**Kafka Source Plugin**

**High level Design Document**

Author: Ajeesh Gopalakrishnakurup

Client: AWS

Version Number: 1.0

Release Date: 24-01-2023

**Approving Authority Signatures**

|  |  |  |
| --- | --- | --- |
| **Approver** | **Approval Signature** |  |
|  | Signed:  Date: |  |
|  | Signed:  Date: |  |

**Document History**

Version History

| Version No | Date | Summary of Changes | Authority |
| --- | --- | --- | --- |
| 1.0 | 24-01-2023 | Initial draft version | Ajeesh Gopalakrishnakurup |
|  | 09-02-2023 | Incorporated the review comments | Ajeesh Gopalakrishnakurup |
|  | 05-04-2023 | Incorporated the Multi topic changes | Ajeesh Gopalakrishnakurup |
|  | 19-05-2023 | Incorporated the latest YAML changes | Ajeesh Gopalakrishnakurup |
|  | 25-07-2023 | Incorporated the latest changes | Ajeesh Gopalakrishnakurup |

**Table Of Contents**

[**1.** **Problem Statement** 4](#_Toc141181141)

[**2.** **Introduction** 4](#_Toc141181142)

[**3.** **Requirement & Dependencies** 4](#_Toc141181143)

[**3.1 Functional Requirements** 4](#_Toc141181144)

[**3.2 Non-Functional Requirements** 4](#_Toc141181145)

[**4.** **Out of Scope** 5](#_Toc141181146)

[**5.** **Customer Experience** 5](#_Toc141181147)

[**Use Cases:** 5](#_Toc141181148)

[**6.** **High-Level Design** 6](#_Toc141181149)

[**6.1 Authentication Approach** 8](#_Toc141181150)

[**6.2 Serialization and Deserialization** 9](#_Toc141181151)

[**6.3 Data-prepper instance with Multiple Brokers and Multiple Topics** 11](#_Toc141181152)

[**6.4 Possible Combinations of Broker, Topic, and the Data-prepper instance use**  **cases** 12](#_Toc141181153)

[**6.4 Exception Handling** 13](#_Toc141181154)

[**7.** **Low-Level Design** 15](#_Toc141181155)

[**7.1** **Class Structure Diagram** 16](#_Toc141181156)

[**7.2** **Configuration Settings (Consumer Configs)** 16](#_Toc141181157)

[**7.2.1** **YAML Pipeline Configuration Options** 18](#_Toc141181158)

[**8.** **Dependencies** 19](#_Toc141181159)

[**9.** **Design Alternatives and Decisions** 19](#_Toc141181160)

[**10.** **Risks** 19](#_Toc141181161)

[**11.** **Open Questions** 19](#_Toc141181162)

[**12.** **Acceptance Criteria** 19](#_Toc141181163)

[**13.** **Failure Scenarios** 19](#_Toc141181164)

[**14.** **Integration Test Cases** 20](#_Toc141181165)

[**15.** **Technology Stack** 21](#_Toc141181166)

[**16.** **Metric Types** 22](#_Toc141181167)

[**17.** **Distributed System** 24](#_Toc141181168)

[**18.** **Possible Extensions** 24](#_Toc141181169)

[**19.** **Appendix** 24](#_Toc141181170)

# **Problem Statement**

Implement a source type Data-Prepper plugin to read the topics from Kafka message broker and store it in the Data-prepper buffer.

The requirement is available at <https://github.com/opensearch-project/data-prepper/issues/254>

# **Introduction**

This HLD document will document the technical approach for implementing Kafka source plugin. Following technical areas will be documented:

* **Multi-Threaded Kafka Consumer** - The multi-threaded models might be to process each message from a Kafka topic in a separate thread taken from a thread pool.
* **Authentication** - Kafka Source Plugin will use Simple Authentication and Security Layer (SASL) and here it will support auth mechanisms including PLAIN and OAUTH.
* **Deserialization** - Kafka Source Plugin will provide String deserializer, JSON deserializer and AVRO deserializer. String deserializer will be the default deserializer .

