

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

LƯU MINH THIỆN – 21521460  
NGUYỄN THÀNH ĐĂNG – 21520683  
QUÁCH THỊ HOÀI PHƯƠNG – 21520409



**ĐỒ ÁN MÔN HỌC**  
**GIÁM SÁT HẠ TẦNG CLOUD SỬ DỤNG PROMETHEUS & LOKI KẾT**  
**HỢP GRAFANA & ICINGA CHO ỨNG DỤNG MICROSERVICES**  
**TRÊN K3S**

**MÔN: ĐÁNH GIÁ HIỆU NĂNG HỆ THỐNG MẠNG MÁY TÍNH**  
**LỚP: NT531.P11**

**GIẢNG VIÊN HƯỚNG DẪN**  
**ĐẶNG LÊ BẢO CHƯƠNG**

**TP. HỒ CHÍ MINH, 2024**

## MỤC LỤC

Chương 1. GIỚI THIỆU.....	12
1.1. Lý do chọn đề tài .....	12
1.2. Phạm vi nghiên cứu .....	12
1.3. Tóm tắt đồ án.....	13
Chương 2. CƠ SỞ LÝ THUYẾT.....	13
2.1. Monitoring tools.....	13
2.1.1. Prometheus .....	13
2.1.2. Grafana Loki.....	16
2.1.3. Grafana .....	19
2.1.4. Icinga.....	20
2.2. K3s.....	24
2.3. Microservices .....	27
2.4. AWS.....	29
2.5. Infrastructure as Code (IaC) .....	30
2.5.1. Terraform.....	30
2.5.2. Ansible .....	31
Chương 3. HIỆN THỰC ĐỀ TÀI.....	32
3.1. Mô hình mạng.....	32
3.2. Ứng dụng microservices.....	34
3.3. Triển khai cơ sở hạ tầng và ứng dụng .....	36
3.3.1. Triển khai cơ sở hạ tầng .....	36
3.3.2. Triển khai ứng dụng.....	41
3.4. Cấu hình giám sát cho hệ thống.....	43

3.4.1.	Prometheus .....	43
3.4.1.1.	Prometheus trên monitoring instance.....	43
3.4.1.2.	Prometheus trên cụm K3s.....	48
3.4.2.	Grafana Loki.....	51
3.4.2.1.	Grafana Loki tên monitoring instance .....	51
3.4.2.2.	Grafana Loki trên cụm K3s.....	56
3.4.3.	Grafana .....	59
3.4.4.	Icinga2 .....	72
3.5.	Triển khai các kịch bản kiểm thử .....	81
3.5.1.	Đánh giá hiệu suất của hệ thống phân phối tải .....	81
3.5.2.	Đánh giá tài nguyên của cụm K3s.....	84
3.5.3.	Đánh giá hệ thống cảnh báo.....	87
3.5.3.1.	Kiểm tra cảnh báo khi tài nguyên bị quá tải .....	87
3.5.3.2.	Kiểm tra cảnh báo khi dịch vụ hoặc server không khả dụng.	92
Chương 4.	KẾT LUẬN .....	96
4.1.	Kết quả đạt được.....	96
4.1.1.	So sánh.....	96
4.1.2.	Đánh giá chung .....	97
4.2.	Kết luận .....	98

## **DANH MỤC HÌNH VẼ**

Hình 1.	Prometheus.....	13
Hình 2.	Prometheus model .....	14
Hình 3.	Prometheus workflow .....	16
Hình 4.	Grafana Loki .....	16
Hình 5.	Grafana Loki model .....	17
Hình 6.	Grafana Loki workflow.....	18
Hình 7.	Grafana.....	19
Hình 8.	Icinga .....	20
Hình 9.	Kiến trúc Icinga2 .....	22
Hình 10.	Icinga workflow.....	24
Hình 11.	K3s .....	24
Hình 12.	Kiến trúc K3S.....	25
Hình 13.	Kiến trúc Microservice.....	27
Hình 14.	AWS .....	29
Hình 15.	Terraform .....	30
Hình 16.	Ansible .....	31
Hình 17.	Mô hình mạng .....	32
Hình 18.	Mô hình ứng dụng microservices .....	34
Hình 19.	Danh sách các API .....	35
Hình 20.	Triển khai hạ tầng bằng Terraform.....	36
Hình 21.	Kiểm tra trạng thái các instance trên AWS console.....	36
Hình 22.	Thông tin resource map của VPC .....	36
Hình 23.	Kết nối VPN đến hạ tầng để cấu hình .....	37
Hình 24.	File inventory.....	37
Hình 25.	Cấu hình cho các server bằng Ansible .....	38
Hình 26.	Mapping domain bằng Cloudflare .....	39
Hình 27.	Cấu hình HA Proxy.....	40

Hình 28.	Thông tin cấu hình để kết nối đến cụm K3s .....	40
Hình 29.	Kiểm tra thông tin các node của cụm K3s .....	40
Hình 30.	Lấy thông tin port của Traefik thông qua service.....	40
Hình 31.	Build các service thành các image .....	41
Hình 32.	Docker Hub chứa các image của ứng dụng.....	41
Hình 33.	Các file manifest triển khai lên cụm K3s.....	42
Hình 34.	Nội dung file kustomization.yml định nghĩa các resource sẽ triển khai	
	42	
Hình 35.	Kiểm tra thông tin resource của ứng dụng sau khi triển khai thành công	43
Hình 36.	Thư mục cấu hình của Prometheus .....	43
Hình 37.	Cấu hình scrape của Prometheus.....	44
Hình 38.	Cấu hình web của Prometheus.....	44
Hình 39.	Tệp systemd của Prometheus .....	45
Hình 40.	Cấu trúc thư mục lưu trữ của Prometheus .....	45
Hình 41.	Thư mục chứa các file liên quan của mỗi data block.....	46
Hình 42.	File chunk của data block .....	46
Hình 43.	WAL (Write-Ahead Log) .....	47
Hình 44.	Tệp systemd của Node Exporter .....	47
Hình 45.	Kiểm tra trạng thái Node Exporter .....	48
Hình 46.	Kiểm tra trạng thái các target trên Prometheus web UI .....	48
Hình 47.	Nội dung file values.yml của Prometheus .....	49
Hình 48.	Nội dung file ingress.yml của Prometheus .....	50
Hình 49.	Kiểm tra thông tin các resource của Prometheus sau khi triển khai thành công	50
Hình 50.	Kiểm tra trạng thái các target trên Prometheus web UI của cụm K3s	
	51	
Hình 51.	Nội dung file cấu hình của Loki.....	52
Hình 52.	Cấu trúc thư mục lưu trữ của Loki .....	54

Hình 53.	Kiểm tra trạng thái sẵn sàng của Loki.....	54
Hình 54.	Cấu hình của Promtail.....	55
Hình 55.	File lưu trạng thái vị trí đọc cuối cùng của tệp log .....	55
Hình 56.	Kiểm tra trạng thái Promtail .....	56
Hình 57.	Nội dung file values.yml của Loki .....	57
Hình 58.	Nội dung file ingress.yml của Loki.....	58
Hình 59.	Nội dung file values.yml của Promtail .....	58
Hình 60.	Kiểm tra thông tin các resource của Loki sau khi triển khai thành công	
	59	
Hình 61.	Đường dẫn thư mục chứa file cấu hình Grafana .....	59
Hình 62.	Kiểm tra trạng thái Grafana.....	59
Hình 63.	Màn hình đăng nhập của Grafana .....	60
Hình 64.	Thông tin các data source.....	60
Hình 65.	Thêm data source.....	60
Hình 66.	Chọn kiểu data source.....	61
Hình 67.	Nhập thông tin cho data source.....	61
Hình 68.	Kiểm tra kết nối đến data source .....	62
Hình 69.	Cấu hình xác thực cho data source.....	62
Hình 70.	Thêm dashboard.....	62
Hình 71.	Nhập URL, ID hoặc JSON model của dashboard .....	63
Hình 72.	Nhập thông tin và chọn data source tương ứng.....	63
Hình 73.	Dashboard sau khi được import.....	64
Hình 74.	Dashboard cho Prometheus .....	64
Hình 75.	Dashboard cho Prometheus trên cụm K3s .....	65
Hình 76.	Dashboard cho Loki .....	65
Hình 77.	Dashboard cho Loki trên cụm K3s .....	66
Hình 78.	Tạo Telegram bot.....	67
Hình 79.	Tạo nhóm chat Telegram để nhận thông báo từ bot .....	67
Hình 80.	Xem chat ID thông qua Telegram web.....	68

Hình 81. Tạo contact point.....	68
Hình 82. Nhập thông tin cho contact point.....	68
Hình 83. Tạo rule cảnh báo.....	69
Hình 84. Định nghĩa truy vấn và điều kiện cảnh báo .....	69
Hình 85. Thiết lập hành vi đánh giá .....	70
Hình 86. Cấu hình các nhãn và thông báo .....	71
Hình 87. Cấu hình tin nhắn thông báo .....	71
Hình 88. Kết quả sau khi tạo tất cả rule cảnh báo .....	72
Hình 89. Cấu hình Influxdb.....	72
Hình 90. Màn hình dashboard của Icinga.....	73
Hình 91. Cấu hình hosts và endpoints của ứng dụng.....	74
Hình 92. Plugin từ thư viện của Nagios.....	74
Hình 93. Giám sát dịch vụ API .....	75
Hình 94. Giám sát mạng .....	75
Hình 95. Giám sát hệ thống.....	76
Hình 96. Giám sát dịch vụ web.....	76
Hình 97. Định nghĩa các Service Groups.....	77
Hình 98. Giám sát Kubernetes.....	77
Hình 99. Cấu hình host Telegram .....	78
Hình 100. Cấu hình service Telegram .....	78
Hình 101. Kết quả cấu hình thông báo qua kênh Telegram .....	79
Hình 102. Theo dõi các dịch vụ hệ thống.....	79
Hình 103. Theo dõi dịch vụ trong K3S (1) .....	80
Hình 104. Theo dõi dịch vụ trong K3S (2) .....	80
Hình 105. Mô phỏng lượng lớn người dùng truy cập .....	81
Hình 106. Kiểm tra số lượng traffic ở giao diện HA Proxy stats .....	82
Hình 107. Theo dõi log thu được của Loki trong kịch bản 1.....	82
Hình 108. Theo dõi metric thu được của Prometheus trong kịch bản 1 .....	82
Hình 109. Số lượng yêu cầu qua HA Proxy đến các worker node .....	83

Hình 110.	Khả năng cân bằng tải của HA Proxy gần như bằng nhau .....	83
Hình 111.	Kết quả thu thập log của Loki trong kịch bản 1 .....	83
Hình 112.	Kết quả thu thập metric của Prometheus trong kịch bản 1 .....	84
Hình 113.	Kiểm tra quá trình tạo pod bằng kubectl .....	85
Hình 114.	Theo dõi log thu được của Loki trong kịch bản 2 .....	85
Hình 115.	Theo dõi metric thu được của Prometheus trong kịch bản 2 .....	85
Hình 116.	Các pod được phân phối đều trên các node .....	86
Hình 117.	Tài nguyên CPU, RAM được sử dụng để chạy các pod mới trên từng node .....	86
Hình 118.	Tạo traffic bằng Locust .....	86
Hình 119.	Kết quả thu thập log của Loki trong kịch bản 2 .....	87
Hình 120.	Kết quả thu thập metric của Prometheus trong kịch bản 2 .....	87
Hình 121.	Kiểm tra thông tin node đang chạy pod của service .....	88
Hình 122.	Biểu đồ mô phỏng của Locust .....	88
Hình 123.	Theo dõi log thu được của service .....	89
Hình 124.	Theo dõi metric thu được của node đang chạy pod của service đó .....	89
Hình 125.	Cảnh báo được gửi đến Telegram khi CPU của node vượt ngưỡng .....	90
Hình 126.	Sử dụng stress-ng để gây tải cho server được chọn .....	90
Hình 127.	Theo dõi thông số CPU, RAM .....	90
Hình 128.	Theo dõi trạng thái cảnh báo khi CPU, RAM vượt ngưỡng .....	91
Hình 129.	Cảnh báo được gửi đến Telegram khi server vượt ngưỡng CPU ..	91
Hình 130.	Cảnh báo được gửi đến Telegram khi server vượt ngưỡng RAM ..	91
Hình 131.	Trạng thái healthcheck trước khi chạy của K3s và các endpoint..	92
Hình 132.	Dừng tất cả dịch vụ trên cụm K3s .....	92
Hình 133.	Trạng thái healthcheck thay đổi khi dừng tất cả dịch vụ .....	93
Hình 134.	Cảnh báo được gửi đến Telegram về trạng thái healthcheck của dịch vụ .....	93

Hình 135.	Trạng thái của server trước khi tắt .....	94
Hình 136.	Tắt server trên AWS console .....	94
Hình 137.	Trạng thái của server được cập nhật lại.....	95
Hình 138.	Cảnh báo được gửi đến Telegram về trạng thái healthcheck của server	95

## **DANH MỤC BẢNG**

Bảng 1.	Bảng phân chia địa chỉ IP cho từng instance .....	33
Bảng 2.	Bảng so sánh chi tiết các công cụ giám sát.....	96
Bảng 3.	Bảng phân chia nhiệm vụ .....	101

## **LỜI CẢM ƠN**

Nhóm em xin gửi lời cảm ơn chân thành đến thầy Đặng Lê Bảo Chương - giảng viên bộ môn “Đánh giá hiệu năng Hệ thống Mạng máy tính” trong Khoa Mạng Máy tính và Truyền thông và Trường Đại học Công nghệ Thông tin - ĐHQG-HCM đã trang bị cho nhóm em những kiến thức, kỹ năng cơ bản cần có để hoàn thành đề tài nghiên cứu này.

Tuy nhiên trong quá trình nghiên cứu đề tài, do kiến thức chuyên ngành còn hạn chế nên nhóm vẫn còn nhiều thiếu sót khi tìm hiểu, đánh giá và trình bày về đề tài. Rất mong nhận được sự quan tâm, góp ý của thầy để đề tài của nhóm được đầy đủ và hoàn chỉnh hơn.

Xin chân thành cảm ơn.

## **Chương 1. GIỚI THIỆU**

### **1.1. Lý do chọn đề tài**

Đề tài “**Giám sát hạ tầng cloud sử dụng Prometheus & Loki kết hợp Grafana & Icinga cho ứng dụng microservices trên K3s**” được chọn để đáp ứng nhu cầu thiết yếu trong việc giám sát hệ thống microservices phức tạp trên nền tảng Kubernetes nhẹ (K3s), nhằm đảm bảo hiệu suất và độ ổn định của hệ thống. Sự kết hợp giữa Prometheus để thu thập metric, Loki để quản lý logs, Grafana để trực quan hóa và Icinga để giám sát cảnh báo tạo nên một giải pháp toàn diện, cho phép theo dõi chi tiết, cảnh báo chính xác và phân tích dữ liệu chuyên sâu. Ngoài ra, đề tài này cũng giúp so sánh các ưu, nhược điểm của từng công cụ. Với Icinga – công cụ mạnh mẽ với nhiều chức năng, từ giám sát dịch vụ và mạng lưới, tới cảnh báo, quản lý và theo dõi trạng thái hệ thống theo thời gian thực. Prometheus mạnh trong thu thập metric nhưng thiếu tính năng log. Loki chuyên về logs nhưng không lưu trữ metric. Grafana tập trung vào trực quan hóa dữ liệu nhưng phụ thuộc vào các nguồn dữ liệu bên ngoài. Những điểm khác biệt này giúp làm rõ vai trò và giới hạn của từng công cụ, cho phép chọn lựa và tối ưu hóa giải pháp giám sát phù hợp nhất cho từng loại hệ thống.

### **1.2. Phạm vi nghiên cứu**

Phạm vi nghiên cứu của đề tài tập trung vào việc phát triển một giải pháp giám sát toàn diện cho hệ thống microservices triển khai trên nền tảng K3s:

- Đánh giá các yêu cầu giám sát của hệ thống microservices, bao gồm việc thu thập metric, quản lý logs và cảnh báo.
- Triển khai và cấu hình Prometheus và Grafana để thu thập và trực quan hóa metric, Loki để quản lý logs, và Icinga để giám sát và cảnh báo.
- Thực hiện các bài kiểm tra để đánh giá hiệu suất của giải pháp giám sát, so sánh độ chính xác của cảnh báo và khả năng xử lý dữ liệu của từng công cụ.

- Nghiên cứu và phân tích các ưu, nhược điểm của từng công cụ trong việc giám sát hạ tầng cloud và ứng dụng microservices.
- Đề xuất giải pháp tối ưu hóa dựa trên kết quả phân tích và thực nghiệm.

### **1.3. Tóm tắt đồ án**

Đồ án này nhằm phát triển một giải pháp giám sát hiệu quả cho hạ tầng cloud sử dụng các công cụ phổ biến trong lĩnh vực công nghệ thông tin. Việc lựa chọn kết hợp Prometheus, Loki, Grafana và Icinga không chỉ nhằm tạo ra một hệ thống giám sát đồng bộ mà còn để so sánh hiệu quả giữa các cặp công cụ.

Mục tiêu của nghiên cứu không chỉ là cải thiện khả năng giám sát và phản ứng nhanh với các sự cố mà còn cung cấp cái nhìn sâu sắc về cách mỗi cặp công cụ hoạt động. Qua đó, đề tài sẽ giúp người dùng lựa chọn giải pháp giám sát tối ưu nhất cho từng hệ thống, đồng thời đưa ra những khuyến nghị dựa trên kết quả so sánh giữa các công cụ.

## **Chương 2. CƠ SỞ LÝ THUYẾT**

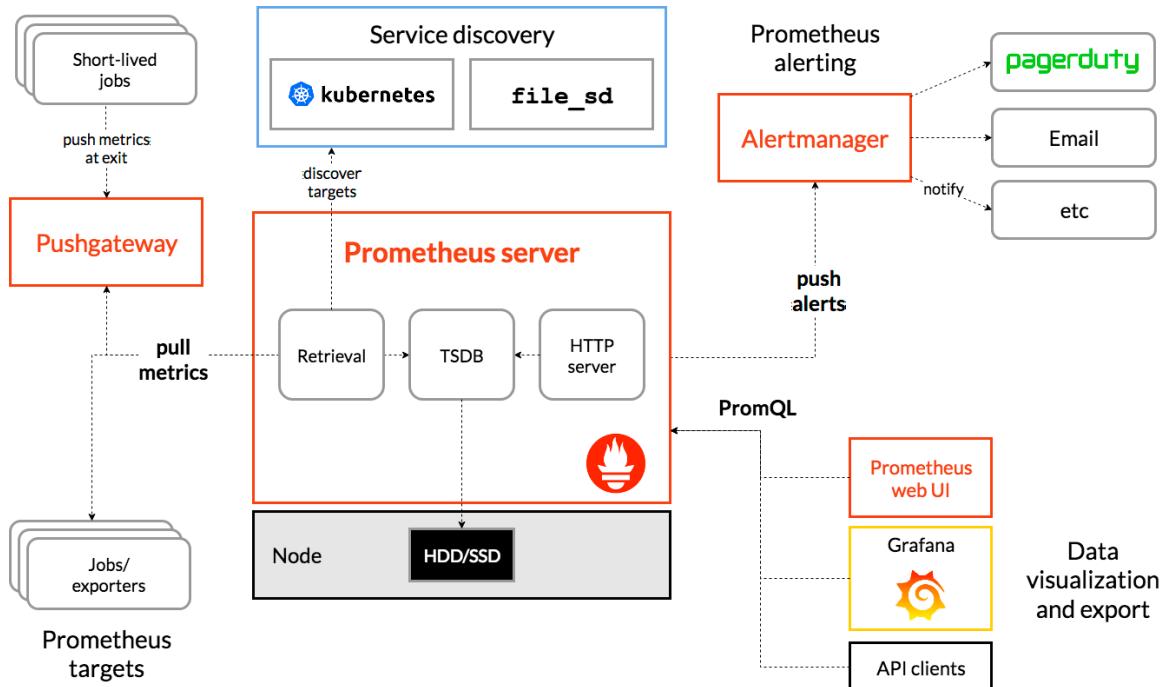
### **2.1. Monitoring tools**

#### **2.1.1. Prometheus**



*Hình 1. Prometheus*

Prometheus là một hệ thống giám sát và cảnh báo mã nguồn mở được phát triển bởi SoundCloud. Prometheus giúp thu thập, lưu trữ, và phân tích các số liệu từ hệ thống ứng dụng.



Hình 2. Prometheus model

Các thành phần chính trong mô hình hoạt động của Prometheus bao gồm:

### Prometheus Server:

Thành phần trung tâm có nhiệm vụ thu thập và lưu trữ dữ liệu từ các endpoint của ứng dụng, sau đó lưu trữ chúng trong cơ sở dữ liệu theo kiểu time-series. Server cũng xử lý các truy vấn từ người dùng thông qua PromQL.

- **Retrieval:**
  - Component chịu trách nhiệm pull metrics từ targets.
  - Thực hiện scraping theo interval được cấu hình.
  - Xử lý service discovery để biết targets cần scrape.
- **TSDB (Time Series Database):**
  - Lưu trữ time series data.
  - Tối ưu hóa cho dữ liệu dạng metrics theo thời gian.
  - Được lưu trên Node's HDD/SSD.
- **HTTP Server:**
  - Expose Prometheus API.

- Cho phép query metrics qua PromQL.
- Phục vụ giao diện web UI.

**Exporters:** Các công cụ này thu thập dữ liệu từ các ứng dụng và dịch vụ, sau đó cung cấp chúng theo định dạng mà Prometheus có thể đọc được. Có nhiều loại Exporters, chẳng hạn như Node Exporter để thu thập số liệu của hệ điều hành, hay các Exporters riêng cho MySQL, Kafka...

- Là các nguồn metrics cần được thu thập.
- Chạy như các jobs hoặc exporters để expose metrics.
- Có thể là các ứng dụng, services, infrastructure components.

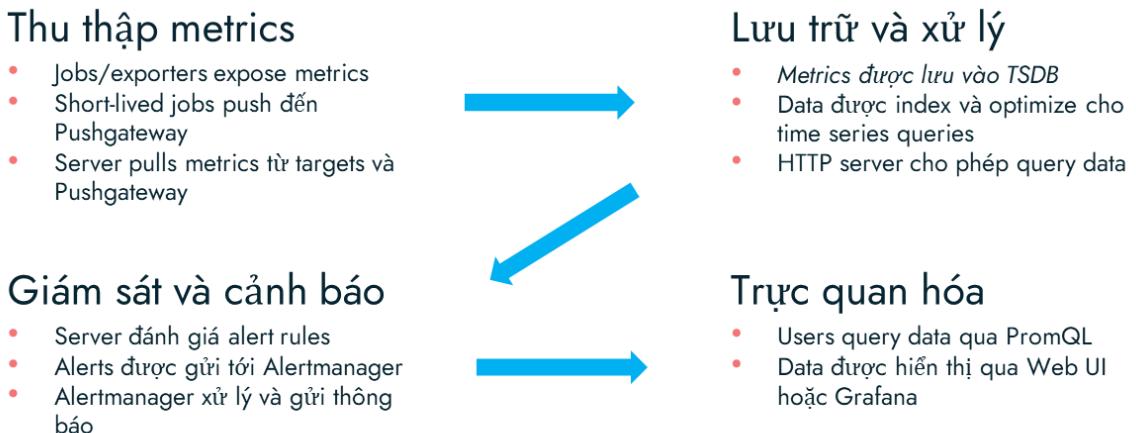
**Push Gateway:** Được sử dụng để thu thập các số liệu từ những dịch vụ có vòng đời ngắn (short-lived jobs). Các ứng dụng gửi dữ liệu trực tiếp tới Push Gateway, sau đó Prometheus sẽ thu thập dữ liệu từ đây. Chức năng chính của Push Gateway:

- Là gateway trung gian để nhận metrics từ short-lived jobs, trong đó short-lived jobs là:
  - Các jobs chạy trong thời gian ngắn.
  - Push metrics tới Pushgateway khi kết thúc.
  - Phù hợp với các batch jobs, cron jobs.
- Lưu trữ tạm thời metrics
- Cho phép Prometheus server pull metrics từ đó

**Alertmanager:** Thành phần quản lý và xử lý các cảnh báo do Prometheus gửi tới. Alertmanager có thể gửi cảnh báo đến email, Telegram, Slack, PagerDuty và nhiều kênh thông báo khác.

- Xử lý và quản lý cảnh báo.
- Nhận alerts từ Prometheus server.
- Có thể gom nhóm, lọc và route alerts.

**Prometheus Query Language (PromQL):** Là ngôn ngữ truy vấn mạnh mẽ, cho phép người dùng trích xuất và phân tích các số liệu đã thu thập.



Hình 3. Prometheus workflow

### 2.1.2. Grafana Loki

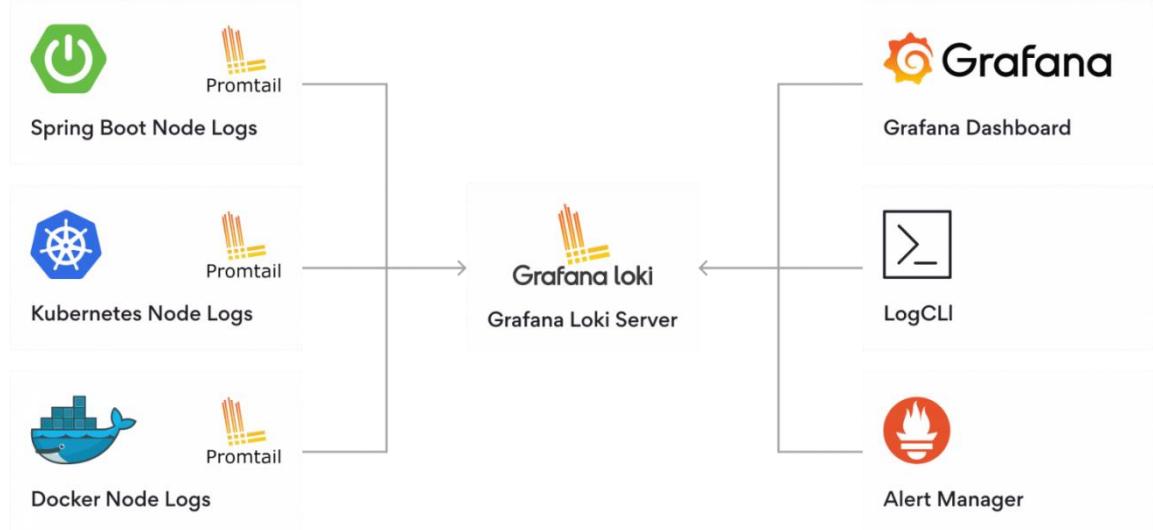


Hình 4. Grafana Loki

Grafana Loki là một hệ thống quản lý và tập hợp log được phát triển bởi Grafana Labs, tập trung vào việc xử lý và truy vấn log với hiệu quả cao và chi phí thấp. Loki được thiết kế để tích hợp mượt mà với Prometheus và Grafana nhằm cung cấp một giải pháp giám sát và phân tích log tối ưu cho các hệ thống phân tán và môi trường cloud-native.

Một vài đặc điểm chính của Grafana Loki:

- Là một hệ thống log aggregation được thiết kế bởi Grafana Labs
- Lấy cảm hứng từ Prometheus, nhưng tập trung vào logs thay vì metrics
- Được thiết kế để hiệu quả về chi phí và dễ vận hành
- Sử dụng cùng labeling system như Prometheus giúp đồng bộ dữ liệu logs với metrics một cách dễ dàng.



Hình 5. *Grafana Loki model*

Các thành phần chính của Grafana Loki:

**Grafana Loki Server:** Đây là thành phần cốt lõi của hệ thống Loki với các nhiệm vụ:

- Nhận và lưu trữ logs từ Promtail.
- Lập chỉ mục (index) logs dựa trên nhãn.
- Xử lý các truy vấn từ Grafana.
- Nén và loại bỏ các logs trùng lặp.
- Quản lý các chính sách lưu trữ logs.

### Log Sources:

- App logs: Logs từ các ứng dụng chạy trên hệ thống.
- Kubernetes Node Logs: Logs từ các nodes trong Kubernetes cluster chứa logs hệ thống, logs container, và sự kiện của Kubernetes.
- Docker Node Logs: Logs từ Docker containers ghi lại đầu ra của container (stdout/stderr) và logs của Docker daemon.

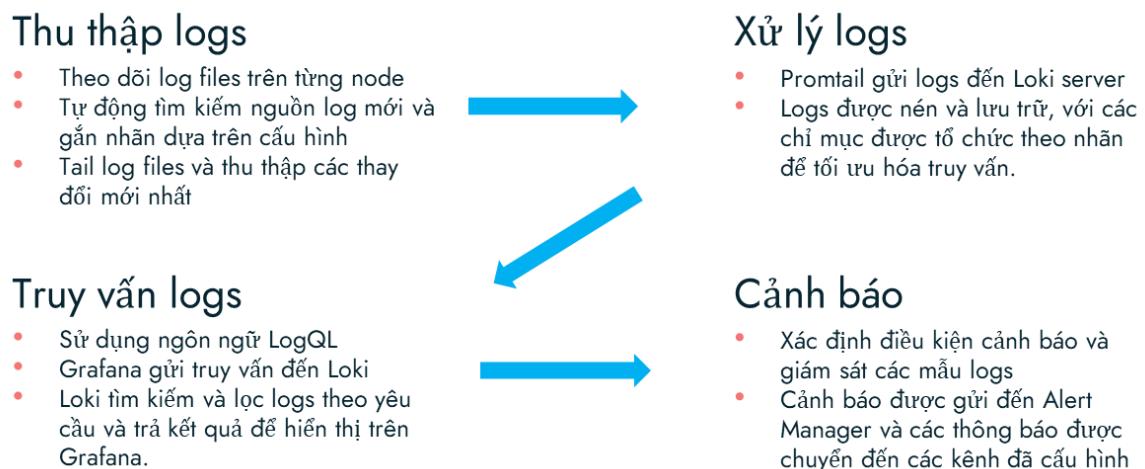
**Promtail:** Promtail là agent thu thập logs, được triển khai trên mỗi node cần thu thập dữ liệu. Nó có các chức năng chính như sau:

- Khám phá các mục tiêu thu thập logs (service discovery).
- Gắn nhãn cho các log streams.

- Đẩy logs đến Loki server để xử lý.
- Theo dõi log files và tích hợp với các dịch vụ phát hiện.

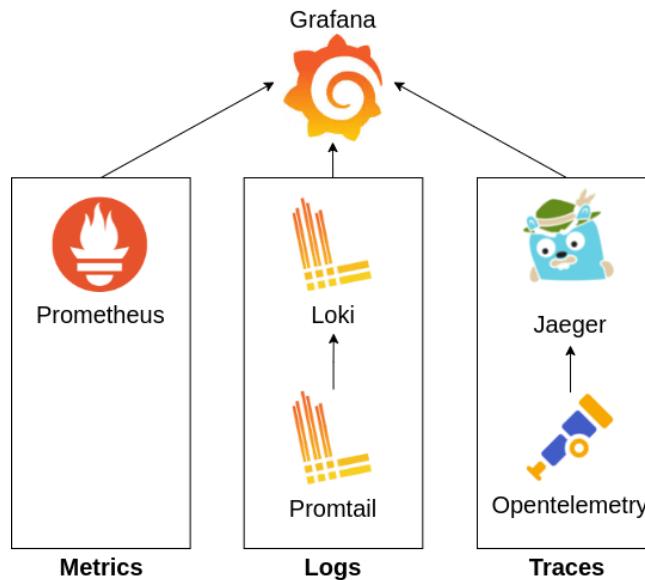
**LogCLI:** Công cụ dòng lệnh cho phép truy vấn Loki, hữu ích cho quá trình gõ lỗi và tự động hóa, đồng thời có thể tích hợp với các script.

**Alert Manager:** Xử lý các cảnh báo dựa trên mẫu logs, tích hợp với các cảnh báo trong Grafana, và quản lý các thông báo qua các kênh khác nhau. Mặc dù Loki không có hệ thống cảnh báo tích hợp như Prometheus, nhưng có thể sử dụng các công cụ bổ trợ như Prometheus hoặc AlertManager để thiết lập cảnh báo dựa trên logs. Cơ chế cảnh báo này được triển khai dựa trên việc phát hiện các mẫu logs cụ thể hoặc các sự kiện đặc biệt.



Hình 6. *Grafana Loki workflow*

### 2.1.3. Grafana



Hình 7. Grafana

Grafana là một công cụ trực quan hóa dữ liệu mã nguồn mở, cho phép người dùng giám sát và phân tích dữ liệu time-series từ các nguồn khác nhau, như Prometheus, Loki, Elasticsearch... Cho phép query, visualize, alert và hiểu về metrics bất kể chúng được lưu trữ ở đâu.

Grafana đặc biệt hữu ích trong việc hiển thị dữ liệu logs, metrics, và cung cấp một nền tảng tập trung để tạo các bảng điều khiển (dashboard) giúp giám sát hệ thống và ứng dụng.

Các thành phần chính của Grafana:

- **Connections:** Tích hợp với nhiều nguồn dữ liệu qua các plugin tích hợp sẵn hoặc từ cộng đồng để kết nối và lấy dữ liệu từ các hệ thống khác như Prometheus, Loki, Elasticsearch...
- **Dashboards:** Giao diện trực quan hóa dữ liệu.
- **Alerting:** Cho phép tạo các quy tắc cảnh báo dựa trên dữ liệu được lấy từ các nguồn tích hợp.
- **Annotations:** Là các ghi chú mà người dùng có thể thêm vào biểu đồ để đánh dấu các sự kiện quan trọng hoặc các thay đổi trong hệ thống.
- **Administration:** Quản lý thông tin xác thực người dùng và phân quyền.

#### 2.1.4. Icinga



Hình 8. *Icinga*

Icinga là một hệ thống giám sát mạng mã nguồn mở, được phát triển từ việc fork Nagios vào năm 2009. Dự án này ra đời nhằm đáp ứng nhu cầu của cộng đồng một cách nhanh chóng và linh hoạt hơn, với phiên bản ổn định 1.0 được phát hành vào tháng 12 năm 2009.

**Chức năng chính:** Icinga kế thừa và phát triển các tính năng của Nagios, đồng thời bổ sung thêm nhiều cải tiến quan trọng. Hệ thống cung cấp khả năng giám sát toàn diện các thành phần mạng, bao gồm việc theo dõi các dịch vụ mạng như SMTP, POP3, HTTP, đồng thời giám sát tài nguyên máy chủ như CPU, dung lượng ổ đĩa và các thiết bị phần cứng. Đặc biệt, Icinga cho phép người dùng tùy chỉnh và phát triển các plugin kiểm tra theo nhu cầu riêng.

**Hệ thống thông báo và trực quan hóa:** Icinga cung cấp hệ thống cảnh báo đa dạng thông qua email, SMS và các phương thức tùy chỉnh. Người dùng có thể lựa chọn giữa hai giao diện: Icinga Classic UI và Icinga Web. Hệ thống báo cáo được xây dựng trên nền tảng Jasper Reports, cho phép tạo báo cáo tự động với nhiều mức độ truy cập khác nhau, kèm theo khả năng hiển thị đồ thị hiệu suất thông qua các công cụ như PNP4Nagios và InGraph.

**Ưu điểm của Icinga2:** Icinga2 mang đến nhiều ưu điểm nổi bật trong lĩnh vực giám sát hệ thống:

- Tính linh hoạt cao trong kiến trúc modular, cho phép dễ dàng mở rộng và tùy chỉnh theo nhu cầu cụ thể của tổ chức. Người dùng có thể thêm hoặc bớt các thành phần mà không ảnh hưởng đến hoạt động của toàn hệ thống.

