

Users Streaming Data Ingestion Using Spark

1. installed confluent kafka & airflow locally following the doc notes. Opened both kafka & airflow dashboard on browser locally
2. we were publishing data to kafka topic using the airflow script

```
9
10 def stream_data():
11     import json
12     from kafka import KafkaProducer
13     import time
14     import logging
15
16     #create producer object
17     producer = KafkaProducer(bootstrap_servers=['broker:29092'], max_block_ms=5000)
18     curr_time = time.time()
19
20     while True:
21         if time.time() > curr_time + 300: #1 minute
22             break
23         try:
24             res = get_data()
25             res = format_data(res)
26
27             producer.send('users_data', json.dumps(res).encode('utf-8'))
28             print("Record Inserted !!")
29         except Exception as e:
30             logging.error(f'An error occured: {e}')
31             continue
32
33
34 with DAG('user_automation',
35         default_args=default_args,
36         schedule_interval='@daily',
37         catchup=False) as dag:
38
39     streaming_task = PythonOperator(
40         task_id='stream_data_from_api',
41         python_callable=stream_data
42     )
```

- Checked both dag file logs and kafka topic messages:

The screenshot displays the Apache Airflow web interface. At the top, the navigation bar includes the Airflow logo, tabs for DAGs, Cluster Activity, Datasets, Security, Browse, Admin, and Docs, and a clock showing 04:20 UTC. Below the navigation bar is a status bar with various task states: deferred, failed, queued, removed, restarting, running, scheduled, shutdown, skipped, success, up_for_reschedule, up_for_retry, upstream_failed, and no_status.

The main content area shows a DAG named 'user_automation' with a task named 'stream_data_from_api'. The task is currently in a 'running' state. The interface includes buttons for 'Clear task', 'Mark state as...', and 'Filter Tasks'. Below these buttons are tabs for 'Details', 'Graph', 'Gantt', 'Code', and 'Logs'. The 'Logs' tab is selected, showing a list of log entries for the task. The log entries are filtered by 'All Levels' and 'All File Sources'. The log content shows the task's execution details, including the task's name, the DAG's name, and the task's status. The log entries are as follows:

```
[2024-01-27, 04:19:57 UTC] {standard_task_runner.py:84} INFO - Running: ['***', 'tasks', 'run', 'user_automation', 'stream_data_from_api', 'scheduled__2024-01-26T00:00:00Z', '2024-01-27, 04:19:57 UTC']
[2024-01-27, 04:19:57 UTC] {standard_task_runner.py:85} INFO - Job 2: Subtask stream_data_from_api
[2024-01-27, 04:19:57 UTC] {task_command.py:415} INFO - Running <TaskInstance: user_automation.stream_data_from_api scheduled__2024-01-26T00:00:00Z__2024-01-27, 04:19:57 UTC> on host
[2024-01-27, 04:19:58 UTC] {taskinstance.py:1660} INFO - Exporting env vars: AIRFLOW_CTX_DAG_OWNER='growdatakills' AIRFLOW_CTX_DAG_ID='user_automation' AIRFLOW_CTX_TASK_ID='stream_data_from_api'
[2024-01-27, 04:19:58 UTC] {kafka_stream.py:45} INFO - Inside Stream Function
[2024-01-27, 04:19:58 UTC] {conn.py:380} INFO - <BrokerConnection node_id=bootstrap-0 host=broker:29092 <connecting> [IPv4 ('172.19.0.8', 29092)]: connecting to broker
[2024-01-27, 04:19:58 UTC] {conn.py:1205} INFO - Probing node bootstrap-0 broker version
[2024-01-27, 04:19:58 UTC] {conn.py:410} INFO - <BrokerConnection node_id=bootstrap-0 host=broker:29092 <connecting> [IPv4 ('172.19.0.8', 29092)]: Connection complete.
[2024-01-27, 04:19:58 UTC] {conn.py:1267} INFO - Broker version identified as 2.5.0
[2024-01-27, 04:19:58 UTC] {conn.py:1268} INFO - Set configuration api_version=(2, 5, 0) to skip auto check_version requests on startup
[2024-01-27, 04:19:58 UTC] {kafka_stream.py:51} INFO - Inside While Loop
[2024-01-27, 04:20:00 UTC] {kafka_stream.py:59} INFO - Record Published
[2024-01-27, 04:20:00 UTC] {conn.py:380} INFO - <BrokerConnection node_id=1 host=broker:29092 <connecting> [IPv4 ('172.19.0.8', 29092)]: connecting to broker:29092 [IPv4 ('172.19.0.8', 29092)]: Connection complete.
[2024-01-27, 04:20:00 UTC] {kafka_stream.py:51} INFO - Inside While Loop
[2024-01-27, 04:20:00 UTC] {conn.py:410} INFO - <BrokerConnection node_id=1 host=broker:29092 <connecting> [IPv4 ('172.19.0.8', 29092)]: Connection complete.
[2024-01-27, 04:20:00 UTC] {conn.py:919} INFO - <BrokerConnection node_id=bootstrap-0 host=broker:29092 <connected> [IPv4 ('172.19.0.8', 29092)]: Closing connection.
[2024-01-27, 04:20:00 UTC] {kafka_stream.py:59} INFO - Record Published
[2024-01-27, 04:20:00 UTC] {kafka_stream.py:51} INFO - Inside While Loop
[2024-01-27, 04:20:01 UTC] {kafka_stream.py:59} INFO - Record Published
[2024-01-27, 04:20:01 UTC] {kafka_stream.py:51} INFO - Inside While Loop
[2024-01-27, 04:20:02 UTC] {kafka_stream.py:59} INFO - Record Published
[2024-01-27, 04:20:02 UTC] {kafka_stream.py:51} INFO - Inside While Loop
[2024-01-27, 04:20:03 UTC] {kafka_stream.py:59} INFO - Record Published
[2024-01-27, 04:20:03 UTC] {kafka_stream.py:51} INFO - Inside While Loop
```

