KIP-883: Add isDeleted flag when stopping a connector

Status

Current state: *Under Discussion*

Discussion thread: *here*

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

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

Motivation

The base class for all Source and Sink connectors is the `org.apache.kafka.connect.connector.Connector` abstract class. This class defines a few abstract methods (e.g. `void start(Map<String, String> props)` and `void stop()`) that are then implemented by Connectors and are invoked as part of their lifecycle.

As connectors interact with external systems, they sometimes need to provision external resources. We can imagine a Sink connector that creates new queues in a Messaging System before writing messages to them, or a Source connector that activates an account before sending requests to a source system, among other use cases. A more concrete example (and one that concerns us in particular) is a source connector that audits database changes by creating an "audit" table and sets up database triggers to insert row-level updates to that table.

There are cases where we might want to cleanup these resources when the connector that uses them is deleted. It can be to save costs (e.g. the external system charges per active account), or compute resources (triggers writing database updates to an audit table that will no longer be read from should be removed, and so does the audit table). Taking other actions, like deleting the connector offsets (as discussed on KIP-875) might also be considered as part of this cleanup process.

The Connector API in Kafka Connect does not provide any hooks to indicate when a connector has been deleted, so it is currently not possible to react to this event. The `stop()` method in its current form cannot be used for this purpose, as a Connector can be stopped and restarted at any point (e.g. when its configuration changes).

Public Interfaces

org.apache.kafka.connect.connector.Connector

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

public abstract class Connector implements Versioned {

    /**
     * Stop this connector.
     */
    public abstract void stop();

    /**
     * Stop this connector, and also indicate if the connector has been deleted.
     * <p>
     * Connectors are not required to override this method, unless they need to perform some cleanup
     * actions in cases where the connector has been deleted.
     *
     * @param isDeleted indicates if the connector has been deleted.
     */
    public void stop(boolean isDeleted) {
        stop();
    }
}
```

Proposed Changes

Add an overload to the `void stop(boolean isDeleted)` method to the Connector public API, with a default implementation that calls the existing `void stop()` method. This new method can then be overridden by connectors that need to take any additional steps (remove assets, delete offsets, etc) as part of the deletion process.

Worker class

Both StandaloneHerder and DistributedHerder invoke methods from the `Worker` class to start/stop the connector instance. This KIP will overload the `Worker#stopAndAwaitConnector(...)` method, passing a flag indicating that the connector has been deleted. This flag be passed to the new `Connector#stop(isDeleted)` method, so Connectors can implement any additional logic needed.

Compatibility, Deprecation, and Migration Plan

The proposed change is fully backward-compatible with existing Kafka Connect releases. The new method added to the public interface includes an default implementation of the new method, so existing connectors don't need to override it if not needed.

Test Plan

Integration tests will be added to make sure that the new method is invoked when a connector is deleted. Mainly:

- Add/update unit tests to WorkerTest and WorkerConnectorTest classes.
- Add integration tests in `ConnectWorkerIntegrationTest` and auxiliary classes (`EmbeddedConnectClusterAssertions`,`SampleSourceConnector`,`SampleSinkConnector` etc.)

Rejected Alternatives

Add new destroy()/onDelete() method to the Connect API

Initially we thought about adding a new destroy() method to the Connector class. The idea was to call this method on WorkerConnector#doShutdown(), right after the connector.stop() is executed. This however presented some questions around the execution guarantees, for example, what the behavior would be when the Connector#stop() method never returned, or the method throws an exception. To make things simpler, an overloaded Connector#stop (boolean isDeleted) was introduced instead, so the expected behavior remains the same as with the current implementation. That is, the method is guaranteed to be called if the connector stops within CONNECTOR_GRACEFUL_SHUTDOWN_TIMEOUT_MS (default: 5 secs)

Delete provisioned resources out-of-band

In theory, users can monitor Kakfa Connect configuration topics to determine if/when a connector has been deleted. Reacting to this event outside of the connector's context is probably not very useful, as there might not be enough contextual information to perform any meaningful action. It is also better to keep these concerns encapsulated within the connector framework itself.