**TRƯỜNG ĐẠI HỌC TÀI NGUYÊN VÀ MÔI TRƯỜNG HÀ NỘI**

**KHOA CÔNG NGHỆ THÔNG TIN**

****

**BÁO CÁO THỰC TẬP TỐT NGHIỆP**

**ĐỀ TÀI**

**NGHIÊN CỨU GIAO THỨC AMQP TRONG TRUYỀN TẢI DỮ LIỆU VỚI RABBITMQ**

**Đơn vị tập tốt nghiệp : Công ty cổ phần SmartOSC**

**Cán bộ hướng dẫn : Nguyễn Hải Hà**

**Sinh viên thực hiện : Đặng Văn Lực**

**Lớp : ĐH6C1**

**Hệ Đại học : Chính qui**

**Khóa học : 2016 - 2020**

**Hà Nội, tháng 03/2020**

# **LỜI CẢM ƠN**

Trên thực tế không có sự thành công nào mà không gắn liền với những sự giúp đỡ mọi người dù ít hay nhiều, dù trực tiếp hay gián tiếp. Trong suốt thời gian học tập nhất là trong quá trình thực tập, em đã nhận được rất nhiều sự quan tâm giúp đỡ nhiệt tình của …

Vì thời gian, điều kiện còn có hạn, em đã cố gắng rất nhiều để hoàn thành đợt thực tập tốt nghiệp, nhưng vẫn còn nhiều hạn chế và không thể tránh khỏi những thiếu sót, mong thầy cô và các bạn có những ý kiến đóng góp để em có thể hoàn thiện và phát triển đề tài hơn.

Em xin chân thành cảm ơn!

**Sinh viên thực hiện**

**Lực**

**Đặng Văn Lực**

**MỤC LỤC**

# **DANH SÁCH CHỮ VIẾT TẮT**

|  |  |  |
| --- | --- | --- |
| **Từ viết tắt** | **Tên tiếng Anh** | **Tên tiếng Việt** |
| IaaS | Infrastructure as a Services | Hạ tầng như một dịch vụ |
| PaaS | Platform as a Services | Nền tảng như một dịch vụ |

**DANH MỤC HÌNH ẢNH**

Hình 1.1: Giao diện A…………………………………………………………………5

# **PHẦN MỞ ĐẦU**

1. **Tính cấp thiết của đề tài**

Ngày nay công nghệ thông tin phát triển rất nhanh chóng, nắm giữ vai trò quan trọng trong việc thúc đấy sự tăng trưởng kinh tế, với sự ra đời của rất nhiều công nghệ mới, các dịch vụ CNTT đáp ứng nhu cầu của người dùng cũng như là các doanh nghiệp,... Tuy nhiên, trong giai đoạn suy thoái kinh tế như hiện nay, thì việc ứng dụng một công nghệ hay một dịch vụ CNTT đáp ứng việc quản lý tốt, hiệu quả dữ liệu của riêng công ty cũng như dữ liệu khách hàng, đối tác là một trong những bài toán được ưu tiên hàng đầu cho doanh nghiệp. Để có thể quản lý được nguồn dữ liệu đó, ban đầu các doanh nghiệp phải đầu tư, tính toán rất nhiều loại chi phí như chi phí cho phần cứng, phần mềm, mạng, chi phí cho quản trị viên, chi phí bảo trì, sửa chữa,…Ngoài ra họ còn phải tính toán khả năng mở rộng, nâng cấp thiết bị; phải kiểm soát việc bảo mật dữ liệu cũng như tính sẵn sàng cao của dữ liệu. Để giải quyết vấn đề trên thì chúng ta thấy được rằng nếu có một nơi tin cậy giúp các doanh nghiệp quản lý tốt nguồn dữ liệu đó, các doanh nghiệp sẽ không còn quan tâm đến cơ sở hạ tầng, công nghệ mà chỉ tập trung chính vào công việc kinh doanh của họ thì sẽ mang lại cho họ hiệu quả và lợi nhuận ngày càng cao hơn. Trong một hệ thống phân tán (distributed system), có rất nhiều thành phần. Nếu muốn các thành phần này giao tiếp được với nhau thì chúng phải biết nhau. Nhưng điều này gây rắc rối cho việc viết code. Một thành phần phải biết quá nhiều đâm ra rất khó maintain, debug.Giải pháp ở đây là thay vì các liên kết trực tiếp, khiến các thành phần phải biết nhau thì sử dụng một liên kết trung gian qua một message broker. Với sự tham gia của message broker thì producer sẽ không hề biết consumer. Nó chỉ việc gửi message đến các queue trong message broker. Consumer chỉ việc đăng ký nhận message từ các queue này.

Tất nhiên, có thể có một giải pháp là sử dụng database để lưu các message trong các temporary table.Tuy nhiên xét về hiệu năng thì không thể bằng message broker vì một số lý do: Tần xuất trao đổi message cao sẽ làm tăng load của database, giảm performance đáng kể. Trong môi trường multithread, database cần có cơ chế lock.Lock cũng làm giảm performance. Sử dụng message broker sẽ không có vấn đề này.

