MỤC LỤC

LỜI MỞ ĐẦU

Trong bối cảnh công nghệ ngày càng phát triển và yêu cầu về việc duy trì hoạt động ổn định của các hệ thống phần mềm trở nên ngày càng quan trọng, việc giám sát và quản lý tình trạng "sức khoẻ" của các dịch vụ trong hạ tầng là yếu tố không thể thiếu. Một hệ thống giám sát hiệu quả không chỉ giúp phát hiện sự cố kịp thời mà còn cung cấp thông tin chi tiết về tài nguyên hệ thống, từ đó hỗ trợ người quản trị đưa ra các quyết định đúng đắn nhằm tối ưu hóa hiệu suất và đảm bảo tính liên tục của các dịch vụ.

Bài toán được đặt ra là xây dựng một hệ thống giám sát sức khoẻ cho các dịch vụ trong môi trường Docker, bao gồm việc giám sát tình trạng hoạt động của các container và các endpoint API, cũng như theo dõi tài nguyên hệ thống như RAM, bộ nhớ, ổ cứng và băng thông mạng. Hệ thống giám sát còn cần cung cấp các biểu đồ lưu lượng truy cập, giúp người quản trị dễ dàng theo dõi và nắm bắt tình hình trong thời gian thực.

Cụ thể, hệ thống giám sát được triển khai với hai container được chạy trong Docker: một chứa API cung cấp thông tin giá vàng tại Việt Nam và một chứa API cập nhật tỷ giá ngoại tệ so với VNĐ. Mục tiêu là xây dựng một giải pháp giám sát toàn diện, có thể theo dõi tình trạng sức khỏe của các container và các endpoint API, đồng thời hiển thị dữ liệu tài nguyên và lưu lượng truy cập theo thời gian thực.

Báo cáo này sẽ mô tả chi tiết quá trình xây dựng cũng như cách thức triển khai và cài đặt hệ thống giám sát, đồng thời đánh giá hiệu quả và tính khả thi của giải pháp trong môi trường thực tế.

Từ khóa: *Container, Giám sát sức khỏe, Theo dõi tài nguyên hệ thống, Giám sát tình trạng hoạt động*

I. Giới thiệu chung

1. Đặt vấn đề

Trong quá trình quản lý các hệ thống dịch vụ, đặc biệt là các dịch vụ web và API chạy trong môi trường như Docker, một trong những thách thức lớn là giám sát tình trạng "sức khoẻ" của các dịch vụ đó. Các sự cố phát hiện muộn có thể dẫn đến gián đoạn dịch vụ và ảnh hưởng đến trải nghiệm sử dụng. Do vậy, cần có công cụ giám sát hiệu quả, giúp rút ngắn thời gian phát hiện lỗi, cảnh báo khi gặp sự cố và khôi phục dịch vụ.

Để giải quyết những vấn đề trên, một giải pháp giám sát được áp dụng qua mẫu thiết kế Health Endpoint Monitoring là rất cần thiết. Health Endpoint Monitoring[1] cung cấp khả năng giám sát trạng thái các dịch vụ và các API endpoint trong thời gian thực, giúp giám sát tài nguyên hệ thống và cung cấp các biểu đồ theo dõi lưu lượng truy cập; từ đó giúp giải quyết các vấn đề tồn tại trong quá trình giám sát và quản lý hệ thống, đồng thời đảm bảo sự ổn định và hiệu quả của toàn bộ hệ thống dịch vụ.

2. Mục tiêu của hệ thống

Mục tiêu chính của hệ thống giám sát sức khỏe là cung cấp một giải pháp toàn diện và hiệu quả, giúp theo dõi và quản lý tình trạng hoạt động các dịch vụ và tài nguyên hệ thống trong thời gian thực. Cụ thể, hệ thống sẽ tập trung vào các mục tiêu:

- **Giám sát tình trạng hoạt động của các container Docker:** Hệ thống sẽ theo dõi và giám sát tình trạng "up/down" của các container Docker; đảm bảo rằng tất cả các container chạy các dịch vụ API đều được kiểm tra thường xuyên và nhận được cảnh báo trước và trong khi gặp sự cố, đảm bảo các dịch vụ luôn sẵn sàng hoạt động.
- **Giám sát tình trạng hoạt động của các API endpoints:** Hệ thống sẽ giám sát trạng thái của các API endpoints. Bằng cách kiểm tra thường xuyên, hệ thống có thể phát hiện các lỗi hoặc sự cố tại các endpoint, cung cấp thông tin về độ trễ của API, đảm bảo hoạt động ổn định, cung cấp dữ liệu chính xác cho người dùng cuối và đánh giá hiệu suất của các dịch vụ.
- **Theo dõi các chỉ số tài nguyên hệ thống:** Mục tiêu tiếp theo là giám sát các chỉ số tài nguyên hệ thống, gồm bộ nhớ (RAM), CPU, ổ cứng (Disk) và băng thông mạng; giúp phát hiện kịp thời tình trạng quá tải tài nguyên hoặc các vấn đề hiệu suất trong quá trình vận hành, tối ưu hóa phân bổ tài nguyên cho các container và dịch vụ, tránh tình trạng quá tải gây ra sự cố hoặc gián đoạn dịch vụ.

- **Xây dựng biểu đồ lưu lượng truy cập:** Một mục tiêu quan trọng khác của hệ thống là xây dựng và hiển thị các biểu đồ lưu lượng truy cập API; giúp người quản trị dễ dàng theo dõi sự thay đổi trong lượng truy cập theo thời gian thực, phân tích các xu hướng sử dụng dịch vụ, và phát hiện các đột biến trong lưu lượng truy cập có thể ảnh hưởng đến hiệu suất của hệ thống.

3. Tổng quan về hạ tầng cần giám sát

Hạ tầng giám sát sử dụng Docker để triển khai các container chứa các API endpoint, giúp đóng gói ứng dụng và dịch vụ cùng với các phụ thuộc vào môi trường tách biệt. Hai container này chạy độc lập, mỗi container đảm nhiệm một dịch vụ API cụ thể và có thể dễ dàng triển khai, dừng hoặc cập nhật mà không làm gián đoạn hoạt động của các dịch vụ khác trong hệ thống, đồng thời cũng dễ dàng mở rộng hạ tầng khi có nhu cầu.

