Project: Design and Implement Scalable Data Pipelines for Consumer Segmentation Application

Objective:

Design and implement scalable, efficient, and reliable data pipelines to support a consumer segmentation application. Each pipeline should handle **data ingestion, transformation, storage, and retrieval** to enable **real-time or batch processing**.

Task 1: Kafka Producer Simulation (OOP-based Implementation with Schema Registry Integration)

Develop a **Kafka producer in Python** to simulate **user activity events** for a consumer segmentation application. The producer should generate and send events to Kafka topics representing the following activities:

1. Movies Events (Using Spark Producer)

- Batch process Movies.txt file and orginize data for **Spark Streaming producer** to produce **movie-related events** to Kafka - event_name = "movies_catalog_enriched".
- Use **Apache Spark** to read and process the `Movies.txt` file.
- **ListPrice** should be parsed to extract the correct price, as it may contain multiple formats (e.g., `4999USD$49.99`). Only the **dollar value (`$49.99`)** should be used.

User Registration

Each **User Registration** event must include:

- `timestamp`: Event creation timestamp (ISO 8601 format).
- `event_name`: String, must be `"consumer_registration"`.
- `user_id`: Unique consumer ID (integer from `1` to `1,000,000`), randomly generated.
- `age`: Randomized user age within a reasonable range (`18–95`).
- `masked_email`: Email address with domain masked (e.g., `****@gmail.com`).
- `preferred_language`: Randomly assigned from `["eng", "geo", "□□"]`.

Sign In Event

Each **Sign In** event must include:

- `timestamp`: Event creation timestamp (ISO 8601 format).
- `event_name`: String, must be `"sign_in"`.
- `user_id`: Consumer ID associated with the event.

Sign Out Event

Each **Sign Out** event must include:

- `timestamp`: Event creation timestamp (ISO 8601 format).
- `event_name`: String, must be `"sign_out"`.

- `user_id`: Consumer ID associated with the event.

Item View Event

Each **Item View** event must include:

- `timestamp`: Event creation timestamp (ISO 8601 format).
- `event_name`: String, must be `"item_view"`.
- `item_id`: String, chosen randomly from `Movies.txt` file content.
- `user_id`: Randomly selected from already created users.

Add to Cart Event

Each **Add to Cart** event must include:

- `timestamp`: Event creation timestamp (ISO 8601 format).
- `event_name`: String, must be `"added_to_cart"`.
- `item_id`: String, chosen randomly from `Movies.txt` file content.
- `user_id`: Randomly selected from already created users.
- `cart_id`: String, uuid random.

Checkout Event

Each **Checkout** event must include:

- `timestamp`: Event creation timestamp (ISO 8601 format).
- `event_name`: String, must be `"checkout_to_cart"`.
- `user_id`: Randomly selected from already created consumers.
- `cart_id`: String, from existing cart_id set.
- `payment_method`: String, randomly choose from ["Cash", "Card"].

Task 2: Consume Kafka Topics Using Spark Structured Streaming and Load Data to Snowflake

1. Consume Kafka Topics

- Use **Spark Structured Streaming** to consume the Kafka topics created in **Task 1**.
- Process the events in **real-time** and transform them as needed for downstream systems.
- Ensure the ingestion of data from Kafka topics such as:
  - `consumer_registration_topic`
  - `sign_in_topic`
  - `item_view_topic`
  - `checkout_topic`
  - `added_to_cart`
  - `checkout_to_cart`
  - `movies_catalog_enriched`

2. Data Transformation

- Transform the data to match the desired schema, ensuring consistency between the Kafka messages and the target Snowflake table.
- Handle any necessary data cleaning or enrichment (e.g., parsing price, combining related fields).
- Use correct data types when defining tables.

3. Load Data to Snowflake

- Use **Spark Snowflake Connector** to load the transformed data into **Snowflake** for analytics and reporting.
- Ensure the pipeline supports **micro-batch processing** for real-time ingestion into Snowflake.
- Store the data in appropriate Snowflake tables (e.g., `consumer_events`, `item_view_events`, `cart_events`, `etc`).

Task 3: Orchestrate Movie File Processing with Apache Airflow

1. Orchestrate Movie Files Ingestion

- Use **Apache Airflow** to create a **DAG** that orchestrates the processing of movie files (e.g., `Movies.txt`).
- The DAG should:
  - Trigger on a scheduled basis or when new movie files are available.
  - Process movie files using **Apache Spark** to extract movie details.
  - Ensure that the file processing, transformation steps are executed in sequence.

2. Movie File Processing Steps

- Read the movie files (e.g., `Movies.txt`) and process them with **Spark** to generate event data for Kafka.
- Use Airflow to manage and monitor the status of the movie file processing steps, ensuring they are completed successfully.

3. Integrate with Kafka Producer

- After processing the movie file, ensure that the movie-related events are produced to Kafka (as in **Task 1**), enabling real-time processing.

Task 4: Create Transformations Using DBT Cloud for Data Mart Creation

1. Set Up DBT Cloud Project

- Set up a **DBT Cloud project** and connect it to your Snowflake instance.
- Configure the appropriate Snowflake warehouse and database for transformations.

2. Create Source Models

- Define **source models** in DBT to pull data from Snowflake tables (e.g., `consumer_events`, `item_view_events`, `cart_events`) created in **Task 2**.
- Use DBT’s **source configuration** to define the raw data sources for the transformation pipeline.

3. Build Transformation Models

- Create **transformation models** that aggregate and structure the raw event data into a **data mart** suitable for analytics.
  - **Aggregated User Activity Data**: Create a model to aggregate user activity over time (e.g., number of items viewed, added to cart, or purchased).

