

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG



NGÔ VŨ MINH ĐẠT
NGUYỄN ĐOÀN KHẮC HUY

ĐỒ ÁN CHUYÊN NGÀNH

Xây dựng quy trình triển khai tự động dựa trên kiến trúc microservice

Building a deployment process based on microservice architecture

TP. HỒ CHÍ MINH, 2024

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG



NGÔ VŨ MINH ĐẠT – 21521935

NGUYỄN ĐOÀN KHẮC HUY - 21522151

ĐỒ ÁN CHUYÊN NGÀNH

Xây dựng quy trình triển khai tự động dựa trên kiến trúc microservice

Building a deployment process based on microservice architecture

GIẢNG VIÊN HUỐNG DẪN

Ths.Lê Anh Tuấn

TP. HỒ CHÍ MINH, 20

LỜI CẢM ƠN

Đồ án "Phát triển nền tảng dựa trên tính toán cận biên cho các ứng dụng IoT thông minh" là thành quả cho quá trình nỗ lực không ngừng nghỉ của chúng tôi trong việc rèn luyện và trau dồi kiến thức. Đồ án đánh dấu bước tiến quan trọng trong việc áp dụng kiến thức chuyên môn vào giải quyết một vấn đề thực tiễn mang tính đột phá trong lĩnh vực Internet vạn vật (IoT).

Mục lục

Chương 1. Mở đầu	10
1.1. Đặt vấn đề	10
1.2. Mục tiêu của đề tài	10
1.3. Đối tượng nghiên cứu và kết quả mong muốn	10
Chương 2. Cơ sở lý thuyết	11
2.1. Điện toán biên (edge computing)	11
2.1.1. Kiến trúc của Edge computing	11
2.1.2. Lợi ích	12
2.2. Kiến trúc Microservices	12
2.2.1. Kiến trúc một khối (Monolithic architecture)	12
2.2.2. Kiến trúc dịch vụ nhỏ (Microservice architecture)	13
2.2.3. Tại sao nên sử dụng microservice	13
2.3. Sonarqube	14
2.4. Docker	15
2.4.1. Kiến trúc Docker	15
2.4.2. Các thành phần chính của Docker	16
2.4.3. Dockerfile	17
2.5. Kubernetes	18
2.5.1. Giới thiệu	18
2.5.2. Kiến trúc thành phần chính	18
2.5.3. Pod	19
2.5.4. Networking	20
2.5.5. Volume	22
2.5.6. ConfigMap	24
2.5.7. Secret	26
2.6. ArgoCD	27

2.6.1.	Tổng quan	27
2.6.2.	Kiến trúc	28
2.6.3.	Cơ chế hoạt động	29
2.6.4.	Trình bày kết quả cho người dùng thông qua WebUI hoặc API.....	31
2.7.	Rancher	32
2.7.1.	Thành phần chính	32
2.7.2.	Chức năng chính.....	33
2.8.	RKE2.....	33
2.9.	Prometheus.....	35
2.9.1.	Kiến trúc	36
2.9.2.	Vai trò của Prometheus trong quản lý K8s.....	36
2.10.	Grafana	37
Chương 3.	Triển khai hệ thống	38
3.1.	Sơ đồ hệ thống	38
3.2.	Sourcecode microservice	38
3.3.	Deployment Repository	39
3.4.	Gitlab.....	39
3.5.	Pipeline repository	40
3.6.	Private Registry.....	41
3.7.	Argo CD và Kubernets Cluster.....	42
3.8.	Prometheus và Grafana	43
3.9.	Triển khai các service trên kubernetes.....	46
3.9.1.	Triển khai mô hình cluster.....	46
3.9.2.	Triển khai ứng dụng lên kubernetes	48

DANH MỤC HÌNH

Hình 2.1.1: Kiến Trúc tổng quan của mô hình Edge Computing	11
Hình 2.2.1: Kiến trúc monolithic	12
Hình 2.2.2: Kiến trúc microservice.....	13
Hình 2.3.1: Sonarqube overview	15
Hình 2.4.1: Kiến trúc tổng quan của Docker [4]	16
Hình 2.4.2: Chi tiết file Dockerfile	17
Hình 2.5.1: Kiến trúc Kubernetes.....	18
Hình 2.5.2 Luồng hoạt động networking của Kubernetes	20
Hình 2.5.3: File yaml về đối tượng kubernetes “Service”	21
Hình 2.5.4: Đối tượng kubernetes “Endpoints”.....	21
Hình 2.5.5: Nhiều Service có thể truy cập được thông qua 1 Ingress.	22
Hình 2.5.6: Mô tả volume mount trong Kubernetes.....	23
Hình 2.5.7: Mô tả mối quan hệ PersistentVolume và PersistentVolumeClaim.....	23
Hình 2.5.8: Luồng hoạt động và mối quan hệ giữa PV, PVC và SC.....	24
Hình 2.5.9: File yaml mẫu cấu hình ConfigMap.....	25
Hình 2.5.10: File yaml mẫu sử dụng ConfigMap trong Pod.	25
Hình 2.5.11: File yaml về secret loại DockerConfig.	26
Hình 2.5.12: File yaml về secret loại Opaque.	26
Hình 2.6.1: Kiến trúc của Argo CD	28
Hình 2.6.2: Mô hình hóa quá trình triển khai dự án bằng ArgoCD.....	29
Hình 2.6.3: Mô hình duy trì và tự động điều chỉnh Kubernetes.....	30
Hình 2.6.4: Sau khi triển khai ứng dụng trên ArgoCD.....	31
Hình 2.7.1: Logo Rancher.	32
Hình 2.8.1: Logo RKE2.....	33
Hình 2.9.1: Logo Prometheus.....	35
Hình 2.9.2: Kiến trúc Prometheus.....	36
Hình 2.10.1: Logo Grafana.....	37
Hình 3.1.1: Sơ đồ hệ thống.....	38
Hình 3.2.1: Repo chứa tất cả dự án microservice.....	39
Hình 3.3.1: Repo chứa file cấu hình Kuberneete của tất cả microservices.	39
Hình 3.4.1: Gitlab runner.....	40
Hình 3.5.1: ví dụ về Pipeline Repository.....	40
Hình 3.5.2: Cài đặt mẫu của User Service.....	41
Hình 3.6.1: Registry Harbor.	42

Hình 3.7.1: File cấu hình Argocd được cài đặt trên Cluster.....	43
Hình 3.8.1: Giao diện quản lý tổng cluster.....	44
Hình 3.8.2: Giao diện quản lý network của mod-edges.	44
Hình 3.8.3: Thông báo thông qua telegram.	45
Hình 3.8.4: Thông báo thông qua Slack.	45
Hình 3.9.1: Số lượng máy ảo sử dụng.	47
Hình 3.9.2: Các kubernetes nodes sau khi triển khai.....	47
Hình 3.9.3: Rancher dashboard.	47
Hình 3.9.4: Số lượng pod mà đồ án sử dụng.	48
Hình 3.9.5: UI login.....	49
Hình 3.9.6: Trang chính của ứng dụng.	49
Hình 3.9.7: AI service.....	50
Hình 3.9.8: Authentication service	50
Hình 3.9.9: User service.	51
Hình 3.9.10: Task service	51
Hình 3.9.11: File service.....	52

DANH MỤC TỪ VIẾT TẮT

Kí hiệu chữ viết tắt	Chữ viết đầy đủ
API	Application programming interface
JSON	JavaScript Object Notation
HTML	Hypertext Markup Language
XML	Extensible Markup Language
ORM	Object-Relational Mapping
YOLO	You Only Look Once
CI/CD	Continuous Integration/ Continuous Deployment
CI/CD	Continuous Integration/ Continuous Delivery

DANH MỤC CÁC THUẬT NGỮ

Tên thuật ngữ	Ý nghĩa
Business logic	Logic nghiệp vụ: là việc chuyển đổi các quy tắc doanh nghiệp thành các đoạn code trong ứng dụng, mô tả việc data sẽ được xử lý như thế nào.
Application server	Là Server cung cấp các tài nguyên và dịch vụ cần thiết để chạy ứng dụng
Decomposing	Là quá trình tách các thành phần của ứng dụng có kiến trúc một khối thành các ứng dụng có dịch vụ nhỏ
Unbundling	Là quá trình đóng gói các mảng các dịch vụ nhỏ bao gồm mã nguồn, các file cấu hình, database thành một đơn vị nguyên khối (self-contained unit)
Production	Là môi trường cho người dùng cuối cùng. Môi trường này cần đảm bảo không được lỗi và tối ưu để đưa đến cho người dùng
Gitlab Runner	Là “lính” đứng lắng nghe lệnh từ Gitlab server để chạy các công việc. Gitlab runner là một phần mềm có thể cài trên bất kỳ máy nào.
Gitlab Executor	Là chế độ đứng ra thực hiện công việc, tùy thuộc vào quy trình sẽ sử dụng các chế độ khác nhau.

Route	Điều hướng request đến nơi khác.
-------	----------------------------------

Chương 1. Mở đầu

1.1. ĐẶT VĂN ĐỀ

Trong thời đại hiện tại, sự phát triển nhanh chóng của công nghệ, phần cứng và phần mềm đã thúc đẩy sự phát triển của các ứng dụng thông minh dựa trên tính toán cận biên (edge computing). Để đáp ứng nhu cầu thiết yếu của con người, nhóm của chúng tôi đã quyết định xây dựng một hệ thống cận biên thông minh, tích hợp công nghệ trí tuệ nhân tạo vào quá trình nhận diện. Hệ thống này được thiết kế dựa trên kiến trúc microservices, giúp giải quyết các vấn đề liên quan đến việc sử dụng tài nguyên nhiều hơn trên điện toán đám mây, đồng thời giảm chi phí hoạt động và đảm bảo thời gian thực. Đồng thời, chúng tôi sẽ tích hợp quy trình triển khai CI/CD tự động tích hợp với Kubernetes (K8s) để đảm bảo việc triển khai và quản lý ứng dụng một cách hiệu quả.

1.2. MỤC TIÊU CỦA ĐỀ TÀI

- Áp dụng kiến thức đã học vào đề tài nhóm thực hiện.
- Xây dựng được sản phẩm hoàn chỉnh bao gồm backend và frontend.
- Tạo nên một sản phẩm hoàn chỉnh có thể phục vụ cộng đồng qua các dịch vụ thiết thực, giao diện thân thiện và dễ thao tác.
- Xây dựng quy trình Devops CICD Pipeline và Kubernetes.
- Có hệ thống thông báo, logging, đánh giá khi phát hiện vi phạm sử dụng AI.

1.3. ĐỐI TƯỢNG NGHIÊN CỨU VÀ KẾT QUẢ MONG MUỐN

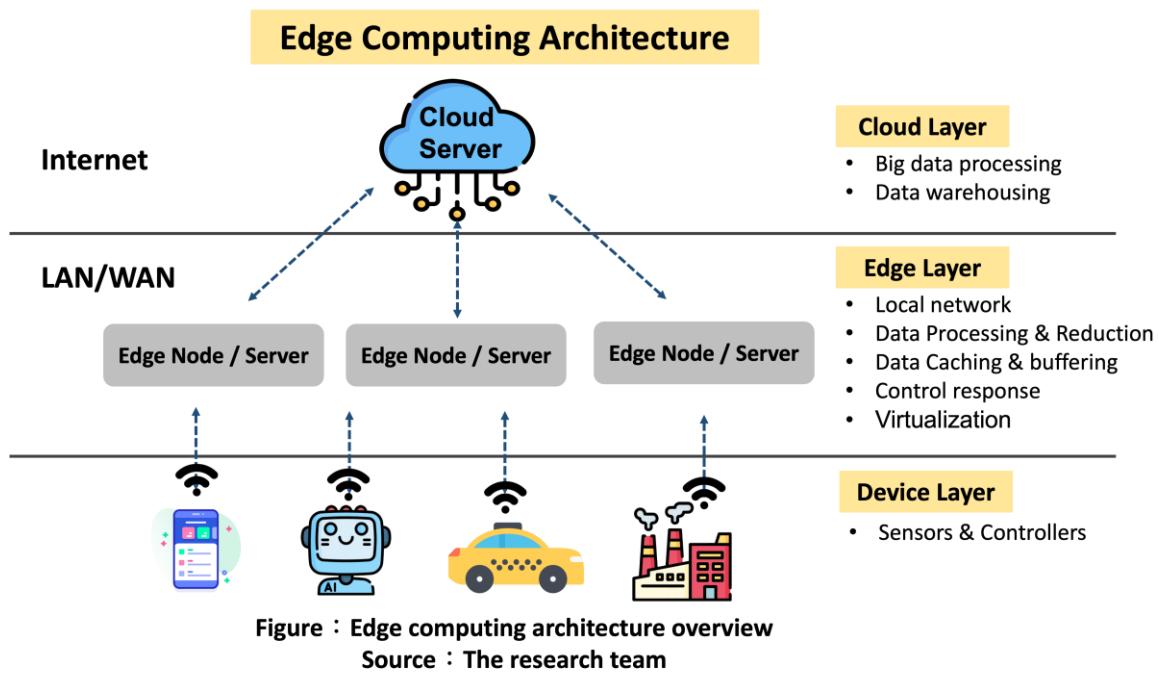
- Đối tượng nghiên cứu: Mô hình deep learning sử dụng Yolo, Thiết bị nhúng jetson Nano và camera.
- Kết quả mong muốn đạt được: Giải quyết được bài toán nhận diện người đeo khẩu trang trong thời gian thực.

Chương 2. Cơ sở lý thuyết

2.1. ĐIỆN TOÁN BIÊN (EDGE COMPUTING)

Điện toán biên (Edge Computing) là một mô hình điện toán phân tán mang tính đột phá, trong đó việc xử lý dữ liệu và lưu trữ thông tin được thực hiện tại “rìa” của mạng, gần với nguồn dữ liệu được tạo ra. Mô hình này khác biệt so với mô hình điện toán đám mây truyền thống, nơi dữ liệu được tập trung xử lý tại các trung tâm dữ liệu lớn và xa xôi. [1]

2.1.1. Kiến trúc của Edge computing



Hình 2.1.1: Kiến Trúc tổng quan của mô hình Edge Computing

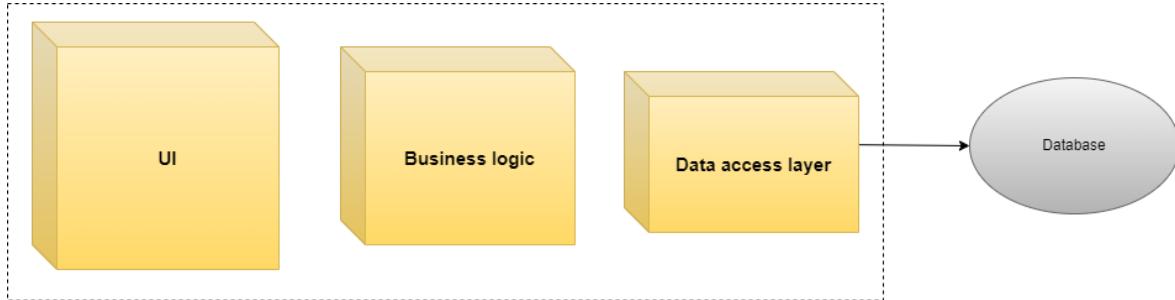
- Tầng cloud (cloud layer): là một cloud của bên thứ 3 hoặc tự host, hoặc đôi khi là một trung tâm dữ liệu. Chúng có vai trò quan trọng trong việc lưu trữ và chạy các ứng dụng, điều phối và quản lý các nút biên (edge node) [1].
- Tầng biên (Edge layer): là nơi xử lý và lưu trữ dữ liệu ngay tại điểm thu thập dữ liệu, giúp giảm độ trễ và tiết kiệm băng thông [1].
- Thiết bị biên (Device layer): là thiết bị có khả năng tính toán như máy ATM, máy ảnh số, hoặc ô tô, được tích hợp sẵn khả năng xử lý dữ liệu [1].

2.1.2. Lợi ích

- Với đồ án là xây dựng framework cho các thiết bị IOT thông minh thì sử dụng kiến trúc edge computing mang lại các lợi ích như:
- Giảm Độ Trễ (Latency): Edge Computing cho phép xử lý dữ liệu ngay tại nguồn, giúp giảm độ trễ trong việc truyền dữ liệu đến trung tâm dữ liệu.
- Tiết Kiệm Băng Thông: Việc xử lý dữ liệu tại chỗ giúp giảm lượng dữ liệu cần truyền qua mạng, từ đó tiết kiệm băng thông và giảm chi phí liên quan.
- Tăng Cường Bảo Mật: Khi dữ liệu được xử lý tại biên mạng, ít phải di chuyển qua mạng, giảm nguy cơ bị tấn công hoặc rò rỉ thông tin.
- Cải Thiện Độ Tin Cậy: Xử lý dữ liệu gần nguồn giúp giảm phụ thuộc vào kết nối mạng liên tục và độ tin cậy của trung tâm dữ liệu xa.
- Hỗ Trợ IoT và Ứng Dụng Thời Gian Thực: Edge Computing phù hợp với các ứng dụng IoT cần phản hồi tức.