Container đầu tiên chứa một API endpoint cung cấp thông tin về giá vàng tại Việt Nam từ các công ty SJC, DOJI và PNJ theo từng thời điểm. Container này sẽ được giám sát để đảm bảo API hoạt động ổn định, không có lỗi và cung cấp thông tin chính xác, đồng thời theo dõi tình trạng tài nguyên hệ thống của container.

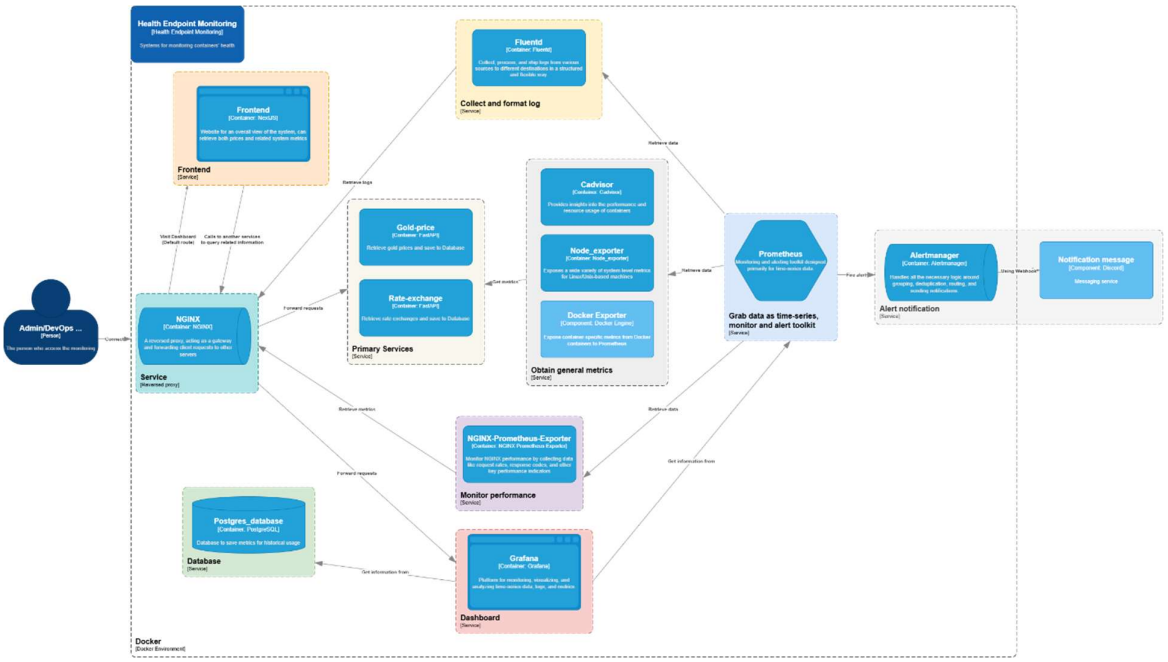
Container thứ hai chứa một dịch vụ API cung cấp thông tin về tỷ giá ngoại tệ so với Việt Nam Đồng (VNĐ) từ các ngân hàng. Tương tự như container chứa API giá vàng, container này cũng sẽ được giám sát để đảm bảo API luôn hoạt động chính xác và hiệu quả. Dữ liệu tỷ giá cần được cập nhật thường xuyên để đáp ứng nhu cầu sử dụng và hệ thống giám sát sẽ theo dõi tình trạng hoạt động để phát hiện kịp thời các vấn đề.

II. Kiến trúc tổng thể của hệ thống, các công nghệ sử dụng và ứng dụng mẫu thiết kế Health Endpoint Monitoring

Tất cả các thành phần của hệ thống đều được tách ra, tương ứng thành các container khác nhau, đảm bảo mỗi container chỉ thực hiện một nhiệm vụ duy nhất; giúp hệ thống không bị lỗi hoàn toàn khi một vài dịch vụ gặp sự cố.

Để kiểm tra sức khỏe cho các container chứa API, cần phải kiểm tra cả khả năng phản hồi về API endpoint (như endpoint tới trang web lấy thông tin có gặp sự cố không), cũng như thông số về các tài nguyên mà chúng đã sử dụng. Về mặt kiểm tra endpoint có thể lấy được thông tin hay không, các hàm kiểm tra sẽ được cài đặt trực tiếp trong

các container đảm nhiệm API endpoint tương ứng, tuy nhiên để có thể kiểm tra chính xác, không thể chỉ dựa vào mã phản hồi mà API trả về (ví dụ như 200, 201, 400...). Đây chính là điểm mà mẫu thiết kế Health Endpoint Monitoring đã nhắc tới, cần phải kiểm tra toàn bộ nội dung của Response trả về, từ đó phát hiện ra lỗi, vị trí của lỗi, mức ảnh hưởng tới các thành phần khác.



Hình 1. Kiến trúc hệ thống

Việc kiểm tra thông số tài nguyên của các container gặp khó khăn khi chúng không thể phản hồi do sự cố. Do đó việc triển khai một hệ thống giám sát "sức khỏe" từ bên ngoài container là vô cùng cần thiết. Hệ thống này sử dụng các container chuyên biệt để thu thập dữ liệu từ những container dịch vụ như cAdvisor, Node Exporter và chính Docker Exporter được lấy từ Docker Engine. Các dịch vụ này cung cấp thông tin chi tiết về việc sử dụng tài nguyên hệ thống như CPU, bộ nhớ, mạng và I/O ổ cứng trong thời gian thực. Cụ thể, Node Exporter[2] cung cấp thông tin về tài nguyên hệ thống của máy chủ và các tiến trình liên quan, cAdvisor cung cấp thông tin về tài nguyên của các container, trong khi Docker Engine[3] cung cấp cái nhìn tổng quan về các container đang chạy trong hệ thống.

