KIP-896: Remove old client protocol API versions in Kafka 4.0

Status

Current state: Adopted

Discussion thread: here

JIRAs:

- Apache Kafka 3.7: Unable to render Jira issues macro, execution error.
- Apache Kafka 4.0: Unable to render Jira issues macro, execution error.

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

Motivation

The Apache Kafka project has retained support for all protocol API versions since Apache Kafka 0.8.0 (released 9+ years ago on December 2013). As time goes on, the cost of maintaining support for all these versions goes up (both in code complexity and the testing matrix) and the value goes down (as users switch to clients that rely on newer protocol API versions). Apache Kafka 4.0 presents a good opportunity to re-evaluate this policy since it includes other modernization initiatives like dropping support for zk mode and dropping support for message formats v0 and v1 (KIP-724). No date has been set for Apache Kafka 4.0, but it's safe to say it won't happen before the end of 2023.

We propose setting the new baseline for protocol API versions to Apache Kafka 2.1 (released 4+ years ago on Nov 2018). More concretely, for each protocol API, we will only keep the most recent version supported by Apache Kafka 2.1 as well as any version introduced after Apache Kafka 2.1. We deviate from the proposed baseline in a number of cases based on existing clients support. In order to determine client support for the various protocol API versions, we take the latest client release at the end of 2021 - this gives it a better chance of being widely used.

Apache Kafka 2.1 has been chosen as the baseline because it was released many years ago *and* it includes support for KIP-320 (required for proper fencing support when consuming) and KIP-110 (zstd is a popular compression mechanism).

Since some protocol APIs are used both for clients and inter-broker replication (e.g. Fetch, ListOffsets, etc.), this proposal also implies a minimum `inter.broker.protocol.version` of 2.1. That said, the removal of zk mode (also planned for Apache Kafka 4.0) will have even stricter requirements when it comes to this.

We believe there is merit in having a general protocol API versioning policy that can be applied mechanically on each new major version - we intend to submit a separate KIP for that. It would only apply from Apache Kafka 5.0 to give the ecosystem plenty of time to adapt.

Public Interfaces

Remove support for the following protocol API versions (the range is inclusive) in **Apache Kafka 4.0**:

- Produce: V0-V6
  - librdkafka: V7
  - KafkaJS: V7
  - Sarama: V7
  - kafka-python: V7
- Fetch: V0-V3 due to clients (kafka-python)

    - librdkafka: V11
    - KafkaJS: V11
    - Sarama: V11
    - kafka-python: V4
- ListOffset: V0 due to clients (librdkafka, KafkaJS, Sarama, kafka-python) support – V0-V3 per the baseline
    - librdkafka: V2
    - KafkaJS: V3
    - Sarama: V2
    - kafka-python: V1
- Metadata: V0-V3 due to clients (librdkafka, KafkaJS, Sarama, kafka-python) support – V0-V6 per the baseline
    - librdkafka: V4
    - KafkaJS: V6
    - Sarama: V5
    - kafka-python: V5
- OffsetCommit: V0-V1 due to clients (KafkaJS, Sarama, kafka-python) support – V0-V5 per the baseline
    - librdkafka: V7
    - KafkaJS: V5
    - Sarama: V4
    - kafka-python: V2
- OffsetFetch: V0 due to clients (KafkaJS, kafka-python) support – V0-V4 per the baseline
    - librdkafka: V7
    - KafkaJS: V4
    - Sarama: V7
    - kafka-python: V1
- FindCoordinator: none due to clients (Sarama, kafka-python) support – V0-V1 per the baseline
    - librdkafka: V2
    - KafkaJS: V2
    - Sarama: V1
    - kafka-python: V0
- JoinGroup: V0-V1 due to clients (Sarama, kafka-python) support – V0-V2 per the baseline
    - librdkafka: V5
    - KafkaJS: V5
    - Sarama: V2
    - kafka-python: V2
- Heartbeat: none due to clients (Sarama, kafka-python) support – V0-V1 per the baseline
    - librdkafka: V3
    - KafkaJS: V3
    - Sarama: V0
    - kafka-python: V1
- LeaveGroup: none due to clients (librdkafka, Sarama, kafka-python) support – V0-V1 per the baseline
    - librdkafka: V1
    - KafkaJS: V3
    - Sarama: V0
    - kafka-python: V1
- SyncGroup: none due to clients (Sarama) support – V0-V1 per the baseline
    - librdkafka: V3
    - KafkaJS: V3
    - Sarama: V0
    - kafka-python: V1
- DescribeGroups: none due to clients (librdkafka, Sarama) support – V0-V1 per the baseline
    - librdkafka: V0
    - KafkaJS: V2
    - Sarama: V0
    - kafka-python: V3
- ListGroups: none due to clients (librdkafka, Sarama) support – V0-V1 per the baseline
    - librdkafka: V0
    - KafkaJS: V2
    - Sarama: V0
    - kafka-python: V2
- SaslHandshake: none due to clients (kafka-python) - V0 per the baseline
    - librdkafka: V1
    - KafkaJS: V1
    - Sarama: V1
    - kafka-python: V0
- ApiVersions: none due to clients (kafka-python) - V0-V1 per the baseline
    - librdkafka: V3
    - KafkaJS: V2
    - Sarama: V3
    - kafka-python: V0
- CreateTopics: V0-V1 due to clients (Sarama) support – V0-V2 per the baseline
    - librdkafka: V4
    - KafkaJS: V3
    - Sarama: V2
    - kafka-python: V3
- DeleteTopics: V0 due to clients (librdkafka, KafkaJS, Sarama) support – V0-V2 per the baseline
    - librdkafka: V1
    - KafkaJS: V1
    - Sarama: V1
    - kafka-python: V3

