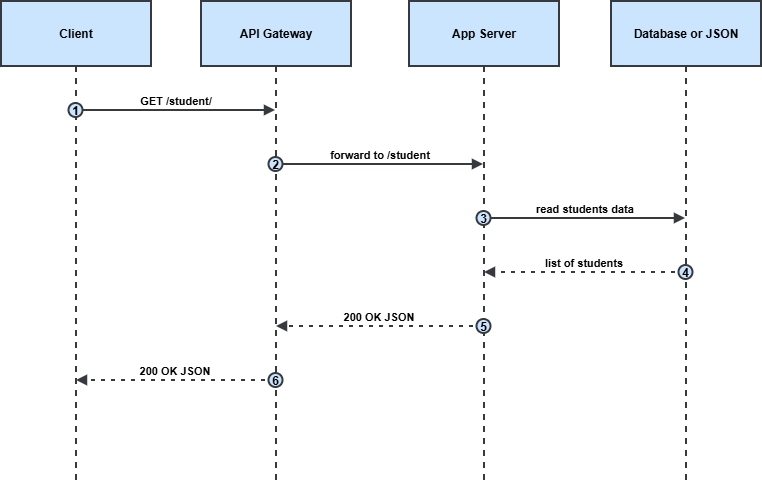
}

1. Proxy forward request tới web-frontend-server trên network cloud-net.
2. Nginx của web server trả về index.html (MyMiniCloud – Home).
3. Proxy chuyển tiếp response về cho client.

Flow này chứng minh reverse proxy hoạt động đúng: client chỉ thấy một endpoint duy nhất, trong khi web server thực sự nằm ở container khác.

### Flow API /student qua API Gateway

Mục tiêu: hiện thực yêu cầu mở rộng Backend API và route /student/ trong bảng “Cloud App Integration Practice”.



Hình 5: Flow request API /student qua API Gateway tới Application Server và nguồn dữ liệu.

Luồng này thể hiện đầy đủ sự phối hợp giữa API Gateway, Application Server và Database/Object storage, đồng thời minh họa khái niệm API endpoint và HTTP JSON response.

### Flow đăng nhập người dùng vào hệ thống qua Keycloak

Mục tiêu của luồng này là mô tả cách người dùng thực hiện đăng nhập (login) vào hệ thống thông qua **Keycloak** – authentication-identity-server, đi qua **API Gateway** và sử dụng **DNS nội bộ** để phân giải tên miền dịch vụ. Qua đó, luồng này minh họa rõ vai trò của tầng Auth và DNS trong kiến trúc MyMiniCloud.

**Các bước thực hiện luồng đăng nhập:**

1. **Người dùng mở trình duyệt**, truy cập địa chỉ http://localhost/auth/ (hoặc một liên kết “Login” trong trang web frontend).
2. **API Gateway (api-gateway-proxy-server) nhận request** tại cổng 80, áp dụng rule:
3. location /auth/ {
4. proxy\_pass http://authentication-identity-server:8080/;
5. }

và forward request tới container Keycloak trên mạng cloud-net.

1. **DNS nội bộ của Docker** (và Bind9 nếu dùng tên như keycloak.cloud.local) chịu trách nhiệm phân giải hostname authentication-identity-server thành địa chỉ IP container tương ứng.
2. **Keycloak trả về trang login**, hiển thị form username/password cho user.
3. Người dùng nhập thông tin tài khoản (ví dụ user sv01 thuộc realm MSSV) và bấm “Sign In”.
4. Trình duyệt gửi lại request POST tới /auth/realms/<realm>/protocol/openid-connect/auth (hoặc endpoint tương ứng), tiếp tục đi qua API Gateway và tới Keycloak.
5. **Keycloak kiểm tra thông tin đăng nhập** (so sánh với dữ liệu trong database nội bộ của Keycloak). Nếu đúng, nó tạo **session và token** (ID token, access token) cho người dùng.
6. Keycloak redirect trình duyệt về ứng dụng (web hoặc app) kèm theo mã ủy quyền (authorization code) hoặc token (tùy flow), hoàn tất quá trình đăng nhập một lần (SSO).
7. Từ thời điểm này, các request từ trình duyệt đến backend có thể kèm token để chứng thực với các dịch vụ phía sau.



Hình 6:Flow đăng nhập người dùng vào hệ thống thông qua API Gateway và Keycloak

### Flow truy cập file tĩnh trên MinIO qua API Gateway

Luồng này mô tả cách người dùng truy cập một file (ảnh avatar, tài liệu PDF,…) lưu trữ ở **MinIO –** object-storage-server, thông qua API Gateway và có thể kết hợp với DNS nội bộ. Đây là ví dụ cụ thể cho việc tích hợp dịch vụ lưu trữ đối tượng vào ứng dụng cloud.

**Các bước của luồng truy cập file:**

1. Người dùng truy cập trang web frontend (ví dụ /profile.html) qua http://localhost/. Trang web này có **thẻ <**img**> hoặc <**a**>** trỏ tới đường dẫn /storage/profile-pics/avatar-sv01.png.
2. Trình duyệt gửi **request GET** /storage/profile-pics/avatar-sv01.png tới host localhost.
3. **API Gateway** nhận request tại cổng 80, áp dụng cấu hình:

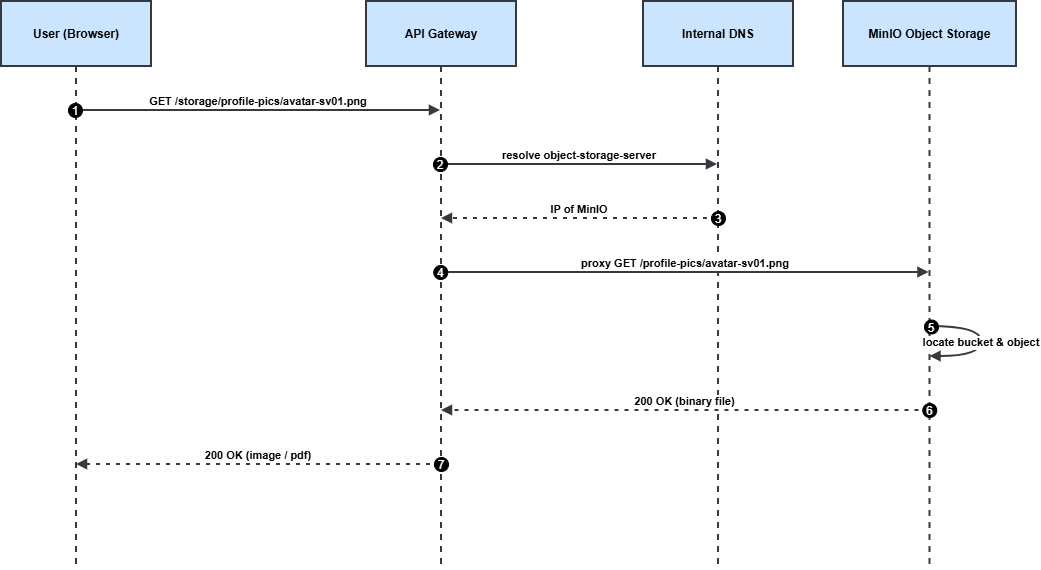
location /storage/ {

proxy\_pass http://object-storage-server:9000/;

}

và forward request tới MinIO qua mạng cloud-net.

1. **Docker DNS** (hoặc Bind9 nếu dùng tên minio.cloud.local) phân giải hostname object-storage-server thành IP container MinIO.
2. **MinIO kiểm tra bucket và object:**
   * Xác định bucket tương ứng (ví dụ profile-pics).
   * Kiểm tra quyền truy cập (public bucket hoặc pre-signed URL).
3. MinIO trả về **HTTP 200 OK** kèm theo nội dung file (ảnh, PDF,…) cho API Gateway.
4. API Gateway chuyển tiếp response về cho trình duyệt.
5. Trình duyệt hiển thị ảnh avatar trên trang hoặc cho phép người dùng tải file về máy.



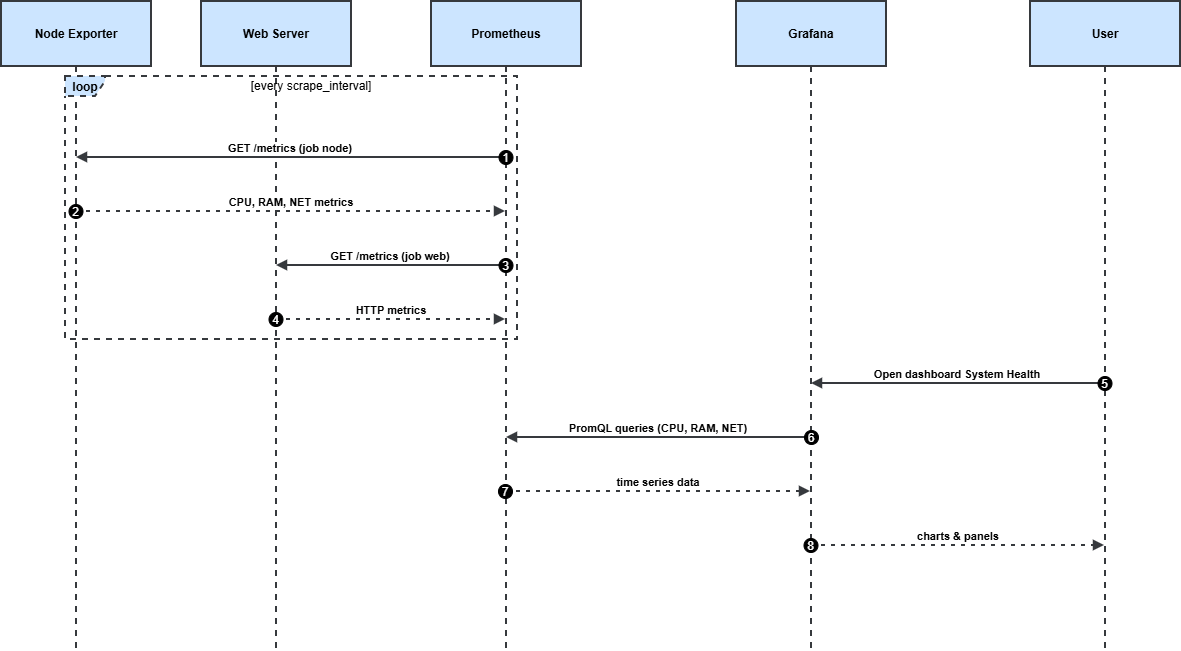
Hình 7: Flow truy cập file tĩnh trên MinIO qua API Gateway.

### Flow giám sát tài nguyên với Node Exporter – Prometheus – Grafana

Luồng này mô tả chi tiết cách **Prometheus thu thập metrics** từ monitoring-node-exporter-server và web/app, sau đó **Grafana truy vấn dữ liệu** từ Prometheus để hiển thị trên dashboard “System Health of MSSV”. Qua đó thể hiện rõ vai trò của tầng giám sát trong MyMiniCloud.

**Các bước trong luồng giám sát:**

1. **Node Exporter** chạy trên container monitoring-node-exporter-server, expose endpoint /metrics tại cổng 9100, cung cấp các số liệu CPU, RAM, network của host/container.
2. Trong file prometheus.yml, cấu hình hai job:
   * job\_name: 'node' với target monitoring-node-exporter-server:9100.
   * job\_name: 'web' với target web-frontend-server:80 (giám sát thêm HTTP metrics).
3. **Định kỳ (ví dụ mỗi 15 giây)**, Prometheus gửi HTTP GET /metrics tới từng target trong các job này và lưu dữ liệu dạng time-series vào database nội bộ.
4. Người dùng mở **Grafana** tại http://localhost:3000/, vào dashboard “System Health of MSSV”.
5. Mỗi panel trên dashboard (CPU, Memory, Network) được cấu hình một truy vấn PromQL tới nguồn dữ liệu Prometheus, ví dụ:
   * CPU: 100 - (avg by (instance)(rate(node\_cpu\_seconds\_total{mode="idle"}[5m])) \* 100)
   * Memory: (1 - node\_memory\_MemAvailable\_bytes / node\_memory\_MemTotal\_bytes) \* 100
   * Network: sum by (instance)(rate(node\_network\_receive\_bytes\_total[5m]))
6. Grafana gửi các truy vấn đó đến Prometheus, nhận về các chuỗi thời gian (time-series) tương ứng.
7. Grafana vẽ các biểu đồ đường (line chart) và hiển thị cho người dùng; khi hệ thống chịu tải, các biểu đồ sẽ thay đổi theo thời gian thực.



Hình 8: Flow thu thập và trực quan hóa metrics giữa Node Exporter, Prometheus và Grafana

## Đánh giá kiến trúc MyMiniCloud theo yêu cầu đề

Từ phân tích trên có thể thấy:

* Kiến trúc MyMiniCloud bám sát mô hình tổng thể trong đề (Reverse Proxy → Web/App/Auth → Database/Storage → Monitoring/Logging → DNS), đồng thời hiện thực đủ 9 loại server bắt buộc.
* Cấu trúc triển khai sử dụng đúng các thực hành hiện đại: container hóa bằng Docker, orchestration bằng Docker Compose, network riêng cloud-net, monitor bằng Prometheus/Grafana, auth bằng Keycloak, object storage bằng MinIO.

