

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



NGUYỄN THỊ MINH THỦY

PHÁT TRIỂN HỆ PHÂN TÁN VỚI KIẾN TRÚC VI DỊCH VỤ

CHUYÊN NGÀNH : HỆ THỐNG THÔNG TIN

MÃ SỐ: 60.48.01.04

LUẬN VĂN THẠC SĨ KỸ THUẬT

(Theo định hướng ứng dụng)

NGƯỜI HƯỚNG DẪN KHOA HỌC: PGS. TS. HÀ HẢI NAM

HÀ NỘI - 2016

Luận văn được hoàn thành tại:

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

Người hướng dẫn khoa học: PGS. TS. Hà Hải Nam

Phản biện 1:

Phản biện 2:

Luận văn sẽ được bảo vệ trước Hội đồng chấm luận văn thạc sĩ tại
Học viện Công nghệ Bưu chính Viễn thông

Vào lúc: ... giờ ngày tháng năm

Có thể tìm hiểu luận văn tại:

- Thư viện của Học viện Công nghệ Bưu chính Viễn thông

MỞ ĐẦU

Tính cấp thiết của đề tài

Kiến trúc hướng vi dịch vụ (Microservice Architectural Style - MAS) là một cách xây dựng hệ thống phần mềm. Một hệ thống vi dịch vụ bao gồm nhiều dịch vụ nhỏ mà giao tiếp với nhau qua mạng. Mỗi dịch vụ có một vòng đời phát triển phần mềm riêng biệt. Để làm việc một cách chính xác, các dịch vụ này phải phụ thuộc vào nhau.

Để đánh giá lợi thế của hệ thống vi dịch vụ luận văn đã nghiên cứu và xây dựng một nguyên mẫu – một sản phẩm phần mềm dựa trên hệ thống vi dịch vụ. Mẫu được xây dựng dựa trên một chức năng đã tồn tại của hệ thống bán hàng “Customer Sales System” (KSS) của công ty Alm Branch. Thông qua nguyên mẫu, luận văn sẽ nghiên cứu các cách để cải thiện và đạt được độ sẵn sàng cao với chi phí bảo trì thấp.

Tổng quan về vấn đề nghiên cứu

Luận văn thạc sĩ này sẽ nghiên cứu các chiến lược về độ sẵn sàng và bảo trì trong một hệ thống microservice. Một nguyên mẫu microservice nhỏ được xây dựng dựa trên một chức năng thực tế “create meeting from absalon” nằm trong hệ thống nguyên khối về quản lý bán hàng (KSS) của Alm Brand. Nguyên mẫu này sẽ là nền tảng cho việc nghiên cứu các cách thức đảm bảo độ sẵn sàng trong những hệ thống có tính rủi ro cao.

Mục đích, đối tượng, phạm vi và phương pháp nghiên cứu

Phương pháp chính được thực hiện trong luận văn thạc sĩ này là sử dụng mô hình lặp đi lặp lại việc tạo nguyên mẫu. Quá trình này sẽ giúp tôi có kinh nghiệm nhanh hơn khi tiếp cận kiểu kiến trúc microservice, cùng với đó các vấn đề nổi bật cũng được xác định sớm để xem xét.

Cấu trúc luận văn

Nội dung của luận văn được trình bày trong ba phần chính như sau:

1. Phần mở đầu
2. Phần nội dung: bao gồm ba chương

CHƯƠNG 1: TỔNG QUAN VỀ KIẾN TRÚC VI DỊCH VỤ

**CHƯƠNG 2: GIẢI PHÁP MÔ HÌNH HÓA HỆ PHÂN TÁN VỚI
KIẾN TRÚC VI DỊCH VỤ**

CHƯƠNG 3: THỬ NGHIỆM VÀ ĐÁNH GIÁ

3. Phần kết luận

CHƯƠNG I. TỔNG QUAN VỀ HỆ THỐNG VI DỊCH VỤ

1.1 Giới thiệu về hệ thống vi dịch vụ

Hệ thống vi dịch vụ (Microservice System – MS) đã nổi lên trong vài năm qua để mô tả một cách đặc biệt của thiết kế các ứng dụng phần mềm như là một hệ thống các dịch vụ được triển khai độc lập và giao tiếp với nhau qua mạng.

