

Xây dựng Microservices bằng ASP.NET Core - Lập kế hoạch

Trung Nguyen 03/05/2021 10 min read

Nội dung chính của bài viết

Share



Danh sách các bài viết:

- **Phần 1:** Lập kế hoạch (bài viết này).
- **Phần 2:** Định hình kiến trúc microservices với CQRS và MediatR.
- **Phần 3:** Khám phá dịch vụ với Eureka.
- **Phần 4:** Xây dựng API Gateway với Ocelot.
- **Phần 5:** Thao tác với database PostgreSQL bằng Marten.
- **Phần 6:** Giao tiếp máy chủ thời gian thực với SignalR và RabbitMQ.
- **Phần 7:** Transaction outbox với RabbitMQ.

Trong khi Java bị đình trệ, nền tảng .NET đã phát triển rất nhanh chóng. Một số thư viện mã nguồn mở đã được tạo và áp dụng rộng rãi như [NHibernate](#), [Castle Project](#) hoặc [log4net](#). Mọi thứ thậm chí còn tốt hơn với ASP.NET MVC.

Nhưng đầu đó vào khoảng năm 2014, mọi thứ đã thay đổi khi Microsoft bắt đầu đầu tư mạnh mẽ vào .NET Core. Ý tưởng này có vẻ tuyệt vời - ứng dụng .NET chạy trên tất cả các nền tảng chính.

Nhưng phải mất nhiều năm để đạt được và những bản phát hành đầu tiên hơi đáng thất vọng. Vì vậy, nhiều API bị thiếu khiến nhiều thư viện yêu thích của chúng tôi không được chuyển đổi.

Các phiên bản đầu tiên của Entity Framework Core thiếu các tính năng quan trọng. Tương lai của .NET có vẻ đáng ngờ. Cùng lúc đó, Java 8 ra đời và mang lại sự mới mẻ cho ngôn ngữ này, cũng là lúc cách tiếp cận kiến trúc dựa trên microservices ra đời. Điều này cùng với [Spring Boot](#) đã thuyết phục tôi quay trở lại vùng đất Java.

Nhưng các bản phát hành .NET gần đây - đặc biệt là .NET Core 2.x và sáng kiến .NET Standard đã thuyết phục tôi rằng .NET Core đã trở lại. Vì vậy, tôi cùng với các đồng đội của mình quyết định khám phá các khả năng và thách thức của việc xây dựng các giải pháp microservices dựa trên .NET Core.

Chúng tôi có nhiều kinh nghiệm trong lĩnh vực microservices khi chúng tôi phát triển và triển khai các hệ thống theo cách tiếp cận kiến trúc này từ năm 2015. Chúng tôi muốn chia sẻ với bạn tùy chọn khả thi để xử lý các tác vụ liên quan đến microservices điển hình như khám phá dịch

Subscribe

Kế hoạch

Trong loạt bài viết này, chúng tôi sẽ giới thiệu cho các bạn các tác vụ điển hình cần thiết để xây dựng giải pháp dựa trên microservices.

Các tác vụ này bao gồm:

Thiết kế kiến trúc bên trong của một microservices với:

- [CQRS](#).
- Event Sourcing.

Truy cập các nguồn dữ liệu với:

- Entity Framework Core với PostgreSQL.
- NHibernate.
- Marten với PostgreSQL.
- NEST với Elasticsearch.

Giao tiếp giữa các microservices:

- Đồng bộ với các cuộc gọi HTTP REST trực tiếp.
- Không đồng bộ với RabbitMQ.

Và các tác vụ quan trọng khác như:

- API Gateway với Ocelot.
- Khám phá dịch vụ với Eureka.
- Khả năng phục hồi với Polly.
- Ghi log bằng Serilog.
- Bảo mật các dịch vụ với JWT.
- Chạy tác vụ nền với Hangfire.
- Tổng hợp log với ELK.
- Số liệu và giám sát.
- CI/CD với Azure DevOps.

Triển khai các dịch vụ tới:

- Azure.
- Kubernetes.

Tình huống kinh doanh

Chúng ta sẽ xây dựng một hệ thống rất đơn giản cho các đại lý bảo hiểm để bán các loại sản phẩm bảo hiểm khác nhau.

