KIP-298: Error Handling in Connect

Status

Current state: *Accepted*

Discussion thread: *here*

JIRA: *here*

Released: 2.0.0

The proposal discussed in this KIP is implemented in this pull request.

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

Motivation

There are several places in Connect during which failures may occur. Any failure to deserialize, convert, process, or read/write a record in Kafka Connect can cause a task to fail. Although some errors can be addressed with transformations or custom converters that check for malformed data, in general it is difficult to ensure correct and valid data or to tell Connect to skip problematic records.

Connect should allow users to configure how failures should be handled during all phases of processing records. Some failures such as the lack of availability of some external components may be resolved by simply retrying, while other errors should be recorded and the problem records simply skipped. Where possible, Connect should be able to record the error and optionally include the problematic records and configuration states of the connector, transform and converter. Since no single solution works for everyone, all of this error handling behavior should be configurable.

This proposal aims to change the Connect framework to allow it to automatically deal with errors while processing records in a Connector. By default, Connect will fail immediately when an error occurs, which is the previous Connect behavior. Therefore, all new behaviors must be explicitly enabled.

Public Interfaces

Several new behaviors for handling and reporting errors are introduced, and all must be configured in the individual connector configurations.

Retry on Failure

Connect will attempt to retry the failed operation for a configurable total duration, starting with a fixed duration (value of `300ms`) and with exponential backoff between each retry.

Configuration Name	Description	Default Value	Domain
errors.retry.timeout	The total duration a failed operation will be retried for.	0	[-1, 0, 1, ... Long.MAX_VALUE], where -1 means infinite duration.
errors.retry.delay.max.ms	The maximum delay between two consecutive retries (in milliseconds). Jitter will be added to the delay once this limit is reached to prevent any thundering herd issues.	60000	[1, ... Long.MAX_VALUE]

Task Tolerance Limits

Tolerate up to a configurable number of errors in a task. A failed operation is declared to be an error only if Connect has exhausted all retry options. If the task fails to successfully perform an operation on a record within tolerance limit, the record is skipped. Once the tolerance limit is reached, the task will fail. Tolerance limits can be configured using the following new properties:

Config Option	Description	Default Value	Domain

errors.tolerance	Fail the task if we exceed specified number of errors in the observed duration.	none	[none, all].

Log Error Context

The error context and processing information can be logged along with the standard application logs using the following configuration properties:

Config Option	Description	Default Value	Domain
errors.log.enable	Log the error context along with the other application logs. This context includes details about the failed operation, and the record which caused the failure.	false	Boolean
errors.log.include.messages	Whether to include the Connect Record in every log. This is useful if users do not want records to be written to log files because they contain sensitive information, or are simply very large. If this property is disabled, Connect will still log some minimal information about the record (for example, the source partition and offset if it is a SourceRecord, and Kafka topic and offset if it is a SinkRecord).	false	Boolean

Dead Letter Queue (for Sink Connectors only)

For sink connectors, we will write the original record (from the Kafka topic the sink connector is consuming from) that failed in the converter or transformation step into a configurable Kafka topic.

Config Option	Description	Default Value	Domain
errors.deadletterqueue.topic.name	The name of the dead letter queue topic. If not set, this feature will be disabled.	""	A valid Kafka topic name
errors.deadletterqueue.topic.replication.factor	Replication factor used to create the dead letter queue topic when it doesn't already exist.	3	[1 ... Short.MAX_VALUE]
errors.deadletterqueue.context.headers.enable	If true, multiple headers will be added to annotate the record with the error context	false	Boolean

If the property `errors.deadletterqueue.context.headers.enable` is set to **`true`**, the following headers will be added to the produced raw message (only if they don't already exist in the message). All values will be serialized as UTF-8 strings.

Header Name	Description
__connect.errors.topic	Name of the topic that contained the message.
__connect.errors.partition	The numeric ID of the partition in the original topic that contained the message (encoded as a UTF-8 string).
__connect.errors.offset	The numeric value of the message offset in the original topic (encoded as a UTF-8 string).
__connect.errors.connector.name	The name of the connector which encountered the error.
__connect.errors.task.id	The numeric ID of the task that encountered the error (encoded as a UTF-8 string).
__connect.errors.stage	The name of the stage where the error occurred.
__connect.errors.class.name	The fully qualified name of the class that caused the error.
__connect.errors.exception.class.name	The fully qualified classname of the exception that was thrown during the execution.
__connect.errors.exception.message	The message in the exception.
__connect.errors.exception.stacktrace	The stacktrace of the exception.

