📘 Aula - AWS - Lambda (Funções como serviço ou FaaS)

> *O AWS Lambda é um serviço de computação sem servidor que permite executar código sem provisionar ou gerenciar servidores, criando lógica de dimensionamento de cluster com reconhecimento de workloads, mantendo integrações de eventos ou gerenciando tempos de execução. Com o Lambda, você pode executar o código para praticamente qualquer tipo de aplicação ou serviço de back-end, tudo sem precisar de administração.*

O AWS Lambda é serviço da Amazon serverless (traduzido literalmente do inglês, significa “sem servidor”). Isso não é verdade. A arquitetura Serverless ainda é baseada em servidores, porém o desenvolvedor não precisa gerenciar o servidor. O serverless também pode ser conhecido como FaaS(Function as a Service).

A vantagem do AWS Lambda que é gratuito para até 1 Milhão de solicitações por mês. Isso mesmo após os 12 meses de alguns serviços gratuitos.

Porque Usar?

Uma das principais razões para se utilizar o serverless é para tirar a carga de operações que podem ser custosas do backend. Como um processamento e operações que possam demorar muito tempo ou que consumam muito recursos da máquina como memória ou processador.

Colocando essas operações em um serviço serverless permite que essa carga seja tirada da sua aplicação, assim diminuindo o impacto que teria sobre outros usuários, além de reduzir as especificações técnicas da máquina que a sua aplicação necessita.

Configurando o Lambda

1. Pesquisa pelo serviço Lambda na barra de pesquisa da AWS

2. Aqui você terá algumas opções como:

- **Criar do zero:** utilizado para criar funções do zero.

- **Usar em esquema:** são exemplos de funções para atividades recorrentes na AWS.

- **Imagem de contêiner:** caso você queira utilizar uma imagem de contêiner salva no Amazon ECR (Elastic Container Registry).

- **Explorar o repositório de aplicativos sem servidor:** são exemplos de aplicações Serverless. Neste temos exemplos de arquiteturas utilizando os serviços da AWS e você consegue facilmente instalar e rodar.

