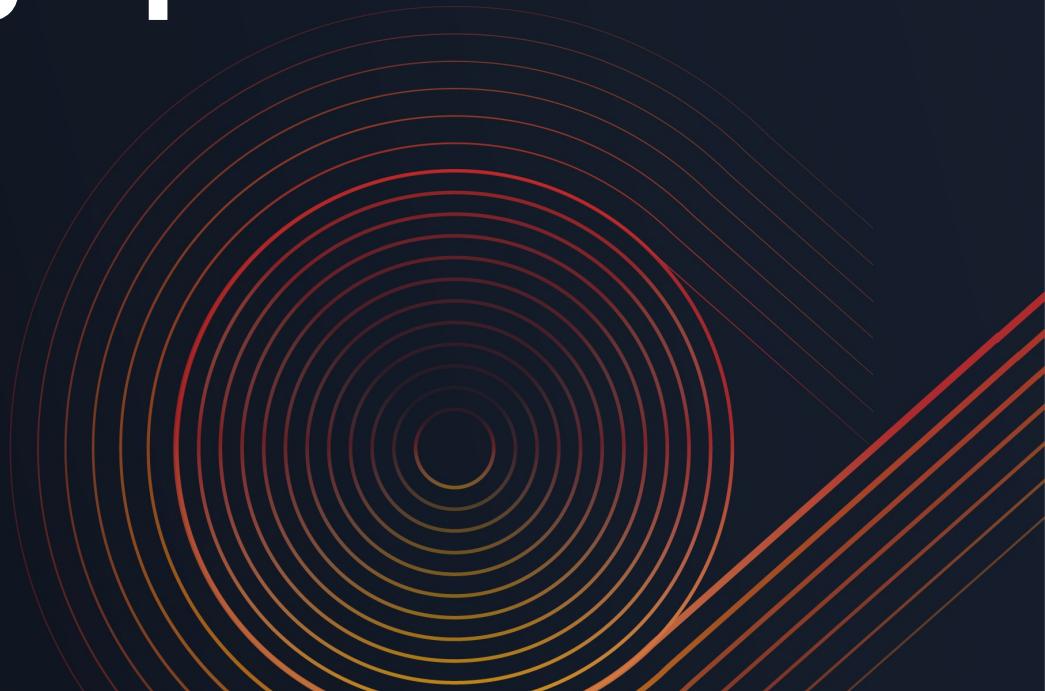




# Arquiteturas e Padrões de Design para Aplicações Modernas

Peterson Larentis

Enterprise Solutions Architect, AWS



# Apresentação



/peterson-larentis



Peterson Larentis é **Arquiteto de Soluções na AWS e professor de MBA na FIAP**, pragmático e entusiasta por **Cloud Computing, Serverless, DevOps e desenvolvimento Ágil**.

## Head Trainer da iWorks Education

Na AWS auxilia os maiores e-commerces da américa latina a construírem arquiteturas seguras, confiáveis, elásticas e escaláveis com excelência operacional utilizando as melhores práticas ágeis de engenharia de software e é **responsável pela Track de DevOps no AWS Summit Brasil**.

Na FIAP ministra disciplinas de Serverless, Microservices, DevOps e Engenharia de Software ágil nos MBA em Cloud Computing, MBA em Full Stack Development e MBA em Business Agility.

Formado em Análise e Desenvolvimento de Sistemas possui MBA em Marketing Digital pela Fundação Getúlio Vargas e 6 AWS Certifications.

# Tópicos

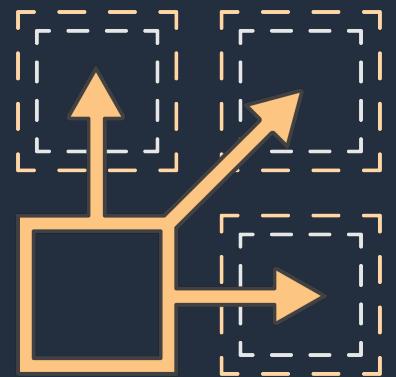
- The Twelve-App Factor
- Aplicações Orientadas a Eventos
- Serviços AWS e Padrões de Arquitetura

# 3 Principais Pilares da Modernização

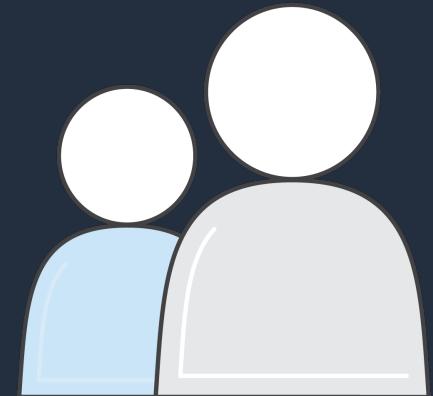


Operações e governança em escala

Automatizar, habilitar e fazer  
autoatendimento



Tecnologia e Arquitetura  
Funções de negócios independentes



Pessoas, Processo e Cultura

Organizado para Valor

A modernização é a refatoração da tecnologia legada combinando infraestrutura, arquitetura e padrões de organização modernos para maximizar a resiliência, a eficiência da engenharia e a agilidade dos negócios.

# O objetivo final



Valor para o cliente



Confiabilidade



Resiliência



Escalabilidade



*Agilidade*

# Principais desafios segundo o State of DevOps Report 2020 apresentado pela Puppet e CircleCI:

Os principais desafios para **implementar entrega contínua (continuous delivery)** reportado por todos os grupos:

**“Cobertura de teste incompleta:** Escrever bons testes que cubram todos os cenários possíveis é quase impossível, especialmente em ambientes complexos onde há um número infinito de comportamentos de usuário, dependências, arquiteturas dinâmicas e muito mais.”

**“Mentalidade organizacional:** Não é nenhuma surpresa que a mentalidade organizacional seja um desafio comum em cada grupo. Ouvimos muito isso em nosso trabalho com empresas, e é a única coisa com a qual os líderes seniores e profissionais estão de acordo.”

