Large Scale Data Management

ASSIGNMENT 2

Stylianos Giagkos | f3352410

Large Scale Data Management
ASSIGNMENT 1
Stylianos Giagkos | f3352410

PREAMBLE In this project, you will use the Apache Spark framework and the Apache Cassandra NoSQL database. The scope is to create a Structured Streaming Spark process that consumes Kafka messages and uses Cassandra as a sink to persist information. You are encouraged to use the attached Vagrantfile that will provide you with Apache Spark (installed in the VM), Apache Kafka (running as a container and accessible through port 29092) and Apache Cassandra (running as a container and accessible through port 9042).

PART I: In this part you will create using Python and Kafka a stream of movies rated by a group of people. You should create with the use of the Python Faker library a list of names (at least 10) and use a list of movies (see attached file 'movies.csv') to periodically generate a movie for each of them (at most per minute). In addition to the Faker produced names you should also include your own name in the list (hard-coded). Your Python script should produce a Kafka stream with these movies. The information to be included in the stream comprises the name of the person watching the movie, the name of the movie, the current time, and a random movie rating (1-10). You should consult file 'python-kafka-example.py' that will help you develop your own script.

1. Producer (part1_kafka_stream_producer.py):

Part 1 Code

- This code is a Python script that uses asynchronous programming to produce (send) movie ratings to a Kafka topic. Here's a breakdown of what each part does:

1. **Imports**:
    - json: Used to serialize Python objects into JSON format.
    - asyncio: Provides asynchronous programming support for managing concurrent tasks.
    - random: Used for generating random choices, like selecting names and movie titles.
    - aiokafka.AIOKafkaProducer: Kafka producer that supports asynchronous operations with asyncio.
    - faker.Faker: Generates fake data such as random names for simulation.
    - pandas: For reading and manipulating CSV data.
    - datetime: For working with timestamps.
2. **Loading Movie Titles**:
    - The code loads movie titles from a CSV file (movies.csv) located at /vagrant/data/movies.csv. It reads the file into a pandas DataFrame and then converts the first column (movie titles) into a list (movies_list).
3. **Faker Instance**:
    - The Faker library is used to generate random fake names. It creates a list of 14 fake names, followed by one hardcoded name ("Stelios Giagkos").
4. **Kafka Topic**:
    - The Kafka topic to which the movie ratings will be sent is defined as 'movies_ratings'.

5. **Serializer Function**:
    - The serializer function converts a Python dictionary (the movie rating data) into a JSON string and encodes it into bytes. This is necessary because Kafka producers expect the data to be in byte format.
6. **Produce Function**:
    - This is an asynchronous function that produces movie rating data.
    - It creates an instance of AIOKafkaProducer, specifying the Kafka broker address (localhost:29092), a serializer function, and using gzip compression for the messages.
    - The while True loop continuously sends movie rating data every 10 seconds. Inside the loop:
        - For each iteration, it selects a random name from the names list and a random movie title from the movies_list.
        - It also generates a random movie rating between 1 and 10.
        - It records the current timestamp using datetime.now().isoformat().
        - This data is then packaged into a dictionary (data) and sent to the Kafka topic 'movies_ratings'.
        - The data is printed to the console with a "Sent" message.
        - The loop waits for 10 seconds before sending the next batch of 15 messages.
7. **Asynchronous Execution**:
    - The asyncio.get_event_loop() gets the event loop for asynchronous execution.
    - The loop.run_until_complete(produce()) starts the producer function and runs it until completion. Since produce() is an infinite loop, the script will keep running and sending messages until manually stopped.

- **In summary**: This script simulates the production of movie ratings, sending random movie names, user names, ratings (from 1 to 10), and timestamps to a Kafka topic (movies_ratings) every 10 seconds. Each batch consists of 15 entries, and it runs indefinitely until the script is stopped.