2.2. KIẾN TRÚC MICROSERVICES

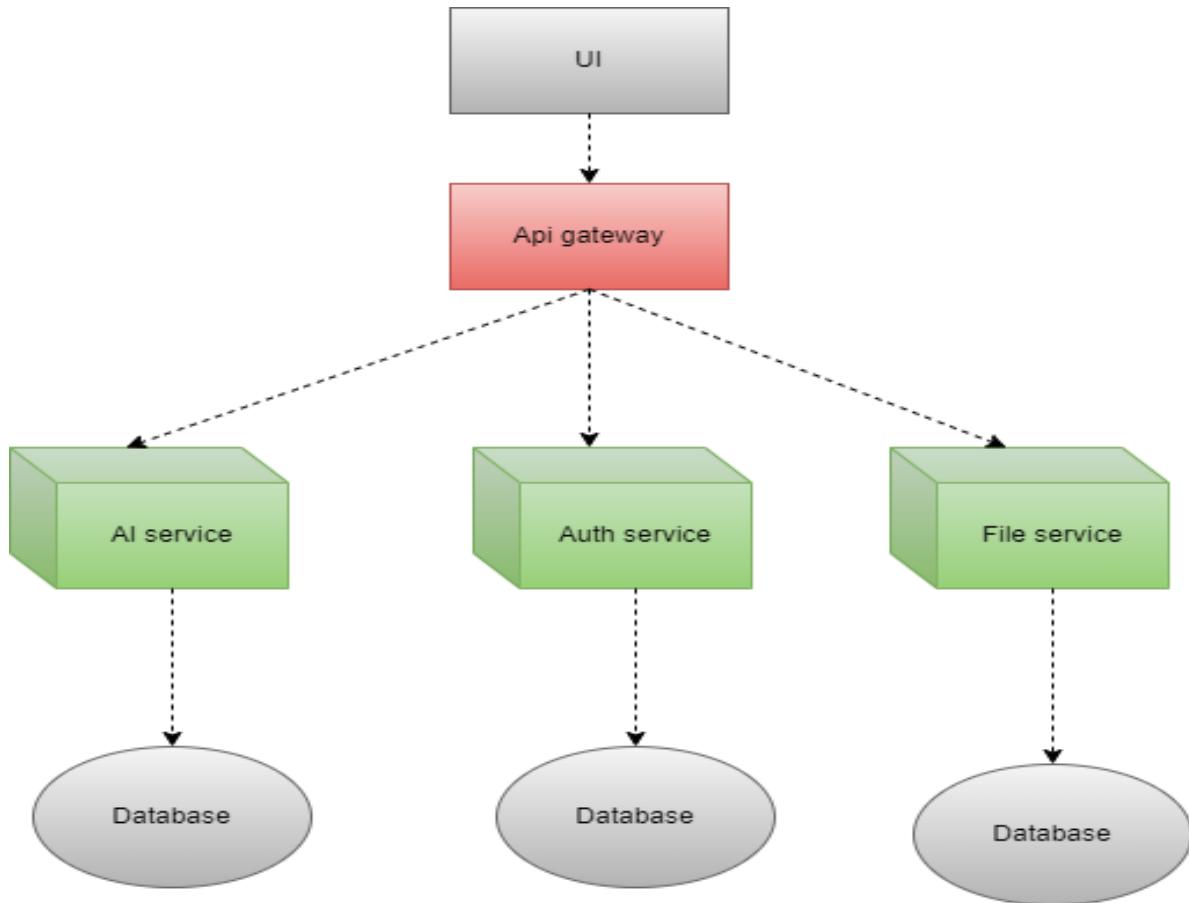
2.2.1. Kiến trúc một khối (Monolithic architecture)



Hình 2.2.1: Kiến trúc monolithic

Kiến trúc một khối là dạng phát triển ứng dụng, và khi triển khai lên thì ở 1 dạng khối duy nhất. Tất cả các UI, các logic nghiệp vụ (business logic), các logic để truy cập database sẽ được đóng gói thành 1 ứng dụng duy nhất và sẽ được triển khai trên máy chủ ứng dụng (Application server) [2].

2.2.2. Kiến trúc dịch vụ nhỏ (Microservice architecture)



Hình 2.2.2: Kiến trúc microservice

Kiến trúc microservices là một phương pháp thiết kế phần mềm tiên tiến, hướng đến việc phân chia ứng dụng một khối (Monolithic) thành các dịch vụ nhỏ (microservice), độc lập và tự chủ. Mỗi dịch vụ được thiết kế để thực hiện một chức năng cụ thể, sở hữu logic nghiệp vụ riêng biệt và được phát triển, triển khai, vận hành độc lập với các dịch vụ khác [3].

Khi viết ứng dụng microservice là quá trình phân tách (decomposing) và đóng gói (unbundling). Đầu tiên là phân tách các thành phần của ứng dụng có kiến trúc một khối thành các ứng dụng microservice. Thứ 2 là đóng gói các módules các microservices bao gồm mã nguồn, các file cấu hình, database thành một đơn vị nguyên khái (self-contained unit) [3].

2.2.3. Tại sao nên sử dụng microservice

Khi ứng dụng một khối ngày càng lớn thì sẽ gặp các vấn đề sau đây [3]

- Integration Complexity: Các ứng dụng cần kết nối với nhiều dịch vụ và cơ sở dữ liệu khác nhau, khiến việc xây dựng chúng trở nên khó khăn hơn .

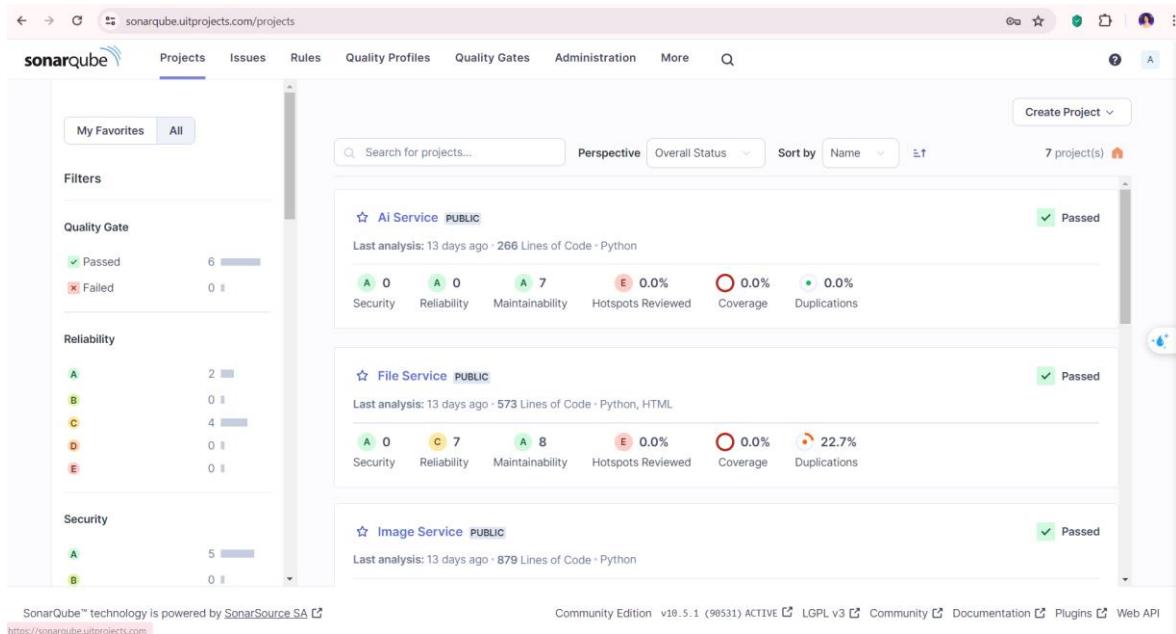
- Faster Delivery: Khách hàng mong muốn các tính năng mới nhanh chóng, vì vậy phần mềm cần được phát hành thành các phần nhỏ hơn thường xuyên hơn.
- Khả năng mở rộng: Các ứng dụng cần xử lý những thay đổi không thể dự đoán đối với lưu lượng truy cập của người dùng, cần mở rộng lên hoặc xuống tùy theo nhu cầu.
- Tính khả dụng cao: Ứng dụng phải luôn sẵn sàng để tránh mất khách hàng vào tay đối thủ cạnh tranh

Do đó với kiến trúc microservice, có thể xây dựng cả một hệ thống với các tính chất như sau: [3]

- Linh hoạt (flexible): Các dịch vụ rời rạc có thể được tích hợp và sắp xếp lại để nhanh chóng cung cấp các chức năng mới. Đơn vị mã càng nhỏ thì càng dễ thay đổi và thời gian để kiểm thử và triển khai mã càng ít.
- Dàn hồi (resilient): Các dịch vụ rời rạc giúp ứng dụng không còn là một "gói hỗn độn" (ball of mud), nơi sự cố ở một phần của ứng dụng có thể khiến toàn bộ ứng dụng bị lỗi. Các sự cố có thể được cô lập trong một phần nhỏ của ứng dụng và được khắc phục trước khi toàn bộ ứng dụng dừng hoạt động. Điều này cũng cho phép ứng dụng giảm hiệu suất một cách linh hoạt trong trường hợp xảy ra lỗi không thể phục hồi.
- Khả năng mở rộng: Các dịch vụ rời rạc có thể dễ dàng phân bố theo chiều ngang trên nhiều máy chủ, giúp việc mở rộng các tính năng/dịch vụ một cách linh hoạt.

2.3. SONARQUBE

Sonarqube là công cụ kiểm tra chất lượng mã nguồn trong môi trường phát triển phần mềm, Sonarqube hỗ trợ nhiều loại ngôn ngữ, có thể quét và phát hiện các lỗ hỏng, bugs lỗi logic và đưa báo cáo đến cho người dùng. Công cụ này có thể cấu hình để tích hợp vào quy trình CI của dự án, giúp đảm bảo code chất lượng nhất trước khi đưa đến cho người dùng cuối.



Hình 2.3.1: Sonarqube overview

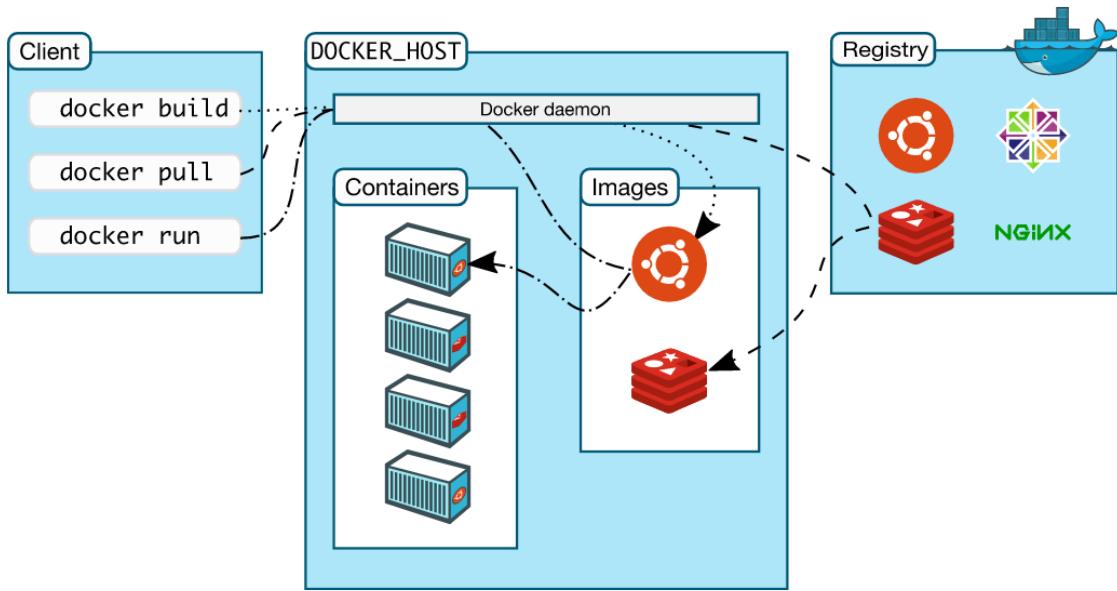
2.4. DOCKER

Docker là nền tảng cho phép đóng gói ứng dụng và môi trường vào một container, giúp việc triển khai trở nên nhất quán trên mọi môi trường. Dễ dàng tích hợp với quy trình CI.

2.4.1. Kiến trúc Docker

Docker sử dụng kiến trúc client – server. Docker client sử dụng một REST API (qua UNIX socket, hoặc cổng mạng) để có thể giao tiếp với Docker daemon, tiến trình này thực hiện công việc tạo, chạy và phân phối các Docker container. Docker client và daemon có thể chạy trên cùng một hệ thống hoặc có thể kết nối Docker client với Docker daemon từ xa [4].

2.4.2. Các thành phần chính của Docker



Hình 2.4.1: Kiến trúc tổng quan của Docker [4]

- **Docker daemon (dockerd):** lắng nghe các yêu cầu của người dùng thông qua Docker API và quản lý các đối tượng Docker như image, container, network và volume. Một daemon cũng có thể giao tiếp với các daemon khác để quản lý các Docker service [4].
- **Docker client:** là cách thức người dùng tương tác với Docker. Khi sử dụng các câu lệnh như docker run, client thông qua Docker API sẽ gửi các lệnh này đến dockerd, nơi thực hiện chúng. Docker client có thể giao tiếp với nhiều hơn một Docker daemon [4].
- **Docker registry:** là nơi lưu trữ các Docker image. Docker Hub là nơi lưu trữ Docker image công khai (public registry) mà bất kỳ ai cũng có thể sử dụng và Docker được định cấu hình mặc định để tìm image trên Docker Hub. Ngoài ra, có thể cấu hình các registry riêng tư khác như opensource Harbor để lưu trữ Docker image [4].
- **Docker Image:** là template read-only (chỉ cho phép đọc) với các hướng dẫn để tạo Docker container. Image sẽ được sử dụng để đóng gói các ứng dụng và các thành phần đi kèm của ứng dụng, được lưu trữ ở server hoặc trên registry. Khi pull image từ dockerhub, thật chất là tải Dockerfile của người khác hoặc bên thứ 3 về, sau đó docker daemon sử dụng các câu lệnh hướng dẫn trong Dockerfile để tạo Container. [4]
- **Docker container:** Là một “thùng chứa”, mà nhờ công nghệ của docker mà “thùng chứa” này chỉ chứa tất cả source code và các công cụ, môi trường cần thiết để ứng dụng này chạy. Có thể tương tác thông qua các lệnh docker start, stop, [4]

2.4.3. Dockerfile

Dockerfile là một file ghi ra các lệnh để hướng dẫn docker daemon build container.

- Dưới đây là ví dụ dockerfile của auth-service được sử dụng trong đồ án.

The screenshot shows a code editor window with a dark theme. The tab bar at the top says "auth-service\ Dockerfile". The code itself is a Dockerfile with the following content:

```
1 FROM python:3.8-slim
2 ENV PIP_DISABLE_PIP_VERSION_CHECK=1
3 ENV PYTHONDONTWRITEBYTECODE=1
4 ENV PYTHONUNBUFFERED=1
5 WORKDIR /app
6 RUN apt-get update && \
7     apt-get install libffi-dev libpq-dev gcc -y && \
8     apt-get clean && \
9 COPY ./requirements.txt ./requirements.txt
10 RUN python -m pip install -r ./requirements.txt
11 COPY ./app app
12 RUN groupadd -r auth-service && \
13     useradd -r -g auth-service -d /app auth-service && \
14     chown -R auth-service:auth-service /app && \
15     chmod -R 700 /app && \
16 RUN rm -rf /tmp/* && \
17     rm -rf /var/tmp/* && \
18     rm -rf /root/.cache/* \
19     rm -rf /var/lib/apt/lists/*
20 CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]
21 EXPOSE 8000
```

Hình 2.4.2: Chi tiết file Dockerfile.

Sau đó sử dụng câu lệnh “docker build .” . File DockerFile này sẽ được gửi đến docker Daemon (docker server), Daemon sẽ đọc file DockerFile rồi tiến hành làm những thao tác sau (giải thích theo line) [5].

- 1: FROM keyword chỉ định bản image mà dự án auth-service này sử dụng, version nên lớn hơn version dự án sử dụng, và ưu tiên sử dụng image có tag là alpine. Image alpine thường tối ưu nhẹ, phù hợp với triển khai trên môi trường production
- 6 – 8: Cài các thư viện, công cụ cần thiết để ứng dụng chạy
- 9 - 11: Cài đặt các thư viện Python mà dự án sử dụng, sau đó copy mã nguồn vào image
- 12 – 15: Vì mặc định container khi chạy sẽ sử dụng quyền root, điều này khá nguy hiểm nên cần tạo một user với đủ quyền hoạt động

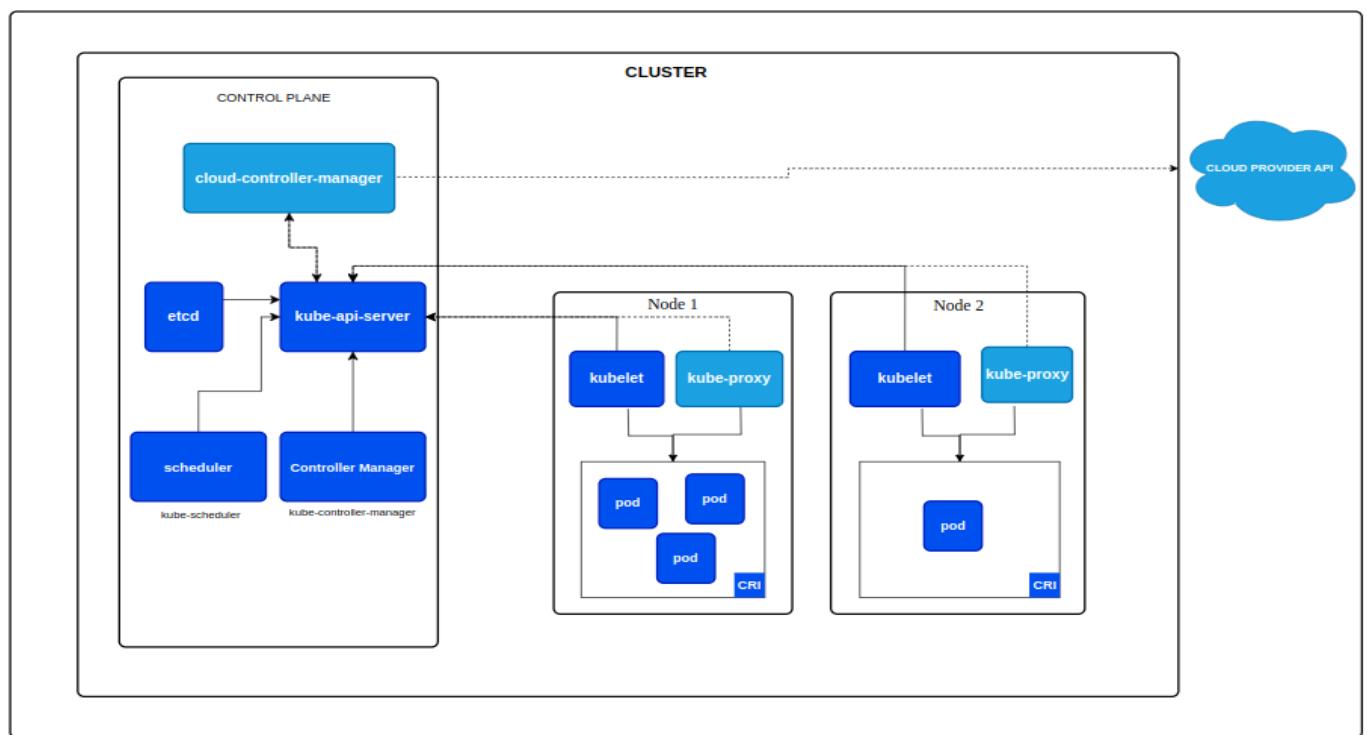
- 16 – 19: Xoá các file không cần thiết để tối ưu kích thước image
- 20 – 21: Expose port container và câu lệnh để container khởi động

2.5. KUBERNETES

2.5.1. Giới thiệu

Kubernetes là một hệ thống orchestration container mã nguồn mở để tự động hóa việc triển khai, mở rộng và quản lý phần mềm. Được thiết kế ban đầu bởi Google, dự án hiện do một cộng đồng những người đóng góp trên toàn thế giới bảo trì và thương hiệu được nắm giữ bởi Cloud Native Computing Foundation.