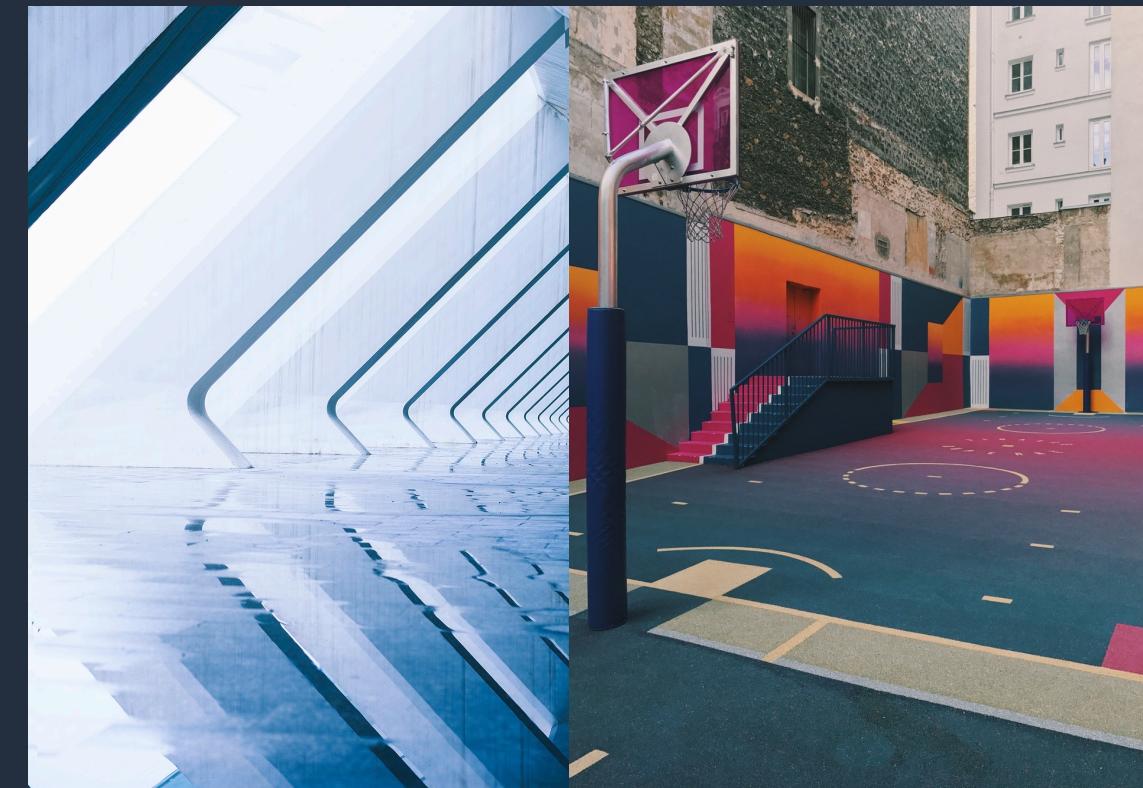
**“Arquitetura de aplicação fortemente acoplada:** A arquitetura de aplicação fortemente acoplada é uma das principais restrições para as equipes de entrega. Atualizar sua aplicação ou serviço requer coordenação com outras equipes, retardando a entrega de cada equipe devido a dependências complexas. **O acoplamento fraco significa que os aplicativos são mais modulares, para que as equipes possam entregar em seu próprio ritmo, usando seus próprios fluxos de trabalho**”

# Diferença entre Arquitetura e Design

Arquitetura de software é processo de converter características de software como flexibilidade, escalabilidade, viabilidade, reutilização e segurança em solução estruturada que atenda às expectativas de negócios.

Exemplos de padrões de Arquitetura de Software:

- ★ Microserviços
- ★ Serverless
- ★ Event-Driven
- ★ Hexagonal

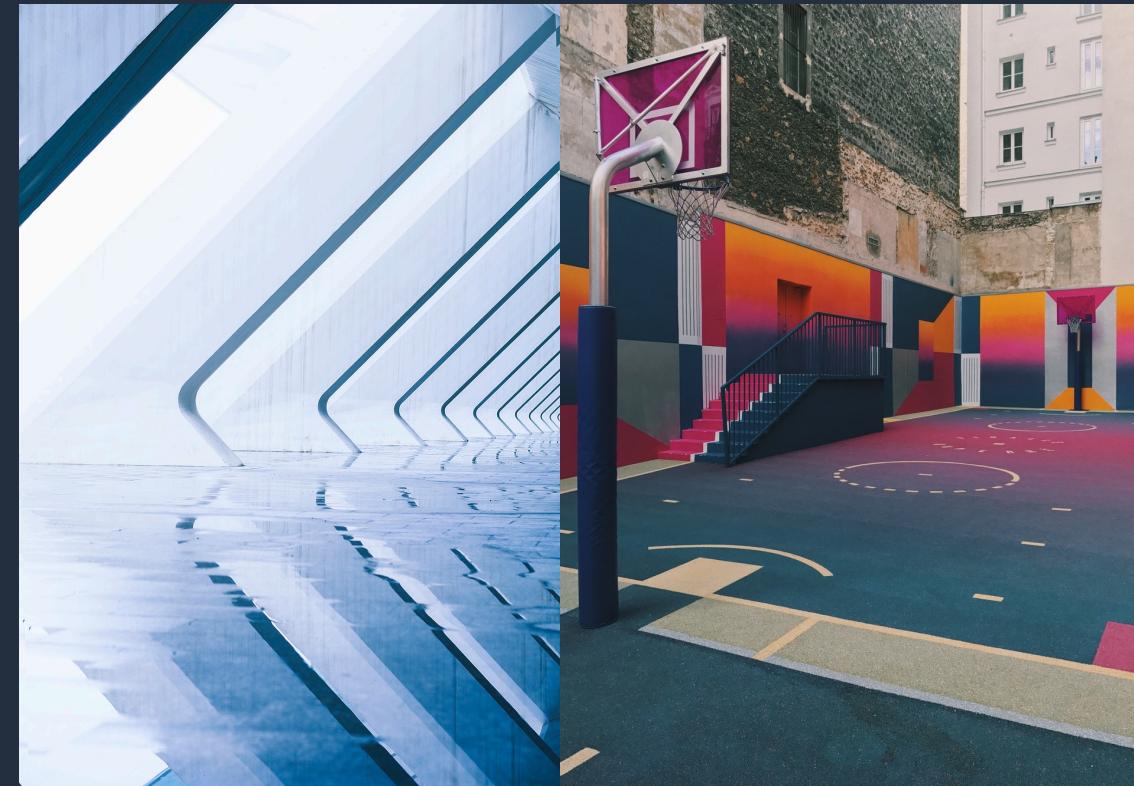


# Diferença entre Arquitetura e Design

O design do software é responsável pelo design no nível de código, como o que cada módulo está fazendo, o escopo das classes e o objetivos das funções.

