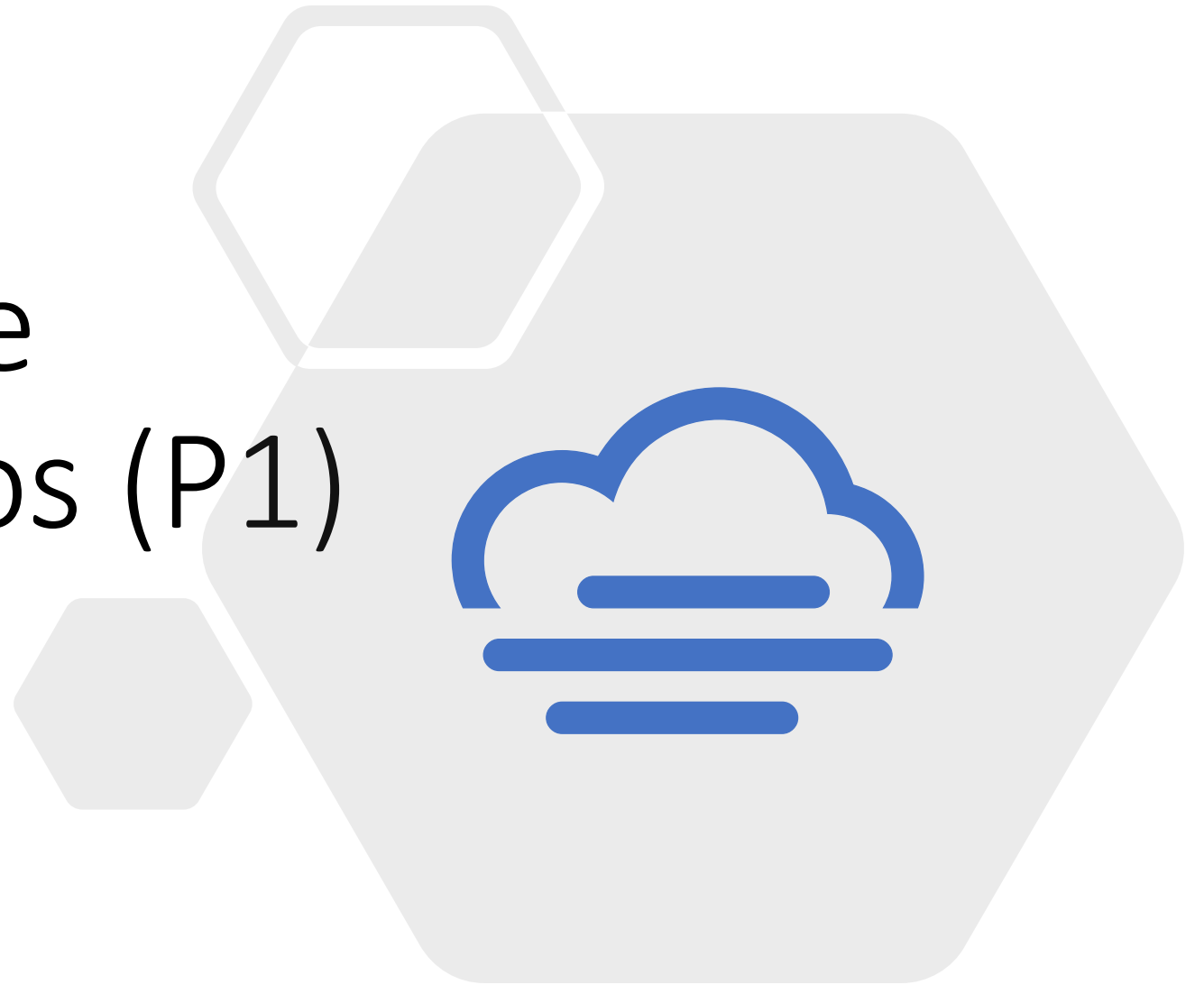


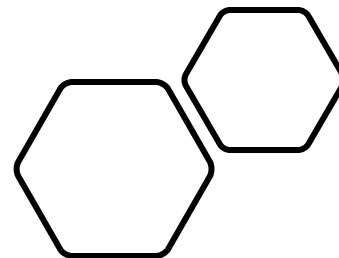
Arquitetura de Micros serviços (P1)

Prof. Bernardo Copstein

Prof. Júlio Machado



Introdução a arquitetura de Micros serviços



Leituras recomendadas:

- Sommerville, Ian. Engineering Software Products – An Introduction to Modern Software Engineering. Pearson, 2019. Capítulo 6.
- Newman, Sam. Building Microservices. O'Reilly Media Inc, 2015.