2.5.2. Kiến trúc thành phần chính



Hình 2.5.1: Kiến trúc Kubernetes.

2.5.2.1. Control plane

- **kube-api-server:** là interface chính, đại diện cho control plane và cluster.
- **Etdc:** là interface chính, đại diện cho control plane và cluster.
- **kube-scheduler:** đảm nhận việc scheduling(tiến trình mà chọn một node trong cluster để chạy container).
- **kube-controller-manager:** có trách nhiệm quản lý các utility(tiện ích) trên k8s.

- **cloud-controller-manager(option):** cung cấp interface giữa k8s và các mô hình cloud(AWS, Azure, Google Cloud).

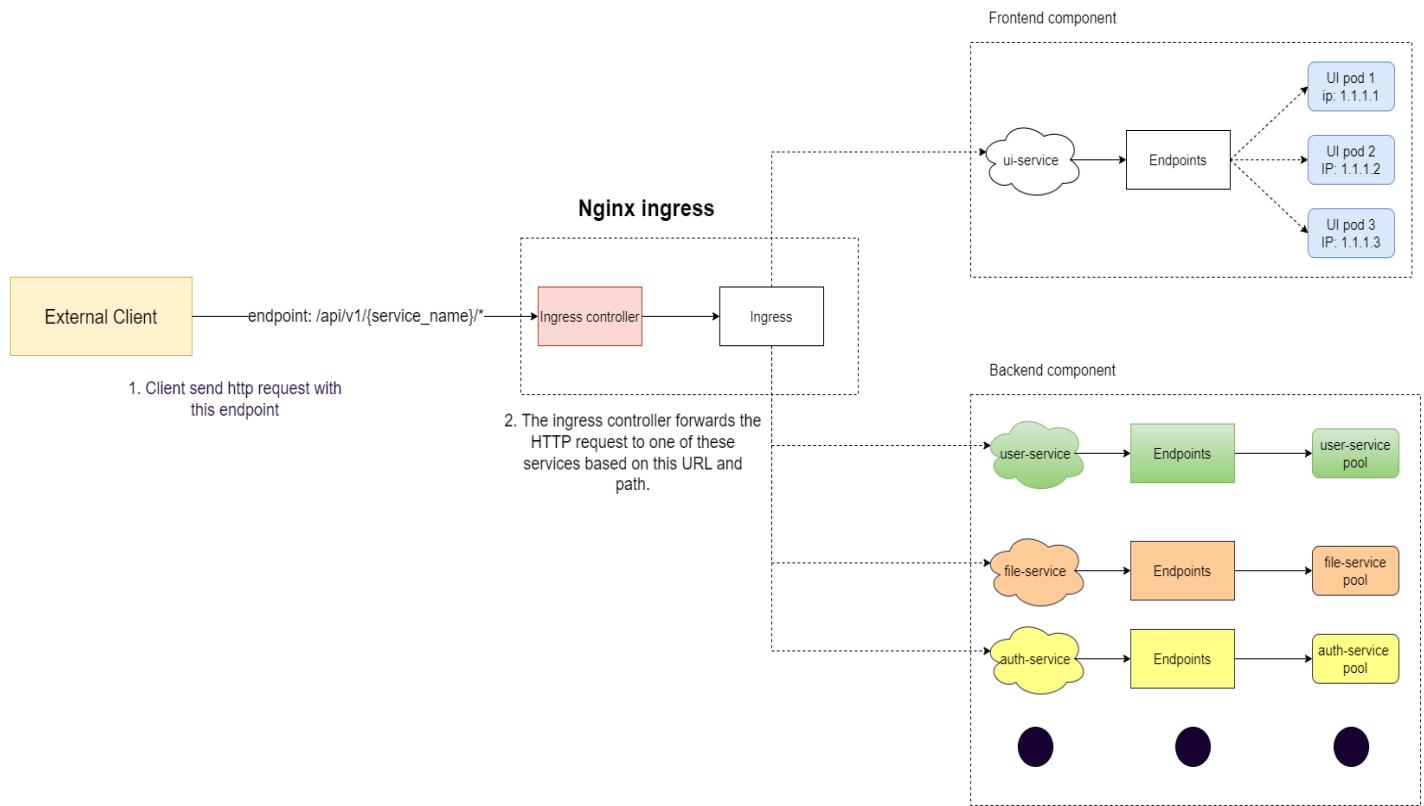
2.5.2.2. Node

- Node là các máy đang chạy hay đó là nơi chứa các container được quản lý bởi control plane.
- **Kubelet:** có nhiệm vụ liên lạc với control plane và chắc chắn rằng container đang được chạy trên node mà control plane chỉ định.
- **container runtime:** là một phần mềm chạy container, không phải là một phần của k8s, nhưng để chạy container trên mỗi node thì k8s cần có container runtime.
- **kube-proxy:** đảm nhận nhiệm vụ liên quan đến cung cấp mạng giữa các container và services trong cluster.

2.5.3. Pod

- Pod là đơn vị nhỏ nhất trong Kubernetes, 1 Pod có thể chứa 1 hay nhiều container hoạt động đồng thời. Tuỳ thuộc vào tính chất dự án mà có thể chọn 1 pod chạy với nhiều container hoặc một container. Thường là sẽ là một do dễ quản lí và kiểm soát phần logging dễ hơn khi 1 container bị down. [7]
- Các container trong cùng một pod sẽ share cùng một tài nguyên, cùng môi trường, cùng địa chỉ IP. Có thể cấu hình để giới hạn tài nguyên sử dụng của pod. [7]
- Container: là các image đã được push lên Dockerhub hoặc private registry.
- Về việc triển khai các pod này lên node nào sẽ được tự động chọn bởi thành phần Schedule, Pod có thể ở node 1 khi khởi tạo, nhưng có thể bị phá huỷ và triển khai tự động lại trên Node 2 [8].

2.5.4. Networking



Hình 2.5.2 Luồng hoạt động networking của Kubernetes.

2.5.4.1. Service

- Để các pod giao tiếp được với nhau mà không lo về việc địa chỉ ip của pod bị thay đổi khi pod restart. (pod bị restart trong Kubernetes có nghĩa là pod đó sẽ bị xoá và thay thế bằng một pod mới hoàn toàn khác, và địa chỉ của chúng sẽ bị thay đổi. Thì service được sinh ra để giải quyết vấn đề này [8].)
- Ở phần sourcecode, chỉ code phần \$name_service, sau đó deploy trên Kubernetes, các service sẽ giao tiếp được với nhau thông qua dns server của Kubernetes mà không cần quá quan tâm địa chỉ ip là gì.
- Các Service biết nên forward đến pod nào nhờ vào việc cấu hình sao cho match Label khi giữa các pod và Service. Ví dụ file cấu hình user-service trong đồ án:

```

1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: user-service-svc
5   annotations:
6     cloud.google.com/neg: '{"ingress": true}'
7   labels:
8     service: user-service-app
9   namespace: mod-edges
10
11 spec:
12   type: ClusterIP
13   selector:
14     app: user-service-app
15 ports:
16   - targetPort: 8002
17     name: http
18     port: 8002
19     protocol: TCP

```

```

1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: user-service-deployment
5   namespace: mod-edges
6 spec:
7   replicas: 3
8   selector:
9     matchLabels:
10    app: user-service-app
11   template:
12     metadata:
13       labels:
14         app: user-service-app
15   spec:
16     containers:
17       - image: registry-harbor.ngodat0103.me/mod-edges/user-service:1.0.0
18         name: user-service
19         ports:
20           - containerPort: 8002
21         envFrom:
22           - configMapRef:

```

Hình 2.5.3: File yaml về đối tượng kubernetes “Service”

2.5.4.2. Endpoints (endpointSlice)

- Vì các pod có thể scale lên làm 2 - 3 - 4 pod tùy thuộc vào độ tải của ứng dụng, và thế là sẽ có nhiều địa chỉ ip. Thành phần Endpoints sinh ra để lưu danh sách địa chỉ ip của các pod. [8]

```

akira@legion:/mnt/c/Users/ngovu$ kubectl get endpoints -n mod-edges
NAME                           ENDPOINTS
ai-service-db-svc              10.244.1.8:5432
ai-service-svc                 10.244.1.7:8003
auth-service-svc               10.244.2.7:8000
file-service-db-svc            10.244.2.6:5432
file-service-svc               10.244.2.10:8005
release-name-ingress-nginx-controller 10.244.0.5:443,10.244.0.5:80
release-name-ingress-nginx-controller-admission 10.244.0.5:8443
task-service-db-svc            10.244.2.8:5432
task-service-svc               10.244.0.3:8001
ui-app-svc                      10.244.0.4:3000
user-service-db-svc             10.244.1.6:5432
user-service-svc                10.244.0.7:8002,10.244.1.9:8002,10.244.2.9:8002
akira@legion:/mnt/c/Users/ngovu$ kubectl describe endpoints user-service-svc -n mod-edges
Name:           user-service-svc
Namespace:      mod-edges
Labels:         app.kubernetes.io/instance=mod-edges-dev
Annotations:   endpoints.kubernetes.io/last-change-trigger-time: 2024-06-03T12:03:35Z
Subsets:
  Addresses:    10.244.0.7,10.244.1.9,10.244.2.9
  NotReadyAddresses: <none>
  Ports:
    Name  Port  Protocol
    --  --  --
    http  8002  TCP

```

Hình 2.5.4: Đối tượng kubernetes “Endpoints”.

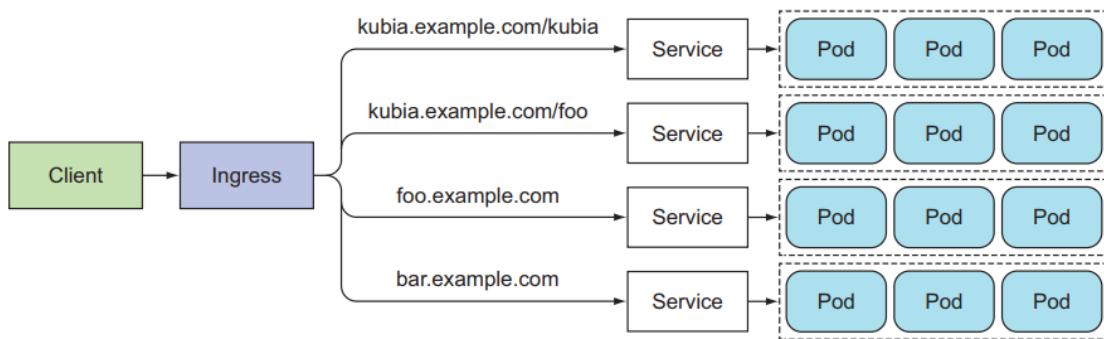
Sau đó, Service sẽ sử dụng danh sách các địa chỉ này để forward đến 1 trong các pod, từ đó tăng khả năng cân bằng tải.

2.5.4.3. Ingress Controller

Để cho phép client từ bên ngoài có thể truy cập được các Service bên trong. Có 2 cách để thực hiện là sử dụng LoadBalancer và Ingress. Nhưng Loadbalancer có đặc điểm là mỗi lần tạo là phải yêu cầu cấp loadbalancer từ Cloud. Trong khi Ingress chỉ yêu cầu có một Loadbalancer. Để giải quyết hạn chế của Loadbalancer thì đồ án sử dụng Ingress [8].

Ingress hoạt động ở tầng application, điều đó có nghĩa ingress có thể can thiệp vào phần payload, để có thể cung cấp những tính năng như cookie-based session [8].

Khi một request đến Ingress, Ingress sẽ check xem Service nào nên forward các request đến. Ingress có thể sử dụng rule là phân biệt host domain và path để route đến Service khác.



Hình 2.5.5: Nhiều Service có thể truy cập được thông qua 1 Ingress.

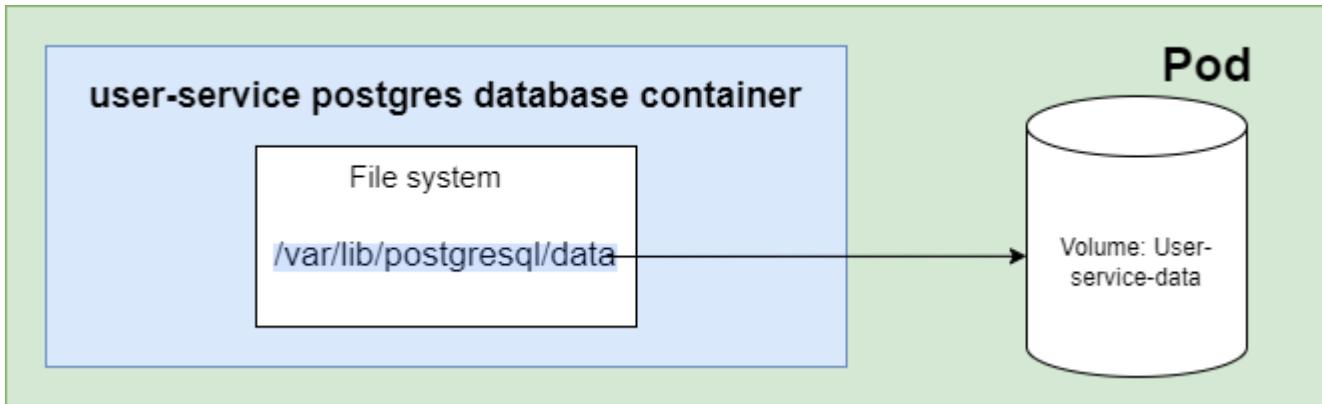
2.5.4.4. Tổng kết

Client gửi http request -> đến ingress -> ingress sẽ check các route được định nghĩa trong file cấu hình -> forward đến Service -> Service sẽ check các danh sách địa chỉ ip trong Endpoints, sau đó sử dụng một trong các địa chỉ này sẽ forward đến 1 trong các Pod thông qua cơ chế Round Robin.

2.5.5. Volume

Khi container bị hỏng và khi container khởi động lại thì sẽ không thấy được dữ liệu trước đó mà container hỏng để lại. Bởi vì container sẽ sử dụng cái data mà lúc tại thời điểm nạp vào docker image. Do đó Volume được sinh ra để giải quyết vấn đề này.

2.5.5.1. Cơ chế hoạt động



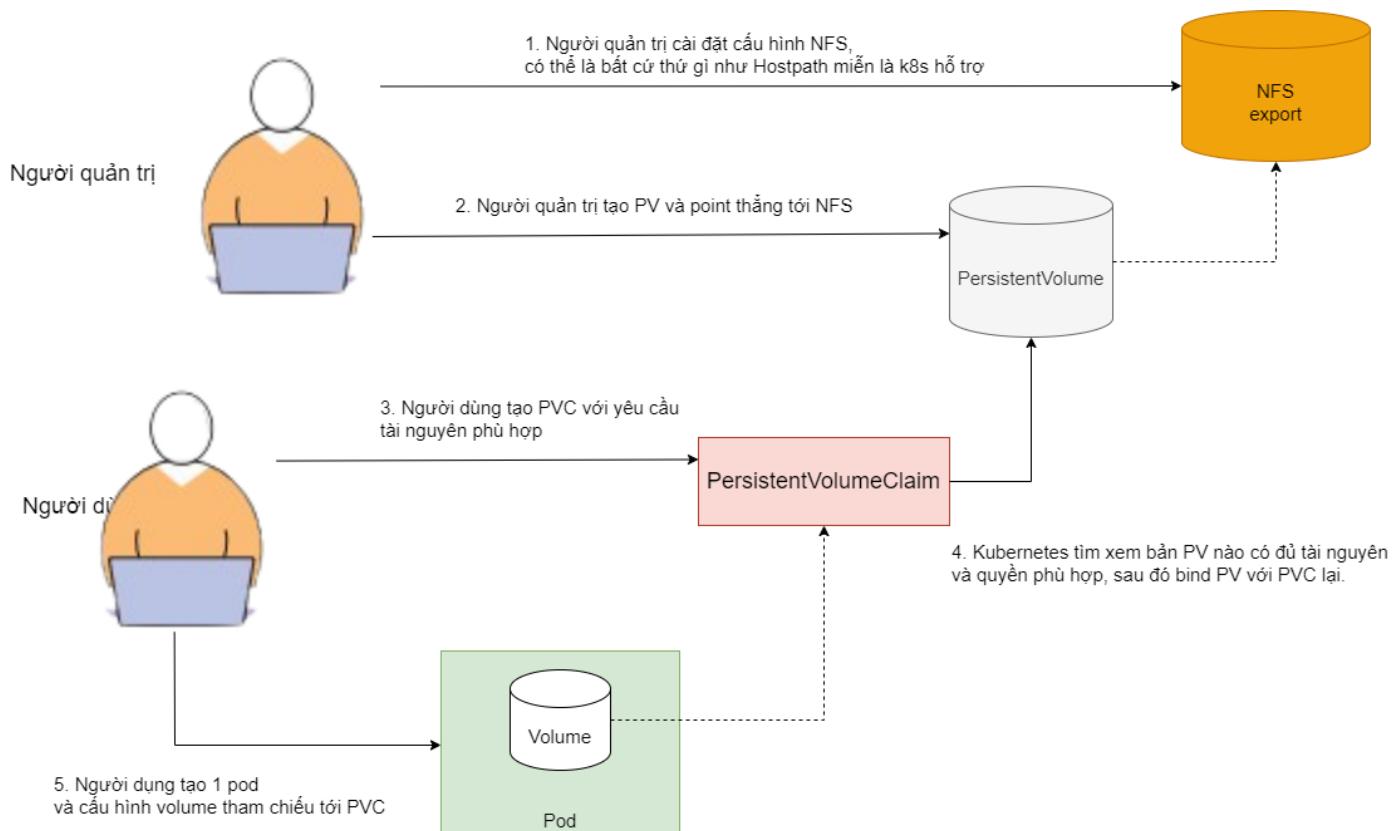
Hình 2.5.6: Mô tả volume mount trong Kubernetes.

