

ĐẠI HỌC BÁCH KHOA HÀ NỘI

ĐỒ ÁN TỐT NGHIỆP

Xây dựng hệ thống quản lý ảo hóa chức năng
mạng(MANO-NFV)

THÁI BẢO TRUNG

trungtb204858@sis.hust.edu.vn

Ngành: Kỹ thuật máy tính

Giảng viên hướng dẫn: TS. Trần Hoàng Hải

Chữ kí GVHD

Khoa: Kỹ thuật máy tính

Trường: Công nghệ Thông tin và Truyền thông

HÀ NỘI, 06/2024

LỜI CAM KẾT

Tôi – *Thái Bảo Trung* – cam kết Đồ án Tốt nghiệp (ĐATN) là công trình nghiên cứu của bản thân tôi dưới sự hướng dẫn của *Tiến Sĩ Trần Hoàng Hải*. Các kết quả nêu trong ĐATN là trung thực, là thành quả của riêng tôi, không sao chép theo bất kỳ công trình nào khác. Tất cả những tham khảo trong ĐATN – bao gồm hình ảnh, bảng biểu, số liệu, và các câu từ trích dẫn – đều được ghi rõ ràng và đầy đủ nguồn gốc trong danh mục tài liệu tham khảo. Tôi xin hoàn toàn chịu trách nhiệm với dù chỉ một sao chép vi phạm quy chế của nhà trường.

Hà Nội, ngày tháng năm

Tác giả ĐATN

Họ và tên sinh viên

LỜI CẢM ƠN

Em xin gửi lời cảm ơn chân thành và sâu sắc nhất tới thầy Trần Hoàng Hải-Phó Giám đốc Trung tâm Mạng thông tin, Giảng viên Khoa Kỹ thuật máy tính , Trường Công nghệ thông tin và Truyền thông - Đại học Bách Khoa Hà Nội đã giúp đỡ và luôn tận tình chỉ bảo, hướng dẫn để em có thể hoàn thành tốt đồ án tốt nghiệp này. Em xin chân thành cảm ơn các thầy, cô, CBNV Đại học Bách khoa Hà Nội đã truyền đạt những kiến thức, kinh nghiệm và kỹ năng không chỉ trong quá trình làm đồ án này mà còn trong cả quá trình học tập rèn luyện tại trường. Em cũng xin gửi lời cảm ơn tới anh Nguyễn Xuân Chính-trung tâm nghiên cứu OCS- Tổng Công Ty Công Nghiệp Công Nghệ Cao Viettel đã tận tình hướng dẫn, chỉ bảo về vấn đề chuyên môn, cảm ơn gia đình, bạn bè đã ủng hộ, động viên em trong suốt quá trình học tập vừa qua để em có ngày hôm nay được thực hiện đồ án này. Do trong quá trình nghiên cứu, tìm hiểu và thực nghiệm đồ án chắc chắn không thể tránh khỏi những sai sót nhất định, em rất mong nhận được sự góp ý của thầy, cô giáo và các bạn để đồ án được hoàn chỉnh hơn. Xin trân trọng cảm ơn!

TÓM TẮT NỘI DUNG ĐỒ ÁN

Trong bối cảnh mạng viễn thông hiện đại, sự phát triển nhanh chóng của các dịch vụ và ứng dụng đòi hỏi một cơ sở hạ tầng linh hoạt và hiệu quả. Quản lý các chức năng mạng ảo (NFV-MANO) trở thành một trong những vấn đề trọng yếu cần được giải quyết. Hiện nay, việc triển khai NFV-MANO vẫn gặp nhiều thách thức, bao gồm khả năng tương tác giữa các thành phần khác nhau, khả năng mở rộng, và tối ưu hóa tài nguyên. Một số hướng tiếp cận như sử dụng các kiến trúc phần mềm phân tán, áp dụng công nghệ đám mây, và phát triển các giao thức quản lý tiên tiến đã được đề xuất nhằm cải thiện hiệu suất và hiệu quả của hệ thống NFV-MANO. Tuy nhiên, những giải pháp này vẫn tồn tại hạn chế về độ phức tạp, khó khăn trong triển khai thực tế và chi phí cao.

Đóng góp chính của ĐATN này là xây dựng một mô hình NFV-MANO linh hoạt và hiệu quả, giúp giảm thiểu độ phức tạp và tăng cường khả năng mở rộng của hệ thống. Kết quả thử nghiệm cho thấy giải pháp đề xuất đã cải thiện hiệu suất quản lý tài nguyên, giảm thiểu thời gian triển khai và chi phí vận hành, mở ra tiềm năng ứng dụng rộng rãi trong các mạng viễn thông hiện đại.

- Chương 1: Giới thiệu về đề tài: Tổng quan về các vấn đề trong việc quản lý các chức năng mạng ảo hóa, đưa ra vấn đề cần giải quyết và phương án đề xuất.
- Chương 2: Phân tích và đặc tả chức năng : Phân tích và đặc tả chi tiết các chức năng có trong hệ thống. - Chương 3: Công nghệ sử dụng: Mô tả các công nghệ sử dụng, các ưu và nhược điểm của các công nghệ đó. -Chương4:Thiết kế, triển khai và đánh giá hệ thống :Phân tích, thiết kế mô hình triển khai và thực nghiệm đánh giá chức năng - Chương 5: Các giải pháp đóng góp nổi bật: Mô tả các đóng góp cải tiến, giúp ích nổi bật đã thực hiện. - Chương 6: Kết luận và hướng phát triển: Trình bày các kết quả đạt được và phương hướng phát triển trong tương lai.

Sinh viên thực hiện
(Ký và ghi rõ họ tên)

ABSTRACT

In the modern telecommunications network landscape, the rapid development of services and applications requires a flexible and efficient infrastructure. Managing virtual network functions (NFV-MANO) becomes one of the key issues that need to be addressed. Currently, implementing NFV-MANO still faces many challenges, including interoperability between different components, scalability, and resource optimization. Several approaches such as using distributed software architectures, applying cloud technology, and developing advanced management protocols have been proposed to improve the performance and efficiency of NFV-systems. MANO. However, these solutions still have limitations in complexity, difficulty in practical implementation and high cost.

The main contribution of this DATN is to build a flexible and efficient NFV-MANO model, which helps reduce complexity and enhance the scalability of the system. Experimental results show that the proposed solution improves resource management performance, reduces deployment time and operating costs, opening up the potential for widespread application in modern telecommunications networks.

- Chapter 1: Introduction to the topic: Overview of issues in managing virtualized network functions, presenting problems to be solved and proposed solutions. - Chapter 2: Analysis and specification of functions: Analysis and detailed specification of functions in the system. - Chapter 3: Technologies used: Describes the technologies used, the advantages and disadvantages of those technologies. -Chapter 4: System design, implementation and evaluation: Analysis, design of deployment models and experimental evaluation of functions - Chapter 5: Outstanding contribution solutions: Describe the outstanding improvement and helpful contributions that have been made. - Chapter 6: Conclusion and development direction: Presenting the achieved results and future development directions.

MỤC LỤC

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI.....	1
1.1 Đặt vấn đề.....	1
1.2 Định hướng giải pháp.....	1
1.3 Bố cục đồ án	2
CHƯƠNG 2. PHÂN TÍCH VÀ ĐẶC TẢ CHỨC NĂNG.....	3
2.1 Tổng quan chức năng	3
2.1.1 Sơ đồ hệ thống.....	3
2.1.2 Biểu đồ use case tổng quát	4
2.1.3 Biểu đồ use case phân rã chức năng Quản lý vòng đời VNF	5
2.1.4 Biểu đồ use case phân rã chức năng thu thập, giám sát hiệu năng....	6
2.1.5 Biểu đồ use case phân rã chức năng Quản lý cảnh báo	6
2.2 Đặc tả chức năng	7
2.2.1 Đặc tả use case Create VNF Instance	7
2.2.2 Đặc tả use case Instantiate VNF instance.....	8
2.2.3 Đặc tả use case Terminate VNF Instance	9
2.2.4 Đặc tả use case Healing VNF Instance	10
2.2.5 Đặc tả use case Scale VNFC.....	11
2.2.6 Đặc tả use case LCMOPOCC VNF Instance.....	12
2.3 Yêu cầu phi chức năng	12
CHƯƠNG 3. CÔNG NGHỆ SỬ DỤNG.....	13
3.1 Kubernetes.....	13
3.2 Prometheus	14
3.3 Grafana.....	16
3.4 Framework Quarkus.....	17

3.5 TypeScript	18
3.6 Fabric8 Kubernetes Java Client	19
3.7 SnakeYAML	20
CHƯƠNG 4. THIẾT KẾ, TRIỂN KHAI VÀ ĐÁNH GIÁ HỆ THỐNG	22
4.1 Thiết kế kiến trúc.....	22
4.1.1 Lựa chọn kiến trúc phần mềm	22
4.1.2 Thiết kế chi tiết gói	24
4.2 Thiết kế chi tiết.....	26
4.2.1 Thiết kế lớp	26
4.2.2 Thiết kế cơ sở dữ liệu	29
4.3 Xây dựng ứng dụng.....	33
4.3.1 Thư viện và công cụ sử dụng	33
4.3.2 Kết quả đạt được	34
4.3.3 Minh họa các chức năng chính	34
4.4 Kiểm thử.....	42
CHƯƠNG 5. CÁC GIẢI PHÁP VÀ ĐÓNG GÓP NỔI BẬT	44
5.1 Giải pháp triển khai ứng dụng dưới dạng container [1]	44
5.2 Giải pháp quản lý vòng đời các chức năng mạng ảo hóa	45
5.3 Giải pháp thu thập và giám sát hiệu năng hệ thống.....	46
5.4 Giải pháp Quản lý cảnh báo lỗi hệ thống	46
CHƯƠNG 6. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN	48
6.1 Kết luận	48
6.2 Hướng phát triển.....	48
TÀI LIỆU THAM KHẢO.....	49

DANH MỤC HÌNH VẼ

Hình 2.1	Biểu đồ deployment của hệ thống	3
Hình 2.2	Biểu đồ use case tổng quát	4
Hình 2.3	Biểu đồ use case phân rã chức năng Quản lý vòng đời VNF . .	5
Hình 2.4	Biểu đồ use case phân rã chức năng thu thập, giám sát hiệu năng hệ thống	6
Hình 2.5	Biểu đồ use case phân rã chức năng Quản lý cảnh báo lỗi . . .	6
Hình 4.1	Biểu đồ chi tiết gói Model	24
Hình 4.2	Biểu đồ chi tiết gói Controller	25
Hình 4.3	Thiết kế chi tiết lớp VNFD	26
Hình 4.4	Thiết kế chi tiết lớp VNF Instance	27
Hình 4.5	Thiết kế chi tiết lớp VNFc	28
Hình 4.6	Giao diện đăng nhập người dùng	34
Hình 4.7	Giao diện chức năng tạo VNFD	35
Hình 4.8	Giao diện chức năng khởi tạo VNF Instance	35
Hình 4.9	Giao diện chức năng bật VNF Instance	36
Hình 4.10	Giao diện chức năng tắt VNF Instance	36
Hình 4.11	Giao diện chức năng phục hồi VNF Instance	37
Hình 4.12	Giao diện chức năng xóa VNF Instance	37
Hình 4.13	Giao diện chức năng mở rộng VNF Instance	38
Hình 4.14	Giao diện chức năng mở rộng VNF Instance	38
Hình 4.15	Giao diện chức năng xem chi tiết các thành phần trong VNF Instance	39
Hình 4.16	Giao diện chức năng xem chi tiết các thành phần trong VNF Instance	39
Hình 4.17	Giao diện chức năng LCMOPOCC	40
Hình 4.18	Giao diện chức năng LCMOPOCC	40
Hình 4.19	Giao diện chức năng thu thập, giám sát hiệu năng hệ thống . .	41
Hình 4.20	Giao diện chức năng quản lý cảnh báo lỗi hệ thống	41

DANH MỤC BẢNG BIỂU

Bảng 2.1	Đặc tả usecase Create VNF instance	7
Bảng 2.2	Đặc tả usecase Instantiate VNF instance	8
Bảng 2.3	Đặc tả usecase Terminate VNF instance	9
Bảng 2.4	Đặc tả usecase Healing VNF instance	10
Bảng 2.5	Đặc tả usecase Scale VNFc	11
Bảng 2.6	Đặc tả usecase LCMOPOCC VNF instance	12
Bảng 4.1	Đặc tả lớp VNFD	26
Bảng 4.2	Đặc tả lớp VNF Instance	27
Bảng 4.3	Đặc tả lớp VNFD	28
Bảng 4.4	Đặc tả cơ sở dữ liệu VNF instance	29
Bảng 4.5	Đặc tả cơ sở dữ liệu VNFc	29
Bảng 4.6	Đặc tả cơ sở dữ liệu VNFD	29
Bảng 4.7	Đặc tả cơ sở dữ liệu LCMOPOCC	30
Bảng 4.8	Đặc tả cơ sở dữ liệu Deployment	30
Bảng 4.9	Đặc tả cơ sở dữ liệu Service	30
Bảng 4.10	Đặc tả cơ sở dữ liệu VNF Configmap	31
Bảng 4.11	Đặc tả cơ sở dữ liệu Secret	32
Bảng 4.12	Đặc tả cơ sở dữ liệu Container	32
Bảng 4.13	Danh sách thư viện và công cụ sử dụng	33
Bảng 4.14	Kiểm thử chức năng tạo VNF instance	42
Bảng 4.15	Kiểm thử chức năng bật VNF instance	42
Bảng 4.16	Kiểm thử chức năng scale VNF instance	43

DANH MỤC THUẬT NGỮ VÀ TỪ VIẾT TẮT

Viết tắt	Tên tiếng Anh	Tên tiếng Việt
API	Application Programming Interface	Giao diện lập trình ứng dụng
IOT	Internet Of Things	Mạng lưới kết nối vạn vật internet
NFV	Network functional Virtualization	Ảo hóa chức năng mạng
NFV-MANO	Network functional Virtualization Management and Orchestration	Hệ thống quản lý ảo hóa chức năng mạng
NFVI	NFV Infrastructure	Cơ sở hạ tầng ảo hóa chức năng mạng
VNF	Virtualized Network Function	Hàm chức năng mạng được ảo hóa
VNFc	Virtualized Network Function Component	Thành phần chức năng mạng ảo hóa
PNF	Physical Network Function	Hàm chức năng mạng vật lý
EMS	Element Management System	Hệ thống quản lý phần tử mạng
VIM	Virtualized Infrastructure Manager	Trình quản lý cơ sở hạ tầng ảo hóa
VNFM	Virtual Network Function Manager	Quản lý hàm chức năng mạng ảo
VNFD	Virtualized Network Function Descriptor	Bản mô tả hàm chức năng mạng ảo hóa
KVM	Kernel-based Virtual Machine	Máy ảo dựa trên nhân
OSS/BSS	Operation/Business Support System	Hệ thống hỗ trợ vận hành/kinh doanh
NS	Network Service	Dịch vụ mạng
ETSI	European Telecommunications Standards Institute	Viện tiêu chuẩn viễn thông Châu Âu
3GPP	3rd Generation Partnership Project	Dự án hợp tác thế hệ thứ 3
IETF	Internet Engineering Task Force	Lực lượng đặc nhiệm kỹ thuật Internet
NAT	Network Address Translation	Chuyển đổi địa chỉ mạng
IaaS	Infrastructure as a Service	Cơ sở hạ tầng dưới dạng dịch vụ
PaaS	Platform as a Service	Nền tảng dưới dạng dịch vụ
NF	Network Function	Hàm chức năng mạng

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI

1.1 Đặt vấn đề

Trong bối cảnh các nhà cung cấp dịch vụ mạng ngày càng phụ thuộc vào hạ tầng mạng ảo hóa (NFV - Network Functions Virtualization) để cải thiện hiệu quả và giảm chi phí, các giải pháp quản lý và điều phối NFV, hay NFV-MANO (NFV Management and Orchestration), đã trở thành yếu tố then chốt. Tuy nhiên, mặc dù NFV-MANO mang lại nhiều lợi ích, các giải pháp hiện có cũng không thiếu những hạn chế và thách thức đáng kể.

