KIP-767: Connect Latency Metrics

Status

Current state: *Under Discussion*

Discussion thread: *here*

JIRA:

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

Motivation

For many sink connectors, a key indicator is the latency between message arrival in the source and the Connector's successful persistence to the sink. At present Connect does not provide a common way to introduce this metric.

This KIP proposes to introduce a latency metric that measures the average and max latency of individual records handled by Connect.

In the case of SourceConnectors, the runtime can already calculate these metrics as it has access to the futures returned by the Kafka producer and can compare time at completion against the SourceRecord's timestamp.

The runtime does not currently have a way to capture the time at which a SinkRecord is persisted to the sink as the put method returns void. Some connector implementations send the records asynchronously after put which is encouraged as per the java docs "Put the records in the sink. Usually this should send the records to the sink asynchronously and immediately return" and thus we cannot simply measure the time elapsed after put returns. We must introduce a method by which the connector can signal a record has been acked by the sink. To do so, we propose to expand upon the idea of the ErrantRecordReporter introduced in Kafka 2.6 to include a SinkRecordReporter.

Public Interfaces

SinkRecordReporter

Define a new SinkRecordReporter interface and class returned by the SinkTaskContext:

```
public interface SinkRecordReporter {
  # calls the existing ErrantRecordReporter.report
  Future<Void> reportErrantRecord(SinkRecord record, Throwable error);

  # trigger the SinkConnectorInterceptor.onSuccess
  void reportSuccessfulRecord(SinkRecord record);
}
```

SinkTaskContext

The SinkTaskContext needs to expose the new SinkRecordReporter in the same way it did for the ErrantRecordReporter:

```
public interface SinkTaskContext {
  ...


  # Java docs will include usage example of catching NoSuchMethodError and NoClassDefError
  # for compatibility with older Connect Runtime versions
  default SinkRecordReporter sinkRecordReporter() {
    return null;
  }
}
```

Proposed Changes

This SinkRecordReporter will provide methods to indicate the completion of an individual record as it is acked from the Sink as well as a method to indicate that a record has resulted in an error. In the case of an error, the SinkRecordReporter will call the existing ErrantRecordReporter. Calling the ErrantRecordReporter maintains compatibility for connectors already using the ErrantRecordReporter as well as provides it's functionality to any connectors that switch over to exclusively using the SinkRecordReporter.

Compatibility, Deprecation, and Migration Plan

The SinkRecordReporter will delegate reportErrantRecord calls to the existing ErrantRecordReporter.report method to preserve compatibility with that method. We will mark the SinkTaskContext.errantRecordReporter as deprecated in favor of using the SinkRecordReporter.

Calls to SinkTaskContext.sinkRecordReporter from Connector implementations would throw NoClassDefErrors when the Runtime has not been upgraded. The javadocs will provide examples of catching and handling these exceptions. This is the same handling used for the introduction of the ErrantRecordReporter.

In order to provide some metrics for SinkConnector implementations that do not yet use the SinkRecordReporter, we will calculate default average and max latency metrics by using the time after flush completes as the time of record completion for all uncommitted records. This calculation may be dominated by the commit interval for fast sink puts, but is the closest we can approximate until the Connector adopts use of the SinkRecordReporter.

Rejected Alternatives

The author considered exposing the Kafka metrics and sensors libraries to the task contexts to allow each connector to define relevant metrics, but opted instead to standardize the latency metrics so they are common across all Connector implementations. This prevents there being a separate metric name for effectively the same latency measurement for every connector as they all implement the metric separately.