KIP-161: streams deserialization exception handlers

Status

Current state: *Adopted (1.0.0)*

Discussion thread: *here*

JIRA: *here*

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

Motivation

When processing billions of records a day, it is likely that some of these records may be corrupt or that the serialization logic may be incorrect and buggy or it cannot handle all record types. We call such records “poison pills” in this KIP. In this cases, users might want some options with how to handle poison pills. Currently either the entire streams pipeline is closed upon such a record, or the user has to do lots of extra work with operations like flatMap, branch etc to circumvent the exceptions.

Proposed Changes

We propose to give users more control on how to handle poison pills as well as good default options they can just pick from. In a nutshell, this proposal calls for skipping over all bad records and calling an exception handler for each of them. The handler may in turn decide to just log the error, or fail the pipeline. The Rejected Alternatives contains more complex handling options that can be done at a latter stage. Note that all processing nodes are linked to just one handler.

The basic interface added will be:

```
/**
 * An interface that allows user code to inspect a record that has failed deserialization
 */
public interface DeserializationExceptionHandler extends Configurable {
 /**
  * Inspect a record and the exception received
  * @param context processor context
  * @param record record that failed deserialization
  * @param exception the actual exception
  */
 DeserializationHandlerResponse handle(ProcessorContext context, ConsumerRecord<byte[], byte[]> record,
Exception exception);
}

public enum DeserializationHandlerResponse {
        /* continue with processing */
        CONTINUE(1),
        /* fail the processing and stop */
        FAIL(2);
}
```

Users will be able to set an exception handler through the following new config option which points to a class name.

```
public static final String DEFAULT_DESERIALIZATION_EXCEPTION_HANDLER_CLASS_CONFIG = "default.deserialization.
exception.handler"
```

Two implementations of the interface will be provided. The default will be the LogAndFailExceptionHandler as seen below:

```
// logs the error and returns CONTINUE
public class LogAndContinueExceptionHandler implements RecordExceptionHandler {...}

// logs the error and returns FAIL
public class LogAndFailExceptionHandler implements RecordExceptionHandler {...}
// Then in StreamsConfig.java:

.define(DEFAULT_DESERIALIZATION_EXCEPTION_HANDLER_CLASS_CONFIG,
        Type.CLASS,
        LogAndFailExceptionHandler.class.getName(),
        Importance.MEDIUM,
        DEFAULT_DESERIALIZATION_EXCEPTION_HANDLER_CLASS_DOC)
```

In addition, we also added a metric that keeps track of the number of records skipped per source node.

Compatibility, Deprecation, and Migration Plan

- *The way poison pills are handled today is that the uncaught exception handler eventually catches them and the streams pipeline will shut down. We can choose to maintain that default method for backwards compatibility using HandlerResponse.FAIL by default.*

Rejected Alternatives

- **Allow retries**. Basically for each record exception, allow the processing logic to retry it. In this case RecordExceptionHandler.handle() would return not void, but an enum with RETRY as an option. This options leads to quite a bit of complexity since each processing node needs to implement a custom retry logic. Furthermore we don't have enough evidence to know failure types for which retrying would solve the issue. We might want to do this once we have more evidence that it's useful.
- **Allow record substitution**. If a record is corrupt, why not have the exception handler inject a good record to substitute the corrupt one? There are two options here. The record could be injected before the processing logic: 1) process->exception->inject record->reprocess new injected record->forward.... Alternatively the record could be injected after the processing logic: 2) process->exception->inject record and forward it to next processing node.

  Again this is theoretically viable but if it's not clear if it's useful. We can consider adding such logic at a later stage.
- **Allow per-node handlers**. Note that the proposal above uses one exception handler. We could allow for each node to have it's own handler (e.g., map has a handler, a reduce has a different handler). It is not clear if the complexity is worth it. We can consider this in a later stage if there are sufficient use cases.
- **Allow dead letter queue**. We could send each record to a special topic. Note that this will still be possible with this KIP, but we won't provide such functionality in the library for now and just stick with the basic building blocks.