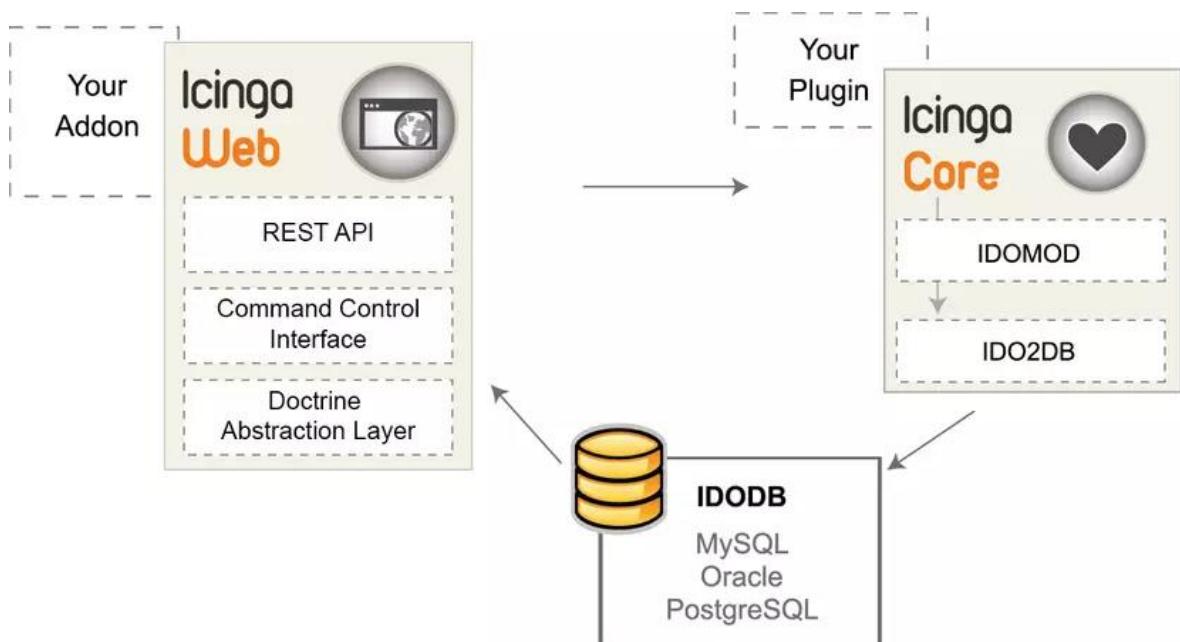
- Khả năng tích hợp mạnh mẽ với các công cụ bên thứ ba như Grafana, Elasticsearch, và các hệ thống cơ sở dữ liệu khác nhau (MySQL, PostgreSQL, Oracle). Điều này tạo ra một hệ sinh thái đa dạng và phong phú cho việc giám sát và phân tích.
- Cơ chế phân tán hiệu quả cho phép giám sát các hệ thống quy mô lớn với nhiều vị trí địa lý khác nhau. Icinga2 có thể duy trì hiệu suất ổn định ngay cả khi số lượng host và service được giám sát tăng lên đáng kể.
- Khả năng tự động hóa cao thông qua API RESTful và các công cụ tự động hóa như Puppet, Chef, và Ansible. Điều này giúp đơn giản hóa quá trình triển khai và quản lý cấu hình trên quy mô lớn.
- Cộng đồng người dùng lớn và tích cực, thường xuyên đóng góp các plugin mới và chia sẻ giải pháp cho các vấn đề phổ biến. Điều này đảm bảo hệ thống luôn được cập nhật và cải tiến liên tục.
- Tính bảo mật cao với các tính năng như mã hóa SSL/TLS cho communication giữa các node, xác thực hai lớp, và phân quyền chi tiết cho người dùng.
- Khả năng template hóa mạnh mẽ, cho phép tái sử dụng các cấu hình và giảm thiểu công sức khi triển khai trên nhiều hệ thống tương tự nhau.
- Hiệu năng cao trong việc xử lý và phân tích dữ liệu giám sát, với khả năng xử lý hàng nghìn check đồng thời mà không gây tải nặng cho hệ thống.
- Giao diện web trực quan và dễ sử dụng, cung cấp dashboard tùy chỉnh và các công cụ báo cáo chuyên nghiệp, giúp người quản trị dễ dàng nắm bắt tình trạng hệ thống.
- Chi phí triển khai thấp do là giải pháp mã nguồn mở, phù hợp với nhiều quy mô doanh nghiệp khác nhau từ nhỏ đến lớn.

**Nhược điểm của Icinga2:** Mặc dù mạnh mẽ, Icinga2 vẫn tồn tại một số hạn chế:

- Quá trình cài đặt và cấu hình khá phức tạp, đòi hỏi kiến thức chuyên sâu về hệ thống.
- Tài liệu hướng dẫn đôi khi không được cập nhật kịp thời với các phiên bản mới.

- Tốn nhiều thời gian để tối ưu hóa các cảnh báo, tránh các false positive.
- Krang giới hạn về khả năng tùy chỉnh giao diện người dùng.
- Đường cong học tập dốc, đặc biệt đối với người mới bắt đầu.
- Một số tính năng nâng cao yêu cầu phải cài đặt thêm plugin từ bên thứ ba.

**Kiến trúc:** Icinga được thiết kế theo mô hình kiến trúc phân tầng, trong đó mỗi thành phần đều có vai trò và trách nhiệm riêng biệt. Kiến trúc này cho phép hệ thống hoạt động một cách độc lập và linh hoạt, đồng thời dễ dàng mở rộng khi cần thiết.



Hình 9. Kiến trúc Icinga2

### Các thành phần chính:

- Icinga Web Interface:
  - Là lớp giao diện người dùng được xây dựng hiện đại, tương tác với người dùng cuối
  - REST API đóng vai trò quan trọng trong việc cung cấp interface chuẩn để các ứng dụng bên ngoài có thể tương tác với hệ thống
  - Command Control Interface cho phép quản trị viên thực hiện các thao tác điều khiển và cấu hình hệ thống

- Doctrine Abstraction Layer tạo ra một lớp trùu tượng giữa ứng dụng và cơ sở dữ liệu, giúp việc thay đổi CSDL trở nên dễ dàng hơn
- Icinga Core:
  - Là "trái tim" của hệ thống, nơi xử lý tất cả các logic giám sát
  - IDOMOD (Icinga Data Output Module) có nhiệm vụ xử lý và định dạng dữ liệu từ engine giám sát
  - IDO2DB là cầu nối chuyển đổi dữ liệu từ IDOMOD sang định dạng phù hợp để lưu trữ trong CSDL
  - Thực hiện các nhiệm vụ giám sát, kiểm tra và phân tích trạng thái của các đối tượng được giám sát
- IDODB (Icinga Data Output Database):
  - Là nơi lưu trữ tất cả dữ liệu giám sát và cấu hình của hệ thống
  - Hỗ trợ đa nền tảng CSDL (MySQL, PostgreSQL, Oracle) giúp tổ chức linh hoạt trong việc lựa chọn công nghệ phù hợp
  - Lưu trữ lịch sử giám sát, trạng thái, metrics và các thông tin cấu hình

**Quy trình hoạt động:** Icinga2 hoạt động theo mô hình client-server. Trên server trung tâm, Icinga2 liên tục thu thập thông tin từ các agent được cài đặt trên các máy chủ cần giám sát. Quá trình này được thực hiện thông qua các check command định kỳ, với kết quả được lưu trữ và phân tích. Khi phát hiện bất thường, hệ thống sẽ kích hoạt các notification rule đã được cấu hình để thông báo cho người quản trị. Các bước chi tiết như sau:

- Thu thập dữ liệu:
  - Icinga Core liên tục thực hiện các check commands
  - Dữ liệu được thu thập từ nhiều nguồn khác nhau (hosts, services, applications)
  - Các plugin có thể mở rộng khả năng thu thập dữ liệu
- Xử lý dữ liệu:
  - IDOMOD nhận dữ liệu thô từ Core

- Thực hiện các bước xử lý và chuẩn hóa dữ liệu
- IDO2DB chuyển đổi dữ liệu sang format phù hợp với CSDL
- Lưu trữ dữ liệu:
  - Dữ liệu được lưu vào IDODB
  - Hỗ trợ nhiều loại CSDL khác nhau
  - Đảm bảo tính nhất quán và toàn vẹn dữ liệu
- Hiển thị và tương tác:
  - Web Interface truy xuất dữ liệu thông qua Doctrine Layer
  - REST API cung cấp interface cho các ứng dụng bên ngoài
  - Command Control Interface xử lý các lệnh từ người dùng.

## Thu thập dữ liệu

- Các plugin giám sát gửi thông tin về trạng thái hệ thống
- Giám sát: CPU, RAM, disk, services, network traffic...
- Hỗ trợ nhiều loại plugin: Nagios plugins, custom plugins...

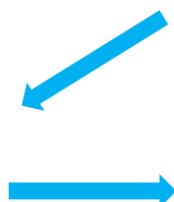


## Xử lý dữ liệu

- Core (IDOMOD + IDO2DB): Chuyển đổi và chuẩn hóa dữ liệu giám sát
- Phân tích trạng thái: OK, Warning, Critical, Unknown
- Kiểm tra ngưỡng cảnh báo và thực hiện notify

## Lưu trữ dữ liệu

- IDODB: Cập nhật vào cơ sở dữ liệu để lưu trữ lịch sử
- Hỗ trợ: MySQL, PostgreSQL, Oracle
- Lưu logs, metrics, events và performance data



## Hiển thị kết quả

- Icinga Web: Truy xuất và trình bày thông tin qua giao diện web
- Dashboard trực quan: graphs, charts, status maps
- Hỗ trợ REST API cho tích hợp bên thứ 3 Cung cấp reporting và alert management

Hình 10. Icinga workflow

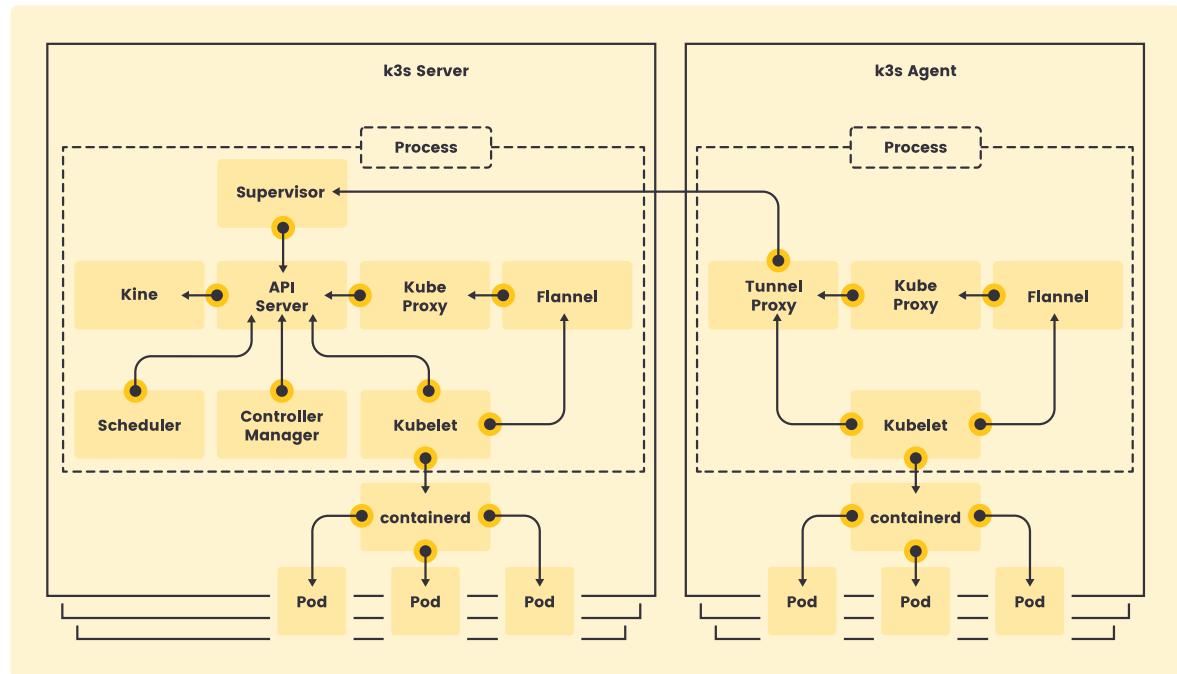
## 2.2. K3s



Hình 11. K3s

K3s là một bản phân phối Kubernetes nhẹ được phát triển bởi Rancher Labs, hiện là một phần của SUSE. Ra mắt vào năm 2019, K3s được thiết kế đặc biệt cho các môi trường edge computing, IoT và các hệ thống có tài nguyên hạn chế. Với kích thước nhị phân chỉ khoảng 50MB, K3s mang đến một giải pháp container orchestration đầy đủ tính năng nhưng vẫn duy trì được tính đơn giản và hiệu quả.

**Đặc điểm và Kiến trúc:** K3s đạt được kích thước nhỏ gọn bằng cách loại bỏ các tính năng cũ, các driver không thiết yếu và các plugins bên thứ ba khỏi Kubernetes tiêu chuẩn. Thay vào đó, nó tích hợp nhiều thành phần thiết yếu vào một quy trình đơn lẻ. Kiến trúc này bao gồm containerd làm container runtime, Flannel cho networking, và SQLite làm datastore mặc định, mặc dù vẫn hỗ trợ etcd và các database khác.



Hình 12. Kiến trúc K3S

K3s được thiết kế với kiến trúc đơn giản nhưng mạnh mẽ, tập trung vào hai thành phần chính:

- Server Node (Control Plane) và Agent Nodes (Worker Nodes). Server Node đóng vai trò là bộ não của hệ thống, chứa các components thiết yếu

như API Server để xử lý requests, Controller Manager để quản lý các controller, và Scheduler để lập lịch cho pods. Đặc biệt, K3s tích hợp containerd làm container runtime và Flannel cho network overlay, cùng với SQLite làm datastore mặc định, mặc dù vẫn hỗ trợ etcd cho các deployment lớn hơn.

- Về phía Agent Nodes, chúng chạy các thành phần worker cơ bản như Kubelet để quản lý containers, Kube-proxy cho network rules, và cũng sử dụng containerd cùng Flannel. Điểm đặc biệt trong kiến trúc K3s là việc đóng gói tất cả components vào một binary duy nhất, giúp giảm đáng kể độ phức tạp trong việc triển khai và quản lý.

K3s cung cấp tính năng high availability thông qua khả năng chạy nhiều server nodes với automatic failover khi sử dụng external database. Về mặt bảo mật, kiến trúc này tích hợp sẵn TLS encryption, RBAC, và token-based node registration. Đặc biệt, K3s tối ưu hóa việc sử dụng tài nguyên, làm cho nó trở thành lựa chọn lý tưởng cho edge computing và các môi trường có tài nguyên hạn chế.

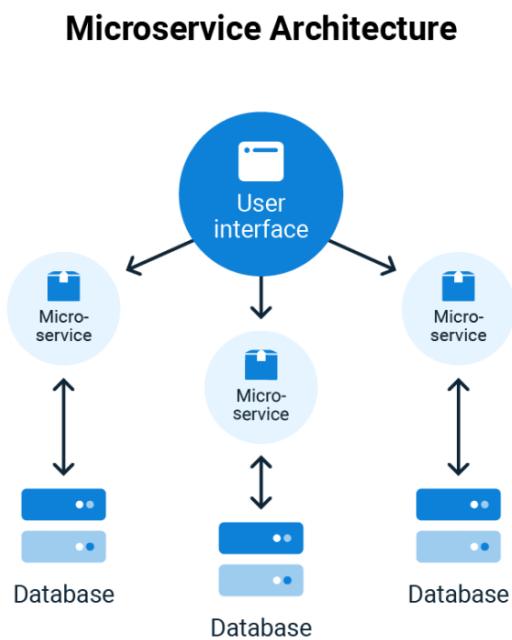
Luồng hoạt động của K3s bắt đầu từ việc khởi tạo server node, sau đó các agent nodes đăng ký thông qua token bảo mật. API requests được xử lý bởi API server, trong khi scheduler phân phối workloads và các controllers đảm bảo desired state của hệ thống. Toàn bộ kiến trúc được thiết kế để duy trì sự cân bằng giữa tính đơn giản và khả năng mở rộng, đồng thời vẫn đảm bảo tính tương thích với Kubernetes tiêu chuẩn.

**Tính năng chính:** K3s duy trì hầu hết các tính năng quan trọng của Kubernetes tiêu chuẩn trong khi vẫn đơn giản hóa quá trình triển khai và quản lý. Nó cung cấp load balancing tích hợp thông qua ServiceLB, quản lý storage thông qua Local Path Provisioner, và hệ thống quản lý chứng chỉ tích hợp. Điều đặc biệt là K3s tự động xử lý việc cấu hình TLS và bảo mật, giúp giảm thiểu công sức setup ban đầu.

K3s đặc biệt phù hợp cho các trường hợp sử dụng như edge computing, nơi tài nguyên hạn chế là một thách thức. Nó được sử dụng rộng rãi trong các môi trường IoT, các branch office từ xa, và các hệ thống embedded. Khả năng chạy trên các thiết bị ARM như Raspberry Pi làm cho nó trở thành lựa chọn lý tưởng cho các dự án development và testing quy mô nhỏ.

**Yêu cầu hệ thống và hiệu năng:** K3s có thể chạy trên các hệ thống với chỉ 512MB RAM và 200MB disk space. Server node yêu cầu tối thiểu 1 CPU core, trong khi agent node có thể chạy với resources thấp hơn. So với Kubernetes tiêu chuẩn, K3s sử dụng ít memory hơn đáng kể và khởi động nhanh hơn, thường chỉ trong vài giây.

### 2.3. Microservices



Hình 13. Kiến trúc Microservice

Microservices là một phương pháp phát triển phần mềm, trong đó một ứng dụng lớn được chia thành các dịch vụ nhỏ, độc lập với nhau. Mỗi dịch vụ thực hiện một chức năng cụ thể và giao tiếp với các dịch vụ khác thông qua các APIs.

### **Đặc điểm của Microservices:**

- Độc lập và tách biệt: Các microservices có thể được phát triển, triển khai và mở rộng độc lập mà không ảnh hưởng đến các phần khác của hệ thống.
- Định hướng theo chức năng: Mỗi microservice tập trung vào một chức năng cụ thể.
- Công nghệ đa dạng: Các microservices có thể được phát triển bằng nhiều ngôn ngữ lập trình và công nghệ khác nhau.

### **Kiến trúc Microservices so với kiến trúc Monolithic:**

- Monolithic: Là mô hình kiến trúc truyền thống, trong đó toàn bộ ứng dụng được phát triển và triển khai như một khối duy nhất. Điều này có thể gây ra nhiều khó khăn trong việc bảo trì, mở rộng và triển khai.
- Microservices: Chia nhỏ ứng dụng thành các dịch vụ nhỏ, độc lập. Mỗi dịch vụ có thể được phát triển, triển khai và mở rộng một cách độc lập.

### **Lợi ích của Microservices:**

- Do các dịch vụ được tách biệt, việc bảo trì và mở rộng có thể được thực hiện dễ dàng hơn.
- Các dịch vụ có thể được triển khai độc lập, giúp dễ dàng thực hiện triển khai liên tục.
- Lỗi ở một dịch vụ không làm sụp đổ toàn bộ hệ thống, giúp hệ thống có khả năng chịu lỗi cao hơn.
- Mỗi đội phát triển có thể chọn công nghệ phù hợp nhất cho dịch vụ mà họ phát triển.

## 2.4. AWS



Hình 14. AWS

Amazon Web Services (AWS) là một trong những nhà cung cấp dịch vụ điện toán đám mây hàng đầu trên thế giới. AWS cung cấp một loạt các dịch vụ điện toán đám mây, từ lưu trữ đến tính toán, từ máy ảo đến máy học và trí tuệ nhân tạo.

Một vài dịch vụ phổ biến của AWS:

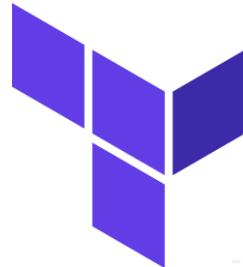
- Amazon EC2 (Elastic Compute Cloud): Cung cấp các máy ảo có khả năng mở rộng linh hoạt trên đám mây của AWS.
- Amazon S3 (Simple Storage Service): Dịch vụ lưu trữ đối tượng đáng tin cậy, an toàn và có khả năng mở rộng.
- Amazon ECS (Elastic Container Service): Dịch vụ quản lý và triển khai các container trên AWS.
- Amazon EKS (Elastic Kubernetes Service): Dịch vụ được quản lý hoàn toàn giúp triển khai, quản lý và vận hành Kubernetes trên AWS.

Lợi ích của AWS:

- AWS cung cấp khả năng mở rộng linh hoạt để đáp ứng nhu cầu tăng trưởng của doanh nghiệp.
- AWS cho phép trả tiền theo mức sử dụng, giúp giảm thiểu các chi phí vốn ban đầu.
- AWS cung cấp một loạt các dịch vụ linh hoạt để phù hợp với mọi loại ứng dụng và yêu cầu kỹ thuật.
- AWS cung cấp một môi trường đám mây an toàn, đáng tin cậy và tuân thủ các tiêu chuẩn bảo mật quốc tế.

## 2.5. Infrastructure as Code (IaC)

### 2.5.1. Terraform



Hình 15. Terraform

Terraform là một công cụ mã nguồn mở được phát triển bởi HashiCorp, giúp tự động hóa việc quản lý cơ sở hạ tầng dưới dạng mã. Terraform cho phép các nhà phát triển và quản trị hệ thống định nghĩa hạ tầng của mình bằng các file cấu hình có thể đọc được và dễ dàng quản lý.

Đặc điểm và lợi ích của Terraform:

- Terraform có thể tương tác với nhiều nhà cung cấp điện toán đám mây khác nhau như AWS, Azure, Google Cloud, và nhiều hệ thống cơ sở hạ tầng khác.
- Các module trong Terraform cho phép tái sử dụng và chia sẻ cấu hình giữa các dự án và nhóm.
- Terraform cho phép tự động hóa các công việc như tạo, cập nhật, và xóa các tài nguyên cơ sở hạ tầng.
- Terraform ghi nhật ký tất cả các thay đổi được thực hiện và cung cấp cơ chế xác nhận trước khi triển khai thay đổi.

Một vài thành phần chính của Terraform: Providers (cung cấp API để tương tác với các dịch vụ đám mây hoặc hệ thống), Modules (tập hợp các cấu hình được tái sử dụng), Resources (các phần tử cơ bản được quản lý, như máy ảo, mạng), Data Sources (lấy thông tin từ bên ngoài để sử dụng trong cấu hình), State (tệp lưu trữ trạng thái cơ sở hạ tầng).

### 2.5.2. Ansible



Hình 16. Ansible

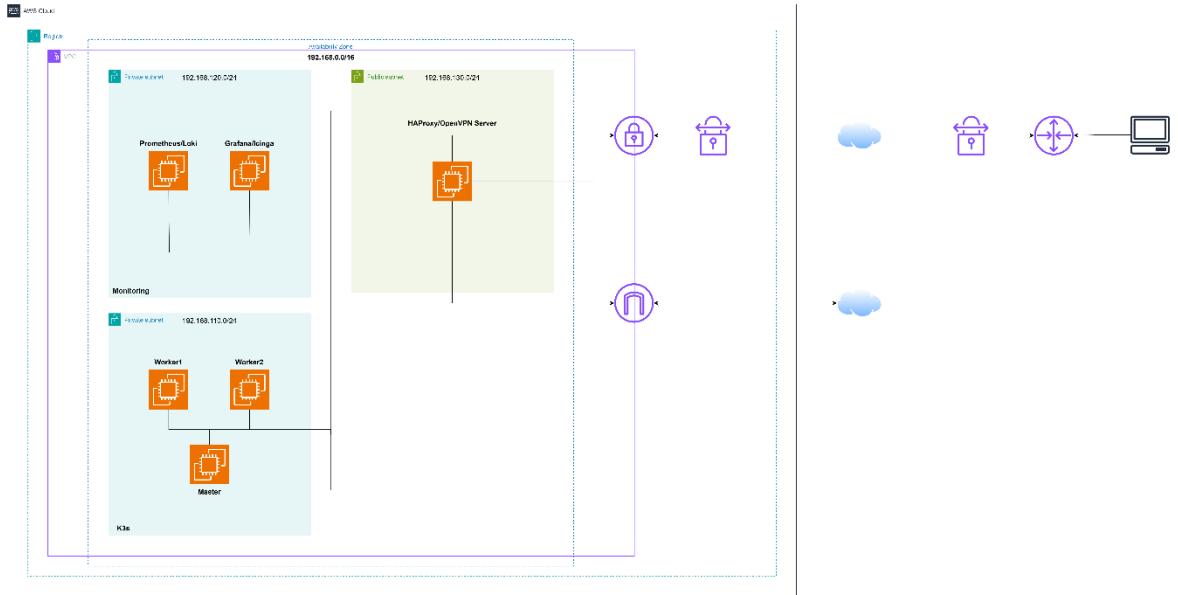
Ansible là một công cụ tự động hóa mã nguồn mở, được thiết kế để tự động hóa việc quản lý cấu hình, triển khai ứng dụng, và các tác vụ phức tạp khác. Ansible nổi bật nhờ tính đơn giản, không yêu cầu cài đặt các agents trên các máy đích, và khả năng mở rộng mạnh mẽ. Ansible sử dụng ngôn ngữ YAML để định nghĩa các kịch bản (playbooks) dễ đọc, giúp các quản trị viên hệ thống và nhà phát triển dễ dàng viết và duy trì các kịch bản tự động hóa.

Ansible được sử dụng rộng rãi trong nhiều lĩnh vực khác nhau bao gồm:

- Quản lý cấu hình: Ansible giúp tự động hóa việc cài đặt và cấu hình phần mềm trên các máy chủ, đảm bảo rằng tất cả các máy chủ đều có cấu hình nhất quán.
- Triển khai ứng dụng: Ansible có thể tự động hóa quá trình triển khai ứng dụng, từ việc chuẩn bị môi trường, cài đặt phần mềm, đến cấu hình và khởi động ứng dụng.
- Quản lý hạ tầng: Ansible có thể quản lý và điều phối các tài nguyên hạ tầng, chẳng hạn như tạo và cấu hình máy ảo, thiết lập mạng và lưu trữ.
- Tự động hóa bảo trì: Ansible giúp tự động hóa các tác vụ bảo trì định kỳ, như cập nhật phần mềm, kiểm tra tính khả dụng của dịch vụ, và sao lưu dữ liệu.
- Orchestration: Ansible có thể điều phối các công việc phức tạp liên quan đến nhiều hệ thống khác nhau, chẳng hạn như triển khai ứng dụng đa tầng hoặc quản lý container.

## Chương 3. HIỆN THỰC ĐỀ TÀI

### 3.1. Mô hình mạng



Hình 17. Mô hình mạng

Sử dụng AWS để triển khai hạ tầng cloud cho các EC2 instance nằm trong 1 VPC với 3 subnet mang vai trò riêng, trong đó:

- VPC (Virtual Private Cloud): Môi trường mạng ảo trên AWS có thể tự do cấu hình địa chỉ mạng theo ý muốn, trong đây sẽ bao gồm các subnet:
  - Private subnet cho cụm K3s: Vùng mạng dùng để triển khai các node dành cho việc triển khai ứng dụng lên cụm, trong đó cụm K3s sẽ bao gồm 3 EC2 instance tương ứng với 3 node (1 master, 2 worker).
  - Private subnet cho monitoring: Vùng mạng dành cho việc giám sát toàn bộ hạ tầng, gồm 2 EC2 instance triển khai các server như Prometheus, Grafana Loki, Icinga, Grafana.
  - Public subnet: Vùng mạng có thể truy cập trực tiếp từ internet đóng vai trò là đầu ra internet duy nhất cho VPC, trong đây sẽ chứa 1 EC2 instance được gán EIP mang địa chỉ IP public và được xem là NAT instance có khả năng forward các gói tin nhận được có

source/destination IP từ nhiều nơi. Ngoài ra đối với mô hình đã thiết kế, ở instance này sẽ triển khai các server như:

- HA Proxy: Đóng vai trò là một máy chủ web giúp các server bên trong hạ tầng có thể được truy cập từ bên ngoài thông qua domain được mapping, cấu hình các ACL để đối với từng domain sẽ được trả về backend tương ứng. Ngoài ra, đối với ứng dụng được triển khai bên trong cụm K3s, HA proxy sẽ là load balancer giúp phân phối traffic đến các node bên trong cụm một cách đồng đều nhằm cân bằng tải cho cụm.
- OpenVPN: Triển khai VPN nhằm mục đích có thể kết nối an toàn đến hạ tầng từ máy local để phục vụ cho việc kiểm thử, cấu hình một cách nhanh chóng.

Lý do cho việc triển khai mô hình trên:

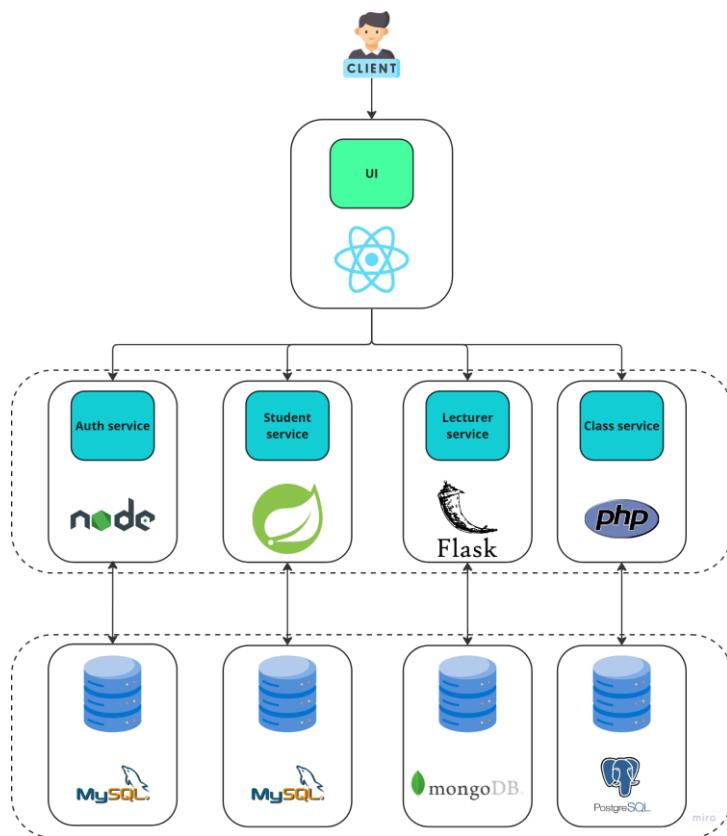
- Dễ dàng quản lý, tách biệt các tài nguyên và có thể tùy chỉnh các thông số cho hạ tầng một cách linh hoạt.
- Giám sát hệ thống dễ dàng, giúp theo dõi và phân tích hiệu suất của hạ tầng mà không làm lộ thông tin nhạy cảm ra bên ngoài.
- Khả năng bảo mật khi các instance nằm bên trong private subnet giúp che giấu địa chỉ IP thực khi đi ra ngoài internet, phục vụ kết nối VPN an toàn chỉ được truy cập từ các máy nhất định được cung cấp VPN profile đảm bảo dữ liệu được truyền tải một cách an toàn.
- Tiết kiệm chi phí khi chỉ cần 1 EIP và không cần phải triển khai thêm NAT gateway.

**Bảng 1. Bảng phân chia địa chỉ IP cho từng instance**

VPC	Subnet	Host	Server(s)
192.168.0.0/16	192.168.110.0/24	192.168.110.10	K3s Master
		192.168.110.11	K3s Worker 1

		192.168.110.12	K3s Worker 2
192.168.120.0/24		192.168.120.20	Icinga/Grafana
		192.168.120.21	Prometheus/Loki
	192.168.130.0/24	192.168.130.30	HAProxy

### 3.2. Ứng dụng microservices



Hình 18. Mô hình ứng dụng microservices

Các tính năng đã triển khai:

- **Auth service:**

- Ngôn ngữ: JavaScript (NodeJs, ExpressJs)
- Cơ sở dữ liệu: MySQL
- Chức năng: Xử lý các chức năng liên quan đến xác thực người dùng, bao gồm đăng nhập, đăng ký, và quản lý phiên làm việc của người dùng.

- **Student service:**

- Ngôn ngữ: Java (Spring Boot framework)
- Cơ sở dữ liệu: MySQL
- Chức năng: Quản lý thông tin sinh viên, bao gồm lưu trữ, truy xuất, cập nhật và xóa thông tin sinh viên.

- **Lecturer service:**

- Ngôn ngữ: Python (Flask)
- Cơ sở dữ liệu: MongoDB
- Chức năng: Quản lý thông tin giảng viên, bao gồm lưu trữ, truy xuất, cập nhật và xóa thông tin giảng viên.

- **Class service:**

- Ngôn ngữ: PHP
- Cơ sở dữ liệu: PostgreSQL
- Chức năng: Quản lý thông tin lớp học, bao gồm lưu trữ, truy xuất, cập nhật và xóa thông tin lớp học.

Danh sách các API:

Service	Base URL	Trạng thái	URL(s)
Auth service	<a href="https://auth-api.th1enlm02.live">https://auth-api.th1enlm02.live</a>	GET	<a href="https://auth-api.th1enlm02.live">https://auth-api.th1enlm02.live</a>
		POST	<a href="https://auth-api.th1enlm02.live/login">https://auth-api.th1enlm02.live/login</a>
		POST	<a href="https://auth-api.th1enlm02.live/register">https://auth-api.th1enlm02.live/register</a>
		GET	<a href="https://auth-api.th1enlm02.live/logout">https://auth-api.th1enlm02.live/logout</a>
Student service	<a href="https://student-api.th1enlm02.live">https://student-api.th1enlm02.live</a>	GET	<a href="https://student-api.th1enlm02.live/student/getAll">https://student-api.th1enlm02.live/student/getAll</a>
		POST	<a href="https://student-api.th1enlm02.live/student/add">https://student-api.th1enlm02.live/student/add</a>
		PUT	<a href="https://student-api.th1enlm02.live/student/{id}">https://student-api.th1enlm02.live/student/{id}</a>
		DELETE	<a href="https://student-api.th1enlm02.live/student/{id}">https://student-api.th1enlm02.live/student/{id}</a>
Lecturer service	<a href="https://lecturer-api.th1enlm02.live">https://lecturer-api.th1enlm02.live</a>	GET	<a href="https://lecturer-api.th1enlm02.live/lecturer/getAll">https://lecturer-api.th1enlm02.live/lecturer/getAll</a>
		POST	<a href="https://lecturer-api.th1enlm02.live/lecturer/add">https://lecturer-api.th1enlm02.live/lecturer/add</a>
		PUT	<a href="https://lecturer-api.th1enlm02.live/lecturer/{id}">https://lecturer-api.th1enlm02.live/lecturer/{id}</a>
		DELETE	<a href="https://lecturer-api.th1enlm02.live/lecturer/{id}">https://lecturer-api.th1enlm02.live/lecturer/{id}</a>
Class service	<a href="https://class-api.th1enlm02.live">https://class-api.th1enlm02.live</a>	GET	<a href="https://class-api.th1enlm02.live/class/getAll">https://class-api.th1enlm02.live/class/getAll</a>
		POST	<a href="https://class-api.th1enlm02.live/class/add">https://class-api.th1enlm02.live/class/add</a>
		PUT	<a href="https://class-api.th1enlm02.live/class/{classCode}">https://class-api.th1enlm02.live/class/{classCode}</a>
		DELETE	<a href="https://class-api.th1enlm02.live/class/{classCode}">https://class-api.th1enlm02.live/class/{classCode}</a>

Hình 19. Danh sách các API

### 3.3. Triển khai cơ sở hạ tầng và ứng dụng

#### 3.3.1. Triển khai cơ sở hạ tầng

Sử dụng Terraform định nghĩa các resource sẽ triển khai cho hệ thống, sau đó apply để triển khai toàn bộ cơ sở hạ tầng.

```
Apply complete! Resources: 53 added, 0 changed, 0 destroyed.

Outputs:

bastion_instance_ip_public = [
  "34.229.50.228",
]
instance_ips = {
  "bastion_instance" = [
    "192.168.130.30",
  ]
  "cluster_instance" = [
    "192.168.110.10",
    "192.168.110.11",
    "192.168.110.12",
  ]
  "monitoring_backend_instance" = [
    "192.168.120.21",
  ]
  "monitoring_frontend_instance" = [
    "192.168.120.20",
  ]
}
```

Hình 20. Triển khai hạ tầng bằng Terraform

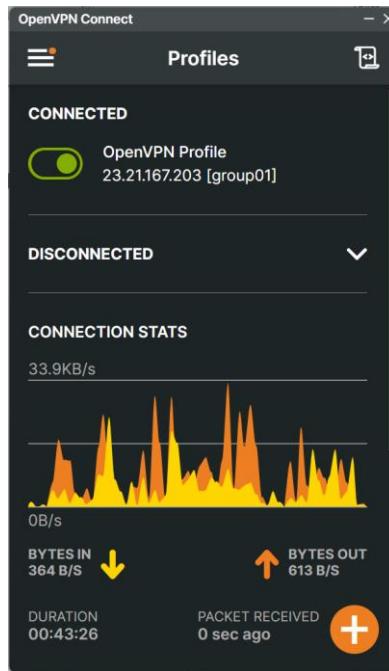
Sau khi triển khai hạ tầng thành công, truy cập vào AWS console để kiểm tra, các instance đều ở trạng thái running.