1. **Tổng quan về đề tài nghiên cứu**

Hiện nay, việc ứng dụng công nghệ thông tin ở các doanh nghiệp Việt Nam đã trở nên phổ biến, và điện toán đám mây đã bắt đầu có những tín hiệu lạc quan. Tuy nhiên hầu hết các doanh nghiệp đang sử dụng các dịch vụ điện toán đám mây hiện nay là các công ty công nghệ hàng đầu trong nước như: FPT, Quang Trung, … còn đại đa số các doanh nghiệp vừa và nhỏ vẫn còn e dè với điện toán đám mây, vẫn chưa hiểu rõ hết về mô hình này nên việc sử dụng các dịch vụ điện toán đám mây vẫn còn hạn chế.

Vừa qua, các "đại gia" công nghệ như Microsoft, IBM, HP, Intel ... đua nhau quảng cáo rầm rộ các sản phẩm, dịch vụ, giải pháp sử dụng công nghệ ảo hóa với những cam kết rằng ảo hóa là sự phát triển cho tương lai, giúp các doanh nghiệp nâng cao hiệu quả với chi phí thấp nhất.... trong bối cảnh phần lớn doanh nghiệp Việt Nam "lơ mơ" về khái niệm này.

Theo khảo sát gần đây của Enterprise Strategy Group, tại thị trường Mỹ, 28% DN có kế hoạch sử dụng môi trường ảo hóa sẽ thực hiện ảo hóa máy chủ trong vòng 6 tháng tới và 42% có kế hoạch khai thác ảo hóa trong năm sau. Các ban ngành IT tại Mỹ đang sử dụng ảo hóa đã ảo hóa 24% số máy chủ và dự kiến con số này sẽ tăng lên 45% vào năm 2009. Ở những quốc gia phát triển như Singapore đã có khoảng 40% DN trang bị kỹ thuật này.

VMware ESXI Server đã nhanh chóng chiếm lĩnh thị trường ảo hóa với 60% máy chủ ảo hóa là sử dụng VMware ESX Server. Tiếp theo là đến Windows Server 2008 tích hợp Windows Virtualization.

Em nhận thấy đây là một đề tài hay, bổ ích và vô cùng thực tế, có thể cung cấp cho em kiến thức về việc giao tiếp giữa các thành phần trong một hệ thống lớn.

1. **Mục đích nghiên cứu**

* Tìm hiểu kiến trúc của giao thức AMQP
* Tìm hiểu về AMQP với RabbitMQ

1. **Đối tượng nghiên cứu và phạm vi nghiên cứu**

*Đối tượng nghiên cứu gồm có:*

* Cơ sở lý thuyết về các giao thức truyền tải dữ liệu thông dụng trong internet
* RabbitMQ và AMQP

*Phạm vi nghiên cứu:*

* Khái niệm, đặc điểm, cách thức hoạt động, mô hình triển khai, ưu điểm, nhược điểm của các giao thức truyền tải dữ liệu thông dụng trong internet.
* Khái niệm, kiến trúc hoạt động của AMQP với RabbitMQ

1. **Phương pháp nghiên cứu**

* Phương pháp nghiên cứu lý thuyết : tiến hành thu nhập các tài liệu, thông tin liên quan đến đề tài.
* Tổng hợp và phân tích tích tài liệu để đưa ra cơ sở lý thuyết về phương thức truyền tải AMQP với RabbitMQ
* Xây dựng demo về việc sử dụng giao thức AMQP với RabbitMQ

1. **Những đóng góp của báo cáo**

Đồ án hoàn thành sẽ có những đóng góp chủ yếu sau đây:

* Hệ thống hóa những vấn đề lý thuyết cơ bản về các giao thức truyền tải dữ liệu thông dụng trong internet.
* Một số vấn đề cơ bản về giao thức AMQP với RabbitMQ.
* Cách sử dụng giao thức AMQP trong RabbitMQ.

1. **Kết cấu của báo cáo**

*Đồ án gồm có 3 chương như sau:*

**Chương 1: Tổng quan về các giao thức truyền tải dữ liệu thông dụng trong internet**

Chương này trình bày tổng quan về các giao thức truyền tải dữ liệu thông dụng trong internet; ưu điểm và nhược điểm của từng phương thức truyền tải.

**Chương 2**: **AMQP với RabbitMQ.**

Chương này sẽ trình bày về microservice ,sử dụng phương thức truyền tải AMQP với RabbitMQ

**Chương 3: Xây dựng demo sử dụng phương thức AMQP trong RabbitMQ với AspCore 3.1**

Trong chương này giới thiệu về ASP.NET Core và dựng demo trên framework này

# **CHƯƠNG 1: TỔNG QUAN VỀ CÁC GIAO THỨC TRUYỀN TẢI DỮ LIỆU THÔNG DỤNG TRONG INTERNET**

**Internet of Things** (IoTs) là thuật ngữ rất phổ biến hiện nay, dùng để chỉ một tập hợp các thiết bị có khả năng kết nối internet và kết nối được với nhau để thực hiện một công việc nào đó. Để khai thác hết được tiềm năng của mô hình IoT, các thiết bị kết nối cần phải giao tiếp bằng các giao thức nhẹ mà không làm tiêu tốn quá nhiều tài nguyên của CPU. Dưới đây cung cấp tổng quan về 5 giao thức truyền tải dữ liệu phổ biến được sử dụng trong các mô hình Internet of Things.

