

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

**NGUYỄN VŨ ĐẠI DƯƠNG - 22520308
LÊ HỮU KHÁNH - 22520636
BÙI QUỐC MINH - 22520855
VŨ HUỲNH KIỀU NGÂN - 22520938**

**ĐỒ ÁN MÔN HỌC
HỆ TÍNH TOÁN PHÂN BỐ**

OpenFaaS (FaaS)

BÁO CÁO CUỐI KỲ

**GIẢNG VIÊN HƯỚNG DẪN
THS. BÙI THANH BÌNH**

TP. HỒ CHÍ MINH, NĂM 2025

Mục lục

Mục lục	i
1 Giới thiệu đề tài	1
1.1 Lý do chọn đề tài	1
1.2 Mục đích thực hiện đề tài	1
1.3 Đối tượng và phạm vi nghiên cứu của đề tài	2
1.3.1 Đối tượng nghiên cứu	2
1.3.2 Phạm vi kỹ thuật:	2
1.3.3 Giới hạn nghiên cứu:	2
2 Cơ sở lý thuyết	3
2.1 Giới thiệu về OpenFaaS	3
2.1.1 Kiến trúc của OpenFaaS	4
2.1.2 Các tính năng nổi bật của OpenFaaS	4
2.1.3 Quy trình triển khai một function trên OpenFaaS	5
2.1.4 Lý do lựa chọn OpenFaaS	5
2.1.5 Hạn chế	6
2.2 Kubernetes và K3s	6
2.2.1 Kubernetes	6
2.2.2 K3s – Phiên bản nhẹ của Kubernetes	7
2.2.3 So sánh Kubernetes và K3s	7
2.2.4 Vai trò của Kubernetes/K3s trong OpenFaaS	8
2.3 Công cụ hỗ trợ triển khai	8
2.3.1 arkade	8
2.3.2 faas-cli	8
2.3.3 Docker / Containerd	9
2.3.4 GitLab CI/CD và Webhook	9
2.3.5 Prometheus và Grafana	9

2.3.6	Các công cụ hỗ trợ khác	10
3	Thiết kế và xây dựng hệ thống	11
3.1	Kiến trúc tổng thể hệ thống	11
3.2	Chi tiết các luồng xử lý	12
3.2.1	Luồng xử lý 1: Quy đổi tiền tệ (currency function)	12
3.2.2	Luồng xử lý 2: Theo dõi giá chứng khoán (stock function)	12
3.2.3	Luồng xử lý 3: Nhận sự kiện từ GitLab → gửi Telegram (webhook function)	13
3.2.4	Luồng giám sát hệ thống (Prometheus + Grafana)	13
4	Triển khai hệ thống	14
4.1	Triển khai K3s Cluster	14
4.1.1	Triển khai Infrastructure và cài đặt các dependencies	14
4.1.2	Cài đặt K3s	15
4.1.3	Kiểm tra cài đặt	15
4.2	Cài đặt và triển khai OpenFaaS	16
4.2.1	Cài đặt OpenFaaS	16
4.2.2	Triển khai OpenFaaS	17
4.3	Triển khai chức năng và tạo website	19
4.3.1	Tạo chức năng quy đổi tiền tệ với ExchangeRate-API	19
4.3.2	Tạo chức năng theo dõi chứng khoán	21
4.3.3	Tạo website	25
4.3.4	Tạo Webhooks function	27
4.4	Cài đặt hệ thống giám sát cho OpenFaaS	32
4.4.1	Prometheus	32
4.4.2	Grafana	33
	Tài liệu tham khảo	36

Chương 1

Giới thiệu đề tài

1.1 Lý do chọn đề tài

Trong thời đại công nghệ thông tin bùng nổ, việc tối ưu hóa hạ tầng hệ thống và tối giản công tác quản trị tài nguyên đang trở thành yêu cầu cấp thiết đối với các doanh nghiệp và tổ chức. Mô hình **Function-as-a-Service (FaaS)** nổi lên như một giải pháp hiệu quả, cho phép các nhà phát triển tập trung hoàn toàn vào xây dựng và phát triển chức năng mà không cần quan tâm đến việc vận hành và quản lý máy chủ nền tảng.

OpenFaaS – một nền tảng mã nguồn mở triển khai mô hình FaaS trên Kubernetes và Docker – được đánh giá cao nhờ tính linh hoạt, khả năng mở rộng và sự hỗ trợ cộng đồng mạnh mẽ. Với mong muốn tiếp cận và tìm hiểu sâu hơn về các mô hình tính toán phân tán hiện đại, cũng như vận dụng kiến thức đã học vào thực tế, nhóm chúng em quyết định chọn đề tài "**OpenFaaS (FaaS)**" làm nội dung nghiên cứu cho đồ án môn học Hệ Tính Toán Phân Bố.

1.2 Mục đích thực hiện đề tài

Thông qua đề tài này, nhóm hướng đến các mục tiêu cụ thể sau:

- Tìm hiểu nguyên lý hoạt động và kiến trúc của mô hình Function-as-a-Service (FaaS).
- Triển khai thành công OpenFaaS trên hệ thống Kubernetes nhẹ (K3s Cluster) trong môi trường Google Cloud Platform.
- Xây dựng một số chức năng serverless thực tế như quy đổi tiền tệ, theo dõi chứng khoán.
- Thiết lập hệ thống giám sát hoạt động của OpenFaaS thông qua Prometheus và Grafana.

- Rèn luyện kỹ năng thiết kế, triển khai và vận hành hệ thống phân tán trên môi trường đám mây.

1.3 Đối tượng và phạm vi nghiên cứu của đề tài

1.3.1 Đối tượng nghiên cứu

- Nghiên cứu mô hình FaaS và kiến trúc serverless.
- Tìm hiểu và triển khai hệ thống OpenFaaS trên nền tảng Kubernetes (K3s).
- Triển khai các chức năng serverless mẫu trên OpenFaaS.

1.3.2 Phạm vi kỹ thuật:

- Sử dụng Google Cloud Platform để triển khai hạ tầng máy ảo cho cụm K3s.
- Sử dụng K3s để triển khai Kubernetes Cluster nhẹ.
- Cài đặt OpenFaaS và xây dựng các hàm serverless mẫu (currency-converter, stock-tracker).
- Thiết lập hệ thống giám sát với Prometheus và Grafana.
- Các vấn đề về tối ưu hóa hiệu suất, bảo mật hệ thống và triển khai quy mô lớn ngoài phạm vi nghiên cứu của đồ án.

1.3.3 Giới hạn nghiên cứu:

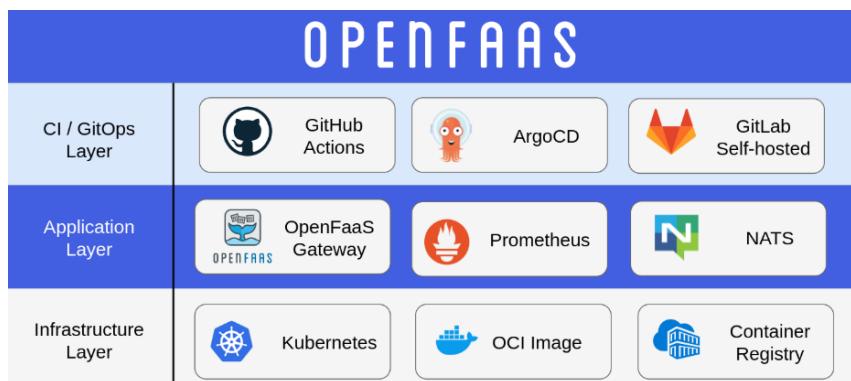
- Chỉ tập trung vào việc triển khai và vận hành OpenFaaS trên môi trường cụ thể (K3s Cluster GCP).
- Các chức năng serverless xây dựng ở mức độ cơ bản, mang tính minh họa cho cơ chế hoạt động.
- Không đi sâu vào tối ưu hóa cluster Kubernetes cho sản xuất hoặc các chiến lược CI/CD phức tạp.

Chương 2

Cơ sở lý thuyết

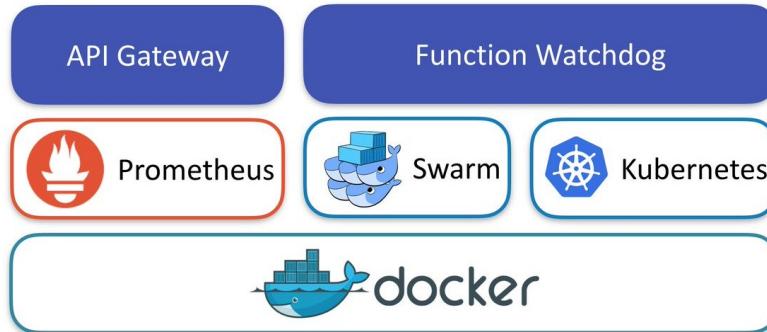
2.1 Giới thiệu về OpenFaaS

OpenFaaS (Open Function as a Service) là một nền tảng mã nguồn mở cho phép triển khai và quản lý các hàm serverless (FaaS) một cách dễ dàng trên nền tảng container như Docker và hệ sinh thái Kubernetes. Được phát triển bởi cộng đồng nguồn mở và sáng lập bởi Alex Ellis, OpenFaaS nhanh chóng trở thành một trong những dự án phổ biến nhất trong lĩnh vực serverless mã nguồn mở.



Hình 2.1: Sơ đồ phân lớp kiến trúc tổng quan của OpenFaaS

2.1.1 Kiến trúc của OpenFaaS



Hình 2.2: Kiến trúc của OpenFaaS

OpenFaaS được xây dựng theo mô hình microservices với các thành phần chính như sau:

- **Gateway:** Là trung tâm điều phối, cung cấp REST API cho việc quản lý function và nhận request từ người dùng. Gateway cũng tích hợp Prometheus để thu thập metric.
- **Function Watchdog:** Là một lightweight HTTP server chạy trong mỗi function container, có nhiệm vụ nhận HTTP request, chuyển đến function handler và trả kết quả.
- **faas-netes (hoặc faas-swarm):** Là một trình điều phối, chịu trách nhiệm triển khai các function trên nền tảng Kubernetes hoặc Docker Swarm.
- **CLI (Command Line Interface):** Cho phép người dùng tương tác với hệ thống OpenFaaS, bao gồm tạo, deploy, invoke, hoặc remove function.
- **UI Portal:** Giao diện người dùng web cho phép xem, quản lý và thực thi các function một cách trực quan.
- **Prometheus:** Công cụ giám sát được tích hợp sẵn, thu thập số liệu về số lần gọi và hiệu suất các function.
- **Docker / Container Runtime:** Mỗi function được đóng gói trong container và chạy trên môi trường như Docker Swarm hoặc Kubernetes.