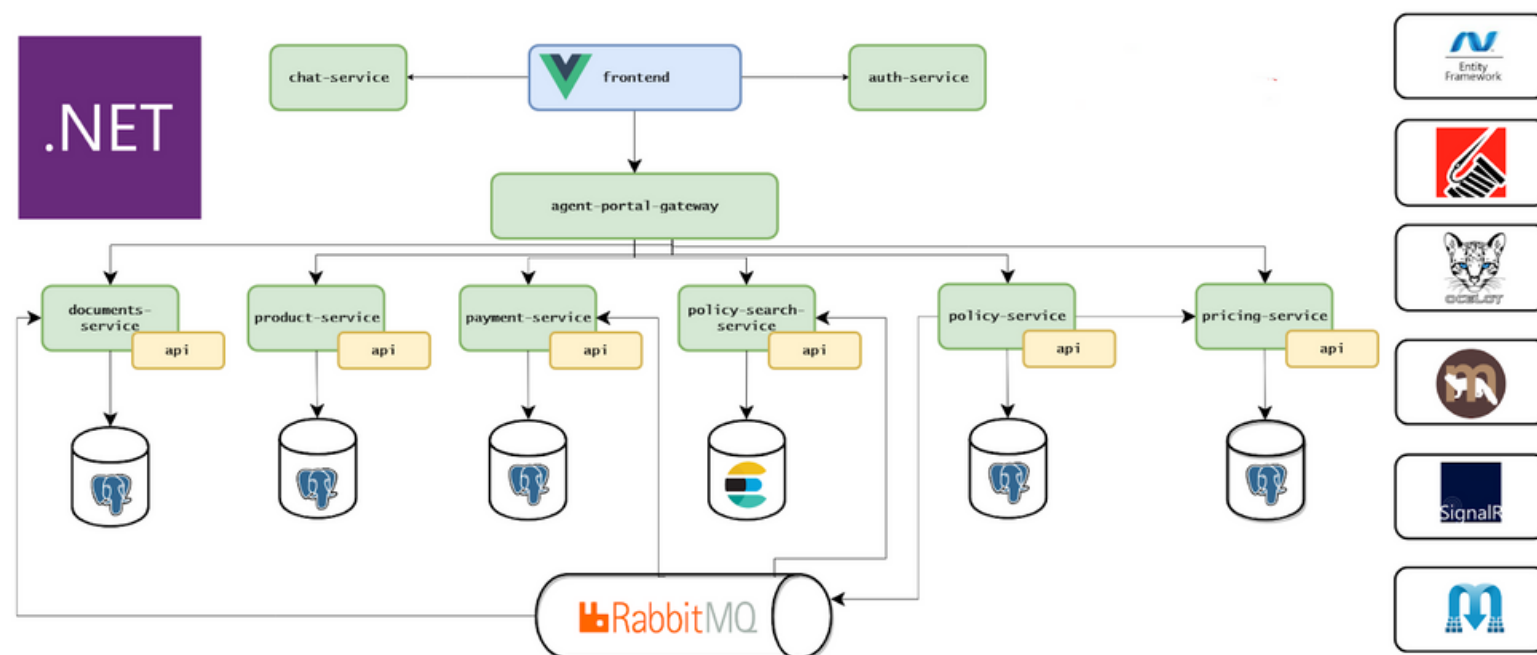
Đại lý bảo hiểm sẽ phải đăng nhập và hệ thống sẽ hiển thị cho họ danh sách các sản phẩm họ có thể bán. Đại lý sẽ được xem sản phẩm và tìm được sản phẩm phù hợp với khách hàng của mình. Sau đó, họ có thể tạo offer và hệ thống sẽ tính giá dựa trên các thông số được cung cấp.

Cổng thông tin cũng sẽ cung cấp cho họ khả năng tìm kiếm và xem offer và chính sách.

Cổng thông tin cũng sẽ có một số tính năng mạng xã hội cơ bản như chat cho các đại lý.

Kiến trúc hệ thống

Bạn có thể tìm thấy sơ đồ kiến trúc của hệ thống của chúng tôi bên dưới.



Hệ thống bao gồm:

- **Frontend** - một ứng dụng SPA viết bằng VueJS cung cấp cho các đại lý bảo hiểm khả năng lựa chọn sản phẩm thích hợp cho khách hàng của họ, tính toán giá cả, tạo offer và kết thúc quy trình bán hàng bằng cách chuyển đổi offer thành chính sách. Ứng dụng này cũng cung cấp các chức năng tìm kiếm và xem các chính sách và offer. Giao diện người dùng làm việc với các dịch vụ backend thông qua API Gateway.
- **Agent Portal API Gateway** - là một microservices đặc biệt có mục đích chính là che giấu sự phức tạp của cấu trúc dịch vụ back office cơ bản khỏi ứng dụng client. Thông thường, chúng tôi tạo một API Gateway dành riêng cho từng ứng dụng client. Trong trường hợp trong tương lai, chúng tôi thêm một ứng dụng di động Xamarin vào hệ thống của mình, chúng tôi sẽ cần phải xây dựng một API Gateway dành riêng cho nó. API Gateway cũng cung cấp hàng rào bảo mật và không cho phép chuyển yêu cầu chưa được xác thực đến các dịch vụ backend. Một cách sử dụng phổ biến khác của API Gateway là tổng hợp nội dung từ nhiều dịch vụ.
- **Auth Service** - một dịch vụ chịu trách nhiệm xác thực người dùng. Hệ thống bảo mật của chúng tôi sẽ dựa trên mã thông báo [JWT](#). Sau khi xác thực người dùng thành công, dịch vụ xác thực sẽ phát hành mã token được sử dụng để kiểm tra quyền của người dùng và các sản phẩm có sẵn.
- **Chat Service** - một dịch vụ sử dụng SignalR để cung cấp cho các đại lý khả năng chat với nhau.
- **Pricing Service** - một dịch vụ chịu trách nhiệm tính giá cho sản phẩm bảo hiểm nhất định dựa trên tham số của nó.
- **Policy Service** - dịch vụ chính trong hệ thống của chúng tôi. Nó chịu trách nhiệm tạo ra các offer và chính sách. Nó sử dụng Pricing Service thông qua các cuộc gọi HTTP REST để tính giá. Khi một chính sách được tạo, nó sẽ gửi sự kiện không đồng bộ đến event bus (RabbitMQ) để các dịch vụ khác có thể phản ứng.
- **Policy Search Service** - mục đích duy nhất của dịch vụ này là để hiển thị các chính sách tìm kiếm. Dịch vụ này đăng ký các sự kiện liên quan đến vòng đời chính sách và các sản phẩm.

quan đến chính sách. Nó dùng kỹ các sự kiện trên quan đến chính sách, tạo tài khoản chính sách khi chính sách được tạo, đăng ký các khoản thanh toán dự kiến. Nó cũng có một công việc nền rất đơn giản là phân tích cú pháp tệp với các khoản thanh toán đến và chỉ định nó cho tài khoản chính sách thích hợp.





- **Product Service** - đây là một danh mục sản phẩm. Nó cung cấp thông tin cơ bản về từng sản phẩm bảo hiểm và các thông số của nó có thể được tùy chỉnh trong khi tạo offer cho khách hàng.
- **Document Service** - dịch vụ này sử dụng JS Report để tạo chứng chỉ pdf.





























Như bạn có thể thấy, các dịch vụ được chia tách theo nghiệp vụ kinh doanh. Chúng tôi cũng có một dịch vụ kỹ thuật - dịch vụ tài liệu. Việc giới thiệu các dịch vụ kỹ thuật có ý nghĩa khi chúng yêu cầu khả năng mở rộng / khả năng phục hồi và chúng tôi muốn giảm thời gian cập nhật / sửa chữa hoặc chi phí cấp phép.

Mỗi microservice được xây dựng xung quanh [ngữ cảnh bị ràng buộc](#) trong các điều khoảnDDD. Ngoài ra, bạn có thể quan sát thấy chúng hợp tác để đạt được mục tiêu kinh doanh chính - bán sản phẩm bảo hiểm cho khách hàng cuối cùng.

Cấu trúc giải pháp

Dưới đây là một phần của solution của chúng tôi mở trong [Rider IDE](#). Như bạn có thể thấy đối với mỗi microservice, chúng tôi tạo ra ba dự án: một cho định nghĩa API, một cho việc triển khai và một cho các thử nghiệm. Bạn có thể tìm thấy mã nguồn cho dự án của chúng tôi tại đây trên [GitHub của chúng tôi](#). Lưu ý rằng nó đang được phát triển và mã sẽ thay đổi khi chúng ta tiến hành loạt bài của mình.



