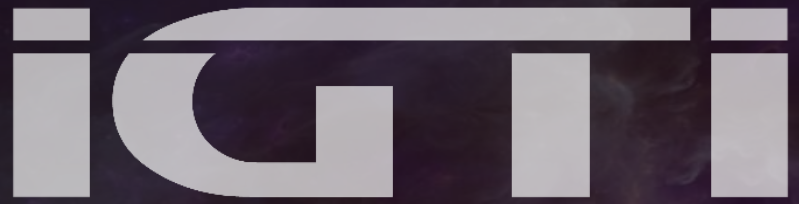


A aula interativa do **Módul 3 - Bootcamp Arquiteto de Software** começará em breve!

Atenção:

1) Você entrará na aula com o microfone e o vídeo DESABILITADOS.
2) Apenas a nossa equipe poderá habilitar seu microfone e seu vídeo em momentos de interatividade, indicados pelo professor.
3) Utilize o recurso Q&A para dúvidas técnicas. Nossos tutores e monitores estarão prontos para te responder e as perguntas não se perderão no chat.
4) Para garantir a pontuação da aula, no momento em que o professor sinalizar, você deverá ir até o ambiente de aprendizagem e responder a enquete de presença. Não é necessário encerrar a reunião do Zoom, apenas minimize a janela.

Design Patterns, Estilos e Padrões Arquiteturais

Segunda Aula Interativa

Prof. Vagner Clmentino dos Santos

Nesta aula

- Tipos de Estilo Arquiteturais.
- Falácias da Computação Distribuída.
- Padrões Arquiteturais Históricos e Modernos.
- Exercício de fixação.
- Espaço para dúvidas.

Acordos

- Pratique a Paciência
- Pratique a Empatia
- Pratique a Curiosidade
- Pratique o Networking

Espaço para dúvidas

https://bit.ly/2HB30D0

Tipos de Estilos Arquiteturais

- Os estilos de arquitetura podem ser classificados como monolítico ou distribuído
- No monolítico é possível identificar um único deployment
- No distribuído o build resulta em dois ou mais “executáveis” que estão conectados por meio de protocolos de acesso remoto

Exemplos

Tipos de Estilos Arquiteturais	
Monolítico	Distribuído
• **Arquitetura em Camada** • **Pipeline** • **Microkernel**	• Arquitetura Orientada à Eventos • Arquitetura Orientada à Serviço • Microsserviços

Características do estilo monolítico

1. Mais simples e barato
2. Menor tempo de desenvolvimento
3. Bom ponto de partida

Características do estilo distribuído

1

Mais poderoso em termos de desempenho, escalabilidade e disponibilidade

2

Desenvolvimento mais complexo

3

Traz consigo diversos “trade-offs”

Falácias da Computação Distribuída

- A rede é confiável
- Latência é zero
- A largura de banda (bandwidth) é infinita
- A rede é segura
- A topologia da rede (roteadores, hubs, switch, firewall etc.) nunca muda
- Existe um único administrador (dono) do sistema para colaborar e se comunicar
- O custo em dinheiro de transportar algo pela rede, como uma chamada Restful, é zero

Padrões Históricos

- Padrões de Aplicações Corporativas
- Padrões de Integração
- ESB
- Web services

Padrões Modernos

- Circuit Breaker
- Command and Query Responsibility Segregation (CQRS)
- Event Sourcing
- Sidecar
- Backend-for-Frontend

Problema

1

Os sistemas distribuídos devem ser projetados levando em consideração possíveis falhas.

2

Serviços remotos podem não responder por vários motivos.

3

Implementar novas tentativas pode ajudar, contudo, em casos de falha total, é inútil tentar novamente.