Cách hoạt động là ta cấu hình 1 cái Volume, sau đó sẽ mount cái volume này vào đường dẫn, /var/lib/postgresql/data. Như vậy, mọi hoạt động read / write thực chất sẽ ghi thẳng vào khu vực volume đã được định nghĩa. Một khi container restart thì sẽ mount tới Volume hoạt động tiếp, không bị mất data.

Mount có nghĩa là tất cả sự thay đổi bên trong một thư mục sẽ mapping tới đích bên kia.

2.5.5.2. PersistentVolume (PV), PersistentVolumeClaim (PVC) và Persistent Volume Provisioner.

Để cho phép ứng dụng gửi yêu cầu data về lưu trữ mà không cần quan tâm về cơ sở hạ tầng. Kubernetes tạo ra PersistentVolume và PersistentVolumeClaim.

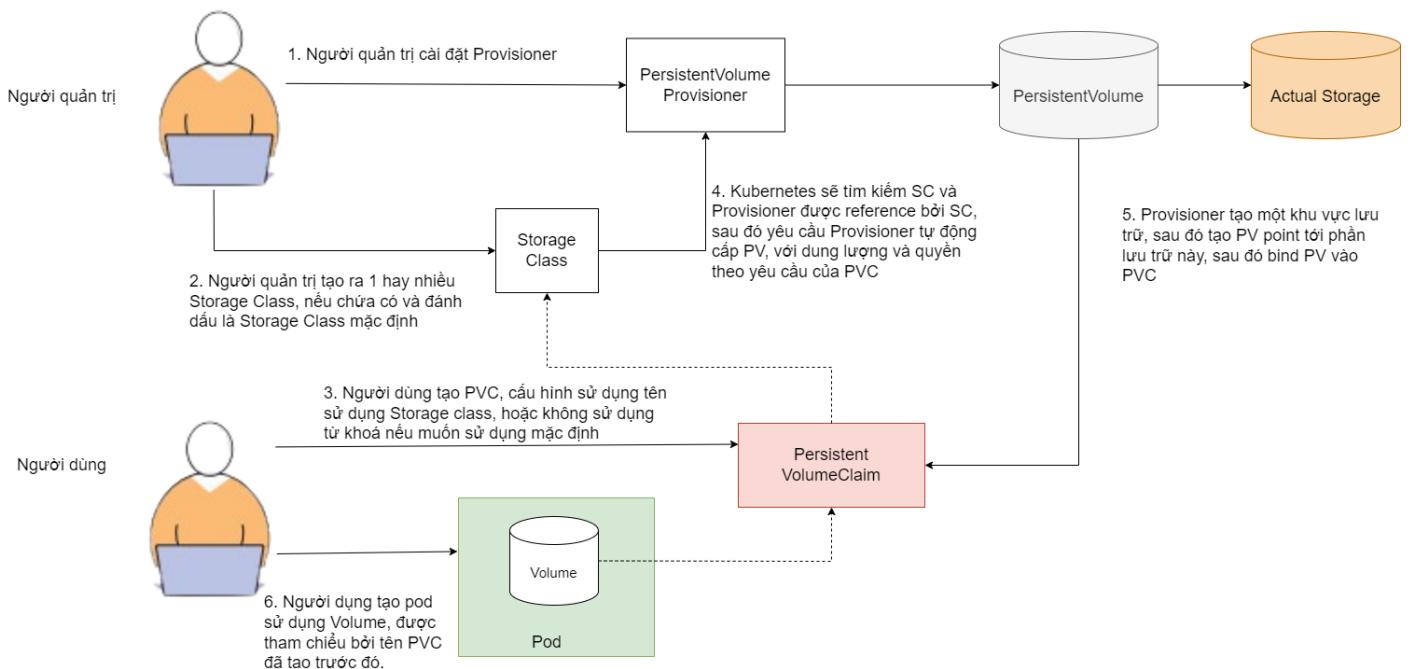


Hình 2.5.7: Mô tả mối quan hệ PersistentVolume và PersistentVolumeClaim.

- Người quản trị cluster tạo ra cấu hình PV có sẵn.
- Người dùng hay bên dev sử dụng file cấu hình PersistentVolumeClaim để yêu cầu không gian lưu trữ.
- Kubernetes sẽ tìm bản PV phù hợp và ràng buộc vào PVC.
- Sau đó cấu hình mountPath để lắp đường dẫn đến PVC.

2.5.5.3. Tự động cấp PersistentVolume

Nhưng với cấu hình này có nhược điểm là người quản trị phải cấu hình bằng tay từng PersistentVolume, Cho nên một resource mới của Kubernetes ra đời để cho phép tự động cấp PV khi người dùng request từ PVC.



Hình 2.5.8: Luồng hoạt động và mối quan hệ giữa PV, PVC và SC.

2.5.6. ConfigMap

- ConfigMap là một đối tượng API dùng để lưu trữ dữ liệu không bí mật theo cặp key-value. Các Pod có thể sử dụng ConfigMap dưới dạng biến môi trường, đối số dòng lệnh hoặc dưới dạng tệp cấu hình trong một ổ đĩa. ConfigMap cho phép tách cấu hình dành riêng cho môi trường khỏi container image.
- ConfigMap không được thiết kế để chứa khôi dữ liệu lớn. Dữ liệu được lưu trữ trong ConfigMap không thể vượt quá 1 MiB.

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: game-demo
data:
  # property-like keys; each key maps to a simple value
  player_initial_lives: "3"
  ui_properties_file_name: "user-interface.properties"

  # file-like keys
  game.properties: |
    enemy.types=aliens,monsters
    player.maximum-lives=5
  user-interface.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true

```

Hình 2.5.9: File yaml mẫu cấu hình ConfigMap.

```

apiVersion: v1
kind: Pod
metadata:
  name: configmap-demo-pod
spec:
  containers:
    - name: demo
      image: alpine
      command: ["sleep", "3600"]
      env:
        # Define the environment variable
        - name: PLAYER_INITIAL_LIVES # Notice that the case is different here
          # from the key name in the ConfigMap.
        valueFrom:
          configMapKeyRef:
            name: game-demo           # The ConfigMap this value comes from.
            key: player_initial_lives # The key to fetch.
      - name: UI_PROPERTIES_FILE_NAME
        valueFrom:
          configMapKeyRef:
            name: game-demo
            key: ui_properties_file_name
  volumeMounts:
    - name: config
      mountPath: "/config"
      readOnly: true

```

Hình 2.5.10: File yaml mẫu sử dụng ConfigMap trong Pod.

2.5.7. Secret

- Secret là một đối tượng tương tự với ConfigMap trong Kubernetes nhưng khác ở điểm được thiết kế để lưu các dữ liệu bí mật. Có nhiệm vụ lưu các thông tin nhạy cảm như mật khẩu, token, key và các thông tin có thể ảnh hưởng nghiêm trọng đến toàn bộ hệ thống.
- Các thông tin này sẽ được đưa vào bên trong các container hoặc các pod, có nghĩa là không cần phải khai báo các thông tin nhạy cảm vào trong code giúp cho các thông tin đó được bảo mật trong hệ thống của Kubernetes.
- Cơ chế hoạt động của secret là lưu dữ liệu theo cặp key-value, value sẽ được mã hóa bằng base64.
- Các loại secret được sử dụng trong đồ án
 - + Docker config: dùng để lưu thông tin của harbor registry.

```
apiVersion: v1
data:
  .dockerconfigjson: ewogICAgICAgICJhd
kind: Secret
metadata:
  name: regcred
  namespace: mod-edges
type: kubernetes.io/dockerconfigjson
```

Hình 2.5.11: File yaml về secret loại DockerConfig.

- + Opaque: là dữ liệu mặc định nếu không chỉ định rõ ràng loại secret, dùng để lưu các loại dữ liệu định danh như tài khoản, mật khẩu.