2.1.2 Các tính năng nổi bật của OpenFaaS

- **Hỗ trợ đa nền tảng:** Chạy được trên Kubernetes, Docker Swarm, hoặc K3s – một phiên bản lightweight của Kubernetes.

- **Không giới hạn ngôn ngữ:** Có thể triển khai function viết bằng bất kỳ ngôn ngữ nào hỗ trợ HTTP hoặc stdio (Python, Node.js, Go, C#, Java,...).
- **Hệ sinh thái mở rộng:** Hỗ trợ các template có sẵn, Prometheus monitoring, auto-scaling, và tích hợp GitOps (qua OpenFaaS Cloud).
- **Khả năng mở rộng cao:** Có thể tự động mở rộng số lượng bản sao function tùy theo lưu lượng, nhờ tích hợp cơ chế auto-scaling dựa trên số lượng request hoặc metric từ Prometheus.
- **Hỗ trợ CI/CD và GitOps:** Có thể tích hợp với GitLab, GitHub để triển khai function tự động khi có thay đổi mã nguồn.
- **Dễ sử dụng:** Với CLI thân thiện, cùng khả năng deploy function chỉ với một dòng lệnh (faas-cli up), việc triển khai ứng dụng serverless trở nên rất đơn giản.

2.1.3 Quy trình triển khai một function trên OpenFaaS

1. **Tạo function:** Sử dụng faas-cli new để tạo một function dựa trên template ngôn ngữ mong muốn.
2. **Viết logic:** Chỉnh sửa file handler.py, handler.js... để định nghĩa logic của function.
3. **Xây dựng function:** Dùng lệnh faas-cli build để build image Docker từ function.
4. **Deploy function:** Sử dụng faas-cli deploy hoặc faas-cli up để triển khai function lên cụm Kubernetes hoặc Docker Swarm.
5. **Invoke function:** Gửi request qua REST API, CLI hoặc giao diện UI để thực thi function.

2.1.4 Lý do lựa chọn OpenFaaS

OpenFaaS là lựa chọn lý tưởng cho các tổ chức, cá nhân muốn triển khai mô hình serverless mà không phụ thuộc vào nền tảng đám mây thương mại như AWS Lambda, Google Cloud Functions. Các ưu điểm khiến OpenFaaS trở thành nền tảng nổi bật bao gồm:

- **Mã nguồn mở và minh bạch,** dễ dàng tùy biến.
- **Triển khai linh hoạt** trên nhiều môi trường khác nhau, từ cloud đến on-premise.
- **Phù hợp với Kubernetes,** dễ tích hợp vào hệ sinh thái cloud-native hiện đại.
- **Cộng đồng hỗ trợ mạnh mẽ,** tài liệu đầy đủ và liên tục cập nhật.

2.1.5 Hạn chế

Dù có nhiều lợi thế, OpenFaaS vẫn tồn tại một số hạn chế so với các nền tảng thương mại:

- Cần kiến thức về Kubernetes hoặc Docker để triển khai.
- Thiếu một số tính năng quản lý nâng cao so với AWS Lambda (ví dụ: kiểm soát IAM, theo dõi chuyên sâu).
- Phải tự cấu hình hệ thống auto-scaling, bảo mật, giám sát nếu không dùng OpenFaaS Cloud.

2.2 Kubernetes và K3s

2.2.1 Kubernetes

Kubernetes là một nền tảng mã nguồn mở được phát triển bởi Google để tự động hóa việc triển khai, mở rộng và quản lý container. Được xem là tiêu chuẩn thực tế (de facto standard) trong quản lý container hiện nay, Kubernetes cung cấp cơ chế điều phối (orchestration) linh hoạt, đảm bảo tính sẵn sàng cao và khả năng mở rộng cho các ứng dụng cloud-native.

Các thành phần chính của Kubernetes bao gồm:

Master Node: Chịu trách nhiệm điều phối toàn bộ cluster. Bao gồm các thành phần như:

- *kube-apiserver*: Giao diện chính giữa client và hệ thống.
- *etcd*: Lưu trữ cấu hình và trạng thái cluster.
- *kube-scheduler*: Gán pod cho node thích hợp.
- *kube-controller-manager*: Quản lý các controller như replica, endpoint, node, v.v.

Worker Node: Là nơi thực thi container, chứa:

- *kubelet*: Giao tiếp với master để nhận lệnh.
- *kube-proxy*: Quản lý kết nối mạng cho pod.
- *container runtime (thường là Docker hoặc containerd)*: Thực thi container thực tế.

Kubernetes hỗ trợ các tính năng quan trọng như self-healing, load balancing, horizontal scaling, rolling updates, và secret/config management, giúp ứng dụng hoạt động ổn định và bảo mật trong môi trường phân tán [1].

2.2.2 K3s – Phiên bản nhẹ của Kubernetes

K3s là một phân phối nhẹ của Kubernetes được phát triển bởi Rancher Labs, thiết kế đặc biệt cho các môi trường tài nguyên hạn chế, IoT, edge computing hoặc phát triển thử nghiệm. K3s giúp đơn giản hóa quá trình cài đặt và vận hành Kubernetes mà vẫn giữ được đầy đủ chức năng cốt lõi.

Ưu điểm của K3s

- Gọn nhẹ:** Kích thước nhị phân <100 MB.
- Cài đặt đơn giản: Một dòng lệnh để cài đặt cả master và worker node.
- Tích hợp sẵn các thành phần cần thiết:** Như containerd, Flannel (network), CoreDNS, Traefik, v.v.
- Hỗ trợ embedded database:** Dùng sqlite, etcd hoặc kine thay vì etcd độc lập, phù hợp với môi trường nhỏ.
- Hỗ trợ ARM:** Chạy tốt trên Raspberry Pi hoặc thiết bị nhúng.

Nhờ tính đơn giản và hiệu suất cao, K3s đặc biệt phù hợp để triển khai OpenFaaS cho các dự án nhỏ, bài lab, hoặc hệ thống phân tán quy mô vừa và nhỏ trong môi trường đám mây như Google Cloud Platform hoặc AWS Lightsail [2].

2.2.3 So sánh Kubernetes và K3s

Tiêu chí	Kubernetes	K3s
Mục tiêu sử dụng	Sản xuất quy mô lớn	Edge, dev, test, IoT
Dung lượng nhị phân	300 MB	<100 MB
Cài đặt và vận hành	Phức tạp hơn	Đơn giản, 1 dòng lệnh
Quản lý etcd	Tự triển khai	Tích hợp sqlite/kine
Yêu cầu tài nguyên	Cao hơn	Thấp hơn nhiều
Phù hợp với OpenFaaS	Tốt	Rất phù hợp (gọn, dễ setup)

Bảng 2.1: So sánh Kubernetes và K3s

2.2.4 Vai trò của Kubernetes/K3s trong OpenFaaS

OpenFaaS sử dụng Kubernetes hoặc K3s như nền tảng điều phối để triển khai, mở rộng và quản lý các function dưới dạng container. Khi người dùng deploy function, Gateway sẽ gửi lệnh đến faas-netes (một controller của OpenFaaS), và nó sẽ tạo pod mới trên Kubernetes/K3s để thực thi function.

K3s đặc biệt phù hợp với OpenFaaS nhờ khả năng triển khai nhanh chóng, dung lượng nhỏ, tiêu tốn ít RAM/CPU, đồng thời vẫn hỗ trợ đầy đủ các công cụ giám sát như Prometheus, Grafana, và cơ chế scale function hiệu quả.

2.3 Công cụ hỗ trợ triển khai

Trong quá trình triển khai OpenFaaS trên nền tảng Kubernetes nhẹ như K3s, nhiều công cụ hỗ trợ đã được sử dụng để tự động hóa, đơn giản hóa thao tác, và đảm bảo tính nhất quán trong suốt vòng đời phát triển – triển khai – giám sát hệ thống. Dưới đây là các công cụ chính được sử dụng:

2.3.1 arkade

Arkade là một công cụ dòng lệnh mã nguồn mở giúp cài đặt các ứng dụng Kubernetes phổ biến một cách nhanh chóng và dễ dàng bằng các lệnh đơn giản. Arkade đóng vai trò quan trọng trong việc rút ngắn thời gian thiết lập hệ thống, đặc biệt hữu ích cho các môi trường phát triển, thử nghiệm, hoặc học tập.

Ví dụ, việc cài đặt OpenFaaS trên K3s chỉ cần một lệnh đơn giản:

```
arkade install openfaas
```

Arkade sẽ tự động tải và triển khai tất cả các thành phần cần thiết như Gateway, Prometheus, và các CRD kèm theo [3].

2.3.2 faas-cli

faas-cli là công cụ dòng lệnh chính thức để quản lý các function trong hệ thống OpenFaaS. Nó hỗ trợ đầy đủ các chức năng như:

- Tạo function từ template (*faas-cli new*).
- Build và đóng gói container image (*faas-cli build*).
- Deploy function (*faas-cli deploy*).

- Gửi request tới function (*faas-cli invoke*).
- Gỡ bỏ function (*faas-cli remove*).

Ngoài ra, faas-cli còn hỗ trợ việc tạo template tùy chỉnh, tương thích với các ngôn ngữ lập trình phổ biến như Python, Node.js, Go, C#, Java...[4].

2.3.3 Docker / Containerd

Các function của OpenFaaS được đóng gói dưới dạng container. Do đó, Docker hoặc containerd được sử dụng để:

- Build image từ mã nguồn function.
- Push lên container registry.
- Chạy container tương ứng trong môi trường Kubernetes.

Docker còn hỗ trợ các thao tác kiểm thử local trước khi deploy function thực tế.

2.3.4 GitLab CI/CD và Webhook

Trong hệ thống triển khai mẫu, GitLab self-hosted được sử dụng để:

- Quản lý mã nguồn function.
- Tích hợp webhook đến OpenFaaS để tự động gửi thông báo (qua Telegram).
- Hỗ trợ triển khai thủ công hoặc semi-automated dựa trên các sự kiện push hoặc merge request.

GitLab CI/CD kết hợp với Webhook và các hàm serverless là một hướng tiếp cận phổ biến trong mô hình GitOps – triển khai hệ thống theo sự kiện từ Git [5].

