KIP-915: Txn and Group Coordinator Downgrade Foundation

Status

Current state: Accepted

Discussion thread: here

JIRA: here

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

Motivation

Today, we don't have a downgrade story for transaction and group coordinators, and it becomes extremely difficult to downgrade once we add new fields to any of the existing record types used in their respective topics (__transaction_state and __consumer_offsets). Adding new record types or new fields is not backward compatible because older coordinators do not expect any change in the schema.

- Adding new record types: Future changes to both __consumer_offsets and __transaction_state topics may introduce new record types that are unknown to the existing coordinators. Today, the transaction coordinator fails when it tries to deserialize unknown record keys into TransactionLogKey and the group coordinator throws an IllegalStateException when the record key is unknown.
- Adding new fields to existing record types: We may introduce new fields to existing *Value* records in both topics (TransactionLogValue, GroupMetadataValue, and OffsetCommitValue) and bump their versions. The existing implementation of both coordinators throw an exception when the record type is known but the version is not supported.

The only way to downgrade is to reformat these records to older versions which is a complicated process and is further explored in Rejected Alternatives. KIP-848 introduced a new group coordinator that is able to conduct incremental, cooperative rebalances led by the server. KIP-848 significantly changes how the group coordinator works and therefore is the primary motivator for this KIP. This KIP proposes to lay a generic foundation for both coordinators so that future KIPs can illustrate simple downgrade paths.

Proposed Changes

Ignore unknown record types

This KIP proposes to ignore unknown record keys which allows the downgraded coordinator to proceed with loading the rest of the partition. As we cannot write tombstones for unknown keys, these records will be stored in the logs until the coordinators are upgraded. However, this KIP prioritizes the simplicity of ignoring them because we expect downgrades to be non-permanent.

Bump non-flexible Value record types to flexible versions

In this KIP we propose to bump each of these records to a flexible version and backport this to earlier releases, specifically 3.0.3, 3.1.3, 3.2.4, 3.3.3, and 3.4.1 (note: we will need commitment from release managers to perform the minor releases). We will also apply this to 3.5 if the patch is merged to trunk before the code freeze date. Backporting this patch to earlier releases is acceptable because the change is small and is very low risk. One limitation is that we will be unable to downgrade to a version lower than 3.X. We will mention in future release notes where a new record type or a new field is introduced that we can only downgrade to the aforementioned versions. Note that once a new tagged field is introduced in a later version, that version can never be downgraded to below the listed versions.

We will only deserialize with the flexible version but serialize with the highest known non-flexible version. This is because users upgrading to one of these versions may want to downgrade but if we serialize with the flexible version they won't be able to downgrade back to an earlier version. Deserializing a flexible version to a non-flexible version will fail.

We will rely on tagged fields (introduced in KIP-482) which allows additions and deletions of new fields without needing to bump versions. Once a version is flexible, deserializing tagged fields is straightforward as they are automatically ignored. We do not touch *Key* record types because keys are considered fixed and optional fields in keys do not make much sense.

This KIP opens the door to backward compatible changes to the record schemas through the use of tagged fields. Any future tagged fields will not require a version bump and older brokers can simply ignore the tagged fields they do not understand. **Note that introducing a new non-tagged field or removing an existing non-tagged field in the future will not be backward compatible**.

Compatible changes

Compatible changes are straightforward; the added tagged field is truly optional and is not required for the correctness of the coordinator. These fields will be ignored by the downgraded coordinator.

Incompatible changes

There are two cases: 1) we introduce a tagged field that is required for correctness, i.e. a new field that enforces correct access and without it results in incorrect coordinator behavior / data loss or 2) a non-tagged field is added. For both incompatible changes, we propose to have a version bump (already required for non-tagged fields) and add a new *isBackwardCompatible* field to the MetadataVersion enum so that the operator, if they decide, can downgrade with the *--force* option knowing that the downgrade is not backward compatible.

For 1), we can still maintain compatibility with the proposed 3.X versions above. We have 3 versions: a) pre-915 version, b) 915 version (3.X versions above), and c) post-915 version (this would add a new tagged field without changing the record version). We can do a 2-step upgrade by upgrading all (a) brokers to (b), then upgrade them to (c).

Public Interfaces

__*consumer_offsets*

GroupMetadataValue.json

Bump to flexible version

```
// KIP-915: bumping the version will no longer make this record backward compatible.
{
  "type": "data",
  "name": "GroupMetadataValue",
  "validVersions": "0-4",
  "flexibleVersions": "4+",
  "fields": [
    { "name": "protocolType", "versions": "0+", "type": "string"},
    { "name": "generation", "versions": "0+", "type": "int32" },
    { "name": "protocol", "versions": "0+", "type": "string", "nullableVersions": "0+" },
    { "name": "leader", "versions": "0+", "type": "string", "nullableVersions": "0+" },
    { "name": "currentStateTimestamp", "versions": "2+", "type": "int64", "default": -1, "ignorable": true},
    { "name": "members", "versions": "0+", "type": "[]MemberMetadata" }
  ],
  "commonStructs": [
    {
      "name": "MemberMetadata",
      "versions": "0-4",
      "fields": [
        { "name": "memberId", "versions": "0+", "type": "string" },
        { "name": "groupInstanceId", "versions": "3+", "type": "string", "default": "null", "nullableVersions":
"3+", "ignorable": true},
        { "name": "clientId", "versions": "0+", "type": "string" },
        { "name": "clientHost", "versions": "0+", "type": "string" },
        { "name": "rebalanceTimeout", "versions": "1+", "type": "int32", "ignorable": true},
        { "name": "sessionTimeout", "versions": "0+", "type": "int32" },
        { "name": "subscription", "versions": "0+", "type": "bytes" },
        { "name": "assignment", "versions": "0+", "type": "bytes" }
      ]
    }
  ]
}
```