```
apiVersion: v1
kind: Secret
metadata:
  name: regcred
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: repository
type: Opaque
stringData:
  url: "http://yoururl.com/"
  username: "your username"
  password: "your password"
```

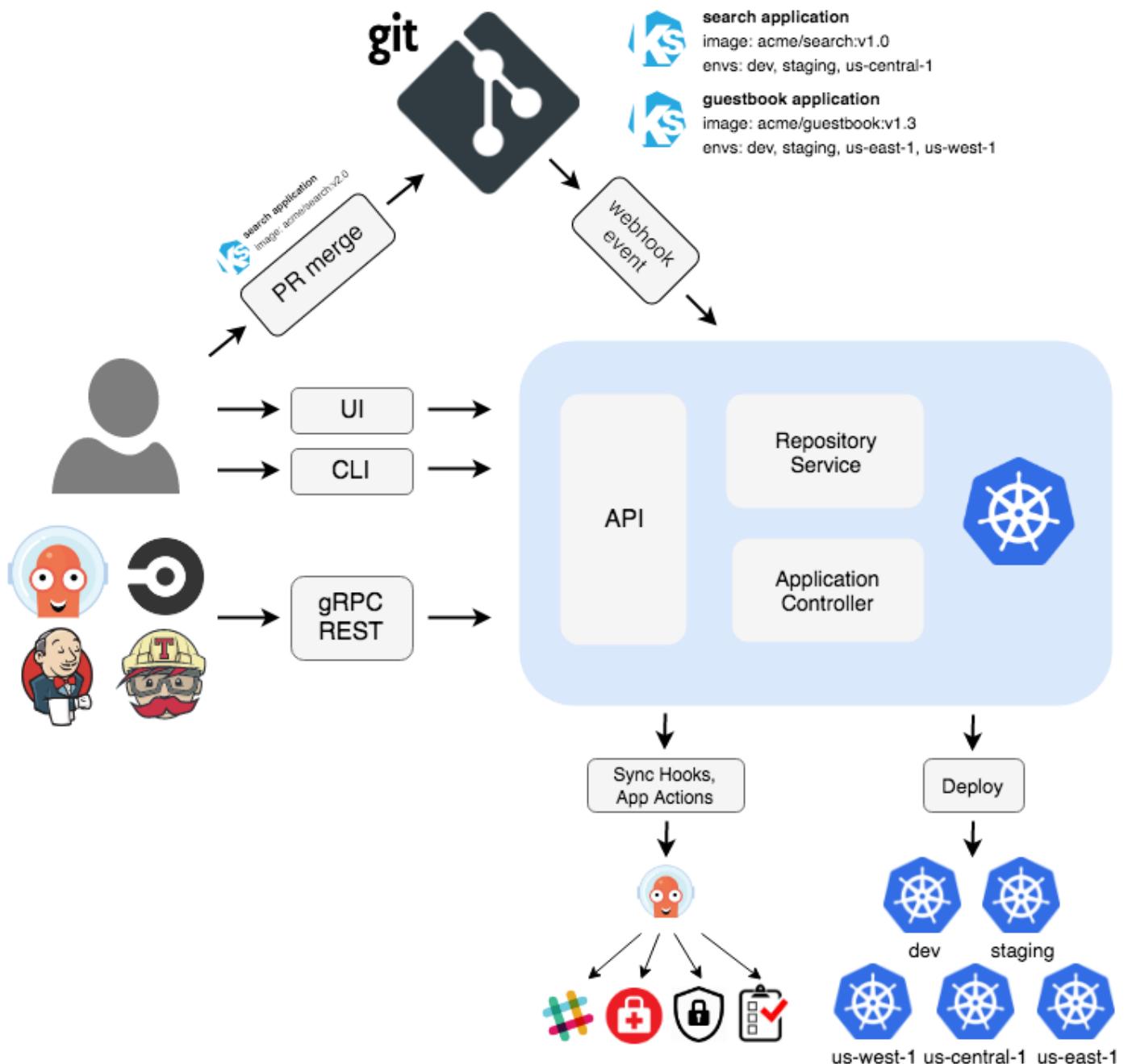
Hình 2.5.12: File yaml về secret loại Opaque.

2.6. ARGOCD

2.6.1. Tổng quan

- ArgoCD là một mã nguồn mở container-native workflow engine phục vụ việc deploy service trên Kubernetes.
- ArgoCD được triển khai trên Kubernetes như một Kubernetes CRD (Custom Resource Definition)
- ArgoCD là một công cụ dễ sử dụng cho phép các nhóm phát triển triển khai và quản lý các ứng dụng mà không cần phải tìm hiểu nhiều về Kubernetes và không cần toàn quyền truy cập vào hệ thống Kubernetes.

2.6.2. Kiến trúc



Hình 2.6.1: Kiến trúc của Argo CD

ArgoCd có 3 thành phần chính là Api server, Repository server, Application controller

2.6.2.1. API Server

Là một gRPC/REST server cung cấp các APIs, các APIs này được sử dụng bởi Argocd Web UI, CLI, và các hệ thống CI/CD khác tích hợp. API Server có thể thực hiện các chức năng như: Quản lý ứng dụng và báo cáo trạng thái, thực hiện các hành động như sync, rollback hoặc các tác vụ do người dùng định nghĩa quản lý các credential của repository và cluster (được lưu trữ như thành phần Secret trong Kubernetes), có hỗ trợ xác thực, ủy quyền và lắng nghe các sự kiện WebHook

2.6.2.2. Repository Server

Là một internal service duy trì một local cache của git repository giữ các application manifests. Chịu trách nhiệm tạo ra và trả về các kubernetes manifests dựa vào các thông tin đầu vào như repository URL, revision (commit, tag, branch), application path (đường dẫn tới thư mục chứa manifest của ứng dụng được khai báo) và template specific settings: parameters, helm values.yaml

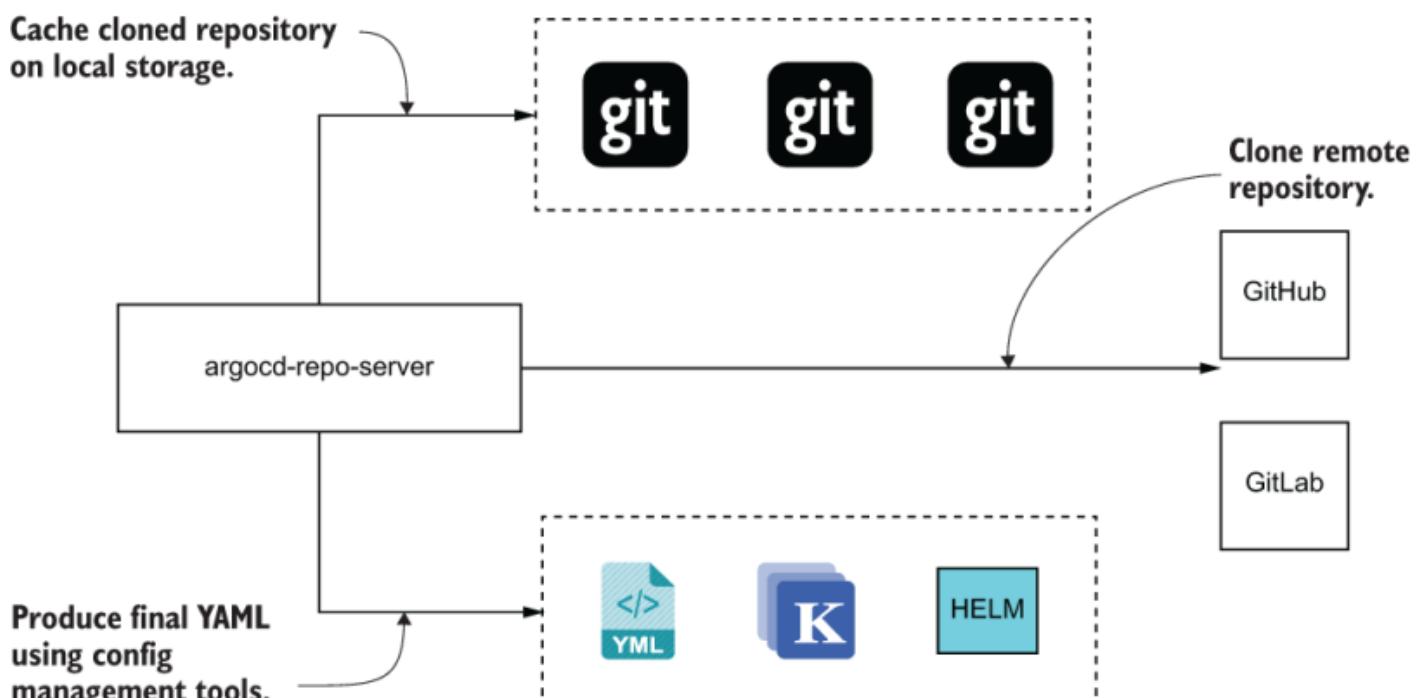
2.6.2.3. Application Controller

Là một kubernetes controller tiếp tục thực hiện việc theo dõi trạng thái của ứng dụng đang chạy (trên kubernetes) và so sánh với trạng thái mong muốn (được định nghĩa trong git repo). Phát hiện tình trạng OutOfSync của service và tùy chọn thực hiện các hành động sửa chữa, chịu trách nhiệm cho việc gọi bất kỳ user-defined hooks cho các lifecycle events (PreSync, Sync, PostSync)

2.6.3. Cơ chế hoạt động

Luồng xử lý của ArgoCD được chia thành 3 giai đoạn, mỗi giai đoạn được xử lý bởi một thành phần được liệt kê ở trên. 3 giai đoạn của gồm:

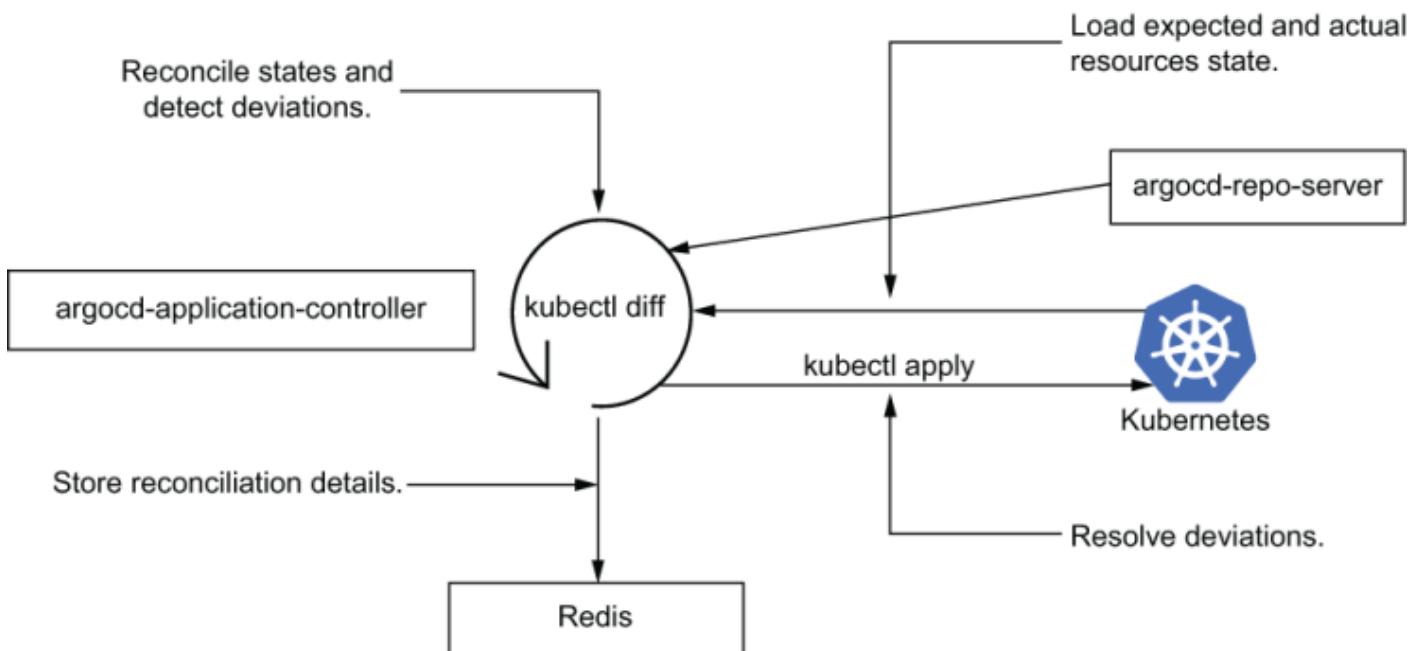
2.6.3.1. Lấy source manifest từ remote git repo



Hình 2.6.2: Mô hình hóa quá trình triển khai dự án bằng ArgoCD

ArgoCD lấy thông tin file manifest từ remote git repo thông qua bước git pull. Sau đó lưu thông tin này vào local storage của ArgoCD repository server. Tạo ra manifest phục vụ cho Argocd applicatoin controller sử dụng ở giai đoạn tiếp theo.

2.6.3.2. Phát hiện và khắc phục những thông tin sai lệch giữa manifest đang chạy với manifest trên git

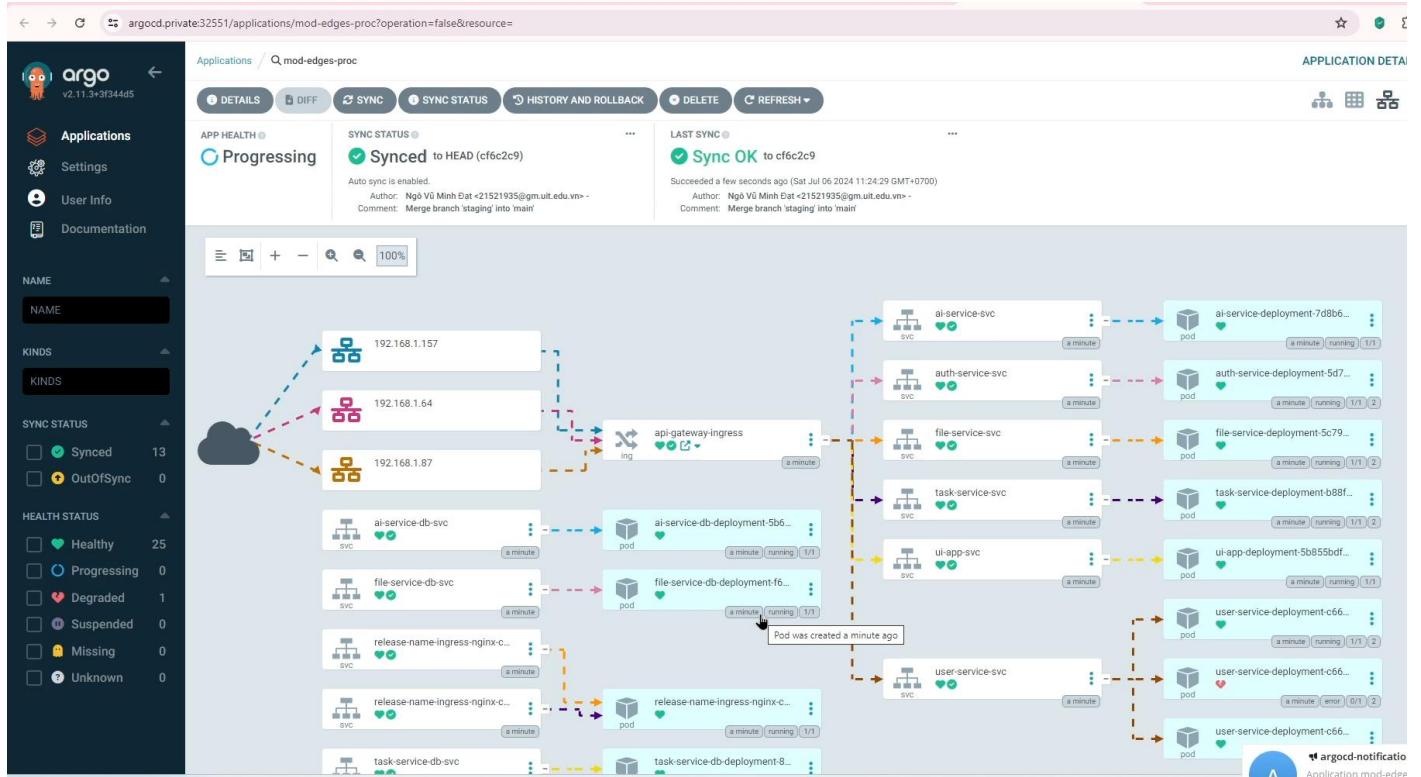


Hình 2.6.3: Mô hình duy trì và tự động điều chỉnh Kubernetes.

Đọc file manifest đang chạy trên target cluster (cụm k8s nơi mà ứng dụng được deploy). Gọi lấy yaml manifest từ argocd repo server So sánh 2 manifest (thông manifest trên repo và manifest thực tế chạy trên cụm) Thực hiện kubectl apply để apply manifest lên k8s đích nếu các manifest có sự khác biệt. Sau đó cập nhật thông tin application (manifest, sync, health status) và lưu vào redis.

2.6.4. Trình bày kết quả cho người dùng thông qua WebUI hoặc API

- Argo CD sở hữu giao diện người dùng (UI) trực quan và mạnh mẽ, giúp người dùng dễ dàng quản lý việc triển khai ứng dụng Kubernetes theo mô hình GitOps. UI cung cấp nhiều tính năng hữu ích



Hình 2.6.4: Sau khi triển khai ứng dụng trên ArgoCD

2.7. RANCHER



Hình 2.7.1: Logo Rancher.

Rancher là một công cụ nguồn mở cung cấp giao diện nền web để quản lý các container, bao gồm triển khai ứng dụng, quản lý tài nguyên, theo dõi, giám sát tình trạng chung của cluster... Đơn giản hóa việc quản lý K8S.

2.7.1. Thành phần chính

Rancher server: trung tâm của Rancher cluster bao gồm các thành phần như etcd, authentication proxy, API Rancher server, cluster control. Chức năng chính là cho phép người dùng quản lý, giám sát cung cấp các cluster khác thông qua UI.

Rancher K8S Engine: tạo các RKE cluster.

Cluster control : chịu trách nhiệm thiết lập các liên lạc an toàn giữa Rancher server và từng cluster.

Authentication Proxy: xác thực người gọi bằng các dịch vụ xác thực từ bên thứ 3 sau đó chuyển tiếp lệnh đến cluster thích hợp.

Tác nhân node: thực hiện một số thao tác trên cluster do Rancher khởi chạy.

2.7.2. Chức năng chính

- Triển khai và quản lý các Kubernetes cluster trên nhiều nền tảng, bao gồm các dịch vụ cloud được quản lý (EKS, AKS, GKE), các nhà cung cấp khác (DOKS, LKE, ACK, CCE, OKE, TKE) và on-premises.
- Tạo trình điều khiển tùy chỉnh để hỗ trợ hầu như mọi nền tảng Kubernetes hiện có.
- Cung cấp và cài đặt Kubernetes on-premises hoặc trên cloud.
- Thực thi bảo mật cấp doanh nghiệp bằng bảng điều khiển trung tâm.
- Hỗ trợ Active Directory, LDAP và SAML.
- Quản lý tất cả các Kubernetes cluster từ một giao diện duy nhất.

2.8. RKE2



Hình 2.8.1: Logo RKE2.

RKE2, còn được gọi là RKE Chính phủ, là phiên bản tiếp theo của Rancher Kubernetes Engine, phân phối Kubernetes thế hệ mới tập trung vào bảo mật và tuân thủ cho các tổ chức thuộc Chính phủ Hoa Kỳ. Được gọi là RKE2 vì đây là phiên bản tiếp theo của Rancher Kubernetes Engine dành cho các trường hợp sử dụng trung tâm dữ liệu. Bản phân phối chạy độc lập hoặc tích hợp vào Rancher. Tính năng cung cấp tự động các cụm RKE2 mới có sẵn trong Rancher v2.6+[11].

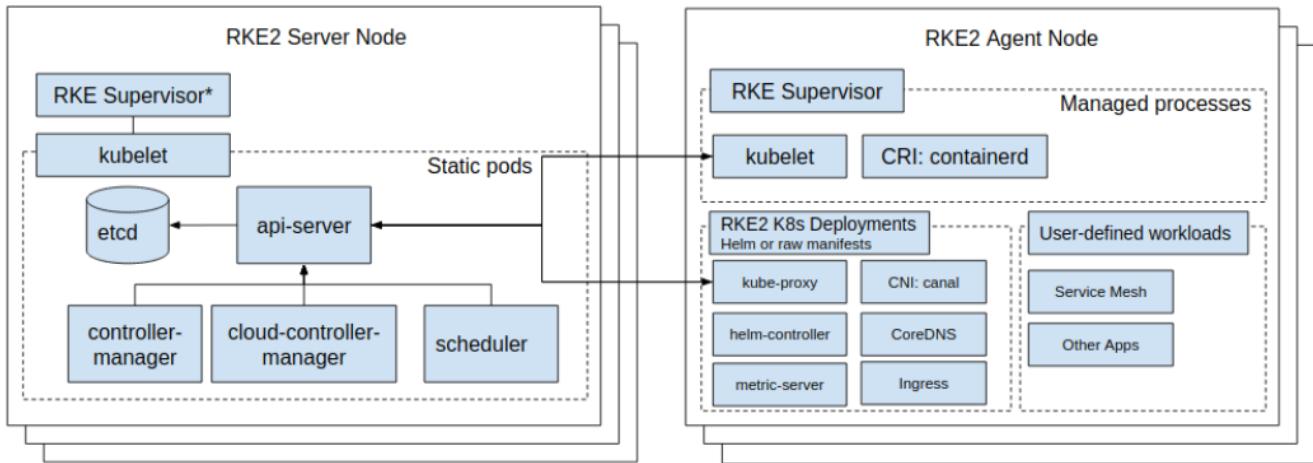


Figure 2.8.1: Kiến trúc RKE2

Cung cấp các tính năng sau:

- Thiết lập **mặc định và tùy chọn cấu hình** cho phép cụm vượt qua CIS Kubernetes Benchmark với sự can thiệp tối thiểu của người vận hành. CIS Kubernetes Benchmark là một bộ hướng dẫn thực hành bảo mật được công nhận rộng rãi dành cho các cụm Kubernetes. RKE2 đi kèm với các cấu hình mặc định giúp bạn dễ dàng đáp ứng các yêu cầu bảo mật này.
- **Cho phép tuân thủ FIPS 140-2.** FIPS 140-2 là tiêu chuẩn của Viện Tiêu chuẩn và Công nghệ Hoa Kỳ (NIST) dành cho các mô-đun mật mã. RKE2 được xây dựng để hỗ trợ tiêu chuẩn này, giúp bạn đáp ứng các yêu cầu về bảo mật dữ liệu nghiêm ngặt.
- Hỗ trợ **chính sách SELinux** và thực thi nhãn **Multi-Category Security (MCS)**. SELinux là một cơ chế kiểm soát truy cập bắt buộc (MAC) trên Linux giúp cải thiện bảo mật hệ thống. MCS là một tính năng SELinux cho phép gán nhãn bảo mật nhiều mức độ cho các đối tượng, tăng cường khả năng kiểm soát truy cập hơn nữa.
- Quét thường xuyên các thành phần để **tìm lỗ hổng bảo mật (CVE)** bằng trivy trong quy trình xây dựng của chúng. Trivy là một trình quét lỗ hổng mã nguồn containerized. Bằng cách tích hợp trivy, RKE2 đảm bảo các thành phần được cập nhật và vá lỗi kịp thời.

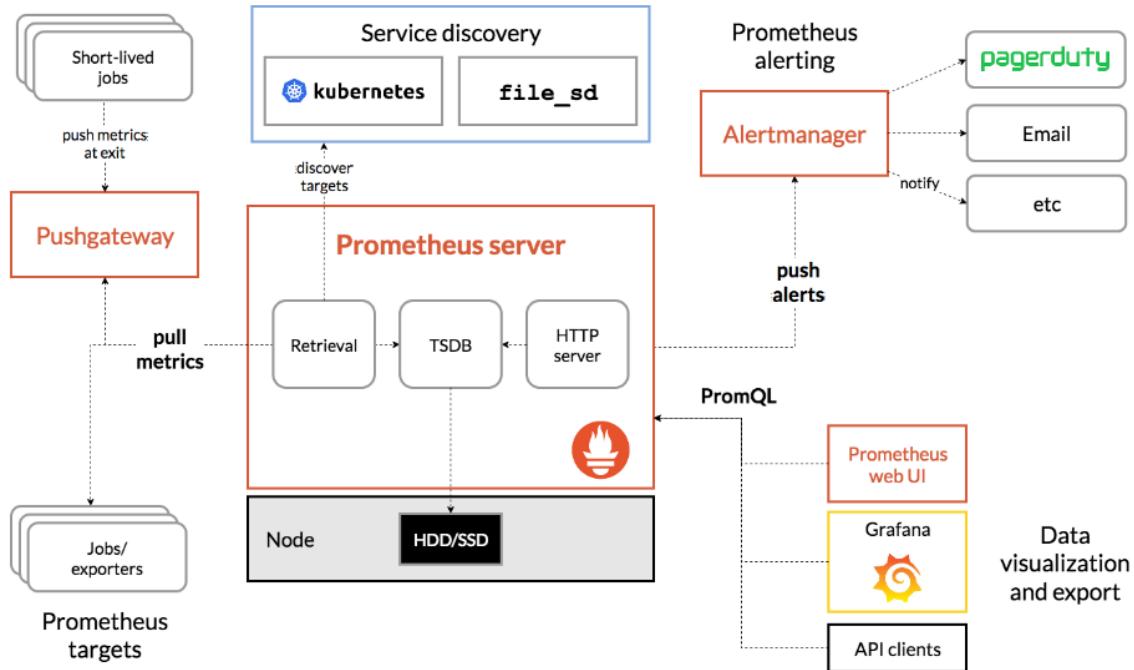
2.9. PROMETHEUS



Hình 2.9.1: Logo Prometheus.

Prometheus là một bộ công cụ theo dõi, giám sát và cảnh báo hệ thống có mã nguồn mở được xây dựng ban đầu bởi SoundCloud. Prometheus có khả năng thu thập và lưu trữ các dữ liệu metric từ các ứng dụng, hệ thống và cơ sở hạ tầng IT thông qua kênh trực tiếp hoặc dịch vụ Pushgateway trung gian và lưu trữ ở các local máy chủ. Prometheus hỗ trợ rất nhiều bộ template giám sát với các mã nguồn mở, giúp việc triển khai và cấu hình giám sát nhanh chóng, hiệu quả. Đi kèm với Prometheus thì cần có Alert Manager để có thể phát hiện được các vấn đề của hệ thống [9].

2.9.1. Kiến trúc



Hình 2.9.2: Kiến trúc Prometheus.

- Thành phần chính là Prometheus server:
 - + Time series Database: lưu trữ thông tin metrics của đối tượng giám sát.
 - + Data Retrieval Worker: lấy thông tin metric từ các đối tượng giám sát và lưu vào database.
 - + HTTP Server API: tiếp nhận các truy vấn lấy dữ liệu được lưu ở DB, hiển thị dữ liệu lên dashboard.
- Thư viện cho các ứng dụng.
- Push Gateway Prometheus: hỗ trợ các đối tượng có thời gian thực hiện ngắn, giúp các metric đẩy về pushgateway rồi về Prometheus server khi Prometheus không thể chủ động lấy dữ liệu.
- Alert Manager: quản lý, xử lý các cảnh báo.

2.9.2. Vai trò của Prometheus trong quản lý K8s

- Giám sát hiệu suất của các K8s cluster và ứng dụng chạy trên K8s.
- Theo dõi tài nguyên CPU, memory, storage của các node trong cluster.