2.3.5 Prometheus và Grafana

Hai công cụ giám sát phổ biến:

- **Prometheus:** Thu thập các metric từ OpenFaaS Gateway, autoscaler, và function runtime để phục vụ giám sát hiệu suất.
- **Grafana:** Trực quan hóa các dữ liệu từ Prometheus dưới dạng biểu đồ, dashboard, hỗ trợ phát hiện lỗi hoặc cảnh báo theo thời gian thực.

Trong hệ thống, Prometheus được cấu hình tự động thông qua OpenFaaS Helm chart, trong khi Grafana được cài đặt thủ công để tạo các dashboard theo nhu cầu thực tế.

2.3.6 Các công cụ hỗ trợ khác

Ngoài ra, nhóm còn sử dụng một số công cụ hỗ trợ khác như:

- **curl:** Gửi request HTTP để kiểm tra function.
- **kubectl:** Tương tác với K3s/Kubernetes cluster để kiểm tra pod, service, log.
- **Vercel:** Deploy giao diện web frontend để tương tác với các function từ người dùng cuối.

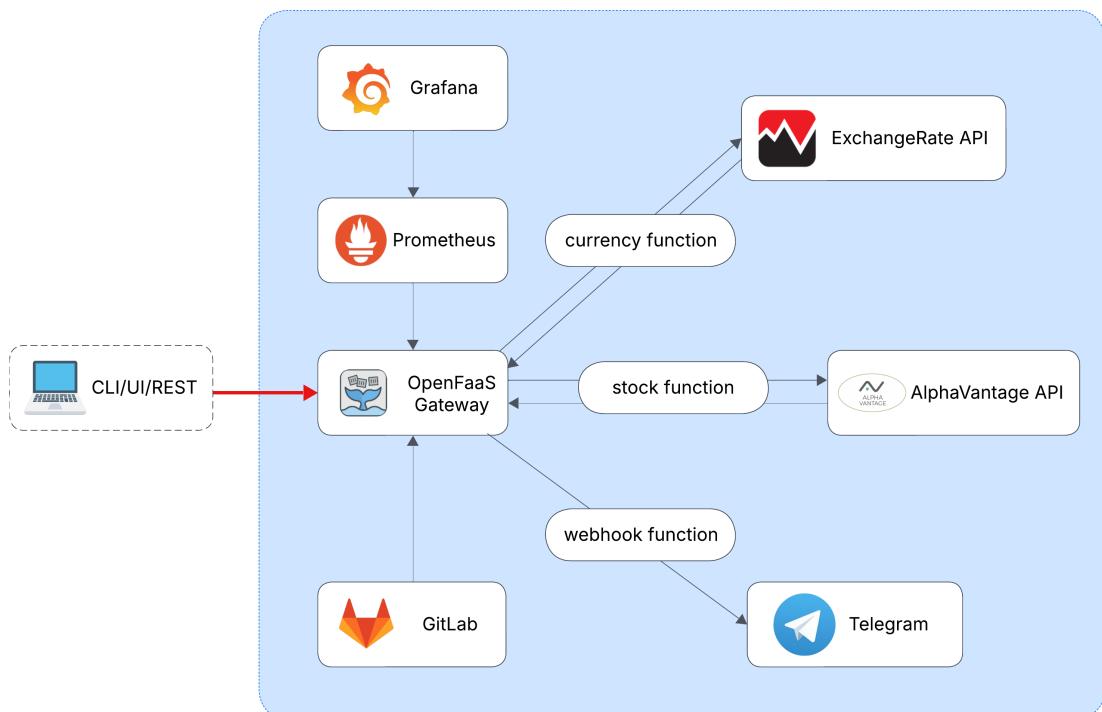
Chương 3

Thiết kế và xây dựng hệ thống

3.1 Kiến trúc tổng thể hệ thống

Hệ thống serverless sử dụng OpenFaaS trên cụm K3s Cluster (1 master, 2 worker), cung cấp 3 chức năng: quy đổi tiền tệ, theo dõi chứng khoán, và nhận webhook từ GitLab để gửi thông báo qua Telegram.

Các thành phần được triển khai và tương tác trong một kiến trúc tổng thể như hình sau:



Hình 3.1: Kiến trúc tổng quan hệ thống OpenFaaS trên Kubernetes Cluster

- **OpenFaaS Gateway:** Trung tâm tiếp nhận request, quản lý và phân phối các function.
- **Các function:** Được triển khai dưới dạng container độc lập trên Kubernetes.
- **Prometheus + Grafana:** Hệ thống giám sát thu thập và hiển thị metric.
- **Người dùng:** Giao tiếp với hệ thống thông qua trình duyệt web hoặc API frontend.
- **Dịch vụ bên ngoài:** Gồm AlphaVantage (cung cấp dữ liệu chứng khoán), ExchangeRate-API (cung cấp tỉ giá), GitLab (nguồn sự kiện webhook), Telegram (nhận cảnh báo).

3.2 Chi tiết các luồng xử lý

Hệ thống có ba chức năng chính với các luồng xử lý sau:

3.2.1 Luồng xử lý 1: Quy đổi tiền tệ (currency function)

1. Người dùng truy cập giao diện web, nhập loại tiền cần chuyển đổi.
2. Trình duyệt gửi yêu cầu HTTP đến API frontend (proxy).
3. API gửi request đến OpenFaaS Gateway.
4. Gateway gọi currency function.
5. Function gọi API từ nhà cung cấp tỉ giá (ExchangeRate-API).
6. Nhận kết quả và trả về cho người dùng.

- **Dữ liệu đầu vào:** Mã tiền tệ nguồn và đích.
- **Dữ liệu đầu ra:** Tỉ giá quy đổi mới nhất.
- **API bên ngoài:** ExchangeRate-API.

3.2.2 Luồng xử lý 2: Theo dõi giá chứng khoán (stock function)

1. Người dùng truy cập web, nhập mã cổ phiếu cần tra cứu.
2. Trình duyệt gửi request đến API frontend.
3. API gửi request đến OpenFaaS Gateway.
4. Gateway kích hoạt stock function.

5. Function thực hiện truy vấn API của AlphaVantage để lấy dữ liệu thị trường.

6. Kết quả được trả về giao diện người dùng.

- **Dữ liệu đầu vào:** Mã chứng khoán.
- **Dữ liệu đầu ra:** Giá và biến động cổ phiếu hiện tại.
- **API bên ngoài:** AlphaVantage.

3.2.3 Luồng xử lý 3: Nhận sự kiện từ GitLab → gửi Telegram (webhook function)

1. Khi có sự kiện xảy ra trong GitLab (push, merge request, issue...), GitLab gọi webhook đến OpenFaaS Gateway.

2. Gateway kích hoạt webhook function.

3. Function phân tích payload nhận được từ GitLab.

4. Dựa trên loại sự kiện, soạn tin nhắn phù hợp.

5. Gửi thông báo qua Telegram thông qua Bot API.

- **Dữ liệu đầu vào:** Payload từ GitLab Webhook.
- **Dữ liệu đầu ra:** Tin nhắn gửi qua Telegram.
- **API bên ngoài:** Telegram Bot API.

3.2.4 Luồng giám sát hệ thống (Prometheus + Grafana)

1. Prometheus liên tục thu thập metric từ OpenFaaS Gateway và các function.

2. Grafana đọc dữ liệu từ Prometheus và hiển thị dưới dạng dashboard.

3. Người dùng có thể giám sát thông lượng, thời gian phản hồi, số lần invoke, v.v.

- **Dữ liệu thu thập:** CPU, RAM, tỉ lệ lỗi, số lượng request.
- **Giao diện giám sát:** Grafana UI.

Chương 4

Triển khai hệ thống

4.1 Triển khai K3s Cluster

4.1.1 Triển khai Infrastructure và cài đặt các dependencies

Ở đồ án này, nhóm sẽ thực hiện triển khai Cluster trên Google Cloud Platform. Cluster gồm 3 máy ảo: 1 máy master và 2 máy worker.

VM instances								Create instance	Import VM	Refresh							
Instances		Observability		Instance schedules													
VM instances																	
Filter Enter property name or value																	
Status	Name ↑	Zone	Recommendations	In use by	Internal IP	External IP	Connect	SSH	⋮	⋮	⋮						
<input type="checkbox"/>	<input checked="" type="checkbox"/> openfaas	asia-southeast1-c			10.148.0.2 (nic0)	34.142.235.231 (nic0)	SSH	⋮	⋮	⋮	⋮						
<input type="checkbox"/>	<input checked="" type="checkbox"/> worker1	asia-southeast1-b			10.148.0.3 (nic0)	34.143.131.146 (nic0)	SSH	⋮	⋮	⋮	⋮						
<input type="checkbox"/>	<input checked="" type="checkbox"/> worker2	asia-southeast1-b			10.148.0.4 (nic0)	35.197.159.236 (nic0)	SSH	⋮	⋮	⋮	⋮						

Hình 4.1: Các máy ảo trong infrastructure

Cấu hình firewall allow các port: 6443 (Kubernetes API), 10250-10252 (kubelet), 30000-32767 (NodePort) và cài đặt Docker trên tất cả các máy ảo.

```

g22520636@openfaas:~$ docker version
Client: Docker Engine - Community
  Version:           28.1.1
  API version:      1.49
  Go version:       go1.23.8
  Git commit:       4eba377
  Built:            Fri Apr 18 09:52:57 2025
  OS/Arch:          linux/amd64
  Context:          default

Server: Docker Engine - Community
  Engine:
    Version:          28.1.1
    API version:     1.49 (minimum version 1.24)
    Go version:      go1.23.8
    Git commit:      01f442b
    Built:           Fri Apr 18 09:52:57 2025
    OS/Arch:         linux/amd64
    Experimental:   false
  containerd:
    Version:          1.7.27
    GitCommit:        05044ec0a9a75232cad458027ca83437aae3f4da
  runc:
    Version:          1.2.5
    GitCommit:        v1.2.5-0-g59923ef
  docker-init:
    Version:          0.19.0
    GitCommit:        de40ad0

```

Hình 4.2: Docker Version

4.1.2 Cài đặt K3s

Sử dụng dòng lệnh sau cho máy Master Node:

```
curl -sfL https://get.k3s.io | sh -s - --token nhom8
```

