Imp points

1) While sending a record give partition number- so that message will directly go to that partition if u don’t give that provide at least key so that partition will be calculated based on that key, if u don’t give key , then in round robin fashion it will send messages to all partitions

2) Concept of producer buffer - if producer is sending 1 million records in 2-3 seconds , it won’t send all 1 m records to broker 1 by 1 by hitting 1 million times– we have concept of producer buffer all the records will be in some buffer if buffer reaches max size then it will send in batch, but in real time if we are sending 1 individual message then how come it was sent so fast?? This I am also not sure.

3) Every message in kafka is automatically timestamped , producer api will set the time to the message which is going to send

4) add logic to reject duplicate messages incase of retries

5) Configure enable.idempotence= true for exactly once scenario to avoid duplicates, producer must enable this to avoid duplicates from producer retries, incase even if message already received by broker and if the acknowledgement was lost and if it wasn’t received by producer he may retry which is a duplicate scenario, so to avoid such scenarios keep the flag =true

6) Mandatorily Implement transactions in kafka-- send the messages using transactions – so that in a transaction either if all messages will be sent or no message will be sent, Caution: **- setting a unique transaction id is mandatory for each producer instance,**

And make sure consumer reads only committed data , set isolation levels to read\_comitted

**7) While consuming (ex:- acc transfer request )- make sure consumer should commit the offset frequently**

If he don’t do that- if that consumer went down and came back as he didn’t commit the offset he will read the same message again and debit the money from one of the account

So its danger consumer should always commit the offset

8**)Commit the offsets after consuming** configure commit.interval.ms=100/200ms so that consumer will commit the offsets what ever he has read for each and every 100-200ms

Always think about this worst cases

1. What if the broker went down without committing the offsets

Doubts

1. If producer send 1 million messages , then will we receive 1 million acknowledgements for all messages, if yes but how do we know which acknowledgement is for which message among those million messages

Producer

Producer always sends the data to the leader partition, similarly consumer always consumes from the leader partition

When you are creating a producer its mandatory to tell the application id, then only the broker will come to know from which ap these messages are coming from, so set this ID while configuring the producer

Properties props = new Properties();

props.put(ProducerConfig.CLIENT\_ID\_CONFIG, "HelloProducer");// this tell which producer is this

props.put(ProducerConfig.BOOTSTRAP\_SERVERS\_CONFIG, localhost:9092,localhost:9093);

props.put(ProducerConfig.KEY\_SERIALIZER\_CLASS\_CONFIG, IntegerSerializer.class.getName());

props.put(ProducerConfig.VALUE\_SERIALIZER\_CLASS\_CONFIG, StringSerializer.class.getName());

props.put(ProducerConfig.TRANSACTIONAL\_ID\_CONFIG,”diff for each producer instance”);

props.put(ProducerConfig.TRANSACTIONAL\_TIMEOUT\_CONFIG,12000);

KafkaProducer<Integer, String> producer = new KafkaProducer<>(props);

logger.info("Start sending messages...");

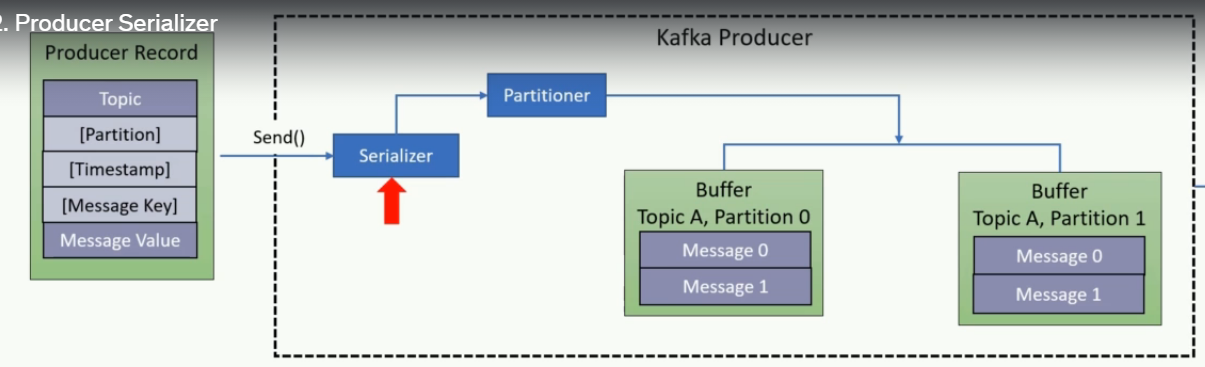
for (int i = 1; i <= AppConfigs.numEvents; i++) {

producer.send(new ProducerRecord<>(AppConfigs.topicName, i, "Simple Message-" + i));