# **Requirement & Dependencies**

## **3.1 Functional Requirements**

1. The Kafka source plugin should be able to support many authentication mechanisms including PLAIN and OAUTH.
2. It should support the authentication protocols like PLAINTEXT, SASL\_PLAINTEXT and SASL\_SSL.
3. Data Prepper should be able to receive events from Kafka using a source which acts as a Kafka consumer.
4. The Kafka consumer should support most or all the consumer configurations.
5. The Kafka consumer should be able to support deserializing objects directly into Events.

## **3.2 Non-Functional Requirements**

*[Scaling and Performance related data need to be included]*

# **Out of Scope**

*[To be filled]*

# **Customer Experience**

## **Use Cases:**

1. Every request from the Kafka source plugin to the Kafka broker should be authenticated or not, based on the user configuration.
2. The Kafka source plugin should be able to read the topic from Kafka.
3. Data transfer between the Kafka source plugin and the Kafka broker should be encrypted.
4. The Kafka source plugin should be able to log the error message or exceptions in case of any exceptions or errors occur.
5. The Kafka plugin should be able to make a thread-safe request to the Kafka broker.
6. The user should be able to configure all or most of the Kafka Consumer properties.
7. There should be no duplicate data in the Data Prepper buffer in the case of multiple instances accessing the same topic.
8. There should be "at-least-once" processing use case need to ensure all the messages in the consumer code (Kafka Source Plugin) are successfully processed before performing another .poll() call.

# **High-Level Design**

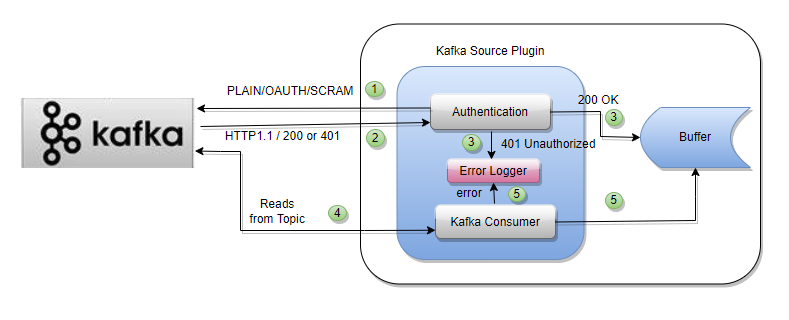


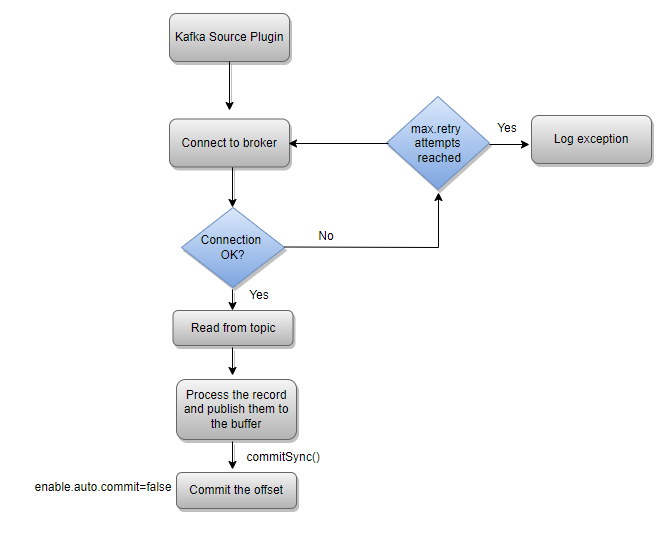
Figure 1: High Level diagram for Kafka plugin

The high-level data flow between the Kafka plugin and the Kafka broker is as follows.

1. Once the Kafka source plugin will be able to establish a connection with the bootstrap server , then the Kafka source will start to read data from the topic and write to the buffer.
2. If the authentication is not successful from the bootstrap server, then write a proper error message to the Data Prepper error logger.
3. **Manual Offset Committing :** Kafka Source Plugin willset the consumer config property

***enable. auto. commit = false*** and will use the Kafka API **commitSync()** which will help the user to commit consumed offsets until it is completed the processing. Hence the Kafka Source Plugin can make sure that when records should be considered as consumed and hence commit their offsets.

The following diagram demonstrates the commit strategy and error handling in Kafka Source Plugin.