# CẤU HÌNH & DOCKER FILE

## Cấu trúc thư mục dự án

Dự án MiniCloud được tổ chức theo từng nhóm dịch vụ, mỗi thư mục tương

ứng với một “server logic” hoặc cụm chức năng:

MiniCloud/

├─ docker-compose.yml # Khai báo toàn bộ service

├─ frontend/

├─ backend/

├─ database/ # MySQL + script init

├─ auth/ # Keycloak

├─ storage-server/ # MinIO (Object Storage)

├─ dns-server/ # Bind9 (DNS nội bộ)

├─ monitoring/ # Prometheus, Node Exporter

├─ logging/ # Grafana + provisioning

└─ load\_balancer/ # Nginx làm API Gateway / Proxy

## File docker-compose và định nghĩa service

Hệ thống sử dụng tệp cấu hình docker-compose.yml để triển khai toàn bộ các container. Một mạng nội bộ tên cloud-net (driver bridge) được tạo kèm dải IP tĩnh, giúp tối ưu cấu hình DNS, routing và hỗ trợ quá trình giám sát, debug trong môi trường mô phỏng cloud.

Các service được định nghĩa tương ứng với mô hình hệ thống, trong đó hai frontend (web-frontend-server1 và web-frontend-server2) được xây dựng từ thư mục frontend/, chạy ứng dụng Next.js trong môi trường production. Các frontend ánh xạ ra những cổng khác nhau trên máy host (3001 và 3002), đồng thời được proxy thực hiện cân bằng tải để xử lý request từ người dùng.

Service backend được build từ thư mục backend/, vận hành API ứng dụng trên cổng 8081. Thành phần này phụ thuộc vào database và chỉ khởi chạy sau khi MySQL đạt trạng thái sẵn sàng thông qua healthcheck. Các biến môi trường của backend cấu hình tham số kết nối Keycloak, thông tin database, pool giới hạn kết nối và cổng truy cập API.

Cơ sở dữ liệu MySQL được triển khai dưới dạng container database, expose cổng 3306 trong mạng nội bộ và map ra cổng 3307 trên host. Việc mount thư mục database/init/ vào container giúp hệ thống tự động chạy script khởi tạo dữ liệu ban đầu. Healthcheck đảm bảo quá trình khởi tạo được xác nhận trước khi backend được phép truy cập.

Thành phần xác thực auth sử dụng Keycloak, ánh xạ ra cổng 8082 và cấu hình tài khoản admin qua biến môi trường. Thư mục lưu trữ auth/data được bind mount để đảm bảo dữ liệu người dùng, realm và cấu hình Keycloak vẫn tồn tại sau khi khởi động lại hệ thống.

Dịch vụ lưu trữ storage-server vận hành MinIO, xuất bản API lưu trữ và giao diện điều khiển ở cổng 9000 và 9001. Service này mount thư mục dữ liệu thật ra host, đồng thời thiết lập tài khoản quản trị và thư mục lưu trữ qua biến môi trường.

DNS nội bộ được xây dựng thông qua dns-server sử dụng Bind9, expose cổng 53 cả TCP và UDP. Thư mục cấu hình zone files và named.conf được mount trực tiếp, cho phép định nghĩa hostname tĩnh cho các service. Điều này giúp routing dựa trên tên thay vì IP, hỗ trợ đúng logic mô phỏng hạ tầng cloud.

Hệ thống giám sát được triển khai gồm Node Exporter và Prometheus. Node Exporter expose metrics của hệ thống qua cổng 9100, trong khi Prometheus chạy ở cổng 9090 và mount file cấu hình monitoring/prometheus.yml. Các thành phần này cho phép quan sát trạng thái hoạt động của cluster và container.

Thành phần logging sử dụng Grafana, ánh xạ ra cổng 3120 để truy cập dashboard. Cấu hình provisioning cùng dữ liệu ứng dụng được map ra volume nhằm bảo toàn dashboard, user setup và thông số hiển thị sau mỗi lần restart.

Thành phần cuối là proxy chịu trách nhiệm load balancing và đóng vai trò API Gateway. Service sử dụng Nginx build từ thư mục load\_balancer, ánh xạ cổng 80 của host, tiếp nhận yêu cầu từ người dùng và phân phối tới frontend, backend, dịch vụ xác thực và lưu trữ. Ngoài ra, proxy cũng mount nội dung tĩnh từ frontend để phục vụ truy cập blog nội bộ.

Bên cạnh định nghĩa container, file docker-compose còn khai báo nhiều thành phần thiết yếu gồm: biến môi trường cho các dịch vụ database, MinIO, Keycloak và backend; cơ chế lưu trữ bền vững dữ liệu bằng volume/bind mount; mount cấu hình cho DNS, Prometheus và Nginx; cùng cấu hình IPAM cho phép phân bổ địa chỉ IP tĩnh trên mạng cloud-net.

Từ đó, hệ thống docker-compose mô phỏng một kiến trúc cloud tích hợp nhiều lớp dịch vụ, tách biệt logic xử lý, lưu trữ, xác thực, giám sát và điều phối, đảm bảo tính phân tán, quan sát được và sẵn sàng mở rộng.

## Dockerfile của từng nhóm server

Trong MiniCloud, hệ thống kết hợp hai cách triển khai dịch vụ: một số service sử dụng image chính thức từ Docker Hub, trong khi những thành phần cốt lõi được build từ Dockerfile tự viết nhằm phù hợp logic nghiệp vụ và mô phỏng môi trường cloud thực tế. Các Dockerfile được xây dựng cho frontend, backend, storage server, DNS server và load balancer.

Thành phần Web Frontend trong thư mục frontend/ được triển khai dựa trên base image Node (Alpine). Dockerfile cài đặt dependency của dự án, thực hiện build ứng dụng Next.js và khởi chạy server production tại cổng nội bộ 3000. Cơ chế này đảm bảo frontend chạy nhẹ, tối ưu tài nguyên và phù hợp với mô hình triển khai container.

Thành phần Backend ứng dụng, đặt tại thư mục backend/, sử dụng base image Node. Dockerfile thực hiện copy mã nguồn Express, cài đặt dependency của API và expose cổng 8081 trước khi chạy file chính index.js. Nhờ đó, backend hoạt động độc lập như một microservice, phục vụ các API truy vấn dữ liệu và xác thực.

Dịch vụ lưu trữ đối tượng trong thư mục storage-server/ xây dựng từ base image Alpine. Dockerfile cài đặt nhị phân MinIO và cấu hình entrypoint khởi chạy minio server /data --console-address :9001, từ đó hệ thống cung cấp endpoint API và bảng điều khiển web của MinIO phục vụ lưu trữ phân tán.

Server DNS nội bộ trong thư mục dns-server/ dựa trên base image Debian. Dockerfile cài đặt Bind9, copy file cấu hình named.conf cùng toàn bộ zone records, đồng thời cấu hình Bind chạy ở chế độ foreground để đảm bảo container giữ vòng đời và ghi log trực tiếp. Thành phần này đóng vai trò phân giải tên miền nội bộ giữa các service theo dải IP tĩnh cloud-net.

Thành phần load balancer/API Gateway được xây dựng trong thư mục load\_balancer/, sử dụng base image nginx:stable. Dockerfile cài đặt thêm các công cụ debug cần thiết như ping, dig và curl để hỗ trợ kiểm tra mạng trong môi trường phát triển, copy file nginx.conf làm cấu hình mặc định và expose cổng 80 phục vụ điều phối request từ người dùng vào hệ thống.

Các dịch vụ còn lại như MySQL, Keycloak, Prometheus, Node Exporter và Grafana không yêu cầu Dockerfile riêng vì chúng sử dụng image chính thức từ Docker Hub. Những thành phần này chủ yếu được cấu hình thông qua docker-compose.yml cùng cơ chế mount file cấu hình, mapping volume hoặc thiết lập biến môi trường để đáp ứng logic vận hành của MiniCloud.

Cách tiếp cận này thể hiện sự phân tách giữa dịch vụ tự xây dựng (nơi hệ thống cần kiểm soát chi tiết) và dịch vụ hạ tầng (nơi image chuẩn đã đáp ứng đầy đủ), đồng thời phản ánh mô hình triển khai microservices trong môi trường cloud hiện đại.

## Các file cấu hình chính

Trong kiến trúc MiniCloud, các thành phần hạ tầng được cấu hình thông qua các file định nghĩa riêng, đóng vai trò điều phối request, phân giải tên miền, thu thập metric và hiển thị dashboard giám sát.

Thành phần Proxy/API Gateway sử dụng Nginx, với file cấu hình đặt tại load\_balancer/nginx.conf. Cấu hình này định nghĩa upstream pool cho hai frontend, cho phép cân bằng tải giữa web-frontend-server1 và web-frontend-server2. Proxy phân tuyến request theo đường dẫn: trang chủ và giao diện người dùng được chuyển vào cụm frontend; các request /api/ được forward tới backend chạy trên cổng 8081; luồng /auth/ được điều hướng đến Keycloak tại địa chỉ /auth nội bộ; nội dung blog dạng HTML tĩnh được phục vụ trực tiếp từ thư mục mount (frontend/public/blog) tại blog-static, giúp proxy có thể trả file tĩnh mà không cần backend xử lý.

Thành phần DNS nội bộ sử dụng Bind9 thông qua hai file cấu hình dns-server/named.conf và zone records tại dns-server/zones/. Zone được khai báo dưới tên miền cloud.local, ánh xạ hostname các service như frontend-1.cloud.local, frontend-2.cloud.local, backend.cloud.local, auth.cloud.local, database.cloud.local và storage.cloud.local đến IP tĩnh tương ứng trong mạng cloud-net. Cách tiếp cận này cho phép tầng ứng dụng truy cập service theo DNS thay vì phụ thuộc vào IP cố định, phù hợp mô hình hạ tầng ảo hóa.

Phần monitoring được triển khai với Prometheus, đọc cấu hình từ monitoring/prometheus.yml. Hệ thống khai báo nhiều job scrape: Node Exporter để thu thập metric tài nguyên máy chủ tại monitoring-node-exporter-server:9100; bản thân Prometheus giám sát chính nó qua endpoint nội bộ; cụm frontend được thiết lập endpoint thu thập metric tại /api/metrics trên cả web-frontend-server1 và web-frontend-server2. Nhờ đó, hệ thống có cái nhìn tổng quan về tải ứng dụng, phản hồi frontend và tài nguyên host.

Lớp logging và hiển thị dashboard sử dụng Grafana. Cấu hình provisioning nằm tại logging/provisioning/, trong đó datasource được cấu hình trỏ đến Prometheus thông qua địa chỉ http://monitoring-prometheus-server:9090. Đồng thời, dashboard được tự động load từ thư mục logging/grafana-data, giúp hệ thống khởi động lại vẫn giữ các bảng giám sát. Dashboard mẫu hiển thị các panel quan sát sức khỏe hệ thống như CPU, RAM, độ trễ và tốc độ mạng, phản ánh trạng thái hoạt động thực tế của MiniCloud.

Thông qua sự phối hợp giữa Proxy, DNS, Monitoring và Logging, hệ thống mô phỏng mô hình cloud thực tế, trong đó request được điều hướng thông minh, tên miền được phân giải nội bộ, trạng thái container được theo dõi, và dữ liệu vận hành được trực quan hóa trên dashboard phục vụ phân tích và đánh giá hiệu năng.

# DEMO & KIỂM THỬ

## Môi trường chạy và cách khởi động hệ thống

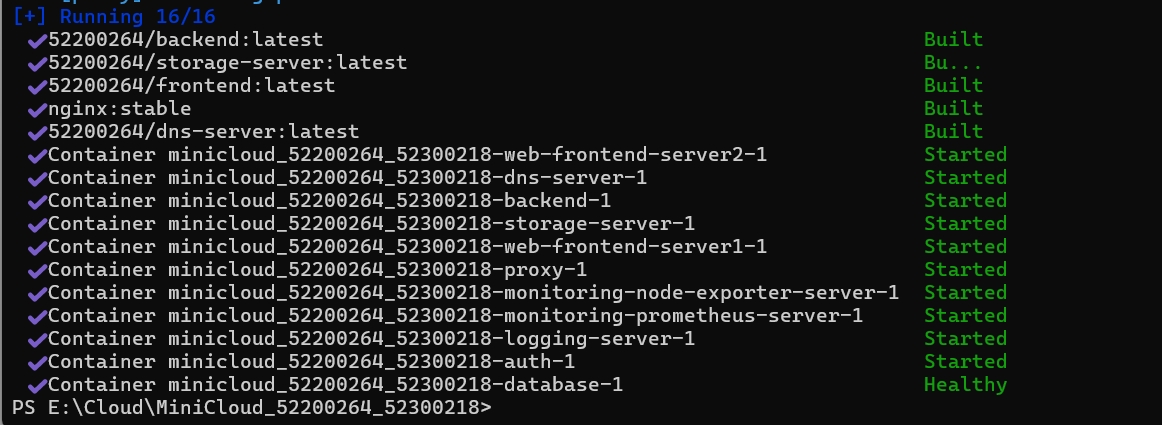
Môi trường demo (máy local):

