Data Engineering-in-class-Activity #2

Here's a summary of the purchases:

Customers:

1. eabara
2. htanaka
3. jsmith
4. jbernard
5. awalther
6. sgarcia

Products:

1. alarm clock
2. t-shirts
3. batteries
4. book
5. gift card

Purchases:

1. eabara: 4 alarm clocks, 5 t-shirts, 3 batteries, 0 books, 4 gift cards
2. htanaka: 5 alarm clocks, 3 t-shirts, 3 batteries, 5 books, 4 gift cards
3. jsmith: 5 alarm clocks, 3 t-shirts, 3 batteries, 6 books, 5 gift cards
4. jbernard: 6 alarm clocks, 7 t-shirts, 5 batteries, 6 books, 4 gift cards
5. awalther: 7 alarm clocks, 6 t-shirts, 5 batteries, 5 books, 3 gift cards
6. sgarcia: 6 alarm clocks, 6 t-shirts, 6 batteries, 5 books, 6 gift cards

Exploring Confluent Cloud and Reporting Findings

Introduction:
During our in-class activity, we have been tasked with exploring the Confluent Cloud site and gathering information on the features, services, and tools it offers. This report aims to summarize the findings and present them for discussion during the class session and as part of Assignment 1.
Findings:

1. Overview of Confluent Cloud:
2. Confluent Cloud is a fully managed, cloud-native Kafka service that allows users to easily build event-driven applications using Apache Kafka. The platform offers scalability, security, and flexibility, making it a suitable choice for businesses looking to manage real-time data streams.
3. Key Features:
4. a. Fully Managed Service: Confluent Cloud manages the underlying infrastructure, enabling users to focus on building applications without worrying about the complexities of managing a Kafka cluster.
5. b. Elastic Scalability: The platform allows users to scale their Kafka clusters up or down based on their requirements, ensuring optimal resource usage and cost-efficiency.
6. c. Global Availability: Confluent Cloud is available across multiple cloud providers and regions, providing users with the flexibility to deploy their applications close to their end-users.
7. d. Security: The platform offers built-in security features, such as data encryption at rest and in transit, as well as support for role-based access control and private networking.
8. Confluent Cloud Ecosystem:
9. a. Connectors: Confluent Cloud provides a range of pre-built connectors that allow users to easily integrate their Kafka clusters with various data sources and sinks, such as databases, storage systems, and messaging services.
10. b. ksqlDB: This is a fully managed, serverless event-streaming database that enables users to perform real-time analytics and processing of their data streams using SQL-like queries.
11. c. Schema Registry: Confluent Cloud's Schema Registry helps users manage and enforce schemas for their Kafka topics, ensuring data compatibility and consistency across their applications.
12. d. Monitoring and Alerting: The platform offers built-in monitoring and alerting tools, enabling users to track the performance of their Kafka clusters and receive notifications about potential issues.
13. Pricing:
14. Confluent Cloud offers a variety of pricing plans, including pay-as-you-go and committed-use discounts, catering to different business sizes and requirements. Users can choose between the Basic, Standard, or Dedicated clusters, each with its own set of features and performance characteristics.

Conclusion:
Our exploration of the Confluent Cloud site has provided us with valuable insights into the platform's features, ecosystem, and pricing. With its fully managed service, elastic scalability, global availability, and robust security features, Confluent Cloud presents a compelling option for businesses looking to build event-driven applications using Apache Kafka.

G section:

1. What does Producer.flush() do?

Producer.flush() is a method in the Kafka Producer API that ensures all the messages in the producer's buffer are sent to the Kafka brokers before the method returns. It blocks the execution of the program until all the messages are acknowledged by the brokers. This method is particularly useful when you want to guarantee that all messages have been sent before proceeding to the next step or ending the program.

2. What happens if you do not call producer.flush()?

If you do not call producer.flush(), messages may still be sent to the Kafka brokers, but there is no guarantee that all messages in the buffer will be sent immediately. The producer might wait for more messages to fill the buffer or until a specified timeout is reached. This could lead to data loss if the producer is shut down before all messages are sent, or messages could be sent with a delay.

3. What happens if you call producer.flush() after sending each record?

Calling producer.flush() after sending each record ensures that every message is sent to the Kafka brokers immediately, without waiting for the buffer to fill or a timeout to occur. This provides a stronger guarantee that messages are sent as soon as they are produced, but it may lead to increased latency and reduced throughput, as the producer has to wait for each message to be acknowledged by the brokers before proceeding.

4. What happens if you wait for 2 seconds after every 5th record send, and you call flush only after every 15 record sends, and you have a consumer running concurrently? Specifically, does the consumer receive each message immediately? only after a flush? Something else?

In this scenario, the producer sends messages in batches of 15 and calls flush() after every 15 records, introducing a delay of 2 seconds after every 5th record. The consumer will receive messages in the following manner:

- The first 5 records will be sent without any delay.
- The next 5 records will be sent with a 2-second delay after the 5th record.
- The last 5 records will be sent with a 2-second delay after the 10th record.
- After sending the 15th record, the producer will call flush(), ensuring all messages in the buffer are sent to the Kafka brokers.

In this case, the consumer will receive messages as they are sent by the producer. The consumer might receive messages in smaller batches or with delays, depending on when the producer calls flush() and when the delays are introduced. Messages will not necessarily be received by the consumer immediately after they are produced, but they will be received in the order they were sent, as Kafka maintains message order within a partition.