Các container này và Docker Exporter sẽ được kết nối với Prometheus[4], một container chuyên việc thu thập các số liệu (metrics) theo thời gian thực. Ở đây, Prometheus đóng vai trò như một nơi tổng hợp dữ liệu từ rất nhiều nguồn, và bằng PromQL (Prometheus Query Language), ta có thể lấy được dữ liệu và lọc theo những tham số nhất định đã cài đặt. Ngoài ra, Prometheus còn có chức năng là một hệ thống cảnh báo, có thể kích hoạt những cảnh báo về tình trạng của các container khi chỉ số của chúng vượt quá ngưỡng đã định theo những quy tắc mà người quản trị đã cài đặt.

Kết hợp với thành phần có tên là Alertmanager để có thể gửi các cảnh báo đã được kích hoạt trực tiếp tới quản trị viên thông qua email, webhook hoặc những hệ thống cảnh báo khác. Tất cả dữ liệu từ Prometheus sẽ làm đầu vào cho Grafana[5], một container chuyên biệt dùng để giám sát và trực quan hóa dữ liệu, cho phép tạo ra các bảng điều khiển (dashboards) trực quan và tổng thể để phân tích và theo dõi hệ thống.

Thêm vào đó, hệ thống sử dụng NGINX như một reverse proxy giúp tối ưu hóa việc định tuyến các yêu cầu đến các dịch vụ tương ứng thông qua URL định sẵn, đồng thời giảm thiểu số lượng cổng cần mở trên máy chủ và tăng cường tính tiện dụng cho các API. Tất cả các yêu cầu được gửi đến máy chủ đều được xử lý thông qua NGINX và gần như mọi yêu cầu này đều được ghi lại vào nhật ký truy cập, do đó chính NGINX cũng có thể cung cấp những số liệu liên quan về số lượng kết nối đang mở, tỷ lệ lỗi HTTP, lưu lượng truy cập,... NGINX Prometheus Exporter được sử dụng nhằm thu thập số liệu từ NGINX và chuyển đổi chúng sang định dạng tương thích với Prometheus. Ngoài ra, thông qua nhật ký truy cập của NGINX, các chỉ số quan trọng như thời gian phản hồi (response time) hay mã phản hồi HTTP (200, 400,...) có thể được trích xuất. Quá trình này được hỗ trợ bởi Fluentd, một container chuyên thu thập, xử lý, và chuyển tiếp dữ liệu của nhật ký đến các hệ thống giám sát. Các thông số như thời gian phản hồi sẽ giúp hệ thống giám sát và phân tích hiệu suất của các container liên quan, chính là một trong những tiêu chí mà mẫu thiết kế Health Endpoint Monitoring đã đề cập. Chẳng hạn, khi thời gian phản hồi tăng đột ngột, đây có thể là dấu hiệu của sự cố trong container hoặc hệ thống mạng.

Hệ thống còn sử dụng PostgreSQL, là hệ quản trị cơ sở dữ liệu quan hệ (RDBMS) hỗ trợ nhiều tính năng và tương thích với nhiều loại dữ liệu khác nhau, để lưu lại lịch sử giá vàng và tỷ giá so với VNĐ sau khi lấy dữ liệu từ các trang web về. Cuối cùng, nhằm giúp quá trình xem thông tin trên các dashboard khác nhau trở nên thuận tiện, cũng như xem các thông tin giá vàng và tỷ giá, hệ thống cũng tích hợp container đảm nhiệm vai trò Frontend sử dụng framework NextJS, có khả năng tối ưu hóa trải nghiệm người dùng, container này sẽ gọi tới các endpoint do NGINX đã định sẵn, lấy dữ liệu trả về dưới dạng JSON rồi hiển thị lên trình duyệt.

III. Triển khai, cài đặt hệ thống

1. Container chứa dịch vụ

Hệ thống cung cấp hai dịch vụ được triển khai dưới dạng các container trên nền tảng Docker: dịch vụ cung cấp tỷ giá hối đoái VND và dịch vụ cung cấp giá vàng ở Việt Nam. Dữ liệu tỷ giá hối đoái được thu thập từ nguồn trang apithanhtoan.com, với tỷ giá hối đoái của 6 ngân hàng bao gồm Vietcombank, Techcombank, BIDV, Agribank, VietInbank, TPBank. Dịch vụ cung cấp giá vàng cung cấp giá mua và bán của 3 công ty vàng lớn là SJC, DOJI, PNJ. Những dữ liệu này được lấy dữ liệu từ trang chủ của các

công ty trên và chuẩn hóa theo định dạng phù hợp với hệ thống. Sau đây là mô tả các endpoint:

- Container *gold-price* (cung cấp giá vàng ở Việt Nam):
  - */getprices:* endpoint này yêu cầu phương phức GET và phản hồi lại giá vàng mua và bán của tất cả các công ty trên hệ thống.
  - */getprice/[company_code]:* endpoint này yêu cầu phương thức GET và phản hồi lại giá vàng mua và bán của công ty có mã là *[company_code]*.
- Container *rate-exchange* (cung cấp tỷ giá hối đoái của VND):
  - */getexchangerates*: endpoint này yêu cầu phương thức GET và phản hồi lại tỷ giá hối đoái của tất cả các ngân hàng có trên hệ thống.
  - */getexchangerate/[bank_code]:* endpoint này yêu cầu phương thức GET và phản hồi lại tỷ giá hối đoái của ngân hàng có mã là *[bank_code]*.

2. NGINX: Reversed Proxy

Các đường dẫn được thiết lập ở NGINX là:

- *"/"*: dịch vụ web ở cổng 3000.
- *"/gold-price/":* dịch vụ cung cấp giá vàng ở cổng 8000.
- *"/rate-exchange/":* dịch vụ cung cấp tỷ giá hối đoái ở cổng 8000.
- *"/grafana/":* container Grafana ở cổng 3000.

