KIP-78: Cluster Id

Status

Current state: *Adopted*

Discussion thread: http://kafka.markmail.org/thread/glcyw3bvngtvobbs

JIRA: KAFKA-4093

GitHub PR*:* https://github.com/apache/kafka/pull/1830

Released: 0.10.1.0

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

Motivation

It is increasingly common for organisations to have multiple Kafka clusters and there are a number of situations where it would be useful if the different clusters could be uniquely identified: monitoring, auditing, log aggregation, preventing clients and brokers from connecting to the wrong cluster and more. This KIP focuses on the monitoring and auditing use cases while ensuring that the design can be evolved to satisfy other use cases (this is covered in more detail in the "Potential Future Improvements" section).

One point worth covering is whether we already have something that could be used to uniquely identify a cluster. Today, each Kafka Cluster contains a set of brokers and an associated set of Zookeeper instances. You can identify each broker by a hostname or IP address (or a set of hostnames and IP addresses) and port (or ports), and can identify each Zookeeper instance by a hostname or IP address (or a set of hostnames and IP addresses) and port (or ports). Unfortunately, sets of hostnames, IP addresses, and ports are not a great way to uniquely identify a cluster as they might change over time. Brokers (and zookeeper instances) might die and be replaced with new instances on new machines, or might be moved to different machines. Clusters might be consolidated together, or hosts might be reused in new clusters.

Given that, we propose the introduction of a new concept: a Kafka cluster id.

Goals

- All Kafka clusters should have a cluster id (i.e. it should not be optional) in order to avoid complex logic in downstream tools that require it.
- No additional work should be required of users when creating or upgrading a Kafka cluster (e.g. no additional mandatory configs should be introduced).
- The cluster id should be unique and immutable so that external tools can associate a message with a cluster. If the cluster id is allowed to change, the correlation between messages and clusters breaks (i.e. a single cluster will appear as multiple clusters).
- It is acceptable for users to be able to set or update the cluster id via expert-level tools even though that can cause the uniqueness and immutability guarantees to be violated. Users are recommended not to do this apart from exceptional situations.
- The cluster id should be available via broker metrics so that it's easily exported to monitoring tools with existing metric reporters (e.g. JmxReporter).
- The cluster id should be exposed to client and broker metric reporters so that they can tag, categorise or namespace metrics based on the cluster id.
- Client interceptors should have access to the cluster id so that they can associate it with the message metadata being tracked.
- Client serializers should have access to the cluster id so that they can include it in the message, if desired (e.g. as part of a standard message header).

Public Interfaces

Overview

We propose adding the `cluster_id` to the Metadata response, storing it in ZooKeeper, exposing it via metrics, to serializers and client interceptors. The `cluster_id` should be unique, immutable, auto-generated and it may be used in situations where the allowed character set is restricted. Given this, we limit the allowed characters to the following regular expression [a-zA-Z0-9_\-]+, which corresponds to the characters used by the URL-safe Base64 variant.

Protocol

We will introduce a `cluster_id` field to `MetadataResponse` and we will make no changes to `MetadataRequest`. Both will be bumped to version 2:

```
Metadata Request (Version: 2) => [topics]
  topics => STRING


Metadata Response (Version: 2) => cluster_id [brokers] controller_id [topic_metadata]

  cluster_id => STRING
  brokers => node_id host port rack
    node_id => INT32
    host => STRING
    port => INT32
    rack => NULLABLE_STRING
  controller_id => INT32
  topic_metadata => topic_error_code topic is_internal [partition_metadata]
    topic_error_code => INT16
    topic => STRING
    is_internal => BOOLEAN
    partition_metadata => partition_error_code partition_id leader [replicas] [isr]
      partition_error_code => INT16
      partition_id => INT32
      leader => INT32
      replicas => INT32
      isr => INT32
```

Java APIs

We will add an `id` field to the `Cluster` class, which is populated from `MetadataResponse` and accessible from the internal `Metadata` class. We will also introduce the following class and interface:

```
package org.apache.kafka.common;

class ClusterResource {

  private final String clusterId;

  public ClusterResource(String clusterId) {

    this.clusterId = clusterId;

  }

  public String clusterId() {

    return clusterId;

  }

}


package org.apache.kafka.common;

interface ClusterResourceListener {

  void onUpdate(ClusterResource cluster);

}
```

Client interceptors, serializers and metric reporters who *optionally* implement the interface `ClusterResourceListener` will receive an instance of `ClusterResource` once the cluster id is available. For clients, that is when a client receives a metadata response (that means that methods like `ProducerInterceptor.onSend` may be called before the cluster id is available). For brokers, that is during start-up, shortly after the connection to ZooKeeper is established (the broker case is only relevant for metric reporters).

ZooKeeper and Tools

We propose the introduction of a new ZooKeeper znode /cluster/id where we store a unique and immutable id automatically generated by the first broker that is able to create the znode. For consistency with other znodes stored by Kafka, we will store the data as JSON. An example follows:

```
{
  "version": 1,
  "id": "vPeOCWypqUOSepEvx0cbog"
}
```

Proposed Changes

Broker

We update `KafkaServer.startup()` to invoke `getOrGenerateClusterId()` after `initZk()`.