1.1.1 Đặc điểm của hệ thống vi dịch vụ (Microservice System)

- a, Được cấu thành từ các dịch vụ*
- b, Được tổ chức dựa trên các giao dịch*
- c, Microservice là một sản phẩm không phải là một dự án*
- d, Dữ liệu được quản lý không tập trung (Decentralized)*
- e, Đa ngôn ngữ*
- f, Triển khai liên tục*
- g, Khép kín*
- h, Liên lạc ngoài dịch vụ*
- i, Nguyên tắc chuyên biệt*

1.1.2 Ưu điểm của kiến trúc hệ thống vi dịch vụ

- Giảm thiểu sự gia tăng phức tạp rối rắm hệ thống lớn
- Chia nhỏ ứng dụng một khối công kênh thành các dịch vụ nhỏ dễ quản lý, bảo trì nâng cấp, tự do chọn, nâng cấp công nghệ mới
- Mỗi dịch vụ nhỏ sẽ định ra ranh giới rõ ràng dưới dạng RPC hay API hướng thông điệp

1.1.3 Nhược điểm của kiến trúc hệ thống vi dịch vụ

Nhược điểm đầu tiên của microservices cũng chính từ tên gọi của nó. Microservice nhấn mạnh kích thước nhỏ gọn của dịch vụ. Một số lập trình đề xuất dịch vụ siêu nhỏ cỡ dưới 100 dòng code. Chia quá nhiều sẽ dẫn đến manh

mún, vụn vặt, khó kiểm soát. Việc lưu dữ liệu cục bộ bên trong những dịch vụ quá nhỏ sẽ khiến dữ liệu phân tán quá mức cần thiết.

1.2 So sánh kiến trúc vi dịch vụ với kiến trúc liên quan

1.2.1 Kiến trúc hệ thống nguyên khối (Monolithic System)

- a, Một cơ sở mã nguồn*
- b, Ngôn ngữ cụ thể*
- c, Mở rộng ngang*
- d, Kiến trúc 3 lớp*
- e, Tiến trình đơn*
- f, Các thành phần đan xen*
- g, Vòng đời triển khai dài*

1.3 Mô hình nguyên mẫu

- Nghiên cứu chiến lược đảm bảo độ sẵn sàng cho hệ thống vi dịch vụ sử dụng kiến trúc vi dịch vụ.
- Nghiên cứu chiến lược thay đổi giúp giảm thiểu chi phí bảo trì thông qua việc thiết kế và cài đặt một nền tảng vi dịch vụ chung nào đó.

1.4 Các đặc trưng của giao dịch trong hệ thống phân tán

Sử dụng giao dịch (transactions) trong các hệ thống n-lớp truyền thống như là hệ thống nguyên khối là tương đối đơn giản. Khi chia một hệ thống như vậy thành các dịch vụ độc lập như là các microservices, sẽ có nhiều vấn đề cần xem xét đến.

1.4.1 Mô hình “hai giai đoạn”

Một giải pháp được sử dụng rộng rãi là sử dụng xác nhận thực hiện “hai giai đoạn”, thành phần điều phối đầu tiên sẽ kiểm tra nếu tất cả các dịch vụ tham gia đã sẵn sàng xác nhận thực hiện (commit). Trên cơ sở kết quả bỏ phiếu

của các dịch vụ để đảm bảo tính thống nhất để hủy bỏ hoặc xác nhận thực hiện với tất cả các dịch vụ.

1.4.2 Mô hình Saga

Một giải pháp khác để đạt được sự nhất quán giữa các dịch vụ là sử dụng mô hình Saga.

