Streaming de dados com Apache Kafka e Spark Streaming

# Introdução ao Apache Kafka

Em ambientes de TI, é comum haver sistemas que precisam se comunicar entre si para trocar informações com o forte desafio de garantir que essas informações cheguem de forma confiável e eficiente ao seu destino final.

Para resolver esse problema existem diversas ferramentas de mensageria, entre elas o Apache Kafka, que é uma plataforma distribuída de streaming de dados. O Kafka é muito utilizado em arquiteturas de microsserviços e big data, onde é necessário processar grandes volumes de dados em tempo real.

# Funcionamento do Apache Kafka

## Apache Kafka

(Fonte: https://lankydan.dev/intro-to-kafka-ordering-related-records)

O Kafka é uma plataforma de streaming de dados distribuída, que funciona como um intermediário entre sistemas produtores e consumidores de informações. Ele é baseado em tópicos, que são canais de comunicação onde os dados são enviados e recebidos. Um tópico é dividido em partições, que são armazenadas em vários servidores. Cada partição é replicada em um número configurável de servidores para garantir alta disponibilidade e tolerância a falhas.

Os produtores de informações enviam mensagens para um tópico, que são armazenadas em uma partição e são entregues a um ou mais consumidores que se inscrevem no tópico. O Kafka, por usa vez, garante que as mensagens são entregues na ordem em que foram produzidas em cada partição.

O Kafka tem um modelo de pub/sub (publicação/assinatura), que significa que os consumidores se inscrevem em tópicos específicos e recebem as mensagens que são publicadas nesses tópicos. Isso permite que vários consumidores possam ler as mensagens do mesmo tópico, com cada um mantendo seu próprio offset (posição) na partição.

## Apache Kafka no Databricks

Nesta seção, será apresentado o funcionamento prático do Apache Kafka utilizando o Databricks.

Em um novo notebook, o primeiro passo é instalar o Kafka no ambiente do Databricks com o comando de download dos arquivos de instalação. Na exemplo em questão, a versão 3.5.1 do Kafka será utilizada:

%sh

sudo wget https://downloads.apache.org/kafka/3.5.1/kafka\_2.12-3.5.1.tgz

Esse processo pode levar algum tempo, e o primeiro resultado será apresentado da seguinte maneira:

zookeeper\_1

Após isso, com o comando a seguir, o arquivo é inicializado:

%sh

tar -xvf kafka\_2.12-3.5.1.tgz

zookeeper\_2

Com todo esse processo concluído, é necessário executar e manter o Zookeeper, um serviço de coordenação e gerenciamento fundamental para o funcionamento do Kafka, em execução.

%sh

./kafka\_2.12-3.5.1/bin/zookeeper-server-start.sh ./kafka\_2.12-3.5.1/config/zookeeper.properties

zookeeper\_3

Em outro notebook, é preciso executar o servidor Kafka e também mantê-lo em execução:

%sh

./kafka\_2.12-3.5.1/bin/kafka-server-start.sh ./kafka\_2.12-3.5.1/config/server.properties

server

Com o Zookeeper e o servidor em execução, um novo tópico com o nome "primeiro\_topico" será criado em outro notebook:

%sh

./kafka\_2.12-3.5.1/bin/kafka-topics.sh --bootstrap-server localhost:9092 --create --topic primeiro\_topico --partitions 1 --replication-factor 1

create\_topic\_1

Após a criação do primeiro tópico, todo o conteúdo será deletado e, em seguida, o console do Kafka será iniciado para imprimir todo o conteúdo do tópico em tempo real:

%sh

./kafka\_2.12-3.5.1/bin/kafka-topics.sh --bootstrap-server localhost:9092 --delete --topic primeiro\_topico

%sh

./kafka\_2.12-3.5.1/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic primeiro\_topico --from-beginning

create\_topic\_2

Agora será criado um produtor para produzir conteúdo para o tópico "primeiro\_topico" a partir da API do Kafka para Python. O primeiro passo é instalar o "kafka-python" com o comando "pip install":

%sh

pip install --upgrade pip

pip install kafka-python

Depois é necessário importar o "KafkaProducer" da biblioteca do Kafka, bem como a biblioteca para trabalhar com JSON para criar o conteúdo na estrutura esperada pelo Kafka:

from kafka import KafkaProducer

import json

producer\_1

Para criar o produtor, basta chamar a função "KafkaProducer" com o endereço do servidor local e a porta 9092, que é a padrão. A mensagem a ser criada será um JSON com três campos: "id", "nome" e "idade". A mensagem a ser enviada é previamente definida como um dicionário Python e, posteriormente, convertida em uma string JSON formatada por meio da função "json.dumps". Em seguida, essa string é codificada em bytes, utilizando o formato de codificação UTF-8. Esse processo é especialmente útil quando se deseja transmitir dados em formato JSON pela rede ou armazená-los em um arquivo, pois muitas vezes é necessário trabalhar com bytes em vez de strings. Por fim, basta utilizar a função "producer.send" para enviar a mensagem do produtor para o tópico "primeiro\_topico":

producer = KafkaProducer(bootstrap\_servers=["localhost:9092"])

topic = "primeiro\_topico"

json\_data = {"id":1, "nome": "Matheus", "idade": 27}

mensagem = json.dumps(json\_data).encode('utf-8')

producer.send(topic, mensagem)