+

•

○


Decomposição de sistemas

- Saber como decompor um sistema em componentes é uma das principais tarefas de um arquiteto de software
 - Podem ser desenvolvidos em paralelo
 - Podem ser reusados e/ou substituídos se a tecnologia sob a qual foram construídos mudar
 - Podem ser distribuídos em diferentes computadores



Decomposição x “cloud based”



- Entre as vantagens de software baseado em nuvem podemos citar:
 - Escalabilidade
 - Confiabilidade
 - Elasticidade (gestão dos recursos)
 - Para que se possa usufruir dessas vantagens são necessários componentes que possam:
 - ser replicados
 - executados em paralelo
 - movidos de um servidor para outro
 - Para tanto:
 - Não é possível usar objetos convencionais
 - Serão necessários serviços sem estado que mantenham toda a informação persistente em um banco de dados
- 

A Evolução do desenvolvimento de sistemas

DDD nos mostrou a importância de representar o mundo real no código

Os conceitos de integração contínua nos mostraram a melhor maneira de colocar um software em produção

As arquiteturas Hexagonal e Limpa nos levaram para além do modelo de camadas

As plataformas virtuais permitiram provisionar e alterar o tamanho das máquinas de forma automática

Algumas grandes organizações entenderam a importância de equipes pequenas

Microserviços

Micros serviços emergiram da evolução nas formas de desenvolver sistemas

Não foram inventados ou descritos antes de surgirem

Emergiram como uma tendência ou um padrão a partir do uso real

Mas existem apenas em função de tudo que veio antes

+

•

○

Micros serviços

- Uma arquitetura de micros serviços se constitui em uma coleção de pequenos serviços autônomos autocontidos, de granularidade fina, e que implementam uma única peça de responsabilidade de negócio
- Não é uma questão de tamanho, é uma questão de escopo