Trước hết, giá thành của các giải pháp NFV-MANO hiện nay khá cao, gây khó khăn cho các doanh nghiệp vừa và nhỏ khi muốn triển khai. Chi phí đầu tư ban đầu cho các giải pháp này bao gồm phần cứng, phần mềm, và các dịch vụ triển khai, cấu hình, bảo trì. Đối với nhiều doanh nghiệp, đặc biệt là những doanh nghiệp mới hoặc có quy mô nhỏ, việc chi trả cho những khoản đầu tư lớn này là một thách thức không nhỏ.

Thêm vào đó, việc quản lý và điều phối các chức năng mạng ảo (VNFs) phức tạp cũng đặt ra nhiều thách thức. Mặc dù NFV-MANO có khả năng tự động hóa và tối ưu hóa quá trình này, nhưng thực tế cho thấy việc duy trì và vận hành các VNFs một cách hiệu quả vẫn gặp nhiều khó khăn. Sự phức tạp trong việc giám sát và khắc phục sự cố đòi hỏi các công cụ quản lý tiên tiến và đội ngũ kỹ thuật có chuyên môn cao, làm tăng chi phí vận hành và quản lý.

Những thách thức này đặt ra yêu cầu cấp thiết cho việc cải tiến và phát triển các giải pháp NFV-MANO hiệu quả hơn, vừa đảm bảo tính kinh tế, vừa đáp ứng được các yêu cầu kỹ thuật và vận hành ngày càng cao. Điều này bao gồm việc giảm chi phí triển khai, cải thiện tính tương thích và dễ dàng tích hợp, nâng cao tính linh hoạt và khả năng mở rộng, cũng như đơn giản hóa quá trình quản lý các VNFs.

1.2 Định hướng giải pháp

Hướng tiếp cận được lựa chọn trong đề tài này là sử dụng nền tảng quản lý tài nguyên container như Kubernetes để cải thiện khả năng mở rộng và linh hoạt của hệ thống NFV-MANO. Lý do chọn hướng này là vì Kubernetes cung cấp khả năng tự động hóa việc triển khai, mở rộng và quản lý các ứng dụng container.

Giải pháp đề xuất trong đề tài bao gồm việc thiết kế và triển khai một hệ thống NFV-MANO với các thành phần chính như bộ quản lý vòng đời dịch vụ, hệ thống thu thập và giám sát hiệu năng hệ thống. Hệ thống được thử nghiệm trên một môi trường mô phỏng để đánh giá hiệu suất và khả năng mở rộng.

Các chức năng chính của NFV-MANO trong phạm vi đề án bao gồm quản lý vòng đời của các VNF, thu thập, giám sát hiệu năng của hệ thống và quản lý cảnh báo lỗi về hệ thống nếu có.

1.3 Bố cục đề án

Phần còn lại của báo cáo đề án tốt nghiệp này được tổ chức như sau.

Chương 2 trình bày về thiết kế hệ thống và đặc tả các usecase trong hệ thống

Trong Chương 3, em giới thiệu về các công nghệ, nền tảng mà em sử dụng trong phạm vi đề án này.

Chương 4 em sẽ trình bày chi tiết về thiết kế cơ sở dữ liệu và các kết quả mà em đã thực hiện được trong phạm vi đề án.

Trong chương 5, em sẽ mô tả các ưu điểm, đóng góp của công nghệ NFV đối với các doanh nghiệp viễn thông cũng như trong đời sống thực tiễn.

Cuối cùng ở chương 6, em sẽ tổng kết lại các kết quả đã đạt được trong phạm vi đề án, đánh giá ưu nhược điểm của hệ thống. Đồng thời đưa ra các hướng định hướng phát triển tiếp theo để cải thiện và phát triển hệ thống NFV-MANO.

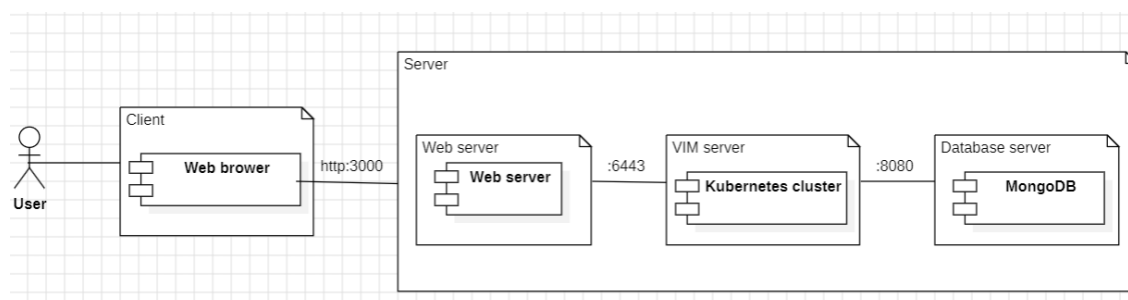
CHƯƠNG 2. PHÂN TÍCH VÀ ĐẶC TẢ CHỨC NĂNG

2.1 Tổng quan chức năng

Quản lý vòng đời của các thực thể VNF (VNF instances). Cụ thể, hệ thống NFV-MANO có các chức năng chính sau:

1. Quản lý vòng đời của VNF (khởi tạo/hủy, bật/tắt, phục hồi khi có sự cố), mở rộng (scale up) hay thu hẹp (scale down) VNF khi cần.
2. Thu thập, giám sát hiệu năng hệ thống
3. Quản lý các cảnh báo lỗi (nếu có).

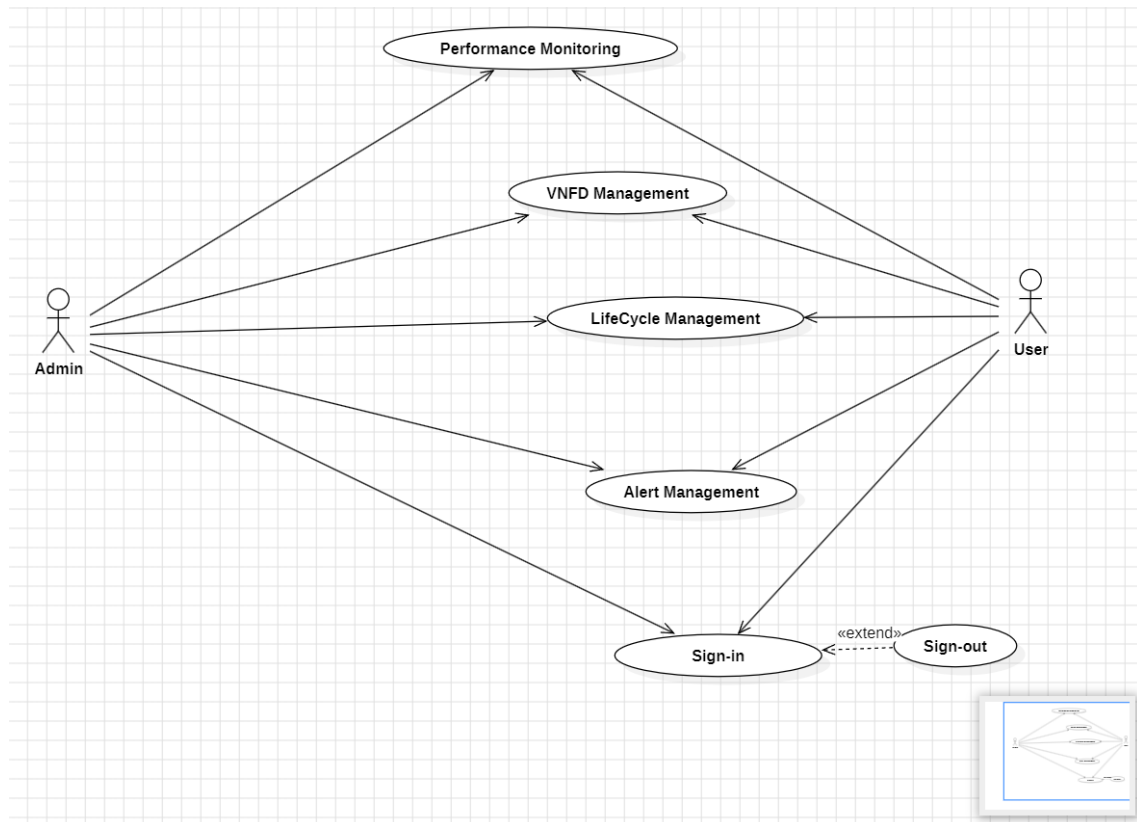
2.1.1 Sơ đồ hệ thống



Hình 2.1: Biểu đồ deployment của hệ thống

Hình 2.1 thể hiện mô hình triển khai của hệ thống. Trong đó hệ thống có 4 khối logic: client - webserver - VIM server - database server. Trong đó Client là các thiết bị người dùng sử dụng như máy tính case, laptop cá nhân, ... Web server là một máy chủ đọc các dữ liệu truy vấn từ database và render ra giao diện cho người dùng, cũng là nơi cập nhật database theo tương tác từ phía client. VIM server là hệ thống quản lý các container. Database server là máy chủ chứa hệ quản trị cơ sở dữ liệu.

2.1.2 Biểu đồ use case tổng quát

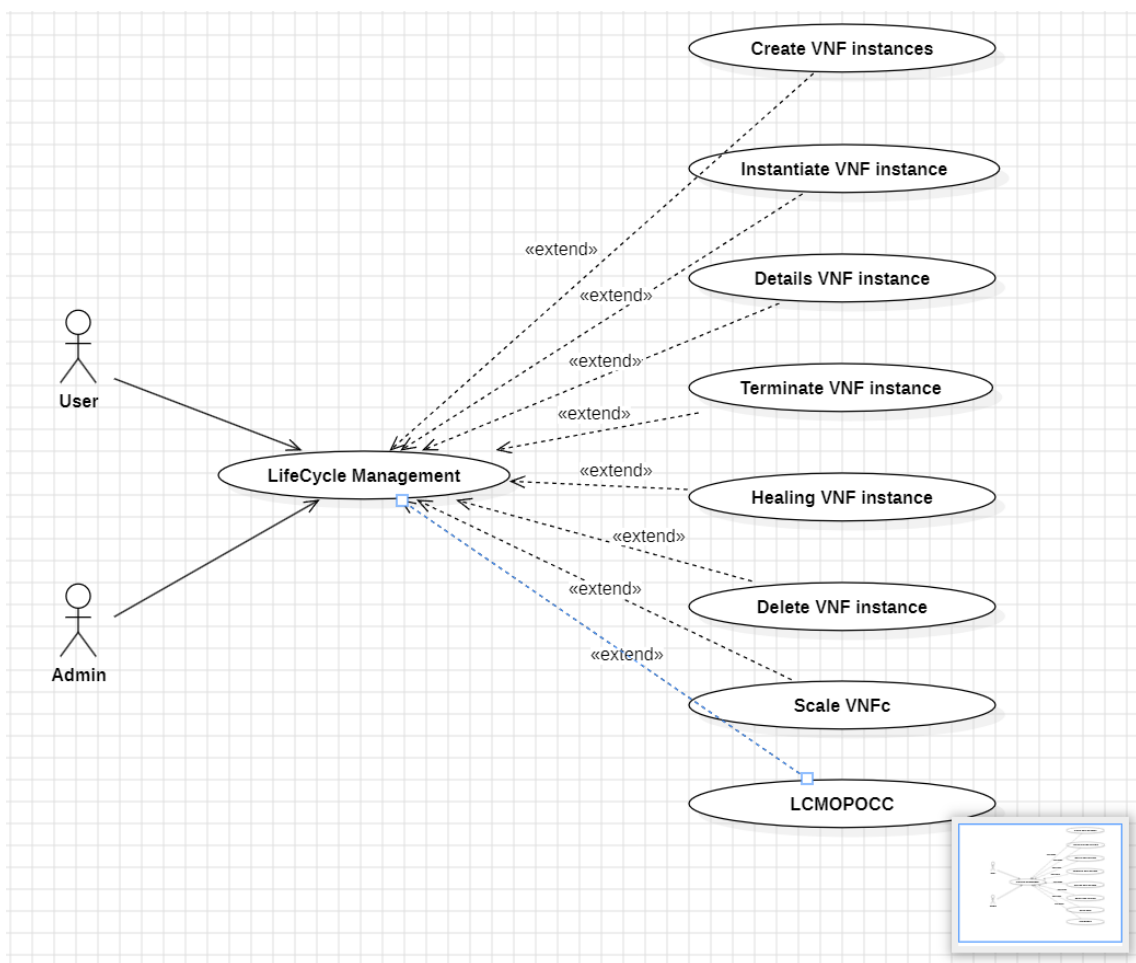


Hình 2.2: Biểu đồ use case tổng quát

Hình 2.2 trình bày biểu đồ use case tổng quát cho hệ thống quản lý các hàm chức năng mạng ảo hóa. Hệ thống cho phép người dùng đăng nhập, đăng xuất, quản lý vòng đời của các VNF instance, thu thập, giám sát hiệu năng và quản lý cảnh báo lỗi của hệ thống.

