KIP-268: Simplify Kafka Streams Rebalance Metadata Upgrade

Status

Current state: Accepted (vote)

Discussion thread: [DISCUSS] KIP-268: Simplify Kafka Streams Rebalance Metadata Upgrade

JIRA: KAFKA-6054

Released: 2.0

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

Motivation

Kafka Streams exchanges custom metadata during rebalance. This metadata has a magic version number encode to allow upgrading of the metadata format. In the past we upgraded the metadata format once (in 0.10.1.0 release), but did not design a proper upgrade path. This implies, that upgrading from 0.10.0.x versions, is only possible if the whole application is shut down first (ie, rolling bounces don't work).

In 2.0, we plan to upgrade the metadata format again and thus need to fix KAFKA-6054. The current implementation forces us, to design an upgrade path with two rolling bounced. This also requires that users configure the application correctly during upgrade (see details below). This is cumbersome and error prone for the user. Thus, we also propose a simplified upgrade path with no requirements for users to configure Kafka Streams correctly for upgrade and also to allow for single rolling bounce upgrades.

Note, that this KIP is requirement for KIP-258 and KIP-262.

Public Interfaces

For the upgrade from 0.10.0.x, ..., 1.1.x version to 2.0 version, we need to add a new configuration parameter `upgrade.from` that will be `null` by default and can take the following values:

- `null`: no upgrade required (if a new application is started or upgrade was done already)
- `"0.10.0"`, `"0.10.1"`, `"0.10.2"`, `"0.11.0"`, `"1.0"`, and `"1.1"`: for upgrading from 0.10.0.x,..., 1.1.x to 2.0

Note, that the above proposal only fixes KAFKA-6054 in 2.0. If we want to have fixes for versions 0.10.1.x, ...,1.1.x for KAFKA-6054, we would need to back port only one required value:

- `null`: for no upgrade required
- `"0.10.0"`: for upgrading from 0.10.0.x to any version of 0.10.1.x, ..., 1.1.x

Proposed Changes

We add the above config for upgrading from 0.10.0.x, ..., 1.1.x to 2.0 and future releases. We describe the upgrade path in detail below. To enable single rolling bounce upgrade with no required configuration of the applications, we add a second magic byte encoding "supported version" into the rebalance metadata that allows us to implement a "version probing" step. If we want to change the metadata in pre-2.0 releases again, this "version probing" step allows for single rolling bounce upgrade (details below).

Note, that the currently used rebalance metadata version are 1 (0.10.0.x) and 2 (0.10.1.x, ..., 1.1.x). We increase the metadata version number to 3 in 2.0 release.

Rebalance Metadata

```
// Metadata v2

SubscriptionMetadata => VersionNumber ClientId PreviousAssignedTasks StandbyTasks UserEndpoint
  VersionNumber => int32
  ClientId => int64 int64     // UUID most signification bits + least significant bits
  PreviousAssignedTasks => [TaskId]
  StandbyTasks => [TaskId]
  UserEndpoint => string     // user specified endpoint for Interactive Queries

AssignmentMetadata => VersionNumber AssignedTasks AssignedStandbyTasks GlobalAssignment
  VersionNumber => int32
  AssignedTasks => [TaskId]
  AssignedStandbyTasks => [TaskId]
  GlobalAssignment => [Host Port [Topic Partition]]]     // metadata for Interactive Queries

TaskId => GroupId Partition
  GroupId => int32
  Partition => int32

Host => string
Port => int32
Topic => string
Partitio => int32


// Metadata v3

SubscriptionMetadata => VersionNumber SupportedVersionNumber ClientId PreviousAssignedTasks StandbyTasks
UserEndpoint
  VersionNumber => int32
  SupportedVersionNumber => int32     // NEW
  ClientId => int64 int64
  PreviousAssignedTasks => numTasks [TaskId]
  StandbyTasks => numTasks [TaskId]
  UserEndpoint => string


AssignmentMetadata => VersionNumber SupportedVersionNumber AssignedTasks AssignedStandbyTasks GlobalAssignment
  VersionNumber => int32
  SupportedVersionNumber => int32     // NEW
  AssignedTasks => [TaskId]
  AssignedStandbyTasks => [TaskId]
  GlobalAssignment => [Host Port [Topic Partition]]]
```

Upgrading to 2.0:

1. prepare a jar hot swap from old version to 2.0; Kafka Streams need to be configured with `upgrade.from="<old.version>"`for startup
2. do a rolling bounce to get the new jar and config in place for each instance
    a. config `"upgrade.from"` tells the application to send `Subscription` using old endocing (version 1 or 2), to be compatible with potential version 1 or 2 leader in the group
    b. instances will receive a corresponding version 1 or 2 `Assigment` in this stage
3. user prepares a second round of rebalance; this time, the configuration parameter `upgrade.from` must be removed for new startup
4. do a second rolling bounce for each instance to get new config (ie, `upgrade.from` removed)
    a. bounced instance can send a version 3 `Subscription` as we know that leader understands version 3
    b. the leader sends version 1 or 2 `Assigment` back as long as at least one version 1 or 2 `Subsription` is received
    c. if the leader receives only version 3 `Subscirption,` it send version 3 `Assigment` and the upgrad is completed

