KIP-384: Add config for incompatible changes to persistent metadata

Status

Current state: *Under Discussion*

Discussion thread:

JIRA: 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

Kafka has several persistent formats that are used to represent cluster and partition metadata. For example, consumer offsets and group metadata are stored in the __consumer_offsets topic using an internal versioned schema. Occasionally we make incompatible changes to these formats. The config `inter.broker.protocol.version` is currently used to tell when it is safe for brokers to begin using the new schema version. The problem is that once an incompatible schema format has begun to be used, it is no longer possible to downgrade the cluster. The goal of this proposal is to decouple the persistent format from the inter-broker protocol version so that an upgrade can be tested without compromising the ability to downgrade the cluster if needed.

Public Interfaces

We will add a new broker configuration `persistent.metadata.version` which is used to control backwards incompatible changes to any persistent metadata stored by the broker. This includes the following:

1. Schemas for internal topics (e.g. __consumer_offsets and __transaction_state).
2. Metadata stored in zookeeper.
3. Partition metadata (index files, producer snapshot, leader epoch checkpoint)
4. Log metadata spanning partitions (replication checkpoint, cleaner checkpoint, etc)

The schemas for this metadata can still be upgraded using `inter.broker.protocol.version`, but only if the change is compatible. For example, changes to the metadata stored in Zookeeper is often compatible. In this case, the changes will take effect immediately when `inter.broker.protocol.version` is updated since we expect graceful downgrade to still be possible and we want users to be able to test as many new features as possible before committing to an upgraded version.

The valid values of `persistent.metadata.version` will be consistent with `log.message.format.version` and `inter.broker.protocol.version`. For example, "2.1.0" can be used to upgrade to the metadata version supported in 2.1.0 and beyond. The default value will conservatively match the value specified by `log.message.format.version`. The reasoning is that `log.message.format.version` already specifies a minimum compatible version in order to understand the message format in use.

Proposed Changes

In order to allow for graceful downgrade of Kafka, we will add a new configuration `persistent.metadata.version` which is used for incompatible changes to any of the persistent metadata schemas. When a user is testing an upgrade to

1. Bump the inter-broker format version and validate behavior
2. Once the user is confident with the behavior, they can bump `persistent.metadata.version` to the latest version and roll the cluster.
3. Finally, if there are any message format changes, the user can ensure clients have been upgraded in order to avoid conversion costs before doing another roll to upgrade the message format version.

Note that during the roll in the second step, there will be some brokers which have an updated `persistent.metadata.version` and some which do not. Since some schemas are propagated between brokers, replicas will generally take the format from the leader. For example, if the __consumer_offsets schema is bumped and a replica receives the new version before it has updated its own `persistent.metadata.version`, then it will nevertheless take the version that was written by the leader.

It is possible to update `inter.broker.protocol.version` and `persistent.metadata.version` at the same time, but doing so will prevent the possibility of downgrade. In general, once `persistent.metadata.version` has been updated to the latest version, users cannot downgrade.

Note that we require that the log message format version to not be higher than what is specified by `persistent.metadata.version`.

Compatibility, Deprecation, and Migration Plan

This change is compatible with previous versions. Obviously it can only be used for incompatible schema changes going forward.

Rejected Alternatives

1. We considered overloading `inter.broker.protocol.version` to include changes to the persistent metadata format. The main drawback is that users cannot test features which depend on the inter-broker protocol behavior without committing to the upgrade. For example, KIP-320 makes changes to replication protocol which may require additional testing.
2. We also considered overloading `log.message.format.version`. The problem here is first that this configuration can be overridden at the topic level, which does not make sense for global metadata formats. Also, upgrading the message format version is tied to performance due to the cost of conversion, so it should be treated as a separate concern. We also felt generally that there was a lot of confusion about this configuration and overloading it for other purposes would only add to that confusion.