> **Aviso**
>
> Para esse exemplo usaremos a Criação de funções do zero

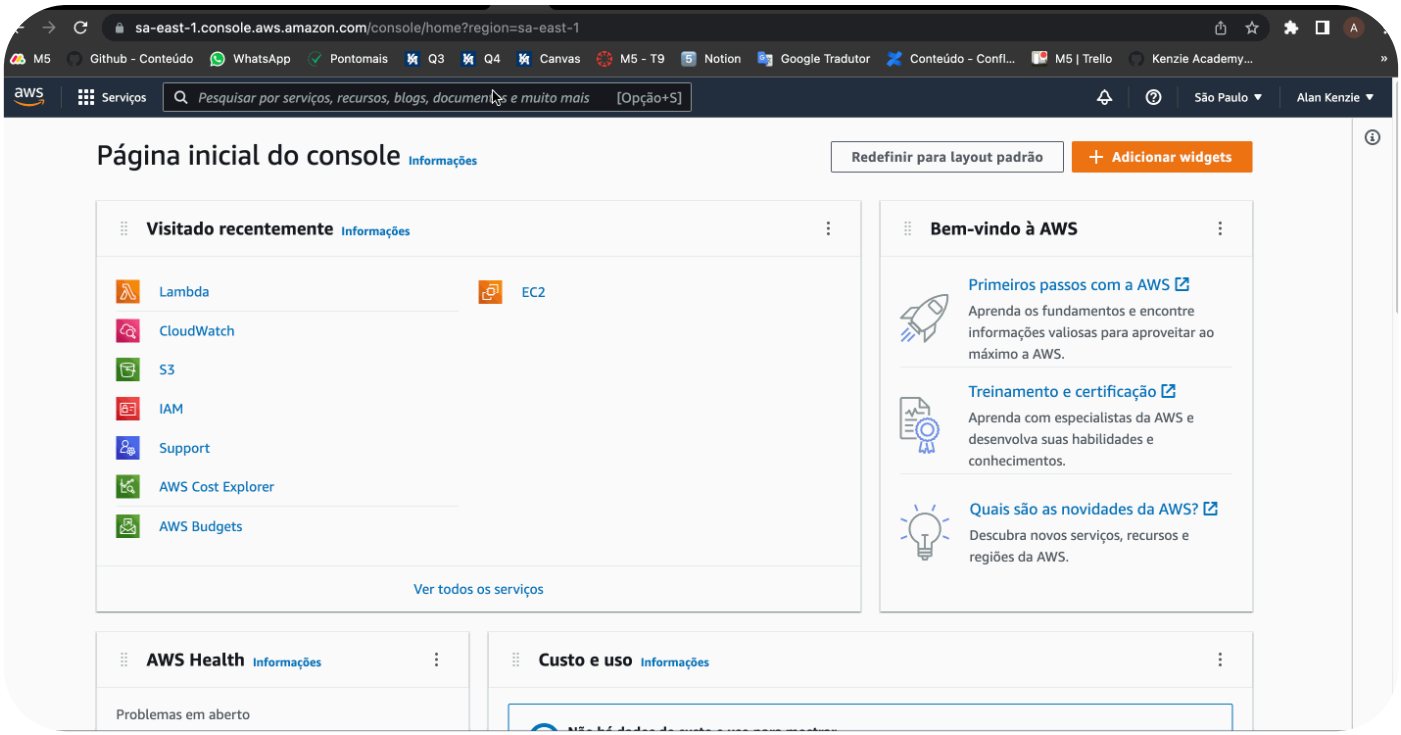
3. Dê um nome para a sua função - o nome da função deve conter somente letras, números, hifens (-) e sublinhados (_).

4. Escolha a linguagem a ser utilizada

5. Clique em **Criar função**

> **Aviso**
>
> Para Permissões, o Lambda criará automaticamente uma função de execução

básica para que a função do Lambda possa acessar o CloudWatch para logs.



Testando nossa função

Uma função de teste foi criada e conseguimos ter acesso ao painel de controle com diversas funções, inclusive para testarmos.

O código que o Lambda gerou para nós foi o seguinte:

Copiar para área de transferência

```
import json


def lambda_handler(event, context):
    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!'
    }
```

- **event:** é o parâmetro principal. Neste argumento recebemos os dados envidados pelos serviços da AWS podendo conter dados de uma mensagem recebida no SQS e até mesmo dados de um request repassado pelo API Gateway.

- **context:** neste recebemos um objeto LambdaContext. Temos neste objeto informações do contexto de invocação deste lambda como a quantidade de memória configurada até o ARN de quem invocou esta função. **Aqui temos uma explicação detalhada deste objeto**.

Vamos adicionar alguns *prints* à esse código, para que consigamos visualizar os logs da aplicação, juntamente com a mensagem de sucesso.