Características gerais

Pequenos,
independentes e
fracamente
acoplados

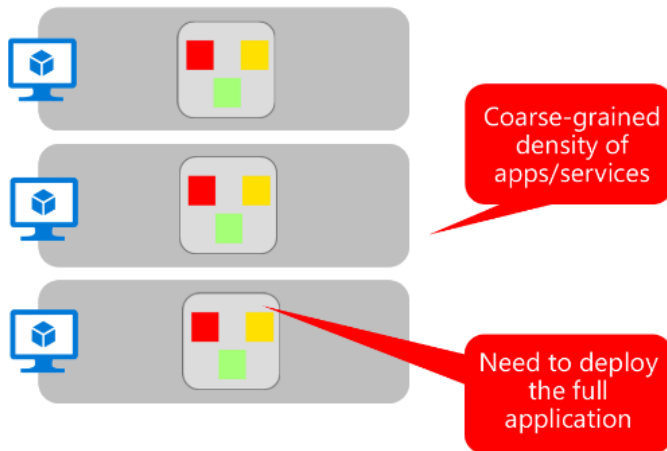
Mantidos em
bases-de-código
separadas

Implantados de
forma
independente de
outros serviços

Micros serviços

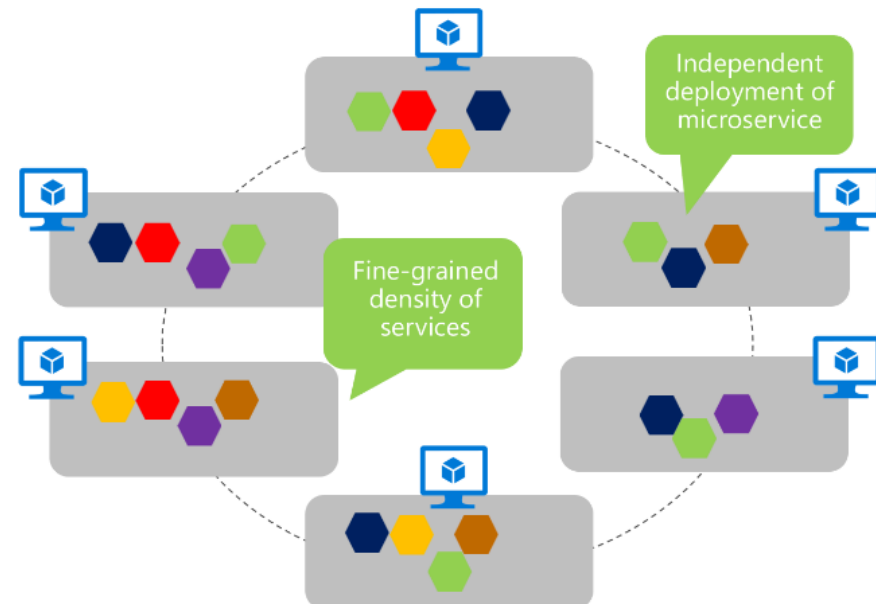
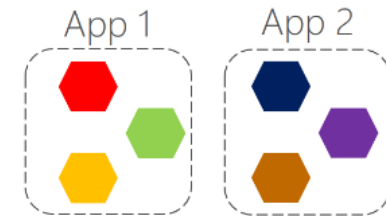
Monolithic deployment approach

- A traditional application has most of its functionality within a few processes that are componentized with layers and libraries.
- Scales by cloning the app on multiple servers/VMs



Microservices application approach

- A microservice application segregates functionality into separate smaller services.
- Scales out by **deploying each service independently** with multiple instances across servers/VMs



Características (I)



Autocontido

Os microsserviços não têm dependências externas. Eles gerenciam seus próprios dados e implementam sua própria interface de comunicação.



Leve

Os microsserviços se comunicam usando protocolos leves, para que as sobrecargas de comunicação de serviço sejam baixas.



Independente de implementação

Os microsserviços podem ser implementados usando diferentes linguagens de programação e podem usar diferentes tecnologias (por exemplo, diferentes tipos de banco de dados) em sua implementação.



Implantável de forma independente

Cada microsserviço é executado em seu próprio processo e pode ser implantado de forma independente, usando sistemas automatizados.



Orientado à Negócios

Os microsserviços devem implementar recursos e necessidades de negócios, em vez de simplesmente fornecer um serviço técnico.


Características (II)

Um micro serviço bem projetado deve ter alta coesão e baixo acoplamento.

- A coesão é uma medida do número de relacionamentos que as partes de um componente têm entre si. Alta coesão significa que todas as partes necessárias para fornecer a funcionalidade do componente estão incluídas no componente.
- O acoplamento é uma medida do número de relacionamentos que um componente tem com outros componentes do sistema. Baixo acoplamento significa que os componentes não têm muitos relacionamentos com outros componentes.

Cada micro serviço deve ter uma única responsabilidade, ou seja, deve fazer apenas uma coisa e deve fazê-lo bem.