- Giám sát sức khỏe của các pod và service.
- Phát hiện các vấn đề về hiệu suất và đưa ra cảnh báo.

2.10. GRAFANA



Hình 2.10.1: Logo Grafana.

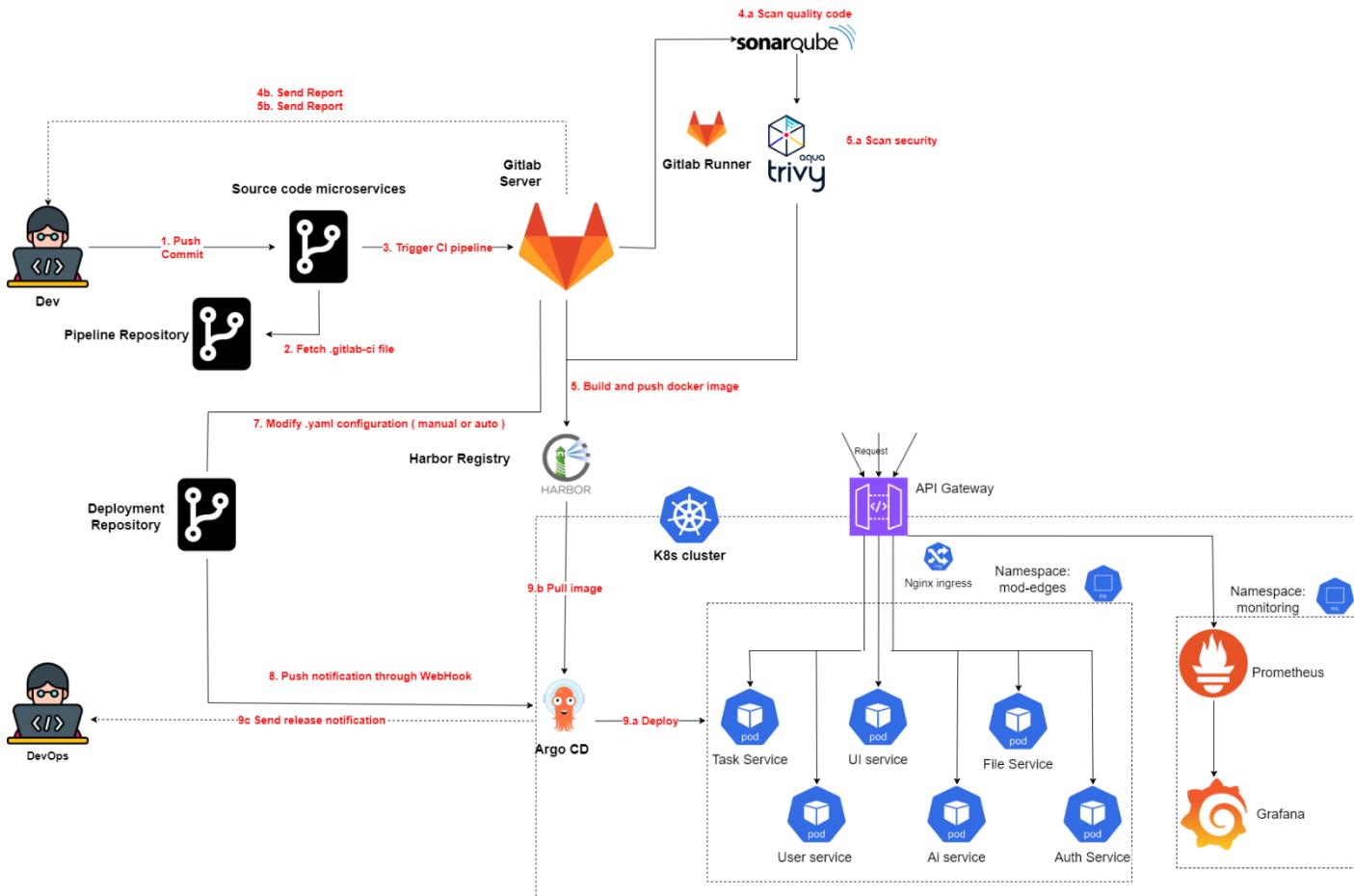
Grafana là một nền tảng để xây dựng các analytics và monitoring hay nói cách khác grafana là một vizualizer hiển thị các metric dưới dạng các biểu đồ hoặc đồ thị, được tập hợp lại thành một dashboard có tính tùy biến cao hỗ trợ theo dõi tình trạng của node một cách dễ dàng hơn. Đây là một nền tảng mã nguồn mở có tính ứng dụng cao trên bất kỳ lĩnh vực nào có thể thu thập dữ liệu theo dòng thời gian [10].

Vai trò của Grafana trong quản lí K8S:

- Trực quan hóa các dữ liệu của Prometheus.
- Giám sát hiệu suất thu thập dữ liệu từ nhiều nguồn khác nhau trong cụm Kubernetes, bao gồm:
 - + Metrics: CPU, RAM, ổ đĩa, mạng, v.v.
 - + Logs: Kubernetes logs, application logs, v.v.
 - + Events: Kubernetes events, pod events, node event,...
- Phân tích sự cố giúp xác định nguyên nhân của các vấn đề.

Chương 3. Triển khai hệ thống

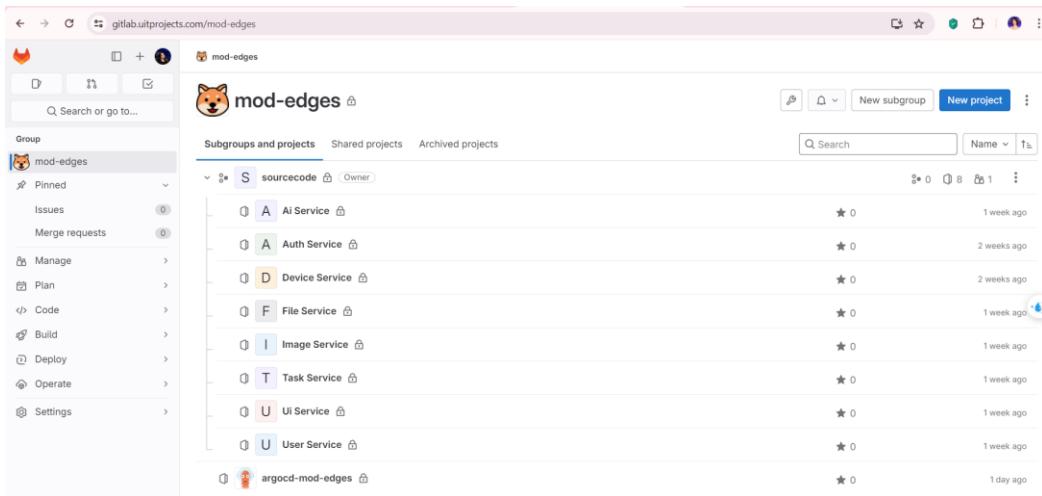
3.1. SƠ ĐỒ HỆ THỐNG



Hình 3.1.1: Sơ đồ hệ thống.

3.2. SOURCECODE MICROSERVICE

- Group trong đó bên trong chứa từng repo nhỏ, mỗi repo sẽ tương ứng với 1 dự án microservice.



Hình 3.2.1: Repo chứa tất cả dự án microservice.

- Mỗi Repo trong đây đã được cấu hình quy trình CI bao gồm quá trình scan code, và build và push vào private registry.

3.3. DEPLOYMENT REPOSITORY

- Repo tập chung chứa các file cấu hình liên quan đến việc triển khai trên Kubernetes.

Name	Last commit	Last update
dev	config(k8s): update	1 day ago
.gitignore	change Loadbalancer to nodeport 2	3 days ago
README.md	Initial commit	1 month ago
application.yaml	config(argocd): update K8s configuration file	2 weeks ago
argocd-installation.sh	config(k8s): update	1 day ago

Hình 3.3.1: Repo chứa file cấu hình Kubernetes của tất cả microservices.

3.4. GITLAB

- Nơi lưu trữ sourcecode và tích hợp quy trình Devops. Để hỗ trợ cho quy trình CI của dự án. Gitlab được cấu hình 1 runner với 2 chế độ như sau.

The screenshot shows the 'Runners' page in the GitLab interface. At the top, there are filters for 'All' (2), 'Group' (2), and 'Project' (0). A search bar and a 'Created date' dropdown are also present. A toggle switch for 'Show only inherited' is turned on. Below this, a summary shows 2 Online, 0 Offline, and 0 Stale runners.

Status	Runner	Owner	Actions
Online	#44 (CKLqmLiB) Group Version 16.10.0 Last contact: 43 minutes ago IP 192.168.1.202 8 jobs Created by Ngô Vũ Minh Đạt 2 weeks ago	mod-edges	Edit Pause Delete
Online	#42 (-Ym4WzNFR) Group Version 16.10.0 Last contact: 58 minutes ago IP 192.168.1.202 68 jobs Created by Ngô Vũ Minh Đạt 3 weeks ago	mod-edges	Edit Pause Delete

Hình 3.4.1: Gitlab runner.

- Shell: chạy câu lệnh trực tiếp như đang tương tác với terminal
- Docker: chạy các câu lệnh trong môi trường container cụ thể, được sử dụng với Sonarqube để check chất lượng code.
- Quá trình CI với scan code sẽ chạy với executor là Docker để cô lập môi trường và tránh bị conflic với các thư viện của Sonarqube và thư viện sẵn có trên host chạy runner.
- Sử dụng shell trong quá trình Docker build, Docker push để tận dụng cache layer của Docker, tránh mất thời gian vì phải chạy lại từ đầu cho những lần chạy CI tiếp theo.

3.5. PIPELINE REPOSITORY

The screenshot shows the 'Pipeline' repository page in the GitLab interface. The sidebar includes 'Project' (pipeline), 'Issues' (0), 'Merge requests' (0), 'CI/CD settings', 'Manage', 'Plan', 'Code', 'Build', 'Secure', 'Deploy', 'Operate', 'Monitor', 'Analyze', and 'Settings'. The main area shows a summary for the 'pipeline' project: 5 Commits, 1 Branch, 0 Tags, 7 KiB Project Storage. A modal window titled 'Auto DevOps' is open, stating it will automatically build, test, and deploy your application based on a predefined CI/CD configuration. It includes a 'Enable in settings' button. Below the modal, the pipeline details are shown: config(): delete 1 (authored 2 hours ago). The pipeline has a main branch with several stages: README, Add LICENSE, Add CHANGELOG, Add CONTRIBUTING, Add Kubernetes cluster, Set up CI/CD, and Add Wiki. A 'Configure Integrations' section is also visible. A table lists files with their last commit and update times:

Name	Last commit	Last update
README.md	Initial commit	3 hours ago
general-pipeline.yml	config(): delete 1	2 hours ago
README.md		

Hình 3.5.1: ví dụ về Pipeline Repository

- Repo chưa tập trung tất cả file liên quan tới pipeline, hạn chế can thiệp file cấu hình từ bên Dev, và có các Project khác có thể sử dụng chung một file CI mà không cần phải cấu hình tay trên từng project microservice khác.
- Các Project muốn sử dụng thì cần cấu hình include file pipeline này

mod-edges / sourcecode / User Service / CI/CD Settings

General pipelines

Customize your pipeline configuration.

Public pipelines
Allow public access to pipelines and job details, including output logs and artifacts. [?](#)

Auto-cancel redundant pipelines
Pipelines for new changes cause older pending or running pipelines on the same branch to be cancelled. [?](#)

Prevent outdated deployment jobs
When a deployment job is successful, prevent older deployment jobs that are still pending. [?](#)

Allow job retries for rollback deployments
Allow job retries even if the deployment job is outdated. [?](#)

Use separate caches for protected branches
Unprotected branches will not have access to the cache from protected branches. [?](#)

CI/CD configuration file

`general-pipeline.yml@mod-edges/pipeline`

The name of the CI/CD configuration file. A path relative to the root directory is optional (for example `my/path/.myfile.yml`). [?](#)

Git strategy

Choose which Git strategy to use when fetching the project. [?](#)

git clone
For each job, clone the repository.

git fetch
For each job, re-use the project workspace. If the workspace doesn't exist, use `git clone`.

Git shallow clone

20

The number of changes to fetch from GitLab when cloning a repository. Lower values can speed up pipeline execution. Set to `0` or blank to fetch all branches and tags for each job. [?](#)

Hình 3.5.2: Cài đặt mẫu của User Service.

3.6. PRIVATE REGISTRY

Để có thể lưu trữ các Docker image một cách riêng tư. Nhóm sử dụng Opensource Harbor để dựng ra một nơi lưu trữ các bản image.

The screenshot shows the Harbor Registry interface. At the top, there's a header bar with a search bar, language selection (English), and user account (admin). Below the header, the main navigation menu includes 'Projects', 'Logs', 'Administration', 'Users', 'Robot Accounts', 'Registries', 'Replications', 'Distributions', 'Labels', 'Project Quotas', 'Interrogation Services', 'Clean Up', 'Job Service Dashboard', and 'Configuration'. A 'LIGHT' button is also present. The main content area displays the 'mod-edges' project details, which are private and have used 8.90GiB of unlimited quota. The 'Repositories' tab is selected, showing a table with columns: Name, Artifacts, Pulls, and Last Modified Time. The table lists several repositories: mod-edges/ui-service, mod-edges/image-service, mod-edges/task-service, mod-edges/file-service, mod-edges/device-service, mod-edges/auth-service, mod-edges/ai-service, and mod-edges/user-service. Each row shows the number of artifacts, pull requests, and the last modified time.

Name	Artifacts	Pulls	Last Modified Time
mod-edges/ui-service	1	10	5/29/24, 12:54 AM
mod-edges/image-service	2	2	5/13/24, 3:07 PM
mod-edges/task-service	2	11	5/29/24, 12:54 AM
mod-edges/file-service	2	9	5/29/24, 12:54 AM
mod-edges/device-service	1	0	5/13/24, 12:27 PM
mod-edges/auth-service	2	10	5/29/24, 12:54 AM
mod-edges/ai-service	2	12	5/29/24, 12:54 AM
mod-edges/user-service	1	10	5/29/24, 12:54 AM

Hình 3.6.1: Registry Harbor.

3.7. ARGO CD VÀ KUBERNETS CLUSTER

Đối với quy trình triển khai, Nhóm sử dụng công cụ ArgoCD. Cài đặt Argocd như là một thành phần của Kubernetes cluster, sau đó cấu hình ArgoCD theo dõi Deployment Repository với thông tin Credential Gitlab phù hợp. Sau đó Argocd tự động triển khai với toàn bộ file cấu hình k8s được định bên trong folder dev.

- Với cấu hình mặc định thì Argocd sẽ check repo deployment 3p một lần. Điều này sẽ hơi lâu. Nhóm đang dự tính chuyển sang sử dụng WebHook.
- Cấu hình WebHook ở gitlab, bắn sự kiện cho Argocd, sau đó Argocd tự động check repo. Với cấu hình như này đáp ứng được nhu cầu thay đổi nhanh.

