HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

Lê Quang Thành

NGHIÊN CỨU ỨNG DỤNG CÔNG NGHỆ KUBERNETES CHO VIỆC TRIỂN KHAI HỆ THỐNG SOA TRONG HỆ THỐNG NGÂN HÀNG

Chuyên ngành: Hệ thống thông tin

Mã số: 8.48.01.04

ĐỀ ÁN TỐT NGHIỆP THẠC SĨ KỸ THUẬT

(Theo định hướng ứng dụng)

NGƯỜI HƯỚNG DẪN KHOA HỌC

TS. VŨ VĂN THỎA

HÀ NỘI – 2025

LỜI CAM ĐOAN

Tôi cam đoan rằng công trình nghiên cứu này là thành quả của công sức cá nhân của tôi và không sao chép từ bất kỳ nguồn nào khác. Tất cả thông tin được trình bày trong đề án này đều là sản phẩm của công việc cá nhân hoặc được tổng hợp từ nhiều nguồn tài liệu khác nhau. Mọi tài liệu tham khảo đều được trích dẫn một cách hợp pháp và có nguồn gốc rõ ràng. Các dữ liệu và kết quả được trình bày trong đề án đều là trung thực và chưa từng được công bố trong bất kỳ công trình nghiên cứu nào khác.

Tác giả đề án

Lê Quang Thành

LỜI CẢM ƠN

Lời đầu tiên, học viên xin gửi lời cảm ơn chân thành tới Học viện Công nghệ Bưu chính Viễn thông, Khoa Đào tạo Sau đại học cùng toàn thể các thầy cô của Học viện đã tạo điều kiện thuận lợi và trực tiếp giảng dạy, hỗ trợ em trong suốt quá trình học tập chương trình Thạc sĩ Kỹ thuật.

Em đặc biệt biết ơn TS. VŨ VĂN THỎA, người đã dành nhiều thời gian, tâm huyết hướng dẫn tận tình và đưa ra những góp ý quý báu giúp em hoàn thiện bản đề cương đề án này.

Cuối cùng, em xin bày tỏ lòng biết ơn sâu sắc tới gia đình, người thân và bạn bè đã luôn đồng hành, động viên và hỗ trợ em trong suốt quá trình học tập và hoàn thành đề án.

MỤC LỤC

LỜI CAM ĐOAN i

LỜI CẢM ƠN ii

MỤC LỤC iii

DANH MỤC TỪ VIẾT TẮT vi

DANH SÁCH BẢNG vii

DANH SÁCH HÌNH VẼ viii

MỞ ĐẦU 1

CHƯƠNG I: TỔNG QUAN VỀ HỆ THỐNG SOA VÀ CÁC CÔNG NGHỆ LIÊN QUAN 3

1.1 Hệ thống SOA 3

1.1.1 Khái niệm 3

1.1.2. Đặc điểm 3

1.1.3. Thành phần cơ bản 5

1.2. TIBCO BusinessWorks 5 5

1.2.1. Đặc điểm nổi bật 6

1.2.2. Kiến trúc hoạt động 6

1.2.3. Hạn chế của TIBCO BusinessWorks 5 6

1.3. TIBCO BusinessWorks 6 8

1.3.1. Đặc điểm nổi bật của TIBCO BW6 so với BW5 8

1.3.2. So sánh BW5 và BW6 9

1.4. Docker và Kubernetes 9

1.4.1. Docker 9

1.4.2. Kubernetes 11

1.5. Kết luận chương 1 12
CHƯƠNG II: NGHIÊN CỨU TRIỂN KHAI HỆ THỐNG SOA TRONG NGÂN HÀNG....14
2.1. Tổng quan về mô hình SOA trong ngân hàng....14
2.2. Thực trạng triển khai trục tích hợp bằng TIBCO BW5 16
2.2.1 Thực trạng....16
2.2.2. Hạn chế 19
2.3. Đề xuất mô hình triển khai trục tích hợp bằng TIBCO BW6 kết hợp Kubernetes 20
2.3.1. Mô hình đề xuất 20
2.3.2. Đặc điểm nổi bật....21
2.3.3. Khó khăn và hạn chế 22
2.4. Kết luận chương 2 23
CHƯƠNG III: TRIỂN KHAI HỆ THỐNG SOA TRÊN NỀN TẢNG KUBERNETES TẠI NGÂN HÀNG BIDV 24
3.1. Tổng quan kiến trúc....24
3.2. Mô hình CI/CD....28
3.2.1. Quy trình CI....28
3.2.2. Quy trình CD 31
3.3. Thực nghiệm triển khai services 34
3.3.1. Sơ đồ concept OpenShift kết hợp Kubernetes....34
3.3.2. Service gọi đến lõi ngân hàng....37
3.3.3. Đọc log và ghi log vào log tập trung 43
3.3.4. Bảng điều khiển giám sát kiểm soát (Control Monitoring Dashboard)....46

3.4. Đánh giá hiệu quả của sử dụng TIBCO BW6 kết hợp với Openshift và Kubernetes 48
3.4.1. Khả năng Auto Scaling 48
3.4.2. Quản lý log tập trung 49
3.4.3. Dashboard monitoring 50
3.4.4. Năng lực CI/CD 51
3.5. Thử nghiệm test tải trên Openshift 52
3.5.1. Cấu hình 52
3.5.2. Các tiêu chuẩn SLA/PAC 52
3.5.3. Kịch bản kiểm thử 53
3.7. Kết luận chương 3 55
KẾT LUẬN 56
1. Các kết quả đạt được của đề án 56
2. Hướng nghiên cứu tiếp theo 57
DANH MỤC TÀI LIỆU THAM KHẢO 58

DANH MỤC TỪ VIẾT TẮT

STT	Từ viết tắt	Tiếng Anh	Tiếng Việt
1	SOA	Service-Oriented Architecture	Kiến trúc hướng dịch vụ
2	HPA	Horizontal Pod Autoscaler	Cơ chế tự động mở rộng Pod theo chiều ngang
3	HTTP	HyperText Transfer Protocol	Giao thức truyền tải siêu văn bản
4	SOAP	Simple Object Access Protocol	Giao thức truy cập đối tượng đơn giản
5	REST	Representational State Transfer	Phong cách kiến trúc REST
6	WSDL	Web Services Description Language	Ngôn ngữ mô tả dịch vụ web
7	GUI	Graphical User Interface	Giao diện người dùng đồ họa
8	ESB	Enterprise Service Bus	Trục dịch vụ doanh nghiệp
9	API	Application Programming Interface	Giao diện lập trình ứng dụng
10	gRPC	Google Remote Procedure Call	Giao thức RPC hiệu năng cao do Google phát triển
11	UAT	User Acceptance Testing	Kiểm thử chấp nhận người dùng

DANH SÁCH BẢNG

Bảng 1.1: So sánh giữa TIBCO BW5 và TIBCO BW6.....9
Bảng 3.1: Bảng mô tả các kết nối giữa SOA 2.0 và các hệ thống liên quan .27

DANH SÁCH HÌNH VẼ

Hình 2.1: Tổng quan về hệ thống SOA trong ngân hàng....15
Hình 2.2: Hiện trạng triển khai dịch vụ khi sử dụng TIBCO BW5....18
Hình 2.3: Mô hình triển khai dịch vụ khi sử dụng TIBCO BW6....20
Hình 3.1: Tổng quan kiến trúc SOA....26
Hình 3.2: Mô hình CI/CD đơn giản....28
Hình 3.3: Quy trình CI/CD trên Gitlab....28
Hình 3.4: Quy trình CI trên Gitlab....29
Hình 3.5: Logic code trong quy trình CI....29
Hình 3.6: Logic code tải image lên đám mây....31
Hình 3.7: Pha CD trên Gitlab....32
Hình 3.8: Logic của pha CD....33
Hình 3.9: Logic của pha CD (tiếp)....33
Hình 3.10: Giao diện ArgoCD....34
Hình 3.11: Tổng quan về Openshift....35
Hình 3.12: Tổng quan cấu trúc Openshift....36
Hình 3.13: Cấu trúc service gọi đến lõi ngân hàng....37
Hình 3.14: Toàn bộ hệ thống SOA phổ thông....37
Hình 3.15: Ví dụ về HPA....38
Hình 3.16: Tài nguyên của 1 service....39
Hình 3.17: Nội dung HorizontalPodAutoscaler....40
Hình 3.18: Nội dung Deployment....41
Hình 3.19: ConfigMaps....42
Hình 3.20: Routes....42
Hình 3.21: Services....42
Hình 3.22: Ví dụ log của 1 service....43
Hình 3.23: Mô hình ELK....44
Hình 3.24: Giao diện Kafka....45
Hình 3.25: Giao diện Elasticsearch....45

Hình 3.26: Dashboard của Openshift.....47
Hình 3.27: Dashboard của 1 service TIBCO BW5.....48
Hình 3.28: Danh sách node của 1 service TIBCO BW6.....49
Hình 3.29: HPA của 1 service.....49
Hình 3.30: Log phân tán trên TIBCO BW5.....49
Hình 3.31: Thông tin sơ sài của service trên TIBCO BW5.....50

MỞ ĐẦU

Trong thời đại chuyển đổi số đã và đang trở lên mạnh mẽ trong lĩnh vực ngành ngân hàng, việc nâng cấp, hiện đại hóa hạ tầng công nghệ thông tin đóng vai trò vô cùng quan trọng nhằm tăng cường khả năng cạnh tranh, tính linh hoạt và đảm bảo hiệu quả vận hành. Một trong những nền tảng tích hợp phổ biến được các ngân hàng sử dụng nhiều năm qua là kiến trúc định hướng dịch vụ (SOA – Service-Oriented Architecture), cho phép các dịch vụ nghiệp vụ được xây dựng, triển khai và tích hợp một cách linh hoạt.

Tuy nhiên, thực tế triển khai SOA trên các nền tảng truyền thống như TIBCO BusinessWorks 5 (BW5) đang bộc lộ nhiều hạn chế. Việc triển khai thủ công, khó mở rộng, thiếu sự linh hoạt trong việc cập nhật và bảo trì hệ thống, cũng như không phù hợp với mô hình DevOps hiện đại, đã đặt ra yêu cầu cấp thiết phải có giải pháp tối ưu hơn. Trong khi đó, sự ra đời của TIBCO BW6 với khả năng tích hợp sẵn Docker và hỗ trợ triển khai trên nền tảng Kubernetes đã mở ra một hướng tiếp cận mới trong việc xây dựng các hệ thống SOA linh hoạt, hiện đại, dễ quản lý và phù hợp với xu hướng cloud-native hiện nay.

Việc nghiên cứu ứng dụng Kubernetes vào triển khai SOA trong hệ thống ngân hàng là cần thiết và có tính thời sự, vừa có ý nghĩa lý luận trong bối cảnh phát triển công nghệ tích hợp phân tán, vừa có ý nghĩa thực tiễn khi giúp các tổ chức tài chính nâng cao hiệu suất, tiết kiệm tài nguyên và gia tăng tính ổn định của hệ thống.

Chính vì vậy, học viên chọn đề tài "**Ứng dụng công nghệ Kubernetes cho việc triển khai hệ thống SOA trong hệ thống ngân hàng**" để thực hiện đề án tốt nghiệp thạc sĩ chuyên ngành Hệ thống thông tin.

Đề tài nhằm làm rõ vai trò của SOA trong ngân hàng và hệ thống hóa kiến thức về TIBCO BW5, BW6 cùng công nghệ Kubernetes. Nghiên cứu phân tích lợi ích, thách thức và khả năng ứng dụng Kubernetes trong triển khai SOA, đồng thời so sánh giữa mô hình BW5 truyền thống và BW6 container hóa. Trên cơ sở đó, đề tài đề xuất mô hình triển khai BW6 trên Kubernetes và đánh giá tính hiệu quả, khả thi khi áp dụng vào thực tiễn ngành ngân hàng.

Đề tài tập trung nghiên cứu hệ thống tích hợp SOA trong môi trường ngân hàng, đặc biệt là nền tảng TIBCO BusinessWorks BW6 cùng các công nghệ container hóa và điều phối Kubernetes. Phạm vi nghiên cứu được xác định trong không gian của hệ thống ngân hàng hoặc môi trường mô phỏng có cấu trúc tương đương, thực hiện trong giai đoạn 2024–2025 nhằm phản ánh bối cảnh công nghệ hiện tại và xu hướng phát triển trong thời gian tới. Về kỹ thuật, nghiên cứu chủ yếu tập trung vào triển khai, quản trị và đánh giá hiệu quả của hệ thống SOA khi chuyển đổi từ BW5 sang BW6 kết hợp Kubernetes

Nội dung đề án được trình bày thành ba chương theo cấu trúc sau:

Chương 1: Tổng quan về hệ thống SOA và các công nghệ liên quan

Chương này trình bày cơ sở lý luận và các khái niệm nền tảng liên quan đến đề tài như SOA, TIBCO BusinessWorks, Docker và Kubernetes.

Chương 2: Nghiên cứu triển khai hệ thống SOA trong ngân hàng

Chương này trình bày tổng quan thực trạng triển khai hệ SOA trong các ngân hàng, với trọng tâm là phân tích hai mô hình triển khai phổ biến hiện nay: TIBCO BusinessWorks 5 (BW5) và TIBCO BusinessWorks 6 (BW6). Trên cơ sở lý thuyết và khảo sát thực tế, chương này đánh giá những hạn chế của mô hình cũ và phân tích tiềm năng của việc ứng dụng công nghệ Kubernetes vào triển khai hệ thống SOA, từ đó làm rõ tính cấp thiết của hướng tiếp cận mới trong bối cảnh hiện đại hóa hạ tầng công nghệ ngân hàng.

Chương 3: Triển khai hệ thống SOA trên nền tảng Kubernetes tại ngân hàng BIDV

Chương này đề xuất mô hình triển khai hệ thống SOA sử dụng TIBCO BW6 trên nền tảng Kubernetes, đồng thời trình bày kết quả mô phỏng hoặc thử nghiệm để chứng minh tính khả thi.

CHƯƠNG I: TỔNG QUAN VỀ HỆ THỐNG SOA VÀ CÁC CÔNG NGHỆ LIÊN QUAN

1.1 Hệ thống SOA

1.1.1 Khái niệm

Kiến trúc hướng dịch vụ (SOA) là một phương pháp phát triển phần mềm trong đó các chức năng được đóng gói thành các "dịch vụ" độc lập, sau đó được sử dụng để xây dựng các ứng dụng doanh nghiệp. Mỗi dịch vụ đảm nhiệm một nghiệp vụ cụ thể và có khả năng giao tiếp, phối hợp với các dịch vụ khác, bất kể chúng được triển khai trên nền tảng hay ngôn ngữ lập trình nào. Nhờ SOA, nhà phát triển có thể tái sử dụng cùng một dịch vụ cho nhiều hệ thống hoặc kết hợp nhiều dịch vụ nhỏ lẻ để xử lý những tác vụ phức tạp hơn.

Chẳng hạn, nếu nhiều quy trình nghiệp vụ trong doanh nghiệp đều cần chức năng xác thực người dùng, thay vì phải triển khai lại mã nguồn cho từng quy trình, tổ chức chỉ cần xây dựng một dịch vụ xác thực chung và tái sử dụng nó cho mọi ứng dụng. Tương tự, trong lĩnh vực y tế, các hệ thống như quản lý bệnh nhân hay hồ sơ y tế điện tử (EHR) đều cần chức năng đăng ký bệnh nhân; thay vì tự xử lý riêng lẻ, các hệ thống này có thể gọi tới một dịch vụ đăng ký dùng chung để thực hiện nghiệp vụ đó.

1.1.2. Đặc điểm

Hệ thống SOA (Kiến trúc hướng dịch vụ) là một mô hình thiết kế phần mềm, trong đó các thành phần ứng dụng được tổ chức thành các dịch vụ rời rạc, có thể tái sử dụng và độc lập với nhau. Những đặc điểm nổi bật của hệ thống SOA bao gồm:

a) Phân tách lỏng lẻo (Loose coupling):

Các dịch vụ trong SOA có thể hoạt động và giao tiếp với nhau mà không cần biết chi tiết về cách thức triển khai của nhau.

Ưu điểm: Khi một dịch vụ thay đổi hoặc được cập nhật, các dịch vụ khác sử dụng nó sẽ không bị ảnh hưởng, giúp giảm thiểu rủi ro và tăng tính linh hoạt.

Ví dụ: Một ứng dụng thương mại điện tử có dịch vụ xử lý thanh toán. Nếu dịch vụ này được nâng cấp, các dịch vụ khác như quản lý giỏ hàng vẫn hoạt động bình thường mà không cần thay đổi gì.

b) Khả năng tái sử dụng (Reusability)

Các dịch vụ được thiết kế để thực hiện một chức năng kinh doanh cụ thể và có thể được sử dụng lại trong nhiều ứng dụng khác nhau.

Ưu điểm: Tiết kiệm thời gian và chi phí phát triển, vì các nhà phát triển không cần phải xây dựng lại các chức năng đã có sẵn.

Ví dụ: Dịch vụ xác thực người dùng có thể được sử dụng lại cho nhiều ứng dụng khác nhau của công ty, từ ứng dụng web đến ứng dụng di động.

c) Khả năng tương tác (Interoperability)

Các dịch vụ trong SOA có thể giao tiếp với nhau dù được phát triển bằng các ngôn ngữ lập trình hoặc trên các nền tảng khác nhau.

Cơ chế: Các dịch vụ sử dụng các giao thức truyền thông chuẩn như HTTP, SOAP, hoặc REST để trao đổi dữ liệu.

Ví dụ: Một dịch vụ được xây dựng bằng Java có thể dễ dàng trao đổi dữ liệu với một dịch vụ khác được viết bằng .NET, cho phép tích hợp các hệ thống cũ (legacy systems) với các công nghệ mới.

d) Tính trừu tượng (Abstraction)

Chi tiết về cách triển khai của một dịch vụ được ẩn đi khỏi người dùng hoặc ứng dụng sử dụng nó.

Lợi ích: Các nhà phát triển chỉ cần biết giao diện (interface) và chức năng của dịch vụ để sử dụng, giúp đơn giản hóa quá trình phát triển.

Ví dụ: Khi sử dụng dịch vụ kiểm tra tín dụng, nhà phát triển chỉ cần gửi thông tin khách hàng và nhận kết quả, mà không cần biết bên trong dịch vụ đó xử lý dữ liệu như thế nào.

e) Khả năng mở rộng (Scalability)

Các dịch vụ có thể được mở rộng độc lập với nhau, cho phép hệ thống tăng hoặc giảm hiệu suất tùy theo nhu cầu.

Cách thức: Khi một dịch vụ gặp phải tải lớn, nhà quản trị có thể khởi chạy thêm các phiên bản (instance) của dịch vụ đó để đáp ứng nhu cầu mà không ảnh hưởng đến các dịch vụ khác.

f) Được đóng gói (Self-contained)

Mỗi dịch vụ là một đơn vị chức năng độc lập, chứa tất cả logic và dữ liệu cần thiết để thực hiện công việc của nó.

Ưu điểm: Cho phép các dịch vụ được phát triển, triển khai và bảo trì một cách độc lập, giúp cô lập lỗi và tăng tính ổn định của hệ thống

1.1.3. Thành phần cơ bản

Các thành phần cơ bản của hệ thống SOA bao gồm: nhà cung cấp dịch vụ, người tiêu dùng dịch vụ, hợp đồng dịch vụ và kho lưu trữ dịch vụ.

Nhà cung cấp dịch vụ (Service Provider): là một ứng dụng hoặc hệ thống tạo ra và cung cấp một dịch vụ cụ thể. Việc này tạo, quản lý và bảo trì dịch vụ, đồng thời công bố dịch vụ đó lên kho lưu trữ để các ứng dụng khác có thể sử dụng.

Người tiêu dùng dịch vụ (Service Consumer): là một ứng dụng, hệ thống hoặc người dùng khác tìm kiếm và gọi các dịch vụ do nhà cung cấp cung cấp. Tìm kiếm dịch vụ phù hợp trong kho lưu trữ, rồi kết nối với nhà cung cấp để sử dụng chức năng của dịch vụ.

Hợp đồng dịch vụ (Service Contract): là một thỏa thuận chính thức xác định cách thức nhà cung cấp và người tiêu dùng tương tác với nhau. Hợp đồng có thể được định nghĩa bằng một tài liệu ngôn ngữ như WSDL (Web Services Description Language), xác định các yêu cầu và phản hồi cần thiết để giao tiếp với dịch vụ.

Kho lưu trữ dịch vụ (Service Registry): hay còn gọi là kho chứa dịch vụ (Service Repository), là một danh bạ trung tâm lưu trữ thông tin về tất cả các dịch vụ có sẵn trong hệ thống SOA. Kho cho phép người tiêu dùng dịch vụ khám phá và tìm kiếm các dịch vụ dựa trên nhu cầu của họ.

1.2. TIBCO BusinessWorks 5

TIBCO BusinessWorks 5 (hay còn gọi là TIBCO BW 5.x) là một nền tảng tích hợp và tự động hóa quy trình kinh doanh mạnh mẽ, được phát triển bởi TIBCO

Software Inc. Đây là một công cụ có lịch sử lâu đời, đã được nhiều doanh nghiệp sử dụng rộng rãi trong hàng thập kỷ để xây dựng các giải pháp tích hợp đa dạng.

1.2.1. Đặc điểm nổi bật

Phát triển dựa trên giao diện đồ họa (GUI): TIBCO BusinessWorks 5 cho phép người dùng thiết kế và phát triển các quy trình tích hợp một cách trực quan, sử dụng các thao tác kéo-và-thả mà không cần viết nhiều mã lập trình.

