Resize imagem

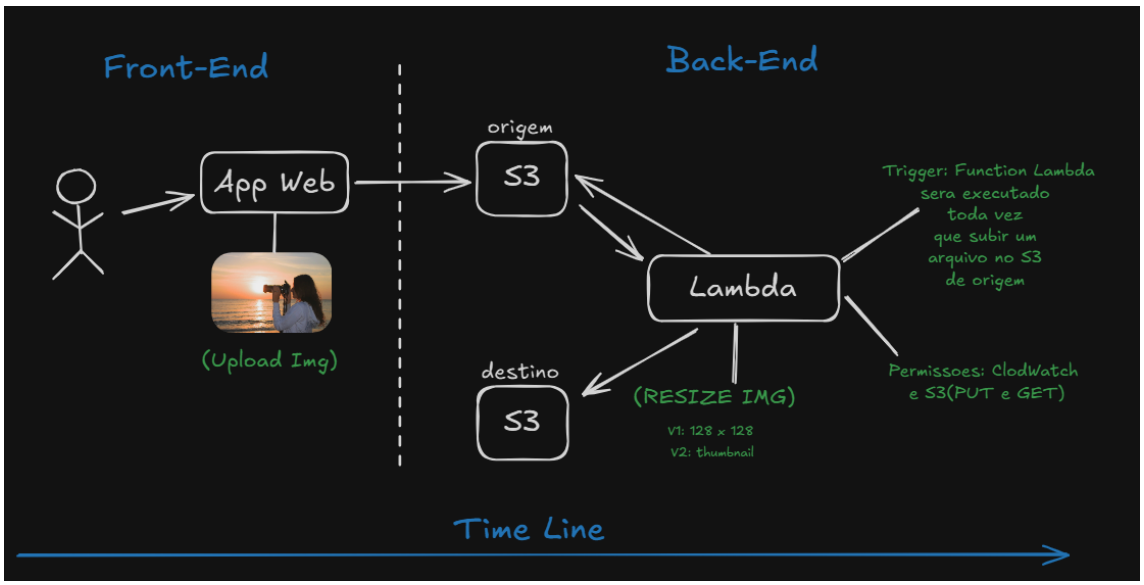
Apresentação Projeto

Nesse projeto, iremos automatizar o tamanho de um **upload** de uma **imagem**!!

Pois em alguns momentos, pode ser que em uma aplicação web por exemplo, um usuário realize um **upload** de uma **imagem muito grande**, totalmente fora dos padrões aceitos da nossa aplicação!

E para esse reajuste do tamanho da imagem não ficar para o usuário, ou até mesmo pela aplicação em si, vamos criar um **script python**, automatizando quando o processo de **upload** de uma **imagem** no serviço de **Bucket S3**, reajustando o tamanho da **imagem** automaticamente!