Sử dụng dòng lệnh sau cho máy Worker Node:

```
curl -sfL https://get.k3s.io | K3S_URL=https://MASTER_NODE_IP
K3S_TOKEN=nhom8 sh -
```

```

g22520636@worker1:~$ curl -sfL https://get.k3s.io | K3S_URL=https://10.148.0.2:6443 K3S_TOKEN=nhom8 sh -
[INFO] Finding release for channel stable
[INFO] Using v1.32.3+k3s1 as release
[INFO] Downloading hash https://github.com/k3s-io/k3s/releases/download/v1.32.3+k3s1/sha256sum-amd64.txt
[INFO] Downloading binary https://github.com/k3s-io/k3s/releases/download/v1.32.3+k3s1/k3s
[INFO] Verifying binary download
[INFO] Installing k3s to /usr/local/bin/k3s
[INFO] Skipping installation of SELinux RPM
[INFO] Creating /usr/local/bin/kubectl symlink to k3s
[INFO] Creating /usr/local/bin/crictl symlink to k3s
[INFO] Skipping /usr/local/bin/ctr symlink to k3s, command exists in PATH at /usr/bin/ctr
[INFO] Creating killall script /usr/local/bin/k3s-killall.sh
[INFO] Creating uninstall script /usr/local/bin/k3s-agent-uninstall.sh
[INFO] env: Creating environment file /etc/systemd/system/k3s-agent.service.env
[INFO] systemd: Creating service file /etc/systemd/system/k3s-agent.service
[INFO] systemd: Enabling k3s-agent unit
Created symlink /etc/systemd/system/multi-user.target.wants/k3s-agent.service → /etc/systemd/system/k3s-agent.service.
[INFO] systemd: Starting k3s-agent

```

Hình 4.3: Cài đặt K3s trên Worker Node

4.1.3 Kiểm tra cài đặt

Sau khi cài đặt xong, dùng lệnh sau để kiểm tra. Thấy rằng đã có 2 máy worker đang hoạt động:

```
sudo kubectl get pod -A -o wide
```

```
g22520636@openfaas:~$ sudo kubectl get pod -A -o wide
NAMESPACE     NAME                           READY   STATUS    RESTARTS   AGE     IP          NODE      NOMINATED NODE   READINESS GATES
kube-system   coredns-ff8999c5-cxqgo        1/1     Running   0          10m    10.42.0.4   openfaas   <none>        <none>
kube-system   helm-install-traefik-hg9c4   0/1     Completed  1       10m    10.42.0.3   openfaas   <none>        <none>
kube-system   helm-install-traefik-crd-jrr25  0/1     Completed  1       10m    10.42.0.2   openfaas   <none>        <none>
kube-system   local-path-provisioner-774c665dc-lt5s8 1/1     Running   0          10m    10.42.0.6   openfaas   <none>        <none>
kube-system   metrics-server-6f4c6e75d5-9cnt9  1/1     Running   0          10m    10.42.0.5   openfaas   <none>        <none>
kube-system   svc1b-traefik-63282256-gwbj4   2/2     Running   0          10m    10.42.0.7   openfaas   <none>        <none>
kube-system   svc1b-traefik-63282256-hb6xw   2/2     Running   0          100s   10.42.1.2   worker1   <none>        <none>
kube-system   svc1b-traefik-63282256-ngrj9   2/2     Running   0          95s    10.42.2.2   worker2   <none>        <none>
kube-system   traefik-67bfb46dc9-s9jk9     1/1     Running   0          10m    10.42.0.8   openfaas   <none>        <none>
g22520636@openfaas:~$
```

Hình 4.4: Kiểm tra Worker

4.2 Cài đặt và triển khai OpenFaaS

4.2.1 Cài đặt OpenFaaS

Đầu tiên, cài đặt Arkade bằng lệnh:

```
curl -sLS https://get.arkade.dev | sudo sh
```

```
g22520636@openfaas:~$ arkade
[Logo]
Open Source Marketplace For Developer Tools

Usage:
  arkade [flags]
  arkade [command]

Available Commands:
  chart      Chart utilities
  completion Output shell completion for the given shell (bash or zsh)
  get        The get command downloads a tool
  help       Help about any command
  info       Find info about a Kubernetes app
  install    Install Kubernetes apps from helm charts or YAML files
  oci        oci apps
  system    System apps
  uninstall  Uninstall apps installed with arkade
  update    Replace the running binary with an updated version
  version   Print the version

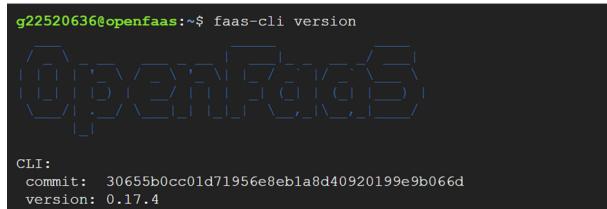
Flags:
  -h, --help   help for arkade

Use "arkade [command] --help" for more information about a command.
```

Hình 4.5: Kiểm tra Arkade

Tiếp tục, cài đặt OpenFaaS CLI bằng lệnh:

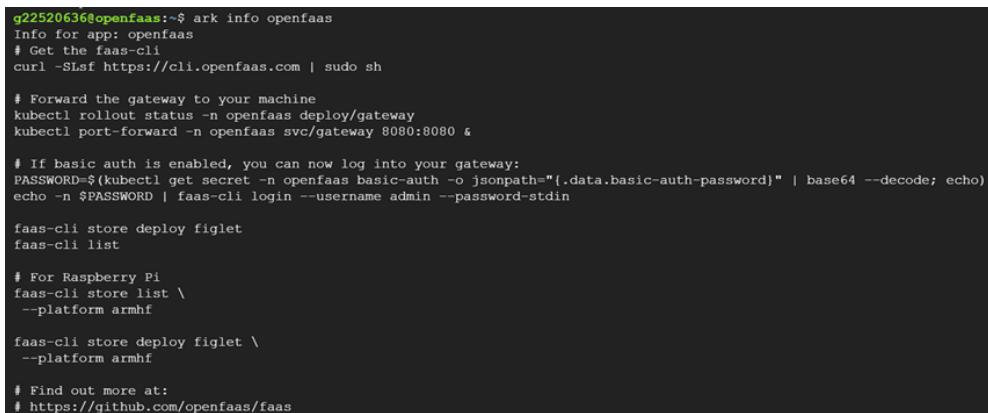
```
arkade get faas-cli
```



Hình 4.6: Kiểm tra OpenFaaS CLI

Sau đó, cài đặt OpenFaaS bằng lệnh:

```
arakde install openfaas
```



Hình 4.7: Kiểm tra thông tin OpenFaaS

4.2.2 Triển khai OpenFaaS

Tạo một service bằng file *openfaas-ui-lb-service.yaml* như sau để expose gateway ra ngoài K3s Cluster:

```
GNU nano 7.2                                     openfaas-ui-lb-service.yaml *
```

```
apiVersion: v1
kind: Service
metadata:
  name: openfaas-gateway
  namespace: openfaas
spec:
  type: NodePort
  ports:
  - name: http
    port: 80
    targetPort: 8080
    nodePort: 30080
  selector:
    app: gateway
```

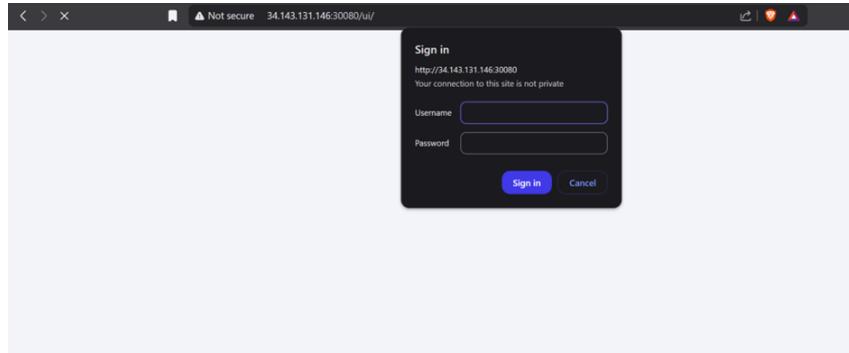
Hình 4.8: File openfaas-ui-lb-service.yaml

Thực hiện lệnh `kubectl apply -f openfaas-ui-lb-service.yaml` để tiến hành tạo service. Kiểm tra service, thấy rằng nó đã hoạt động:

```
g22520636@openfaas:~$ sudo kubectl get svc -n openfaas
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)        AGE
alertmanager   ClusterIP 10.43.27.47  <none>        9093/TCP     3d16h
gateway        ClusterIP 10.43.158.224 <none>        8080/TCP     3d16h
gateway-external NodePort 10.43.31.149  <none>        8080:31112/TCP 3d16h
nats           ClusterIP 10.43.10.112 <none>        4222/TCP     3d16h
openfaas-gateway NodePort 10.43.161.134 <none>        80:30080/TCP  3d15h
prometheus    ClusterIP 10.43.84.51  <none>        9090/TCP     3d16h
g22520636@openfaas:~$
```

Hình 4.9: Kiểm tra Service

Truy cập vào OpenFaaS Portal, thấy rằng nó yêu cầu đăng nhập bằng tài khoản và mật khẩu:



Hình 4.10: Màn hình đăng nhập OpenFaaS Portal

Thực hiện lấy mật khẩu đăng nhập bằng lệnh:

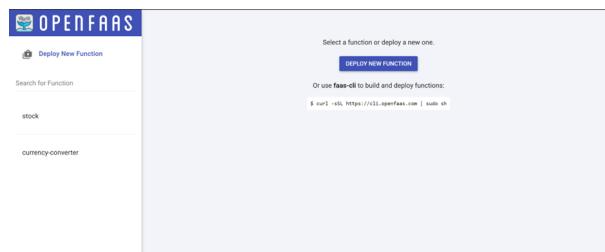
```
PASSWORD=$(kubectl get secret -n openfaas basic-auth -o
  jsonpath=".data.basic-auth-password" | base64 --decode;
echo)
```

Lệnh này sẽ giúp lưu mật khẩu vào biến PASSWORD:

```
g22520636@openfaas:~$ echo $PASSWORD
6CJAuQLFhn2o
```

Hình 4.11: Lấy mật khẩu cho OpenFaaS Portal

Lúc này, đăng nhập đã thành công với username là admin và password được lấy ở phía trên:



Hình 4.12: Giao diện chính của OpenFaaS Portal

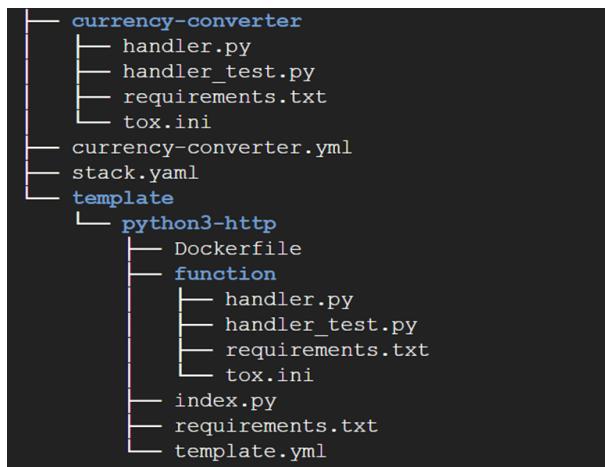
4.3 Triển khai chức năng và tạo website

4.3.1 Tạo chức năng quy đổi tiền tệ với ExchangeRate-API

Đầu tiên, tạo 1 template python3-http với lệnh:

```
sudo faas-cli new currency-converter --lang python3-http
```

Lúc này, thấy rằng nó đã tạo các file và thư mục như sau:



Hình 4.13: Thư mục được tạo dựa trên template python3-http

Handler.py là đoạn mã để tiến hành function, ở function này sẽ tiến hành gọi API từ API Provider là ExchangeRate-API để lấy thông tin. Tiến hành chỉnh sửa file handler.py trong thư mục currency-converter như sau:

```

GNU nano 7.2
handler.py

import requests
import json
import os

def handle(event, context):
    try:
        # Parse input JSON from event.body
        data = json.loads(event.body) # Use event.body instead of event
        amount = float(data.get("amount"))
        from_currency = data.get("from").upper()
        to_currency = data.get("to").upper()

        # Get API key from environment variable
        api_key = os.getenv("EXCHANGE_RATE_API_KEY")
        if not api_key:
            return json.dumps({"error": "API key not configured"})

        # Call ExchangeRate-API with timeout
        api_url = f"https://v6.exchangerate-api.com/v6/{api_key}/latest/{from_currency}"
        response = requests.get(api_url, timeout=10)
        response_data = response.json()

        # Check if API call was successful
        if response_data.get("result") != "success":
            return json.dumps({"error": "Failed to fetch exchange rates"})

        # Get conversion rate and calculate
        rate = response_data["conversion_rates"].get(to_currency)
        if not rate:
            return json.dumps({"error": f"Currency {to_currency} not supported"})

        converted_amount = amount * rate

        # Return result
        return json.dumps({
            "converted_amount": round(converted_amount, 2),
            "from": from_currency,
            "to": to_currency,
            "rate": rate
        })
    except ValueError as ve:
        return json.dumps({"error": "Invalid input: amount must be a number"})
    except requests.exceptions.RequestException as re:
        return json.dumps({"error": f"API request failed: {str(re)}"})
    except Exception as e:
        return json.dumps({"error": str(e)})

```

Hình 4.14: Nội dung file handler.py

Tiến hành sửa file như sau để định nghĩa function và thêm API Key:

```

g22520636@openfaas:~/openfaas$ cat currency-converter.yml
version: 1.0
provider:
  name: openfaas
  gateway: http://127.0.0.1:30080
functions:
  currency-converter:
    lang: python3-http
    handler: ./currency-converter
    image: khanhle04/currency-converter:latest
    environment:
      EXCHANGE RATE API KEY: a009187e2759f91d090a51ca

```

Hình 4.15: Nội dung file currency-converter.yml

Cuối cùng, build và deploy function trên bằng lệnh:

```
faas-cli up -f currency-converter.yml
```

```

g22520636@openfaas:~/openfaas$ faas-cli up -f currency-converter.yml
[0] > Building currency-converter...
Building: khanhle04/currency-converter:latest with python3-http template. Please wait..
2023/04/21 02:49:57 Build flags: [build --tag khanhle04/currency-converter:latest .]
#0 building with "default" instance using docker driver

#1 [internal] load build definition from Dockerfile
#1 transferring dockerfile: 1.69kB done
#1 WARN: RedundantTargetPlatform: Setting platform to predefined ${TARGETPLATFORM:-linux/amd64} in FROM is redundant as this is the default behavior (line 2)
#1 WARN: RedundantTargetPlatform: Setting platform to predefined ${TARGETPLATFORM:-linux/amd64} in FROM is redundant as this is the default behavior (line 3)
#1 DONE 0.0s

#2 [internal] load metadata for ghcr.io/openfaas/of-watchdog:0.10.7
#2 DONE 0.5s

#3 [internal] load metadata for docker.io/library/python:3.12-alpine
#3 DONE 0.7s

#4 [internal] load .dockerrigore
#4 transferring context: 2B done
#4 DONE 0.0s

#5 [watchdog 1/1] FROM ghcr.io/openfaas/of-watchdog:0.10.7@sha256:5ac3b18c1afad2ef85c4013c9acbd20746c2bc75b39fb0cf40e6f566d3096249
#5 DONE 0.0s

#6 [build 1/16] FROM docker.io/library/python:3.12-alpine@sha256:c08bfdbff9184cdff225497bac12b2c0dac1d24bbe13287cf7d99f1116cf43
#6 DONE 0.0s

#7 [internal] load build context
#7 transferring context: 4.99kB done
#7 DONE 0.0s

#8 [build 11/16] RUN mkdir -p function
#8 CACHED

#9 [build 5/16] RUN addgroup -S app && adduser app -S -G app
#9 CACHED

#10 [build 3/16] RUN chmod +x /usr/bin/fwatchdog
#10 CACHED

#11 [build 12/16] RUN touch ./function/__init__.py
#11 CACHED

#12 [build 6/16] RUN chown app /home/app

```

Hình 4.16: Build và deploy function quy đổi tiền tệ

Kiểm tra function, thấy rằng nó đã hoạt động:

Function	Invocations	Replicas
currency-converter	15	1

Hình 4.17: Kiểm tra function bằng command

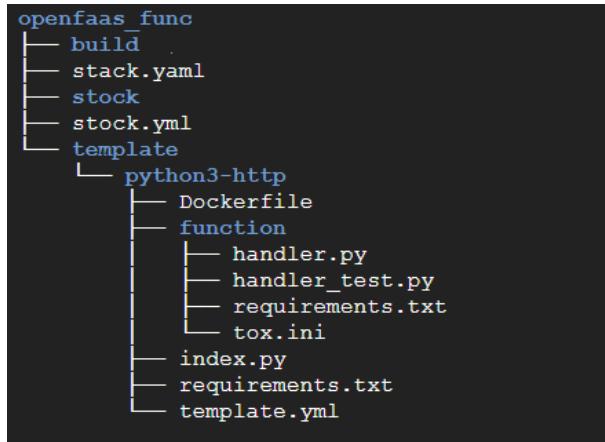
Hình 4.18: Kiểm tra function trên OpenFaaS Portal

4.3.2 Tạo chức năng theo dõi chứng khoán

Đầu tiên, tạo 1 template python3-http với lệnh:

```
sudo faas-cli new stock --lang python3-http
```

Lúc này, thấy rằng nó đã tạo các file và thư mục như sau:



Hình 4.19: Thư mục được tạo dựa trên template python3-http

Handler.py là đoạn mã để tiến hành function, ở function này sẽ tiến hành gọi API từ API Provider là AlphaVantageStockAPI để lấy thông tin. Tiến hành chỉnh sửa file handler.py trong thư mục stock như sau:

```

import requests
import json
import os

def handle(event, context):
    try:
        symbol = event.body.decode('utf-8').strip().strip('"').upper()
        if not symbol:
            return json.dumps({"error": "Stock symbol is required"})

        api_key = os.getenv("STOCK_API_KEY")
        url = f"https://www.alphavantage.co/query"
        params = {
            "function": "TIME_SERIES_DAILY",
            "symbol": symbol,
            "apikey": api_key
        }

        response = requests.get(url, params=params)
        print("API Response:", response.json())
        data = response.json()

        if "Error Message" in data:
            return json.dumps({"error": "Invalid stock symbol or API error"})

        time_series = data["Time Series (Daily)"]
        latest_date = sorted(time_series.keys())[-1]
        latest_close = time_series[latest_date]["4. close"]

        output = {
            "stock": symbol,
            "date": latest_date,
            "closing_price": f"${latest_close}",
            "full_data": time_series
        }

        return json.dumps(output, indent=2)

    except Exception as e:
        return json.dumps({"error": str(e)})

```

Hình 4.20: Nội dung file handler.py

Tiến hành sửa file như sau để định nghĩa function và thêm API Key:

```

GNU nano 7.2
version: 1.0
provider:
  name: openfaas
  gateway: http://34.143.131.146:30080
functions:
  stock:
    lang: python3-http
    handler: ./stock
    image: duongnvd/stock:latest
    environment:
      STOCK_API_KEY: 4J3KGOX75ELRMCQD

```

Hình 4.21: Nội dung file stock.yml

Cuối cùng, build và deploy function trên bằng lệnh:

```

faas-cli up -f stock.yml

```

```

root@openfaas:/home/g22520308/openfaas_func# faas-cli up -f stock.yaml
[0] Building stock
[0] Docker daemon is using hostpath volumes. Please wait...
2025/04/24 08:47:17 Build stage: Docker --tag dockerfaas/stock:latest ...
[0] Building with "default" instances using docker drives

[1] [internal] load build definition from Dockerfile
[1] [internal] transferring dockerfile: 1.69kB done
[1] [internal] redundantTARGETPLATFORM setting platform to predefined #!TARGETPLATFORM:-linux/amd64) in FROM is redundant as this is the default behavior (line 2)
[1] [internal] redundantTARGETPLATFORM setting platform to predefined #!TARGETPLATFORM:-linux/amd64) in FROM is redundant as this is the default behavior (line 3)
[1] DONE 0.0s

[2] [auth] library/python:pull token for registry-1.docker.io
[2] DONE 0.0s

[3] [internal] load metadata for gcr.io/openfaas/of-watchdog:0.10.7
[3] DONE 0.0s

[4] [internal] load metadata for docker.io/library/python:3.12-alpine
[4] DONE 1.4s

[5] [internal] load .dockerrcignore
[5] [internal] transferring contexts: 2B done
[5] DONE 0.0s

[6] [watchdog 1/1] FROM gcr.io/openfaas/of-watchdog:0.10.7@sha256:5ec3b18c1afad2ed95c4013c9acbd20744c2bc75b39fb0cf40e6f566d3096249
[6] DONE 0.0s

[7] [build 1/16] FROM docker.io/library/python:3.12-alpine@sha256:e08fd8effe9194ed64235497bae12b2e04da1d34bae13297e07d99451116e643
[7] DONE 0.0s

[8] [internal] load build context
[8] [internal] transferring contexts: 4.46kB done
[8] DONE 0.0s

[9] [build 13/16] WORKDIR /home/app/function/
[9] CACHED

[10] [build 16/16] COPY --chown=app:app function/requirements.txt .
[10] CACHED

[11] [build 14/16] COPY --chown=app:app function/
[11] CACHED

[12] [build 3/16] RUN chmod +x /usr/bin/watchdog
[12] CACHED

[13] [build 2/16] COPY --from=watchdog /watchdog /usr/bin/watchdog
[13] CACHED

[14] [build 7/16] WORKDIR /home/app/
[14] CACHED

[15] [build 8/16] COPY --chown=app:app index.py .
[15] CACHED

[16] [build 9/16] COPY --chown=app:app requirements.txt .
[16] CACHED

