KIP-762: Delete Committed Connect Records

Status

Current state: *Under Discussion*

Discussion thread:

JIRA:

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

Motivation

While Kafka Connect's primary use-case is probably to move data in and out of Kafka, it is also useful for building pipelines where Kafka is just incidental. We can move data between Sources and Sinks leveraging Kafka's durability, even when the resulting Kafka topics are not otherwise used. In such scenarios, it doesn't make sense to keep data around in Kafka longer than necessary for the pipeline.

In order to reduce the storage space required for these pipelines, we could use a shorter retention period. However, there is no way to guarantee that SinkConnectors are able to keep up with the retention period. This yields a problematic and unnecessary trade-off between storage space and durability.

Instead, the Connect runtime could automatically delete records it knows it has already processed. This would be challenging for individual SinkConnector implementations to handle, since multiple SinkConnectors could be reading the same topics, and we don't want one SinkConnector to truncate a topic that another SinkConnector is still reading. Since the Connect runtime knows the state of all Connectors, it is safe for the runtime to delete records once they are read by all SinkConnectors.

Public Interfaces

We'll add a pair of new Worker configuration properties:

- delete.committed.enabled = <boolean>: must be true to enable this feature. Default: false.
- delete.committed.topics = <topic pattern list>: topics to automatically truncate. Default: empty list (i.e. no topics matched)

Proposed Changes

When delete.committed is enabled, each Worker will asynchronously truncate topic-partitions after they are flushed. To avoid deleting records prematurely, the following conditions must be met:

- The topic must be processed by some SinkConnector. This will trivially be the case for topic-partitions that are being flushed by a Worker.
- The topic name must be included in / matched by delete.committed.topics. This may be configured to ".*", in which case all topics are candidates to be automatically truncated.
- For a candidate topic to be truncated, the Connect worker must be the *only consumer group* subscribing to the topic. The Worker will query Kafka for all consumer groups and eliminate any topics that have other active consumer groups. This sanity check will be performed only once for each topic (when the worker first encounters the topic) unless "active topics" is reset (see KIP-558). The worker will *log a warning* anytime it discovers such topics and will refuse to truncate them. This helps prevent delete.committed from accidentally truncating topics that may have usage outside Connect, but is not a guarantee.
- There must be no outstanding record deletion requests for this Worker.

The frequency at which Workers truncate topics will be influenced primarily by offset.flush.interval, but deletion is not guaranteed to occur at this frequency.

Compatibility, Deprecation, and Migration Plan

This feature must be explicitly enabled.

Rejected Alternatives

- We could use a Connector-level configuration to control which topics are truncated for finer-grained control. However: 1) topics are truncated by Workers, not individual Connectors; 2) this feature is most useful for pipelines that involve multiple Connectors that process the same topics.

- We could eliminate the delete.committed.topics configuration for simplicity, but it is difficult to guarantee that a given topic is safe to truncate; thus, we let operators configure topics explicitly if they prefer. We recommend using some convention like ".*\.intermediate" or ACLs to prevent misuse of these intermediate topics.
- We could instead rely on rate limiting (see KIP-731) to keep our intermediate topics small. However, Connect rate limiting is meant to be for protecting external systems (sources/sinks), not Kafka or Connect.
- We could rely on broker-side quotas to keep our intermediate topics small. However, this would mean we'd need to continually update our quotas to maintain a target queue depth.
- We could add backpressure from Sinks to Sources. This is a common pattern in other systems. However, Connect doesn't necessarily colocate Sinks and Sources that are processing the same topic-partitions, so Workers would need to apply backpressure in a distributed manner. Moreover, such tight coupling between sources and sinks would be unusual in the Kafka ecosystem, and may confuse operators that observe Sources inexplicably lagging.
- We could tag individual records with metadata when they are produced by SourceConnectors, ensuring that we only delete records that have gone through a SourceSink pipeline. This would perhaps make sense if Kafka supported deleting individual records. However, since we can only truncate topic-partitions up to a given offset, we don't gain anything by tracking individual records.