KIP-195: AdminClient.createPartitions

Status

Current state: Accepted

Discussion thread: here

JIRA: KAFKA-5856

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

Motivation

Describe the problems you are trying to solve.

As described in KIP-4 and KIP-117 it is desirable to have network protocols and Java AdminClient APIs for administration of a Kafka cluster. One such administrative action is to increase the number of partitions of a topic. This action that can also be performed using `kafka-topics.sh --alter --topic ... --partitions ...` This KIP does not propose to change that tool, simply add an equivalent AdminClient API. Note it is not currently possible to decrease the number of partitions using the tool, and likewise this KIP only proposes to add an API for partition count increase.

Doing this is enable future work to refactor the `TopicCommand`/`kafka-topics.sh` to function via a connection to a broker rather than interacting directly with ZooKeeper.

Public Interfaces

New network protocol APIs will be added:

- `CreatePartitionsRequest` and `CreatePartitionsResponse`

The `AdminClient` API will have new methods added (plus overloads for options):

- `createPartitions(Map<String, NewPartition> newPartitions)`

Proposed Changes

AdminClient: createPartitions()

This API supports the use case of increasing the partition count via `kafka-topics.sh --alter --partitions ...`

Notes:

- This API is synchronous in the sense that the client can assume that the partition count has been changed (or the request was rejected) once they have obtained the result for the topic from the `CreatePartitionsResult`.

```
/**
 * <p>Increase the number of partitions of the topics given as the keys of {@code newPartitions}
 * according to the corresponding values.</p>
 */
public CreatePartitionsResult createPartitions(Map<String, NewPartitions> newPartitions,
                    CreatePartitionsOptions options)
public CreatePartitionsResult createPartitions(Map<String, NewPartitions> newPartitions)
```

Where:

```
/** Describes new partitions for a particular topic. */
public class NewPartitions {
    private int newNumPartitions;
    private List<List<Integer>> assignments;
    private NewPartitions(int newNumPartitions) { ... }

    /**
     * Increase the number of partitions to the given {@code newCount}.
     * The assignment of new replicas to brokers will be decided by the broker.</p>
     */
    public static NewPartitions increaseTo(int newCount) { ... }

    /**
     * <p>Increase the number of partitions to the given {@code newCount}
     * assigning the new partitions according to the given {@code newAssignments}.
     * The length of {@code newAssignments} should equal {@code newCount - oldCount}, since
     * the assignment of existing partitions are not changed.
     * Each inner list of {@code newAssignments} should have a length equal to
     * the topic's replication factor.
     * The first broker id in each inner list is the "preferred replica".</p>
     *
     * <p>For example, suppose a topic currently has a replication factor of 2, and
     * has 3 partitions. The number of partitions can be increased to 4
     * (with broker 1 being the preferred replica for the new partition) using a
     * {@code PartitionCount} constructed like this:</p>
     *
     * <pre><code>NewPartitions.increaseTo(4, Arrays.asList(Arrays.asList(1, 2))</code></pre>
     *
     */
    public static NewPartitions increaseTo(int newCount, List<List<Integer>> newAssignments) { ... }
}


public class CreatePartitionsOptions {
    public CreatePartitionsOptions() { ... }
    public Integer timeoutMs() { ... }
    public CreatePartitionsOptions timeoutMs(Integer timeoutMs) { ... }
    public boolean validateOnly() { ... }
    /**
     * Validate the request only: Do not actually change any partition counts.
     */
    public CreatePartitionsOptions validateOnly() { ... }
}

public class CreatePartitionsResult {
    // package access constructor
    Map<String, KafkaFuture<Void>> values() { ... }
    KafkaFuture<Void> all() { ... }
}
```

Network Protocol: CreatePartitionsRequest and CreatePartitionsResponse

The `CreatePartitionsRequest` is used to increase the partition count for a batch of topics, and is the basis for the `AdminClient.createPartitions()` method.

The request must be sent to the controller.

The request will require the `ALTER` operation on the `Topic` resource.

After validating the request the broker calls `AdminUtils.addPartitions()` which ultimately updates the topic partition assignment znode (`/brokers/topics/${topic}`).

The controller then waits for the change to the number of partitions to be reflected in its metadata cache before sending the `CreatePartitionsResponse`.

```
CreatePartitionsRequest => [topic_partition_count] timeout
  topic_partition_count => topic partition_count
    topic => STRING
    partition_count => count [assignment]
      count => INT32
      assignment => [INT32]
  timeout => INT32
```

Where

Field	Description
`topic`	the name of a topic
`count`	the new partition count
`assignment`	a list of assigned brokers (one list for each new partition)
`timeout`	The maximum time to await a response in ms.

Note: When a `NewPartitions` is constructed without a `newAssignments` array it results in a null `assignment` array in the `CreatePartitionsRequest`.

The response provides an error code and message for each of the topics present in the request.

```
CreatePartitionsResponse => throttle_time_ms [topic_partition_count_error]
  topic_partition_count_error => topic error_code error_message
    topic => STRING
    error_code => INT16
    error_message => NULLABLE_STRING
```

Where

Field	Description
`throttle_time_ms`	duration in milliseconds for which the request was throttled
`topic`	the name of a topic in the request
`error_code`	an error code for that topic
`error_message`	more detailed information about any error for that topic

Anticipated errors:

- `TOPIC_AUTHORIZATION_FAILED` (29) The user lacked Alter on the topic
- `INVALID_TOPIC_EXCEPTION` (17) If the topic doesn't exist
- `INVALID_PARTITIONS` (37) If the partition `count` was <= the current partition count for the topic.
- INVALID_REPLICA_ASSIGNMENT (39) if the size of any of the lists contained in the `partitions` list was not equal to the topic replication factor.
- `INVALID_REQUEST` (42) If duplicate topics appeared in the request, or the size of the `partitions` list did not equal the number of new partitions
- `REASSIGNMENT_IN_PROGRESS` (new) If a partition reassignment is in progress. It is necessary to prevent increasing partitions at the same time so that we can be sure the partition has a meaningful replication factor.
- `NONE` (0) The topic partition count was changed successfully.

Compatibility, Deprecation, and Migration Plan

This is a new API and won't directly affect existing users.

Rejected Alternatives

`NewPartitions` is inconsistent because it takes a number of partitions, but only assignments for the new partitions. One is absolute and the other is a difference. The reasons for this are:

- `NewPartitions` could take an increment, rather than the new "absolute" number of partitions. But this makes the request non-idempotent, with consequent possibilities of a double increment. This would be particularly bad because it's not possible to *decrease* the partition count.

- `NewPartitions` could take a complete assignment for both old and new partitions. This would incorrectly suggest that the request could increase the number of partitions and effect a reassignment of the existing partitions at the same time. The server would have to either ignore the old partitions (in which case why were they required to be provided?) or validate them (in which case the client has to know the old assignment in order to add more, which is needlessly difficult).

Numerous names were considered: increasePartitions, increatePartitionCount, increaseNumPartitions, addPartitions. It was felt that createPartitions() successfully implied that only an increase was possible, and was consistent with createTopics. Simiarly numerous names were considered for NewPartitions. The name of the static factory methods was chosen to alleviate the awkward semantics mentioned above, making it clear that the number argument was the new total partition count, and not an increment.

Consideration was given to whether to support non-consecutive partition ids. No use cases for non-consecutive partition ids were identified, so this is not supported.