Trong mô hình này chứa một dịch vụ điều phối, là thành phần chính có trách nhiệm kiểm soát sự tham gia của các microservice. Mô hình này được áp dụng theo nguyên tắc, tất cả các dịch vụ tham gia vào quá trình xử lý một hành động đều có hoạt động đền bù. Khi một hoặc nhiều dịch vụ tham gia thất bại khi xử lý, dịch vụ điều phối sẽ yêu cầu các dịch vụ còn lại thực hiện hoạt động bồi thường tương ứng.

1.4.3 Mô hình đặt chỗ (Reservation)

Mô hình đặt chỗ là giải pháp thứ ba để đạt được sự nhất quán giữa các dịch vụ. Không giống với mô hình Saga, mô hình đặt chỗ không chứa một thành phần điều phối trung

1.5 Kết luận chương 1

Chương 1 đã đưa ra được các khái niệm chung nhất và các đặc điểm về hệ thống vi dịch vụ cũng như hệ thống nguyên khối. Ngoài ra, chương I còn nêu lên được mục đích xây dựng mô hình nguyên mẫu cũng như đưa ra một mô hình nguyên mẫu được xây dựng một cách tuần tự lặp đi lặp lại được sử dụng xuyên suốt luận văn. Đồng thời, chương I của luận văn đã đưa ra các loại giao dịch phân tán để làm nền tảng cho sự phân tích hệ thống của chương II.

CHƯƠNG II. GIẢI PHÁP MÔ HÌNH HÓA HỆ PHÂN TÁN VỚI KIẾN TRÚC VI DỊCH VỤ

2.1 Mô hình hóa các dịch vụ

Nguyên mẫu là sản phẩm phần mềm cuối cùng của luận văn thạc sỹ này. Thông qua nguyên mẫu, luận văn sẽ nghiên cứu cách cách để cải thiện và đạt được độ sẵn sàng cao, chi phí bảo trì thấp. Trong quá trình phát triển nguyên mẫu, vấn đề cần xem xét về làm thế nào để đạt được kết quả mong muốn về độ sẵn sàng và chi phí bảo trì sẽ được trình bày chi tiết và các giải pháp được lựa chọn cài đặt được coi là một phần của nguyên mẫu. Chức năng “create meeting from Absalon” hiện tại sẽ bao gồm một số microservice tiềm năng.

2.2 Giải pháp tích hợp trong hệ phân tán với kiến trúc vi dịch vụ

2.2.1 Phân tích kiến trúc hệ thống bán hàng (KSS)

Thành phần máy chủ KSS là một phần trong một hệ thống ứng dụng lớn hơn là hệ thống bán hàng KSS hiện đang được khai thác. Hệ thống bán hàng (KSS) là công cụ chính được sử dụng bởi các giao dịch viên tại các trung tâm tiếp thị qua điện thoại của Alm Brand.

a, Máy chủ KSS

Máy chủ KSS là thành phần chính và là backend của hệ thống KSS. Nó cung cấp tất cả các chức năng thông qua danh sách các web services. Tất cả các yêu cầu và tương tác (nội bộ và từ bên ngoài) với hệ thống KSS đều thông qua thành phần máy chủ KSS.

b, Absalon

Absalon là một ứng dụng frontend và là công cụ chính được sử dụng của các đại lý bảo hiểm. Nó chứa một thành phần KSS chuyên dụng để tích hợp với máy chủ KSS cung cấp chức năng hiển thị, sửa đổi lịch của các đại lý KSS và thông tin về khách hàng.

2.2.2 Phân tích thành phần chức năng “Create meeting from Absolon”

Chức năng “Create meeting from Absolon” là một trong những chức năng phức tạp nhất và có tác động tới hầu hết các thành phần trong hệ thống KSS. Nó cũng là một trong những chức năng sử dụng đầu tiên và là chức năng chính khi một người dùng tương tác với hệ thống. Do vậy chức năng này là một ứng cử viên lý tưởng cho việc tạo nguyên mẫu thử nghiệm.