Metrics

The following new metrics will monitor the number of failures, and the behavior of the response handler. Specifically, the following set of counters:

- for counting the failures at various stages.
- for how the framework handled this error.
    - number of messages retried
    - number of records skipped
    - number of messages logged to file or the dead letter queue

MBean name: `kafka.connect:type=task-error-metrics,connector=([-.\w]+),task=([-.\w]+)`

Metric/Attribute Name	Description	Implemented
total-record-failures	Total number of failures seen by this task.	2.0.0

total-record-errors	Total number of errors seen by this task.	2.0.0
total-records-skipped	Total number of records skipped by this task.	2.0.0
total-retries	Total number of retries made by this task.	2.0.0
total-errors-logged	The number of messages that was logged into either the dead letter queue or with Log4j.	2.0.0
deadletterqueue-produce-requests	Number of produce requests to the dead letter queue.	2.0.0
deadletterqueue-produce-failures	Number of records which failed to produce correctly to the dead letter queue.	2.0.0
last-error-timestamp	The timestamp when the last error occurred in this task.	2.0.0

Proposed Changes

A connector consists of multiple stages. For source connectors, Connect retrieves the records from the connector, applies zero or more transformations, uses the converters to serialize each record's key, value, and headers, and finally writes each record to Kafka. For sink connectors, Connect reads the topic(s), uses the converters to deserialize each record's key, value, and headers, and for each record applies zero or more transformations and delivers the records to the sink connector. In this proposal, we will specifically deal with the following failure scenarios which can occur during these stages:

Operation	Will Retry?	Tolerated Exceptions
Transformation	only on org.apache.kafka.connect.errors.RetriableException	java.lang.Exception
Key, Value and Header Converter	only on org.apache.kafka.connect.errors.RetriableException	java.lang.Exception
Kafka Produce and Consume	only on org.apache.kafka.common.errors.RetriableException	only on org.apache.kafka.connect.errors.RetriableException, fail task otherwise.
put() in SinkTask and poll() in SourceTask	only on org.apache.kafka.connect.errors.RetriableException	only on org.apache.kafka.connect.errors.RetriableException, fail task otherwise.

There are two behavioral changes introduced by this KIP. First, a failure in any stage will be reattempted, thereby "blocking" the connector. This helps in situations where time is needed to manually update an external system, such as manually correcting a schema in the Schema Registry. More complex causes, such as requiring code changes or corruptions in the data that can't be fixed externally, will require the worker to be stopped, data to be fixed and then the Connector to be restarted. In the case of data corruption, the topic might need to be cleaned up too. If the retry limit for a failure is reached, then the tolerance limit is used to determine if this record should be skipped, or if the task is to be killed. The second behavioral change is introduced in how we report these failures. Currently, only the exception which kills the task is written with the application logs. With the additions presented in this KIP, we are logging details about the failed operation along with the bad record. We are also introducing an option to write bad records into a dead letter queue for Sink Connectors. This would write the original key, value and headers of failed records into a configured Kafka topic.

Example 1: Fail Fast

To maintain backward compatibility, by default a Connector task will fail immediately upon an error or exception. This reflects the same behavior as earlier releases, ensuring that existing installations work the same way. Although it is not necessary to add extra configuration properties, the following properties may be added to a connector configuration to achieve this older behavior:

```
# disable retries on failure
errors.retry.timeout=0

# do not log the error and their contexts
errors.log.enable=false

# do not record errors in a dead letter queue topic
errors.deadletterqueue.topic.name=

# Fail on first error
errors.tolerance=none
```

Example 2: Record and Skip

The following configuration shows how to setup error handling with multiple retries, logging both to the application logs and a Kafka topic with infinite tolerance:

```
# retry for at most 10 minutes times waiting up to 30 seconds between consecutive failures
errors.retry.timeout=600000
errors.retry.delay.max.ms=30000

# log error context along with application logs, but do not include configs and messages
errors.log.enable=true
errors.log.include.messages=false

# produce error context into the Kafka topic
errors.deadletterqueue.topic.name=my-connector-errors

# Tolerate all errors.
errors.tolerance=all
```

Compatibility, Deprecation, and Migration Plan

The current behavior in Connect is to kill the task on the first error in any stage. As mentioned above, this will remain the default behavior if connector configurations are not changed.

Rejected Alternatives

Correcting records in the handler: the handler will not be responsible for providing corrected records. For sink records, the user can correct records and write the corrected records back to the origin Kafka topics using the dead letter queue mentioned above. For source records, the user can analyze the error messages and fix the data at the source.

Allow per-stage error handler: This would have provided finer grained error handling. But comes at the expense of more configuration, and users having to ensure that the different stages are using compatible error handling. It is also not evident that this is more useful than simply taking the most flexible of handlers and applying it across all stages.

Interceptors for Erroneous Records: Similar to ProducerInterceptor and ConsumerInterceptor, we could potentially add ErrorInterceptors too. But given that the handlers subsumes most of the functionalities here, we decided to not provide this feature.

Defining and configuring properties in the worker config: Firstly, it is out of place to specify error handling properties in a worker config when it will never be used by a worker (since all the failures are handled at the Connector level). Secondly, adding inheritance in configurations adds a level of complexity which can be avoided at this stage of development.

Write records that fail in the put() step of a sink connector to the dead letter queue: since sink connectors can chose to batch records in a put() method, it is not clear what errors are caused by what records (they might be because of records that were immediately written to put(), or by some previous records that were processed later). Also, there might be connection issues that are not handled by the connector, and simply bubbled up as IOException (for example). Effectively, errors sent back to the framework from the put() method currently do not have sufficient context to determine the problematic records (if any). Addressing these issues would need a separate KIP.