Microserviços **serverless**:
diferentes designs, mesmas dores

SREday 2025

Felipe **KiKo**

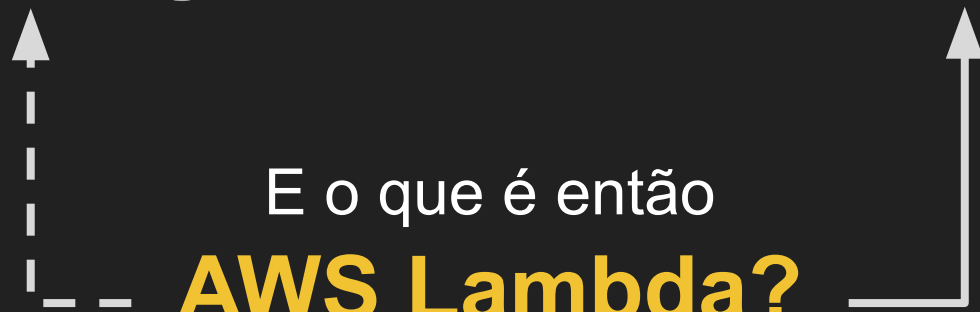
Mas, antes de tudo…

O que é

Microserviços?

O que é

Serveless?



E o que é então

AWS Lambda?

ECS Fargate?

Qual é o problema?

Uso “correto” de **FaaS** / **CaaS**

Microserviços

1. Definir o contrato da API;
2. Modularidade e Integração;
3. Síncrona ou assíncrona;
4. Stack;
5. …

Até chegar na pergunta…qual melhor **arquitetura**?

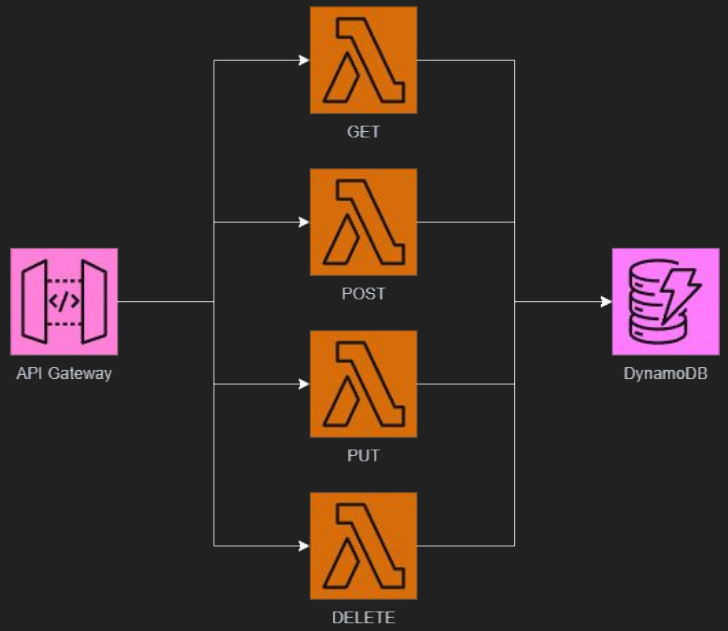
Abordagens

As mais comuns

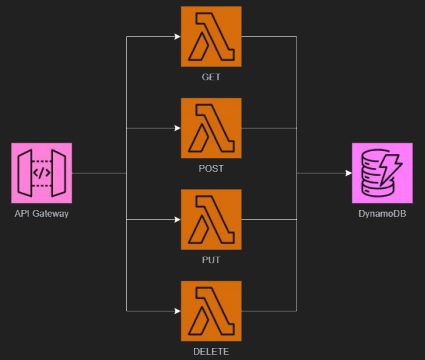
- **Single-Responsibility** Lambda Functions
- Lambda-lith (**One Single** Lambda Function)
- …
- **Read and Write** Functions

Single responsibility Lambda functions

Lida somente com uma **única responsabilidade**



Single responsibility Lambda functions



Vantagens

- Arquitetura por cada função (stack)
- Custo: Cada uma com memória e timeout
- Testes isolados
- Problemas separados por função (-bugs)
- Deploys independentes
- Debug fácil: Métricas e logs individuais
- Cold start: Execução mais rápida

Desvantagens

- Cold start...sim você terá vários!
- Manutenção complexa
- Código compartilhado (Lambda layers?)
- Ambiente de desenvolvimento
- Atualização de runtime / issues de segurança
- Muuuuuuitos containers
- Observabilidade (se não for bem feita!)

Lambda-lith: One single Lambda function

~~*Um anel para todos governar, ops…*~~

Um **Backend** para todos os **verbos** e / ou **métodos**

Lambda-lith: One single Lambda function

Vantagens

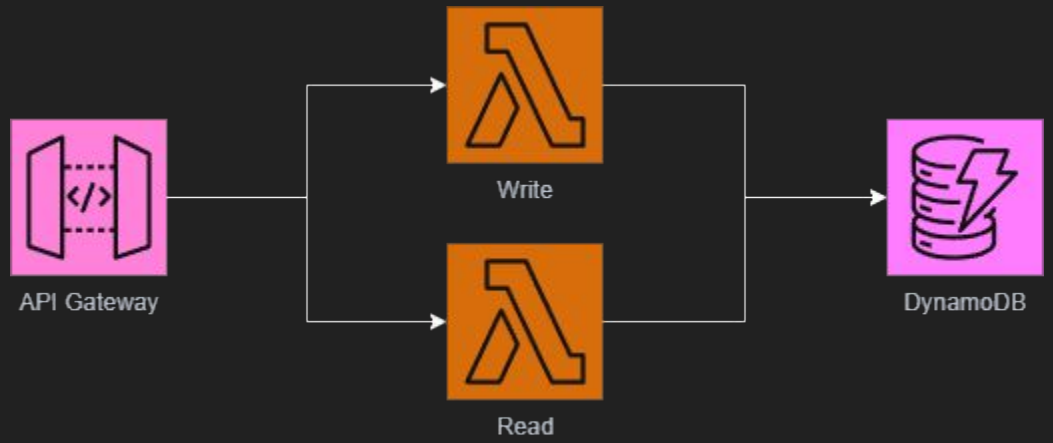
- Manutenção mais fácil
- Modularização em nível de código
- Cold start reduzido (mas depende)
- Observabilidade / APM