}

|  |
| --- |
| Topic –mandatory field because if u don’t know topic name where u will send the message |
| Message value - Mandatory field |
| Message key –optional |
| Partition - |
|  |

Producer buffer



If producer is sending 1 million records at a time producer wont really send them all at once

It will be stored in a buffer, once buffer max capacity is filled then only it will send

Doubt:- let’s say if buffer max capacity is 1MB if we send 1-2 messages definitely it wont fill , in that case to full the capacity it take hours, but how are we getting instantly and why are we not waiting till buffer is filled

Partitioner

You can also Create a custom partitioner and provide that name to that

Props.put(ProducerConfig.partitioner\_Class\_CONFIG,MyPartitioner.class.getName());

Message timestamp

For every message in Kafka time stamp will be there among below 2

1. **Create Time** TimeStamp set by producer – its create time like when producer is going to send a message then producer will set this time to the message

Message.timestamp.type=0 this is the default and recommended one

1. **Log Append Time** Time stamp set by broker on arrival to the broker –In this case time stamp will be set by the broker when he received it

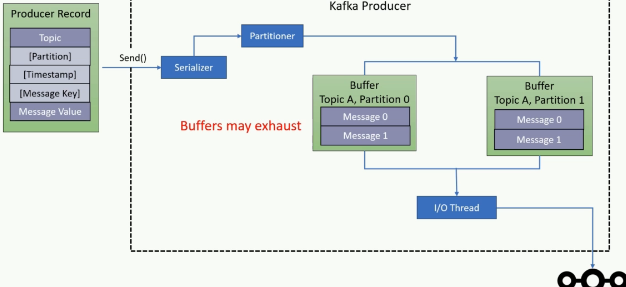
Message.timestamp.type=1

Producer buffer

kafkaTemplate.send(topicName,message) // this will actually send the message to the buffer partition , it will not immediately send the message to the partition, IO threads are real senders when the buffer max capacity is filled then IO threads will send the data from buffer to cluster.

The default producer memory is 32 MB , if u want you can expand the producer buffer memory

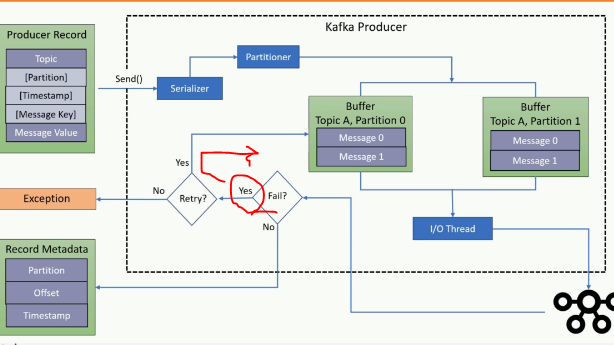
Buffer.memory



Producer retries

Generally if we send 1 million messages we should get acknowledgements for each message we received, lets say if we have configured acks=all means leader will send positive acknowledgements only when all followers / all ISR’s received it, lets say leader received the message but followers didn’t received it incase failed acknowledgement will be sent back, and as usual producer will retry to send the same message again now I don’t know what happens then will broker accepts it?

Ok for now understand that it will retry when it gets a negative acknowledgement



Summary

1. Producer.send(key,value);- Seriailization of key and value , decide the partition number

First the key and value will be serialized by the respective serializer

If partition is there it will put that serialized message into that partition buffer ,else it will decide the partition based on the key , if key is also absent then in round robin fashion message will be sent to one of the partition

1. Serialized message will sit into the producer buffer
2. IO thread will send the message to the cluster
3. If the producer received the acknowledgement then no retry , if the producer didn’t received the acknowledgement then retry
4. It will retry for configured number of times, still not working ha then then throw an exception

**Scaling producer**

A single kafka broker is enough to receive 100-1000 messages per second

If you have a trading application where u should send the data at high speed , like 10,000 messages per second then u may need to use multi-threaded producer

Ex:- if u have 2 files which Is having bulk data in each file

Then create 2 threads ,whereas each thread will read parallelly from each file and each thread after reading one line it will send that line to producer buffer internally once that entire producer buffer is reached then all will be sent at once

Duplicate message –atleast once vs atmost once

Below are the scenarios of coming duplicate messages

1. Generally when we send a message by the producer , broker after receiving the message he should send acknowledge for the received message , incase broker while sending back acknowledgement to the producer if some issue came, then that ack wont reach to the producer in that case producer will retry sending the message which is already received
2. In case of replications – generally replicas are considered as follower brokers , generally these followers will be little behind the leader , like if leader consumed or read 100 messages followers would have read 70-80 messages only , lets say if acks=all means when a message received by the broker 1) leader broker will consume happily 2) when these followers are out of sync and since they didn’t consume properly since ack=all ( means send back ack only when all followers consumed the message) if one of broker has missed to consume it and in this case timeout as we didn’t send the ack back to the producer and producer will think msg not received and he will resend he wont consider that message was already consumed by the leader broker or not , from his side since ack didn’t came he will resend in this case this is duplicate message will be sent to broker even though leader has already consumed the same msg he will get it.