- DeleteRecords: none due to clients (Sarama) support – V0 per the baseline
  - librdkafka: V1
  - KafkaJS: V1
  - Sarama: V0
  - kafka-python: none
- InitProducerId: none due to clients (Sarama) support – V0 per the baseline
  - librdkafka: V4
  - KafkaJS: V1
  - Sarama: V0
  - kafka-python: none
- OffsetForLeaderEpoch: V0-V1
  - librdkafka: none
  - KafkaJS: none
  - Sarama: none
  - kafka-python: none
- AddPartitionsToTxn: none due to clients (librdkafka, Sarama) support – V0 per the baseline
  - librdkafka: V0
  - KafkaJS: V1
  - Sarama: V0
  - kafka-python: none
- AddOffsetsToTxn: none due to clients (librdkafka, Sarama) support – V0 per the baseline
  - librdkafka: V0
  - KafkaJS: V1
  - Sarama: V0
  - kafka-python: none
- EndTxn: none due to clients (Sarama) support – V0 per the baseline
  - librdkafka: V1
  - KafkaJS: V1
  - Sarama: V0
  - kafka-python: none
- WriteTxnMarkers: none due to baseline
- TxnOffsetCommit: none due to clients (KafkaJS, Sarama) support – V0-V1 per the baseline
  - librdkafka: V3
  - KafkaJS: V1
  - Sarama: V0
  - kafka-python: none
- DescribeAcls: V0
  - librdkafka: none
  - KafkaJS: V1
  - Sarama: V1
  - kafka-python: V1
- CreateAcls: V0
  - librdkafka: none
  - KafkaJS: V1
  - Sarama: V1
  - kafka-python: V1
- DeleteAcls: V0
  - librdkafka: none
  - KafkaJS: V1
  - Sarama: V1
  - kafka-python: V1
- DescribeConfigs: V0 due to clients (librdkafka) support – V0-V1 per the baseline
  - librdkafka: V1
  - KafkaJS: V2
  - Sarama: V2
  - kafka-python: V2
- AlterConfigs: none due to clients (librdkafka, Sarama) support – V0 per the baseline
  - librdkafka: V0
  - KafkaJS: V1
  - Sarama: V0
  - kafka-python: V1
- AlterReplicaLogDirs: V0
  - librdkafka: none
  - KafkaJS: none
  - Sarama: none
  - kafka-python: none
- DescribeLogDirs: V0
  - librdkafka: none
  - KafkaJS: none
  - Sarama: V1
  - kafka-python: none
- SaslAuthenticate: none due to baseline
  - librdkafka: V0
  - KafkaJS: V1
  - Sarama: V1
  - kafka-python: none
- CreatePartitions: none due to clients (librdkafka, Sarama) support – V0 per the baseline
  - librdkafka: V0
  - KafkaJS: V1

    - Sarama: V0
    - kafka-python: V1
- CreateDelegationToken: V0
    - librdkafka: none
    - KafkaJS: none
    - Sarama: none
    - kafka-python: none
- RenewDelegationToken: V0
    - librdkafka: none
    - KafkaJS: none
    - Sarama: none
    - kafka-python: none
- ExpireDelegationToken: V0
    - librdkafka: none
    - KafkaJS: none
    - Sarama: none
    - kafka-python: none
- DescribeDelegationToken: V0
    - librdkafka: none
    - KafkaJS: none
    - Sarama: none
    - kafka-python: none
- DeleteGroups: V0
    - librdkafka: V1
    - KafkaJS: V1
    - Sarama: V0
    - kafka-python: V1

Support for the **raw unversioned direct SASL protocol that preceded SaslHandshake and KIP-43** will also be removed in Apache Kafka 4.0.