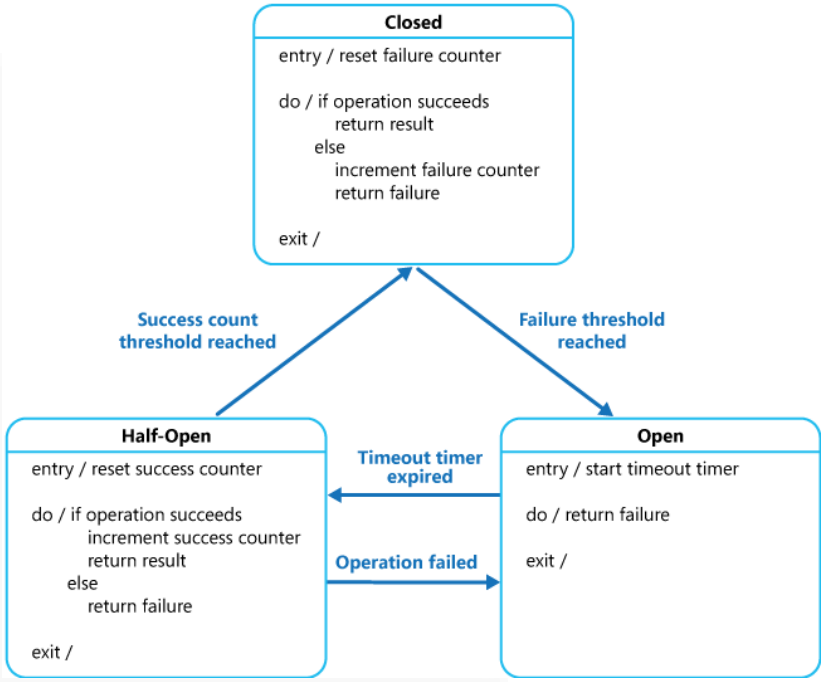
Circuit Breaker

Pode impedir que um aplicativo execute repetidamente uma operação em que a falha é provável.

Atua como um proxy que monitora o número de falhas recentes ocorridas e decide se a operação deve continuar ou se deve retornar uma exceção.

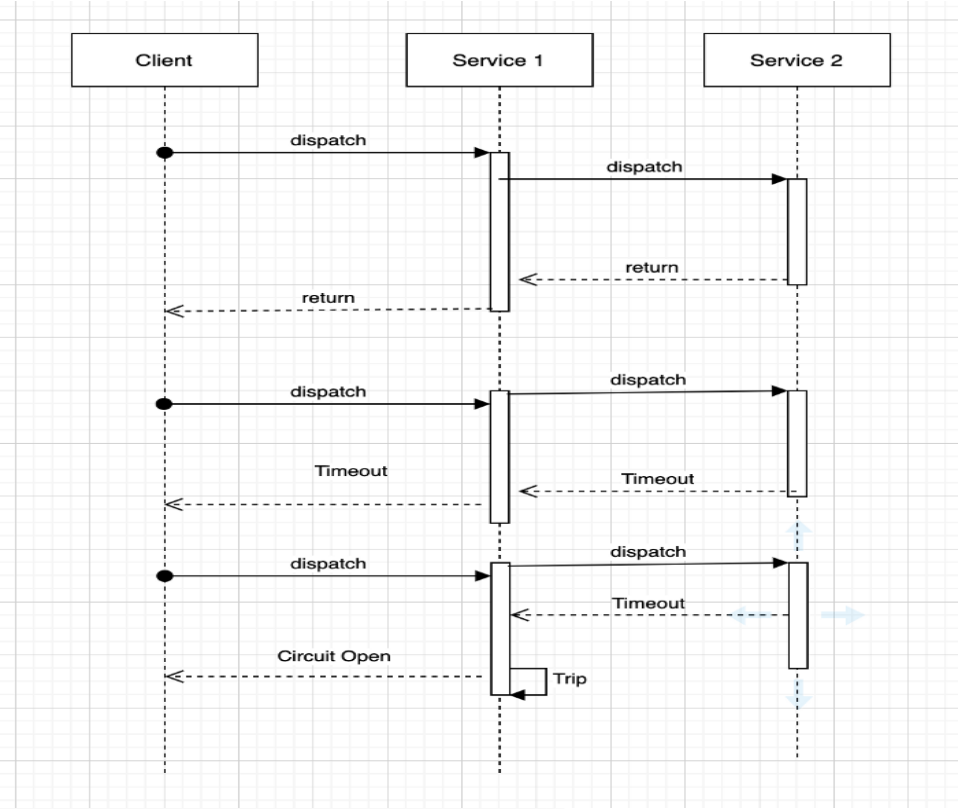
O proxy pode ser implementado como uma máquina de estados: **Fechado**, **Aberto** e **Entreaberto**