To avoid this w have 1 worst solution that is configure zero retries= atmost Once

So this is bad because even if something went wrong it wont retry leads to data loss

**Best solution is u implement the duplicate record identification** mechanism based on the co relation id

Kafka is by default atleast once- means retry always

Solution for duplicate retry-idempotent producer

This solution won’t work if application itself is wantedly sending the duplicate message

This is only for retry mechanism, if u configure this while retrying this the producer will not send the duplicates, I don’t know how it works internally but configure this

In case when acknowledgment didn’t received then producer will retry right so in those cases if u configure this while retrying it will ensure no duplicates

Enable.idempotence=true

Transactions in Kafka

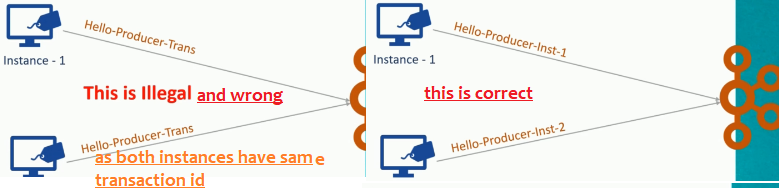
1. Transaction **means – either all or nothing** - It says either all messages within the transaction are committed or nothing is committed
2. Transactions should be committed if all parts are succeeded ,
3. only committed data can be consumed by the consumer, if u have sent to broker and if u didn’t commit the data that data may present in the broker , but the consumer cannot consume it for this u might need to set the isolation level to read\_comitted
4. Transactional id which is set to the producer instance must be unique even in cluster environment- each producer instance must have a different transaction id to uniquely identify that producer

!!!! Whenever u are using the transactions in Kafka make sure u should have different transaction id for each producer, especially if u scale the application in our prod if we

Have 4 instances of our application then we will have 4 producers then each producer should have different transaction number

Internally it is similar to database commit, like first all instructions will be executed in database once u commit changes are saved, here also in a transaction if we are sending

100 of messages first all messages will go to the kafka and after committing it will persist



You cant run 2 instances of producer with same transaction id-try to know what will happen if we keep same

props.put(ProducerConfig.TRANSACTIONAL\_ID\_CONFIG,” Hello-Producer-Trans”);

if u set as above all producer instances when this appn running on a cluster all instances may take same transaction id

Assignment:- what will happen when 2 producer instances have transaction id

On say producer.initTransactions(); , producer.beginTransaction()

Whether u start the transaction or not, by default all the message will be written to broker with abort flag, once u commit the flag will be changed to **“c”** for each message and only messages which are marked as committed can be read by the consumer

When u say kproducer.rollback(); as every producer is identified by its transaction id, as messages are already written to the broker, broker finds all the messages written with that transaction id and all those messages will be rollbackedby the broker ,

Now think if 2 producers have same transaction id which is the uniquely identifiable , when 2 producers wrote messages to the broker, if 1 producer issued rollback , as messages are written with same transaction id which producer messages should be rollbacked?

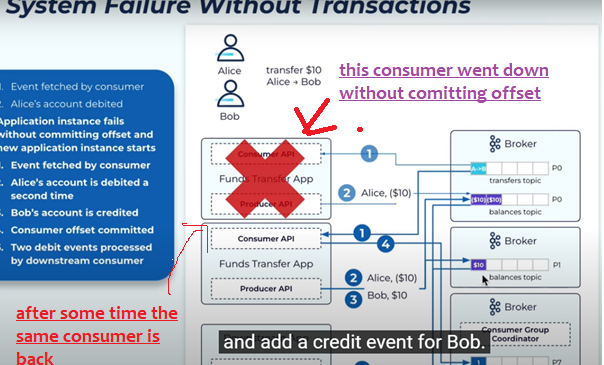
Hence each producer instance should have different transaction id

Ans:- application wont even start if another producer also have same transaction id So in prod also when second instance cant start- because when u say

what is producer.beginTransaction()?

