Implementation of Kafka: Flight Alert

Semester project report for Advance Database Systems, Dr. Xiaofei Zhang instructing.

Likhitha Javvaji – U00841140
Department of Computer Science
University of Memphis
ljavvaji@memphis.edu

Naveenreddy Narayana - U00864162
Department of Data Science
University of Memphis
nnrayana@memphis.edu

ABSTRACT:

Apache Kafka is a distributed publish-subscribe messaging system and a reliable queue that can handle an enormous amount of data and enables you pass messages from one endpoint to another. Both offline and online message consumption are suitable for Kafka. So, the goal of our project is to implement an end-to-end use case to send alerts about the respective Flight Details by using Kafka as the streaming platform.

1. Introduction:

Motivation:

As in the airline industry, huge amount of real time data will be processed daily as they have many sources of real-time data on flights like, travel updates, checkin information, discounts on upcoming flights, departure and arrival schedules, and other crucial information that must be analyzed and immediately sent to multiple systems and apps. So, as Kafka offers a fault-tolerant, scalable architecture that can manage massive real-time data stream volumes. So, it will be helpful for the airlines to offer the above sources in real-time by using Kafka, as it will be improving customer experience overall and keeping them updated.

Objectives:[1]

- Efficient Processing: The system should be able to process an enormous quantity of flight data and deliver alerts quickly and without any kind of delay.
- Personalized Alerts: Users should be able to receive customized alerts from the system depending on their selected airlines, point of origin, about discounts etc.
- Reliability: Users should receive alerts from the system even if there is a temporary system breakdown, and it should be dependable.

These objectives can be easily handled by Kafka as it handles huge amounts of data in real time. So, Implementing Kafka in flights alerts is a very good choice for our project.

Challenges:

Real-time Data Processing: Processing the data in real-time is essential since flight information must be accurate. This involves taking in enormous volumes of data, filtering and combining it, and then setting off necessary alerts.

Data Integration: The information needed for flight alerts is gathered from a variety of sources, including airlines, airports, predictions of the weather, and more. It can be difficult to combine all these data sources and ensure that they are correct and current.

Scalability: The system should be able to handle rising traffic and scale efficiently as the number of users and flights increases.

Kafka can address the above challenges as its ability is to handle huge volumes of data. It guarantees to give accurate results within a short span of time and ensures the message is delivered exactly once and in the correct order. So, this is one more reason to use Kafka as our streaming platform.

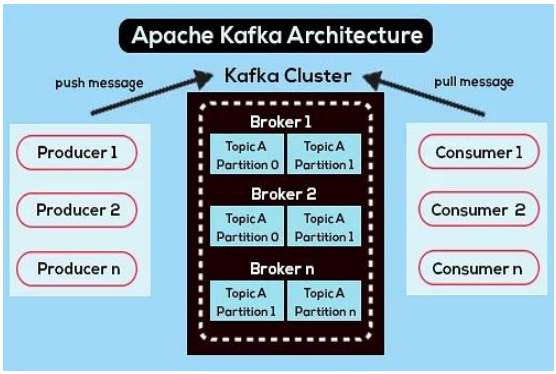
1.1 Apache Kafka:

An open-source distributed publish-subscribe messaging system called Apache Kafka was created with a particular objective of handling real-time streaming data for distributed streaming, pipelining, and replay of data feeds for quick, scalable techniques. A broker-based system called Kafka works by keeping data streams as records across a server cluster. Kafka servers can connect to several data centers and offer data durability by grouping streams of messages into topics and distributing them among various server instances. To avoid data loss, Kafka messages are replicated across the cluster and persistently stored on disk. The synchronization service Zookeeper is the basis upon which Kafka is constructed. [1]

1.2 Architecture:

Kafka maintains key-value messages that arrive from a fixed number of producers. The data can be divided into many "partitions" inside various "topics". Messages are indexed and stored along with a timestamp in a partition in strict order according to their offsets, or the position of a message within a partition. Messages from partitions can be received by other processes referred to as "consumers". For stream processing, Kafka provides the Streams API, which enables developers to create Java programs that read data from Kafka and return results to Kafka.[2]

Let's see each of the components that was mentioned in the architecture.[2]



Topic: A Kafka topic is a collection of messages that belong to a certain category or feed. Kafka stores information in the form of topics. Producers write their to the topics, and consumers read the data from these topics.

Broker: One or more servers that are referred to as brokers make up a Kafka cluster. A broker in Kafka serves as a container that can host several topics with various partitions. Brokers in the Kafka cluster are identified by a particular integer ID. A connection to any one of the cluster's Kafka brokers implies a connection across the entire cluster.

Consumer: Consumer read the data from the topics that is from the Kafka cluster. When a consumer has prepared to read the message, the broker must be notified to retrieve the data. According to Kafka, a consumer group is a collection of users who collect information on the same topic.