- Use case Quản lý vòng đời VNF instance(LifeCycle Management): Khởi tạo(Create)/hủy(Delete), bật(Instantiate)/tắt(Terminate), phục hồi khi có sự cố(healing), mở rộng(Scale)
- Use case thu thập, giám sát hiệu năng(Performance Management): Thu thập, giám sát các thông tin về hiệu suất hoạt động của hệ thống
- Use case Quản lý cảnh báo(Alert Management): Thông báo lỗi của hệ thống nếu có
- Use case VNFD Management: Khởi tạo, quản lý các VNFD

2.1.3 Biểu đồ use case phân rã chức năng Quản lý vòng đời VNF

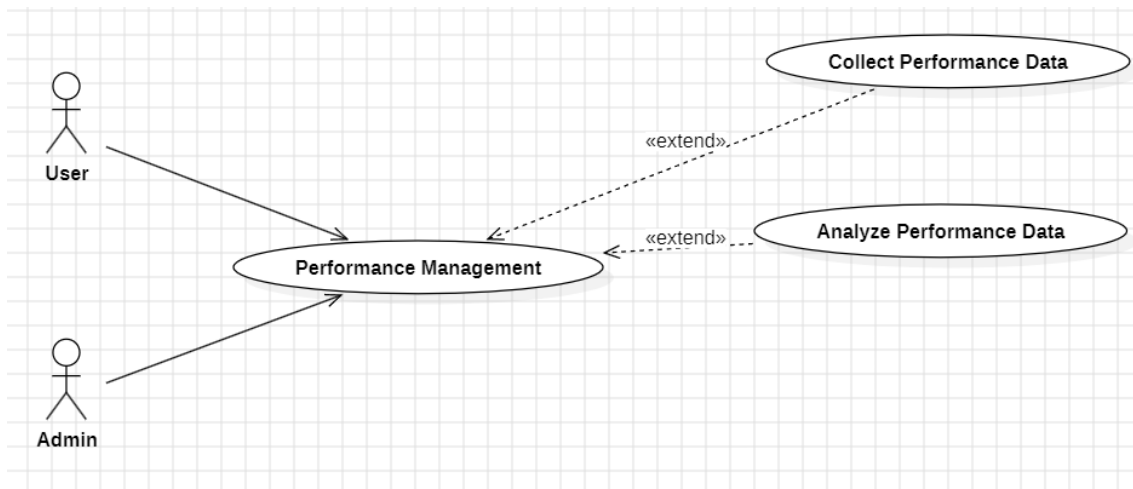


Hình 2.3: Biểu đồ use case phân rã chức năng Quản lý vòng đời VNF

Hình 2.3 trình bày biểu đồ usecase phân rã chức năng Quản lý vòng đời VNF, bao gồm:

- Create VNF instance: Khởi tạo 1 VNF instance
- Instantiate VNF instance: Bật VNF instance
- Details VNF instance: Xem chi tiết các VNFc có trong VNF instance
- Terminate VNF instance: Tắt VNF instance
- Healing VNF instance: khắc phục VNF instance khi có sự cố
- Delete VNF instance: Xóa VNF instance
- Scale VNFc: Mở rộng VNF instance khi cần thiết
- LCMOPOCC: Truy vết, hiển thị thông tin người dùng thao tác với các VNFc có trong VNF instance

2.1.4 Biểu đồ use case phân rã chức năng thu thập, giám sát hiệu năng



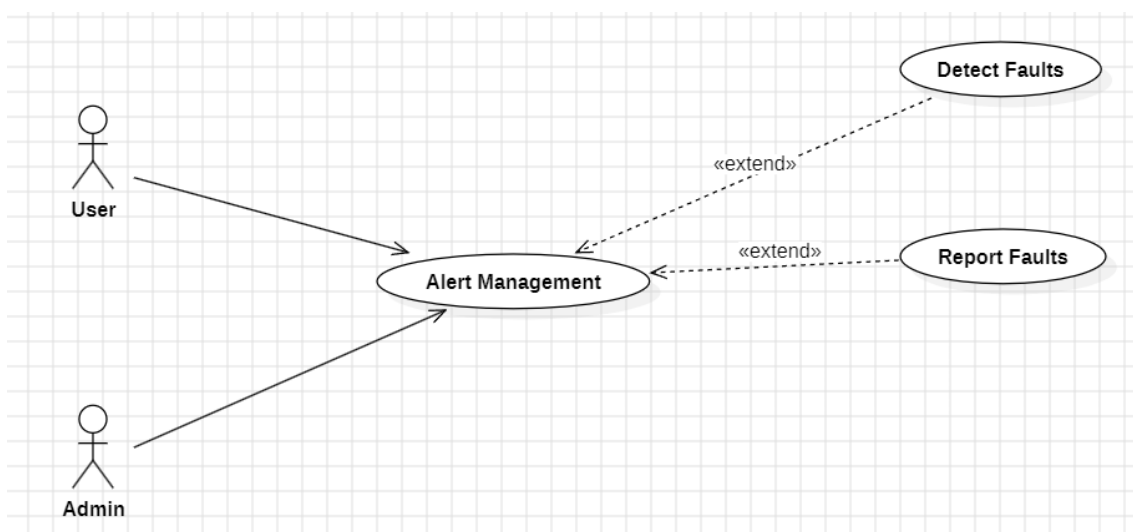
Hình 2.4: Biểu đồ use case phân rã chức năng thu thập, giám sát hiệu năng hệ thống

Hình 2.4 trình bày biểu đồ usecase phân rã chức năng thu thập, giám sát hiệu năng hệ thống, bao gồm:

- Collect Performance Data: Thu thập dữ liệu hiệu năng hệ thống
- Analyze Performance Data: Phân tích, biểu diễn dữ liệu hiệu năng hệ thống

Chức năng này sử dụng công cụ mã nguồn mở Prometheus để có thể thu thập các thông số về hiệu năng của hệ thống và biểu diễn các thông số này thành các biểu đồ trực quan trên hệ thống Grafana.

2.1.5 Biểu đồ use case phân rã chức năng Quản lý cảnh báo



Hình 2.5: Biểu đồ use case phân rã chức năng Quản lý cảnh báo lỗi

Hình 2.5 trình bày biểu đồ usecase phân rã chức năng Quản lý cảnh báo lỗi hệ thống, bao gồm:

- Detect Faults: Phát hiện lỗi
- Report Faults: Thông báo cảnh báo đến người dùng

Chức năng này sẽ giám sát các thông số của hệ thống như cpu, ram, ...Nếu như có node nào trong hệ thống kubernetes đang chạy có vấn đề như cpu quá cao, ram quá thấp, hệ thống sẽ gửi cảnh báo đến với prometheus và gmail của người sử dụng hệ thống, qua đó có thể xử lý vấn đề một cách kịp thời. Ngoài ra, khi có VNF Instance xảy ra vấn đề, ví dụ như các Pod chạy VNF bị lỗi và VNF Instance không hoạt động, hệ thống cũng sẽ gửi cảnh báo đến người sử dụng.

2.2 Đặc tả chức năng

2.2.1 Đặc tả use case Create VNF Instance

Tác nhân	User		
Mô tả	Khởi tạo VNF Instances		
Tiền điều kiện	Đã có VNFD		
Luồng sự kiện chính	STT	Thực hiện bởi	Hành động
	1	User	Chọn chức năng Create VNF Instances
	2	User	Nhập thông tin về tên VNF instance, mô tả về VNF instance, tên VNFD
	3	User	Nhấn nút Create
	4	Hệ thống	Kiểm tra soát lỗi nhập
	5	Hệ thống	Hiển thị kết quả lên màn hình
Luồng sự kiện thay thế	STT	Thực hiện bởi	Hành động
	1	Hệ thống	Thông báo lỗi: chưa nhập đủ thông tin
Hậu điều kiện	Không		

Bảng 2.1: Đặc tả usecase Create VNF instance

2.2.2 Đặc tả use case Instantiate VNF instance

Tác nhân	User		
Mô tả	Bật VNF Instances		
Tiền điều kiện	Đã tạo VNF instance		
Luồng sự kiện chính	STT	Thực hiện bởi	Hành động
	1	User	Chọn VNF instance cần bật
	2	User	Chọn chức năng Instantiate VNF instance
	3	User	Nhấn nút Instantiate
	4	Hệ thống	Tiến hành khởi tạo các tài nguyên kubernetes
	5	Hệ thống	Hiển thị kết quả bật thành công hay thất bại lên màn hình
Luồng sự kiện thay thế	STT	Thực hiện bởi	Hành động
	1	Hệ thống	Không có
Hậu điều kiện	Không		

Bảng 2.2: Đặc tả usecase Instantiate VNF instance

2.2.3 Đặc tả use case Terminate VNF Instance

Tác nhân	User		
Mô tả	Tắt VNF Instances		
Tiền điều kiện	Đã bật VNF instance		
Luồng sự kiện chính	STT	Thực hiện bởi	Hành động
	1	User	Chọn VNF instance cần tắt
	2	User	Chọn chức năng Terminate VNF instance
	3	User	Nhấn nút Terminate
	4	Hệ thống	Tiến hành xóa các tài nguyên kubernetes
	5	Hệ thống	Hiển thị kết quả tắt thành công hay thất bại lên màn hình
Luồng sự kiện thay thế	STT	Thực hiện bởi	Hành động
	1	Hệ thống	Không có
Hậu điều kiện	Không		

Bảng 2.3: Đặc tả usecase Terminate VNF instance

2.2.4 Đặc tả use case Healing VNF Instance

Tác nhân	User		
Mô tả	Phục hồi VNF Instances		
Tiền điều kiện	Đã bật VNF instance		
Luồng sự kiện chính	STT	Thực hiện bởi	Hành động
	1	User	Chọn VNF instance cần healing
	2	User	Chọn chức năng Healing VNF instance
	3	User	Nhấn nút Healing
	4	Hệ thống	Tiến hành xóa các tài nguyên kubernetes trước đó của VNF instance và tạo lại
	5	Hệ thống	Hiển thị kết quả healing thành công hay thất bại lên màn hình
Luồng sự kiện thay thế	STT	Thực hiện bởi	Hành động
	1	Hệ thống	Không có
Hậu điều kiện	Không		

Bảng 2.4: Đặc tả usecase Healing VNF instance

2.2.5 Đặc tả use case Scale VNFC

Tác nhân	User		
Mô tả	Mở rộng số lượng VNFC trong VNF instance		
Tiền điều kiện	Đã bật VNF instance		
Luồng sự kiện chính	STT	Thực hiện bởi	Hành động
	1	User	Chọn VNF instance cần scale
	2	User	Chọn chức năng Scale VNF instance
	3	User	Chọn VNFC và số lượng VNFC cần scale trong VNF instance
	4	User	Nhấn nút Scale
	5	Hệ thống	Tiến hành tạo thêm VNFC trong VNF instance
	6	Hệ thống	Hiển thị kết quả scale thành công hay thất bại lên màn hình
Luồng sự kiện thay thế	STT	Thực hiện bởi	Hành động
	1	Hệ thống	Không có
Hậu điều kiện	Không		

Bảng 2.5: Đặc tả usecase Scale VNFC

2.2.6 Đặc tả use case LCMOPOCC VNF Instance

Tác nhân	User		
Mô tả	Hiển thị các thông tin mà user đã tương tác với VNFC trong VNF instance		
Tiền điều kiện	VNF instance chưa bị xóa		
Luồng sự kiện chính	STT	Thực hiện bởi	Hành động
	1	User	Chọn VNF instance cần xem
	2	User	Chọn chức năng LCMOPOCC
	3	Hệ thống	Hiển thị các thông tin LCMOPOCC của VNF instance lên màn hình
Luồng sự kiện thay thế	STT	Thực hiện bởi	Hành động
	1	Hệ thống	Không có
Hậu điều kiện	Không		

Bảng 2.6: Đặc tả usecase LCMOPOCC VNF instance

2.3 Yêu cầu phi chức năng

- Giao diện dễ dùng, chạy được trên các trình duyệt web phổ biến hiện nay như Google Chrome, Opera,
- Xử lý truy vấn nhanh, chức năng vận hành ổn định
- Hệ thống có thể hoạt động liên tục 24/24

CHƯƠNG 3. CÔNG NGHỆ SỬ DỤNG

3.1 Kubernetes

Kubernetes[1] là một mã nguồn mở được dùng để tự động triển khai hệ thống, scaling, quản lý các container. Nó thực sự là một hệ thống mạnh mẽ, được phát triển bởi Google. Đây là một hệ sinh thái lớn, phát triển nhanh chóng với các dịch vụ, sự hỗ trợ và công cụ có sẵn rất phong phú.

Tên gọi Kubernetes có nguồn gốc từ tiếng Hy Lạp, có ý nghĩa là người lái tàu hoặc hoa tiêu. Google mở mã nguồn Kubernetes từ năm 2014. Kubernetes xây dựng dựa trên một thập kỷ rưỡi kinh nghiệm mà Google có được với việc vận hành một khối lượng lớn workload trong thực tế, kết hợp với các ý tưởng và thực tiễn tốt nhất từ cộng đồng.

Tại sao bạn cần Kubernetes và nó có thể làm những gì?

Các container là một cách tốt để đóng gói và chạy các ứng dụng của bạn. Trong môi trường production, bạn cần quản lý các container chạy các ứng dụng và đảm bảo rằng không có khoảng thời gian downtime. Ví dụ, nếu một container bị tắt đi, một container khác cần phải khởi động lên. Điều này sẽ dễ dàng hơn nếu được xử lý bởi một hệ thống.

Đó là cách Kubernetes đến với chúng ta. Kubernetes cung cấp cho bạn một framework để chạy các hệ phân tán một cách mạnh mẽ. Nó đảm nhiệm việc nhân rộng và chuyển đổi dự phòng cho ứng dụng của bạn, cung cấp các mẫu deployment và hơn thế nữa. Ví dụ, Kubernetes có thể dễ dàng quản lý một triển khai canary cho hệ thống của bạn.

Kubernetes cung cấp cho bạn:

- **Service discovery và cân bằng tải**

Kubernetes có thể expose một container sử dụng DNS hoặc địa chỉ IP của riêng nó. Nếu lượng traffic truy cập đến một container cao, Kubernetes có thể cân bằng tải và phân phối lưu lượng mạng (network traffic) để việc triển khai được ổn định.

- **Tự động rollouts và rollbacks**

Bạn có thể mô tả trạng thái mong muốn cho các container được triển khai dùng Kubernetes và nó có thể thay đổi trạng thái thực tế sang trạng thái mong muốn với tần suất được kiểm soát. Ví dụ, bạn có thể tự động hoá Kubernetes để tạo mới các container cho việc triển khai của bạn, xóa các container hiện

có và áp dụng tất cả các resource của chúng vào container mới.

- **Đóng gói tự động**

Bạn cung cấp cho Kubernetes một cluster gồm các node mà nó có thể sử dụng để chạy các tác vụ được đóng gói (containerized task). Bạn cho Kubernetes biết mỗi container cần bao nhiêu CPU và bộ nhớ (RAM).

- **Tự phục hồi**

Kubernetes khởi động lại các containers bị lỗi, thay thế các container, xóa các container không phản hồi lại cấu hình health check do người dùng xác định và không cho các client biết đến chúng cho đến khi chúng sẵn sàng hoạt động.

- **Quản lý cấu hình và bảo mật**

Kubernetes cho phép bạn lưu trữ và quản lý các thông tin nhạy cảm như: password, OAuth token và SSH key. Bạn có thể triển khai và cập nhật lại secret và cấu hình ứng dụng mà không cần build lại các container image và không để lộ secret trong cấu hình stack của bạn

3.2 Prometheus

Prometheus[2] là bộ công cụ cảnh báo và giám sát hệ thống nguồn mở ban đầu được xây dựng tại SoundCloud. Kể từ khi thành lập vào năm 2012, nhiều công ty và tổ chức đã áp dụng Prometheus và dự án có cộng đồng người dùng và nhà phát triển rất tích cực. Nó hiện là một dự án nguồn mở độc lập và được duy trì độc lập với bất kỳ công ty nào. Để nhấn mạnh điều này và làm rõ cơ cấu quản trị của dự án, Prometheus đã gia nhập Cloud Native Computing Foundation vào năm 2016 với tư cách là dự án được tổ chức thứ hai, sau Kubernetes.

Prometheus thu thập và lưu trữ số liệu của mình dưới dạng dữ liệu chuỗi thời gian, tức là thông tin số liệu được lưu trữ cùng với dấu thời gian mà thông tin đó được ghi lại, cùng với các cặp khóa-giá trị tùy chọn được gọi là nhãn.

a, Các Tính Năng Chính của Prometheus

1. **Thu Thập Dữ Liệu:** Prometheus thu thập dữ liệu bằng cách quét (scraping) các điểm cuối (endpoints) được phơi bày bởi các dịch vụ. Dữ liệu này thường được thu thập dưới dạng chỉ số (metrics) và lưu trữ trong cơ sở dữ liệu chuỗi thời gian.
2. **Ngôn Ngữ Truy Vấn PromQL:** Prometheus sử dụng PromQL (Prometheus Query Language) để truy vấn và phân tích dữ liệu. PromQL là một ngôn ngữ mạnh mẽ cho phép người dùng thực hiện các phép tính phức tạp trên các chuỗi thời gian.