Hỗ trợ SOA và ESB: Đây là một công cụ lý tưởng để triển khai Kiến trúc hướng dịch vụ (SOA) và xây dựng Enterprise Service Bus (ESB), giúp các ứng dụng và hệ thống khác nhau trong doanh nghiệp giao tiếp với nhau một cách hiệu quả

Tích hợp đa dạng: BW 5 cung cấp một thư viện phong phú các "kết nối" (connector) và "khối" (palette) để kết nối với nhiều loại hệ thống khác nhau, bao gồm các ứng dụng tại chỗ (on-premise), cơ sở dữ liệu, dịch vụ web (SOAP và REST), và nhiều công nghệ khác.

Quản lý và giám sát: Nền tảng này cung cấp các công cụ quản lý và giám sát, cho phép người dùng theo dõi và quản lý hiệu suất của các quy trình đã được triển khai

1.2.2. Kiến trúc hoạt động

TIBCO BW 5 hoạt động dựa trên một số thành phần chính, bao gồm:

TIBCO Designer: Một môi trường phát triển dựa trên Eclipse, nơi người dùng tạo ra các quy trình tích hợp bằng cách sử dụng giao diện đồ họa.

TIBCO Engine: Công cụ thực thi các quy trình đã được thiết kế và triển khai.

TIBCO Administrator: Giao diện quản trị trên nền web, dùng để triển khai, quản lý, và giám sát các ứng dụng BW đã được đưa vào vận hành.

TIBCO Runtime Agent (TRA): Môi trường để triển khai các ứng dụng BW trên từng máy chủ.

1.2.3. Hạn chế của TIBCO BusinessWorks 5

Khi triển khai trên nền tảng TIBCO BusinessWorks 5 (BW5), doanh nghiệp có thể gặp một số hạn chế đáng kể, đặc biệt khi so sánh với các nền tảng tích hợp hiện

đại hơn như TIBCO BusinessWorks 6 (BW6) hoặc các giải pháp mã nguồn mở. Các hạn chế chính bao gồm:

a) Phức tạp trong quản lý và triển khai

Quản trị miền (Domain Administration): BW5 yêu cầu sử dụng một công cụ riêng biệt là Domain Utility để tạo và quản lý các miền (domain), không tích hợp trực tiếp vào môi trường phát triển như BW6. Điều này làm tăng độ phức tạp trong khâu quản trị.

Phụ thuộc vào TRA: Các máy chủ chạy ứng dụng BW5 cần phải cài đặt TIBCO Runtime Agent (TRA), một thành phần riêng biệt để triển khai và chạy các ứng dụng. Trong khi đó, các phiên bản mới hơn đã tích hợp hoặc đơn giản hóa quá trình này.

Triển khai thủ công và khó tự động hóa: BW5 thiếu các công cụ và plugin tích hợp sẵn để hỗ trợ quy trình Tích hợp và Triển khai liên tục (CI/CD) một cách hiệu quả. Nhiều quy trình triển khai phải thực hiện thủ công hoặc thông qua các công cụ kịch bản cũ như Apache Ant, gây tốn thời gian và dễ xảy ra lỗi.

b) Thiếu khả năng hỗ trợ các kiến trúc hiện đại

Không phù hợp cho Microservices: BW5 được thiết kế theo kiến trúc nguyên khối (monolithic) của ESB truyền thống. Mỗi ứng dụng BW5 là một tập hợp các quy trình lớn, điều này không phù hợp với xu hướng kiến trúc microservices hiện nay, nơi các ứng dụng được xây dựng từ các dịch vụ nhỏ, độc lập.

Triển khai trên Cloud và Container khó khăn: Mặc dù TIBCO đã cung cấp giải pháp để chạy các ứng dụng BW5 trên container, nhưng bản chất thiết kế của BW5 không hướng đến kiến trúc container. Các ứng dụng BW5 thường có kích thước lớn và không phải lúc nào cũng tối ưu cho môi trường container.

Quản lý trạng thái phức tạp: Trong các kiến trúc hiện đại, các dịch vụ thường là không trạng thái (stateless). BW5 có thể quản lý trạng thái, nhưng việc này phức tạp hơn và không tương thích với các nguyên tắc phát triển ứng dụng trên đám mây (cloud-native).

c) Hạn chế về mặt công nghệ và hiệu năng

Quản lý luồng xử lý HTTP: Việc xử lý các luồng HTTP (REST hoặc SOAP/HTTP) có thể gây ra các vấn đề về hiệu suất và lỗi "out-of-memory" nếu không được cấu hình đúng cách, đặc biệt khi có lưu lượng truy cập cao.

Giao diện phát triển lỗi thời: Môi trường thiết kế TIBCO Designer trong BW5 có giao diện người dùng cũ kỹ, không trực quan và dễ gây nhầm lẫn. So với BW6 dựa trên Eclipse với các tính năng hiện đại hơn, trải nghiệm phát triển trên BW5 kém hơn đáng kể.

1.3. TIBCO BusinessWorks 6

TIBCO BusinessWorks 6 (BW6) là phiên bản tiếp theo của TIBCO BusinessWorks 5 (BW5), được thiết kế lại hoàn toàn để đáp ứng các yêu cầu của kiến trúc tích hợp hiện đại. BW6 không chỉ kế thừa các khả năng tích hợp mạnh mẽ của BW5 mà còn tập trung vào việc hỗ trợ các công nghệ mới như điện toán đám mây (cloud computing), kiến trúc microservices và tự động hóa quy trình

1.3.1. Đặc điểm nổi bật của TIBCO BW6 so với BW5

a) Hỗ trợ kiến trúc hiện đại:

Microservices: Khác với kiến trúc nguyên khối của BW5, BW6 được thiết kế để tạo ra các dịch vụ nhỏ, độc lập (microservices). Các dịch vụ này có thể được triển khai và quản lý một cách riêng biệt, giúp tăng tính linh hoạt và khả năng mở rộng.

API-first: BW6 hỗ trợ cách tiếp cận phát triển API-first, giúp các doanh nghiệp xây dựng các ứng dụng hiện đại và dễ dàng kết nối với các thiết bị di động, web và IoT.

b) Sẵn sàng cho môi trường đám mây và container:

Triển khai linh hoạt: BW6 cho phép triển khai ứng dụng trên nhiều môi trường khác nhau, từ máy chủ tại chỗ (on-premise) cho đến các nền tảng đám mây như Docker, Kubernetes và TIBCO Cloud Integration.

Hỗ trợ CI/CD: Với các plugin Maven tích hợp, BW6 giúp tự động hóa quy trình Tích hợp và Triển khai liên tục (CI/CD), giúp giảm thiểu các thao tác thủ công và tăng tốc độ triển khai.

Quản lý đơn giản hơn: BW6 sử dụng TIBCO Enterprise Administrator (TEA) và tiện ích bwadmin để quản lý các miền (domain), không còn phụ thuộc vào Domain Utility như BW5. Việc này giúp quản lý tập trung và hiệu quả hơn.

Đơn giản hóa triển khai: BW6 không yêu cầu cài đặt TIBCO Runtime Agent (TRA) trên mỗi máy chủ, giúp đơn giản hóa quy trình triển khai và giảm thiểu các rủi ro liên quan đến phiên bản.

1.3.2. So sánh BW5 và BW6

Đặc điểm	TIBCO BusinessWorks 5 (BW5)	TIBCO BusinessWorks 6 (BW6)
Môi trường phát triển	TIBCO Designer (Eclipse cũ)	TIBCO Business Studio (Eclipse hiện đại)
Kiến trúc	ESB truyền thống, nguyên khối	Hỗ trợ microservices, API-first
Triển khai	Phụ thuộc vào TRA, phức tạp	Đơn giản hóa, hỗ trợ container (Docker, Kubernetes)
Quản lý	Sử dụng Domain Utility, phức tạp	Sử dụng TEA và bwadmin, quản lý tập trung
Tự động hóa	Hỗ trợ hạn chế, chủ yếu thủ công	Tích hợp Maven plugin, hỗ trợ CI/CD
Hỗ trợ REST	Có thể triển khai, nhưng phức tạp hơn	Hỗ trợ REST framework tích hợp, đơn giản hơn

Bảng 1.1: So sánh giữa TIBCO BW5 và TIBCO BW6

1.4. Docker và Kubernetes

1.4.1. Docker

a) Khái niệm

Docker là một nền tảng ảo hóa ở cấp độ hệ điều hành (OS-level virtualization) cho phép đóng gói, triển khai và vận hành ứng dụng trong các "container" – những môi trường độc lập chứa toàn bộ mã nguồn, thư viện và các cấu hình cần thiết. Mục tiêu của Docker là đảm bảo ứng dụng có thể hoạt động ổn định

và nhất quán trên mọi môi trường, từ máy phát triển, máy kiểm thử cho đến hệ thống triển khai thực tế (production)

b) Nguyên lý và cấu trúc

Docker hoạt động dựa trên cơ chế chia sẻ nhân hệ điều hành (kernel sharing), trong đó nhiều container có thể cùng sử dụng một hệ điều hành chủ (host OS) nhưng vẫn được cách ly hoàn toàn nhờ các công nghệ như Namespaces và Control Groups (cgroups) của Linux.

Cấu trúc của Docker bao gồm ba thành phần chính:

Docker Client: giao diện dòng lệnh hoặc API cho phép người dùng tương tác với Docker.

Docker Daemon (dockerd): dịch vụ nền chịu trách nhiệm xây dựng, chạy và quản lý container.

Docker Image & Container: image là khuôn mẫu (template) được tạo từ Dockerfile, còn container là thể hiện (instance) đang chạy của image đó.

Nhờ mô hình này, Docker có thể tạo, sao chép và triển khai các container nhanh chóng, giúp quá trình phát triển và vận hành trở nên linh hoạt và tự động hóa hơn.

c) Đặc điểm

Docker nổi bật ở khả năng mở rộng linh hoạt, cho phép triển khai hàng trăm container trên cùng một máy chủ hoặc cụm máy chủ mà vẫn đảm bảo hiệu năng. Hệ thống hạ tầng của Docker hỗ trợ việc tích hợp dễ dàng với các công cụ quản lý container như Kubernetes hoặc Openshift, giúp tự động phân phối tải, giám sát và mở rộng quy mô theo nhu cầu. Ngoài ra, Docker còn tối ưu việc sử dụng tài nguyên, giảm chi phí vận hành và rút ngắn thời gian triển khai ứng dụng so với mô hình máy ảo truyền thống.

Các yếu tố ảnh hưởng đến việc triển khai: hạ tầng, khả năng tích hợp, năng lực đội ngũ, chi phí chuyển đổi.

1.4.2. Kubernetes

a) Khái niệm

Kubernetes (thường được viết tắt là K8s) là một hệ thống mã nguồn mở do Google phát triển và hiện được duy trì bởi Cloud Native Computing Foundation (CNCF), dùng để tự động hóa việc triển khai, mở rộng và quản lý các ứng dụng container. Kubernetes đóng vai trò như một nền tảng điều phối (orchestration platform) giúp quản lý hàng loạt container Docker hoặc các container runtime khác, đảm bảo tính sẵn sàng cao, khả năng mở rộng và cân bằng tải cho ứng dụng trong môi trường phân tán

b) Nguyên lý và cấu trúc

Kubernetes hoạt động theo mô hình kiến trúc chủ–tớ (master–node). Trong đó, Control Plane (Master) chịu trách nhiệm điều phối toàn bộ hệ thống, còn Worker Nodes là nơi thực thi các container.

Các thành phần chính của Kubernetes bao gồm:

API Server: cổng giao tiếp trung tâm cho phép người dùng và các thành phần khác tương tác với hệ thống.

Scheduler: đảm nhận việc phân bổ container đến các node phù hợp.

Controller Manager: giám sát và duy trì trạng thái mong muốn của hệ thống.

Kubelet: tiến trình chạy trên mỗi node để quản lý container.

Kube Proxy: thực hiện định tuyến và cân bằng tải trong mạng nội bộ của cluster.

Kubernetes sử dụng Pod làm đơn vị triển khai nhỏ nhất – mỗi pod có thể chứa một hoặc nhiều container hoạt động cùng nhau. Thông qua các khái niệm như Deployment, Service, ReplicaSet, và Namespace, Kubernetes cung cấp khả năng quản lý vòng đời ứng dụng một cách tự động, ổn định và có khả năng phục hồi cao.

c) Đặc điểm

Kubernetes nổi bật với khả năng mở rộng quy mô động (auto-scaling), cân bằng tải (load balancing) và tự phục hồi (self-healing), giúp đảm bảo ứng dụng luôn

hoạt động ổn định trong điều kiện thay đổi tài nguyên hoặc sự cố. Hệ thống cho phép triển khai đa nền tảng trên hạ tầng vật lý, máy ảo hoặc đám mây (Cloud/Hybrid Cloud), đồng thời dễ dàng tích hợp với các công cụ CI/CD để tự động hóa quy trình triển khai phần mềm. Nhờ khả năng quản lý hàng nghìn container và hỗ trợ đa môi trường, Kubernetes đã trở thành tiêu chuẩn công nghệ cho việc triển khai ứng dụng theo kiến trúc microservices hiện nay.

1.5. Kết luận chương 1

Trong kiến trúc hướng dịch vụ (SOA – Service-Oriented Architecture), các ứng dụng được chia nhỏ thành nhiều dịch vụ độc lập, mỗi dịch vụ đảm nhận một chức năng cụ thể và giao tiếp với nhau thông qua các giao thức chuẩn (thường là HTTP hoặc gRPC). Để đảm bảo các dịch vụ này có thể triển khai, vận hành và mở rộng một cách hiệu quả, việc áp dụng các công nghệ container hóa như Docker và Kubernetes đóng vai trò đặc biệt quan trọng

Docker giúp đóng gói từng dịch vụ của SOA thành một container riêng biệt, chứa toàn bộ mã nguồn, cấu hình và thư viện cần thiết. Nhờ đó, mỗi dịch vụ có thể được triển khai một cách độc lập, đồng nhất và dễ tái sử dụng trên các môi trường khác nhau. Việc container hóa còn giúp giảm thiểu xung đột môi trường, rút ngắn thời gian triển khai, đồng thời hỗ trợ tự động hóa trong quá trình phát triển và kiểm thử

Trong khi đó, Kubernetes đóng vai trò là nền tảng điều phối và quản lý các dịch vụ container hóa trong toàn bộ hệ thống SOA. Với khả năng tự động cân bằng tải, mở rộng quy mô, tự phục hồi khi xảy ra lỗi và quản lý vòng đời dịch vụ, Kubernetes đảm bảo các thành phần SOA luôn duy trì tính sẵn sàng, khả năng mở rộng và độ tin cậy cao. Bên cạnh đó, Kubernetes còn hỗ trợ triển khai theo mô hình microservices, vốn là sự phát triển tự nhiên và hiện đại hóa của SOA, giúp tổ chức dễ dàng tích hợp, cập nhật hoặc thay thế các dịch vụ mà không ảnh hưởng đến toàn bộ hệ thống

Tổng hợp lại, sự kết hợp giữa Docker và Kubernetes đã tạo nên một hạ tầng triển khai linh hoạt, tự động hóa và tối ưu hóa cho kiến trúc SOA. Docker đảm bảo

tính di động và độc lập của từng dịch vụ, còn Kubernetes đảm bảo khả năng quản lý, mở rộng và duy trì hệ thống ở quy mô lớn. Đây chính là nền tảng kỹ thuật quan trọng giúp SOA vận hành hiệu quả trong môi trường công nghệ hiện đại, đặc biệt là trong các mô hình DevOps và Cloud Computing.

CHƯƠNG II: NGHIÊN CỨU TRIỂN KHAI HỆ THỐNG SOA TRONG NGÂN HÀNG

Trong bối cảnh ngành ngân hàng đang đẩy mạnh chuyển đổi số và hiện đại hóa hạ tầng công nghệ thông tin, kiến trúc định hướng dịch vụ (Service-Oriented Architecture – SOA) đóng vai trò quan trọng trong việc xây dựng các hệ thống linh hoạt, có khả năng mở rộng và dễ dàng tích hợp với các nền tảng mới. Chương này tập trung nghiên cứu thực trạng triển khai hệ thống SOA trong các ngân hàng Việt Nam, với trọng tâm phân tích hai mô hình triển khai phổ biến là TIBCO BusinessWorks 5 (BW5) và TIBCO BusinessWorks 6 (BW6).

Trên cơ sở tổng hợp lý thuyết và kết quả khảo sát thực tế, chương này đánh giá toàn diện ưu điểm, hạn chế và các vấn đề tồn tại trong mô hình BW5, đồng thời làm rõ những cải tiến nổi bật của BW6 trong việc hỗ trợ phát triển, quản lý và vận hành các dịch vụ SOA. Tiếp đó, đề án xem xét tiềm năng ứng dụng công nghệ container hóa và nền tảng Kubernetes trong triển khai SOA, như một hướng tiếp cận mới giúp nâng cao hiệu quả sử dụng tài nguyên, khả năng mở rộng và độ tin cậy của hệ thống.

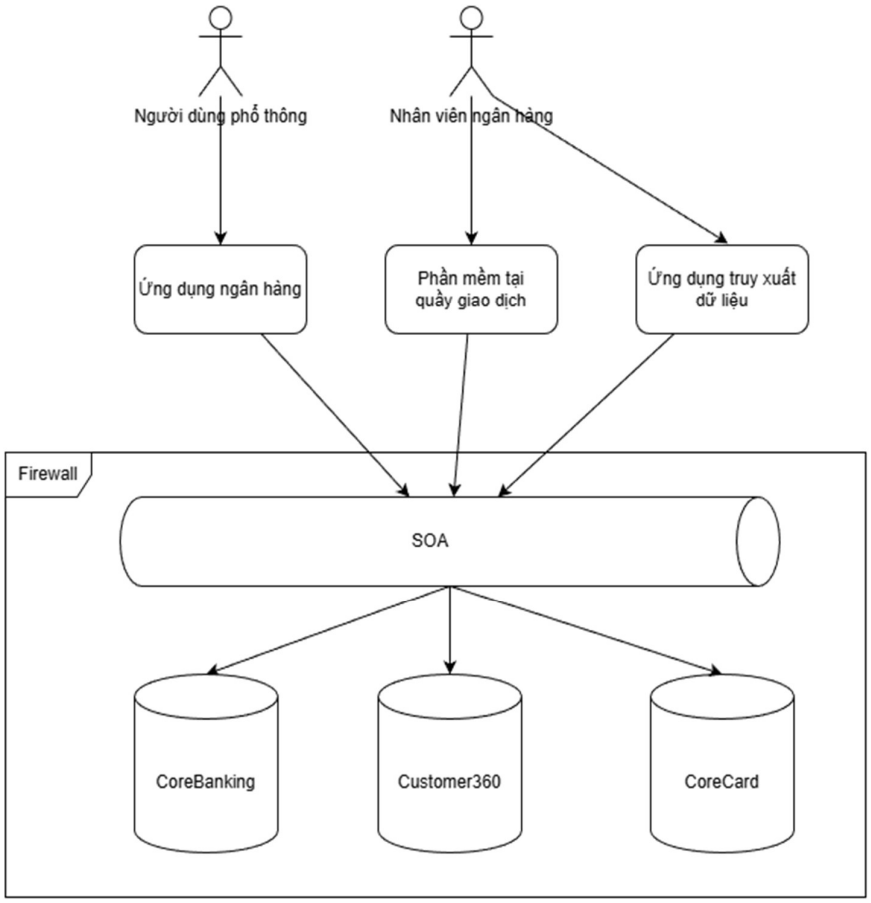
Thông qua quá trình phân tích, chương này hướng đến việc làm rõ tính cấp thiết của việc chuyển đổi sang mô hình triển khai SOA hiện đại, dựa trên nền tảng công nghệ tiên tiến, nhằm đáp ứng yêu cầu ngày càng cao về tốc độ, an toàn và tính linh hoạt trong hoạt động của các ngân hàng hiện nay.

2.1. Tổng quan về mô hình SOA trong ngân hàng

Trong môi trường ngân hàng hiện đại, nơi các hệ thống nghiệp vụ ngày càng đa dạng và khối lượng giao dịch tăng nhanh, việc xây dựng một kiến trúc công nghệ linh hoạt, có khả năng tích hợp cao là yêu cầu tất yếu. Kiến trúc hướng dịch vụ (SOA) được xem là giải pháp hiệu quả giúp các ngân hàng chuẩn hóa quy trình trao đổi dữ liệu, tách biệt các chức năng nghiệp vụ và nâng cao khả năng tái sử dụng dịch vụ.

SOA cho phép các hệ thống lõi của ngân hàng (như Core Banking, Customer360, Core Card) hoạt động độc lập nhưng vẫn có thể kết nối với nhau thông qua một lớp trung gian tích hợp. Nhờ đó, các ứng dụng phục vụ người dùng – bao

gồm ứng dụng ngân hàng điện tử, phần mềm tại quầy giao dịch, hay các công cụ truy xuất dữ liệu nội bộ – đều có thể truy cập dịch vụ một cách thống nhất và an toàn, mà không cần quan tâm đến sự khác biệt về nền tảng kỹ thuật phía dưới …



Hình 2.1: Tổng quan về hệ thống SOA trong ngân hàng

Hình trên minh họa cấu trúc tổng thể của hệ thống ngân hàng hiện nay. Trong mô hình này, SOA đóng vai trò như lớp trung gian tích hợp (Integration Layer), kết nối giữa các ứng dụng nghiệp vụ ở tầng người dùng và các hệ thống lõi (Core Systems) ở tầng hạ tầng.

