Building Microservices On .NET Core – Part 1 The Plan

14/01/2019

I liked .NET technology from its inception. In fact I left the dark star of overxmlized J2EE development to join forces of rebellion around 2004. Over the years my team here at Altkom Software & Consulting built and maintained more and more complex business solutions for insurance and banking. While Java was in stagnation .NET platform developed very quickly. Several open source libraries were created and widely adopted like for example NHibernate, Castle Project or log4net. Things get even better with ASP.NET MVC. But somewhere around 2014 things changed as Microsoft started to heavily invest in .NET Core. The idea seemed great – have .NET running on all major platforms. But it took many years to achieve and first releases were a bit disappointing. So many missing APIs caused that many of our favourites libraries were not ported. First versions of Entity Framework lacked critical features. The future of .NET seemed doubtful. At the same time Java 8 came out and brought coolness back to the language, also microservice based approach to architecture emerged. This together with Spring Boot convinced me to move back to Java land.

But recent .NET releases – especially .NET Core 2.x and .NET standard initiative convinced me that .NET core is back in the business. So I with my teammates decided to explore possibilities and challenges of building microservice based solutions on .NET Core.

We have a lot of experience in the microservice area as we develop and deploy systems in this architectural approach since 2015. We wanted to find out possible options for dealing with typical microsevice related tasks like service discovery, service communication synchronous and asynchronous, data access to various data sources (relational and noSQL), logging, resilience, error handling, security with JWT and api gateway building.

The Plan

In this series of articles we are going to walk through typical tasks required to build microservices based solution.

These tasks include:

- Designing internal architecture of a microservice with
  - CQRS
  - Event Sourcing
- Accessing data sources with
  - Entity Framework Core with PostgreSQL
  - NHibernate
  - Marten with PostgreSQL
  - NEST with ElasticSearch
- Communication between microservices
  - Synchronous with direct HTTP REST calls
  - Asynchronous with RabbitMQ
- API Gateways with Ocelot
- Service discovery with Eureka
- Resilience with Polly
- Logging with Serilog
- Securing services with JWT
- Running background jobs with Hangfire
- Log aggregation with ELK
- Metrics and monitoring
- CI/CD with Azure DevOps
- Deploying services to
  - Azure
  - Kubernetes

List of published articles

Part 2: Shaping microservice internal architecture with CQRS and MediatR
Part 3: Service Discovery with Eureka
Part 4: Building API Gateways With Ocelot

Business Case

We are going to build very simplified system for insurance agents to sell various kind of insurance products.

Insurance agents will have to log in and system will present them list of products they can sell. Agents will be able to view products and find a product appropriate for their customers. Then they can create an offer and system will calculate a price based on provided parameters.

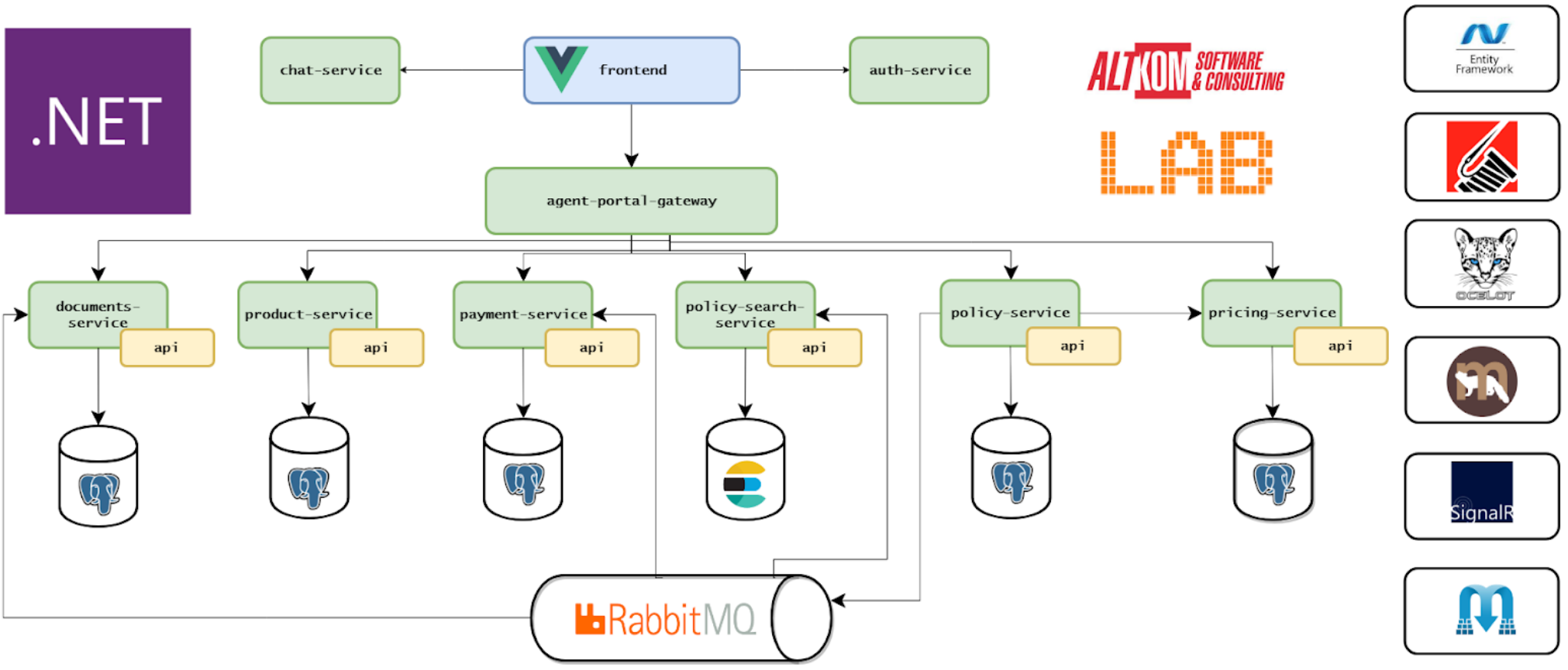
Finally agent will be able to confirm the sale by converting offer to policy and printing pdf certificate.