• Hệ điều hành: Windows 11

• Docker Engine: v29.0.1; Docker-compose: v2.40.3-desktop.1

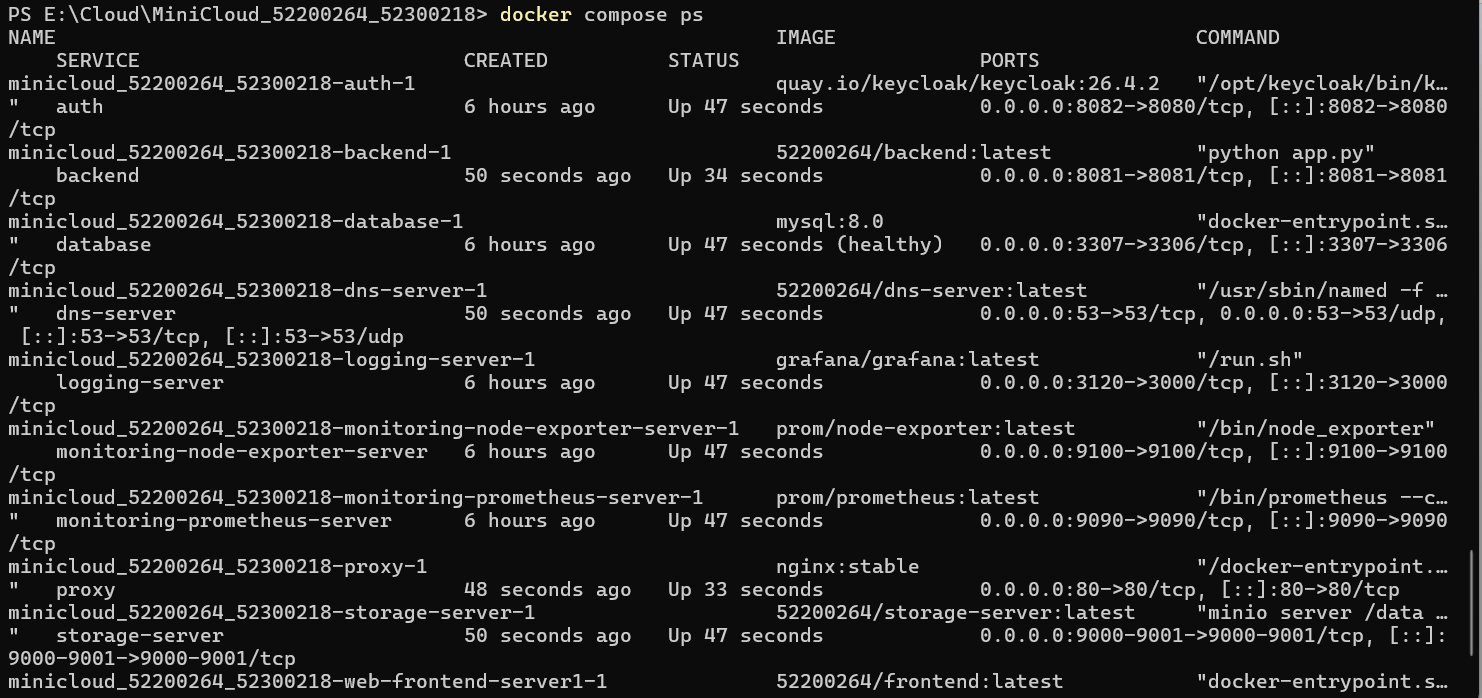
• RAM: 16G, Disk free > 50G

Build và chạy toàn bộ container: “docker compose up -d --build”



Hình 4.1: Kết quả build thành công project

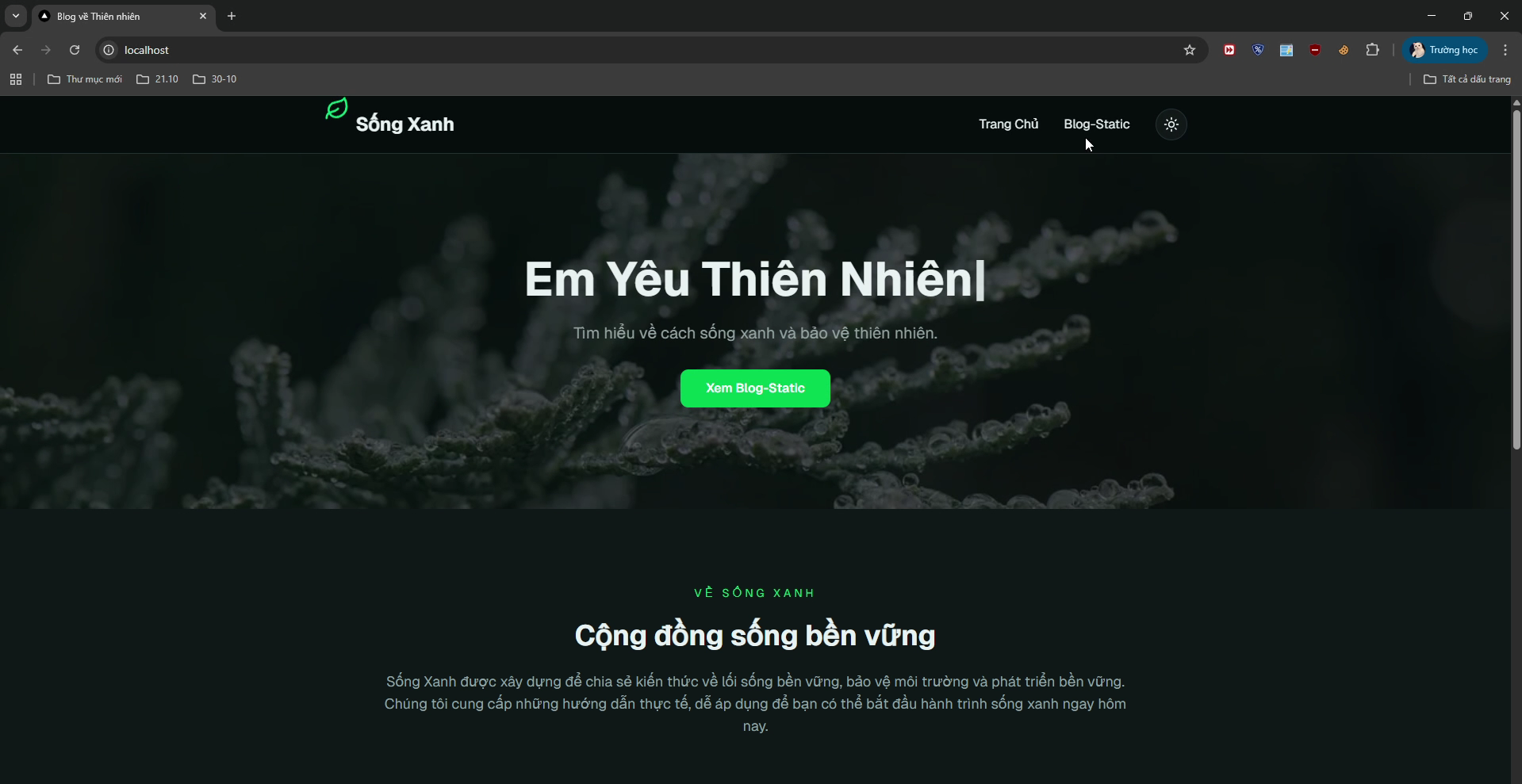
Kiểm tra trạng thái: “docker compose ps”



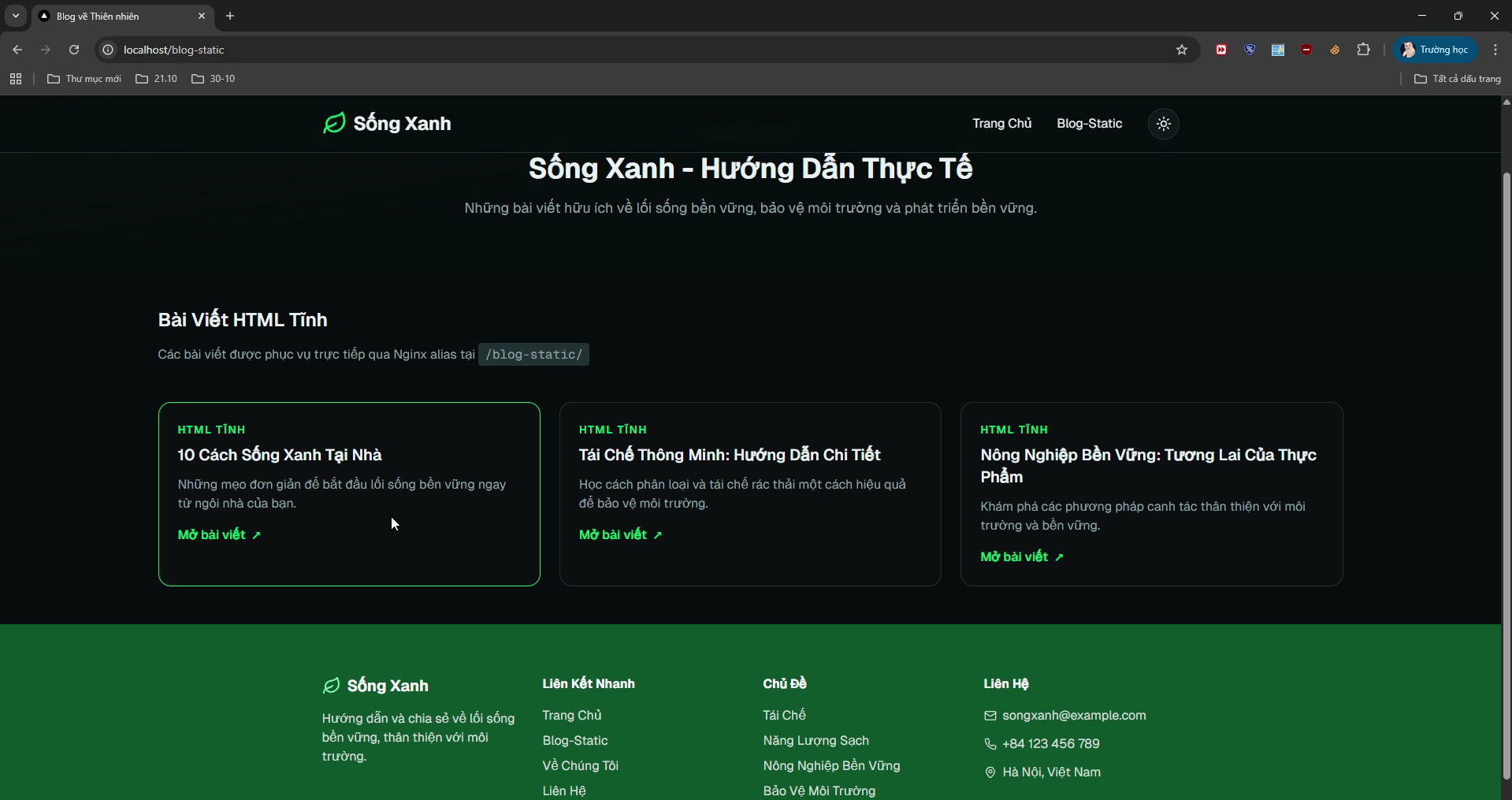
Hình 4.2: Xem trạng thái của các service

## Kiểm thử chức năng từng server

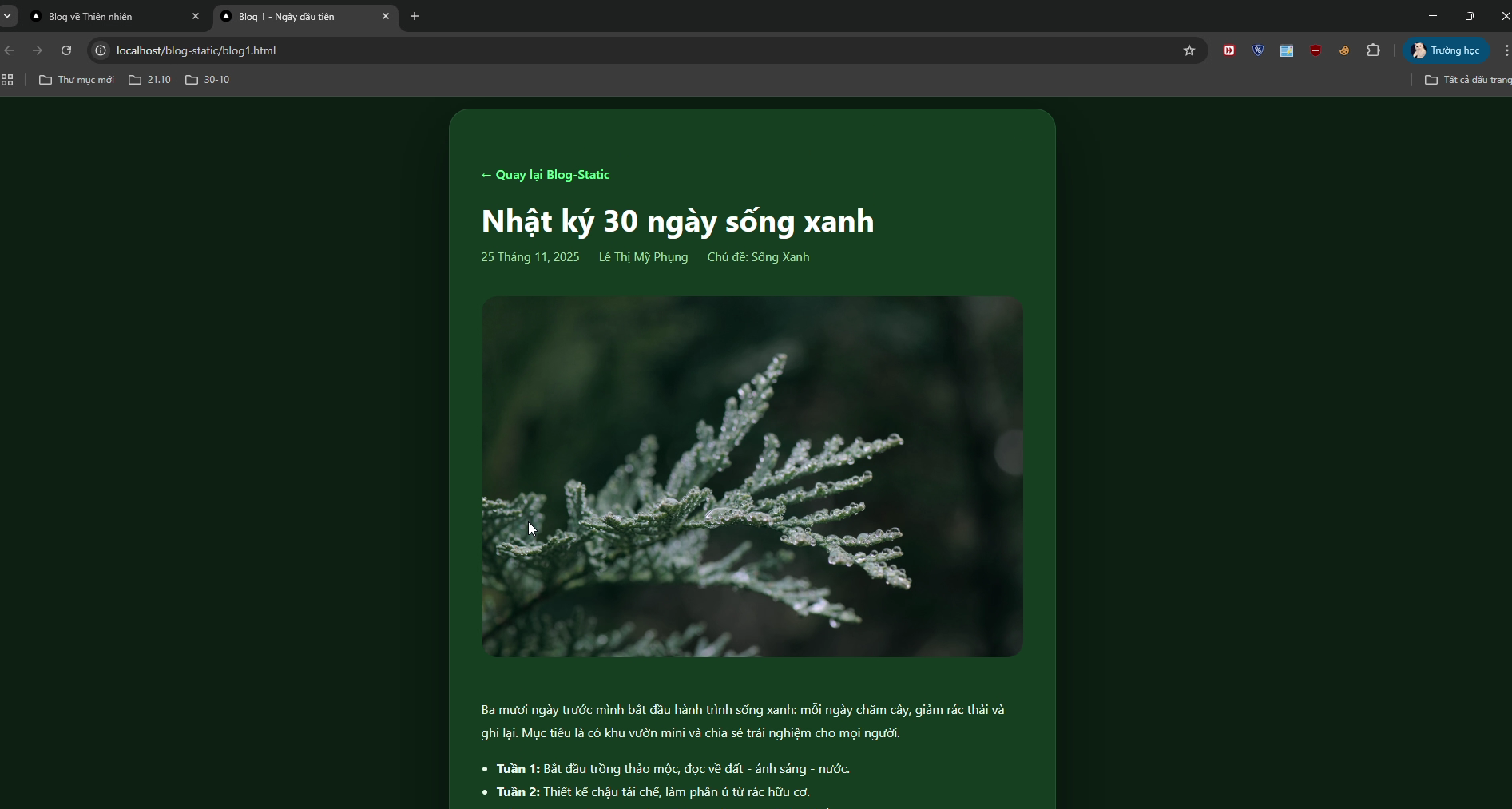
### Web Server (Web Frontend)



