Kafka

Consumer Offset

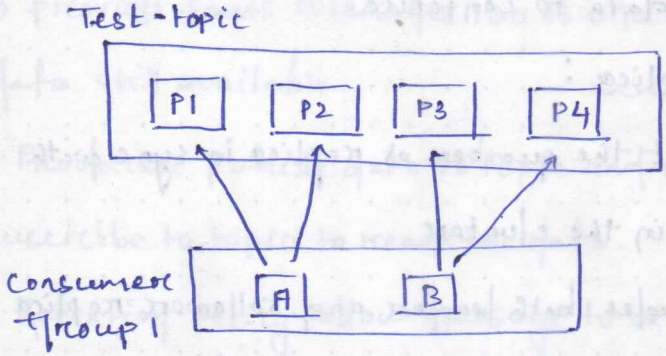
1. From beginning
2. latest
3. specific offset (Programatically possible)

→ Consumer offset is stored in a internal topic — consumer_offset.

Consumer Group

→ group.id is mandatory.
→ group.id plays a major role when it comes to scalable msg consumption.

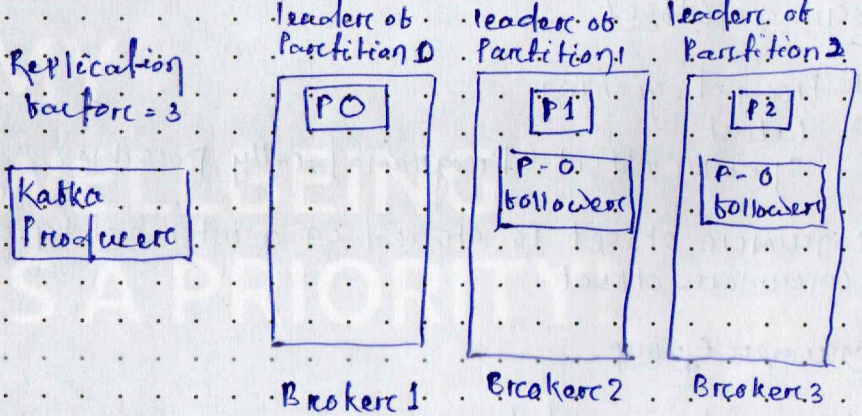
→ Each different app will have unique consumer group.



→ No need to manage explicitely as it will be manage with same group id.

→ Kafka Brokers manage the consumer group.

Replication



- Same happen for other partition, Here clone of partition saved in other broker as followers.

- If broker 0 goes down zookeeper will assign other broker partition as leader and it will use the cloned data to continue.

In-Sync Replica :

→ Represent the number of replica in sync with each other in the cluster.

- Includes both leader and follower replica.

→ Recomended value is always greater than 1.

→ Ideal value is ISR = Replication Factore.

→ This can be controlled by "mini. insync. replicas"

- It can be set at brokers or topic level.

Note:

Kafka topic:

It is a catagory or stream where messages are sent by producer and consumed by consumer.

→ Each topic is split into ~~practioners~~ Partitions, which helps kafka scale by distributing data accross multiple brokers in a cluster. These ~~practioners~~ Partitions also ensure message order within them and assign offset to each which act as unique identifier.

→ Kafka topic can be replicate across brokers to provide fault tolerance, so if one fail other data still available.

→ Producer publish data to topic and consumers suscribe to topic to read the data.

→ Retention policy allow message to be stored in topic for a configurable time. Even consumer read the data will be available after that.

Partitions :

Partitions are a way to split data within a topic, enable kafka to scale horizontaly, ensuring message ordering within partitions, support

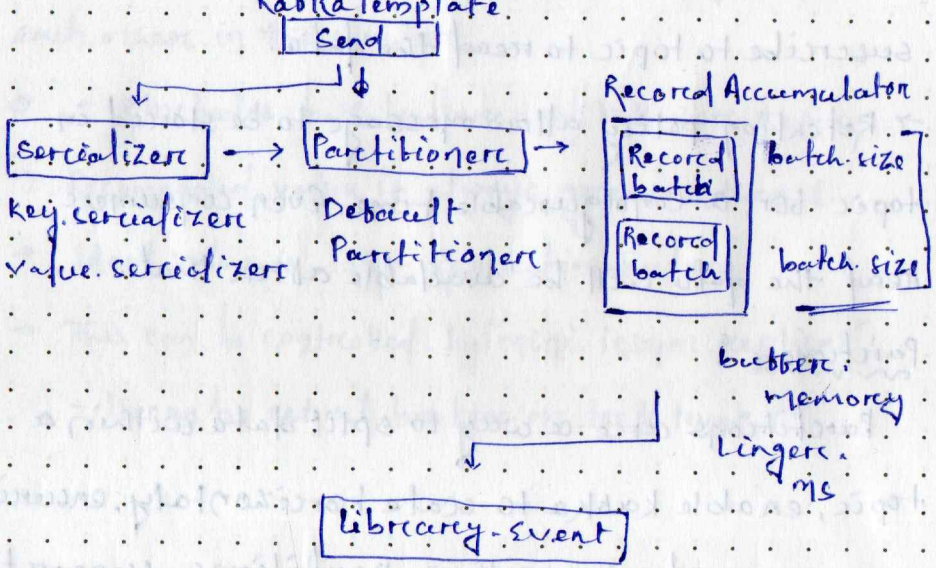
parallel consumption and fault tolerance. Number of topic partitions, configure for a topic has a significant impact on performance and scalability.

Kafka Brokers:

Kafka broker is a server that is part of an apache kafka cluster and is responsible for receiving, storing and serving data. The Brokers act as intermediate between consumers and producers.

→ Topics are being created inside it.

KafkaTemplate.send()



Configure KafkaTemplate:

Mandatory values:

→ bootstrap-servers
→ key-serializers
→ value-serializers

Kafka Admin

- Create Topic programatically.
- Part of SpringKafka.
- How to create topic from code?
  - → Create a bean of type KafkaAdmin in spring Configuration.
  - → Create a bean of type NewTopic in spring Configuration.

what are automated test?

- Run against your code base
- Run as part of build
- Easy capture bug
- Types of Automation
  - → unit test
  - → Integration test
  - → End to End test

Integration Test :

- Test combines the different layers of the code and verify behaviour is working as expected.

Why unit Test ?

- Unit test are handy to mock external dependency.
- Unit test are faster to Integration test
- Cover scenario that not possible in Integration test.

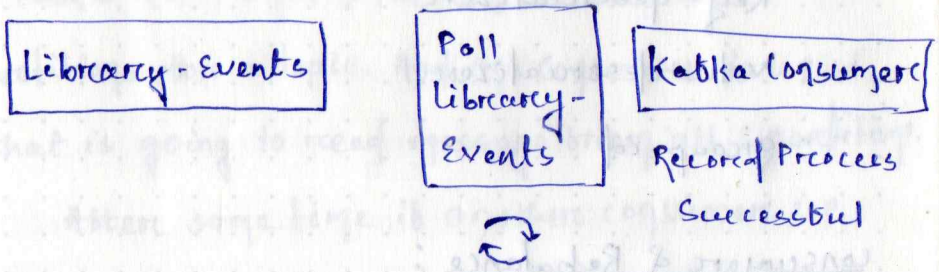
Kafka Producer Configuration :

- acks
  - acks = 0, 1 and -1
  - acks = 1 guarantees message is written to a leader
  - acks = -1 guarantee message is written to leader and all the replica (Default)
  - acks = 0 No guarantee

→ with acks 0, the application put success once it is pushed and don't wait about topic write.

- retries :
  - Integer value = [0 - 2147483647]
  - In spring default value is 2147483647.
- retry.backoff.ms
  - Integer value represented in millis.
  - Default value is 100ms.

Kafka Topic



MessageListenerContainer :

- KafkaMessageListenerContainer
- ConcurrentMessageListenerContainer

@KafkaListener Annotation :

→ Use ConcurrentMessageListnerContainer behind the scenes.

KafkaMessageListenerContainer :

→ Implement of MessageListenerContainer
→ Poll the record
→ Commit the offsets
→ Single threaded

ConcurrentMessageListenerContainer:

→ Represents multiple KafkaMessageListener-Container.

Properties required for kafka consumers:

- bootstrap.servers
- key-deserializer
- value-deserializer
- group-id

Consumers & Rebalance:

Consumer Groups:

Kafka consumer group is a collection of consumers that work together to consume messages from a kafka topic.

→ Each consumer group is assigned one or more partition, ensures that each message is processed by only one consumer.

→ Consumer groups managed their offset, so in case of failure allow them to read from correct offset.

→ Multiple groups can independently consumed from the same topic without interfering to each other.

Rebalance:

→ Kafka consumer concept, that changing the partition ownership from one to another.

Let's the broker is up and running with 3 partitions of a topic. And one consumer is up so that is going to read message from all 3 partitions.

After some time if another consumer up for the same topic with same group id then one request will send to group-co-ordinator and that verify and rebalance the partition by re-distributing between two consumer.

Consumer Offset:

After one record was processed by consumer it commit the offset that so that if consumer goes down it not going to read it from begin.

→ In spring there are many types of acknowledge available. Default one is batch.

Note: For java base configuration go to "KafkaAnnotationDrivenConfiguration.java" and find "KafkaListenerContainerFactory" as this the method work behind bare kafka.

`spring.kafka.listener.ack-mode = manual`

→ Other value available BATCH (default), RECORD, TIME, COUNT, COUNT_TIME, MANUAL, MANUAL_IMMEDIATE.

Concurrent Consumers:

→ Create multiple consumer in different thread which help better process.

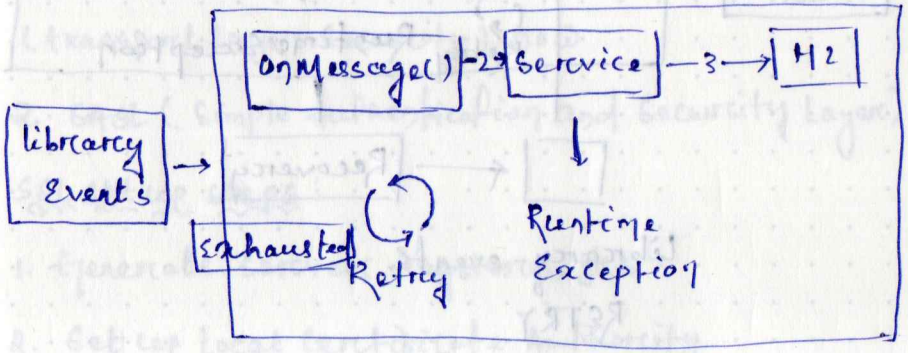
`spring.kafka.listener.concurrency = 3`

```
@KafkaListener(topics = "topic", groupId = "group",
    ackMode = "MANUAL", concurrency = "3")
```

Some other property:

- max-poll-records: Control how many records can be polled per request.
- heartbeat-interval-ms: The interval (in millisecond) at which the consumer sends a heart beat to broker to let it know that it's alive and still processing.
- Session-timeout-ms: Maximum time a consumer can go without sending a heart beat to broker before it concider dead.
- Fetch-min-bytes: The consumer will wait the fetch the batch from broker until atleast this amount of data available.
- batch-max-wait-ms: If broker don't have min bytes mentioned and then max wait limit over then consumer stop waiting and batch data.

Recovery in kafka Consumer:



Recovery

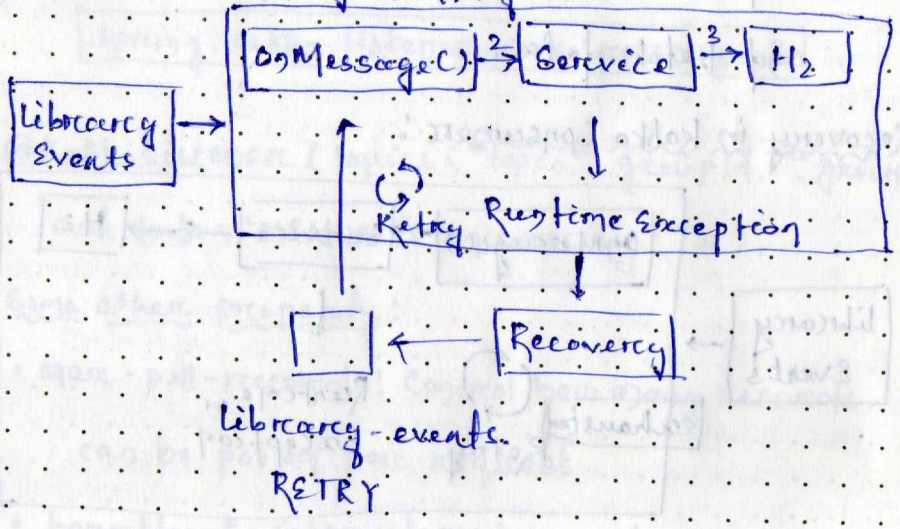
Recovery type:

1. Process the failed records again
   eg: Service the consumer interacts with temperarily down.
2. Discard the message and move on.
   eg: Parsing errors, Invalid events.

Process the message again:

1. Publish the fail message to a retry topic.
2. Saved the failed message in DB and retry with scheduler.

Publish the message to Retry Topic:



Discard the message:

1. Publish the failed record in to Dead Letters topic for tracking.
2. Saved to DB for tracking.

Note: Create some project to understand the code properly.

Kafka Producers Errors

1. Kafka cluster not available
2. If acks = all, and some brokers not available
3. mini.insync.replicas config
   - eg: mini.insync.replicas = 2 but only one broker available.

Kafka Security

Kafka support two popular protocols:

1. SSL (Secured Socket Layer), which is TLS (transport Layer Security) now
2. SASL (Simple Authentication and Security Layer)

SSL set up steps:

1. Generate server.keystore.jks
2. Set up Local Certificate Authority
3. Create CSR (Certificate Signing Request)
4. sign the SSL certificate ~~to server.keystore bits~~

5. Add the signed SSL certificate to server.
keystore file
6. Configure the SSL cert. in our kafka broker.
7. Create client.truststore.jks the client.