Points on Kafka Broker-partitions

## How Kafka is secure

Because Kafka uses SSL based communication where SSL is a standard technology for encrypting the data that is being sent over the network

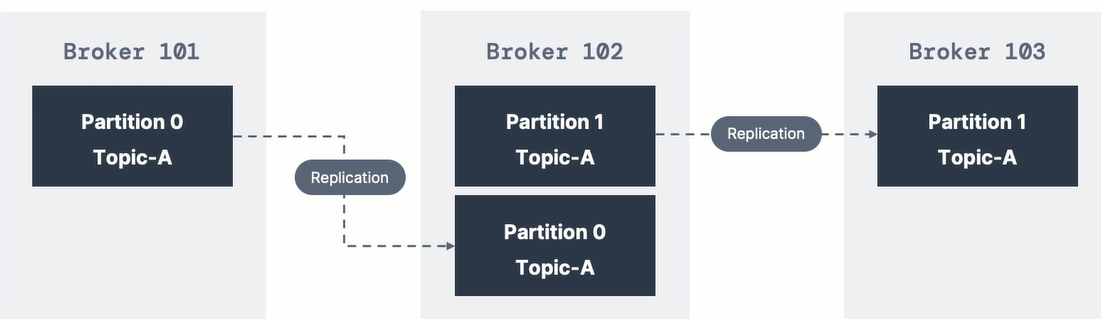
Kafka is always a cluster its set of nodes running on different machines

1. For every message u send u will definitely get acknowledgement
2. And kafkaTemplate.send(msg,callback) and attach a call back u will get a metadata where u can find that message went into which partition number, and what was the offset id for it
3. If u send 1 million messages, u will get 1 million acknowledgements, and attach a callback for each message , for all 1 million messages callback will be fired and u can see partition number and offset id of each stored messages
4. What if leader is down ? always we should maintain some in sync replicas

Means those replicas will continuously ask leader for new data and they will keep themselves in sync with each other. So when leader is down if 1 broker is in In sync replica list then that ISR can become the leader as previous , leader is down , always cfg minimum ISR list =2+

Assignment:- send 1 lakh message and see all the acknowledgements

1. Topics are durable its not like queues once a consumer consumes they are not lost we can retain then, topics stores the data in log files we can configure the retention policy for those messages
2. 1 topic will have many partitions –each partition will be on a separate node to protect from broker failures



If you see the above image if topic-A is having 2 partitions , both partitions are on different node to

1. If all partitions are on same broker, if that broker goes down all partitions will not be available
2. since partitions got splitted and stored even if that broker went down all partitions will not go down

## Kafka references

1. Kafka consumer html java docs

<https://kafka.apache.org/23/javadoc/index.html?org/apache/kafka/clients/consumer/KafkaConsumer.html>

1. Read documentation for each class u use , this is very much recommended

Producer

1. Partition is determined by the key- While sending messages ProducerRecord, send partition num , if u send it will go to that partition only, else it will use the key , Partition number will decide the key same like hash code ,
2. If no key then all messages will be evenly distributed to all partitions in round robin fashion   
   If all messages are having same key they all will be sitting on same partition   
   use case:- lets say if we are sending employee objects send emp objects having the key as employee work location so that all hyd emp data will go to partition -1 , all employee objects having Chennai work location those emp objects will goto partition-2.. so if u want data to flow to certain partitions then prefer sending the key along with the messages
3. If we are sending data we always send to leader broker, consumer should request for the data and he should provide acknowledgement once he properly consumed the message , else producer may think and he may retry and consumer should commit the offset after consuming that message

Questions

1) Can we tune partition size -mostly hyd employee Objects will go to partition-1 , if all data goes to p-1 for hyd then partition should be bigg right

Glimpse of kafka

1. Broker responsibility is to acknowledge for those messages received, if producer didn’t receive ack he will send the same message again
2. Consumer responsibility is to commit the offset frequently all those committed offsets will be stored in the topic of broker

What is an Event in Kafka World?

Event is anything that is happened, manual event or automatic event

When user clicks something it’s also an event or any user interaction is an event

Ex:- transaction is an event, sensor data this is also an event

# Kafka properties

General properties:- key.deserializer", value.deserializer

Deserializers and serializers are common

props.setProperty("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");

props.setProperty("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");

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

Note:- every kafka message will be having a corelation id

1. **Auto timestamping**There are 2 types of timestamps to which a message can be set

At origin we can set the timestamp

Setting producer time

**Message.timestamp.type=0** this is the default and recommended one means producer will set the time to that message when he is sending to the broker

Setting broker time 1 means , timestamp will be set to the TimeStamp field of ProducerRecord not by us ,by broker it will be set when the message received by the broker

1. Buffer.memory=32MB this is producer buffer memory ,

This is a producer config and can be set to properties obj which we give to producer

Producer.send(ProducerRecord); will actually send the record to producer buffer

IO threads are responsible to send the data from buffer to the cluster

1. Min in sync replicas must be configured =2

**Min**.insync.replicas=2 its like in a team ensuring remaining team members are having equal knowledge as like team leader, so that if team leader goes offline remaining 1 of few members can become next team leader

--config min.insync.replicas=2 this is a topic/ broker config

So while creating a topic u have to add to the existing script

It will make sure always 2 replicas are present in the ISR list, if we don’t do this

Then incase if follower is far behind the broker then they will be removed from the ISR list

And eventhough u cfgd acks=null, since no body is present in ISR list, broker will give ack only when it is received by the broker itself

#story already written in another word file named “kafka fresco notes”

1. Acks=all

Means broker will give back acknowledgement back to the producer only when all the followers in the IN sync replicas also received the message

1. Enable.idempotence=true (this is a producer configuration)

This is only for producer retries, in this case producer while sending the messages it will stamp the producer id and message sequence number , so when producer resends ,he will resend with **same producer id and same sequence id** and broker will identify and rejects it

This is for exactly once scenario to avoid duplicates, in atleast scenario there is a chance of duplicate occurances , eventhough message received successfully if broker while sending back acknowledgement if it lost, producer will retry sending the same message which is a duplicate scenario , to avoid that set this flag,

1. props.setProperty("group.id", "test"); every consumer must belong to a consumer group
2. props.setProperty("enable.auto.commit", "false"); this is a consumer property this is by default true which means **AUTO-OFFSET-COMMITTING** means consumer after consuming the messages he will send back the acknowledgement immediately to the broker for each and every message

if u wantedly make it is a false then its programmer responsibility and he should decide when to commit

Read full documentation for entire stuff

<https://kafka.apache.org/23/javadoc/index.html?org/apache/kafka/clients/consumer/KafkaConsumer.html>

1. spring config properties to avoid poison pill scenarios

key-deserializer: org.springframework.kafka.support.serializer.ErrorHandlingDeserializer

value-deserializer: org.springframework.kafka.support.serializer.ErrorHandling Deserializer

spring.deserializer.key.delegate.class: org.apache.kafka.common.serialization. String Deserializer spring.deserializer.value.delegate.class: io.confluent.kafka.serializers. KafkaAvroDeserializer

1. log.segment.bytes

here segment is nothing but a file, once a file/segment is opened we will continuously write to that file, each file maximum size is 1GB, once 1GB is filled that’s it we will create a new segment

1. log.segment.ms =1week (to create a new segment or new file)

its maximum wait time to create a new segment

generally kafka has a habit of creating more segments, here I gave as 1 week, maximum kafka will wait up to 1 week ,after that whether file max size 1GB reached or not it won’t consider it will create a new file or segment

Broker properties

Since these are broker properties try keeping these in C:\kafka\_2.13-3.3.1\config\server.properties

**log.cleanup.policy=delete** //But I don’t know how kafka is deleting the messages based on which time is it the header time it will delete based on the age of data , default is 1 week

* **log.retention.hours** = 168 // Deleting based on hours
* **log.retention.minutes** = 2 //Delete for every minute
* **log.retention.ms** =1 //deleting for every few milliseconds
* **log.cleanup.policy**=compact //Deleting based on duplicate key instead of delting based on time/deleting old messages

log.retention.bytes = -1//infinite

# Practice – assignments

1. Auto timestamping –set the producer time and setting the broker time to the message
2. Practice 03- multi threaded producer which demonstrates 1 kafka producer object is enough , multiple threads can use same kafka producer objects

# Kafka standards

Kafka can process million messages per second with cluster of brokers

1. Producer batching – send a batch of records to broker at once
2. Sending in a Transaction 🡪In an application ,if you are sending 4 messages to kafka make sure you are sending all of them in a transaction
3. Don’t tolerate loss of messages
4. Don’t tolerate any duplicate messaging – implement producer idempotence
5. Consumer Batching + Implement kafka batch listener – instead of reading single message and saving 1 record at once

We should poll only once for every 5 or 10 sec, so that for that 10 sec, if 1,000 messages are accumulated, we can fetch and store all those messages at once in db, so that insertion will be very fast

# Kafka problems and solutions

1. Poison pill scenario (Deserialization exceptions)

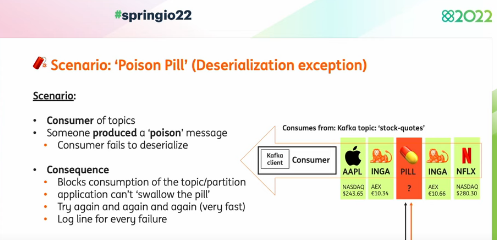
Poison pill is nothing but a message which cannot be deserialized by your current deserializer

1. Lack of proper exception handling

Poison pill scenario

Let’s say , your topic may be consuming only string messages and you are using value deserializer as “stringDeserializer” and suddenly if a json object arrives to the topic and your String Deserializer is not capable of deserializing that json object , so any message which your deserializer is not capable of deserializing it is called a poison pill

The main problem is you would have con



Problem

Since that message can’t be deserialized spring kafka consumer will retry that for 10 times almost

And then it will stop, because for that message it cannot commit the offset/for that message broker will not send any ack and broker will think consumer didn’t received the message and broker also will send the same message again and again to the consumer

Solution

1. Configure the spring related error handling deserializer
2. For later purposes , Better store that message into a dead letter topic or store that bloody message to the mongodb tables

If u use spring kafka when a consumer faces deserialization exception it will not retry because what is the use of retry for 1o times , if u retry for 20 times also same exception will come again and again

1. Configure the Error serializer and deserializer
2. key-deserializer: org.apache.kafka.common.serialization.StringDeserializer
3. value-deserializer: io.confluent.kafka.serializers.KafkaAvroDeserializer

If error comes the above deserializer will handle, if error didn’t came those request will be delegated to

Their own classes who are below

**spring.deserializer.key.delegate.class**: org.apache.kafka.common.serialization.String Deserializer

**spring.deserializer.value.delegate.class**: io.confluent.kafka.serializers. KafkaAvroDeserializer

after configuring those properties, when a deserialization exception hits it will not retry for that many times

Publishing that pill/ message to the dead letter topic

@Bean

public DefaultErrorHandler errorHandler(DeadLetterPublishingRecoverer recoverer) {

return new DefaultErrorHandler(recoverer);

}

Configure this bean and it will put the pills / undeserializable messages into the dead letter topics

### Exception handling retry scenarios

<https://github.com/manideep-vv/forked-spring-io-barcelona-2022-spring-kafka-beyond-the-basics/blob/main/spring-kafka-consumer/src/main/java/nl/jtim/spring/kafka/consumer/config/KafkaExceptionHandlingConfiguration.java>

use proper backoff and then retry ,

|  |
| --- |
|  |
| import org.springframework.util.backoff.ExponentialBackOff; |
|  |

@Bean

public DefaultErrorHandler errorHandlerWithExponentialBackOff(DeadLetterPublishingRecoverer recoverer) {

ExponentialBackOff backOff = new ExponentialBackOff(2\_000, 1.5);

backOff.setMaxElapsedTime(MINUTES.toMillis(2));

return new DefaultErrorHandler(recoverer, backOff);

}

This says every time before u retry we should wait for 2 minutes

@Bean

public DefaultErrorHandler errorHandler(DeadLetterPublishingRecoverer recoverer) {

return new DefaultErrorHandler(recoverer, new FixedBackOff(0, 5));

}

Validate the data

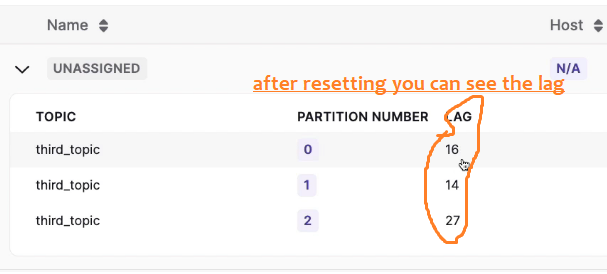
First after consuming validate the data

### Some Question & Answers

1. How to say if that message which we kept in that topic has been consumed or not?

Simple:- check the lag for that consumer group , it shows what was the topics offset , means in that each partition has how many messages and till which offsets they have occupied with messages

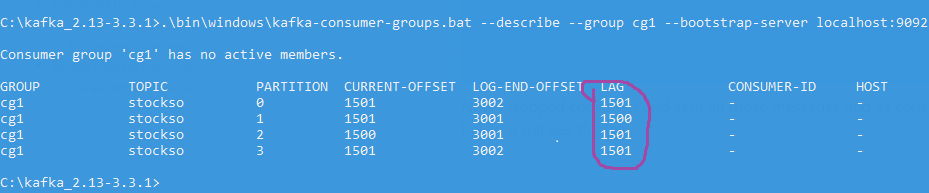
And check the lag by saying till which offset that consumer group have consumed if there is any lag it means messages are not consumed by that consumer group



Here the lag is 16,14,17 for each partition , says that consumer group is yet to consume that many messages from each partition

1. Suppose in a topic if u have consumed all those messages and if u want to consume all those messages again then what u will do
2. Either change the consumer group and start which is telling like I am a new consumer want all messages freshly
3. Or in production or UAT, as u cant change the consumers as those names are fixed u have to reset the offsets and make those pointing to zero
4. If u want to know the project status , u will see the JIRA board , where u can find how many developers are there and how much work did they completed

Similarly, if u want to know the status here of each individual consumer & if u want to know how messages are there in each partition and how many messages each consumer consumed and what is the lag for each partition then u must describe the consumer group



In above see if u describe the consumer group , the partition end offset is 3000 but current offset is 1500, so consumer is either yet to consume 1500 messages or yet to acknowledge for those 1500, what ever the reason but the consumer didn’t commit the offsets for those 1500 messages, means he didn’t consumed or he didn’t gave acknowledgement

# Kafka Best practices

1. Make sure ur code should work for both avro and json formats, it should be easily portable
2. Sending data in transactions

When ur application is producing messages to kafka then use transactions in kafka by setting unique transactional id for each and every producer

1. producer batching

If u are producing large volumes of data, u can use producer batching

Means we will send data one by one, but IO threads will wait for linger.ms (some 400/X ms) or for some size, once that is met then io threads will transfer all data at once to broker

1. Batch listener - instead of listening in serial fashion , we will listen as a batch of records, don’t poll every millisecond to get 1 records, poll only for every 5 seconds and get 100’s of records at once

batch.size

The producer will attempt to batch records together into fewer requests whenever multiple records are being sent to the same partition. This helps performance on both the client and the server. This configuration controls the default batch size in bytes.

A small batch size will make batching less common and may reduce throughput (a batch size of zero will disable batching entirely). A very large batch size may use memory a bit more wastefully as we will always allocate a buffer of the specified batch size in anticipation of additional records

Linger.ms

This setting accomplishes this by adding a small amount of artificial delay—that is, rather than immediately sending out a record, the producer will wait for up to the given delay to allow other records to be sent so that the sends can be batched together.

Lets say u gave linger.ms=5s , means eventhough this line producer.send(new producerRecord()) ran , producer will not send this record immediately to broker

Producer will keep it in buffer for linger.ms and then IO threads will send data to broker as a batch of records, this setting is to send the data in batches

1. If u are producing large volumes of data, then u can use multi-threaded producer
2. Attaching callbacks- while sending data attach a callback, so that u will know message is sent successfully or not

producer.send(record,callback);

1. Handling duplicate data scenario- Assume duplicates will come / consider the case of same message is being retried and then solve the problem
2. Send data in Avro format – avro is very fast as less data will be transferred as schema will not be sent along with data unlike json

When same message will be retried to re consumed again and again?

1. After consuming the message, when consumer is sending back ack to broker that ack is lost due to network fluctuations, when broker didn’t received ack he will resend the same message again to consumer
2. Sometimes when consumer won’t send back ack wantedly when no follower is in sync and when only leader consumer alone consuming those messages, when followers are completely out of sync in this case also consumer wont send back ack to broker

Ex:- in sdp appn we uniquely identified a message using kafka header, we implemented database unique constraint on a column, so if same comes again since constraint is there db will unique constraint exception

1. Deciding how many folders🡪 A developer should decide how many folders/partitions they want, if u give too many ( 2,000 partitions ) either 2,000 consumers/ app instances or there Must be 1 consumer must read from 500 partitions, so be cautious while finalising no of partitions
2. Decide how to process messages in Dead letter queue
3. Be careful while deciding segment size-

Low segment size🡪 A smaller segment size means that files must be closed and allocated more often, which reduces the overall efficiency of disk writes. Having a small segment size (ex:- 1MB/2MB means only few messages will be written into it, if u have huge data kafka should create multiple files) would mean while reading also if consumers are reading huge data then reading messages Kafka has to keep a lot of files open which may lead to **Too many open files**error**.**

* 1. **Time-based Retention:** Set a specific time period (e.g., 7 days).
  2. **Size-based Retention:** Define a maximum log size (e.g., 10GB).
  3. **Combined Approach:** Use both time and size limits for added control.

1. Make sure u have a retry topic, after consuming a message, if it fails because if our target service is down or if db is down, then we will not be able to save that message ,

Then in that case we should send that message to 1st retry topic /2nd retry topic,

1. While designing always consider if order of messages is important – message-1- account creation, m-2 – account details updating , what if the 1st msg went to DLQ and what happens if

We try to process 2nd message

Misc

1. We should frequently commit offsets

Lets say- we have 3 partitions, 1 consumer of a consumer group reading from 1 partition,

That consumer should frequently commit that offsets (a way to tell broker that I have read till this message)

And consumer should send acknowledgement only when that message has been processed successfully

1. Dfg

Kafka features

1. Kafka allows us to **fetch the data from any offset number**

|  |  |
| --- | --- |
| .index file  (something like hashmap , keys as offset number, value as partition+file segment) | .timeindex |
| Used forFinding messages by offset | Finding messages by timestamp |
| Analogy - Book index by chapter number | |  | | --- | | Historical timeline | | .timeindex file contains the offset number and what time that message arrived | |

1. **Time based fetching** – topic might have lot of data, but if u want to fetch message only after certain time onwards then we can use this .timindex file

Efficient Time-based Seeks: When a consumer needs to start consuming messages from a specific point in time, Kafka can leverage the .timeindex file. It allows for a quick lookup of the offset associated with that timestamp. This is significantly faster than linearly searching through the entire .log file.

while (true) {

ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(FETCH\_TIMEOUT\_MS));

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

// Check if message timestamp is after the specified start time

if (**record.timestamp()** >= startTimeInMillis) {

System.out.printf("Key: %s, Value: %s, Timestamp: %d\n", record.key(), record.value(), record.timestamp());

// Process the message here

}

}

consumer.commitSync()

1. **Offset number based fetching** – for this, internally kafka will use .index file

long offset = 42;

// Create Kafka consumer

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

// Create a list containing TopicPartition with desired topic and offset

TopicPartition topicPartition = new TopicPartition(topic, 0); // Assuming partition 0

List<TopicPartition> partitions = Collections.singletonList(topicPartition);

// Assign the consumer to the specific topic partition

consumer.assign(partitions);

// Seek to the desired offset

consumer.seek(topicPartition, offset);

When this will be useful

1. **Inserting the record with timestamp** – producer while inserting the record he can choose some timestamp, so that from consumer

can see at what time producer has sent that message to that topic / at what time broker received that message to that topic

1. Choose the right log clean up policy

|  |  |
| --- | --- |
| Log.cleanup.**policy=compact**  To delete messages based on key | Log.cleanup.**policy=delete**  To delete messages based on time /based on retention period |

Df