KIP-709: Extend OffsetFetch requests to accept multiple group ids.

Status

Current state: *"Accepted"*

Discussion thread: *here*

JIRA: ⚠ Unable to render Jira issues macro, execution error. , ⚠ *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

Currently the OffsetFetch API supports fetching offsets for a single consumer group id. This is fine from a consumer client perspective but there are classes of problems (mainly monitoring and management) that require information about multiple consumer groups. Currently these applications must submit a separate request for each group to the group coordinator. This KIP streamlines this process so that a single request can be made to fetch offsets for multiple groups. This carries the following advantages:

1. It reduces request overhead
2. It simplifies client side code

Public Interfaces

We will bump the OffsetFetch API to a new version. The new schemas are provided below. These introduce a group_id level return, note that previous top level error codes will now be return at a per-group level:

Below is the new JSON for the OffsetFetchRequest. Here we are adding a new OffsetFetchRequestGroup type which has a string for the groupId as well as a list of OffsetFetchRequestTopics

```
{
  "apiKey": 9,
  "type": "request",
  "listeners": ["zkBroker", "broker"],
  "name": "OffsetFetchRequest",
  // In version 0, the request read offsets from ZK.
  //
  // Starting in version 1, the broker supports fetching offsets from the internal __consumer_offsets topic.
  //
  // Starting in version 2, the request can contain a null topics array to indicate that offsets
  // for all topics should be fetched. It also returns a top level error code
  // for group or coordinator level errors.
  //
  // Version 3, 4, and 5 are the same as version 2.
  //
  // Version 6 is the first flexible version.
  //
  // Version 7 is adding the require stable flag.
  //
  // Version 8 is adding support for fetching offsets for multiple groups
  "validVersions": "0-8",
  "flexibleVersions": "6+",
  "fields": [
                { "name": "GroupIds", "type": "[]OffsetFetchRequestGroup", "versions": "8+", // new
                 "about": "Each group we would like to fetch offsets for", "fields": [
                        { "name": "groupId", "type": "string", "versions": "8+", "entityType": "groupId",
        "about": "The group ID."}, // new
                        { "name": "Topics", "type": "[]OffsetFetchRequestTopic", "versions": "0+",
"nullableVersions": "2+",
      "about": "Each topic we would like to fetch offsets for, or null to fetch offsets for all topics.",
"fields": [
              { "name": "Name", "type": "string", "versions": "0+", "entityType": "topicName",
                "about": "The topic name."},
              { "name": "PartitionIndexes", "type": "[]int32", "versions": "0+",
                "about": "The partition indexes we would like to fetch offsets for." }
                         ]}
                ]},
    { "name": "RequireStable", "type": "bool", "versions": "7+", "default": "false",
     "about": "Whether broker should hold on returning unstable offsets but set a retriable error code for the
partition."}
  ]
}
```

Below is the JSON for the new OffsetFetchResponse. We are adding a new type called OffsetFetchResponseGroup, which adds a new string for the GroupId, as well as a list of OffsetFetchResponseTopics. In addition, we have moved the error code as a field within OffsetFetchResponseGroup so that we can give back group level error codes in the response.

```
{
  "apiKey": 9,
  "type": "response",
  "name": "OffsetFetchResponse",
  // Version 1 is the same as version 0.
  //
  // Version 2 adds a top-level error code.
  //
  // Version 3 adds the throttle time.
  //
  // Starting in version 4, on quota violation, brokers send out responses before throttling.
  //
  // Version 5 adds the leader epoch to the committed offset.
  //
  // Version 6 is the first flexible version.
  //
  // Version 7 adds pending offset commit as new error response on partition level.
  //
  // Version 8 is adding support for fetching offsets for multiple groups
"validVersions": "0-8",
  "flexibleVersions": "6+",
  "fields": [
    { "name": "ThrottleTimeMs", "type": "int32", "versions": "3+", "ignorable": true,
      "about": "The 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": "GroupIds", "type": "[]OffsetFetchResponseGroup", "versions": "8+", // new
                 "about": "The responses per group id.", "fields": [
                        { "name": "groupId", "type": "string", "versions": "8+", "entityType": "groupId", // new
          "about": "The group ID." },
                        { "name": "Topics", "type": "[]OffsetFetchResponseTopic", "versions": "0+",
      "about": "The responses per topic.", "fields": [
        { "name": "Name", "type": "string", "versions": "0+", "entityType": "topicName",
          "about": "The topic name." },
        { "name": "Partitions", "type": "[]OffsetFetchResponsePartition", "versions": "0+",
          "about": "The responses per partition", "fields": [
          { "name": "PartitionIndex", "type": "int32", "versions": "0+",
            "about": "The partition index." },
          { "name": "CommittedOffset", "type": "int64", "versions": "0+",
            "about": "The committed message offset." },
          { "name": "CommittedLeaderEpoch", "type": "int32", "versions": "5+", "default": "-1",
            "ignorable": true, "about": "The leader epoch." },
          { "name": "Metadata", "type": "string", "versions": "0+", "nullableVersions": "0+",
            "about": "The partition metadata." },
          { "name": "ErrorCode", "type": "int16", "versions": "0+",
            "about": "The error code, or 0 if there was no error." }
          ]}
      ]},
                  { "name": "ErrorCode", "type": "int16", "versions": "8+", "default": "0", "ignorable": true,
// moved inside OffsetFetchResponseGroup
        "about": "The group-level error code, or 0 if there was no error." }
      ]}
  ]
}
```