```

1 apiVersion: argoproj.io/v1alpha1
2 kind: Application
3 metadata:
4   name: mod-edges-dev
5   namespace: argocd
6 spec:
7   project: default
8
9   source:
10    directory:
11      recurse: true
12    repoURL: http://gitlab.vitprojects.com/mod-edges/argocd-mod-edges.git
13    targetRevision: HEAD
14    path: dev/
15   destination:
16     server: https://kubernetes.default.svc
17     namespace: mod-edges
18   syncPolicy:
19     syncOptions:
20       - CreateNamespace=true
21     automated:
22       prune: true
23       selfHeal: true

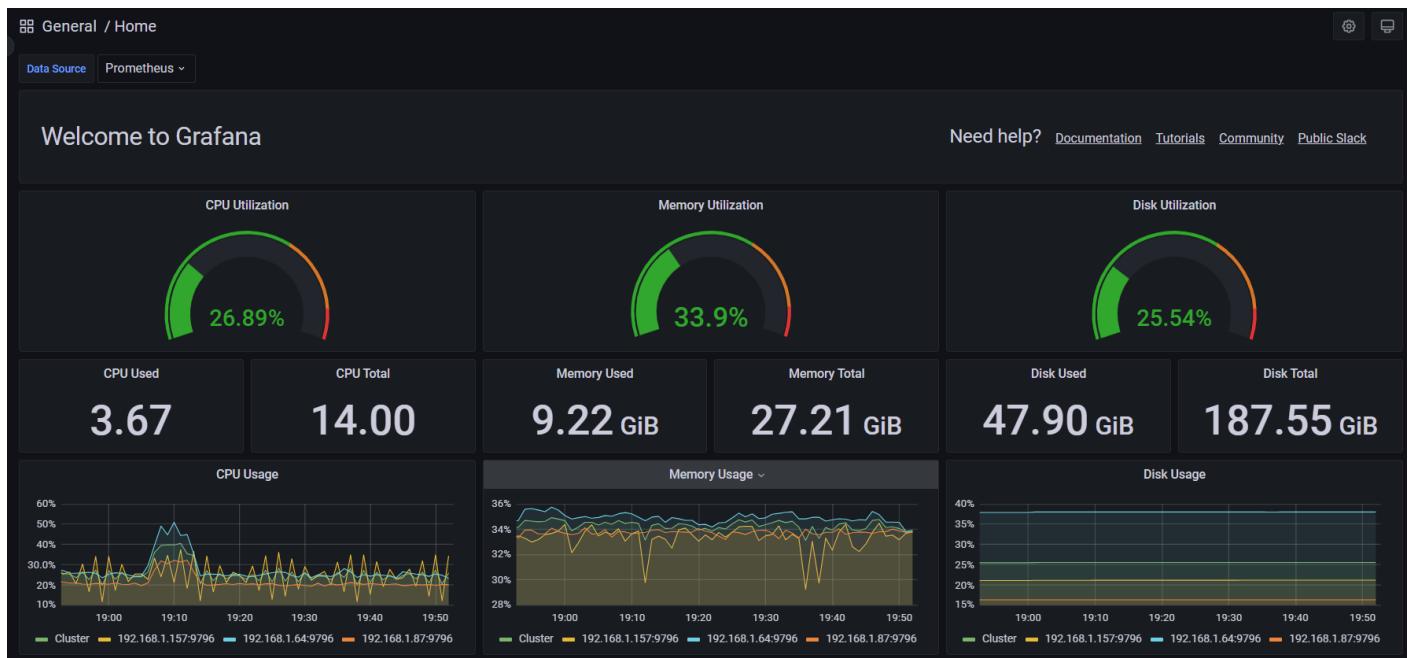
```

Hình 3.7.1: File cấu hình Argocd được cài đặt trên Cluster.

3.8. PROMETHEUS VÀ GRAFANA

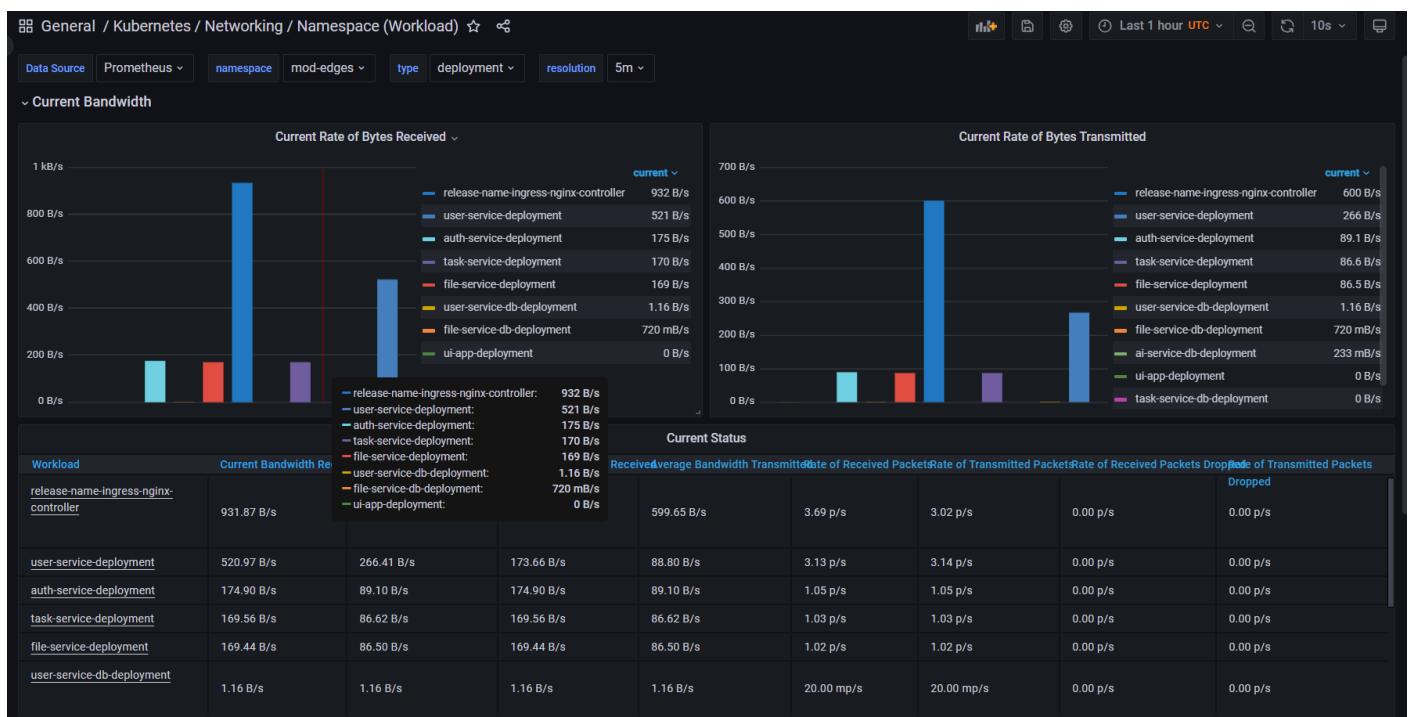
Để quản lý hệ thống một cách trực quan và luôn sẵn sàng cho những tình huống xấu nhất như việc bị tấn công mạng DDos, các dịch vụ bị lỗi, RAM hoặc CPU bị sử dụng quá mức. Cần một công cụ quản lý hạ tầng và ứng dụng nhóm chọn sử dụng Prometheus để thu thập dữ liệu và trực quan hóa dữ liệu bằng Grafana gọi chung là Monitoring.

- Một số chức năng mà nhóm sử dụng:
 - + Quản lý các tài nguyên của cluster.



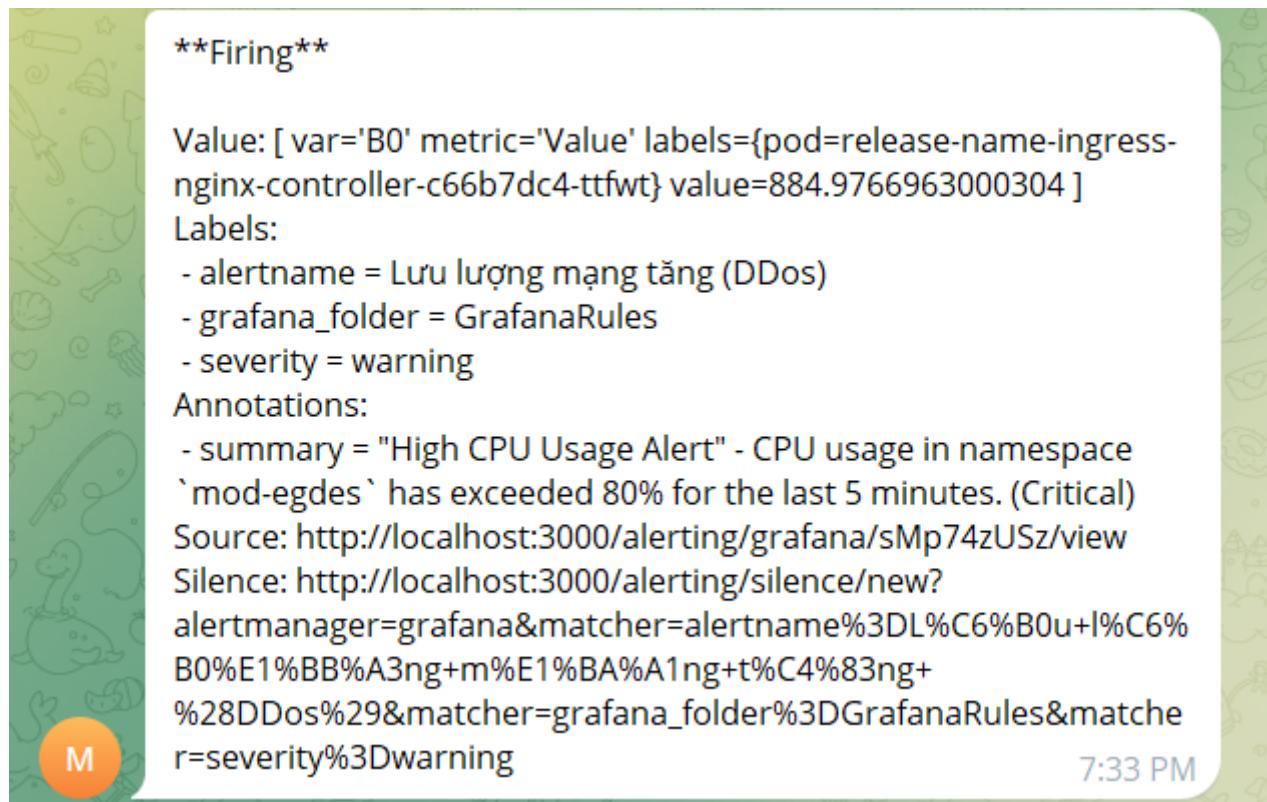
Hình 3.8.1: Giao diện quản lý tổng cluster.

- + Quản lý network của ứng dụng.



Hình 3.8.2: Giao diện quản lý network của mod-edges.

- + Cảnh báo thông qua Telegram và Slack.



Hình 3.8.3: Thông báo thông qua telegram.

alert

+ Add a bookmark

[RESOLVED] cattle-monitoring-system/rancher-monitoring-prometh Today

Grafana APP 8:01 PM

[FIRING:1] Lưu lượng mạng tăng (DDos) GrafanaRules (warning)

[FIRING:1] Lưu lượng mạng tăng (DDos) GrafanaRules (warning)

Firing

Value: [var='B0' metric='Value' labels={pod=release-name-ingress-nginx-controller-c66b7dc4-ttfwt} value=883.0153571548851]

Labels:

- alername = Lưu lượng mạng tăng (DDos)
- grafana_folder = GrafanaRules
- severity = warning

Annotations:

- summary = Úng dụng mod-edges đang bị truy cập quá mức hãy mau kiểm tra!!!!

Source: <http://localhost:3000/alerting/grafana/sMp74zUSz/view>

Silence: [http://localhost:3000/alerting/silence/new?](http://localhost:3000/alerting/silence/new?alertmanager=grafana&matcher=alername%3DL%C6%B0u+l%C6%B0%E1%BB%A3ng+m%E1%BA%A1ng+t%C4%83ng+%28DDos%29&matcher=grafana_folder%3DGrafana)

alertmanager=grafana&matcher=alername%3DL%C6%B0u+l%C6%B0%E1%BB%A3ng+m%E1%BA%A1ng+t%C4%83ng+%28DDos%29&matcher=grafana_folder%3DGrafana

aRules&matcher=severity%3Dwarning

Show less

Grafana v9.1.5 | Today at 8:01 PM

Hình 3.8.4: Thông báo thông qua Slack.

3.9. TRIỂN KHAI CÁC SERVICE TRÊN KUBERNETES

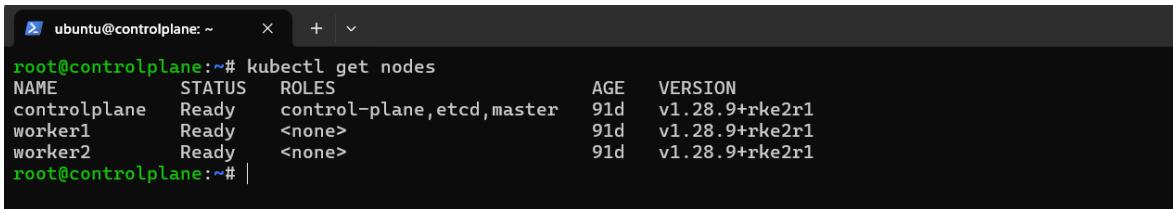
3.9.1. Triển khai mô hình cluster

- Đồ án triển khai mô hình cluster, sử dụng các dịch vụ máy ảo của Openstack.
- Tài nguyên mà đồ án sử dụng:
 - + Máy controlplane:
 - Tên: ControlPlane
 - Hệ điều hành: Ubuntu server 22.04.4 LTS
 - RAM: 16GB
 - VCPUs: 8 VCPU
 - Disk: 60GB
 - Floating IP: 192.168.120.58
 - + Máy agent 1:
 - Tên: Worker1
 - Hệ điều hành: Ubuntu server 22.04.4 LTS
 - RAM: 4GB
 - VCPUs: 2 VCPU
 - Disk: 60GB
 - Floating IP: 192.168.120.107
 - + Máy agent 2:
 - Tên: Worker2
 - Hệ điều hành: Ubuntu server 22.04.4 LTS
 - RAM: 8GB
 - VCPUs: 4 VCPU
 - Disk: 60GB
 - Floating IP: 192.168.120.158

Instance Name	Image Name	IP Address	Flavor	Key Pair	Status	Availability Zone	Task	Power State	Age	Actions
ControlPlane	Instance PreSetup	192.168.1.64, 192.168.120.58	d60.xl8	-	Active	Serv03	None	Running	2 days, 10 hours	<button>Create Snapshot</button>
Worker2	Instance PreSetup	192.168.1.157, 192.168.120.158	d60.l4	-	Active	Serv03	None	Running	2 days, 10 hours	<button>Create Snapshot</button>
Worker1	Instance PreSetup	192.168.1.87, 192.168.120.107	d60.m2	-	Active	Serv03	None	Running	2 days, 10 hours	<button>Create Snapshot</button>

Hình 3.9.1: Số lượng máy ảo sử dụng.

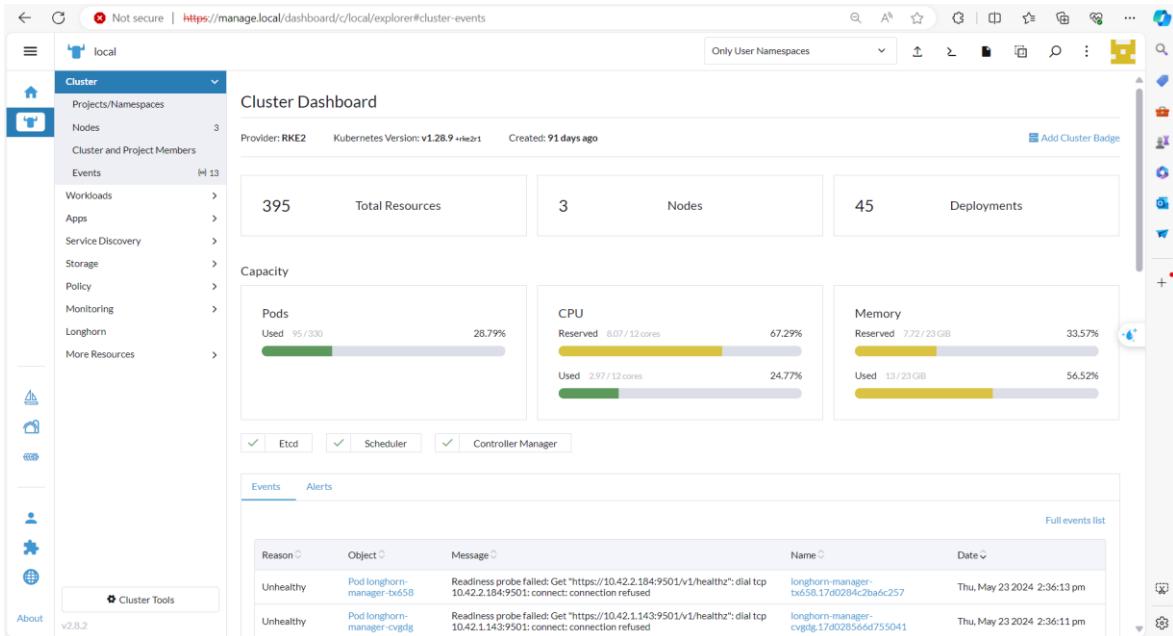
- Sau khi chuẩn bị tài nguyên và môi trường, kết nối các tài nguyên trên thành một cluster kubernetes với RKE2.



