KIP-496: Administrative API to delete consumer offsets

Status

Current state: Adopted

Discussion thread: here

JIRA: Unable to render Jira issues macro, execution error. . Some of the metrics are documented in

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

Traditionally, committed offsets in Kafka were expired based on a configurable retention time. In KIP-211, retention semantics were changed to take into account group status. Basically as long as a group is still active, no offsets would be expired. This addressed the problem of losing committed offsets for low-volume partitions which rarely have new data, but it introduced two new problems:

1. The expiration timestamp of an offset was removed from the OffsetCommit protocol. Some users were relying on the ability to set this in order to force expiration of committed offsets which were no longer in use.
2. When a consumer subscription changes, we are still left with the committed offsets of the previous subscription. These will never be cleaned up as long as the group remains active. We were aware of this problem in KIP-211, but the solution was not implemented because the coordinator is presently agnostic to join group metadata and we were unclear about the compatibility implications of changing that.

The impact of these problems is that it is difficult to rely on lag monitoring using committed offsets. If a consumer is not subscribed to a topic, the lag will just grow. Also, it prevents the broker from cleaning up unused state.

Public Interfaces

Request/Response

We will add a new API to delete committed offsets.

The request schema is documented below:

```
{
  "apiKey": X,
  "type": "request",
  "name": "OffsetDeleteRequest",
  "validVersions": "0",
  "fields": [
    { "name": "GroupId", "type": "string", "versions": "0+",
      "about": "The unique group identifier." },
     { "name": "Topics", "type": "[]OffsetDeleteRequestTopic", "versions": "0+",
      "about": "The topics to delete offsets for",  "fields": [
      { "name": "Name", "type": "string", "versions": "0+",
        "about": "The topic name." },
      { "name": "Partitions", "type": "[]OffsetDeleteRequestPartition", "versions": "0+",
        "about": "Each partition to delete offsets for.", "fields": [
        { "name": "PartitionIndex", "type": "int32", "versions": "0+",
          "about": "The partition index." },
      ]}
    ]}
  ]
```

And here is the response schema:

```
{
  "apiKey": X,
  "type": "response",
  "name": "OffsetDeleteResponse",
  "validVersions": "0",
  "fields": [    { "name": "ErrorCode", "type": "int16", "versions": "0+",
      "about": "The top-level error code, or 0 if there was no error." },
    { "name": "ThrottleTimeMs", "type": "int32", "versions": "0+", "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": "Topics", "type": "[]OffsetDeleteResponseTopic", "versions": "0+",
      "about": "The responses for each topic.", "fields": [
      { "name": "Name", "type": "string", "versions": "0+",
        "about": "The topic name." },
      { "name": "Partitions", "type": "[]OffsetDeleteResponsePartition", "versions": "0+",
        "about": "The responses for each partition in the topic.",  "fields": [
        { "name": "PartitionIndex", "type": "int32", "versions": "0+",
          "about": "The partition index." },
        { "name": "ErrorCode", "type": "int16", "versions": "0+",
          "about": "The error code, or 0 if there was no error." }
      ]}
    ]}
  ]
}
```

The error sent back to the client when a group is actively subscribed to the topic is documented bellow:

```
public enum Errors {
        ...
        GROUP_SUBSCRIBED_TO_TOPIC(86, "The consumer group is actively subscribed to the topic",
GroupSubscribedToTopicException::new);
        ...
}

public class GroupSubscribedToTopicException extends ApiException {
    public GroupSubscribedToTopicException(String message) {
        super(message);
    }
}
```

Admin Client

This capability will be exposed through the admin client in the following API:

```
interface Admin {

  /**
   * Delete committed offsets for a set of partitions in a consumer group. This will
   * succeed at the partition level only if the group is not actively subscribed
   * to the corresponding topic.
   */
  DeleteConsumerGroupOffsetsResult deleteConsumerGroupOffsets(
     String groupId,
     Set<TopicPartition> partitions,
     DeleteConsumerGroupOffsetsOptions options)
}

@InterfaceStability.Evolving
public class DeleteConsumerGroupOffsetsResult {
  public KafkaFuture<Void> partitionResult(TopicPartition partition);
  public KafkaFuture<Void> all();
}

@InterfaceStability.Evolving
public class DeleteConsumerGroupOffsetsOptions extends AbstractOptions<DeleteConsumerGroupOffsetsOptions> {
}
```

Consumer Group Delete Offset options

Main action

--delete-offsets

This action should be executed independently from other Consumer Group actions.