Các thông số được lưu lại của mỗi yêu cầu đến máy chủ là

- *`$request_method`:* đây là tham số về phương thức của yêu cầu được gửi đến máy chủ (VD: GET, POST, PUT, ...).
- *`$request`:* địa chỉ của yêu cầu mà máy khách gửi đến máy chủ.
- *`$status`:* trạng thái HTTP của phản hồi từ máy chủ về máy khách (VD: 200, 201, 404, ...).
- *`$upstream_status`:* trạng thái HTTP của phản hồi từ dịch vụ về máy chủ NGINX.
- *`$upstream_cache_status`:* trạng thái bộ đệm của máy chủ NGINX cho yêu cầu đó từ máy khách.
- *`$upstream_response_time`:* thời gian phản hồi của dịch vụ đến NGINX tình từ thời điểm NGINX bắt đầu gửi yêu cầu đến dịch vụ và hoàn tất nhận phản hồi từ dịch vụ đó.

3. Container lấy thông số của hệ thống: Cadvisor, Node Exporter, Fluentd

Fluentd có khả năng phân tích nhật ký (log) từ đó cung cấp các thông số của các dịch vụ hoạt động bên trong các container. Do đó ta sẽ cần cấu hình sử dụng Regexp (regular expression) cho Fluentd để có thể chắt lọc ra các thông số cần thiết cho hệ thông từ nhật ký truy cập của NGINX. File cấu hình của Fluentd có thể được mount vào thông qua Docker.

Bên cạnh đó ta sẽ tạo một phân vùng lưu trữ chia sẻ, phân vùng lưu trữ này có thể được truy cập vào bởi cả NGINX và Fluentd, từ đó giúp Fluentd có thể đọc được nhật ký ghi bởi NGINX. CAdvisor và Node Exporter ta sử dụng image được build sẵn không cần tùy chỉnh thêm.

4. Container dùng để xử lý các metrics: Prometheus, Alertmanager, Grafana

Các container này cũng sẽ được mount sẵn file cấu hình. Thêm vào đó bảng điều khiển từ Grafana được cũng được xây dựng sẵn và được mount vào container Grafana thông qua docker.

Prometheus được cấu hình để thu thập dữ liệu từ những nguồn đã thiết lập mỗi 15 giây. Các nguồn dữ liệu sẽ được thiết lập sẵn là:

- *Docker*: ở *host.docker.internal:9323/metrics*
- *Cadvisor* ở *cadvisor:8080/metrics*
- *Node exporter* ở *node-exporter:9100/metrics*
- *Nginx prometheus exporter* ở *nginx-prometheus-exporter:9113/metrics*
- *Fluentd* ở *fluentd:24231/metrics*

Prometheus cũng được thiết lập để kiểm tra các cảnh báo dựa trên dữ liệu sẵn có mỗi 15 giây. Các cảnh bảo được thiết lập sẵn bao gồm:

- *ContainerKilled*: cảnh báo này sẽ được kích hoạt khi container đã bị tắt trong vòng 30 giây.
- *ContainerHighCpuUtilization*: cảnh báo này sẽ được kích hoạt khi phần trăm sử dụng CPU cao quá 80%.
- *ContainerHighMemoryUsage*: cảnh báo này sẽ được kích hoạt khi phần trăm sử dụng RAM cao quá 80%.
- *ContainerVolumeUsage*: cảnh báo này sẽ được kích hoạt khi phần trăm sử dụng phân vùng lưu trữ cao quá 80%.

- *ContainerHighThrottleRate*: cảnh báo này sẽ được kích hoạt khi tỷ lệ sử dụng CPU của bất kỳ container nào vượt quá 80% trong thời gian liên tục là 5 phút, cho biết container đang bị "nghẽn" do thiếu CPU.

Các thống số và cảnh báo sẽ được hiển thị trực quan trên bảng điều khiển từ Grafana dưới dạng các biểu đồ và bảng. Bên cạnh đó, các cảnh báo cũng sẽ được tự động gửi trực tiếp tới quản trị viên thông qua công cụ Alertmanager bằng nhiều phương thức khác nhau, trong đó phổ biến nhất là webhook và email. Hook và email này có thể được cung cấp bởi một máy chủ riêng biệt hoặc dịch vụ của một bên thứ ba. Trong phạm vi dự án này, ta sẽ tiến hành thử nghiệm gửi thông báo thông qua nền tảng Discord.

5. Container dịch vụ web: Frontend

Dịch vụ web được phát triển sử dụng framework Nextjs chạy trên nền tảng Nodejs. Với mỗi trang, các thành phần giao diện người dùng được tách riêng, cho phép sử dụng lại trong các trang khác nhau. Với trang cung cấp giá vàng, hai bảng cung cấp trạng thái và dữ liệu được đặt cạnh nhau, tạo một giao diện nhỏ gọn và tổng quát. Với trang cung cấp thông tin về tỷ giá ngoại tệ, các thành phần như dẫn đường (breadcrumbs) được kết hợp cùng các nút bấm để điều hướng dễ dàng, đồng thời tách riêng các trang cung cấp tỷ giá chi tiết của từng ngân hàng. Trong các trang cung cấp số liệu, các bảng điều khiển Grafana được nhúng trực tiếp vào trong trang web, điều này mang lại trải nghiệm người dùng liền mạch và dễ dàng.

IV. Kết quả kiểm tra

Ở mục này, kết quả cũng như cách các giá trị được trả về theo định dạng sẽ được phân loại, tương ứng với chức năng của các container có liên quan. Trong những trường hợp thực tế, có những lúc hệ thống sẽ không phản hồi hoặc bị sập, thì phần này cũng sẽ có những kết quả tương ứng khi đó hệ thống trả về.

