escola britânica de artes criativas & tecnologia

---

# **Módulo** | Computação em Nuvem II

Caderno de **Aula**
Professor André Perez

---

# Tópicos

1. AWS Lambda;
2. AWS Step Functions;
3. AWS EventBridge.

---

# Aulas

## 1. AWS Lambda

O AWS Lambda é um serviço de computação orientado a evento e sem servidor. Permite a execução de uma função em diversas linguagens de programação que utilize entre 128 MB e 10 GB de memória RAM e que dure, no máximo, 15 minutos. É conhecido com FaaS (*function as a service*) ou função como serviço.

> O nome do serviço é inspirado em funções lambda do paradigma funcional.

### 1.1. Funcionamento

1. Ao criar uma função, o *runtime* (linguagem de programação + versão) é escolhida;
2. A função sempre recebe dois parametros: `event` (*input*) e `context` (execução);
3. Código refatorado precisa ser implantado (*deploy*);
4. Código pode ser testado;
5. Recursos (memória e tempo de execução) podem ser configurados;
6. Funções precisam de permissão para acessar outros serviços (*buckets*, tabelas, etc.) via AWS IAM;
7. Logs de execução são armazenados no AWS CloudWatch.

## 1.2. Preço

O AWS Lambda cobra por quantidade de execução, tempo de execução e memória alocada. Sobre a quantidade de execução, o preço é de 0,20 USD por milhão de execuções (1,11 BRL). Já sobre tempo de execução e memória alocada, o preço é de 0,0000166667 USD por GB por segundo (0.00009 BRL). Você sempre deve consultar o preço na página *web* do serviço (link).

## 1.3. Atividade

Extrair dados da site da B3 através de uma API:

In [ ]:

```
import json
from datetime import datetime

import requests

# -- setup

URL = 'https://www2.cetip.com.br/' +
      'ConsultarTaxaDi/ConsultarTaxaDICetip.aspx'

# -- extract

try:
  response = requests.get(URL)
  response.raise_for_status()
except Exception as exc:
  raise exc
else:
  data = json.loads(response.text)
  print(f'1 - {data}')

# -- transform

data['taxa'] = data['taxa'].replace(',', '.')
data['indice'] = data['indice'].
    replace('.', '').
    replace(',', '.')

data['dataTaxa'] = datetime.
    strptime(data['dataTaxa'], '%d/%m/%Y').
    strftime('%Y-%m-%d')

data['dataIndice'] = datetime.
    strptime(data['dataIndice'], '%d/%m/%Y').
    strftime('%Y-%m-%d')

data['dataReferencia'] = datetime.now().strftime('%Y-%m-%d')

data_csv = ','.join([v for v in data.values()])

print(f'2 - {data}')
print(f'3 - {data_csv}')
```

```
1 - {'taxa': '9,15', 'dataTaxa': '20/01/2022', 'indice': '33.871,84', 'data
Indice': '21/01/2022'}
2 - {'taxa': '9.15', 'dataTaxa': '2022-01-20', 'indice': '33871.84', 'dataI
ndice': '2022-01-21', 'dataReferencia': '2022-01-21'}
3 - 9.15,2022-01-20,33871.84,2022-01-21,2022-01-21
```

Vamos dividir essa etapa em duas: extração e transformação. Logo, temos que:

- Criar um *bucket* no AWS S3 para salvar o dado original ( `bronze` );
- Criar uma função AWS Lambda para extrair o dado original;
- Criar um *bucket* no AWS S3 para salvar o dado transformado ( `silver` );
- Criar uma função AWS Lambda para transformar o dado original;
- Criar uma função AWS Lambda para criar uma tabela no AWS Athena apontando para o *bucket* do dado transformado.

Vamos também usar o pacote Python boto3, o SDK (*software development kit*) da AWS para interação com os serviços da plataforma. A documentação pode ser encontrada neste link. Exemplo:

```
import boto3

client = boto3.client('s3')
client.upload_file(Filename='<nome-do-arquivo>', Bucket='<nome-do-bucket>', Key='<nome-do-objeto>')

client = boto3.client('athena')
client.start_query_execution(
  QueryString='SELECT * FROM <nome-da-tabela> LIMIT 10',
  ResultConfiguration={'OutputLocation': 's3://<nome-do-bucket-de-resultados>/'}
)
```

AWS Lambda para *bucket* **bronze**:

In [ ]:

```
import json
import logging
from datetime import datetime

import boto3
import urllib3
from botocore.exceptions import ClientError


def lambda_handler(event, context) -> bool:

  # -- setup

URL = 'https://www2.cetip.com.br/' +
      'ConsultarTaxaDi/ConsultarTaxaDICetip.aspx'
  BRONZE_BUCKET = 'modulo-39-ebac-bronze'

  client = boto3.client('s3')

  date = datetime.now().strftime('%Y-%m-%d')
  filename_json = f'stock-exchange-{date}.json'

  # -- extract

  try:
    http = urllib3.PoolManager()
    response = http.request(url=URL, method='get')
  except Exception as exc:
    raise exc
  else:
    data = json.loads(response.data.decode())
    logging.info(msg=data)

  # -- transform

  ...

  # -- load

  try:
      with open(
          f'/tmp/{filename_json}',
          mode='w',
          encoding='utf8'
      ) as fp:
          json.dump(data, fp)
      client.upload_file(
          Filename=f'/tmp/{filename_json}',
          Bucket=BRONZE_BUCKET,
          Key=filename_json
      )
  except ClientError as exc:
      raise exc

  return json.dumps(dict(status=True))
```