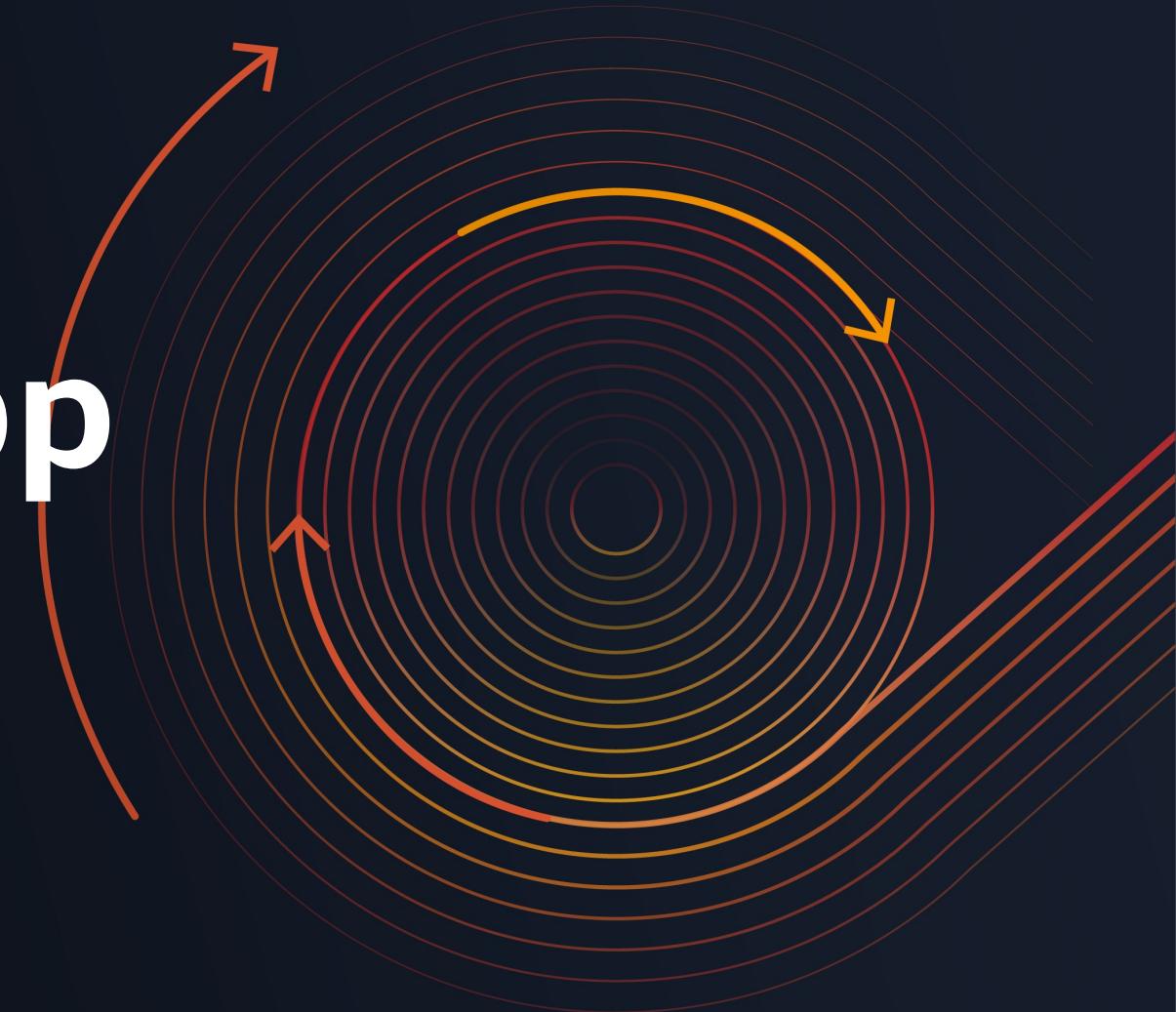
## Exemplos de padrões de Design

- ★ SOLID
- ★ Factory
- ★ Adapter
- ★ Object Calisthenics



# The Twelve-Factor App

Aplicação 12 fatores



Princípios  
Twelve-Factor App



Arquitetura baseada  
em Microserviços



=



Arquitetura de alta  
disponibilidade,  
escalabilidade e  
portabilidade



# THE TWELVE-FACTOR APP

## INTRODUÇÃO

Na era moderna, software é comumente entregue como um serviço: denominados *web apps*, ou *software-como-serviço*. A aplicação doze-fatores é uma metodologia para construir softwares-como-serviço que:

- Usam formatos **declarativos** para automatizar a configuração inicial, minimizar tempo e custo para novos desenvolvedores participarem do projeto;
- Tem um **contrato claro** com o sistema operacional que o suporta, oferecendo **portabilidade máxima** entre ambientes que o executem;
- São adequados para **implantação** em modernas **plataformas em nuvem**, evitando a necessidade por servidores e administração do sistema;
- **Minimizam a divergência** entre desenvolvimento e produção, permitindo a **implantação contínua** para máxima agilidade;
- E podem **escalar** sem significativas mudanças em ferramentas, arquiteturas, ou práticas de desenvolvimento.

A metodologia doze-fatores pode ser aplicada a aplicações escritas em qualquer linguagem de programação, e que utilizem qualquer combinação de serviços de suportes (banco de dados, filas, cache de memória, etc).