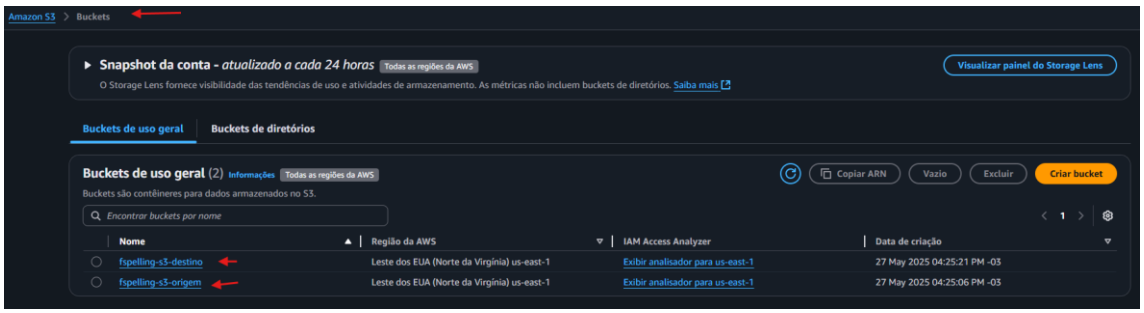
Time line Projeto



Criação buckets S3

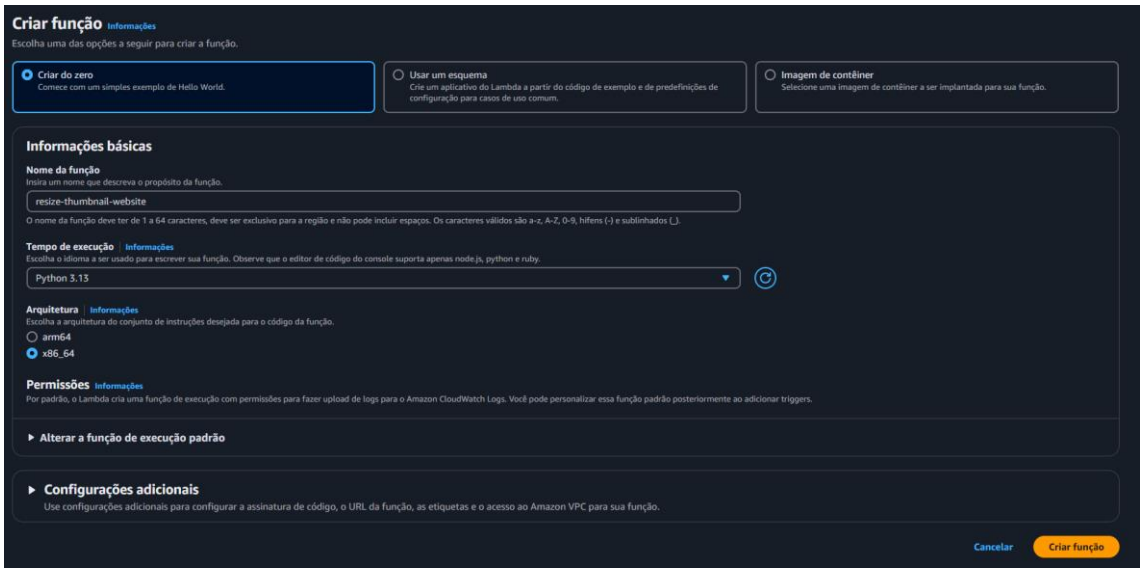
Primeiramente precisamos criar os nossos **buckets** (**storages**) **S3**, tanto para o repositório **bucket** de **origem** quanto o de **destino**!

Dito isso segue a criação simples desses **buckets**!

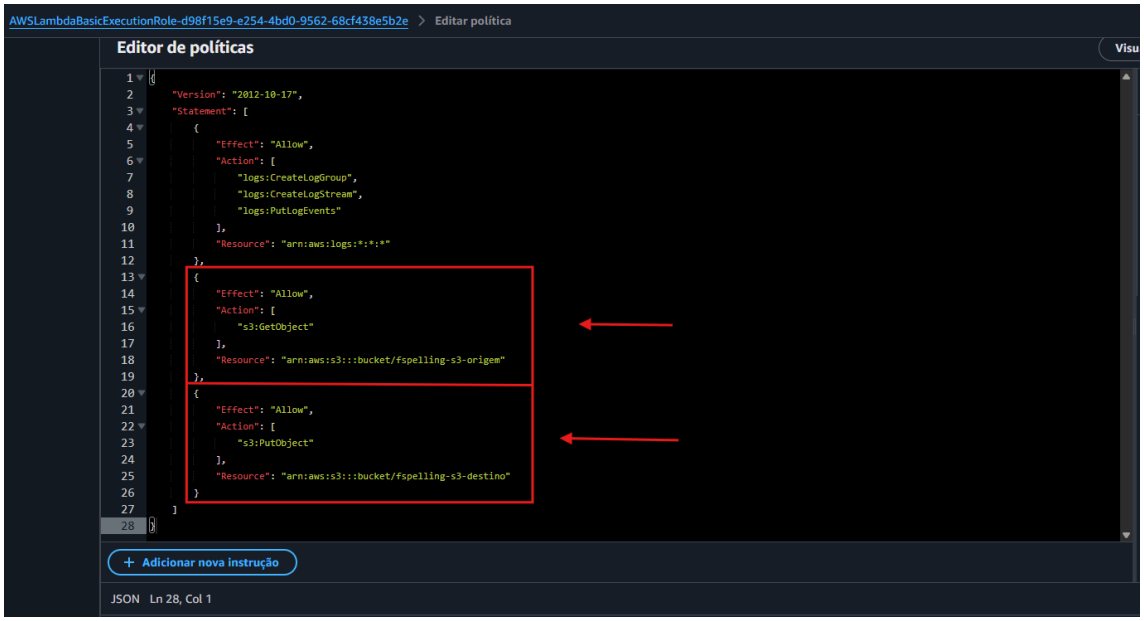


Criação função lambda

Com os **buckets S3** criados, vamos criar a nossa **função lambda**!! Logo segue a criação de forma simples a criação da nossa **função lambda**!!