****

**Figure 6.4.1** Commit strategy and error handling in Kafka Source Plugin

**The Kafka Source Plugin proposes three strategies to handle failures.**

1. Under normal situation, the Kafka Source Plugin will process each record from the topic and publishes the result to the buffer.
2. Records that cannot be processed, for example, those that don’t have the expected format or are missing required attributes, are logged into the Data Prepper logger and commit the offset. In this case, the failed records never consider for a retry.
3. If an error occurs while processing the records or publishing the records into the buffer, Kafka plugin will send failing messages to another Kafka topic (say retry topic) where a retry instance of a Kafka plugin application periodically attempts to process the events.

## **6.1 Authentication Approach**

* SASL is framework that will be used for the protocols: **PLAINTEXT,** **SASSL\_PLAINTEXT** and **SASL\_SSL** for communicating with the brokers. Default protocol will be **PLAINTEXT**.
* Kafka source will be supporting the **PLAIN** and **AUTHBEARER** authentication mechanisms.
* Securing the Apache Kafka Cluster using SSL, SASL and ACL (Access Control List).

The Java Authentication and Authorization Service (JAAS) API will supply the user authentication and authorization services for the Kafka broker.

***Sample configs for SASL\_OAUTH will look like:***

**kafka\_server\_jaas.conf**  
KafkaServer {  
    org.apache.kafka.common.security.oauthbearer.OAuthBearerLoginModule required  
    unsecuredLoginStringClaim\_sub="admin";  
};  
  
**server.properties**  
listeners=SASL\_SSL://host.name:port  
security.inter.broker.protocol=SASL\_SSL  
sasl.mechanism.inter.broker.protocol=OAUTHBEARER  
sasl.enabled.mechanisms=OAUTHBEARER  
  
**consumer.properties**  
security.protocol=SASL\_SSL (or SASL\_PLAINTEXT if non-production)  
sasl.mechanism=OAUTHBEARER

## **6.2 Serialization and Deserialization**

Kafka Source Plugin will read messages from Kafka topic which are made up of bytes and so a Deserializer will be needed for the consumer to indicate how to transform these bytes back into some objects or data and they will be used on the key and the value of the message.

The Kafka Source Plugin will use a StringDeserializer to transform the bytes into a string, JsonDeserializer to transform the bytes into Json, and Avro Deserializer into Avro formats.

A Schema registry will be used for deserializing Json and Avro data formats.

The package **"org.apache.kafka.common.serialization.<de*SerializerName*>"** helps to do consumer deserialization and the Consumer code that uses this deserializer will look like the below code snippet.

***String Deserializer***

*Properties props = new Properties();*

*props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer")*

*props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer")*

***JSON Deserializer***

ConsumerConfig.***KEY\_DESERIALIZER\_CLASS\_CONFIG***,"org.apache.kafka.common.serialization.

StringDeserializer)

ConsumerConfig.***VALUE\_DESERIALIZER\_CLASS\_CONFIG***,<Json\_Deserializer\_class>.**class**)

***AVRO Deserializer***

props.put(ConsumerConfig.***VALUE\_DESERIALIZER\_CLASS\_CONFIG***,

KafkaAvroDeserializer.**class**.getName())

props.put(KafkaAvroDeserializerConfig.SPECIFIC\_AVRO\_READER\_CONFIG, "true")

props.put(KafkaAvroDeserializerConfig.SCHEMA\_REGISTRY\_URL\_CONFIG, "http://localhost:8081")

***com.fasterxml.jackson.core.\**** package will be used for JSON deserializer and ***io.confluent.kafka.serializers.\**** package will be used for Avro deserializer.

When the object needs to send to Kafka is not always a string or integer, it is a choice of either using a generic serialization library like JSON, Avro, Thrift or Protobuf to create records. The Kafka source plugin will be using existing deserializers such as String, JSON and Apache AVRO.