# The Twelve-Factor app manifesto

## I. Codebase

One codebase tracked in revision control, many deploys

## II. Dependencies

Explicitly declare and isolate dependencies

## III. Config

Store config in the environment

## IV. Backing services

Treat backing services as attached resources

## V. Build, release, run

Strictly separate build and run stages

## VI. Processes

Execute the app as one or more stateless processes

## VII. Port binding

Export services via port binding

## VIII. Concurrency

Scale out via the process model

## IX. Disposability

Maximize robustness with fast startup and graceful shutdown

## X. Dev/prod parity

Keep development, staging, and production as similar as possible

## XI. Logs

Treat logs as event streams

## XII. Admin processes

Run admin/management tasks as one-off processes

# The Twelve-Factor App

Isolation and dependency management

Codebase

Dependencies

Config

Deployment automation

Build, release, run

Dev/prod parity

Compute and invocation

Concurrency

Processes

Port-binding

Disposability

Application support

Logs

Administrative processes

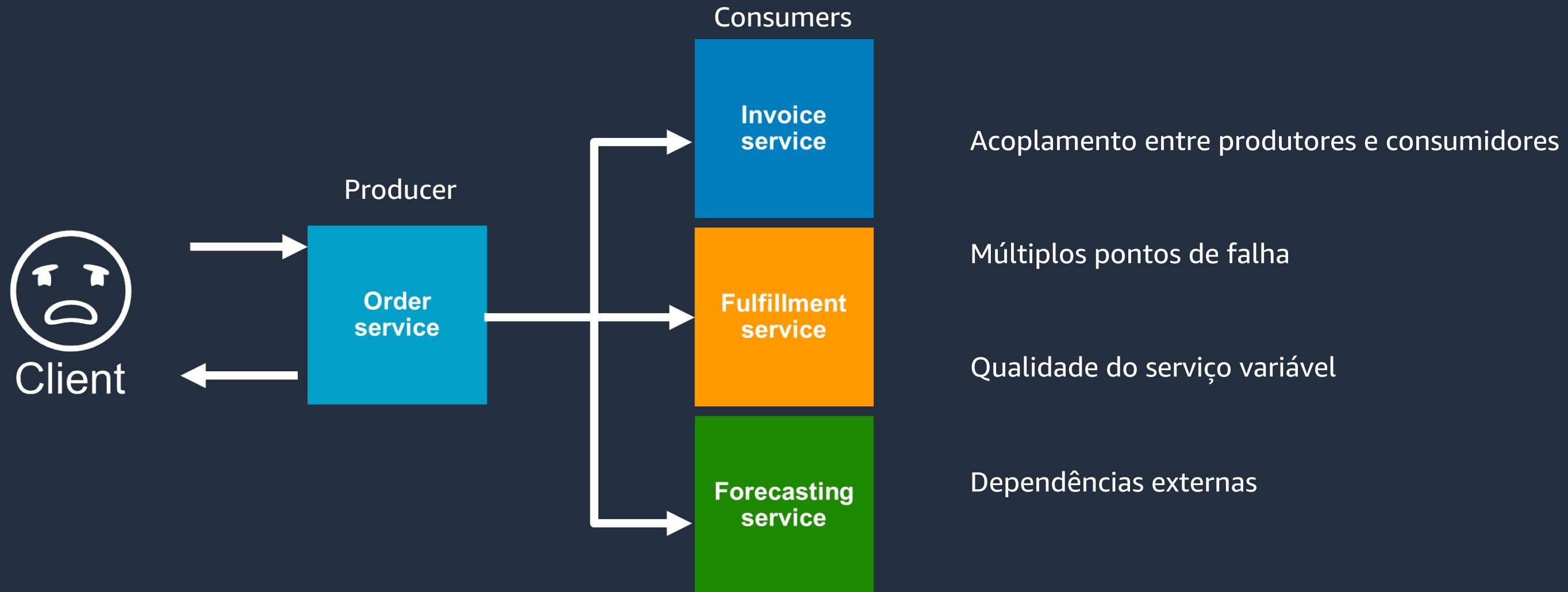
Backing services

# Aplicações orientadas a eventos

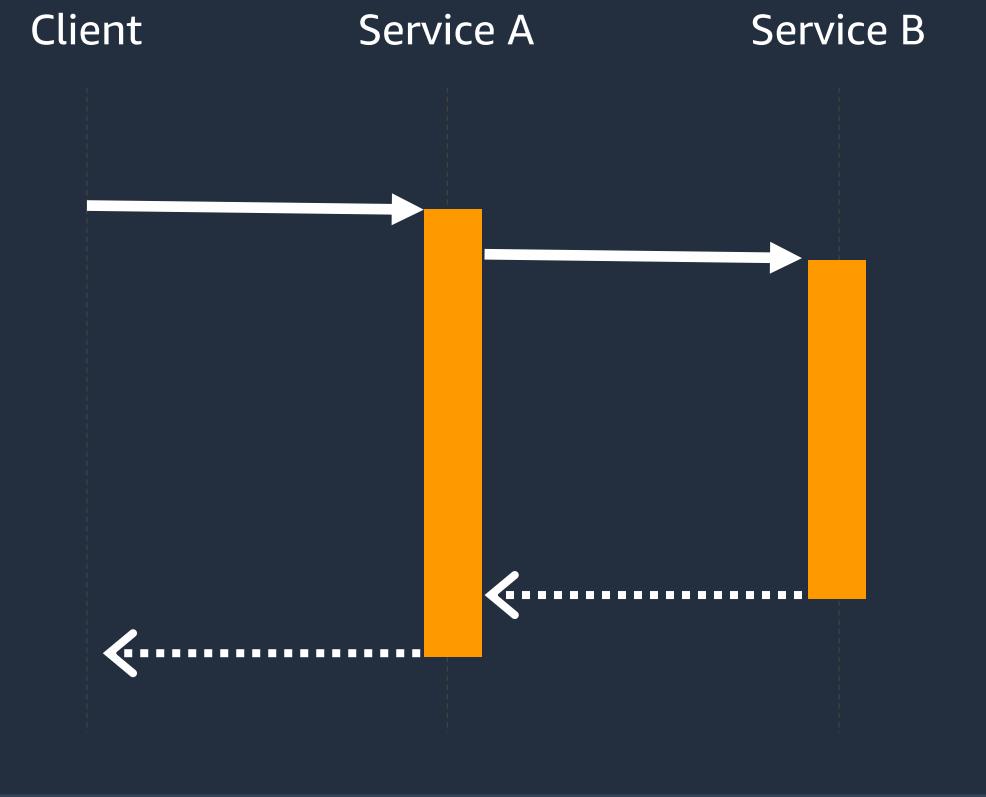
# Aplicações orientadas a eventos

- Usam eventos para acionar e comunicar entre serviços desacoplados
- Um evento é uma alteração no estado ou uma atualização
- As mensagens contêm os detalhes dessa alteração no estado ou atualização (transação / fluxo)
- As arquiteturas orientadas a eventos possuem três componentes-chave: produtores de eventos, roteadores de eventos e consumidores de eventos.

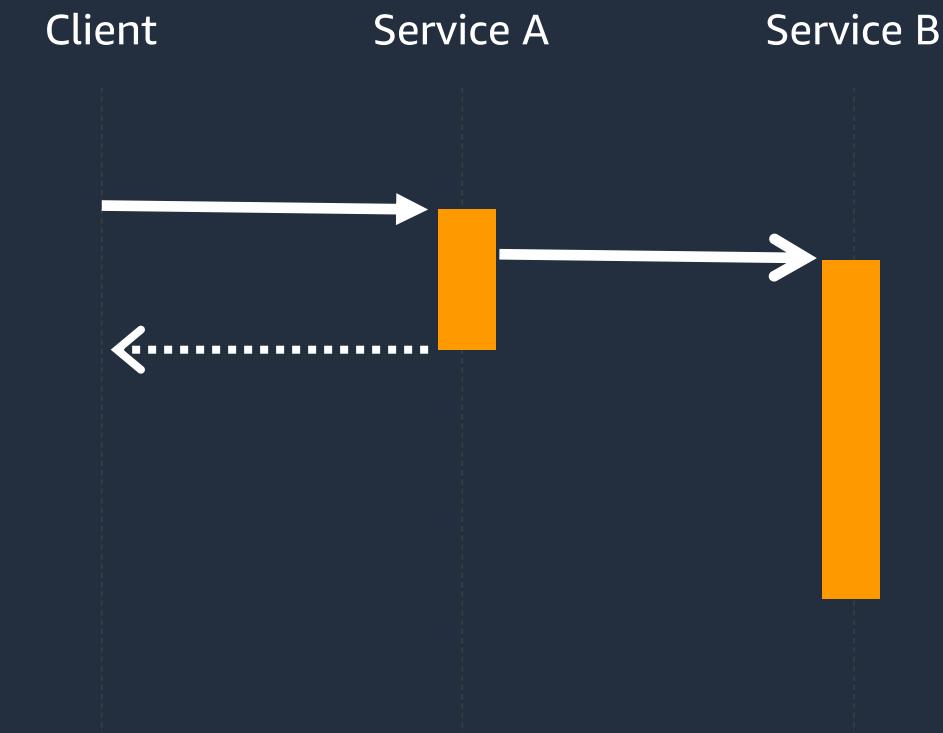
# Desafios em sistemas distribuídos (API síncrona)