Ở tầng trên cùng, người dùng bao gồm khách hàng và nhân viên ngân hàng tương tác với hệ thống thông qua nhiều kênh khác nhau như ứng dụng ngân hàng điện tử, phần mềm tại quầy giao dịch, hay ứng dụng truy xuất dữ liệu nội bộ. Các ứng dụng này gửi yêu cầu nghiệp vụ đến lớp SOA để truy cập thông tin và dịch vụ được cung cấp bởi hệ thống lõi.

Bên trong khu vực được bảo vệ bởi Firewall, lớp SOA đảm nhận chức năng điều phối, chuyển đổi và định tuyến dữ liệu giữa các thành phần, đảm bảo việc giao tiếp diễn ra an toàn, ổn định và tuân thủ các chính sách bảo mật của ngân hàng. Lớp này giúp chuẩn hóa cách thức tích hợp, giảm thiểu sự phụ thuộc giữa các hệ thống và tạo điều kiện cho việc mở rộng, bảo trì hoặc thay thế thành phần mà không ảnh hưởng đến toàn bộ hệ thống.

Tầng dưới cùng bao gồm các hệ thống lõi như Core Banking, Customer360, Core Card, … là nơi xử lý các nghiệp vụ trọng yếu như quản lý tài khoản, giao dịch, thông tin khách hàng và dữ liệu thẻ. Thông qua lớp SOA, các hệ thống này có thể chia sẻ và cung cấp dữ liệu cho nhiều kênh ứng dụng khác nhau một cách nhất quán.

Nhờ cơ chế hoạt động này, SOA giúp ngân hàng hình thành một hệ sinh thái dịch vụ thống nhất, linh hoạt và dễ mở rộng, đồng thời nâng cao khả năng kết nối với các nền tảng công nghệ hiện đại như microservices, API Gateway, cloud-native hay Open Banking, góp phần thúc đẩy quá trình chuyển đổi số toàn diện trong ngành ngân hàng.

2.2. Thực trạng triển khai trục tích hợp bằng TIBCO BW5

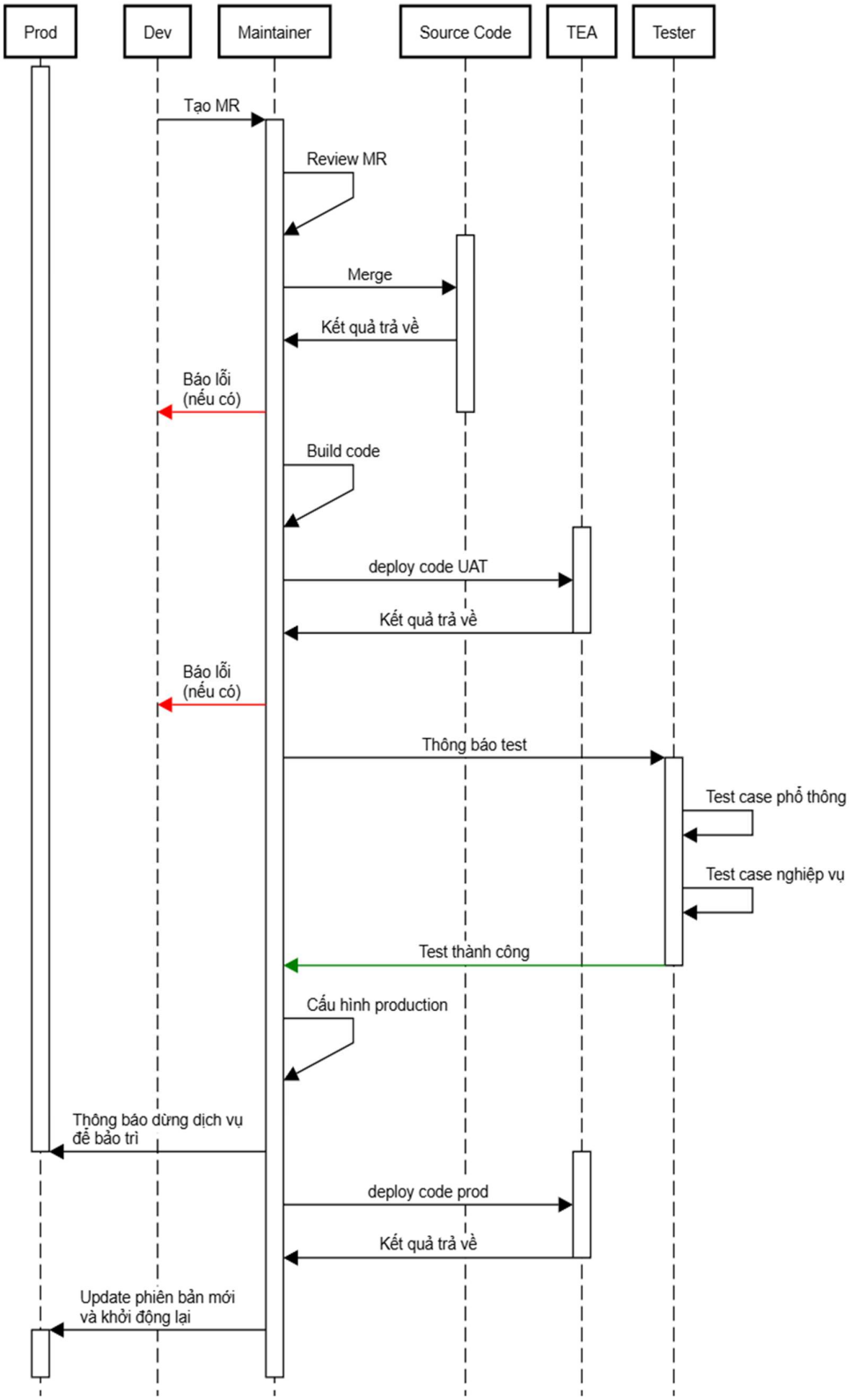
2.2.1 Thực trạng

Quy trình triển khai dịch vụ trong môi trường TIBCO BusinessWorks 5 (BW5) được vận hành theo cơ chế thủ công nhiều bước và đòi hỏi sự phối hợp giữa nhiều vai trò khác nhau, bao gồm nhà phát triển (Dev), người duy trì hệ thống (Maintainer), bộ phận kiểm thử (Tester) và môi trường quản trị TEA (TIBCO Enterprise Administrator). Toàn bộ quá trình triển khai, kiểm thử, và chuyển đổi môi trường đều được thực hiện tuần tự, không có khả năng tự động hóa.

Cụ thể, quy trình bắt đầu khi lập trình viên tạo Merge Request (MR) để yêu cầu hợp nhất mã nguồn. Maintainer thực hiện review, merge và build code trên hệ thống quản lý mã nguồn (thường là GitLab hoặc SVN). Sau khi build thành công, mã nguồn được triển khai (deploy) lên môi trường UAT (User Acceptance Test) thông qua công cụ quản trị TEA.

Tại môi trường UAT, kiểm thử viên (Tester) tiến hành hai cấp độ kiểm thử gồm test case phổ thông (kiểm tra tính ổn định, giao diện, và kết nối dịch vụ) và test case nghiệp vụ (kiểm tra logic xử lý, dữ liệu và luồng giao dịch thực tế). Nếu phát hiện lỗi, kết quả được phản hồi lại cho nhóm phát triển để chỉnh sửa và gửi lại MR mới. Quy trình này có thể phải lặp lại nhiều lần cho đến khi tất cả các trường hợp kiểm thử đạt yêu cầu.

Khi dịch vụ được kiểm thử thành công, Maintainer tiến hành cấu hình môi trường production, thông báo dừng dịch vụ bảo trì, sau đó triển khai mã nguồn chính thức (deploy code prod). Cuối cùng, hệ thống được cập nhật phiên bản mới và khởi động lại để đưa dịch vụ vào vận hành.



Hình 1.2: Hiện trạng triển khai dịch vụ khi sử dụng TIBCO BW5

2.2.2. Hạn chế

Mặc dù quy trình này đảm bảo tính kiểm soát chặt chẽ và giúp giảm rủi ro trong từng giai đoạn triển khai, song trong thực tế vận hành tại các ngân hàng, mô hình này bộc lộ nhiều hạn chế rõ rệt:

Thứ nhất, quy trình triển khai hoàn toàn phụ thuộc vào thao tác thủ công của con người, từ bước merge, build, deploy đến cấu hình môi trường. Điều này khiến thời gian triển khai kéo dài, dễ phát sinh lỗi vận hành và không đảm bảo tính nhất quán giữa các môi trường.

Thứ hai, thiếu tích hợp CI/CD khiến quy trình không thể tự động hóa kiểm thử hay triển khai. Mỗi thay đổi dù nhỏ cũng phải qua nhiều bước xác nhận, làm chậm tiến độ phát hành dịch vụ và không phù hợp với mô hình phát triển Agile/DevOps.

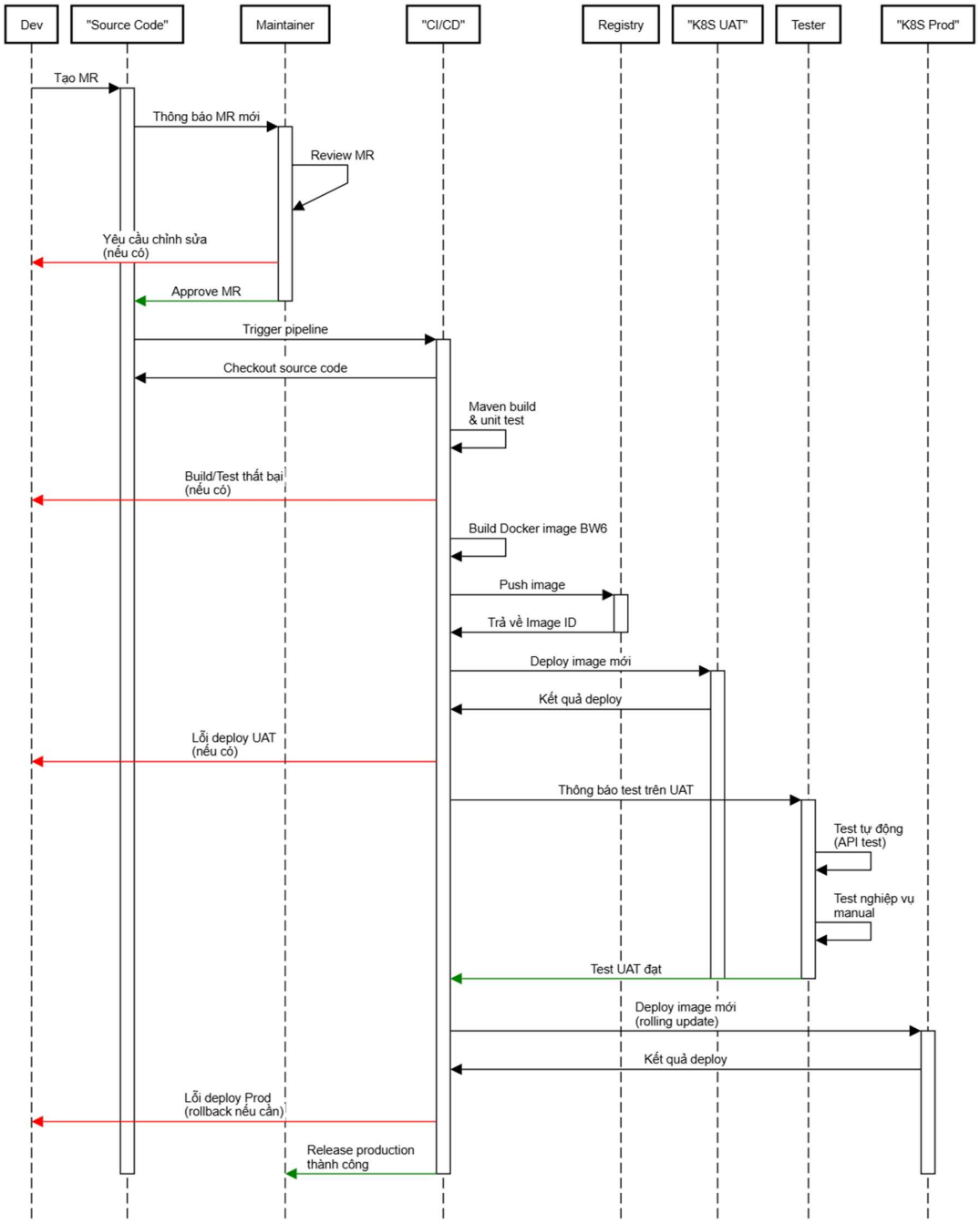
Thứ ba, việc kiểm thử UAT phụ thuộc hoàn toàn vào thao tác của tester, không có pipeline tự động ghi nhận kết quả hoặc rollback khi thất bại. Điều này làm giảm tính chủ động trong quản lý rủi ro và tăng khối lượng công việc thủ công cho cả đội phát triển lẫn vận hành.

Cuối cùng, trong môi trường production, BW5 không có khả năng auto-scaling hoặc auto-recovery. Khi xảy ra lỗi, dịch vụ phải được dừng thủ công, khởi động lại hoặc cập nhật phiên bản mới, gây gián đoạn dịch vụ (downtime) và tăng áp lực lên đội vận hành.

Những bất tiện này cho thấy TIBCO BW5 không còn phù hợp với yêu cầu tích hợp hiện đại, đặc biệt trong bối cảnh các ngân hàng đang chuyển dịch mạnh sang mô hình container hóa, CI/CD, và microservices. Việc chuyển đổi sang TIBCO BW6 hoặc kiến trúc triển khai dựa trên Kubernetes và Docker là bước đi cần thiết nhằm nâng cao tính linh hoạt, tự động hóa và khả năng mở rộng động cho hệ thống SOA ngân hàng.

2.3. Đề xuất mô hình triển khai trục tích hợp bằng TIBCO BW6 kết hợp Kubernetes

2.3.1. Mô hình đề xuất



Hình 2.2: Mô hình triển khai dịch vụ khi sử dụng TIBCO BW6

Sau giai đoạn vận hành trục SOA bằng TIBCO BW5 với nhiều hạn chế về khả năng tự động hóa và mở rộng, các ngân hàng đã từng bước chuyển đổi sang TIBCO BusinessWorks 6 (BW6) – nền tảng thế hệ mới được thiết kế phù hợp với xu hướng container hóa, DevOps và CI/CD. Mô hình triển khai BW6 thể hiện trong hình dưới đây cho thấy sự thay đổi toàn diện từ cách quản lý mã nguồn, quy trình build, kiểm thử cho đến triển khai và giám sát dịch vụ.

Trong mô hình này, toàn bộ quy trình phát triển và triển khai dịch vụ được tự động hóa thông qua pipeline CI/CD. Khi nhà phát triển (Dev) tạo yêu cầu hợp nhất mã nguồn (Merge Request – MR) trên hệ thống quản lý mã nguồn (như GitLab hoặc Bitbucket), Maintainer sẽ tiến hành review và approve MR. Ngay sau khi MR được chấp thuận, pipeline CI/CD được tự động kích hoạt để thực hiện chuỗi tác vụ bao gồm: checkout source code, build và unit test bằng Maven, đóng gói ứng dụng TIBCO BW6 dưới dạng Docker image, và đẩy image lên Registry trung tâm.

Từ Registry, pipeline tiếp tục triển khai image mới lên môi trường UAT (User Acceptance Test) trong cụm Kubernetes (K8S UAT). Tại đây, các bài kiểm thử tự động (API test) và kiểm thử nghiệp vụ thủ công (manual test) được thực hiện nhằm đảm bảo chất lượng dịch vụ trước khi phát hành. Khi toàn bộ kiểm thử đạt yêu cầu, pipeline sẽ tự động triển khai image mới sang môi trường sản xuất (K8S Prod) bằng cơ chế rolling update, đảm bảo không gián đoạn dịch vụ trong quá trình nâng cấp.

2.3.2. Đặc điểm nổi bật

So với thế hệ BW5, mô hình triển khai BW6 mang lại nhiều ưu điểm vượt trội:

Tự động hóa toàn bộ quy trình phát triển – kiểm thử – triển khai, rút ngắn thời gian release và giảm thiểu lỗi do thao tác thủ công.

Độ phụ thuộc vào maintainer giảm đi đáng kể

Tích hợp chặt chẽ với công cụ DevOps như Jenkins, GitLab CI,... giúp thiết lập pipeline CI/CD chuẩn hóa.

Hỗ trợ containerization với Docker và Kubernetes, cho phép mở rộng linh hoạt (auto-scaling), tự phục hồi (auto-recovery), và tối ưu tài nguyên hệ thống.

Cơ chế logging và monitoring tập trung thông qua các công cụ như Prometheus, Grafana hoặc ELK stack, giúp giám sát toàn bộ vòng đời dịch vụ.

Nhờ các cải tiến này, TIBCO BW6 trở thành nền tảng tích hợp trung tâm (integration backbone) trong kiến trúc SOA hiện đại của ngân hàng, đáp ứng tốt yêu cầu về tốc độ, linh hoạt, tính sẵn sàng cao và khả năng mở rộng động – những yếu tố cốt lõi trong tiến trình chuyển đổi số và hiện đại hóa hạ tầng công nghệ ngân hàng.

2.3.3. Khó khăn và hạn chế

Mặc dù mô hình triển khai trục tích hợp bằng TIBCO BusinessWorks 6 (BW6) mang lại nhiều lợi ích rõ rệt về mặt kỹ thuật và vận hành, song trong giai đoạn thí điểm chuyển đổi từ BW5 sang BW6, các ngân hàng vẫn phải đối mặt với nhiều khó khăn và thách thức thực tế.

Thứ nhất, thiếu nhân lực có chuyên môn về container và Kubernetes là trở ngại lớn nhất trong quá trình triển khai. Do hạ tầng mới dựa hoàn toàn trên công nghệ container hóa, việc thiết lập, cấu hình và giám sát hệ thống đòi hỏi đội ngũ kỹ thuật có kiến thức sâu về DevOps, CI/CD, và vận hành cụm Kubernetes. Tuy nhiên, nguồn nhân lực này tại nhiều ngân hàng hiện còn hạn chế, dẫn đến tiến độ triển khai chậm và phụ thuộc nhiều vào đơn vị tư vấn bên ngoài.

Thứ hai, chi phí đầu tư ban đầu lớn cũng là yếu tố gây cản trở cho việc áp dụng rộng rãi. Việc chuyển đổi sang BW6 không chỉ đòi hỏi đầu tư vào phần mềm bản quyền, mà còn cần nâng cấp hạ tầng máy chủ, thiết bị lưu trữ, và hệ thống giám sát mới. Bên cạnh đó, do hệ thống BW5 hiện tại vẫn vận hành ổn định, nhiều ngân hàng còn tâm lý e ngại thay đổi, lo ngại rủi ro khi di chuyển các dịch vụ nghiệp vụ quan trọng sang nền tảng mới.

Thứ ba, thiếu mô hình chuẩn và quy trình triển khai rõ ràng khiến việc chuyển đổi gặp khó khăn trong giai đoạn đầu. Mỗi đơn vị triển khai thường phải tự thiết kế pipeline, quy trình đóng gói và chiến lược migrate riêng, dẫn đến thiếu tính đồng nhất và khó khăn trong việc quản lý, bảo trì lâu dài.

Nhìn chung, dù mô hình BW6 mang lại nền tảng kỹ thuật hiện đại hơn, quá trình chuyển đổi từ BW5 sang BW6 vẫn cần được thực hiện từng bước, có lộ trình và

hỗ trợ đào tạo bài bản, nhằm đảm bảo tính ổn định của hệ thống ngân hàng trong quá trình hiện đại hóa trục tích hợp SOA.

2.4. Kết luận chương 2

Chương đã hệ thống hóa các khái niệm cốt lõi về SOA trong ngân hàng, chỉ ra vai trò quan trọng của trục tích hợp trong việc kết nối các hệ thống nghiệp vụ, chuẩn hóa giao tiếp, nâng cao khả năng tái sử dụng dịch vụ và hỗ trợ chiến lược chuyển đổi số. Trên cơ sở đó, đề án đi sâu phân tích thực trạng triển khai trục tích hợp dựa trên TIBCO BusinessWorks 5, làm rõ những điểm mạnh nhưng đồng thời cũng nhận diện những hạn chế còn tồn tại như kiến trúc đơn khối, khó mở rộng, khó giám sát, chi phí vận hành cao và chưa đáp ứng tốt yêu cầu linh hoạt trong bối cảnh số lượng giao dịch tăng nhanh.

Từ các vấn đề thực tiễn, chương tiếp tục đề xuất mô hình triển khai trục tích hợp mới dựa trên TIBCO BusinessWorks 6 kết hợp nền tảng Kubernetes. Mô hình này thể hiện nhiều ưu điểm nổi bật như khả năng triển khai dạng container, tự động mở rộng, CI/CD, cải thiện hiệu năng, tăng tính linh hoạt và đảm bảo tính sẵn sàng cao của hệ thống. Tuy nhiên, chương cũng chỉ ra những khó khăn và thách thức cần cân nhắc khi triển khai, bao gồm yêu cầu đầu tư hạ tầng, năng lực đội ngũ, thay đổi quy trình vận hành và các rủi ro chuyển đổi hệ thống.

Tổng thể, Chương II đã xây dựng được bức tranh toàn diện về hiện trạng và định hướng phát triển hệ thống SOA tại ngân hàng. Đây là cơ sở quan trọng để Chương III tiếp tục phân tích, đánh giá và đề xuất giải pháp triển khai chi tiết, hoàn thiện kiến trúc trục tích hợp theo định hướng của BIDV, linh hoạt và bền vững trong tương lai.