Part 1 Output

```
vagrant@vagrant:/vagrant/part1$ python3 part1_kafka_stream_producer.py
Sent: {'name': 'Michael Sanchez', 'movie': 'Anjelah Johnson: Not Fancy', 'rating': 9, 'timestamp': '2025-03-13T19:26:50.86321
}
Sent: {'name': 'David Rosario', 'movie': 'Hera Pheri', 'rating': 10, 'timestamp': '2025-03-13T19:26:50.863344'}
Sent: {'name': 'Steven Smith', 'movie': 'Jenny Slate: Stage Fright', 'rating': 7, 'timestamp': '2025-03-13T19:26:50.863379'}
Sent: {'name': 'Ronald Jones', 'movie': 'Thimmarusu', 'rating': 4, 'timestamp': '2025-03-13T19:26:50.863405'}
Sent: {'name': 'Ronald Jones', 'movie': 'Sí, Mi Amor', 'rating': 5, 'timestamp': '2025-03-13T19:26:50.863427'}
Sent: {'name': 'Laura Baker', 'movie': 'The Unknown Saint', 'rating': 9, 'timestamp': '2025-03-13T19:26:50.863452'}
Sent: {'name': 'Tina Black', 'movie': 'Kung Fu Panda 2', 'rating': 5, 'timestamp': '2025-03-13T19:26:50.863473'}
Sent: {'name': 'Courtney Young', 'movie': 'P Se PM Tak', 'rating': 4, 'timestamp': '2025-03-13T19:26:50.863492'}
Sent: {'name': 'Tina Black', 'movie': 'Amanda Knox', 'rating': 5, 'timestamp': '2025-03-13T19:26:50.863512'}
Sent: {'name': 'Courtney Young', 'movie': 'Army of the Dead', 'rating': 3, 'timestamp': '2025-03-13T19:26:50.863532'}
Sent: {'name': 'Laura Baker', 'movie': 'New York Minute', 'rating': 9, 'timestamp': '2025-03-13T19:26:50.863552'}
Sent: {'name': 'Courtney Young', 'movie': 'Why Do Fools Fall in Love', 'rating': 10, 'timestamp': '2025-03-13T19:26:50.863571'
Sent: {'name': 'Michael Walker', 'movie': 'Beats', 'rating': 10, 'timestamp': '2025-03-13T19:26:50.863590'}
Sent: {'name': 'Steven Smith', 'movie': '9to5: The Story of a Movement', 'rating': 7, 'timestamp': '2025-03-13T19:26:50.863749
}
Sent: {'name': 'Steven Smith', 'movie': 'Humpty Sharma Ki Dulhania', 'rating': 6, 'timestamp': '2025-03-13T19:26:50.863820'}
Sent: {'name': 'Stelios Giagkos', 'movie': 'Singham', 'rating': 7, 'timestamp': '2025-03-13T19:27:00.871072'}
```

2. Consumer (part2_kafka_stream_consumer.py):

Part 2 Code

- This code is a PySpark-based streaming application that reads movie ratings data from Kafka, enriches it with movie details from a CSV file, and writes the enriched data into a Cassandra database in real-time. Below is a detailed explanation of each part:

- **1. Importing Required Libraries:**

- SparkSession: Used to initialize the Spark session.
- StructType, StructField: These are used to define the schema of incoming Kafka messages.
- StringType, IntegerType, TimestampType: Data types for the schema fields.
- col, from_json, expr: Functions for manipulating DataFrames, parsing JSON data, and expressing SQL-like operations.

- **2. Initializing Spark Session:**

- A Spark session (spark) is created using SparkSession.builder. The application is named "KafkaCassandraStream".
- The spark.jars.packages config specifies the dependency to enable Spark to read data from Kafka (spark-sql-kafka).
- spark.sparkContext.setLogLevel("ERROR") sets the logging level to only show error messages (reducing verbosity).

- **3. Defining the Schema for the Incoming Data:**

- The movie_rating_schema defines the structure of the incoming data, with four fields: name (String), movie (String), rating (Integer), and timestamp (Timestamp).
- This schema will be used to parse the JSON data from Kafka.

- **4. Loading the Movie Details from CSV:**

- The CSV file /vagrant/data/netflix.csv containing movie details is loaded into a DataFrame (movies_df).
- The .cache() method is used to store the movie details DataFrame in memory, improving performance when it is used in repeated operations like the join.

