KIP-69 - Kafka Schema Registry

Status

Current state: *"Draft"*

Discussion thread: *pending*

JIRA: *here*

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

Motivation

Schema registry allows producers to define a schema for the messages that are being sent to kafka and consumers to apply the schema to read from.This will greatly benefit all the clients from streaming, batch usecases to understand the schema of a message and perform actions accordingly.

Design - Approach - 1

Adding a Schema for a topic

Users can use kafka-topics.sh tool to add a schema for a topic. This schema will be part of topic config.

Schema will contain version and as the schema evolves Kafka will update the version to a higher version.

Querying for a Schema

Schema will be part of TopicMetadataRequest and Java producer and consumer clients will have that in their local cache after a succesful TopicMetadataRequest.

Other clients can rely on sending a SchemaRequest . This SchemaRequest will contain one or more topics along with version of schema they would like to access.

Serializers, Deserializers

Once the Producer have the serializer plugged in, it will use this schema to validate and serialize the bytes and producer will add schema version to the Message.

Similarly on the Consumer side depending on which schema.version does the message have , it applies that particular schema to deserialize.

If the message doesn't adhere to a schema than the serializer / deserializer throw an exception.

class InvalidMessageException {}

Configuration

Users needs to add schema.version for each topic and clients will apply the schema with that version.

If there is no schema available than we will throw following exception

class SchemaNotAvailableException {}

Design - Approach - 2

The disadvantage of having a schema validation being done on client side is every client has to write a serializer. To overcome this issue we could move this responsibility to broker side. In this design option adding and querying a schema will not change from above Approach - 1. The broker will instantiate a single instance of Serializer on topic creation and initialize it with the Schema specification. When a produce request is sent to the broker , it will invoke the schema validator's validate method with message pay load. The schema validator will validate the payload against the current version of schema for the topic. If the validation is successful message will be stored with one additional message metadata schemaVersion = current version that it was validated against.

Schema evolution

As the schema is versioned it will be updated with a new version. The brokers will use zookeeper watcher and update the latest schema version. Broker will have to keep 2 schemas and serializers current for a configurable amount of time to ensure smooth migration from one version to another but after the time out the new version will become current and all incoming messages will have to adhere to new schema. Given broker always stores the schema version as part of the metadata the consumer side can leverage version to figure out which version of schema a message is part of.

Advantages

* As this validation is server side, we only have to write a serializer for a format once.
* Topic config leveraged as schema registry so no extra dependency.
* Pluggable model so we can support a small number of formats and users can extend it to any format they like.

Disadvantages:

* The broker will have perf degradation depending on how much overhead the serializer introduces for schema validation. For some formats it will be pretty small and for some format it will be pretty heavy. This is a conscious choice that user will have to make when selecting the formats.
* At any given point there will one live version of schema and users won't be able to select which version they would like to apply.