KIP-170: Enhanced TopicCreatePolicy and introduction of TopicDeletePolicy

Status

Current state: *"retired" - superseded by KIP-201: Rationalising Policy interfaces*

Discussion thread: *here*

JIRA: *KAFKA-5497*

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

Motivation

As stated in KIP-108, the operators of a Kafka cluster need to validate users' self-service topic creation and deletion.

In the case of IBM MessageHub we found that the decision about permitting a topic creation requires more information than just whether the request metadata falls within acceptable boundaries.
We also needed to validate that against the current amount of resources already used in the cluster (eg number of partitions).

We also found that the action of deleting a topic had to be validated against the topic being in use by other services/applications.
This last point has the implicit assumption that such usages can be easily checked against a registry of some sort.

This KIP proposes to extend KIP-108 with :

- *Provide the Create Topic Policy with a convenient API for querying the cluster topics' metadata*
- *Introduce a Delete Topic Policy*
- *A newer DeleteTopicRequest protocol message version that can include a boolean for verification only*
- *A newer DeleteTopicResponse message version that can include an error message to be returned to the user*

NOTE:

The initial interface proposed for querying the cluster metadata has intentionally been kept with somewhat narrow capabilities (what MessageHub actually required)
rather than being provided with speculative generality. We await major community feedback on this point.

The Delete policy we had to implement had no requirements on querying cluster metadata, but it should be easy to make it similar to the Create interface

KIP authors Edoardo Comar and Mickael Maison

Public Interfaces

New Java interfaces in the client project corresponding to pluggable policies :

```
package org.apache.kafka.server.policy;
/**
 * Enhanced topic creation policy
 */
public interface CreateTopicPolicyV2 extends CreateTopicPolicy {
  /**
   * A queryable provider of metadata
   */
  interface TopicPartitionMetadataProvider {
      /**
       * @returns a Map with all topics in the cluster and their corresponding number of partitions
       */
      Map<String, Integer> topicsPartitionCount();
      /**
       * checks if the topic is marked for deletion
       */
         boolean isTopicMarkedForDeletion(String topicName);
  }
  public void setResourceProvider(TopicPartitionMetadataProvider provider);
}

/**
 * A topic deletion policy
 */
public interface DeleteTopicPolicy extends Configurable, AutoCloseable {
    void validate(String topicName) throws PolicyViolationException;
}
```

New configuration in `server.properties` :

`delete.topic.policy.class.name` : The delete topic policy class that should be used for validation. The class should implement the `org.apache.kafka.server.policy.DeleteTopicPolicy` interface.

New versions of DeleteTopicRequest/DeleteTopicResponse protocol messages:

```
DeleteTopics Request (Version: 1) => [topics] timeout validate_only
  topics => STRING
  timeout => INT32
  validate_only => BOOLEAN

DeleteTopics Response (Version: 1) => [topic_error_codes]
  topic_error_codes => topic error_code error_message
    topic => STRING
    error_code => INT16
    error_message => NULLABLE_STRING
```

Proposed Changes

`kafka.server.AdminManager` will be enhanced

- to instantiate a TopicDeletePolicy - in a similar fashion as what it currently does for the topic creation
- to invoke the valdate method of TopicDeletePolicy when handing a request to delete topics
- to skip performing the actual deletion (via `AdminUtils.deleteTopic`) if the request is for validate only

- check if a topic creation policy implements `CreateTopicPolicyV2` and in that case set a `CreateTopicPolicyV2.TopicPartitionMetadataProvider` on the policy

The TopicPartitionMetadataProvider interface could be implemented by the kafka.server.AdminManager itself or by another class, depending on how rich the Provider interface will turn out to be.

For the initial proposal, both provider methods

```
  Map<String, Integer> topicsPartitionCount();
  boolean isTopicMarkedForDeletion(String topicName);
```

can be easily implemented by `AdminManager`

The metadata itself can be retrieved by enhancing the `MetadataCache` held by the `AdminManager`.

Compatibility, Deprecation, and Migration Plan

- *What impact (if any) will there be on existing users?*
  - *None*
- *If we are changing behavior how will we phase out the older behavior?*
  - *N/A*
- *If we need special migration tools, describe them here.*
  - *Not needed*
- *When will we remove the existing behavior?*
  - *N/A*

Rejected Alternatives

If there are alternative ways of accomplishing the same thing, what were they? The purpose of this section is to motivate why the design is the way it is and not some other way.