***Confluent Schema Registry will be used for Apache AVRO deserilaizer to read the records.***

**Flow diagram for Avro records deserialization**

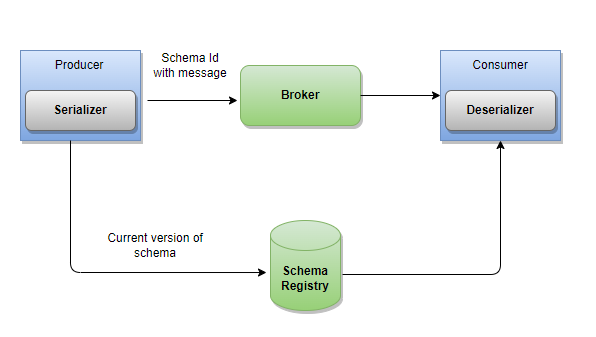
****

Figure:6.2.1 Serialization and deserialization of Avro records

## **6.3 Data-prepper instance with Multiple Brokers and Multiple Topics**

The below diagram explains each consumer groups having two consumers each and both the consumer groups listening on the Kafka Topic A and Topic B. Each broker has two partitions P1&P2 and P3&P4 for the Topic A and Topic B respectively.Consumer1 receives data from partitions P1 and Consumer2 receives data from P2, Consumer3 will be getting data from P3 and Consumer 4 will be getting from partition P4. Two consumers from a consumer group never receives the data from the same partition at the same time.

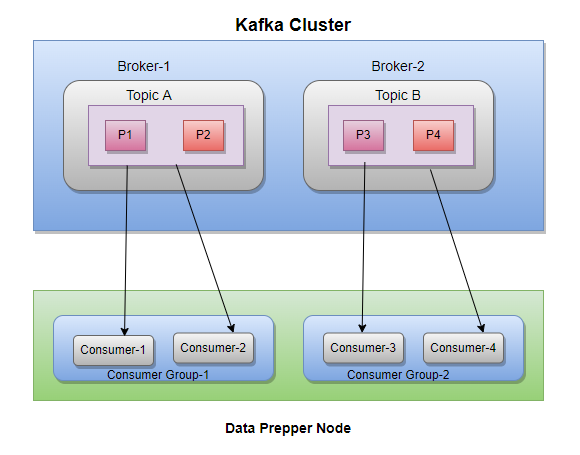
****

Figure 6.3. Multiple consumer groups trying to read data from multiple brokers with multiple topics.

## **6.4 Possible Combinations of Broker, Topic, and the Data-prepper instance use** **cases**

Combination of Single Broker and Single Topic with Single/Multiple Data Prepper Instances

|  |  |  |
| --- | --- | --- |
|  | **Single Broker** | **Single Topic** |
| **Single DP Instance** |  |  |
| **Multiple DP Instance** |  |  |

Combination of Multiple Broker and Multiple Topic with Single/Multiple Data Prepper Instances

|  |  |  |
| --- | --- | --- |
|  | **Multiple Broker** | **Multiple Topic** |
| **Single DP Instance** |  |  |
| **Multiple DP Instance** |  |  |

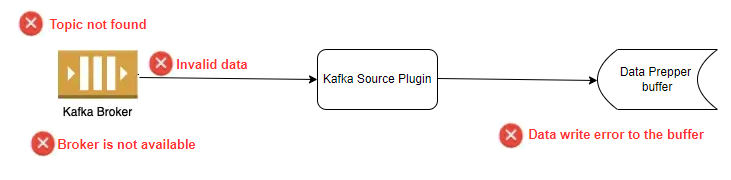
## **6.4 Exception Handling**

The Kafka Source plugin need to deal with certain kinds of exceptions while reading the data from a Kafka topic. Every exception will be logged in and it will be used for future analysis as well.

The following exceptions or error scenarios will be handled in the Kafka source side.

* 1. The Kafka broker is not available.
  2. The authentication fails to the Kafka broker/ clusters.
  3. The authentication fails the Schema Registry
  4. The given topic is not available.
  5. The given Schema registry URL is not available.
  6. The given Schema version is not available.
  7. The user makes any TYPO mistake in the supplied configurations.
  8. Any invalid data while reading from the topic.
  9. Timeout occurs during the Schema registry connectivity.

**A few of the frequently occurred errors while consuming data from Kafka broker.**

****

**Possible errors checklist:** Here is a list of possible errors that may occur while reading the data from the Kafka broker.

[*https://kafka.apache.org/0100/javadoc/org/apache/kafka/common/errors/package-summary.html*](https://kafka.apache.org/0100/javadoc/org/apache/kafka/common/errors/package-summary.html)

**A Retriable Exception**will be used to handle the failure scenarios and the failed read operation can be reattempted. A list of possible retry related exceptions are given in the below URL.