OffsetCommitValue.json

KIP-848 bumps to a flexible version 4 and adds the topicId field. This KIP proposes to solely bump to flexible version and for KIP-848 to add topicId as a tagged field instead.

```
// KIP-915: bumping the version will no longer make this record backward compatible.
{
  "type": "data",
  "name": "OffsetCommitValue",
  "validVersions": "0-4",
  "flexibleVersions": "4+",
  "fields": [
    { "name": "offset", "type": "int64", "versions": "0+" },
    { "name": "leaderEpoch", "type": "int32", "versions": "3+", "default": -1, "ignorable": true},
    { "name": "metadata", "type": "string", "versions": "0+" },    { "name": "commitTimestamp", "type":
"int64", "versions": "0+" },
    { "name": "expireTimestamp", "type": "int64", "versions": "1", "default": -1, "ignorable": true}
  ]
}
```

__transaction_state

TransactionLogValue.json

Bump to flexible version

```
// KIP-915: bumping the version will no longer make this record backward compatible.
{
  "type": "data",
  "name": "TransactionLogValue",
  "validVersions": "0-1",
  "flexibleVersions": "1+",
  "fields": [
    { "name": "ProducerId", "type": "int64", "versions": "0+",
      "about": "Producer id in use by the transactional id"},
    { "name": "ProducerEpoch", "type": "int16", "versions": "0+",
      "about": "Epoch associated with the producer id"},
    { "name": "TransactionTimeoutMs", "type": "int32", "versions": "0+",
      "about": "Transaction timeout in milliseconds"},
    { "name": "TransactionStatus", "type": "int8", "versions": "0+",
      "about": "TransactionState the transaction is in"},
    { "name": "TransactionPartitions", "type": "[]PartitionsSchema", "versions": "0+", "nullableVersions": "0+",
      "about": "Set of partitions involved in the transaction", "fields": [
      { "name": "Topic", "type": "string", "versions": "0+"},
      { "name": "PartitionIds", "type": "[]int32", "versions": "0+"}]},
    { "name": "TransactionLastUpdateTimestampMs", "type": "int64", "versions": "0+",
      "about": "Time the transaction was last updated"},
    { "name": "TransactionStartTimestampMs", "type": "int64", "versions": "0+",
      "about": "Time the transaction was started"}
  ]
}
```

Compatibility, Deprecation, and Migration Plan

The compatibility plan is explored in proposed changes. We will backport this to all minor 3.X versions: 3.0.3, 3.1.3, 3.2.4, 3.3.3, and 3.4.1 . Downgrades to lower versions will be incompatible and will be explicitly stated in future release notes when new fields/records are introduced.

Test Plan

Rejected Alternatives

Rejected Alternative: upgraded coordinator deletes new and downgrades existing record types

Instead of the downgraded coordinator deleting the new record types when loading the partition, we can have the new coordinator delete the new record types before shutting down. This is possible with KIP-584 (feature flag) versioning approach: the operator downgrades the coordinator version which triggers coordinators to perform deletions for the new record types. We can ensure that all partitions will be compacted even if a broker is down since the partitions will have migrated to an online broker. Once coordinators append tombstones for the new record types they can explicitly trigger compaction. This introduces additional time spent cleaning up during downgrades. More importantly, coordinators need to downgrade *Value* records so that the downgraded coordinator can load committed offsets. This means group coordinators need to rewrite all offset commits with the old format, including transactional offset commits.

Rewriting transactional offset commits complicates the downgrade path:

- If a transactional offset commit is in progress, we need to abort it before reformatting but we don't have a mechanism in place to trigger a server side abort. Furthermore, we will need to add logic so that the coordinator is notified when a transaction is aborted to proceed with the rewrite.
- Producers perspective: we would either have to make the rewrite completely invisible to the producer or have the producer retry after aborting it from the server side. Both paths are complex and require additional investigation.
- Definition of a rewrite: should we consider translating the transaction start time / deadline when rewriting?

We also need a separate logic to downgrade the __transaction_state *Value* record, TransactionLogValue, but it should be simpler.

The benefit of this approach is that future record types are deleted. The proposed approach to ignore new records only works because the coordinator deletes new record types when a group is converted from new to old. However, we may introduce new record types that are not deleted during this conversion. Another benefit is that there are no strict requirements for *Value* records. We don't have to only add taggedFields (which this KIP requires) since these records will be rewritten anyways. Having the upgraded coordinator explicitly rewrite new record types and downgrade is future proof and there are no version downgrade barriers like we do for the proposed design.