```

Hình 4.22: Build và deploy function xem giá chứng khoán

Kiểm tra function, thấy rằng nó đã hoạt động:

```

root@openfaas:/home/g22520308/openfaas_func# faas-cli list
Function          Invocations   Replicas
currency-converter    165        1
stock              178        1
root@openfaas:/home/g22520308/openfaas_func#

```

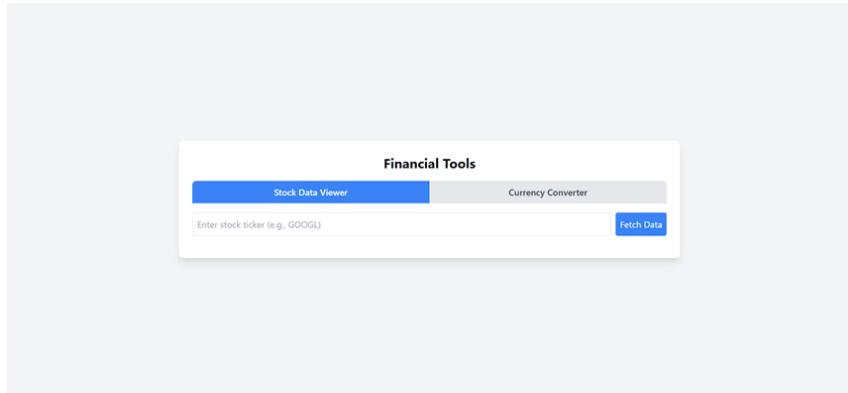
Hình 4.23: Kiểm tra function bằng command

Function	Invocations	Replicas
currency-converter	165	1
stock	178	1

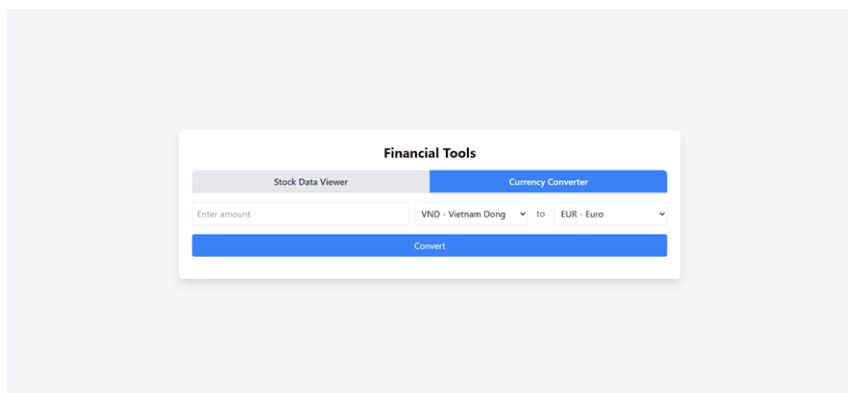
Hình 4.24: Kiểm tra function trên OpenFaaS Portal

4.3.3 Tạo website

Tiến hành tạo 1 web site với giao diện đơn giản sử dụng html và tailwindcss để hiển thị các thông tin và ô nhập liệu:



Hình 4.25: Giao diện web chức năng quy đổi tiền tệ



Hình 4.26: Giao diện web chức năng theo dõi chứng khoán

Sử dụng ngôn ngữ JavaScript xây dựng API proxy handler, dùng để trung gian nhận request từ frontend và gọi đến các dịch vụ phía backend (ở đây là các function của OpenFaaS):

```

export default async function handler(req, res) {
    if (req.method !== 'POST') {
        return res.status(405).json({ error: 'Method not allowed' });
    }

    let targetUrl, bodyToSend;

    if (typeof req.body === 'string') {
        targetUrl = 'http://34.143.131.146:30080/function/stock';
        bodyToSend = req.body;
    }

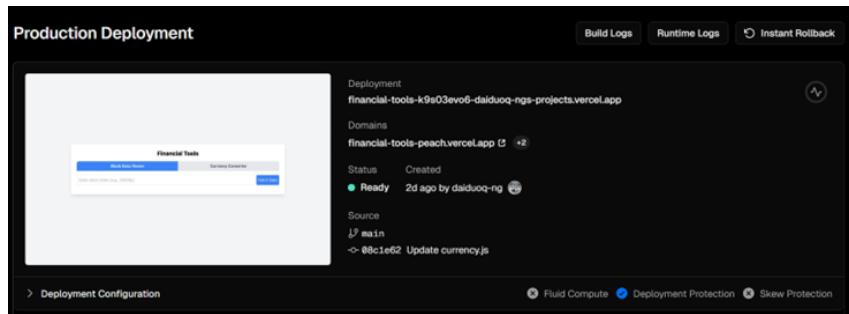
    else if (typeof req.body === 'object' && req.body !== null && req.body.type === 'currency') {
        targetUrl = 'http://34.143.131.146:30080/function/currency-converter';
        const { type, ...rest } = req.body;
        bodyToSend = rest;
    }

    else {
        return res.status(400).json({ error: 'Invalid request body: Must be a string or a currency object' });
    }
}

```

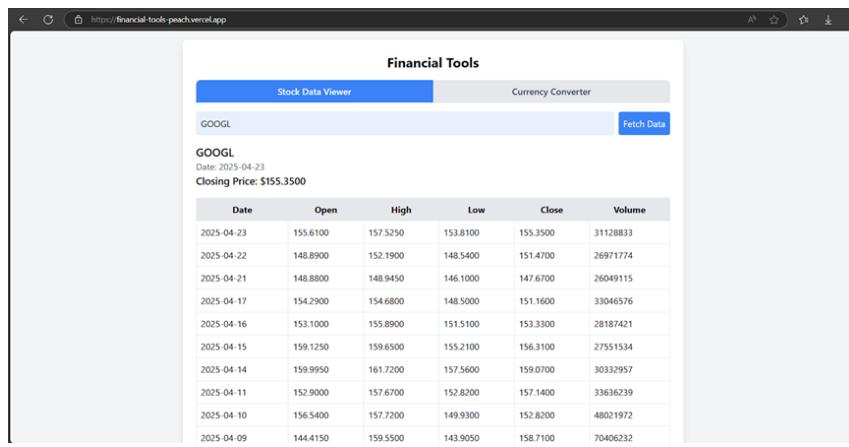
Hình 4.27: Nội dung file API proxy handler

Tiến hành deploy lên Vercel để có thể truy cập trực tiếp trang web từ Internet:

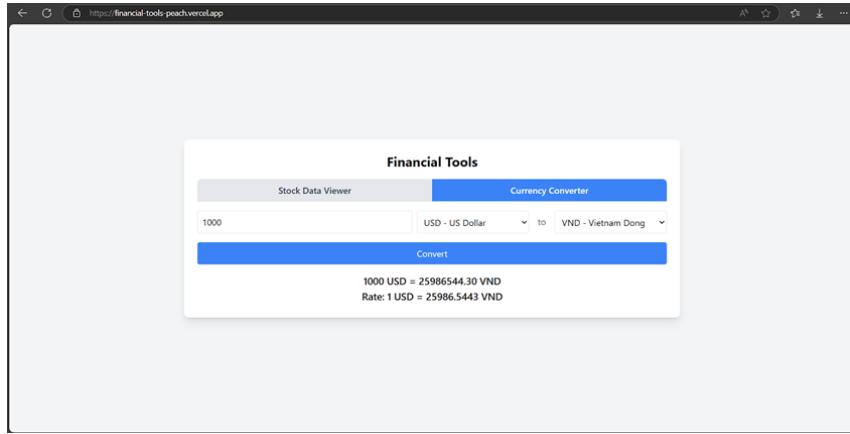


Hình 4.28: Deploy website lên Vercel

Ta có thể truy cập và sử dụng các tính năng tương ứng với các function đã định nghĩa:



Hình 4.29: Chức năng quy đổi tiền tệ



Hình 4.30: Chức năng theo dõi giá chứng khoán

4.3.4 Tạo Webhooks function

Tiến hành cài đặt GitLab trong Kubernetes cluster:

```
g22520308@openfaas:~$ helm repo add gitlab https://charts.gitlab.io/
"gitlab" has been added to your repositories
g22520308@openfaas:~$ helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "gitlab" chart repository
Update Complete. *Happy Helming!*
g22520308@openfaas:~$ kubectl create namespace gitlab
namespace/gitlab created
```

Hình 4.31: Thêm và cập nhật GitLab trên OpenFaaS và namespace

```
g22520308@openfaas:~$ helm install gitlab gitlab/gitlab \
--namespace gitlab \
--set global.hosts.domain=34.143.131.146.nip.io \
--set global.hosts.externalIP=34.143.131.146 \
--set certmanager-issuer.email=daiduongnguyen102@gmail.com
```

Hình 4.32: Thiết lập cài đặt GitLab trên OpenFaaS

Tiến hành tạo 1 template python3-http với lệnh:

```
sudo faas-cli new gitlab-telegram-notify --lang python3-http
```

```

g22520308@openfaas:~$ sudo faas-cli new gitlab-telegram-notify --lang python3-http
Fetch templates from repository: https://github.com/openfaas/python-flask-template
Wrote 1 template(s) : [python3-http] from https://github.com/openfaas/python-flask-template
Folder: gitlab-telegram-notify created.

Function created in folder: gitlab-telegram-notify
Stack file written: stack.yaml
g22520308@openfaas:~$ 