Estados



Fonte: https://docs.microsoft.com/en-us/azure/architecture/patterns/circuit-breaker

Funcionamento



Fonte: https://medium.com/better-programming/modern-day-architecture-design-patterns-for-software-professionals-9056ee1ed977#4c75

Quando usar

- Para impedir que um aplicativo tente invocar um serviço remoto ou acessar um recurso compartilhado, se a operação tiver alta probabilidade de falhar.

- Para tratar o acesso a recursos particulares locais em um aplicativo, como a estrutura de dados na memória.
- Como substituto para o tratamento de exceções na lógica de negócios dos seus aplicativos.

Problema

1

Nas arquiteturas tradicionais, o mesmo modelo de dados é usado para consultar e atualizar o repositório de dados

2

Em geral, leitura e escrita são assimétricas, com requisitos de desempenho e de escala muito diferentes

3

Utilizar uma única fonte de armazenamento para leitura e escrita pode acarretar em problemas de desempenho.

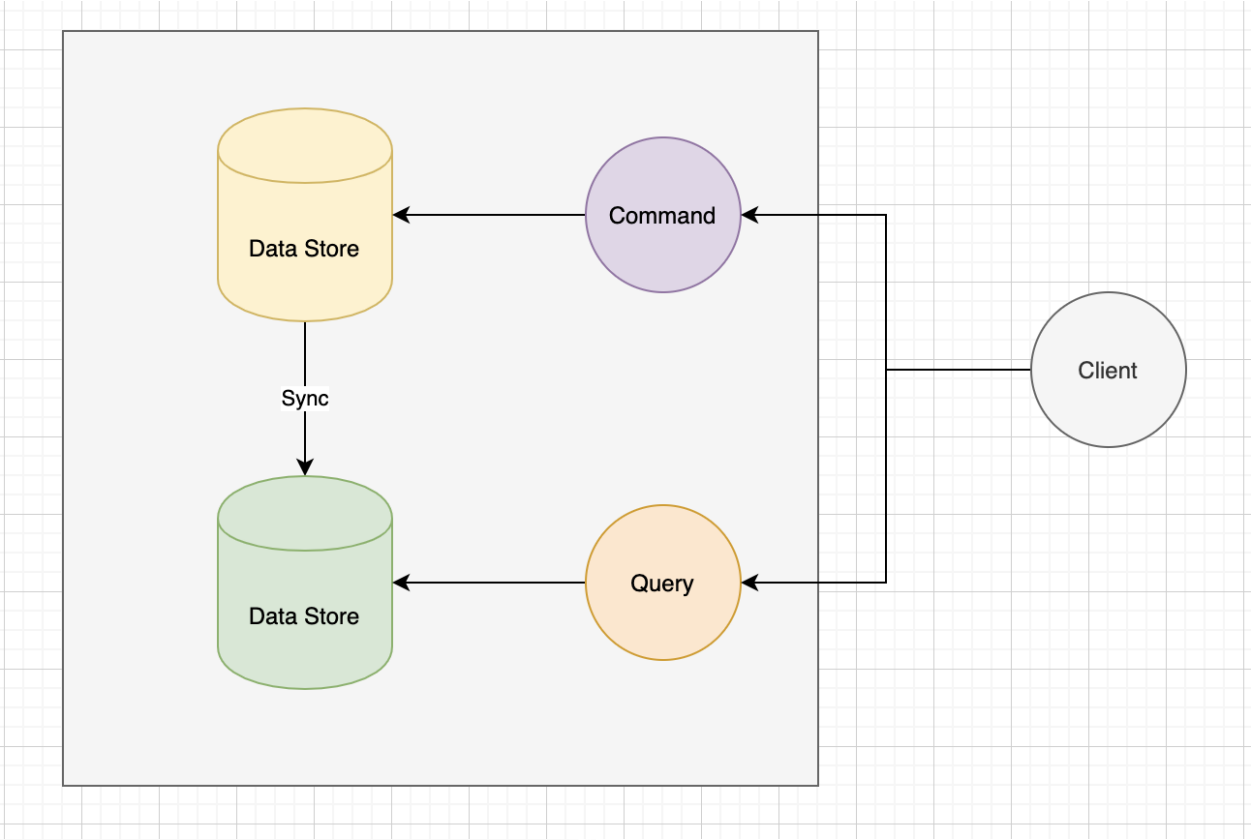
Command and Query Responsibility Segregation (CQRS)