Nhiệm vụ rõ ràng của chức năng này là tạo ra một cuộc họp và đặt một cuộc gặp trong lịch của các đại lý bảo hiểm trên cả hệ thống lịch của hệ thống Notes và KSS. Lưu ý là “Create meeting from Absolon” có thể được khởi tạo từ cả 02 hệ thống là Absolon lẫn KSS mobile.

“Create meeting from Absolon” được khởi tạo bằng cách gọi web services với tên gọi là: AbsalonFactoryWS.bookMeeting. Các chức năng chính của “Create meeting from Absolon” được mô tả như sau:

1. Lưu dữ liệu (Persist)
2. Tạo cuộc hẹn (CreateAppointment)
3. Tạo cuộc hẹn (CreateAppointment)
4. Hợp nhất (Merge)
5. Khởi tạo cuộc gặp (InitiateMeeting)
6. Tạo thư mời (PrintGenericLetter)
7. Gửi thư mời (SendEmail)
8. Ghi nhận sự kiện (SetHaendelse)
9. Lưu dữ liệu (Persist)
10. Thêm cuộc gặp (Insert)

2.2.3 Phân tích nguyên mẫu hệ thống KSS

a, Kiến trúc thành phần trong nguyên mẫu của hệ thống KSS

b, Kiến trúc và khung hệ thống KSS

✓ Kiến trúc các thành phần thư viện dùng chung

- ✓ *Kiến trúc các thành phần thư viện cơ sở*
- ✓ *Kiến trúc các thành phần Microservice và Máy chủ KSS*

c, Giao thức đặt chỗ với Microservice cho hệ thống KSS

2.2.4 Sơ đồ lớp chi tiết

2.2.5 Biểu đồ tuần tự

2.3 Kết luận chương 2

Một mục tiêu của nguyên mẫu là xây dựng lại một phần của phiên bản đang hoạt động hiện tại của chức năng “create meeting from Absalon”. Và trong suốt quá trình đó, đồng thời xây dựng một khung làm việc (framework) microservice chung và một giao thức có thể áp dụng đáp ứng như yêu cầu của Alm Brand

CHƯƠNG III. CÀI ĐẶT VÀ THỬ NGHIỆM

3.1 Cài đặt các vi dịch vụ

3.1.1 Apache Kafka

3.1.2 Cài đặt các dịch vụ

a. **Producer Kafka**, dưới đây là đoạn mã nguồn để phát triển một producer

```
private ProducerConfig getKafkaProducerConfiguration() {
    Properties props = new Properties();
    props.put("metadata.broker.list", "ubuntu:9092");
    props.put("serializer.class", "kafka.serializer.StringEncoder");
    props.put("zookeeper.session.timeout.ms", "5000");
    props.put("request.timeout.ms", "5000");
    props.put("request.required.acks", "1");
    props.put("auto.commit.interval.ms", "1000");
    return new ProducerConfig(props);
}
```

đơn giản để phân phát thông điệp văn bản tới một hàng đợi Kafka.

b. Kafka Consumer

Kafka Consumer được bắt đầu như một luồng chạy riêng biệt với một vòng lặp vô hạn lắng nghe các thông điệp mới.

- **Cấu hình consumer:** Đoạn mã dưới đây là cấu hình cho consumer. Đối tượng consumer kết nối tới một máy chủ zookeeper (trung gian), có chức năng phân.
- **Lớp Cosumer:** Lớp consumer dưới đây thể hiện mã nguồn đầy đủ phiên bản đầu tiên của nguyên mẫu Kafka consumer. Với kết quả đầu ra đơn giản là hiển thị bất kỳ thông điệp nào lấy từ hàng đợi “verification-topic” được lớp producer sử dụng để phân phát thông điệp