# Melhore a capacidade de resposta e reduza as dependências

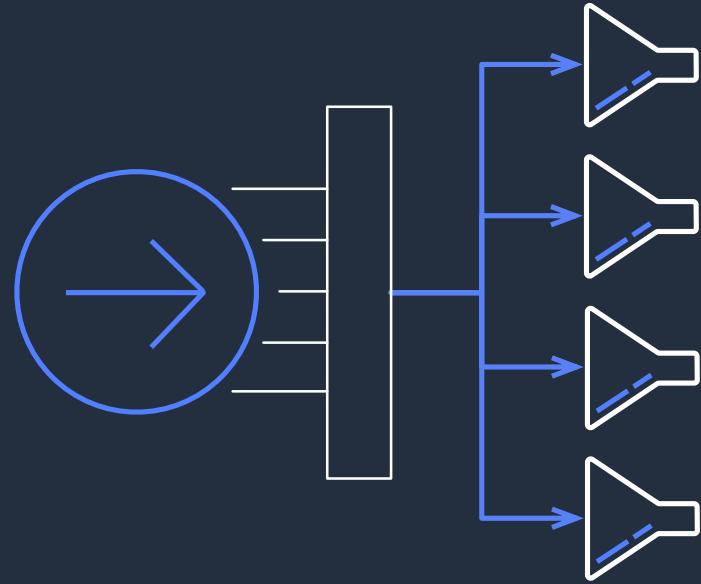


Comandos  
Síncronos

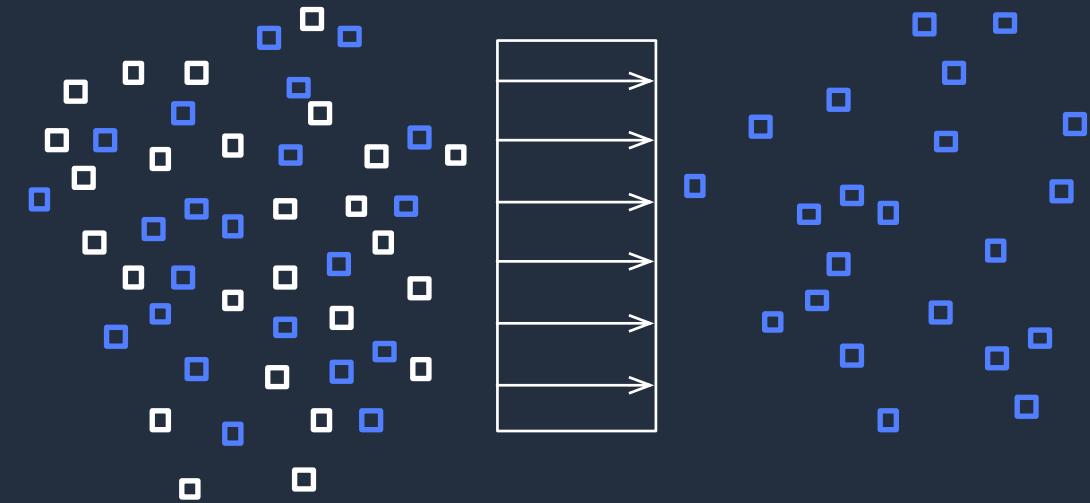


Comandos  
Assíncronos

# Desacople os serviços com roteadores de eventos



Produtores abstratos  
e consumidores



Seleciona e filtra  
eventos

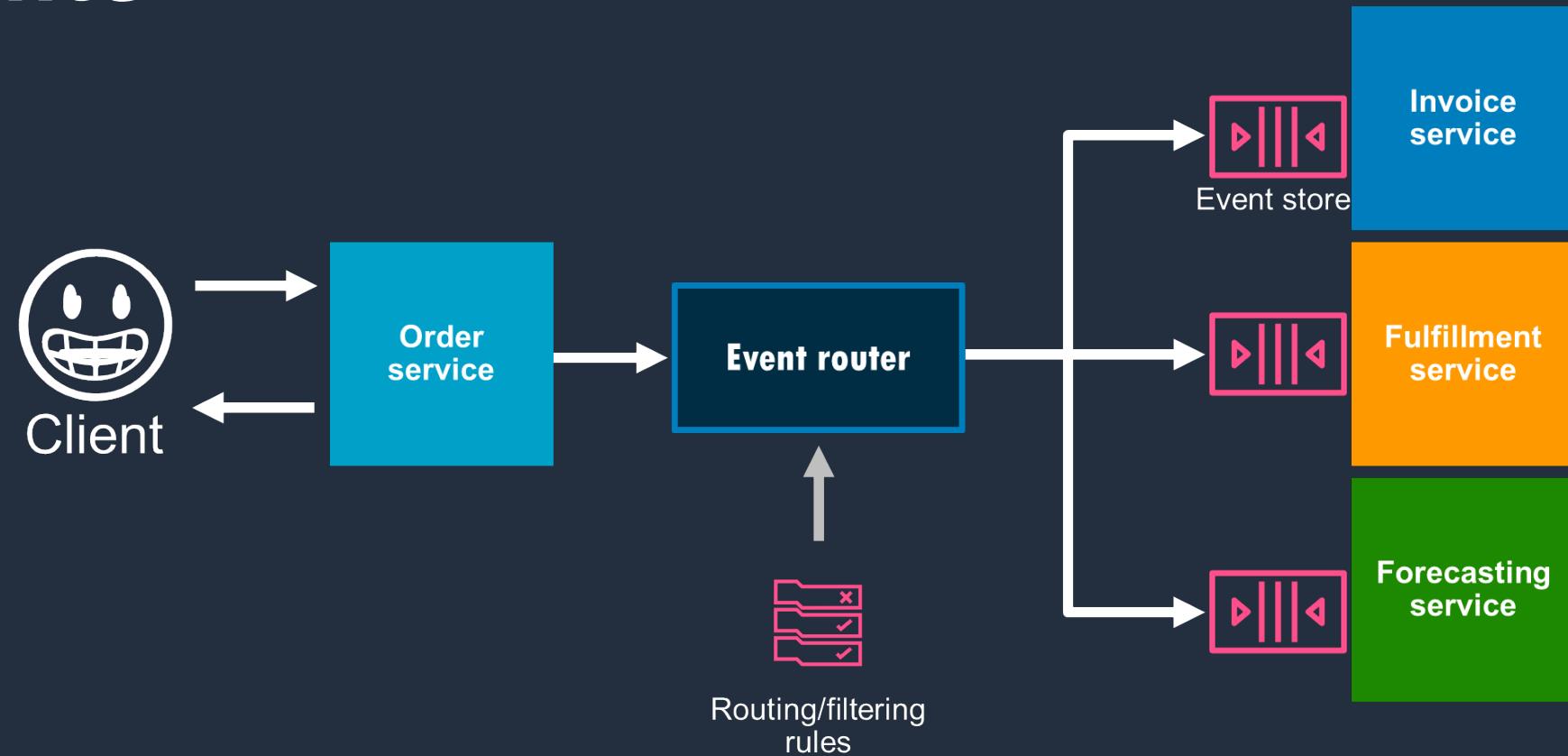
# Melhore a resiliência e a escalabilidade com armazenadores de eventos