<https://kafka.apache.org/31/javadoc/org/apache/kafka/common/errors/RetriableException.html>

***Summarize the technical parameters detail in a nutshell to read data from a Kafka topic.***

|  |  |
| --- | --- |
| Supported message format | Json /String/Avro |
| Deserializer in considerations | Json/String/Avro |
| Auth mechanisms | PLAIN and AUTHBEARER |
| Auth protocols | PLAINTEXT, SASL\_PLAINTEXT and SASL\_SSL |
| Configurations to read Kafka topic | Most or all of the Consumer Configs (see Section 7.2 Configuration Settings) |
|  |  |

***Kafka source plugin will be using Schema Registry for deserializing Jason and Avro data formats.***

# **Low-Level Design**

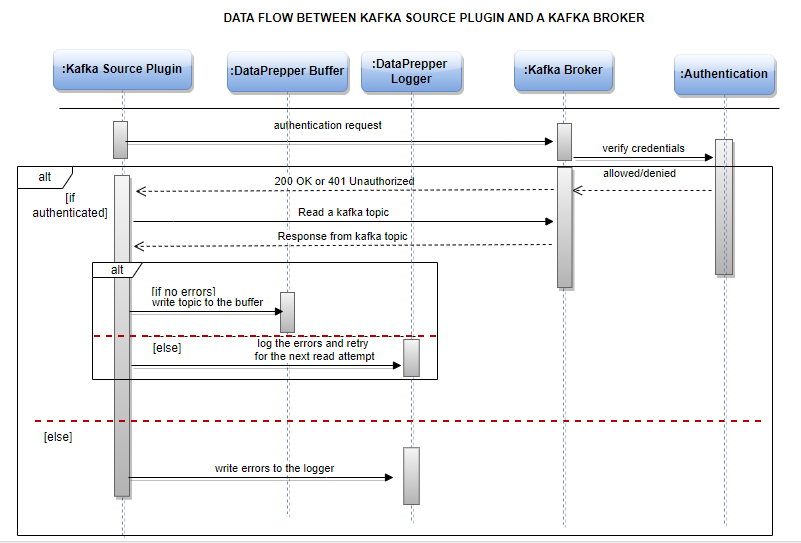
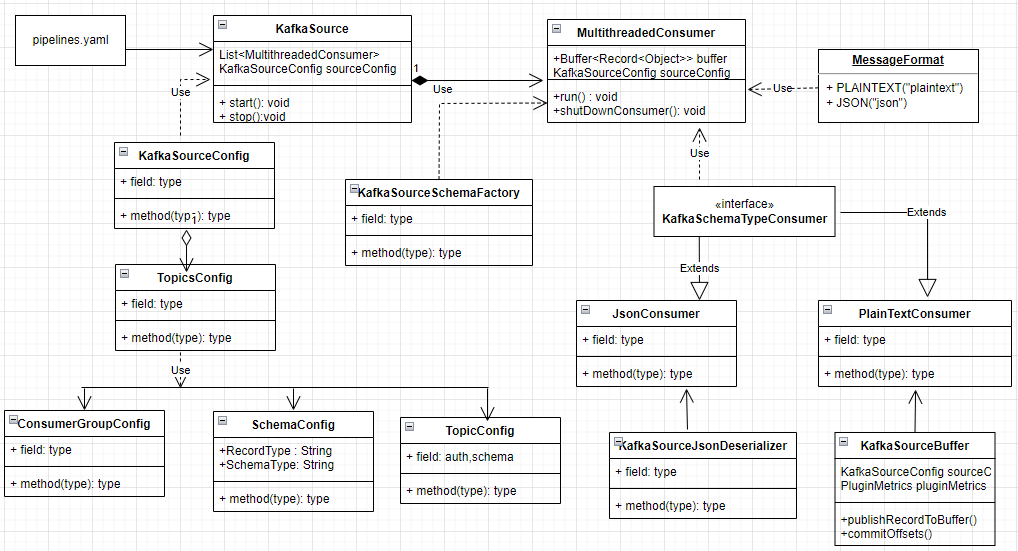


Figure 2: Data flow from a client to Kafka broker

1. The Kafka Source Plugin will be sending the auth credentials (auth token request to the auth server in the case of OAuth) to the Kafka broker.
2. If the authentication is a success, proceed with the read from the Kafka topic, otherwise, log the authentication failure exception.
3. Process the message from the topic and publish it to the buffer if there are no errors.
4. commit the latest offset returned from the Kafka topic and throwing an exception if commit fails for some reason
5. If there are any errors while processing the message or any IO exception while publishing the message to the buffer, log the exception and retry for the next read attempt.