- **5. Reading the Kafka Stream:**

- spark.readStream is used to read data from the Kafka topic 'movies_ratings'.
- The option("kafka.bootstrap.servers", "localhost:29092") configures the Kafka server.
- option("startingOffsets", "latest") ensures that the stream reads new messages starting from the latest offset in the Kafka topic.

- **6. Parsing the Kafka Data:**

- The Kafka message (value) is in byte format. It is first cast to a string using CAST(value AS STRING).
- The from_json function is used to parse the string into a structured format based on the movie_rating_schema. This results in a DataFrame with columns like name, movie, rating, and timestamp.

– **7. Renaming the Rating Column:**

- To avoid ambiguity with the rating column from both the Kafka data and the movie details DataFrame, the rating column in the Kafka data is renamed to rating_movie.

– **8. Joining with the Movie Details:**

- The parsed DataFrame (parsed_df) is joined with movies_df (the movie details DataFrame) on the movie title (parsed_df.movie == movies_df.title).
- The join is an inner join, meaning only rows with matching movie titles will be included.
- A unique ID (id) is generated using expr("uuid()").
- Additional columns such as show_id, rating_movie, release_year, and director are selected from the joined DataFrame. These come from the movies_df DataFrame.

– **9. Writing the Stream to Cassandra:**

- The function write_to_cassandra is defined to write the processed data to Cassandra.
- The data is written to the movie_ratings table in the netflix keyspace using batch_df.write.format("org.apache.spark.sql.cassandra").
- The mode("append") ensures that data is appended to the table rather than overwriting it.

– **10. Configuring the Stream to Write Every 30 Seconds:**

- The writeStream function is used to set up the streaming query.
- outputMode("append") ensures that new data is appended to the output.
- .foreachBatch(write_to_cassandra) writes each batch of data to Cassandra.
- .trigger(processingTime='30 seconds') specifies that the data should be processed and written every 30 seconds.
- query.awaitTermination() keeps the query running indefinitely, processing and writing new data as it arrives.

– **11. Cassandra Schema:**

- The Cassandra CREATE KEYSPACE and CREATE TABLE commands are used to set up the schema for storing movie ratings in Cassandra.
- The keyspace netflix is created with a simple replication strategy, and the movie_ratings table is created with the columns id, name, timestamp, show_id, movie, rating_movie, duration, rating, release_year, director, and country.

- The primary key for the table is (name, timestamp, id), with name as the partition key and timestamp as the clustering key. This ensures that data for each person is grouped by name and ordered by the timestamp in descending order.

- **Summary:**

- This code uses Apache Spark to stream movie ratings from Kafka, enriches the data with movie details from a CSV file, and writes the enriched data into a Cassandra database. The data is processed in batches every 30 seconds and stored with a unique ID, ready for further analysis or querying in the Cassandra database.

3. Details about Cassandra data model

DESCRIBE KEYSPACE netflix;

The netflix keyspace is configured to efficiently store movie ratings data with performance and data integrity in mind. The movie_ratings table uses a composite primary key of name, timestamp, and id, with columns including country, director, movie, rating, rating_movie, release_year, and show_id. The table is optimized for read performance, utilizing caching for keys and employing the SizeTieredCompactionStrategy for efficient data storage. LZ4 compression is used for faster access, and durability is ensured through durable_writes = true and a garbage collection grace period of 10 days. The setup also includes a 99th percentile speculative retry for read queries, balancing fast, reliable access with optimized performance under load.