* 1. **MQTT (Message Queue Telemetry Transport)**

MQTT là một giao thức mã nguồn mở để truyền các messages giữa nhiều Client (Publisher và Subscriber) thông qua một Broker trung gian, được thiết kế để đơn giản và dễ dàng triễn khai. Kiến trúc MQTT dựa trên Broker trung gian và sử dụng kết nối TCP long-lived từ các Client đến Broker.

MQTT hỗ trợ tổ chức hệ thống theo các Topics có tính phân cấp, như một hệ thống tập tin (vd: /Home/kitchen/humidity), cung cấp nhiều lựa chọn điều khiển và QoS (Quality of Service).

MQTT là một giao thức khá nhẹ nên có thể được sử dụng cho truyền thông 2 chiều thông qua các mạng có độ trễ cao và độ tin cậy thấp, nó cũng tương thích với các thiết bị tiêu thụ điện năng thấp.



Ưu điểm :

* Kết nối riêng rẽ
* Khả năng mở rộng



Nhược điểm :

* Thời gian tách biệt (Time decoupling)
* Đồng bộ riêng rẽ (Synchronization decoupling)
* Điểm trung gian (broker) không cần thông báo về trạng thái gửi thông điệp. Do đó không có cách nào để phát hiện xem thông điệp đã gửi đúng hay chưa.
* Publisher không hề biết gì về trạng thái của subscribe và ngược lại. Vậy làm sao chúng ta có thể đảm bảo mọi thứ đều ổn
* Những kẻ xấu (malicious publisher) có thể gửi những thông điệp xấu, và các subcribers sẽ truy cập vào những thứ mà họ không nên nhận

Cơ chế tổng quan

* MQTT hoạt động theo cơ chế client/server, nơi mà mỗi cảm biến là một khách hàng (client) và kết nối đến một máy chủ, có thể hiểu như một điểm trung gian (broker), thông qua giao thức TCP (Transmission Control Protocol). Broker chịu trách nhiệm điều phối tất cả các thông điệp giữa phía gửi đến đúng phía nhận.
* MQTT là giao thức định hướng bản tin. Mỗi bản tin là một đoạn rời rạc của tín hiệu và broker không thể nhìn thấy. Mỗi bản tin được publish một địa chỉ, có thể hiểu như một kênh (Topic). Client đăng kí vào một vài kênh để nhận/gửi dữ liệu, gọi là subscribe. Client có thể subcribe vào nhiều kênh. Mỗi client sẽ nhận được dữ liệu khi bất kì trạm nào khác gửi dữ liệu vào kênh đã đăng kí. Khi một client gửi một bản tin đến một kênh nào đó gọi là publish

Ví dụ :



* 1. **CoAP (Constrained Applications Protocol)**

CoAP là một giao thức truyền tải tài liệu theo mô hình client/server dự trên internet tương tự như giao thức HTTP nhưng được thiết kế cho các thiết bị ràng buộc. Giao thức này hỗ trợ một giao thức one-to-one để chuyển đổi trạng thái thông tin giữa client và server.

CoAP sử dụng UDP (User Datagram Protocol), không hỗ trợ TCP, ngoài ra còn hỗ trợ địa chỉ broadcast và multicast, truyền thông CoAP thông qua các datagram phi kết nối (connectionless) có thể được sử dụng trên các giao thức truyền thông dựa trên các gói.

UDP có thể dễ dàng triển khai trên các vi điều khiển hơn TCP nhưng các công cụ bảo mật như SSL/TSL không có sẵn, tuy nhiên ta có thể sử dụng Datagram Transport Layer Security (DTLS) để thay thế..



Cơ chế tổng quan :

* CoAP theo mô hình client/server. Client gửi yêu cầu đến máy chủ, sau đó máy chủ gửi lại phản hồi. Client có thể GET, PUT, POST và DELETE các tài nguyên.
* CoAP có tính linh động và hỗ trợ đàm phán nội dung. Điều này cho phép client và máy chủ có thể nâng cấp, thêm mới một cách độc lập mà ko ảnh hưởng gì đến phía còn lại.
* CoAP đưa ra các yêu cầu quan sát tài nguyên. Cả hai bên đều có thể tác động hoặc xóa các yêu cầu quan sát. Khi cờ quan sát được thiết lập, máy chủ vẫn có thể tiếp tục hồi đáp sau khi các dữ liệu đã truyền đi.
* Với CoAP, máy chủ cung cấp một hệ thống các tài nguyên cho phép client khám phá tài nguyên và các loại phương tiện truyền thông.
* Trong CoAP một nút cảm biến thường là một máy chủ. Chúng có khả năng nhận các gói tin gửi đến để hoạt động đúng đằng sau NAT, thiết bị đầu tiên phải gửi yêu cầu đến máy chủ, như được thực hiện trong LWM2M, cho phép các router liên kết chúng lại.