## **Class Structure Diagram**



## **Configuration Settings (Consumer Configs)**

|  |  |  |  |
| --- | --- | --- | --- |
| No | **Configurations** | **Description** | **Importance** |
| 1 | key\_deserializer | Deserializer class for key | High |
| 2 | [value\_deserializer](https://kafka.apache.org/documentation/#consumerconfigs_value.deserializer) | Deserializer class for value | High |
| 3 | [bootstrap\_servers](https://kafka.apache.org/documentation/#consumerconfigs_bootstrap.servers) | A list of host/port pairs to use for establishing the initial connection to the Kafka cluster. | High |
| 4 | [fetch\_min\_bytes](https://kafka.apache.org/documentation/#consumerconfigs_fetch.min.bytes) | The minimum amount of data the server should return for a fetch request. | High |
| 5 | [group\_id](https://kafka.apache.org/documentation/#consumerconfigs_group.id) | A unique string that identifies the consumer group this consumer belongs to. | High |
| 6 | [heartbeat\_interval\_ms](https://kafka.apache.org/documentation/#consumerconfigs_heartbeat.interval.ms) | The expected time between heartbeats to the consumer coordinator when  using Kafka's group management facilities | High |
| 8 | [session\_timeout\_ms](https://kafka.apache.org/documentation/#consumerconfigs_session.timeout.ms) | The timeout used to detect client failures when using Kafka's group management facility. | High |
| 9 | [enable\_auto\_commit](https://kafka.apache.org/documentation/#consumerconfigs_enable.auto.commit) | If true, the consumer's offset will be periodically committed in the background. | Medium |
| 10 | [fetch\_max\_bytes](https://kafka.apache.org/documentation/#consumerconfigs_fetch.max.bytes) | The maximum amount of data the server should return for a fetch request | Medium |
| 11 | [max\_poll\_interval.ms](https://kafka.apache.org/documentation/#consumerconfigs_max.poll.interval.ms) | The maximum delay between invocations of poll () when using consumer group management | Medium |
| 12 | [max\_poll\_records](https://kafka.apache.org/documentation/#consumerconfigs_max.poll.records) | The maximum number of records returned in a single call to poll () | Medium |
| 13 | security\_protocol | Protocol used to communicate with brokers are PLAINTEXT, SASL\_PLAINTEXT, SASL\_SSL | Medium |
| 14 | client\_id | An id string to pass to the server when making requests | Low |
| 15 | fetch\_max\_wait\_ms | The maximum amount of time the server will block before answering the fetch request | Low |
| 16 | retry\_backoff\_ms | The amount of time to wait before attempting to retry a failed request to a given topic partition | Low |