4. Next the data was streaming from kafka and ingesting into cassandra. For this I wrote a `spark_stream` script where first I established a spark connection. Inside spark connection we included all the jar files using which spark can communicate with both kafka and cassandra. Used jar files based on our pyspark version.

```
def create_spark_connection():
    s_conn = None

    try:
        print("Creating spark session!")
        print("*****")
        s_conn = SparkSession.builder \
            .appName('SparkDataStreaming') \
            .config('spark.jars.packages', "com.datastax.spark:spark-cassandra-connector_2.12:3.4.0,"
                                             "org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.0") \
            .config('spark.cassandra.connection.host', 'localhost') \
            .getOrCreate()

        s_conn.sparkContext.setLogLevel("ERROR")
        print("Spark connection created successfully!")
        print("*****")
    except Exception as e:
        print(f"Couldn't create the spark session due to exception {e}")
        print("*****")

    return s_conn
```

5. Once spark connection is done, next we created dataframe by connecting & reading data from kafka topic. So we were getting data in the form of key value pair where we were using custom schema to get a clean row columnar based dataframe.

```
def connect_to_kafka(spark_conn):
    spark_df = None
    try:
        spark_df = spark_conn.readStream \
            .format('kafka') \
            .option('kafka.bootstrap.servers', 'localhost:9092') \
            .option('subscribe', 'users_data') \
            .option('startingOffsets', 'latest') \
            .load()

        print("kafka dataframe created successfully")
        print("*****")
    except Exception as e:
        print(f"kafka dataframe could not be created because: {e}")
        print("*****")

    return spark_df

def create_selection_df_from_kafka(spark_df):
    schema = StructType([
        StructField("id", StringType(), False),
        StructField("first_name", StringType(), False),
        StructField("last_name", StringType(), False),
        StructField("gender", StringType(), False),
        StructField("address", StringType(), False),
        StructField("post_code", StringType(), False),
        StructField("email", StringType(), False),
        StructField("username", StringType(), False),
        StructField("registered_date", StringType(), False),
        StructField("phone", StringType(), False),
        StructField("picture", StringType(), False)
    ])

    sel = spark_df.selectExpr("CAST(value AS STRING)") \
        .select(from_json(col('value'), schema).alias('data')).select("data.*")

    return sel
```

6. Next we created cassandra connection. We created keyspace and table using connection.

```
def create_spark_connection():
    s_conn = None

    try:
        print("Creating spark session!")
        print("*****")
        s_conn = SparkSession.builder \
            .appName('SparkDataStreaming') \
            .config('spark.jars.packages', "com.datastax.spark:spark-cassandra-connector_2.12:3.4.0,"
                                             "org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.0") \
            .config('spark.cassandra.connection.host', 'localhost') \
            .getOrCreate()

        s_conn.sparkContext.setLogLevel("ERROR")
        print("Spark connection created successfully!")
        print("*****")
    except Exception as e:
        print(f"Couldn't create the spark session due to exception {e}")
        print("*****")

    return s_conn

def create_keyspace(session):
    session.execute("""
        CREATE KEYSPACE IF NOT EXISTS spark_streams
        WITH replication = {'class': 'SimpleStrategy', 'replication_factor': '1'};
    """)

    print("Cassandra - Keyspace created successfully!")
    print("*****")

def create_table(session):
    session.execute("""
        CREATE TABLE IF NOT EXISTS spark_streams.created_users (
            id TEXT PRIMARY KEY,
            first_name TEXT,
            last_name TEXT,
            gender TEXT,
            address TEXT,
            post_code TEXT,
            email TEXT,
            username TEXT,
            registered_date TEXT,
            phone TEXT,
            picture TEXT);
    """)

    print("Cassandra Table created successfully!")
    print("*****")
```