Instances (6) <a href="#">Info</a>							
Name		Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
<input type="checkbox"/>	NT531.P11-cluster-instance-1	i-06b5c39b8c3378577	<span>Running</span> <a href="#">View details</a> <a href="#">Logs</a>	t2.medium	<span>2/2 checks passed</span> <a href="#">View alarms</a> +	us-east-1a	-
<input type="checkbox"/>	NT531.P11-cluster-instance-2	i-04ce18cc2072cf19e	<span>Running</span> <a href="#">View details</a> <a href="#">Logs</a>	t2.medium	<span>2/2 checks passed</span> <a href="#">View alarms</a> +	us-east-1a	-
<input type="checkbox"/>	NT531.P11-cluster-instance-0	i-020a3ad5897704356	<span>Running</span> <a href="#">View details</a> <a href="#">Logs</a>	t2.medium	<span>2/2 checks passed</span> <a href="#">View alarms</a> +	us-east-1a	-
<input type="checkbox"/>	NT531.P11-bastion-instance-0	i-0114482b8e8a29910	<span>Running</span> <a href="#">View details</a> <a href="#">Logs</a>	t2.small	<span>2/2 checks passed</span> <a href="#">View alarms</a> +	us-east-1a	ec2-34-229-50-22
<input type="checkbox"/>	NT531.P11-monitoring-fe-instance-0	i-0ab22fc6df4fa1b8d	<span>Running</span> <a href="#">View details</a> <a href="#">Logs</a>	t2.medium	<span>2/2 checks passed</span> <a href="#">View alarms</a> +	us-east-1a	-
<input type="checkbox"/>	NT531.P11-monitoring-be-instanc...	i-05c1d4ba866b2c1e1	<span>Running</span> <a href="#">View details</a> <a href="#">Logs</a>	t2.medium	<span>2/2 checks passed</span> <a href="#">View alarms</a> +	us-east-1a	-

Hình 21. Kiểm tra trạng thái các instance trên AWS console



Hình 22. Thông tin resource map của VPC



Hình 23. Kết nối VPN đến hổ tăng để cấu hình

File inventory chứa thông tin của các instance sẽ được cấu hình tự động bằng Ansible:

```

inventory.yml
1  all:
2    children:
3      bastion:
4        monitoring:
5        cluster:
6        vars:
7          ansible_user: ubuntu
8          ansible_ssh_common_args: '-o StrictHostKeyChecking=no'
9          ansible_python_interpreter: /usr/bin/python3
10         project: group01
11
12       bastion:
13         hosts:
14           bastion_host:
15             ansible_host: 192.168.130.30 # i-0114482b8e8a29910
16         vars:
17           subnet: public-subnet
18
19       monitoring:
20         hosts:
21           monitoring_fe_host:
22             ansible_host: 192.168.120.20 # i-0ab22fc6df4fa1b8d
23           monitoring_be_host:
24             ansible_host: 192.168.120.21 # i-05c1d4ba866b2c1e1
25         vars:
26           subnet: private-subnet-1
27
28     cluster:
29       hosts:
30         k3s_master_host:
31           ansible_host: 192.168.110.10 # i-020a3ad5897704356
32         k3s_worker_host_1:
33           ansible_host: 192.168.110.11 # i-06b5c39b8c3378577
34         k3s_worker_host_2:
35           ansible_host: 192.168.110.12 # i-04ce18cc2072cf19e
36       vars:
37         subnet: private-subnet-0
38         k3s_version: v1.30.2+k3s1
39         token: "QEpvPaNbGujxZB2w3k9F7tgJ8syAQR"
40         api_endpoint: "{{ hostvars[groups['cluster'][0]]['ansible_host'] }}"
41         extra_server_args: ""
42         extra_agent_args: ""

```

Hình 24. File inventory

Chạy ansible-playbook để tiến hành cấu hình cho các instance, trong đó mỗi server sẽ có role tương ứng, trong từng role sẽ chứa các thư mục có cấu trúc theo quy định và sẽ tự động lấy thông tin các vars, files, tasks, handlers để cấu hình.

The screenshot shows the Visual Studio Code interface with the following details:

- Explorer Panel:** Shows the file structure with `inventory.yml` as the active editor.
- Terminal:** Shows the command `ansible hub [WSL:Ubuntu]` and the output of the Ansible playbooks.
- Bottom Status Bar:** Displays memory usage (10.84%), CPU usage (15m 5s 712ms), and system status.

```
File Edit Selection View Go Run Terminal Help
File Edit Selection View Go Run Terminal Help
inventory > inventory.yml
inventory > inventory.yml
  1: all:
  2:   children:
  3:     bastion:
  4:       hosts:
  5:         - cluster:
  6:           vars:
  7:             ansible_user: ubuntu
  8:             ansible_ssh_common_args: '-o StrictHostKeyChecking=no'
  9:             ansible_python_interpreter: /usr/bin/python3
 10:            project: GroupB
 11:
 12:   bastions:
 13:     hosts:
 14:       bastion_host:
 15:         ansible_host: 192.168.130.30 # l-0ce7e5ae8506f4ed
 16:         vars:
 17:           subnet: public-subnet
 18:
 19:     monitoring:
 20:       hosts:
 21:         monitoring_fe_host:
 22:           ansible_host: 192.168.120.20 # l-0c3df9bf4f33ca16c
 23:         monitoring_be_host:
 24:           ansible_host: 192.168.120.21 # l-0fc415cb43d919f09
 25:         vars:
 26:           subnet: private-subnet-1
 27:
 28:   cluster:
 29:     hosts:
 30:       k3s_master_host:
 31:         ansible_host: 192.168.110.10 # l-0ea31fc297aeff3a28
 32:       k3s_worker_host_1:
 33:         ansible_host: 192.168.110.11 # l-0d2eeff00d5a600147
 34:       k3s_worker_host_2:
 35:         ansible_host: 192.168.110.12 # l-0fffb8a209c1abbb
 36:         vars:
 37:           subnet: private-subnet-0
 38:           k3s_version: v1.39.2+k3s1
 39:           token: "dgvPMqHguyx5zBw3k9F7tgJ8syAQ="
 40:           extra_hosts: "[{{ hostvars[groups['cluster'][0]]['ansible_host'] }}]"
 41:           extra_server_args:
 42:             extra_agent_args: ""
```

PLAY RECAP \*\*\*\*\*  
k3s\_master\_host : ok=26 changed=18 unreachable=0 failed=0 skipped=19 rescued=0 ignored=1  
k3s\_worker\_host\_1 : ok=14 changed=8 unreachable=0 failed=0 skipped=9 rescued=0 ignored=1  
k3s\_worker\_host\_2 : ok=15 changed=9 unreachable=0 failed=0 skipped=9 rescued=0 ignored=1  
monitoring\_fe\_host : ok=56 changed=41 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

Hình 25. Cấu hình cho các server bằng Ansible

Sau khi xác định được thông tin các instance, cụ thể là public instance trong đó có cấu hình HA Proxy, sử dụng Cloudflare để mapping domain với địa chỉ IP public của public instance và dùng HA Proxy để reverse proxy đến backend server tương ứng.

The screenshot shows the Cloudflare DNS management interface for the domain **th1enlm02.live**. At the top, there are navigation links for **DNS Setup: Full**, **Import and Export**, and **Dashboard Display Settings**. Below the header, there is a search bar labeled **Search DNS Records** with a placeholder **Q**, and buttons for **Add filter**, **Search**, and **Add record**.

Type	Name	Content	Proxy status	TTL	Actions
A	th1enlm02.live	23.21.167.203	DNS only	Auto	Edit
CNAME	application	th1enlm02.live	DNS only	Auto	Edit
CNAME	auth-api	th1enlm02.live	DNS only	Auto	Edit
CNAME	class-api	th1enlm02.live	DNS only	Auto	Edit
CNAME	grafana	th1enlm02.live	DNS only	Auto	Edit
CNAME	icinga	th1enlm02.live	DNS only	Auto	Edit
CNAME	lecturer-api	th1enlm02.live	DNS only	Auto	Edit
CNAME	loki-cluster	th1enlm02.live	DNS only	Auto	Edit
CNAME	loki	th1enlm02.live	DNS only	Auto	Edit
CNAME	prometheus-cluster	th1enlm02.live	DNS only	Auto	Edit
CNAME	prometheus	th1enlm02.live	DNS only	Auto	Edit
CNAME	student-api	th1enlm02.live	DNS only	Auto	Edit

*Hình 26. Mapping domain bằng Cloudflare*

Cấu hình HA Proxy lắng nghe trên port 80 và 443. Trong đó chứa thông tin các ACL và backend tương ứng cho domain của từng server đã triển khai. Default backend sẽ trỏ vào backend chứa thông tin các worker node của cụm K3s, trong đó mặc định K3s sử dụng ingress controller là Traefik nên port sẽ lắng nghe trên các worker node là port của Traefik được expose bởi vì các service của ứng dụng sử dụng các ingress để xác định các rule chứa thông tin host là domain để truy cập được từ bên ngoài.

```
frontend https-in
  bind :80
  bind :443 ssl crt /etc/haproxy/certs

  mode http
  acl is_cors_preflight method OPTIONS
  http-response set-header Access-Control-Allow-Origin https://application.thienlm02.live
  http-response set-header Access-Control-Allow-Methods "GET, POST, PUT, DELETE, OPTIONS"
  http-response set-header Access-Control-Allow-Headers "Content-Type, Authorization, X-Theme, Client-Id, Client-Secret, fineract-platform-tenantid"
  http-response set-header Access-Control-Max-Age "3600" if is_cors_preflight

  acl ACL_prometheus hdr(host) -i prometheus.thienlm02.live www.prometheus.thienlm02.live
  use_backend backend_prometheus if ACL_prometheus

  acl ACL_loki hdr(host) -i loki.thienlm02.live www.loki.thienlm02.live
  use_backend backend_loki if ACL_loki

  acl ACL_grafana hdr(host) -i grafana.thienlm02.live www.grafana.thienlm02.live
  use_backend backend_grafana if ACL_grafana

  acl ACL_icinga hdr(host) -i icinga.thienlm02.live www.icinga.thienlm02.live
  use_backend backend_icinga if ACL_icinga

  default_backend k3s_cluster
  redirect scheme https code 301 if !{ ssl_fc }

backend k3s_cluster
  #server master_node_http 192.168.110.10:31930 check
  server worker_node1_http 192.168.110.11:31930 check
  server worker_node2_http 192.168.110.12:31930 check

backend backend_prometheus
  server prometheus 192.168.120.21:9090 check

backend backend_loki
  server loki 192.168.120.21:3100 check

backend backend_grafana
  server grafana 192.168.120.20:3000 check

backend backend_icinga
  server icinga 192.168.120.20:80 check
ubuntu@ip-192-168-130-36:/etc/haproxy$ |
```

Hình 27. Cấu hình HA Proxy

Ngoài ra sau khi cụm K3s đã được triển khai, đảm bảo KUBECONFIG đã được cấu hình và OpenVPN đang bật để có thể kết nối đến API server của cụm và thực hiện các thao tác liên quan.

Hình 28. Thông tin cấu hình để kết nối đến cum K3s

Kubernetes Cluster Status											
Name	Status	Roles	Age	Version	Internal-IP	External-IP	OS-Image	Kernel-Version	Container-Runtime	Annotations	
ip-192-168-110-10	Ready	control-plane,etccd,master	3d18h	v1.30.2+k3s1	192.168.110.10	<none>	Ubuntu 20.04.6 LTS	5.15.0-1071-aws	containerd://1.7.17-k3s1		
ip-192-168-110-11	Ready	<none>	3d18h	v1.30.2+k3s1	192.168.110.11	<none>	Ubuntu 20.04.6 LTS	5.15.0-1071-aws	containerd://1.7.17-k3s1		
ip-192-168-110-12	Ready	<none>	3d18h	v1.30.2+k3s1	192.168.110.12	<none>	Ubuntu 20.04.6 LTS	5.15.0-1071-aws	containerd://1.7.17-k3s1		

Hình 29. Kiểm tra thông tin các node của cụm K3s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
traefik	LoadBalancer	10.43.152.97	192.168.110.10,192.168.110.11,192.168.110.12	80:31930/TCP,443:30514/TCP	3d18h

Hình 30. Lấy thông tin port của Traefik thông qua service

### 3.3.2. Triển khai ứng dụng

Chuẩn bị source code của ứng dụng và file docker-compose.yml trong đó chứa thông tin của các service sau đó tiến hành build thành các image bằng Docker.

The screenshot shows the VS Code interface with the following details:

- EXPLORER**: Shows the project structure for "SIMPLE-MICROSERVICES-APPLICATION".
- OPEN EDITORS**: Shows the "docker-compose.yml" file.
- docker-compose.yml** content (partial):

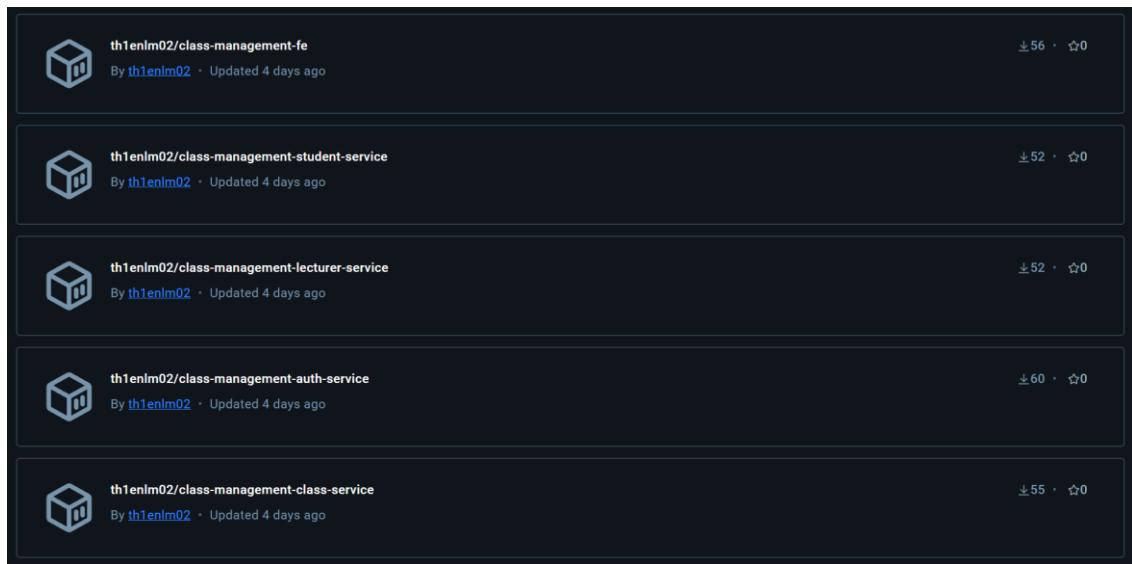
```
version: '3.8'
services:
  auth-service-mysql:
    image: mysql:8.3.0
    container_name: auth-service-mysql-container
    restart: on-failure
    ports:
      - "3305:3306"
    networks:
      - backend-network
    environment:
      - MYSQL_DATABASE=${MYSQL_DATABASE_AUTH}
      - MYSQL_USER=${MYSQL_USER}
      - MYSQL_PASSWORD=${MYSQL_PASSWORD}
      - MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASSWORD}
      - MYSQL_HOST=auth-service-mysql-container
    volumes:
      - auth-service-mysql-data:/var/lib/mysql
      - ./class-management-auth-service/db/init.sql:/docker-entrypoint-initdb.d/init.sql:ro
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:3306"]
      interval: 60s
      timeout: 20s
      retries: 5
      start_period: 30s
  student-service-mysql:
    image: mysql:8.3.0
    container_name: student-service-mysql-container
    restart: on-failure
```

- TERMINAL**: Shows the command "t-fpt :: argocd" and a list of Docker images:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
thienlm02/class-management-fe	latest	4cf95694d60c	4 days ago	503MB
thienlm02/class-management-auth-service	latest	c3daaf4153d7	4 days ago	146MB
thienlm02/class-management-class-service	latest	a39b9acfbac	5 days ago	584MB
thienlm02/class-management-student-service	latest	8a5486a80bf0	5 days ago	486MB
thienlm02/class-management-lecturer-service	latest	2fe1f7b73b67	5 days ago	1.08GB

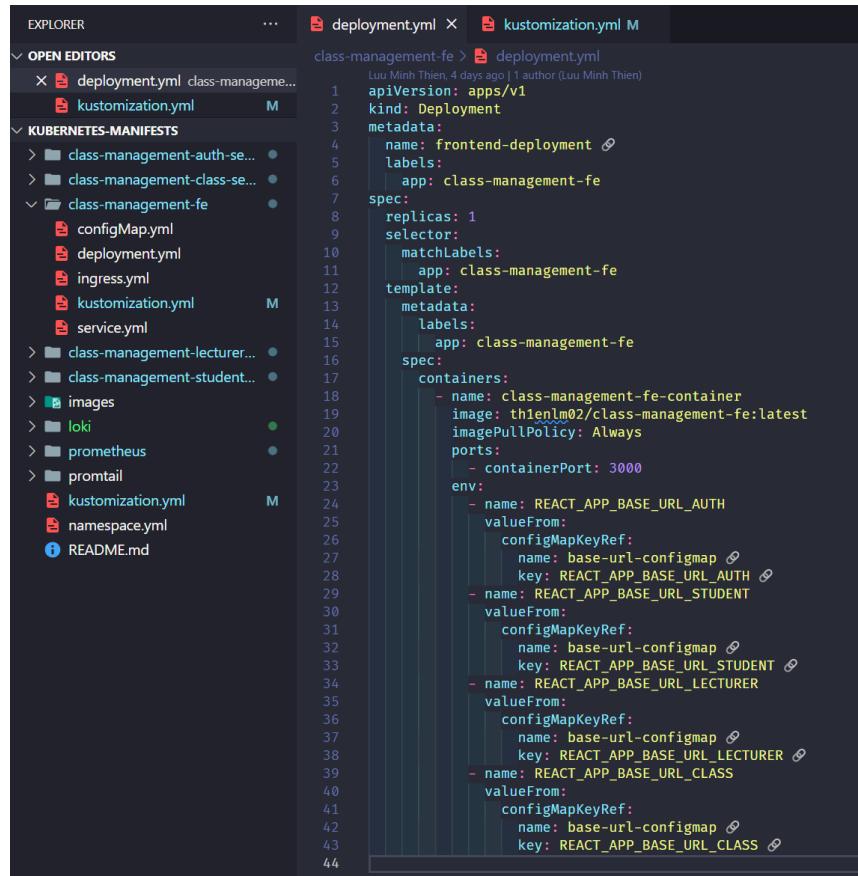
Hình 31. Build các service thành các image

Sau đó, chạy lệnh 'docker-compose push' để push các image lên Docker Hub.



Hình 32. Docker Hub chứa các image của ứng dụng

Chuẩn bị các file manifest triển khai lên cụm K3s, để dễ dàng quản lý từng module hóa các service thành các thư mục chứa các file manifest cho từng service.



```

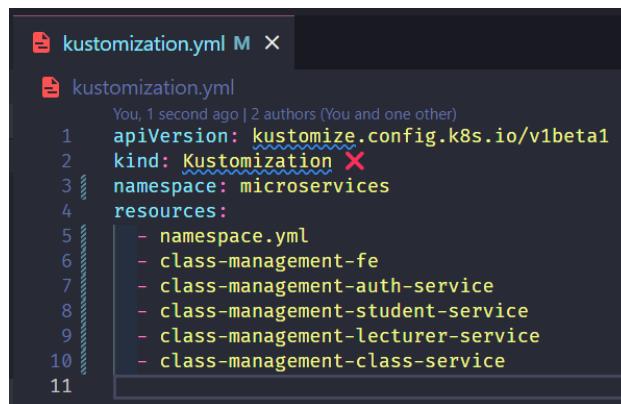
EXPLORER
OPEN EDITORS
KUBERNETES-MANIFESTS
deployment.yaml
kustomization.yml M
class-management-fe
configMap.yaml
deployment.yaml
ingress.yaml
kustomization.yml M
service.yaml
class-management-auth-service
class-management-class-service
class-management-student-service
class-management-lecturer-service
images
loki
prometheus
promtail
kustomization.yml M
namespace.yaml
README.md

deployment.yaml > deployment.yml
Luu Minh Thien, 4 days ago | 1 author (Luu Minh Thien)
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: frontend-deployment
5   labels:
6     app: class-management-fe
7 spec:
8   replicas: 1
9   selector:
10    matchLabels:
11      app: class-management-fe
12   template:
13     metadata:
14       labels:
15         app: class-management-fe
16   spec:
17     containers:
18       - name: class-management-fe-container
19         image: thienlm02/class-management-fe:latest
20         imagePullPolicy: Always
21       ports:
22         - containerPort: 3000
23       env:
24         - name: REACT_APP_BASE_URL_AUTH
25           valueFrom:
26             configMapKeyRef:
27               name: base-url-configmap
28               key: REACT_APP_BASE_URL_AUTH
29         - name: REACT_APP_BASE_URL_STUDENT
30           valueFrom:
31             configMapKeyRef:
32               name: base-url-configmap
33               key: REACT_APP_BASE_URL_STUDENT
34         - name: REACT_APP_BASE_URL_LECTURER
35           valueFrom:
36             configMapKeyRef:
37               name: base-url-configmap
38               key: REACT_APP_BASE_URL_LECTURER
39         - name: REACT_APP_BASE_URL_CLASS
40           valueFrom:
41             configMapKeyRef:
42               name: base-url-configmap
43               key: REACT_APP_BASE_URL_CLASS
44

```

Hình 33. Các file manifest triển khai lên cụm K3s

Triển khai các service lên cụm K3s, sử dụng Kustomization để dễ dàng quản lý và tổ chức các tài nguyên cần triển khai trong file kustomization.yaml và có thể apply trực tiếp với kubectl với tùy chọn -k và chỉ định đường dẫn chứa file.



```

kustomization.yaml M X
kustomization.yaml
You, 1 second ago | 2 authors (You and one other)
1 apiVersion: kustomize.config.k8s.io/v1beta1
2 kind: Kustomization X
3 namespace: microservices
4 resources:
5   - namespace.yaml
6   - class-management-fe
7   - class-management-auth-service
8   - class-management-student-service
9   - class-management-lecturer-service
10  - class-management-class-service
11

```

Hình 34. Nội dung file kustomization.yaml định nghĩa các resource sẽ triển khai

Kết quả lệnh kubectl get pods -o yaml										
NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES		
auth-deployment-6b968bc45-tw7lf	1/1	Running	4 (51m ago)	2d23h	10.42.1.19	ip-192-168-110-11	<none>	<none>		
auth-mysql-deployment-f9c4bf4d5-v7wld	1/1	Running	13 (51m ago)	4d11h	10.42.1.18	ip-192-168-110-11	<none>	<none>		
class-deployment-6977cc845b-pv68r	1/1	Running	5 (51m ago)	2d23h	10.42.2.208	ip-192-168-110-12	<none>	<none>		
class-postgresql-deployment-7b8f5cf8d-vzpw5	1/1	Running	4 (51m ago)	2d23h	10.42.1.16	ip-192-168-110-11	<none>	<none>		
frontend-deployment-74477dcc99-cs8kv	1/1	Running	4 (51m ago)	2d23h	10.42.1.23	ip-192-168-110-11	<none>	<none>		
lecturer-deployment-7645d94cfc-2qzjx	1/1	Running	4 (51m ago)	2d23h	10.42.1.12	ip-192-168-110-11	<none>	<none>		
lecturer-mongodb-deployment-66f6b697b4-skmkc	1/1	Running	13 (51m ago)	4d11h	10.42.1.11	ip-192-168-110-11	<none>	<none>		
student-deployment-f85fd4965-vkj36	1/1	Running	4 (51m ago)	2d23h	10.42.1.17	ip-192-168-110-11	<none>	<none>		
student-mysql-deployment-98666fc68-h8kvd	1/1	Running	6 (51m ago)	2d23h	10.42.1.13	ip-192-168-110-11	<none>	<none>		
Kết quả lệnh kubectl get svc -o yaml										
NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE	SELECTOR				
auth-service	ClusterIP	10.43.62.185	<none>	3077/TCP	4d11h	app=class-management-auth-service				
auth-service-mysql	ClusterIP	10.43.183.104	<none>	3306/TCP	4d11h	app=auth-service-mysql				
class-service	ClusterIP	10.43.2.94	<none>	8000/TCP	4d11h	app=class-management-class-service				
class-service-postgresql	ClusterIP	10.43.185.207	<none>	5432/TCP	4d11h	app=class-service-postgresql				
frontend-service	ClusterIP	10.43.190.214	<none>	3000/TCP	4d11h	app=class-management-fe				
lecturer-service	ClusterIP	10.43.44.204	<none>	5000/TCP	4d11h	app=class-management-lecturer-service				
lecturer-service-mongodb	ClusterIP	10.43.128.172	<none>	27017/TCP	4d11h	app=lecturer-service-mongodb				
student-service	ClusterIP	10.43.139.131	<none>	8080/TCP	4d11h	app=class-management-student-service				
student-service-mysql	ClusterIP	10.43.10.140	<none>	3307/TCP	4d11h	app=student-service-mysql				
Kết quả lệnh kubectl get svc -o yaml										
NAME	CLASS	HOSTS	ADDRESS				PORTS	AGE		
auth-service-ingress	traefik	auth-api.thienlm02.live	192.168.110.10,192.168.110.11,192.168.110.12	80	4d11h					
class-service-ingress	traefik	class-api.thienlm02.live	192.168.110.10,192.168.110.11,192.168.110.12	80	4d11h					
frontend-ingress	traefik	application.thienlm02.live	192.168.110.10,192.168.110.11,192.168.110.12	80	4d11h					
lecturer-service-ingress	traefik	lecturer-api.thienlm02.live	192.168.110.10,192.168.110.11,192.168.110.12	80	4d11h					
student-service-ingress	traefik	student-api.thienlm02.live	192.168.110.10,192.168.110.11,192.168.110.12	80	4d11h					

Hình 35. Kiểm tra thông tin resource của ứng dụng sau khi triển khai thành công

### 3.4. Cấu hình giám sát cho hệ thống

#### 3.4.1. Prometheus trên monitoring instance

Thông tin cấu hình của Prometheus tại đường dẫn “/etc/prometheus” chứa các tệp cấu hình quan trọng và các thư mục hỗ trợ cho việc cấu hình và quản lý Prometheus.

```
root@ip-192-168-120-21:/etc/prometheus# ll
total 36
drwxrwx--- 7 root prometheus 4096 Nov 13 09:12 .
drwxr-xr-x 120 root root      4096 Nov 14 06:41 ..
drwxr-xr-x  2 root root      4096 Nov   9 15:33 console_libraries/
drwxr-xr-x  2 root root      4096 Nov   9 15:34 consoles/
drwxrwx--- 2 root prometheus 4096 Nov   9 13:58 file_sd/
-rw-r----- 1 root prometheus 1311 Nov 13 09:12 prometheus.yml
drwxrwx--- 2 root prometheus 4096 Nov   9 15:35 rules/
drwxrwx--- 2 root prometheus 4096 Nov   9 13:58 scrapes/
-rw-r----- 1 root prometheus 133 Nov   9 15:35 web.yml
root@ip-192-168-120-21:/etc/prometheus#
```

Hình 36. Thư mục cấu hình của Prometheus

Phần cấu hình cho Prometheus định nghĩa thông tin và các thông số để thu thập dữ liệu từ các nguồn scrape nằm trong file prometheus.yml. Trong đó, mục targets sẽ chứa thông tin các server đã cài đặt Node Exporter bao gồm địa chỉ IP và port mà Node Exporter đã expose (ở đây là port 9200).

```

scrape_configs:
  - job_name: prometheus
    metrics_path: /metrics
    scrape_interval: 5m
    basic_auth:
      username: "admin"
      password: "admin@group01"
    static_configs:
      - targets:
          - 192.168.120.21:9090
  - job_name: node
    file_sd_configs:
      - files:
          - /etc/prometheus/file_sd/node.yml

  - job_name: node_exporter
    scrape_interval: 15s
    basic_auth:
      username: "admin"
      password: "admin@group01"
    static_configs:
      - targets: [ '192.168.130.30:9200' ]
        labels:
          server: group01-bastion_host
      - targets: [ '192.168.120.20:9200' ]
        labels:
          server: group01-monitoring_fe_host
      - targets: [ '192.168.21:9200' ]
        labels:
          server: group01-monitoring_be_host
      - targets: [ '192.168.110.10:9200' ]
        labels:
          server: group01-k3s_master_host
      - targets: [ '192.168.110.11:9200' ]
        labels:
          server: group01-k3s_worker_host_1
      - targets: [ '192.168.110.12:9200' ]
        labels:
          server: group01-k3s_worker_host_2

  scrape_config_files:
    - scrapes/*
root@ip-192-168-120-21:/etc/prometheus# |

```

*Hình 37. Cấu hình scrape của Prometheus*

Ngoài ra, tệp web.yml chứa các cấu hình liên quan đến giao diện web của Prometheus, đối với tệp này sẽ cấu hình basic authentication để bảo mật kết nối đến server và mật khẩu phải dưới dạng hash.

```

root@ip-192-168-120-21:/etc/prometheus# cat web.yml
tls_server_config: {}
http_server_config: {}
basic_auth_users:
  admin: $2a$12$HR07l5oVaH/HGntovg4rh.bFL6JSzu6kzlllahW3juLwhkIg9jDMO

```

*Hình 38. Cấu hình web của Prometheus*

Hơn nữa, có thể kiểm tra lại tệp cấu hình systemd của Prometheus tại đường dẫn “/etc/systemd/system/prometheus.service” để đảm bảo các tùy chọn được cấu hình đúng. Trong đó, đối với các tùy chọn cho phần lưu trữ chứa đường dẫn lưu trữ time-series database, thời gian lưu trữ (365d tức là 365 ngày), kích thước lưu trữ (0 tức là không giới hạn kích thước lưu trữ).

```

root@ip-192-168-120-21:/etc/prometheus# cat /etc/systemd/system/prometheus.service
#
# Ansible managed
#
[Unit]
Description=Prometheus
After=network-online.target
Requires=local-fs.target
After=local-fs.target

[Service]
Type=simple
Environment="GOMAXPROCS=2"
User=prometheus
Group=prometheus
ExecReload=/bin/kill -HUP $MAINPID
ExecStart=/usr/local/bin/prometheus \
--storage.tsdb.path=/var/lib/prometheus \
--storage.tsdb.retention.time=365d \
--storage.tsdb.retention.size=0 \
--web.config.file=/etc/prometheus/web.yml \
--web.console.libraries=/etc/prometheus/console_libraries \
--web.console.templates=/etc/prometheus/consoles \
--web.listen-address=0.0.0.0:9090 \
--web.external-url= \
--config.file=/etc/prometheus/prometheus.yml

CapabilityBoundingSet=CAP_SET_UID
LimitNOFILE=65000
LockPersonality=true
NoNewPrivileges=true
MemoryDenyWriteExecute=true
PrivateDevices=true
PrivateTmp=true
ProtectHome=true
RemoveIPC=true
RestrictSUIDSGID=true
#SystemCallFilter=@signal @timer

ReadWriteDirectories=/var/lib/prometheus

ProtectSystem=full

SyslogIdentifier=prometheus
Restart=always
TimeoutStopSec=600s

[Install]

```

Hình 39. Tệp systemd của Prometheus

Prometheus lưu trữ dữ liệu thời gian thực (time-series data) theo định dạng tối ưu để hỗ trợ truy vấn nhanh và hiệu quả. Đường dẫn chứa thông tin liên quan đến lưu trữ của Prometheus lưu trữ tại thư mục “/var/lib/prometheus”.

```

root@ip-192-168-120-21:~# ll /var/lib/prometheus/
total 64
drwxr-xr-x 11 prometheus prometheus 4096 Nov 14 05:45 .
drwxr-xr-x 55 root      root        4096 Nov 14 06:40 ..
drwxr-xr-x  3 prometheus prometheus 4096 Nov 12 07:00 01JCFJR686KJR2ZNE7E313NEZV/
drwxr-xr-x  3 prometheus prometheus 4096 Nov 13 07:44 01JCJ7Q2VQCCFC76TCVA5HGMCY/
drwxr-xr-x  3 prometheus prometheus 4096 Nov 13 10:44 01JCJ30NGPJP5S3WQK04NC2KNP/
drwxr-xr-x  3 prometheus prometheus 4096 Nov 13 11:00 01JCJWA0VBQDBPX4E96S2DAH/
drwxr-xr-x  3 prometheus prometheus 4096 Nov 13 11:00 01JCJJWB5ZMKBRW0W9KM9DRE3N/
drwxr-xr-x  3 prometheus prometheus 4096 Nov 14 03:45 01JCMCE5RJTS1CWY4AWDWQE6BR/
drwxr-xr-x  3 prometheus prometheus 4096 Nov 14 05:45 01JCMK9ZZVN35F9DVKYN9YNTK/
drwxr-xr-x  2 prometheus prometheus  0 Nov 14 03:45 chunks_head/
-rw-r--r--  1 prometheus prometheus    0 Nov 14 05:48 queries.active
-rw-r--r--  1 prometheus prometheus 20001 Nov 14 06:45 queries.active
drwxr-xr-x  3 prometheus prometheus 4096 Nov 14 05:45 wal/

```

Hình 40. Cấu trúc thư mục lưu trữ của Prometheus

Cấu trúc lưu trữ của Prometheus gồm các thành phần chính sau:

- Data blocks: Prometheus chia dữ liệu thành các khối, thường có thời lượng là 2 giờ. Mỗi khối là một thư mục riêng biệt. Các khối có tên dạng các ID chứa các tệp con:
  - index: Chứa chỉ mục của các series và giá trị timestamp, giúp tìm kiếm và truy vấn dữ liệu nhanh chóng.
  - meta.json: Tệp chứa thông tin meta về khối dữ liệu, như thời gian bắt đầu và kết thúc của khối, số lượng series.
  - tombstones: Chứa thông tin về các dữ liệu đã bị xóa hoặc không còn hợp lệ.
  - chunks: Dữ liệu trong mỗi khối được chia thành các chunks nén để giảm thiểu dung lượng. Thư mục chunks/ lưu trữ các chunk, với mỗi chunk là một tệp riêng chứa dữ liệu thô được nén lại để truy vấn nhanh và tiết kiệm dung lượng.

```
root@ip-192-168-120-21:~# ll /var/lib/prometheus/01JCFJR686KJR2ZNE7E313NEZV/
total 7344
drwxr-xr-x  3 prometheus prometheus    4096 Nov 12 07:00 ./
drwxr-xr-x 11 prometheus prometheus    4096 Nov 14 05:45 ../
drwxr-xr-x  2 prometheus prometheus    4096 Nov 12 07:00 chunks/
-rw-r--r--  1 prometheus prometheus 7495807 Nov 12 07:00 index
-rw-r--r--  1 prometheus prometheus   1264 Nov 12 07:00 meta.json
-rw-r--r--  1 prometheus prometheus      9 Nov 12 07:00 tombstones
```

