KIP-377: TopicCommand to use AdminClient

Status

Current state: *Accepted (vote thread)*

Discussion thread: *here (and the original discussion is here)*

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

Released: 2.2

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

Motivation

Currently `kafka-topics.sh` uses only direct Zookeeper connections which is not really desired compared to the AdminClient. This change would aim to add capability to the TopicCommand to be able to connect to a broker using the AdminClient.

This is part of KIP-4 which outlines the importance of exposing admin operations via the Kafka protocol:

- Allows clients in any language to administrate Kafka (Wire protocol is supported by any language)
- Provides public client for performing admin operations
- Ensures integration test code in other projects and clients maintains compatibility
- Prevents users from needing to use the Command classes and work around standard output and system exits
- Removing the need for admin scripts (`kafka-configs.sh`, `kafka-topics.sh`, etc) to talk directly to Zookeeper.
- Allows ZNodes to be completely locked down via ACLs
- Further hides the Zookeeper details of Kafka

Public Interfaces

Command-line Options

A few extra options will be added to kafka-configs.sh:

`--bootstrap-server` option will be added to accept config changes. This will would accept a list of brokers that the internal AdminClient would use.

`--command-config` option will be also added. This would accept a file argument that points to the AdminClient connection properties file (such as SSL, buffers, etc.).

Bootstrap Server Option

```
val bootstrapServerOpt = parser.accepts("bootstrap-server", "REQUIRED: The Kafka servers to connect to,
separated by commas, for instance "localhost:9091,localhost:9092". In case of providing this, a direct
Zookeeper connection won't be required.")
                      .withRequiredArg
                      .describedAs("server to connect to")
                      .ofType(classOf[String])
val commandConfigOpt = parser.accepts("command-config", "Property file containing connection configs to be
passed to the AdminClient. " +
                      "This is used only with --bootstrap-server option.")
                      .withRequiredArg
                      .describedAs("command config property file")
                      .ofType(classOf[String])
```

Proposed Changes

The change proposed in this KIP is to add an extra option as stated above and to migrate create, delete, list and describe operations to use a broker connection. This would be a backward compatible change, leaving the zookeeper option available and fully working until a further point in time but would deprecate it as part of this KIP.

Specific behavior changes

- Providing `--bootstrap-server` and `--zookeeper` together would result in an exception as they should be mutually exclusive.
- Deleting an internal topic is allowed by the protocol and thus by this command as well. If this feature is not desirable, then ALCs must be set accordingly on the server side or the `delete.topic.enable=false` topic level config must be set.

Compatibility, Deprecation, And Migration Plan

This KIP won't implement topic config alternation as that is deprecated in the TopicCommand and should be done by `kafka-configs.sh`. The only alternation we allow is changing the partition number for topics.

No other existing behavior would be removed and the implementation would be done in a backward compatible way.

Also retrieving the list of topics that are marked for deletion won't be implemented now as currently it's not possible to retrieve via any protocols. This conversation is part of KIP-142. The implementation will add a note regarding this in its output.

As part of the `--zookeeper` option deprecation we will mark it as deprecated in the command help but also print out a warning message about using this deprecated option.

Test Plan

The existing tests will be run with the `--bootstrap-server` mode too. Additionally we can refactor some of the `kafka-topics.sh` usages in the smokes to use the AdminClient mode.

Rejected Alternatives

Protocol Changes

Topics Marked For Deletion

Currently `KafkaAdminClient.describeTopics()` and `KafkaAdminClient.listTopics()` uses the Metadata protocol to acquire topic information. The returned response however won't contain the topics that are under deletion but couldn't complete yet (for instance because of some replicas offline), therefore it is not possible to implement the current command's "marked for deletion" feature. To get around this there were several alternatives that can be seen below but during the discussion we decided to keep this work in KIP-142. The idea in this KIP was that we could introduce some changes in the Metadata protocol, such as:

- Cache topics that are under deletion but some of their replicas are offline.
- Create a new error, called `TOPIC__DELETION_IN_PROGRESS`

- Bump the Metadata request version. The format of the protocol won't change, only the fact that there is a new Error type that we're introducing, but that requires bumping the protocol as old clients won't be able to handle it and most probably end up in an `UNKNOWN_SERVER_ERROR`.
- Smarten up the `KafkaApis.handleTopicMetadataRequest` to also return the list of topics under deletion with the above error

TOPIC_DELETION_IN_PROGRESS

```
public enum Errors {
    // ...
    TOPIC_DELETION_IN_PROGRESS(74, "Topic deletion is in progress.",
        TopicDeletionInProgressException::new);
    // ...
}


public class TopicDeletionInProgressException extends ApiException {

    private static final long serialVersionUID = 4767321103391338488L;

    public TopicDeletionInProgressException(String message) {
        super(message);
    }
}
```

AlterTopics Protocol

At an early stage of the KIP discussion it occured that there is a need for a protocol that would handle topic partition changes, such as increasing the partition number. It got rejected as a similar api, called CreatePartitions already exists and we don't need a new protocol.

For archiving purposes here is the protocol:

AlterTopics Protocol Request and Response

```
AlterTopics Request (version: 0) => validate_only [topic_change]
  validate_only => BOOLEAN
  topic_change => topic_name target_partition_number
    topic_name => STRING
    target_partition_number => INT32

AlterTopics Response (version: 0) => throttle_time_ms [topic_change_result]
  throttle_time_ms => INT32
  topic_change_result => topic_name error_code error_message
    topic_name => STRING
    error_code => INT16
    error_message => NULLABLE_STRING
```

Authorization implications:

- From the authorization perspective using this protocol would require an ALTER operation on a Topic resource. Both currently available and therefore can be used.