---

Retenha as mensagens até que o serviço esteja disponível para processar

# Confiabilidade, resiliência e escalabilidade de forma independente



Amazon VP / Distinguished Engineer  
Messaging, Eventing & Orchestration

# Arquiteturas orientadas a mensagens e eventos em FinServ



## Bancos

*Gestão transacional e de caixa*



## Pagamentos



## Mercados de Capitais

*Distribuição de dados de mercado*



## Seguro

*Pedidos de empréstimo e processamento de reclamações*

Arquiteturas levemente acopladas, escala maior, sistemas mais tolerantes a falhas com menos dependências → **inovação mais rápida**

# Serviços AWS e Padrões de Arquiteturas



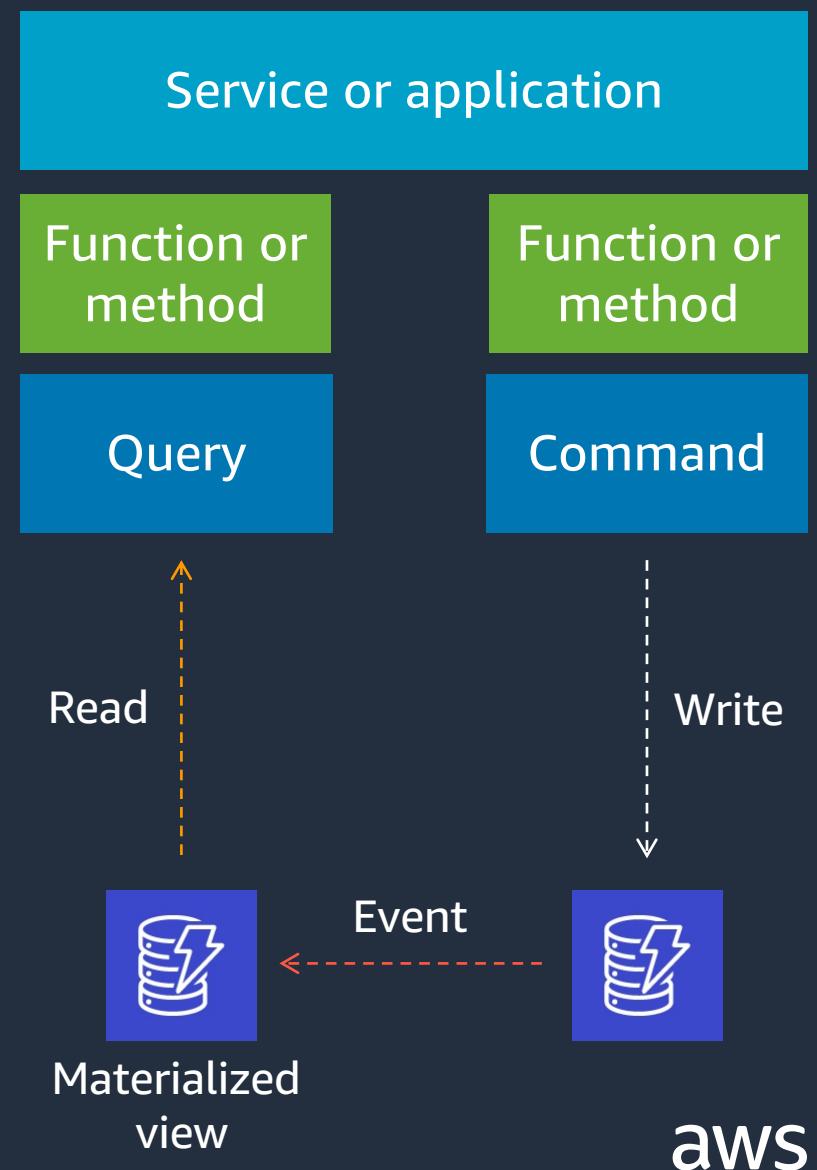
# CQRS

- Um modelo para atualizar as informações e um modelo separado para ler informações.
- Para separar as transações e o estado geral, CQRS + Event Sourcing

## Benefícios:

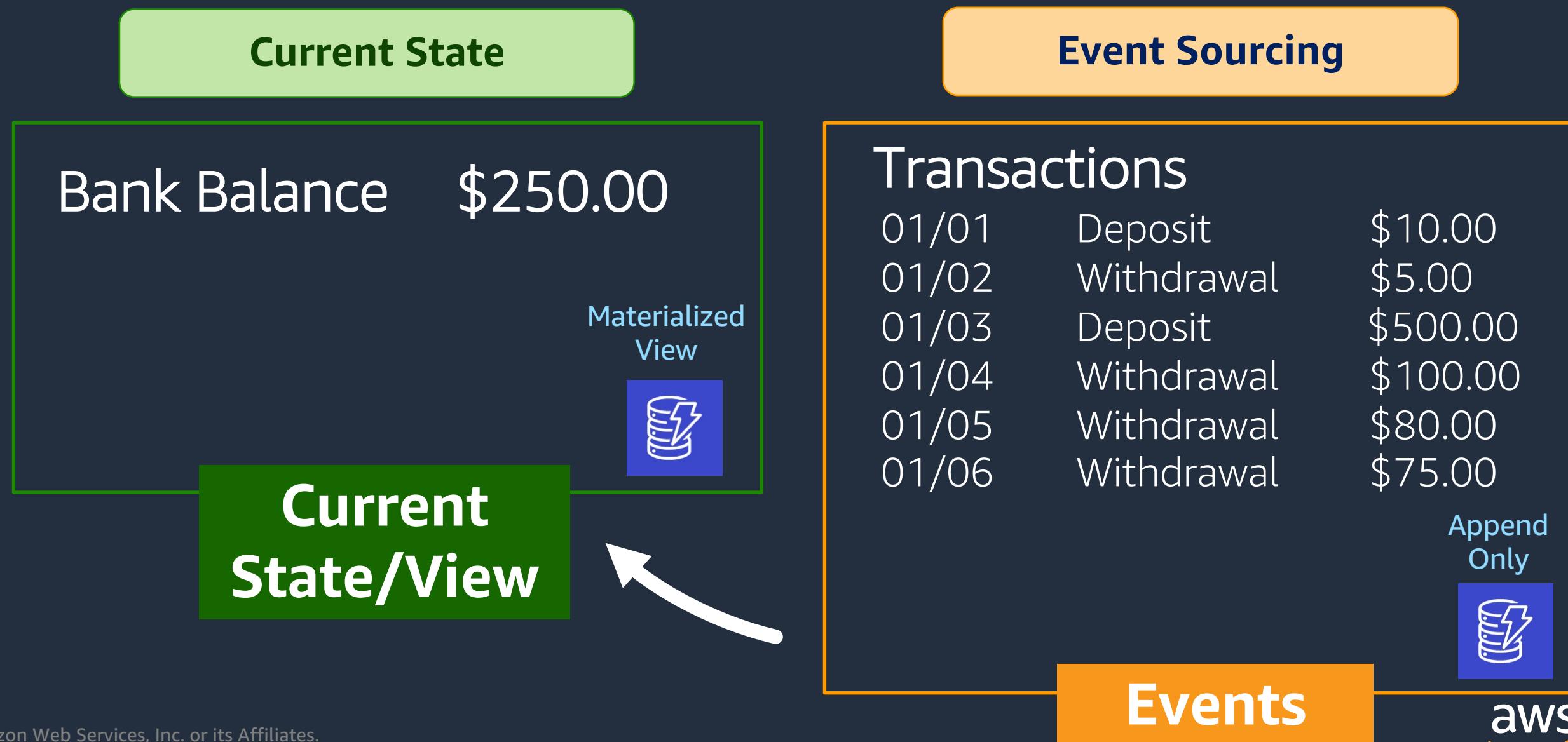
- 🎯 Distribuir e dimensionar leituras e gravações
- 🎯 Aproveite a eventual consistência
- 🎯 Flexibilidade em modelos de leitura e gravação
- 🎯 Promove e apoia a persistência poliglota