1. Kết quả lấy dữ liệu từ các nguồn của container chứa API endpoint

Kết quả trả về của các container chứa API endpoint sẽ có dạng như sau:

1.1. Container chứa API giá vàng:

Kết quả lấy dữ liệu giá vàng từ website của các công ty sẽ được gói vào 1 object được tạo từ class có tên là Price, gồm:

- *Company_code*: Biểu thị cho tên công ty, dưới dạng viết tắt (VD: SJC)
- *Buy*: Giá vàng mua vào của công ty, đơn vị VNĐ
- *Sell*: Giá vàng bán ra của công ty, đơn vị VNĐ

Kết quả trả về khi hệ thống hoạt động bình thường:

```
{
    "company_code": "SJC",
    "buy": 85000000,
    "sell": 87000000
}
```


Hình 2. Kết quả API giá vàng khi trả về

Kết quả trả về khi hệ thống gặp sự cố (mất mạng, website của công ty không phản hồi ...):


```
{
    "detail": {
        "company_code": "SJC",
        "error": "Internal Server Error"
    }
}
```


Hình 3. Kết quả API giá vàng trả về khi gặp lỗi

Ngoài ra, bản thân container cũng đã được cài đặt hàm để có thể trả về tình trạng của các website endpoint để lấy giá vàng tương ứng. Hàm kiểm tra này sẽ gồm 3 biến:

- *provider_connection*: kết nối tới website
- *parse_data*: kiểm tra dữ liệu đúng dạng có thể xử lý chưa
- *data_response*: dữ liệu đã được trả về chưa.

Nếu hệ thống hoạt động bình thường, tất cả các biến sẽ có giá trị True, nếu bị lỗi kết nối thì cả ba biến đều có giá trị False, còn nếu chỉ bị lỗi về giá trị thì *parse_data* sẽ nhận giá trị False.


```
    {
        "company_code": "SJC",
        "provider_connection": true,
        "parse_data": true,
        "endpoint_response": true
    },
```


Hình 4. Kết quả kiểm tra các endpoint mà API giá vàng sử dụng

1.2. Container chứa API tỷ giá ngoại tệ so với VNĐ

Do thông tin giá vàng được lấy từ trang apithanhtoan.com nên các trường thông tin hầu hết sẽ được lấy ra từ nội dung gốc, và các trường quan trọng trong thông tin mà API trả về như sau:

- *success*: Nhận giá trị True/False để biểu thị trạng thái lấy dữ liệu thành công hay thất bại.
- *bank_name, bank_code, swift_code*: Các mã của ngân hàng tương ứng

- *rate_exchanges*: Danh sách chứa thông tin tỷ giá so với VND. Các phần tử con trong mảng là 1 object gồm những trường thông tin quan trọng sau:
    - *currency_name, currency_code*: Tên và kí hiệu viết tắt của loại tiền đó
    - *rate_buy, rate_transfer, rate_sell*: Chỉ số tỷ giá tương ứng với VND

Kết quả trả về khi hệ thống hoạt động bình thường:


```
{
    "success": true,
    "bank_name": "Ng%C3%A2n%20h%C3%A0ng%20TMCP%20Ngo%E1%BA%A1i%20Th%C6%B0%C6%A1ng%20Vi%E1%BB%87t%20Nam%20%28VIETCOMBANK%29",
    "bank_name_short": "Vietcombank",
    "bank_code": "BFTV",
    "swift_code": "BFTVVNVX",
    "rate_exchanges": [
        {
            "currency_name": "US%20DOLLAR%20%20%20%20%20%20%20%20%20%20%20%20",
            "currency_name_vi": "Dollar%20M%E1%BB%B9",
            "currency_code": "USD",
            "rate_buy": "25160.00",
            "rate_transfer": "25190.00",
            "rate_sell": "25512.00",
            "logo": "https://apithanhtoan.com/assets-iframe/images/flags/32/United-States.png"
        },
```


Hình 5. Kết quả trả về của API cập nhật tỷ giá VND khi hoạt động bình thường

Kết quả trả về khi hệ thống gặp lỗi (không kết nối được tới web ...):

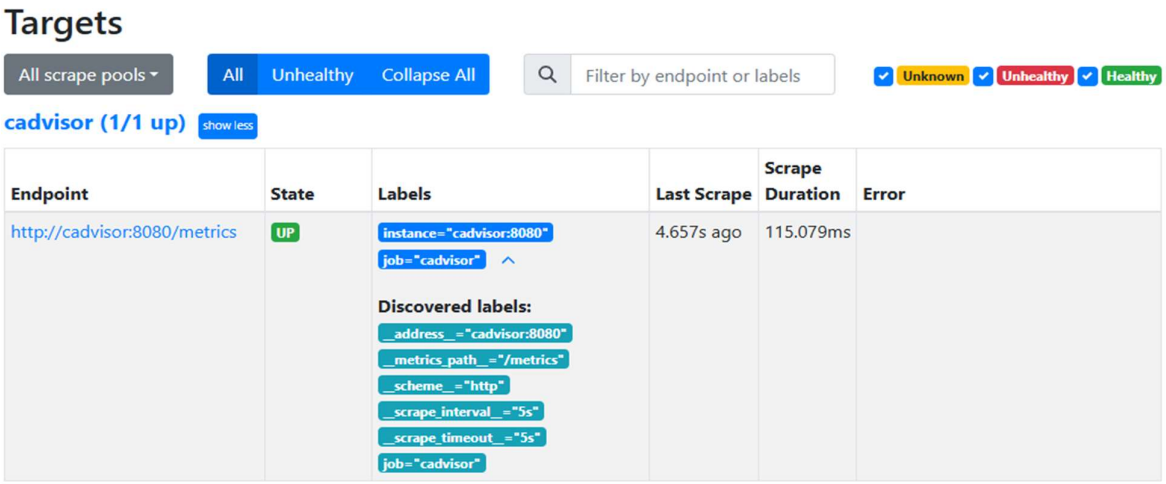

```
{
    "success": false,
    "bank_name_short": "bidv",
    "error": "HTTPSConnectionPool(host='apithanhtoan.com', port=443): Max retries exceeded
        with url: /ty-gia/bidv/ (Caused by NameResolutionError(\"<urllib3.connection.
        HTTPSConnection object at 0x7f2362bde840>: Failed to resolve 'apithanhtoan.com' (
        [Errno -3] Temporary failure in name resolution)\"))"
}
```


Hình 6. Kết quả trả về của API cập nhật tỷ giá VND khi gặp lỗi

Container này cũng đã cài đặt hàm để có thể trả về tình trạng của các website endpoint để lấy tỷ giá tương ứng, do chi tiết giống với hàm của container cập nhật giá vàng.