Hạn chế :

* Sử dụng CoAP khi HTTP quá tải băng thông : Việc sử dụng CoAP không lớn như HTTP, vì vậy nó có thể giới hạn các tùy chọn phần mềm và phần cứng của ta. Có các giải pháp để chuyển đổi các thông báo CoAP sang và từ HTTP giúp các giải pháp CoAP tương thích hơn.
  1. **AMQP (Advanced Message Queue Protocol)**

****

AMQP là một giao thức làm trung gian cho các gói tin trên lớp ứng dụng với mục đích thay thế các hệ thống truyền tin độc quyền và không tương thích. Các tính năng chính của AMQP là định hướng message, hàng đợi, định tuyến (bao gồm point-to-point và publish-subscribe) có độ tin cậy và bảo mật cao. Các hoạt động sẽ được thực hiện thông qua broker, nó cung cấp khả năng điều khiển luồng (Flow Control).

Một trong các Message Broker phổ biến là RabbitMQ, được lập trình bằng ngôn ngữ Erlang, RabbitMQ cung cấp cho lập trình viên một phương tiện trung gian để giao tiếp giữa nhiều thành phần trong một hệ thống lớn.

Không giống như các giao thức khác, AMQP là một giao thức có dây (wire-protocol), có khả năng diễn tả các message phù hợp với định dạng dữ liệu, có thể triển khai với rất nhiều loại ngôn ngữ lập trình.

AMQP kết nối qua:

* Tổ chức – ứng dụng trong các tổ chức khác nhau
* Công nghệ – ứng dụng trên các nền tảng khác nhau
* Thời gian: Hỗ trợ giao tiếp bất đồng bộ
* Không gian – hoạt động ở khoảng cách xa hoặc trên các mạng nghèo nàn



* 1. **DDS (Data Distribution Service)**



DDS là một ngôn ngữ trung gian dựa vào dữ liệu tập trung được sử dụng để cho phép khả năng mở rộng, thời gian thực, độ tin cậy cao và trao đổi dữ liệu tương tác.

Đây là một giao thức phi tập trung (broker-less) với truyền thông ngang hàng trực tiếp theo kiểu peer-to-peer giữa các publishers và subscribers và được thiết kế để trở thành một ngôn ngữ và hệ điều hành độc lập. DDS gửi và nhận dữ liệu, sự kiện, và thông tin lệnh trên UDP nhưng cũng có thể chạy trên các giao thức truyền tải khác như IP Multicast, TCP / IP, bộ nhớ chia sẻ … DDS hỗ trợ các kết nối được quản lý many-to-many theo thời gian thực và ngoài ra còn hỗ trợ dò tìm tự động (automatic discovery). Các ứng dụng sử dụng DDS cho truyền thông được tách riêng và không yêu cầu sự can thiệp từ các ứng dụng của người dùng, có thể đơn giản hóa việc lập trình mạng phức tạp. Các tham số QoS được sử dụng để xác định các cơ chế tự dò tìm của nó được thiết lập một lần.

* 1. **XMPP (Extensible Messaging và Presence Protocol)**



XMPP (trước đây gọi là “Jabber”) là giao thức truyền thông dùng cho định hướng tin nhắn trung gian dựa trên ngôn ngữ XML.

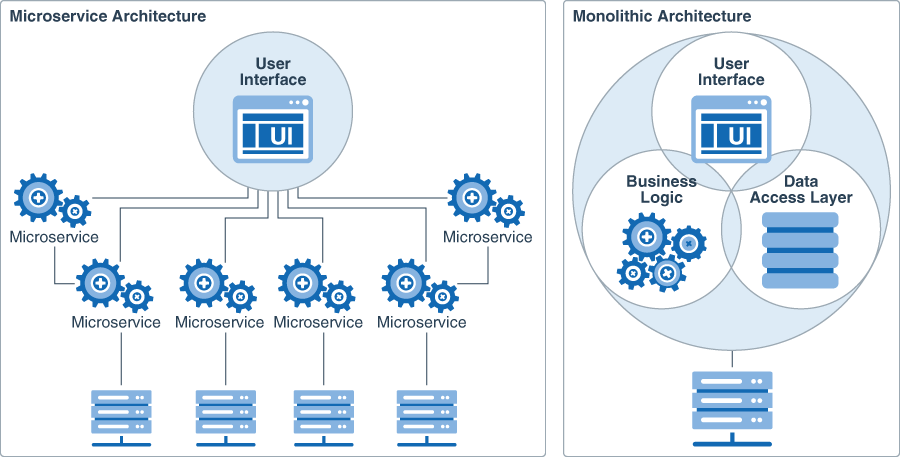
XMPP là mô hình phân quyền client-server phi tập trung, được sử dụng cho các ứng dụng nhắn tin văn bản. Có thể nói XMPP gần như là thời gian thực và có thể mở rộng đến hàng trăm hàng nghìn nút. Dữ liệu nhị phân phải được mã hóa base64 trước khi nó được truyền đi trong băng tần. XMPP tương tự như MQTT, có thể chạy trên nền tảng TCP.