CQRS separa leituras e gravações em modelos diferentes, usando **commands** e **queries**.

Commands devem ser baseados em tarefas e podem ser processados de maneira assíncrona

Queries não devem modificar o banco e devem retornar um DTO com nenhum conhecimento do domínio

Funcionamento



Fonte: https://medium.com/better-programming/modern-day-architecture-design-patterns-for-software-professionals-9056ee1ed977#4c75

Quando usar

- Aplicação esperando um grande número de leituras e escritas
- Quando você quer fazer o tunning das operações de leitura e escrita

- O domínio ou as regras de negócio são simples
- Em um CRUD onde as operações de acesso a dados são suficientes.

Problema

1. As aplicações trabalham com dados mantendo o estado atual e atualizando-o quando necessário

2. Atualizações diretas no banco de dados podem diminuir o desempenho e a capacidade de resposta

3. Atualizações frequentes aumentam a probabilidade de inconsistência nos dados

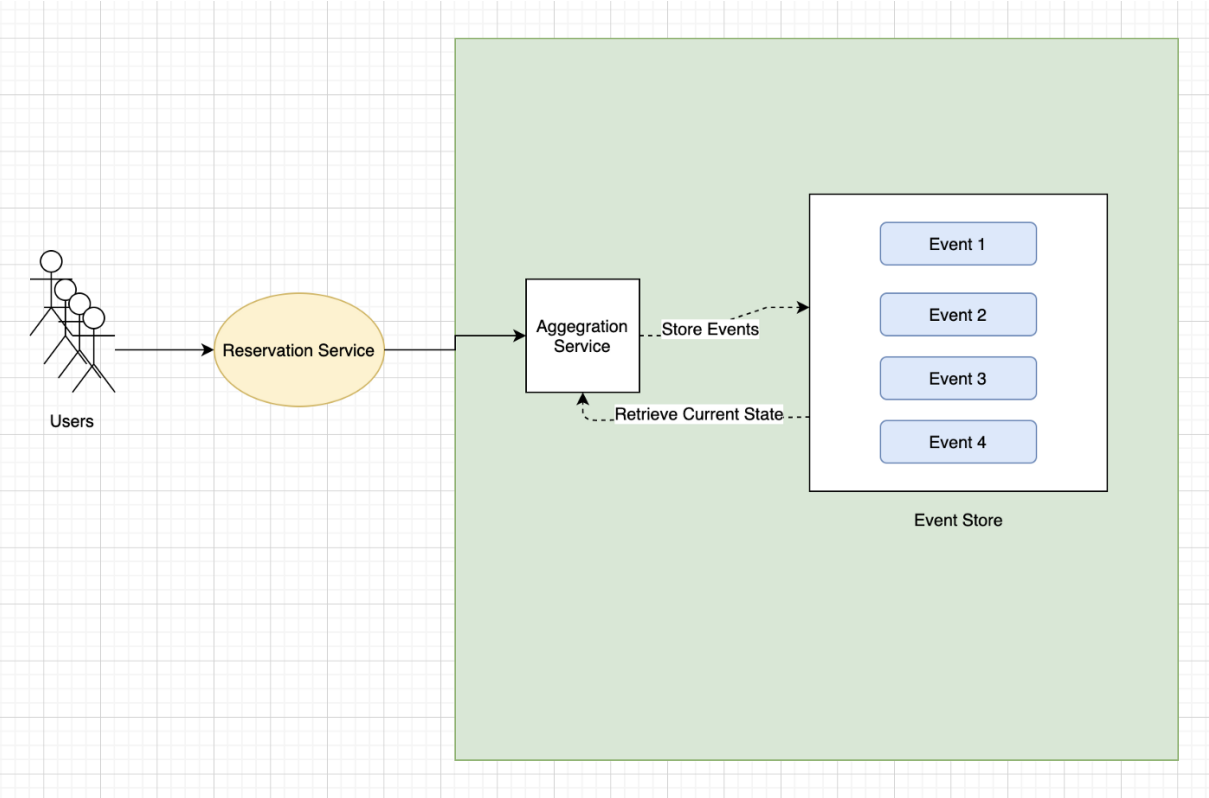
Event Sourcing

Uma sequência de eventos é armazenada como um journal (log sequencial)

Uma visão agregada do journal dá o estado atual da aplicação.

Usado para sistemas que não podem arcar com o lock (isolamento) dos dados

Funcionamento



Fonte: https://medium.com/better-programming/modern-day-architecture-design-patterns-for-software-professionals-9056ee1ed977#4c75

Quando usar

- Quando as operações regulares de CRUD não são suficientemente boas
- Tipicamente adequado para sistemas de reserva ou sistemas de comércio eletrônico
- Quando há uma exigência de forte auditoria

- Domínios pequenos ou simples que funcionam bem com os mecanismos tradicionais de gerenciamento de dados
- Sistemas em que há uma baixa ocorrência de atualizações nos mesmos dados

Problema

1

Aplicações dependem de interesses transversais (ex. monitoramento, log, configuração etc.)

2

Um interrupção dessa funcionalidades poderia afetar alguns componentes ou toda a aplicação

3

Essas tarefas periféricas podem ser implementadas como componentes ou serviços separados