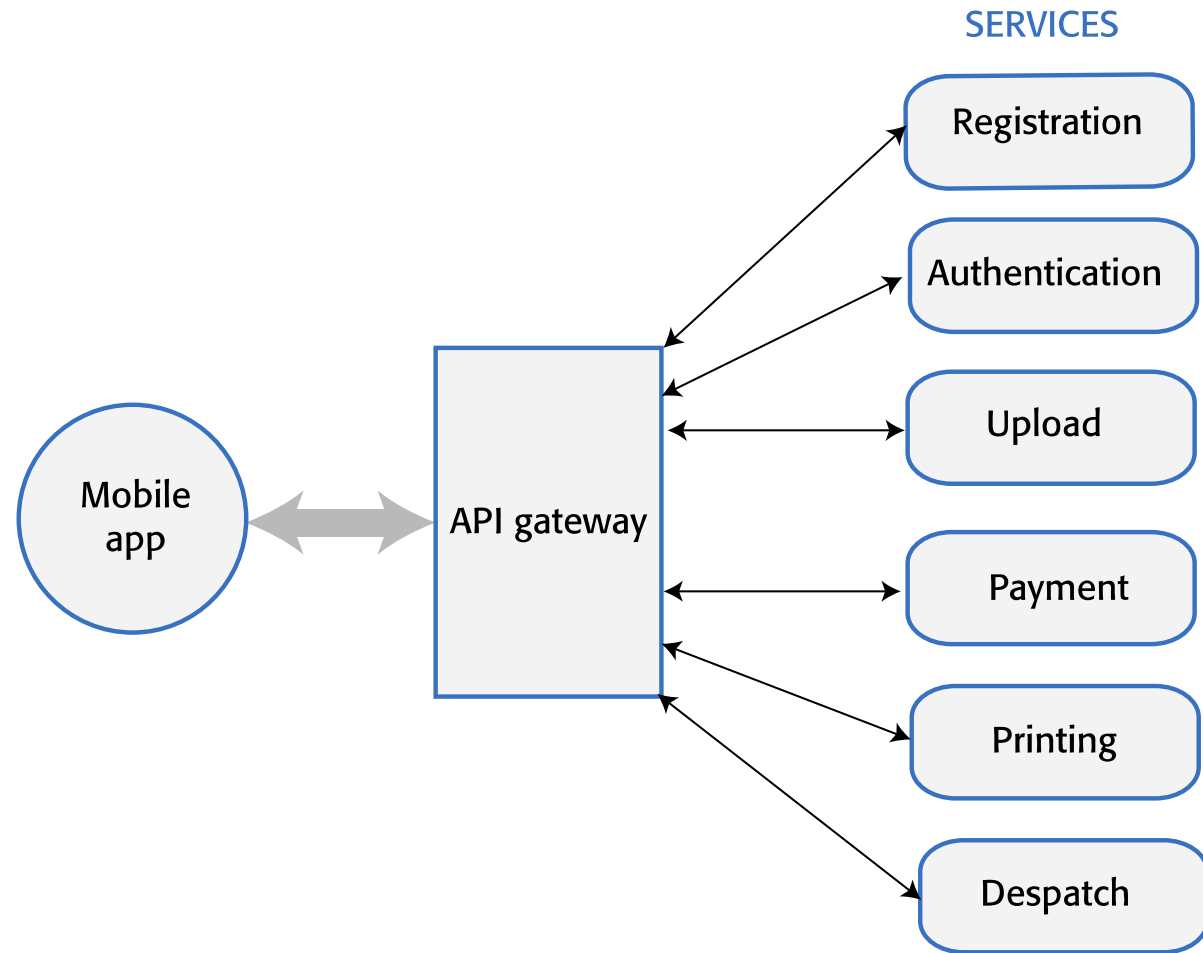
- No entanto, 'uma coisa só' é difícil de definir de uma forma que seja aplicável a todos os serviços.
- A responsabilidade nem sempre significa uma atividade única e funcional.