These transactions are initiated by the producers, if the same producer instance has initiated the transaction and if he didn’t close that transaction, this producer.beginTransaction() statement will close or it may abort un finished transaction if exists

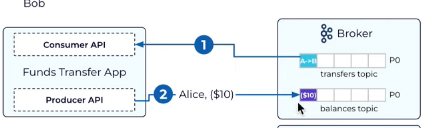
Producer.commitTRansaction(); producer.abortTransaction();

s

In the above image see , as there is no transaction enabled debit request message has been kept 2 times into the queue.

#### Without transaction what is the issue

Let us say a money transfer event, there are no transactions,



1. Step -1 in above message is a message transfer event which the consumer must read
2. consumer after consuming the message it should produce 2 events –debit event, credit event

First consumer will read a message called message transfer event from A --> B and consumer didn’t commit the offset after reading and consumer kept a money debit message from Alice into another topic, and suddenly it went down

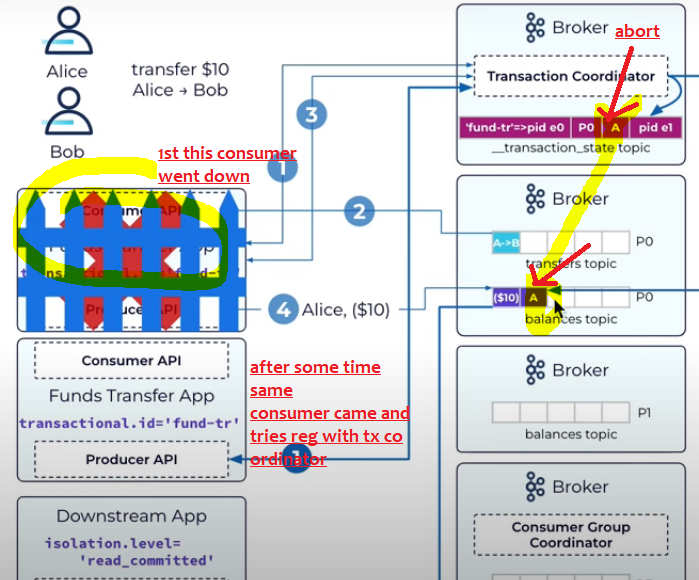
And after sometime it is back and since last time that consumer didn’t commit the offset it will read the same transfer message again **from Alice 🡪 Bob and** puts **Alice money debit request message** into the topic second time which is wrong () and keeps money credit message to Bob in another topic

If first time itself if transaction are there, since credit message to BOB has failed, debit tx wouldn’t have committed issue would have solved, hence transactions are very important, since there is no transaction that’s why this duplicate message came

So when transactions are there even when application fails it will guarantee only single time message will be delivered

#### Exceptions during a transaction

1. first every producer instance should have a transaction id even in a cluster environment also with that this producer it will register to a transaction co-ordinator and get the unique producer instance id



Now see as per above image 1 st consumer read the message and stored in topic and didn’t commit offsets and went offline and after same consumer came and registers with transaction co ordinator and see this transaction id is already registers and *after seeing all that transaction co ordinator tries to abort all pending transactions and those messages which are* ***already written to the broker will be marked as aborted*.**

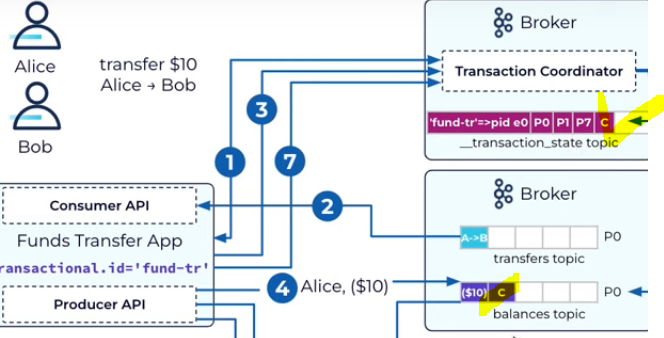
Q) U may get a doubt if these messages are already written what if they are consumed in between ,

A) Simple these messages are not in committed state, consumer reads those messages which are in committed state

#### 2- Successful or committed transaction

Success or failure , during a transaction in both cases messages will be written to kafka broker

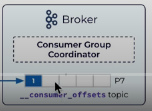
If success and when you **commit the transaction**, then commit marker will be marked on that message see below message, if transaction is aborted then message will be **marked as abort** refer above



Note:- even though both messages are in the broker ,consumer can see only committed messages

Committed offset storage

When the consumer committed the offsets all will be stored in the broker

offsets should be maintained by broker, not consumer, like the team lead should know how far the team member has worked

if consumer stored the offsets , if that consumer went down we don’t know how far the consumer has read, if broker know then he will give the offset number to new consumer to consume from that instance