- ▼  DotNetMicroservicesPoc · 14 projects
 - ▶  PaymentService
 - ▶  PaymentService.Api
 - ▶  PaymentService.Test
 - ▶  PolicySearchService
 - ▶  PolicySearchService.Api
 - ▼  PolicyService
 - ▶  Dependencies
 - ▶  Properties
 - ▶  Commands
 - ▶  Controllers
 - ▶  DataAccess
 - ▶  Domain
 - ▶  Messaging
 - ▶  Queries
 - ▶  RestClients
 - ▶  wwwroot
 - ▶  appsettings.Development.json
 - ▶  appsettings.json
 - ▶  Program.cs
 - ▶  Startup.cs
 - ▶  PolicyService.Api
 - ▶  PolicyService.Tests
 - ▶  PricingService
 - ▶  PricingService.Api
 - ▶  PricingService.Test
 - ▶  ProductService
 - ▶  ProductService Api

Dự án định nghĩa API chứa các class và interface mô tả các khả năng của dịch vụ được tiếp xúc với thế giới bên ngoài dưới dạng các lệnh mà nó có thể xử lý, các truy vấn mà nó có thể trả lời, các sự kiện mà nó phát ra và các lớp đối tượng truyền dữ liệu mà nó thể hiện. Chúng ta có thể coi đây là các định nghĩa cổng trong [kiến trúc cổng và bộ điều hợp](#).

Dự án triển khai chứa trình xử lý lệnh, trình xử lý truy vấn và trình xử lý thông báo cùng cung cấp chức năng dịch vụ. Hầu hết logic nghiệp vụ đi vào phần mô hình miền. Bộ điều hợp để nói chuyện với thế giới bên ngoài được triển khai dưới dạng controller (để xử lý các yêu cầu HTTP đến), listener (cho các sự kiện được phân phối qua hàng đợi) và REST client (để xử lý các yêu cầu HTTP đi).

Dự án thử nghiệm chứa cả unit test và integration test.

Tóm lược

Theo chúng tôi .NET Core là một nền tảng tuyệt vời để xây dựng microservices và trong loạt bài viết này, chúng tôi muốn chứng minh điều đó. Có một hệ sinh thái phong phú của các công cụ và thư viện mã nguồn mở xung quanh nó. Bản thân ngôn ngữ C# là một công cụ tuyệt vời.

Nền tảng và các thư viện đều là mã nguồn mở và các tính năng mới được cung cấp với tốc độ rất cao.

Ngoài ra, những cải tiến của .NET Core 2.x xung quanh việc sử dụng bộ nhớ và hiệu suất làm cho [nó thậm chí còn hấp dẫn hơn](#).

Trong các bài viết tiếp theo, chúng tôi sẽ trình bày các phát hiện và giải pháp của chúng tôi cho các tác vụ phổ biến liên quan đến việc phát triển và triển khai microservices.

Xây dựng Microservices bằng ASP.NET Core - Định hình kiến trúc Microservices với CQRS và MediatR

Trong bài viết này, chúng ta sẽ thiết kế kiến trúc bên trong của Microservices với mẫu CQRS và MediatR trong ASP.NET Core.

 Trung Nguyen • Comdy

Microservices

Bài viết được dịch từ bài gốc ở [đây](#).

Nếu **Comdy** hữu ích và giúp bạn tiết kiệm thời gian

Bạn có thể vui lòng **tắt trình chặn quảng cáo** ❤️ để hỗ trợ chúng tôi duy trì hoạt động của trang web.

Bài Viết Liên Quan:

Xây dựng Microservices bằng ASP.NET Core - Transaction Outbox với RabbitMQ

Trong bài viết này, chúng ta sẽ tìm hiểu cách sử dụng outbox pattern để triển khai distributed transaction trong kiến trúc microservices.

👁 12 min read

[Xem Thêm >](#)

Xây dựng Microservices bằng ASP.NET Core - Giao tiếp máy chủ thời gian thực với SignalR và RabbitMQ

Trong bài viết này, chúng tôi sẽ chỉ cho bạn cách bạn có thể kết hợp SignalR và RabbitMQ để xây dựng giao tiếp máy chủ-máy khách thời gian thực.

👁 14 min read

[Xem Thêm >](#)

Xây dựng Microservices bằng ASP.NET Core - Thao tác với database bằng Marten

Trong bài viết này, chúng ta sẽ tìm hiểu về truy cập dữ liệu và cách lưu trữ dữ liệu một cách hiệu quả trong microservices sử dụng Marten và PostgreSQL.

👁 26 min read

[Xem Thêm >](#)

Xây dựng Microservices bằng ASP.NET Core - Xây dựng API Gateway với Ocelot

Trong bài viết này, chúng ta sẽ tìm hiểu về API Gateway trong kiến trúc microservices và cách xây dựng API Gateway bằng Ocelot.

👁 13 min read

[Xem Thêm >](#)

[Xem tất cả bài viết về Microservices](#)

[Subscribe](#)