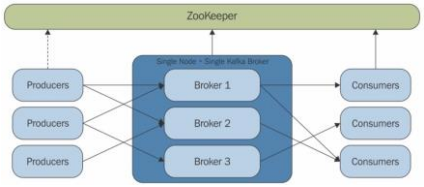
Producer: Producer pushes messages to one or more Kafka topics. They send data to the Kafka cluster. The broker receives a message that a Kafka producer publishes to Kafka and appends it to a specified partition. Producers have the option to publish messages to a certain partition.

Partitions: Kafka divides each topic into a variable number of partitions. Multiple consumers can read data from the same topic in parallel because of partitions. The partitions are divided according to sequence. When defining a topic, the number of partitions is stated; however, this number can be altered at later stages.

Partition Offset: Kafka provides partitions to messages or records. Each record is given an offset, which specifies its location within the partition. The offset value connected to a record can be used to identify it specifically within its partition. A partition offset only has relevance within the partition.

Zookeeper:

Zookeeper is a distributed coordination service that is used by Kafka to manage and maintain its cluster state. Zookeeper is a separate system from Kafka, but it is critical for Kafka to function properly. The cluster's Kafka brokers are managed and coordinated by Zookeeper. It handles the brokers by keeping a list and it utilizes the list to manage them. The Zookeeper is used to alert the producer and consumer when new brokers are created or when the current broker that is present in the Kafka cluster fails. This information will be used by the producer and consumer to make decisions in the coordination of the Kafka cluster. The details about the Kafka cluster and consumer details are stored in the Zookeeper. [3]



2. Related Work:

The blog "Kafka for Live Commerce: Transforming Retail Shopping in the Metaverse" [5] discusses the effects of live commerce on the retail shopping experience and how Kafka may be utilized as a streaming platform to enable real-time communication between companies and their consumers. The blog starts off by describing live commerce and describing how it differs from traditional e-commerce. The benefits of live commerce are then discussed, including improved customer engagement and higher conversion rates.

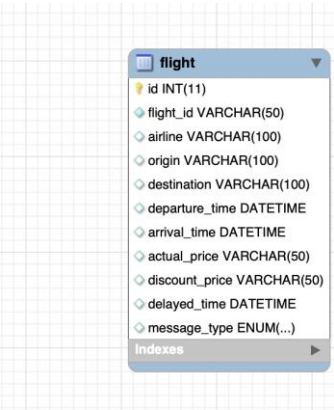
The blog then goes on to explain how Kafka may be used to support live commerce by giving consumers and sellers access to real-time data streams. The article illustrates some of Kafka's most important characteristics, like scalability and fault tolerance, which make it the ideal basis for live commerce applications. The blog offers a few instances of how Kafka has been implemented to the retail sector to facilitate real-time communication between sellers and customers.

As we all know, in the areas like E-commerce, Finance, social media, Health Care, Airline Industry etc. handles huge amounts of data. These are the main platforms that require continuous streaming of data and need to send the data to users correctly within the accurate without causing delay. So, for these concepts as discussed above Kafka is the best streaming platform.

By reading the above blog, we got an idea for the project to work on the concept of Airline Industry. As we get different types of alerts from the airlines like about the upcoming trip, departure time, arrival time, check-in, about the discounts offered by different airlines. So, we thought of a concept to use Kafka as our streaming platform to send the flight alerts for the particular user in the accurate time.

3. Solution:

- As an initial step to send the messages to the producer we created a table in the database called flight. In that table we have created different topics like flight_alert, flight_discount_alert, flight_delay_alert, flight_checkin_alert. These are stored in the attribute messgae_type with data type ENUM. We have stored a few details about the flight by using the INSERT query.



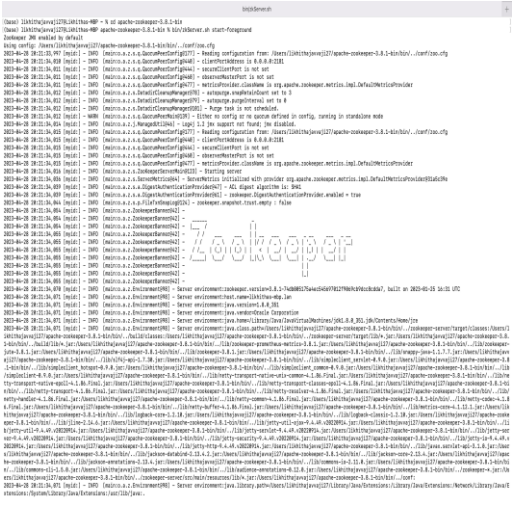
- After the successful creation of table, we have created topics in the command prompt by using below the command.
- In the third step we created a class called FlightKafkaProducer. Here the producer collects the data from database and pushes the data to the respective Kafka topics which can then be consumed by consumers in real time. Here in this class, we created a flight properties object which is used to specify various configuration properties for the Kafka producer. Used one more concept that is producerconfig which is a set of key-value pairs that define the behavior of a Kafka producer [3]. By using the concept of JDBC [4] the producer connects to the database, stores them and sends those details to the respective topics. Now the data will be stored in the topic.
- In the fourth step we created a class called FlightKafkaConsumer. Here the consumer subscribes to one or more topics and reads the data that was stored in the topic. As alike producers in the consumer class also we used the concept of Kafka properties, consumer config for the key-value pairs to define the behavior of a Kafka consumer [3].
- In the Producer class calculated the time that a message is taking to store in the topic and checked the offset that it belongs to. In the same way calculated the time in the Consumer class, here it is evaluating the time that consumer is taking to subscribe and read the data from the particular topic.