Copiar para área de transferência

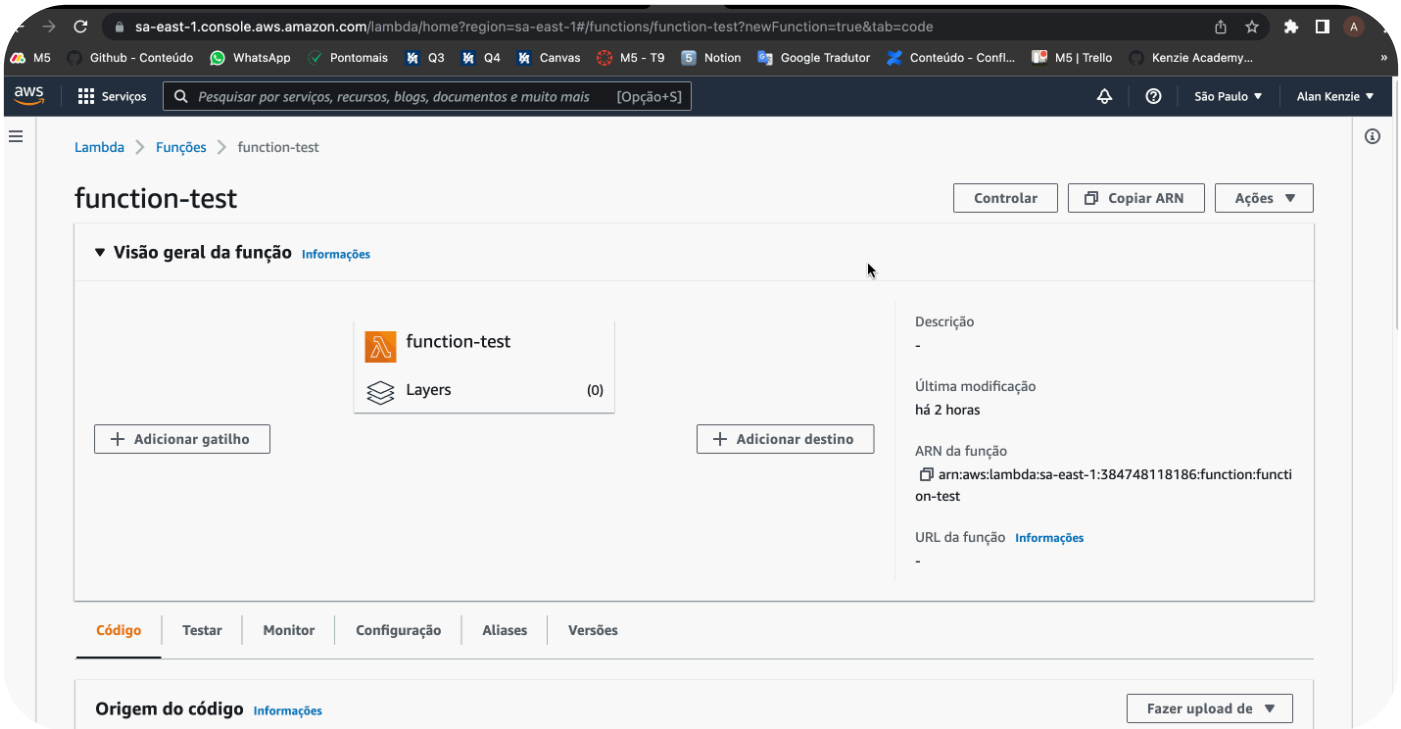
```
import json

def lambda_handler(event, context):
    print('Hello from lambda!')
    print(event)
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!'
    }
```

Agora podemos testar nossa função:

1. Primeiramente, clique em **Deploy** para que nossas alterações tenham efeito
2. Após, clique em **Test**
3. Configure um evento de teste

- Dê um nome para o evento
- Clique em **Salvar**

> **Aviso**
>
> O JSON do evento será o event da nossa função.
>
> Quando você executa um evento de teste no console, o Lambda invoca sua função de maneira síncrona com o evento de teste. O tempo de execução da função converte o documento JSON em um objeto e o transmite ao método do manipulador do código para processamento.