CHƯƠNG III: TRIỂN KHAI HỆ THỐNG SOA TRÊN NỀN TẢNG KUBERNETES TẠI NGÂN HÀNG BIDV

Sau khi nghiên cứu tổng quan kiến trúc SOA trong lĩnh vực ngân hàng và phân tích mô hình triển khai trục tích hợp bằng TIBCO BusinessWorks 5 (BW5) và TIBCO BusinessWorks 6 (BW6), chương này tập trung trình bày mô hình triển khai thực tế hệ thống SOA trên nền tảng Kubernetes tại Ngân hàng TMCP Đầu tư và Phát triển Việt Nam (BIDV). Đây là một bước tiến quan trọng trong quá trình hiện đại hóa hạ tầng tích hợp và chuyển đổi sang kiến trúc điện toán đám mây của ngân hàng.

Việc áp dụng Kubernetes không chỉ giúp BIDV tự động hóa quy trình triển khai và quản lý dịch vụ, mà còn mang lại khả năng mở rộng linh hoạt, tối ưu tài nguyên, và tăng tính sẵn sàng (high availability) cho toàn bộ hệ thống SOA. Bằng cách kết hợp nền tảng container hóa (Docker) và pipeline CI/CD tự động, BIDV hướng đến mục tiêu xây dựng một hệ sinh thái tích hợp hiện đại, thống nhất và có khả năng phục hồi cao – đáp ứng yêu cầu vận hành 24/7 và khối lượng giao dịch ngày càng lớn.

Chương này sẽ trình bày chi tiết kiến trúc tổng thể của hệ thống SOA trên Kubernetes, bao gồm cấu trúc hạ tầng, cơ chế triển khai, quản lý tài nguyên, và giám sát dịch vụ. Tiếp đó, nội dung sẽ đi sâu vào quy trình triển khai thực tế tại BIDV, đánh giá hiệu quả đạt được cũng như những khó khăn, thách thức và kinh nghiệm thực tiễn rút ra trong quá trình triển khai hệ thống.

3.1. Tổng quan kiến trúc

Hình dưới đây mô tả kiến trúc tổng thể của hệ thống tích hợp SOA được triển khai trên nền tảng OpenShift (Kubernetes) tại Ngân hàng BIDV, với trọng tâm là việc ứng dụng TIBCO BusinessWorks 6 (BW6) trong môi trường container hóa để đảm bảo khả năng mở rộng, linh hoạt và quản lý tập trung.

Toàn bộ hệ thống được chia thành nhiều namespace độc lập trong cụm OpenShift, trong đó Namespace SOA chịu trách nhiệm xử lý các dịch vụ nghiệp vụ chung, còn Namespace SOA 2.0 – CDE (Cardholder Data Environment) được tách

biệt để phục vụ các nghiệp vụ liên quan đến thẻ quốc tế, đảm bảo tính bảo mật và tuân thủ các chuẩn an toàn dữ liệu.

Ở tầng giao tiếp, các ứng dụng đầu cuối (Request Application) như Customer App, Teller App, CRM, DKTT,… gửi yêu cầu thông qua các giao thức SOAP/JMS hoặc HTTP/HTTPS đến hàng đợi tin nhắn (TIBCO EMS Queue). Queue này đóng vai trò trung gian truyền thông bất đồng bộ, giúp cân bằng tải và giảm độ phụ thuộc trực tiếp giữa các ứng dụng và dịch vụ xử lý.

Bên trong namespace SOA, hệ thống được chia thành hai nhóm chính:

Core Module – bao gồm các dịch vụ lõi như Core Profile Request/Reply RO và Core Profile Request/Reply RW, đảm nhận việc truy xuất và cập nhật thông tin hồ sơ khách hàng, đồng thời kết nối trực tiếp với hệ thống Core Banking thông qua các máy chủ Primary Server và Report Server.

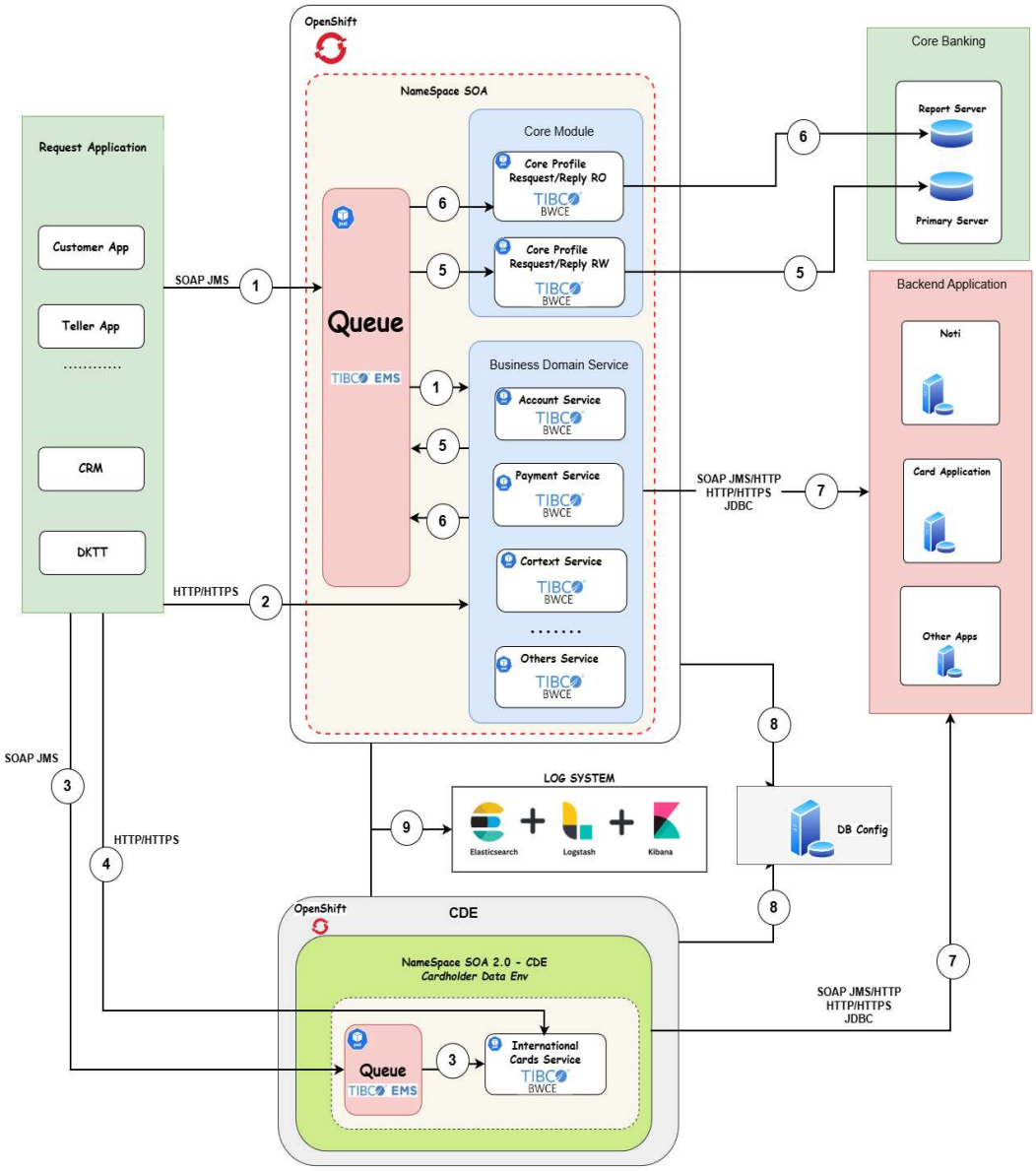
Business Domain Service – bao gồm các dịch vụ nghiệp vụ như Account Service, Payment Service, Contract Service, và các dịch vụ khác. Các dịch vụ này được triển khai dưới dạng container độc lập, sử dụng TIBCO BW6 CE làm công cụ xử lý và tích hợp dữ liệu từ nhiều hệ thống backend (Noti, Hệ thống thẻ, DB Config v.v.) qua các giao thức SOAP, JMS, HTTP/HTTPS hoặc JDBC.

Một thành phần quan trọng trong kiến trúc là hệ thống giám sát và ghi log tập trung (LOG SYSTEM) được xây dựng trên nền tảng ELK Stack (Elasticsearch – Logstash – Kibana). Thành phần này cho phép thu thập, phân tích và trực quan hóa log từ tất cả các container trong cụm Kubernetes, giúp bộ phận vận hành dễ dàng theo dõi, giám sát và xử lý sự cố trong thời gian thực.

Ngoài ra, namespace SOA 2.0 – CDE được triển khai riêng biệt để đảm bảo an toàn dữ liệu thẻ (PCI DSS compliance). Tại đây, các dịch vụ thẻ quốc tế (International Cards Service) cũng hoạt động thông qua TIBCO EMS Queue tương tự như namespace SOA, nhưng được cô lập hoàn toàn về mặt tài nguyên và mạng để tăng tính bảo mật.

Nhờ cấu trúc phân tách rõ ràng giữa các tầng và dịch vụ, mô hình này mang lại tính linh hoạt, khả năng mở rộng động và độ sẵn sàng cao, đồng thời cho phép

BIDV dễ dàng mở rộng thêm các dịch vụ mới, triển khai rolling update không gián đoạn, và quản lý tập trung toàn bộ hệ thống SOA trên một nền tảng thống nhất.

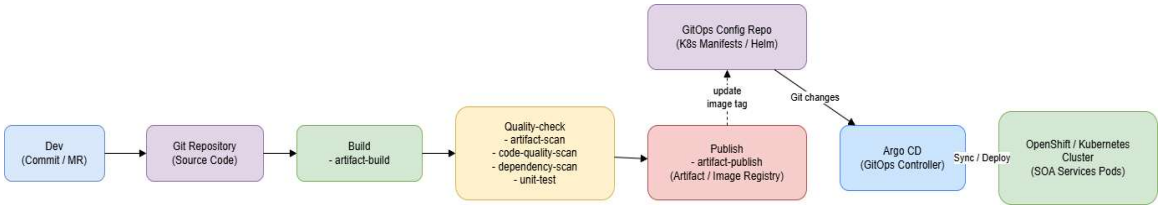


Hình 3.1: Tổng quan kiến trúc SOA

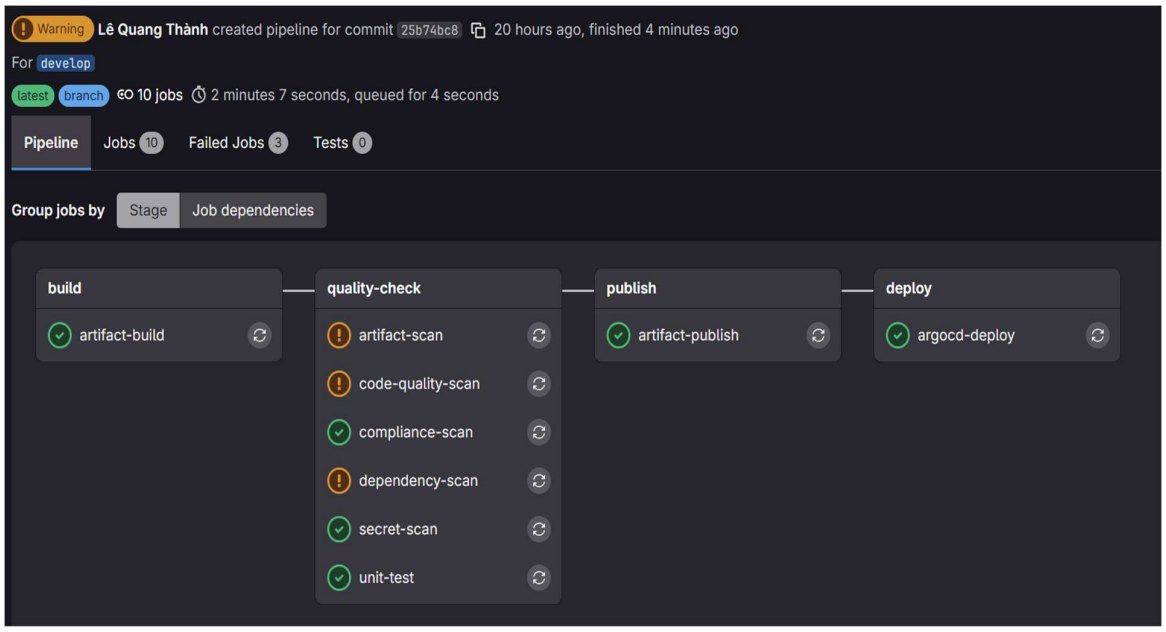
STT	Nguồn	Giao thức/Chuẩn kết nối	Đích	Mô tả
1	Request Application	SOAP JMS	SOA	Các ứng dụng tích hợp SOA qua giao thức SOAP over JMS
2	Request Application	RESTful API	SOA	Kết nối tới SOA theo chuẩn RESTful API, xác thực qua hệ thống QLToken
3	Request Application	SOAP JMS	SOA	Các ứng dụng CDE tích hợp SOA qua giao thức SOAP over JMS. Phục vụ các chương trình, nghiệp vụ xử lý yêu cầu số thẻ tường minh
4	Request Application	RESTful API	SOA	Các ứng dụng CDE tích hợp SOA. Phục vụ các chương trình, nghiệp vụ xử lý yêu cầu số thẻ tường minh theo chuẩn RESTful API, xác thực qua hệ thống QLToken
5	SOA	SOAP JMS JDBC/MRPC	Core Primary	B1. Push dữ liệu để gửi vào queue xử lý RW khi gọi vào Core Profile B2. Pull dữ liệu để thực hiện gọi vào Primary Server của Core Profile
6	SOA	SOAP JMS JDBC/MRPC	Core Report	B1. Push dữ liệu để gửi vào queue xử lý RO khi gọi vào Core Profile B2. Pull dữ liệu để thực hiện gọi vào Report Server của Core Profile
7	SOA	SOAP JMS/HTTP RESTful API JDBC	Backend Application	SOA tích hợp đến các ứng dụng nội bộ BIDV theo phương thức ứng dụng cung cấp
8	SOA	JDBC	Config DB	Kết nối Database
9	SOA	TCP/IP (Kafka Queue)	Log tập trung	Forwarding log từ hệ thống SOA vào Kafka vào hệ thống log tập trung ELK

Bảng 3.1: Bảng mô tả các kết nối giữa SOA 2.0 và các hệ thống liên quan

3.2. Mô hình CI/CD



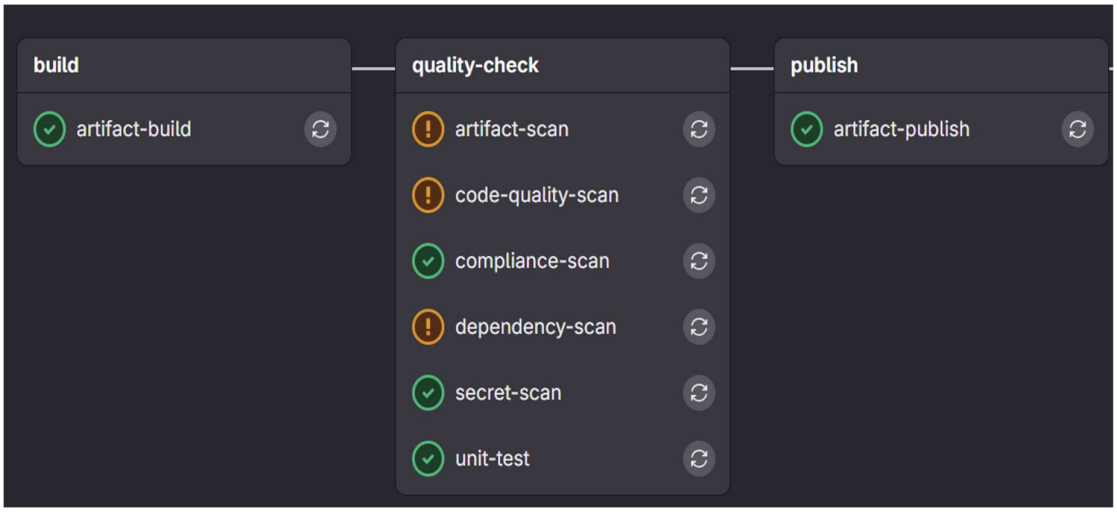
Hình 3.2: Mô hình CI/CD đơn giản



Hình 3.3: Quy trình CI/CD trên Gitlab

3.2.1. Quy trình CI

Pha CI (Continuous Integration) bắt đầu từ thời điểm lập trình viên (Dev) thực hiện commit hoặc tạo Merge Request (MR) lên Git Repository chứa mã nguồn dịch vụ SOA. Mỗi thay đổi trên nhánh được cấu hình sẽ tự động kích hoạt pipeline CI, bảo đảm rằng mọi phiên bản mã đều phải đi qua cùng một quy trình kiểm tra trước khi được chấp nhận.



Hình 3.4: Quy trình CI trên Gitlab

Sau khi nhận tín hiệu từ Git, pipeline chuyển sang giai đoạn Build – artifact-build. Ở bước này, hệ thống CI (GitLab CI/Jenkins) tự động biên dịch mã nguồn, chạy các bước đóng gói cần thiết (ví dụ: Maven/Gradle build đối với Java, build BW6 app, tạo Docker image, v.v.) và sinh ra một artifact triển khai được. Artifact này mang phiên bản (tag) cụ thể, giúp truy vết chính xác giữa mã nguồn và gói triển khai sau này. Giai đoạn build giúp phát hiện sớm các lỗi cú pháp, lỗi phụ thuộc (missing dependency) và đảm bảo mã có thể biên dịch thành công trên môi trường chuẩn.

```
def podLabel = "freeops-7_0_build_maven-image-build"

podTemplate(label: podLabel, yaml: POD_YAML) {
  node(podLabel) {

    // giữ đúng path workspace giống log
    ws('/home/jenkins/agent/builds') {


      stage('Init') {···
      }

      stage('Warnup') {···
      }

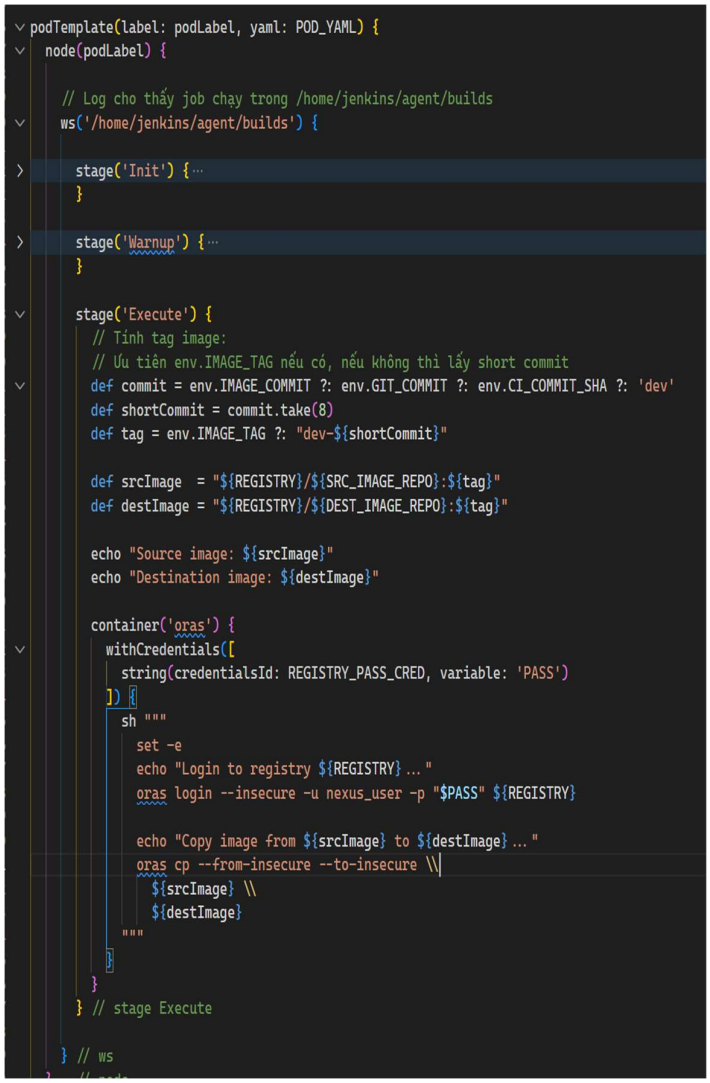
      stage('Execute - Maven Build') {···
      }

      stage('Execute - Build & Push Image') {···
      }
    } // ws
  }   // node
}
```

Hình 3.5: Logic code trong quy trình CI

Tiếp theo, artifact được đưa vào khối Quality-check, nơi tập hợp nhiều job kiểm tra chất lượng được thực thi tự động. Các bước như artifact-scan, code-quality-scan, dependency-scan giúp quét lỗ hổng bảo mật trong bản build và các thư viện phụ thuộc, kiểm tra chất lượng mã (code smell, bug, duplication), cũng như mức độ tuân thủ chuẩn coding và quy định nội bộ (compliance-scan). Song song đó, secret-scan được sử dụng để phát hiện nguy cơ rò rỉ thông tin nhạy cảm (mật khẩu, khóa truy cập…) trong repo, còn unit-test chạy bộ kiểm thử đơn vị nhằm xác nhận các chức năng cốt lõi vẫn hoạt động đúng sau mỗi thay đổi. Chỉ khi toàn bộ kiểm tra quan trọng đạt yêu cầu, pipeline mới được phép chuyển sang bước kế tiếp.

