AWS MSK Kafka to Lambda Integration Using IAM Authentication

# Pipeline Overview

We implemented an event-based trigger on AWS Lambda that was initially successful. When a message arrives on the Kafka topic, Lambda gets triggered automatically to process the message. However, the current requirement is to modify the setup so that the Lambda function runs once a day, connects to MSK, fetches the latest messages from the Kafka topic, and stores them in S3. This change will help aggregate the messages daily for downstream processing.

# Python Approach

Initially, we attempted to implement the solution using Python. The goal was to connect AWS MSK with AWS Lambda using IAM Authentication (SASL/AWS\_MSK\_IAM mechanism). This would ensure secure consumption of Kafka messages within Lambda, and the messages would then be processed for downstream applications like S3.

## Challenges Encountered with Python

Several issues were encountered when using Python with the required libraries. We explored the following approaches and libraries:

1. \*\*GLIBC Error with confluent-kafka\*\*

Error Message:   
{ "errorMessage": "Unable to import module 'lambda\_function': /lib64/libpthread.so.0: version GLIBC\_2.28 not found (required by /var/task/confluent\_kafka/../confluent\_kafka.libs/librdkafka-bc407dcd.so.1)", "errorType": "Runtime.ImportModuleError", "requestId": "", "stackTrace": [] }  
  
This error suggests that the Lambda runtime environment is missing certain native libraries (specifically GLIBC\_2.28) required by confluent-kafka. The issue stems from the Lambda environment not supporting some of the native dependencies required by the confluent-kafka library. This prevents the function from initializing properly.

2. \*\*SASL Mechanism Error with kafka-python\*\*

Error Message:   
{ "errorMessage": "Error processing messages: sasl\_mechanism must be in PLAIN, GSSAPI, OAUTHBEARER, SCRAM-SHA-256, SCRAM-SHA-512", "errorType": "SaslMechanismError", "requestId": "", "stackTrace": [] }  
  
This error occurs when trying to use the kafka-python library with the SASL mechanism for AWS MSK IAM authentication. The library only supports PLAIN, GSSAPI, OAUTHBEARER, SCRAM-SHA-256, and SCRAM-SHA-512, which means the AWS\_MSK\_IAM mechanism isn't supported.

3. \*\*Unrecognized Broker Version Error\*\*

Error Message:   
{ "errorMessage": "UnrecognizedBrokerVersion", "errorType": "UnrecognizedBrokerVersion", "requestId": "", "stackTrace": [] }  
  
This error indicates that the version of the Kafka broker isn't recognized by the Python Kafka client. The issue arises due to compatibility problems between the version of the Kafka client and MSK Kafka brokers. Despite our attempts to update versions, this issue persisted, leading to the decision to explore alternatives.

# Node.js Approach

After facing numerous issues with Python, we switched to Node.js for connecting AWS MSK with Lambda. We used `kafkajs` and `aws-msk-iam-auth` libraries for managing the Kafka connection and IAM-based authentication.

## Challenges Encountered with Node.js

We also faced some challenges with the Node.js approach, including initial errors related to IAM authentication and topic subscription.

1. \*\*IAM Authentication and Kafka Client Issues\*\*

In the beginning, we encountered issues related to establishing a connection with the Kafka client using IAM. The Kafka client had issues identifying the proper authentication mechanism, causing the connection to fail. The `aws-msk-iam-auth` plugin helped resolve this.

2. \*\*Successful Connection with kafkajs and aws-msk-iam-auth\*\*

Once the correct authentication mechanism was configured using `aws-msk-iam-auth`, the connection to the MSK broker was successfully established. The consumer subscribed to the correct topic, and messages were successfully fetched from Kafka.

## Libraries Used in Node.js Approach

1. `kafkajs` - A modern client library for Apache Kafka implemented in Node.js.  
2. `aws-msk-iam-auth` - A plugin for enabling IAM-based authentication with MSK Kafka.

# Final Working Solution

After resolving the issues, we successfully established a connection to MSK, consumed messages from the Kafka topic, and processed them in the Lambda function. The solution involves using Node.js with `kafkajs` and `aws-msk-iam-auth` to establish a secure connection and handle Kafka message consumption effectively.