Portal will also give them ability to search and view offer and policies.

Portal will also have some basic social network features like chat for agents.

Solution Architecture

You can find architectural diagram of our system below.



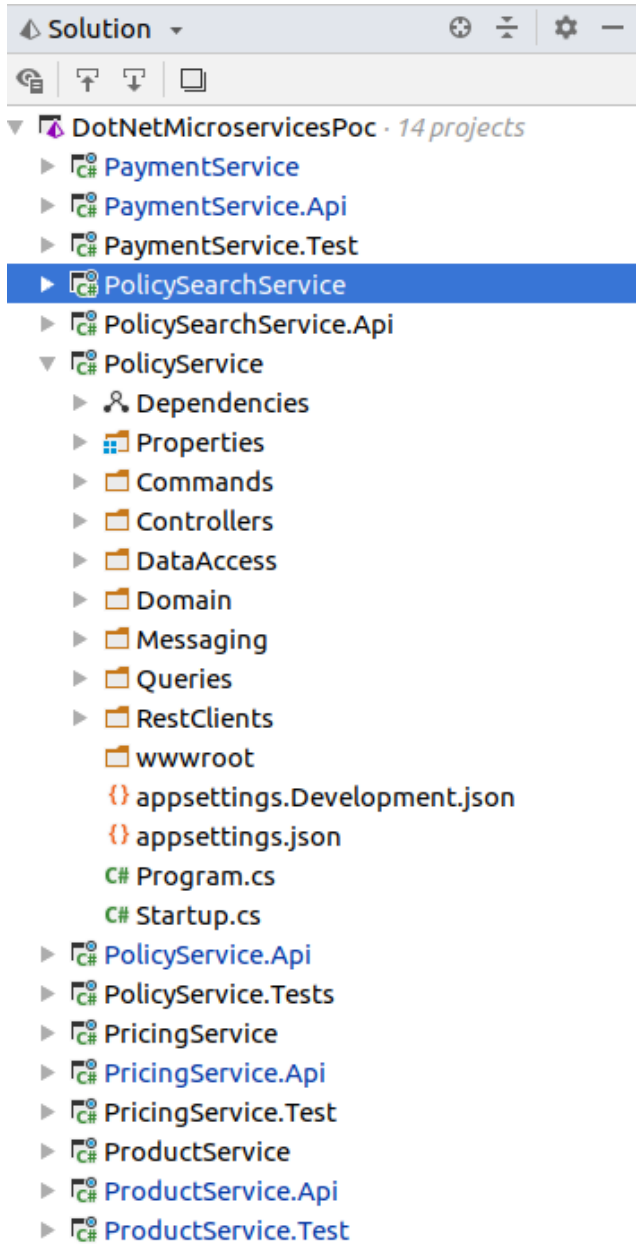
The system consists of:

- **Frontend** – a VueJS Single Page Application that provides insurance agents ability to select appropriate product for their customers, calculate price, create an offer and conclude the sales process by converting offer to policy. This application also provides search and view functions for policies and offers. Frontend talks to backend services via api-gateway.
- **Agent Portal API Gateway** – is a special microservice whose main purpose it to hide complexity of the underlying back office services structure from client application. Usually we create a dedicated API gateway for each client app. In case in the future we add a Xamarin mobile app to our system, we will need to build a dedicated API gateway for it. API gateway provides also security barrier and does not allow unauthenticated request to be passed to backend services. Another popular usage of API gateways is content aggregation from multiple services.
- **Auth Service** – a service responsible for users authentication. Our security system will be based on JWT tokens. Once user identifies himself correctly, auth service issues a token that is further use to check user permission and available products.
- **Chat Service** – a service that uses SignalR to give agents ability to chat with each other.
- **Pricing Service** – a service responsible for calculation of price for given insurance product based on its parametrization.
- **Policy Service** – main service in our system. It is responsible for creation of offers and policies. It uses Pricing Service via REST http calls to calculate the price. Once a policy is created it sends asynchronous event to the event bus (RabbitMQ) so other services can react.
- **Policy Search Service** – a sole purpose of this service is to expose search functionality. This service subscribes to events related to policy lifecycle and indexes given policy in ElasticSearch to provide advanced search capabilities.
- **Payment Service** – this service is responsible for managing financial operations related to policy. It subscribes to policy related events, creates policy account when policy is created, registers expected payments. It also has a very simplified background job that parses file with incoming payments and assigns it to proper policy account.
- **Product Service** – this is a product catalog. It provides basic information about each insurance product and its parameters that can be customized while creating an offer for a customer.
- **Document Service** – this service uses JS Report to generate pdf certificates.