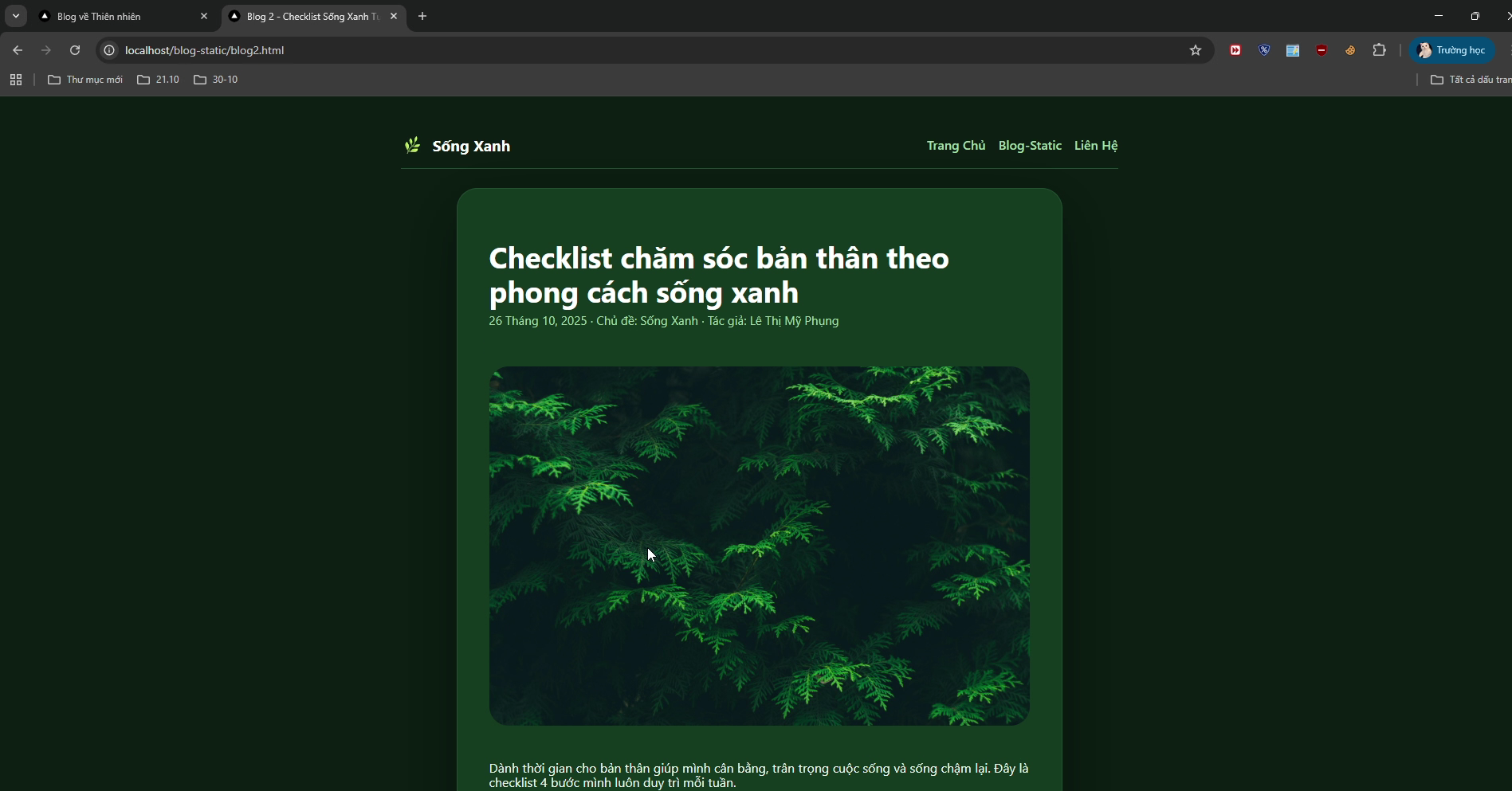
Hình 4.3: Truy cập http://localhost/ → giao diện Next.js hiển thị bình thường



Hình 4.4: http://localhost/blog – danh sách các blog, có cả tĩnh & động

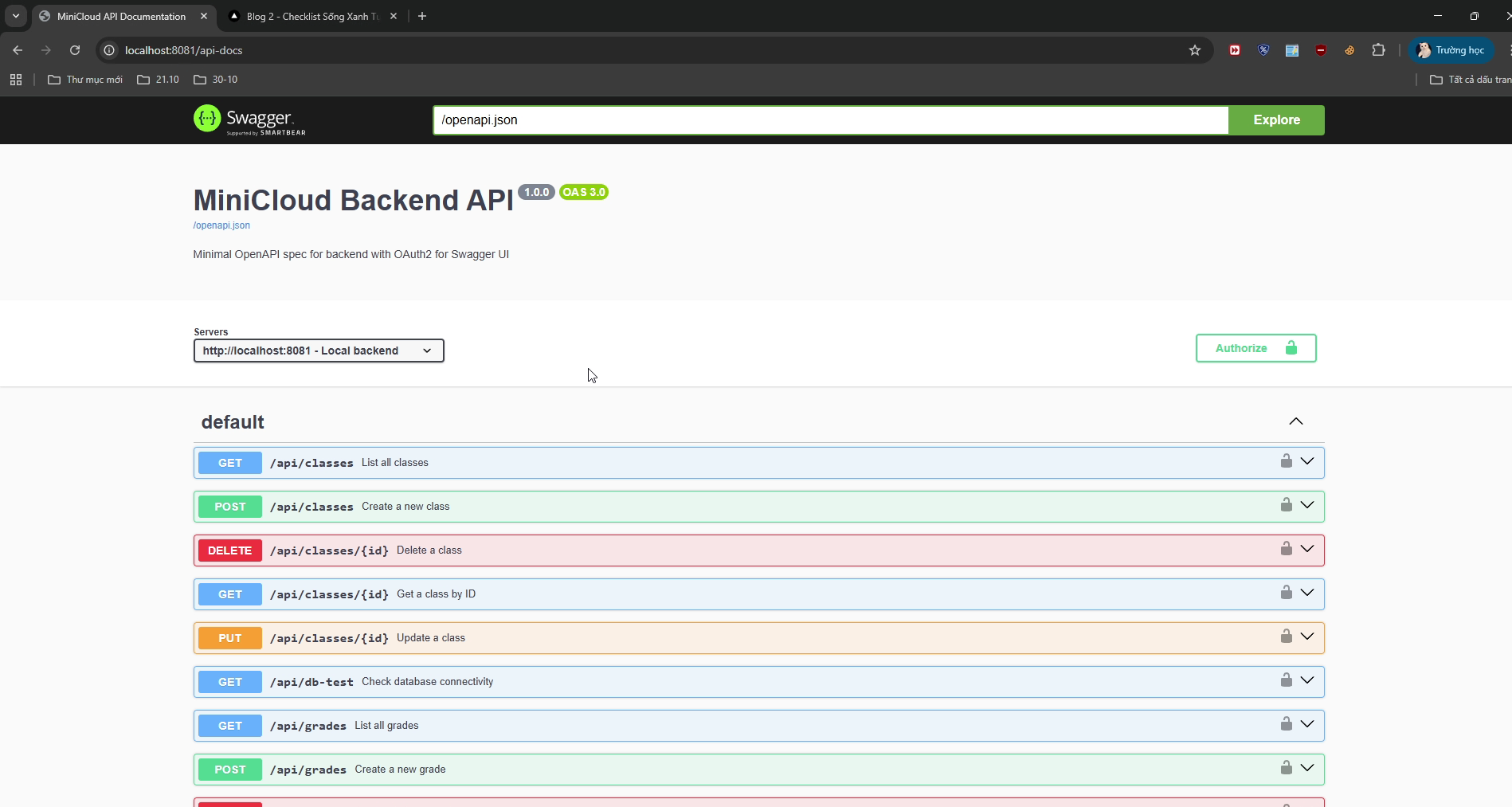


Hình 4.5: Truy cập http://localhost/blog-static/blog1.html → hiển thị một bài blog