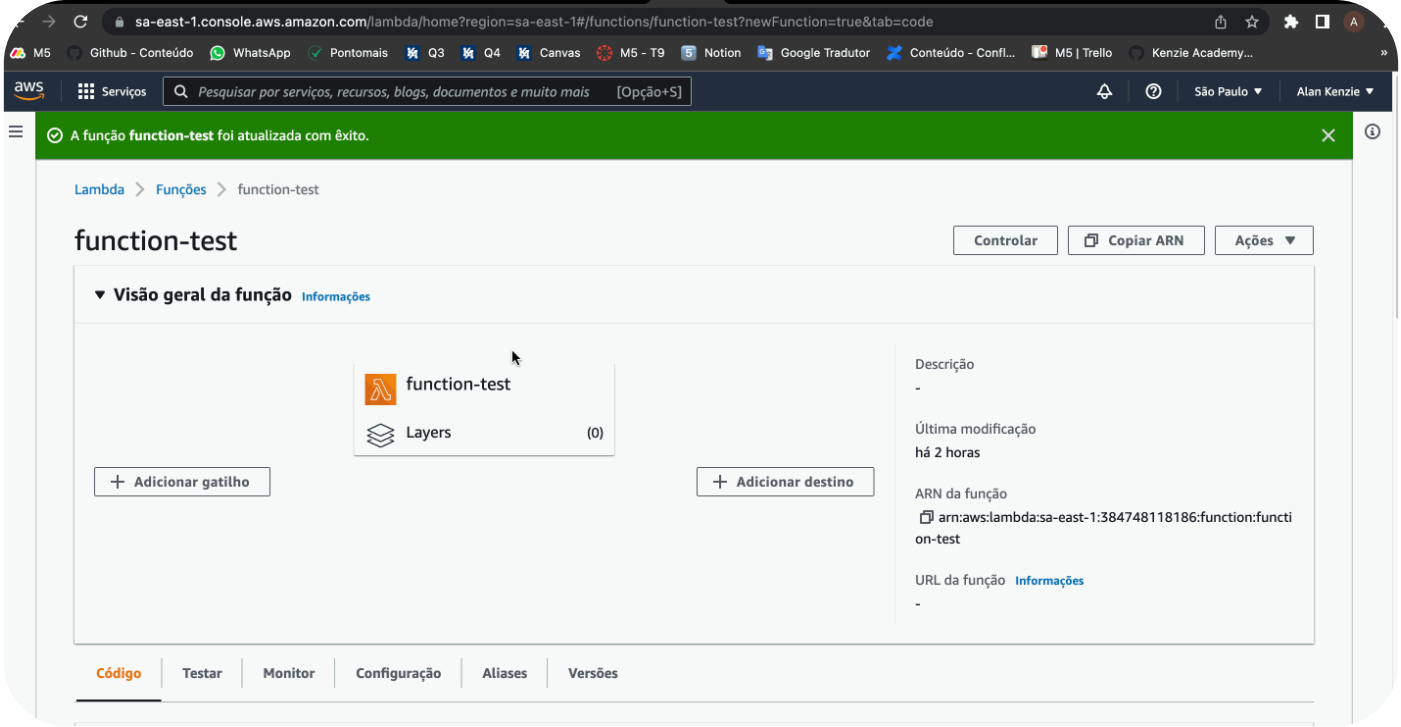


Atualize sua função para que ela falhe! Dessa vez, deixaremos a função print sem o fechamento dos parêntesis, para vermos um caso de falha!

Copiar para área de transferência

```
import json

def lambda_handler(event, context):
    print('Hello from lambda!'
    print(event)
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!'
    }
```