2. Hiển thị thông tin trạng thái từ các endpoint kết nối tới Prometheus:

Trong hệ thống, có nhiều container đóng vai trò lấy các thông số tổng quan như CPU, bộ nhớ, mạng và I/O ổ cứng của các container chứa API endpoint mà hệ thống muốn kiểm tra, tất cả chúng đều được kết nối tới Prometheus, và khi kết nối thành công, chúng sẽ được hiển thị như sau:



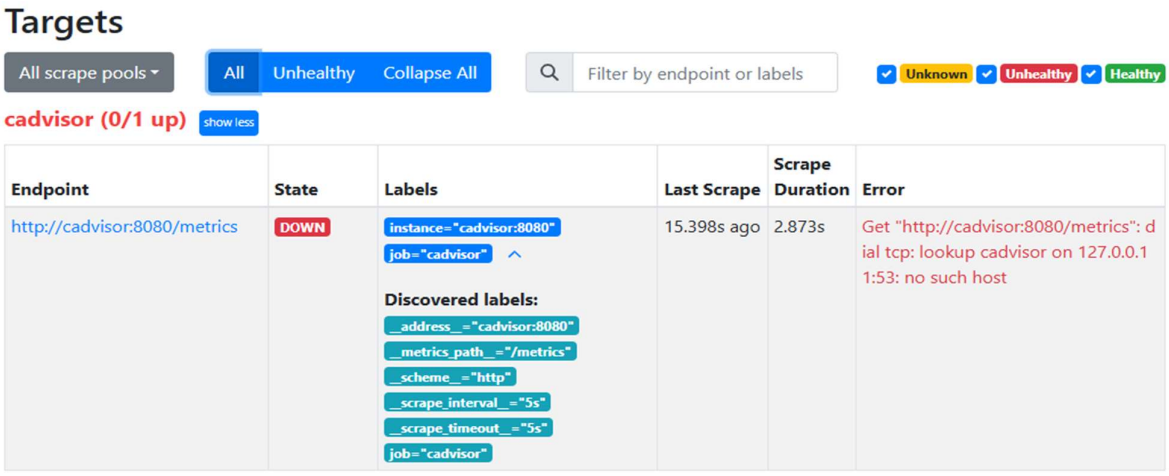
Hình 7. Container Cadvisor đã kết nối thành công tới Prometheus

Một số các thông số quan trọng liên quan:

- *instance:* Thể hiện của container dưới dạng *hostname:port*
- *state:* Chỉ trạng thái up/down của endpoint tương ứng
- *__metric_path__*: Địa chỉ dẫn tới nơi trả về kết quả mà instance lấy được
- *scrape:* Các thông số liên quan tới thời gian thu thập và trích xuất dữ liệu
- *error:* Lỗi xảy ra (nếu có)

Các container khác khi kết nối thành công tới Prometheus cũng sẽ hiển thị với định dạng tương tự như hình minh họa phía trên.

Tuy nhiên, trong trường hợp có container nào gặp lỗi, không phản hồi tới Prometheus, thì kết quả sẽ được hiển thị như sau:



Hình 8. Trạng thái container kết nối tới Prometheus khi không phản hồi

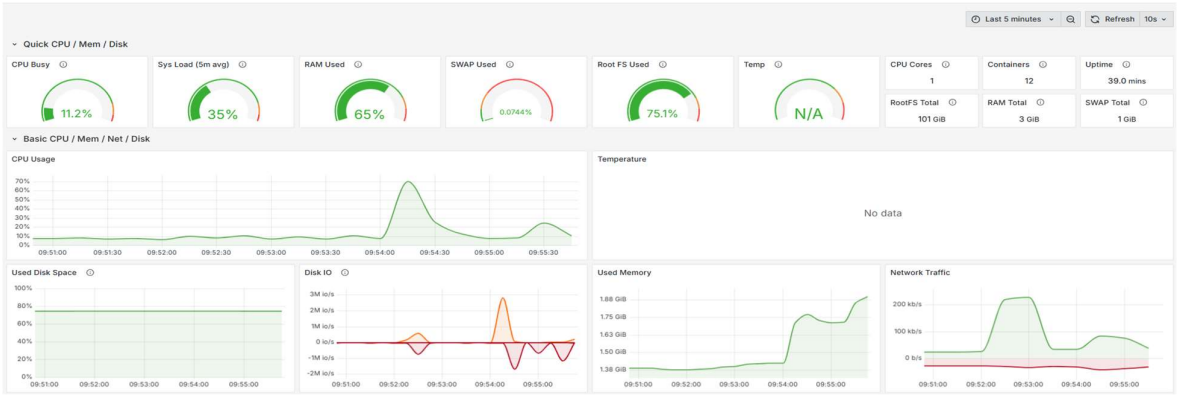
3. Hiển thị thông tin tài nguyên máy chủ và container trên dashboard

Do các endpoint khi kết nối tới Prometheus chỉ trả về định dạng như một file text với các thông số được cập nhật khi làm mới trang, chẳng hạn như hình dưới đây:

```
# HELP nginx_connections_active Active client connections
# TYPE nginx_connections_active gauge
nginx_connections_active 1
```