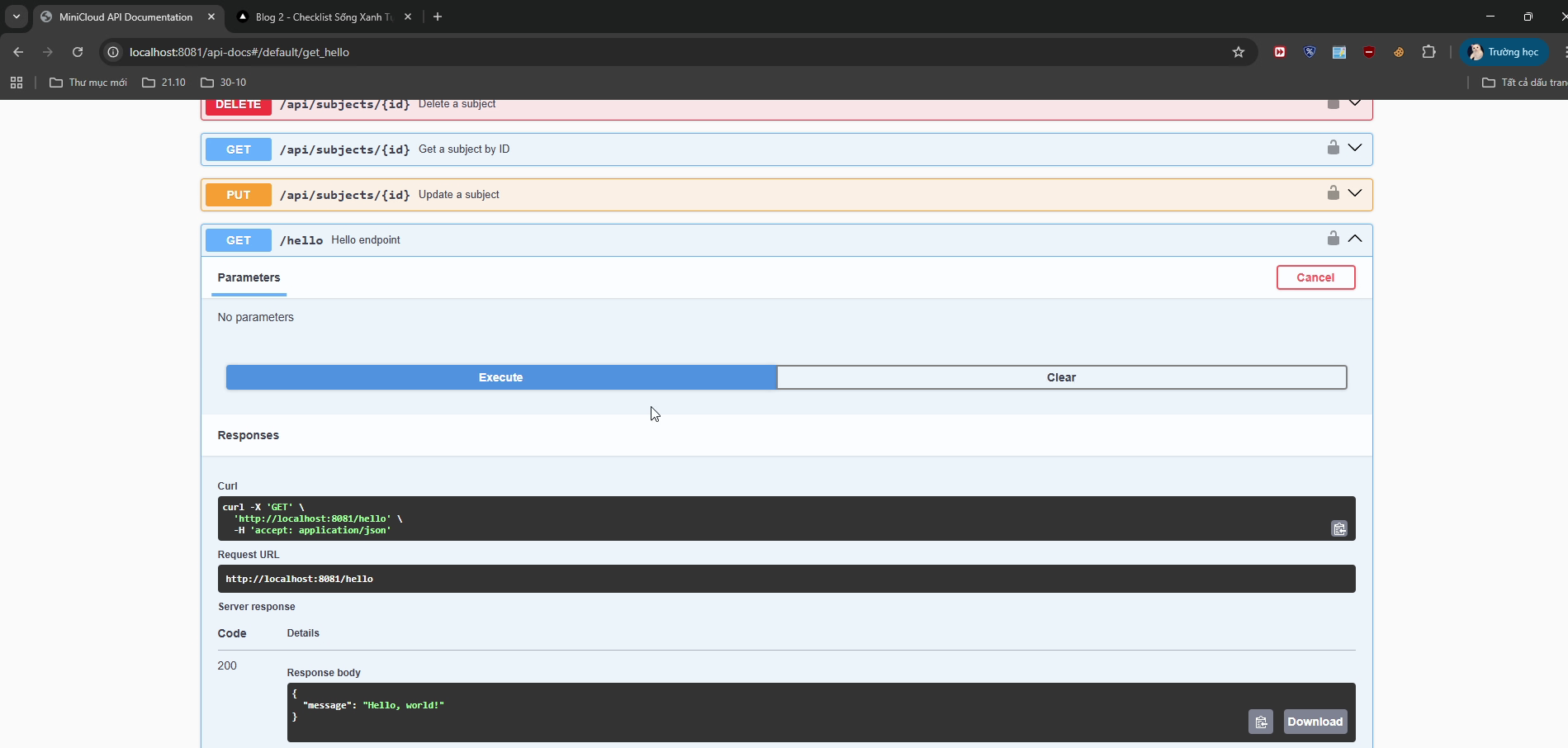


Hình 4.6: Truy cập http://localhost/blog-static/blog2.html

### App Server (Application Backend)



Hình 4.7: Truy cập <http://localhost:8081/>

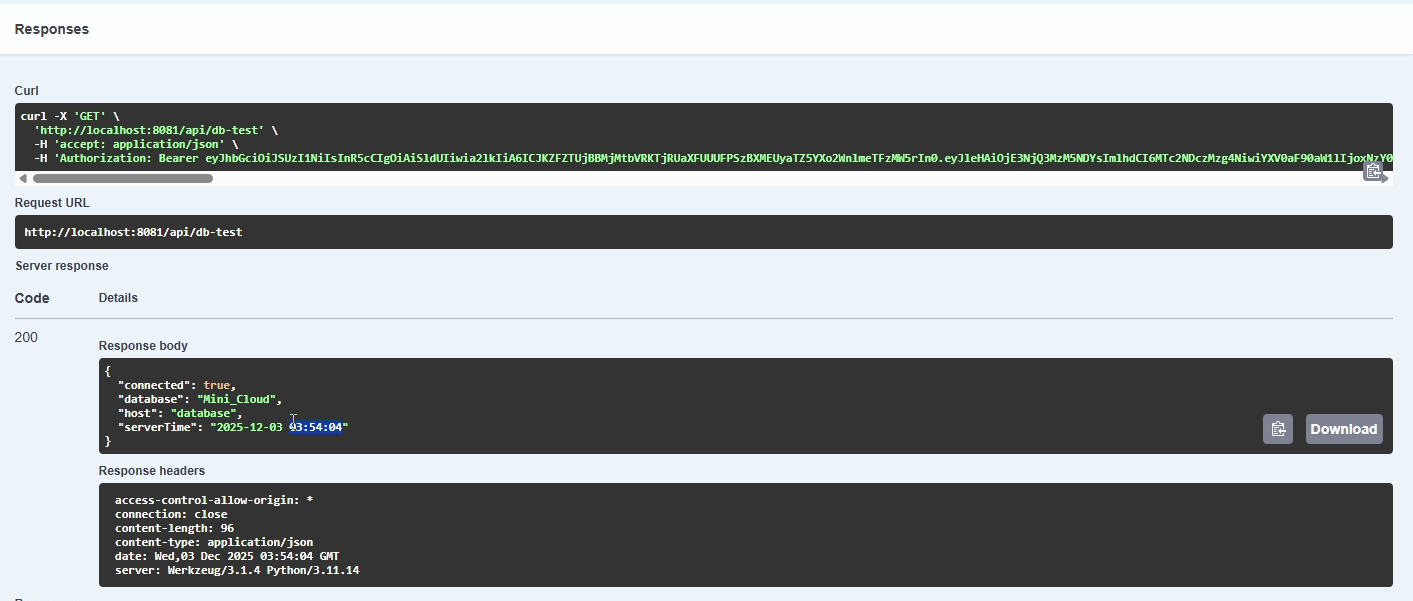


Hình 4.8: Kết quả Gọi API public



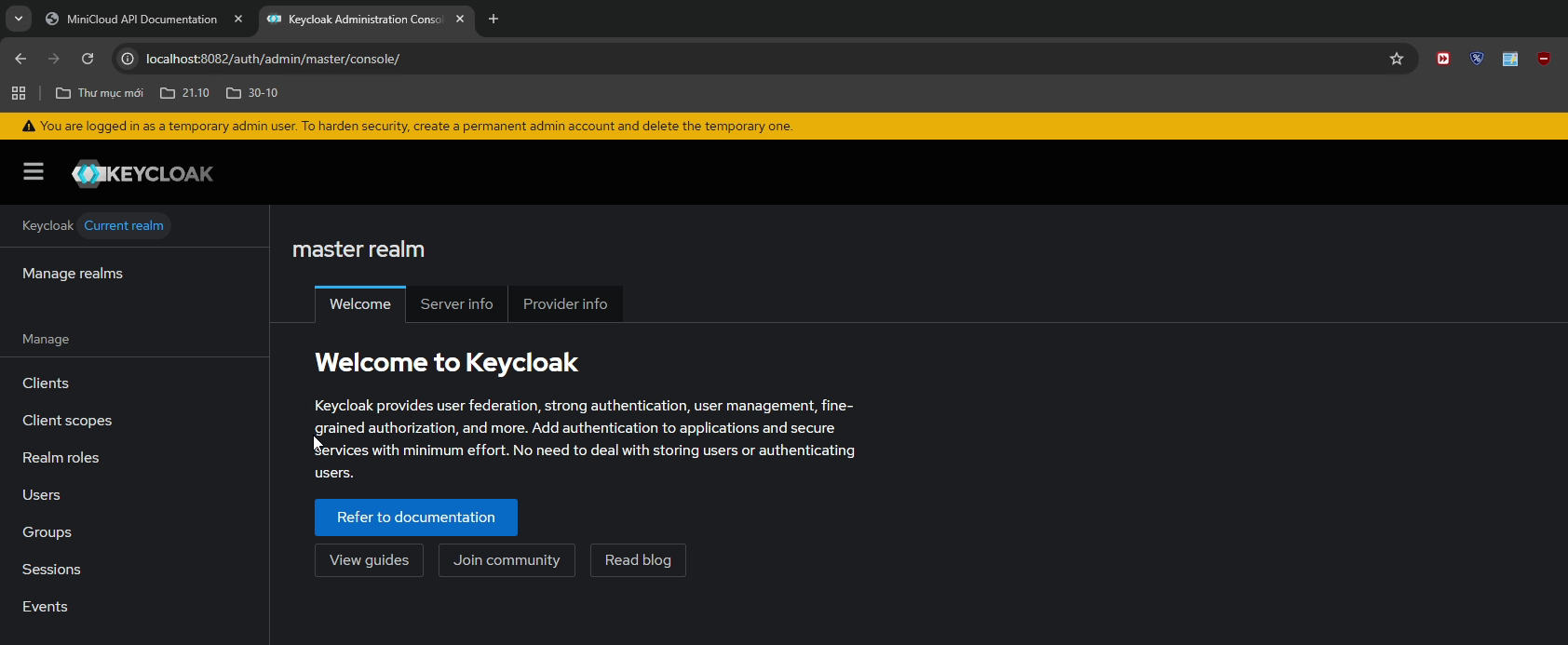
Hình 4.9: Kết quả gọi API /student qua proxy20

### Database Server (MySQL)

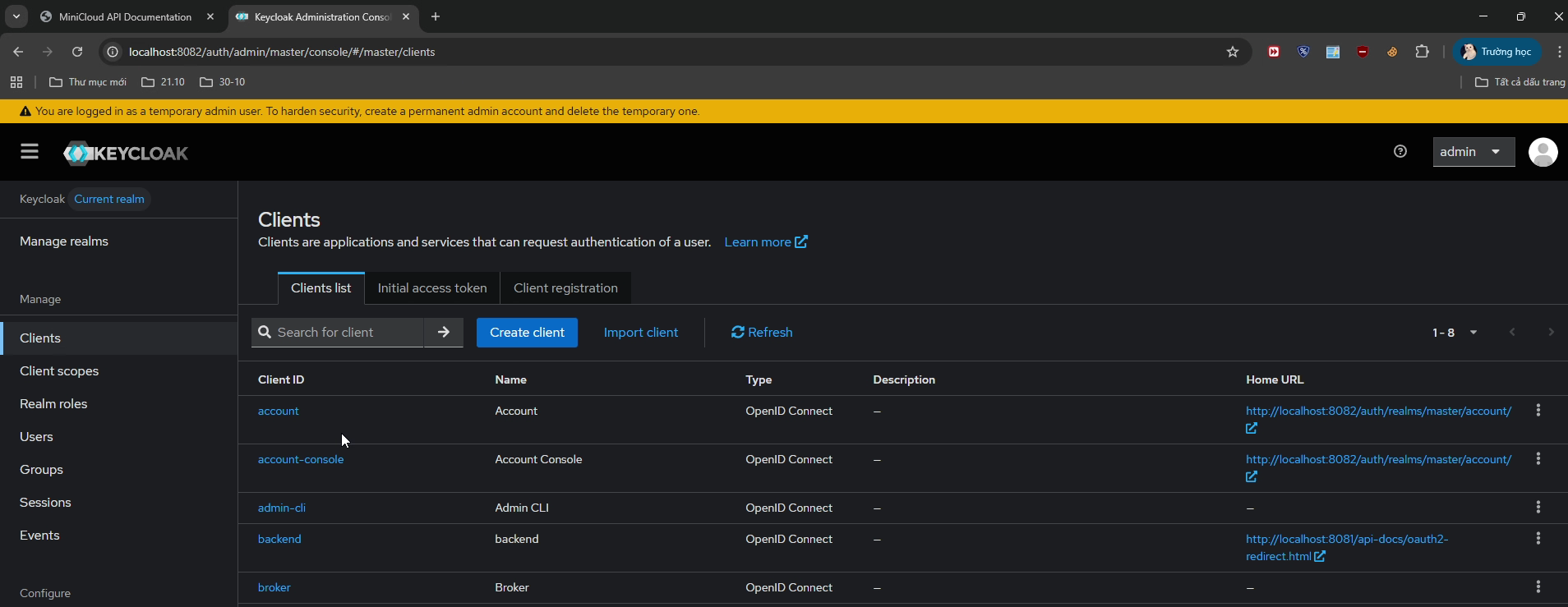


Hình 4.10: Kết quả test database

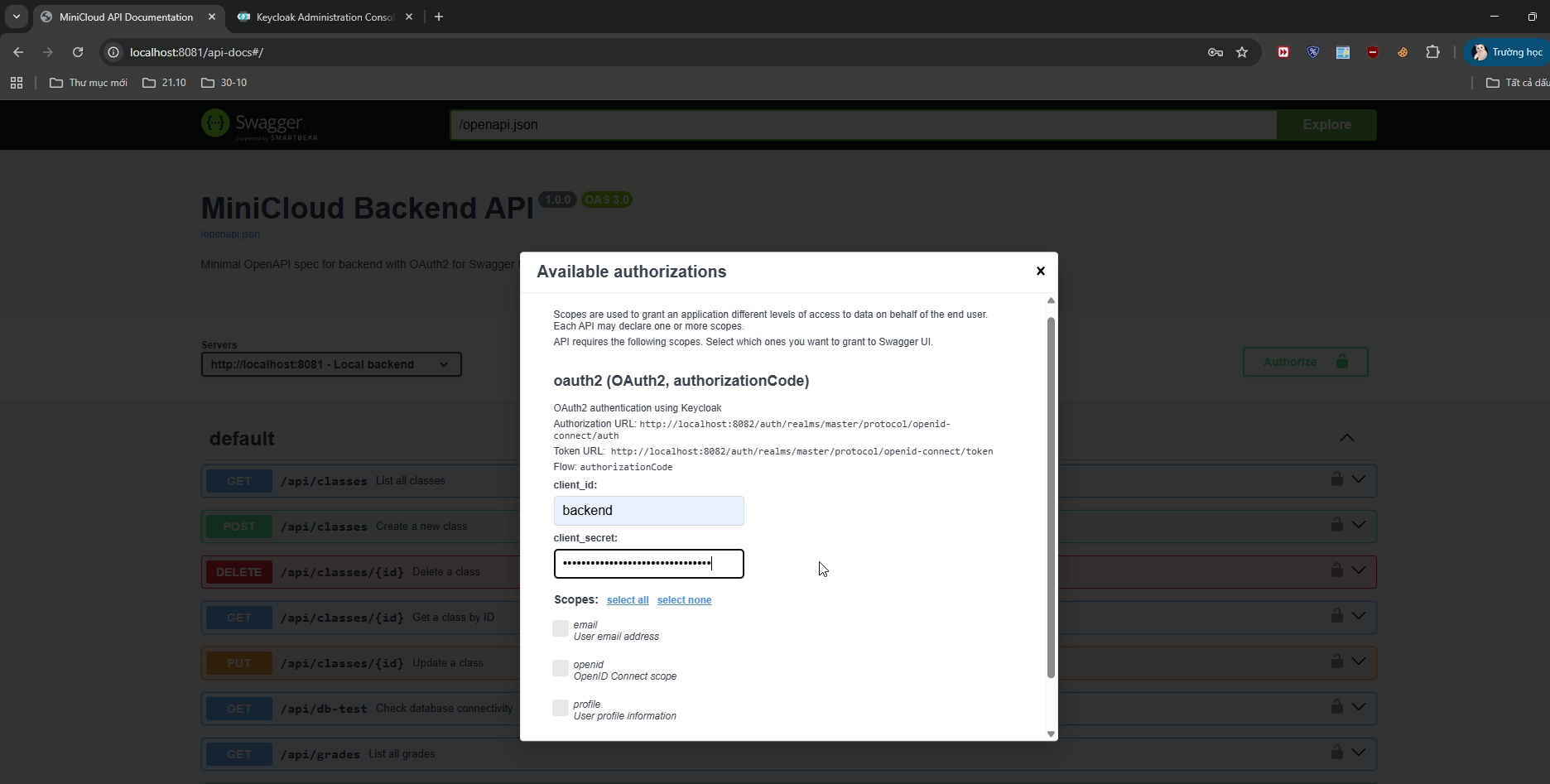
### Auth Server (Keycloak)