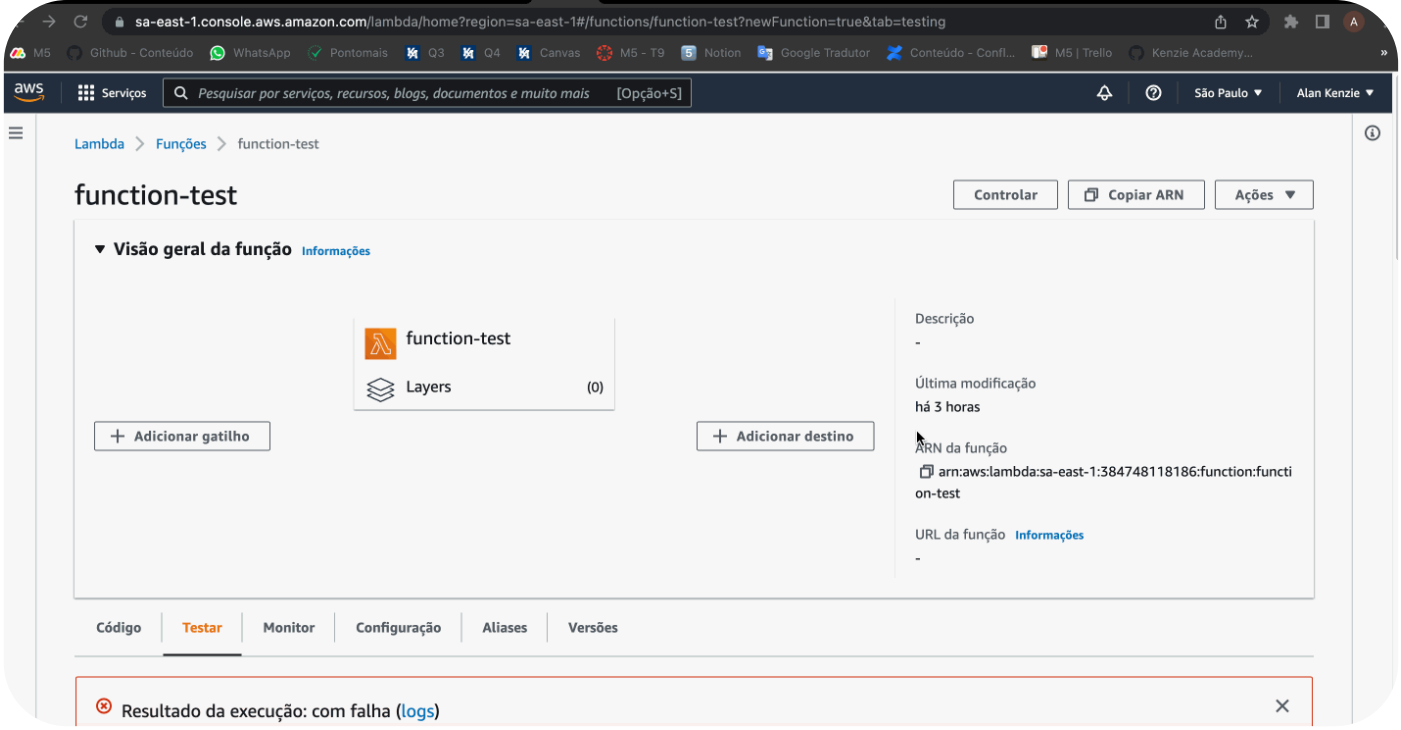
- Clique em **Deploy**
- Clique em **Test**



Visualizando os logs com CloudWatch

O Amazon CloudWatch monitora os recursos da Amazon Web Services (AWS) e as aplicações executadas na AWS em tempo real. Você pode usar o CloudWatch para coletar e monitorar métricas, que são as variáveis que é possível medir para avaliar seus recursos e suas aplicações.