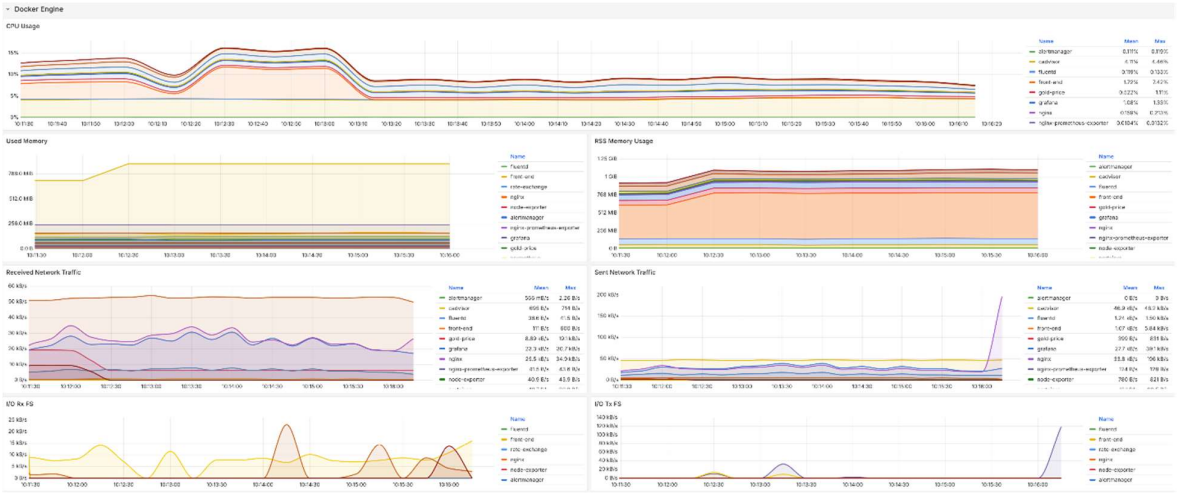
Hình 9. Dạng metric được export mà Prometheus đọc

Nên để có thể nắm được tình trạng hệ thống một cách toàn diện, những trang tổng quan (dashboard) cần được xây dựng, và đây là một dashboard tổng quan về các thông số cơ bản như CPU, bộ nhớ, mạng, ổ cứng:



Hình 10. Dashboard biểu thị các thông số cơ bản của toàn hệ thống

Dashboard cũng có thể biểu thị thông số phân theo các container đang chạy:



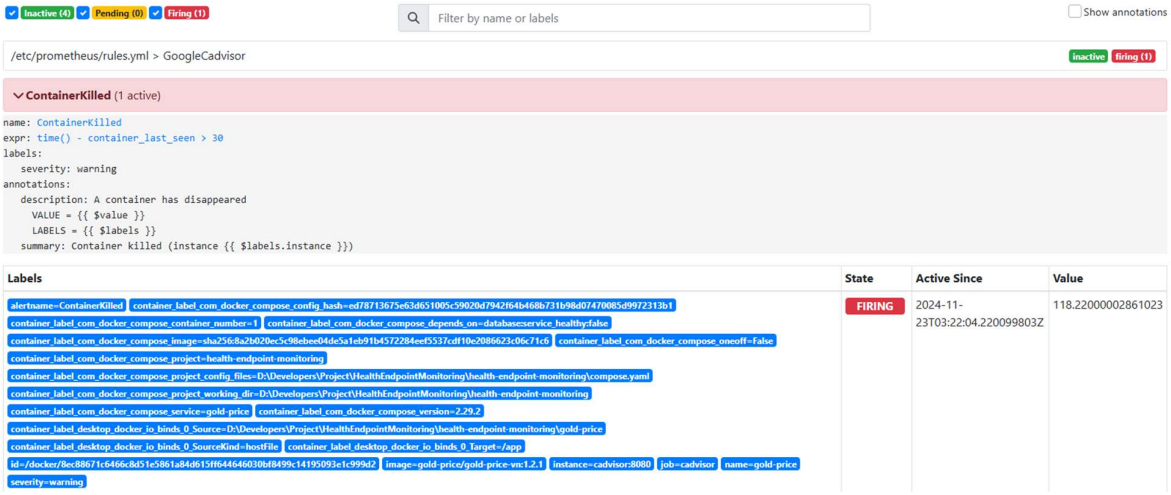
Hình 11. Trạng thái hệ thống theo các container tương ứng

4. Hiển thị trạng thái khi bình thường và khi có cảnh báo

Khi hệ thống hoạt động bình thường, không có cảnh báo nào được gửi đi, như trong hình kết quả dưới đây:

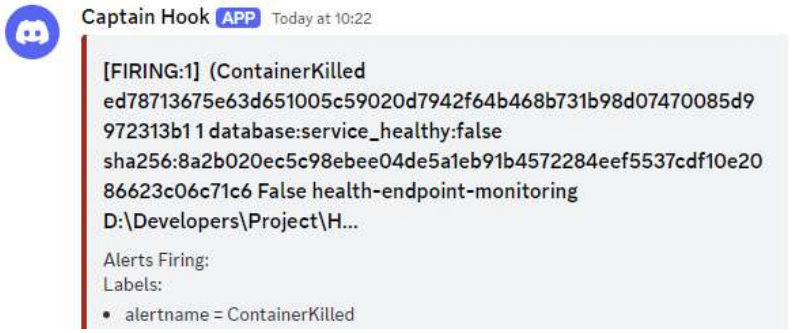
Hình 12. Trạng thái của các cảnh báo khi hệ thống hoạt động bình thường

Tuy nhiên, khi có thành phần gặp sự cố, những cảnh báo liên quan sẽ được kích hoạt, và các cảnh báo sẽ được gửi tới cả Prometheus:



Hình 13. Các cảnh báo được phát đi khi container gặp sự cố

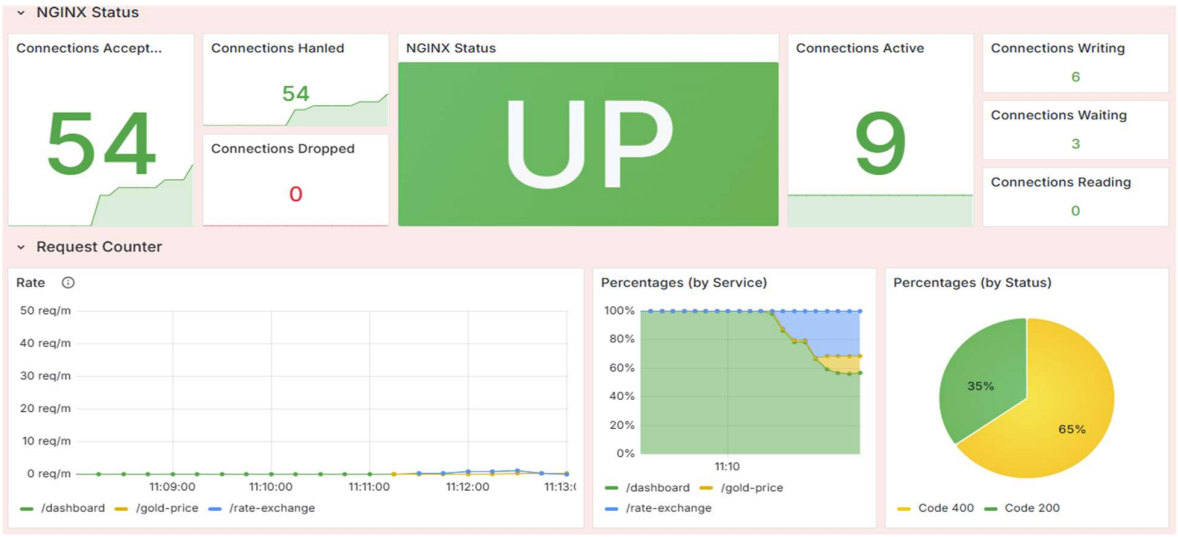
Cũng như sẽ được gửi tới webhook và từ đó thông báo trên Discord:



Hình 14. Cảnh báo nhận được trên Discord

5. Biểu đồ lưu lượng truy cập, tổng số kết nối, thời gian phản hồi

Do tất cả đều được truy cập qua NGINX, nên ta có thể lấy ra được những thông số như số lượng kết nối, kết nối tới endpoint nào, số lượng loại mã phản hồi (200, 404, ...) như trong hình vẽ sau:



Hình 15. Biểu đồ hiển thị trạng thái và thông số của NGINX

Ngoài ra, ta cũng có thể lấy được các thông số về thời gian phản hồi của các API khi truy cập qua NGINX:

Hình 16. Thời gian phản hồi và các URL truy cập tương ứng

V. Đánh giá và kết luận

1. Đánh giá hiệu quả hệ thống giám sát

Tính hiệu quả và độ chính xác của hệ thống giám sát

Hệ thống giám sát hoạt động hiệu quả, theo dõi sức khỏe của hai container API (giá vàng và tỷ giá ngoại tệ) một cách độc lập, không có điểm thất bại duy nhất (single point failure). Các API có khả năng tự kiểm tra sức khỏe và duy trì hoạt động ngay cả khi website nguồn gặp sự cố.

Khả năng phát hiện sự cố và cảnh báo

Hệ thống có khả năng phát hiện sớm các sự cố và gửi cảnh báo kịp thời khi có vấn đề như vượt ngưỡng CPU, RAM, hoặc container không phản hồi, giúp người quản trị nhanh chóng xử lý.

Giám sát các chỉ số cơ bản và hiệu suất hệ thống

Các chỉ số về CPU, RAM, số lượng kết nối và thời gian phản hồi được giám sát và hiển thị rõ ràng cho từng container, giúp người quản trị nắm bắt trạng thái hoạt động của hệ thống một cách nhanh chóng và chính xác.

2. Kết luận và các hướng phát triển

2.1. Tổng kết các kết quả đạt được

Hệ thống giám sát đã hoàn thành tốt nhiệm vụ theo dõi sức khỏe của các container API, giúp phát hiện và cảnh báo kịp thời khi có sự cố xảy ra. Các chỉ số về hiệu suất, kết nối, và thời gian phản hồi đều được giám sát liên tục và hiển thị rõ ràng. Nhờ vậy, hệ thống đã đảm bảo sự ổn định và liên tục cho các dịch vụ API.

2.2. Các hướng phát triển trong tương lai

Trong tương lai, hệ thống giám sát có thể mở rộng để theo dõi nhiều container hơn, đặc biệt là khi có sự gia tăng số lượng API và dịch vụ trong hệ thống. Hệ thống cũng có thể được cải thiện để tự động hóa việc phản hồi sự cố. Ví dụ, khi phát hiện sự cố về tài nguyên (CPU, RAM), hệ thống có thể tự động thực hiện các hành động như restart container hoặc chuyển hướng traffic tới các container dự phòng mà không cần sự can thiệp của quản trị viên. Để nâng cao khả năng phân tích và theo dõi, có thể thêm chức năng vẽ biểu đồ lịch sử giá vàng và tỷ giá ngoại tệ. Hiện tại, dữ liệu đã được ghi lại trong cơ sở dữ liệu, do đó, hệ thống có thể được mở rộng để hiển thị các biểu đồ và báo cáo lịch sử của giá vàng và tỷ giá theo thời gian, giúp người dùng dễ dàng theo dõi và phân tích các xu hướng. Ngoài các công cụ hiện có, việc tích hợp thêm các công cụ giám sát như ELK stack (Elasticsearch, Logstash) có thể giúp hệ thống giám sát trở nên mạnh mẽ hơn trong việc phân tích log và theo dõi sự kiện của toàn bộ hệ thống.

VI. Tham khảo

1. Mã nguồn

Mã nguồn github dự án: Mã nguồn Github

2. Thông tin tài liệu tham khảo

[1] Tài liệu mẫu thiếu kế Health Endpoint Monitoring: Tài liệu mẫu thiết kế của Azure

[2] Tài liệu công cụ Node Exporter: Đường dẫn đến tài liệu Node Exporter

[3] Tài liệu Docker Engine: Đường dẫn đến tài liệu Docker Engine

[4] Tài liệu công cụ Prometheus: Đường dẫn đến tài liệu Prometheus

[5] Tài liệu công cụ Grafana: Đường dẫn đến tài liệu Grafana

[6] Công cụ cAdvisor: Trang Github của cAdvisor

[7] NGINX Prometheus Exporter: Trang Github của NGINX Prometheus Exporter

[8] Công cụ fluentd: Tài liệu Fluentd