Após a criação da função lambda, como já sabemos, o **IAM Role** padrão dará permissão apenas para o serviço do **CloudWatch**. Logo precisaremos editar a política dessa role, adicionando as permissões referente ao serviço do **S3** também!



Veja que foi adicionado **2 novas permissões**:

- S3:GetObject: permitir **buscar** arquivos no repositório (**fspelling-s3-origem**) do **bucket S3**.
- S3:PutObject: permitir **adicionar/alterar** arquivos no repositório (**fspelling-s3-destino**) do **bucket S3**.

Configurar trigger na função lambda

Com a **função lambda** criada, chegou a hora de configurar uma **trigger** para ela!!

Pois como já foi citado no **time line** do projeto, toda vez que for persistido um arquivo no repositório (**fspelling-s3-origem**) do **S3**, deverá ser executado a nossa **função lambda** através de um **evento**, que será vinculado por essa **trigger**.

Dito isso, segue a configuração da **trigger** da nossa **função lambda**:

Segue abaixo toda a explicação de cada **configuração** que foi utilizado para a criação dessa **trigger** para a nossa **função lambda**:

- **Origem da trigger**: Serviço de origem que será disparado o evento, que consequentemente irá executar a nossa **função lambda**, que nesse caso o serviço de origem será o **S3**.

- **Bucket**: Como o serviço de origem será **S3**, logo é necessário informar qual **bucket** que será vinculado, que nesse caso será o **fspelling-s3-origem**.

- **Tipo de evento**: Tipo de evento desse **bucket** do **S3** que será vinculado com a nossa **função lambda**, que nesse caso será **todos os eventos de criação de objeto**. Ou seja, todo arquivo que for persistido nesse **bucket**, será executado a nossa função lambda, através da invocação desses eventos!

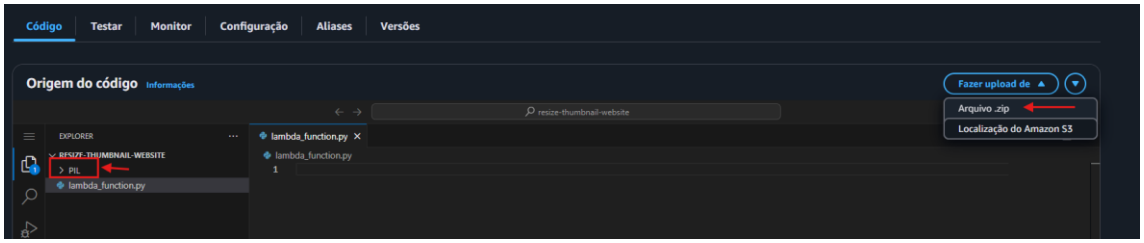
- **Prefixo - Opcional**: Quando é necessário informar que os eventos só serão aceitos, caso os arquivos forem persistidos em um **diretório específico**!

- **Sufixo - Opcional**: Quando é necessário informar que os **eventos** só serão aceitos, caso os arquivos tiverem uma **extensão específica**!

Criação script função lambda

Primeiramente antes de criar o nosso **script**, precisaremos importar manualmente uma **biblioteca python** referente a manipulação de imagens, que será o **PIL**, pois o **aws lambda** não tem acesso nativo com essa biblioteca!

Dito isso, vamos baixar a **biblioteca** do **PIL** em nossa máquina, e consequentemente vamos empacotar em um arquivo **.zip** para realizar o **upload da biblioteca**! Pois o **aws lambda** só aceita realizar uploads manuais através de arquivos **.zip**!!



Agora sim podemos começar a criação do nosso **script**, e para começar vamos importar todas as bibliotecas necessárias para a execução do nosso script:

```
lambda_function.py
1   import os
2   import tempfile
3   import boto3
4   from PIL import Image
```

- **pil**: biblioteca para manipular **tamanho de imagens**.
- **os**: biblioteca para buscar **variáveis de ambiente**.
- **boto3**: biblioteca para ter integração total com os serviços da **aws**, que nesse caso será com o serviço do **S3**.
- **tempfile**: biblioteca para criar nome de arquivo ou diretórios temporários de forma simples e segura.