3. **Cảnh Báo (Alerting):** Prometheus có hệ thống cảnh báo tích hợp, cho phép người dùng thiết lập các quy tắc cảnh báo dựa trên các chỉ số và điều kiện nhất định. Khi các điều kiện này được thỏa mãn, hệ thống cảnh báo sẽ gửi thông báo qua các kênh khác nhau như email, Slack, hoặc PagerDuty.
4. **Khả Năng Mở Rộng:** Prometheus hỗ trợ các tính năng mở rộng thông qua việc tích hợp với các công cụ và dịch vụ khác. Nó có thể tích hợp với các hệ thống như Kubernetes, Docker, và nhiều công cụ giám sát khác.
5. **Khả Năng Tự Chữa Lành:** Prometheus được thiết kế để có thể tự phục hồi sau các sự cố. Nó có thể chạy trong môi trường không đáng tin cậy và có khả năng tái khởi động mà không mất dữ liệu.
6. **Bộ Nhớ Tạm Thời:** Prometheus lưu trữ dữ liệu trên đĩa cục bộ, giúp giảm thiểu sự phụ thuộc vào các dịch vụ bên ngoài và cải thiện hiệu suất truy vấn.

b, Ứng Dụng Thực Tế

- **Giám Sát Hệ Thống và Ứng Dụng:** Prometheus thường được sử dụng để giám sát hiệu suất của các máy chủ, ứng dụng, và dịch vụ. Các chỉ số như CPU, bộ nhớ, lưu lượng mạng, và hiệu suất ứng dụng đều có thể được thu thập và phân tích.
- **Quản Lý Cơ Sở Hạ Tầng:** Trong các môi trường đám mây và container, Prometheus có thể giám sát và quản lý các tài nguyên, đảm bảo rằng hệ thống hoạt động hiệu quả và ổn định.
- **Phân Tích Hiệu Suất:** Các nhà phát triển và quản trị viên hệ thống sử dụng Prometheus để phân tích hiệu suất và xác định các điểm nghẽn trong hệ thống, từ đó tối ưu hóa các ứng dụng và dịch vụ.

c, Lợi Ích

- **Giám Sát Chủ Động:** Với khả năng cảnh báo mạnh mẽ, Prometheus giúp người dùng phát hiện và xử lý các vấn đề trước khi chúng ảnh hưởng đến người dùng cuối.
- **Tích Hợp Linh Hoạt:** Prometheus có thể tích hợp với nhiều công cụ và dịch vụ khác, từ đó cung cấp một hệ sinh thái giám sát toàn diện.
- **Dễ Dàng Mở Rộng:** Khả năng mở rộng của Prometheus cho phép người dùng tăng cường khả năng giám sát khi hệ thống của họ phát triển.
- **Hiệu Quả Chi Phí:** Là một công cụ mã nguồn mở, Prometheus giúp giảm chi phí cho việc giám sát và cảnh báo, đồng thời cung cấp các tính năng mạnh mẽ tương đương với các giải pháp thương mại.

3.3 Grafana

Grafana[3] là một nền tảng mã nguồn mở dùng để giám sát và phân tích dữ liệu. Nó cung cấp các công cụ để trực quan hóa dữ liệu, theo dõi các chỉ số, và nhận thông báo về các sự kiện quan trọng. Grafana thường được sử dụng để theo dõi hiệu suất hệ thống, ứng dụng, và các dịch vụ khác.

a, Các Tính Năng Chính của Grafana

1. **Trực Quan Hóa Dữ Liệu:** Grafana hỗ trợ nhiều loại biểu đồ và đồ thị khác nhau như biểu đồ đường, biểu đồ cột, bản đồ nhiệt (heatmaps), và nhiều loại biểu đồ tùy chỉnh khác. Người dùng có thể dễ dàng tạo các bảng điều khiển (dashboard) để hiển thị dữ liệu từ nhiều nguồn khác nhau.
2. **Hỗ Trợ Nhiều Nguồn Dữ Liệu:** Grafana hỗ trợ nhiều loại cơ sở dữ liệu khác nhau như Prometheus, Graphite, Elasticsearch, InfluxDB, MySQL, PostgreSQL, và nhiều nguồn dữ liệu khác. Điều này cho phép người dùng kết hợp và phân tích dữ liệu từ nhiều nguồn trong một giao diện duy nhất.
3. **Cảnh Báo (Alerting):** Grafana cung cấp các công cụ để thiết lập cảnh báo dựa trên các ngưỡng (threshold) và điều kiện nhất định. Khi các điều kiện này được thỏa mãn, hệ thống có thể gửi thông báo qua email, Slack, PagerDuty, và nhiều kênh khác.
4. **Khả Năng Mở Rộng (Plugins):** Grafana có một hệ thống plugin mạnh mẽ cho phép người dùng mở rộng tính năng của nó. Các plugin có thể thêm nguồn dữ liệu mới, các loại hình trực quan hóa mới, hoặc các tích hợp với các công cụ và dịch vụ khác.
5. **Xác Thực và Quản Lý Người Dùng:** Grafana hỗ trợ nhiều phương thức xác thực và quản lý người dùng, bao gồm LDAP, OAuth, và các hệ thống xác thực khác. Điều này giúp bảo mật các bảng điều khiển và dữ liệu quan trọng.

b, Ứng Dụng Thực Tế

Giám Sát Hệ Thống: Các quản trị viên hệ thống sử dụng Grafana để theo dõi hiệu suất của các máy chủ, mạng, và các dịch vụ khác. Bằng cách thiết lập các bảng điều khiển, họ có thể nhanh chóng phát hiện và xử lý các vấn đề.

Giám Sát Ứng Dụng: Các nhà phát triển sử dụng Grafana để theo dõi hiệu suất và sức khỏe của các ứng dụng, đảm bảo chúng hoạt động ổn định và hiệu quả.

c, Lợi Ích

- **Tăng Cường Hiệu Quả:** Bằng cách cung cấp một nền tảng trực quan và dễ sử dụng, Grafana giúp người dùng nhanh chóng hiểu và hành động dựa trên

dữ liệu của họ.

- **Giảm Thời Gian Phản Ứng:** Với các công cụ cảnh báo mạnh mẽ, người dùng có thể nhận thông báo ngay lập tức khi có vấn đề phát sinh, giúp giảm thời gian phản ứng và khắc phục sự cố.

3.4 Framework Quarkus

Quarkus[4] là một framework Java hiện đại được thiết kế đặc biệt cho môi trường điện toán đám mây và container. Được phát triển bởi Red Hat, Quarkus nhằm mục tiêu tối ưu hóa hiệu suất, kích thước bộ nhớ và tốc độ khởi động của các ứng dụng Java, giúp chúng hoạt động hiệu quả hơn trong các môi trường đám mây và serverless.

a, Các Tính Năng Chính của Quarkus

1. **Hiệu Suất Cao:** Quarkus được thiết kế để khởi động nhanh chóng và sử dụng ít bộ nhớ hơn so với các framework Java truyền thống. Điều này làm cho Quarkus trở nên lý tưởng cho các ứng dụng serverless và microservices.
2. **Developer Joy:** Quarkus cung cấp trải nghiệm phát triển tuyệt vời với các tính năng như reload nhanh (live reload), phát hiện lỗi khi code (hot deployment), và các công cụ hỗ trợ phát triển mạnh mẽ. Điều này giúp tăng năng suất và giảm thời gian phát triển.
3. **Hỗ Trợ Nhiều Tiêu Chuẩn và API:** Quarkus hỗ trợ nhiều tiêu chuẩn Java và API như JAX-RS, CDI, JPA, Hibernate, và nhiều công nghệ khác. Điều này giúp các nhà phát triển dễ dàng chuyển đổi và tích hợp với các ứng dụng hiện có.
4. **Tích Hợp GraalVM:** Quarkus tích hợp chặt chẽ với GraalVM để biên dịch các ứng dụng Java xuống mã máy (native binaries). Điều này không chỉ cải thiện tốc độ khởi động mà còn giảm thiểu sử dụng bộ nhớ.
5. **Cloud-Native và Kubernetes-Native:** Quarkus được thiết kế để hoạt động tốt với các công nghệ đám mây như Kubernetes và OpenShift. Nó cung cấp các công cụ và tích hợp để triển khai, giám sát, và quản lý ứng dụng trong môi trường đám mây.

b, Ứng Dụng Thực Tế

- **Microservices:** Quarkus rất phù hợp cho việc phát triển các ứng dụng microservices nhờ vào khả năng khởi động nhanh và sử dụng ít bộ nhớ.
- **Serverless Computing:** Với khả năng khởi động nhanh và hiệu suất cao, Quarkus là lựa chọn lý tưởng cho các ứng dụng serverless, nơi thời gian phản

hồi nhanh là rất quan trọng.

c, Lợi Ích

- **Tối Ưu Hóa Chi Phí:** Việc sử dụng ít bộ nhớ và khởi động nhanh giúp giảm chi phí hạ tầng, đặc biệt là trong các môi trường đám mây nơi tài nguyên được sử dụng tính theo giờ hoặc theo phút.
- **Nâng Cao Hiệu Suất:** Các ứng dụng Quarkus có thể xử lý tải công việc nặng với hiệu suất cao và thời gian phản hồi nhanh, cải thiện trải nghiệm người dùng cuối.
- **Dễ Dàng Triển Khai:** Với khả năng tích hợp chặt chẽ với Kubernetes và các nền tảng đám mây, Quarkus giúp đơn giản hóa quy trình triển khai và quản lý ứng dụng.
- **Cộng Đồng Hỗ Trợ Mạnh Mẽ:** Quarkus có một cộng đồng phát triển sôi động và được hỗ trợ bởi Red Hat, đảm bảo rằng các nhà phát triển có thể tiếp cận tài liệu, hỗ trợ, và các bản cập nhật thường xuyên.

3.5 TypeScript

TypeScript[5] là một ngôn ngữ lập trình mã nguồn mở được phát triển bởi Microsoft. TypeScript là một phần mở rộng của JavaScript, bổ sung thêm các tính năng như kiểu tĩnh và các công cụ hỗ trợ phát triển mạnh mẽ, giúp lập trình viên viết mã dễ bảo trì và ít lỗi hơn.

a, Các Tính Năng Chính của TypeScript

1. **Kiểu Tĩnh (Static Typing):** TypeScript cho phép định nghĩa kiểu cho các biến, hàm, và đối tượng. Điều này giúp phát hiện lỗi trong quá trình viết mã thay vì đợi đến khi chạy chương trình mới phát hiện.
2. **Hỗ Trợ ECMAScript Mới Nhất:** TypeScript hỗ trợ tất cả các tính năng mới nhất của ECMAScript, bao gồm các tính năng của ES6/ES7 và các phiên bản sau này. Điều này giúp các lập trình viên sử dụng các tính năng hiện đại trong JavaScript.
3. **Khả Năng Tương Thích Ngược:** Mã TypeScript có thể được biên dịch xuống JavaScript thuần túy (plain JavaScript), đảm bảo rằng nó có thể chạy trên bất kỳ môi trường nào hỗ trợ JavaScript.
4. **Công Cụ Phát Triển Mạnh Mẽ:** TypeScript đi kèm với các công cụ phát triển mạnh mẽ như tự động hoàn thành mã, kiểm tra lỗi theo thời gian thực, và khả năng tái cấu trúc mã dễ dàng. Các công cụ này giúp tăng năng suất lập trình viên và cải thiện chất lượng mã.

5. **Hỗ Trợ Module:** TypeScript hỗ trợ hệ thống module, cho phép tổ chức mã theo các đơn vị độc lập và dễ quản lý. Điều này giúp phát triển và bảo trì các ứng dụng lớn dễ dàng hơn.

b, Ứng Dụng Thực Tế

- **Phát Triển Web:** TypeScript thường được sử dụng trong các dự án phát triển web, đặc biệt là với các framework hiện đại như Angular, React, và Vue.js. Việc sử dụng TypeScript giúp giảm thiểu lỗi và cải thiện khả năng bảo trì mã.
- **Phát Triển Backend:** TypeScript cũng được sử dụng trong phát triển backend với các framework như Node.js, NestJS. Nó cung cấp các công cụ mạnh mẽ để xây dựng các ứng dụng server-side hiệu quả và an toàn.
- **Ứng Dụng Di Động:** Với sự hỗ trợ từ các framework như React Native, TypeScript có thể được sử dụng để phát triển ứng dụng di động, mang lại lợi ích của kiểu tĩnh cho các dự án di động.

c, Lợi Ích

- **Phát Hiện Lỗi Sớm:** Với kiểu tĩnh, các lỗi thường gặp trong JavaScript có thể được phát hiện ngay khi viết mã, giúp giảm thiểu lỗi runtime và tăng chất lượng mã.
- **Tăng Năng Suất:** Các công cụ phát triển của TypeScript như tự động hoàn thành mã và kiểm tra lỗi theo thời gian thực giúp lập trình viên làm việc hiệu quả hơn.
- **Cải Thiện Bảo Trì:** TypeScript giúp mã dễ đọc và dễ bảo trì hơn, đặc biệt là trong các dự án lớn với nhiều lập trình viên tham gia.
- **Hỗ Trợ Tốt Hơn cho Các Thư Viện và Framework:** TypeScript có khả năng tương thích tốt với nhiều thư viện và framework phổ biến, mang lại lợi ích của kiểu tĩnh cho các dự án sử dụng các công cụ này.

3.6 Fabric8 Kubernetes Java Client

Fabric8 Kubernetes Java Client[6] là một thư viện Java mã nguồn mở, cung cấp một cách thức tiện lợi để tương tác với Kubernetes API từ ứng dụng Java. Đây là một phần của dự án Fabric8, một bộ công cụ và nền tảng dành cho việc xây dựng, triển khai và quản lý các ứng dụng container.

Dưới đây là một số điểm nổi bật về thư viện Fabric8 Kubernetes Java Client:

1. **Tích hợp dễ dàng:** Thư viện này cung cấp một API Java để tương tác với Kubernetes một cách trực tiếp, giúp lập trình viên Java có thể quản lý các tài nguyên Kubernetes như Pods, Services, Deployments, ConfigMaps, và nhiều

tài nguyên khác một cách dễ dàng.

2. **Hỗ trợ đa phiên bản:** Fabric8 Kubernetes Client hỗ trợ nhiều phiên bản của Kubernetes API, giúp đảm bảo tính tương thích với các phiên bản Kubernetes khác nhau.
3. **CRUD Operations:** Thư viện này hỗ trợ đầy đủ các thao tác CRUD (Create, Read, Update, Delete) cho hầu hết các đối tượng của Kubernetes.
4. **DSL (Domain Specific Language):** Cung cấp một DSL mạnh mẽ để tạo ra các đối tượng Kubernetes một cách dễ dàng và rõ ràng.
5. **Tích hợp tốt với Spring Boot:** Thư viện này có thể dễ dàng tích hợp với các ứng dụng Spring Boot, giúp phát triển các microservices dễ dàng hơn.
6. **Hỗ trợ Asynchronous và Watchers:** Fabric8 Kubernetes Client hỗ trợ các tác vụ bất đồng bộ (asynchronous) và cơ chế theo dõi (watchers) để nhận thông báo khi có sự thay đổi trong các tài nguyên Kubernetes.
7. **Quản lý cấu hình:** Hỗ trợ quản lý cấu hình ứng dụng thông qua ConfigMaps và Secrets của Kubernetes.
8. **Hỗ trợ Kubernetes Custom Resources:** Fabric8 Kubernetes Client hỗ trợ tạo và quản lý các tài nguyên tùy chỉnh (Custom Resources) của Kubernetes, giúp mở rộng khả năng của Kubernetes.

3.7 SnakeYAML

SnakeYAML[7] là một thư viện Java mã nguồn mở, dùng để phân tích cú pháp (parse) và tạo các tệp YAML. YAML (YAML Ain't Markup Language) là một ngôn ngữ đánh dấu dạng dữ liệu tuần tự, thường được sử dụng cho các tệp cấu hình và truyền dữ liệu giữa các ứng dụng.

