Kafka Producer and Consumer Implementation Documentation

By Aditya Pandiarajan
adityapandiarajan7@gmail.com

Overview

This document provides a detailed explanation of a Kafka-based data streaming implementation. The system consists of a producer that fetches data from an external API and sends it to Kafka, and a consumer that reads messages from Kafka and uploads them to an AWS S3 bucket.

Kafka Producer Code

Functionality:

The producer fetches data from an API and sends it to a Kafka topic at regular intervals.

Key Components:

1. **Kafka Broker**: 54.211.215.139:9092
2. **Topic Name**: task
3. **API Endpoint**: https://files.polygon.io/api/v1/your_endpoint_here
4. **Message Sending Interval**: 2 seconds

Code Explanation:

1. **Fetch Data from API**:
    - The FetchDataFromAPI(url string) ([]map[string]string, error) function sends an HTTP request to the API.
    - It decodes the JSON response into a slice of maps.
2. **Produce Messages to Kafka**:
    - Establishes a Kafka producer using sarama.NewSyncProducer().
    - Retrieves data from the API.
    - Randomly selects an item from the dataset.
    - Converts the selected data into a JSON message.
    - Sends the message to the Kafka topic task.

- Repeats the process every 2 seconds.

Code Snippet:

```
// Sending message to Kafka
topic := "task"
msg := &sarama.ProducerMessage{
   Topic: topic,
   Value: sarama.StringEncoder(message),
}
producer.SendMessage(msg)
```

Kafka Consumer Code

Functionality:

The consumer reads messages from Kafka and uploads them to an AWS S3 bucket.

Key Components:

1. **Kafka Broker**: 54.211.215.139:9092
2. **Topic Name**: task
3. **Consumer Group**: consumer-group-1
4. **AWS S3 Bucket**: task-aditya
5. **AWS Region**: us-east-1

Code Explanation:

1. **Consume Messages from Kafka**:
   - Establishes a Kafka consumer using sarama.NewConsumerGroup().
   - Reads messages from the topic task.
   - Generates a unique filename for each message.
   - Calls uploadToS3() to store the message as a file in S3.
2. **Upload Messages to S3**:
   - The uploadToS3(fileName string, data []byte) error function:
     - Establishes an AWS session.
     - Uploads the received message to S3 with a unique filename.

Code Snippet:

```
// Uploading message to S3
_, err = s3Client.PutObject(&s3.PutObjectInput{
   Bucket: aws.String(s3BucketName),
```

```
    Key:    aws.String(fileName),
    Body:   bytes.NewReader(data),
    ACL:    aws.String("private"),
})
```

Expected Output

- **Kafka Producer**:
    - Sends messages to Kafka every 2 seconds.
    - Example log output:
    - Message sent: {"symbol":"AAPL", "price":175.23}
- **Kafka Consumer**:
    - Reads messages from Kafka.
    - Uploads them to AWS S3.
    - Example log output:
    - Received message: {"symbol":"AAPL", "price":175.23}
    - Uploaded stock_market_1710963847.json to S3

Conclusion

This implementation successfully demonstrates real-time data streaming using Kafka. The producer fetches stock market data from an API and sends it to Kafka, while the consumer processes the messages and uploads them to AWS S3 for storage.