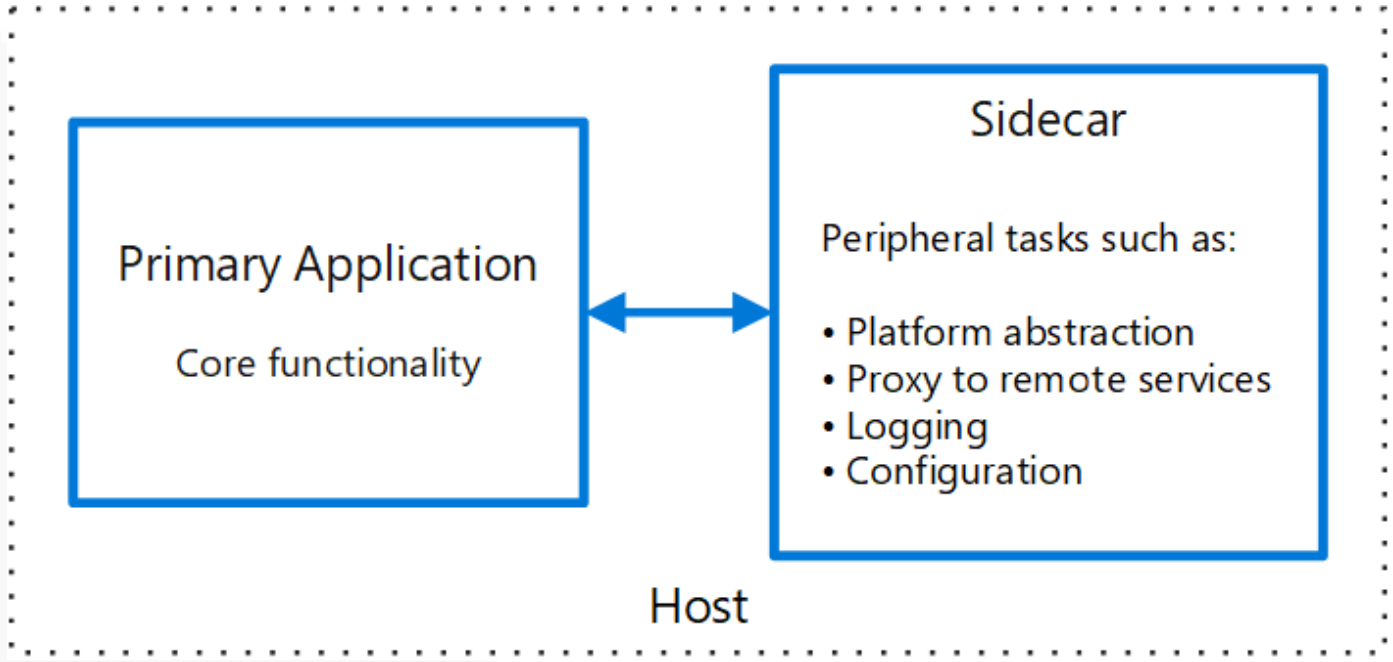
Sidecar

As funcionalidades principais (domínio) ficam na aplicação principal, em seu próprio processo ou container