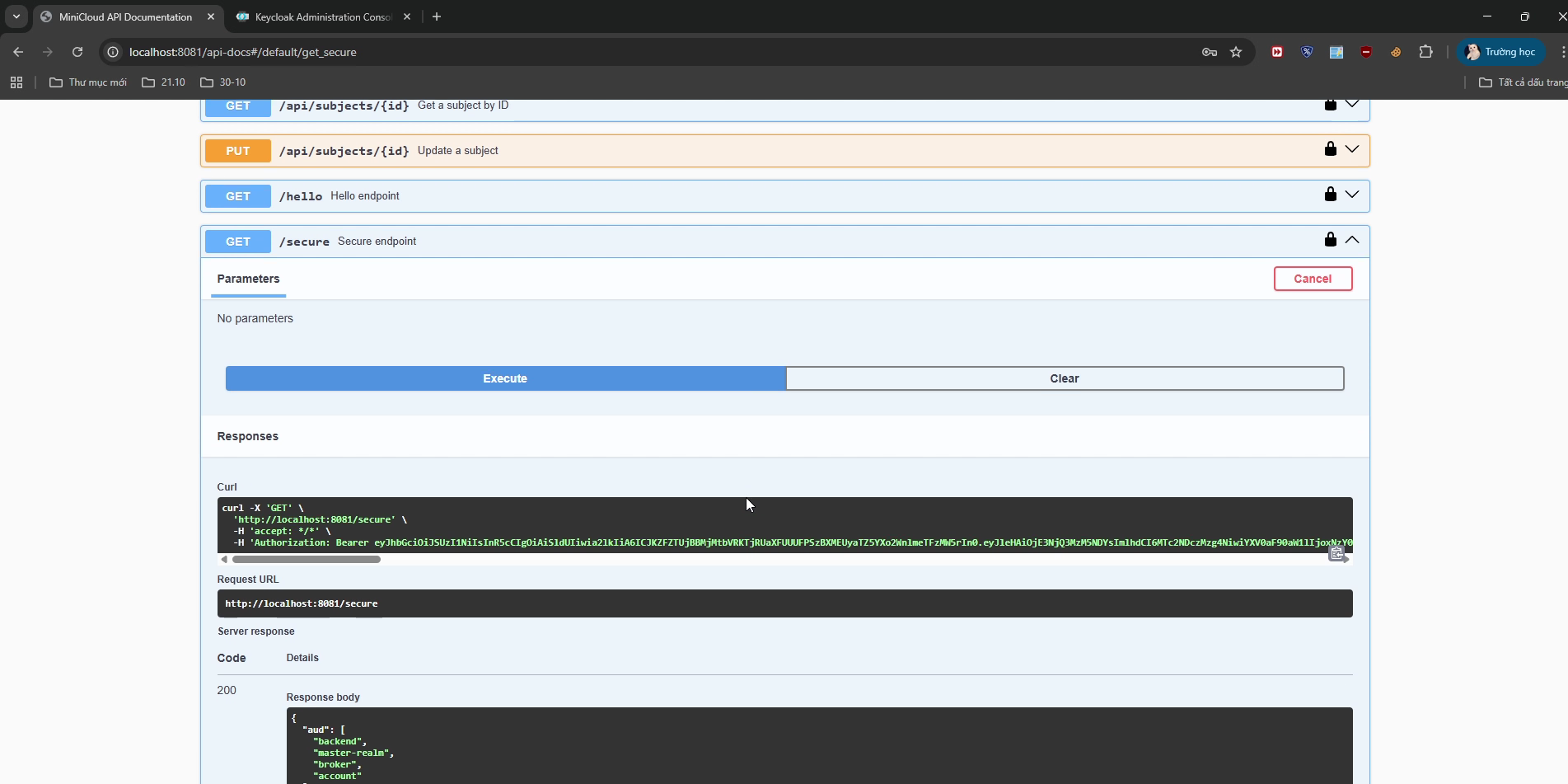
Hình 4.11: Giao diện quản trị hiển thị trang Dashboard Keycloak (admin)



Hình 4.12: Các user đã tạo trong Keycloak

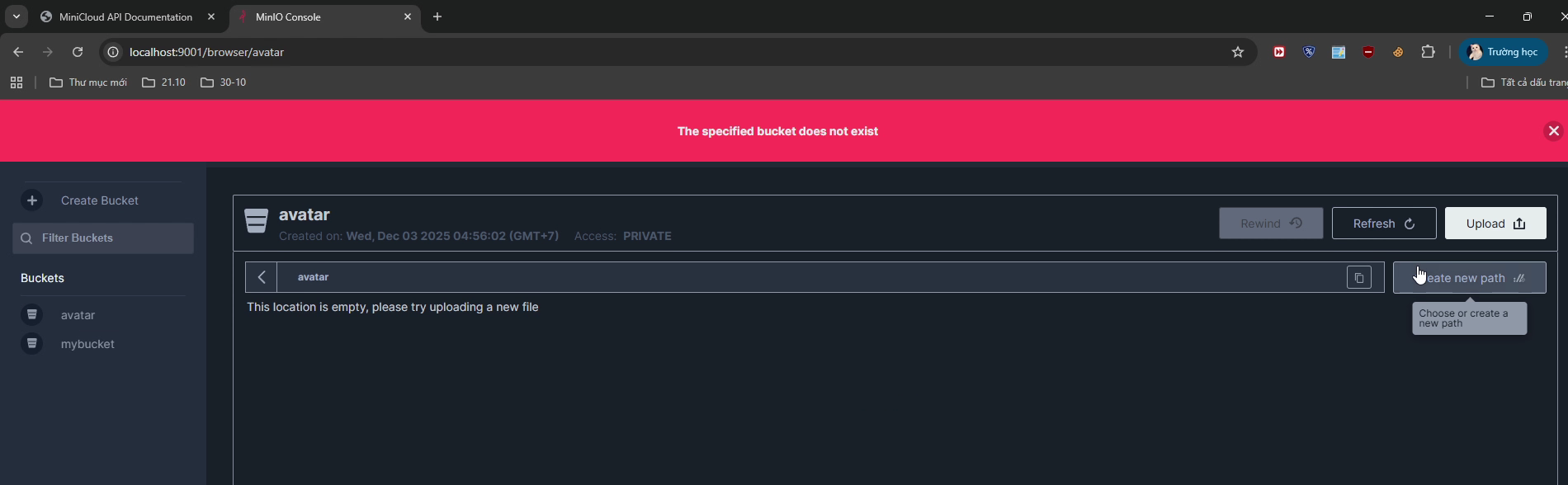


Hình 4.13: Lấy secret key từ client

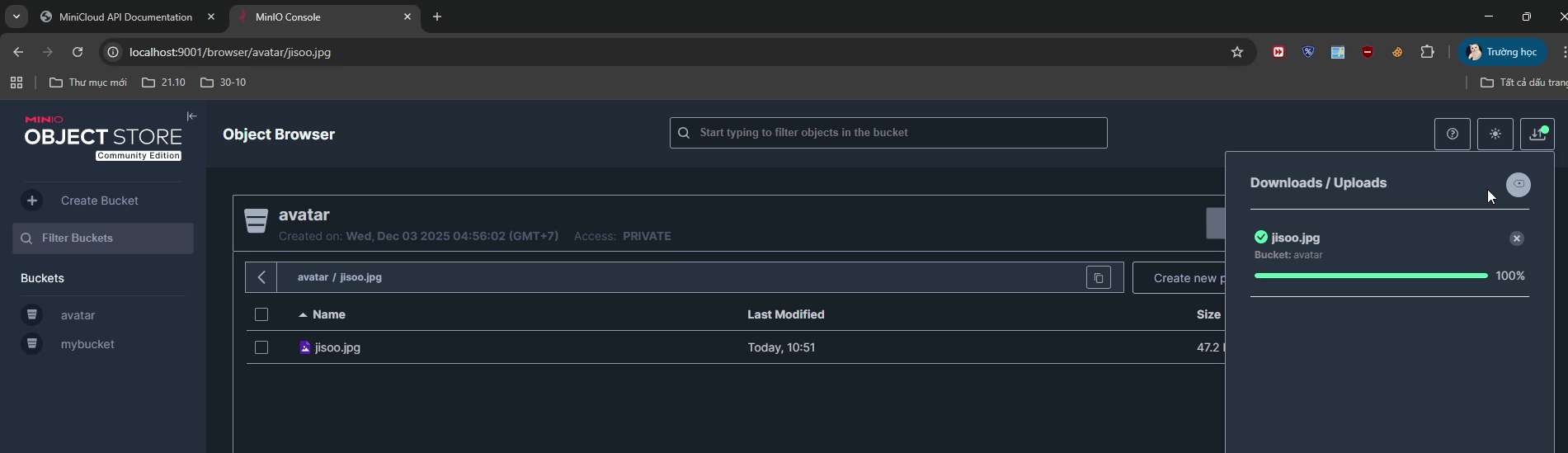


Hình 4.14: Xác thực backend từ keycloak (/secure)