```

Hình 4.33: Tạo 1 template mới trên OpenFaaS

Handler.py là đoạn mã để thực hiện function, ở function này sẽ tiến hành tiếp nhận và xử lý dữ liệu từ webhook GitLab. Tiến hành chỉnh sửa file *handler.py* trong thư mục *gitlab-telegram-notify* như sau:

```

import json
import os
import requests
import logging

# Set up logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

def handle(event, context=None):
    """Handle GitLab webhook and send notification to Telegram"""
    logger.info(f"Received event: {event}")
    logger.info(f"Event type: {type(event)}")

    if isinstance(event, dict) and 'body' in event:
        req = event['body']
    elif hasattr(event, 'body'):
        req = event.body
    else:
        req = event

    if isinstance(req, bytes):
        req = req.decode('utf-8')
        gitlab_data = json.loads(req)
    elif isinstance(req, str):
        gitlab_data = json.loads(req)
    elif isinstance(req, dict):
        gitlab_data = req
    else:
        gitlab_data = {}

    logger.info(f"GitLab data: {gitlab_data}")

    event_type = gitlab_data.get("object_kind", "unknown")
    project = gitlab_data.get("project", {}).get("name", "unknown")

```

```

elif isinstance(req, dict):
    gitlab_data = req
else:
    gitlab_data = {}

logger.info(f"GitLab data: {gitlab_data}")

event_type = gitlab_data.get("object_kind", "unknown")
project = gitlab_data.get("project", {}).get("name", "unknown")
if event_type == "push":
    user = gitlab_data.get("user_name", "someone")
    message = f"{user} đã push code vào dự án {project}"
elif event_type == "merge_request":
    user = gitlab_data.get("user", {}).get("name", "someone")
    mr_title = gitlab_data.get("object_attributes", {}).get("title", "untitled")
    message = f"{user} đã tạo merge request '{mr_title}' trong dự án {project}"
elif event_type == "issue":
    user = gitlab_data.get("user", {}).get("name", "someone")
    issue_title = gitlab_data.get("object_attributes", {}).get("title", "untitled")
    message = f"{user} đã tạo issue '{issue_title}' trong dự án {project}"
else:
    message = f"⚠️ Có sự kiện {event_type} mới trong dự án {project}"

bot_token = os.environ.get("TELEGRAM_BOT_TOKEN")
chat_id = os.environ.get("TELEGRAM_CHAT_ID")

telegram_api_url = f"https://api.telegram.org/bot{bot_token}/sendMessage"
telegram_data = {
    "chat_id": chat_id,
    "text": message,
    "parse_mode": "HTML"
}

requests.post(telegram_api_url, json=telegram_data)

```

Hình 4.34: Nội dung file handler.py

File **Handler.py** có chức năng chính là xử lý payload từ webhook GitLab khi có sự kiện như *push*, *merge request* hoặc *issue*, và gửi thông báo đến Telegram thông qua Telegram Bot API. Cụ thể như sau:

Nhận và xử lý dữ liệu webhook:

- Đầu tiên, nhận dữ liệu webhook từ GitLab gửi đến (qua tham số event), sau đó kiểm tra và giải mã nội dung (body) thành một đối tượng JSON (gitlab_data).

Phân tích loại sự kiện:

- Nếu sự kiện là *push*, lấy thông tin người dùng và dự án rồi tạo nội dung tin nhắn thông báo việc đẩy code.
- Nếu sự kiện là *merge request*, lấy thông tin người tạo *merge request* và tiêu đề để soạn nội dung thông báo.
- Nếu sự kiện là *issue*, lấy thông tin người tạo *issue* và tiêu đề để soạn nội dung tin nhắn.
- Các sự kiện khác sẽ được gửi với nội dung thông báo chung.

Gửi tin nhắn Telegram:

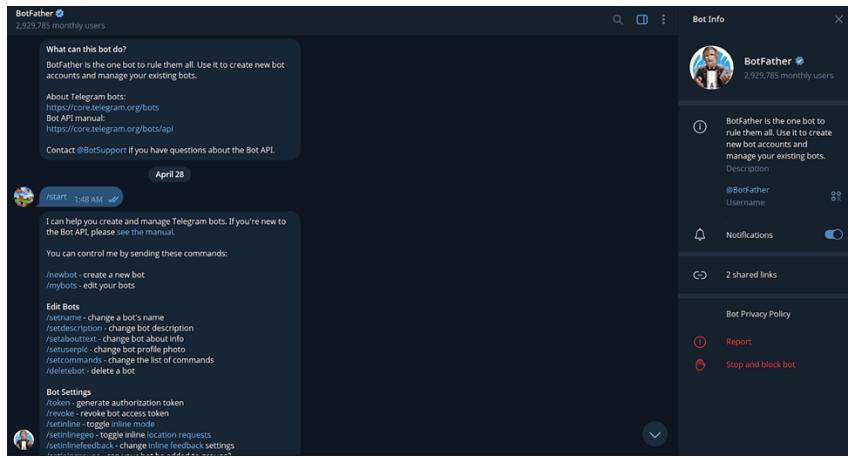
- Lấy TELEGRAM_BOT_TOKEN và TELEGRAM_CHAT_ID từ biến môi trường.

- Gọi API Telegram `https://api.telegram.org/bot<token>/sendMessage` bằng phương thức POST, đính kèm `chat_id` và nội dung message vào payload gửi đi.

Trong đó:

- `TELEGRAM_BOT_TOKEN` là token của bot Telegram được dùng để xác thực.
- `TELEGRAM_CHAT_ID` là ID của người nhận hoặc nhóm sẽ nhận tin nhắn.

Tiếp theo, tạo một bot Telegram để nhận thông báo từ các sự kiện của GitLab:



Hình 4.35: Tạo bot trong Telegram bằng BotFather

Tiến hành sửa file `gitlab-telegram-notify.yml` như sau để định nghĩa function:

```
GNU nano 7.2                                     gitlab-telegram-notify.yml
version: 1.0
provider:
  name: openfaas
  gateway: http://34.143.131.146:30080
functions:
  gitlab-telegram-notify:
    lang: python3-http
    handler: ./gitlab-telegram-notify
    image: duongnvd/gitlab-telegram-notify:latest
    environment:
      TELEGRAM_BOT_TOKEN: "7973064027:AAHfhhsryHgj5D0_cjDKlkIQMnJ-bur4kGXl"
      TELEGRAM_CHAT_ID: "1216414407"
    secrets:
      - telegram-secrets
```

Hình 4.36: Nội dung file gitlab-telegram-notify.yml

Cuối cùng, build và deploy function trên bằng lệnh:

```
faas-cli up -f gitlab-telegram-notify.yml
```

```

Image: duongnvd/gitlab-telegram-notify:latest built.
[0] < Building gitlab-telegram-notify done in 0.90s.
[0] Worker done.

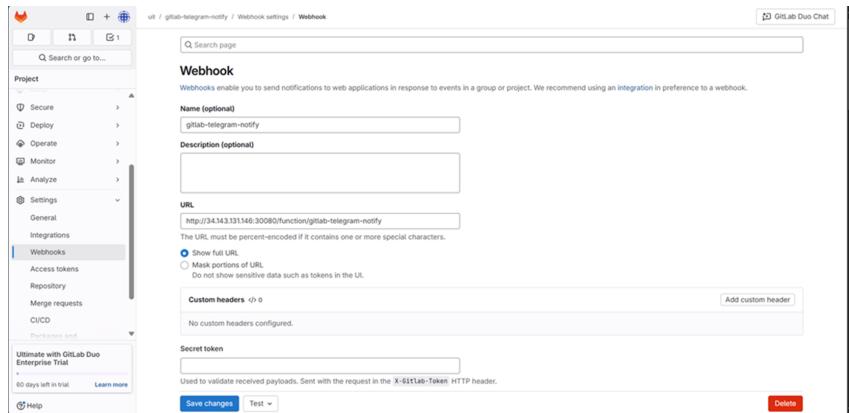
Total build time: 0.90s
[0] > Pushing gitlab-telegram-notify [duongnvd/gitlab-telegram-notify:latest]
The push refers to repository [docker.io/duongnvd/gitlab-telegram-notify]
5f70bf18a086: Layer already exists
b847285278ca: Layer already exists
200af6f1c42: Layer already exists
451946de7f0d: Layer already exists
78c934192848: Layer already exists
d04ae4af2f43: Layer already exists
584ed2df7f7e: Layer already exists
d296cd4a69c: Layer already exists
097e1d3e06c8: Layer already exists
aac18114cdcf: Layer already exists
551e36404a7e: Layer already exists
a67b040aaa65: Layer already exists
e6778ff47b23: Layer already exists
035c120d2311: Layer already exists
c5f83227c988: Layer already exists
5a73889e102c: Layer already exists
2748e349efdf5: Layer already exists
08000c18d16d: Layer already exists
latest: digest: sha256:84fde95e94834cel144671f0b42a5ae7a607055d8545551d2d51f68c43dd7d size: 4483
[0] < Pushing gitlab-telegram-notify [duongnvd/gitlab-telegram-notify:latest] done.
[0] Worker done.
Deploying: gitlab-telegram-notify.
WARNING! You are not using an encrypted connection to the gateway, consider using HTTPS.

Deployed. 202 Accepted.
URL: http://34.143.131.146:30080/function/gitlab-telegram-notify
root@openfaas:/home/g22520308#

```

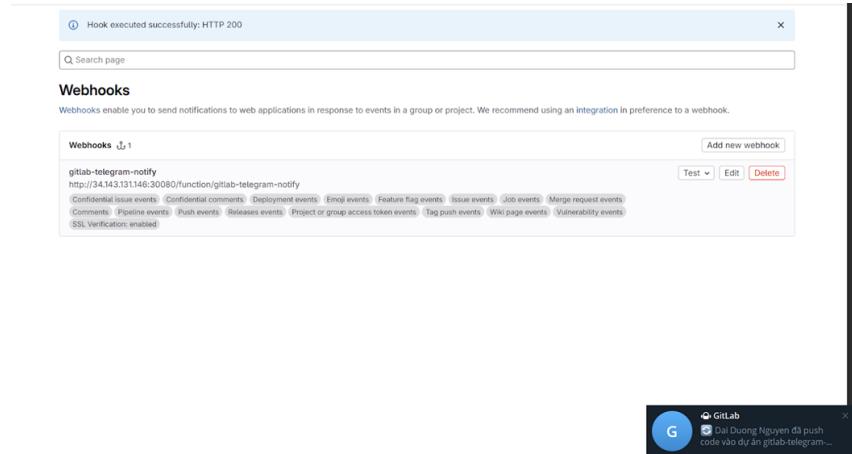
Hình 4.37: Build và deploy function webhooks