Ở cuối pha CI là giai đoạn Publish – artifact-publish, nơi artifact đã "qua kiểm định" được đẩy lên kho lưu trữ trung tâm (artifact registry hoặc container registry). Việc chỉ publish các artifact đã vượt qua đầy đủ các bước build và quality-check giúp bảo đảm rằng mọi phiên bản được sử dụng cho triển khai (CD) đều đã được kiểm chứng về chất lượng và bảo mật. Đồng thời, kho artifact đóng vai trò nền tảng để các môi trường sau này (UAT, Production) có thể sử dụng cùng một gói triển khai, đảm bảo tính nhất quán xuyên suốt toàn bộ vòng đời dịch vụ SOA.



```
podTemplate(label: podLabel, yaml: POD_YAML) {
  node(podLabel) {

    // Log cho thấy job chạy trong /home/jenkins/agent/builds
    ws('/home/jenkins/agent/builds') {

      stage('Init') {···
      }

      stage('Warnup') {···
      }

      stage('Execute') {
        // Tính tag image:
        // Ưu tiên env.IMAGE_TAG nếu có, nếu không thì lấy short commit
        def commit = env.IMAGE_COMMIT ?: env.GIT_COMMIT ?: env.CI_COMMIT_SHA ?: 'dev'
        def shortCommit = commit.take(8)
        def tag = env.IMAGE_TAG ?: "dev-${shortCommit}"

        def srcImage  = "${REGISTRY}/${SRC_IMAGE_REPO}:${tag}"
        def destImage = "${REGISTRY}/${DEST_IMAGE_REPO}:${tag}"

        echo "Source image: ${srcImage}"
        echo "Destination image: ${destImage}"

        container('oras') {
          withCredentials([
            string(credentialsId: REGISTRY_PASS_CRED, variable: 'PASS')
          ]) {
            sh """
              set -e
              echo "Login to registry ${REGISTRY} ... "
              oras login --insecure -u nexus_user -p "$PASS" ${REGISTRY}

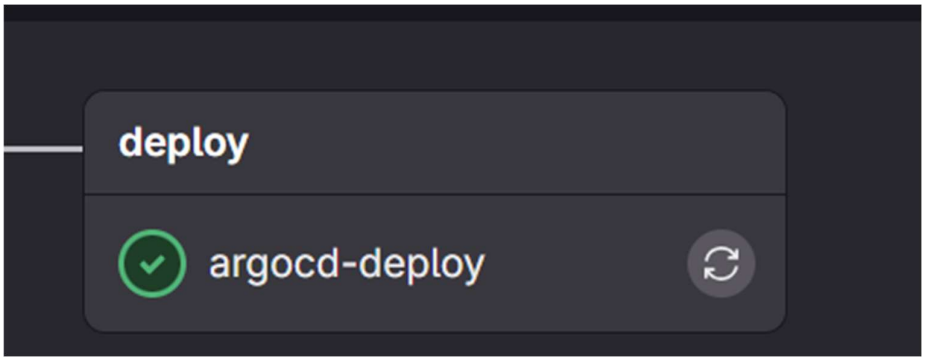
              echo "Copy image from ${srcImage} to ${destImage} ... "
              oras cp --from-insecure --to-insecure \\
                ${srcImage} \\
                ${destImage}
            """
          }
        }
      } // stage Execute

    } // ws
```

Hình 3.6: Logic code tải image lên đám mây

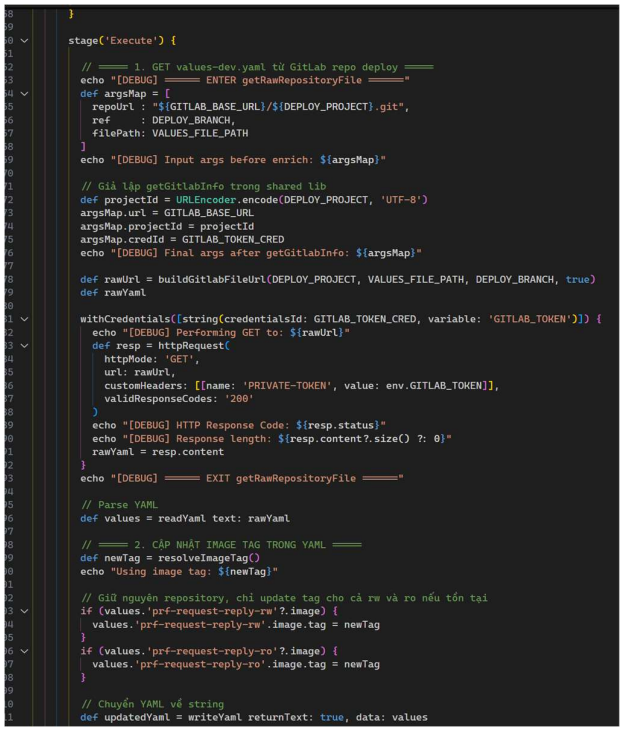
3.2.2. Quy trình CD

Pha CD (Continuous Deployment) bắt đầu ngay sau khi artifact đã được publish thành công lên kho lưu trữ trung tâm. Thay vì triển khai trực tiếp artifact vào môi trường chạy, hệ thống sử dụng mô hình GitOps, trong đó toàn bộ cấu hình triển khai Kubernetes (Helm chart) được quản lý trong GitOps Config Repository. Khi pipeline hoàn tất bước publish, hệ thống sẽ tự động cập nhật thông tin phiên bản artifact mới (ví dụ: tag của Docker image) vào repo cấu hình này. Mọi thay đổi tại đây đều được kiểm soát bằng Git, bảo đảm tính minh bạch, khả năng truy vết và phục vụ kiểm toán.



Hình 3.7: Pha CD trên Gitlab

Ngay khi GitOps repository được cập nhật, công cụ Argo CD – hoạt động như một GitOps Controller – liên tục theo dõi sự thay đổi tại kho cấu hình. Argo CD phát hiện phiên bản mới của cấu hình triển khai và so sánh với trạng thái thực tế đang chạy trong cụm OpenShift/Kubernetes. Nếu phát hiện sự chênh lệch, Argo CD sẽ tự động kích hoạt quá trình đồng bộ (sync) để triển khai phiên bản mới vào cụm. Quá trình này bao gồm cập nhật Deployment, StatefulSet, ConfigMap, Secret… tùy theo thay đổi, đồng thời hỗ trợ chiến lược rolling update để bảo đảm quá trình nâng cấp diễn ra mượt mà, không gián đoạn dịch vụ.



```
}

stage('Execute') {

  // ═══ 1. GET values-dev.yaml từ GitLab repo deploy ═══
  echo "[DEBUG] ═══ ENTER getRawRepositoryFile ═══"
  def argsMap = [
    repoUrl : "${GITLAB_BASE_URL}/${DEPLOY_PROJECT}.git",
    ref     : DEPLOY_BRANCH,
    filePath: VALUES_FILE_PATH
  ]
  echo "[DEBUG] Input args before enrich: ${argsMap}"

  // Giả lập getGitlabInfo trong shared lib
  def projectId = URLEncoder.encode(DEPLOY_PROJECT, 'UTF-8')
  argsMap.url = GITLAB_BASE_URL
  argsMap.projectId = projectId
  argsMap.credId = GITLAB_TOKEN_CRED
  echo "[DEBUG] Final args after getGitlabInfo: ${argsMap}"

  def rawUrl = buildGitlabFileUrl(DEPLOY_PROJECT, VALUES_FILE_PATH, DEPLOY_BRANCH, true)
  def rawYaml

  withCredentials([string(credentialsId: GITLAB_TOKEN_CRED, variable: 'GITLAB_TOKEN')]) {
    echo "[DEBUG] Performing GET to: ${rawUrl}"
    def resp = httpRequest(
      httpMode: 'GET',
      url: rawUrl,
      customHeaders: [[name: 'PRIVATE-TOKEN', value: env.GITLAB_TOKEN]],
      validResponseCodes: '200'
    )
    echo "[DEBUG] HTTP Response Code: ${resp.status}"
    echo "[DEBUG] Response length: ${resp.content?.size() ?: 0}"
    rawYaml = resp.content
  }
  echo "[DEBUG] ═══ EXIT getRawRepositoryFile ═══"

  // Parse YAML
  def values = readYaml text: rawYaml

  // ═══ 2. CẬP NHẬT IMAGE TAG TRONG YAML ═══
  def newTag = resolveImageTag()
  echo "Using image tag: ${newTag}"

  // Giữ nguyên repository, chỉ update tag cho cả rw và ro nếu tồn tại
  if (values.'prf-request-reply-rw'?.image) {
    values.'prf-request-reply-rw'.image.tag = newTag
  }
  if (values.'prf-request-reply-ro'?.image) {
    values.'prf-request-reply-ro'.image.tag = newTag
  }

  // Chuyển YAML về string
  def updatedYaml = writeYaml returnText: true, data: values
```

Hình 3.8: Logic của pha CD



```
// ═══ 3. PUT LẠI FILE values-dev.yaml LÊN GITLAB ═══
echo "[DEBUG] ═══ ENTER updateRepositoryFile ═══"

def fileUrl = buildGitlabFileUrl(DEPLOY_PROJECT, VALUES_FILE_PATH, DEPLOY_BRANCH, false)

def payload = [
  branch        : DEPLOY_BRANCH,
  commit_message: "Update image tag to ${newTag} for prf-request-reply",
  content       : updatedYaml,
  encoding      : "text"
]
def jsonBody = JsonOutput.toJson(payload)

withCredentials([string(credentialsId: GITLAB_TOKEN_CRED, variable: 'GITLAB_TOKEN')]) {
  def putResp = httpRequest(
    httpMode: 'PUT',
    url: fileUrl,
    customHeaders: [[name: 'PRIVATE-TOKEN', value: env.GITLAB_TOKEN]],
    contentType: 'APPLICATION_JSON',
    requestBody: jsonBody,
    validResponseCodes: '200'
  )
  echo "[DEBUG] PUT Response Code: ${putResp.status}"
  // Nếu cần đọc JSON response:
  def respJson = readJSON text: putResp.content
  echo "[DEBUG] update file result: ${respJson}"
}

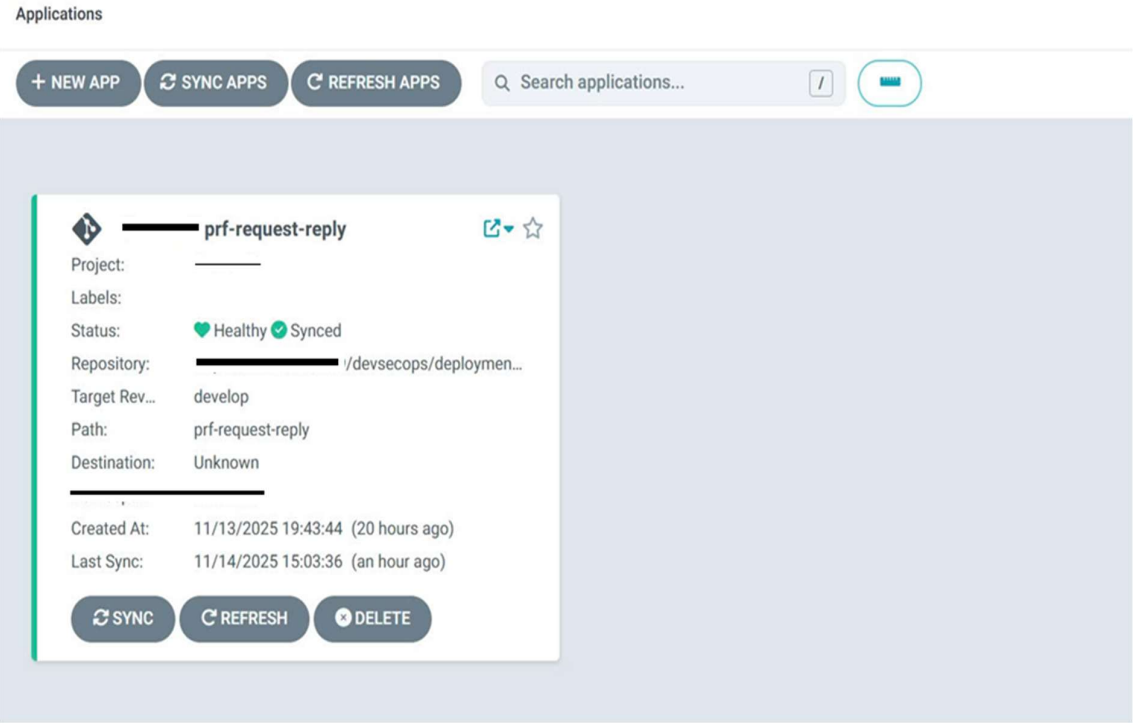
echo "[DEBUG] ═══ EXIT updateRepositoryFile ═══"

// ═══ 4. ARGOCD SYNC APP ═══
container('argocd') {
  withCredentials([
    usernamePassword(
      credentialsId: ARGOCD_CRED_ID,
      usernameVariable: 'ARGOCD_USER',
      passwordVariable: 'ARGOCD_PASS'
    )
  ]) {
    sh """
      set -e
      argocd login ${ARGOCD_SERVER} \\
        --username "$ARGOCD_USER" \\
        --password "$ARGOCD_PASS" \\
        --insecure

      argocd app sync ${ARGOCD_APP_NAME} \\
        --prune \\
        --timeout 300
    """
  }
```

Hình 3.9: Logic của pha CD (tiếp)

Trong quá trình triển khai, Argo CD liên tục giám sát trạng thái Pod, tài nguyên và các điều kiện sẵn sàng (readiness/liveness). Nếu triển khai gặp lỗi hoặc Pod khởi chạy không thành công, Argo CD có khả năng tự động rollback về phiên bản ổn định trước đó dựa trên trạng thái được ghi nhận trong GitOps repository. Điều này giúp giảm thiểu rủi ro khi triển khai và tăng mức độ tin cậy của hệ thống.



Hình 3.10: Giao diện ArgoCD

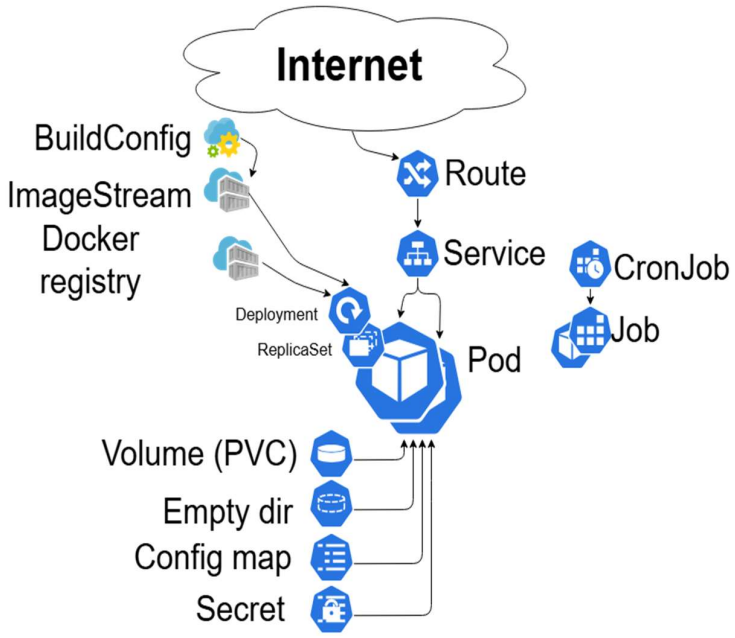
Nhờ sử dụng mô hình GitOps và Argo CD, quy trình CD đạt được mức độ tự động hóa rất cao, bảo đảm rằng mọi phiên bản được triển khai ra môi trường thực thi đều là phiên bản đã được kiểm duyệt trong Git, đồng thời hạn chế tối đa sự can thiệp thủ công. Điều này góp phần nâng cao độ ổn định, khả năng kiểm soát và hiệu quả vận hành của các dịch vụ SOA triển khai trên nền tảng OpenShift/Kubernetes.

3.3. Thực nghiệm triển khai services

3.3.1. Sơ đồ concept OpenShift kết hợp Kubernetes

Hình dưới đây mô tả kiến trúc tổng quan của hệ thống ứng dụng được triển khai trên nền tảng Kubernetes trong môi trường điện toán đám mây. Kiến trúc này thể hiện đầy đủ các lớp chức năng từ phía người dùng, lớp điều phối tài nguyên, cho đến các dịch vụ hỗ trợ vận hành và lưu trữ dữ liệu. Mục tiêu của mô hình là đảm bảo khả năng triển khai linh hoạt, tự động hóa quá trình quản lý vòng đời ứng dụng và tối ưu hiệu năng trong điều kiện hệ thống có số lượng dịch vụ lớn và tải biến động liên tục.

Ở tầng trên cùng, người dùng hoặc các ứng dụng bên ngoài truy cập vào hệ thống thông qua Internet. Các yêu cầu này được chuyển tới hệ thống thông qua các thành phần điều hướng như Ingress Controller hoặc API Gateway. Đây là lớp trung gian thực hiện chức năng điều phối lưu lượng, cân bằng tải và kiểm soát bảo mật đầu vào trước khi yêu cầu được chuyển vào cụm Kubernetes.



Hình 3.11: Tổng quan về Openshift

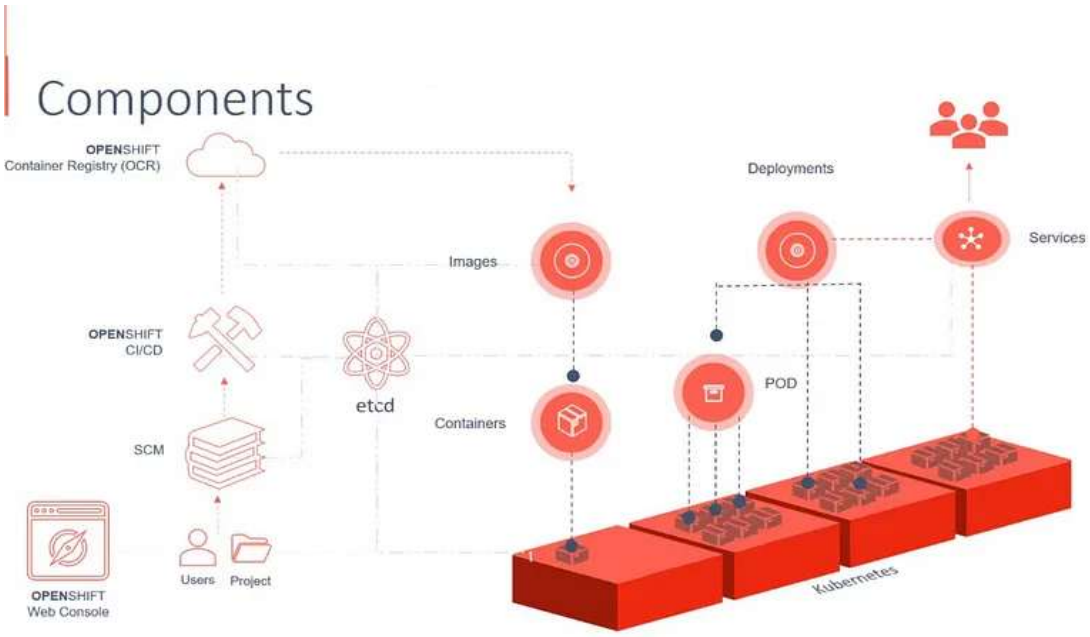
Phía trong cụm, ứng dụng được triển khai dưới dạng các container – đơn vị đóng gói nhẹ, độc lập và dễ mở rộng. Kubernetes quản lý các container thông qua các đối tượng như Pod, ReplicaSet, Deployment, đảm bảo khả năng mở rộng động (auto-scaling), khôi phục khi xảy ra lỗi (self-healing), và phân phối tài nguyên hợp lý giữa các node trong cụm. Nhờ đó, hệ thống đạt được tính linh hoạt và sẵn sàng cao trong môi trường vận hành thực tế.

Bao quanh khối xử lý trung tâm là các dịch vụ hỗ trợ của Kubernetes, bao gồm:

- Service (ClusterIP/NodePort/LoadBalancer): định tuyến yêu cầu đến các Pod phù hợp.
- ConfigMap và Secret: quản lý cấu hình ứng dụng và thông tin nhạy cảm.

- Persistent Volume và Persistent Volume Claim: cung cấp cơ chế lưu trữ dữ liệu bền vững.
- Job và CronJob: thực thi các tác vụ theo lịch hoặc tác vụ nền.

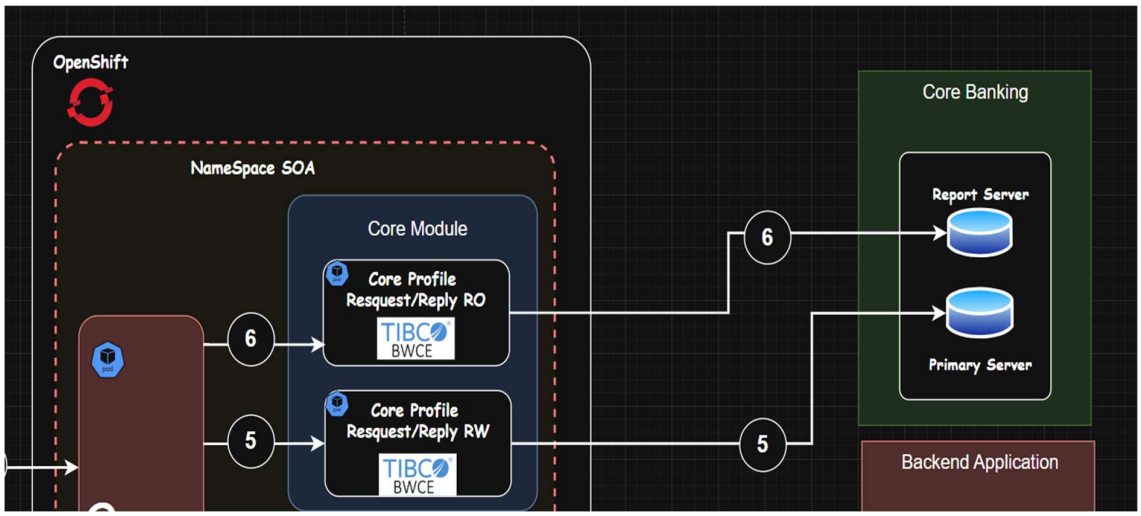
Các thành phần này hoạt động phối hợp nhằm đảm bảo môi trường vận hành thống nhất, chuẩn hóa và dễ quản lý.