Ao executar este breve segmento de código, o resultado é imediatamente exibido no console do Kafka:

topic\_res\_1

Ao enviar uma nova mensagem utilizando o mesmo código, mas com um conteúdo diferente, o console também irá exibir essa nova mensagem:

topic\_res\_2

Além de visualizar o conteúdo de um topico a partir do console do Kafka, também é possível consumir o conteúdo de um produtor a partir da criação de um consumidor. O primeiro passo é criar um consumidor também utilizando a API do Python em outro notebook:

%sh

pip install --upgrade pip

pip install kafka-python

Para criar o consumidor, é simplesmente necessário empregar a função "KafkaConsumer", fornecendo o nome do tópico e o endereço do servidor. A fim de visualizar o conteúdo das mensagens recebidas pelo consumidor, um "loop for" será utilizado, permanecendo em execução com uma instrução "print" para manter a exibição dos conteúdos de forma contínua:

from kafka import KafkaConsumer

consumer = KafkaConsumer("primeiro\_topico", bootstrap\_servers=["localhost:9092"])

for message in consumer:

print(message)

O resultado é exibido da seguinte maneira:

consumer\_0 consumer\_1 consumer\_2

Com a criação do consumidor, é importante ressaltar que o console do Kafka utilizado anteriormente não deixará de receber as mensagens:

topic\_res\_after\_consumer

Introdução ao Spark Streaming

O Apache Spark Streaming é um sistema de processamento de streaming escalável e tolerante a falhas que suporta nativamente cargas de trabalho em lote e streaming. O Spark Streaming é uma extensão da API principal do Spark que permite que engenheiros cientistas de dados processem dados em tempo real de várias fontes diferentes, como o Apache Kafka, Apache Flume ou o Amazon Kineses.

O Spark Streaming fornece uma abstração de alto nível chamada fluxo discretizado ou DStream, que representa um fluxo contínuo de dados. Os DStreams podem ser criados a partir de fluxos de dados de entrada de outras fontes, ou aplicando operações de alto nível em outros DStreams. Internamente, um DStream é representado como uma sequência de RDDs.

Em resumo, o spark streaming divide uma entrada de dados em pequenos batchs e fornece uma enorme e poderosa capacidade de processamento com latência de até um segundo.

Spark Streaming no Databricks

O Spark Streaming será demonstrado na prática no ambiente Databricks a seguir, utilizando o PySpark e a função "readStream".

O primeiro exemplo consistirá em configurar um fluxo para consumir dados de um tópico do Kafka denominado "primeiro\_topico". A função "readStream" é inicializada com o formato "kafka" e as opções necessárias, como o nome do tópico e o endereço do servidor. Os resultados do fluxo são armazenados no DataFrame Spark "df". A instrução "df.printSchema()" exibe a estrutura do esquema do DataFrame, incluindo o nome e o tipo das colunas. Em seguida, um novo DataFrame "df\_res" é criado, contendo o conteúdo JSON das mensagens do tópico, seguindo um esquema estrutural predefinido pela variável "schema". Por fim, o comando "display" é utilizado para apresentar em tempo real os resultados do tópico.

from pyspark.sql.functions import col, from\_json

from pyspark.sql.types import IntegerType, StringType, StructType

df = spark.readStream \

.format("kafka") \

.option("kafka.bootstrap.servers", "localhost:9092") \

.option("subscribe", "primeiro\_topico") \

.option("startingOffsets", "earliest") \

.load()

df.printSchema()

schema = StructType() \

.add("id", IntegerType()) \

.add("nome", StringType()) \

.add("idade", IntegerType())

df\_res = df.selectExpr("CAST(value AS STRING)") \

.select(from\_json(col("value"), schema).alias("data")) \

.select("data.\*")

display(df\_res)

Os resultados do streaming é apresentado abaixo:

spark\_streaming\_0 spark\_streaming\_1

No segundo exemplo, um fluxo de streaming é desenvolvido para consumir as mensagens de um tópico Kafka denominado "primeiro\_topico". Uma nova coluna de informação, chamada "nova informação", é adicionada e o resultado é armazenado em outro tópico Kafka denominado "topico\_resultado".