Agora vamos definir uma variável **SIZE**, onde será pré-definido o tamanho do **resize** da **imagem** para **128x128**, além da nossa **variável de ambiente** referente ao **bucket** de **destino** do **S3** que salvaremos essa nova imagem redimensionada!

```
6   SIZE = (128, 128)
7   DEST_BUCKET = os.environ['DEST_BUCKET']
```

Agora precisamos definir a nossa variável referente ao **objeto** do **S3**, e para isso dessa vez utilizaremos a função **client()**, por ser da **documentação oficial** da **AWS**, e nesse caso terá uma **performance** levemente melhor também!

```
9   s3_client = boto3.client('s3')
```

Enfim vamos criar a nossa função principal do lambda, **lambda_handler()**.

E para iniciar a nossa implementação da função, vamos realizar um **for loop** referente aos registros de eventos que possamos obter no disparo dessa função! E como queremos garantir que o objeto do disparo do evento seja de um **objeto S3**, faremos então uma **validação simples**, ficando assim:

```
for record in event['Records']:
    if 's3' in record:
        |
```

Sendo um evento de um **objeto S3**, logo vamos obter os dados referente ao nome do nosso **bucket S3** além do nome do arquivo (**key**) que estão associados no disparo do evento.

```
if 's3' in record:
    source_bucket = record['s3']['bucket']['name']
    key = record['s3']['object']['key']
```

Com esses dados referente ao **bucket** do **s3** e do **nome do arquivo da imagem** obtido, vamos criar uma **outra variável** referente **ao nome do novo arquivo** que será **da imagem redimensionada**.

```
if 's3' in record:
    source_bucket = record['s3']['bucket']['name']
    key = record['s3']['object']['key']
    thumb = f'thumb-{key}'
```

Agora vamos utilizar a biblioteca do **tempfile**, onde criaremos um **diretório temporário** para poder realizar o **download** da **imagem** do **bucket s3 de origem**, obtido no evento anteriormente, para podermos posteriormente realizar o **redimensionamento da imagem**!

```
with tempfile.TemporaryDirectory() as tempdir:
    download_path = os.path.join(tempdir, key)
    s3.download_path(source_bucket, key, download_path)
```

Veja que utilizamos a função **download_path()** do **objeto S3** para que possamos realizar o download em si da imagem nesse diretório temporário que acabamos de criar!

Após o download da imagem do **bucket S3** de **origem**, vamos realizar o processo de **redimensionar a imagem** que está nesse diretório temporário e consequentemente realizar o **upload** da **imagem** no **bucket S3** de **destino**!

Dito isso, vamos chamar uma função interna que iremos criar ainda (**generate_thumbnail()**), que será responsável por redimensionar a imagem em si e salvar essa nova imagem nesse mesmo diretório temporário que estamos trabalhando.

```
with tempfile.TemporaryDirectory() as tempdir:
    download_path = os.path.join(tempdir, key)
    s3.download_path(source_bucket, key, download_path)

    upload_path = os.path.join(tempdir, thumb)
    generate_thumbnail(download_path, upload_path)
```

Após gerar essa imagem nova redimensionada, **que iremos implementar ainda**, vamos realizar o upload enfim da imagem redimensionada no **bucket de destino s3**, e para isso utilizaremos a função **upload_file()** do objeto **s3_client**.

```
with tempfile.TemporaryDirectory() as tempdir:
    download_path = os.path.join(tempdir, key)
    s3.download_path(source_bucket, key, download_path)

    upload_path = os.path.join(tempdir, thumb)
    generate_thumbnail(download_path, upload_path)

    s3.upload_path(DEST_BUCKET, thumb, upload_path)
    print(f'Thumbnail image saved at {DEST_BUCKET}/{thumb}')
```

E para finalizar vamos implementar a função **generate_thumbnail()**, que como já foi citado acima, será responsável por **redimensionar a imagem** em si e **salvar** essa nova imagem nesse mesmo diretório temporário.

```
def generate_thumbnail(source_path, dest_path):
    print('Generating thumbnail from:', source_path)

    with Image.open(source_path) as image:
        image.thumbnail(SIZE)
        image.save(dest_path)
```

Veja que usamos a função **open()** da biblioteca **PIL**, para que possamos abrir a imagem e redimensionar o seu tamanho. E esse redimensionamento da imagem será feito através do método **thumbnail()**. E para finalizar e **salvar** em si essas alterações da imagem no diretório de destino, informado no parâmetro da função, usaremos o método **save()**.

E pronto!! Nosso script está criado com sucesso!!