# **CHƯƠNG 2: AMQP VỚI RabbitMQ**

**2.1 Microservice là gì**

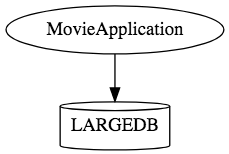
Thực tế có nhiều định nghĩa khác nhau đối với Microservices nhưng hiểu theo cách đơn giản thì, microservice là một kiếu kiến trúc phần mềm. Các module trong phần mềm này được chia thành các service rất nhỏ (microservice). Mỗi service sẽ được đặt trên một server riêng -> dễ dàng để nâng cấp và scale ứng dụng.

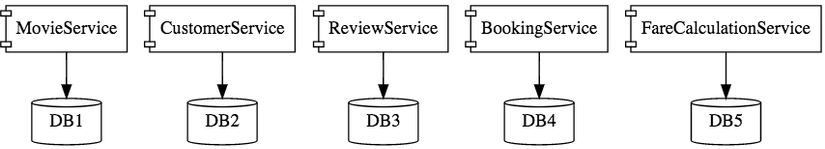
### *2.1.1. Kiến trúc*



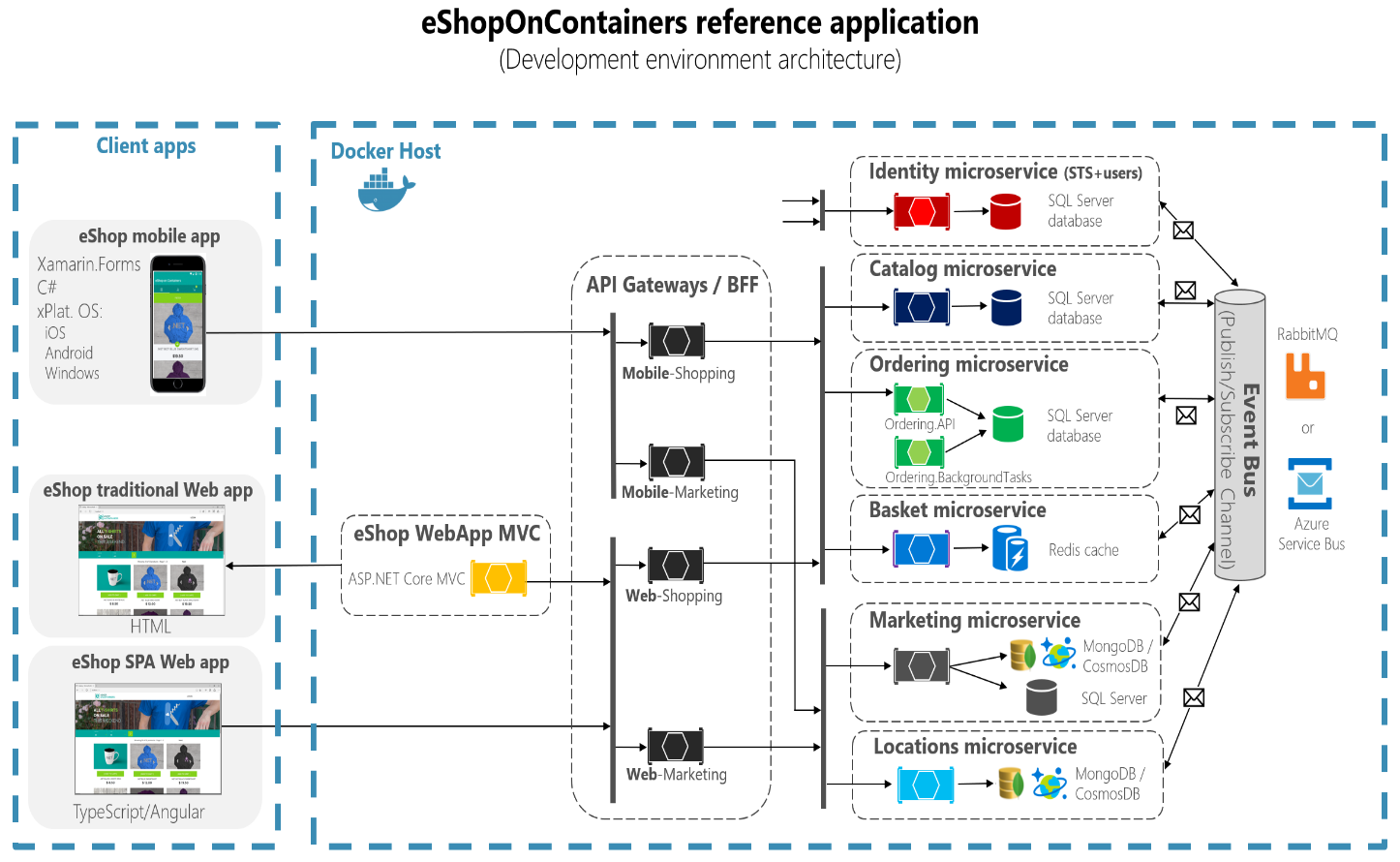
Khác biệt với kiến trúc Monolith (một khối), hay vì gom tất cả module thành một khối (monolith), ta tách các module thành những service siêu nhỏ. Mỗi service sẽ được đặt trên một server riêng (Có thể dùng server cloud như AWS hoặc Azure), giao tiếp với nhau thông qua mạng (Gửi nhận message qua giao thức HTTP hoặc sử dụng MessageQueue)...