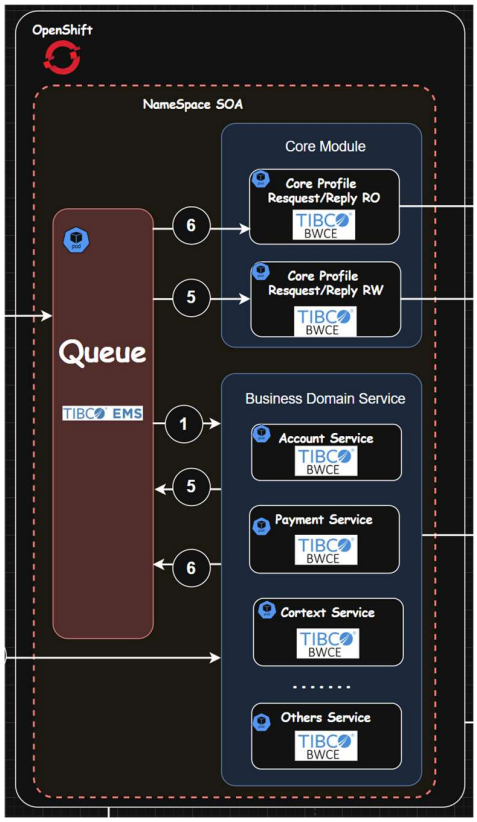
Hình 3.12: Tổng quan cấu trúc Openshift

Cuối cùng, tầng lưu trữ và tích hợp backend đóng vai trò xử lý dữ liệu và cung cấp dịch vụ cho các thành phần phía trên. Đây có thể là cơ sở dữ liệu quan hệ, hệ thống message queue hoặc các ứng dụng lõi khác. Kubernetes cho phép kết nối linh hoạt tới các hệ thống này thông qua cơ chế service hoặc kết nối trực tiếp thông qua API.

3.3.2. Service gọi đến lõi ngân hàng



Hình 3.13: Cấu trúc service gọi đến lõi ngân hàng



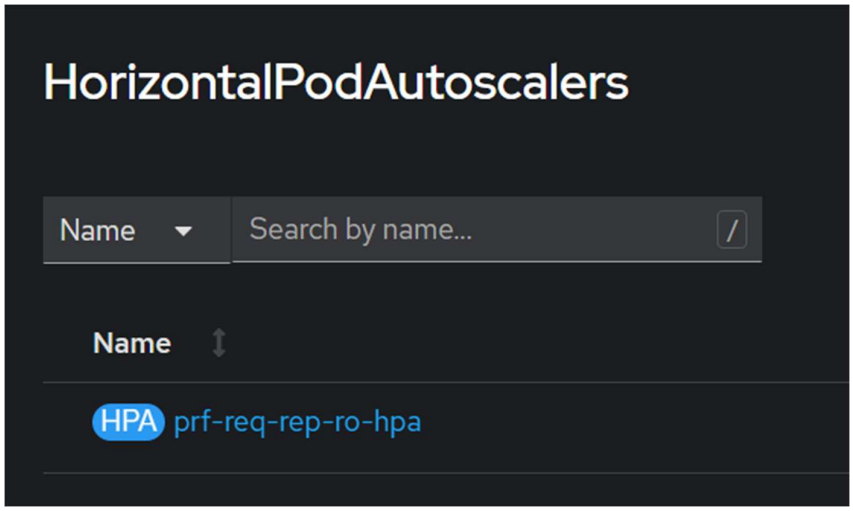
Hình 3.14: Toàn bộ hệ thống SOA phổ thông

Bài toán:

Đây là service dùng chung, 70% dịch vụ của ngân hàng đều đi qua service này để gọi vào CoreBanking nên nguồn tài nguyên cần phải tối ưu, tập trung cho dịch vụ này.

Vậy, bài toán được giải quyết bằng việc có nhiều pod để hỗ trợ cân tải. Đồng thời mã nguồn bên trong đều phải được tối ưu một cách hợp lý

Đứng về phía hạ tầng, dịch vụ này có những thành phần như sau:



Hình 3.15: Ví dụ về HPA

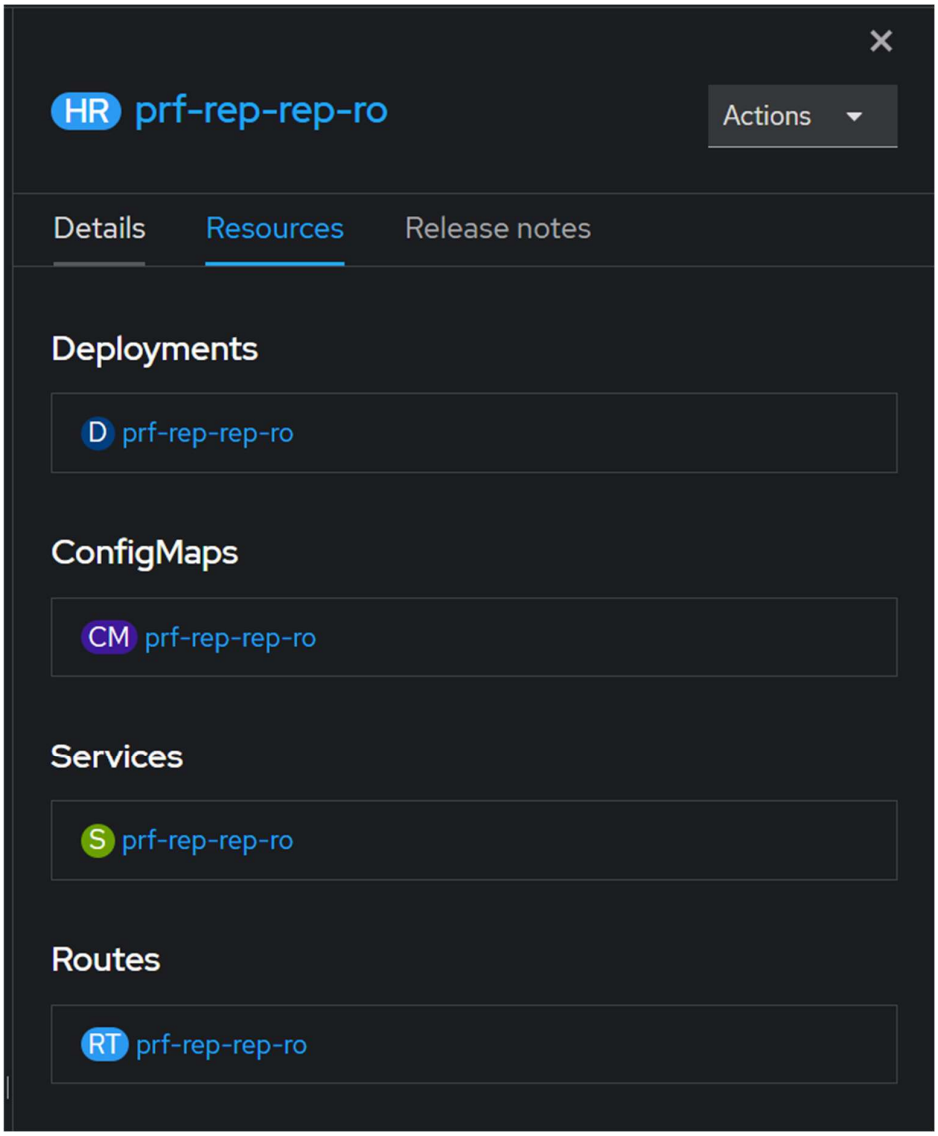
HorizontalPodAutoscaler là cơ chế trong Kubernetes tự động tăng hoặc giảm số lượng Pod dựa trên mức sử dụng tài nguyên như CPU hoặc RAM. Nhờ đó, hệ thống luôn duy trì hiệu năng ổn định và tối ưu tài nguyên khi tải biến động.

Deployment: dùng để quản lý vòng đời của các Pod trong OpenShift. Chúng giúp thực hiện triển khai, cập nhật phiên bản (rolling update), mở rộng hoặc thu nhỏ số lượng Pod một cách tự động. Đồng thời tại đây khai báo những tham số được dùng ví dụ như ConfigMaps và Secrets

ConfigMaps: dùng để lưu trữ các cấu hình không nhạy cảm như tham số ứng dụng, URL kết nối, biến môi trường. Tại đây sẽ lưu lại những tham số gọi đến dịch vụ lõi ngân hàng.

Services: cung cấp điểm truy cập ổn định (stable endpoint) cho các Pod phía sau, bất kể Pod có thay đổi hay được khởi động lại. Chúng đảm nhiệm chức năng định tuyến lưu lượng và cân bằng tải, đảm bảo các Pod nhận được request đúng cách trong hệ thống phân tán.

Routes cho phép các ứng dụng trong OpenShift phơi bày dịch vụ ra bên ngoài cụm Kubernetes ra mạng nội bộ. Đây là cơ chế điều hướng lưu lượng từ bên ngoài vào ứng dụng thông qua HTTP/HTTPS, đồng thời hỗ trợ các chính sách bảo mật và cân bằng tải nâng cao.



Hình 3.16: Tài nguyên của 1 service

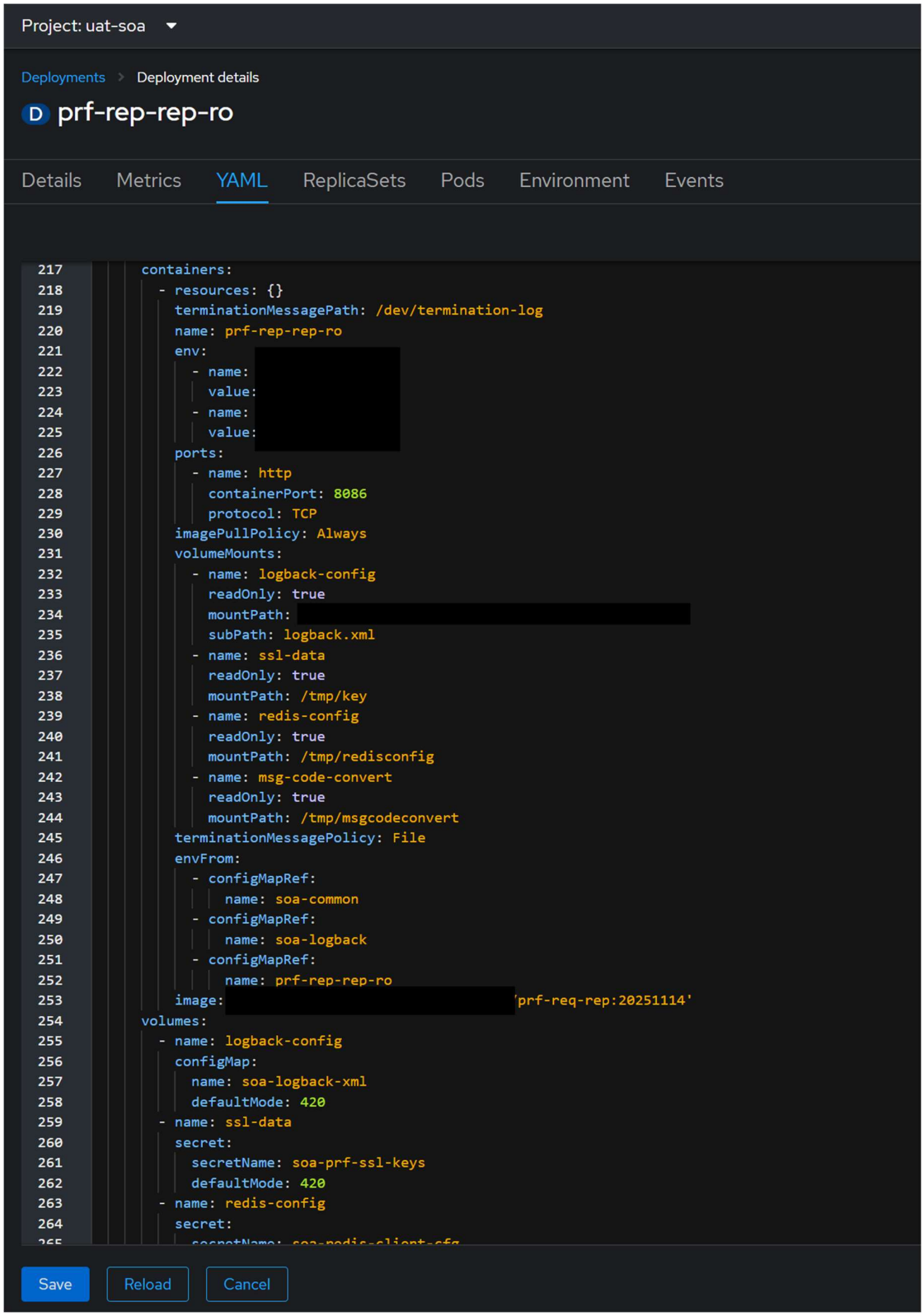
```yaml
spec:
  scaleTargetRef:
    kind: Deployment
    name: prf-req-rep-ro
    apiVersion: apps/v1
  minReplicas: 1
  maxReplicas: 5
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 70
```

Hình 3.17: Nội dung HorizontalPodAutoscaler

- mỉnReplicas, maxReplicas: số lượng pod tối thiểu và tối đa cho deployment
- Khi CPU > 70% → tự động scale thêm Pod

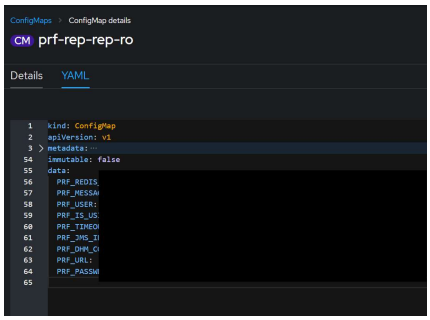
Nội dung Deployment

- Name: Tên container trong pod
- Env: Định nghĩa biến môi trường cho container
- Ports: Thông tin ports được phơi ra của dịch vụ
- VolumeMounts: Danh sách các volume được gắn vào container.
- EnvFrom: Dùng để import ConfigMap hoặc Secret vào container dưới dạng biến môi trường.
- Image: Docker image để chạy dịch vụ BW6

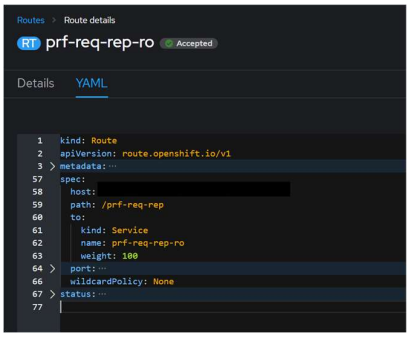


Hình 3.18: Nội dung Deployment

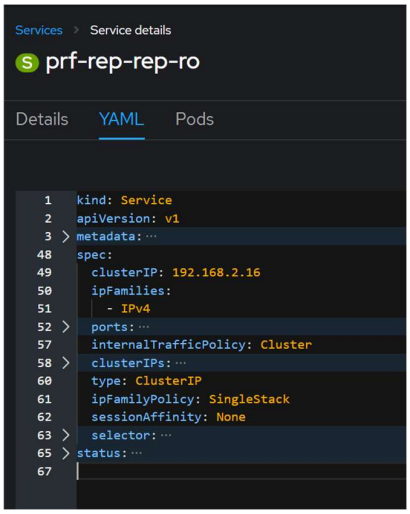
Nội dung ConfigMaps, Services, Routes



Hình 3.19: ConfigMaps



Hình 3.20: Routes



Hình 3.21: Services

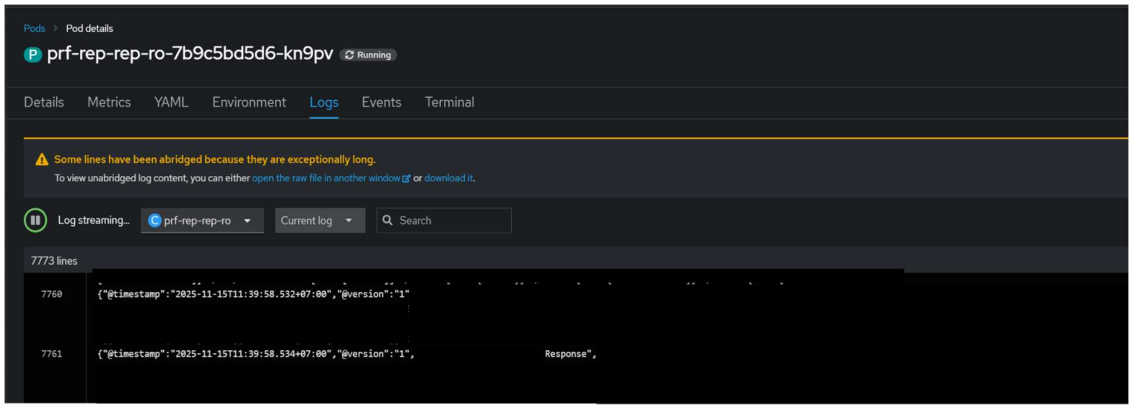
Nhìn chung, các tài nguyên như Deployment, Service, Route, ConfigMap và Secret đóng vai trò nền tảng trong việc triển khai và vận hành các dịch vụ SOA trên OpenShift/Kubernetes. Sự phối hợp giữa các thành phần này giúp hệ thống đạt được tính linh hoạt trong cấu hình, khả năng mở rộng động, tính sẵn sàng cao và khả năng tách biệt môi trường vận hành khỏi cấu trúc ứng dụng. Việc quản lý đúng chuẩn các tài nguyên này không chỉ bảo đảm dịch vụ hoạt động ổn định, mà còn là tiền đề quan trọng để triển khai các cơ chế giám sát và vận hành nâng cao.

Trên cơ sở đó, một thành phần không thể thiếu trong hệ thống phân tán hiện đại là cơ chế ghi log và giám sát tập trung. Đây là yếu tố giúp các nhóm phát triển và

vận hành nắm bắt được hành vi của ứng dụng, phát hiện sớm lỗi phát sinh và đánh giá hiệu suất theo thời gian thực. Phần tiếp theo sẽ trình bày chi tiết phương pháp đọc log từ Pod, cơ chế thu thập log tự động và mô hình tích hợp hệ thống log tập trung trong môi trường Kubernetes.

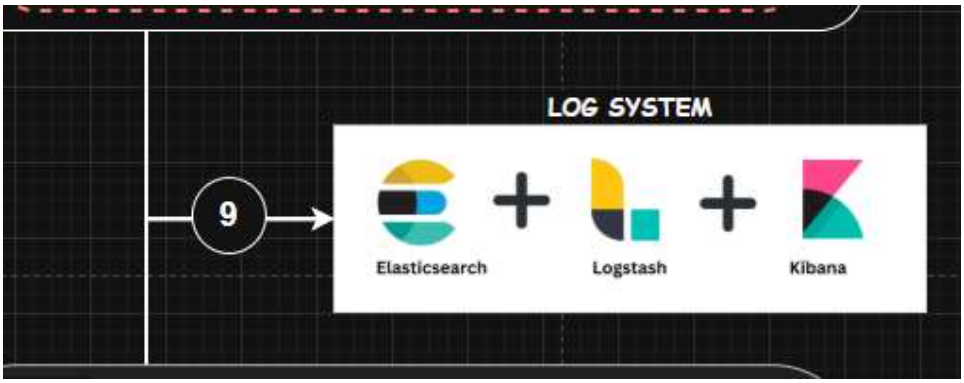
3.3.3. Đọc log và ghi log vào log tập trung

Trong quá trình hoạt động, Pod sinh ra các bản ghi (log) phản ánh trạng thái và hành vi của ứng dụng. Các log này được lưu trực tiếp tại container dưới dạng dòng xuất chuẩn (stdout/stderr) và được Kubernetes thu thập tập trung. Cơ chế ghi log của Pod cho phép đội ngũ vận hành dễ dàng theo dõi, phân tích lỗi và đánh giá hiệu năng ứng dụng, đồng thời hỗ trợ tích hợp với các hệ thống giám sát tập trung như ELK để nâng cao khả năng quan sát toàn hệ thống.