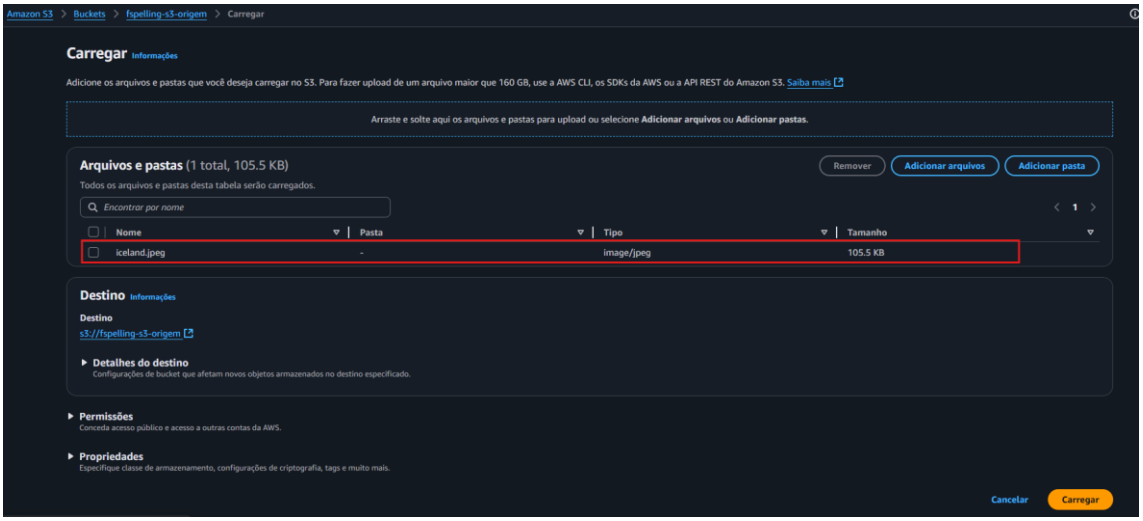
lambda_function.py

```python
import os
import tempfile
import boto3
from PIL import Image

SIZE = (128, 128)
DEST_BUCKET = os.environ['DEST_BUCKET']
s3_client = boto3.client('s3')


def lambda_handler(event, context):
    for record in event['Records']:
        if 's3' in record:
            source_bucket = record['s3']['bucket']['name']
            key = record['s3']['object']['key']
            thumb = f'thumb-{key}'

            with tempfile.TemporaryDirectory() as tempdir:
                download_path = os.path.join(tempdir, key)
                s3.download_path(source_bucket, key, download_path)

                upload_path = os.path.join(tempdir, thumb)
                generate_thumbnail(download_path, upload_path)

                s3.upload_path(DEST_BUCKET, thumb, upload_path)
                print(f'Thumbnail image saved at {DEST_BUCKET}/{thumb}')


def generate_thumbnail(source_path, dest_path):
    print('Generating thumbnail from:', source_path)

    with Image.open(source_path) as image:
        image.thumbnail(SIZE)
        image.save(dest_path)

```

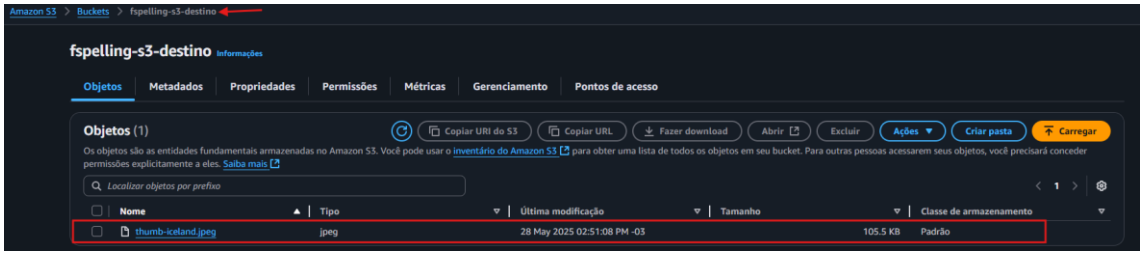
Realizando upload de imagens no S3

Enfim vamos testar a execução da nossa **função lambda**, onde fara o **redimensionamento** da imagem ao **subirmos** uma **imagem** do **bucket de origem do S3**!

Dito isso, vamos subir uma imagem “grande” e ver esse redimensionamento automático executado pela **função lambda**, graças a **trigger** associada aos **eventos** de objeto de criação do **S3**!



Após o upload da imagem no **bucket de origem do S3** veremos que foi gerado uma **thumb** dessa imagem no **bucket de destino do S3**!!



Logo sabemos que a nossa **função lambda** foi executada corretamente!!