Hình 41. Thư mục chứa các file liên quan của mỗi data block

```
root@ip-192-168-120-21:~# ll /var/lib/prometheus/01JCFJR686KJR2ZNE7E313NEZV/chunks/
total 66936
drwxr-xr-x  2 prometheus prometheus    4096 Nov 12 07:00 ./
drwxr-xr-x  3 prometheus prometheus    4096 Nov 12 07:00 ../
-rw-r--r--  1 prometheus prometheus 68528780 Nov 12 07:00 000001
```

Hình 42. File chunk của data block

- WAL (Write-Ahead Log): Thư mục WAL lưu trữ dữ liệu gần đây và đóng vai trò là bộ nhớ đệm, cho phép Prometheus lưu tạm thời các thay đổi trước khi chúng được ghi vào khối dữ liệu vĩnh viễn và được dùng để khôi phục dữ liệu trong trường hợp Prometheus gặp sự cố.

```

root@ip-192-168-120-21:/var/lib/prometheus/wal# ll
total 47440
drwxr-xr-x 3 prometheus prometheus    4096 Nov 14 07:00 ../
drwxr-xr-x 7 prometheus prometheus    4096 Nov 14 07:00 ...
-rw-r--r-- 1 prometheus prometheus 27426816 Nov 14 05:45 00000044
-rw-r--r-- 1 prometheus prometheus 16941056 Nov 14 07:00 00000045
-rw-r--r-- 1 prometheus prometheus 4193189 Nov 14 07:18 00000046
drwxr-xr-x 2 prometheus prometheus    4096 Nov 14 07:00 checkpoint.00000043/
root@ip-192-168-120-21:/var/lib/prometheus/wal# |

```

Hình 43. WAL (Write-Ahead Log)

- chunk\_head: Thư mục lưu các chunk hiện tại trong bộ nhớ mà chưa được chuyển thành khối dữ liệu. Dữ liệu này sẽ chuyển vào các khối 2 giờ khi đạt giới hạn về thời gian.

```

root@ip-192-168-120-21:/var/lib/prometheus/chunks_head# ll
total 3660
drwxr-xr-x 2 prometheus prometheus    4096 Nov 14 07:01 ../
drwxr-xr-x 7 prometheus prometheus    4096 Nov 14 07:00 ...
-rw-r--r-- 1 prometheus prometheus 3608376 Nov 14 07:01 000002
-rw-r--r-- 1 prometheus prometheus 131072 Nov 14 07:01 000003
root@ip-192-168-120-21:/var/lib/prometheus/chunks_head# |

```

Để đảm bảo Prometheus lấy được metric, kiểm tra Node Exporter trên các target server đang hoạt động và expose đúng port.

```

root@ip-192-168-120-21:~# cat /etc/systemd/system/node_exporter.service
#
# Ansible managed
#
[Unit]
Description=Prometheus Node Exporter
After=network-online.target

[Service]
Type=simple
User=node-exp
Group=node-exp
ExecStart=/usr/local/bin/node_exporter \
    '--collector.systemd' \
    '--collector.textfile' \
    '--collector.textfile.directory=/var/lib/node_exporter' \
    '--web.listen-address=0.0.0.0:9200' \
    '--web.telemetry-path=/metrics'

SyslogIdentifier=node_exporter
Restart=always
RestartSec=1
StartLimitInterval=0

ProtectHome=yes
NoNewPrivileges=yes

ProtectSystem=full

[Install]
WantedBy=multi-user.target
root@ip-192-168-120-21:~# |

```

Hình 44. Tệp systemd của Node Exporter

```

ubuntu@ip-192-168-130-30:/etc/node_exporter$ systemctl status node_exporter.service
● node_exporter.service - Prometheus Node Exporter
   Loaded: loaded (/etc/systemd/system/node_exporter.service; enabled; vendor preset: enabled)
     Active: active (running) since Thu 2024-11-14 03:45:55 UTC; 1h 17min ago
       Main PID: 517 (node_exporter)
          Tasks: 4 (limit: 2333)
         Memory: 22.7M
        CGroup: /system.slice/node_exporter.service
                  └─517 /usr/local/bin/node_exporter --collector.systemd --collector.textfile --collector.textfile.directory=/var/lib/node_exporter --web.listen-address=0.0.0.0:9200

Nov 14 03:45:56 ip-192-168-130-30 node_exporter[517]: ts=2024-11-14T03:45:56.142Z caller=node_exporter.go:118 level=info collector=time
Nov 14 03:45:56 ip-192-168-130-30 node_exporter[517]: ts=2024-11-14T03:45:56.142Z caller=node_exporter.go:118 level=info collector:time
Nov 14 03:45:56 ip-192-168-130-30 node_exporter[517]: ts=2024-11-14T03:45:56.142Z caller=node_exporter.go:118 level=info collector:udp_queues
Nov 14 03:45:56 ip-192-168-130-30 node_exporter[517]: ts=2024-11-14T03:45:56.142Z caller=node_exporter.go:118 level=info collector:uname
Nov 14 03:45:56 ip-192-168-130-30 node_exporter[517]: ts=2024-11-14T03:45:56.142Z caller=node_exporter.go:118 level=info collector:vmsstat
Nov 14 03:45:56 ip-192-168-130-30 node_exporter[517]: ts=2024-11-14T03:45:56.142Z caller=node_exporter.go:118 level=info collector=watchdog
Nov 14 03:45:56 ip-192-168-130-30 node_exporter[517]: ts=2024-11-14T03:45:56.142Z caller=node_exporter.go:118 level=info collector=xfs
Nov 14 03:45:56 ip-192-168-130-30 node_exporter[517]: ts=2024-11-14T03:45:56.142Z caller=node_exporter.go:118 level=info collector:zfs
Nov 14 03:45:56 ip-192-168-130-30 node_exporter[517]: ts=2024-11-14T03:45:56.163Z caller=tls_config.go:313 level=info msg="Listening on" address=[::]:9200
Nov 14 03:45:56 ip-192-168-130-30 node_exporter[517]: ts=2024-11-14T03:45:56.163Z caller=tls_config.go:316 level=info msg="TLS is disabled." http2=false address=[::]:9200
ubuntu@ip-192-168-130-30:/etc/node_exporter$ 

```

Hình 45. Kiểm tra trạng thái Node Exporter

Có thể truy cập vào domain của Prometheus tại đường dẫn <https://prometheus.th1enlm02.live> để kiểm tra trạng thái của các target server:

Targets					
All scrape pools	All	Unhealthy	Collapse All	Filter by endpoint or labels	
<b>node_exporter (6/6 up)</b>					
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
<a href="http://192.168.110.11:9200/metrics">http://192.168.110.11:9200/metrics</a>	UP	instance="192.168.110.11:9200" job="node exporter" server="group01-k3s-worker-host-1"	7.15s ago	107.049ms	
<a href="http://192.168.110.12:9200/metrics">http://192.168.110.12:9200/metrics</a>	UP	instance="192.168.110.12:9200" job="node exporter" server="group01-k3s-worker-host-2"	1.595s ago	85.603ms	
<a href="http://192.168.130.30:9200/metrics">http://192.168.130.30:9200/metrics</a>	UP	instance="192.168.130.30:9200" job="node exporter" server="group01-hastion-host"	1.633s ago	60.840ms	
<a href="http://192.168.120.20:9200/metrics">http://192.168.120.20:9200/metrics</a>	UP	instance="192.168.120.20:9200" job="node exporter" server="group01-monitoring-fs-host"	5.389s ago	69.802ms	
<a href="http://192.168.120.21:9200/metrics">http://192.168.120.21:9200/metrics</a>	UP	instance="192.168.120.21:9200" job="node exporter" server="group01-monitoring-lb-host"	10.74s ago	65.515ms	
<a href="http://192.168.110.10:9200/metrics">http://192.168.110.10:9200/metrics</a>	UP	instance="192.168.110.10:9200" job="node exporter" server="group01-k3s-master-host"	3.491s ago	77.848ms	
<b>prometheus (1/1 up)</b>					
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
<a href="http://192.168.120.21:9090/metrics">http://192.168.120.21:9090/metrics</a>	UP	instance="192.168.120.21:9090" job="prometheus"	26.363s ago	333.545ms	

Hình 46. Kiểm tra trạng thái các target trên Prometheus web UI

### 3.4.1.2. Prometheus trên cụm K3s

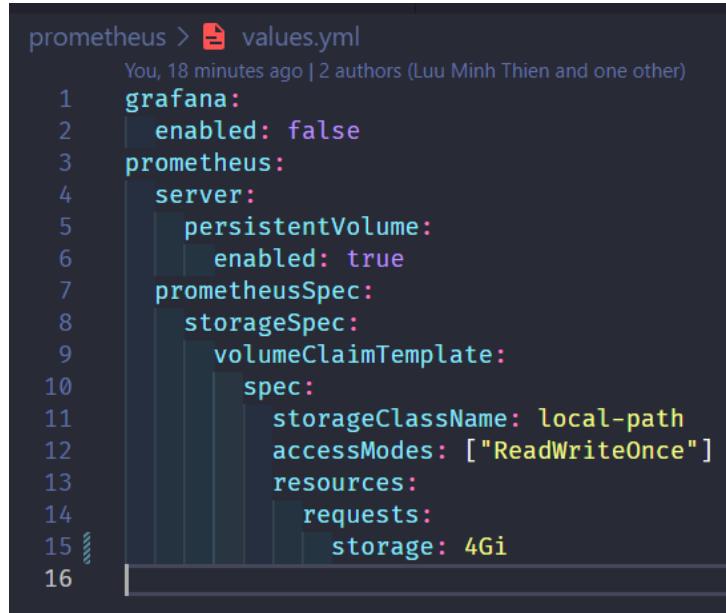
Sử dụng Helm để cài đặt Prometheus trên cụm K3s, chạy các lệnh sau để thêm và cập nhật kho lưu trữ của Prometheus:

```

helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo update prometheus-community

```

Tạo file values.yml để override lại cấu hình của Prometheus, trong đó enable cấu hình persistent volume và storage classname sẽ sử dụng local-path, tức là Prometheus sẽ lưu trữ dữ liệu ngay tại node đang chạy.



```
prometheus > values.yml
You, 18 minutes ago | 2 authors (Luu Minh Thien and one other)
1  grafana:
2    enabled: false
3  prometheus:
4    server:
5      persistentVolume:
6        enabled: true
7      prometheusSpec:
8        storageSpec:
9          volumeClaimTemplate:
10         spec:
11           storageClassName: local-path
12           accessModes: [ "ReadWriteOnce" ]
13           resources:
14             requests:
15               storage: 4Gi
16
```

Hình 47. Nội dung file values.yml của Prometheus

Sau khi có thông tin values cho Prometheus Helm chart thì chạy lệnh sau để triển khai Prometheus lên cụm K3s:

```
helm install --namespace monitoring prometheus-stack -f values.yml
prometheus-community/kube-prometheus-stack --create-namespace
```

Để có thể sử dụng domain truy cập vào Prometheus trên cụm K3s, tiến hành cấu hình Ingress cho Prometheus, trong đó định nghĩa rule khi truy cập vào domain đang mapping cho Prometheus trên cụm thì sẽ trả vào service có tên là prometheus-operated với port là 9090, nó là một service do Prometheus Operator tạo ra để quản lý và duy trì các pod của Prometheus.

```

prometheus > ⌂ ingress.yml
Luu Minh Thien, 4 days ago | 1 author (Luu Minh Thien)
1  apiVersion: networking.k8s.io/v1
2  kind: Ingress
3  metadata:
4    name: prometheus-ingress
5    labels:
6      name: prometheus-ingress
7    annotations:
8      ingress.kubernetes.io/auth-type: "basic"
9      ingress.kubernetes.io/auth-secret: "prometheus-basic-auth"
10   spec:
11     rules:
12       - host: prometheus-cluster.th1enlm02.live
13         http:
14           paths:
15             - pathType: Prefix
16               path: "/"
17               backend:
18                 service:
19                   name: prometheus-operated
20                   port:
21                     number: 9090
22

```

Hình 48. Nội dung file ingress.yml của Prometheus

Sau khi triển khai thành công, có thể kiểm tra thông tin các resource trên cụm K3s và truy cập vào domain để kiểm tra trạng thái của các target đã up chưa tại đường dẫn <https://prometheus-cluster.th1enlm02.live>.

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS G
<b>ATES</b>								
alertmanager-prometheus-stack-kube-prom-alertmanager-0	2/2	Running	26 (9m52s ago)	5d18h	10.42.0.204	ip-192-168-110-10	<none>	<none>
prometheus-prometheus-stack-kube-prom-prometheus-0	2/2	Running	24 (9m26s ago)	5d18h	10.42.2.8	ip-192-168-110-12	<none>	<none>
prometheus-stack-kube-prom-operator-69c57f884b-pw4lf	1/1	Running	14 (9m23s ago)	5d18h	10.42.1.126	ip-192-168-110-11	<none>	<none>
prometheus-stack-kube-state-metrics-6f44fb6dc6-mcjk2	1/1	Running	15 (9m52s ago)	5d18h	10.42.0.202	ip-192-168-110-10	<none>	<none>
prometheus-stack-prometheus-node-exporter-tbdc9	1/1	Running	12 (9m23s ago)	5d18h	192.168.110.11	ip-192-168-110-11	<none>	<none>
prometheus-stack-prometheus-node-exporter-vrfqb	1/1	Running	12 (9m26s ago)	5d18h	192.168.110.12	ip-192-168-110-12	<none>	<none>
prometheus-stack-prometheus-node-exporter-vznlml	1/1	Running	15 (9m52s ago)	5d18h	192.168.110.10	ip-192-168-110-10	<none>	<none>
<b>K8SVC</b> -o wide								
NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE	SELECTOR		
alertmanager-operated	ClusterIP	None	<none>	9093/TCP,9094/TCP,9094/UDP	5d18h	app.kubernetes.io/name=alertmanager		
prometheus-operated	ClusterIP	None	<none>	9090/TCP	5d18h	app.kubernetes.io/name=prometheus		
prometheus-stack-kube-prom-alertmanager	ClusterIP	10.43.14.4	<none>	9093/TCP,8080/TCP	5d18h	alertmanager=prometheus-stack-kube-prom-al		
ertmanager, app.kubernetes.io/name=alertmanager								
prometheus-stack-kube-prom-operator	ClusterIP	10.43.72.73	<none>	443/TCP	5d18h	app=kube-prometheus-stack-operator,release		
-prometheus-stack								
prometheus-stack-kube-prom-prometheus	ClusterIP	10.43.38.235	<none>	9090/TCP,8080/TCP	5d18h	app.kubernetes.io/name=prometheus,operator		
.prometheus.io/name=prometheus-stack-kube-prom-prometheus								
prometheus-stack-kube-state-metrics	ClusterIP	10.43.146.127	<none>	8080/TCP	5d18h	app.kubernetes.io/instance=prometheus-stac		
k,app.kubernetes.io/name=kube-state-metrics								
prometheus-stack-prometheus-node-exporter	ClusterIP	10.43.42.72	<none>	9100/TCP	5d18h	app.kubernetes.io/instance=prometheus-stac		
k,app.kubernetes.io/name=prometheus-node-exporter								
<b>INGING</b> -o wide								
NAME	CLASS	HOSTS	ADDRESS		PORTS	AGE		
prometheus-ingress	traefik	prometheus-cluster.th1enlm02.live	192.168.110.10,192.168.110.11,192.168.110.12		80	5d12h		

Hình 49. Kiểm tra thông tin các resource của Prometheus sau khi triển khai thành công

Hình 50. Kiểm tra trạng thái các target trên Prometheus web UI của cụm K3s

### 3.4.2. Grafana Loki

#### 3.4.2.1. Grafana Loki tên monitoring instance

File cấu hình Loki được lưu trữ tại đường dẫn “/etc/loki/config.yml” trong đó sẽ chứa các thông tin cấu hình cho Loki server, bao gồm:

- server: Thiết lập các thông số cho máy chủ Loki, bao gồm mức ghi log, địa chỉ IP và cổng lắng nghe cho giao diện HTTP và gRPC.
- query\_range: Cấu hình bộ nhớ đệm cho các kết quả truy vấn, giúp cải thiện hiệu suất truy vấn log.
- pattern\_ingester: Kích hoạt và thiết lập chức năng tiếp nhận log với các thuộc tính về pattern và vòng đời.
- storage\_config: Định nghĩa nơi và cách Loki lưu trữ dữ liệu log (index và chunks) bằng boltdb và filesystem:
  - boltdb giúp lưu trữ chỉ mục và cải thiện tốc độ tìm kiếm log.
  - filesystem sẽ lưu trữ dữ liệu log thu được trên thư mục hệ thống dưới dạng các file chunk tương tự Prometheus.

- schema\_config: Cấu hình schema, giúp Loki xác định cách lưu trữ và truy xuất dữ liệu theo thời gian. Trong đó loại cơ sở dữ liệu là tsdb (time-series database), sử dụng nơi lưu trữ là thư mục hệ thống để lưu trữ các đối tượng, phần period sẽ xác định thời gian lưu trữ các file chunk là 24 giờ.
- limits\_config: Đặt giới hạn về tài nguyên, như số lượng luồng log tối đa cho mỗi người dùng để tối ưu hóa hiệu suất.
- common: Thiết lập thông số chung cho instance Loki, bao gồm địa chỉ, thư mục lưu trữ và yếu tố sao lưu dữ liệu.

```

root@ip-192-168-120-21:/etc/loki# cat config.yml
target: all
auth_enabled: False
server:
  log_level: info
  http_listen_address: 0.0.0.0
  http_listen_port: 3100
  grpc_listen_port: 9096

query_range:
  results_cache:
    cache:
      embedded_cache:
        enabled: true
        max_size_mb: 100

pattern_ingester:
  enabled: true
  lifecycler:
    ring:
      kvstore:
        store: inmemory

storage_config:
  boltdb_shipper:
    active_index_directory: /tmp/loki/boltdb-shipper-active
    cache_location: /tmp/loki/boltdb-shipper-cache
  filesystem:
    directory: /tmp/loki/chunks

schema_config:
  configs:
    - from: 2020-10-24
      store: tsdb
      object_store: filesystem
      schema: v13
      index:
        prefix: index_
        period: 24h

limits_config:
  max_global_streams_per_user: 10000

common:
  instance_addr: 127.0.0.1
  path_prefix: /var/lib/loki
  replication_factor: 1
  ring:
    kvstore:
      store: inmemory

```

Hình 51. Nội dung file cấu hình của Loki

Đối với phần lưu trữ của Loki, thay vì lưu trữ log đầy đủ như các hệ thống khác, Loki lưu trữ dữ liệu dưới dạng chunks và chỉ mục tối thiểu. Cách thức lưu trữ log của Loki:

- Chunks dữ liệu log:
  - Log được chia thành chunks — khối dữ liệu nhỏ nén để tối ưu không gian lưu trữ.
  - Mỗi chunk chứa log từ một khoảng thời gian cố định (thường là 1 giờ, nhưng có thể cấu hình).
  - Các chunks được lưu dưới dạng tệp hoặc trong các object store (như S3, GCS) hoặc filesystem.
  - Cách hoạt động:
    - Khi một log đến Loki, nó sẽ được gắn nhãn (label).
    - Log được lưu thành log stream dựa trên nhãn của nó.
    - Mỗi dòng log sẽ tạo ra các chunks riêng biệt.
- Indexing:
  - Loki chỉ lập chỉ mục các nhãn và thời gian log.
  - Không lập chỉ mục toàn bộ nội dung log (như Elasticsearch) giúp Loki tiết kiệm dung lượng.
  - Chỉ mục được lưu trong kvstore: Dùng cơ chế lưu trữ “in-memory” hoặc lưu trữ ngoài (Consul, Etcd).
- Schema lưu trữ: Quy định cách Loki tổ chức dữ liệu:
  - Loại lưu trữ (filesystem, boltdb-shipper, S3).
  - Phân vùng theo thời gian (ví dụ: 1 ngày).
  - Định dạng chỉ mục (v11, v12, v13).

```

root@ip-192-168-120-21:/tmp/loki/chunks# ll
total 20
drwxr-xr-x  4 loki nogroup 4096 Nov 16 03:31 .
drwxr-xr-x  3 loki nogroup 4096 Nov 16 03:30 ..
drwxr-xr-x 18 loki nogroup 4096 Nov 16 04:06 fake/
drwxr-xr-x  4 loki nogroup 4096 Nov 16 03:46 index/
-rw-r--r--  1 loki nogroup 259 Nov 16 03:31 loki_cluster_seed.json
root@ip-192-168-120-21:/tmp/loki/chunks# ll index/
total 16
drwxr-xr-x  4 loki nogroup 4096 Nov 16 03:46 .
drwxr-xr-x  4 loki nogroup 4096 Nov 16 03:31 ..
drwxr-xr-x  3 loki nogroup 4096 Nov 16 03:50 index_20042/
drwxr-xr-x  3 loki nogroup 4096 Nov 16 03:50 index_20043/
root@ip-192-168-120-21:/tmp/loki/chunks# ll fake/
total 72
drwxr-xr-x 18 loki nogroup 4096 Nov 16 04:06 .
drwxr-xr-x  4 loki nogroup 4096 Nov 16 03:31 ..
drwxr-xr-x  2 loki nogroup 4096 Nov 16 03:31 1d549d17e2a57918/
drwxr-xr-x  2 loki nogroup 4096 Nov 16 03:37 3d3e86bd7a4f616/
drwxr-xr-x  2 loki nogroup 4096 Nov 16 03:31 3e2d1525814faa4/
drwxr-xr-x  2 loki nogroup 4096 Nov 16 04:06 4c2ce8634b4b16f2/
drwxr-xr-x  2 loki nogroup 4096 Nov 16 03:38 4e3e208c4f67c186/
drwxr-xr-x  2 loki nogroup 4096 Nov 16 04:06 60d7e3002d4c5bf8/
drwxr-xr-x  2 loki nogroup 4096 Nov 16 03:38 65ff084ecbf16b69/
drwxr-xr-x  2 loki nogroup 4096 Nov 16 03:37 71b6a10dfb44938b/
drwxr-xr-x  2 loki nogroup 4096 Nov 16 03:36 87d0acd16cc300fc/
drwxr-xr-x  2 loki nogroup 4096 Nov 16 03:37 a0f63d93c833a9b8/
drwxr-xr-x  2 loki nogroup 4096 Nov 16 04:05 bc5bb4eab7a06d48/
drwxr-xr-x  2 loki nogroup 4096 Nov 16 04:06 c1a6061c2aaa8a8a/
drwxr-xr-x  2 loki nogroup 4096 Nov 16 03:31 cdb7698a2a33c6a7/
drwxr-xr-x  2 loki nogroup 4096 Nov 16 03:36 d1bb146a3f36caee/
drwxr-xr-x  2 loki nogroup 4096 Nov 16 03:37 e9a0f84bc6a4a07c/
drwxr-xr-x  2 loki nogroup 4096 Nov 16 03:38 ef7bf6c7e0886659/
root@ip-192-168-120-21:/tmp/loki/chunks# |

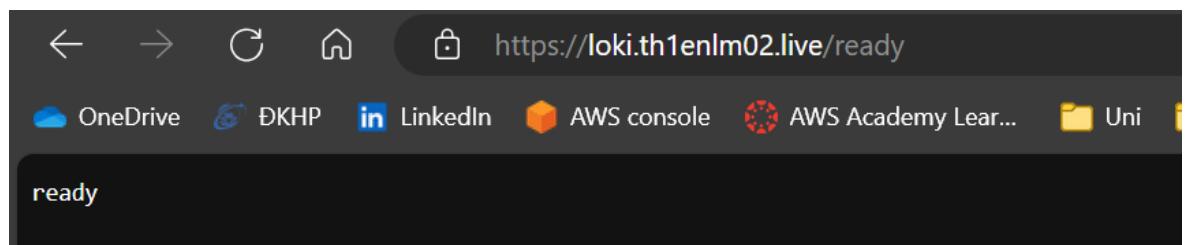
```

Hình 52. Cấu trúc thư mục lưu trữ của Loki

Về thời gian lưu trữ log của Loki, mặc định Loki không có thời gian lưu trữ log cố định, thời gian log được giữ lại phụ thuộc vào cách cấu hình hệ thống lưu trữ trong phần storage\_config và limits\_config của tệp cấu hình Loki. Nếu không cấu hình thời gian lưu trữ, Loki sẽ giữ log mãi mãi trên bộ lưu trữ được chỉ định cho đến khi tự tay xóa dữ liệu hoặc không gian lưu trữ bị đầy.

Truy cập vào đường dẫn sau để kiểm tra trạng thái sẵn sàng của Loki:

<https://loki.th1enlm02.live/ready>



Hình 53. Kiểm tra trạng thái sẵn sàng của Loki

Promtail sẽ cài đặt trên các instance cần lấy log, có nhiệm vụ thu thập và xử lý dữ liệu log sau đó gửi đến Loki. Cấu hình Promtail sẽ lấy log của HA Proxy tại thư mục lưu trữ log của nó: “/var/log/haproxy.log”

```
ubuntu@ip-192-168-130-30:/etc/promtail$ cat config.yml
server:
  http_listen_port: 9080
  http_listen_address: 0.0.0.0

positions:
  filename: /var/lib/promtail/positions.yaml

clients:
  - url: http://192.168.120.21:3100/loki/api/v1/push

scrape_configs:
  - job_name: haproxy
    pipeline_stages:
      - regex:
          expression: '^(\?P<timestamp>\w+ \d+ \d+:\d+:\d+) (\?P<host>[^ ]+) (\?P<process>[^ ]+)([\?P<pid>\d+]) (\?P<client_ip>[^ ]+)(\?P<client_port>\d+) \[(\?P<request_date>[^ ]+)\] (\?P<frontend>[^ ]+)- (\?P<backend>[^ ]+)/(\?P<server>[^ ]+) (\?P<time_tq>\d+)/(\?P<time_tw>\d+)/(\?P<time_tc>\d+)/(\?P<time_tr>\d+)/(\?P<time_tt>\d+) (\?P<http_status>\d+) (\?P<bytes>\d+) (\?P<captured_request_cookie>[^ ]+) (\?P<captured_response_cookie>[^ ]+) (\?P<termination_state>[^ ]+) (\?P<actconn>\d+)/(\?P<fcconn>\d+)/(\?P<bccn>\d+)/(\?P<srvcn>\d+)/(\?P<retries>\d+) (\?P<srv_queue>\d+)/(\?P<backend_queue>\d+) "(?P<method>[^ ]+) (\?P<uri>[^ ]+)" (\?P<protocol>[^ ]+)"
        - timestamp:
            source: timestamp
            format: Jan 02 15:04:05
        - labels:
            client_ip: null
            frontend: null
            backend: null
            server: null
            http_status: null
            method: null
            uri: null
        - output:
            source: message
      static_configs:
        - targets:
            - localhost
            labels:
              project: group01
              subnet: public-subnet
              hostname: bastion_host
              service_name: haproxy
              __path__: /var/log/haproxy.log
```

Hình 54. Cấu hình của Promtail

Các thành phần chính trong cấu hình Promtail:

- server: Promtail chạy một HTTP server trên cổng 9080 để cung cấp thông tin trạng thái và metrics.
- positions: Lưu trữ vị trí đọc cuối cùng từ tệp log để đảm bảo khi Promtail khởi động lại, nó không đọc lại dữ liệu đã thu thập. File lưu trạng thái “/var/lib/promtail/positions.yaml”

```
root@ip-192-168-130-30:/etc/promtail# cat /var/lib/promtail/positions.yaml
positions:
  /var/log/haproxy.log: "230505"
```

Hình 55. File lưu trạng thái vị trí đọc cuối cùng của tệp log

- clients: URL đích mà Promtail sẽ push log đến, ở đây sẽ là địa chỉ của Loki server.
- scrape\_configs: Cấu hình các nguồn log và cách xử lý dữ liệu log.

- pipeline\_stages:
  - regex: Sử dụng biểu thức chính quy để trích xuất các thông tin từ log.
  - timestamp: Xác định nguồn thời gian từ trường timestamp trong log.
  - labels: Gán nhãn từ các trường trích xuất để tổ chức log trong Loki. Các nhãn này thường được thêm vào hoặc thay đổi dựa trên nội dung của log và gán cho các log đã qua xử lý.
  - output: Chỉ giữ lại nội dung log thô trong trường message.
- static\_configs:
  - targets: Chỉ định nơi sẽ thu thập log, localhost tức là trên máy cục bộ.
  - labels: Các nhãn ở đây sẽ được gắn cho tất cả các log được thu thập từ một nguồn, giúp xác định nguồn gốc của log và các thuộc tính của dịch vụ hoặc hệ thống đang ghi log. Ngoài ra, trong đây có nhãn là ‘\_path\_’ chỉ định đường dẫn tệp log cần theo dõi và khi tệp log thay đổi, Promtail sẽ chỉ đọc phần mới.

```
root@ip-192-168-130-30:/etc/promtail# systemctl status promtail
● promtail.service - Promtail service
   Loaded: loaded (/etc/systemd/system/promtail.service; enabled; vendor preset: enabled)
   Active: active (running) since Sat 2024-11-16 03:30:53 UTC; 3h 42min ago
     Main PID: 7 (promtail)
        Tasks: 7 (limit: 2353)
       Memory: 80.6M
      CGroup: /system.slice/promtail.service
              └─ 467 /usr/bin/promtail -config.file /etc/promtail/config.yml

Nov 16 03:30:55 ip-192-168-130-30 promtail[467]: level=info ts=2024-11-16T03:30:55.290669326Z caller=reload.go:133 msg="Reloading configuration file" md5sum=831d8bb50575
Nov 16 03:30:55 ip-192-168-130-30 promtail[467]: level=info ts=2024-11-16T03:30:55.323811285Z caller=server.go:351 msg="server listening on addresses" http=[::]:9080 grpc=[::]:9051
Nov 16 03:30:55 ip-192-168-130-30 promtail[467]: level=info ts=2024-11-16T03:30:55.323965882Z caller=main.go:173 msg="Starting Promtail" version="(version=3.2.1, branch=releas
Nov 16 03:30:55 ip-192-168-130-30 promtail[467]: level=warn ts=2024-11-16T03:30:55.326857215Z caller=watchConfig.go:263 msg="enable watchConfig"
Nov 16 03:31:00 ip-192-168-130-30 promtail[467]: level=info ts=2024-11-16T03:31:00.321808235Z caller=filtargetmanager.go:372 msg="Adding target" key="/var/log/haproxy.log"
Nov 16 03:31:00 ip-192-168-130-30 promtail[467]: level=info ts=2024-11-16T03:31:00.331916475Z caller=filtarget.go:343 msg="watching new directory" directory=/var/log/haproxy.log
Nov 16 03:31:00 ip-192-168-130-30 promtail[467]: level=info ts=2024-11-16T03:31:00.332040176Z caller=log.go:168 level=info msg="Seeked /var/log/haproxy.log - 81Offset:546568 Whence:0"
Nov 16 03:31:00 ip-192-168-130-30 promtail[467]: level=info ts=2024-11-16T03:31:00.332967998Z caller=tailer.go:147 component=tailer msg="tail routine: started" path=/var/log/haproxy.log
Nov 16 03:31:00 ip-192-168-130-30 promtail[467]: ts=2024-11-16T03:31:00.583073851Z caller=log.go:168 level=info msg="Re-opening truncated file /var/log/haproxy.log ...
Nov 16 03:31:00 ip-192-168-130-30 promtail[467]: ts=2024-11-16T03:31:00.58315802Z caller=log.go:168 level=info msg="Successfully reopened truncated /var/log/haproxy.log"
```

Hình 56. Kiểm tra trạng thái Promtail

### 3.4.2.2. Grafana Loki trên cụm K3s

Sử dụng Helm để cài đặt Loki trên cụm K3s, chạy các lệnh sau để thêm và cập nhật kho lưu trữ của Grafana:

```
helm repo add grafana https://grafana.github.io/helm-charts
helm repo update grafana
```

Tạo file values.yml để override lại cấu hình của Loki, trong đó đặt số lượng bản sao cho dữ liệu là 3 giúp tăng cường khả năng chịu lỗi và đảm bảo tính sẵn sàng. Bật chế độ single-binary để Loki hoạt động dưới dạng một tiến trình duy nhất, giúp đơn giản hóa cấu hình và triển khai. Đối với cấu hình lưu trữ của Loki:

- Cấu hình schema cho dữ liệu của Loki:
  - Lưu trữ dữ liệu dưới dạng tsdb (time-series database).
  - Sử dụng S3 làm hệ thống lưu trữ đối tượng cho Loki.
  - Bật MinIO để sử dụng làm dịch vụ lưu trữ S3 nội bộ.

```
loki > values.yml
You, 11 seconds ago | 2 authors (You and one other)
1 loki:
2   auth-enabled: false
3   commonConfig:
4     replication_factor: 3
5   schemaConfig:
6     configs:
7       - from: "2024-04-01"
8         store: tsdb
9         object_store: s3
10        schema: v13
11        index:
12          prefix: loki_index_
13          period: 24h
14   singleBinary:
15     enabled: true
16
17 minio:
18   enabled: true
19
20 deploymentMode: SingleBinary
21
22 singleBinary:
23   replicas: 3
24
25 backend:
26   replicas: 0
27 read:
28   replicas: 0
29 write:
30   replicas: 0
```

Hình 57. Nội dung file values.yml của Loki

Sau khi có thông tin values cho Loki Helm chart thì chạy lệnh sau để triển khai Loki lên cụm K3s:

```
helm install loki grafana/loki --namespace loki --version 6.16.0 -f values.yaml
```

Để có thể sử dụng domain truy cập vào Loki trên cụm K3s, tiến hành cấu hình Ingress cho Loki, trong đó định nghĩa rule khi truy cập vào domain đang mapping cho Loki trên cụm thì sẽ trả vào service có tên là loki-gateway với port là 80, service này cung cấp một API HTTP để nhận dữ liệu log từ các client như Promtail.

```
loki > ingress.yaml
1  apiVersion: networking.k8s.io/v1
2  kind: Ingress
3  metadata:
4    name: loki-ingress
5    labels:
6      name: loki-ingress
7    annotations:
8      ingress.kubernetes.io/auth-type: "basic"
9      ingress.kubernetes.io/auth-secret: "loki-basic-auth"
10   spec:
11     rules:
12       - host: loki-cluster.thienlm02.live
13         http:
14           paths:
15             - pathType: Prefix
16               path: "/"
17               backend:
18                 service:
19                   name: loki-gateway
20                   port:
21                     number: 80
22
```

Hình 58. Nội dung file ingress.yaml của Loki

Tiếp theo, tạo file values.yml để cấu hình cho Promtail trong đó chỉ định đường dẫn đến loki-gateway để đẩy log cho Loki.

```
promtail > values.yaml
You, 23 seconds ago | 2 authors (Luu Minh Thien and one other)
1 config:
2   logLevel: info
3   serverPort: 3101
4   clients:
5     - url: http://loki-gateway.loki/api/v1/push
6       tenant_id: 'fake'
```

Hình 59. Nội dung file values.yaml của Promtail

Triển khai Promtail Helm chart với các giá trị trên lên cụm K3s bằng lệnh sau:

```
helm install promtail grafana/promtail --namespace loki -f
values.yaml
```

