The Ins and Outs of Writing Kafka KIPs

Lucia Cerchie
Developer Advocate
LinkedIn https://www.linkedin.com/in/luciacerchie/

Matthias J. Sax
Software Engineer | Apache Kafka Committer and PMC member
Twitter/𝕏 @MatthiasJSax

What is a KIP?

- KIP: Kafka Improvement Proposal
  https://cwiki.apache.org/confluence/display/KAFKA/Kafka+Improvement+Proposals

- Design document (or one-pager) about any major change in Kafka

What is considered a "major change" that needs a KIP?

Any of the following should be considered a major change:

- Any major new feature, subsystem, or piece of functionality
- Any change that impacts the public interfaces of the project

What are the "public interfaces" of the project?

All of the following are public interfaces that people build around:

- Binary log format
- The network protocol and api behavior
- Any class for which the build generates Javadoc. This includes (but is not limited to; look for `javadoc {}` sections in `build.gradle` for the complete list):
  - org/apache/kafka/common/serialization
  - org/apache/kafka/common
  - org/apache/kafka/common/errors
  - org/apache/kafka/clients/producer
  - org/apache/kafka/clients/consumer (eventually, once stable)
- Configuration, especially client configuration
- Monitoring
- Command line tools and arguments

What is the goal of a KIP?

- Standard software engineering practices to
  - Ensure good design
  - Design before you implement
- Foster collaboration within the community
- Technical documentation!
- Project governance

The KIP Process

- Design a solution and write the KIP itself
- Start a discussion thread on the dev mailing list
- Start a vote thread on the dev mailing list
- Implement the KIP
- Celebrate your contribution!

The Structure of a KIP

https://cwiki.apache.org/confluence/display/KAFKA/KIP-Template

- Meta information
- Motivation
- Public Interfaces
- Proposed Changes
- Compatibility, Deprecation, and Migration Plan
- Test Plan
- Documentation
- Rejected Alternatives

Hands on Experience

KIP-941: Allow range queries to accept null bounds

https://cwiki.apache.org/confluence/display/KAFKA/KIP-941%3A+Range+queries+to+accept+null+lower+and+upper+bounds

KIP-941: Range queries to accept null lower and upper bounds

Created by Lucia Cerchie, last modified on Aug 08, 2023

- Status
- Motivation
- Public Interfaces
- Proposed Changes
- Compatibility, Deprecation, and Migration Plan
- Test Plan
- Rejected Alternatives

Status

Current state: *["Adopted"]*

Discussion thread: https://lists.apache.org/thread/wnn2qrb7vglw11bdm2cdlkm1430b75zc

JIRA: ~~KAFKA-15126~~ - Change range queries to accept null lower and upper bounds RESOLVED

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

Motivation

Within the RangeQuery class in the Interactive Query API (which allows you to leverage the state of your application from outside your application), there are methods for getting the upper and lower bounds. When web client requests come in with query params (which become those bounds), it's common for those params to be null. We want developers to just be able to pass in the upper/lower bounds if they want instead of implementing their own logic to avoid getting the whole range.

An example of the logic they can avoid after this KIP is implemented is below:

```
private RangeQuery<String, ValueAndTimestamp<StockTransactionAggregation>> createRangeQ
        if (isBlank(lower) && isBlank(upper)) {
            return RangeQuery.withNoBounds();
        } else if (!isBlank(lower) && isBlank(upper)) {
            return RangeQuery.withLowerBound(lower);
        } else if (isBlank(lower) && !isBlank(upper)) {
```

KIP-714: Client metrics observability

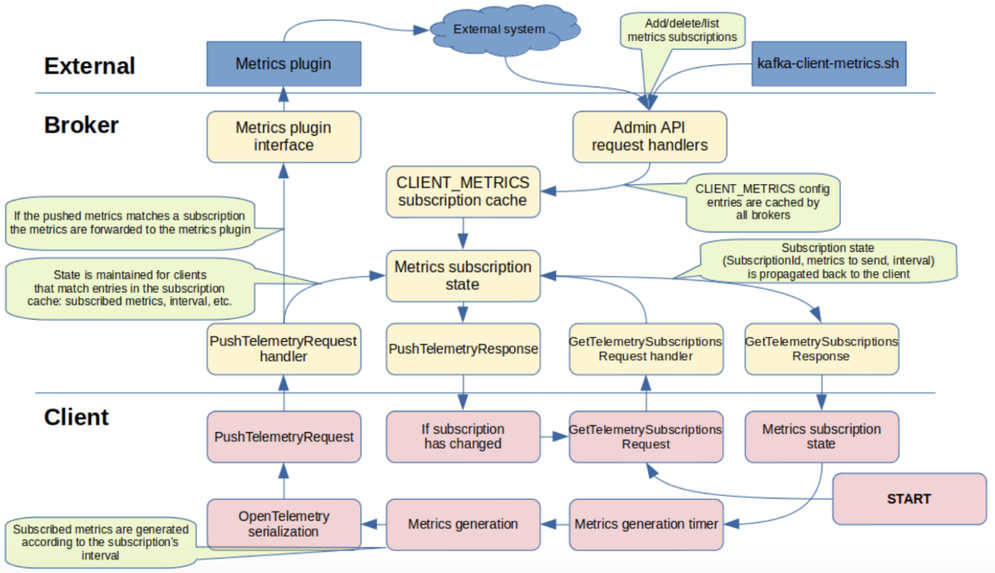
https://cwiki.apache.org/confluence/display/KAFKA/KIP-714%3A+Client+metrics+and+observability

Proposed Changes

Overview

This feature is made up of the following components:

- **GetTelemetrySubscriptions RPC** - protocol request used by the client to acquire its initial Client instance ID and to continually retrieve updated metrics subscriptions.
- **PushTelemetry RPC** - protocol request used by the client to push metrics to any broker it is connected to.
- **Standard and required metrics** - a set of standardized metrics that all supporting clients should provide.
- **AdminAPI config** - the AdminAPI configuration interface with a new `CLIENT_METRICS` resource type is used to manage metrics subscriptions.
- **Client metrics plugin / extending the MetricsReporter interface** - a broker plugin interface that performs something meaningful with the metrics. This plugin will typically forward the metrics to a time-series database. It is recommended that any broker-side processing is kept to a minimum for scalability and performance reasons.