### Storage Server (MinIO)

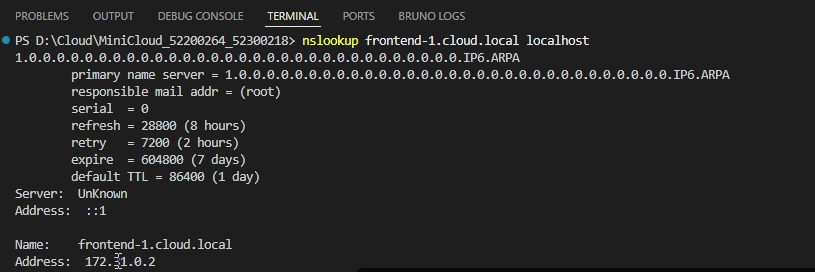


Hình 4.15: Tạo bucket avatar, upload file ảnh, lấy URL public object



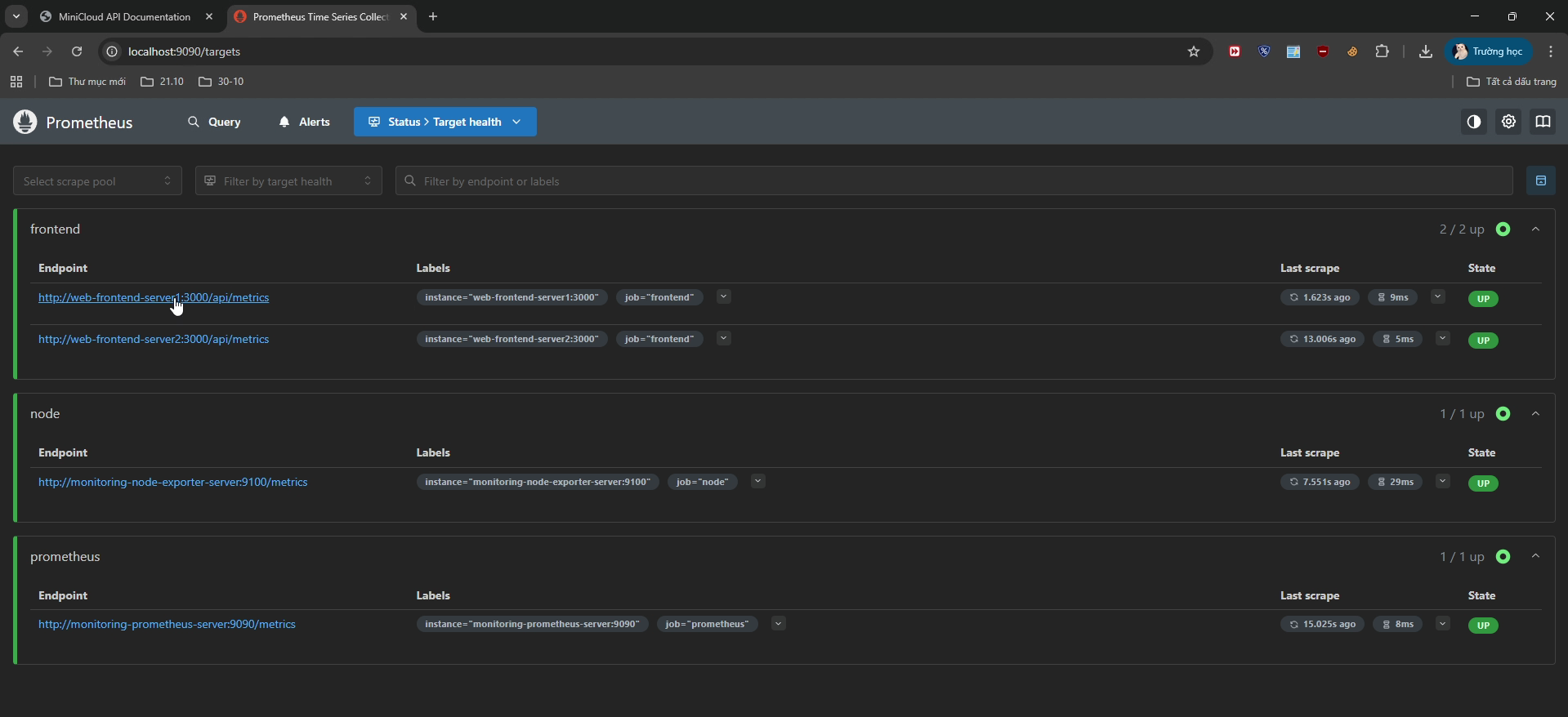
Hình 4.16: Upload thành công

### DNS Server (Bind9)

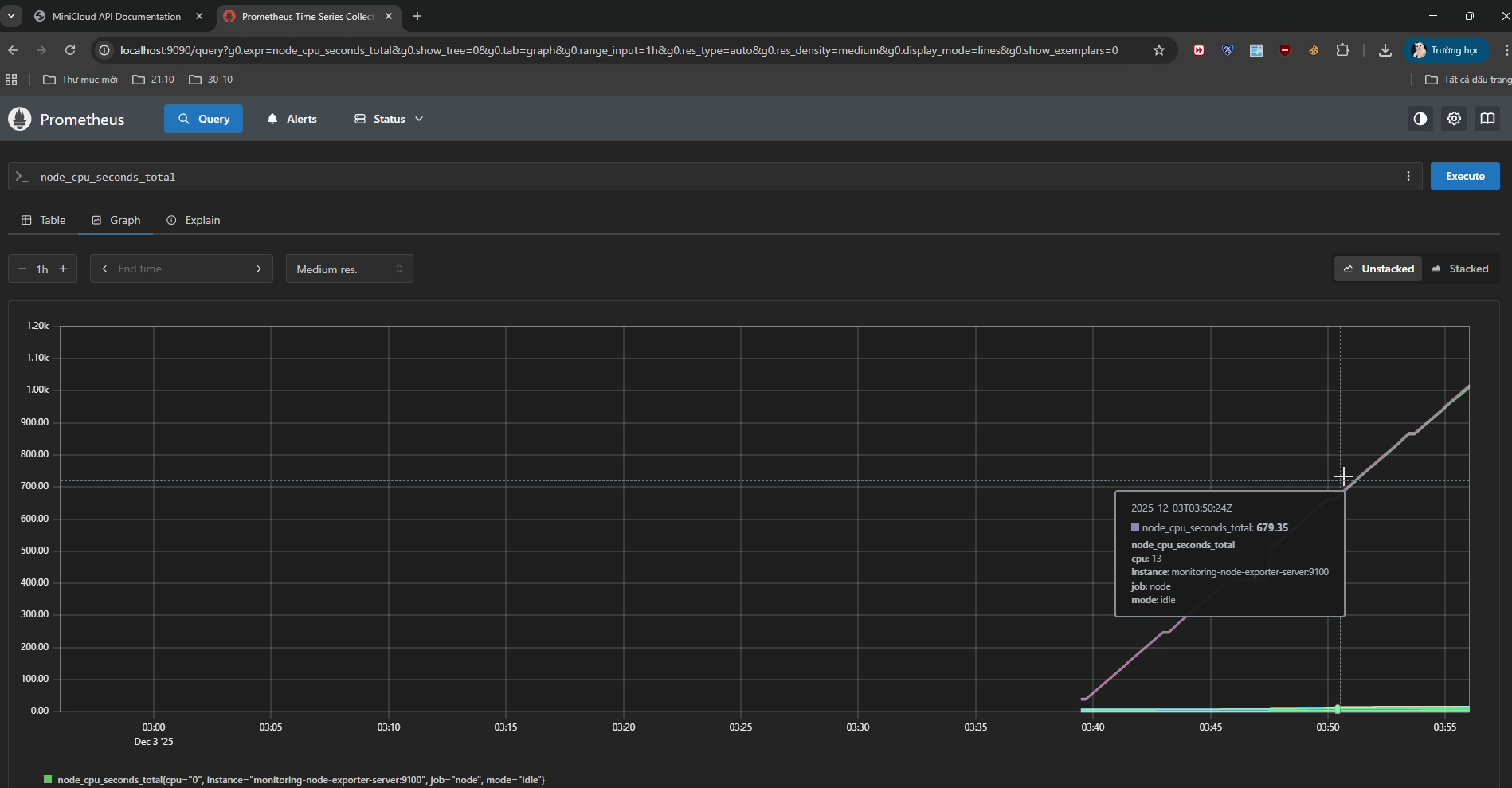


Hình 4.18: Trả về IP theo file cấu hình DNS

### Monitoring (Prometheus)

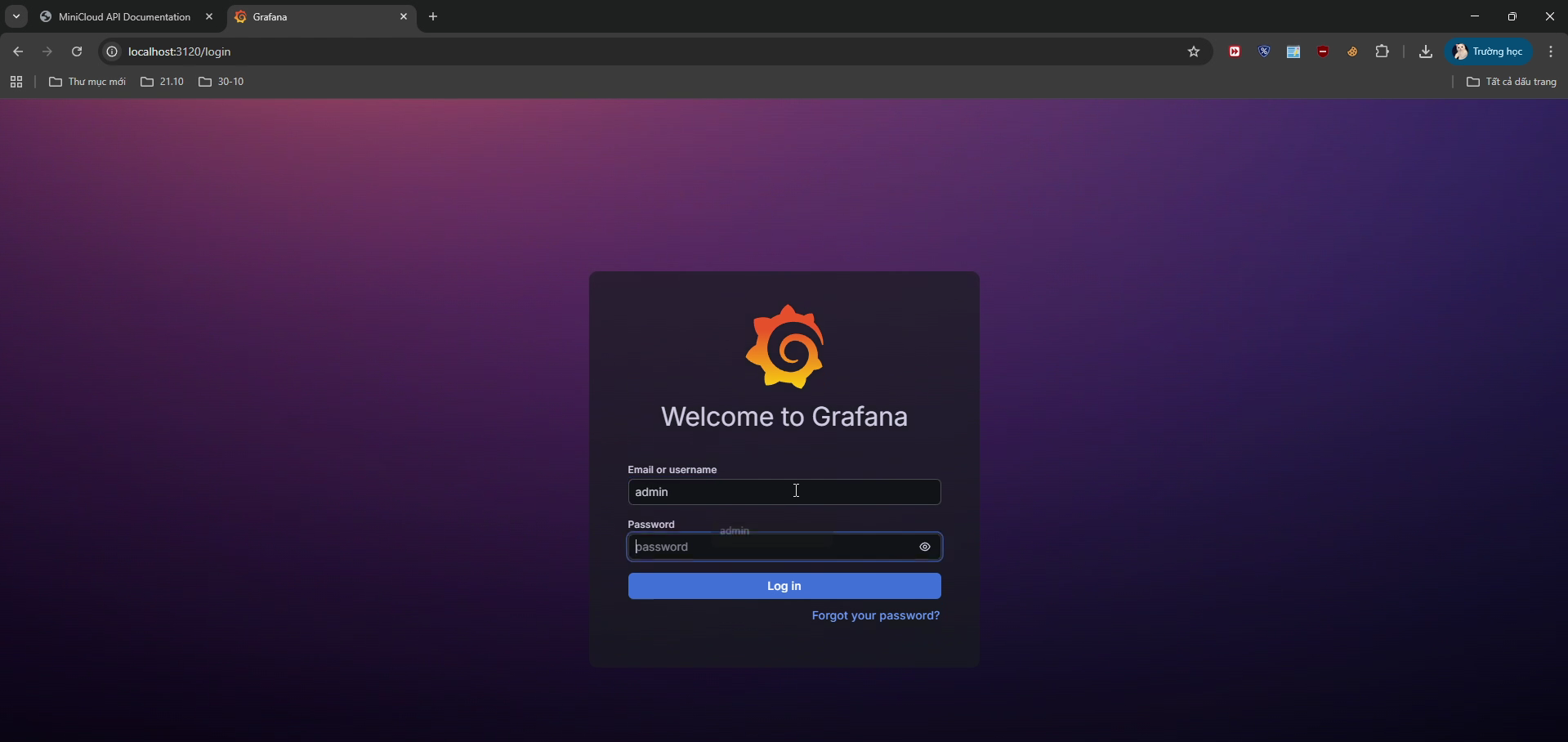


Hình 4.19: Các target trong Prometheus đều có trạng thái “UP”

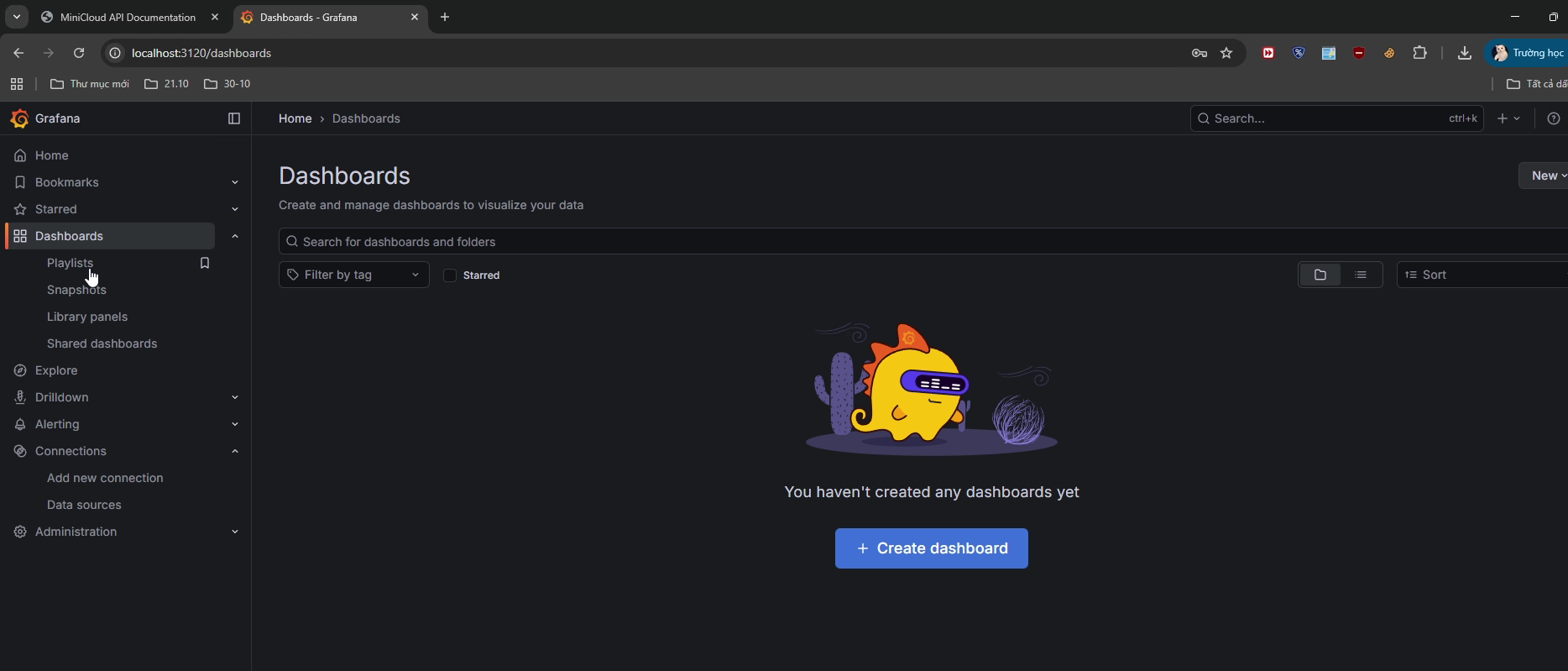


Hình 4.20: Truy vấn “node\_cpu\_seconds\_total” trong tab “Graph”

### Logging / Grafana

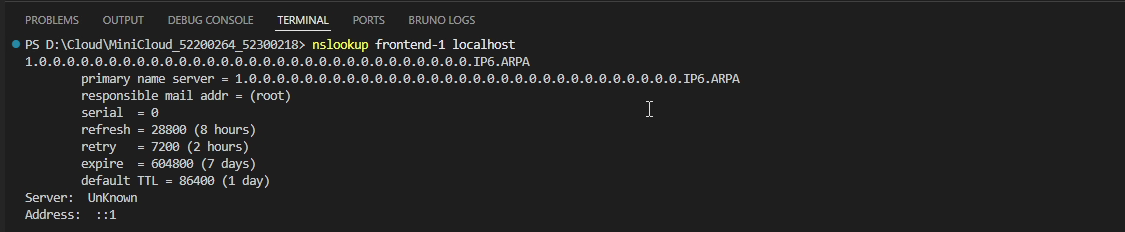


Hình 4.21: Đăng nhập Grafana.



Hình 4.22: Dashboard của Grafana.

### Reverse Proxy / API Gateway + Load Balancer



Hình 4.23: Logs của server frontend

# ĐÁNH GIÁ VÀ PHÂN TÍCH

## Mức độ đáp ứng yêu cầu đề tài

### Hoàn thành yêu cầu cốt lõi

Hệ thống MiniCloud cơ bản cần mô phỏng 9 loại server, tất cả chạy trên cùng Docker network cloud-net và được khởi động bằng một file docker-compose.yml duy nhất.

Hệ thống MyMiniCloud đã đáp ứng đầy đủ các yêu cầu này:

* **9 loại server bắt buộc** đều được triển khai thành các container riêng biệt:
  + web-frontend-server – Web Server (Nginx static site).
  + application-backend-server – Application Server (Flask API).
  + relational-database-server – Database Server (MariaDB).
  + authentication-identity-server – Authentication / Identity Server (Keycloak).
  + object-storage-server – Object Storage Server (MinIO).
  + internal-dns-server – DNS / Name Service (Bind9).
  + monitoring-node-exporter-server – Agent thu thập metrics.
  + monitoring-prometheus-server + monitoring-grafana-dashboard-server – Monitoring & Visualization Server (Prometheus + Grafana).
  + api-gateway-proxy-server – Reverse Proxy / API Gateway (Nginx).