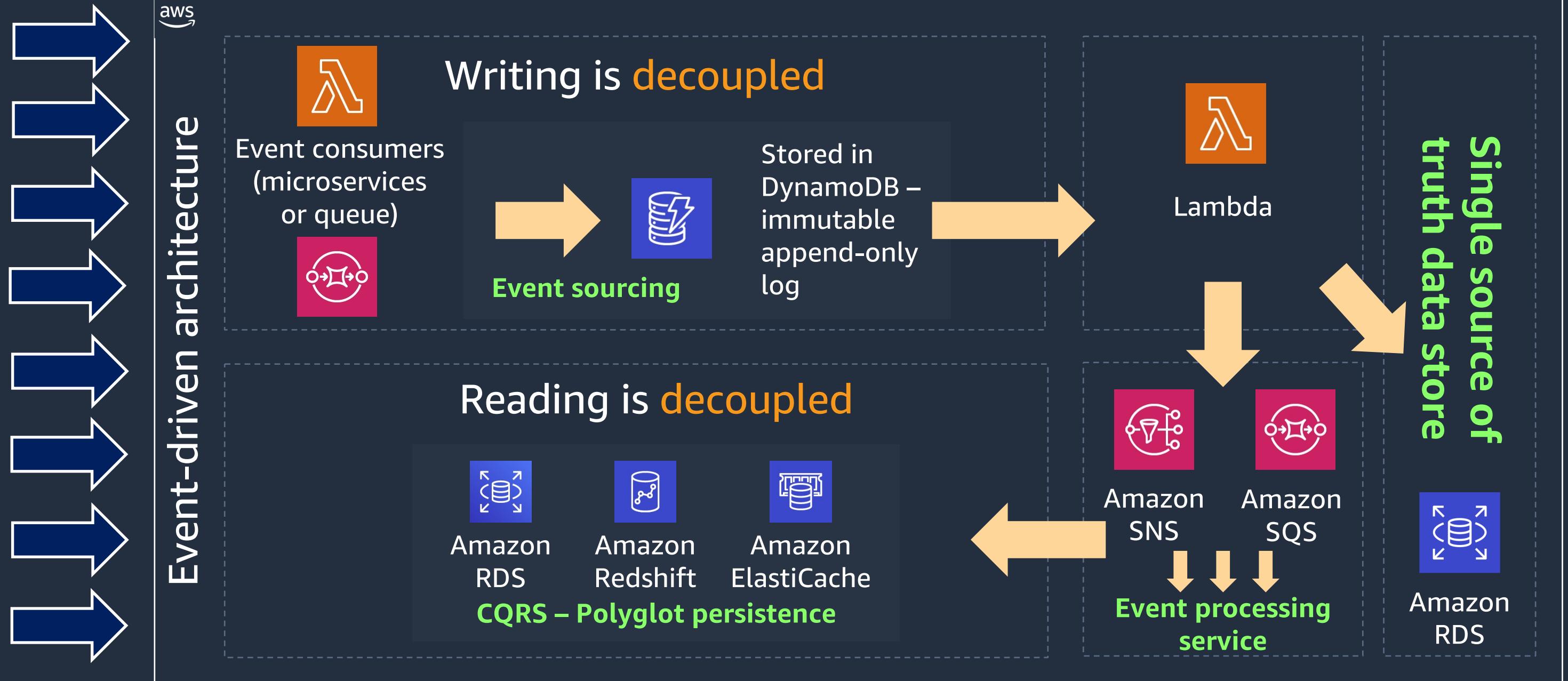
## CQRS



# Event sourcing Database

Capture todas as alterações no estado de um aplicativo como uma sequência de eventos.





# Serviços de eventos e mensagens da AWS

	Event Store		Event Router	
	Queues	Streams	Topics	Bus
AWS Native				
Managed Open Source				

# Amazon Simple Queue Service (SQS)



## O QUE É

Serviço de enfileiramento de mensagens confiável, simples, flexível e totalmente gerenciado para troca contínua de mensagens em qualquer volume e lugar

(Standard e FIFO)



Origem do evento

## CASO DE USO

Construir microsserviços, sistemas distribuídos, e aplicações serverless de forma desacoplada e altamente escalável na nuvem

## CAPACIDADES

Escalabilidade quase infinita sem necessidade de pré-provisionamento

SQS é #1 event-trigger para Lambda

25B mensagens por hora



# Amazon Simple Notification Service (SNS)

## O QUE É

Serviço de publish/subscribe de mensagens e push notification para dispositivos móveis simples, flexível e totalmente gerenciado com alta taxa de transferência e confiabilidade de entrega. **Novo! FIFO**

## CASO DE USO

Notificar vários aplicações inscritas  
Replicar dados entre regiões  
Invocar várias etapas nas cargas de trabalho  
Processamento Paralelo  
Gatilho serverless para ações

## CAPACIDADES

Entrega **altamente confiável** de qualquer volume de mensagens para qualquer número de destinatários através de vários protocolos

Targets: Lambda, SQS, HTTP, SMS, Mobile Push, E-mail

SNS é um event-trigger para Lambda



Origem do evento

# Amazon EventBridge



## O QUE É

Event bus simples, flexível, totalmente gerenciado, com modelo de cobrança pay as you go, que facilita a ingestão e o processamento de dados oriundos de **Serviços da AWS**, suas aplicações, e Aplicações SaaS.



Origem do evento

## CASO DE USO

Reducir a complexidade de ter que escrever integrações ponto a ponto entre serviços.