AWS Lambda para *bucket* **silver**:

In [ ]:

```
import json
```

```
from datetime import datetime

import boto3
from botocore.exceptions import ClientError


def lambda_handler(event, context) -> bool:

  # -- setup

  BRONZE_BUCKET = 'modulo-39-ebac-bronze'
  SILVER_BUCKET = 'modulo-39-ebac-silver'

  client = boto3.client('s3')

  date = datetime.now().strftime('%Y-%m-%d')
  filename_csv = f'stock-exchange-{date}.csv'
  filename_json = f'stock-exchange-{date}.json'

  # -- extract

  client.download_file(
      BRONZE_BUCKET,
      filename_json,
      f'/tmp/{filename_json}'
  )

  with open(
      f"/tmp/{filename_json}",
      mode='r',
      encoding='utf8'
  ) as fp:
      data = json.load(fp)

  # -- transform

  data['taxa'] = data['taxa'].
    replace(',', '.')
  data['indice'] = data['indice'].
    replace('.', '').
    replace(',', '.')

  data['dataTaxa'] = datetime.
    strptime(data['dataTaxa'], '%d/%m/%Y').
    strftime('%Y-%m-%d')

  data['dataIndice'] = datetime.
    strptime(data['dataIndice'], '%d/%m/%Y').
    strftime('%Y-%m-%d')

  # -- load

  try:
      with open(
          f'/tmp/{filename_csv}',
          mode='w',
          encoding='utf8'
      ) as fp:
          fp.write(','.join([v for v in data.values()]))
      client.upload_file(
```

```
            Filename=f'/tmp/{filename_csv}',
            Bucket=SILVER_BUCKET,
            Key=f'data_referencia={date}/{filename_csv}'
        )
    except ClientError as exc:
        raise exc

    return json.dumps(dict(status=True))
```

AWS Lambda para tabela:

In [ ]:

```
import json
from datetime import datetime

import boto3
from botocore.exceptions import ClientError


def lambda_handler(event, context) -> bool:

  # -- setup

  SILVER_BUCKET = 'modulo-39-ebac-silver'

  query = f"""
  CREATE EXTERNAL TABLE IF NOT EXISTS cdi (
    taxa double,
    data_taxa string,
    indice double,
    data_indice string
  )
  PARTITIONED BY (
    data_referencia string
  )
  ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
  WITH SERDEPROPERTIES ('separatorChar'=',')
  LOCATION 's3://{SILVER_BUCKET}/'
  """

  client = boto3.client('athena')

  # -- create

  try:
    client.start_query_execution(
      QueryString=query,
      ResultConfiguration={
          'OutputLocation': 's3://modulo-38-ebac-athena-results/'
      }
    )
  except ClientError as exc:
    raise exc

  # -- update

  try:
    client.start_query_execution(
      QueryString='MSCK REPAIR TABLE cdi',
      ResultConfiguration={
          'OutputLocation': 's3://modulo-38-ebac-athena-results/'
      }
    )
  except ClientError as exc:
    raise exc

  return json.dumps(dict(status=True))
```

## 2. AWS Step Functions

O AWS Step Functions é um serviço de orquestração de serviços sem servidor. No contexto de dados, pode atuar como orquestrador de *pipeline* de dados. O serviço é inspirado no famoso projeto *open source* Apache Airflow.

## 2.1. Funcionamento

1. Uma *task* ou tarefa é uma unidade de processamento;
2. Uma *state machine* é a definição de um fluxo de tarefas;
3. Tasks e state machine são definidas com a linguagem JSON.

## 2.2. Preço

O AWS Step Functions cobra por transição de estado. O preço é de 0,0375 USD por 1.000 transições de estado (0,21 BRL). Você sempre deve consultar o preço na página *web* do serviço (link).

## 2.3. Atividade

- Criar uma `state machine` com três *tasks*, uma com cada função lambda criada na item 1.3.

# 3. AWS EventBridge

O AWS EventBridge é um serviço de barramento de eventos sem servidor. Ele permite que multiplos serviços da AWS se comuniquem de maneira assíncrona através da publicação/consumo de eventos.

## 3.1. Funcionamento

1. Uma *rule* ou regra "escuta" eventos e os roteia para serviços da AWS;
2. Eventos podem ser temporais ou publicações de outros serviços da AWS;
3. Eventos também podem se publicados serviços externos (Shopfy, Zendesk, etc.).

## 3.2. Preço

O AWS EventBridge é gratuito para publicação/consumo de eventos entre serviços da AWS. Você sempre deve consultar o preço na página web do serviço (link).

## 3.3. Atividade

- Criar um evento que inicie a *state machine* do item 2.3.