10/31/2022

Muhammad Osama U41261655

Apache Kafka

CS 777 Term Paper Report

Table of Contents

[Kafka Introduction: 2](#_Toc118135076)

[Why Kafka? 2](#_Toc118135077)

[Apache Kafka: Use cases 4](#_Toc118135078)

[Examples: 4](#_Toc118135079)

[Kafka Components 5](#_Toc118135080)

[Code 13](#_Toc118135081)

Table of Figures

[Figure 1: Data Transfer without Apache Kafka 2](#_Toc118135133)

[Figure 2: Data Transfer with Apache Kafka 3](#_Toc118135134)

[Figure 3: Apache Kafka Uses 3](#_Toc118135135)

[Figure 4: Kafka Architecture 5](#_Toc118135136)

[Figure 5: Kafka Topic 5](#_Toc118135137)

[Figure 6: Kafka Producer 6](#_Toc118135138)

[Figure 7: Message Format 7](#_Toc118135139)

[Figure 8: Kafka Consumer 8](#_Toc118135140)

[Figure 9: Kafka Consumer Group 9](#_Toc118135141)

[Figure 10: Kafka Broker 10](#_Toc118135142)

[Figure 11: Topic Replication & Leader Broker 11](#_Toc118135143)

[Figure 12: Zookeeper Ensemble 12](#_Toc118135144)

# Kafka Introduction:

Kafka is a distributed event store and stream processing platform. It is an open-source system developed by the Apache Software Foundation and is written in Java and Scala. The aim of Kafka is to provide unified, high-throughput, low-latency platform for handling real-time data. Kafka can connect to external systems (for data import/export) via Kafka Connect and provides the Kafka Streams libraries for stream processing applications. Kafka uses a binary TCP-based protocol that is optimized for efficiency and relies on a "message set" abstraction that naturally groups messages together to reduce the overhead of the network roundtrip.

# Why Kafka?

Companies have different source systems from where they ingest data and then transform it and load it into target systems where the data is used. This can be seen from the image below, in which we have 4 source systems and 4 target systems. In this case, we will have to write 16 integrations as each target system is taking data from each source system. Each integration will come with difficulties around:

1. Protocol – how the data is transported
2. Data format – how the data is parsed
3. Data scheme & evolution – how the data is shipped and may change

Due to these integrations, each source system will be having an increased load from the connections.

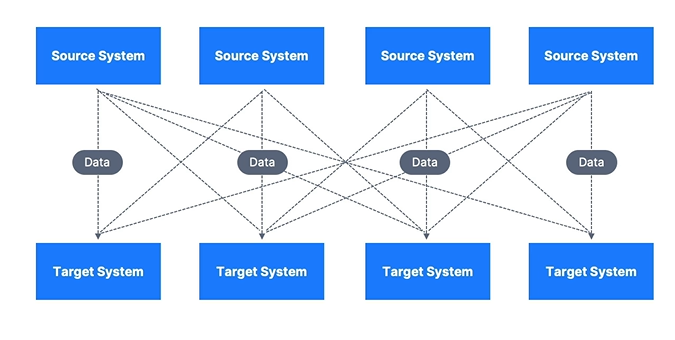


Figure 1: Data Transfer without Apache Kafka

This problem is now solved by Kafka. The image below shows Apache Kafka as an intermediary between the source and the target systems.

Diagram

Description automatically generated

Figure 2: Data Transfer with Apache Kafka

Diagram

Description automatically generated

Figure 3: Apache Kafka Uses

Now the source systems are responsible for sending data which is called producing data into Apache Kafka. So now, Apache Kafka has a stream of data of from all the source systems within it and if the target systems need to receive data, they can tap into the data of Apache Kafka. This is called consuming data from Apache Kafka. The purpose of Apache Kafka is basically to send and receive data like a transportation tool.

Following are some of the properties of Apache Kafka:

1. Created by LinkedIn, now, Open-Source Project mainly maintained by Confluent, IBM, Cloudera.
2. Distributed, resilient architecture, fault tolerant
3. Horizontal Scalability:
   1. Can scale to 100s of brokers.
   2. Can scale to millions of messages per second
4. High performance (latency of less than 10ms) – real time
5. Used by 2000+ firms, 80% of the Fortune 100

# Apache Kafka: Use cases

1. Messaging System
2. Activity Tracking
3. Gather metrics from many different locations
4. Application Logs gathering
5. Stream processing
6. De-coupling of system dependencies
7. Integration with Spark, Flink, Storm, Hadoop and many other Big Data technologies
8. Micro-services

# Examples:

1. Netflix uses Kafka to apply recommendations in real-time while you are watching TV shows.
2. Uber uses Kafka to gather user, taxi, and trip data in real-time to compute and forecast demand and compute surge pricing in real-time.
3. LinkedIn uses Kafka to prevent spam, collect user interactions to make better connection recommendations in real time.