```
ubuntu@controlplane: ~
root@controlplane:~# kubectl get nodes
NAME      STATUS   ROLES      AGE      VERSION
controlplane   Ready    control-plane,etcfd,master   91d     v1.28.9+rke2r1
worker1      Ready    <none>     91d     v1.28.9+rke2r1
worker2      Ready    <none>     91d     v1.28.9+rke2r1
root@controlplane:~# |
```

Hình 3.9.2: Các kubernetes nodes sau khi triển khai.

- Sử dụng kubernetes rancher web dashboard.



Cluster Dashboard

Provider: RKE2 Kubernetes Version: v1.28.9+rke2r1 Created: 91 days ago

Events [13]

Reason	Object	Message	Name	Date
Unhealthy	Pod longhorn-manager-bx658	Readiness probe failed: Get "https://10.42.2.184:9501/v1/healthz": dial tcp 10.42.2.184:9501: connect: connection refused	longhorn-manager-tx658.17d0284c2ba6c257	Thu, May 23 2024 2:36:13 pm
Unhealthy	Pod longhorn-manager-cvgdg	Readiness probe failed: Get "https://10.42.1.143:9501/v1/healthz": dial tcp 10.42.1.143:9501: connect: connection refused	longhorn-manager-cvgdg.17d028566d755041	Thu, May 23 2024 2:36:11 pm

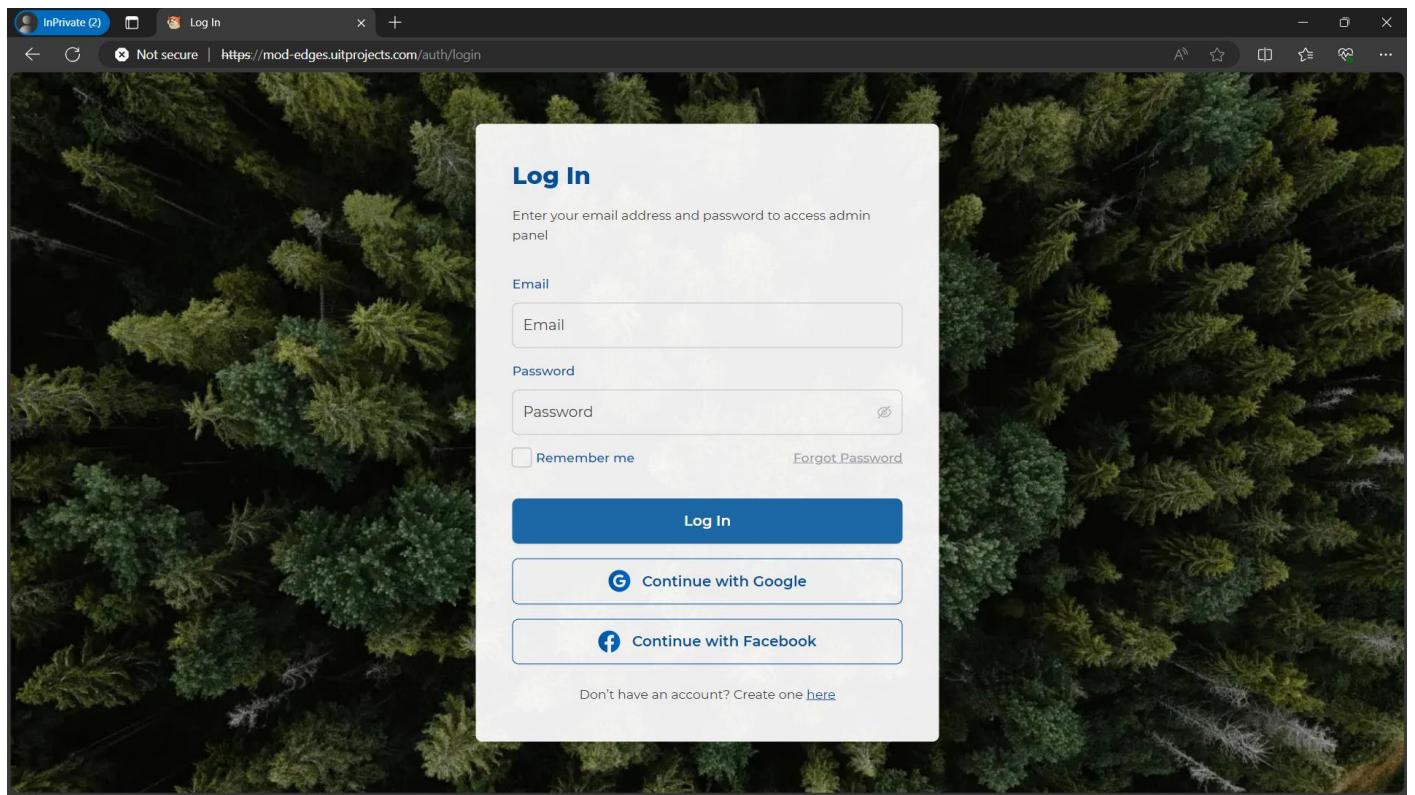
Hình 3.9.3: Rancher dashboard.

3.9.2. Triển khai ứng dụng lên kubernetes

- Sau khi triển khai ứng dụng

State	Name	Image	Ready	Restarts	IP	Node	Age
Namespace: mod-edges							
Running	ai-service-db-deployment-5b6684c65c-vdrh5	postgres	1/1	0	10.42.0.94	controlplane	21 hours
Running	ai-service-deployment-7d8b665fc-r4jxv	registry-harbor.ngodat0103.me/mod-edges/ai-service:1.0.0	1/1	0	10.42.0.91	controlplane	21 hours
Running	auth-service-deployment-5d7cdf6694-qcc5c	registry-harbor.ngodat0103.me/mod-edges/auth-service:1.0.0	1/1	0	10.42.0.95	controlplane	21 hours
Running	file-service-db-deployment-f655ffbf-cpqlz	postgres	1/1	0	10.42.0.93	controlplane	21 hours
Running	file-service-deployment-5c799687f9-vsscm	registry-harbor.ngodat0103.me/mod-edges/file-service:1.0.0	1/1	0	10.42.0.92	controlplane	21 hours
Completed	release-name-ingress-nginx-admission-create-6zxwl	registry.k8s.io/ingress-nginx/kube-webhook-certgen:v1.4.0@sha256:44d1d0e9...	0/1	0	10.42.0.89	controlplane	21 hours
Running	release-name-ingress-nginx-controller-c66b7dc4-zctk6	registry.k8s.io/ingress-nginx/controller:v1.10.0@sha256:42b3f0e5...	1/1	0	10.42.1.23	worker2	21 hours
Running	task-service-db-deployment-8546b68f79-sshsv	postgres	1/1	0	10.42.0.99	controlplane	21 hours
Running	task-service-deployment-b88fd7d69-xvsfd	registry-harbor.ngodat0103.me/mod-edges/task-service:1.0.0	1/1	0	10.42.0.96	controlplane	21 hours
Running	ui-app-deployment-5b855bdf5f-xtlc5	registry-harbor.ngodat0103.me/mod-edges/ui-service:1.0.0	1/1	0	10.42.0.97	controlplane	21 hours
Running	user-service-db-deployment-56d6bc8f5f-jzdz9	postgres	1/1	0	10.42.0.98	controlplane	21 hours
Running	user-service-deployment-dfd98d8f4-5fnhh	registry-harbor.ngodat0103.me/mod-edges/user-service:1.0.0	1/1	0	10.42.0.90	controlplane	21 hours
Running	user-service-deployment-dfd98d8f4-pl5gs	registry-harbor.ngodat0103.me/mod-edges/user-service:1.0.0	1/1	3 (21h ago)	10.42.2.20	worker1	21 hours
Running	user-service-deployment-dfd98d8f4-wp6vs	registry-harbor.ngodat0103.me/mod-edges/user-service:1.0.0	1/1	2 (21h ago)	10.42.1.22	worker2	21 hours

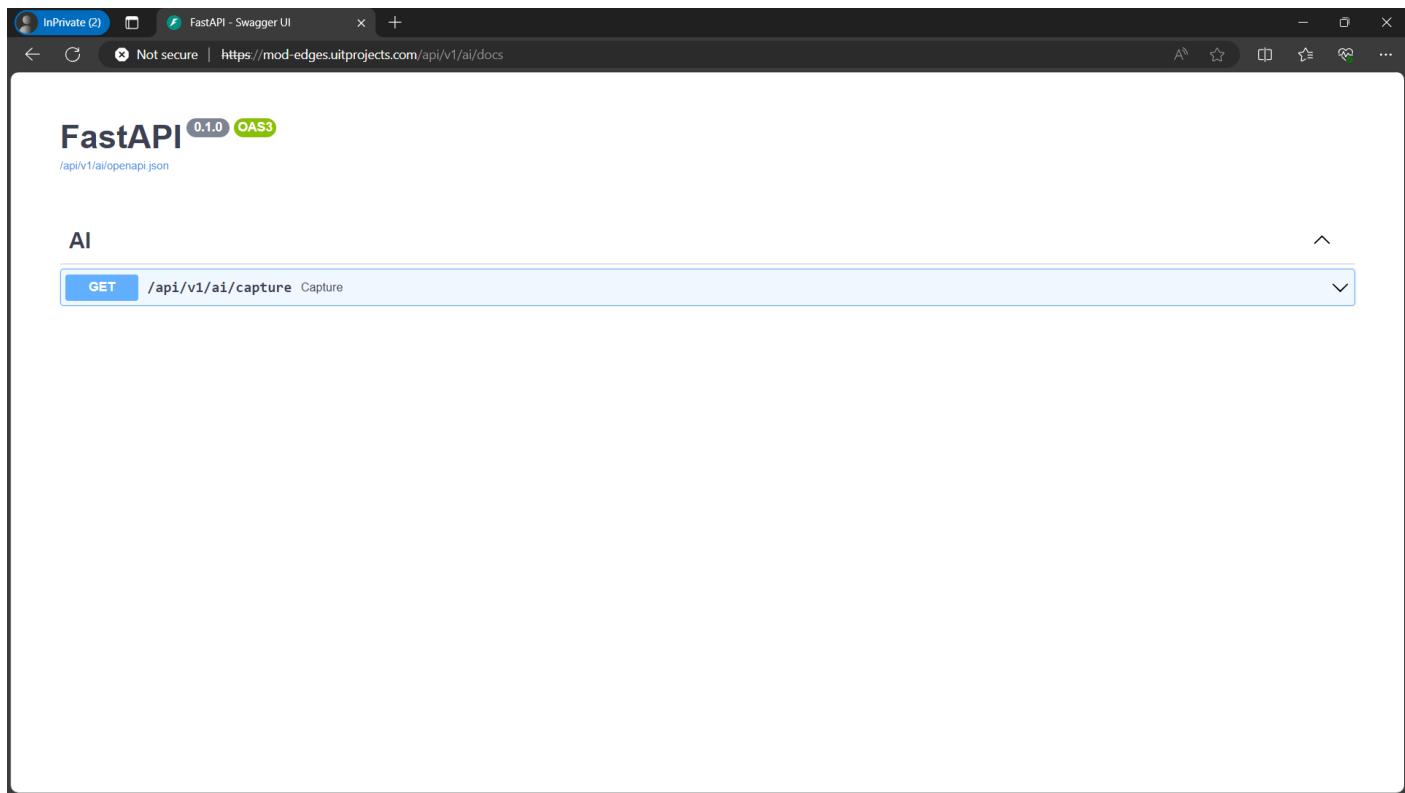
Hình 3.9.4: Số lượng pod mà đã án sử dụng.



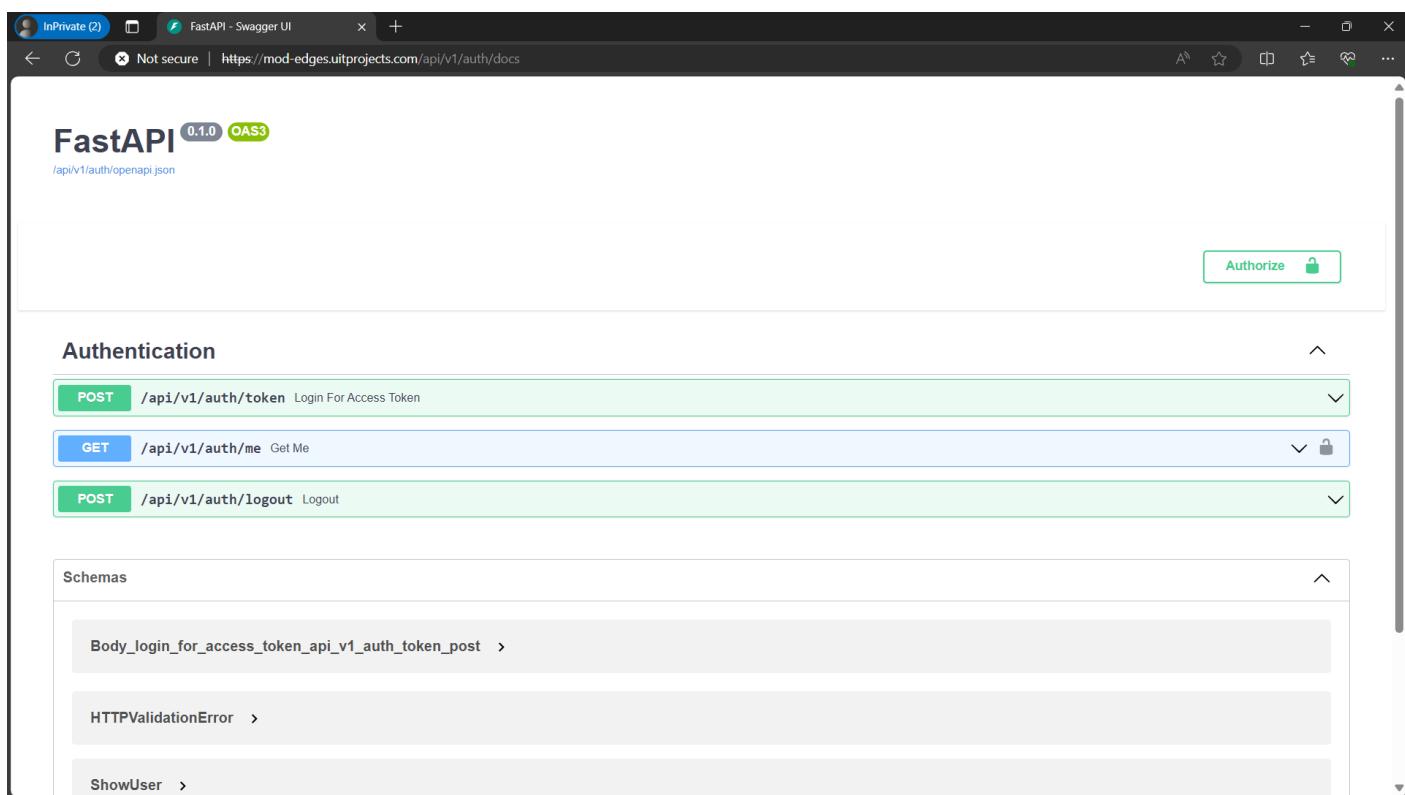
Hình 3.9.5: UI login.

A screenshot of a web-based application interface titled "MOD Edge". On the left, there is a sidebar with icons for "Users", "Files", "Device", and "Apps". The main content area has a search bar at the top. Below it, there are two sections: "Personal Info" and "Contact Info". The "Personal Info" section contains fields for "First Name" (with placeholder "string"), "Last Name" (with placeholder "string"), "Nickname" (with placeholder "string"), "Sex" (with a dropdown menu showing "Female" selected), "Birthday" (with placeholder "06/07/2024" and a calendar icon), and "Language" (with a dropdown menu). The "Contact Info" section contains fields for "Phone" (empty) and "Email" (with placeholder "user@example.com").

Hình 3.9.6: Trang chính của ứng dụng.



Hình 3.9.7: AI service.



Hình 3.9.8: Authentication service.

The screenshot shows the FastAPI - Swagger UI interface for the User service. At the top, it displays "FastAPI 0.1.0 OAS3" and the URL "Not secure | https://mod-edges.ultratechprojects.com/api/v1/user/docs". A green "Authorize" button is located in the top right corner. The main content area is organized into sections: "User" and "Role".

User

- POST** /api/v1/user/all Get a list of all users
- POST** /api/v1/user/signup Create User
- GET** /api/v1/user/me Get details of currently logged in user
- DELETE** /api/v1/user/{username} Delete User
- POST** /api/v1/user/add-role Add Role

Role

- GET** /api/v1/role/all Get a list of all roles
- POST** /api/v1/role/create-role Create Role

Hình 3.9.9: User service.

The screenshot shows the FastAPI - Swagger UI interface for the Task service. At the top, it displays "FastAPI 0.1.0 OAS3" and the URL "Not secure | https://mod-edges.ultratechprojects.com/api/v1/task/docs". A green "Authorize" button is located in the top right corner. The main content area is organized into a single section: "Task".

Task

- POST** /api/v1/task Get All Tasks
- POST** /api/v1/task/create-task Create New Task
- POST** /api/v1/task/service/{service} Get All Tasks By Service
- POST** /api/v1/task/update/{id} Update Task
- POST** /api/v1/task/change-status/{id} Change Task Status
- DELETE** /api/v1/task/delete/{id} Delete Task

Schemas

Hình 3.9.10: Task service.

The screenshot shows the FastAPI Swagger UI interface. At the top, it displays "FastAPI 0.1.0 OAS3" and the URL "Not secure | https://mod-edges.uitprojects.com/api/v1/file/docs". On the right side, there is an "Authorize" button with a lock icon. Below the header, the word "File" is highlighted in bold. A list of API endpoints is shown under the "File" category:

- POST /api/v1/file/upload** Upload File
- GET /api/v1/file/{file_id}** View File
- DELETE /api/v1/file/{file_id}** Delete File
- POST /api/v1/file/all** Get All Files
- POST /api/v1/file/my-files** Get My Files
- GET /api/v1/file/{file_id}/detail** Get File Detail
- GET /api/v1/file/{file_id}/download** Download File

At the bottom left, there is a "Schemas" section. On the right side of the interface, there are collapse/expand arrows and a scroll bar.

Hình 3.9.11: File service.

Phụ lục

PHỤ LỤC 1. Link demo 52

PHỤ LỤC 1. Link demo

<https://drive.google.com/drive/folders/1SckzC1HGs78unueHxmfAbmtKDf3eR4Cc?usp=sharing>

Tài liệu tham khảo

- [1] "Edge Computing là gì? 5 thành phần cơ bản trong hệ sinh thái Edge computing," viettelidc, 23 02 2022. [Online]. Available: <https://www.viettelidc.com.vn/tin-tuc/edge-computing-la-gi-5-thanh-phan-co-ban-trong-he-sinh-thai-edge-computing-3071>. [Accessed 11 6 2024].
- [2] J. Carnell and L. H. SanChez, Spring Microservice In Action, 2nd ed., NEW YORK: Manning, 2021, pp. 3-5.
- [3] J. Carnell and L. H. SanChez, Spring Microservice In Action, 2nd ed., Manning, 2021, pp. 5-6.
- [4] "Giới thiệu tổng quan về kiến trúc Docker," TNHH Giải pháp Viễn thông TEL4VN, 1 1 2021. [Online]. Available: <https://tel4vn.edu.vn/blog/gioi-thieu-tong-quan-ve-kien-truc-cua-docker/>. [Accessed 11 6 2024].
- [5] "Docker Reference | Docker Docs," Docker, Inc, [Online]. Available: <https://docs.docker.com/reference/dockerfile/>. [Accessed 6 11 2024].
- [6] M. Luksa, Kubernetes In Action, 1st ed., New York: Manning, 2018, pp. 56-57.
- [7] M. Luksa, Kubernetes In Action, 1st ed., New York: Manning, 2018, p. 44.
- [8] M. Luksa, Kubernetes In Action, 1st ed., New York: Manning, 2018, pp. 120-149.

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ
THÔNG TIN**

Độc Lập - Tự Do - Hạnh Phúc

TÊN ĐỀ TÀI:

- **Tiếng việt:** Xây dựng quy trình triển khai tự động dựa trên kiến trúc microservice
- **Tiếng anh:** Building a deployment process based on microservice architecture

Cán bộ hướng dẫn:

Thời gian thực hiện: Từ ngày 3/1/2024 đến ngày 4/7/2024

Sinh viên thực hiện:

Ngô Vũ Minh Đạt – 21521935

Nguyễn Đoàn Khắc Huy – 21522151

Nội dung đề tài:

- Đề tài xây dựng quy trình triển khai tự động dựa trên kiến trúc microservice
- Triển khai quy trình Devops CI/CD
- Kết quả mong muốn đạt được:
 - Tích hợp quy trình DevOps theo từng microservice
 - Triển khai tự động triển khai trên các môi trường dev, staging, production
 - Triển khai tự động cập nhật phần mềm
 - Lưu trữ và quản lý dự án với Git

Kế hoạch thực hiện

1. Giai đoạn 1 (01/2024-02/2024) : Tìm hiểu các kiến trúc microservice, thử nghiệm triển khai một vài chức năng cơ bản của microservices.
2. Giai đoạn 2 (02/2024-03/2024) : Triển khai hạ tầng sử dụng OpenStack
3. Giai đoạn 3 (03/2024-04/2024): Triển khai quy trình CI cho từng Microservice
4. Giai đoạn 4 (4/2024-5/2024): Tích hợp quy trình CD sử dụng ArgoCD, mở rộng thành quy trình DevSecOps
5. Giai đoạn 5: (5/2024-6/2024): Viết báo cáo

Xác nhận của CBHD (Ký tên và ghi rõ họ tên)	TP. HCM, ngày....thángnăm..... Sinh viên (Ký tên và ghi rõ họ tên)
-----------------------------------------------------------	------------------------------------------------------------------------------------------------------