Sau khi triển khai thành công, có thể kiểm tra thông tin các resource trên cụm K3s.

```
> kubectl -o wide
NAME          READY   STATUS    RESTARTS   AGE     IP           NODE   NOMINATED NODE   READINESS GATES
loki-0        2/2     Running   10 (6m11s ago)   2d1h   10.42.0.219   ip-192-168-110-10   <none>   <none>
loki-1        2/2     Running   10 (5m52s ago)   2d1h   10.42.2.14    ip-192-168-110-12   <none>   <none>
loki-2        2/2     Running   10 (5m51s ago)   2d1h   10.42.1.145   ip-192-168-110-11   <none>   <none>
loki-canary-h94cb   1/1     Running   5 (5m52s ago)    2d1h   10.42.2.15    ip-192-168-110-12   <none>   <none>
loki-canary-hdxk8   1/1     Running   5 (6m11s ago)    2d1h   10.42.0.216   ip-192-168-110-10   <none>   <none>
loki-chunks-cache-jqvkk   1/1     Running   5 (5m51s ago)    2d1h   10.42.1.142   ip-192-168-110-11   <none>   <none>
loki-chunks-cache-0        2/2     Running   10 (6m11s ago)   2d1h   10.42.0.215   ip-192-168-110-10   <none>   <none>
loki-gateway-8d764c899-wrqss  1/1     Running   10 (5m43s ago)   2d1h   10.42.2.12     ip-192-168-110-12   <none>   <none>
loki-minio-0        1/1     Running   5 (5m51s ago)    2d1h   10.42.1.141   ip-192-168-110-11   <none>   <none>
loki-results-cache-0       2/2     Running   10 (5m52s ago)   2d1h   10.42.2.16     ip-192-168-110-12   <none>   <none>
promtail-57qf2      1/1     Running   5 (5m51s ago)    2d1h   10.42.1.146   ip-192-168-110-11   <none>   <none>
promtail-d94xq      1/1     Running   5 (6m11s ago)    2d1h   10.42.0.212   ip-192-168-110-10   <none>   <none>
promtail-f1j8c      1/1     Running   5 (5m52s ago)    2d1h   10.42.2.10     ip-192-168-110-12   <none>   <none>
> kubectl get svc -o wide
NAME            TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE     SELECTOR
loki             ClusterIP  10.43.157.71  <none>        3100/TCP   2d1h   app.kubernetes.io/component=single-binary,app.kubernetes.io/instance=loki,app.kubernetes.io/name=loki
loki-canary     ClusterIP  10.43.131.12  <none>        3500/TCP   2d1h   app.kubernetes.io/component=canary,app.kubernetes.io/instance=loki,app.kubernetes.io/name=loki
loki-chunks-cache   ClusterIP  None         <none>        11211/TCP, 9150/TCP  2d1h   app.kubernetes.io/component=memcached-chunks-cache,app.kubernetes.io/instance=loki
loki-gateway     ClusterIP  10.43.113.206 <none>        80/TCP     2d1h   app.kubernetes.io/component=gateway,app.kubernetes.io/instance=loki,app.kubernetes.io/name=loki
loki-ingress     ClusterIP  10.43.192.199 <none>        9000/TCP, 9001/TCP, 7946/TCP  2d1h   app=minio,release=loki
loki-minio-console ClusterIP  10.43.250.152 <none>        9000/TCP   2d1h   app=minio,release=loki
loki-minio-svc     ClusterIP  None         <none>        9000/TCP   2d1h   app=minio,release=loki
loki-results-cache ClusterIP  None         <none>        11211/TCP, 9150/TCP  2d1h   app.kubernetes.io/component=memcached-results-cache,app.kubernetes.io/instance=loki,app.kubernetes.io/name=loki
> kubectl get svc -o wide
NAME            CLASS   HOSTS          ADDRESS          PORTS   AGE
loki-ingress   traefik loki-cluster.th1enlm02.live 192.168.110.10,192.168.110.11,192.168.110.12 80   4d23h
```

Hình 60. Kiểm tra thông tin các resource của Loki sau khi triển khai thành công

### 3.4.3. Grafana

Đường dẫn file cấu hình Grafana “/etc/grafana/grafana.ini”, mặc định Grafana sau khi cài đặt sẽ chạy dưới port 3000.

```
root@ip-192-168-120-20:/etc/grafana# ls
grafana.ini  ldap.toml  provisioning
```

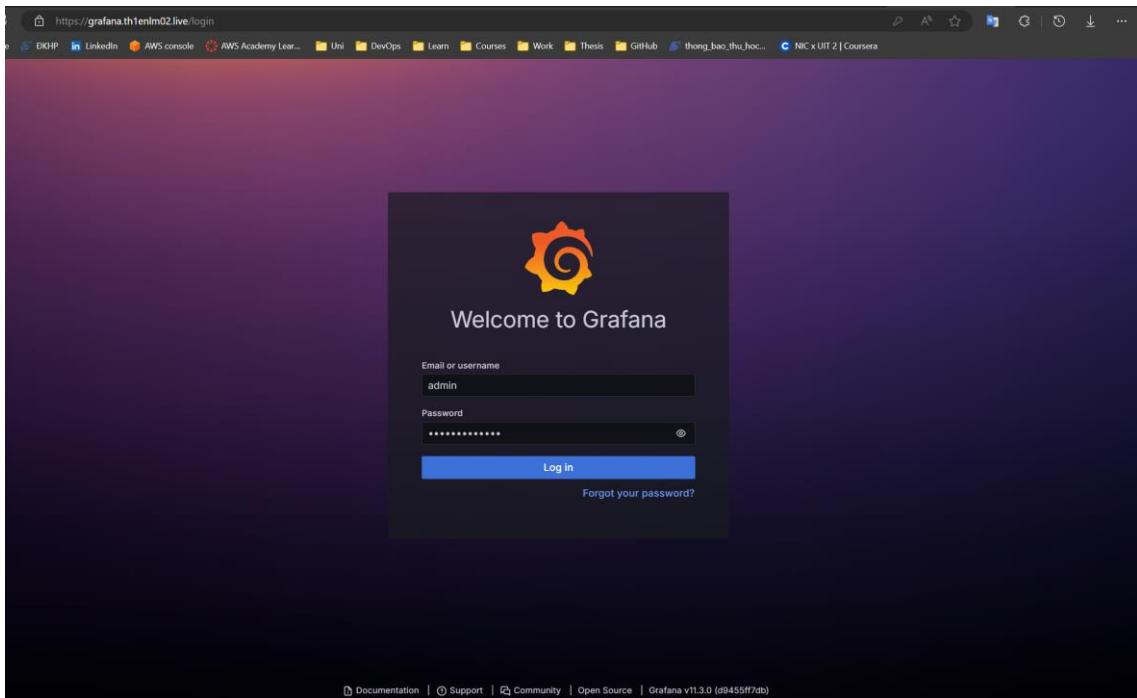
Hình 61. Đường dẫn thư mục chứa file cấu hình Grafana

```
root@ip-192-168-120-20:/etc/grafana# systemctl status grafana-server
● grafana-server.service - Grafana instance
   Loaded: loaded (/lib/systemd/system/grafana-server.service; enabled; vendor preset: enabled)
   Active: active (running) since Sat 2024-11-16 08:29:10 UTC; 54min ago
     Docs: http://docs.grafana.org
     Main PID: 27367 (grafana)
        Tasks: 11 (limit: 4671)
       Memory: 66.0M
      CGroup: /system.slice/grafana-server.service
              └─27367 /usr/share/grafana/bin/grafana server --config=/etc/grafana/grafana.ini --pidfile=/run/grafana/grafana-server.pid --packaging-deb cfg=default.paths.lo

Nov 16 08:39:10 ip-192-168-120-20 grafana[27367]: logger=plugins.update.checker t=2024-11-16T08:39:10.94114953Z level=info msg="Update check succeeded" duration=42.450102ms
Nov 16 08:49:10 ip-192-168-120-20 grafana[27367]: logger=cleanup t=2024-11-16T08:49:10.808784000Z level=info msg="Completed cleanup jobs" duration=11.348606ms
Nov 16 08:49:10 ip-192-168-120-20 grafana[27367]: logger=plugins.update.checker t=2024-11-16T08:49:10.950614235Z level=info msg="Update check succeeded" duration=51.957296ms
Nov 16 08:59:10 ip-192-168-120-20 grafana[27367]: logger=cleanup t=2024-11-16T08:59:10.834568349Z level=info msg="Completed cleanup jobs" duration=36.608058ms
Nov 16 08:59:10 ip-192-168-120-20 grafana[27367]: logger=plugins.update.checker t=2024-11-16T08:59:10.952856172Z level=info msg="Update check succeeded" duration=53.908222ms
Nov 16 09:00:33 ip-192-168-120-20 grafana[27367]: logger=infra.usagestats t=2024-11-16T09:00:33.787174168Z level=info msg="Usage stats are ready to report"
Nov 16 09:09:10 ip-192-168-120-20 grafana[27367]: logger=cleanup t=2024-11-16T09:09:10.809802424Z level=info msg="Completed cleanup jobs" duration=11.582442ms
Nov 16 09:09:10 ip-192-168-120-20 grafana[27367]: logger=plugins.update.checker t=2024-11-16T09:09:10.946808964Z level=info msg="Update check succeeded" duration=47.850922ms
Nov 16 09:19:10 ip-192-168-120-20 grafana[27367]: logger=cleanup t=2024-11-16T09:19:10.809483122Z level=info msg="Completed cleanup jobs" duration=12.055285ms
Nov 16 09:19:10 ip-192-168-120-20 grafana[27367]: logger=plugins.update.checker t=2024-11-16T09:19:10.948520219Z level=info msg="Update check succeeded" duration=50.209608ms
```

Hình 62. Kiểm tra trạng thái Grafana

Truy cập vào Grafana web UI với đường dẫn: <https://grafana.th1enlm02.live>



Hình 63. Màn hình đăng nhập của Grafana

Kết nối đến các data source để lấy dữ liệu từ các nguồn như Prometheus, Loki, Icinga.

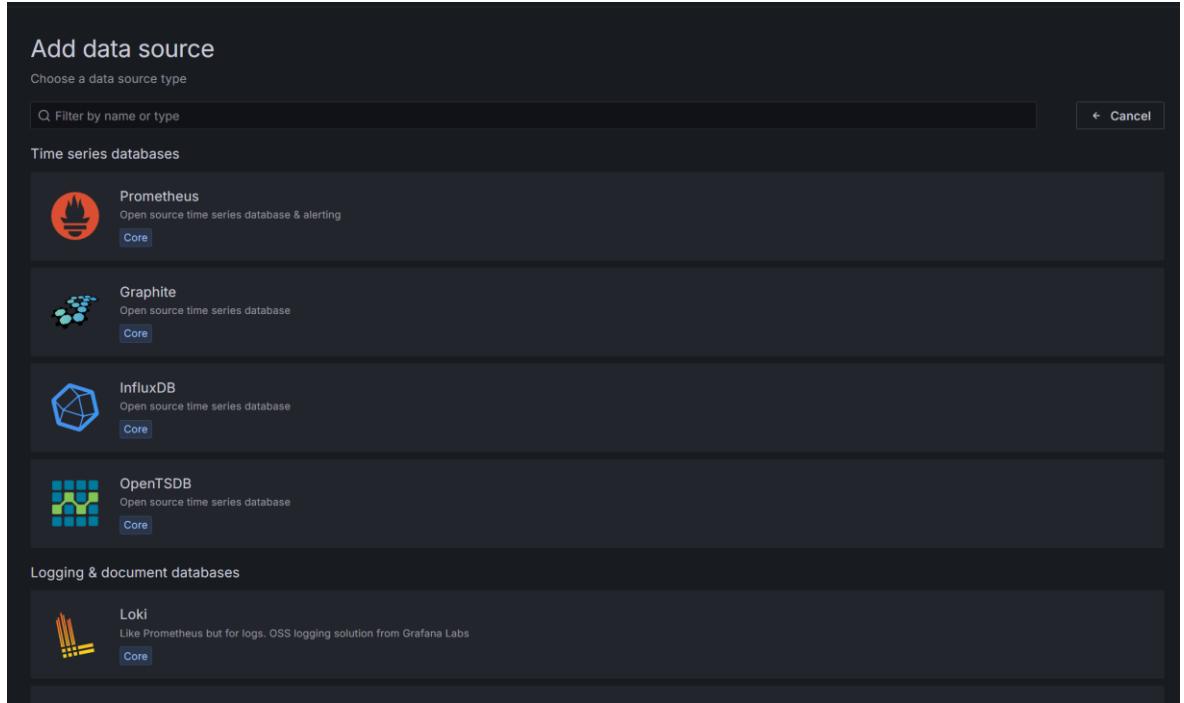
Hình 64. Thông tin các data source

Các bước để kết nối đến một data source:

**Bước 1.** Ở phần Connection → Chọn Data Sources → Thêm data source.

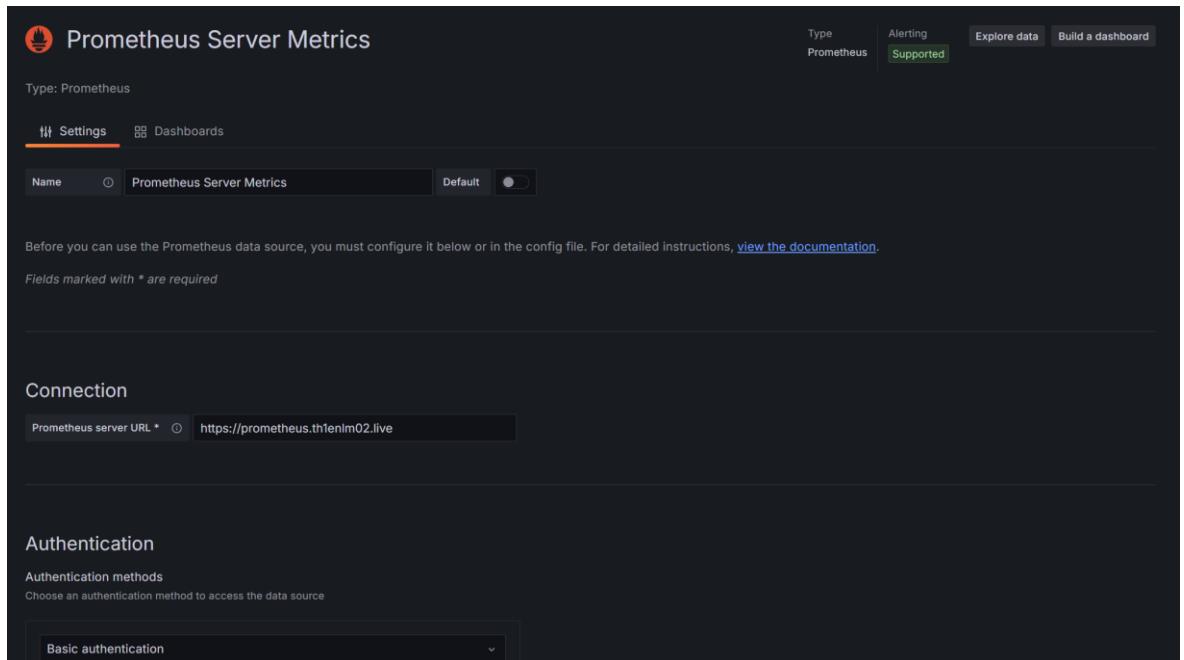
Hình 65. Thêm data source

## Bước 2. Chọn kiểu data source.



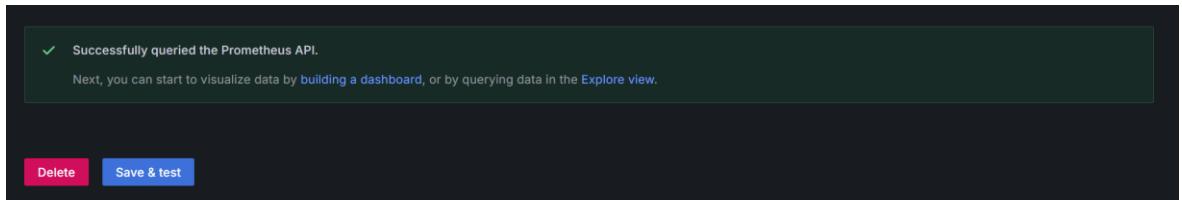
Hình 66. Chọn kiểu data source

## Bước 3. Nhập các thông tin cho data source, trong đó mục Connection là bắt buộc và nhập đường dẫn của data source để query dữ liệu.



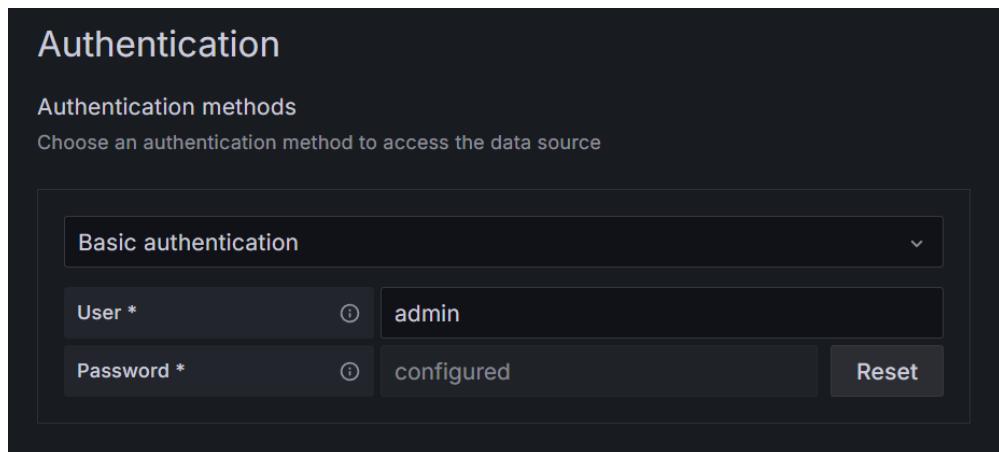
Hình 67. Nhập thông tin cho data source

#### Bước 4. Lưu và kiểm tra kết nối đến data source.



Hình 68. Kiểm tra kết nối đến data source

#### Bước 5. (Optional) Ngoài ra, nếu có cấu hình basic authentication thì nhập thông tin xác thực ở mục Authentication.



Hình 69. Cấu hình xác thực cho data source

Sau khi kết nối được với các data source thành công, tiến hành visualize bằng cách tạo các dashboard cho từng data source.

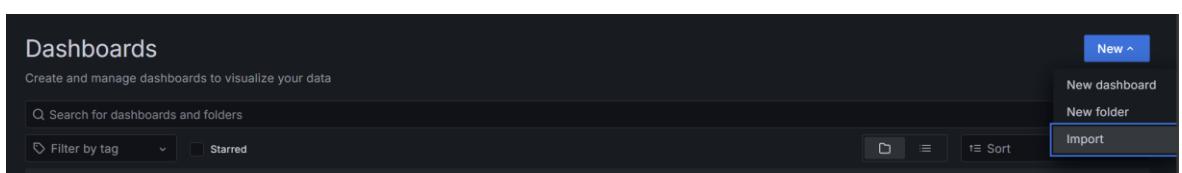
Đối với mỗi dashboard sẽ có template riêng và được cung cấp bởi Grafana, có thể import vào bằng cách nhập URL, ID hoặc JSON model của dashboard đó.

Các bước để import một dashboard:

#### Bước 1. Truy cập vào đường dẫn để lựa chọn dashboard mong muốn tại

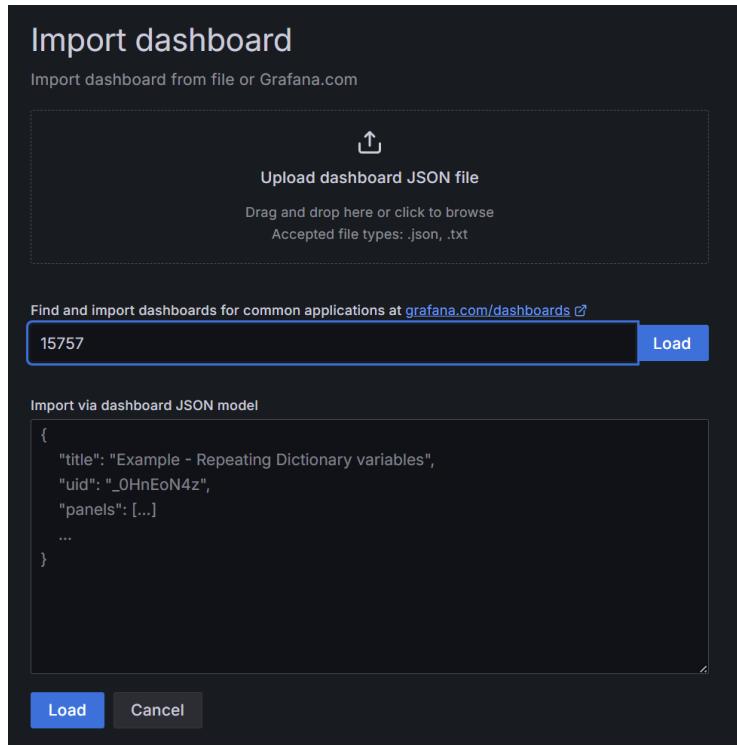
<https://grafana.com/grafana/dashboards/>

#### Bước 2. Chọn mục Dashboard → Ở New chọn Import.



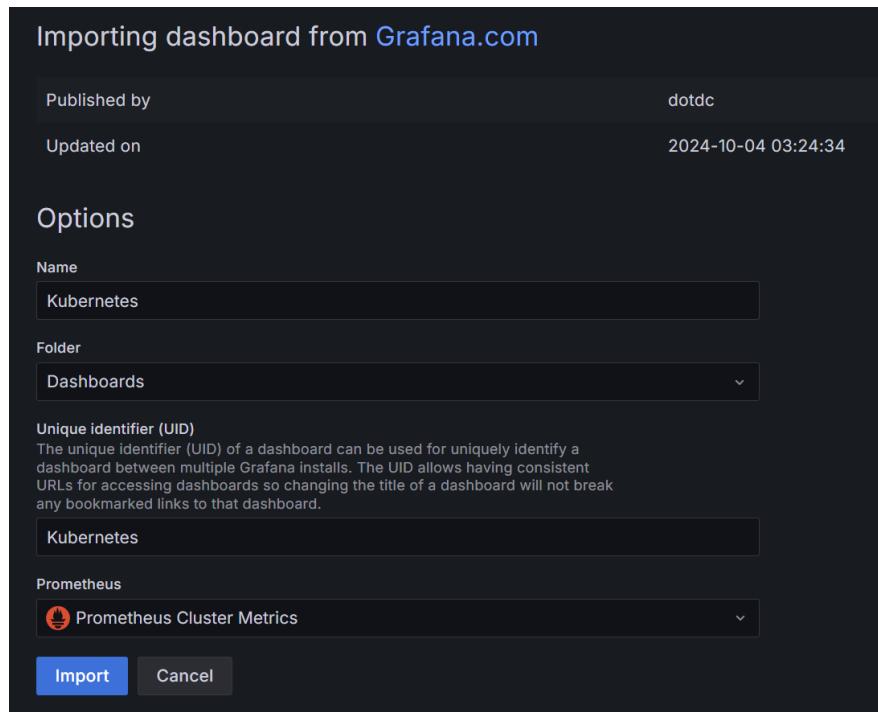
Hình 70. Thêm dashboard

**Bước 3.** Nhập URL, ID hoặc JSON model → Chọn Load.



Hình 71. Nhập URL, ID hoặc JSON model của dashboard

**Bước 4.** Nhập thông tin và chọn data source tương ứng → Chọn Import.



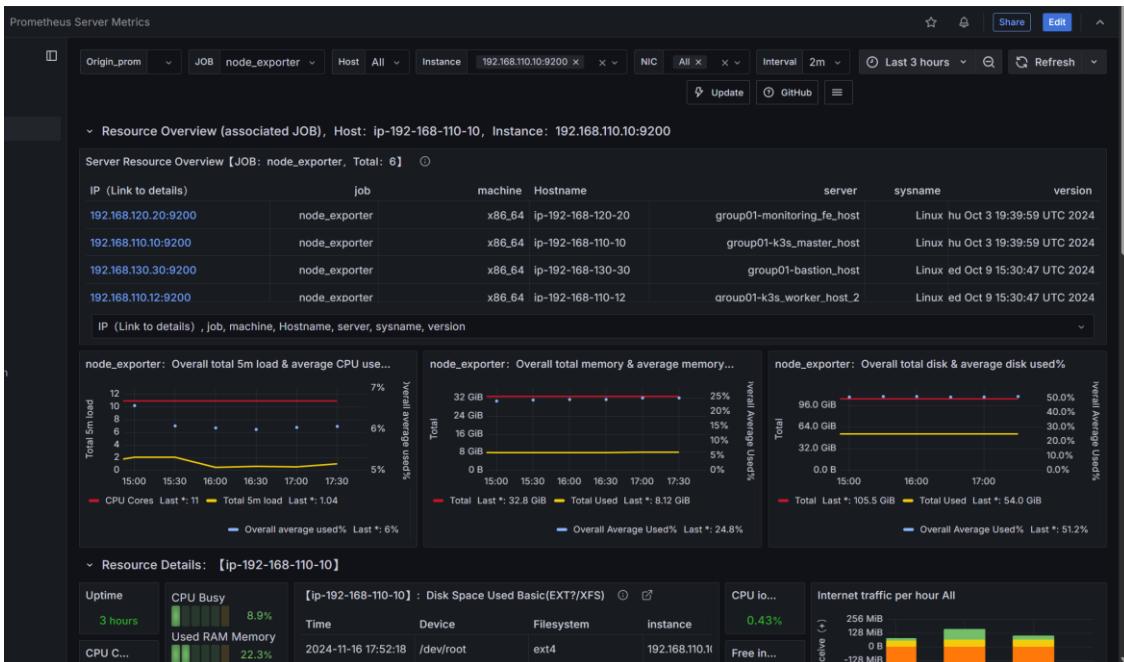
Hình 72. Nhập thông tin và chọn data source tương ứng

## Bước 5. Giao diện của dashboard sau khi được import cùng với data source.

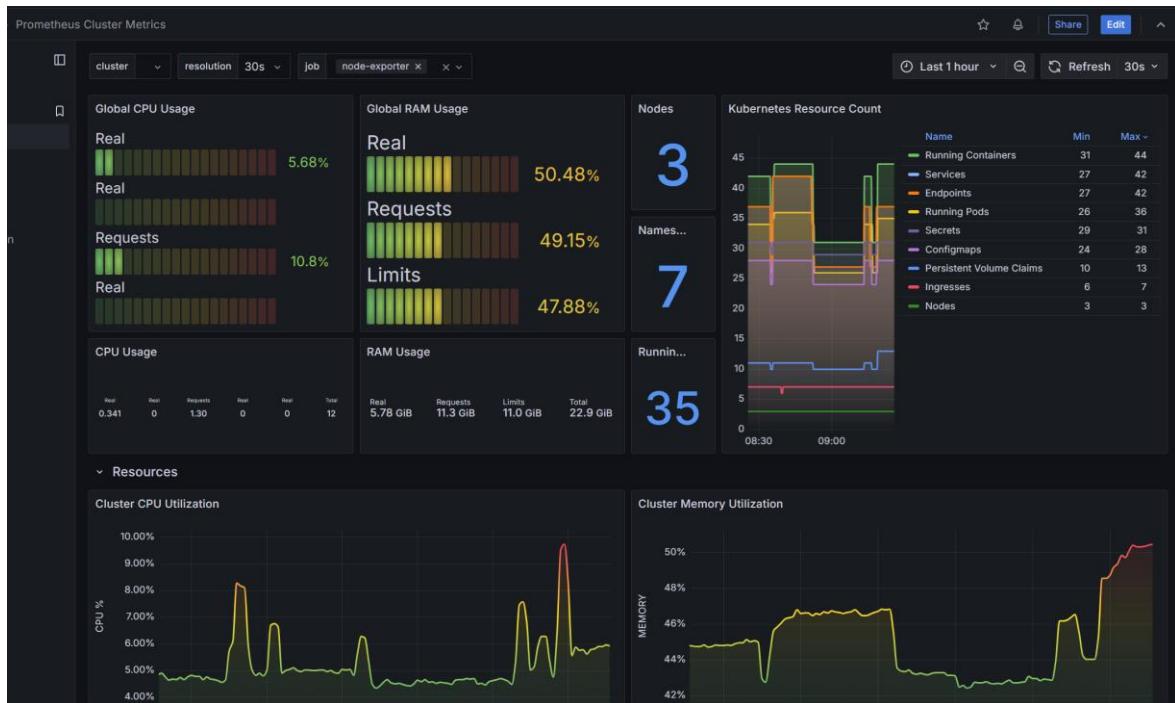


Hình 73. Dashboard sau khi được import

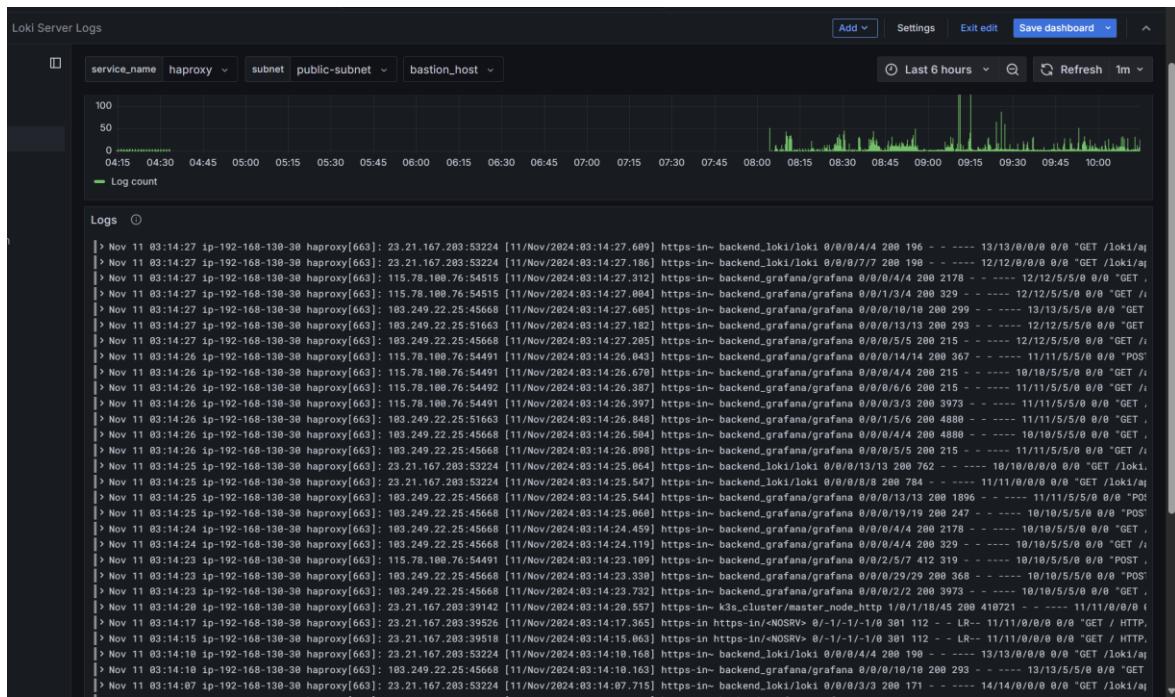
Sau khi thêm các dashboard cần thiết, có thể tùy chỉnh thông tin và các giá trị để hiển thị giao diện theo ý muốn. Sau khi kết nối thành công đến các data source và tạo các dashboard cho từng data source thành công, dưới đây là kết quả:



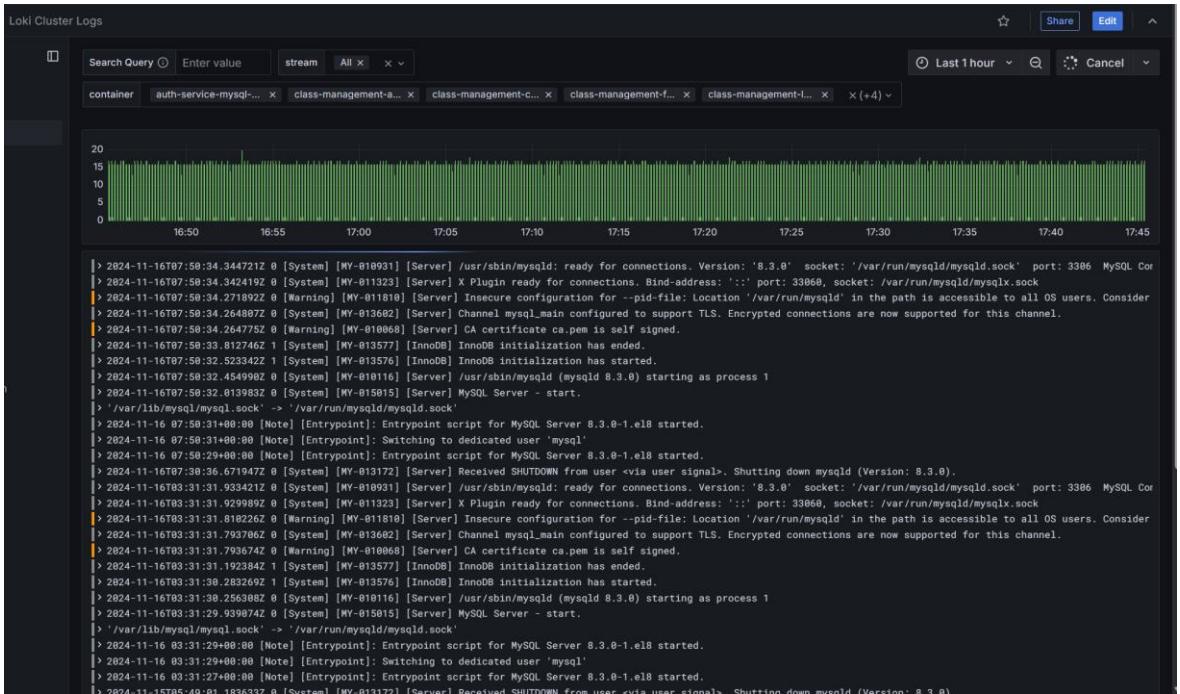
Hình 74. Dashboard cho Prometheus



Hình 75. Dashboard cho Prometheus trên cụm K3s



Hình 76. Dashboard cho Loki



Hình 77. Dashboard cho Loki trên cụm K3s

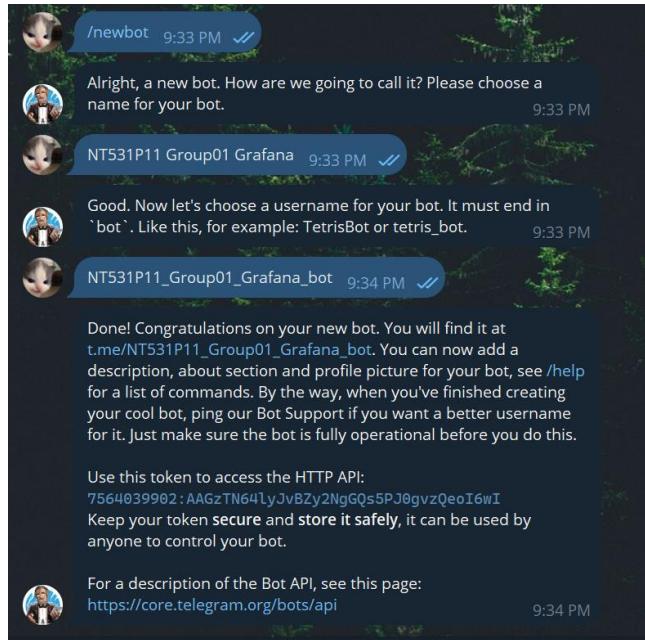
Trong đó, đối với data source được query từ Prometheus và Loki trên cụm K3s thì có thể lấy được dữ liệu các resource của cụm, dựa vào đó có thể tùy chỉnh các giá trị query để hiển thị các thông tin mong muốn.

Các bước cấu hình alert cho Grafana để gửi thông báo đến Telegram khi đạt đến các ngưỡng RAM, CPU, Disk quy định:

#### Bước 1. Tạo Telegram bot bằng BotFather bằng cách thực hiện lần lượt các

bước sau:

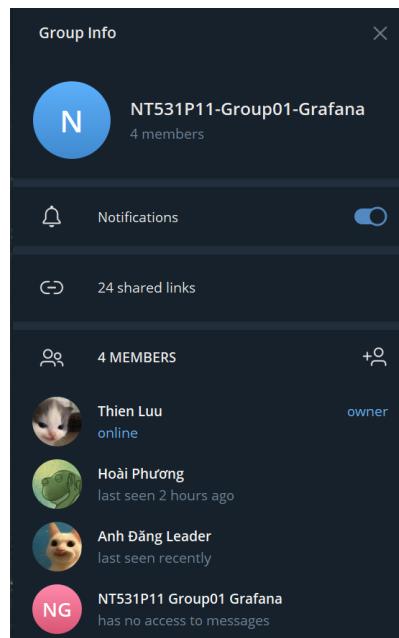
- Nhập “/newbot” để tạo bot mới.
- Nhập tên cho bot.
- Nhập username cho bot, chú ý đối với username thì phải kết thúc bằng từ “bot”.



