Kafka Python

Created by Mahidhar

Reference : Kafka Python Docs

Producer - Using Kafka Python

- A Kafka client publishes records to the Kafka cluster.
- The producer consists of a pool of buffer space that holds records that haven't yet been transmitted to the server as well as a background I/O thread that is responsible for turning these records into requests and transmitting them to the cluster.
- Kafka Producer has ['DEFAULT_CONFIG', '__class__', '__del__', '__delattr__', '__dict__', '__doc__', '__format__', '__getattribute__', '__hash__', '__init__', '__module__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', '_cleanup_factory', '_ensure_valid_record_size', '_partition', '_serialize', '_unregister_cleanup', '_wait_on_metadata',
 'close', 'flush', 'metrics', 'partitions_for', 'send']

Contd ...

- send() -Method is asynchronous.
- When called it adds the record to a buffer of pending record sends and immediately returns. This allows the producer to batch together individual records for efficiency.
- The 'acks' config controls the criteria under which requests are considered complete. The "all" setting will result in blocking on the full commit of the record, the slowest but most durable setting.
- If the request fails, the producer can automatically retry, unless 'retries' is configured to 0.
- The producer maintains buffers of unsent records for each partition. These buffers are of a size specified by the 'batch_size' config. Making this larger can result in more batching, but requires more memory

Contd..

- If you want to reduce the number of requests you can set, 'linger_ms' to something greater than 0.This will instruct the producer to wait up to that number of milliseconds before sending a request in hope that more records will arrive to fill up the same batch.
- Note that records that arrive close together in time will generally batch together even with linger_ms=0 so under heavy load batching will occur regardless of the linger configuration; however setting this to something larger than 0 can lead to fewer, more efficient requests when not under maximal load at the cost of a small amount of latency

Contd ...

close(timeout=None) : Close this producer. Parameters timeout (float, optional) – timeout in seconds to wait for completion.

flush(timeout=None) : Invoking this method makes all buffered records immediately available to send (even if linger_ms is greater than 0) and blocks on the completion of the requests associated with these records. The post-condition of flush() is that any previously sent record will have completed (e.g. Future.is_done() == True). A request is considered completed when either it is successfully acknowledged according to the 'acks' configuration for the producer, or it results in an error. Other threads can continue sending messages while one thread is blocked waiting for a flush call to complete; however, no guarantee is made about the completion of messages sent after the flush call begins.

Contd ...

metrics(raw=False): Get metrics on producer performance.

partitions_for(topic): Returns set of all known partitions for the topic.

send(topic, value=None, key=None, partition=None, timestamp_ms=None) :Publish a message to a topic.

Parameters

- *topic (str) – topic where the message will be published*
- *value (optional) – message value. Must be type bytes, or be serializable to bytes via configured value_serializer. If value is None, key is required and message acts as a 'delete'.*

Contd ..

- *partition (int, optional) – optionally specify a partition. If not set, the partition will be selected using the configured 'partitioner'.*
- *key (optional) – a key to associate with the message. Can be used to determine which partition to send the message to. If partition is None (and producer's partitioner config is left as default), then messages with the same key will be delivered to the same partition (but if key is None, partition is chosen randomly). Must be type bytes, or be serializable to bytes via configured key_serializer.*
- *timestamp_ms (int, optional) – epoch milliseconds (from Jan 1 1970 UTC) to use as the message timestamp. Defaults to current time.*

Consumer

- The consumer will transparently handle the failure of servers in the Kafka cluster, and adapt as topic-partitions are created or migrate between brokers.
- It also interacts with the assigned kafka Group Coordinator node to allow multiple consumers to load balance consumption of topics.
- Kafka Consumer class has multiple methods - 'assign', 'assignment', 'beginning_offsets', 'close', 'commit', 'commit_async', 'committed', 'configure', 'end_offsets', 'fetch_messages', 'get_partition_offsets', 'highwater', 'metrics', 'next', 'offsets', 'offsets_for_times', 'partitions_for_topic', 'pause', 'paused','poll', 'position', 'resume','seek', 'seek_to_beginning', 'seek_to_end', 'set_topic_partitions', 'subscribe', 'subscription', 'task_done', 'topics', 'unsubscribe'