The new level of groups information in the request is reflected in `OffsetFetchRequestData`

```
public class OffsetFetchRequestData implements ApiMessage {
    List<OffsetFetchRequestGroup> groups;
    boolean requireStable;
    private List<RawTaggedField> _unknownTaggedFields;
...
}

public static class OffsetFetchRequestGroup implements Message {
        String groupId;
        List<OffsetFetchResponseTopic> topics;
}
```

The new level of groups information in the response is reflected in `OffsetFetchResponseData`

```
public class OffsetFetchResponseData implements ApiMessage {
    int throttleTimeMs;
    List<OffsetFetchResponseGroup> groups; //replaces from List<OffsetFetchResponseTopic> topics;
    short errorCode;
    private List<RawTaggedField> _unknownTaggedFields;
...
}

public static class OffsetFetchResponseGroup implements Message {
        String groupId;
        List<OffsetFetchRequestTopic> topics;
}
```

The following will be added to `Admin.java`:

```

    /**
     * List the consumer group offsets available in the cluster for the specified consumer groups.
     *
     * @param groupSpecs Map of consumer group ids to a spec that specifies the topic partitions of the group
to list offsets for.
     *
     * @param options The options to use when listing the consumer group offsets.
     * @return The ListConsumerGroupOffsetsResult
     */
    ListConsumerGroupOffsetsResult listConsumerGroupOffsets(Map<String, ListConsumerGroupOffsetsSpec>
groupSpecs, ListConsumerGroupOffsetsOptions options);

    /**
     * List the consumer group offsets available in the cluster for the specified groups with the default
options.
     * <p>
     * This is a convenience method for
     * {@link #listConsumerGroupOffsets(Map, ListConsumerGroupOffsetsOptions)} with default options.
     *
     * @param groupSpecs Map of consumer group ids to a spec that specifies the topic partitions of the group
to list offsets for.
     * @return The ListConsumerGroupOffsetsResult.
     */
    default ListConsumerGroupOffsetsResult listConsumerGroupOffsets(Map<String, ListConsumerGroupOffsetsSpec>
groupSpecs) {
        return listConsumerGroupOffsets(groupSpecs, new ListConsumerGroupOffsetsOptions());
    }
```

New class `ListConsumerGroupOffsetsSpec` will be added to specify the partitions for each group. We can add additional per-group filters to this spec in future if required.

ListConsumerGroupOffsetsSpec

```java
/**
 * Specification of consumer group offsets to list using {@link Admin#listConsumerGroupOffsets(java.util.Map)}.
 *
 * The API of this class is evolving, see {@link Admin} for details.
 */
@InterfaceStability.Evolving
public class ListConsumerGroupOffsetsSpec {

    private Collection<TopicPartition> topicPartitions;

    /**
     * Set the topic partitions whose offsets are to be listed for a consumer group.
     * {@code null} includes all topic partitions.
     *
     * @param topicPartitions List of topic partitions to include
     * @return This ListConsumerGroupOffsetSpec
     */
    public ListConsumerGroupOffsetsSpec topicPartitions(Collection<TopicPartition> topicPartitions) {
        this.topicPartitions = topicPartitions;
        return this;
    }

    /**
     * Returns the topic partitions whose offsets are to be listed for a consumer group.
     * {@code null} indicates that offsets of all partitions of the group are to be listed.
     */
    public Collection<TopicPartition> topicPartitions() {
        return topicPartitions;
    }
}
```