Desvantagens

- Cold start com alto tempo para start
- Tamanho do pacote
- Mesma configuração para tudo (MEM e TIMEOUT)

Read and write functions

E se…tentar **juntar** um pouco das outras **duas abordagens**?



Read and write functions

Geralmente…

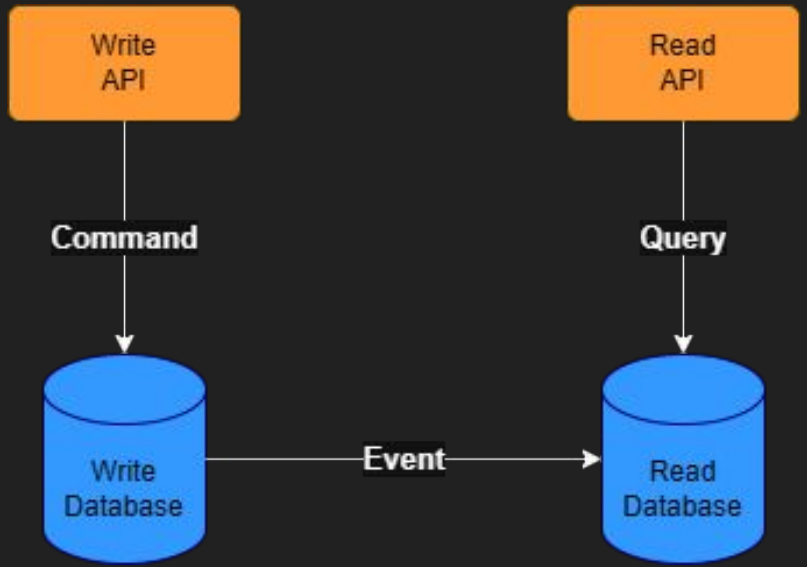
- Uma aplicação **recebe** mais leitura do que escrita…
- A maioria da transações de escrita é mais **complexa** que leitura…
- Nosso banco de dados tem mais **réplica** de leitura do que escrita…
- Do lado de leitura, podemos usar **cache** por exemplo
- Conseguimos **otimizar** as operações de leitura e escrita…

E com isso…

- **Parâmetros diferentes** de memória, timeout, etc
- E usar o **CQRS**

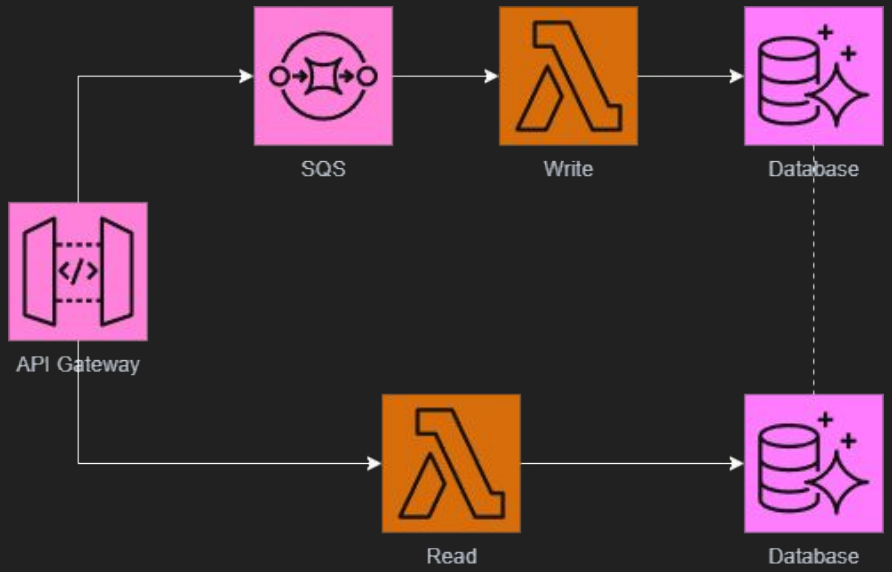
CQRS Patern

Padrão arquitetural usado para **separar** as operações de **leitura** e **escrita**



Read and write functions

E se…ainda colocarmos um **bulk operations**?



Tá…e no final, qual é o melhor?

DEPENDE!

Todas tem trade-offs!

Entender seu negócio

O que você quer **priorizar**?

Manutenção, performance, custo, times, etc..

Alguns critérios de decisão

1. Qual o volume de chamadas reads vs writes?
2. Qual a tolerância à latência / cold starts?
3. Quanto é necessário e crítico a consistência?
4. Qual o seu time de desenvolvimento VS governança de funções?
5. Existe dependências comuns entre funções? Compartilhamento de código?
6. O que fica ou é a complexidade do seu deploy?
7. Evolução esperada (vai crescer muito? variabilidade de carga)?
8. Custo é algo decisório?
9. ...

Ah, **misturar** não é proibido ein!

E se depois precisar mudar...**tá tudo bem**!

Dúvidas?

Obrigado!