Hình 3.22: Ví dụ log của 1 service

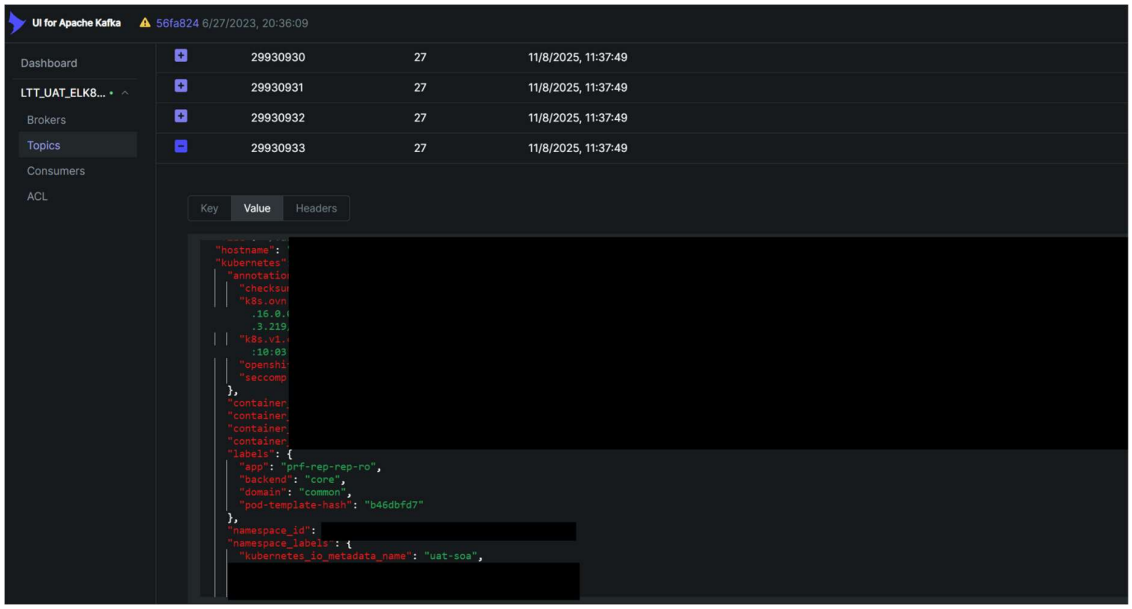
Trong kiến trúc triển khai SOA trên Kubernetes, việc thu thập và phân tích log từ các dịch vụ phân tán là yêu cầu thiết yếu nhằm bảo đảm khả năng quan sát vận hành, giám sát lỗi và tối ưu hiệu năng hệ thống. Để đáp ứng yêu cầu này, hệ thống sử dụng mô hình ghi log tập trung thông qua sự kết hợp giữa Kafka và nền tảng ELK (Elasticsearch – Logstash – Kibana).



Hình 3.23: Mô hình ELK

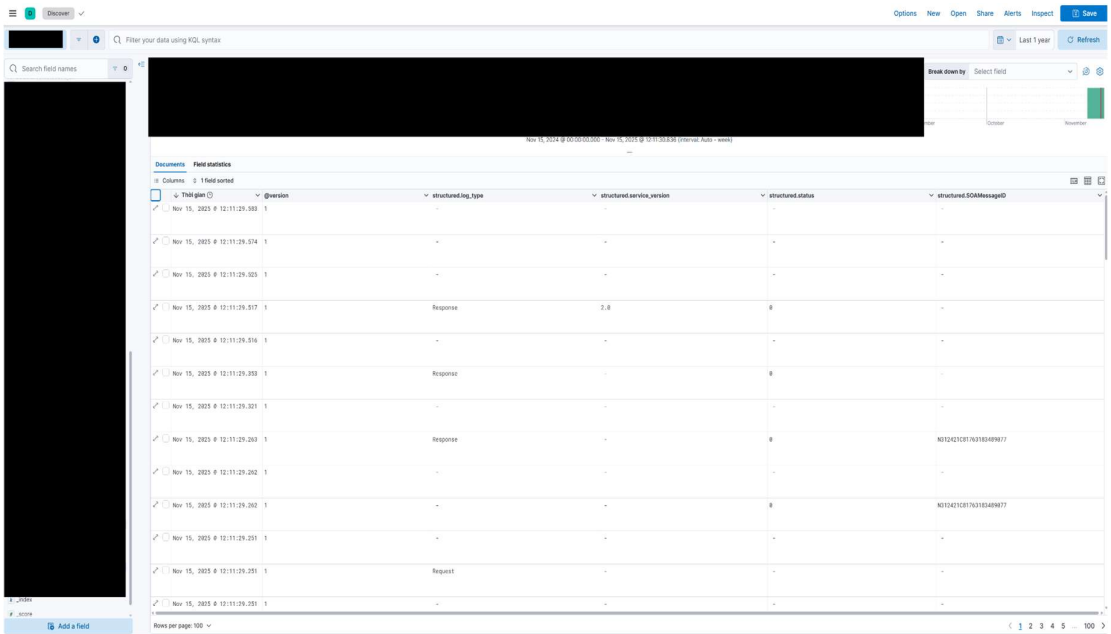
Trong mô hình này, các Pod của dịch vụ SOA xuất log dưới dạng stdout/stderr và log được thu thập bởi Fluentd/Sidecar hoặc cơ chế logging tích hợp của OpenShift. Thay vì gửi trực tiếp vào Logstash, các bản ghi trước tiên được đưa vào Kafka, đóng vai trò là hàng đợi thông điệp có khả năng chịu tải cao. Kafka giúp hệ thống đệm log, tránh tình trạng mất log trong trường hợp Logstash hoặc Elasticsearch quá tải, đồng thời hỗ trợ khả năng mở rộng linh hoạt cho các tác vụ xử lý log theo thời gian thực.

Từ Kafka, các luồng log được Logstash tiêu thụ, xử lý và chuẩn hóa theo một schema thống nhất, giúp loại bỏ dữ liệu dư thừa và bổ sung thông tin về thời gian, nguồn dịch vụ hoặc nhãn môi trường. Sau bước xử lý này, Logstash chuyển log vào Elasticsearch, nơi dữ liệu được lưu trữ dưới dạng chỉ mục và có thể truy vấn hiệu quả với quy mô lớn. Người vận hành truy cập vào Kibana để trực quan hóa các bản ghi, phân tích lỗi, xây dựng bảng điều khiển (dashboard) và theo dõi tình trạng hệ thống theo thời gian thực.



Hình 3.24: Giao diện Kafka

Việc bổ sung Kafka vào pipeline log không chỉ tăng khả năng chịu tải và chống mất log, mà còn tạo nền tảng cho các tính năng nâng cao như phân tích sự kiện theo thời gian thực, giám sát an ninh hoặc cảnh báo tự động. Nhờ đó, hệ thống ghi log tập trung trở nên ổn định, linh hoạt và phù hợp với đặc điểm vận hành của hệ thống SOA hiện đại triển khai trên Kubernetes.

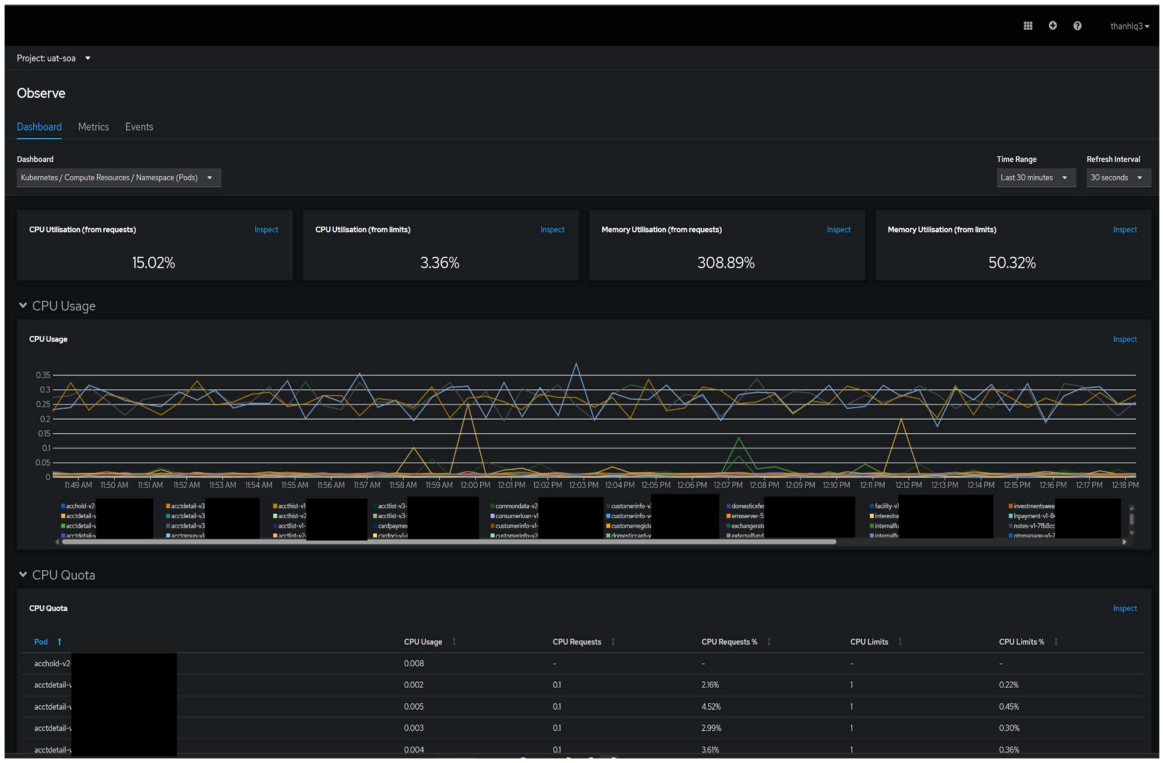


Hình 3.25: Giao diện Elasticsearch

3.3.4. Bảng điều khiển giám sát kiểm soát (Control Monitoring Dashboard)

Trong môi trường triển khai ứng dụng phân tán trên Kubernetes/OpenShift, việc theo dõi trạng thái hoạt động của các tài nguyên và dịch vụ là yếu tố quan trọng nhằm đảm bảo tính ổn định, khả năng mở rộng và hiệu năng của hệ thống. Để đáp ứng yêu cầu này, OpenShift cung cấp Control Monitoring Dashboard – một bảng điều khiển tập trung cho phép đội ngũ vận hành giám sát toàn diện các thành phần của cụm, bao gồm Pod, Deployment, Service, Route, tài nguyên CPU/RAM, trạng thái node, cũng như các cảnh báo liên quan đến tải hệ thống và sự cố phát sinh.

Bảng điều khiển này đóng vai trò như "trung tâm điều phối trực quan" của cụm OpenShift, cung cấp dữ liệu theo thời gian thực thông qua các biểu đồ, chỉ số và cảnh báo tự động. Nhờ đó, người vận hành có thể nhanh chóng phát hiện bất thường, đánh giá mức độ sử dụng tài nguyên, xác định bottleneck, và đưa ra quyết định điều chỉnh cấu hình hoặc mở rộng hệ thống một cách kịp thời. Đây là thành phần then chốt giúp nâng cao hiệu quả vận hành và đảm bảo tính sẵn sàng cao cho các dịch vụ SOA triển khai trên nền tảng OpenShift.



Hình 3.26: Dashboard của Openshift

Tại đây, hệ thống hiển thị tổng hợp nhiều chỉ số quan trọng, bao gồm mức sử dụng CPU và bộ nhớ so với giá trị "requests" và "limits" được khai báo trong cấu hình. Các chỉ số này cho phép người vận hành đánh giá nhanh tình trạng tải hiện tại của cụm, phát hiện các Pod đang sử dụng vượt ngưỡng tài nguyên hoặc không đạt mức hiệu năng mong đợi.

Phần biểu đồ "CPU Usage" cung cấp dữ liệu theo thời gian thực dưới dạng các đường biểu diễn mức tiêu thụ CPU của từng Pod trong namespace. Nhờ cách hiển thị dạng chuỗi thời gian (time series), người vận hành có thể quan sát xu hướng tiêu thụ tài nguyên, nhận diện các đỉnh tải (spikes) bất thường hoặc mô hình sử dụng lặp lại theo thời gian. Các Pod với màu sắc khác nhau giúp phân biệt và theo dõi dễ dàng từng dịch vụ trong hệ thống SOA.

Bên dưới, bảng "CPU Quota" trình bày chi tiết mức sử dụng CPU của từng Pod, bao gồm: giá trị CPU thực tế, yêu cầu CPU (CPU Requests), giới hạn CPU (CPU Limits) và tỷ lệ phần trăm sử dụng so với yêu cầu và giới hạn. Bảng dữ liệu này giúp

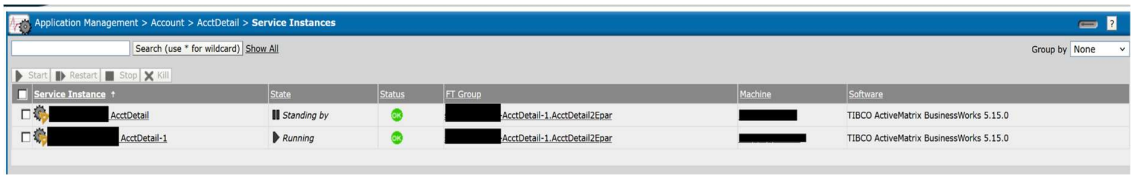
việc so sánh giữa cấu hình và thực tế trở nên minh bạch, hỗ trợ quyết định tối ưu tài nguyên hoặc điều chỉnh cấu hình HPA để mở rộng tự động.

Nhìn chung, dashboard này đóng vai trò như một công cụ theo dõi trung tâm cho toàn bộ namespace, giúp đội vận hành nhanh chóng nắm bắt trạng thái hoạt động của hệ thống, kịp thời phát hiện vấn đề, và đảm bảo các dịch vụ chạy trên nền tảng OpenShift luôn duy trì hiệu năng ổn định và đạt yêu cầu vận hành.

3.4. Đánh giá hiệu quả của sử dụng TIBCO BW6 kết hợp với Openshift và Kubernetes

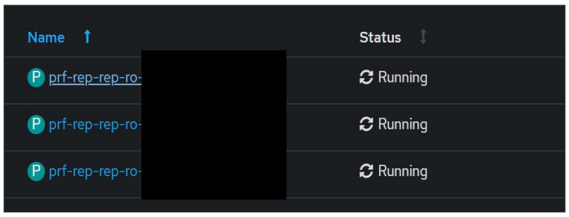
3.4.1. Khả năng Auto Scaling

Trong mô hình BW5, hệ thống triển khai theo dạng máy chủ vật lý hoặc máy ảo cố định, dẫn đến tài nguyên cấp phát không thể tự động mở rộng theo tải. Mỗi khi khối lượng giao dịch tăng cao, các tiến trình BW5 chỉ có thể scale bằng cách tăng CPU/RAM của server hoặc nhân bản thêm BW engine theo cách thủ công. Điều này làm tăng độ trễ xử lý, giảm khả năng đáp ứng theo thời gian thực và tạo áp lực lên đội vận hành. Việc thiếu cơ chế auto scaling thực chất khiến BW5 không phù hợp với môi trường giao dịch biến động lớn trong ngân hàng.



Hình 3.27: Dashboard của 1 service TIBCO BW5

Ngược lại, BW6 khi triển khai trên nền tảng Kubernetes/OpenShift hỗ trợ đầy đủ cơ chế Horizontal Pod Autoscaling (HPA), cho phép dịch vụ tự động mở rộng theo CPU, RAM hoặc thậm chí theo số lượng request. Khi tải tăng, Kubernetes chủ động tạo thêm Pod của dịch vụ BW6 để duy trì hiệu năng ổn định; khi tải giảm, hệ thống thu nhỏ lại giúp tiết kiệm tài nguyên. Cách thức vận hành này mang lại tính đàn hồi cao, phù hợp với đặc thù giao dịch theo phiên (payday, cuối tháng, batch xử lý…). Đây là cải tiến quan trọng giúp nâng cao độ tin cậy cho các dịch vụ SOA.



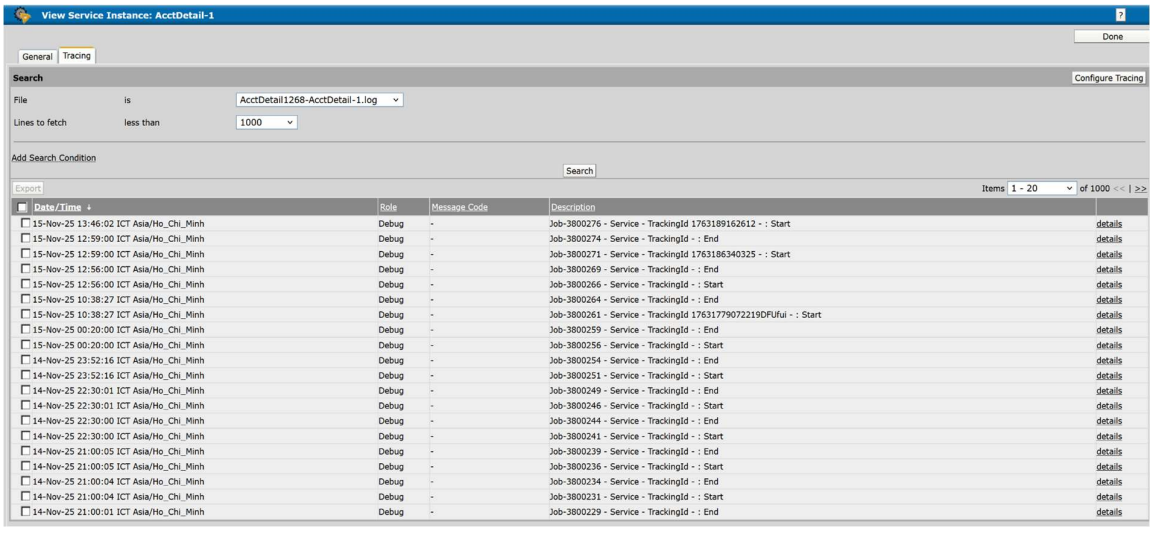
Hình 3.28: Danh sách node của 1 service TIBCO BW6

```yaml
spec:
  scaleTargetRef:
    kind: Deployment
    name: prf-req-rep-ro
    apiVersion: apps/v1
  minReplicas: 1
  maxReplicas: 5
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 70
```

Hình 3.29: HPA của 1 service

3.4.2. Quản lý log tập trung

Trong BW5, log được ghi trực tiếp vào file hệ thống của từng server. Khi số lượng BW engines lớn hoặc trải dài trên nhiều node, việc thu thập log trở nên phân tán, gây khó khăn trong truy vết sự cố, tìm nguyên nhân giao dịch thất bại hoặc theo dõi hiệu năng. Hệ thống BW5 thiếu tích hợp sẵn với các nền tảng logging chuẩn như ELK/Kafka, dẫn đến phải xử lý bằng các script thủ công hoặc agent cài đặt ngoài. Điều này không chỉ làm tăng chi phí vận hành mà còn khiến việc quản trị log trở nên thiếu đồng bộ và khó mở rộng.

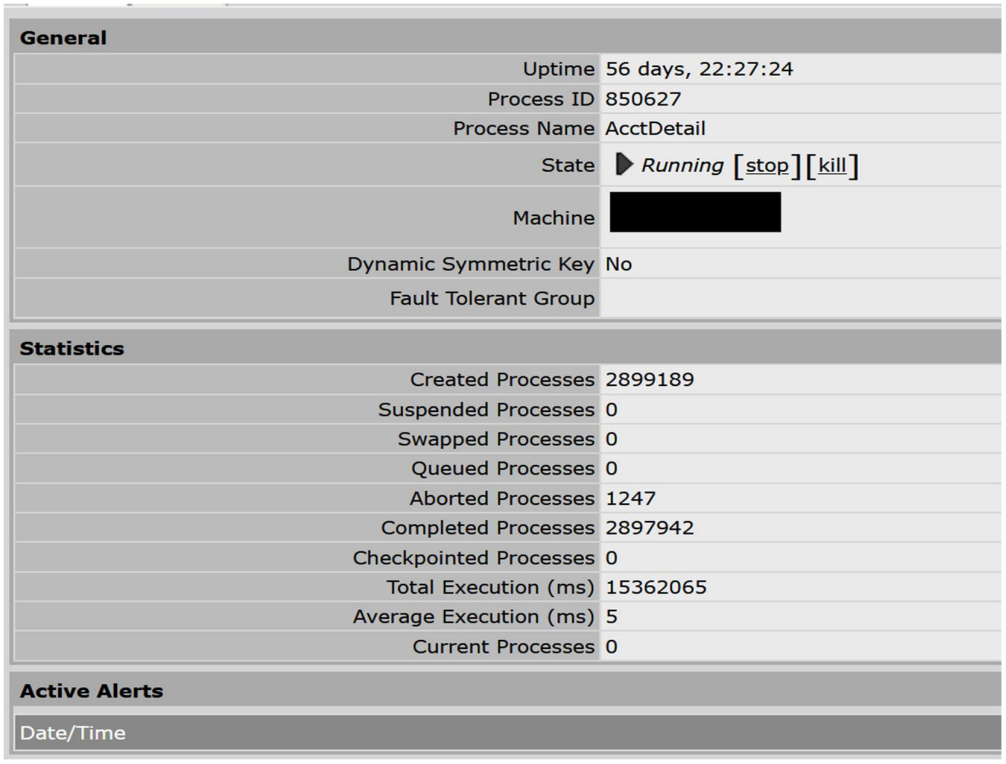


Hình 3.30: Log phân tán trên TIBCO BW5

BW6 hỗ trợ logging theo chuẩn container và tích hợp liền mạch với hệ sinh thái Kubernetes. Mỗi Pod tự động xuất log theo chuẩn stdout/stderr, và OpenShift cho phép gửi log vào hệ thống Kafka → Logstash → Elasticsearch → Kibana để hiển thị tập trung. Nhờ đó, toàn bộ log có thể truy vấn theo dịch vụ, theo Pod, theo thời gian hoặc theo trace ID của giao dịch. Khả năng phân tích log theo real-time, dashboard log và cảnh báo log bất thường là những điểm vượt trội của BW6 so với BW5.

3.4.3. Dashboard monitoring

BW5 sử dụng công cụ quản lý TEA (TIBCO Enterprise Administrator), tuy cung cấp các thông tin cơ bản về trạng thái BW engine nhưng còn nhiều hạn chế: không đo tải ở mức container, không theo dõi tài nguyên chi tiết theo thời gian thực, thiếu biểu đồ CPU/RAM theo pod và không hỗ trợ cảnh báo chủ động. Điều này gây khó khăn cho việc giám sát hiệu năng, đo tải, hoặc phân tích nguyên nhân nghẽn hệ thống.



