KIP-40: ListGroups and DescribeGroup

Status

Current state: *Adopted*

Discussion thread:

JIRA: *KAFKA-2687*

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

Motivation

The capability to inspect consumer group metadata such as the list of members in the group and their partition assignments is crucial for debugging, monitoring, and administration, but there is currently no API which exposes this information for the new consumer. In the initial design of the group management protocol for the new consumer, it was assumed that group metadata would be persisted in Zookeeper by group coordinators. This would have allowed tooling to inspect Zookeeper directly to obtain this metadata in the same way that it does for the old consumer. However, the alternative of storing group metadata in Kafka was suggested in KAFKA-2017, which has several advantages, such as reducing overall load on Zookeeper and maintaining a clean separation of client state (stored in Kafka) from broker state (stored in Zookeeper). At the time of writing, no general agreement has been reached on this issue, so to avoid forcing a hasty decision, the consensus seems to be to revisit the question after the 0.9 release.

With this issue momentarily tabled, we still have the problem of how group metadata can be viewed on an active cluster. Inspecting broker logs to find current group membership and assignments is not really a viable solution, even in the short term. To address this problem, we propose to add two new requests: ListGroups and DescribeGroup. The purpose of the ListGroups API is to get a simple list of the groups that are managed by each broker. To get a list of all groups in the cluster, tools will have to query each broker separately and combine the results. The second API, DescribeGroup, is used to get detailed information about an individual group. This includes a list of the current members and their subscriptions/assignments. To use this API, tools will have to send on GroupMetadataRequest to locate the coordinator for the group, then use DescribeGroup to get the detailed information.

Public Interfaces

Below we show the proposed schemas for ListGroups and DescribeGroup. The ListGroups API takes no arguments and returns a simple list of the groups and their protocol types (e.g. "consumer" or "copycat"). The DescribeGroup API takes a single groupId and returns information on the current status of the group and its members. Below we describe these details in more detail.

ListGroups

```
ListGroupsRequest =>

ListGroupsResponse => [GroupId ProtocolType]
  GroupId => string
  ProtocolType => string
```

DescribeGroup

```
DescribeGroupRequest => GroupId
  GroupId => string

DescribeGroupResponse => ErrorCode State ProtocolType Protocol Leader Members
  ErrorCode => string
  State => string
  ProtocolType => string
  Protocol => string
  Leader => string
  Members => [MemberId Host ClientId MemberMetadata MemberAssignment]
    MemberId => string
    Host => string
    ClientId => string
    MemberMetadata => bytes
    MemberAssignment => bytes
```

Proposed Changes

The ListGroups API is straightforward, so we focus on DescribeGroup below. The request is simple, but the response contains several fields worthy of further detail:

Group States: Below we list the possible group states and how it affects the associated metadata.

- *Dead:* There are no active members in the group. All other group metadata fields will be set to empty.
- *Initializing*: The group is loading metadata from its storage. All other group metadata fields will be set to empty and no member metadata will be returned.
- *Rebalancing*: The group is undergoing a rebalance. The generation, protocolType, and protocol will be set according to the prior generation. No member metadata will be returned.
- *Stable*: The group has a valid generation. Group and member metadata will be set based on the active generation.

To summarize, member metadata is only returned in the Stable state.

Member Metadata: Since the memberId is randomly generated, we must include additional information to help users identify the group member. The client host can be obtained from the session of the member's JoinGroup request and the clientId from the JoinGroup request itself. These fields will stay fixed until the next rebalance.

Since both Copycat and the new consumer use Kafka's group management, the same request can be used by administrators to inspect both types of clients (as well as any future use cases that may come up). The protocolType specifies what kind of group it is and how metadata/assignments should be decoded. For consumer-specific tooling, any groups which do not have a "consumer" protocol type can be ignored.

Error Codes: The following error codes are possible with the DescribeGroup request:

- *COORDINATOR_NOT_AVAILABLE*: The broker could not determine the coordinator for the associated groupId. Under the current implementation, this would happen if the leader of the consumer offsets topic associated with the groupId were unavailable.
- *NOT_COORDINATOR_FOR_GROUP*: The broker is not the coordinator for the associated group. The Coordinator field will be set to the host information of the coordinator. Clients should use this to determine if they need an additional query to satisfy their request.
- *NONE:* The request was satisfied successfully the the group's coordinator.

Compatibility, Deprecation, and Migration Plan

These are new requests, so obviously it will not be possible to use them on an older broker. The easiest way for tooling (such as kafka-consumer-groups.sh) would be to include an option specifying whether the group is for the old consumer or new consumer (for copycat, there is no problem). As far as we can tell, that appears to already be the plan in KAFKA-2490.

Rejected Alternatives

- If the persistence question were settled, it may be possible to query the storage system directly. For example, if group metadata were stored in Zookeeper, then tools could query it directly in the same way that they currently do. However, in line with KIP-4, it seems preferable to have tools depend only on the Kafka API since this decouples them from the storage implementation and allows for simpler access control.
- The initial version of this KIP proposed to extend the GroupMetadata request instead of adding the new request types. The main issue with this is that it leads to a complex request type which simultaneously tries to handle coordinator discovery, group listing, and group description. Separating this use cases into separate requests simplifies the design.