Future upgrades with "version probing" (ie, upgrading from 2.0 to later release — only 'triggers' it metadata version number is increased):

In the current implementation, the group leader fails if it receives a subscription with a higher version number than it understands. We propose to change this: instead of failing, the leader will send an empty assignment back encoding its supported version. This allows the upgraded follower to downgrade its subscription and rejoin the group sending a subscription that the (not yet) upgraded leader understands. As we always encode the leader's supported version in the assignment, after the leader is upgrade and understand the new metadata version, all other instances can switch back to the highest supported metadata version.

Detailed upgrade protocol from metadata version X to Y (with X >= 2.0):

- On startup/rolling-bounce, an instance does not know what version the leader understands and (optimistically) sends an `Subscription` with the latest version Y
- (Old, ie, not yet upgraded) Leader sends empty `Assignment` back to the corresponding instance that sent the newer `Subscription` it does not understand. The `Assignment` metadata only encodes both version numbers (used-version == supported-version) as leader's supported-version X.
- For all other instances the leader sends a regular `Assignment` in version X back.
- If an upgrade follower sends new version number Y `Subscription` and receives version X `Assignment` with "supported-version = X", it can downgrade to X (in-memory flag) and resends a new `Subscription` with old version X to retry joining the group. To force an immediate second rebalance, the follower does an "unsubscribe()/subscribe()/poll()" sequence.
- As long as the leader (before or after upgrade) receives at least one old version X `Subscription` it always sends version `Assignment` X back (the encoded supported version is X before the leader is upgrade and Y after the leader is upgraded).
- If an upgraded instance receives an `Assigment` it always checks the leaders supported-version and update its downgraded "used-version" if possible

Thus, a single rolling bounce without any config settings is sufficient as the leader allows to probe its supported version instead of failing. Note, that if the leader is bounced last, the metadata upgrade only finishes after one more rebalance. We can trigger this rebalance with one more "unsubscribe() /subscribe()/poll()" sequence (to make sure only one instance executes this, the leader should be responsible to trigger this final rebalance – note, if the leader is not bounced last, we can detect this and avoid the additional rebalance).

Compatibility, Deprecation, and Migration Plan

Increasing the rebalance metadata version to 3 is not a backward compatible change per-se. However, the outlined upgrade path allows users to upgrade to 2.0 with zero downtime and two rolling bounces. Note, that an simplified "offline upgrade" is also possible. Instead of setting configs and doing two rolling bounces, all application instances can be stopped. Afterwards, all instances are restarted with the new 2.0 jar.

Test Plan

- unit and integration tests for StreamPartitionsAssigner that must react correctly to configs and received subscription versions
- system tests that perform rolling bounce upgrades as described above
  - this should include failure scenario during the upgrade
  - this should include "simulated upgrades" to metadata version 4, to ensure that the implemented "version probing" work correct for future changes

Test matrix:

from version	to 0.10.1.x	to 0.10.2.x	to 0.11.0.x	to 1.0.x	to 1.1.x	to 2.0.x	to post-2.0.x (simulate metadata version 4)
0.10.0.x	x (*)	x (*)	x (*)	x (*)	x (*)	x	x
0.10.1.x		x (*)	x (*)	x (*)	x (*)	x	x
0.10.2.x			x (*)	x (*)	x (*)	x	x
0.11.0.x				x (*)	x (*)	x	x
1.0.x					x (*)	x	x
1.1.x						x	x
2.0.x							x (tests "version probing")

(*): requires back porting of KAFKA-6054 to older branches

Rejected Alternatives

- use consumer's built-in protocal upgrade mechanism (ie, register multple "assigment strategies")
  - has the disadvantage that we need to implement two `StreamsPartitionAssingor` classes
  - increased network traffic during rebalance
  - encoding "supported version" in metadata (ie, version probing) subsumes this approach for future releases
  - if we want to "disable" the old protocol, a second rebalance is required, too
- Don't fix KAFKA-6054
  - it's a simple fix to include: just add one more accepted values to parameter `upgrade.from`
  - it's s fair question, how many people will run with Streams 0.10.0 – note thought, that if people are "stuck" with 0.10.0 broker, they cannot use 0.10.1 or newer as it's not backwards compatible to 0.10.0 – thus, might be more than expected
- Fix KAFKA-6054 only for 2.0 release
  - it's a relatively simply fix for older releases (main design work is already covered and writing the code is not too complex)
  - it's unclear though if we will have bug-fix releases for older versions; thus nobody might ever be able to get this code (if they don't build from corresponding dev-branches themselves)