7. Then we were writing dataframe into cassandra using spark writeStream

```
def foreach_batch_function(df, epoch_id):
    # Print each micro-batch for debugging
    print("Records inserted..")
    print("\n")
    print(f"Batch {epoch_id}")
    df.show()
    print("*****")

if __name__ == "__main__":
    # create spark connection
    spark_conn = create_spark_connection()

    if spark_conn is not None:
        # connect to kafka with spark connection
        spark_df = connect_to_kafka(spark_conn)
        selection_df = create_selection_df_from_kafka(spark_df)
        session = create_cassandra_connection()

        if session is not None:
            create_keyspace(session)
            create_table(session)

            print("Streaming is being started...")
            print("*****")

            streaming_query = (selection_df.writeStream.foreachBatch(foreach_batch_function)
                               .outputMode("append")
                               .format("org.apache.spark.sql.cassandra")
                               .option('checkpointLocation', '/tmp/checkpoint')
                               .option('keyspace', 'spark_streams')
                               .option('table', 'created_users')
                               .start())

            streaming_query.awaitTermination()
```

8. Checked spark streaming app running successfully

```

org.slf4j#slf4j-api;2.0.7 from central in [default]
org.xerial.snappy#snappy-java;1.1.10.3 from central in [default]
:: evicted modules:
org.slf4j#slf4j-api;1.7.26 by [org.slf4j#slf4j-api;2.0.7] in [default]
com.google.code.findbugs#jsr305;3.0.0 by [com.google.code.findbugs#jsr305;3.0.2] in [default]
-----
|               | modules                | artifacts |
|               | number| search|dwnlded|evicted|| number|dwnlded|
|-----|-----|-----|-----|-----|
| default      | 29 | 0 | 0 | 2 || 27 | 0 |
|-----|-----|-----|-----|-----|

:: retrieving :: org.apache.spark#spark-submit-parent-4509c2bd-b5dd-4a4f-940d-2e55a0ddbee4
confs: [default]
0 artifacts copied, 27 already retrieved (0kB/49ms)
24/01/27 10:14:18 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using built
in-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/01/27 10:14:21 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
24/01/27 10:14:21 WARN Utils: Service 'SparkUI' could not bind on port 4041. Attempting port 4042.
Spark connection created successfully!
*****
kafka dataframe created successfully
*****
Cassandra session created successfully!
*****
Cassandra - Keyspace created successfully!
*****
Cassandra Table created successfully!
*****
Streaming is being started...
*****

```

9. At last, checked Cassandra table records using cql command

```

d/men/61.jpg | 99841 | 2014-05-22T18:34:45.036Z | ticklishpeacock947
48d3cda3-930a-48b1-bdb1-465a737cad91 | 6430 Lanzheronivska, Snyatin, Zaporizka
mitro.yaciv@example.com | Dmitro | male | Yaciv | (067) H04-1366 | https://randomuser.me/api/portraits/me
d/men/38.jpg | 67719 | 2003-05-06T14:40:30.192Z | organicsnake721
09c039d1-0928-4f3f-acee-c7add5c8ce0f | 6956 Čitačka, Brus, Kosovo, Serbia | nast
asiya.dokic@example.com | Nastasiya | female | Dokić | 015-1374-710 | https://randomuser.me/api/portraits/med/
women/78.jpg | 54889 | 2002-09-26T18:37:18.266Z | purpleostrich606
e1bd0a51-3201-4ddc-acd6-5356bb955eed | 2344 Calle de Bravo Murillo, Parla, Ceuta, Spain | mar
garita.diez@example.com | Margarita | female | Diez | 964-512-360 | https://randomuser.me/api/portraits/med/
women/16.jpg | 92533 | 2008-12-06T12:43:38.778Z | ticklishostrich411
cce9f688-1b69-47cd-9636-b109e23dd46e | 5594 Klaarwater, Klein Zundert, Friesland, Netherlands |
wim.denadel@example.com | Wim | male | Den Adel | (085) 1920523 | https://randomuser.me/api/portraits/m
ed/men/3.jpg | 3986 MQ | 2018-02-19T19:11:27.301Z | angryfish772
a1f0b603-6b7e-42b9-a018-8a67f7f2348d | 2667 Main St, Delisle, Manitoba, Canada | ph
ilippe.cote@example.com | Philippe | male | Côté | M26 A28-5240 | https://randomuser.me/api/portraits/me
d/men/40.jpg | T8S 0I2 | 2011-04-05T09:16:31.405Z | happybird557

(10 rows)
cqlsh:spark_streams>

```