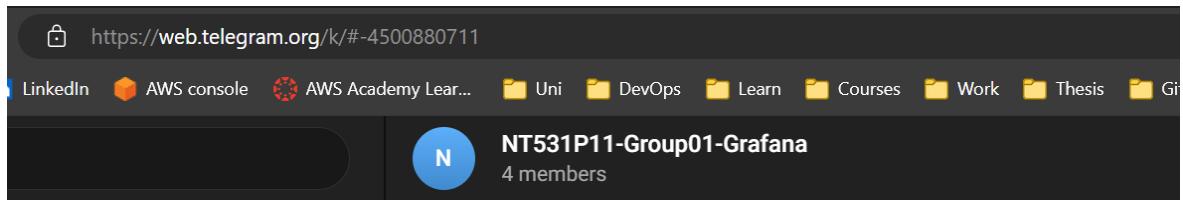
Hình 78. Tạo Telegram bot

⇒ Sau khi tạo bot thành công thì sẽ có API Token của bot đó.

**Bước 2.** Tạo nhóm chat để bot có thể gửi thông báo đến và thêm bot đã tạo trước đó vào nhóm, đồng thời để xem được chat ID thì truy cập vào <https://web.telegram.org> và chọn nhóm chat sau đó xem phần prefix sau dấu “#”.

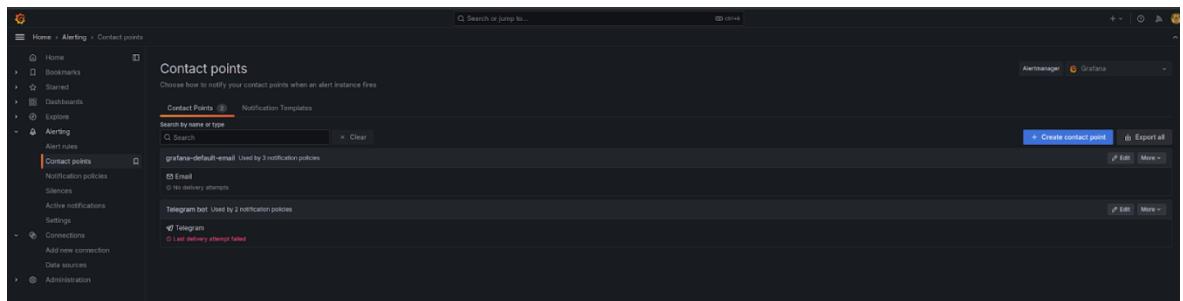


Hình 79. Tạo nhóm chat Telegram để nhận thông báo từ bot



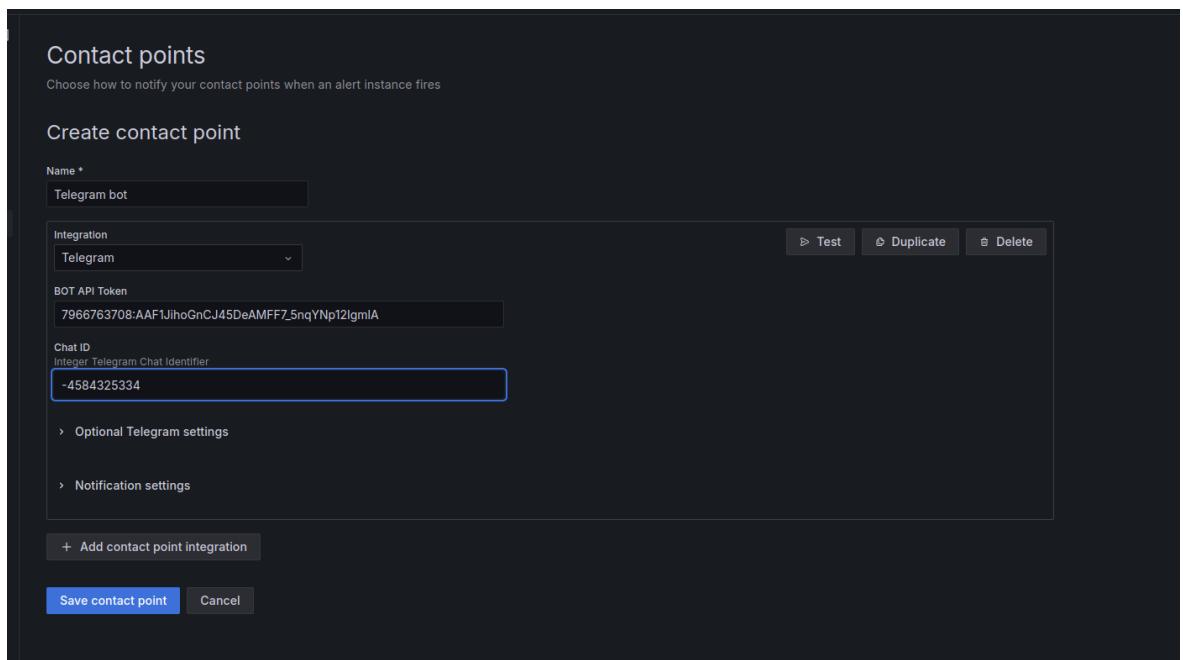
Hình 80. Xem chat ID thông qua Telegram web

**Bước 3.** Chọn mục Alerting → Contact points → Chọn Create contact point.



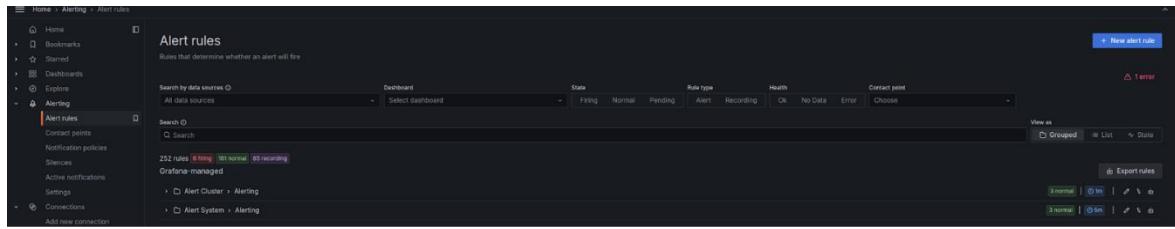
Hình 81. Tạo contact point

**Bước 4.** Nhập thông tin cho contact point, trong đó chọn phương thức cảnh báo ở phần Integration là Telegram → Nhập BOT API Token và chat ID đã tạo ở Telegram → Chọn Save contact point.



Hình 82. Nhập thông tin cho contact point

**Bước 5.** Tạo rule cảnh báo: Alerting → Alert rules → Chọn New alert rule.



Hình 83. Tạo rule cảnh báo

#### Bước 6. Định nghĩa truy vấn và điều kiện cảnh báo:

- Ở phần A:
  - Chọn data source là Prometheus.
  - Chọn Code → Nhập hàm tính toán để lấy điều kiện ở mục Metrics browser sau đó có thể query để kiểm tra.
- Ở phần B: Xử lý, giảm thiểu dữ liệu từ time series thành một giá trị đơn với input lấy từ kết quả query của A và hàm được chọn là lấy giá trị cuối cùng trong chuỗi thời gian.
- Ở phần C: Nhập ngưỡng threshold và điều kiện để trigger cảnh báo với input nhận từ B.

**2. Define query and alert condition**  
Define query and alert condition [Need help?](#)

**A** Prometheus Cluster Metrics [Options](#) 10 minutes, MD = 43200, Min. Interval = 1s [Set as alert condition](#)

**Metrics browser** `(1 - avg(irate(node_cpu_seconds_total{mode="idle"}[5m])) by (instance)) * 100`

[Add query](#)

**Rule type**  
Select where the alert rule will be managed. [Need help?](#)

Grafana-managed  Data source-managed

The alert rule type cannot be changed for an existing rule.

**Expressions**  
Manipulate data returned from queries with math and other operations.

**B Reduce** [Set as alert condition](#)

Takes one or more time series returned from a query or an expression and turns each series into a single number.

**Input** A **Function** Last **Mode** Strict

**C Threshold** [Alert condition](#)

Takes one or more time series returned from a query or an expression and checks if any of the series match the threshold condition.

**Input** B **IS ABOVE** 90

Custom recovery threshold

[Add expression](#) [Preview](#)

Hình 84. Định nghĩa truy vấn và điều kiện cảnh báo

Các hàm query để xử lý đối với các trường hợp:

- CPU overload:

```
(1 - avg(irate(node_cpu_seconds_total{mode="idle"}[5m])) by (instance)) * 100
```

- RAM Usage Over:

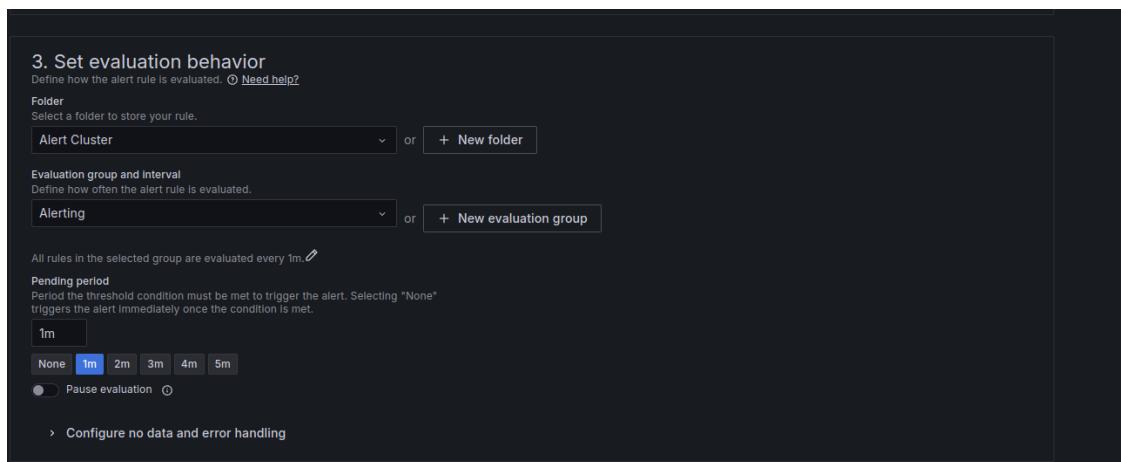
```
avg by (instance) ((node_memory_MemTotal_bytes - node_memory_MemFree_bytes - node_memory_Buffers_bytes - node_memory_Cached_bytes) / node_memory_MemTotal_bytes) * 100
```

- Disk Usage Over:

```
avg by (instance) ((node_filesystem_size_bytes{mountpoint="/" } - node_filesystem_free_bytes{mountpoint="/" }) / node_filesystem_size_bytes{mountpoint="/" }) * 100
```

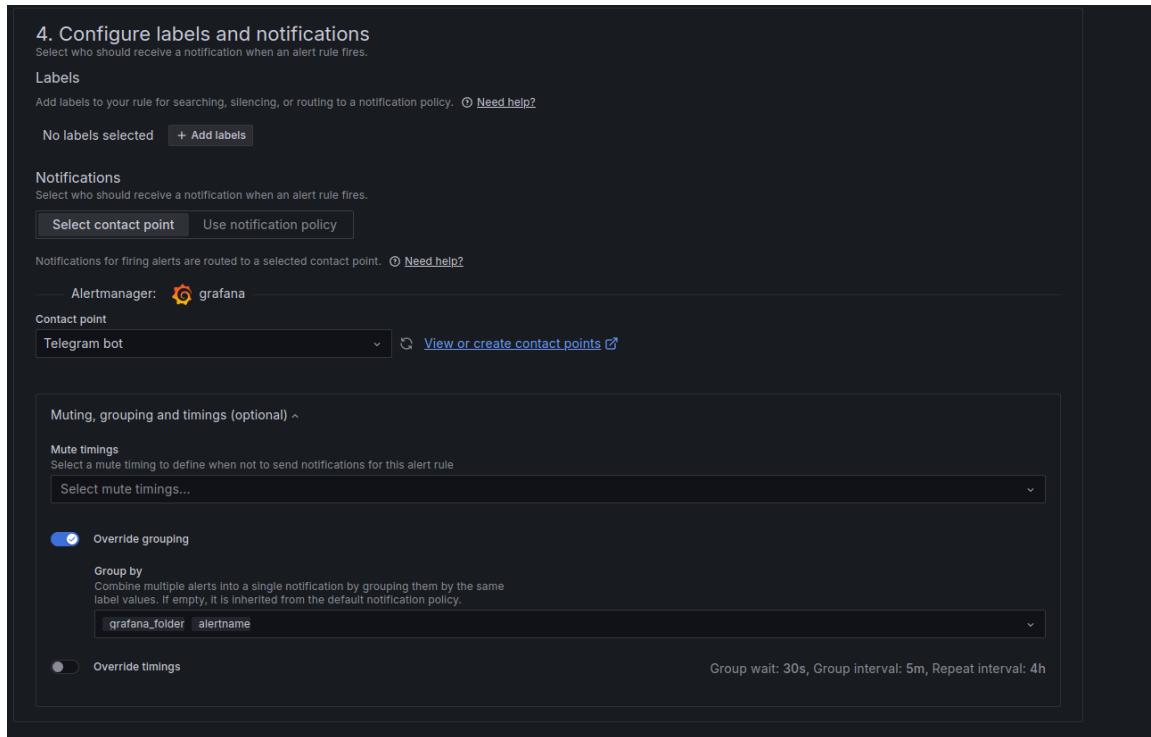
### Bước 7. Thiết lập hành vi đánh giá:

- Folder: Chọn hoặc tạo thư mục để phân loại các quy tắc cảnh báo, giúp dễ quản lý hơn.
- Pending period: Thiết lập khoảng thời gian kiểm tra (interval) giữa mỗi lần cảnh báo, giúp tránh các cảnh báo không cần thiết nếu điều kiện chỉ xảy ra trong thời gian ngắn.



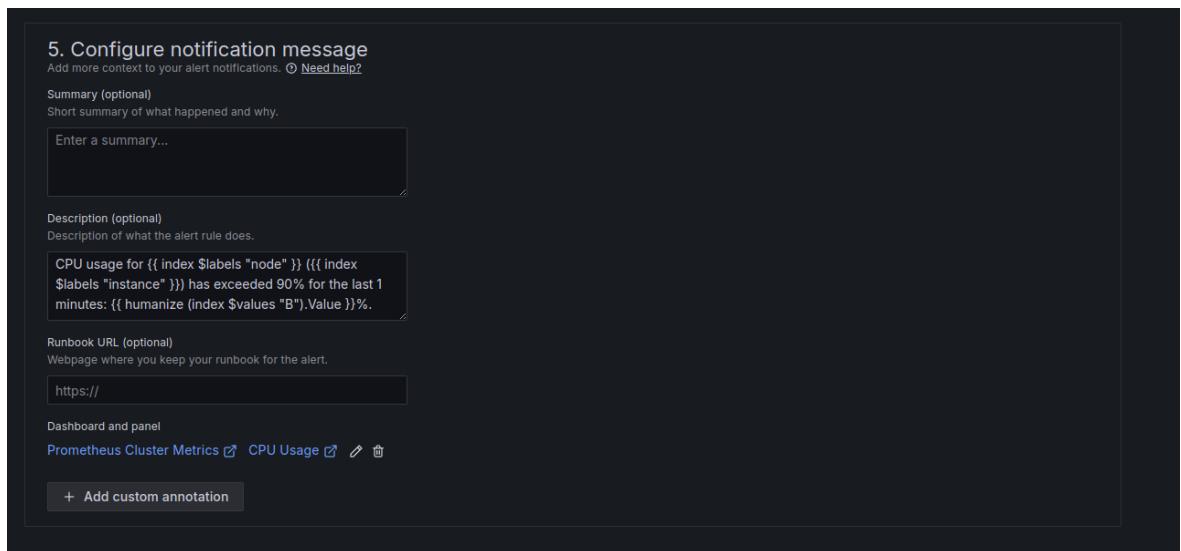
Hình 85. Thiết lập hành vi đánh giá

**Bước 8.** Cấu hình các nhãn và thông báo: Chọn phương thức cảnh báo qua Telegram bot đã cấu hình trước đó trong Contact Points.



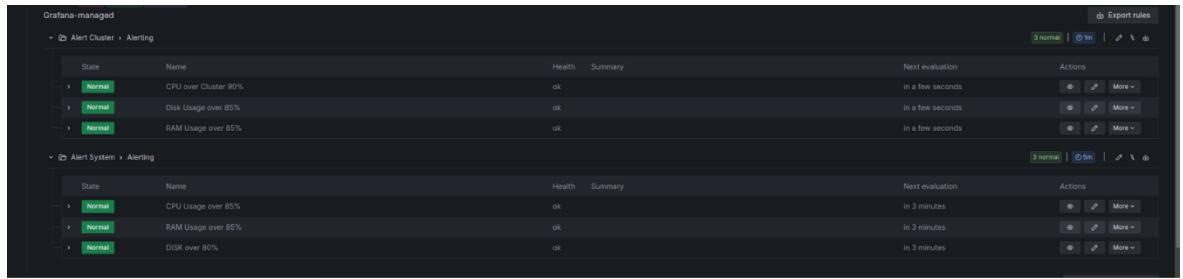
Hình 86. Cấu hình các nhãn và thông báo

**Bước 9.** Cấu hình tin nhắn thông báo: Soạn nội dung tin nhắn sẽ được gửi đến Telegram. Tin nhắn nên chứa tên quy tắc, giá trị và ngưỡng cảnh báo để nắm rõ thông tin.



Hình 87. Cấu hình tin nhắn thông báo

## Bước 10. Bấm Save rule để hoàn tất cấu hình.



Hình 88. Kết quả sau khi tạo tất cả rule cảnh báo

### 3.4.4. Icinga2

Trong Icinga có 3 thành phần chính là Icinga Core, Icinga Web, và Database. Đối với Database sẽ có nhiều lựa chọn để sử dụng. Ở đây nhóm sử dụng InfluxDB để lưu trữ dữ liệu cho Icinga. InfluxDB là một lựa chọn tốt cho việc lưu trữ các dữ liệu đo lường thời gian thực vì nó được thiết kế để xử lý các dữ liệu dạng time-series, như trạng thái hệ thống hoặc hiệu suất dịch vụ mà Icinga thu thập. Dưới đây là phần cấu hình module InfluxDB (icinga2-feature-influxdb)

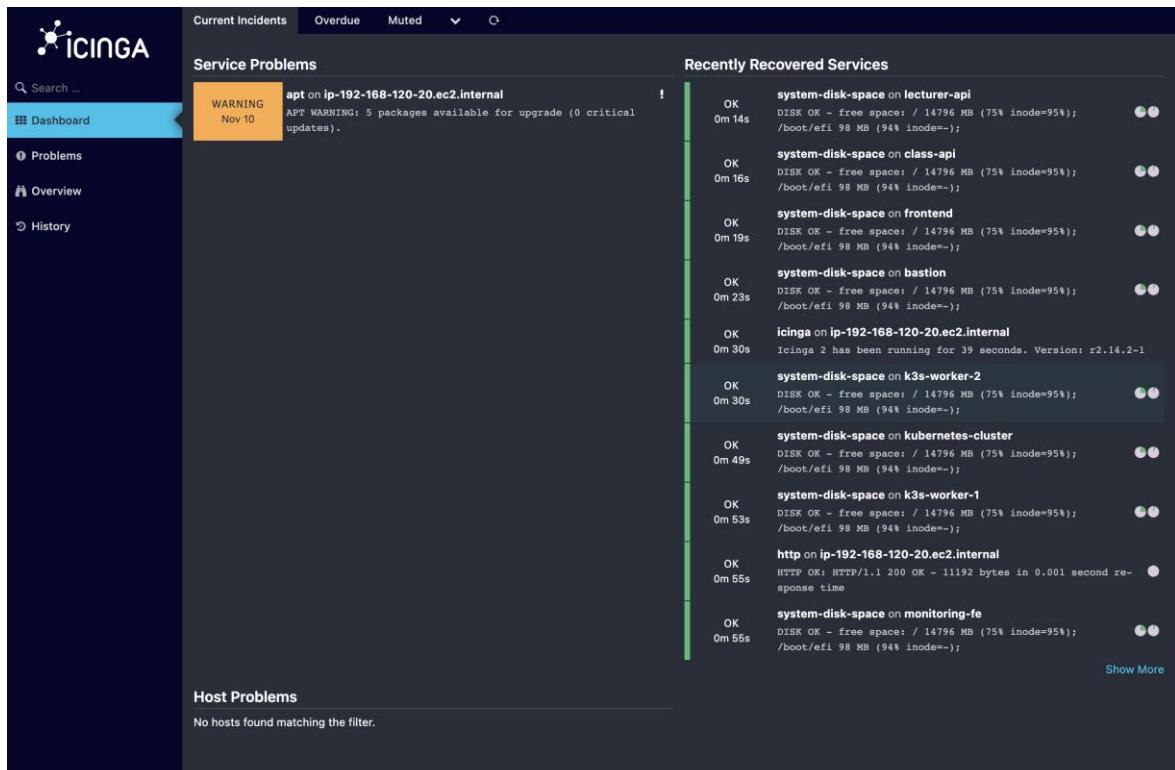
```
main.yml 4, M  ido-mysql.yml 9+, U  influxdb.conf.j2 U  Settings  settings.json
ansible-hub > roles > icinga2 > templates > influxdb.conf.j2
1  object InfluxdbWriter "influxdb" {
2      host = "127.0.0.1"
3      port = 8086
4      database = "icinga2"
5      username = "icinga2"
6      password = "{{ icinga_db_password }}"
7      enable_send_thresholds = true
8      enable_send_metadata = true
9      flush_threshold = 1024
10     flush_interval = 10s
11
12     host_template = {
13         measurement = "$host.check_command$"
14         tags = {
15             |   hostname = "$host.name$"
16         }
17     }
18
19     service_template = {
20         measurement = "$service.check_command$"
21         tags = {
22             |   hostname = "$host.name$"
23             |   service = "$service.name$"
24         }
25     }
26 }
```

Hình 89. Cấu hình Influxdb

Trong đó gồm các thành phần khai báo về:

- Địa chỉ và cổng (host, port) xác định nơi InfluxDB đang chạy.
- Tên cơ sở dữ liệu (database) và thông tin đăng nhập (username, password) được sử dụng để kết nối và ghi dữ liệu vào InfluxDB.
- Các tuỳ chọn enable\_send\_thresholds và enable\_send\_metadata cho phép gửi thêm thông tin về người dùng và metadata của các dịch vụ, giúp phân tích dữ liệu chi tiết hơn.
- Flush threshold và flush interval điều chỉnh số lượng và thời gian giữa các lần đẩy dữ liệu vào InfluxDB.
- host\_template và service\_template chỉ định cách cấu trúc dữ liệu (measurement và tags) khi ghi dữ liệu từ các host và service của Icinga2 vào InfluxDB.

Icingaweb sẽ cung cấp một giao diện giúp người dùng dễ dàng theo dõi những metric thu thập được lưu dưới database.



Hình 90. Màn hình dashboard của Icinga

Cấu hình thông tin các host bao gồm các instance cũng như các endpoint đã deploy của ứng dụng trên cụm K3s.

Hình 91. Cấu hình hosts và endpoints của ứng dụng

Icinga cung cấp nhiều plugin từ thư viện của Nagios cho phép monitoring hệ thống một cách chi tiết.

```
root@ip-192-168-130-30:/usr/lib/nagios/plugins# ls
check_ajp           check_fping      check_load      check_procs     check_users
check_apt           check_ftp       check_log       check_radius   check_v46
check_backuppc      check_game      check_mailq     check_raid     check_wave
check_bgpstate      check_graphite  check_memcached check_rbl      check_webinject
check_breeze         check_haproxy   check_memory    check_real    check_whois
check_by_ssh         check_haproxy_stats check_mongodb  check_redis   check_zone_auth
check_cert_expire   check_host      check_mongodb.py check_rpc     check_zone_rrsig_expiration
check_cert_expire_dir check_hp_bladechassis check_mrtg      check_rta_multi imap_ssl_cert
check_checksums      check_hpasm     check_mrtgtraf  check_running_kernel imap_ssl_cert_epn
check_clamav         check_hpjhd     check_multipath check_sensors negate
check_clamd          check_http      check_mysql    check_simap   pmp-check-aws-rds.py
check_cluster        check_httptest  check_mysql_health check_smstools pmp-check-lvm-snapshots
check_cups           check_icmp      check_mysql_query check_smtp   pmp-check-mongo.py
check_db              check_ide_smart  check_nagios    check_smtp_send pmp-check-mysql-deadlocks
check_debscan        check_iperstatus check_ntfmounts  check_ntp      pmp-check-mysql-deleted-files
check_dhcp            check_ifstatus  check_ntp       check_ntp_time pmp-check-mysql-file-privs
check_dig             check_imap      check_ntp       check_ntp_time pmp-check-mysql-innodb
check_disk            check_imap_quota  check_ntp       check_ntp_time pmp-check-mysql-pidfile
check_disk_smb        check_imap_quota_epn check_ntp_peer  check_ntp_time pmp-check-mysql-processlist
check_dns             check_imap_receive  check_ntp      check_ntp_time pmp-check-mysql-replication-delay
check_dnsec_delegation check_imap_receive_epn check_ntp_time check_ntp_time pmp-check-mysql-replication-running
check_drdbs           check_imapi_sensor check_ntstat   check_ntp_time pmp-check-mysql-status
check_dummy           check_ircd      check_ntp       check_ntp_time pmp-check-mysql-ts-count
check_email_delivery  check_jabber     check_overrr   check_ntp_time pmp-check-pt-table-checksum
check_email_delivery_epn check_ldap      check_packages  check_ntp_time pmp-check-unix-memory
check_entropy         check_ldaps     check_pgsql    check_ntp_time urlize
check_etc_hosts       check_libs      check_ping     check_ntp_time utils.pm
check_etc_resolv      check_libs_ng   check_pollen   check_ntp_time utils.sh
check_file_age        check_librvt   check_pop      check_ntp_time
check_flexlm          check_lm_sensors check_printer  check_ntp_time

```

Hình 92. Plugin từ thư viện của Nagios

Sử dụng các plugin có sẵn để giám sát các dịch vụ Mạng (ping4, ping6, ssh), dịch vụ Web (Web VHosts, Frontend, Api Health Check), và hệ thống (Load, System Processes, Zombie Process, High CPU Processes, Disk Space, Disk Inodes, Users).

Service Group	Service States
API Services	3
Ping Checks	13
System Checks	84
Web Services	1

Service	Description
student-api-health on student-api	HTTP OK: HTTP/1.1 200 OK - 22661 bytes in 0.023 second response time
class-api-health on class-api	HTTP OK: HTTP/1.1 200 OK - 436 bytes in 0.036 second response time
lecturer-api-health on lecturer-api	HTTP OK: HTTP/1.1 200 OK - 9035 bytes in 0.019 second response time

Hình 93. Giám sát dịch vụ API

Service Group	Service States
API Services	3
Ping Checks	13
System Checks	84
Web Services	1

Service	Description
ping4 on bastion	PING OK - Packet loss = 0%, RTA = 0.91 ms
ping4 on class-api	PING OK - Packet loss = 0%, RTA = 1.36 ms
ping4 on student-api	PING OK - Packet loss = 0%, RTA = 1.45 ms
ping4 on lecturer-api	PING OK - Packet loss = 0%, RTA = 2.45 ms
ping4 on frontend	PING OK - Packet loss = 0%, RTA = 1.44 ms
ping4 on k3s-worker-2	PING OK - Packet loss = 0%, RTA = 1.19 ms
ping4 on k3s-worker-1	PING OK - Packet loss = 0%, RTA = 1.33 ms
ping4 on k3s-master	PING OK - Packet loss = 0%, RTA = 0.96 ms
ping4 on kubernetes-cluster	PING OK - Packet loss = 0%, RTA = 1.13 ms
ping4 on monitoring-fe	PING OK - Packet loss = 0%, RTA = 0.03 ms
ping4 on monitoring-be	PING OK - Packet loss = 0%, RTA = 0.79 ms
ping4 on ip-192-168-120-20.ec2.internal	PING OK - Packet loss = 0%, RTA = 0.03 ms
ping6 on ip-192-168-120-20.ec2.internal	PING OK - Packet loss = 0%, RTA = 0.03 ms

Hình 94. Giám sát mạng

Hình 95. Giám sát hệ thống

Hình 96. Giám sát dịch vụ web

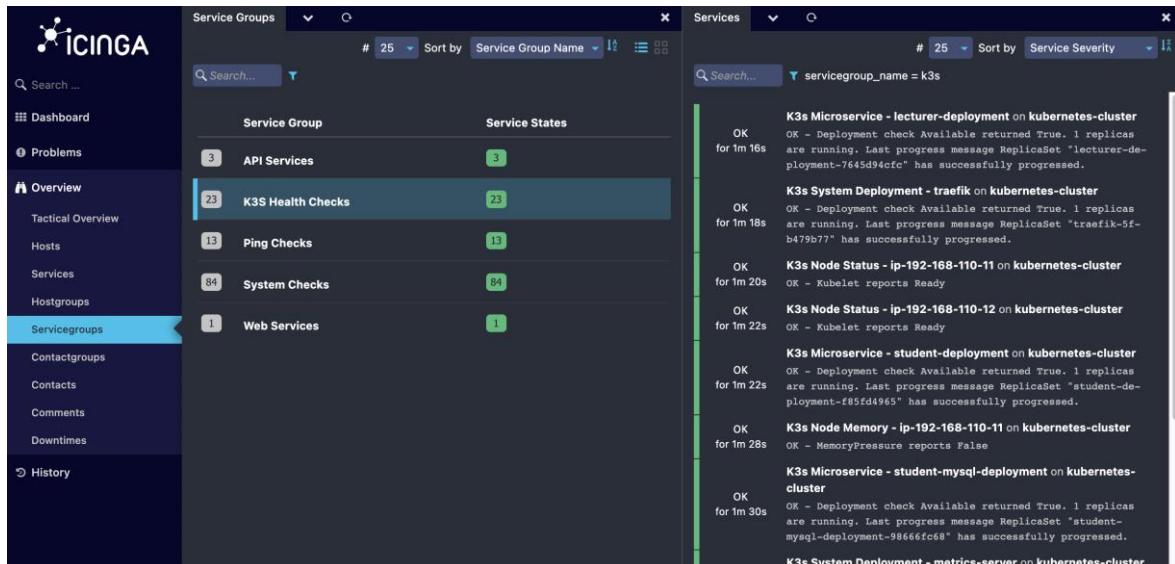
```

1 // Service Groups Definition
2 object ServiceGroup "ping" {
3   display_name = "Ping Checks"
4   assign where match("ping*", service.name)
5 }
6
7 object ServiceGroup "system" {
8   display_name = "System Checks"
9   assign where match("system-*", service.name)
10 }
11
12 object ServiceGroup "web" {
13   display_name = "Web Services"
14   assign where match("http-*", service.name)
15 }
16
17 object ServiceGroup "api" {
18   display_name = "API Services"
19   assign where match("*-api-health", service.name)
20 }
21

```

Hình 97. Định nghĩa các Service Groups

Ngoài ra để giám sát chi tiết hơn trong cluster Kubernetes, bao gồm các dịch vụ về trạng thái node, các deployment quan trọng, các microservices, các daemonset và giám sát bộ nhớ cho các node, nhóm sử dụng thêm một custom plugin check\_kube để thực hiện.