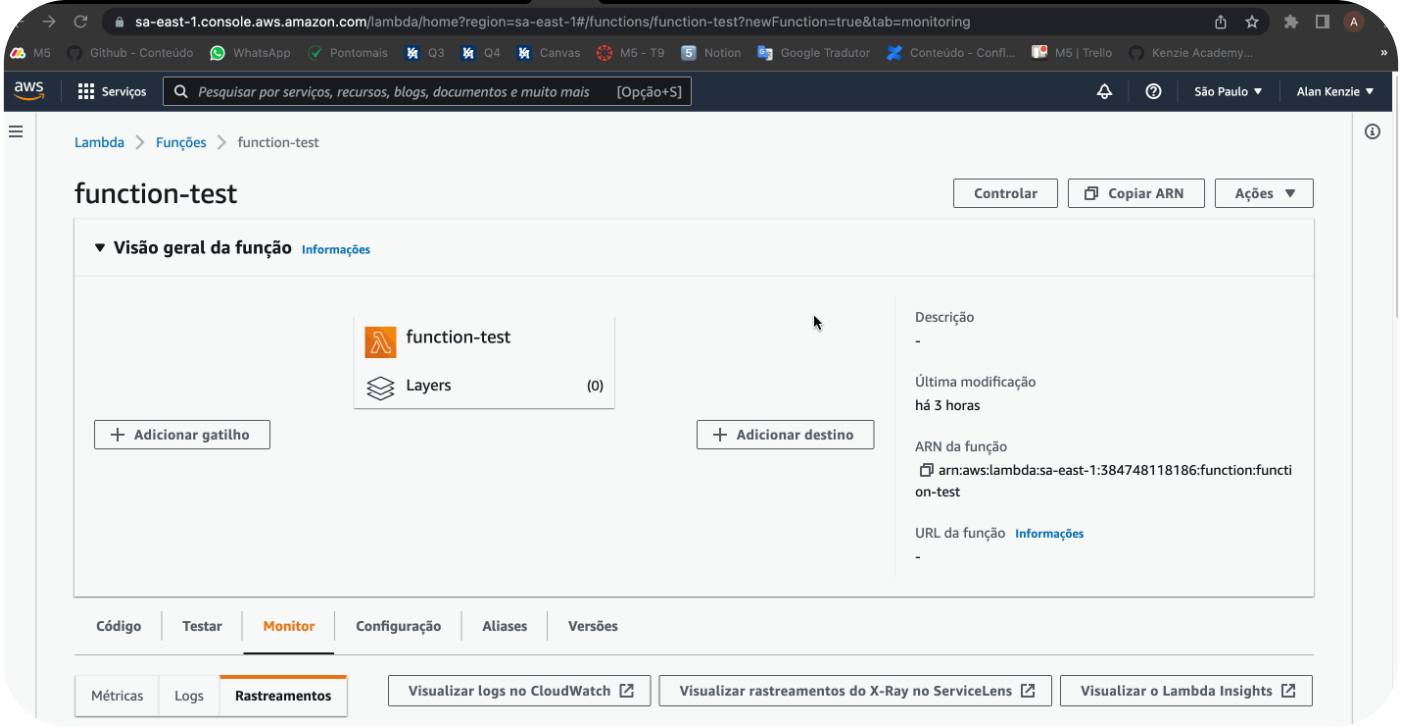
1. Clique na guia **Monitor**



Nessa tela veremos muitas informações a respeito do monitoramento da nossa função Lambda, vamos dar uma olhada nos logs do CloudWatch, cada vez que nossa função é executada, ela cria um novo fluxo de log. Isso facilita a análise do que deu errado em um cenário de falha específico. O fluxo de log mais recente mostra a saída de log que a mensagem de erro usa.

1. Clique em **Visualizar logs no CloudWatch**

Aqui teremos um overview sobre cada log gerado, desde a invocação da funcão, até o report gerado.