Dưới đây là một số điểm nổi bật về SnakeYAML:

1. **Phân tích cú pháp YAML:** SnakeYAML có thể phân tích cú pháp các tệp YAML, chuyển đổi chúng thành các đối tượng Java tương ứng.
2. **Tạo tệp YAML:** Thư viện này cũng hỗ trợ tạo các tệp YAML từ các đối tượng Java, giúp dễ dàng viết dữ liệu vào các tệp YAML.
3. **Hỗ trợ đầy đủ YAML 1.1:** SnakeYAML hỗ trợ đầy đủ các tính năng của YAML 1.1, bao gồm các kiểu dữ liệu phức tạp và các anchor/references.
4. **Dễ sử dụng:** Thư viện này rất dễ sử dụng và tích hợp vào các dự án Java. Nó cung cấp API đơn giản và rõ ràng.
5. **Tùy chỉnh:** SnakeYAML cho phép tùy chỉnh cách mà các tệp YAML được

phân tích và tạo ra, giúp phù hợp với nhiều trường hợp sử dụng khác nhau.

6. **Tương thích với các frameworks:** SnakeYAML có thể được sử dụng với nhiều framework và thư viện Java khác nhau, chẳng hạn như Spring Framework, để quản lý cấu hình ứng dụng.

CHƯƠNG 4. THIẾT KẾ, TRIỂN KHAI VÀ ĐÁNH GIÁ HỆ THỐNG

4.1 Thiết kế kiến trúc

4.1.1 Lựa chọn kiến trúc phần mềm

Kiến trúc MVC (Model-View-Controller)[8] là một trong những kiến trúc phổ biến và lâu đời nhất trong phát triển phần mềm, đặc biệt là trong các ứng dụng web. MVC chia ứng dụng thành ba thành phần chính: Model, View và Controller. Điều này giúp phân tách rõ ràng các thành phần của ứng dụng, từ đó tăng cường khả năng quản lý, bảo trì và mở rộng hệ thống.

Thành Phần Của Kiến Trúc MVC

1. Model:

- **Vai trò:** Model đại diện cho dữ liệu và logic nghiệp vụ của ứng dụng. Nó chịu trách nhiệm quản lý trạng thái của ứng dụng, cập nhật dữ liệu, và thực hiện các tính toán hoặc xử lý dữ liệu.
- **Tính năng:** Kết nối với cơ sở dữ liệu, thao tác và xử lý dữ liệu, lưu trữ trạng thái của ứng dụng.

2. View:

- **Vai trò:** View chịu trách nhiệm về giao diện người dùng (UI). Nó hiển thị dữ liệu từ Model cho người dùng và gửi các hành động của người dùng đến Controller.
- **Tính năng:** Tạo và cập nhật giao diện người dùng, hiển thị thông tin từ Model, không chứa logic xử lý nghiệp vụ.

3. Controller:

- **Vai trò:** Controller đóng vai trò cầu nối giữa View và Model. Nó nhận các yêu cầu từ người dùng thông qua View, xử lý yêu cầu và cập nhật Model tương ứng.
- **Tính năng:** Xử lý các yêu cầu và hành động từ người dùng, xác định logic điều hướng.

Ưu Điểm Của Kiến Trúc MVC

- **Phân Tách Rõ Ràng:** MVC tách biệt rõ ràng các thành phần giao diện người dùng, logic nghiệp vụ và dữ liệu, giúp dễ dàng quản lý và bảo trì ứng dụng.
- **Tái Sử Dụng:** Các thành phần trong MVC có thể được tái sử dụng trong nhiều

phần của ứng dụng hoặc trong các ứng dụng khác.

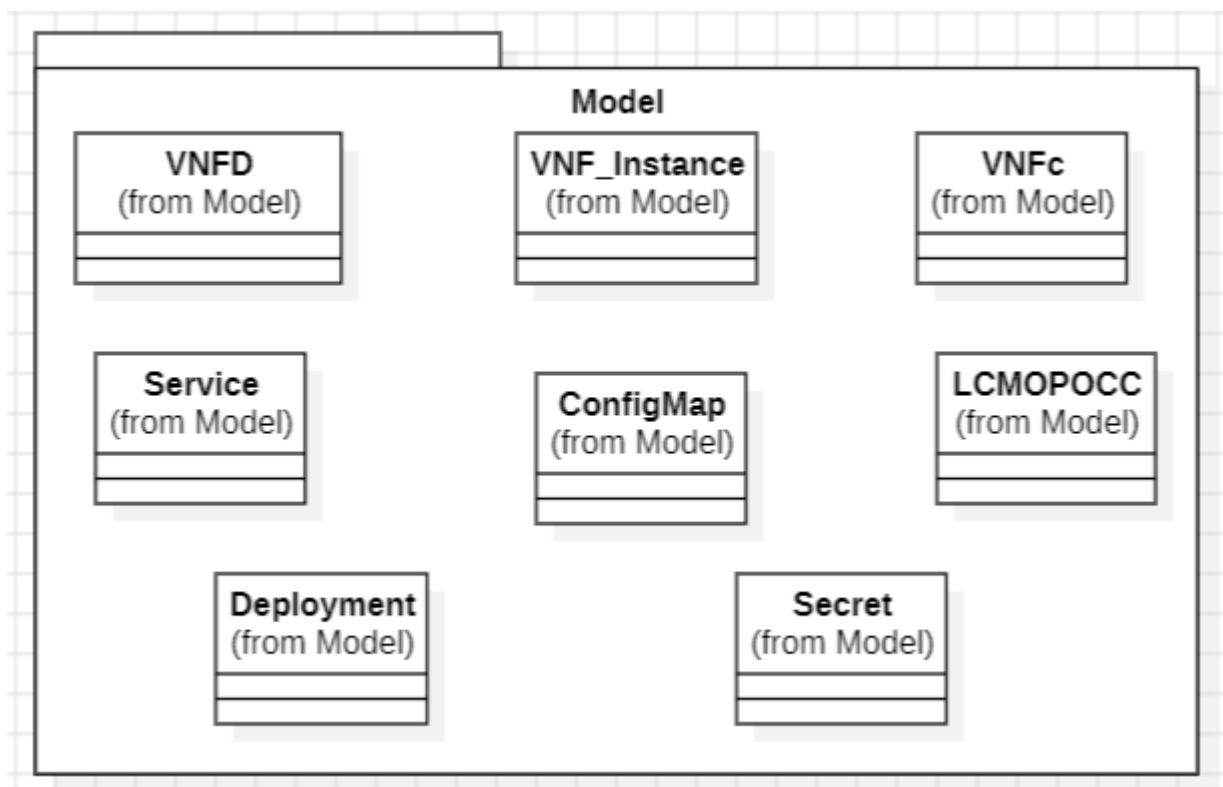
- **Dễ Mở Rộng:** Do có sự phân tách rõ ràng, việc mở rộng ứng dụng trở nên dễ dàng hơn mà không ảnh hưởng đến toàn bộ hệ thống.
- **Phát Triển Song Song:** Các nhóm phát triển có thể làm việc đồng thời trên các thành phần khác nhau (Model, View, Controller) mà ít bị ảnh hưởng lẫn nhau.

Tổng Kết

Kiến trúc MVC là một lựa chọn phổ biến và hiệu quả cho việc phát triển các ứng dụng có yêu cầu về khả năng mở rộng, quản lý và bảo trì cao. Với sự phân tách rõ ràng giữa các thành phần Model, View và Controller, MVC giúp tăng cường khả năng quản lý mã nguồn, đồng thời hỗ trợ phát triển và mở rộng hệ thống một cách linh hoạt và hiệu quả.

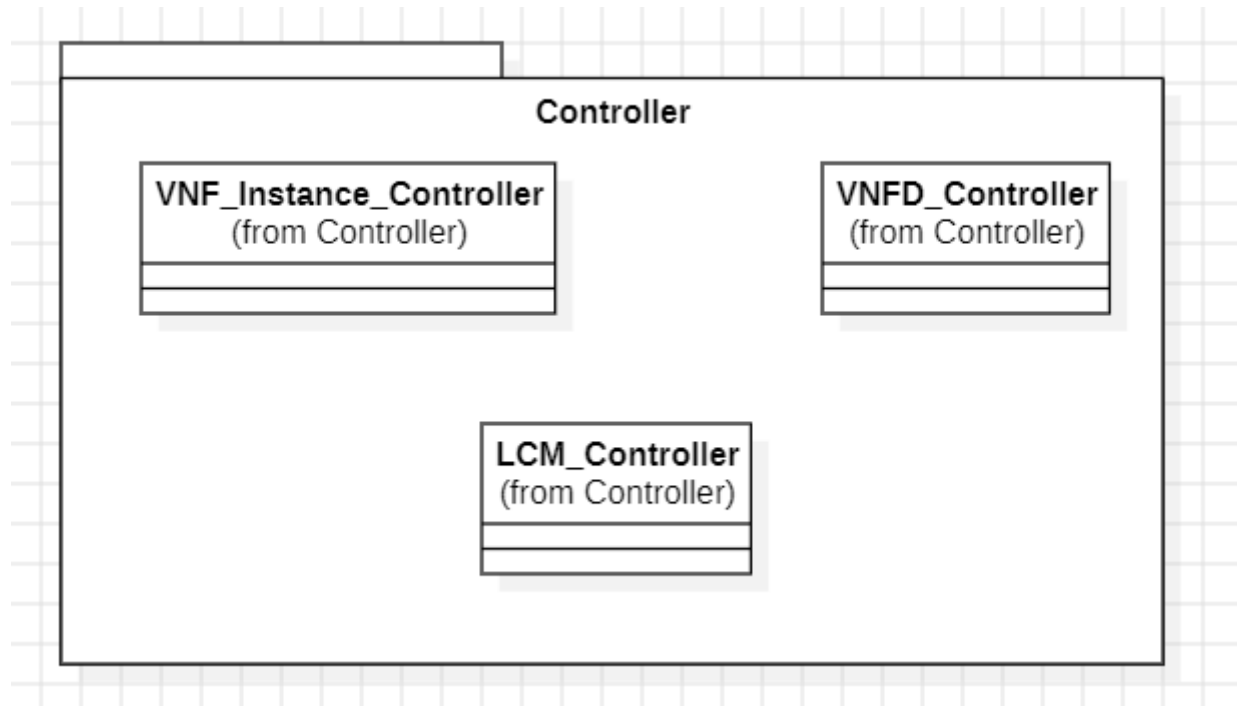
4.1.2 Thiết kế chi tiết gói

Gói Model



Hình 4.1: Biểu đồ chi tiết gói Model

Gói Controller

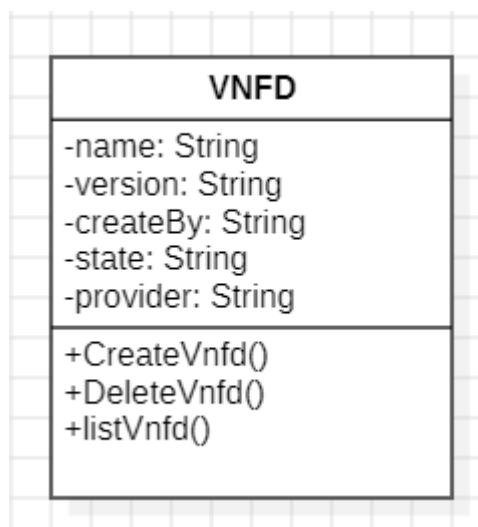


Hình 4.2: Biểu đồ chi tiết gói Controller

4.2 Thiết kế chi tiết

4.2.1 Thiết kế lớp

Lớp VNFD

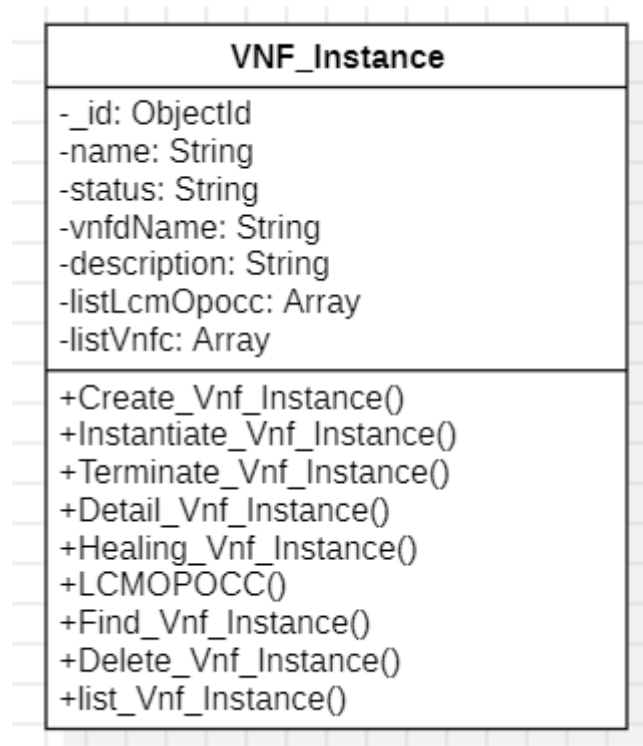


Hình 4.3: Thiết kế chi tiết lớp VNFD

STT	Tên phương thức	Diễn giải
1	CreateVnfd	Tạo VNFD
2	DeleteVnfd	Xóa VNFD
3	ListVnfd	Liệt kê VNFD

Bảng 4.1: Đặc tả lớp VNFD

Lớp VNF Instance

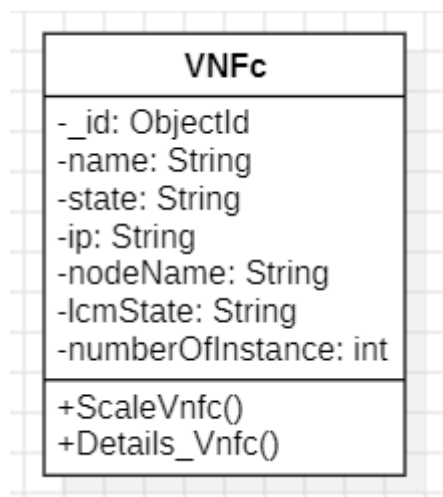


Hình 4.4: Thiết kế chi tiết lớp VNF Instance

STT	Tên phương thức	Diễn giải
1	CreateVnfInstance	Tạo VNF Instance
2	InstantiateVnfInstance	Bật VNF Instance
3	TerminateVnfInstance	Tắt VNF Instance
4	DetailVnfInstance	Xem chi tiết VNF Instance
5	HealingVnfInstance	Phục hồi VNF Instance
6	LCMOPOCC	Xem chi tiết thông tin người dùng tương tác với VNF Instance
7	FindVnfInstance	Tìm kiếm VNF Instance
8	DeleteVnfInstance	Xóa VNF Instance
9	listVnfInstance	Liệt kê VNF Instance

Bảng 4.2: Đặc tả lớp VNF Instance

Lớp VNFC



Hình 4.5: Thiết kế chi tiết lớp VNFC

STT	Tên phương thức	Diễn giải
1	ScaleVnfc	Mở rộng các VNFC
2	DetailVnfc	Xem chi tiết các VNFC

Bảng 4.3: Đặc tả lớp VNFD

4.2.2 Thiết kế cơ sở dữ liệu

Đặc tả chi tiết cơ sở dữ liệu

VNF-Instance

STT	Tên trường	Kiểu dữ liệu	Diễn giải
1	id	String	Id của VNF Instance
2	name	String	Tên của VNF Instance
3	status	String	Trạng thái của VNF Instance (Started hoặc Stopped)
4	description	String	Mô tả về VNF Instance
5	vnfdName	String	Tên của VNFD mà VNF Instance triển khai
6	listVnfc	Array	Tập hợp các VNFC có trong VNF Instance
7	listLcmOpocc	Array	Tập hợp các LcmOpOcc có trong VNF Instance

Bảng 4.4: Đặc tả cơ sở dữ liệu VNF instance

VNFC

STT	Tên trường	Kiểu dữ liệu	Diễn giải
1	id	String	Id của VNFC
2	name	String	Tên của VNFC
3	nodeName	String	Tên của Node trong kubernetes mà VNFC này đang chạy
4	ip	String	Địa chỉ ip của VNFC
5	numberOfInstance	int	Số instance của VNFC
6	state	String	Trạng thái của VNFC (Running, Pending, Terminating)