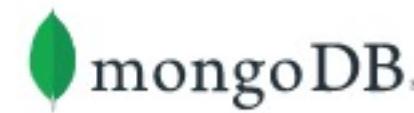
**Disparar ações em mensagens SaaS, executar fluxos de trabalho, aplicar inteligência, auditar e analisar, e sincronizar dados.**

## CAPACIDADES

17 Targets incluindo Lambda, SQS, SNS, Kinesis e Firehose

Schema Registry **stores** uma coleção de esquemas que permite aos desenvolvedores **PESQUISAR/ENCONTRAR/RASTREAR** diferentes esquemas que são usados pelas aplicações

# Mais de 24 integrações SaaS no Amazon EventBridge...



# Event Routers: Eventbridge e/ou SNS

## EVENTBRIDGE

- Reagindo a eventos de aplicações SaaS, serviços da AWS (115) ou aplicações de clientes.
- Schema predefinido: crie regras que são aplicadas em todo o body do evento para filtrar antes de enviar para os consumidores

## SNS

- Publicação de eventos com alta taxa de transferência e baixa latência por outras aplicações, microserviços ou serviços da AWS.
- Fanout muito alto (milhares ou milhões de endpoints)

# Mensagem vs Streaming



# Processamento de mensagens vs processamento de fluxo

## Processamento de mensagens



- A **mensagem** é a unidade de trabalho
- Computação/processamento por mensagem
- A ocorrência da mensagem pode variar
- **Funcionalidade DLQ integrada**
- A mensagem é excluída após o consumo
- **Não há necessidade de rastrear a posição**

## Processamento de fluxo



- O **fluxo de mensagens** é a unidade de trabalho
- Computação complexa em muitas mensagens
- Fluxo constante de mensagens
- **Não há funcionalidade DLQ integrada**
- As mensagens estão disponíveis após o consumo até serem expiradas
- Cada **cliente precisa saber/rastrear sua posição atual no fluxo**

# Mensagens vs Streaming

## Exemplo de mensagens

Transações

Saldo da Conta: \$1000

Débito \$100

Crédito \$200

## Exemplo de streaming

Trilha de Auditoria Bancária

Análises de negociação

Análises de Seguros

Excluindo o custo de invocações Lambda para processar as mensagens, aqui estão algumas projeções de custo para usar SNS vs Kinesis Streams vs DynamoDB Streams como corretor. Estou supondo que a taxa de transferência é consistente e que cada mensagem tem 1 KB de tamanho.

Fonte: <https://theburningmonk.com/2018/04/what-is-the-best-event-source-for-doing-pub-sub-with-aws-lambda/>

monthly cost at 1 msg/s



Kinesis Streams

1 x 60s x 60m x 24hr x 30days  
@ \$0.014 per mil  
+  
24hrs x 30days  
@ \$0.015 per hr

\$10.836



SNS

1 x 60s x 60m x 24hr x 30days  
@ \$0.5 per mil

\$1.296



DynamoDB Streams

1 Write Capacity Unit  
@ \$0.47 per unit

\$0.47

monthly cost at 1,000 msg/s



Kinesis Streams

1k x 60s x 60m x 24hr x 30days  
@ \$0.014 per mil  
+  
24hrs x 30days  
@ \$0.015 per hr

\$47.088



SNS

1k x 60s x 60m x 24hr x 30days  
@ \$0.5 per mil

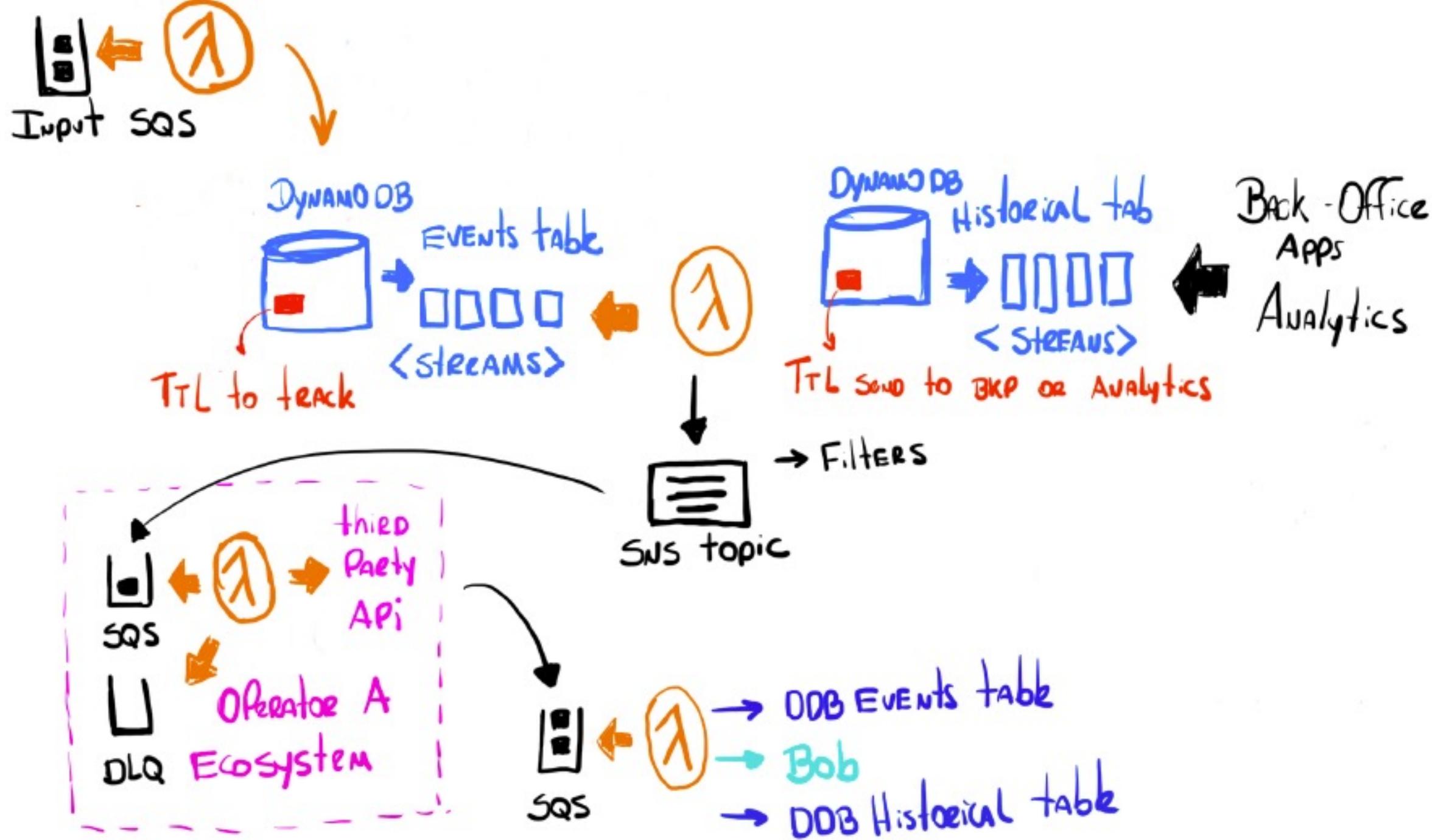
\$1296.00



DynamoDB Streams

1k Write Capacity Unit  
@ \$0.47 per unit

\$470.00



Cross Border Event-Driven tracking Architecture



**Deixe seu Feedback! Sua opinião é muito  
Importante para nós!**



**Obrigado!**

**Repo Git com Apresentação +  
Materiais Adicionais**



Peterson Larentis  
Enterprise Solutions Architect, AWS  
<https://www.linkedin.com/in/peterson-larentis/>