### **YAML Pipeline Configuration Options**

log-pipeline:  
 source:  
 kafka:  
 bootstrap\_servers:  
 - pkc-ymrq7.us-east-2.aws.confluent.cloud:9092  
 client\_dns\_lookup: use\_all\_dns\_ips  
 encryption: plaintext  
 topics:  
 - name: topic-json  
 workers: 5  
 auto\_commit: false  
 auto\_commit\_interval: PT5S  
 session\_timeout: 45000  
 max\_retry\_attempts: 1000  
 auto\_offset\_reset: earliest  
 thread\_waiting\_time: PT1S  
 max\_record\_fetch\_time: PT4S  
 heart\_beat\_interval: PT3S  
 buffer\_default\_timeout: PT5S  
 fetch\_max\_bytes: 52428800  
 fetch\_max\_wait: 500  
 fetch\_min\_bytes: 1  
 retry\_backoff: PT100S  
 consumer\_max\_poll\_records: 500  
 schema:  
 registry\_url: https://psrc-em82q.us-east-2.aws.confluent.cloud  
 version: 1  
 schema\_registry\_api\_key: 7QV2UXHRVNOC6AJD  
 schema\_registry\_api\_secret: 6M9xLZDIfmyBN9cqNm2n9GU23mleiaIHJWqQeA5P4JY   
 basic\_auth\_credentials\_source: USER\_INFO  
 session\_timeout\_ms: 45000  
 aws:  
 msk:  
 arn: service\_arn  
 broker\_connection\_type: public  
 region: us-east-2  
 sts\_role\_arn: sts\_role\_arn  
 authentication:  
 insecure: true/false  
 sasl:  
 msk\_iam: role  
 plaintext:  
 security\_protocol: SASL\_SSL  
 username: 5UH4NID4OENKDIBI  
 password: jCmncn77F9asfox3yhgZLCEwQ5fx8pKiXnszMqdt0y1GLrdZO1V1iz95aIe1UubX  
 oauth:  
 oauth\_client\_id: 0oa9wc21447Pc5vsV5d7  
 oauth\_client\_secret: aGmOfHqIEvBJGDxXAOOcatiE9PvsPgoEePx8IPPa  
 oauth\_login\_server: https://dev-13650048.okta.com  
 oauth\_login\_endpoint: /oauth2/default/v1/token  
 oauth\_login\_grant\_type: refresh\_token  
 oauth\_login\_scope: kafka  
 oauth\_introspect\_server: https://dev-13650048.okta.com  
 oauth\_introspect\_endpoint: /oauth2/default/v1/introspect  
 oauth\_token\_endpoint\_url: https://dev-13650048.okta.com/oauth2/default/v1/token  
 oauth\_security\_protocol: SASL\_SSL  
 oauth\_sasl\_mechanism: OAUTHBEARER  
 oauth\_sasl\_login\_callback\_handler\_class: org.apache.kafka.common.security.oauthbearer.secured.OAuthBearerLoginCallbackHandler  
 oauth\_jwks\_endpoint\_url: https://dev-13650048.okta.com/oauth2/default/v1/keys  
 extension\_logicalCluster: lkc-yggz7j  
 extension\_identityPoolId: pool-RXzn  
 sink:  
 - stdout:

# **Dependencies**

1.Kafka details will be provided by the client to execute the git hub action.

# **Design Alternatives and Decisions**

N/A

# **Risks**

N/A

# **Open Questions**

# **Acceptance Criteria**

1. 85% Code coverage.
2. Successfully perform integration testcases for Kafka Source Plugin.

# **Failure Scenarios**

This section will list down the various failure scenarios that will be handled later.

**Single Node Instance Failure** : When there will be only a single node, with a single consumer, whether the node fails and restarts, or the poll times out, the result is the same. Once the consumer is restarted, this consumer will re-join the consumer group, and it will receive the same batch of messages again.

**Broker Connection Failure**: Kafka Source Plugin is trying to consume data from the Kafka topic but the connection to broker is not established because the broker is not available.

**Poll Timeout**: Processing of each batch of messages consumed on a poll must complete within the **max.poll.interval.ms** value. If this does not complete in time, then the broker has to consider that the consumer has died, and therefore trigger a consumer group rebalance. Then the topic partition is assigned to the other consumer in the consumer group and then consumes the duplicate batch of messages.

# **Integration Test Cases**

1. **Kafka Source Plugin Authentication**

**Positive Scenario :**

Check the authentication to the Kafka broker with the below SASL auth mechanisms and the user should be able to establish a connection to the broker.

* + PLAIN
  + OAuth

**Negative Scenario:**

By adding empty auth configurations for PLAIN, and OAuth and the user shouldn’t be able to login with the broker.

1. **Read records from a Kafka topic.**

**Positive Scenarios**

Check the Kafka broker connection with the consumer configurations to read from a topic and ensure the following workflow shouldn’t break.

* Kafka Source Plugin should be able to read the data from a topic and able to update the offset.
* Kafka Source Plugin should be able to deserialize the message in the expected format.

**Negative Scenarios**

Check the Kafka broker connection with the empty consumer configurations to read from a topic and ensure the following workflow.

* Kafka Source Plugin should not be able to read the data from a topic and should follow a retry attempt
* When retry attempts reached the threshold, log the exception.