Bảng 4.5: Đặc tả cơ sở dữ liệu VNFC

VNFD

STT	Tên trường	Kiểu dữ liệu	Diễn giải
1	id	String	Id của VNFD
1	name	String	Tên của VNFD
1	version	String	Phiên bản của VNFD
1	createBy	String	Tên người tạo VNFD
1	provider	String	Tên doanh nghiệp, nhà cung cấp VNFD
1	vdus	Array	Tập hợp các vdu có trong VNFD

Bảng 4.6: Đặc tả cơ sở dữ liệu VNFD

LCMOPOCC

STT	Tên trường	Kiểu dữ liệu	Diễn giải
1	AffectVNFC	String	Tên của VNFC mà hệ thống tương tác
2	Operation	String	Tên của hoạt động mà hệ thống tương tác với VNF Instance
3	Status	String	Tên của hoạt động mà hệ thống tương tác với VNFC(ADDED, REMOVED)
4	OperationStatus	String	Trạng thái của tương tác(COMPLETED/FAILED)
5	StartTime	Time	Thời gian thực hiện tương tác
6	vdus	Array	Tập hợp các vdu có trong VNFD

Bảng 4.7: Đặc tả cơ sở dữ liệu LCMOPOCC

Deployment

STT	Tên trường	Kiểu dữ liệu	Diễn giải
1	apiVersion	String	Tên version của tài nguyên
2	kind	String	Loại tài nguyên trong kubernetes (Deployment)
3	name	String	Tên của Deployment
4	label	String	Nhãn của Deployment
5	replicas	int	Số lượng instance của Deployment ban đầu
6	container	Container	Mô tả về container chạy Deployment

Bảng 4.8: Đặc tả cơ sở dữ liệu Deployment

Service

STT	Tên trường	Kiểu dữ liệu	Diễn giải
1	apiVersion	String	Tên version của tài nguyên
2	kind	String	Loại tài nguyên trong kubernetes (Service)
3	name	String	Tên của Service
4	port	Port	Mô tả về cổng của Service

Bảng 4.9: Đặc tả cơ sở dữ liệu Service

ConfigMap

STT	Tên trường	Kiểu dữ liệu	Diễn giải
1	apiVersion	String	Tên version của tài nguyên
2	kind	String	Loại tài nguyên trong kubernetes (ConfigMap)
3	name	String	Tên của ConfigMap

Bảng 4.10: Đặc tả cơ sở dữ liệu VNF Configmap

Secret

STT	Tên trường	Kiểu dữ liệu	Diễn giải
1	apiVersion	String	Tên version của tài nguyên
2	kind	String	Loại tài nguyên trong kubernetes (Secret)
3	name	String	Tên của Secret

Bảng 4.11: Đặc tả cơ sở dữ liệu Secret

Container

STT	Tên trường	Kiểu dữ liệu	Diễn giải
1	name	String	Tên của container chạy Deployment
2	image	String	Tên của image mà container chạy
3	containerPort	String	Cổng mà container chạy

Bảng 4.12: Đặc tả cơ sở dữ liệu Container

4.3 Xây dựng ứng dụng

4.3.1 Thư viện và công cụ sử dụng

Mục đích	Công cụ	Địa chỉ URL
IDE lập trình	IntelliJ IDEA Educational	https://www.jetbrains.com/idea/
IDE lập trình	Visual Studio Code	https://code.visualstudio.com/updates/v1
Nền tảng quản lý container	Kubernetes v1.31.2	https://kubernetes.io/
Nền tảng giám sát hệ thống	Prometheus 2.52.0	https://github.com/prometheus/prometheus/releases/download/v2.52.0/prometheus-2.52.0.windows-amd64.zip
Công cụ giám sát, trực quan hóa dữ liệu	Grafana 10.4.1	https://grafana.com/grafana/download/10.4.1
Ngôn ngữ lập trình	TypeScript	https://www.typescriptlang.org/
Framework	ReactJs	https://react.dev/
Ngôn ngữ lập trình	Java 20.0.1	https://www.java.com/en/
Framework	Quarkus	https://quarkus.io/
Thư viện	Fabric8 kubernetes java client	https://fabric8.io/
Thư viện	Keycloak	https://www.keycloak.org/
Database	MongoDB Compass v6.0.5	https://www.mongodb.com/
Nền tảng chạy container	Docker-desktop 25.0.3	https://www.docker.com/products/docker-desktop/

Bảng 4.13: Danh sách thư viện và công cụ sử dụng

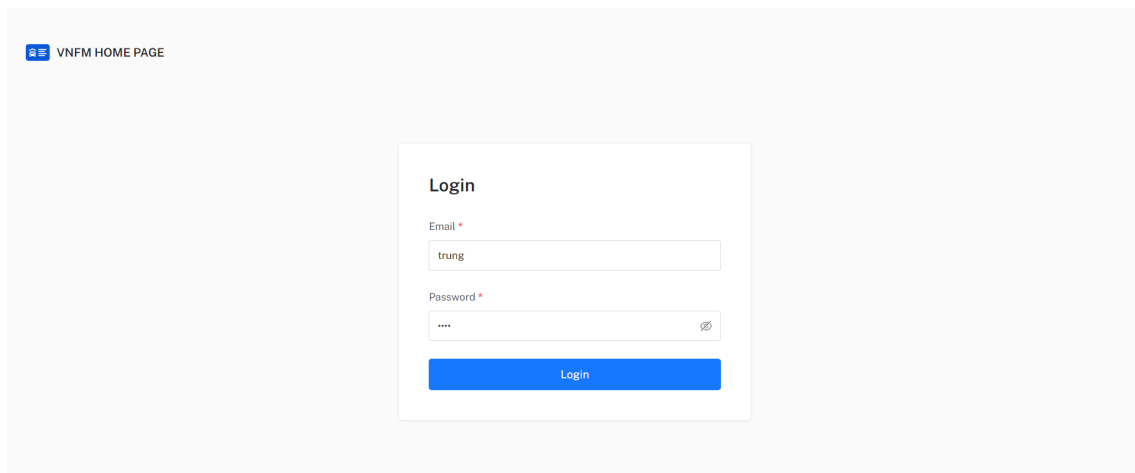
4.3.2 Kết quả đạt được

Trong phạm vi đề án, em đã xây dựng thành công một hệ thống quản lý chức năng mạng ảo hóa(NFV-MANO) cơ bản, bao gồm các chức năng:

- Quản lý vòng đời VNF
- Thu thập, giám sát hiệu năng hệ thống
- Quản lý cảnh báo lỗi hệ thống

4.3.3 Minh họa các chức năng chính

Chức năng đăng nhập người dùng

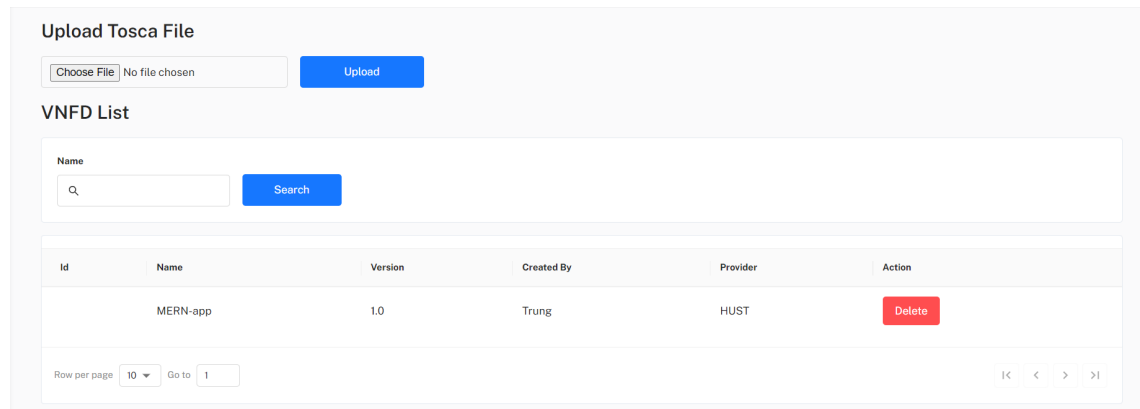


The screenshot shows a web browser window with the title "VNFM HOME PAGE". In the center, there is a "Login" form. The form has two input fields: "Email" with a red asterisk and a red error message, containing the text "trung"; and "Password" with a red asterisk and a red error message, containing four dots. To the right of the password field is a small icon for password visibility. Below the input fields is a blue "Login" button.

Hình 4.6: Giao diện đăng nhập người dùng

Chức năng tạo VNFD

Chức năng tạo VNFD (Virtual Network Function Descriptor) từ file TOSCA (Topology and Orchestration Specification for Cloud Applications) trong hệ thống NFV-MANO là quá trình chuyển đổi mô tả của các chức năng mạng ảo hóa từ định dạng TOSCA sang VNFD, nhằm sử dụng trong quản lý các VNF.

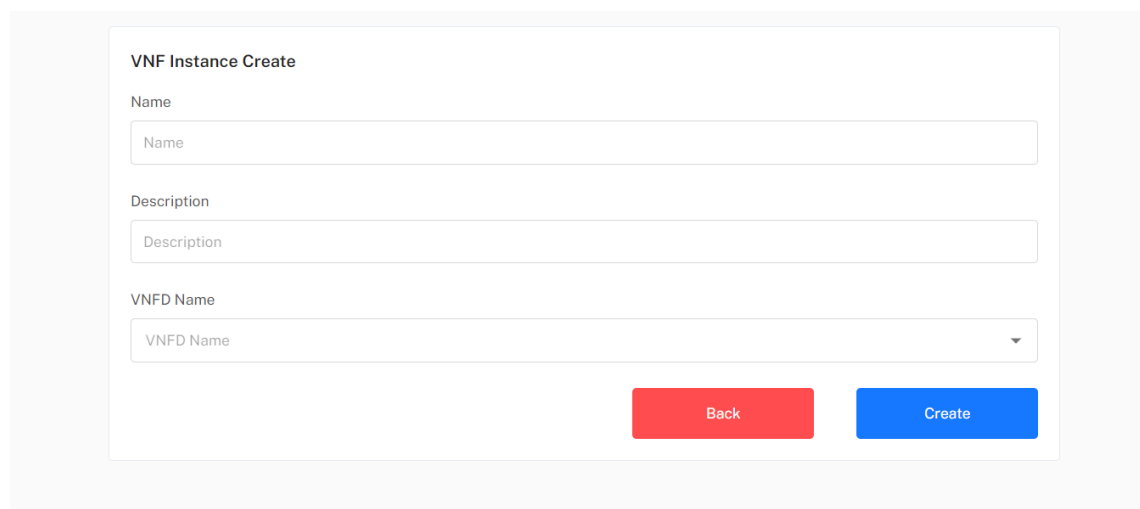


The screenshot displays a web interface for managing VNFDs. At the top, there is a section titled "Upload Tosca File" with a "Choose File" button (showing "No file chosen") and an "Upload" button. Below this is a "VNFD List" section featuring a search bar with a magnifying glass icon and a "Search" button. Underneath the search bar is a table with the following columns: Id, Name, Version, Created By, Provider, and Action. The table contains one entry with the name "MERN-app", version "1.0", created by "Trung", and provider "HUST". The "Action" column for this entry has a red "Delete" button. At the bottom of the table, there is a pagination control showing "Row per page" set to "10" and "Go to" set to "1".

Hình 4.7: Giao diện chức năng tạo VNFD

Chức năng khởi tạo VNF Instance

Khi có yêu cầu triển khai một VNF, MANO sẽ khởi tạo một VNF instance mới dựa trên mô tả VNF (VNFD) và các thông tin về môi trường triển khai.

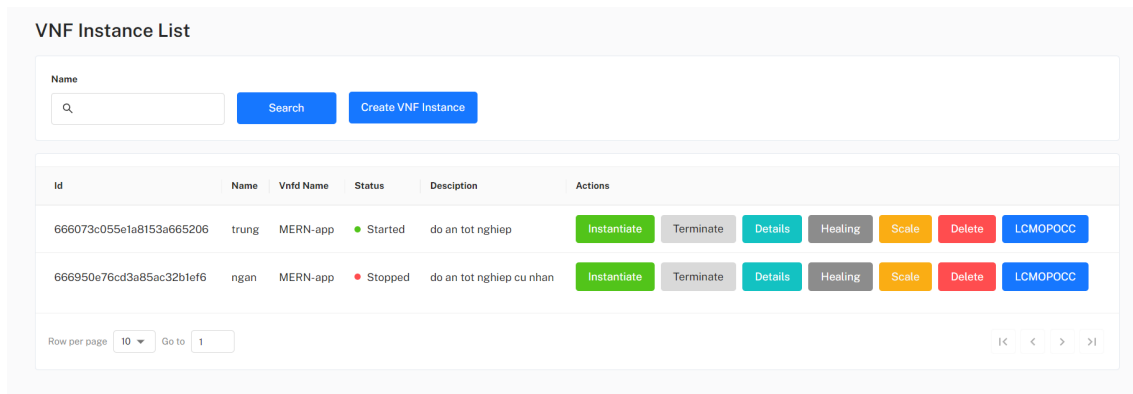


The screenshot shows a "VNF Instance Create" form. It includes three input fields: "Name", "Description", and "VNFD Name". The "VNFD Name" field is a dropdown menu currently showing "VNFD Name". At the bottom right of the form, there are two buttons: a red "Back" button and a blue "Create" button.

Hình 4.8: Giao diện chức năng khởi tạo VNF Instance

Chức năng bật VNF Instance

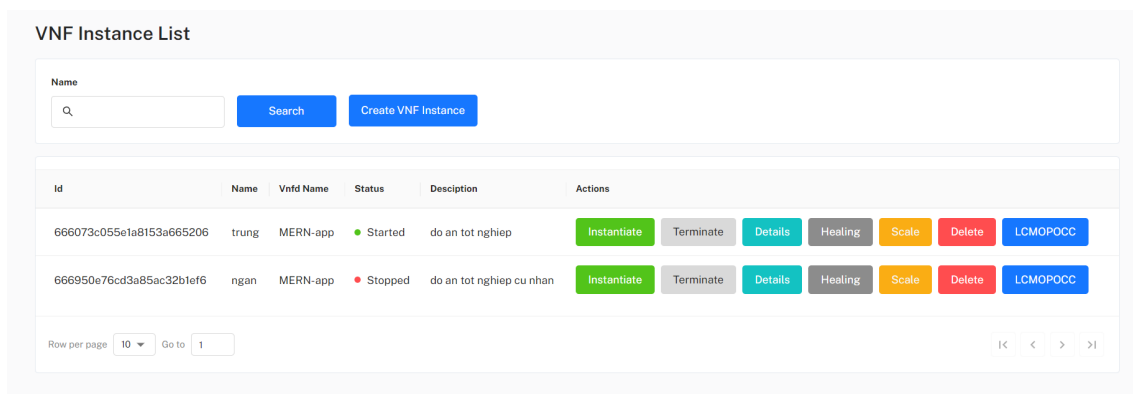
Sau khi khởi tạo VNF Instance, nếu người dùng muốn bật VNF Instance này lên để chạy ứng dụng và cung cấp ứng dụng đến người dùng, người dùng sẽ ấn nút **Instantiate**



Hình 4.9: Giao diện chức năng bật VNF Instance

Chức năng Tắt VNF Instance

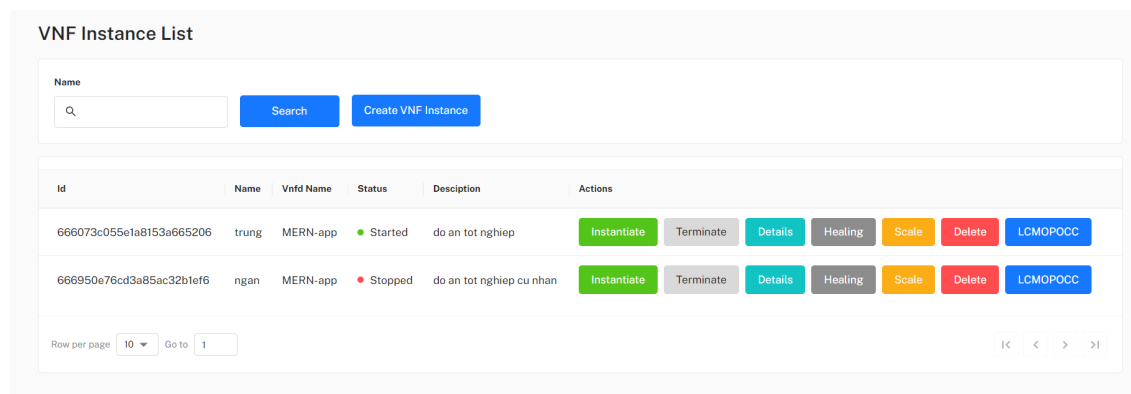
Sau khi VNF Instance đã được bật, nếu người dùng muốn tắt VNF Instance này, người dùng sẽ ấn nút **Terminate**. Sau khi tắt VNF Instance, ứng dụng sẽ không chạy nữa.