Define-se uma interface homogênea de comunicação do serviço com o Sidecar

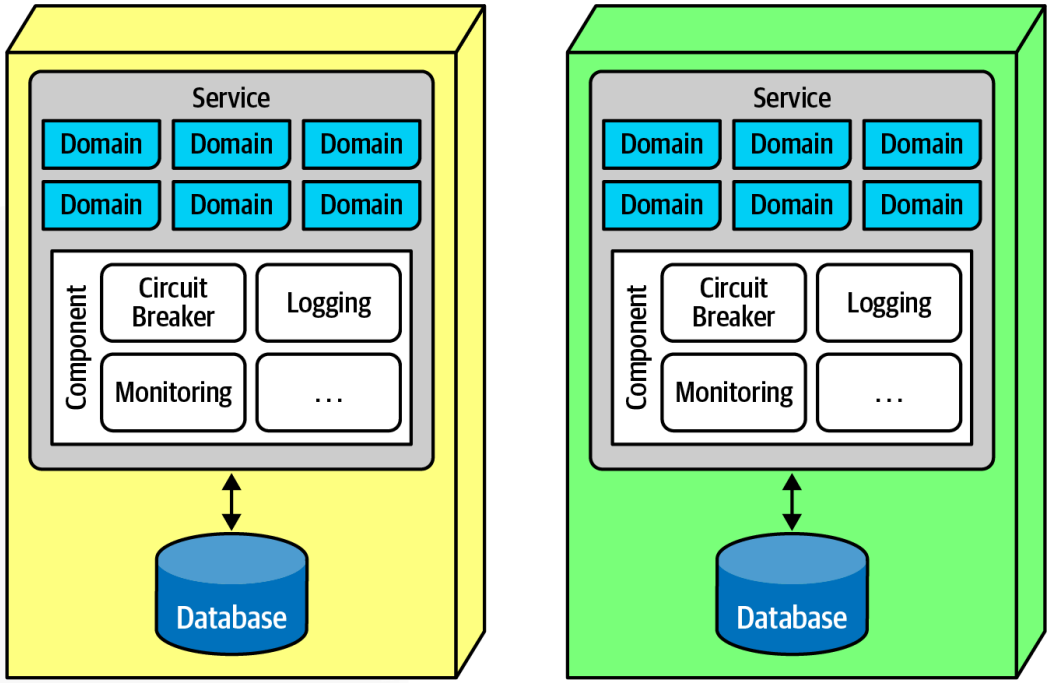
Um serviço com a função de Sidecar não faz parte da aplicação, mas está conectado a ela.

