Relatório do Trabalho 1 de Engenharia de Software 2

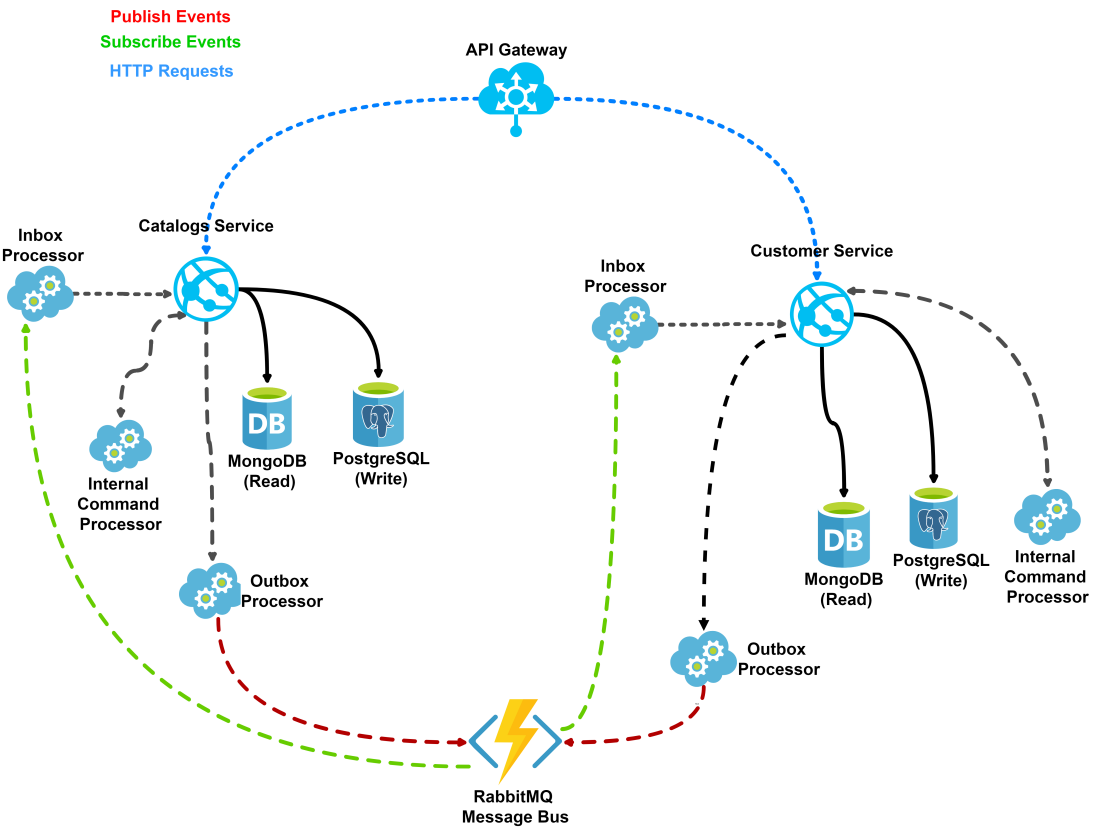
Dhruv Babani
Felipe Gauer
Rafael Toneto
Pontifícia Universidade Católica do Rio Grande do Sul — PUCRS

10 de abril de 2024

1 Uma breve ánalise Critica às fronteiras e Interfaces dos Microsserviços

Neste trabalho, vamos abordar sobre um sistema de microsserviços chamado Food Delivery Microservices, que na qual é um microsserviço fictício de entrega de alimentos, construído com .Net Core e diferentes arquiteturas e tecnologias de software, como arquitetura de microsserviços, arquitetura de fatia vertical, padrão CQRS, design orientado a domínio (DDD), arquitetura orientada a eventos. Para comunicação entre serviços independentes, usamos mensagens assíncronas usando RabbitMQ no topo da biblioteca MassTransit e, às vezes, usamos comunicação síncrona para comunicações em tempo real usando chamadas REST e gRPC.

2 Diagrama de UML



3 Resumo das Escolhas Arquiteturais

Neste projeto, foram selecionais algumas decisões arquiteturais para o desenvolvimento do projeto. Aqui embaixo, estarão listadas todas elas:

- A Arquitetura mostrada acima possui uma API GatewWay pública, que na qual é acessível para os clientes e isso é feito atraves de requisições e respostas HTTP.O gateway de API então roteia a solicitação HTTP para o microsserviço correspondente. A solicitação HTTP é recebida pelo microsserviço que hospeda sua própria API REST.

- Cada microsserviço está rodando no seu proprio AppDomain, e tem direto acesso às suas dependências, tais como, banco de dados, local files e etc. Na verdade, os microsserviços são dissociados uns dos outros e são autônomos. Isso também significa que o microsserviço não depende de outras partes do sistema e pode ser executado independentemente de outros serviços.