Hình dưới đây sẽ minh họa cho việc phần mềm được xây dựng theo kiến trúc Monolith, một ứng dụng sẽ chứa tất cả các thành phần



Còn hình dưới sẽ minh họa việc ứng dụng ở trên khi được xây dựng theo kiến trúc Microservices

Kiến trúc Microservice bao gồm một số thành phần nhỏ, được thiết kế tốt và tương tác qua các message.



### *2.1.2. Ưu điểm của Microservices*

+ Giảm thiểu sự gia tăng phức tạp rối rắm hệ thống lớn.

+ Chia nhỏ ứng dụng một khối cồng kềnh 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.

+ Microservice thúc đẩy tách rạch ròi các khối chức năng (loose coupling - high cohesion), điều rất khó thực hiện với ứng dụng một khối. Nếu muốn loose coupling - high cohesion trong ứng dụng một khối, sẽ phải thiết kế theo Design Pattern (Gang Of Four) và liên tục tái cấu trúc (refactor)

+ Mỗi dịch vụ nhỏ sẽ phát triển dễ hơn, nhanh hơn, dễ viết mã kiểm thử tự động.

### *2.1.3. Nhược điểm của Microservices*

+ 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.

+ Phải xử lý sự cố khi kết nối chậm, lỗi khi thông điệp không gửi được hoặc thông điệp gửi đến nhiều đích đến vào các thời điểm khác nhau.

+ Đảm bảo giao dịch phân tán (distributed transaction) cập nhật dữ liệu đúng đắn (all or none) vào nhiều dịch vụ nhỏ khác nhau khó hơn rất nhiều, đôi khi là không thể so với đảm bảo giao dịch cập nhật vào nhiều bảng trong một cơ sở dữ liệu trung tâm.

+ Theo nguyên tắc CAP (CAP theorem) thì giao dịch phân tán sẽ không thể thỏa mãn cả 3 điều kiện: consistency (dữ liệu ở điểm khác nhau trong mạng phải giống nhau), availablity (yêu cầu gửi đi phải có phúc đáp), partition tolerance (hệ thống vẫn hoạt động được ngay cả khi mạng bị lỗi). Những công nghệ cơ sở dữ liệu phi quan hệ (NoSQL) hay môi giới thông điệp (message broker) tốt nhất hiện nay cũng chưa vượt qua nguyên tắc CAP

+ Kiểm thử tự động một dịch vụ trong kiến trúc microservices đôi khi yêu cầu phải chạy cả các dịch vụ nhỏ khác mà nó phụ thuộc. Do đó khi phân rã ứng dụng một khối thành microservices cần luôn kiểm tra mức độ ràng buộc giữa các dịch vụ mềm dẻo hơn hay cứng nhắc - lệ thuộc hơn. Nếu ràng buộc ít đi, lỏng leo hơn, bạn đi đúng hướng và ngược lại

+ Nếu các dịch vụ nhỏ thiết kế phục thuộc vào nhau theo chuỗi. A gọi B, B gọi C, C gọi D. Nếu một mắt xích có giao tiếp API thay đổi, liệu các mắt xích khác có phải thay đổi theo , việc bảo trì, kiểm thử sẽ phức tạp tương tự ứng dụng một khối. Chính vì thế việc thiết kế dịch vụ không tốt sẽ ảnh hưởng lan truyền đến các dịch vụ khác.

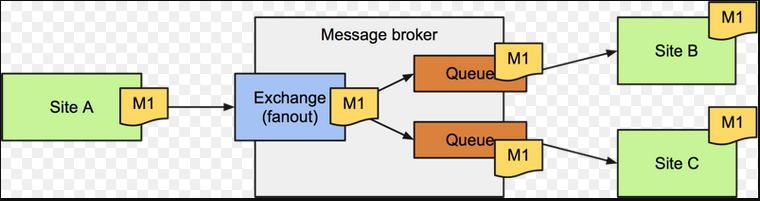
* 1. **RabbitMQ là gì ? Tại sao sử dụng RabbitMQ**

RabbitMQ là một message-queuing software có thể được biết đến như là một người vận chuyển message trung gian hoặc một người quản lí các queue. Nói một cách đơn giản, nó là một phần mềm nơi các queue được định nghĩa, phục vụ cho ứng dụng với mục đích vận chuyển một hoặc nhiều message

Vậy, tại sao chúng ta phải cần đến RabbitMQ? Hãy tưởng tượng, hiện bạn đang có một web service, phải nhận rất rất nhiều request mỗi giây, mà lại phải đảm bảo rằng không có bất cứ một request nào bị mất. Và web service của bạn luôn luôn sẵn sàng tiếp nhận request mới thay vì locked bởi đang xử lí request trước đó. Vậy ý tưởng ở đây là đặt chúng vào một queue giữa web service và processing service. Lúc này sẽ đảm bảo rằng 2 process sẽ hoàn toàn tách rời nhau. Ngoài ra, queue sẽ lưu trữ những request, không bị thiếu sót request nào khi số lượng của chúng trở nên vô cùng lớn

* 1. **Kiến trúc của RabbitMQ , các khái niệm cần biết trong RabbitMQ**

### *2.3.1. Kiến trúc của RabbitMQ*



Có thể hiểu message broker gần như bưu điện. Site A theo cách gọi của rabbitmq là producer (người gửi thông điệp). Site B và Site C theo cách gọi của rabbitmq là consumer (người nhận thông điệp). Producer connect đến message broker để đẩy message. Message sẽ đi qua message broker để đến được consumer. Cấu trúc của message broker chỉ gồm hai phần exchange và queue.

Exchange có nhiều loại. Trong hình vẽ trên exchange type là fanout. Lựa chọn các exchange type khác nhau sẽ dẫn đến khác đối xử khác nhau của message broker với thông điệp nhận được từ producer. Exchange được bind (liên kêt) đến một số queue nhất định. Với exchange type là fanout, message sẽ được broadcast đến các queue được bind với exchange. Consumer sẽ connect đến message broker để lấy message từ các queue.

# **CHƯƠNG 3: Xây dựng demo sử dụng phương thức AMQP trong RabbitMQ với AspCore 3.1**

**3.1 ASP.NET Core là gì**

ASP.NET Core là một open-source mới và framework đa nền tảng (cross-platform) cho việc xây dựng những ứng dụng hiện tại dựa trên kết nối đám mây, giống như web apps, IoT và backend cho mobile.

Ứng dụng ASP.NET Core có thể chạy trên .NET Core hoặc trên phiên bản đầy đủ của .NET Framework. Nó được thiết kế để cung cấp và tối ưu development framework cho những dụng cái mà được triển khai trên đám mây (clound) hoặc chạy on-promise.

Nó bao gồm các thành phần theo hướng module nhằm tối thiểu tài nguyên và chi phí phát triển, như vậy bạn giữ lại được sự mềm giẻo trong việc xây dựng giải pháp của bạn. Bạn có thể phát triển và chạy những ứng dụng ASP.NET Core đa nền tảng trên Windows, Mac và Linux.

**Khác biệt quan trọng của ASP.NET và ASP.NET Core**

|  |  |
| --- | --- |
| **ASP.NET** | **ASP.NET CORE** |
| Phiên bản hiện tại 4.8 | Phiên bản hiện tại 3.1 |
| Nền tảng đã có từ lâu | Hoàn toàn được thiết kế mới |
| Chạy trên .NET Framwork | Chạy trên cả .NET Core và .NET Framework |
| Chỉ trên Windows | Chạy trên tất cả các OS sử dụng .NET Core |
| Nền tảng ổn định với tính năng phong phú | Chưa hoàn chỉnh nhưng mong đợi sẽ hoàn chỉnh trong tương lai |
| WebForms được hỗ trợ | Không hỗ trợ WebForms |
| System.web.dll cồng kềnh | Nhỏ, nhẹ và module hóa |
| Bản quyền của Microsoft | ASP.NET Core là mã nguồn mở |