Exemplo: um sistema de impressão de fotos para dispositivos móveis

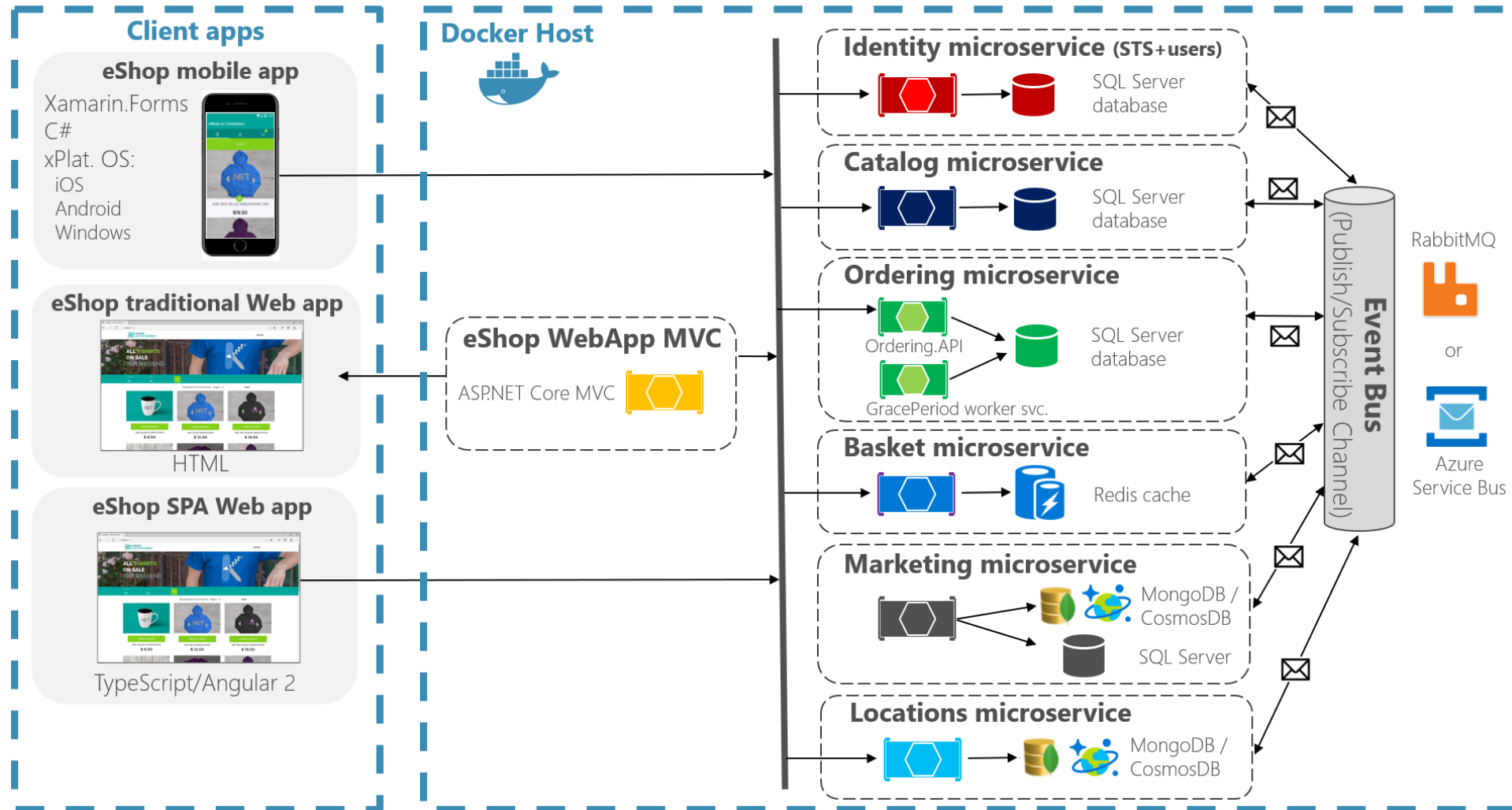
- Os usuários podem fazer upload de fotos para o seu servidor a partir do telefone ou especificar fotos da conta do Instagram que gostariam de imprimir. As impressões podem ser feitas em diferentes tamanhos e em diferentes mídias.
- Os usuários podem escolher o tamanho de impressão e o meio de impressão. Por exemplo, eles podem decidir imprimir uma foto em uma caneca ou em uma camiseta. As impressões ou outras mídias são preparadas e depois entregues em sua casa. Eles pagam pelas impressões usando um serviço de pagamento como Android ou Apple Pay ou registrando um cartão de crédito no provedor de serviços de impressão.

Uma arquitetura
de micros
serviços para um
sistema de
impressão de
fotos



eShopOnContainers reference application

(Development environment architecture)