Hình 98. Giám sát Kubernetes

Mặc định, Icinga 2 hỗ trợ chức năng thông báo qua email khi có sự cố về host hoặc service. Tuy nhiên, đổi với mô hình triển khai sẽ cấu hình bổ sung chức năng thông báo qua Telegram để đảm bảo tính kịp thời và đa dạng hóa kênh cảnh báo. Thông báo qua Telegram giúp người vận hành nhận thông tin nhanh hơn,

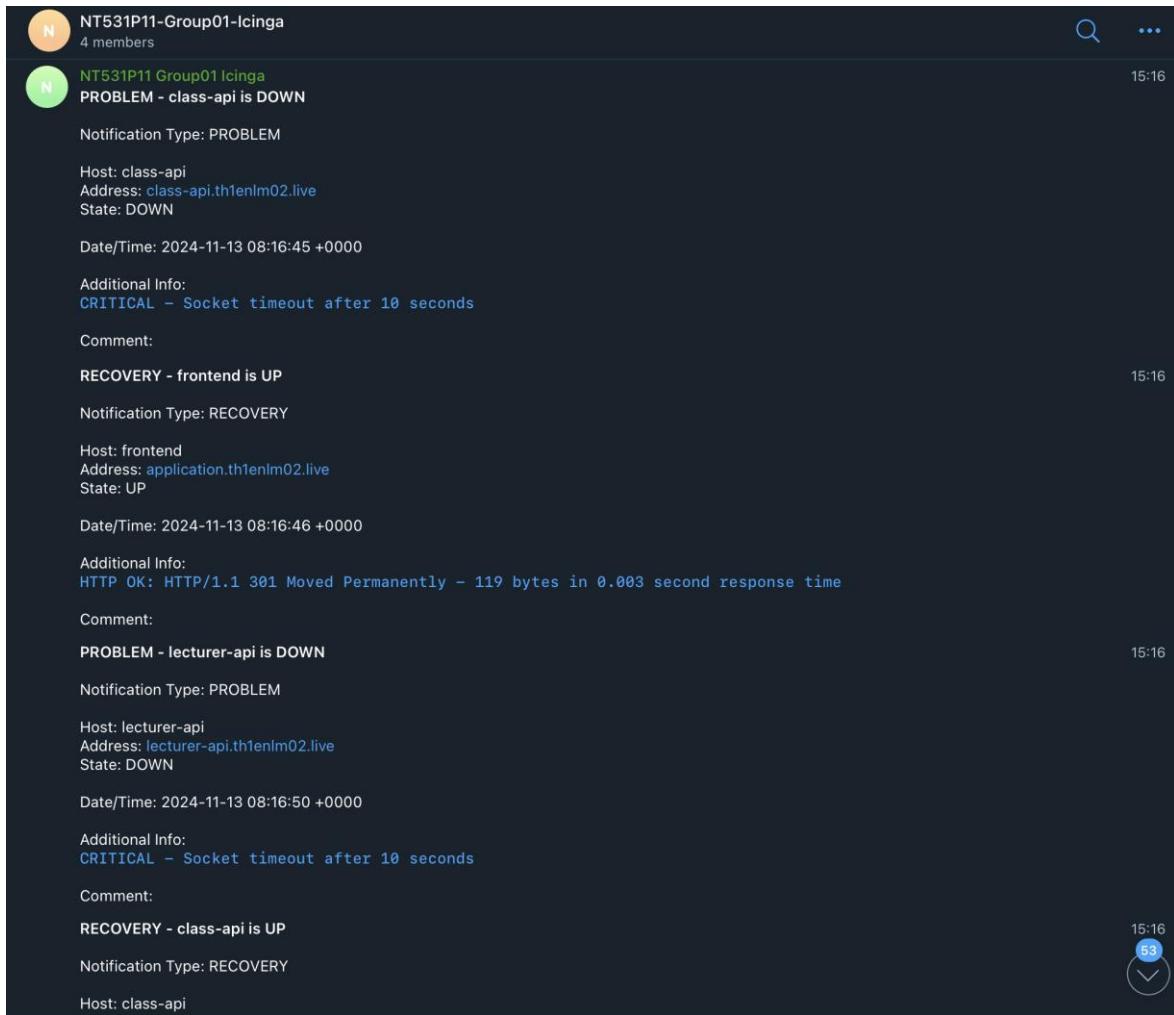
phù hợp với yêu cầu giám sát thời gian thực. Ngoài ra, cấu hình này cho phép tùy biến nội dung thông báo chi tiết, dễ đọc từ đó tối ưu hóa việc quản lý và xử lý sự cố hệ thống.

```
root@ip-192-168-120-20:/etc/icinga2/scripts# ls
mail-host-notification.sh mail-service-notification.sh telegram-host-notification.sh telegram-service-notification.sh
root@ip-192-168-120-20:/etc/icinga2/scripts# cat telegram-host-notification.sh
#!/bin/sh
template=$(cat <<TEMPLATE
*$NOTIFICATIONTYPE - $HOSTDISPLAYNAME is $HOSTSTATE*
Notification Type: $NOTIFICATIONTYPE
Host: $HOSTALIAS
Address: $HOSTADDRESS
State: $HOSTSTATE
Date/Time: $LONGDATETIME
Additional Info:
\$HOSTOUTPUT\`
Comment: [$NOTIFICATIONAUTHORNAME] $NOTIFICATIONCOMMENT
TEMPLATE
)
/usr/bin/curl --silent --output /dev/null \
--data-urlencode "chat_id=${TELEGRAM_CHAT_ID}" \
--data-urlencode "text=${template}" \
--data-urlencode "parse_mode=Markdown" \
"https://api.telegram.org/bot${TELEGRAM_BOT_TOKEN}/sendMessage"
root@ip-192-168-120-20:/etc/icinga2/scripts#
```

Hình 99. Cấu hình host Telegram

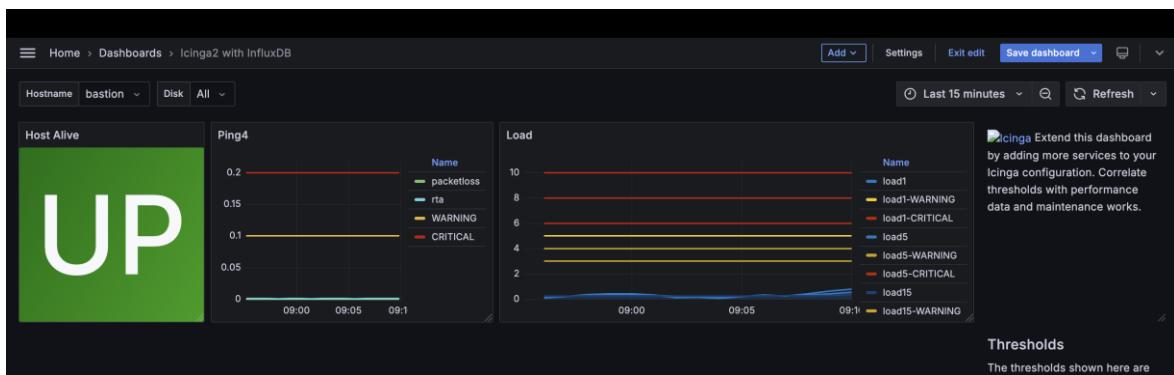
```
root@ip-192-168-120-20:/etc/icinga2/scripts# cat telegram-service-notification.sh
#!/bin/sh
template=$(cat <<TEMPLATE
*$NOTIFICATIONTYPE - $HOSTDISPLAYNAME - $SERVICEDISPLAYNAME is $SERVICESTATE*
Notification Type: $NOTIFICATIONTYPE
Service: $SERVICEDESC
Host: $HOSTALIAS
Address: $HOSTADDRESS
State: $SERVICESTATE
Date/Time: $LONGDATETIME
Additional Info:
\$SERVICEOUTPUT\`
Comment: [$NOTIFICATIONAUTHORNAME] $NOTIFICATIONCOMMENT
TEMPLATE
)
/usr/bin/curl --silent --output /dev/null \
--data-urlencode "chat_id=${TELEGRAM_CHAT_ID}" \
--data-urlencode "text=${template}" \
--data-urlencode "parse_mode=Markdown" \
"https://api.telegram.org/bot${TELEGRAM_BOT_TOKEN}/sendMessage"
```

Hình 100. Cấu hình service Telegram

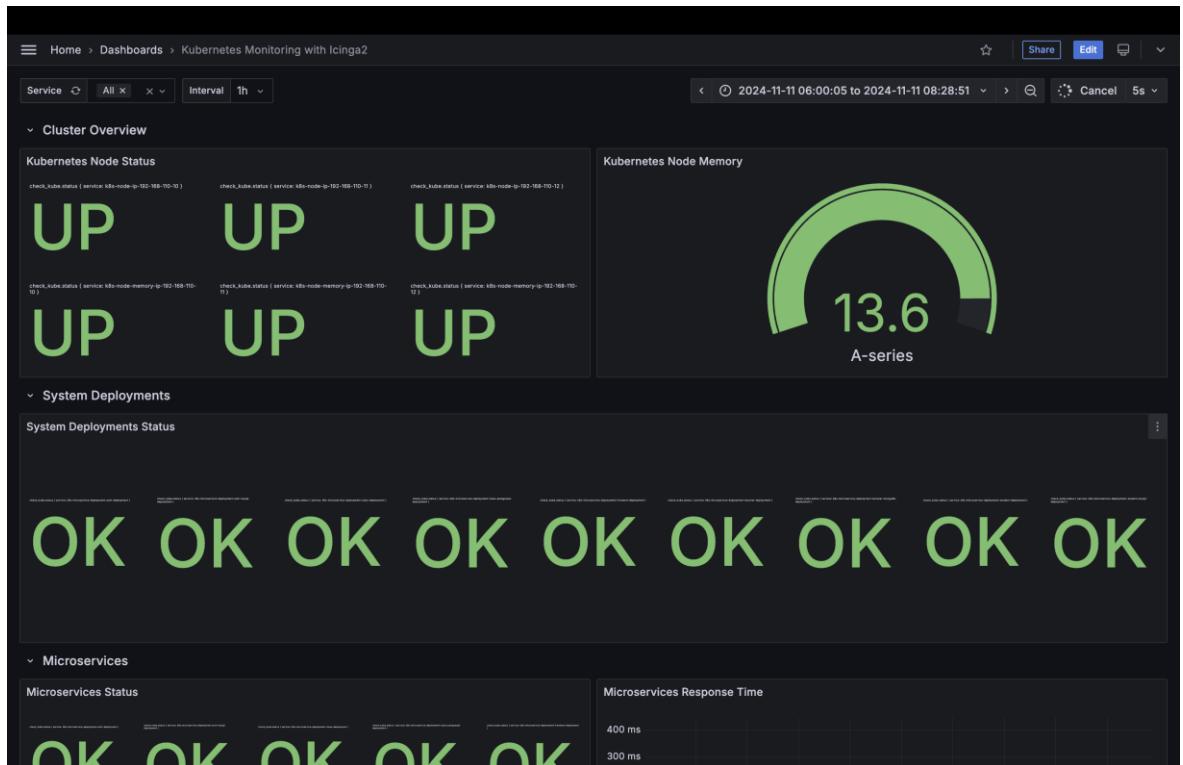


Hình 101. Kết quả cấu hình thông báo qua kênh Telegram

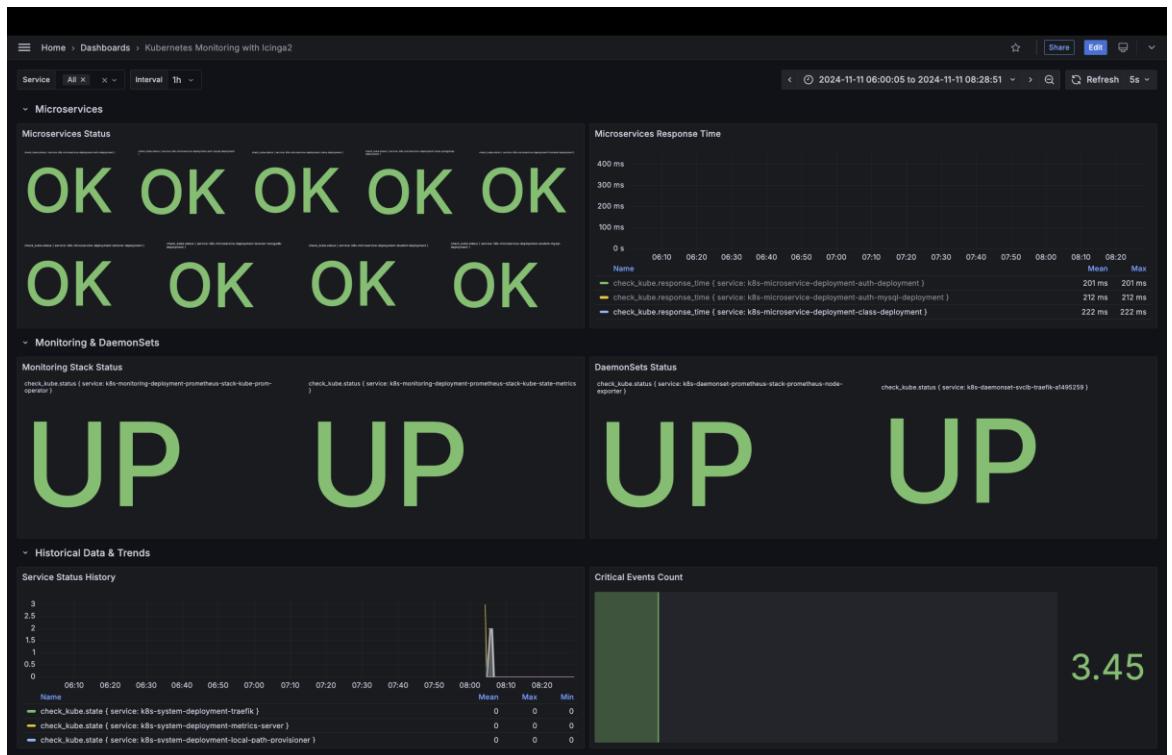
Metric Icinga thu được và lưu vào database. Những metric này có thể sử dụng cho việc mô hình hoá lên Grafana. Tuy nhiên đòi hỏi cấu hình phức tạp qua các lệnh query.



Hình 102. Theo dõi các dịch vụ hệ thống



Hình 103. Theo dõi dịch vụ trong K3S (1)



Hình 104. Theo dõi dịch vụ trong K3S (2)

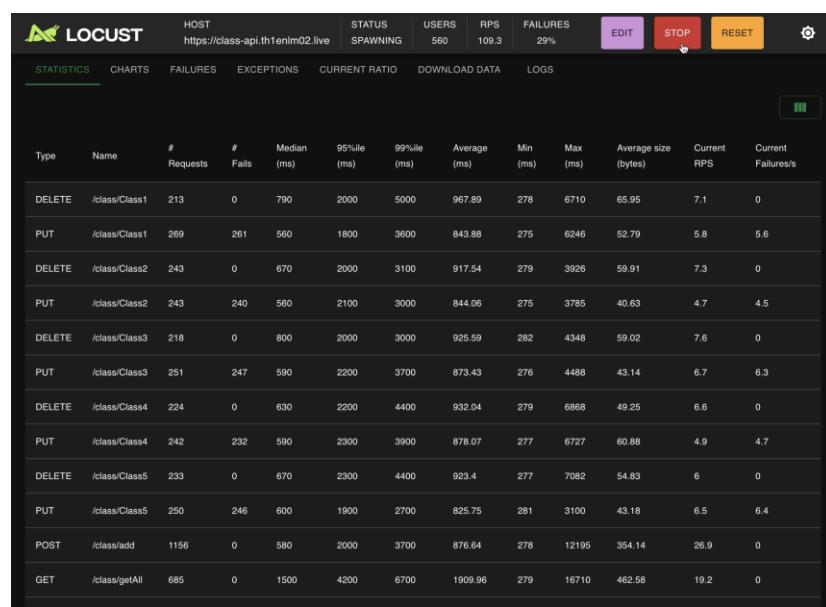
### 3.5. Triển khai các kịch bản kiểm thử

#### 3.5.1. Đánh giá hiệu suất của hệ thống phân phối tải

Để đánh giá hiệu suất của hệ thống phân phối tải sử dụng HA Proxy, sử dụng **Locust** để tạo một lượng lớn người dùng ảo, gửi yêu cầu đến ứng dụng qua HAProxy và theo dõi các thông số quan trọng. Sau đây là một mô tả chi tiết hơn về nội dung yêu cầu:

##### Quy trình kiểm thử:

- Tạo tải giả lập: Sử dụng Locust để mô phỏng một lượng lớn người dùng ảo gửi yêu cầu đồng thời đến ứng dụng thông qua HAProxy. Mục đích là kiểm tra khả năng chịu tải của hệ thống và xem xét hiệu suất phân phối tải của HA Proxy.



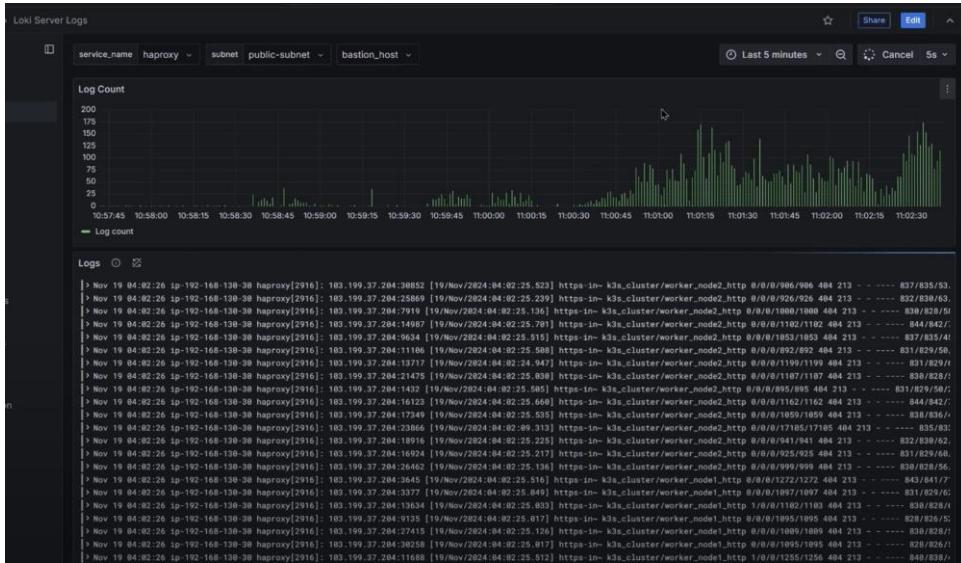
Hình 105. Mô phỏng lượng lớn người dùng truy cập

- Đánh giá hiệu suất phân phối tải: Kiểm tra xem HA Proxy có thể phân phối các yêu cầu đến các worker node một cách hiệu quả như thế nào khi đối mặt với tải cao.
- Giám sát:
  - Sử dụng HA Proxy stats để kiểm tra số lượng traffic phân phối đến các backend.



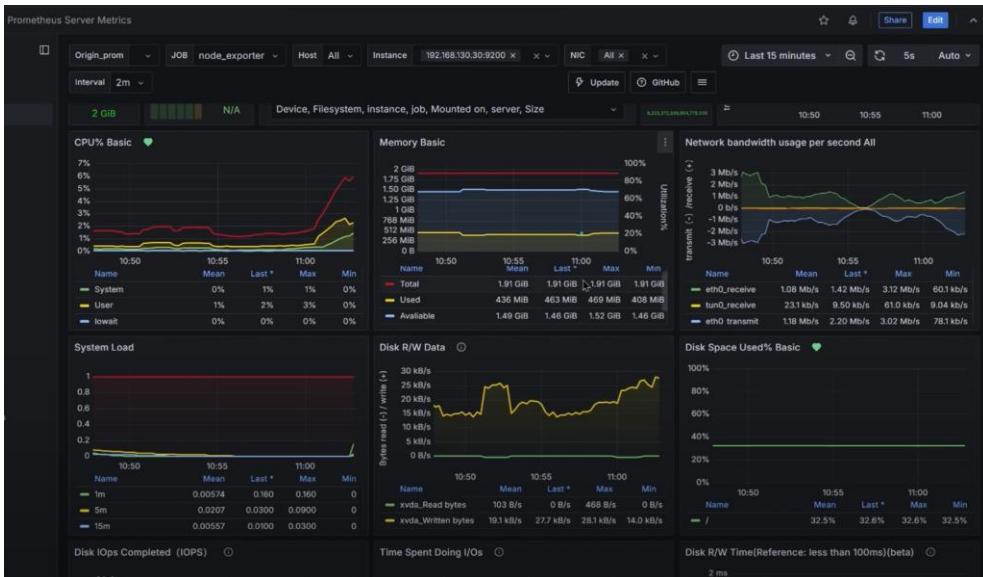
Hình 106. Kiểm tra số lượng traffic ở giao diện HA Proxy stats

- Theo dõi log thu được của Loki trên Grafana.



Hình 107. Theo dõi log thu được của Loki trong kịch bản 1

- Theo dõi metric thu được của Prometheus trên Grafana.



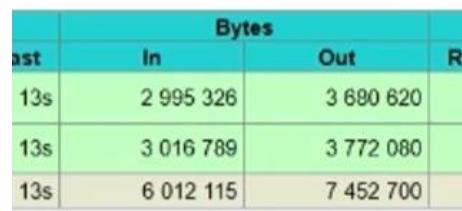
Hình 108. Theo dõi metric thu được của Prometheus trong kịch bản 1

## Các thông số đánh giá:

- Số lượng yêu cầu qua HA Proxy.

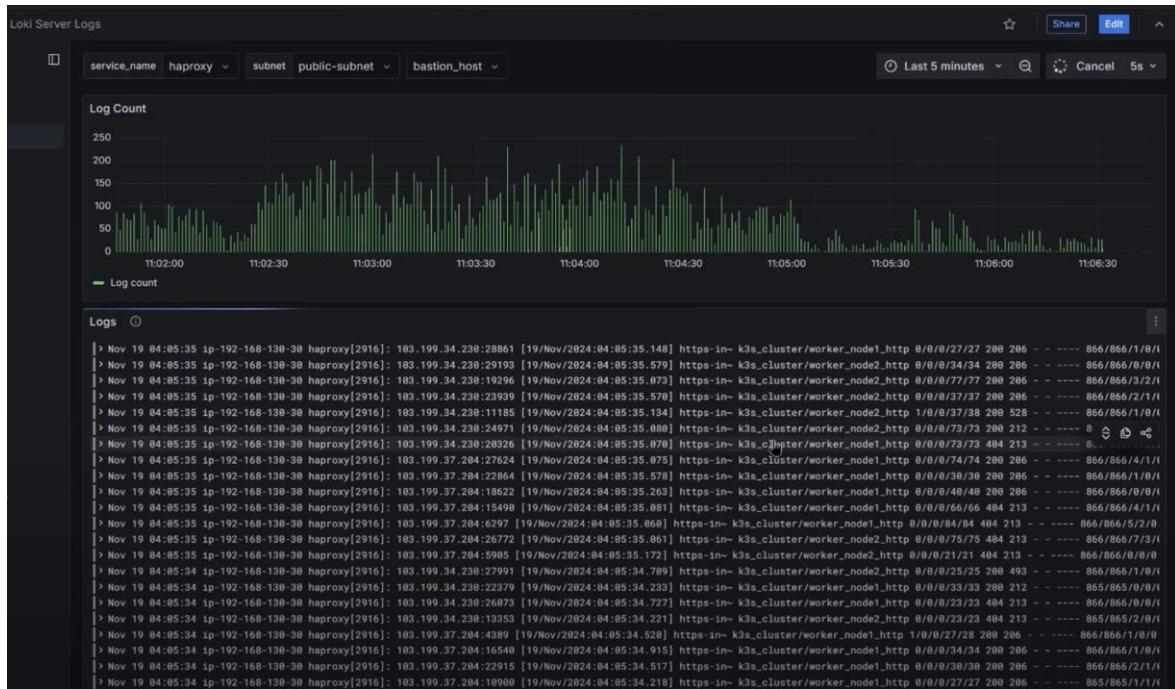
Hình 109. Số lượng yêu cầu qua HA Proxy đến các worker node

- Khả năng phân phối tải của HA Proxy.

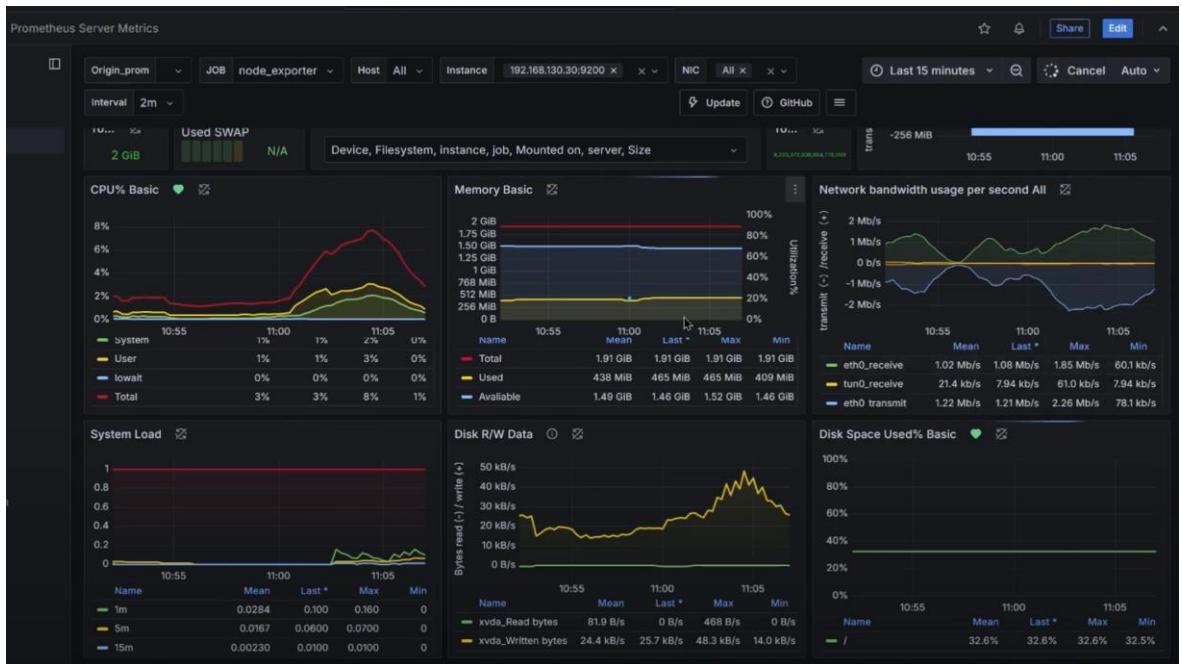


Hình 110. *Khả năng cân bằng tải của HA Proxy gần như bằng nhau*

- Khả năng thu thập log, metric real-time của Loki, Prometheus.



Hình 111. Kết quả thu thập log của Loki trong kịch bản 1



Hình 112. Kết quả thu thập metric của Prometheus trong kịch bản 1

### 3.5.2. Đánh giá tài nguyên của cụm K3s

Để đánh giá khả năng mở rộng, cân bằng tải, hiệu năng mạng và mức độ sử dụng tài nguyên của các node trong cụm K3s khi tăng số lượng replicas của ứng dụng, tiến hành kiểm tra chịu tải có mục đích đánh giá hiệu quả hoạt động của hệ thống khi ứng dụng được mở rộng và có lượng traffic lớn. Mô tả chi tiết các yêu cầu đánh giá như sau:

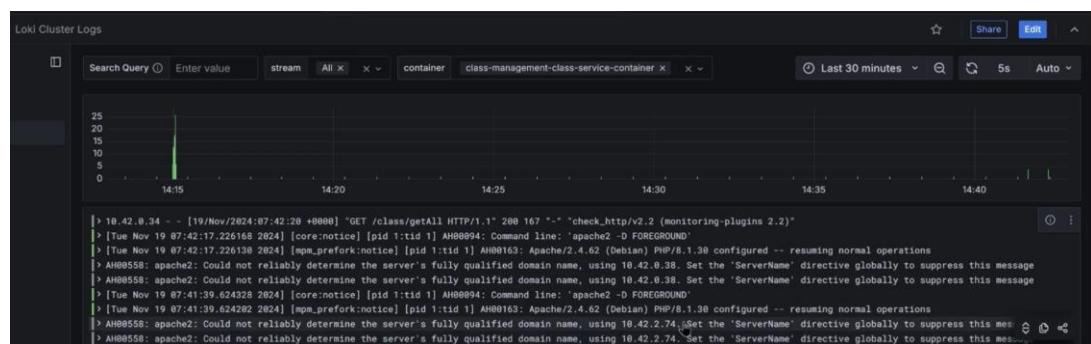
#### Quy trình kiểm thử:

- Tăng số lượng replicas
- Theo dõi quá trình tạo pod
- Đánh giá tài nguyên
- Giám sát:
  - Kiểm tra quá trình tạo pod bằng kubectl.

NAME	READY	STATUS	RESTARTS	AGE
auth-deployment-84d47ccb84-4zwb9	1/1	Running	1 (53m ago)	3h18m
auth-deployment-84d47ccb84-9rwql	1/1	Running	0	23s
auth-deployment-84d47ccb84-xjh5x	1/1	Running	0	23s
auth-mysql-deployment-f9c4abf4d5-8dwq6	1/1	Running	1 (53m ago)	3h10m
class-deployment-6977c845b-b4nxg	0/1	ContainerCreating	0	22s
class-deployment-6977c845b-jh5sp	1/1	Running	0	22s
class-deployment-6977c845b-pgtvc	1/1	Running	1 (53m ago)	3h10m
class-postgres-deployment-7bbaf5fc8d-7trg7	1/1	Running	1 (53m ago)	3h10m
frontend-deployment-7f89fd9484-7sf9b	0/1	ContainerCreating	0	21s
frontend-deployment-7f89fd9484-88sjh	1/1	Running	1 (53m ago)	3h10m
frontend-deployment-7f89fd9484-q8qfn	1/1	Running	0	21s
lecturer-deployment-7645d94fc-m8mfr	0/1	ContainerCreating	0	21s
lecturer-deployment-7645d94fc-q29b6	0/1	ContainerCreating	0	21s
lecturer-deployment-7645d94fc-r76jj	1/1	Running	1 (53m ago)	3h10m
lecturer-mongodb-deployment-66f6eb97b4-jcdcj	1/1	Running	1 (53m ago)	3h10m
student-deployment-f85fd4965-c494k	1/1	Running	0	20s
student-deployment-f85fd4965-fzzh2	0/1	ContainerCreating	0	20s
student-deployment-f85fd4965-sf88p	1/1	Running	4 (53m ago)	3h10m
student-mysql-deployment-98666fc68-2gxbr	1/1	Running	2 (53m ago)	3h10m

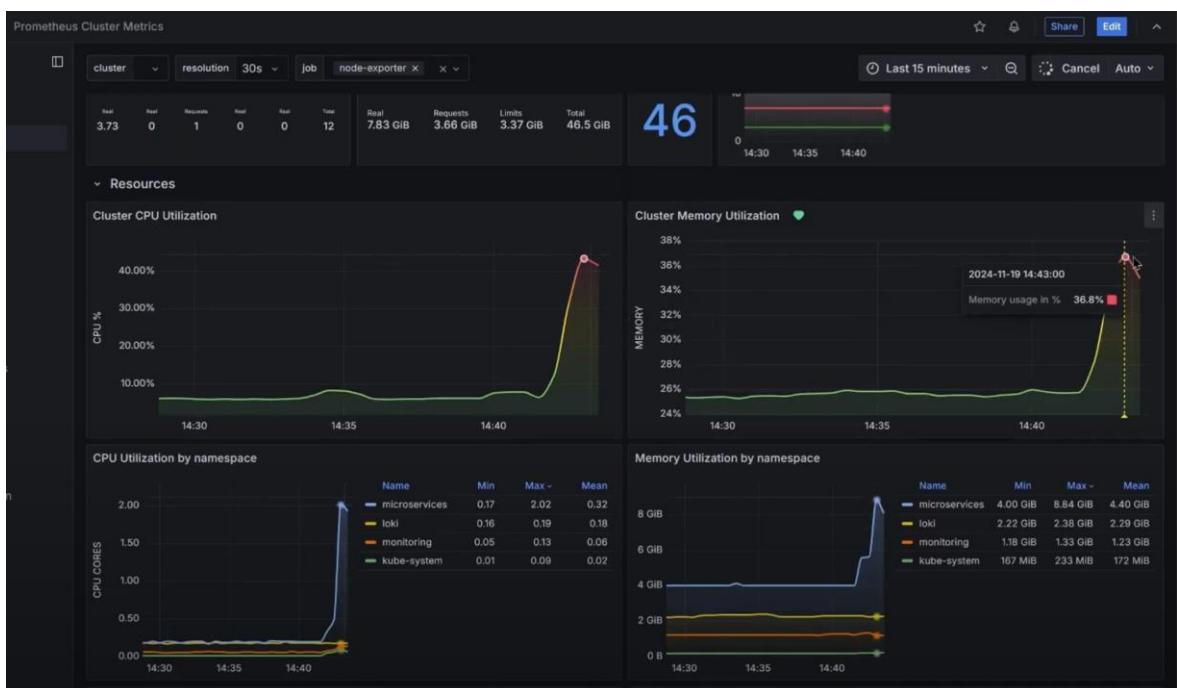
Hình 113. Kiểm tra quá trình tạo pod bằng kubectl

- Theo dõi log của Loki trên Grafana.



Hình 114. Theo dõi log thu được của Loki trong kịch bản 2

- Theo dõi metric thu được của Prometheus trên Grafana.



Hình 115. Theo dõi metric thu được của Prometheus trong kịch bản 2

## Các thông số đánh giá:

- Thời gian khi tất cả các pod được tạo thành công.

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
auth-deployment-84d47ccb48-4zwb9	1/1	Running	0 (104m ago)	3h59m	10.42.2.73	ip-192-168-110-12	<none>	<none>
auth-deployment-84d47ccb48-7lkkw	1/1	Running	0	5m46s	10.42.1.7	ip-192-168-110-11	<none>	<none>
auth-deployment-84d47ccb48-hjml	1/1	Running	0	5m46s	10.42.0.42	ip-192-168-110-10	<none>	<none>
auth-mysql-deployment-f9c4bf4d5-8dwq6	1/1	Running	1 (104m ago)	4h1m	10.42.1.246	ip-192-168-110-11	<none>	<none>
class-deployment-6977cc845b-kv4hg	1/1	Running	0	5m45s	10.42.2.78	ip-192-168-110-12	<none>	<none>
class-deployment-6977cc845b-ptgvc	1/1	Running	0 (104m ago)	4h1m	10.42.1.247	ip-192-168-110-11	<none>	<none>
class-deployment-6977cc845b-tkpp5	1/1	Running	0	5m45s	10.42.0.43	ip-192-168-110-10	<none>	<none>
class-postgresql-deployment-7bb8f5cf8d-7trg7	1/1	Running	1 (104m ago)	4h1m	10.42.1.252	ip-192-168-110-11	<none>	<none>
frontend-deployment-7f89fd9484-88sjh	1/1	Running	1 (104m ago)	4h1m	10.42.1.254	ip-192-168-110-11	<none>	<none>
frontend-deployment-7f89fd9484-svfmc	1/1	Running	0	5m44s	10.42.0.44	ip-192-168-110-10	<none>	<none>
frontend-deployment-7f89fd9484-w4jx2	1/1	Running	0	5m44s	10.42.2.79	ip-192-168-110-12	<none>	<none>
lecturer-deployment-7645d94fcfc-n5gmx	1/1	Running	0	5m43s	10.42.2.80	ip-192-168-110-12	<none>	<none>
lecturer-deployment-7645d94fcfc-r76j	1/1	Running	1 (104m ago)	4h1m	10.42.1.2	ip-192-168-110-11	<none>	<none>
lecturer-deployment-7645d94fcfc-w9rfb	1/1	Running	0	5m43s	10.42.0.45	ip-192-168-110-10	<none>	<none>
lecturer-mongodb-deployment-66fb6b697b4-jcdcj	1/1	Running	1 (104m ago)	4h1m	10.42.1.251	ip-192-168-110-11	<none>	<none>
student-deployment-f85fd4965-ctmvs	1/1	Running	0	5m42s	10.42.0.46	ip-192-168-110-10	<none>	<none>
student-deployment-f85fd4965-sf8bp	1/1	Running	4 (104m ago)	4h1m	10.42.1.4	ip-192-168-110-11	<none>	<none>
student-deployment-f85fd4965-vygrdr	1/1	Running	0	5m42s	10.42.2.81	ip-192-168-110-12	<none>	<none>
student-mysql-deployment-98666fc68-2gxbr	1/1	Running	2 (104m ago)	4h1m	10.42.1.249	ip-192-168-110-11	<none>	<none>

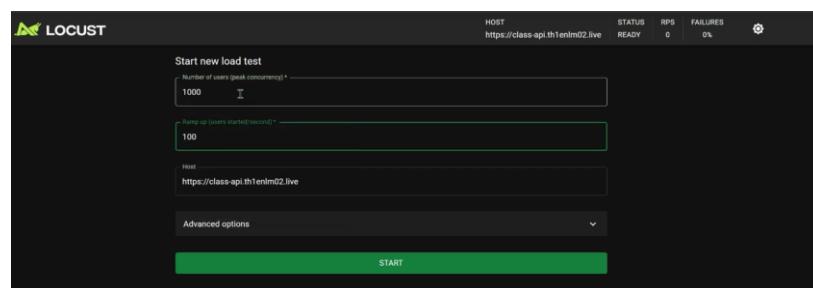
Hình 116. Các pod được phân phối đều trên các node

- Tài nguyên CPU, RAM được sử dụng khi tăng số lượng replicas.

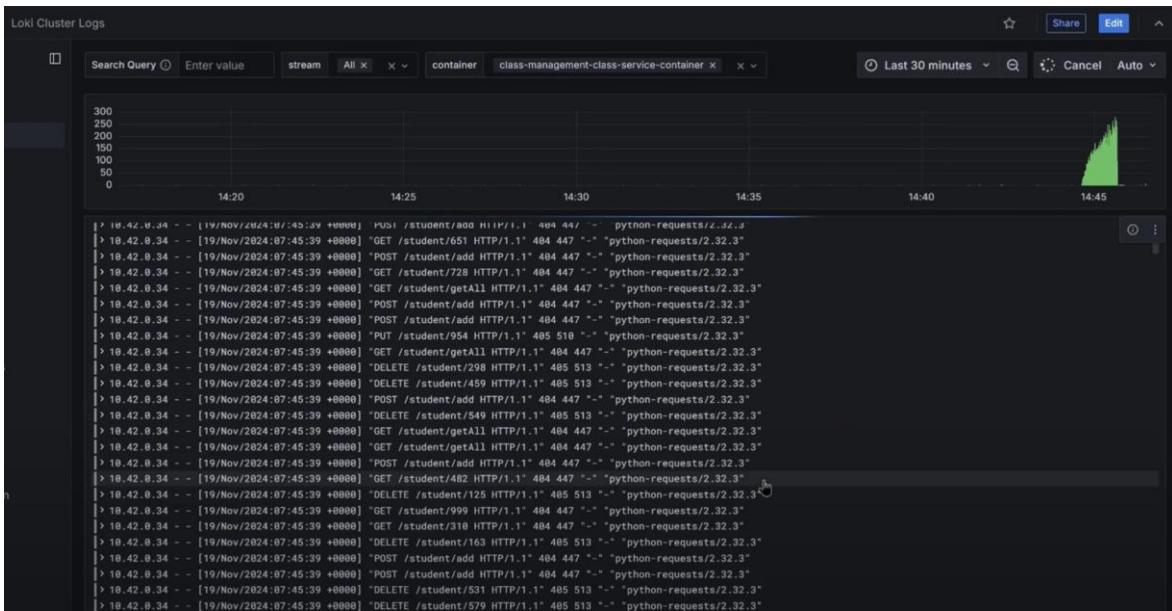


Hình 117. Tài nguyên CPU, RAM được sử dụng để chạy các pod mới trên từng node

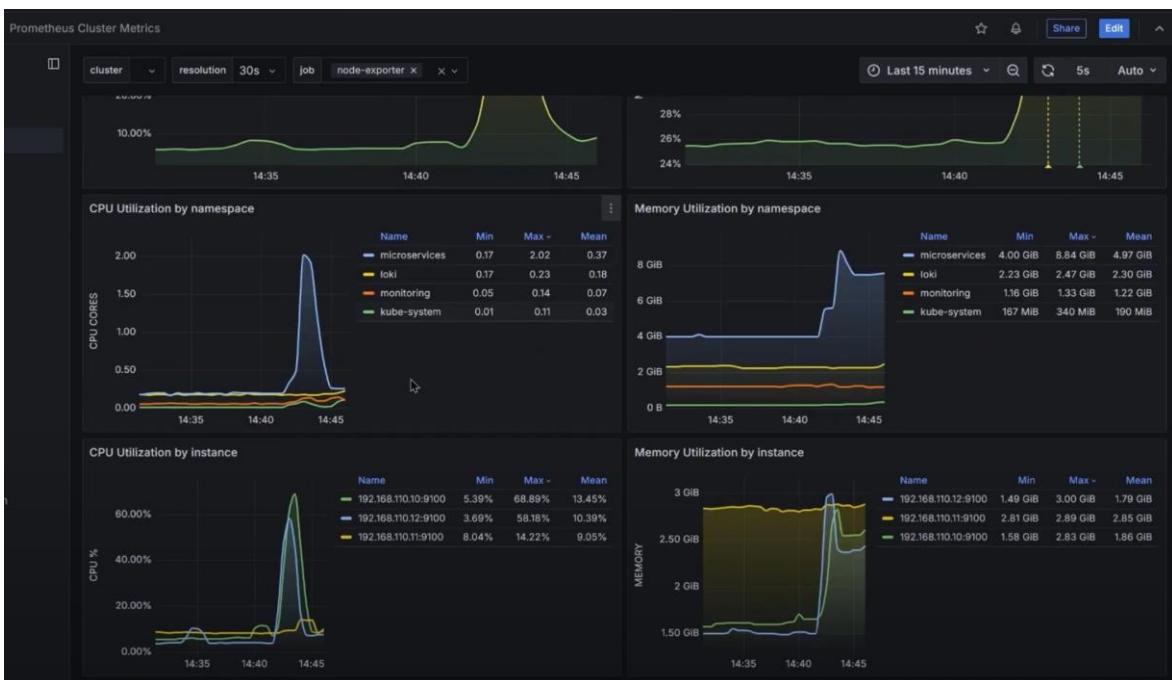
- Tạo traffic để đánh giá khả năng cân bằng tải giữa các node.