Funcionamento



Fonte: https://docs.microsoft.com/en-us/azure/architecture/patterns/sidecar

Sidecar em microsserviços



Fonte: RICHARDS e FORD, 2020

Quando usar

- Trabalhando com múltiplos e heterogêneos microsserviços no mesmo produto
- Você precisa de um serviço que compartilhe o ciclo de vida geral de sua aplicação principal, mas que possa ser atualizado independentemente

- Quando a comunicação entre processos precisa ser otimizada
- Para pequenas aplicações onde o custo de implantação de um sidecar para cada instância não vale a pena

Problema

1

Uma aplicação é inicialmente projetada para atender uma interface web

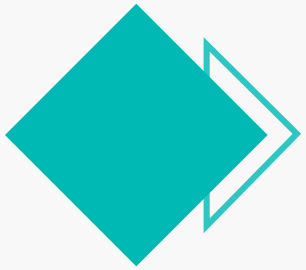
2

Com o aumento da base de usuários é desenvolvido um aplicativo móvel que interage com o mesmo backend

3

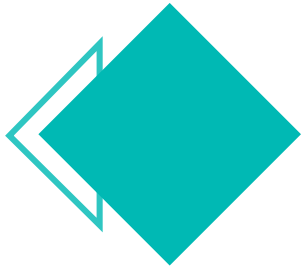
O backend torna-se um serviço para fins gerais, atendendo os requisitos da versão web e mobile

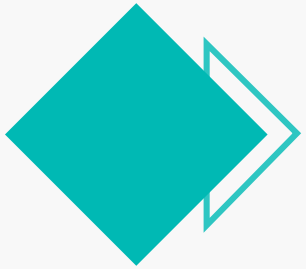
Backend-for-Frontend



Implementa um backend por interface de usuário

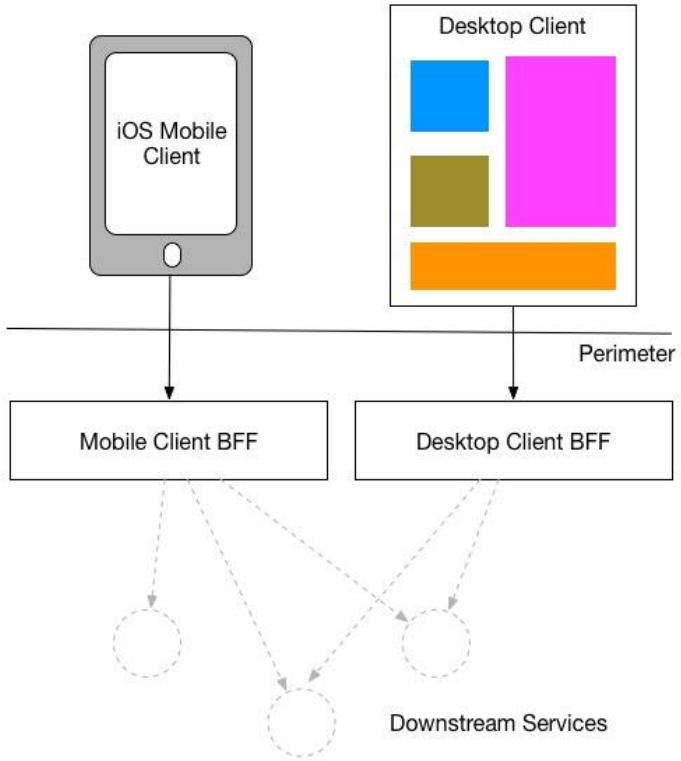
Ajusta o comportamento do backend para melhor atender às necessidades do frontend, sem se preocupar em afetar as demais experiências de usuário





Como cada backend é específico por interface gráfica, ele pode ser otimizado

Funcionamento



Fonte: https://samnewman.io/patterns/architectural/bff

Quando usar

- Um serviço backend de propósito geral está ficando complexo de manter
- O usuário deseja otimizar o backend para as exigências específicas de cada interface de usuário

- Não há muita diferença entre as requisições realizadas pelas diferentes interfaces gráficas ao backend
- Quando apenas uma interface é usada para interagir com o backend.

Vamos exercitar...

Acesse o www.kahoot.it

Digite o código fornecido pelo professor.

Espaço para dúvidas

https://bit.ly/2HB30D0