# Kafka Components

Diagram

Description automatically generated

Figure 4: Kafka Architecture

1. **Kafka Topic**

Table

Description automatically generated

Figure 5: Kafka Topic

* 1. A stream of data.
  2. Defined by its name.
  3. Has partitions into which the messages are stored. The messages in each of the partitions are ordered. The messages in a partition have an incremental id, called the offset.
  4. Kafka topics are immutable, once data is written to a partition, it cannot be changed.
  5. Data is kept only for a limited amount of time (default is one week).
  6. Order is guaranteed only within a partition (not across partitions).
  7. Data is assigned randomly to a partition unless a key is provided.

1. **Kafka Producer**

Graphical user interface, application

Description automatically generated

Figure 6: Kafka Producer

* 1. Producers write data to topics (which are made of partitions).
  2. Producers know to which partitions to write (and which Kafka Broker has it).
  3. In case of Kafka broker failure, the Producers will automatically recover.
  4. Producer has different partition strategies to distribute the data across the partitions. If the messages sent have a key, then the messages with the same key will end up on the same partitions. However, if the key is null then the data is sent in a round robin fashion (partition 0, then 1, then 2…).

Table

Description automatically generated

Figure 7: Message Format

* 1. Kafka only accepts bytes as an input from the producers and sends bytes as an output to the consumer. Therefore, we have message serialization which is transforming data into bytes. This is applied on the message key and value.
  2. Kafka partitioner is a code logic that takes a record and determines to which partition it is sent to.
  3. Key hashing is the process of determining the mapping of a key to a partition.
  4. In the default Kafka partitioner keys are hashed using the murmur2 algorithm.
  5. Producer can choose to receive acknowledgement of data writes. Acks = 0: producer will not wait for acknowledgement, acks = 1: producer will wait for leader acknowledgement, acks = all: leader and the replica acknowledgement.

1. **Kafka Consumer**

Graphical user interface, application, Teams

Description automatically generated

Figure 8: Kafka Consumer

* 1. Consumers read data from a topic (identified by name) – pull model.
  2. Consumers automatically know which broker to read from.
  3. In case of a broker failure, consumers know how to recover.
  4. Data is read in order from low to high from each partition.
  5. Deserializer is used to transform the messages from bytes into objects / data. They are used on the key and the value.
  6. The Consumer needs to know in advance the key and the value type. Therefore, within the life cycle of the topic, the serialization / deserialization type must not change.

1. **Kafka Consumer Group**

Timeline

Description automatically generated with low confidence

Figure 9: Kafka Consumer Group

1. All the consumers reading from a topic make up a consumer group.
2. Each consumer within a group reads from exclusive partitions.
3. If you have more consumers in a consumer group than the number of partitions in the topic from which the data is being read, then the extra consumers sit in an idle state.
4. There can be multiple consumer groups reading from the same topic.
5. Kafka stores the offsets at which a consumer group has been reading.
6. The offsets committed are in a Kafka topic named \_\_consumer\_offsets.
7. When a consumer in a group has processed data received from Kafka, it should be periodically committing the offsets.
8. If a consumer dies, it will be able to read back from where it left off using the committed offsets.
9. By default, the offsets are committed at least once. There are 3 delivery semantics if you want to commit manually:
   * 1. At least once: offsets are committed after the messages are processed. This can result in duplicate processing of messages. We must make sure that the processing is idempotent (processing the messages again, will not impact the system).
     2. At most once: offsets are committed once the messages have been read. This means that if the processing goes wrong, then they would not be read again, and data will be lost.
     3. Exactly one: This is achieved using the transactional API if you go from Kafka to Kafka workflows or using an idempotent consumer if you go from Kafka to an external system.
10. **Kafka Broker**

Graphical user interface, application

Description automatically generated

Figure 10: Kafka Broker

* 1. A Kafka server is called a Kafka broker because it sends and receives data.
  2. A Kafka cluster has multiple brokers.
  3. Each broker is identified with its ID which is an integer.
  4. Each broker contains topic partitions.
  5. After connecting to any broker (called a bootstrap broker), you can connect to the entire cluster.
  6. Each Kafka broker is called a “bootstrap server”.
  7. Each broker knows about all the brokers, topics, and partitions (metadata).

1. **Topic replication factor**

Diagram

Description automatically generated

Figure 11: Topic Replication & Leader Broker

* 1. Topics have a replication factor which is usually greater than 1. This means that the data is backed up on other brokers.
  2. This way if a broker goes down, another broker can server that data.
  3. In Kafka we have the concept of a leader of a partition. Only one broker can be a leader for a given partition. The other brokers have the replica of that partition.
  4. Producers only send data to the broker that is the leader of a partition and consumers can only read data from the leader broker for a partition.
  5. Since Kafka 2.4, it is possible to configure consumers to read from the closest replica as this helps improve latency and decrease network costs if using the cloud.

1. **Zookeeper**

Diagram

Description automatically generated

Figure 12: Zookeeper Ensemble

* 1. It is a software, and it manages the Kafka brokers.
  2. It has a list of the Kafka brokers.
  3. It also helps with leader election when a Kafka broker goes down.
  4. It also sends notifications to Kafka brokers in case of changes for example, deletion of a topic, when a broker goes down, etc.
  5. There can be multiple zookeepers but only one is a leader and the rest are the followers. The followers only have read access.
  6. Zookeeper is not as secure as Kafka, and Kafka is now moving to be run without Zookeeper.

# Code

**Basic code, starting a producer and sending Hello World in a for loop:**

package io.conductor.demos.kafka;

import org.apache.kafka.clients.producer.\*;

import org.apache.kafka.common.serialization.StringSerializer;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import java.util.Properties;

public class ProducerDemoWithCallback {

private static final Logger log = LoggerFactory.getLogger(ProducerDemoWithCallback.class.getSimpleName());

public static void main(String[] args) {

log.info("I am a Kafka Producer");

// create Producer Properties

Properties properties = new Properties();

properties.setProperty(ProducerConfig.BOOTSTRAP\_SERVERS\_CONFIG, "127.0.0.1:9092");

properties.setProperty(ProducerConfig.KEY\_SERIALIZER\_CLASS\_CONFIG, StringSerializer.class.getName());

properties.setProperty(ProducerConfig.VALUE\_SERIALIZER\_CLASS\_CONFIG, StringSerializer.class.getName());

properties.setProperty(ProducerConfig.PARTITIONER\_CLASS\_CONFIG, RoundRobinPartitioner.class.getName());

// create the Producer

KafkaProducer<String, String> producer = new KafkaProducer<>(properties);

// send data - asynchronous

for (int i =0; i<10; i++) {

// Producer Record

ProducerRecord<String, String> producerRecord =

new ProducerRecord<>("demo\_java", "hello world " + i);

producer.send(producerRecord, new Callback() {

@Override

public void onCompletion(RecordMetadata metadata, Exception e) {

// executes every time a record is successfully sent or an exception is thrown

if (e == null) {

// the record was successfully sent

log.info("Received new metadata/ \n" +

"Topic: " + metadata.topic() + "\n" +

"Partition: " + metadata.partition() + "\n" +

"Offset: " + metadata.offset() + "\n" +

"Timestamp: " + metadata.timestamp());

} else {

log.error("Error while producing", e);

}

}

});

try {

Thread.sleep(5000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

// flush and close

producer.flush();

// flush and close

producer.close();

}

}

**Basic Code, starting a consumer and reading data from topic:**

package io.conductor.demos.kafka;

import org.apache.kafka.clients.consumer.ConsumerConfig;

import org.apache.kafka.clients.consumer.ConsumerRecord;

import org.apache.kafka.clients.consumer.ConsumerRecords;

import org.apache.kafka.clients.consumer.KafkaConsumer;

import org.apache.kafka.common.errors.WakeupException;

import org.apache.kafka.common.serialization.StringDeserializer;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import java.time.Duration;

import java.util.Arrays;

import java.util.Properties;

public class ConsumerDemoWithShutdown {

private static final Logger log = LoggerFactory.getLogger(ConsumerDemoWithShutdown.class.getSimpleName());

public static void main(String[] args) {

log.info("I am a Kafka Consumer");

String bootstrapServers = "127.0.0.1:9092";

String groupId = "my-third-application";

String topic = "demo\_java";

// create consumer configs

Properties properties = new Properties();

properties.setProperty(ConsumerConfig.BOOTSTRAP\_SERVERS\_CONFIG, bootstrapServers);

properties.setProperty(ConsumerConfig.KEY\_DESERIALIZER\_CLASS\_CONFIG, StringDeserializer.class.getName());

properties.setProperty(ConsumerConfig.VALUE\_DESERIALIZER\_CLASS\_CONFIG, StringDeserializer.class.getName());

properties.setProperty(ConsumerConfig.GROUP\_ID\_CONFIG, groupId);

properties.setProperty(ConsumerConfig.AUTO\_OFFSET\_RESET\_CONFIG, "earliest");

// create consumer

KafkaConsumer<String, String> consumer = new KafkaConsumer<>(properties);

// get a reference to the current thread

final Thread mainThread = Thread.currentThread();

// adding the shutdown hook

Runtime.getRuntime().addShutdownHook(new Thread() {

public void run() {

log.info("Detected a shutdown, let's exit by calling consume.wakeup()...");

consumer.wakeup();

// join the main thread and allow the execution of the code in the main thread

try {

mainThread.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

});

try {

// subscribe consumer to out topics

consumer.subscribe(Arrays.asList(topic));

// poll for new data

while (true) {

ConsumerRecords<String, String> records =

consumer.poll(Duration.ofMillis(1000));

for (ConsumerRecord<String, String> record: records) {

log.info("Keys: " + record.key() + ", Value: " + record.value());

log.info("Partition: " + record.partition() + ", Offset: " + record.offset());

}

}

} catch (WakeupException e) {

log.info("Wake up exception!");

// we ignore tas this is an expected exception

} catch (Exception e) {

log.error("Unexpected exception");

} finally {

consumer.close(); // this will also commit the offsets if need be

log.info("The consumer is now gracefully closed");

}

}

}

**Advanced Code, producing data into Kafka from a stream (Wikimedia):**

**Implementing Event Handler to read data from a stream in Java:**

package io.conduktor.demos.kafka;

import com.launchdarkly.eventsource.EventHandler;

import com.launchdarkly.eventsource.MessageEvent;

import org.apache.kafka.clients.producer.KafkaProducer;

import org.apache.kafka.clients.producer.ProducerRecord;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class WikimediaChangeHandler implements EventHandler {

KafkaProducer<String, String> kafkaProducer;

String topic;

private final Logger log = LoggerFactory.getLogger(WikimediaChangeHandler.class.getSimpleName());

public WikimediaChangeHandler(KafkaProducer<String, String> kafkaProducer, String topic) {

this.kafkaProducer = kafkaProducer;

this.topic = topic;

}

@Override

public void onOpen() throws Exception {

// nothing here

}

@Override

public void onClosed() {

kafkaProducer.close();

}

@Override

public void onMessage(String event, MessageEvent messageEvent) {

log.info(messageEvent.getData());

// asynchronous

kafkaProducer.send(new ProducerRecord<>(topic, messageEvent.getData()));

}

@Override

public void onComment(String comment) {

// nothing here

}

@Override

public void onError(Throwable t) {

log.error("Error in Stream Reading", t);

}

}

**Implementing the producer to produce data into Kafka:**

package io.conduktor.demos.kafka;

import com.launchdarkly.eventsource.EventHandler;

import com.launchdarkly.eventsource.EventSource;

import org.apache.kafka.clients.producer.KafkaProducer;

import org.apache.kafka.clients.producer.ProducerConfig;

import org.apache.kafka.common.serialization.StringSerializer;

import java.net.URI;

import java.util.Properties;

import java.util.concurrent.TimeUnit;

public class WikimediaChangesProducer {

public static void main(String[] args) throws InterruptedException {

String bootstrapServers = "127.0.0.1:9092";

// create producer properties

Properties properties = new Properties();

properties.setProperty(ProducerConfig.BOOTSTRAP\_SERVERS\_CONFIG, "127.0.0.1:9092");

properties.setProperty(ProducerConfig.KEY\_SERIALIZER\_CLASS\_CONFIG, StringSerializer.class.getName());

properties.setProperty(ProducerConfig.VALUE\_SERIALIZER\_CLASS\_CONFIG, StringSerializer.class.getName());

// create the Producer

KafkaProducer<String, String> producer = new KafkaProducer<>(properties);

String topic = "wikimedia.recentchange";

EventHandler eventHandler = new WikimediaChangeHandler(producer, topic);

String url = "https://stream.wikimedia.org/v2/stream/recentchange";

EventSource.Builder builder = new EventSource.Builder(eventHandler, URI.create(url));

EventSource eventSource = builder.build();

// start the producer in another thread

eventSource.start();

// we produce for 10 minutes and block the program until then

TimeUnit.MINUTES.sleep(10);

}

}

**Advanced Code, reading data from Kafka and sending to OpenSearch:**

package io.conductor.demos.kafka.opensearch;

import org.apache.http.HttpHost;

import org.apache.http.auth.AuthScope;

import org.apache.http.auth.UsernamePasswordCredentials;

import org.apache.http.client.CredentialsProvider;

import org.apache.http.impl.client.BasicCredentialsProvider;

import org.apache.http.impl.client.DefaultConnectionKeepAliveStrategy;

import org.apache.kafka.clients.consumer.ConsumerConfig;

import org.apache.kafka.clients.consumer.ConsumerRecord;

import org.apache.kafka.clients.consumer.ConsumerRecords;

import org.apache.kafka.clients.consumer.KafkaConsumer;

import org.apache.kafka.common.serialization.StringDeserializer;

import org.opensearch.action.index.IndexRequest;

import org.opensearch.action.index.IndexResponse;

import org.opensearch.client.RequestOptions;

import org.opensearch.client.RestClient;

import org.opensearch.client.RestHighLevelClient;

import org.opensearch.client.indices.CreateIndexRequest;

import org.opensearch.client.indices.GetIndexRequest;

import org.opensearch.common.xcontent.XContentType;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import java.io.IOException;

import java.net.URI;

import java.time.Duration;

import java.util.Collections;

import java.util.Properties;

public class OpenSearchConsumer {

public static RestHighLevelClient createOpenSearchClient() {

// String connString = "http://localhost:9200";

String connString = "https://h8rdppngpr:8i1amt38md@kafka-course-8587512405.us-east-1.bonsaisearch.net:443";

// we build a URI from the connection string

RestHighLevelClient restHighLevelClient;

URI connUri = URI.create(connString);

// extract login information if it exists

String userInfo = connUri.getUserInfo();

if (userInfo == null) {

// REST client without security

restHighLevelClient = new RestHighLevelClient(RestClient.builder(new HttpHost(connUri.getHost(), connUri.getPort(), "http")));

} else {

// REST client with security

String[] auth = userInfo.split(":");

CredentialsProvider cp = new BasicCredentialsProvider();

cp.setCredentials(AuthScope.ANY, new UsernamePasswordCredentials(auth[0], auth[1]));

restHighLevelClient = new RestHighLevelClient(

RestClient.builder(new HttpHost(connUri.getHost(), connUri.getPort(), connUri.getScheme()))

.setHttpClientConfigCallback(

httpAsyncClientBuilder -> httpAsyncClientBuilder.setDefaultCredentialsProvider(cp)

.setKeepAliveStrategy(new DefaultConnectionKeepAliveStrategy())));

}

return restHighLevelClient;

}

private static KafkaConsumer<String, String> createKafkaConsumer() {

String bootstrapServers = "127.0.0.1:9092";

String groupId = "consumer-opensearch-demo";

// create consumer configs

Properties properties = new Properties();

properties.setProperty(ConsumerConfig.BOOTSTRAP\_SERVERS\_CONFIG, bootstrapServers);

properties.setProperty(ConsumerConfig.KEY\_DESERIALIZER\_CLASS\_CONFIG, StringDeserializer.class.getName());

properties.setProperty(ConsumerConfig.VALUE\_DESERIALIZER\_CLASS\_CONFIG, StringDeserializer.class.getName());

properties.setProperty(ConsumerConfig.GROUP\_ID\_CONFIG, groupId);

properties.setProperty(ConsumerConfig.AUTO\_OFFSET\_RESET\_CONFIG, "latest");

// create consumer

return new KafkaConsumer<>(properties);

}

public static void main(String[] args) throws IOException {

Logger log = LoggerFactory.getLogger(OpenSearchConsumer.class.getSimpleName());

RestHighLevelClient openSearchClient = createOpenSearchClient();

KafkaConsumer<String, String> consumer = createKafkaConsumer();

try(openSearchClient; consumer) {

boolean indexExists = openSearchClient.indices().exists(new GetIndexRequest("wikimedia"), RequestOptions.DEFAULT);

if (!indexExists) {

CreateIndexRequest createIndexRequest = new CreateIndexRequest("wikimedia");

openSearchClient.indices().create(createIndexRequest, RequestOptions.DEFAULT);

log.info("The Wikimedia index has been created!");

} else {

log.info("The Wikimedia Index already exists");

}

consumer.subscribe(Collections.singleton("wikimedia.recentchange"));

while (true) {

ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis((3000)));

int recordCount = records.count();

log.info("Received: " + recordCount + " records");

for (ConsumerRecord<String, String> record: records) {

try {

IndexRequest indexRequest = new IndexRequest("wikimedia")

.source(record.value(), XContentType.JSON);

IndexResponse response = openSearchClient.index(indexRequest, RequestOptions.DEFAULT);

log.info(response.getId());

} catch (Exception e) {

}

}

}

}

}

}