<https://github.com/dotnet-architecture/eShopOnContainers>

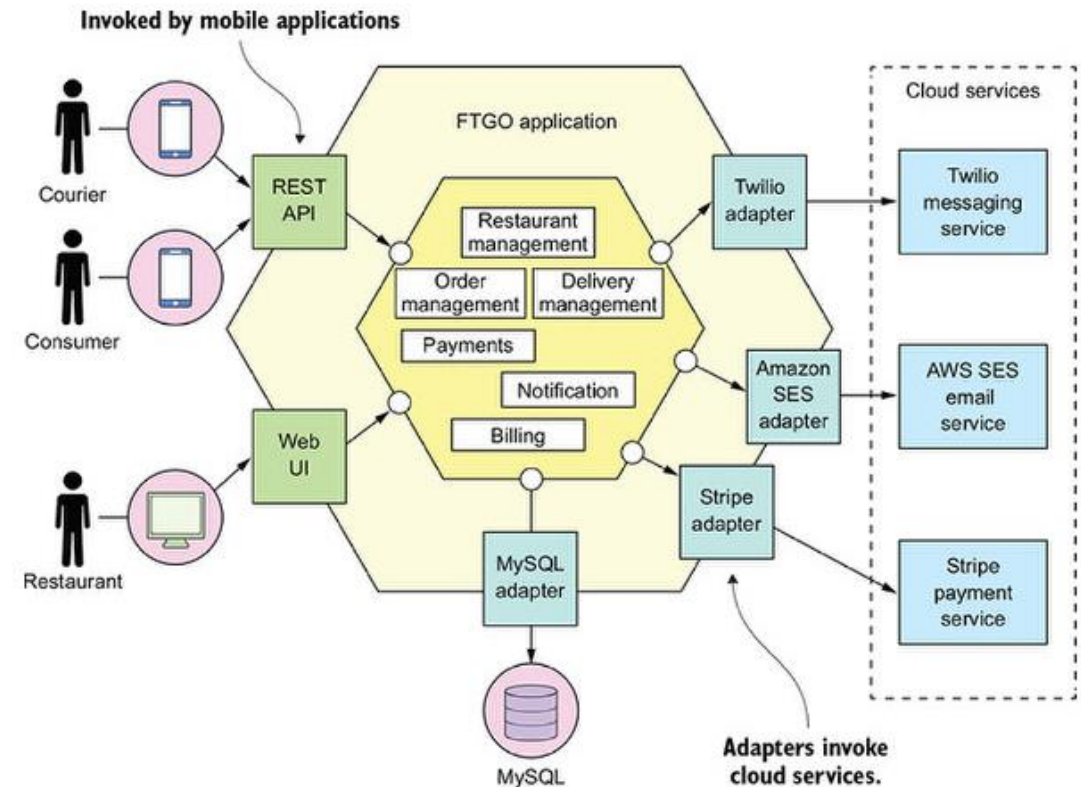
Exemplo monolito → microserviços

Sistema FGTO

- O sistema “Fast To GO” (FGTO) é um sistema que congrega uma cadeia de restaurantes com um serviço de tele-entrega.
- Em princípio é um sistema muito simples: clientes usam um aplicativo mobile pra fazer pedidos em restaurantes locais. Os restaurantes usam um website para editar seus menus e receber os pedidos. Os entregadores usam um aplicativo mobile específico. O FGTO coordena tanto o envio dos pedidos para os restaurantes como os serviços de entrega. Além disso faz a cobrança e prove os pagamentos tanto dos serviços de entrega como dos restaurantes. Este sistema usa vários serviços web, incluindo “Stripe” para pagamentos, “Twilio” para messengeria e “Amazon Simple Email Service” para emails.

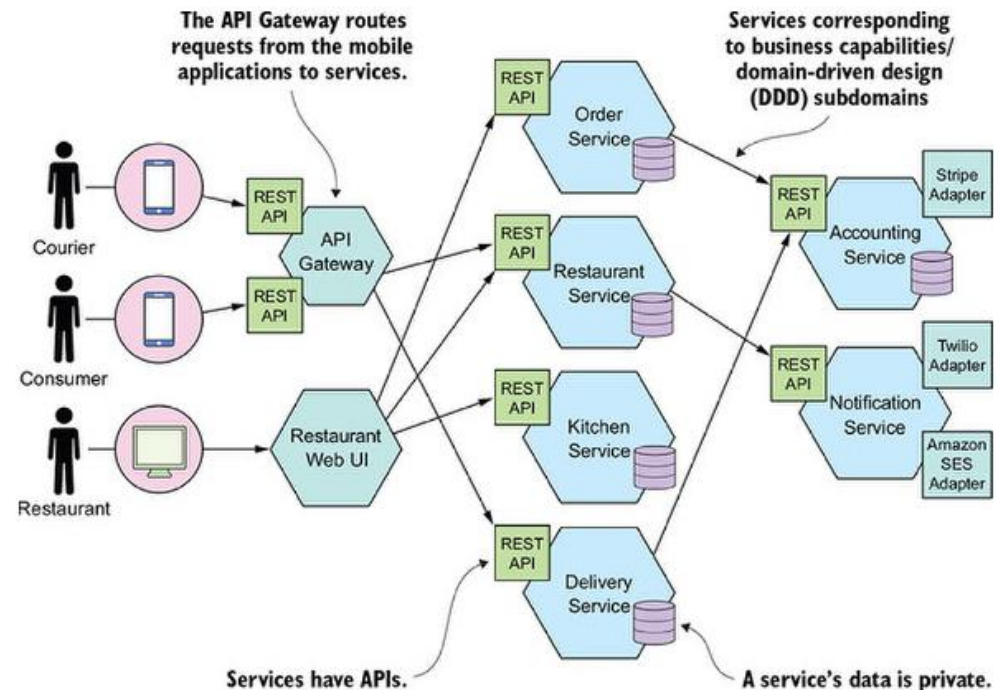
Arquitetura do monolito

- O aplicativo FGTO é um monolito de arquitetura hexagonal. Consiste de uma lógica de negócios cercada de adaptadores que implementam a interface com o usuário e com sistemas externos tais como aplicações mobile, banco de dados, serviços de pagamento em nuvem, email etc.
- Inicialmente foi construído como um monolito Java, consistindo de um único arquivo “.jar”. Com o tempo o Sistema foi se tornando complexo e, apesar dos esforços da equipe, acabou se tornando uma “big ball of mud”.