Hình 118. Tạo traffic bằng Locust



Hình 119. Kết quả thu thập log của Loki trong kịch bản 2



Hình 120. Kết quả thu thập metric của Prometheus trong kịch bản 2

### 3.5.3. Đánh giá hệ thống cảnh báo

#### 3.5.3.1. Kiểm tra cảnh báo khi tài nguyên bị quá tải

Mục tiêu của kịch bản này là kiểm tra khả năng phát hiện và gửi cảnh báo qua Telegram khi tài nguyên CPU và RAM của hệ thống vượt ngưỡng đã cấu hình.

Đồng thời, đánh giá hệ thống giám sát khi ứng dụng trong cụm K3s phải xử lý lượng lớn yêu cầu API.

### Công cụ sử dụng:

- Locust: Tạo tải bằng cách gọi API của ứng dụng.
- stress-ng: Tạo tải giả lập trực tiếp lên CPU và RAM của các server.
- Loki, Prometheus trên Grafana: Theo dõi log và metric hệ thống.

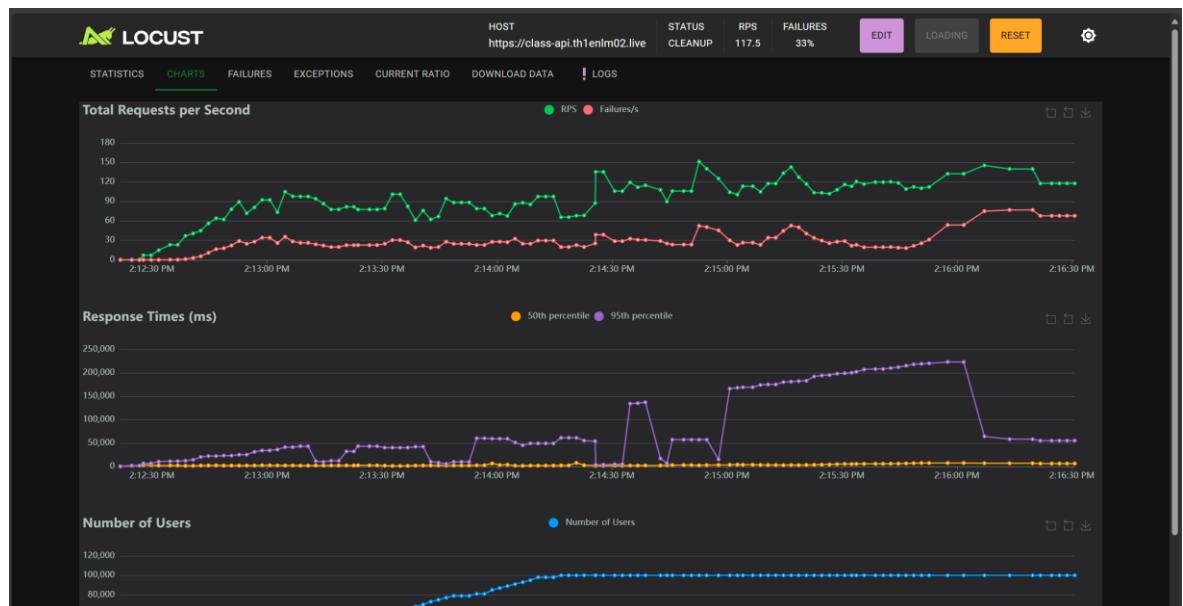
### Quy trình kiểm thử:

*Gọi API vào service trong cụm K3s:*

- Chạy tập lệnh Locust để liên tục gửi yêu cầu HTTP đến API của ứng dụng, với số lượng người dùng tăng dần.

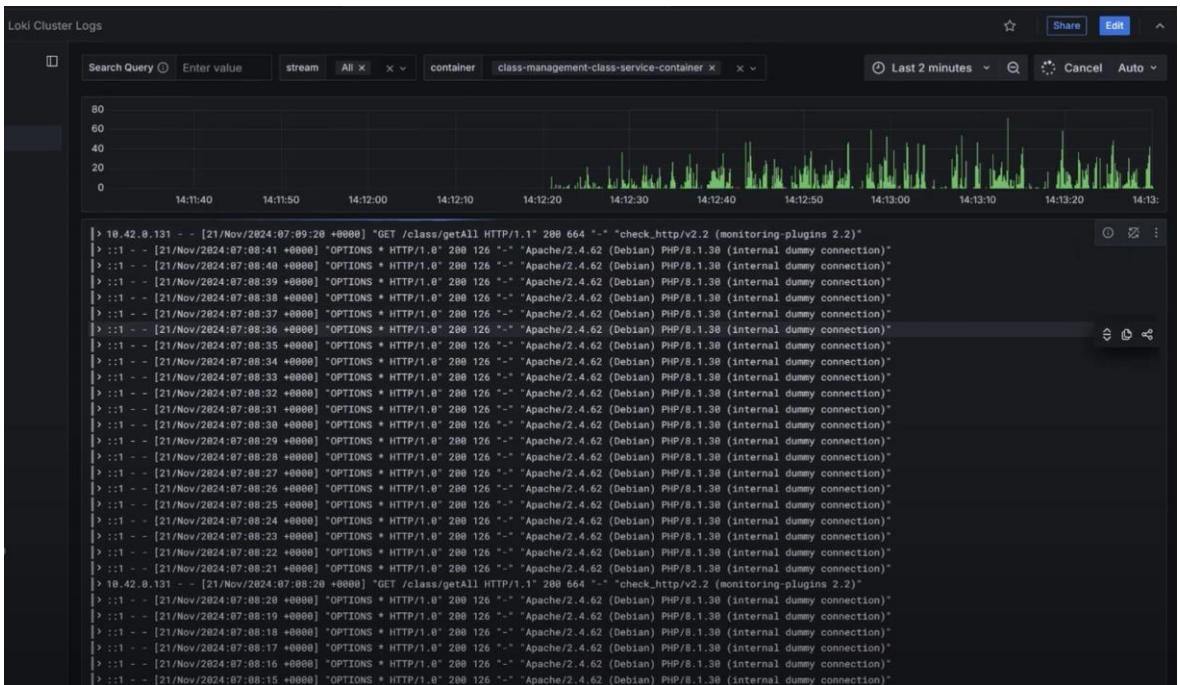
❯ kubectl -o wide											
NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES			
auth-deployment-6b968bc45-52hmt	1/1	Running	2 (3h46m ago)	15h	10.42.1.154	ip-192-168-110-11	<none>	<none>			
auth-mysql-deployment-f9c4bf4d5-w5kfc	1/1	Running	2 (3h46m ago)	15h	10.42.1.151	ip-192-168-110-11	<none>	<none>			
class-deployment-6fc6f757b6-n5tq2	1/1	Running	0	18m	10.42.2.162	ip-192-168-110-12	<none>	<none>			
class-postgresql-deployment-7bb8f5cf8d-9cpkt	1/1	Running	2 (3h46m ago)	15h	10.42.1.161	ip-192-168-110-11	<none>	<none>			
frontend-deployment-7f89fd9484-kjb4	1/1	Running	1 (3h46m ago)	15h	10.42.2.155	ip-192-168-110-12	<none>	<none>			
lecturer-deployment-7645d94cfc-kcq2l	1/1	Running	2 (3h46m ago)	15h	10.42.1.146	ip-192-168-110-11	<none>	<none>			
lecturer-mongodb-deployment-66f6b697b4-l28qb	1/1	Running	2 (3h46m ago)	15h	10.42.1.159	ip-192-168-110-11	<none>	<none>			
student-deployment-f85fd4965-pdhdm	1/1	Running	3 (3h46m ago)	15h	10.42.1.156	ip-192-168-110-11	<none>	<none>			
student-mysql-deployment-98666fc68-zh8xw	1/1	Running	2 (3h46m ago)	15h	10.42.1.153	ip-192-168-110-11	<none>	<none>			

Hình 121. Kiểm tra thông tin node đang chạy pod của service



Hình 122. Biểu đồ mô phỏng của Locust

- Theo dõi các log và metric của ứng dụng trên Grafana.

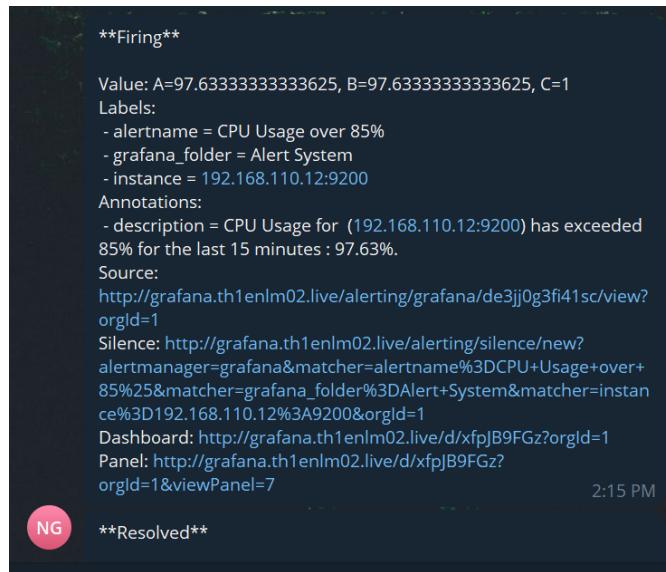


Hình 123. Theo dõi log thu được của service



Hình 124. Theo dõi metric thu được của node đang chạy pod của service đó

- Quan sát các cảnh báo Telegram được gửi từ Prometheus khi các người dùng được kích hoạt (CPU, RAM).



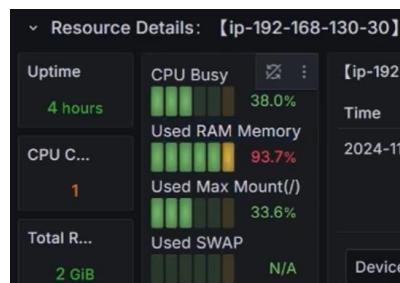
*Hình 125. Cảnh báo được gửi đến Telegram khi CPU của node vượt ngưỡng Gây tải trên server:*

- Chọn một hoặc nhiều server trong hạ tầng để chạy stress-ng.
- Chạy lệnh stress-ng để tạo tải CPU và RAM:

```
stress-ng --vm 2 --vm-bytes 95% --timeout 300s
```

```
ubuntu@ip-192-168-130-30:~$ stress-ng --vm 2 --vm-bytes 95% --timeout 300s
stress-ng: info: [3433] dispatching hogs: 2 vm
```

- Hình 126. Sử dụng stress-ng để gây tải cho server được chọn*
- Theo dõi các log và metric hệ thống của server trên Grafana, trong đó chú ý 2 thông số:
    - CPU Usage (%).
    - Memory Usage (%).

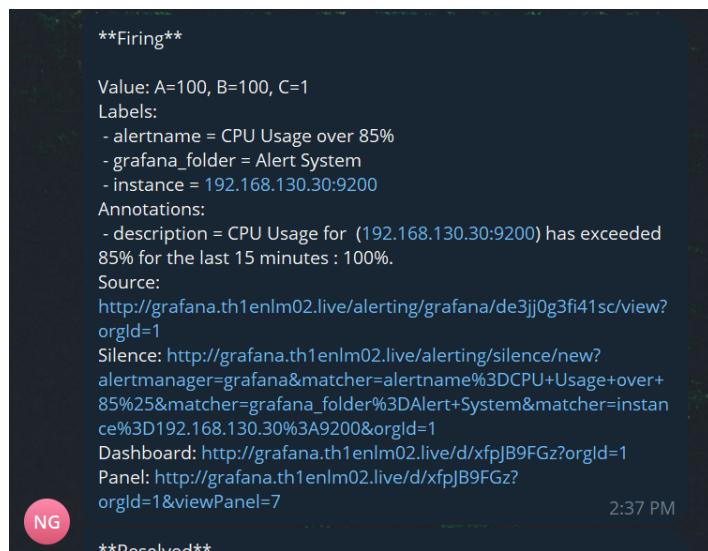


*Hình 127. Theo dõi thông số CPU, RAM*

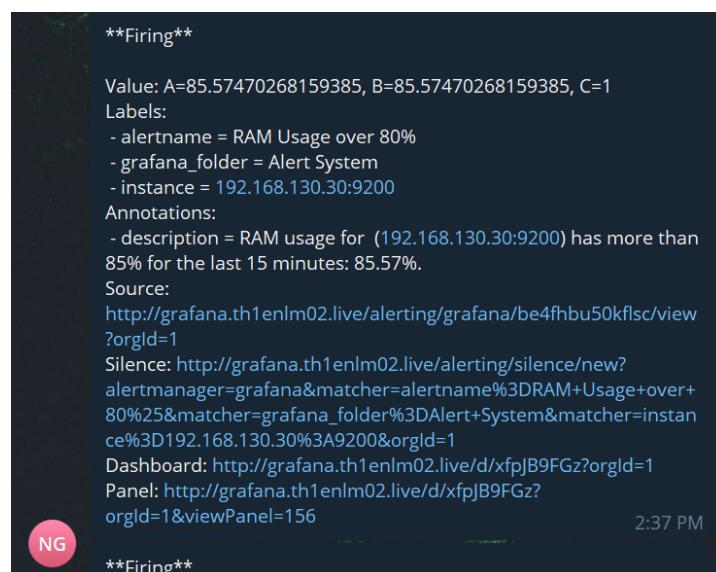
Alert System > Alerting					
State	Name	Health	Summary	Next evaluation	Actions
> <span style="background-color: pink;">Firing</span> for 25s	CPU Usage over 85%	ok		within 10s	<span style="color: pink;">@</span> <span style="color: pink;">Edit</span> More
> <span style="background-color: pink;">Firing</span> for 3h 17m	DISK over 80%	ok		in a few seconds	<span style="color: pink;">@</span> <span style="color: pink;">Edit</span> More
> <span style="background-color: pink;">Firing</span> for 25s	RAM Usage over 80%	ok		in a few seconds	<span style="color: pink;">@</span> <span style="color: pink;">Edit</span> More

Hình 128. Theo dõi trạng thái cảnh báo khi CPU, RAM vượt ngưỡng

- Quan sát các cảnh báo Telegram được gửi từ Prometheus khi CPU hoặc RAM vượt ngưỡng đã cấu hình.



Hình 129. Cảnh báo được gửi đến Telegram khi server vượt ngưỡng CPU



Hình 130. Cảnh báo được gửi đến Telegram khi server vượt ngưỡng RAM

## Các thông số đánh giá:

- Kiểm tra độ chính xác của cảnh báo khi CPU, RAM vượt ngưỡng.
- Đảm bảo nội dung cảnh báo rõ ràng và đúng nguồn.

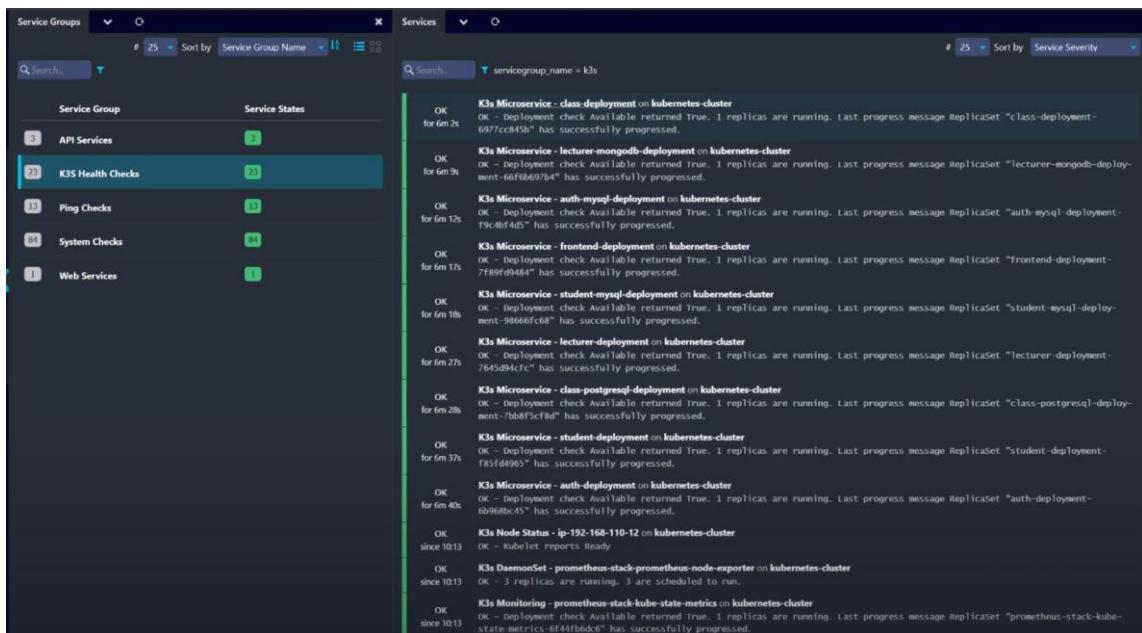
### 3.5.3.2. Kiểm tra cảnh báo khi dịch vụ hoặc server không khả dụng

Kịch bản này nhằm sử dụng Icinga theo dõi trạng thái và kiểm tra khả năng phát hiện và gửi cảnh báo qua Telegram khi:

- Một hoặc nhiều dịch vụ trong K3s bị ngừng hoạt động.
- Một hoặc nhiều server trong hạ tầng bị tắt hoặc mất kết nối.

#### Quy trình kiểm thử:

Dừng dịch vụ trong K3s:



Hình 131. Trạng thái healthcheck trước khi chạy của K3s và các endpoint

- Chạy lệnh ‘kubectl delete’ để dừng các service đã triển khai.

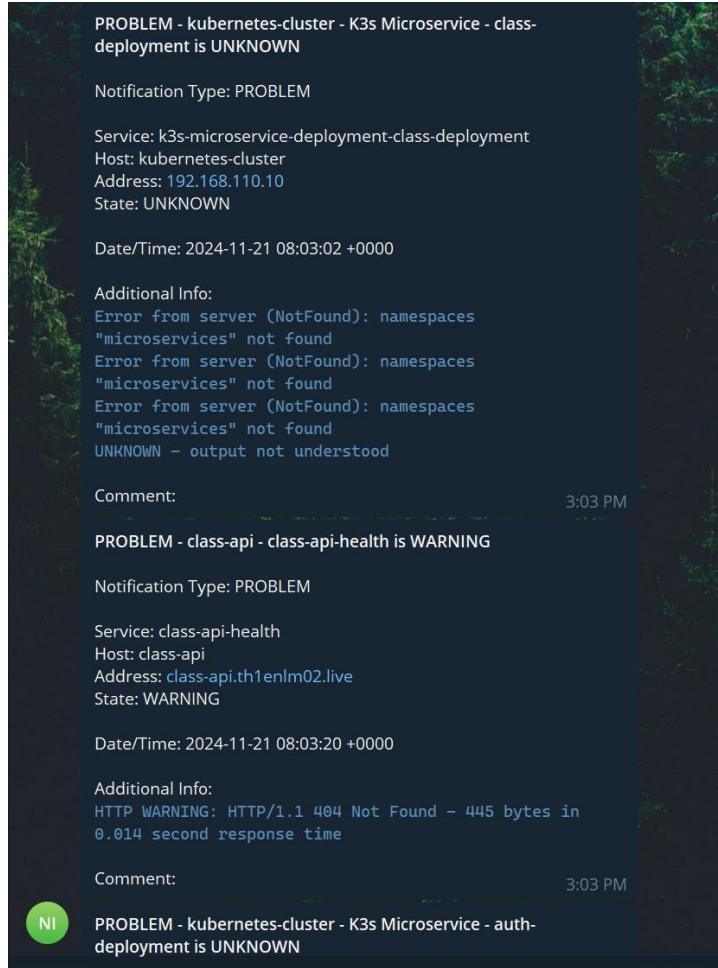
```
> kubectl delete -k .
namespace "microservices" deleted
configmap "auth-mysql-configmap" deleted
configmap "auth-mysql-initdb" deleted
configmap "base-url-configmap" deleted
configmap "class-postgresql-configmap" deleted
configmap "class-postgresql-initdb" deleted
configmap "lecturer-mongodb-configmap" deleted
configmap "student-configmap" deleted
configmap "student-mysql-configmap" deleted
```

Hình 132. Dừng tất cả dịch vụ trên cụm K3s

- Quan sát trên Icinga để xác minh rằng trạng thái dịch vụ được cập nhật.

Hình 133. Trạng thái healthcheck thay đổi khi dừng tất cả dịch vụ

- Kiểm tra thông báo cảnh báo trên Telegram, đảm bảo nội dung rõ ràng và đúng nguồn gốc.



Hình 134. Cảnh báo được gửi đến Telegram về trạng thái healthcheck của dịch vụ

## Tắt server:

- Chọn một hoặc nhiều server trong hạ tầng đang được Icinga giám sát.

The screenshot shows two panels of the Icinga 2 Web interface. The left panel displays 'Service Groups' with items like API Services, K3S Health Checks, Ping Checks (selected), System Checks, and Web Services. The right panel shows 'Services' with a search bar for 'servicegroup\_name = ping'. It lists several services with their status (OK or PING OK), name, and recent check details. For example, 'ping4 on frontend' was OK since 10:18 with a RTA of 1.75 ms. Another entry shows 'ping4 on ip-192-168-120-20.ec2.internal' was OK since Nov 9 with a RTA of 0.03 ms.

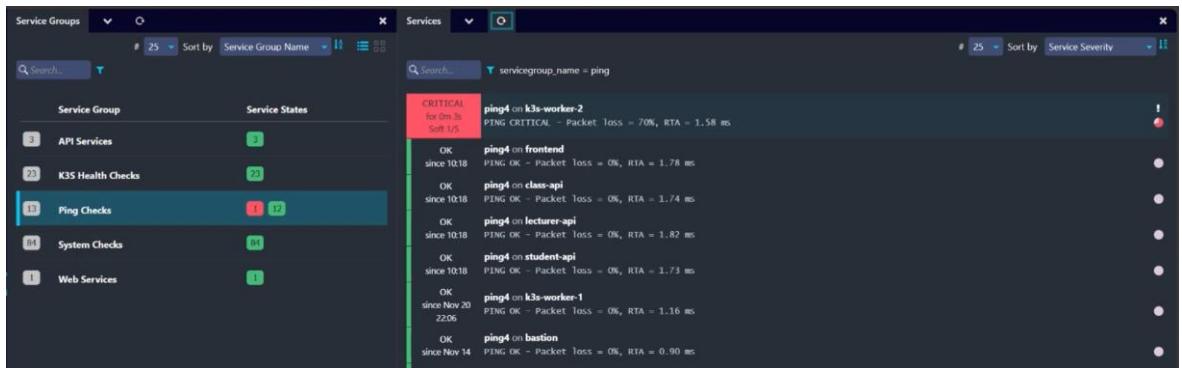
Hình 135. Trạng thái của server trước khi tắt

- Ngừng server thông qua giao diện quản lý hoặc dòng lệnh.

The screenshot shows the AWS CloudWatch Instances console. The top section lists instances with columns for Name, Instance ID, Instance state, Instance type, Status check, Alarm status, Availability Zone, and Public IPv4 DNS. An instance named 'NT531.P11-cluster-instance-2' is selected, shown in the bottom expanded view. This view includes tabs for Details, Status and alarms, Monitoring, Security, Networking, Storage, and Tags. Under 'Details', there's an 'Instance summary' section with fields for Instance ID (i-04ce18cc2072cf19e), Public IPv4 address (192.168.110.12), Private IPv4 addresses (192.168.110.12), Instance state (Stopping), Public IPv4 DNS (192.168.110.12), Hostname type (IP name: ip-192-168-110-12.ec2.internal), Private IP DNS name (ip-192-168-110-12.ec2.internal), Instance type (t2.large), and Elastic IP addresses (none).

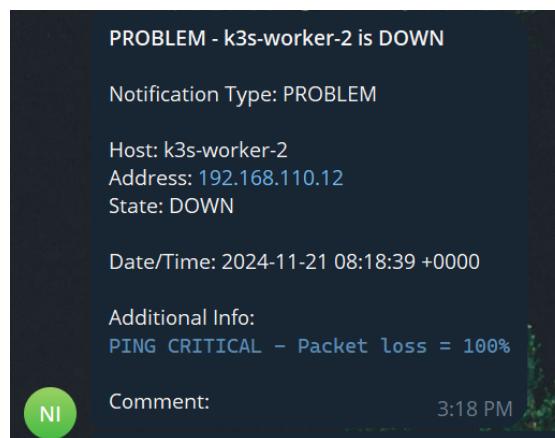
Hình 136. Tắt server trên AWS console

- Quan sát trên Icinga để kiểm tra trạng thái của server.



Hình 137. Trạng thái của server được cập nhật lại

- Xác minh rằng cảnh báo Telegram được gửi khi server bị ngắt kết nối.



Hình 138. Cảnh báo được gửi đến Telegram về trạng thái healthcheck của server

#### Các thông số đánh giá:

- Đảm bảo cảnh báo được gửi đúng khi service hoặc server ngừng hoạt động.
- Kiểm tra nội dung cảnh báo, bao gồm thông tin chính xác về service hoặc server gặp sự cố.
- Theo dõi thời gian từ khi dịch vụ hoặc server ngừng hoạt động đến khi nhận được cảnh báo Telegram.

## Chương 4. KẾT LUẬN

### 4.1. Kết quả đạt được

#### 4.1.1. So sánh

**Bảng 2. Bảng so sánh chi tiết các công cụ giám sát**

Tiêu chí	Icinga	Prometheus/Grafana
Kiến trúc		
Mô hình	Push & Pull based	Pull based
Storage	MySQL/PostgreSQL/...	Time-series DB
Agent	Icinga agent	Exporters
Khả năng Giám sát		
Metrics	Host/Service checks	Time-series data
Độ trễ	1-5 phút	10-15 giây
Custom checks	Plugins (Nagios compatible)	Custom exporters
Performance		
CPU Usage	Thấp (2-5%)	Trung bình (5-15%)
RAM Usage	256MB-1GB	1GB-8GB
Disk I/O	Thấp	Cao
Tính năng		
Querying	Basic filters	PromQL (powerful)
Visualization	Basic Web UI	Grafana integration
Alerting	Built-in, phức tạp	AlertManager, linh hoạt
Khả năng mở rộng		
Horizontal scaling	Khó, cần cấu hình master-slave thủ công, sync config phức tạp	Dễ dàng, chỉ cần thêm Prometheus server mới và config service discovery
Vertical scaling	Dễ, chỉ cần tăng resource cho master node	Trung bình, cần tính toán retention time, storage
High Availability	Đơn giản với mô hình master-slave truyền thống	Phức tạp hơn với federation, thanos, cortex
Tích hợp		
Kubernetes	Bị giới hạn, cần cài đặt thêm plugins, config thủ công	Native support, có sẵn service discovery, auto-config với K8s

Cloud platforms	Cần viết plugins cho từng cloud service	Native integrations, có sẵn exporters cho AWS, GCP, Azure
Third-party tools	Nhiều, chủ yếu plugins từ Nagios	Rất nhiều, hàng nghìn exporters có sẵn, cộng đồng lớn
Triển khai & Quản lý		
Setup time	2-3 ngày, cấu hình chi tiết, rõ ràng qua file config	1-2 ngày, nhanh hơn vì có nhiều tools hỗ trợ
Learning curve	Cao, cấu trúc commands, services rõ ràng	Trung bình, dễ học hơn với PromQL, docs chi tiết
Documentation	Tài liệu ổn định, ít thay đổi	Cập nhật thường xuyên, nhiều examples
Chi phí		
License	Open source	Open source
Maintenance	Trung bình	Ít tốn công maintain vì auto-discovery
Training	Phức tạp hơn về config, plugins, kiến thức sâu về monitoring	Trung bình. PromQL syntax dễ học, tương tự SQL. Cấu hình đơn giản, YAML-based.
Use Cases		
Traditional IT	Rất tốt, vì tập trung vào monitor services	Tốt
Cloud Native	Hạn chế, vẫn dùng được thông qua plugins	Rất tốt vì native support, auto-scaling, service discovery
Microservices	Không phù hợp	Rất phù hợp

#### 4.1.2. Đánh giá chung

**Về hiệu năng giám sát và khả năng thực tế:** Prometheus nổi bật với khả năng thu thập metrics một cách chi tiết và gần như real-time. Hệ thống sử dụng cơ chế pull-based để lấy metrics từ các targets, cho phép theo dõi chi tiết về CPU, memory, disk I/O, network traffic và nhiều metrics khác. Điều này đặc biệt hữu ích trong việc phát hiện sớm các vấn đề về hiệu năng. Trong khi đó, Icinga tập trung vào việc kiểm tra trạng thái của services và hosts theo chu kỳ định kỳ. Mặc

dù cách tiếp cận này đơn giản hơn, nhưng nó có thể dẫn đến độ trễ trong việc phát hiện vấn đề và ít chi tiết hơn trong việc phân tích nguyên nhân gốc rễ.

**Khả năng mở rộng và tích hợp:** Trong môi trường doanh nghiệp hiện đại, Prometheus cho thấy ưu thế vượt trội về khả năng mở rộng. Nó có thể dễ dàng scale horizontally thông qua việc thêm nhiều Prometheus servers và sử dụng service discovery để tự động phát hiện và monitor các targets mới. Đặc biệt trong môi trường Kubernetes, Prometheus hoạt động như "first-class citizen" với khả năng tự động discover và monitor pods, services mới. Ngược lại, Icinga với cấu hình tĩnh truyền thống gặp khó khăn trong việc thích ứng với môi trường dynamic. Việc thêm mới hosts hay services thường đòi hỏi cấu hình tay, làm tăng gánh nặng quản trị trong môi trường lớn.

**Chi phí và độ phức tạp trong vận hành:** Về mặt chi phí vận hành, Icinga thường có lợi thế hơn trong môi trường nhỏ và trung bình. Nó có overhead thấp hơn và không đòi hỏi nhiều tài nguyên như Prometheus. Tuy nhiên, khi xét về long-term, Prometheus lại có lợi thế về automation và self-service capabilities, giúp giảm ánh hưởng trong việc cấu hình tay trong quá trình vận hành. Prometheus cũng có ecosystem phong phú với nhiều exporters có sẵn, trong khi với Icinga, việc phát triển plugins custom có thể tốn nhiều thời gian và công sức hơn.

**Khả năng phân tích và troubleshooting:** Prometheus với PromQL cung cấp công cụ mạnh mẽ để phân tích metrics và troubleshoot issues. Các operators có thể viết các queries phức tạp để phân tích trends, correlate metrics và tạo các alerts thông minh. Icinga, mặc dù đơn giản hơn trong việc setup alerts, lại hạn chế trong khả năng phân tích sâu. Việc troubleshooting với Icinga thường đòi hỏi thêm các tools bổ sung.

## 4.2. Kết luận

Trong môi trường công nghệ hiện đại, việc lựa chọn công cụ giám sát phù hợp đóng vai trò quan trọng đối với sự ổn định và hiệu quả của hệ thống. Qua việc so sánh chi tiết giữa Icinga và Prometheus, có thể thấy mỗi công cụ đều có những

ưu điểm và use cases riêng biệt. Icinga nổi bật với khả năng giám sát truyền thống và hiệu quả trong môi trường on-premise, đặc biệt phù hợp cho các hệ thống legacy với yêu cầu monitoring cơ bản. Trong khi đó, Prometheus thể hiện ưu thế vượt trội trong môi trường cloud-native với khả năng auto-discovery, scaling linh hoạt và tích hợp native với Kubernetes. Việc kết hợp cả hai công cụ trong môi trường hybrid có thể tận dụng được ưu điểm của cả hai platform, trong đó Icinga đảm nhiệm vai trò giám sát services truyền thống, còn Prometheus tập trung vào metrics collection và phân tích performance trong môi trường container hóa. Xu hướng hiện tại nghiêng về việc sử dụng Prometheus trong các dự án mới, đặc biệt là các dự án cloud-native, trong khi Icinga vẫn duy trì vị thế trong các hệ thống enterprise truyền thống.

## TÀI LIỆU THAM KHẢO

- [1] <https://prometheus.io/docs/introduction/overview/>
- [2] <https://grafana.com/docs/loki/latest/>
- [3] <https://grafana.com/docs/>
- [4] <https://icinga.com/docs/>
- [5] <https://registry.terraform.io/providers/hashicorp/aws/latest/docs>
- [6] <https://docs.ansible.com/ansible/latest/index.html>

## PHỤ LỤC

**Bảng 3. Bảng phân chia nhiệm vụ**

Tên thành viên	Nhiệm vụ	Tỉ lệ hoàn thành
Lưu Minh Thiện	<ul style="list-style-type: none"> <li>- Thiết kế cơ sở hạ tầng</li> <li>- Viết script Terraform, Ansible, Kubernetes manifest file</li> <li>- Triển khai ứng dụng web</li> <li>- Cấu hình HA Proxy</li> <li>- Triển khai và troubleshoot cụm K3s</li> <li>- Cấu hình Prometheus, Loki, Grafana</li> <li>- Chuẩn bị slide thuyết trình</li> <li>- Viết báo cáo</li> </ul>	100%
Quách Thị Hoài Phương	<ul style="list-style-type: none"> <li>- Thiết kế cơ sở hạ tầng</li> <li>- Viết script Terraform, Ansible, Locust</li> <li>- Triển khai cụm K3s</li> <li>- Cấu hình Icinga, Grafana</li> <li>- Cấu hình alert cho Icinga</li> <li>- Chuẩn bị slide thuyết trình</li> <li>- Viết báo cáo</li> </ul>	100%
Nguyễn Thành Đăng	<ul style="list-style-type: none"> <li>- Thiết kế cơ sở hạ tầng</li> <li>- Viết script Ansible</li> <li>- Cấu hình HA Proxy</li> <li>- Cấu hình OpenVPN</li> <li>- Cấu hình Prometheus, Loki, Grafana</li> <li>- Cấu hình alert cho Prometheus, Grafana</li> <li>- Viết báo cáo</li> </ul>	100%

**Link slides:**

<https://docs.google.com/presentation/d/1mJlhgMbYEkSuAt63EQdRrTYAWP2iGltd>

**Link video demo chi tiết:** <https://youtu.be/bETCRAICRug>

**Link GitHub:** <https://github.com/NT531-P11-Monitoring-Tools>