KIP-990: Capability to PAUSE Tasks on DeserializationException

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

Presently, Kafka Streams provides users with two options for handling a `DeserializationException` via the `DeserializationExceptionHandler` interface:

1. `FAIL` - throw an Exception that causes the stream thread to fail. This will either cause the whole application instance to exit, or the stream thread will be replaced and restarted. Either way, the failed `Task` will end up being resumed, either by the current instance or after being rebalanced to another, causing a cascading failure until a user intervenes to address the problem.
2. `CONTINUE` - discard the record and continue processing with the next record. This can cause data loss if the record triggering the `DeserializationException` should be considered a valid record. This can happen if an upstream producer changes the record schema in a way that is incompatible with the streams application, or if there is a bug in the `Deserializer` (for example, failing to handle a valid edge-case).

The user can currently choose between data loss, or a cascading failure that usually causes all processing to slowly grind to a halt.

This KIP introduces a third option: `PAUSE`.

Public Interfaces

Modified Interfaces

org.apache.kafka.streams.errors.DeserializationExceptionHandler.DeserializationHandlerResponse

```
        /* pause processing the current Task, but continue other Tasks */
        PAUSE(2, "PAUSE");
```

New Interfaces

org.apache.kafka.streams.errors.LogAndPauseExceptionHandler

```
public class LogAndPauseExceptionHandler implements DeserializationExceptionHandler {
    private static final Logger log = LoggerFactory.getLogger(LogAndPauseExceptionHandler.class);

    @Override
    public DeserializationHandlerResponse handle(final ProcessorContext context,
                                                 final ConsumerRecord<byte[], byte[]> record,
                                                 final Exception exception) {

        log.error("Exception caught during Deserialization, " +
                        "taskId: {}, topic: {}, partition: {}, offset: {}",
                context.taskId(), record.topic(), record.partition(), record.offset(),
                exception);

        return DeserializationHandlerResponse.PAUSE;
    }

    @Override
    public void configure(final Map<String, ?> configs) {
        // ignore
    }
}
```

org.apache.kafka.streams.KafkaStreams

```
/**
 * Resume processing the {@link Task} specified by its {@link TaskId id}.
 * <p>
 * This method resumes a {@link Task} that was {@link Task.State.PAUSED} due to an {@link
 * DeserializationExceptionHandler.DeserializationHandlerResponse.PAUSE error}.
 * <p>
 * If the given {@link Task} is not {@link Task.State.PAUSED}, no action will be taken and
 * {@code false} will be returned.
 * <p>
 * Otherwise, this method will attempt to transition the {@link Task} to {@link Task.State.RUNNING},
 * and return {@code true}, if successful.
 *
 * @return {@code true} if the {@link Task} was {@link Task.State.PAUSED} and was successfully
 *         transitioned to {@link Task.State.RUNNING}, otherwise {@code false}.
 */
public boolean resume(final TaskId task);

/**
 * Gets all of the currently paused Tasks running on threads of this Kafka Streams instance.
 * <p>
 * @return A {@link Set} containing the metadata for Tasks that are assigned to this Kafka
 *         Streams instance, but are currently paused.
 */
public Set<TaskMetadata> pausedLocalTasks() {
    return metadataForLocalThreads().stream()
        .flatMap(thread -> Stream.concat(thread.activeTasks().stream(), thread.standbyTasks().stream()))
        .filter(task -> !task.isRunning())
        .collect(Collectors.toSet());
}
```

org.apache.kafka.streams.processor.TaskMetadata

```
/**
 * Determines if this Task is running or paused.
 * <p>
 * @return {@code true} if this Task is running; {@code false} if it is paused.
 */
public boolean isRunning();
```

Proposed Changes

`DeserializationHandlerResponse.PAUSE` pauses the `Task` that has encountered the error, but continues to process other Tasks normally. When a `Task` is `PAUSED`, it is still assigned as an active `Task` to the instance, but it will not consume or process any records.

Users could observe these errors through their usual observability solutions, by looking for:

1. The ERROR log message accompanying a `DeserializationException`.
2. The consumer failing to consume the subset of partitions that are affected by the error; usually via a "consumer lag" metric.

Once detected, users may intervene by, for example:

1. If the record should be valid: fixing the bug, in the application that causes the record to fail to deserialize. Once the bug has been fixed, the user would shutdown the application, deploy a fixed build and restart it. Once restarted, any `PAUSED` Tasks would automatically start running again from the record that originally produced the error.
2. If the record is invalid (e.g. corrupt data): advancing the consumer offsets, either via an external tool, or by a user-supplied application API. Once the offsets have been advanced, the user could either restart their application instance, or use the `KafkaStreams#reume(TaskId)` API to resume the `PAUSED` Task, if they wish to minimize downtime.

Implementation details

When a `DeserializationExceptionHandler` returns `PAUSE`, the current `Task` will be paused via `InternalProcessorContext`. However, if the Task is a `TaskType.GLOBAL` Task, it will automatically upgrade the response to `FAIL`, as the GlobalTask cannot be PAUSED; pausing the global Task without also pausing all other Tasks on the instance would cause them to work with stale data if they read from or join against any global tables.

When the Task is PAUSED, we will ensure that the offset of the last successfully processed record(s) on that Task are committed. This ensures that:

1. If the user fixes a bug and restarts the application, it will continue from the record that failed, and will not re-process a previously successfully processed record.
2. If the user wants to advance the consumer offsets past the "bad" record, they can simply use: `kafka-consumer-groups --reset-offsets --topic <topic>:<partition> --shift-by 1` to skip the bad message before resuming the `Task`.

Compatibility, Deprecation, and Migration Plan

- Since this is new functionality, it should not modify the behaviour of the system unless the new `PAUSE` response is used in a `DeserializationExceptionHandler`.
- No APIs are deprecated or need migration.

Test Plan

- An integration test will verify that, when pausing a failed Task, the consumer offset of the last successfully processed record(s) are committed.
- A unit test suite will verify that the `LogAndPauseExceptionHandler` properly pauses the `StreamTask`.
    - A unit test will also verify that Global Tasks are never PAUSED.

Rejected Alternatives

No alternatives have been considered.