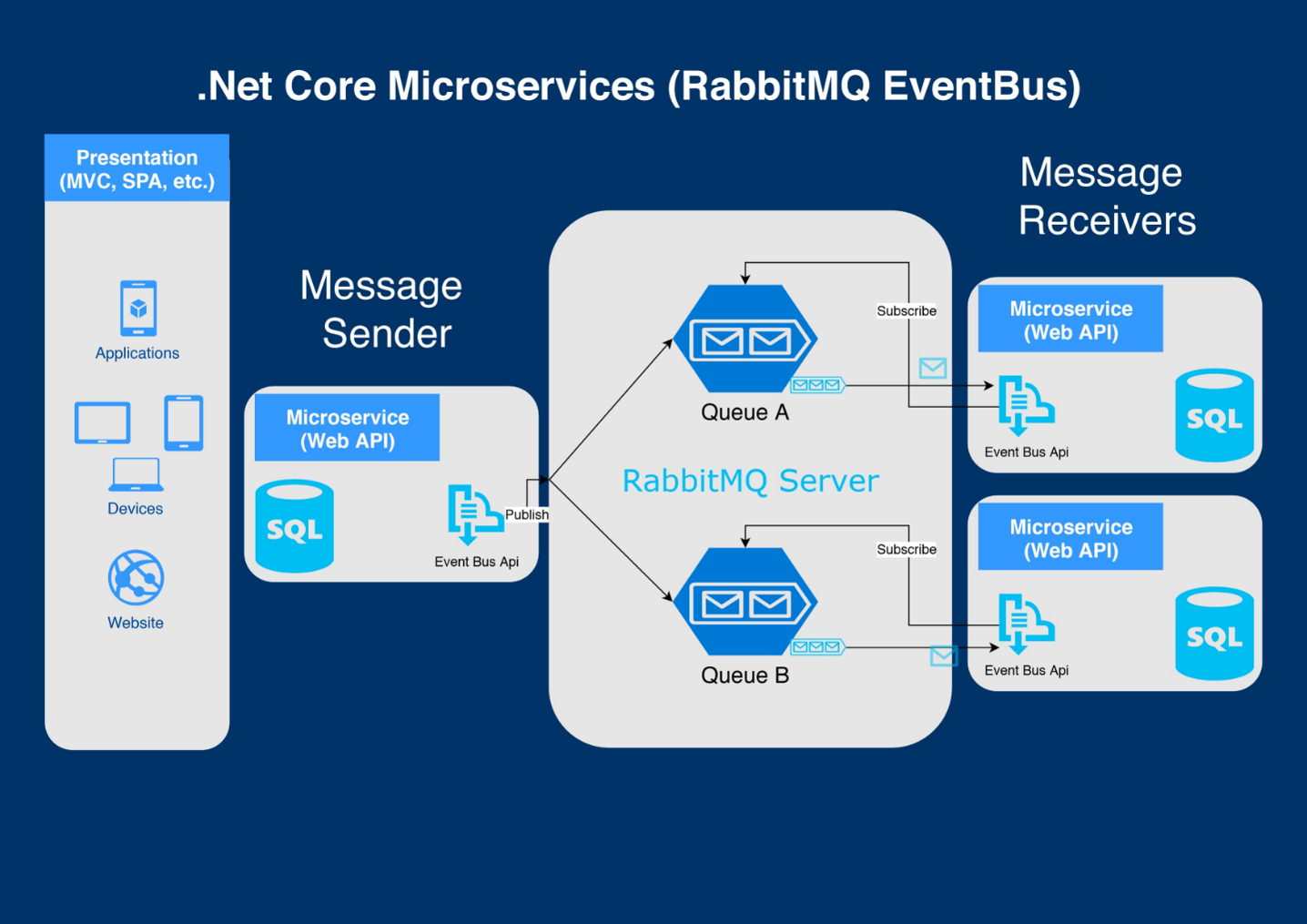
**3.2 Xây dựng demo**

### **3.2.1 Đặt Vấn Đề**

Trong một giao dịch chuyển tiền trong một hệ thống microservice ta cần lưu lại lịch sử giao dịch đó .Giả sử ta có 2 service : service 1 sẽ tiếp nhận giao dịch và service 2 sẽ lưu lại lịch sử của phiên giao dịch này vào database , service 1 và service 2 trên giao thức AMQP sử dụng RabbitMQ

### **3.2.1 Thiết kế**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Allow Nulls** |
| ID | INT | NOT NULL |
| FromAccount | nvarchar(MAX) | NOT NULL |
| ToAccount | nvarchar(MAX) | NOT NULL |
| TransferAmount | decimal(18, 2) | NOT NULL |



# **TÀI LIỆU THAM KHẢO**

[1]. Effective storage managementand data protection for cloudcomputing. IBM Software Thought Leadership White Paper. December 2011;

[2]. IBM presents staged approach to virtualization management. WHITE PAPER | © 2010 IDEAS INTERNATIONAL, INC. June 2010

[3]. IBM WebSphere Application Server Hypervisor Edition. Copyright IBM Corporation 2009. IBM Corporationc Software Group Route 100 Somers, NY 10589 U.S.A. Produced in the United States of America June 2009

[4]. Axel Buecker, Koos Lodewijkx, Harold Moss, Kevin Skapinetz, Michael Waidner Cloud Security Guidance: IBM Recommendations for the Implementation of Cloud Security. © Copyright International Business Machines Corporation 2009.

[5]. IBM WebSphere VirtualEnterprise: Maximize server use while monitoring application health. Copyright IBM Corporation 2011. IBM Corporationc Software Group Route 100 Somers, NY 10589 U.S.A. Produced in the United States of America April 2011

[6]. http:// [www.vmware.com](http://www.vmware.com)

<https://tapit.vn/5-giao-thuc-truyen-tai-du-lieu-trong-internet-things/>

<https://medium.com/@buihuycuong/nh%E1%BB%AFng-%C4%91i%E1%BB%81u-c%E1%BA%A7n-bi%E1%BA%BFt-v%E1%BB%81-rabbitmq-354a37ecf218>

<https://kipalog.com/posts/Tim-hieu-RabbitMQ---Phan-1>  
<https://seal.deha.vn/cac-giao-thuc-tich-hop-he-thong-pho-bien/>

<https://viblo.asia/p/tim-hieu-ve-rabbitmq-OeVKB8bMlkW>

<http://alexvolov.com/2016/06/amqp/>

<https://smartfactoryvn.com/resources/technical-writer/rabbitmq-la-gi/>

<https://www.dds-foundation.org/>

<https://www.dds-foundation.org/why-choose-dds/>