`ListConsumerGroupOffsetsResult` will also change to include a mapping of group id to the map of TopicPartition and OffsetAndMetadata

```
/**
 * The result of the {@link Admin#listConsumerGroupOffsets(Map)} and
 * {@link Admin#listConsumerGroupOffsets(String)} call.
 * <p>
 * The API of this class is evolving, see {@link Admin} for details.
 */
@InterfaceStability.Evolving @InterfaceStability.Evolving
public class ListConsumerGroupOffsetsResult {

    final Map<String, KafkaFuture<Map<TopicPartition, OffsetAndMetadata>>> futures;

    ListConsumerGroupOffsetsResult(final Map<CoordinatorKey, KafkaFuture<Map<TopicPartition,
OffsetAndMetadata>>> futures) {
    }

    /**
     * Return a future which yields a map of topic partitions to OffsetAndMetadata objects.
     * If the group does not have a committed offset for this partition, the corresponding value in the
returned map will be null.
     */
    public KafkaFuture<Map<TopicPartition, OffsetAndMetadata>> partitionsToOffsetAndMetadata() {}

    /**
     * Return a map of group ids to their corresponding futures that yield a map of topic partitions to
     * OffsetAndMetadata objects. If the group doesn't have a committed offset for a specific
     * partition, the corresponding value in the returned map for that group id will be null.
     */
    public KafkaFuture<Map<TopicPartition, OffsetAndMetadata>> partitionsToOffsetAndMetadata(String groupId) {}

    /**
     * Return a future which yields all Map<String, Map<TopicPartition, OffsetAndMetadata> objects,
     * if requests for all the groups succeed.
     */
    public KafkaFuture<Map<String, Map<TopicPartition, OffsetAndMetadata>>> all() {}
```

Proposed Changes

This proposal has 3 parts: 1) extending the wire protocol 2) response handling changes, 3) enhancing the AdminClient to use the new protocol.

Wire Protocol Changes: The OffsetFetch API will be extended to take an collection of consumer group ids (Collection<String>). Previous behaviour will be maintained as single group id requests will be translated to single element arrays inside the AdminClient. The corresponding response does not currently contain the group id of the offsets fetched and so must also be extended to include this information.

Response Handling Changes: This change introduces a new level to represent group level information in org.apache.kafka.common.requests. OffsetFetchResponse and the data structure within this will need to be modified as follows:

The parameter `List<OffsetFetchResponseTopic> topics;` in `OffsetFetchResponseData` now encapsulates a single group's data rather than the entire request and so should be replaced with `List<OffsetFetchResponseGroup> groups;` where `OffsetFetchResponseGroup` is a new class described in Public Interface changes.

Any errors will be communicated at a per group id level using the existing ERROR_CODE field in the response (see above). As any errors produced should already be covered by existing behaviour no new error codes will be required. However, a new scenario is introduced whereby errors can be experienced for a subset of the groups represented in the new call. This scenario is handled by returning separate futures for each group that can be individually examined (see the all() and values()) methods in the Public Interface section. Failures for individual groups will not be automatically retried, it is the responsibility of the caller to react in the appropriate way to the communicated errors. For instance, if the group coordinator changes for one group whilst the request is in progress this will be communicated back to the client and this must resubmit a request to the appropriate new coordinator.

AdminClient changes: Consumer offsets are currently fetched in the listConsumerGroupOffsets method. Additional overloads of this method will be created that take multiple consumer groups as described under Public Interfaces.

Related/Future Work

The changes contained in this KIP refer specifically to offset listing requests. However, in order to perform these requests clients must first find the correct group coordinator. Optimisations for this process are contained in KIP-699 (KIP-699: Update FindCoordinator to resolve multiple Coordinators at a time). These 2 KIPS are not interdependent.

Compatibility, Deprecation, and Migration Plan

As this will introduce a new protocol for the FetchOffset API, we need to ensure we have backwards compatibility with old clients sending requests to the new brokers. We are keeping the old API with the single group id the same, but simply making it call the new batched group ID API with a singleton list. This should ensure that we will have backwards compatibility with old clients. Newer clients connecting to newer brokers will use the new protocol regardless of the Admin API used. Newer clients connecting to older brokers will use the older protocol version, falling back to unbatched mode with multiple requests if the new batched API is used.

Rejected Alternatives

- Replace the existing `listConsumerGroupOffsets` methods with new methods with the multiple group signature. This simplifies the AdminClient interface but it’s assumed that the primary use case for fetching offsets will still be for a single group and so we should favour this in the interfaces.