Hình 4.10: Giao diện chức năng tắt VNF Instance

Chức năng phục hồi VNF Instance

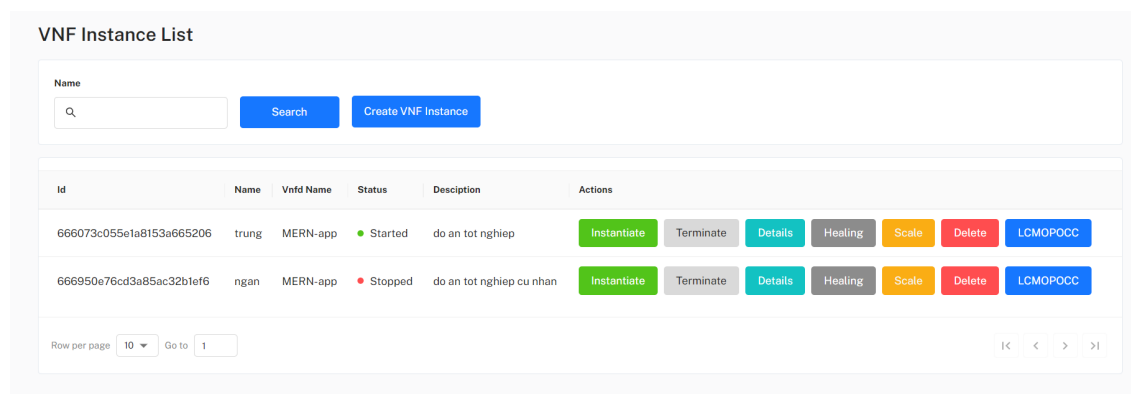
Trong trường hợp VNF Instance đang chạy nhưng có một hoặc nhiều VNFC trong VNF bị lỗi, người dùng có thể sử dụng chức năng phục hồi VNF Instance bằng cách ấn nút **Healing**. Sau đó các VNFC cũ sẽ bị xóa và VIM sẽ tạo ra các VNFC khác, ứng dụng có thể tiếp tục sử dụng.



Hình 4.11: Giao diện chức năng phục hồi VNF Instance

Chức năng Xóa VNF Instance

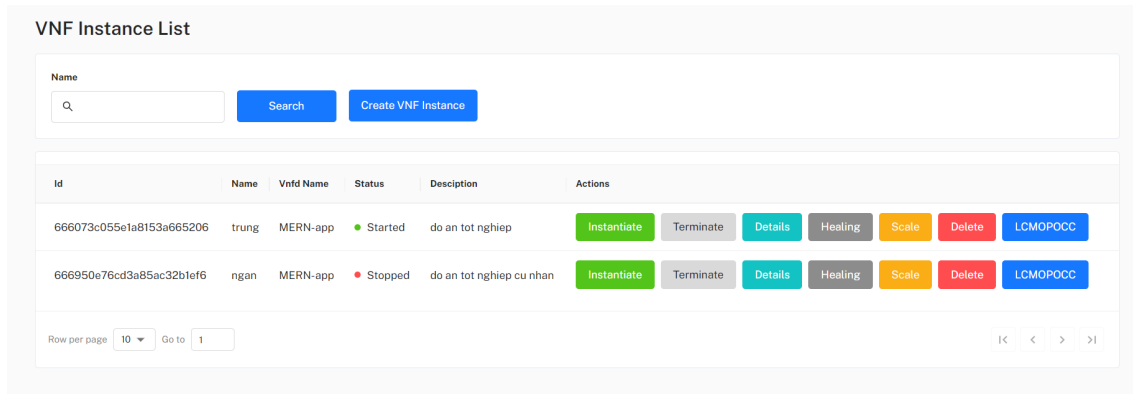
Nếu người dùng không muốn sử dụng VNF Instance này nữa, người dùng có thể sử dụng chức năng xóa VNF Instance bằng cách ấn nút **Delete**. Sau đó VNF Instance này sẽ bị xóa.



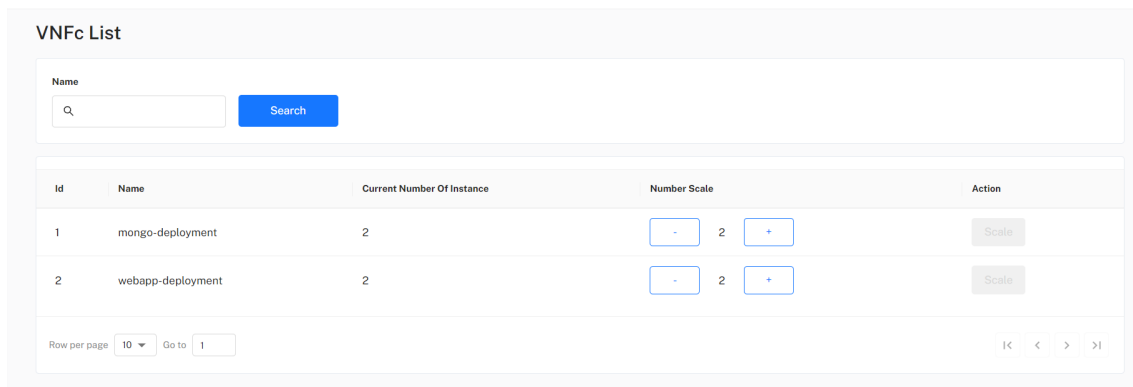
Hình 4.12: Giao diện chức năng xóa VNF Instance

Chức năng mở rộng VNF Instance

Trong trường hợp người dùng muốn tăng tính High Available cho VNF Instance, người dùng có thể sử dụng chức năng mở rộng VNF Instance bằng cách ấn nút **Scale**. Sau đó sẽ chọn VNFC và số lượng VNFC mà ta muốn scale. Sau đó ấn nút **Scale**.



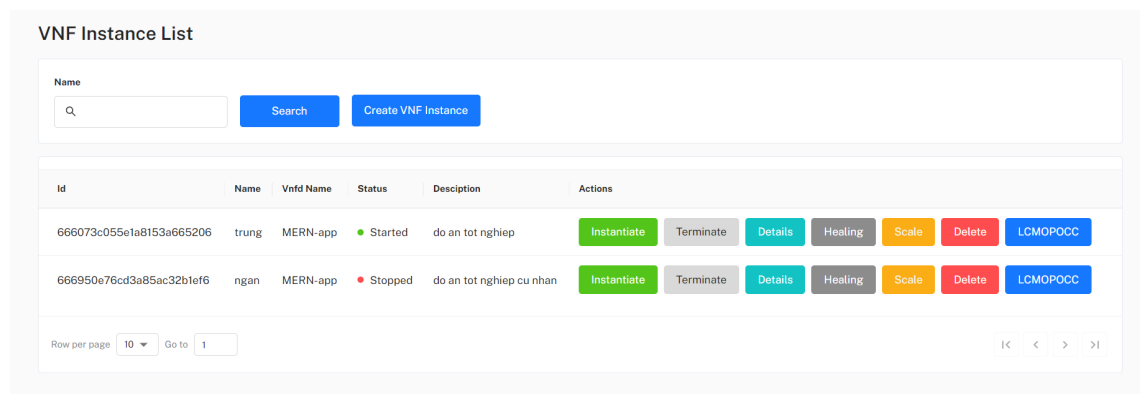
Hình 4.13: Giao diện chức năng mở rộng VNF Instance



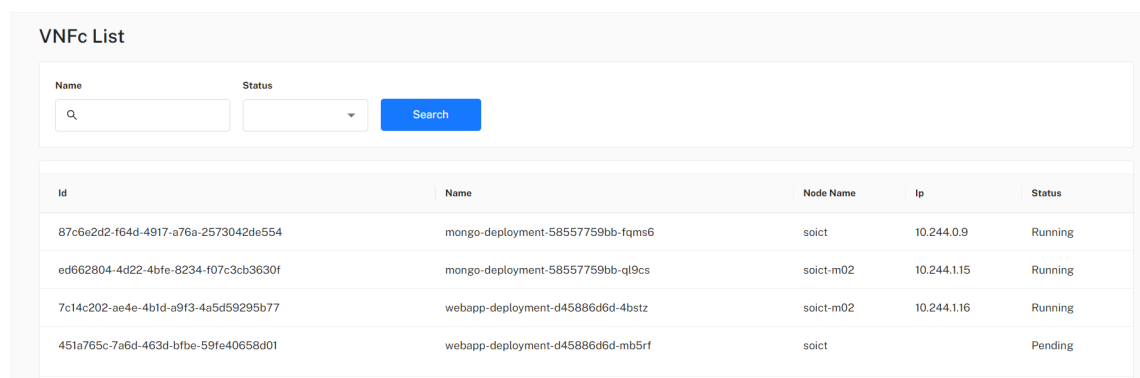
Hình 4.14: Giao diện chức năng mở rộng VNF Instance

Chức năng xem chi tiết các thành phần trong VNF Instance

Nếu người dùng muốn xem chi tiết các VNFC bên trong VNF Instance, người dùng chọn chức năng Detail bằng cách ấn nút **Details**.



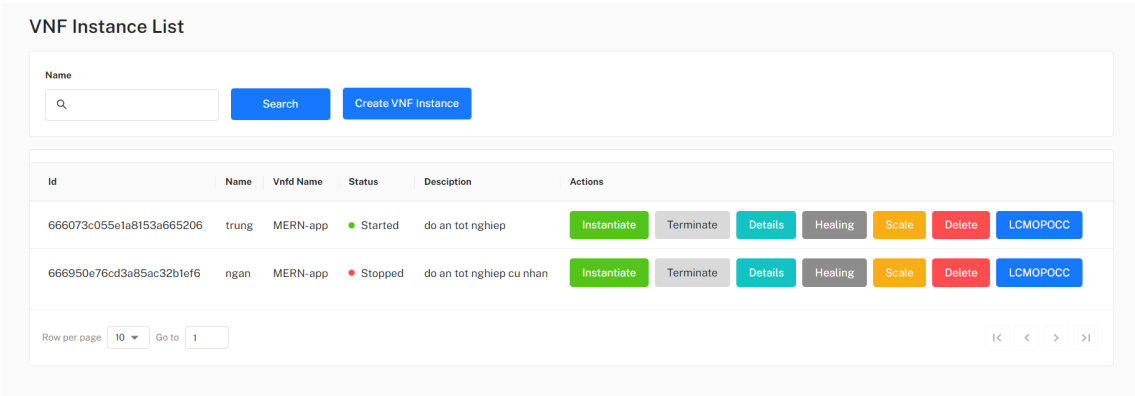
Hình 4.15: Giao diện chức năng xem chi tiết các thành phần trong VNF Instance



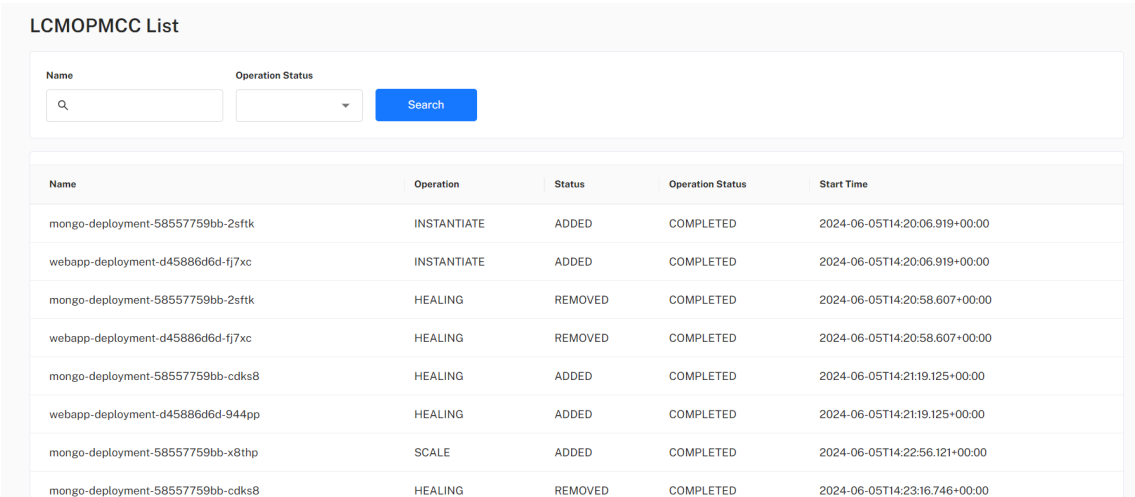
Hình 4.16: Giao diện chức năng xem chi tiết các thành phần trong VNF Instance

Chức năng LCMOPOCC

Nếu người dùng muốn xem các tương tác với VNFC trong VNF Instance, có thể sử dụng chức năng LCMOPOCC bằng cách ấn nút **LCMOPOCC**