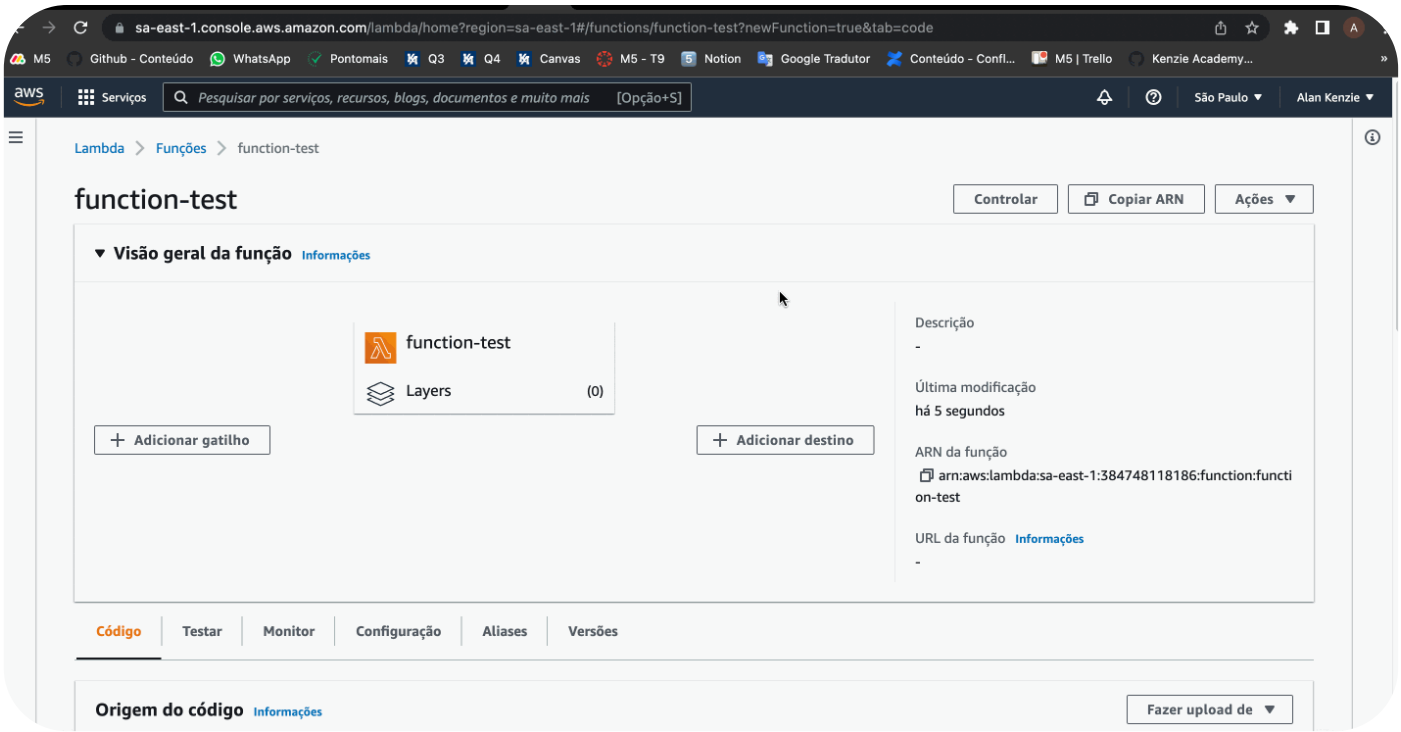
Configurar e acessar variáveis de ambiente

Dependendo de nossa aplicação, pode ser importante ocultarmos informações sensíveis, e o Lambda nos fornece um meio de configurarmos variáveis de ambiente e também veremos como criptografar informações confidenciais em variáveis de ambiente.

1. Primeiramente clique em **Configuração**
2. Depois clique em **Variáveis de ambiente**
3. Clique em **Editar**
4. Clique em **Adicionar variáveis de ambiente**