3.2 Cấu hình lớp Cache

Cấu hình lớp Cache được thực hiện theo cách sau đây:

```
publicclass Cache {

    privatestatic Cache instance;
    private List<Reservation>reservationCache;

    private Cache() {}

    publicstatic Cache getInstance() {
        if(instance == null) {
            instance = new Cache();
        }
        returninstance;
    }

    private List<Reservation> getRequestEventCache() {
        if(reservationCache == null) {
            reservationCache = new ArrayList<Reservation>();
        }
        returnreservationCache;
    }

    publicvoid insert(Reservation reservation) {
        getRequestEventCache().add(reservation);
    }

    publicvoid update(Reservation reservation) {
        for(Reservation reservation1 : getRequestEventCache()) {
            if(reservation1.getUuid().equals(reservation.getUuid())) {
                reservation1.setApplication(reservation.getApplication());
                reservation1.setData(reservation.getData());
                reservation1.setEntryDate(reservation.getEntryDate());
                reservation1.setPrimaryKey(reservation.getPrimaryKey());
            }
        }
    }

    publicvoid remove(String uuid) {
        Reservation reservation = getByCorrelationId(uuid);

        if(reservation != null) {
```

3.3 Cài đặt thực hiện MicroserviceWrapper.onMessage

Dưới đây là phương thức chính trong khung (framework) microservice , phương thức onMessage. Nó xử lý tất cả các thông điệp nhận được trong framework. Tại đây các thông điệp được phân tích để xác định loại thông điệp. Sau đó phương thức onMessage sẽ gọi phương thức xử lý tương ứng

```
@Override
public final void onMessage(String message) throws SystemException {
    JsonObject jsonObj = null;

    ConfirmationEvent confirmationEvent = null;
    ResponseEvent responseEvent = null;
    RequestEvent requestEvent = null;
    // Will happen if the message can't be passed to a JsonObject.
    try {
        jsonObj = new Gson().fromJson(message, JsonElement.class).getAsJsonObject();
    } catch (Exception e) {
        requestEvent = new RequestEvent(); // Dummy - Event is an abstract class, to
one                                     // of the others must do.

        requestEvent.setJson(message);
        handleEvent(requestEvent, false);
        return;
    }
    if (jsonObj.get("reserveKey") != null) {
        confirmationEvent = new GsonUtils<ConfirmationEvent>().fromJSON(message,
ConfirmationEvent.class);
        confirmationEvent.setJson(message);
        if (jsonObj.getAsJsonObject("object") != null) {
            confirmationEvent.setObject(jsonObj.getAsJsonObject("object"));
        }
        handleEvent(confirmationEvent, true);
    } elseif (jsonObj.get("resultCode") != null) {
        responseEvent = new GsonUtils<ResponseEvent>().fromJSON(message,
ResponseEvent.class);
        responseEvent.setJson(message);
        if (jsonObj.getAsJsonObject("object") != null) {
            responseEvent.setObject(jsonObj.getAsJsonObject("object"));
        }
        handleEvent(responseEvent, true);
    } elseif (jsonObj.get("responseQueue") != null) {
        requestEvent = new GsonUtils<RequestEvent>().fromJSON(message,
RequestEvent.class);
        requestEvent.setJson(message);
        if (jsonObj.getAsJsonObject("object") != null) {
            requestEvent.setObject(jsonObj.getAsJsonObject("object"));
        }
        handleEvent(requestEvent, true);
    } else {
        logger.error("Unknown message: " + message);
    }
}
```

handleEvent.

3.4 Triển khai thành phần xử lý ngắt

Thành phần xử lý ngắt được phát triển như một phần của lần thực hiện đầu tiên. *Lớp - CircuitBreaker*

Đây là interface sẽ định nghĩa các chức năng của thành phần xử lý ngắt

- *Lớp CircuitBreakerImpl*

Khi yêu cầu đã được xử lý xong, luồng (thread) sẽ được ngắt.

