Este é um projeto de estudo usando o serviço AWS Kineses Stream e Kineses Firehose. O arquivo producer.ipynb simula a criação de dados em json com a API randomuser e campos gerados aleatoriamente. E o consumer.ipynb simula um consumidor desses dados

Documentação:

Lib requests foi usada para dar acesso a chamadas http para aquisição dos dados da api randomuser

Producer

Lib random para gerar o dado “points” aleatoriamente

Lib time para limitar a quantidade de requisições durante o while true

Lib boto3 para acessar o client da aws

Lib json para enviar os dados formatados em json para os servidores do kinesis

Lib os para aumentar a segurança do arquivo evitando colocar as chaves de api direto no código, mas sim no proprio sistema operacional

Client cria a variavel que acessa o kinesis com as devidas autorizações

Get\_data() faz a coleta dos dados da API random user com alguns tratamentos e no final envia esses dados para a função points\_data()

Points\_data() construi um json com os dados coletados e envia para o kinesis com o Client.put\_records()

Consumer

Shard cria uma instancia da shard selecionada, nesse caso aquela com id = “shardId-000000000002”

Breve explicação de como funciona o Kinesis

Uma stream é dividida em varios fragmentos, as shards. Essa divisao facilita tanto na hora do processamento quanto na aquisição dos dados visto que os dados vão estar paticionados em várias partes. Mas, para facilitar o gerenciamento das shard pode-se usar as keys, elas fazem com que um dado com uma key X sempre aponte para uma shard X. Nesse projeto usei com key = 2

ShardIteratorType serve para configurar a forma de como os dados nas shards vao ser lido. No projeto usei LATEST. Pois, só era interessante os dados mais novos

Parametro ’NextShardIterator’ evita que a shard retorne valores já coletados

Kinesis Firehose

O Firehose da uma alternativa para persistir os dados gerados, ja que depois de 24 horas tudo enviado pelo Kinesis Stream é deletado. Usando essa ferramenta com com o um bucket S3 é possivel fazer a coleta e armazenamento permanente dos dados. Para isso não é necessario nenhuma linha de programação. Apenas um bucket, a stream e um fluxo de entrega do Firehose. O arquivo é salvo a cada X segundos sendo no minimo 60 e maximo 900.

Esse e o arquivo gerado convertido em txt durante o periodo de 60 segundos

This is a study project with AWS Kinesis Stream and Kinesis Firehose. The file producer.ipynb simulate a data creation in json using randomuser API with some random generated parameters. The consumer.ipynb simulate a data consumer.

Documentation

Lib requests was used to let http calls and get datas from randomuser API

Lib random was used to generate a random data to “points”

Lib time limits the quantity the number of requisitions during while true

Lib boto3 lib that provides access to AWS client

Lib json lets send the json formatted datas to kinesis

Lib os lets use enviroment variable in the app increasing the security by don’t using api keys directly on the code

Client is a variable that have access to kinesis with the authorizations

Get\_data() gather the datas from randomuser API with some changes and send to points\_data() function

Points\_data() build a json with gathered datas and send to kinesis with client.put\_records()

Consumer

Shard creates an instance with selected shard, in the case that one with id = “shardId-000000000002”

Short explanation about Kinesis

A Stream is divided by multiple shards. This division let the processing and acquisition a little easier. But to manage this division kinesis use keys to tell to the app which shard get the data. In this project I choose key = 2.

ShardIteratorType is used to manage how the data in the shard will be organized. In this case I used LATEST. Cause just the newest data is important

’NextShardIterator’ parameter Dont let shard returns data already returned

Kinesis Firehose

Firehose is an alternative to persist generated data. In Kinesis all data have 24 hours to expire, after this time them will be deleted. With Firehose you can simply connect a stream with a S3 bucket and the data will be stored. The file is saved every X seconds, minimum 60 and maximum 900 secinds