KIP-NEXT: Add lowest acknowledged offset to BrokerHeartbeatResponse

Status

Current state: *Draft*

Discussion thread:

JIRA:

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

Motivation

Public Interfaces

```
{
  "apiKey": 63,
  "type": "response",
  "name": "BrokerHeartbeatResponse",
  "validVersions": "0-1",
  "flexibleVersions": "0+",
  "fields": [
    { "name": "ThrottleTimeMs", "type": "int32", "versions": "0+",
      "about": "Duration in milliseconds for which the request was throttled due to a quota violation, or zero
if the request did not violate any quota." },
    { "name": "ErrorCode", "type": "int16", "versions": "0+",
      "about": "The error code, or 0 if there was no error." },
    { "name": "IsCaughtUp", "type": "bool", "versions": "0+", "default": "false",
      "about": "True if the broker has approximately caught up with the latest metadata." },
    { "name": "IsFenced", "type": "bool", "versions": "0+", "default": "true",
      "about": "True if the broker is fenced." },
    { "name": "ShouldShutDown", "type": "bool", "versions": "0+",
      "about": "True if the broker should proceed with its shutdown." }
    // New field
    { "name": "LowestAcknowledgedOffset", "type": "int64", "versions": "1+",
      "about": "The lowest metadata offset which has been acknowledged by all unfenced brokers" }
  ]
}
```

Proposed Changes

Brokers now receive the lowest acknowledged metadata offset as part of the heartbeat response from the active controller.

Read-after-write visibility

A common pattern in Kafka is for a broker to forward write RPCs to the controller while serving read RPCs from its local cache. This is an important pattern with regards to scalability, but it introduces a problem of repeatable reads among brokers. For example, if a client is using round-robbin load balancing for its requests to brokers, two subsequent read RPCs (e.g., ListGroups, DescribeConfigs, ApiVersions) may yield different results. In ZooKeeper clusters, this can happen if the brokers process the ZK watches at different times. In KRaft, this can happen if the brokers are at different offsets in the metadata log.

A corollary to this problem is read-after-write consistency with some RPCs in KRaft. Since all write operations are forwarded to the controller, and reads are satisfied by the broker's local copy of the metadata, it is quite easy to be in a situation where a write operation is not immediately visible to the broker that forwarded the request.

In KRaft, we can solve these problems using metadata offsets. If each broker knows how caught up every other broker is, it can avoid exposing metadata that has not been fully replicated by the other brokers. This is analogous to acks=all producers for Kafka data topics.

Broker A IncrementalAlterConfigs Controller commit offset 10

Broker B FetchRequest(offset=8) Controller

Broker A FetchRequest(offset=10) Controller

Broker A HeartbeatResponse(lowest=8) Controller

Broker B HeartbeatResponse(lowest=10) Controller

ApiVersions

Following an update to metadata.version, a broker may begin using a new set of ApiVersions. In KIP-778, we proposed to close and reopen client connections to force renegotiation of compatible ApiVersions. However, one problem with this approach is that brokers will see the metadata.version update at different times. This means the ApiVersions negotiation triggered by broker A reconnecting to broker B may occur before a broker B has seen the updated metadata.version.

By including a lowest acknowledged offset, brokers will know when all other brokers have processed a particular record. In this case, a broker can wait to see a particular lowest acknowledged offset before starting the ApiVersions renegotiation. This ensures that all other brokers will have processed the metadata.version change.

Synchrony

Like all metadata in Kafka, there is some asynchrony with the handling of a given lowest acknowledged offset. The purpose of this offset is not to trigger all brokers to take some action at the same time, but rather to give all brokers a guarantee that all other brokers are at least at a particular point in their timeline of metadata changes.

Compatibility, Deprecation, and Migration Plan

- *What impact (if any) will there be on existing users?*
- *If we are changing behavior how will we phase out the older behavior?*
- *If we need special migration tools, describe them here.*
- *When will we remove the existing behavior?*

Rejected Alternatives

If there are alternative ways of accomplishing the same thing, what were they? The purpose of this section is to motivate why the design is the way it is and not some other way.