5. Digite suas variáveis de ambiente

6. Clique em **Salvar**

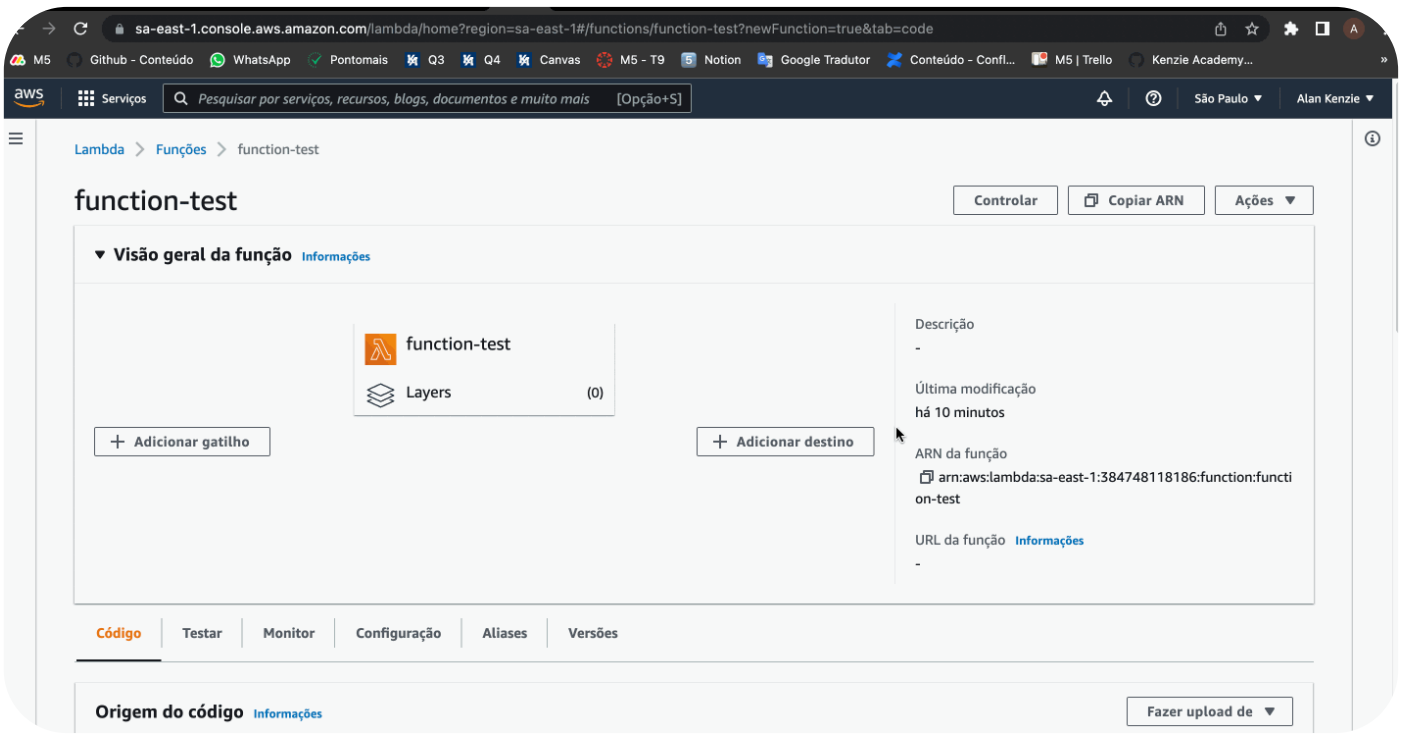


Podemos acessar nossas variáveis de ambiente da mesma maneira que faríamos em nosso código com `os.environ` ou `os.getenv`.

Vamos atualizar nossa funcão para acessarmos nossas variáveis de ambiente.

Copiar para àrea de transferência

```
import os


def lambda_handler(event, context):
    DB_HOST = os.environ.get("DB_HOST")
    DB_USER = os.environ.get("DB_USER")
    DB_PASS = os.environ.get("DB_PASS")
    print(f'Connected to {DB_HOST} as {DB_USER}
    return None
```



Critptografar variáveis de ambiente

Importante!

Você deve ter percebido que quando criamos nossas variáveis de ambiente, logo abaixo temos acesso às configurações de criptografia, para podermos criptografar nossas variáveis de ambiente, precisamos criar uma chave de criptografia na AWS gerenciada pelo usuário, e há um custo de US$ 1.00 (um dólar) por chave gerenciada pelo usuário, não há período de gratuidade, por isso, recomandamos que não façam a criptografia das variáveis de ambiente.

Referências!

AWS Lambda | AWS

Conceitos básicos do AWS Lambda | AWS

Chamada de funções do AWS Lambda | AWS

Objeto de contexto do AWS Lambda em Python | AWS