- Nesta arquitetura usamos o padrão CQRS para separar o modelo de leitura e gravação, além de outras vantagens do CQRS. Por enquanto, não uso o Event Sourcing para simplificar, mas irei usá-lo no futuro para sincronizar o lado de leitura e gravação com o envio de fluxos e usar o recurso de projeção para alguns assinantes sincronizarem seus dados por meio de fluxos enviados e criarem nossos modelos de leitura personalizados nos assinantes lado.

- Aqui eu tenho um modelo de gravação que usa um banco de dados postgres para lidar com melhor consistência e garantia de transação ACID. além deste lado de gravação, eu uso um modelo de leitura que usa MongoDB para melhor desempenho de nosso lado de leitura sem nenhuma junção com o processamento de algum documento aninhado em nosso documento e também melhor escalabilidade com alguns bons recursos de escalonamento do MongoDB.

- Para sincronizar nosso lado de leitura e gravação, temos 2 opções com o uso da Arquitetura Orientada a Eventos (sem usar fluxos de eventos no fornecimento de eventos):

    1. Se nossos lados de leitura estiverem no mesmo serviço, durante o salvamento de dados no lado de gravação, eu salvo um registro de comando interno no armazenamento do processador de comando (como algo que fazemos no padrão de caixa de saída) e depois de confirmar o lado de gravação, nosso gerenciador de processador de comando lê comandos não enviados e envia -los para seus manipuladores de comando no mesmo serviço correspondente e esses manipuladores podem salvar seus modelos de leitura em nosso banco de dados MongoDb como um lado de leitura.
    2. Se nossos Read Sides estiverem em Another Services, publicamos um evento de integração (salvando esta mensagem na caixa de saída) após confirmar nosso write side e todos os nossos assinantes poderão obter este evento e salvá-lo em seus modelos de leitura (MongoDB).

- Aqui usei o Outbox para entrega garantida e pode ser usado como zona de destino para eventos de integração antes de serem publicados no corretor de mensagens.

- O padrão de caixa de saída garante que uma mensagem foi enviada (por exemplo, para uma fila) com êxito pelo menos uma vez. Com este padrão, em vez de publicar diretamente uma mensagem na fila, colocamos-a no armazenamento temporário (por exemplo, tabela de banco de dados) para evitar a perda de qualquer mensagem e algum mecanismo de nova tentativa em caso de falha (Entrega pelo menos uma vez). Por exemplo, quando salvamos dados como parte de uma transação em nosso serviço, também salvamos mensagens (Eventos de Integração) que

posteriormente desejamos processar em outros microsserviços como parte da mesma transação. A lista de mensagens a serem processadas é chamada StoreMessage com Message Delivery Type Outbox que faz parte do nosso serviço MessagePersistence. Essa infraestrutura também oferece suporte ao tipo de entrega de mensagens na caixa de entrada e ao tipo de entrega de mensagens internas (processamento interno).

- Também temos um serviço de background MessagePersistenceBackgroundService que verifica periodicamente nossos StoreMessages no banco de dados e tenta enviar as mensagens para o corretor usando nosso serviço MessagePersistenceService. Após obter a confirmação da publicação (por exemplo, ACK do corretor), ele marca a mensagem como processada para evitar o reenvio. Porém, é possível que não consigamos marcar a mensagem como processada devido a erro de comunicação, por exemplo, o corretor não está disponível. Neste caso nosso MessagePersistenceBackgroundService tenta reenviar as mensagens que não foram processadas e na verdade é uma entrega pelo menos uma vez. Podemos ter certeza de que a mensagem será enviada uma vez, mas também pode ser enviada várias vezes! É por isso que outro nome para essa abordagem é entrega única.

- Também aqui usei RabbitMQ como meu Message Broker para minha comunicação assíncrona entre os microsserviços usando mecanismo de consistência eventual, por enquanto usei ferramentas MassTransit para fazer comunicações de corretor. além dessa consistência eventual, temos uma chamada síncrona com o uso de REST (no futuro usarei gRpc) para nossas necessidades imediatas de consistência.

- Usamos um Api Gateway e aqui usei YARP que é proxy reverso da Microsoft (poderíamos usar envoy, traefik, Ocelot, ...), na frente de nossos serviços, também poderíamos ter vários Api Gateway para atingir o padrão BFF. por exemplo, um Gateway para aplicativos móveis, Um Gateway para aplicativos da web e etc. Com o uso do API Gateway, nossos microsserviços internos são transparentes e o usuário não pode acessá-los diretamente e todas as solicitações serão atendidas por meio deste Gateway. Também poderíamos usar gateway para balanceamento de carga, autenticação e autorização, cache,...

4 Link do repositorio e documentação do Projeto

O link é esse:`https://github.com/mehdihadeli/food-delivery-microservices?tab=readme-ov-file`