Additional information regarding the client releases used for the analysis above:

- librdkafka 1.8.2, released on October 2021 (note that a number of clients are built on top of librdkafka – a few examples are Confluent's clients for Python, .NET & Go, node-rdkafka and rdkafka-ruby).
- KafkaJS 1.15.0, released on November 2020
- Sarama 1.30.1. released on December 2021
- kafka-python 2.0.2, released on Sep 2020

The Java client is part of the Apache Kafka repository, it is released alongside the broker and it generally supports the latest protocol versions exposed by the broker. Given that, we know that the Java client has supported the protocol API versions baseline set by Apache Kafka 2.1 as part of that release in November 2018 and no further analysis is required.

Finally, a couple of observability changes in **Apache Kafka 3.7**:

- Introduce metric `kafka.network:type=RequestMetrics,name=DeprecatedRequestsPerSec,request=(api-name),version=(api-version),clientSoftwareName=(client-software-name),clientSoftwareVersion=(client-software-version)`
- Add boolean field `requestApiVersionDeprecated` to the request header section of the request log (alongside `requestApiKey` , `requestApiVersion` , `requestApiKeyName` , etc.).

Proposed Changes

As stated in the `Public Interfaces` sections, a number of protocol API and inter broker protocol versions would no longer be supported. Kafka brokers would return the UNSUPPORTED_VERSION error when they receive a request with any of the removed API versions. Similarly, the Java clients (producer, consumer and admin) would throw UnsupportedVersionException when interacting with brokers that do not support the newly established minimum protocol API versions. A broker configured with an unsupported inter-broker protocol version would fail during start-up with an error.

Compatibility, Deprecation, and Migration Plan

Users relying on one of the clients included in this document will have to ensure they are using the minimum version described (or a newer version) with the right configuration* before upgrading their Kafka clusters to Apache Kafka 4.0. Users relying on clients not included in this document will have to ensure they are using a version that does not rely on protocol API versions that are being removed as part of this proposal. If the newly introduced `DeprecatedRequestsPerSec` metric is 0, the upgrade is safe. If it's not zero, users can inspect the request log for additional details. The newly introduced field `requestApiVersionDeprecated` makes it convenient to find the relevant request log entries.

As stated previously, the UNSUPPORTED_VERSION error will be returned in case incompatible clients are used with Kafka clusters running Apache Kafka 4.0.

Similarly, users should ensure their brokers are running at least Apache Kafka 2.1 before upgrading their Java clients to Apache Kafka 4.0.

Finally, users should ensure the brokers are running at least Apache Kafka 2.1 and `inter.broker.protocol.version` is set to 2.1 or higher before upgrading brokers to Apache Kafka 4.0. The replacement of zookeeper with KRaft in Apache Kafka 4.0 will have a stricter requirement regarding the inter-broker protocol version, so this implication is unimportant.

* *This is relevant for clients that do not auto negotiate protocol api versions and default to older versions - Sarama is one such client.*

Test Plan

- Client compatibility system tests will updated so that 2.0.x clients fail with an UNSUPPORTED_VERSION error and older versions are removed.
- Client compatibility system tests will updated so that 4.0.x clients fail with an UNSUPPORTED_VERSION error when dealing with Apache Kafka brokers that are too old (2.0 and/or 1.x).
- Broker upgrade tests will be updated so that upgrades from 2.0.x fail with an appropriate error and upgrades from older broker versions are removed.
- Protocol API integration tests will be updated so that the highest unsupported version and lowest supported version are tested and older versions are not.
- Protocol API unit tests will continue to cover all versions, but the expectations will be updated for unsupported versions.
- Test simple flows end to end with the clients mentioned in this proposal against a cluster with this KIP implemented.

Rejected Alternatives

1. Continue with the "support forever" compatibility policy: the benefit cost ratio is low given low usage of the older protocol API versions.