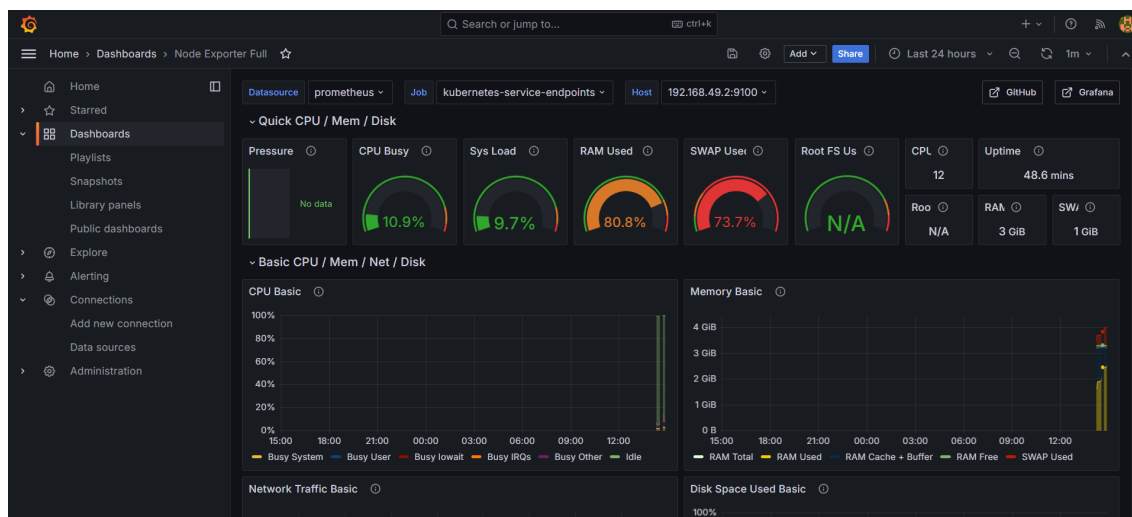
Hình 4.17: Giao diện chức năng LCMOPOCC



Hình 4.18: Giao diện chức năng LCMOPOCC

Chức năng thu thập, giám sát hiệu năng hệ thống

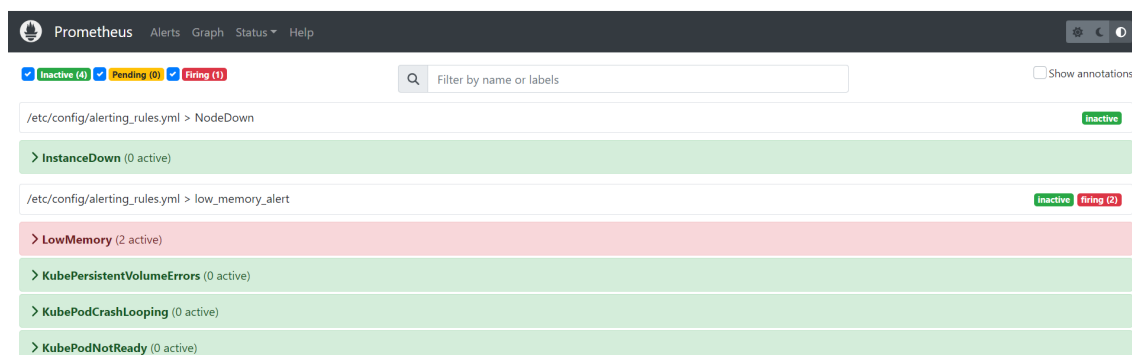
Chức năng thu thập, giám sát hiệu năng hệ thống sẽ thu thập các metric của hệ thống và biểu diễn bằng biểu đồ cho người dùng để quan sát và theo dõi. Người dùng có thể sử dụng chức năng thu thập, giám sát hiệu năng hệ thống bằng cách ấn nút **Performance**



Hình 4.19: Giao diện chức năng thu thập, giám sát hiệu năng hệ thống

Chức năng quản lý cảnh báo lỗi hệ thống

Chức năng quản lý cảnh báo lỗi hệ thống sẽ thông báo lỗi về gmail và prometheus nếu hệ thống có lỗi hoặc cảnh báo nào đó. Người dùng có thể sử dụng chức năng này bằng cách ấn nút **Alert**



Hình 4.20: Giao diện chức năng quản lý cảnh báo lỗi hệ thống

4.4 Kiểm thử

Kiểm thử chức năng tạo VNF Instance

STT	Mục đích	Đầu vào	Đầu ra mong muốn	Kết quả thực tế	Đánh giá
1	Kiểm thử chức năng tạo VNF instance	Tên, mô tả của VNF instance, tên của VNFD	Thông báo tạo VNF instance thành công	Đã có thông báo "Tạo VNF instance thành công"	Đạt
2	Kiểm thử trường hợp không chọn VNFD	Không chọn trường VNFD	Thông báo yêu cầu chọn VNFD	Đã có thông báo "Trường VNFD chưa được chọn"	Đạt

Bảng 4.14: Kiểm thử chức năng tạo VNF instance

Kiểm thử chức năng bật VNF Instance

STT	Mục đích	Đầu vào	Đầu ra mong muốn	Kết quả thực tế	Đánh giá
1	Kiểm thử chức năng bật VNF instance	Tên của VNF instance cần bật	Thông báo bật VNF instance thành công	Đã có thông báo "Bật VNF instance thành công"	Đạt

Bảng 4.15: Kiểm thử chức năng bật VNF instance

Kiểm thử chức năng scale VNF Instance

STT	Mục đích	Đầu vào	Đầu ra mong muốn	Kết quả thực tế	Đánh giá
1	Kiểm thử chức năng scale VNF instance	Tên của VNF instance, tên của VNFC và số lượng VNFC cần scale	Thông báo đã scale VNF instance thành công	Đã có thông báo "Scale VNF instance thành công"	Đạt

Bảng 4.16: Kiểm thử chức năng scale VNF instance

CHƯƠNG 5. CÁC GIẢI PHÁP VÀ ĐÓNG GÓP NỔI BẬT

5.1 Giải pháp triển khai ứng dụng dưới dạng container [1]

Vấn đề

Các cách deployment ứng dụng phổ biến

Thời đại triển khai theo cách truyền thống: Ban đầu, các ứng dụng được chạy trên các máy chủ vật lý. Không có cách nào để xác định ranh giới tài nguyên cho các ứng dụng trong máy chủ vật lý và điều này gây ra sự cố phân bổ tài nguyên. Ví dụ, nếu nhiều ứng dụng cùng chạy trên một máy chủ vật lý, có thể có những trường hợp một ứng dụng sẽ chiếm phần lớn tài nguyên hơn và kết quả là các ứng dụng khác sẽ hoạt động kém đi. Một giải pháp cho điều này sẽ là chạy từng ứng dụng trên một máy chủ vật lý khác nhau. Nhưng giải pháp này không tối ưu vì tài nguyên không được sử dụng đúng mức và rất tốn kém cho các tổ chức để có thể duy trì nhiều máy chủ vật lý như vậy.

Thời đại triển khai ảo hóa: Như một giải pháp, ảo hóa đã được giới thiệu. Nó cho phép bạn chạy nhiều Máy ảo (VM) trên CPU của một máy chủ vật lý. Ảo hóa cho phép các ứng dụng được cô lập giữa các VM và cung cấp mức độ bảo mật và thông tin của một ứng dụng không thể được truy cập tự do bởi một ứng dụng khác.

Ảo hóa cho phép sử dụng tốt hơn các tài nguyên trong một máy chủ vật lý và cho phép khả năng mở rộng tốt hơn vì một ứng dụng có thể được thêm hoặc cập nhật dễ dàng, giảm chi phí phần cứng và hơn thế nữa. Với ảo hóa, bạn có thể có một tập hợp các tài nguyên vật lý dưới dạng một cụm các máy ảo sẵn dùng.

Mỗi VM là một máy tính chạy tất cả các thành phần, bao gồm cả hệ điều hành riêng của nó, bên trên phần cứng được ảo hóa.

Điểm yếu của cách deploy này là do VM được virtualize bằng cách copy cả OS và phần cứng (hardware), nên một server chỉ có thể tạo một số lượng nhỏ VM (4 hoặc 5 với các server bình thường)

Giải pháp

Thời đại triển khai Container: Các container tương tự như các VM, nhưng chúng có tính cô lập để chia sẻ Hệ điều hành (HĐH) giữa các ứng dụng. Do đó, container được coi là nhẹ (lightweight). Giống như máy ảo, một container có hệ thống tệp, CPU, bộ nhớ, không gian tiến trình, v.v. Các container đã trở nên phổ biến vì chúng có thêm nhiều lợi ích, chẳng hạn như:

- Tạo mới và triển khai ứng dụng Agile: gia tăng tính dễ dàng và hiệu quả của

việc tạo các container image so với việc sử dụng VM image.

- **Phát triển, tích hợp và triển khai liên tục:** cung cấp khả năng build và triển khai container image thường xuyên và đáng tin cậy với việc rollbacks dễ dàng, nhanh chóng.
- **Phân biệt giữa Dev và Ops:** tạo các images của các application container tại thời điểm build/release thay vì thời gian triển khai, do đó phân tách các ứng dụng khỏi hạ tầng.
- **Khả năng quan sát không chỉ hiển thị thông tin và các metric ở mức Hệ điều hành, mà còn cả application health và các tín hiệu khác.**
- **Tính nhất quán về môi trường trong suốt quá trình phát triển, testing và trong production:** Chạy tương tự trên laptop như trên cloud.
- **Tính khả chuyển trên cloud và các bản phân phối HĐH:** Chạy trên Ubuntu, RHEL, CoreOS, on-premises, Google Kubernetes Engine và bất kì nơi nào khác.
- **Quản lý tập trung ứng dụng:** Tăng mức độ trừu tượng từ việc chạy một Hệ điều hành trên phần cứng ảo hóa sang chạy một ứng dụng trên một HĐH bằng logical resources.
- **Các micro-services phân tán, elastic:** ứng dụng được phân tách thành các phần nhỏ hơn, độc lập và thể được triển khai và quản lý một cách linh hoạt - chứ không phải một app nguyên khối (monolithic).
- **Cô lập các tài nguyên:** dự đoán hiệu năng ứng dụng

5.2 Giải pháp quản lý vòng đời các chức năng mạng ảo hóa

Vấn đề

- **Phức tạp trong triển khai:** Việc triển khai các chức năng mạng ảo (VNFs) đòi hỏi cấu hình phức tạp và thường xuyên phải tùy chỉnh theo nhu cầu cụ thể của từng môi trường.
- **Khả năng mở rộng:** Khả năng mở rộng nhanh chóng để đáp ứng nhu cầu gia tăng mà không gây gián đoạn dịch vụ là một thách thức lớn.
- **Quản lý và bảo trì liên tục:** Cập nhật và nâng cấp VNFs một cách liên tục để cải thiện hiệu năng, trong khi vẫn phải đảm bảo dịch vụ không bị gián đoạn.

Giải pháp

- **Tự động hóa quy trình triển khai:** Sử dụng các công cụ điều phối (orchestrators) như Kubernetes để tự động hóa việc triển khai, mở rộng và quản lý

VNFs.

5.3 Giải pháp thu thập và giám sát hiệu năng hệ thống

Vấn đề

- **Thu thập số liệu hiệu năng phức tạp:** Các VNFs và hệ thống mạng ảo hóa có nhiều thành phần phức tạp, làm cho việc thu thập và quản lý số liệu hiệu năng trở nên khó khăn.
- **Khả năng lưu trữ và truy vấn dữ liệu:** Lượng dữ liệu giám sát lớn đòi hỏi hệ thống phải có khả năng lưu trữ hiệu quả và truy vấn nhanh chóng để phân tích và giám sát liên tục.
- **Trực quan hóa dữ liệu:** Dữ liệu giám sát cần được trình bày một cách trực quan và dễ hiểu để giúp các quản trị viên hệ thống nhanh chóng nhận diện và xử lý các vấn đề

Giải pháp

Sử dụng Prometheus để thu thập dữ liệu hiệu năng hệ thống

- **Exporter:** Sử dụng các exporter để thu thập số liệu từ các VNFs và hệ thống mạng. Prometheus exporter là các ứng dụng thu thập và cung cấp các số liệu hiệu năng thông qua các endpoint HTTP.
- **Scraping:** Cấu hình Prometheus để thực hiện việc thu thập (scrape) số liệu từ các endpoint được chỉ định ở các khoảng thời gian định kỳ.

Trực quan hóa dữ liệu bằng Grafana:

- **Dashboards:** Tạo và tùy chỉnh các bảng điều khiển (dashboards) trong Grafana để trực quan hóa các số liệu hiệu năng được thu thập bởi Prometheus.
- **Panels:** Sử dụng các loại panel khác nhau (như graph, gauge, heatmap) để hiển thị dữ liệu theo nhiều cách trực quan và dễ hiểu.

5.4 Giải pháp Quản lý cảnh báo lỗi hệ thống

Vấn đề

- **Phát hiện và chẩn đoán lỗi:** Khó khăn trong việc phát hiện và chẩn đoán chính xác nguyên nhân gây ra lỗi trong một hệ thống phức tạp.
- **Chẩn đoán lỗi phức tạp:** Hệ thống mạng phức tạp với nhiều thành phần khác nhau khiến việc chẩn đoán nguyên nhân gây ra lỗi trở nên khó khăn.
- **Quản lý và phân loại cảnh báo:** Khi xảy ra nhiều lỗi cùng một lúc, việc quản lý và phân loại các cảnh báo để xác định mức độ ưu tiên và phản ứng kịp thời là một thách thức.

Giải pháp Phát hiện lỗi kịp thời với Prometheus

- **Hệ thống cảnh báo tự động:** Sử dụng các hệ thống cảnh báo tự động để phát hiện và thông báo ngay lập tức khi xảy ra lỗi.
- **Scraping Interval:** Cấu hình Prometheus với khoảng thời gian thu thập số liệu ngắn để phát hiện các vấn đề nhanh chóng.
- **Alerting Rules:** Thiết lập các quy tắc cảnh báo (alerting rules) để giám sát các số liệu quan trọng và kích hoạt cảnh báo ngay khi phát hiện bất thường.
- **Alertmanager:** Sử dụng Alertmanager để quản lý cảnh báo và gửi thông báo đến các kênh thông báo như Prometheus, email để đảm bảo phản ứng nhanh chóng.

CHƯƠNG 6. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

6.1 Kết luận

Trong đồ án này, em đã thực hiện được nhiều nhóm công việc khác nhau. Bao gồm:

- Về mặt lý thuyết: hiểu được tổng quan về công nghệ ảo hóa chức năng mạng, quá trình phát triển, kiến trúc, các usecase và tiềm năng phát triển ở Việt Nam
- Về mặt thực nghiệm: Xây dựng thành công một hệ thống quản lý và điều phối chức năng mạng ảo hóa(NFV-MANO) cơ bản, bao gồm việc tìm hiểu, xác định yêu cầu của hệ thống, phân tích và thiết kế hệ thống, viết mã nguồn ứng dụng web, v.v...

6.2 Hướng phát triển

Vì thời gian nghiên cứu có hạn, đồng thời NFV là một công nghệ mới và vẫn đang trong giai đoạn hoàn thiện, chính vì thế còn rất nhiều vấn đề chưa thể thực hiện trong phạm vi của đồ án này. Sau đây là một số hướng phát triển cho đề tài về NFV được em đề ra:

Có thể ứng dụng cho các chức năng mạng tổng quát. Trong đồ án này, em đang minh họa VNF cụ thể là 1 website. Trong tương lai, em sẽ phát triển hệ thống của mình để phù hợp với tất cả các VNF theo tiêu chuẩn của ETSI.

Xây dựng thêm chức năng NFVO bao gồm chức năng quản lý các Network Services(NS) và quản lý các gói VNF Packages.

TÀI LIỆU THAM KHẢO

- [1] Cloud Native Computing Founder, *Kubernetes*. [Online]. Available: <https://kubernetes.io/vi/docs/concepts/overview/what-is-kubernetes/> (visited on 06/25/2024).
- [2] SoundCloud, *Prometheus*. [Online]. Available: <https://prometheus.io/docs/introduction/overview/> (visited on 06/25/2024).
- [3] SoundCloud, *Grafana*. [Online]. Available: <https://grafana.com/> (visited on 06/25/2024).
- [4] Red Hat, *Quarkus*. [Online]. Available: <https://quarkus.io/> (visited on 06/25/2024).
- [5] Microsoft, *Typescript*. [Online]. Available: <https://www.typescriptlang.org/> (visited on 06/25/2024).
- [6] Rackspace Hosting, NASA, *Fabric8 kubernetes java client*. [Online]. Available: <https://github.com/fabric8io/kubernetes-client> (visited on 06/25/2024).
- [7] Baeldung, *Parsing yaml with snakeyaml*. [Online]. Available: <https://www.baeldung.com/java-snake-yaml> (visited on 06/25/2024).
- [8] Viblo, *Tìm hiểu mô hình mvc cho người mới bắt đầu*. [Online]. Available: <https://viblo.asia/p/tim-hieu-mo-hinh-mvc-danh-cho-nguoi-moi-bat-dau-cau-truc-va-vi-du-V3m5WLDyK07> (visited on 06/25/2024).