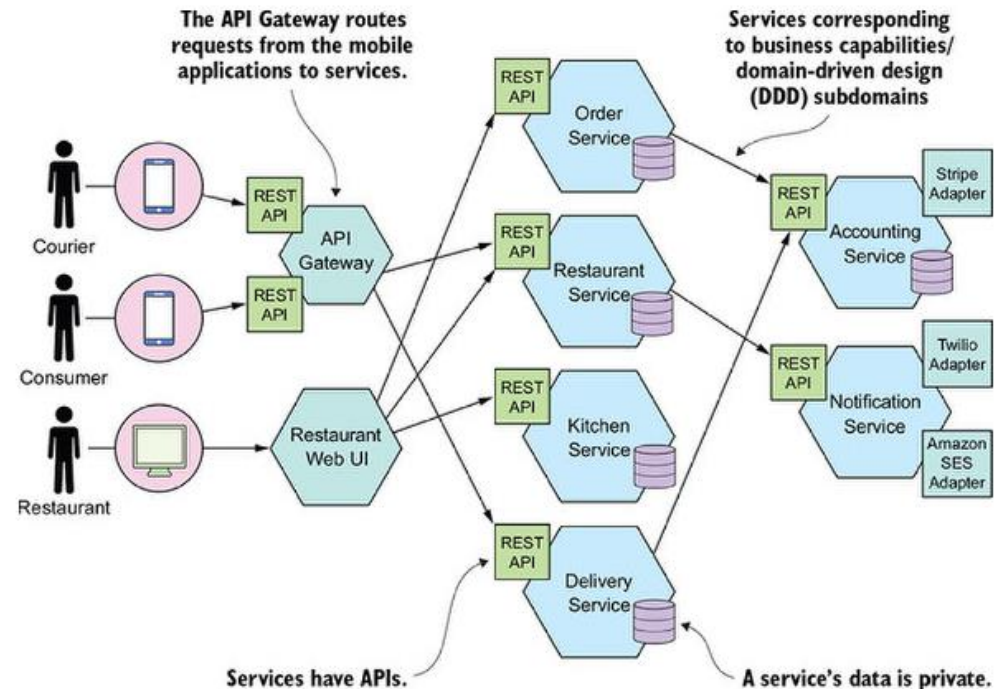
FGTO em uma arquitetura de microserviços

- Os serviços de frontend incluem aplicações mobile para os clientes e entregadores e uma aplicação WEB para os restaurantes
- Um API Gateway faz as vezes de fachada para as aplicações mobile