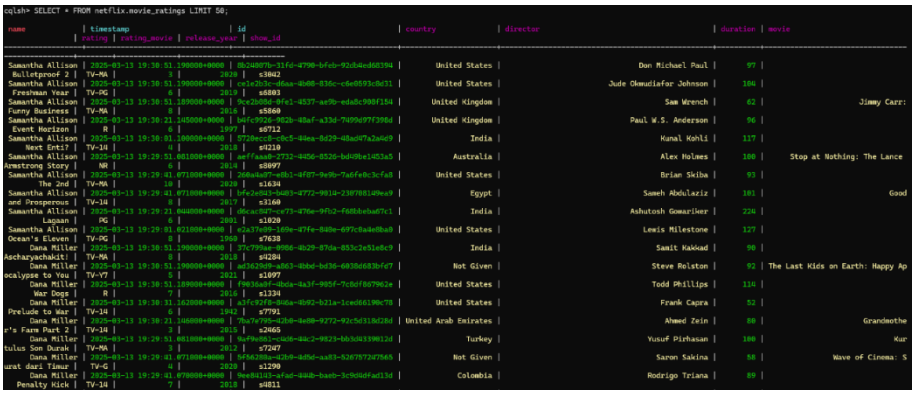
```
cqlsh> DESCRIBE KEYSPACE netflix;

CREATE KEYSPACE netflix WITH replication = {'class': 'SimpleStrategy', 'replication_factor': '1'}  AND durable_writes = true;

CREATE TABLE netflix.movie_ratings (
    name text,
    timestamp timestamp,
    id uuid,
    country text,
    director text,
    duration int,
    movie text,
    rating text,
    rating_movie int,
    release_year int,
    show_id text,
    PRIMARY KEY (name, timestamp, id)
) WITH CLUSTERING ORDER BY (timestamp DESC, id ASC)
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_threshold': '32', 'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '64', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND crc_check_chance = 1.0
    AND dclocal_read_repair_chance = 0.1
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair_chance = 0.0
    AND speculative_retry = '99PERCENTILE';
```

4. A sample of persisted lines (around 50)

SELECT * FROM netflix.movie_ratings LIMIT 50;



5. Two CQL queries and their results

Query 1: Average Duration of Movies Rated by User 'Stelios Giagkos' Between March 10, 2025, 00:00:00 and March 14, 2025, 00:00:00

To get the average movie runtime for a particular hour:

```
cqlsh> SELECT AVG(duration) AS avg_duration
   ... FROM netflix.movie_ratings
   ... WHERE name = 'Stelios Giagkos'
   ...   AND timestamp >= '2025-03-10 00:00:00'
   ...   AND timestamp < '2025-03-14 00:00:00';

 avg_duration
--------------
          101

(1 rows)
cqlsh>
```

SELECT AVG(duration) AS avg_duration

FROM netflix.movie_ratings

WHERE name = 'Stelios Giagkos'

AND timestamp >= '2025-03-10 00:00:00'

AND timestamp < '2025-03-14 00:00:00';

Query 2: List of Movies Rated by You During a Particular Hour

To get the list of movies you rated during the same hour:

```
cqlsh> SELECT movie
   ... FROM netflix.movie_ratings
   ... WHERE name = 'Stelios Giagkos'
   ...   AND timestamp >= '2025-03-10 00:00:00'
   ...   AND timestamp < '2025-03-14 00:00:00';

 movie
-------------------------------------------------
                                      Dolphin Kick
                                     The Last Days
 Unbreakable Kimmy Schmidt: Kimmy vs. the Reverend
                                               Jal
                                            Broken
                                     Roll Red Roll
          Naruto Shippuden the Movie: Blood Prison
                                          Outlawed
                                 3 Türken & ein Baby
                                          Stowaway
              Judy Moody and the Not Bummer Summer
                                            Silent
                                            Wolves
                                The House Next Door
                          Haunting on Fraternity Row
                                           Maktub
                                  Playing for Time
                                        Radiopetti
                                   Sherlock Holmes
                                       The Nut Job
                          Motu Patlu Dino Invasion
                                    My Travel Buddy
                                    Mustang Island
         Trailer Park Boys: Countdown to Liquor Day
                     Dr Jason Leong Hashtag Blessed
                                          YES DAY
                                    Well Done Abba
                                     Teach Us All
                                       Fireflies
                                          Deliha
          Erementari: The Blacksmith and the Devil
                                 3 Türken & ein Baby
                                       Fireworks
                                  The Jungle School
                                  The Vatican Tapes
                                         Piercing
                                           Devrai
                                     Road to Sangam
                The Awakening of Motti Wolkenbruch

(39 rows)
```

SELECT movie

FROM netflix.movie_ratings

WHERE name = 'Stelios Giagkos'

AND timestamp >= '2025-03-10 00:00:00'

AND timestamp < '2025-03-14 00:00:00';