KIP-454: Expansion of the ConnectClusterState interface

Status

Current state: *Accepted* (2.3.0)

Discussion thread: here

JIRA: KAFKA-8231 - Getting issue details... STATUS

PR: https://github.com/apache/kafka/pull/6584

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

Motivation

REST extensions for Kafka Connect were introduced via KIP-285: Connect Rest Extension Plugin. Part of the motivation for this KIP was to allow "complex extensions" to "provide filters that rewrite/validate the connector requests to enforce additional constraints on the connector configurations", and the KIP did include the addition of a `ConnectClusterState` interface which would be used to provide information about the Connect cluster to these REST extensions. However, the information provided by the `ConnectClusterState` interface is somewhat limited at the moment and only includes a list of connectors and the ability to query for the health of a specific connector, which includes the state of the connector as well as the states of its tasks. Expanding the `ConnectClusterState` interface to include information such as the configurations of connectors and their tasks, and the ID of the current Kafka cluster, would make writing these complex extensions easier.

Public Interfaces

As the title of this KIP suggests, the changes will affect the `ConnectClusterState` interface and its only current implementation, the `ConnectClusterStateImpl` class.

Additional methods to add to the `ConnectClusterState` interface:

ConnectClusterState interface - additional methods

```
/**
 * Lookup the current configuration of a connector. This provides the current snapshot of configuration by
querying the underlying
 * herder. A connector returned by previous invocation of {@link #connectors()} may no longer be available and
could result in {@link
 * org.apache.kafka.connect.errors.NotFoundException}.
 *
 * @param connName name of the connector
 * @return the configuration of the connector for the connector name
 * @throws org.apache.kafka.connect.errors.NotFoundException if the requested connector can't be found
 */
Map<String, String> connectorConfig(String connName);

/**
 * Get details about the setup of the Connect cluster.
 * @return a {@link ConnectClusterDetails} object containing information about the cluster
 **/
ConnectClusterDetails clusterDetails();
```

A new `ConnectClusterDetails` interface will be added as well, that contains immutable information about the Connect cluster:

ConnectClusterDetails - new interface

```
package org.apache.kafka.connect.health;

public interface ConnectClusterDetails {

  /**
   * Get the cluster ID of the Kafka cluster backing this Connect cluster.
   * @return the cluster ID of the Kafka cluster backing this connect cluster
   **/
  public String kafkaClusterId();
}
```

This interface will only provide the ID of the backing Kafka cluster for now, but may be expanded in the future to include the mode of the Connect worker (standalone, distributed, perhaps embedded(?)), the group ID of a distributed cluster, or other information.

Proposed Changes

The basic idea here was to add as much information to the `ConnectClusterState` interface as is available via the Connect REST API; this includes all currently-available read-only methods of the `Herder` interface. However, due to lack of a convincing use case, information on task configurations will be left out for now.

For some of these methods, such as `connectorConfig(...)`, communication with the underlying `Herder` will be necessary. The return values will reflect the most up-to-date information that the worker has available locally; the herder will not forward requests to the leader of the worker group. An exception will be thrown if the herder is expecting an impending rebalance. This aligns with the behavior of the current `ConnectClusterStateImpl` class

Right now, it is possible for request filters added via a REST extension to "intercept" new connector configurations that are submitted via the REST API (through a `PUT` request to `/connectors/<name>/config`, or a `POST` request to `/connectors`, for example) and validate and/or modify them.

However, some validation logic, like ensuring that certain connector configuration properties are never modified by unauthorized users, is difficult without knowledge of the current configuration of the connector.

Additionally, the Kafka cluster ID may be useful for the purpose of uniquely identifying a Connect cluster from within a REST extension, since users may be running multiple Kafka clusters and the `group.id` for a distributed Connect cluster may not be sufficient to identify a cluster.

Compatibility, Deprecation, and Migration Plan

Users who have written their own `ConnectClusterState` implementations will have to implement these additional methods if they would like to develop against releases of AK that contain the changes here. Since this interface is already implemented by the Connect framework this is unlikely to be a problem. However, should a developer of a custom `ConnectClusterState` encounter compile-time problems after migrating to a new release that includes these new interface methods, they will have several options available including throwing `UnsupportedOperationExceptions`, providing empty implementations, or providing actual implementations.

Since this is a feature addition and not a bug fix, the targeted version is the upcoming 2.3 release.

Rejected Alternatives

Adding default implementations for new methods

Adding non-default methods to the interface is technically a backwards incompatible change for anyone developing their own `ConnectClusterState` imp lementation. Adding defaults for these methods, even if they immediately throw exceptions, solves this problem. However, that would defeat the purpose of implementing the interface in the first place–users looking up the `ConnectClusterState` javadocs, for example, would expect those methods to be available. In order to reduce confusion for what is likely to be the most common case, no default methods will be added.

Exposing task configurations

Although information on task configurations is readily available from Herder instances, there has yet to be a convincing use case for exposing information on task configurations to REST extensions. This functionality can always be added later in a separate KIP; for now, we'll err on the side of caution and not add support for a feature that may not be used by anyone.

Query the Connect REST API from within the extension itself

For example, to compare a new configuration for a connector to its current configuration, make a `GET` request to `/connectors/<connName>/` and read the current connector configuration from the response. Rejected because the Connect worker's REST interface may be restricted for security purposes, and requiring users to provide extra security configuration information to a REST extension would be painful and may not be possible in all circumstances. Implementation would also be more difficult for writers of the REST extension than necessary. Additionally, if this were a recommended means of collecting information about a Connect cluster from within a REST extension, the `ConnectClusterState` interface in its current form would be entirely redundant.

Create an AdminClient

To deduce the Kafka cluster ID, create an `AdminClient` based on the worker configuration properties provided to the extension in its `configure(...)` method, then get the cluster ID using that client. Rejected because, like above, implementation would be more difficult for writers of the REST extension than necessary.

Consume from the Connect configuration topic

To deduce the configurations of connectors and tasks, read directly from the `config.storage.topic`. Rejected because, like above, implementation would be more difficult for writers of the REST extension than necessary. Also, as with the rejected alternative of querying the Connect REST API from within the extension itself, if this were a recommended means of collecting information about a Connect cluster from within a REST extension, the status information of connectors could be read from the `status.storage.topic`, and the `ConnectClusterState.connectorHealth(...)` method would be partially if not entirely redundant.