The implementation of `getOrGenerateClusterId()` is straightforward. It first tries to get the cluster id from `/cluster/id`. If the znode does not exist, it generates an id via `UUID.randomUUID()`, converts it to a String via URL-safe Base64 encoding and tries to create the znode. If the creation succeeds, the generated id is returned. If it fails with a ZkNodeExistsException, then another broker won the race, so we can just retrieve the cluster id from the expected znode path and return it. The returned cluster id will be passed to registered metric reporters who implement ClusterResourceListener and it will be logged at `info` level.

We will introduce a `clusterId` method to `KafkaServer`, which will be used by `KafkaApis.handleTopicMetadataRequest` to populate the `cluster_id` field in `MetadataResponse` version 2.

Finally, we will add a Yammer metrics `Gauge` that exposes the cluster id via `kafka.cluster:type=Cluster,name=ClusterId` in `KafkaServer`.

It's worth noting that the cluster id will only be stored in ZooKeeper. As such, if the ZooKeeper path is changed or the ZooKeeper state is lost, the admin will have to manually set the `/cluster/id` znode if it wants the cluster id to remain the same. This is deemed acceptable for the monitoring and auditing use cases that this KIP aims to address. This can be improved in the future by also storing the cluster id in each broker after the first connection as per point 3 in the "Potential Future Improvements" section. There are a number of edge cases that need to be considered, so we think it's best to address that in a subsequent KIP.

Clients

Client interceptors, serializers and metric reporters which implement the `ClusterResourceListener` interface (checked via `instanceof`) will receive the current `Cluster` instance after every metadata response received by a consumer or producer. We will also log the cluster id received via the metadata response if it's different from the previously received cluster id (or if it is the first one). The clients will use an implementation of `Metadata.Listener` to hook into metadata updates.

Compatibility, Deprecation, and Migration Plan

A new protocol version will be added for the `Metadata` protocol message, so it should not affect existing clients.

`ClusterResourceListener` is a new interface so existing serializers, client interceptors and metric reports will not be affected unless they choose to implement it.

Tools that want to take advantage of the cluster id while supporting older brokers will have to handle the fact that the older brokers won't have a cluster id.

Test Plan

1. Test that cluster id is generated correctly.
2. Extend `Metadata` protocol tests to check that the cluster id is always returned.
3. Modify the existing upgrade system tests to verify that the cluster id is set after the upgrade.
4. Test that the cluster id is exposed correctly in client and broker metrics.
5. Test that verifies that serializers, client interceptors and metric reporters that implement `ClusterListener` receive the cluster id once a metadata response is received.
6. Test that metric reporters in the broker receive the cluster id once it's available.

Rejected Alternatives

1. Add a method in each interceptor interface that is called on metadata updates to pass the cluster id (and potentially other metadata)

This seems like a good option, but it's not backwards compatible while we still support Java 7. In Java 8, we would have the option to introduce a default method with an empty implementation. This is also specific to interceptors. If we wanted to apply the same approach to metric reporters and serializers, we would probably want a common interface (as in the current proposal).

2. Expose the cluster id in ProducerRecord and ConsumerRecord

This would make it available to some of the client interceptor methods. The cluster id looks out of place in either of these classes as they represent records while the cluster id is a mostly static piece of data.

3. Don't expose the cluster id to client pluggable interfaces

We leave it to the implementations to do the metadata request themselves if they need the data. Since we don't have public API for this yet, it would require using internal request classes or reimplementing them.

4. Pass the cluster id via the configure method of Configurable classes

In order to make the cluster id available via the `configure` method, we would need to delay its invocation until after we get the first metadata response for each client.

The `configure` method of interceptors and serializers would no longer be invoked during the constructor of `KafkaConsumer` and `KafkaProducer`. Instead, it would be invoked after the first metadata response is received. This could pose problems for implementations that were relying on the previous behaviour. Furthermore, the first `ProducerInterceptor.onSend()` call would happen after the first metadata response is received instead of immediately after `KafkaProducer.send()` is called.

5. Pass an instance Cluster to the listener

Using `Cluster` is appealing because it has the information we need and it is already a public class, so we would not need to introduce a new class with a name that can potentially be confusing. There are a few issues, however:

- On the broker, we would have to invoke the listener on every topic/partition change instead of just once during broker startup
- Cluster only includes one endpoint, which makes sense for the client but not the broker where it's not clear which endpoint should be used
- It is unclear if it's useful for serializers, interceptors and metric reporters to know all brokers endpoints and topic/partition metadata

Potential Future Improvements

1. Expose cluster information including cluster id via AdminClient: this is likely to happen as part of KIP-4.
2. Add a human-readable cluster name to complement the id. This is useful, but it's worth exploring separately as there are a few options in the design space. One option that looks promising is to allow user-defined tags on resources (cluster, brokers, topics, etc.). There are also simpler (but less general) alternatives like setting the cluster id via a broker config or via a ZK-based cluster config.
3. Use the cluster id to ensure that brokers are connected to the right cluster: it's useful, but something that can be done later via a separate KIP. One of the discussion points is how the broker knows its cluster id (e.g. via a config or by storing it after the first connection to the cluster).
4. Use the cluster id to ensure that clients are connected to the right cluster: given that clients could potentially connect to multiple clusters for failover reasons (i.e. `bootstrap.servers` can point to a VIP), it may make sense for the config be a list of cluster ids or a cluster id regex.
5. Expose cluster id as a client metric. Kafka Metrics doesn't support non-numeric metric values at the moment so it would have to be extended to support that.