Required Arguments

ID	Argument	Type	Description
1.	--bootstrap-server	Required	Server to connect to.
2.	--group	Required	Consumer Group ID.
3.	--topic	Required	Topics/Partitions: *--topic <topic name>:<partition numbers>* *ex: --topic topic1 --topic topic2:0,1,2*

Execution

Group Error

```
$ kafka-consumer-groups.sh --bootstrap-server localhost:9092 --delete-offsets --group unknown --topic test:0,1
--topic foo

Error: Deletion of offsets failed due to: <error message>
```

Topic/Partition

```
$ kafka-consumer-groups.sh --bootstrap-server localhost:9092 --delete-offsets --group group1 --topic test:0,1 --
topic foo

TOPIC                          PARTITION       STATUS
foo                            Not Provided    Error: <error message>
test                           0               Error: <error message>
test                           1               Error: <error message>
```

Metrics

We are also taking this opportunity to address a gap in metrics reporting. Currently we do not have any visibility about the rate of writes to the internal __consumer_offsets topic. With this KIP, there are four cases to distinguish, so we will add one meter for each.

- `kafka.server:type=group-coordinator-metrics,name=offset-commits`: Marked for every committed offset. (Note this is different than the offset commit request rate which only gives visibility at the request level)
- `kafka.server:type=group-coordinator-metrics,name=offset-expirations`: Marked for every expired offset
- `kafka.server:type=group-coordinator-metrics,name=offset-deletions`: Marked for every administrative deletion
- `kafka.server:type=group-coordinator-metrics,name=group-completed-rebalances`: Marked every time a rebalance completes (which causes a group metadata entry to be appended to the log).

Proposed Changes

To fix the problem described in the motivation, we need to make the group coordinator aware of consumer subscription semantics. The rebalance protocol was designed to be generic so that it could handle use cases beyond the consumer. For example, it is also used by Kafka Connect. Different group implementations are distinguished by a "ProtocolType" field in the JoinGroup request. For the consumer, the protocol type is "consumer." The group coordinator only allows clients of one protocol type to exist in a group.

The schema of group metadata is specific to the protocol type. For the consumer, topic subscriptions are specified in the JoinGroup request using the following schema:

```
GroupMetadata => Version SubscribedTopics UserData
  Version => Int16
  SubscribedTopics => [String]
  UserData => Bytes
```

The risk of letting the coordinator parse this schema is that it will be harder to make changes to it in the future. For example, if we want to add a new field, we will have to be careful to upgrade the broker before the clients which depend on the new field. To address this problem, we propose to let the coordinator parse this data as version 0 only. Effectively it will ignore the value of the version field and just look for the array of subscribed topics. Any trailing bytes will be ignored. Future protocol changes will have to preserve the topic list.

Expiration Semantics: With the ability to parse topic subscriptions, the coordinator will be able to maintain the set of topics that the group is currently interested in. Any committed offset for a partition which is not currently subscribed to is subject to expiration. As is the behavior today, when the group becomes empty, all offsets are subject to expiration.

Deletion Semantics: Any offset which is eligible for expiration may be deleted even if the group is still active. Usually we do not allow committed offset changes while a group is active because we do not have a mechanism to notify the group of the change. However, offsets which are awaiting expiration do not have this problem because they are not being actively consumed. Hence the deletion can be done immediately.

Compatibility, Deprecation, and Migration Plan

This is a backwards compatible change. The major impact is how it affects the future evolution of the consumer group protocol. This is documented above.

Rejected Alternatives

We have considered expiring offsets immediately when the group stops subscribing. The risk is for dynamic groups which have subscriptions changing. If the only member subscribing to a specific topic falls out of the group, then offsets would be lost. Additionally, we wanted to make the behavior consistent with empty group expiration. From an offset expiration perspective, we can treat an empty group as a case where of the subscription is empty, which makes all offsets subject to expiration.