02-streaming-kafka-src-dst-mongodb.ipynb

Vamos começar a análise do notebook.

O notebook que você forneceu parece estar configurado para realizar uma tarefa de streaming de dados utilizando Apache Spark, Kafka e MongoDB. Vou explicar cada etapa do código conforme observado nas células iniciais.

**1. Configuração da Sessão Spark**

python

Copiar código

from pyspark.sql import SparkSession

from pyspark.sql.types import MapType, StringType

from pyspark.sql.functions import from\_json

# Configure Spark session

spark = (SparkSession

.builder

.master('local')

.appName('kafka-mongo-streaming')

# Add Kafka and MongoDB packages. Ensure they are in the same string!

.config("spark.jars.packages",

"org.apache.spark:spark-sql-kafka-0-10\_2.12:3.3.1,"

"org.mongodb.spark:mongo-spark-connector\_2.12:3.0.2")

# MongoDB configuration, including username and password from Compose file

.config("spark.mongodb.input.uri",

"mongodb://root:example@mongo:27017/docstreaming.invoices?authSource=admin")

.config("spark.mongodb.output.uri",

"mongodb://root:example@mongo:27017/docstreaming.invoices?authSource=admin")

.getOrCreate())

# Spark Context

sc = spark.sparkContext

**O que está sendo feito:**

* **Importação de Bibliotecas**: A primeira parte do código importa as bibliotecas necessárias para trabalhar com Spark, incluindo tipos de dados e funções para manipular JSON.
* **Configuração da Sessão Spark**: A sessão Spark é configurada com o nome kafka-mongo-streaming, que será usada para processar streams de dados. Dois pacotes essenciais são adicionados:
  + **Kafka**: Para ler e escrever dados em tópicos Kafka.
  + **MongoDB**: Para ler e gravar dados no MongoDB.
* **Configuração do MongoDB**: As URIs de entrada e saída para o MongoDB são configuradas. Estes apontam para a base de dados docstreaming na coleção invoices, com autenticação configurada.

**2. Leitura de Mensagens do Kafka**

python

Copiar código

# Read the message from the kafka stream

df = spark \

.readStream \

.format("kafka") \

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

.option("subscribe", "ingestion-topic") \

.load()

# convert the binary values to string

df1 = df.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)")

**O que está sendo feito:**

* **Leitura de Stream Kafka**: Este código lê mensagens de um tópico Kafka (ingestion-topic) usando a API de streaming do Spark.
* **Conversão de Dados**: As mensagens Kafka são lidas como binários, então aqui é feita a conversão das chaves (key) e valores (value) para strings.

**3. Criação de uma View Temporária**

python

Copiar código

#Create a temporary view for SparkSQL

df1.createOrReplaceTempView("message")

**O que está sendo feito:**

* **View Temporária**: Uma view temporária chamada message é criada, permitindo que você utilize SQL para consultar o DataFrame resultante.

**4. Exibição dos Dados no Console**

python

Copiar código

# Write out the message to the console of the environment

res = spark.sql("SELECT \* from message")

res.writeStream.format("console") \

.outputMode("append") \

.start()

**O que está sendo feito:**

* **Consulta SQL**: Uma consulta SQL simples é executada para selecionar todas as colunas da view message.
* **Exibição no Console**: Os resultados são escritos no console em modo de saída append, que exibe novos dados conforme eles chegam.

**5. Escrita de Dados de Volta no Kafka**

python

Copiar código

# Write the unvonverted dataframe (no strings)

# message back into Kafka in another topic

ds = df \

.writeStream \

.format("kafka") \

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

.option("topic", "spark-output") \

.option("checkpointLocation", "/tmp") \

.start()

**O que está sendo feito:**

* **Escrita no Kafka**: O DataFrame original (df), antes da conversão de binário para string, é escrito de volta em um novo tópico Kafka chamado spark-output. Isso cria um ciclo onde as mensagens processadas podem ser consumidas por outro sistema.
* **Checkpointing**: Um local de checkpoint (/tmp) é configurado para garantir a tolerância a falhas, permitindo que o processamento continue de onde parou em caso de interrupção.

**Resumo do Projeto**

Este projeto é um pipeline de streaming de dados que:

1. **Lê mensagens de um tópico Kafka**.
2. **Processa e converte as mensagens para um formato legível**.
3. **Armazena as mensagens em uma view temporária para consultas SQL**.
4. **Exibe os dados no console para monitoramento**.
5. **Escreve os dados de volta em um novo tópico Kafka** para serem consumidos novamente.

Esse tipo de fluxo é típico em arquiteturas de dados em tempo real, onde dados são constantemente ingeridos, processados, e redistribuídos para diferentes sistemas. ​