Hình 3.31: Thông tin sơ sài của service trên TIBCO BW5

Còn đối với BW6, khi triển khai trên OpenShift/Kubernetes, engine được hưởng toàn bộ khả năng giám sát từ nền tảng:

- Dashboard CPU Utilization theo từng Pod/dịch vụ
- Dashboard Memory Utilization
- Event log monitoring
- Pod health / restart count
- Deployment status & rolling update status
- Node metrics và cluster capacity

Ngoài ra, BW6 có thể tích hợp Prometheus và Grafana để mở rộng khả năng quan sát. Khả năng theo dõi tài nguyên trực quan theo thời gian thực hỗ trợ đáng kể cho việc phân tích bottleneck và tối ưu hiệu năng dịch vụ SOA.

3.4.4. Năng lực CI/CD

Ở mô hình BW5 truyền thống, quy trình CI/CD còn nhiều hạn chế do phụ thuộc mạnh vào các thao tác thủ công và thiếu sự chuẩn hóa. Việc build, đóng gói và deploy ứng dụng thường được thực hiện trên môi trường máy chủ vật lý hoặc ảo hóa, trong đó TEA chỉ hỗ trợ quản lý runtime mà không hỗ trợ pipeline tự động. Các bước review MR, build code, đẩy code lên UAT/Prod đều do con người vận hành. Điều này khiến thời gian triển khai lâu, dễ phát sinh sai sót thủ công và khó duy trì tính nhất quán giữa các môi trường. Ngoài ra, BW5 thiếu khả năng đóng gói theo chuẩn container, khiến việc tích hợp với các hệ thống CI/CD hiện đại (GitLab CI, Jenkins pipeline, GitOps) gặp nhiều trở ngại.

Trái lại, BW6 khi chạy trên nền tảng OpenShift/Kubernetes được thiết kế để hoạt động hiệu quả với hệ thống CI/CD tự động hóa hoàn chỉnh. Mã nguồn được quản lý tập trung trên Git; mỗi commit hoặc Merge Request sẽ kích hoạt pipeline CI gồm build BW6 application, build Docker image, kiểm tra các tiêu chí chất lượng và publish artifact lên registry. Giai đoạn CD sử dụng mô hình GitOps, trong đó toàn bộ cấu hình triển khai được quản lý trong GitOps repository. Argo CD liên tục đồng bộ giữa repo và môi trường triển khai thực tế, bảo đảm mọi phiên bản BW6 chạy trong cụm đều là phiên bản đã qua kiểm duyệt. Cách tiếp cận này loại bỏ hoàn toàn sự phụ

thuộc vào thao tác thủ công, tăng độ tin cậy, tăng tốc độ triển khai và bảo đảm tính nhất quán giữa DEV–UAT–PROD. Ngoài ra, BW6 hỗ trợ đầy đủ rolling update, rollback tự động, kiểm tra readiness/liveness và triển khai song song nhiều phiên bản dịch vụ (blue–green hoặc canary). Đây là bước tiến vượt trội so với BW5 vốn chỉ deploy được theo kiểu tĩnh trên server.

3.5. Thử nghiệm test tải trên Openshift

3.5.1. Cấu hình

- Máy đẩy tải
  - 2 máy windows
  - 2 máy linux
- Máy chịu tải



Hình 3.32: Thông tin máy chịu tải

- Công cụ kiểm thử
  - Apache JMeter: Kiểm thử hiệu năng
  - Grafana + influxDB+ Telegraf: Công cụ monitor các thông số RT, TPS của từng transaction trong quá trình đẩy tải

3.5.2. Các tiêu chuẩn SLA/PAC

Các tiêu chí đánh giá hiệu năng đã được thống nhất bao gồm:

Thông lượng (throughput): đơn vị tính số lượng giao dịch/giây (tps), thể hiện năng lực xử lý giao dịch đồng thời của hệ thống. Giá trị tps càng cao thì hệ thống càng xử lý được nhiều giao dịch đồng thời.

Thời gian phản hồi: đơn vị tính (mili-giây hoặc giây), thể hiện thời gian xử lý hoàn tất một giao dịch của người dùng từ khi gửi yêu cầu thực hiện tới khi nhận được kết quả phản hồi thành công của ứng dụng. Giá trị càng thấp thì trải nghiệm dịch vụ của người dùng càng tốt và ngược lại.

Các mốc hiệu năng được ghi nhận bao gồm 1 mốc chính:

Mốc 100% kỳ vọng: bao gồm bộ tiêu chí đánh giá hiệu năng (tps và response time) ...

Kết quả kiểm thử hiệu năng được ghi nhận sau mỗi lần thực hiện kiểm thử hiệu năng và chỉ có giá trị phản ánh hiệu năng của các hệ thống liên quan trên môi trường và tại thời điểm cụ thể đó. Sau mỗi hoạt động thay đổi cấu hình, thông số môi trường, các hoạt động tối ưu hiệu năng của từng hệ thống (một phần hay toàn bộ), thực hiện kiểm thử hiệu năng sẽ cho ra kết quả khác nhau.

3.5.3. Kịch bản kiểm thử

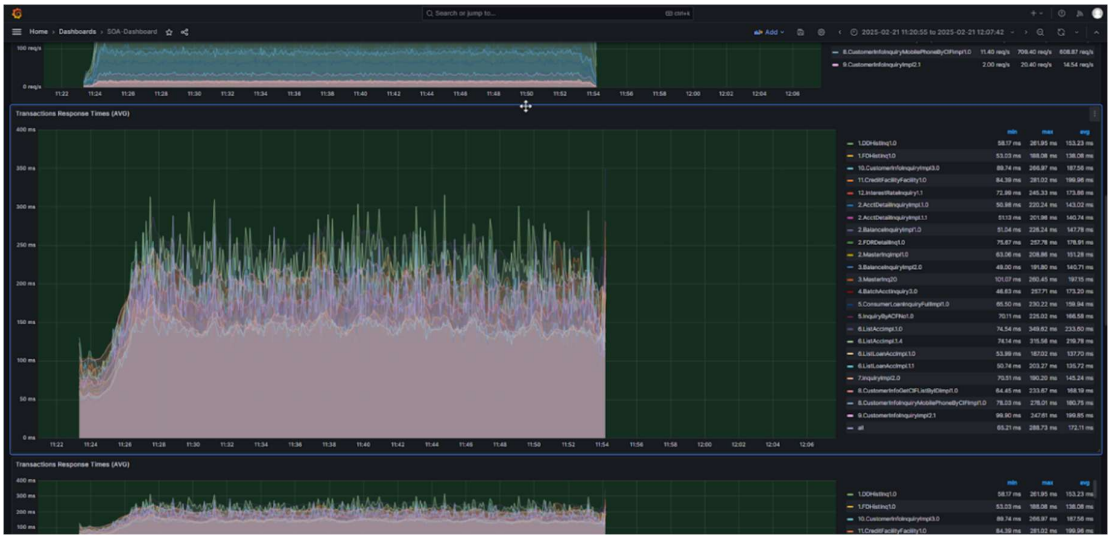
Kiểm thử độc lập ngưỡng Peak TPS - chạy cho từng service độc lập

Luồng	[redacted]	Service	Target TPS (100%)	Target peak TPS	Target RT (ms)	peak TPS ghi nhận	RT ghi nhận	P/F	Note
1			160	208	200	426.08	219.1	P	
			6	15	200	425.91	214.54	P	
			10	15	200	14.9	75.09	P	
			76	98.8	200	99.19	64.45	P	
			2	15	200	14.87	64.41	P	
2			120	156	200	755.9	62.42	P	Riêng sv này k kiểm soát tps
			10	15	200	14.87	89.66	P	
3			70	91	200	91.28	51.23	P	
4			10	15	200	15.86	302.11	P	
			25	32.5	200	32.74	48.37	P	
5			50	65	200	65.48	82.85	P	
			27	35.1	200	35.72	67.7	P	
			3	15	300	15.87	75.99	P	
			422	548.6	300	544.5	73.9	P	
6			12	15.6	200	16	56.78	P	
7			27	35.1	200	35.67	57.34	P	
			14	18.2	200	19	64.67	P	
			6	15	200	15.86	86.08	P	
8			515	669.5	200	694.35	87.82	P	
9				15	200	15.87	120.8	P	
10			70	91	200	91.23	79.64	P	
11			2	15	200	N/A	N/A		DEV cần sửa lại code
12			5	15	200	15.88	79.96	P	

Hình 3.33: So sánh khả năng chịu tải của hệ thống cũ và hệ thống mới



Hình 3.34: Màn hình monitor tổng quát

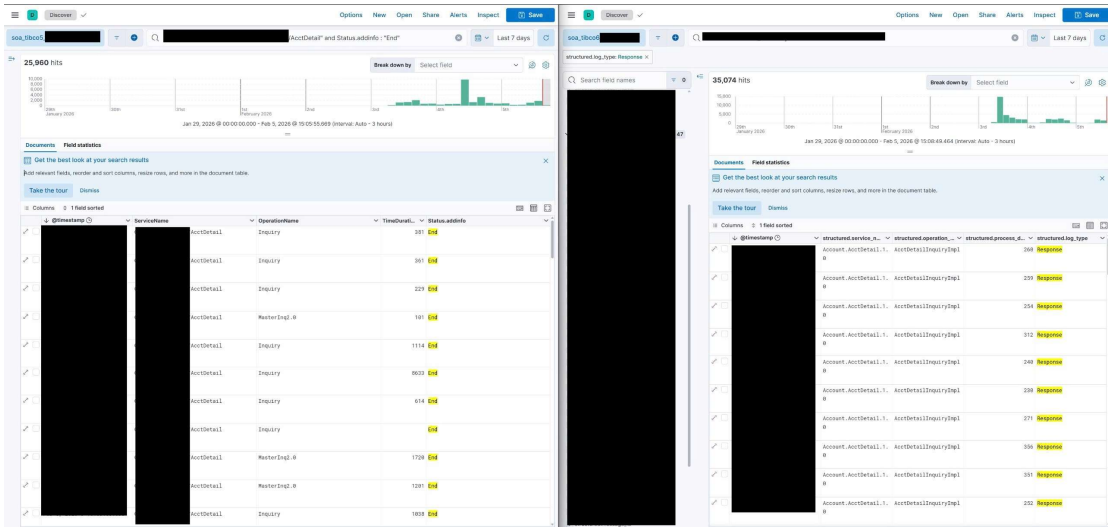


Hình 3.35: Màn hình monitor từng service

Phân tích kết quả: Hệ thống đáp ứng tốt ở ngưỡng 100% và ngưỡng peak TPS kỳ vọng trong điều kiện hạ tầng hoạt động ổn định, các kết nối được duy trì dung quy cách. Thời gian phản hồi ổn định thấp hơn 200ms, riêng một số service có thời gian phản hồi vượt quá kỳ vọng (302,11ms).

Như vậy, ta đã có thể kết luận hệ thống chạy trên Openshift ổn định và có phần nhỉnh hơn hệ thống cũ.

3.6. So sánh thời gian phản hồi



So với TIBCO 6 (Bên phải) thì ta có thể thấy thời gian phản hồi được cải thiện rõ rệt so với TIBCO 5 (bên trái)

3.7. Kết luận chương 3

Kết quả phân tích và thử nghiệm trong chương này cho thấy mô hình triển khai SOA dựa trên TIBCO BW6 kết hợp nền tảng Kubernetes/OpenShift và quy trình CI/CD theo hướng GitOps mang lại những cải thiện rõ rệt so với mô hình truyền thống sử dụng BW5. Hệ thống không chỉ rút ngắn đáng kể thời gian triển khai thông qua tự động hóa toàn bộ vòng đời dịch vụ, mà còn nâng cao khả năng mở rộng động nhờ cơ chế autoscaling linh hoạt của Kubernetes. Bên cạnh đó, việc tích hợp logging tập trung và dashboard giám sát thời gian thực giúp tăng cường khả năng quan sát, hỗ trợ phát hiện và xử lý sự cố nhanh chóng hơn, từ đó giảm đáng kể khối lượng công việc của đội ngũ vận hành.

Nhìn chung, các kết quả thực nghiệm đã chứng minh rằng mô hình đề xuất đáp ứng tốt các yêu cầu về tốc độ, độ tin cậy, tính an toàn và khả năng mở rộng – những tiêu chí cốt lõi của quá trình chuyển đổi số trong ngành ngân hàng. Đây là cơ sở quan trọng để hướng tới việc triển khai rộng rãi kiến trúc SOA hiện đại trên toàn bộ hệ thống, góp phần nâng cao hiệu quả vận hành và chất lượng dịch vụ của ngân hàng trong bối cảnh cạnh tranh và số hóa ngày càng mạnh mẽ.

KẾT LUẬN

1. Các kết quả đạt được của đề án

Đề án đã đạt được nhiều kết quả quan trọng cả về mặt lý luận và thực tiễn. Trước hết, đề án đã xây dựng được nền tảng kiến thức vững chắc về kiến trúc hướng dịch vụ (SOA) và vai trò của containerization trong hiện đại hóa hạ tầng tích hợp. Việc phân tích mối quan hệ giữa SOA – Docker – Kubernetes trong Chương I đã cho thấy sự cần thiết của việc chuyển dịch sang mô hình triển khai dịch vụ linh hoạt, độc lập, có khả năng tự động hóa và tối ưu hóa trong bối cảnh các hệ thống ngân hàng ngày càng mở rộng và phức tạp.

Tiếp theo, Chương II đã đánh giá chi tiết thực trạng trục tích hợp tại ngân hàng thông qua nền tảng TIBCO BusinessWorks 5, đồng thời nhận diện những giới hạn của mô hình kiến trúc đơn khối trong việc đáp ứng yêu cầu về hiệu năng, khả năng mở rộng và chi phí vận hành. Trên cơ sở đó, đề án đã đề xuất mô hình mới dựa trên TIBCO BusinessWorks 6 kết hợp Kubernetes, thể hiện rõ ưu điểm về khả năng triển khai dạng container, tự động mở rộng, áp dụng CI/CD và tăng cường tính linh hoạt của hệ thống. Đặc biệt, đề án cũng chỉ ra các thách thức hiện hữu, tạo tiền đề quan trọng cho việc hoạch định chiến lược chuyển đổi phù hợp.

Ở Chương III, các thử nghiệm và đánh giá thực tế đã chứng minh hiệu quả của mô hình đề xuất: thời gian triển khai được rút ngắn đáng kể nhờ tự động hóa theo hướng GitOps, khả năng mở rộng động được cải thiện mạnh mẽ với autoscaling, và độ tin cậy hệ thống được nâng cao thông qua cơ chế giám sát – logging tập trung. Những kết quả này khẳng định mô hình SOA hiện đại dựa trên BW6 và Kubernetes hoàn toàn phù hợp với yêu cầu vận hành 24/7, khối lượng giao dịch lớn và nhu cầu chuyển đổi số của ngành ngân hàng.

Tổng hợp lại, đề án đã đóng góp một mô hình kiến trúc khả thi, hiện đại, có tính ứng dụng cao, giúp tối ưu hóa vận hành hệ thống tích hợp SOA của ngân hàng. Các kết quả nghiên cứu và thực nghiệm là cơ sở quan trọng để mở rộng triển khai trong tương lai, hướng tới hạ tầng tích hợp linh hoạt, bền vững và đáp ứng tốt chiến lược phát triển dài hạn của BIDV trong kỷ nguyên số.

Quá trình phân tích và thử nghiệm cho thấy việc chuyển đổi từ mô hình BW5 truyền thống sang BW6 kết hợp container hóa và Kubernetes là xu thế tất yếu để nâng cao năng lực công nghệ của các ngân hàng hiện đại. Tuy nhiên, để triển khai thành công, các ngân hàng cần xây dựng lộ trình nâng cấp rõ ràng, bảo đảm sự chuyển đổi không ảnh hưởng đến các dịch vụ đang vận hành. Bên cạnh đầu tư về hạ tầng cloud-native, các ngân hàng cũng cần chú trọng đào tạo nguồn nhân lực về Kubernetes, CI/CD, containerization và các mô hình vận hành DevOps/GitOps nhằm bảo đảm tính chủ động và bền vững trong vận hành. Ngoài ra, cần sớm chuẩn hóa quy trình triển khai, giám sát và bảo mật để hệ thống đạt được tính ổn định, an toàn và hiệu quả lâu dài.

2. Hướng nghiên cứu tiếp theo

Trong tương lai, đề án có thể mở rộng theo các hướng nghiên cứu chuyên sâu hơn như:

Tích hợp các giải pháp API Gateway (Kong, Apigee, Tyk, v.v.) nhằm hoàn thiện kiến trúc tổng thể phục vụ các ứng dụng đa kênh.

Tăng cường nghiên cứu về bảo mật trong môi trường Kubernetes, bao gồm network policy, service mesh, zero-trust, và quản lý khóa/bí mật.

Đánh giá khả năng triển khai trên các nền tảng cloud công cộng như AWS, Azure, GCP nhằm tận dụng các dịch vụ managed Kubernetes.

Nghiên cứu và so sánh chi phí vận hành, hiệu năng thực tế giữa các mô hình triển khai on-premise, hybrid cloud và full cloud-native.

Mở rộng phân tích hiệu năng giữa BW6 và các nền tảng tích hợp khác (Camel, WSO2, MuleSoft) để có góc nhìn đa chiều hơn.

DANH MỤC TÀI LIỆU THAM KHẢO

[1] Viettel IDC (2024). Kubernetes là gì? Chức năng và cơ chế hoạt động của Kubernetes. Viettel IDC, Hà Nội, Việt Nam.

[2] VNG Cloud (2023). 7 lợi ích chính của việc ứng dụng Kubernetes. VNG Cloud, Hà Nội, Việt Nam.

[3] Mustyala Anirudh, Allam Karthik (2023). "Kubernetes Infrastructure and Its Usage in a Banking System", International Journal of Science and Research, 12(5), pp. 1508-1515.

[4] Anonymous (2023). "Cloud-Based Core Banking Systems Using Microservices Architecture", International Journal of Research in Electronics and Computer Engineering, 7(2), pp. 3669-3678.

[5] TIBCO Documentation (2019). Deployment of Order Management on Kubernetes. TIBCO Software Inc., Palo Alto, USA.

[6] TIBCO Community (2020). How to set up TIBCO Platform Dataplane on Kubernetes. TIBCO Software Inc., Palo Alto, USA.

[7] TIBCO YouTube Channel (2016). "TIBCO BusinessWorks Container Edition on Kubernetes". YouTube. Truy cập ngày 27 tháng 7 năm 2025.

[8] TIBCO YouTube Channel (2021). Webinar: Running BusinessWorks on Kubernetes. YouTube. Truy cập ngày 27 tháng 7 năm 2025.

[9] FPT Software (2025). "Kubernetes: No Longer the Secret Sauce for Modern Banking", FPT Software Blog. Truy cập ngày 27 tháng 7 năm 2025.

[10] International Network Group (ING) (2025). Case Study: Using Kubernetes for Container Orchestration in Banking. Truy cập ngày 27 tháng 7 năm 2025.

[11] IJFMR (2019). "Service-Oriented Architecture (SOA) Vs. Microservices", International Journal of Future Microbial Research, 3(1).

[12] Research on Kubernetes Cluster Management (n.d.). "Kubernetes Cluster Management for Cloud Computing Platform", Jurnal Teknik Informatika, Institut Teknologi Sepuluh Nopember. Truy cập ngày 27 tháng 7 năm 2025.

15% Tính tương đồng nói chung

Tổng cộng của tất cả các kết quả trùng khớp, bao gồm cả các nguồn trùng lặp, cho mỗi c...

Đã lọc khỏi Báo cáo

- Mục lục tham khảo
- Văn bản được trích dẫn
- Văn bản được trích dẫn
- Kết quả trùng khớp nhỏ (ít hơn 8 từ)

Nguồn hàng đầu

14% Nguồn Internet
3% Ấn bản
4% Bài tập được nộp (bài của học sinh)

Giảng viên hướng dẫn

Vũ Văn Thỏa

Học viên

Lê Quang Thanh

BỘ KHOA HỌC VÀ CÔNG NGHỆ
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

CỘNG HOÀ XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

BÁO CÁO GIẢI TRÌNH
SỬA CHỮA, HOÀN THIỆN ĐỀ ÁN TỐT NGHIỆP THẠC SĨ

Họ và tên học viên: Lê Quang Thành

Chuyên ngành: Hệ thống Thông tin

Khóa: 2024 đợt I

Tên đề tài: Nghiên cứu ứng dụng công nghệ Kubernetes cho việc triển khai hệ thống SOA trong hệ thống ngân hàng

Người hướng dẫn khoa học: TS. Vũ Văn Thỏa

Ngày bảo vệ: 24/01/2026

Các nội dung học viên đã sửa chữa, bổ sung trong đề án tốt nghiệp theo ý kiến đóng góp của Hội đồng chấm đề án tốt nghiệp:

TT	Ý kiến hội đồng	Sửa chữa của học viên
1	Bổ sung kịch bản kiểm thử để chứng minh giải pháp hiệu quả. Giải thích kết quả	Đã bổ sung ở mục 3.5
2	So sánh thời gian phản hồi, khả năng chịu tải khi chạy nền tảng cũ và mới. Bổ sung thời gian đáp ứng và vấn đề an ninh	Đã bổ sung ở mục 3.5 và 3.6
3	Làm rõ đóng góp trong của đề án trong hệ thống	Đã thêm vào chương 3

Hà Nội, ngày 9 *tháng* 2 *năm* 2026

Ký xác nhận của

CHỦ TỊCH HỘI ĐỒNG CHẤM ĐỀ ÁN	THƯ KÝ HỘI ĐỒNG	NGƯỜI HƯỚNG DẪN KHOA HỌC	HỌC VIÊN
TS. Nguyễn Duy Phương	TS. Đỗ Thị Liên	TS. Vũ Văn Thỏa	Lê Quang Thành