4. Evaluation:

1. Technologies Used:[4]

Programming Language - JAVA
Database - MySQL
IDE - IntelliJ IDEA CE, MySQL Workbench

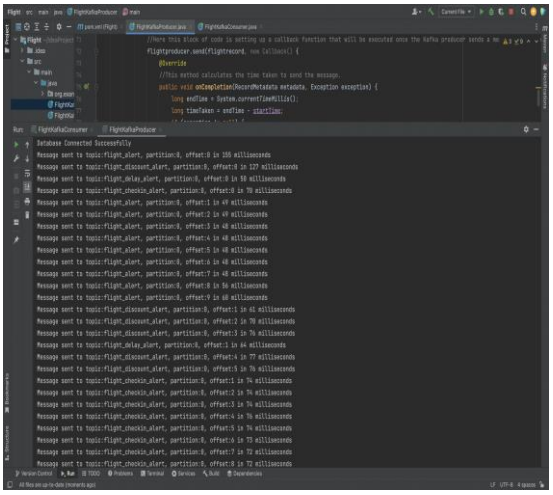
1. First, we need to install Zookeeper and start the Zookeeper session by using the command - **bin/zkServer.sh start-foreground**

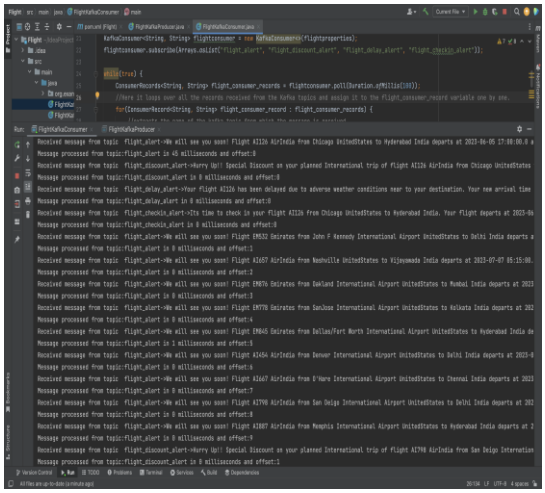


2. Next, we need to install Kafka and start the Kafka by using the command **-bin/kafka-server-start.sh config/server.properties**

3. Next, we need to create the topics - **bin/kafka-topics.sh --create --topic flight_alert --bootstrap-server localhost:9092**

4. Now, first we need to run the consumer class, it will be running background. Once we start the consumer, we start the producer class. Now the producer will send the messages to the topics, consumer will subscribe the topic and read the message from the topic. Here we can see the offset and time calculated in both the classes for each message.





5. Conclusion

In the case of flight alerts, Kafka may be used to interact with numerous data sources (like flight data providers), process the data, and send alerts via a variety of channels (including email, SMS, and mobile apps). Large amounts of data can be managed while ensuring that notifications are sent quickly and effectively by using Kafka. In addition, the fault-tolerant architecture of Kafka can help to ensure that alerts are not lost as a result of disruptions or system failures. Overall, adopting Kafka for flight alerts can assist to increase the precision and timeliness of alerts improving consumer happiness and operational effectiveness for airlines and other airline stakeholders.

Future works:

- As per our project proposal there are no missing components in our implementation.
- In future, we would like to design the front end for our project by using programming languages like HTML, CSS, JavaScript and other frameworks.

References:

[1] Kafka - https://kafka.apache.org/documentation

[2]KafkaArchitecture- https://www.tutorialspoint.com/apache_kafka/apache_kafka_cluster_architecture.htm

[3]Kafka,Zookeeper- https://developers.redhat.com/articles/2022/04/05/developers-guide-using-kafka-java-part-1#kafka_architecture

[4] JDBC - https://www.javatpoint.com/java-jdbc

[5] Related Work blog - https://www.kaiwaehner.de/blog/2021/12/17/kafka-live-commerce-transform-retail-shopping-metaverse/