Cấu hình Webhook trên Gitlab:



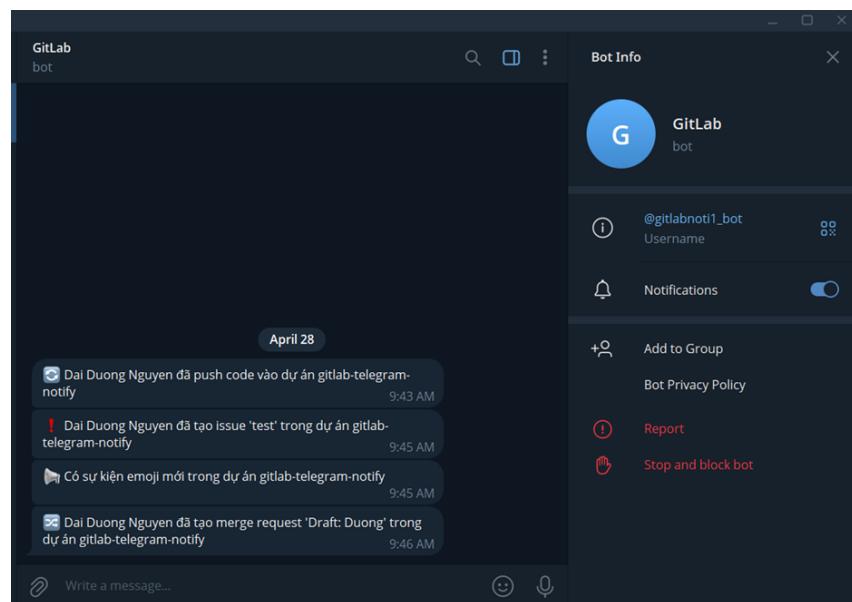
Hình 4.38: Tạo webhook với function vừa deploy

Tiến hành kiểm tra webhook đã được cấu hình đúng hay không:



Hình 4.39: Tạo webhook với function vừa deploy

Ta thấy rằng với mỗi sự kiện mà người dùng thực hiện trên GitLab ta đều nhận được thông báo tương ứng trên Telegram:



Hình 4.40: Thông báo các sự kiện trên Telegram

4.4 Cài đặt hệ thống giám sát cho OpenFaaS

4.4.1 Prometheus

Vì service Prometheus đã được cài đặt sẵn trên OpenFaaS, bây giờ việc ta cần làm là tạo service trên dưới dạng NodePort.

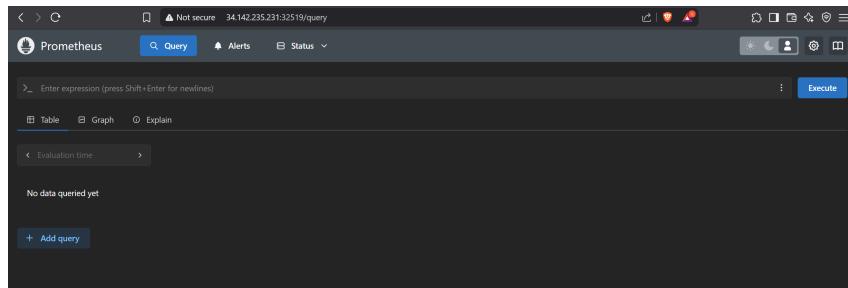
Tạo service trên dưới dạng NodePort bằng lệnh:

```
sudo kubectl expose deployment prometheus -n openfaas --type=NodePort
--name=prometheus-ui
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT (S)	AGE
alertmanager	ClusterIP	10.43.27.47	<none>	9093/TCP	7d20h
gateway	ClusterIP	10.43.158.224	<none>	8080/TCP	7d20h
gateway-external	NodePort	10.43.31.149	<none>	8080:31112/TCP	7d20h
nats	ClusterIP	10.43.10.112	<none>	4222/TCP	7d20h
openfaas-gateway	NodePort	10.43.161.134	<none>	80:30080/TCP	7d19h
prometheus	ClusterIP	10.43.84.51	<none>	9090/TCP	7d20h
prometheus-ui	NodePort	10.43.161.113	<none>	9090:32519/TCP	57s

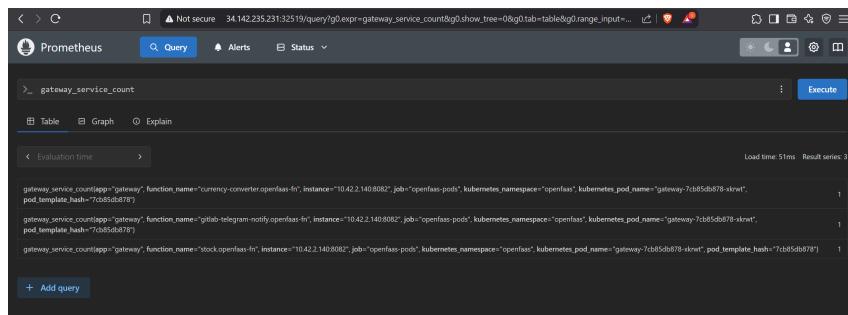
Hình 4.41: Kiểm tra service Prometheus UI

Tiến hành truy cập vào UI trên port 32519, thấy rằng đã truy cập thành công:



Hình 4.42: Màn hình chính của Prometheus UI

Thử nghiệm query một metric, thấy rằng nó đã trả về kết quả, chứng tỏ rằng Prometheus đã hoạt động tốt và không có lỗi:



Hình 4.43: Query Prometheus UI

4.4.2 Grafana

Để có thể cài đặt Grafana trên OpenFaaS, nhóm đã tham khảo nguồn GitHub sau để có thể tiến hành cài đặt:

```
http://github.com/stefanprodan/faas-grafana
```

Thực hiện lệnh sau để tạo pod Grafana trong namespace openfaas, mở cổng 3000 cho container:

```
sudo kubectl run grafana -n openfaas --image=stefanprodan/faas-grafana:4.6.3 --port=3000
```

Tiếp đến, tạo một service loại NodePort tên grafana trong namespace openfaas, trỏ đến pod grafana, expose cổng (mặc định từ pod) qua một NodePort để truy cập từ bên ngoài cluster, sử dụng lệnh sau:

```
sudo kubectl -n openfaas expose pods grafana --type=NodePort --name=grafana
```

Kiểm tra service, thấy rằng đã có service Grafana, hoạt động dưới loại NodePort:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
alertmanager	ClusterIP	10.43.27.47	<none>	9093/TCP	7d21h
gateway	ClusterIP	10.43.158.224	<none>	8080/TCP	7d21h
gateway-external	NodePort	10.43.31.149	<none>	8080:31112/TCP	7d21h
grafana	NodePort	10.43.178.73	<none>	3000:32174/TCP	34s
nats	ClusterIP	10.43.10.112	<none>	4222/TCP	7d21h
openfaas-gateway	NodePort	10.43.161.134	<none>	80:30080/TCP	7d20h
prometheus	ClusterIP	10.43.84.51	<none>	9090/TCP	7d21h
prometheus-ui	NodePort	10.43.161.113	<none>	9090:32519/TCP	70m

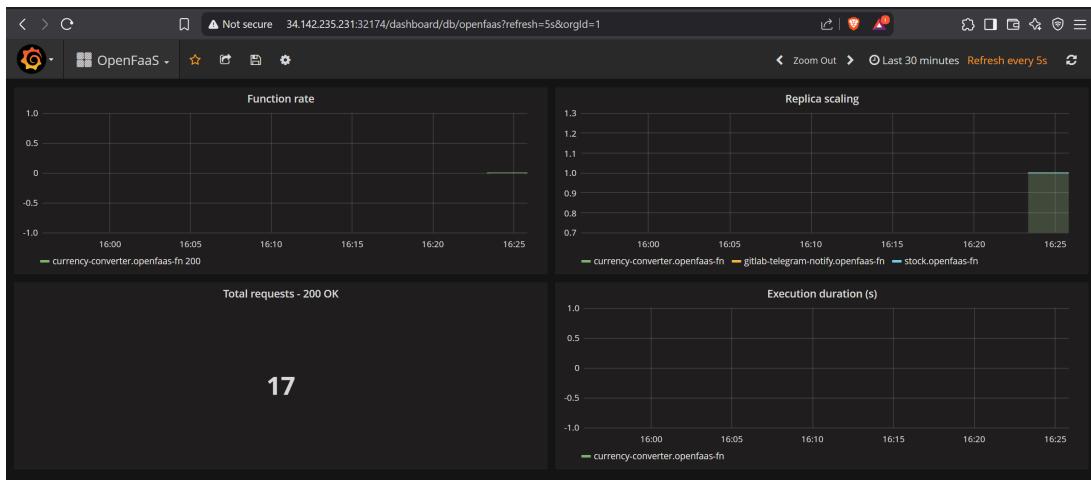
Hình 4.44: Kiểm tra service Grafana

Forward port 32174 tới port 3000 của service Grafana, tiến hành truy cập trên trình duyệt với port 32174, thấy rằng đã truy cập được vào Grafana UI. Đăng nhập bằng tài khoản mật khẩu mặc định là **admin/admin**:

```
g22520636@openfaas:~$ sudo kubectl port-forward -n openfaas svc/grafana 32174:3000
Forwarding from 127.0.0.1:32174 -> 3000
Forwarding from [::1]:32174 -> 3000
```

Hình 4.45: Forward Port để truy cập Grafana UI

Chương 4



Hình 4.46: OpenFaaS Dashboard trên Grafana

Tài liệu tham khảo

- [1] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, “Borg, omega, and kubernetes,” *Commun. ACM*, vol. 59, no. 5, pp. 50–57, Apr. 2016, ISSN: 0001-0782. doi: [10.1145/2890784](https://doi.org/10.1145/2890784). [Online]. Available: <https://doi.org/10.1145/2890784>.
- [2] Rancher Labs, *K3s – Lightweight Kubernetes*, <https://k3s.io/>, Accessed: 2025-04-16, 2025.
- [3] OpenFaaS, *Arkade - the open source kubernetes marketplace*, <https://github.com/alexellis/arkade>, Accessed: 2025-05-01, 2025.
- [4] OpenFaaS Docs, *Faas-cli documentation*, <https://docs.openfaas.com/cli/>, Accessed: 2025-04-25, 2025.
- [5] GitLab, *Gitlab webhooks*, <https://docs.gitlab.com/ee/user/project/integrations/webhooks.html>, Accessed: 2025-04-20, 2025.