**Tất cả container dùng chung mạng nội bộ cloud-net**:Network cloud-net được khai báo trong docker-compose.yml, mọi service đều tham gia network này và giao tiếp với nhau thông qua hostname là tên service. Điều này đúng với yêu cầu kỹ thuật chung của đề.

Quản lý tập trung bằng docker-compose.yml: File docker-compose.yml mô tả đầy đủ image, port, volume, biến môi trường, restart policy và network cho từng server. Nhờ vậy, toàn bộ MiniCloud có thể được dựng lên hoặc dừng lại chỉ với vài lệnh docker compose build, docker compose up -d, docker compose down.

* **Các yêu cầu kỹ thuật khác** cũng được bảo đảm:
  + Mỗi service có container\_name rõ ràng.
  + Port expose phù hợp dải 80xx theo hướng dẫn (Web 8080, App 8085, Auth 8081, MinIO 9000/9001, DNS 1053/udp, Prometheus 9090, NodeExporter 9100, Grafana 3000, Proxy 80).
  + Những service có dữ liệu (MariaDB, MinIO, Bind9, Prometheus, Grafana) đều có volume gắn vào thư mục host.
  + Hầu hết container quan trọng cấu hình restart: unless-stopped, đúng yêu cầu đề.

### Hoàn thành các yêu cầu mở rộng (Cloud App Integration Practice)

Bảng tổng kết cấp độ 2 của đề nêu 9 yêu cầu mở rộng tương ứng 9 server và một yêu cầu bổ sung về load balancing.

Hệ thống MyMiniCloud đã triển khai đầy đủ các mục này:

* **Web Frontend – Blog cá nhân 3 bài**
* Thư mục web-frontend-server/html/blog có trang index.html tổng hợp và tối thiểu 3 bài viết riêng, truy cập qua đường dẫn /blog/.
* Truy cập http://localhost:8080/ hiển thị “MyMiniCloud – Home”, http://localhost:8080/blog/ hiển thị danh sách bài blog, kết quả HTTP 200 OK.
* **Backend API – API /**student **đọc JSON/DB**
* Ứng dụng Flask định nghĩa thêm endpoint /student, đọc dữ liệu sinh viên từ students.json hoặc bảng students trong DB studentdb và trả về JSON.
* Endpoint được truy cập trực tiếp (http://localhost:8085/student) và thông qua proxy (http://localhost/student/).
* **Database – CSDL** studentdb **và bảng** students
* Script khởi tạo trong relational-database-server/init/01\_init.sql tạo database studentdb, bảng students và chèn ít nhất 3 bản ghi với các trường student\_id, fullname, dob, major.
* **Keycloak – Realm riêng, user, client**
* Keycloak có thêm một realm được đặt tên theo MSSV, trong đó tạo ít nhất hai user (sv01, sv02) và một client flask-app kiểu public.
* Realm cung cấp endpoint OpenID Configuration, có thể dùng để thực hiện luồng lấy token test.
* **MinIO – Upload avatar và file PDF**
* Ngoài bucket demo, hệ thống tạo thêm bucket cho avatar và documents.
* Đã upload ảnh avatar (PNG/JPG) và file báo cáo PDF, kiểm thử truy cập qua URL public hoặc pre-signed URL.
* **DNS – Thêm bản ghi & test** dig
* Zone db.cloud.local bổ sung các bản ghi app-backend.cloud.local, minio.cloud.local, keycloak.cloud.local.
* Các truy vấn dig @127.0.0.1 -p 1053 app-backend.cloud.local +short trả về đúng địa chỉ IP đã cấu hình.
* **Prometheus – Thêm scrape target web**
* File prometheus.yml khai báo thêm job\_name: 'web' với target web-frontend-server:80 bên cạnh job node.
* Trang Status → Targets của Prometheus hiển thị cả hai job node và web ở trạng thái UP.
* **Grafana – Dashboard 3 biểu đồ**
* Dashboard “System Health of MSSV” gồm ba panel:
* CPU Usage (%).
* Memory Usage (%).
* Network Receive (bytes/s).
* Mỗi panel sử dụng truy vấn PromQL đúng chuẩn (CPU idle → 100-idle, Memory used = 1-available/total, Network receive = rate(bytes\_total[5m])).
* **Proxy – Route /**student**/ tới backend /**student
* nginx.conf trong api-gateway-proxy-server có location /student/ và proxy\_pass http://application-backend-server:8081/student;.
* Các lệnh curl http://localhost/student và curl http://localhost/student/ trả về JSON danh sách sinh viên, chứng tỏ route hoạt động đúng.
* **Load balancing – Round Robin giữa 2 web server (mục nâng cao)**
* Hệ thống cấu hình Nginx Reverse Proxy với hai upstream web khác nhau (ví dụ web-frontend-server1, web-frontend-server2) và thuật toán mặc định Round Robin; khi tải lại trang nhiều lần, nội dung hoặc header phản hồi luân phiên giữa 2 server, đúng yêu cầu đề.

Như vậy, MyMiniCloud không chỉ đáp ứng phần core mà còn **hoàn thành toàn bộ danh sách mở rộng** của đề tài.

## Ưu điểm của hệ thống

### Kiến trúc mô phỏng sát thực tế

Kiến trúc tổng thể “Reverse Proxy → Web/App/Auth → Database/Storage → Monitoring/Logging → DNS” bám rất sát sơ đồ kiến trúc cloud mẫu trong đề, đồng thời sử dụng các công cụ quen thuộc trong thực tế (Nginx, MariaDB, Keycloak, MinIO, Prometheus, Grafana, Bind9).

Điều này giúp:

* Nhìn vào sơ đồ là thấy rõ đường đi của request và vai trò từng lớp.
* Có thể dễ dàng mở rộng sang các mô hình phức tạp hơn (cluster database, nhiều replica web/app, thêm message queue…).

### Môi trường triển khai thống nhất, dễ tái lập

Nhờ Docker và Docker Compose:

* Toàn bộ hệ thống có thể dựng lại nhanh chóng trên bất kỳ máy nào có Docker, chỉ cần kéo mã nguồn và chạy vài lệnh.
* Cấu hình được quản lý như mã nguồn: mọi thay đổi đều thể hiện qua việc chỉnh sửa Dockerfile, docker-compose.yml, nginx.conf, prometheus.yml, file zone DNS…, rất thuận lợi cho việc review và version control.
* Mỗi service được cô lập, tránh xung đột thư viện và phiên bản.

### Tích hợp đầy đủ monitoring và visualization

Việc triển khai Node Exporter, Prometheus và Grafana mang lại các ưu điểm:

* Nhóm có thể **quan sát trực quan** tình trạng CPU, RAM, lưu lượng mạng của host/container, thay vì chỉ tin vào cảm giác.
* Làm quen với mô hình observability phổ biến trong DevOps/SRE: metric được thu thập bởi exporter, được Prometheus lưu trữ, và Grafana trực quan hóa trên dashboard.

### Gắn kết tốt giữa các thành phần

Một ưu điểm lớn là các phần mở rộng không làm rời rạc hệ thống mà **gắn kết các server với nhau:**

* Blog web có thể nhúng link tới file được lưu trên MinIO.
* API /student lấy dữ liệu từ studentdb trong MariaDB.
* API Gateway route cả web, app, auth, storage, /student/ qua một điểm vào duy nhất.
* DNS nội bộ cung cấp hostname cho web/app/minio/keycloak; Prometheus và Grafana chạy độc lập nhưng đọc dữ liệu từ Node Exporter và web server.

Nhờ đó, MyMiniCloud đáp ứng đúng tinh thần “Cloud App Integration Practice” – không chỉ cài từng server rời rạc mà còn **tích hợp ứng dụng xuyên suốt 9 server.**

## Hạn chế của hệ thống

Dù đã hoàn thành toàn bộ yêu cầu đề, MyMiniCloud vẫn còn một số hạn chế khi so với một hệ thống production thực sự.

### 5.3.1. Chưa có tính sẵn sàng cao (High Availability)

* Toàn bộ container chạy trên **một Docker host**; khi host gặp sự cố (mất điện, lỗi OS), toàn bộ dịch vụ đều dừng.
* Database MariaDB, MinIO, Prometheus chưa được triển khai dạng cluster hoặc replication.

Trong hệ thống thật, cần có nhiều node (cluster Docker/Kubernetes), cơ chế failover, backup/restore tự động để đảm bảo HA.

### Bảo mật còn đơn giản

* Hệ thống chưa cấu hình HTTPS/TLS; các request đi qua HTTP, phù hợp môi trường lab nhưng không an toàn nếu expose ra Internet.
* Các credential ban đầu (như admin/admin của Keycloak, minioadmin /minioadmin của MinIO, admin/admin của Grafana) cần được thay đổi, quản lý bằng secret manager nếu triển khai thật.
* Chưa áp dụng cơ chế phân quyền chi tiết (RBAC) cho bucket MinIO hoặc dashboard Grafana.

### Thiếu logging tập trung và tracing

* Hệ thống chưa triển khai stack logging (ví dụ Loki, ELK) để thu thập và tìm kiếm log từ tất cả container.
* Chưa có distributed tracing (Jaeger, OpenTelemetry), nên khi có lỗi xuyên qua nhiều service thì việc lần theo luồng request sẽ khó khăn hơn.

### Tự động hóa triển khai còn hạn chế

* Việc build/push image và khởi động stack hiện vẫn thực hiện thủ công.
* Chưa tích hợp CI/CD (GitHub Actions, GitLab CI) để tự động build, test, push image lên Docker Hub mỗi khi cập nhật code.

## Khó khăn gặp phải trong quá trình thực hiện

### Khối lượng công nghệ lớn trong thời gian ngắn

Đề tài MyMiniCloud yêu cầu sinh viên cùng lúc làm việc với nhiều công nghệ: Docker, Nginx, Flask, MariaDB, Keycloak, MinIO, Bind9, Prometheus, Grafana, cùng các khái niệm DNS, reverse proxy, metrics, PromQL…

Các khó khăn chính:

* Việc đọc hiểu file cấu hình (nginx.conf, prometheus.yml, db.cloud.local) đòi hỏi phải kết hợp tài liệu môn học và tài liệu chính thức của từng phần mềm.
* Kiến thức về PromQL và cấu hình datasource Grafana ban đầu khá khó, dễ sai cú pháp hoặc chọn sai metric.

### Cấu hình và debug network Docker

Trong quá trình triển khai:

* Có lúc container không thể gọi nhau do sai tên service hoặc quên join network cloud-net.
* Một số port bị trùng với dịch vụ khác trên host, dẫn đến container khởi động nhưng không bind được port.
* Bind9 và MariaDB khá nhạy với quyền truy cập file, nên nếu volume chưa chuẩn quyền hoặc sai đường dẫn dễ khiến container restart liên tục.

Việc giải quyết những vấn đề này giúp nhóm làm quen với:

* Lệnh docker logs, docker exec, docker inspect.
* Cách dùng ping, curl, dig giữa các container để kiểm tra kết nối nội bộ.

### Thiết kế sơ đồ và trình bày báo cáo

Ngoài phần kỹ thuật, nhóm cũng mất nhiều thời gian để:

* Thiết kế các sơ đồ kiến trúc, network, flow request sao cho dễ đọc và đúng yêu cầu “Sơ đồ network, flow request giữa server”.
* Tìm hiểu kiến thức, ôn tập tài liệu, xem thêm tài liệu ngoài để tìm hiểu sâu về đề tài

## Hướng phát triển và mở rộng

Dựa trên kiến trúc hiện tại, MyMiniCloud có thể được phát triển thêm theo nhiều hướng:

1. **Triển khai trên môi trường cloud thực tế**
   * Đưa toàn bộ stack lên một máy ảo EC2, cấu hình domain và HTTPS bằng Let’s Encrypt, qua đó biến MiniCloud thành một demo cloud thực thụ.
2. **Nâng cấp kiến trúc HA và scale-out**
   * Tách database MariaDB thành cluster (Galera hoặc primary/replica).
   * Nhân bản nhiều instance web/app, phía trước là Nginx hoặc một ingress controller nếu chuyển sang Kubernetes.
3. **Bổ sung logging và tracing**
   * Triển khai Loki hoặc ELK stack để gom log từ tất cả container.
   * Kết hợp OpenTelemetry/Jaeger cho phép trace toàn đường đi của request qua Proxy → App → DB.
4. **Tích hợp Auth sâu hơn vào ứng dụng**
   * Kết nối Flask app với Keycloak bằng OIDC (middleware) để bảo vệ một số route, ví dụ /student/secure.
   * Sử dụng role/realm của Keycloak để phân quyền truy cập API.
5. **Hoàn thiện quy trình DevOps**

* Xây dựng pipeline CI/CD: khi push code mới lên Git, hệ thống tự động build image, push lên Docker Hub và deploy lại stack.
* Viết thêm script backup/restore cho MariaDB, MinIO, cấu hình Keycloak và Grafana.

# PHỤ LỤC

Link video demo:

<https://drive.google.com/drive/folders/1eRmr1AZIMJjLZxstWTp5joPefV4GQWzN?usp=sharing>

Link github:

<https://github.com/iifuu04/MiniCloud_52200264_52300218>