As you can see services are decomposed by business capability. We also have one technical service – document service. Introducing technical services makes sense when they require scalability/resilience and we want reduce time to update/fix or licence costs.

Each microservice is built around bounded context in DDD terms. Also you can observe that they cooperate to achieve the main business goal – to sell an insurance product to the end customer.

tests. You can find source code for our project here on our GitHub. Note that it is work in progress and code will change as we progress with our series.



API definition project contains classes and interfaces that describe service capabilities exposed to the external world in form of commands it can handle, queries it can answer, events it emits and data transfer object classes it exposes. We can treat these as port definitions in a ports and adapters architecture.

Implementation project contains command handlers, query handlers and notification handlers which together provide service functionality. Most of the business logic goes into domain model part. Adapters for talking to external world are implemented as controllers (for handling incoming HTTP requests), listeners (for events delivered via queue) and REST Clients (for handling outgoing http requests).

Test project contains both unit tests and integration test.

Summary

In our opinion .NET Core is a great platform for building microservices and in this series of articles we would like to prove it. There is a rich ecosystem of open source tools and libraries around it. The C# language itself is a great tool.

The platform and libraries are all open source and new features are delivered at very high velocity.

Also .NET Core 2.x enhancements around performance and memory usage make it even more attractive.

In the coming articles we are going to present our findings and solutions for common tasks related to microservices implementation and deployment.

Author: Wojciech Suwała, Head Architect, ASC LAB

Tags:

.NET • ASC LAB • Banking software • Banking software providers • Digital banking platform • Digital banking providers • Electronic banking software • Insurance business solutions • Insurance software • Internet banking software • Internet banking system • IT consulting for banks • Mobile banking software • Online banking software • Online banking system • Software development • Software engineering • Software House

6 Add comment

Join the discussion...

4 2 8 5

Subscribe newest oldest most voted

animesh

Guest

As someone who is dipping his toes in microservices (ServiceFabric), I love the premise for this article series and I will be eagerly waiting for rest of the posts. Thank you.

2 Reply 02/07/2018 12:40

Robert Witkowski

Guest

Next parts:
Part 2 Shaping microservice internal architecture with CQRS and MediatR https://altkomsoftware.pl/en/blog/microservices-net-core-cqrs-mediatr/
Part 3 Service Discovery with Eureka https://altkomsoftware.pl/en/blog/service-discovery-eureka/
Part 4 Building API Gateways With Ocelot https://altkomsoftware.pl/en/blog/building-api-gateways-with-ocelot/
Part 5 Marten An Ideal Repository For Your Domain Aggregates https://altkomsoftware.pl/en/blog/building-microservices-domain-aggregates/

1 Reply 11/08/2018 12:41 View Replies (3)

Kieren Johnstone

Guest

I'm wondering if it's worth outlining the pros and cons of microservices, and helping people understand when to use them (i.e. only when there are specific pressures on the development team, each service has a "two-pizza team" size, etc)

1 Reply 02/01/2019 12:45

Art Gorr

Guest

I like the style of the article. Thank you for the contribution.

1 Reply 02/07/2019 12:40

Thao Pham Van

Guest

Awesome, this is a great post that i have ever read. Can you please add some steps into readme.txt file in order to guide how to run docker for each services

0 Reply 12/07/2019 09:26 View Replies (1)

BillyBiro

Guest

Hi. Your plan mentions Event Sourcing, but I'm not seeing any mention or usage of it throughout the blog series. Did you abandon the idea of using Event Sourcing?

0 Reply 21/08/2019 08:01 View Replies (1)

Ha Doan

Guest

Hi, thank you for the tutorial, I'm really love it.
Can I translate your tutorial to Other language, so non-speaking English also can understand?
Thanks

Guest

This series is great but why VueJS. Even though it is not show stopper but it was great if it was ReactJS or Angular. But overall series is superb and hope to learn lot from it.

0 Reply

08/10/2019 21:08

lilianapletwa

Guest

Ulubiona wyszukiwarka kompenzatensat Google

0 Reply

14/10/2019 20:42

Altkom Software & Consulting
Chlodna Street No 51, 00-867 Warsaw
Building: Warsaw Trade Tower

Send us
asc@altkomsoftware.pl

Call us
+48 224 609 931

Patron: