MBA EM **ENGENHARIA DE SOFTWARE**

Serverless I

Prof. José Maria Cesário Jr.

MBA USP ESALQ

A responsabilidade pela idoneidade, originalidade
e licitude dos conteúdos didáticos apresentados é do professor.

MBA USP ESALQ

José Maria Cesário Júnior

- José Maria Cesário Júnior possui mais de 25 anos de experiência na área de TI e atua como Arquiteto de Soluções para Parceiros na AWS Brasil.
- Possui experiência em soluções de Cloud Computing, Indústria 4.0 e IoT Industrial
- Atua desde 2007 como professor universitário de Graduação e Pós Graduação.
- Analista de Sistemas, graduado pela Universidade Metodista de Piracicaba, com especialização em Tecnologia
- MBA em Gestão de Projetos pela FGV
- Especialista em Automação e Controle de Processos Industriais e Agroindustriais pela FEAGRI, UNICAMP
- Mestre em Tecnologia e Inovação pela UNICAMP

https://www.linkedin.com/in/josemariacesariojunior/

AGENDA

Data	Conteúdo
18/09/2025	• Introdução e definições oficiais • Evolução histórica • Padrões importantes e arquiteturas • Características técnicas detalhadas • Comparativo multi-cloud (AWS, GCP, Azure) • Ferramentas e frameworks oficiais • Aplicações práticas • Exercícios

ÍCONES

DEFINIÇÃO OU PONTO DE ATENÇÃO

EXERCÍCIO OU ATIVIDADE PRÁTICA

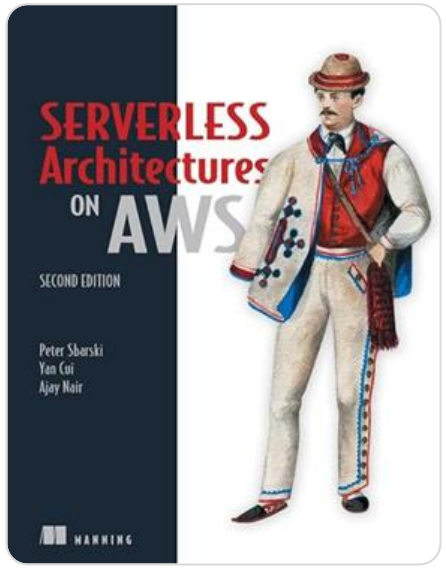
ACESSO AO AMBIENTE NA NUVEM

! IMPORTANTE

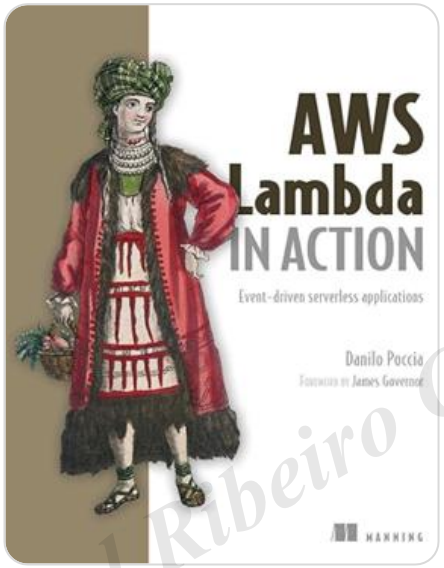
O laboratório prático utilizará recursos da AWS (Amazon Web Services), Microsoft Azure ou GCP (Google Cloud Platform) que podem gerar custos reais. Pontos essenciais:

1. A participação nesta atividade é OPCIONAL
2. Cada aluno é responsável pelo uso de sua própria conta no provedor de serviços Cloud
3. Os provedores de serviços Cloud oferecem um nível gratuito (Free Tier) com limites específicos
4. Custos além do Free Tier serão de responsabilidade individual do usuário
5. Recomendamos verificar os limites do Free Tier antes de iniciar as atividades práticas em https://aws.amazon.com/pt/free/ , https://azure.microsoft.com/pt-br/pricing/free-services, https://cloud.google.com/free/docs/free-cloud-features

LIVROS

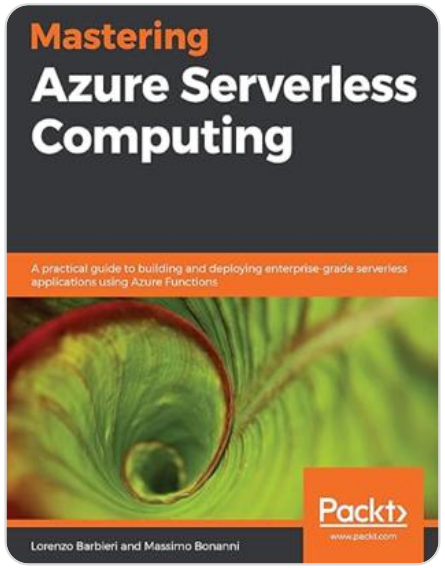


Serverless Architectures on AWS
2ª edição



AWS Lambda in Action: Event-driven serverless applications
1a edição

Building Serverless Applications with Google Cloud Run
1ª Edição



Mastering Azure Serverless Computing
1ª Edição

Referências e Fontes

- Esse material tomou como referência a documentação oficial e aberta dos principais provedores e fabricantes de serviços de computação em nuvem pública do mercado
- AWS Amazon Web Services
- GCP Google Cloud Platform
- Microsoft Azure

Referências Fabricantes

https://docs.aws.amazon.com/whitepapers/latest/serverless-multi-tier-architectures-api-gateway-lambda/introduction.html

https://docs.aws.amazon.com/wellarchitected/latest/serverless-applications-lens/welcome.html

https://serverlessland.com/

Referências Fabricantes

Google Cloud

https://cloud.google.com/architecture/blueprints/serverless-blueprint

https://cloud.google.com/architecture/framework

https://cloud.google.com/serverless?hl=en

Referências Fabricantes

https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-serverless-computing

https://azure.microsoft.com/en-us/solutions/serverless

https://azure.microsoft.com/en-us/products/functions/

FUNDAMENTOS

PARTE 1

O que é Serverless

Serverless computing

- Modelo de desenvolvimento e execução de aplicações que permite aos desenvolvedores criar e executar código sem provisionar ou gerenciar servidores.
- O provedor cloud gerencia dinamicamente a alocação de recursos de máquina, cobrando baseado na quantidade real de recursos consumidos.

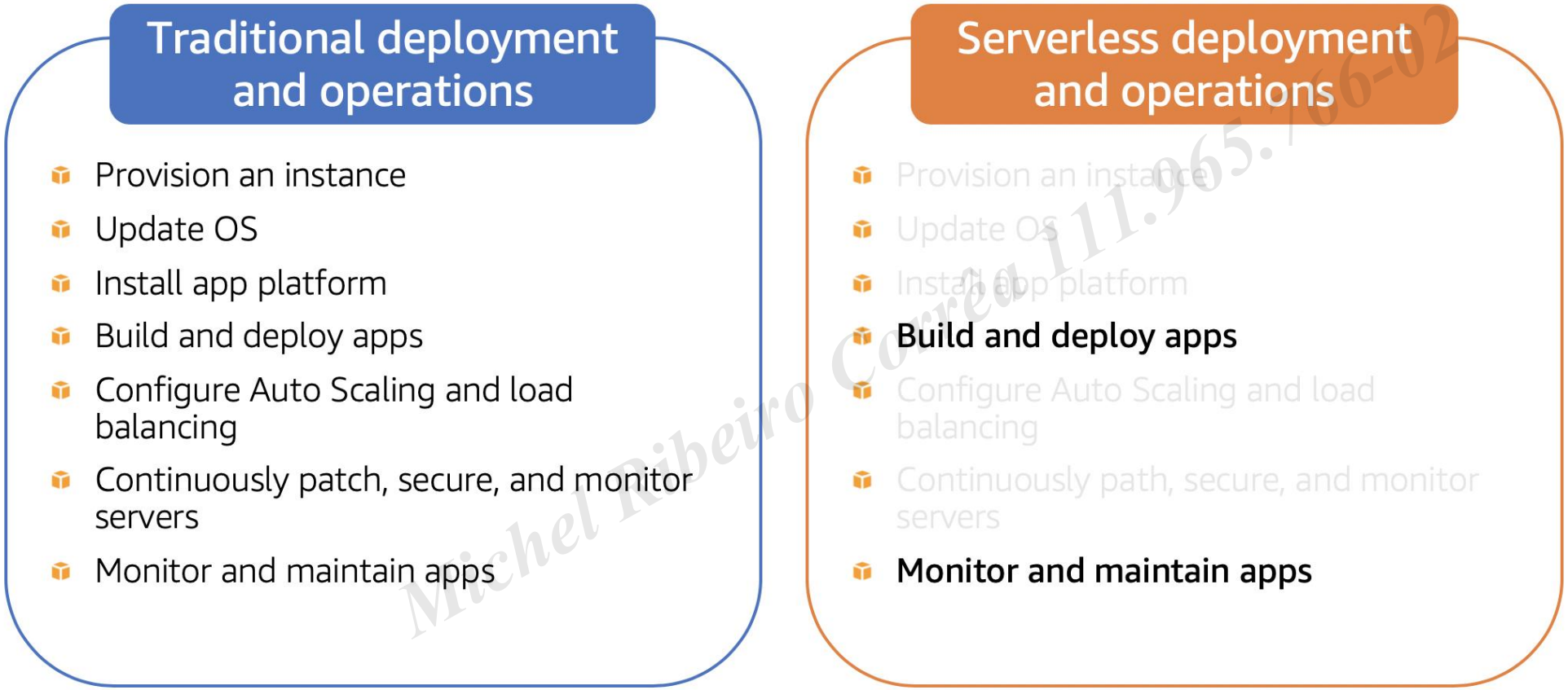
Características fundamentais

- No server management: abstração da infraestrutura
- Flexible scaling: escalabilidade automática de zero a milhares de instâncias
- Pay-per-execution: cobrança por execução, não por capacidade reservada
- Built-in availability: alta disponibilidade e tolerância a falhas nativas

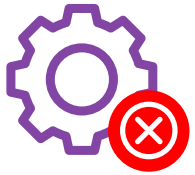
Filosofia serverless framework

- "Build applications that demand attention, not maintenance"
- Construir aplicações que demandem atenção, não manutenção.

O que é Serverless



O que é Serverless



Sem provisionamento ou gerenciamento de servidores

Escalabilidade Automática

Pagamento baseado na quantidade de recursos consumidos

Altamente Disponível e seguro

Evolução Histórica

2014

NASCIMENTO DO AWS LAMBDA

- Introdução do conceito Function as a Service (FaaS)
- Auto-scaling automático e custo zero quando idle
- Mudança paradigmática: de "serverful" para "serverless"

2015

CRIAÇÃO DO SERVERLESS FRAMEWORK

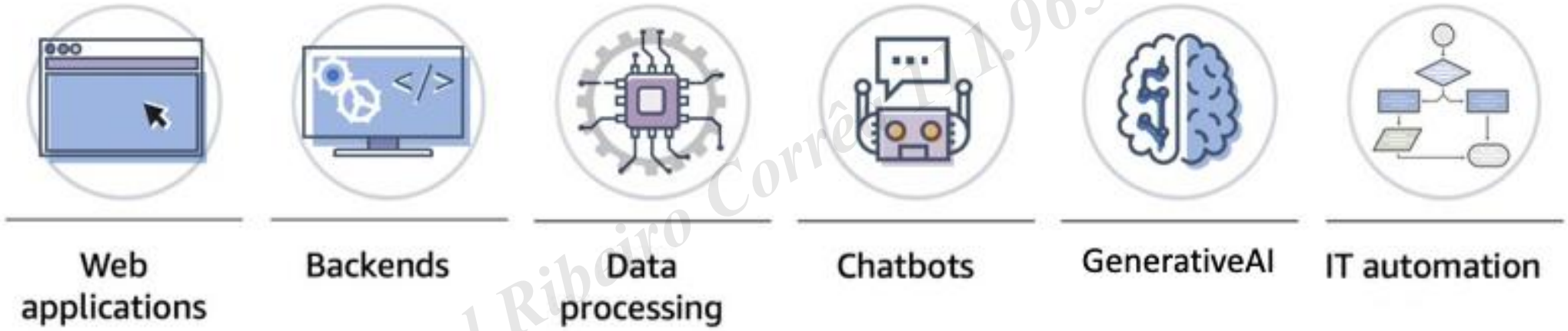
- Streamline do deployment de casos de uso em AWS Lambda
- Introdução do conceito de "serverless architectures"
- Integração de AWS Lambda com infraestrutura cloud

2016 - Presente

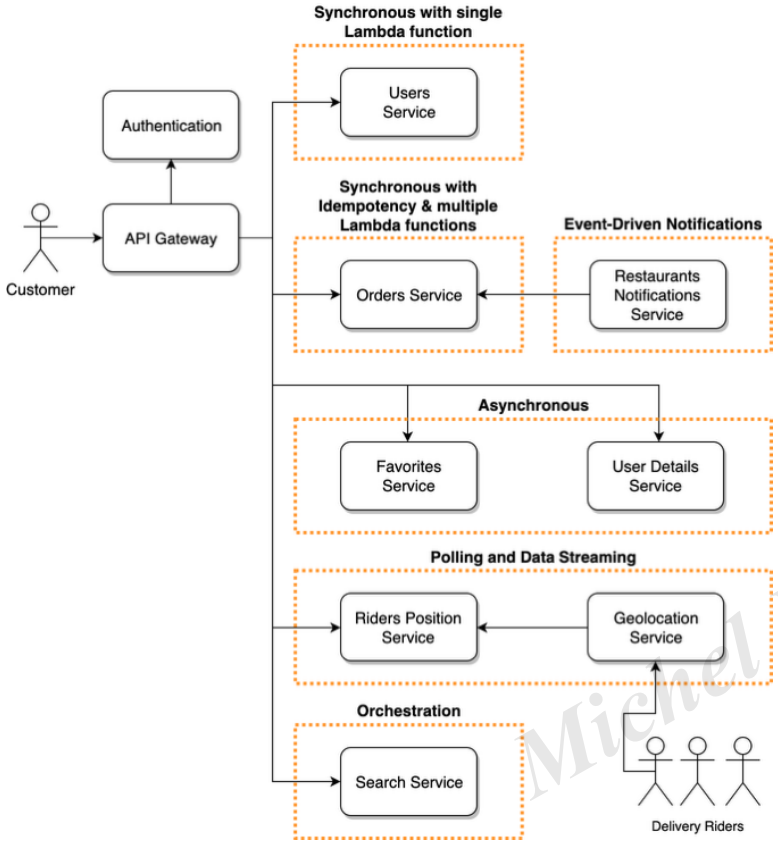
EXPANSÃO MULTI-CLOUD

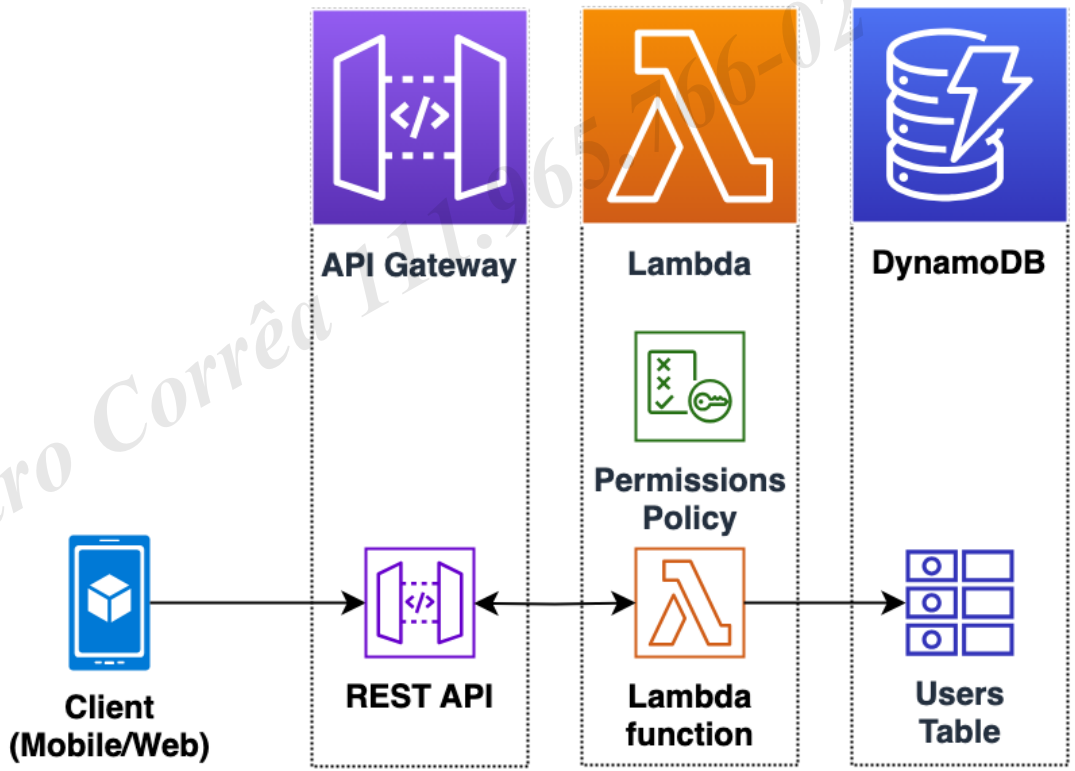
- Google Cloud Functions (2016)
- Azure Functions (2016)
- Transição para modelo SaaS
- Suporte empresarial e governança
- Integração com CI/CD e DevOps

Casos de usos comuns de Aplicações Serverless



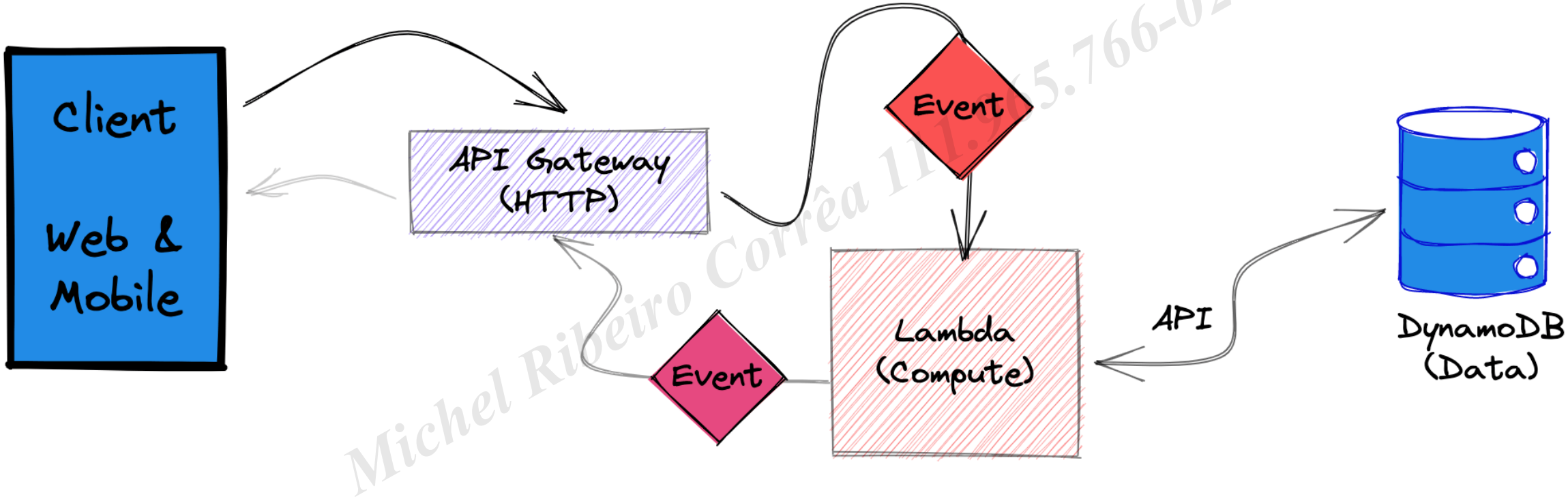
Arquitetura Serverless





Fonte: https://catalog.workshops.aws/serverless-patterns/en-US/business-scenario

Event-Driven Architecture



Fonte: https://catalog.workshops.aws/serverless-patterns/en-US/module1/concepts

Event-Driven Architecture

A arquitetura Serverless oferece flexibilidade de desenvolvimento, escalabilidade e a capacidade de expansão rápida. A chave para esses benefícios é uma arquitetura fracamente acoplada.

Soluções Serverless são baseadas em arquitetura orientada a eventos ou EDA. Em arquiteturas orientadas a eventos, os serviços enviam e recebem eventos que representam mudanças de estado ou uma atualização.

1. O API Gateway serve como ponto de entrada para o microsserviços. Semelhante aos roteadores de URL de aplicativos web tradicionais, o API Gateway recebe a solicitação de entrada para o endpoint /users e a converte em um evento. Esse evento é roteado para uma função Lambda.
2. O serviço Lambda gerencia a função e após inicializar um ambiente de execução, o serviço Lambda invoca a função para manipular o evento de entrada e, possivelmente, muitos outros eventos durante o ciclo de vida da função.
3. A função se conecta a uma tabela de banco de dados Serverless, recupera dados e os agrupa em um evento de resposta JSON.
4. Em vez de enviar a resposta diretamente ao cliente, o evento de resposta é enviado de volta ao API Gateway. O API Gateway então encaminha o corpo do evento ao cliente que fez a chamada para concluir o ciclo de solicitação/resposta.

Event-Driven Architecture

PADRÕES DE COMUNICAÇÃO

- Request/Response: Comunicação síncrona
- Fire-and-forget: Comunicação assíncrona
- Publish/Subscribe: Desacoplamento via tópicos
- Event streaming: Processamento contínuo

FaaS: Function as a Service

- FaaS (Function as a Service) e Serverless não são equivalentes, embora sejam frequentemente usados de forma intercambiável
- **Serverless** é um modelo de computação abrangente, que considera padrões de arquitetura, serviços e ferramentas
- **FaaS é um serviço ou ferramenta, sendo** uma forma específica de implementar o serverless, focando na execução de código em funções pequenas e acionadas por eventos
- Em contraste, PaaS (Platform as a Service) hospeda aplicações completas, com distintos níveis de abstração da infraestrutura

AWS Lambda

CARACTERÍSTICAS TÉCNICAS

- Runtimes suportados: Node.js, Python, Java, Go, C#, Ruby, PowerShell
- Memory allocation: 128MB - 10GB (incrementos de 1MB)
- Execution timeout: 15 minutos máximo
- Package size: 50MB (zipped), 250MB (unzipped)
- Concurrent executions: 1000 (default), configurável até 100,000

EVENT SOURCES PRINCIPAIS

- API Gateway: REST e HTTP APIs
- S3: Object events (create, delete, modify)
- DynamoDB: Stream events
- Kinesis: Data streams e Analytics
- SQS: Queue messages
- SNS: Topic notifications
- EventBridge: Custom events
- CloudWatch: Scheduled events

Azure Functions

CARACTERÍSTICAS TÉCNICAS

- Runtimes suportados: C#, JavaScript, F#, Java, PowerShell, Python, TS
- Memory allocation: 128MB - 1.5GB
- Execution timeout: 5 minutos (Consumption), 30 minutos (Premium)
- Package size: 100MB (compressed)
- Concurrent executions: 200 (Consumption), configurável (Premium)

EVENT SOURCES PRINCIPAIS

- HTTP triggers: REST API endpoints
- Timer triggers: Scheduled execution
- Blob storage: File events
- Queue storage: Message events
- Event Grid: Custom events
- Service Bus: Enterprise messaging
- Cosmos DB: Database events

Fonte: Azure Functions Developer Guide (2023)

Google Cloud Functions

CARACTERÍSTICAS TÉCNICAS

- Runtimes suportados: Node.js, Python, Go, Java, .NET, Ruby, PHP
- Memory allocation: 128MB - 8GB
- Execution timeout: 60 minutos (HTTP), 540 segundos (event-driven)
- Package size: 100MB (compressed), 500MB (uncompressed)
- Concurrent executions: 1000 (default), configurável

EVENT SOURCES PRINCIPAIS

- HTTP triggers: Direct HTTP requests
- Pub/Sub: Message queue events
- Cloud Storage: Bucket events
- Firestore: Database events
- Firebase: Authentication, Analytics, Remote Config
- Cloud Scheduler: Cron jobs
- Cloud Tasks: Task queue events

Fonte: Google Cloud Functions Documentation (2023)

Características Principais

Provedor de Nuvem	AWS	GCP	Azure
Limite de tempo de execução	15 minutos	60 minutos	30 minutos
Limite de Memória	10 Gb	8Gb	1.5 Gb
Limite de tamanho de pacotes	250 Mb	100 Mb	100 Mb

Cada fabricante pode possuir “tiers” ou camadas com limites diferenciados

AWS Lambda

O que é AWS Lambda?

- É possível usar o AWS Lambda para executar código sem provisionar ou gerenciar servidores. O Lambda executa o código em uma infraestrutura de computação de alta disponibilidade e gerencia todos os recursos computacionais, inclusive manutenção do servidor e do sistema operacional, provisionamento de capacidade, escalação automática e registro em log. Você organiza seu código em Funções do Lambda. O serviço do Lambda executa a função somente quando necessário e escala automaticamente.

- Ao usar o Lambda, você é responsável apenas pelo seu código. O Lambda gerencia a frota de computação que oferece um equilíbrio de memória, CPU, rede e outros recursos para executar seu código. Como o Lambda gerencia esses recursos, não é possível fazer acessar servidores ou personalizar o sistema operacional no Runtime fornecido.

Fonte: Adaptado de https://docs.aws.amazon.com/pt_br/lambda/latest/dg/welcome.html

O que é AWS Lambda?

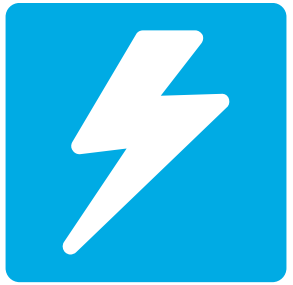
Como o Lambda é um serviço de computação orientado por eventos com tecnologia Serverless, ele usa um paradigma de programação diferente das aplicações Web tradicionais. O seguinte modelo ilustra como o Lambda funciona basicamente:

- Você escreve e organiza o código nas funções do Lambda, que são as peças básicas que você usa para desenvolver uma aplicação do Lambda.
- Você controla a segurança e o acesso por meio de permissões do Lambda, usando perfis de execução para gerenciar com quais serviços da AWS as funções podem interagir e quais políticas de recursos podem interagir com o código.
- As origens de eventos e serviços da AWS acionam as funções do Lambda, transmitindo dados de eventos no formato JSON, os quais as funções processam (isso inclui mapeamentos das origens dos eventos).
- O Lambda executa o código com runtimes específicos da linguagem (como Node.js e Python) em ambientes de execução que empacotam o runtime, as camadas e as extensões.

Fonte: Adaptado de https://docs.aws.amazon.com/pt_br/lambda/latest/dg/welcome.html

O que é AWS Lambda?

Event source



AWS Lambda

Triggers/Gatilhos

Mudanças no estado dos dados

Requisições para endpoints

Mudança de estado de recursos

Node.js
Python
Java
C#
Go
Ruby
BYOR (Bring your own runtime)

Utilizando AWS Lambda
Stateless ou sem estado

- Ao construir funções Lambda, você deve assumir que o ambiente existe apenas para uma única invocação. A função deve inicializar qualquer estado necessário ao ser iniciada pela primeira vez – por exemplo, buscar um carrinho de compras em uma tabela do DynamoDB.
- Ela deve confirmar quaisquer alterações permanentes nos dados antes de sair para um armazenamento durável, como Amazon S3, DynamoDB ou Amazon SQS. Ela não deve depender de nenhuma estrutura de dados existente ou arquivos temporários, ou de qualquer estado interno que seria gerenciado por múltiplas invocações (como contadores ou outros valores calculados e agregados).
- O Lambda fornece um inicializador antes do manipulador, onde você pode inicializar conexões de banco de dados, bibliotecas e outros recursos. Como os ambientes de execução são reutilizados sempre que possível para melhorar o desempenho, você pode amortizar o tempo gasto para inicializar esses recursos em múltiplas invocações. No entanto, você não deve armazenar nenhuma variável ou dado usado na função dentro desse escopo global

Anatomia da Função Lambda

Handler() function

- Function to be executed.

Event object

- Used to pass in event data to the handler.
- Java and C# support simple data types, POJOs/POCOs, and stream input/output.

Context object

- Provides your handler runtime information. (identity, awsRequestId, invokedFuntionArn, clientContext, logStreamName)

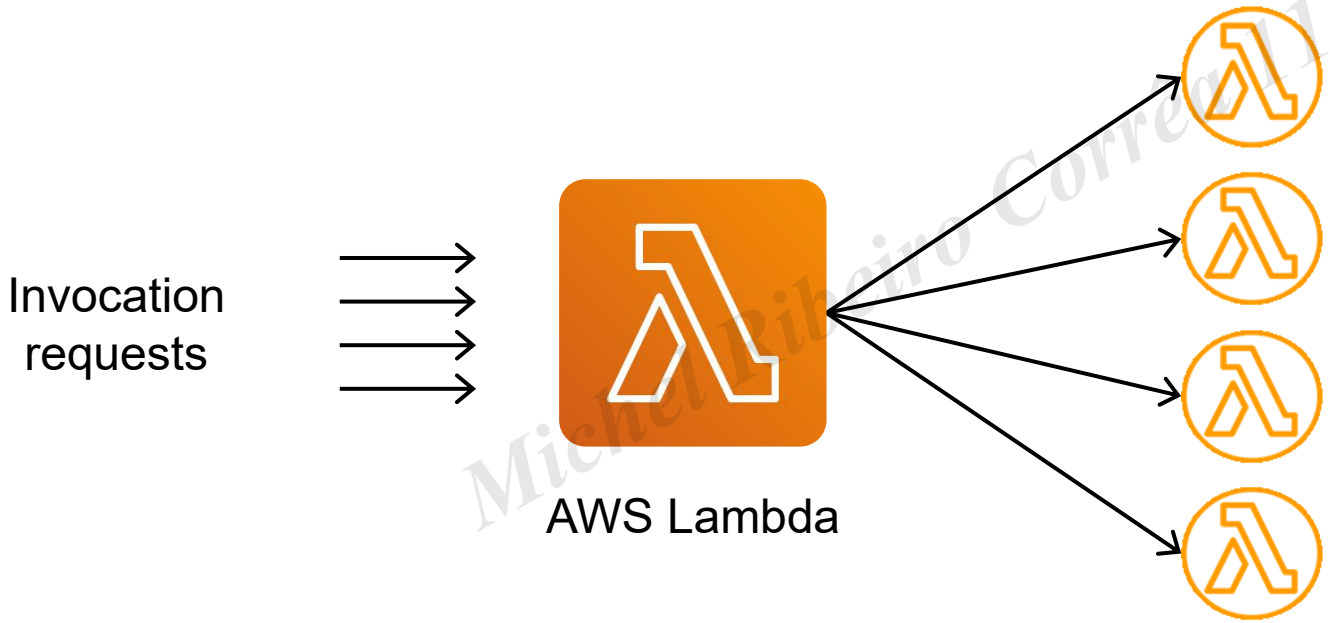
```
public String handleRequest(Book book, Context context) {
    saveBook(book);

    return book.getName() + " saved!";
}
```

```
def my_handler(event, context):
    message = 'Hello {} {}!'.format(event['first_name'],
                                    event['last_name'])
    return {
        'message' : message
    }
```

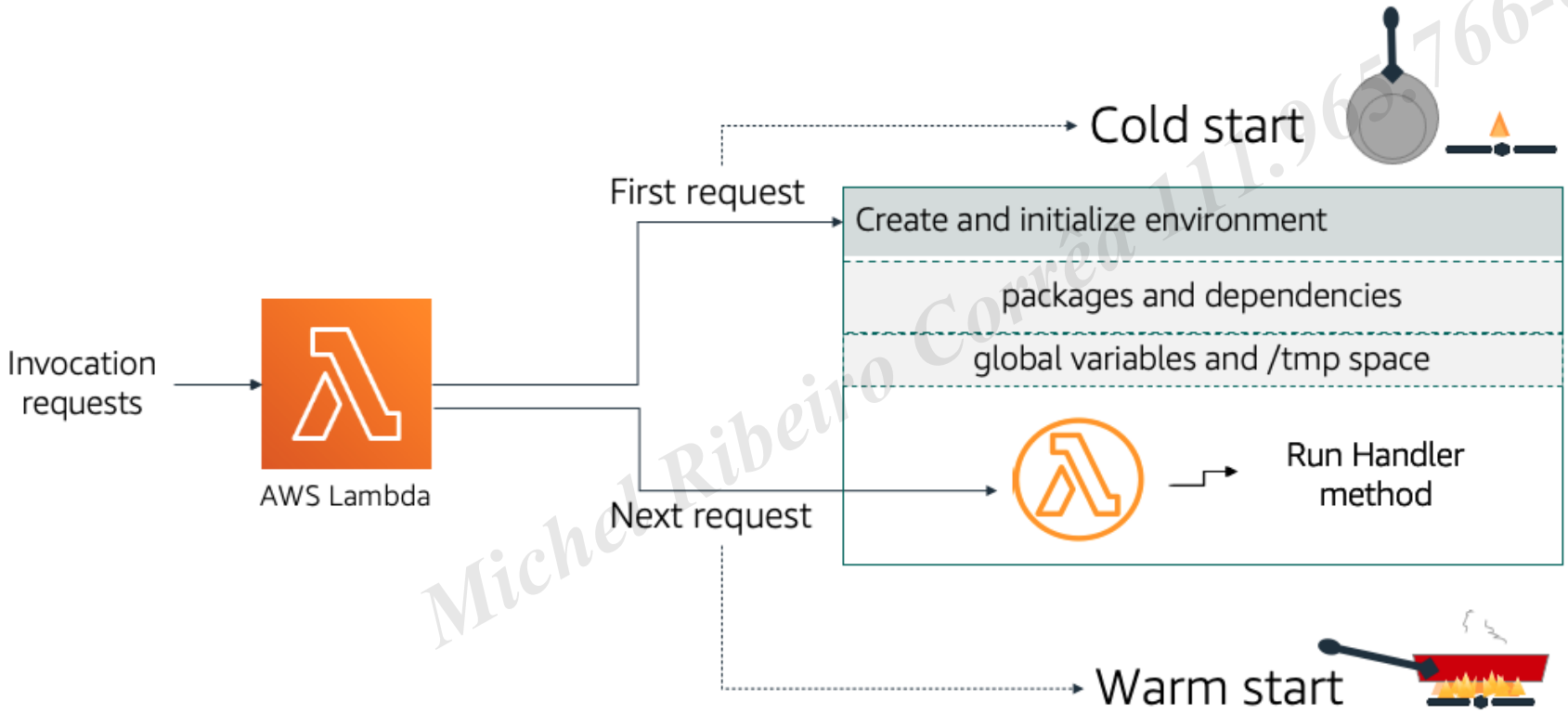
Anatomia da Função Lambda

- Funções Lambda executadas em ambientes efêmeros sob demanda
- **Simultaneidade**: o número de invocações de função em execução ao mesmo tempo
- Triggers ou gatilhos: Eventos que iniciam a execução de uma função serverless



Anatomia da Função Lambda

- Lambda pode criar um novo ambiente "frio" ou reutilizar um "quente"

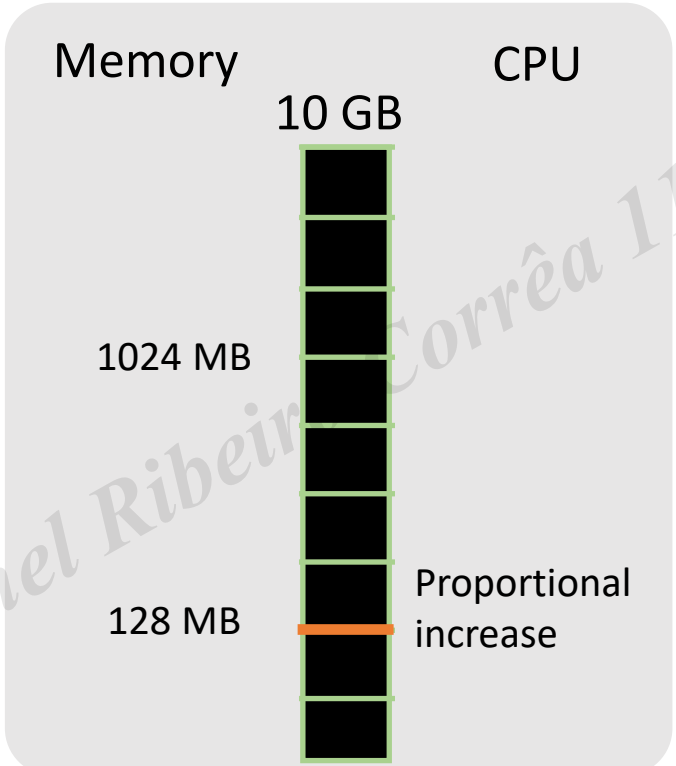


Anatomia da Função Lambda

- Escolha de memória e tempo limite

Function code

Function configuration
- Memory
- Timeout



- A memória determina uma quantidade proporcional de CPU.
- Mais memória tem um custo mais alto.
- O tempo limite equilibra as conclusões bem-sucedidas com o não pagamento por funções “travadas”.
- Uma duração mais longa tem um custo mais alto.
- **Memória mais alta pode significar durações mais curtas e pode resultar em custo geral mais baixo**.

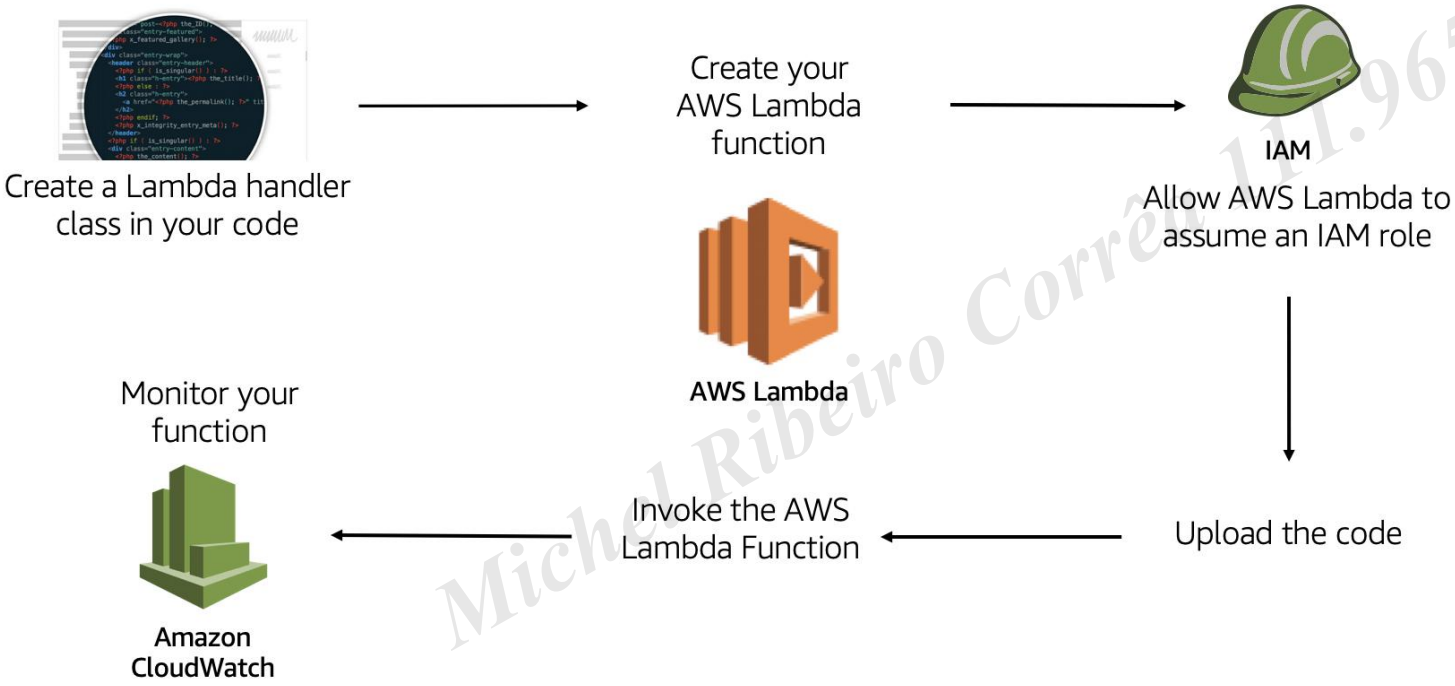
Anatomia da Função Lambda

- Quotas e Limites Lambda

  https://docs.aws.amazon.com/pt_br/lambda/latest/dg/gettingstarted-limits.html

Anatomia da Função Lambda

- Ciclo de desenvolvimento e deploy



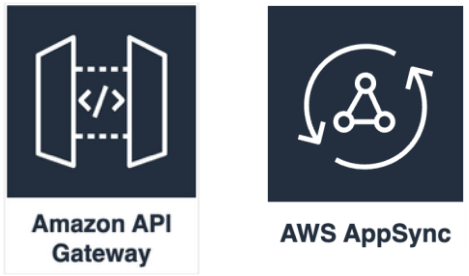
Anatomia da Função Lambda

- Cobrança Granular, baseada no uso
  - O número total de solicitações em todas as suas funções.
  - O período de tempo que seu código é executado.
  - A quantidade de memória que você aloca para sua função Lambda

- https://aws.amazon.com/pt/lambda/pricing/

AWS Serverless Platform

Compute



API Proxy

Storage

Database

Interprocess Messaging

Orchestration

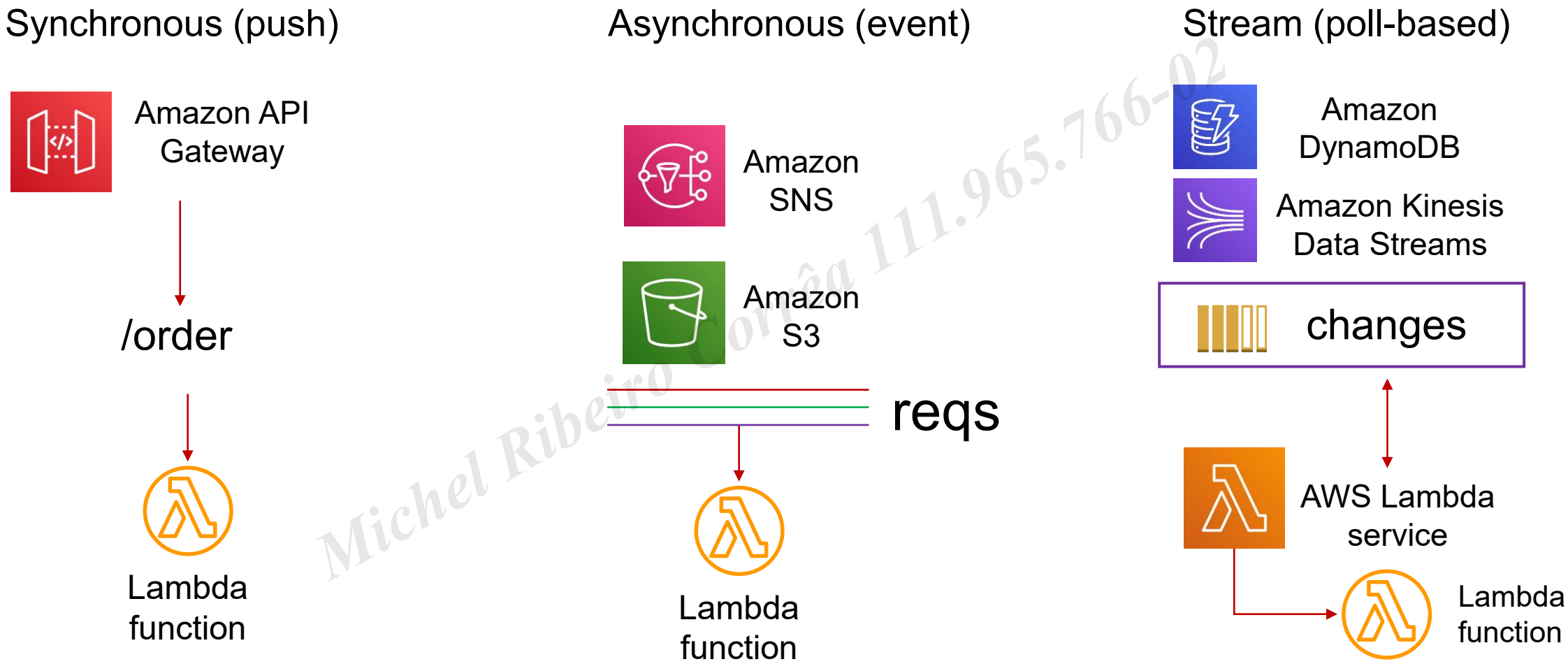
Developer Tools

Exercício

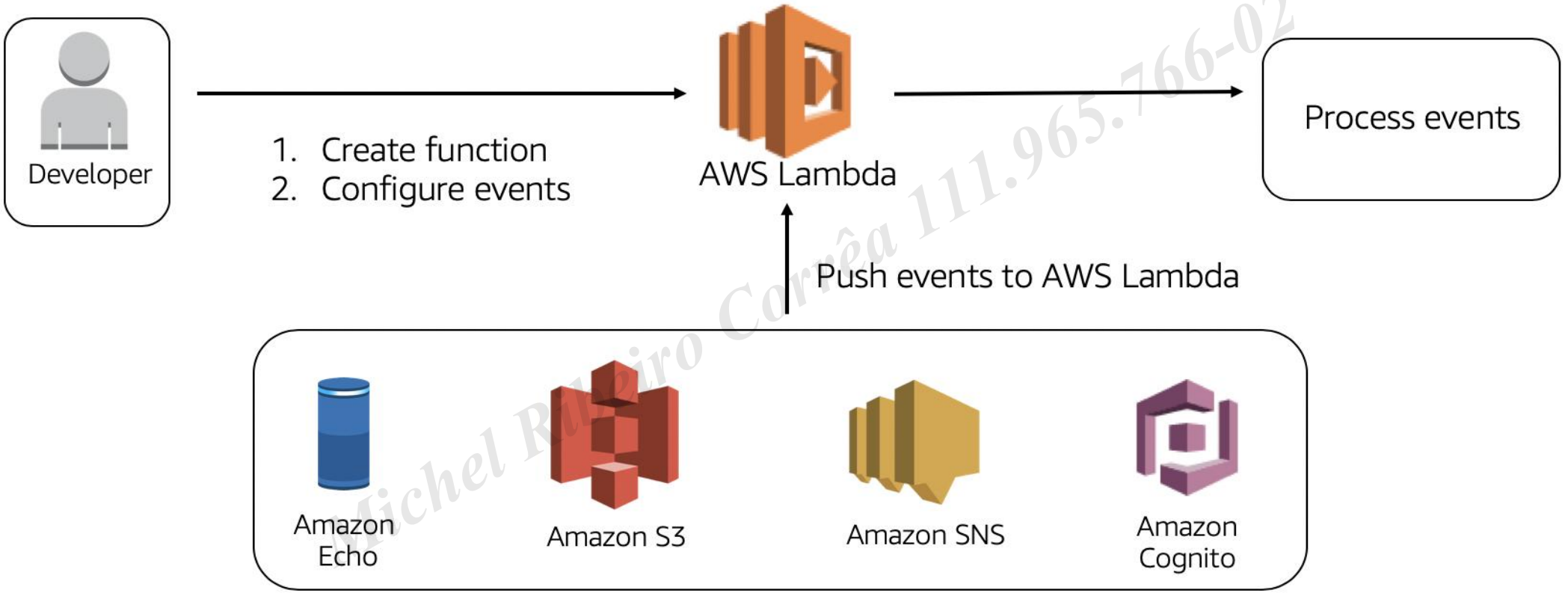
- Crie sua primeira Lambda Function
- https://docs.aws.amazon.com/lambda/latest/dg/getting-started.html

AWS Lambda Execution Model

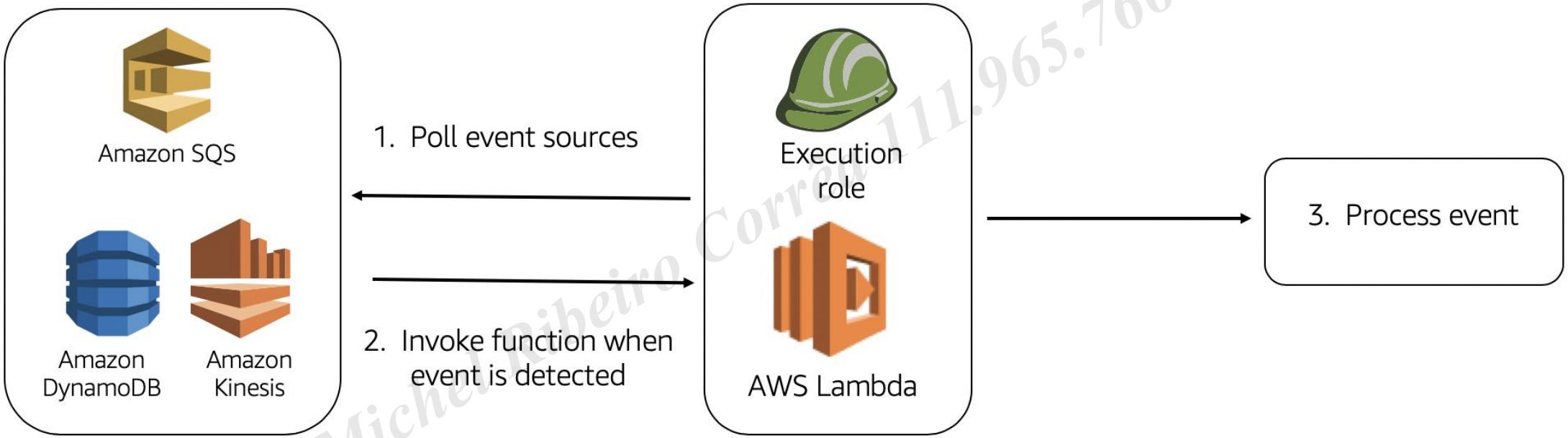
AWS Lambda execution model



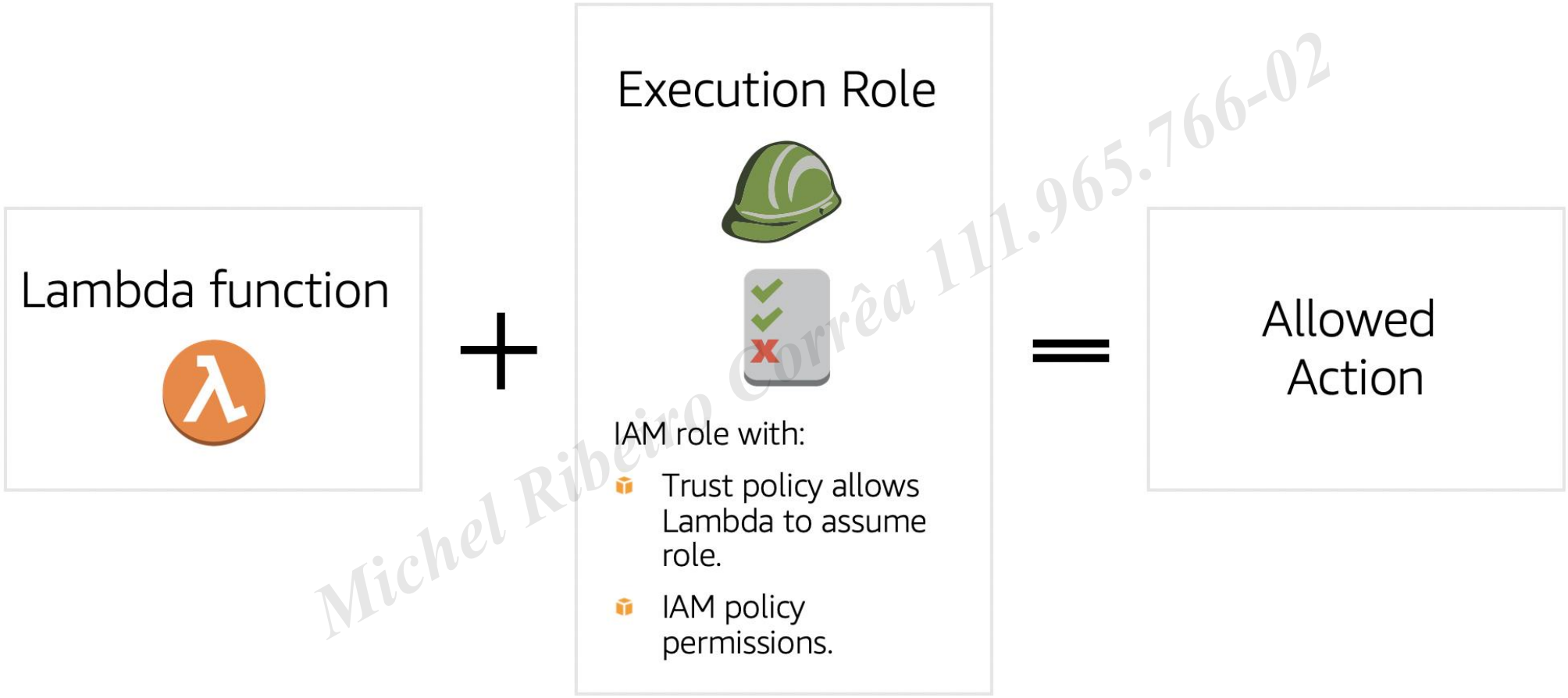
Push Event Model



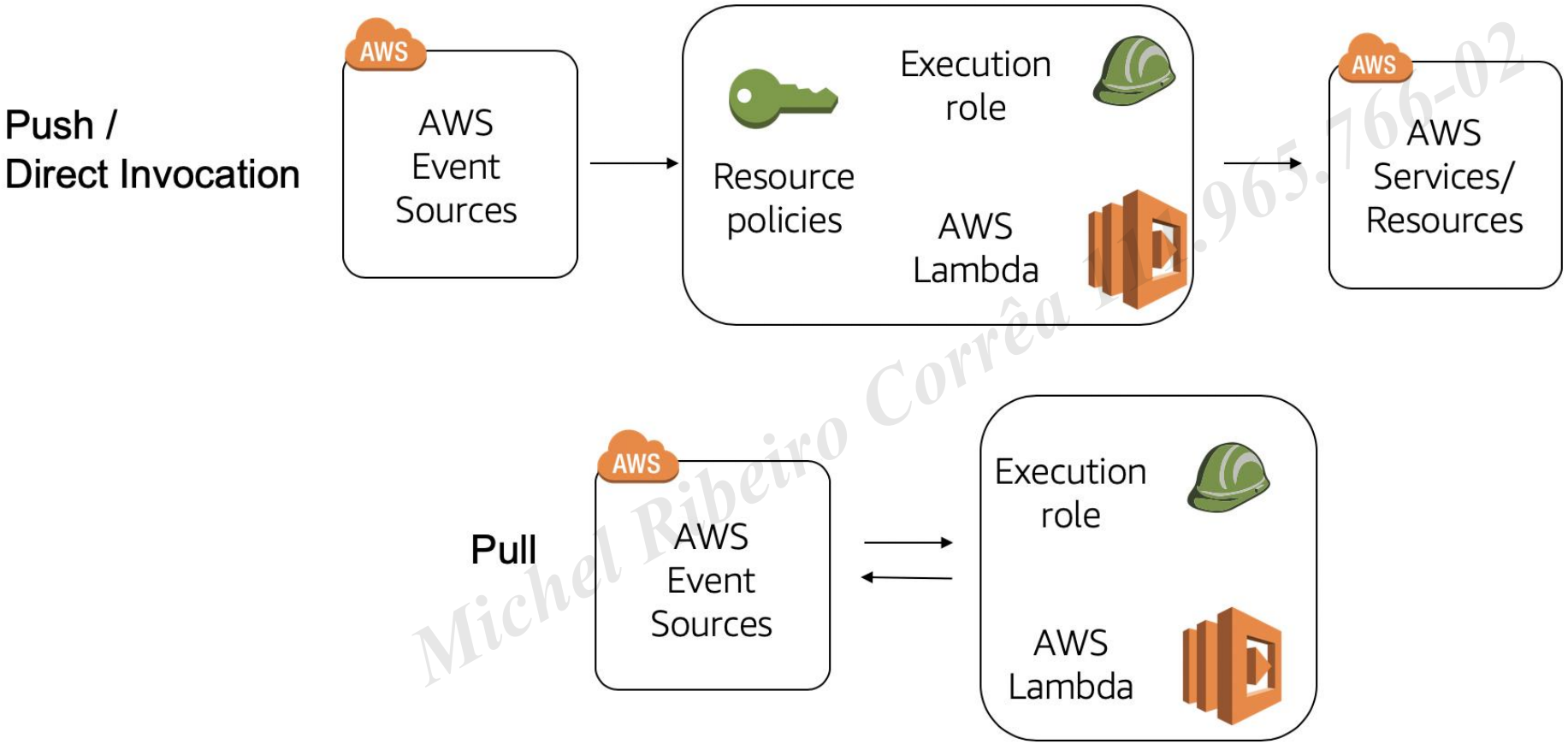
Pull Event Model



Permissão de Execução

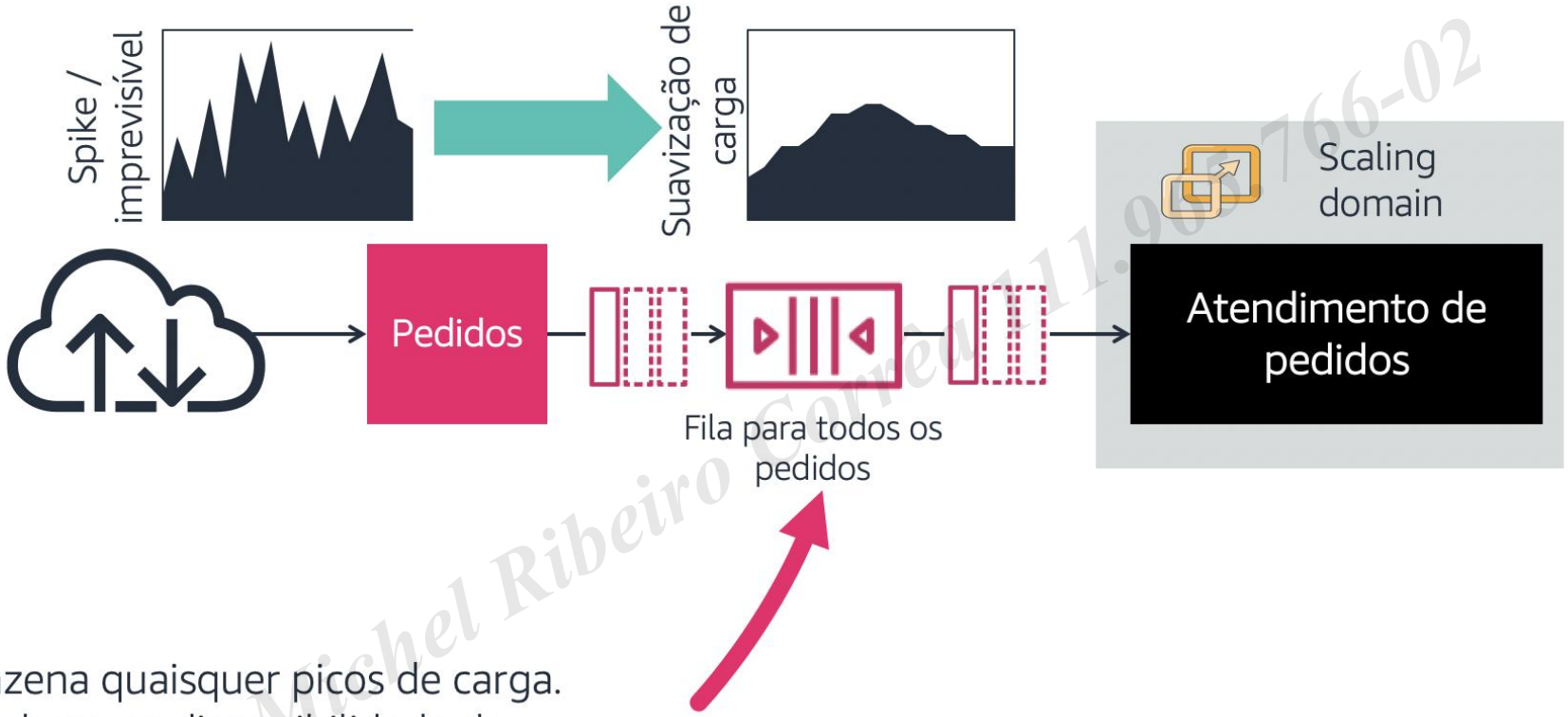


Permissão de Execução



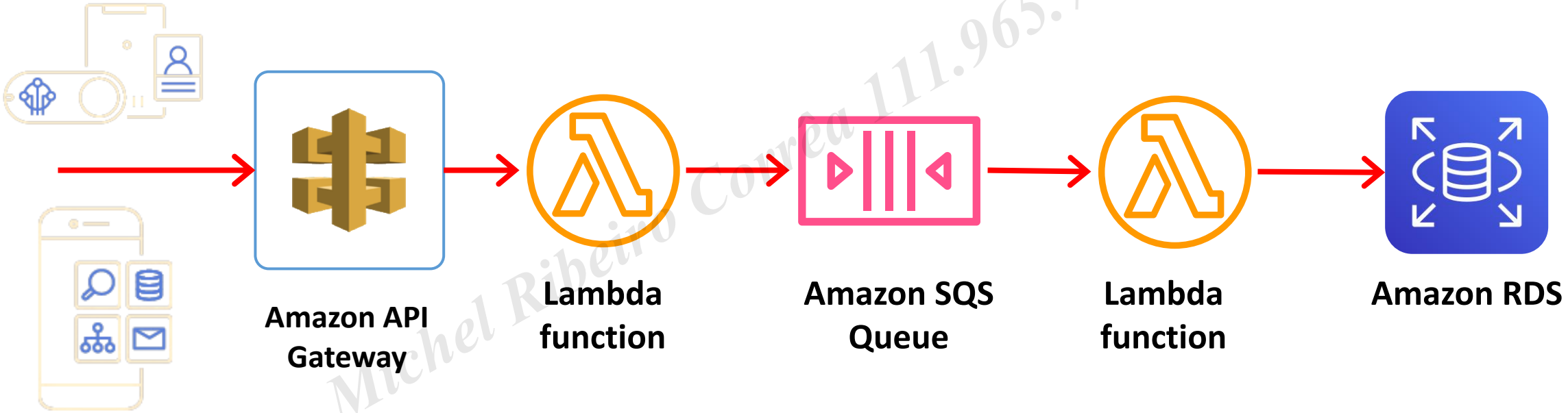
Resiliência e Escala com Event-Driven Architecture

Solução: Desacoplamento Assíncrono

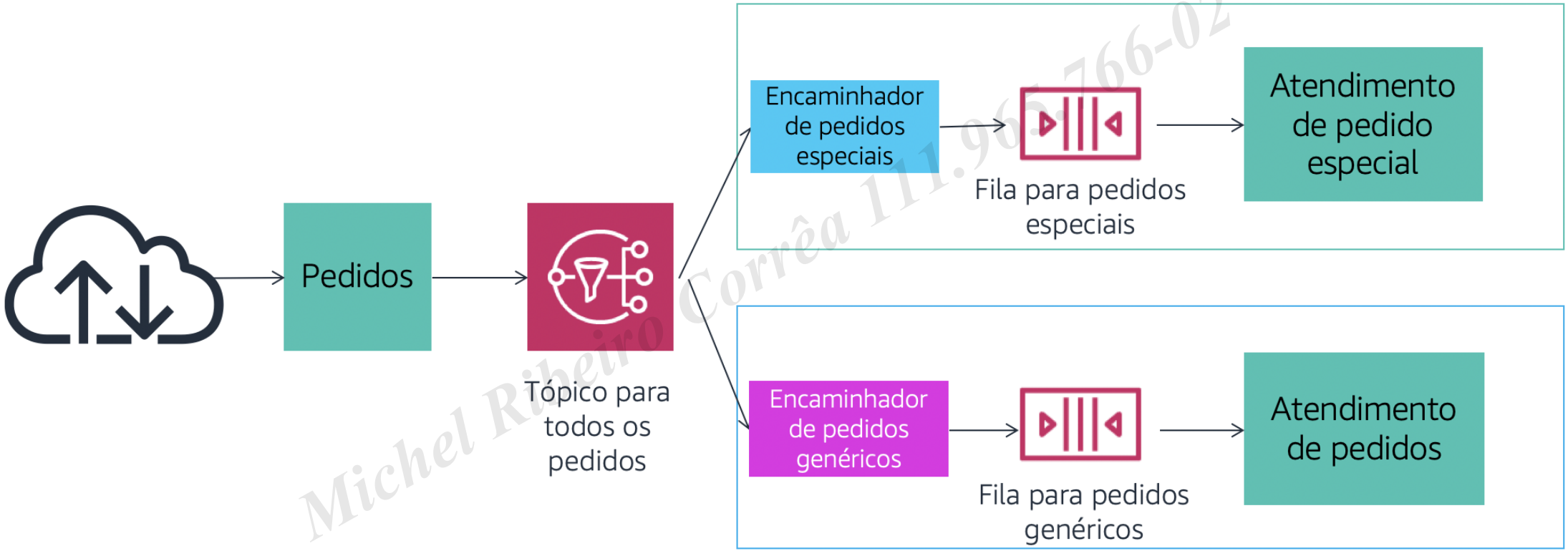