3.5 Cài đặt hệ thống

3.5.1 Cài đặt máy chủ và nguyên mẫu

3.5.2 Cài đặt môi trường JDeveloper của Oracle

3.5.3 Cài đặt Kafka (Máy ảo)

3.5.4 Cài đặt RabbitMQ

3.5.5 Cài đặt SoapUI

3.5.6 Cài đặt Database – MySQLA

3.5.7 Chạy chức năng

3.6 Kết luận chương 3

Chương 3 đã nghiên cứu và trình bày phương pháp cài đặt các dịch vụ theo các bước một cách chi tiết và rõ ràng nhất. Ngoài ra, Chương 3 còn nghiên cứu cách chạy các chức năng và triển khai các thành phần xử lý của hệ thống vi dịch vụ.

KẾT LUẬN

1. Kết quả đạt được

Luận văn đã phát triển một phiên bản cụ thể cho chức năng “create meeting from Absalon”, để có được kinh nghiệm với kiểu kiến trúc microservice và nghiên cứu giải pháp để đảm bảo một mức độ cao về độ sẵn sàng cũng như giảm chi phí bảo trì cần thiết trên các microservice thông qua việc sử dụng lại.

Đồng thời, luận văn làm rõ 2 nội dung sau đây:

a, Nội dung thứ nhất

Nghiên cứu các chiến lược đảm bảo độ sẵn sàng cho trường hợp của hệ thống KSS sử dụng kiến trúc microservice. Đặc biệt ta sẽ triển khai thử nghiệm cùng với mô hình xử lý ngắt (circuit breaker), mô hình vách ngăn (bulkhead) và so sánh kết quả về độ sẵn sàng với kiến trúc nguyên khối của hệ thống KSS hiện tại.

b, Nội dung thứ hai

Nghiên cứu các chiến lược sửa đổi cho trường hợp hệ thống KSS, tập trung vào việc giảm chi phí bảo trì thông qua việc thiết kế và triển khai thực hiện một khung (framework) microservice chung.

2. Hạn chế

- Chi phí vận hành
- Sự phức tạp của hệ thống phân tán
- Thách thức đối với khả năng thử nghiệm

3. Hướng phát triển

Nghiên cứu giải pháp tiềm năng khác hay một chiến lược để giải quyết vấn đề giao dịch phân tán là thay vì yêu cầu nhất quán ngay lập và bổ sung các

chiến lược dự phòng khi gặp sự cố. Chiến lược này cũng đang được sử dụng bởi eBay [29].

TÀI LIỆU THAM KHẢO**Tài liệu Tiếng Anh**

- [1] Alain Abran, J W Moore, P Bourque, R Dupuis, and L L Tripp, 2014. “Software Engineering Body of Knowledge”. In: IEEE Computer Society, Angela Burgess
- [2] C.D. Nielsen, 2015, - Investigate availability and maintainability within a microservice architecture
- [3] J. Kreps LinkedIn Corp, N. Narkhede LinkedIn Corp, J. Rao LinkedIn Corp. Apache Kafka, 2015, - A Distributed Messaging System for Log Processing
- [4] L. Bass, P. Clements & R. Kazman, 2003, - Software Architecture in Practice (2nd Edition)
- [5] Martin Fowler and James Lewis, 2015, - Microservices
- [6] Monolithic Design (Accessed on 04/02/2015)
- [7] Scalability Best Practices: Lessons from eBay (Accessed 08/06/2015)

Website tham khảo

- [8] <http://martinfowler.com/articles/microservices.html>
- [9] <http://omega.cs.iit.edu/~ipro329/methodology.html>
- [10] <http://hosteddocs.ittoolbox.com/PM043004.pdf>
- [11] http://www.sei.cmu.edu/library/assets/whitepapers/SOA_Terminology.pdf
- [12] http://www.w3.org/TR/ws-arch/#service_oriented_architecture
- [13] <http://kafka.apache.org/>