O primeiro passo é criar esse novo tópico denominado "topico\_resultado":

%sh

./kafka\_2.12-3.5.1/bin/kafka-topics.sh --bootstrap-server localhost:9092 --create --topic topico\_resultado --partitions 1 --replication-factor 1

%sh

./kafka\_2.12-3.5.1/bin/kafka-topics.sh --bootstrap-server localhost:9092 --delete --topic topico\_resultado

%sh

./kafka\_2.12-3.5.1/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic topico\_resultado --from-beginning

create\_topic\_res\_1 create\_topic\_res\_2

A construção do fluxo de streaming, utilizando a função "readStream" a partir do tópico "primeiro\_topico", segue a mesma abordagem do exemplo anterior. No entanto, a diferença está na etapa seguinte: em vez de exibir o resultado com o "display", uma nova coluna chamada "nova\_informacao" é acrescentada usando a função "withColumn" do PySpark. Além disso, a função "writeStream" é utilizada para armazenar o resultado final no tópico Kafka denominado "topico\_resultado".

from pyspark.sql.functions import col, from\_json, lit

from pyspark.sql.types import IntegerType, StringType, StructType

df = spark.readStream \

.format("kafka") \

.option("kafka.bootstrap.servers", "localhost:9092") \

.option("subscribe", "primeiro\_topico") \

.option("startingOffsets", "earliest") \

.load()

df.printSchema()

schema = StructType() \

.add("id", IntegerType()) \

.add("nome", StringType()) \

.add("idade", IntegerType())

df\_res = df.selectExpr("CAST(value AS STRING)") \

.select(from\_json(col("value"), schema).alias("data")) \

.select("data.\*")

df\_res = df\_res.withColumn("nova\_informacao", lit("nova\_informacao"))

df\_res.selectExpr("CAST(id AS STRING) AS key", "to\_json(struct(\*)) AS value") \

.writeStream \

.format("kafka") \

.outputMode("append") \

.option("kafka.bootstrap.servers", "localhost:9092") \

.option("checkpointLocation", "/FileStore/topico\_resultado/\_checkpoint") \

.option("topic", "topico\_resultado") \

.start() \

.awaitTermination()

Os resultados do comando com o "writeStream" podem ser validados no console do "topico\_resultado":

create\_topic\_res\_3 create\_topic\_res\_4

No terceiro e último exemplo de utilização do Spark Streaming, os dados consumidos do tópico Kafka "primeiro\_topico" são processados de forma diferente: em vez de serem salvos em outro tópico Kafka, os resultados são armazenados em arquivos no formato Parquet. Arquivos Parquet são um formato de armazenamento de dados colunar altamente eficiente, que é amplamente utilizado para armazenar dados estruturados em Big Data e Análise de Dados, devido à sua compactação, suporte a esquemas complexos e compatibilidade com diversas ferramentas de processamento de dados.

A única alteração no código para este exemplo com arquivos Parquet está na configuração da função "writeStream". Em vez de utilizar o formato Kafka com parâmetros como tópico e servidor, o formato Parquet é adotado, especificando a localização dos arquivos Parquet. A localização escolhida é um diretório disponível para armazenar dados no próprio ambiente do Databricks:

from pyspark.sql.functions import col, from\_json, lit

from pyspark.sql.types import IntegerType, StringType, StructType

df = spark.readStream \

.format("kafka") \

.option("kafka.bootstrap.servers", "localhost:9092") \

.option("subscribe", "primeiro\_topico") \

.option("startingOffsets", "earliest") \

.load()

df.printSchema()

schema = StructType() \

.add("id", IntegerType()) \

.add("nome", StringType()) \

.add("idade", IntegerType())

df\_res = df.selectExpr("CAST(value AS STRING)") \

.select(from\_json(col("value"), schema).alias("data")) \

.select("data.\*")

df\_res = df\_res.withColumn("nova\_informacao", lit("nova\_informacao"))

df\_res.selectExpr("CAST(id AS STRING) AS key", "to\_json(struct(\*)) AS value") \

.writeStream \

.format("parquet") \

.option("path", "/FileStore/streaming\_kafka/data") \

.option("checkpointLocation", "/FileStore/streaming\_kafka/checkpoint") \

.start() \

.awaitTermination()

O resultado é validado em outro notebook por meio do seguinte comando, que lista todos os arquivos no diretório escolhido para o streaming.

dbutils.fs.ls("/FileStore/streaming\_kafka/data")

Além disso, a validação é realizada ao ler todos os arquivos Parquet desse diretório e depois imprime os resultados correspondentes.

file\_location = "/FileStore/streaming\_kafka/data"

file\_type = "parquet"

df\_spark = spark.read.format(file\_type).load(file\_location)

display(df\_spark)

Os primeiros resultados são apresentados abaixo:

parquet\_1 parquet\_2 parquet\_3

Caso o produtor do "primeiro\_topico" adicione mais registros, é possível observar a criação de novos arquivos Parquet no diretório:

parquet\_4 parquet\_5

É importante destacar que, mesmo com a execução contínua das aplicações de streaming, os consumidores dos dados do produtor continuam operando sem qualquer interrupção:

final\_result final\_result\_2

Referências

Documentação oficial do Apache Kafka

Tutorialspoint - Apache Kafka Tutorial

Kafka: The Definitive Guide, de Neha Narkhede, Gwen Shapira e Todd Palino

A documentação da biblioteca Python para o Kafka

Hands-On Kafka, de Arun Kumar Mishra

Apache Kafka gerenciado pela AWS

Artigo What to consider for painless Apache Kafka integration

Artigo Intro to Kafka - Ordering related records

Spark Streaming

Streaming programming guide

Structured Streaming - Definição

Structured Streaming - Documentação