A fila armazena quaisquer picos de carga. Escale com base na disponibilidade de recursos.

Solução: Desacoplamento Assíncrono



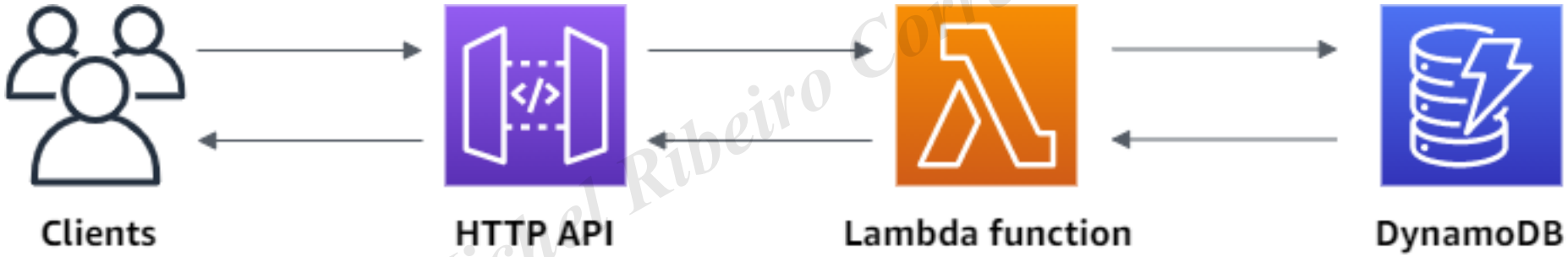
Solução: Fan-out Pattern com Amazon SNS



Exercício

Exercício

- Crie uma API HTTP CRUD com o Lambda e o DynamoDB
- https://docs.aws.amazon.com/pt_br/apigateway/latest/developerguide/http-api-dynamo-db.html



Obrigado!

Prof. José Maria Cesário Jr. | https://www.linkedin.com/in/josemariacesariojunior/

MBA USP ESALQ