Chapter 1 - Kafka Client APIs

Objectives

In this chapter, participants will learn about:

- Apache Kafka's wire protocol
- Supported clients

1.1 Protocol Overview

- Kafka uses a binary protocol over TCP
- The client initiates a socket connection with the broker and keeps it open for subsequent interactions
  - No handshake is required on connection or disconnection
- The client may maintain connections to multiple brokers
- The broker guarantees that on a single TCP connection, requests will be processed in the order they are sent and responses will return in that order as well
  - This ordering is ensured by way of allowing only a single in-flight request per connection
- As a safety mechanism, the broker will reset the socket for requests that exceed a configurable maximum size

1.2 Client Request APIs

- The Kafka client protocol consists of six core client requests APIs:
  - Metadata - describe the currently available brokers (their hostnames/IP and ports), lists leader brokers for partitions
  - Send – send messages to a broker
  - Fetch - read messages from a broker
  - Offsets - get information about the available offsets for a partition
  - Offset Commit - commit a set of offsets for a consumer group

  - Offset Fetch - read a set of offsets for a consumer group
- In essence, the protocol supports Metadata, Produce, Fetch, and Offset functionality

1.3 Kafka 0.9 Client API Extensions

- GroupCoordinator - locate the current coordinator of a consumer group
- JoinGroup - join a group; creating one if there are no active members
- SyncGroup - synchronize state for all members of a group (e.g. distribute partition assignments to consumers)
- Heartbeat - consumer group member health check
- LeaveGroup - leave the group

1.4 Who is Control?

- Kafka message publishing clients decide what message should go on which partition
- Consumers also need to know about the topic and partition they want to fetch a message from
- The requests to publish or fetch data (write/read operations) are handled by the broker that is currently acting as the leader for a given partition
- Kafka supports automatic discovery of the following information about the cluster: the list of topics and their partitions, which broker is the leader for those partitions, and the host and port information for these brokers
  - Automatic discovery is realized through metadata requests that brokers can respond to

1.5 Which Broker Should I Talk to?

- This is the question of proper client connection bootstrapping
- The client simply needs to find just one broker as any brokers will report back to the client the list of all other brokers that exist in the cluster and

the partitions they host

- The common practice is to let the client know about two or more broker URLs (their hostname/IP : port) to bootstrap from
- The clients are automatically notified on any changes in the cluster (a broker went down, etc.) forcing the clients to update their cached metadata about the cluster
  - So the clients do not need to keep polling to see if the cluster has changed
- This notification comes as 2 types of errors:
  - (1) a socket error indicating the client cannot communicate with a particular broker
  - (2) an error code in the response to a request indicating that this broker no longer hosts the partition for which data was requested

1.6 The Bootstrap Cycle of a Client

1. Cycle through the list of "bootstrap" Kafka broker URLs until one the client can connect to is found
2. Fetch cluster metadata
3. Process fetch or produce requests, directing requests to the appropriate broker based on the topic/partitions they send to or fetch from
4. If an error is received, refresh the metadata and try again

1.7 Partition Assignment Strategies

- As we know, the producer assigns a message to a partition (the case of semantic partitioning)
- For that, the producer needs to use some sort of key which makes sense in your problem domain as a mapping between the message and the partition
- One common technique (which also helps with load-balancing) is to use the hash code of your key (e.g. a user id), or the message contents, and

apply a modulo operation by the number of partitions in a topic:

```
partition_id_to_assign = hashCodeOfTheMessage MOD number_of_partitions
```

1.8 Batching

- The common practice is to send one message at a time
- For efficiency, the new Kafka APIs allow to send and fetch messages in batches
- Batching across multiple topics and partitions are also supported
  - A produce request may contain data to append to many partitions
  - A fetch request may pull data from many partitions all at once

1.9 The Common Error Codes

Error	Description	Retriable
OFFSET_OUT_OF_RANGE	The requested offset is not within the range of offsets maintained by the server.	FALSE
CORRUPT_MESSAGE	This message has failed its CRC checksum, exceeds the valid size, or is otherwise corrupt.	TRUE
UNKNOWN_TOPIC_OR_PARTITION	This server does not host this topic-partition.	TRUE
LEADER_NOT_AVAILABLE	There is no leader for this topic-partition as we are in the middle of a leadership election.	TRUE
NOT_ENOUGH_REPLICAS	Messages are rejected since there are fewer in-sync replicas than required.	TRUE
COORDINATOR_NOT_AVAILABLE	The coordinator is not available.	TRUE
REASSIGNMENT_IN_PROGRESS	A partition reassignment is in progress	FALSE

1.10 Request Header

- The request header contains the following fields:

```
api_key        - The id of the request type
api_version    - The version of the API
correlation_id - A user-supplied integer value that will be passed back with the response
client_id -   A user specified identifier for the client making the request
```

1.11 Response Header

- The response header contains just one field:

```
correlation_id - The user-supplied value passed in with the request
```

1.12 Supported Clients (Language Bindings)

C/C++	Perl
Python	stdin/stdout
Go (AKA golang)	PHP
Erlang	Rust
.NET	Alternative Java
Clojure	Storm
Ruby	Scala DSL
Node.js	Clojure
Proxy (HTTP REST, etc)	Swift

- A number of other language bindings has been built on top of the robust high performance C/C++ library with full Kafka protocol support, including Haskell, Node.js, OCaml, PHP, Python, Ruby, and C# / .NET

1.13 Kafka Client Examples

Apache Kafka Java Client

https://docs.confluent.io/kafka-clients/java/current/overview.html

Apache Kafka Python Client

https://docs.confluent.io/kafka-clients/python/current/overview.html

Apache Kafka and Node.js

https://developer.confluent.io/get-started/nodejs/

Other Apache Client Examples

https://docs.confluent.io/platform/current/clients/examples.html

1.14 Summary

- In this chapter we reviewed Kafka client APIs used by both consumers and producers
- We discussed the bootstrapping process for the client to discover information about the cluster
- We reviewed the message-to-partition assignment considerations