FGTO em uma arquitetura de microserviços

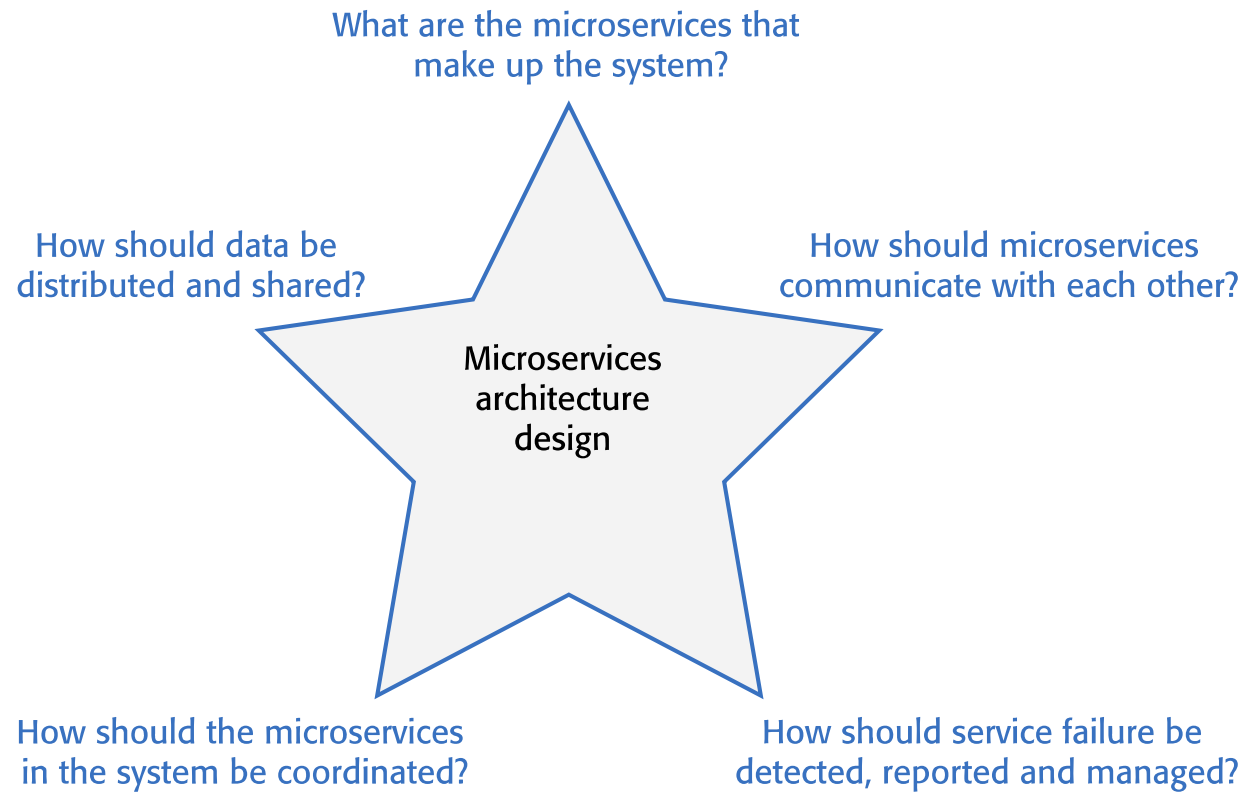
- A lógica de negócios é composta por uma série de “backends”:
 - Order Service—gerencia os pedidos
 - Delivery Service—gerencia as entregas dos restaurantes para os clientes
 - Restaurant Service— mantém a informação sobre os restaurantes
 - Kitchen Service—gerencia os pedidos
 - Accounting Service – gerencia os pagamentos
- A maioria dos serviços corresponde aos módulos do monolito. A diferença é que estão claramente definidos, podem ser desenvolvidos, testados, instalados e escalados individualmente.



Benefícios da arquitetura de microsserviços

- Os micros serviços são autocontidos e executados em processos separados.
- Em sistemas baseados em nuvem, cada micros serviço pode ser implantado em seu próprio contêiner. Isso significa que um micros serviço pode ser interrompido e reiniciado sem afetar outras partes do sistema.
- Se a demanda em um serviço aumentar, as réplicas de serviço podem ser criadas e implantadas rapidamente. Estes não requerem um servidor mais poderoso, então 'scaling-out' é, normalmente, muito mais barato do que 'scaling up'.

Arquitetura de micros serviços - principais questões de design



Diretrizes de decomposição



Equilibre a funcionalidade de grão fino e o desempenho do sistema

Os serviços de função única significam que as alterações são limitadas a menos serviços, mas exigem comunicações de serviço para implementar a funcionalidade do usuário. Isso torna o sistema mais lento devido à necessidade de cada serviço agrupar e desagrupar mensagens enviadas de outros serviços.



Siga o 'princípio de fechamento comum'

Elementos de um sistema que provavelmente serão alterados ao mesmo tempo devem estar localizados dentro do mesmo serviço. A maioria dos requisitos novos e alterados deve, portanto, afetar apenas um único serviço.



Associar serviços a recursos de negócios

Uma capacidade de negócios é uma área discreta de funcionalidade de negócios que é responsabilidade de um indivíduo ou de um grupo. Você deve identificar os serviços necessários para dar suporte a cada recurso de negócios.



Projete serviços para que eles tenham acesso apenas aos dados de que precisam

Se houver uma sobreposição entre os dados usados por diferentes serviços, você precisará de um mecanismo para propagar as alterações de dados para todos os serviços usando os mesmos dados.