KIP-618: Exactly-Once Support for Source Connectors

https://cwiki.apache.org/confluence/display/KAFKA/KIP-618%3A+Exactly-Once+Support+for+Source+Connectors

Rolling upgrade(s)

At most two rolling upgrades will be required to enable exactly-once source support on a Connect cluster.

The first rolling upgrade will be to upgrade every worker to a version of Connect that can provide exactly-once source support, and to set `exactly.once.source.support` to `preparing`.

The second rolling upgrade will be to actually enable exactly-once source support on each worker (by setting `exactly.once.source.support` to `enabled`).

Two upgrades will be required in order to ensure that the internal fencing endpoint is available on every worker before it is required by any of them, and that no non-leader workers are able to write to the config topic during the upgrade.

Downgrade

Two kinds of downgrade are possible. Exactly-once support for a cluster can be disabled by setting `exactly.once.source.support` to either `disabled` or `preparing` for workers in the cluster (a “soft” downgrade), or workers can be reverted to use older versions of the Connect framework that does not support exactly-once sources at all (a “hard” downgrade).

Soft downgrade

Soft downgrades should be possible via a cluster roll, where each worker is shut down, its configuration is altered to disable exactly-once source support, and then it is restarted. During the process, all connectors and their tasks should continue to run and process data, thanks to the same logic that makes a rolling upgrade possible.

Hard downgrade

Offsets accuracy

Because source task offsets on upgraded workers are still written to the worker’s global offsets topic, even if a downgraded worker does not support per-connector offsets topics, it can still pick up on relatively-recent source offsets for its connectors. Some of these offsets may be out-of-date or older than the ones in the connector’s separate offsets topic, but the only consequence of this would be duplicate writes by the connector, which will be possible on any cluster without exactly-once support enabled. Without any writes to the global offsets topic, all records processed by a connector since a switch to a dedicated offsets topic would be re-processed after the downgrade and would likely result in a flood of duplicates. While technically permissible given that the user in this case will have knowingly switched to a version of the Connect framework that doesn't support exactly-once source connectors (and is therefore susceptible to duplicate delivery of records), the user experience in this case could be quite bad, so a little extra effort on the part of Connect to significantly reduce the fallout of downgrades in this case is warranted.

KIP-962: Relax null-key requirement in Kafka Streams

https://cwiki.apache.org/confluence/display/KAFKA/KIP-962%3A+Relax+non-null+key+requirement+in+Kafka+Streams

Compatibility, Deprecation, and Migration Plan

Keeping the old behavior

Users who want to keep the current behavior can prepend a .filter() operator to the aforementioned operators and filter accordingly.

Examples

```
//left join Kstream-Kstream
leftStream
    .filter((key, value) -> key != null)
    .leftJoin(rightStream, (lv, rv) -> join(lv, rv), windows);

//outer join Kstream-Kstream
rightStream
    .filter((key, value) -> key != null);
leftStream
    .filter((key, value) -> key != null)
    .outerJoin(rightStream, (lv, rv) -> join(lv, rv), windows);

//left-foreign-key join Ktable-Ktable
Function<String, String> foreignKeyExtractor = leftValue -> ...
leftTable
    .filter((key, value) -> foreignKeyExtractor.apply(value) != null)
    .leftJoin(rightTable, foreignKeyExtractor, (lv, rv) -> join(lv, rv), Named.as("left-fore

//left join Kstream-Ktable
leftStream
    .filter((key, value) -> key != null)
    .leftJoin(ktable, (k, lv, rv) -> join(lv, rv));

//left join KStream-GlobalTable
KeyValueMapper<String, String, String> keyValueMapper = (k, v) -> ...;
leftStream
    .filter((key, value) -> keyValueMapper.apply(key,value) != null)
    .leftJoin(globalTable, keyValueMapper, (lv, rv) -> join(lv, rv));
```

Formalities

The KIP Process

- Embraces the ASF motto “if it did not happen on the mailing list, it did not happen”
- ”DISCUSS” thread for a KIP to collect feedback, answer question, discuss tradeoffs, and find consensus
  - Anything from ”this is so trivial we skip it” to multiple months
  - Most KIPs take a few weeks
- “VOTE” thread to formally approve a KIP
  - Vote must be open at least 72h
  - Lazy majority vote
  - Need at least three (binding) +1, ie, three committers need to approve
  - ***Non-binding community votes are still very important!***

Lessons Learned

- Overall very well received process
  - Other ASF projects adopted it, too
  - For smaller changes, sometimes a little bit “annoying”
- Great way to teach new contributors
- A lot of people read(!) KIPs
- KIP go into the release notes: great way to highlight important changes
- Challenges:
  - Make sure all KIPs get attention and guidance by committers
  - No clean “reject” process (lots of zombie KIPs)

The Ins and Outs of Writing Kafka KIPs

Lucia Cerchie
Developer Advocate
LinkedIn https://www.linkedin.com/in/luciacerchie/

Matthias J. Sax
Software Engineer | Apache Kafka Committer and PMC member
Twitter/𝕏 @MatthiasJSax