# **Technology Stack**

|  |  |  |
| --- | --- | --- |
| **No** | **Tools** | **Version** |
| 1 | Data Prepper | 2.2 |
| 2 | Kafka | 2.13 |
| 3 | TLS | 1.2 |
| 4 | Java | 11 or 17 |
| 5 | Gradle | 7.6 |
| 6 | Junit | 5.0 |
| 7 | Mockito | 2.0 |
| 8 | Zookeeper | 3.6 |
| 9 | Docker | 20.10 |
| 10 | AWS SDK | 2.17 |
| 11 | OAuth | 2.0 |

# **Metric Types**

The following metrics are available on consumer instances.

1. **Consumer Monitoring Metrics**

|  |  |  |
| --- | --- | --- |
| **METRIC/ATTRIBUTE NAME** | **DESCRIPTION** | **MBEAN NAME** |
| timeBetweenPollAvg | The average delay between invocations of poll(). | kafka.consumer:type=consumer-metrics,client-id=([-.\w]+) |
| timeBetweenPollMax | The max delay between invocations of poll(). | kafka.consumer:type=consumer-metrics,client-id=([-.\w]+) |
| lastPollSecondsAgo | The number of seconds since the last poll() invocation. | kafka.consumer:type=consumer-metrics,client-id=([-.\w]+) |
| pollIdleRatioAvg | The average fraction of time the consumer's poll() is idle as opposed to waiting for the user code to process records. | kafka.consumer:type=consumer-metrics,client-id=([-.\w]+) |
| commitedTimeNsTotal | The total time the Consumer spent in committed in nanoseconds. | kafka.consumer:type=consumer-metrics,client-id=([-.\w]+) |
| commitSyncTimeNsTotal | The total time the Consumer spent committing offsets in nanoseconds (for AOS). | kafka.consumer:type=consumer-metrics,client-id=([-.\w]+) |
| commitLatencyAvg | The average time taken for a commit request | kafka.consumer:type=consumer-coordinator-metrics,client-id=([-.\w]+) |
| commitLatencyMax | The max time taken for a commit request | kafka.consumer:type=consumer-coordinator-metrics,client-id=([-.\w]+) |
| commitRate | The number of commit calls per second | kafka.consumer:type=consumer-coordinator-metrics,client-id=([-.\w]+) |
| commitTotal | The total number of commit calls | kafka.consumer:type=consumer-coordinator-metrics,client-id=([-.\w]+) |
| assignedPartitions | The number of partitions currently assigned to this consumer | kafka.consumer:type=consumer-coordinator-metrics,client-id=([-.\w]+) |
| heartbeatResponseTimeMax | The max time taken to receive a response to a heartbeat request | kafka.consumer:type=consumer-coordinator-metrics,client-id=([-.\w]+) |
| heartbeatRate | The average number of heartbeats per second | kafka.consumer:type=consumer-coordinator-metrics,client-id=([-.\w]+) |
| heartbeatTotal | The total number of heartbeats | kafka.consumer:type=consumer-coordinator-metrics,client-id=([-.\w]+) |
| joinTotal | The total number of group joins | kafka.consumer:type=consumer-coordinator-metrics,client-id=([-.\w]+) |
| rebalanceTotal | The total number of group rebalances participated | kafka.consumer:type=consumer-coordinator-metrics,client-id=([-.\w]+) |
| rebalanceRatePerHour | The number of group rebalance participated per hour | kafka.consumer:type=consumer-coordinator-metrics,client-id=([-.\w]+) |
| failedRebalanceTotal | The total number of failed group rebalances | kafka.consumer:type=consumer-coordinator-metrics,client-id=([-.\w]+) |
| failedRebalanceRatePerHour | The number of failed group rebalance event per hour | kafka.consumer:type=consumer-coordinator-metrics,client-id=([-.\w]+) |

More information will be available on <https://kafka.apache.org/documentation/#monitoring>

# **Distributed System**

A Consumer group will be taking care of the distributed functionality in Kafka. A Consumer should be part of a consumer group, when the consumer performing an operation needs to be scaled up to process in parallel.

Each consumer is part of a group would be processing different data than the other consumers within the same group. This is one of the ways suggested by Kafka to achieve parallel processing in consumers.

# **Possible Extensions**

# **Appendix**