    - **Movie Statistics**: Create a model to aggregate movie statistics such as total views, total sales, and average user ratings.
    - **User Segmentation**: Develop models to identify user segments based on their activity (e.g., high-value customers, frequent browsers, inactive users) - optional.
- Use **DBT’s Jinja macros** and **SQL-based transformations** to join and aggregate data efficiently.

4. Testing and Documentation

- Use **DBT’s testing framework** to validate the transformations and ensure data integrity.
    - Example: Add tests to check that user IDs in the `consumer_events` table match valid IDs in the `users` reference table.
- Document the transformations and models using DBT’s **built-in documentation tools**, making it easy for analysts and stakeholders to understand the logic behind each model (optional).

5. Schedule and Automate the Data Mart Builds

- Set up **DBT Cloud schedules** to automatically run transformations at defined intervals (e.g., daily or hourly).
- Ensure that data is always up-to-date for analytical queries and reporting.

Note

- For this task students are allowed to use DBT Core. If so, please, use other data platform (not Snowflake)

Task 5: User Registration Count with Tumbling and Sliding Windows (Hourly Aggregation)

1. User Registration Count (Tumbling Window)

- Use **Spark Structured Streaming** to compute the **user registration count** every hour using a **tumbling window**.
    - **Tumbling Window**: Aggregates user registration events over fixed, non-overlapping time intervals.
    - Example: For each **1-hour window**, count the number of *checkouts** (i.e., count of distinct `user_id`).

2. User Registration Count (Sliding Window)

- Additionally, use a **sliding window** to compute a **rolling user registration count**.
    - **Sliding Window**: This window slides forward in time (e.g., every 15 minutes) and aggregates user registrations over a 1-hour period.
    - Example: For each **sliding window**, compute the **unique user registrations** in the last **1 hour**.

3. Metrics Aggregation

- Both tumbling and sliding window metrics will be written to a Snowflake table (e.g., `user_registration_metrics`) with the following columns:
    - `timestamp`: Timestamp of the window.

    - `window_type`: Either `tumbling` or `sliding`.
    - `registration_count`: Count of unique user registrations during the window.

4. Performance Optimization

- Optimize the streaming queries for performance to handle high-throughput data, ensuring that the aggregation process scales as the number of user registration events increases (if applicable).

Bonus: Active Sessions Count and Average Session Duration in Real-Time

1. Active Sessions Count

- Use **Spark Structured Streaming** to track the **active sessions** in real-time.
    - Define **session activity** based on **user sign-in** and **sign-out** events.
    - Use the `user_id` and `timestamp` to determine whether a session is currently active.
    - Count the number of **active sessions** in real-time (i.e., how many users are currently logged in).

2. Average Session Duration

- Calculate the **average session duration** for active sessions:
    - For each user session, compute the **time difference** between the **sign-in** and **sign-out** events.
    - Use **Spark Streaming** to compute the average session duration across all sessions in real-time.
    - Output the average session duration to a Snowflake table (e.g., `session_metrics`).

3. Metrics Aggregation

- Store the following metrics in the Snowflake table:
    - `timestamp`: Timestamp of the metric calculation.
    - `active_sessions_count`: Count of active sessions at that timestamp.
    - `average_session_duration`: Average session duration for active sessions.

4. Real-Time Monitoring

- Implement **real-time monitoring** of session metrics to ensure accurate and timely reporting of active session counts and session durations.

Task 6: Calculate Total Sales of Movies

1. Aggregate Movie Sales

- Use **Spark Structured Streaming** to aggregate the **total sales** of movies.
    - **Movie Sales Event**: Each purchase event (from `checkout_topic`) should include details such as `movie_id` and `purchase_amount`.
    - Aggregate sales for each movie by summing the `purchase_amount` for every relevant event.

2. Store Sales Metrics

- Store the **total sales** for each movie in a Snowflake table (e.g., `movie_sales_metrics`).
  - The table should include columns such as:
    - `movie_id`: Unique identifier for each movie.
    - `total_sales`: The total sales value (sum of `purchase_amount`) for that movie.
    - `timestamp`: Timestamp of the aggregation.
  - Perform **windowed aggregations** to compute sales for different time periods (e.g., daily, weekly).

3. Performance Optimization

- Optimize the streaming query to ensure efficient aggregation and minimize latency in calculating movie sales.
- Use **Spark Structured Streaming**'s **stateful operations** for efficient cumulative sum calculations.

Requirements

- **OOP Implementation**: Utilize Object-Oriented Programming (OOP) principles, including encapsulation, inheritance, and polymorphism where applicable.
- **Python & Kafka**: Use Python and the `confluent-kafka` library.
- **Topic Organization**: Define appropriate Kafka topics for different event categories.
- **Avro Format & Schema Registry**: Structure events as Avro-formatted messages and integrate with Confluent Schema Registry for schema validation and compatibility.
- **Event Production**: Ensure messages are successfully produced to Kafka.
- **Randomized Event Generation**: Simulate real-world user interactions with randomly generated event data.
- **Apache Spark Integration**: Use Spark Streaming for processing and producing movie events from the `Movies.txt` file.
- **Spark Structured Streaming**: Consume data from Kafka topics and load the processed data into Snowflake.
- **Snowflake Integration**: Use Spark Snowflake Connector to load data into Snowflake.
- **Apache Airflow**: Create a DAG to orchestrate the movie file processing pipeline.
- **DBT Cloud Integration**: Use DBT Cloud for data transformation and create a data mart for analytics in Snowflake.
- **Windowed Aggregation**: Implement **tumbling and sliding windows** for hourly user registration count aggregation.
- **Active Sessions Count**: Implement tracking of active sessions and average session duration in real-time.
- **Total Sales Calculation**: Aggregate **total sales** of movies and store in Snowflake.