KIP-779: Allow Source Tasks to Handle Producer Exceptions

Status

Current state: Accepted

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

We have Source Connectors bridging in data from systems that we do not have explicit control over. Even with some defensive settings, it is possible we may receive a message that is too large/un-processable for our configured Connect Worker, Kafka Broker, and other eco system components. The current behavior kills the Connector when such an error arises.

KAFKA-8586 addressed the issue of silently failing to produce data by capturing the producer errors and failing the task.

We have no ability to handle these errors inside of the Connector.

We would like to stay alive and keep processing and log out the error/write metadata to a topic/etc... for other systems to pick up for human intervention.

There is extensive error handling/retry capability (KIP-298, KIP-458) but none of it applies to the step of the process if the actual write to Kafka fails.

Public Interfaces

SourceTask.java

```
    /**
     * <p>
     * Commit an individual {@link SourceRecord} when the callback from the producer client is received. This
method is
     * also called when a record is filtered by a transformation or when {@link ConnectorConfig} "errors.
tolerance" is set to "all"
     * and thus will never be ACK'd by a broker.
     * In both cases {@code metadata} will be null.
     * </p>
     * <p>
     * SourceTasks are not required to implement this functionality; Kafka Connect will record offsets
     * automatically. This hook is provided for systems that also need to store offsets internally
     * in their own system.
     * </p>
     * <p>
     * The default implementation just calls {@link #commitRecord(SourceRecord)}, which is a nop by default. It
is
     * not necessary to implement both methods.
     * </p>
     *
     * @param record {@link SourceRecord} that was successfully sent via the producer, filtered by a
transformation, or dropped on producer exception
     * @param metadata {@link RecordMetadata} record metadata returned from the broker, or null if the record
was filtered or if producer exceptions are ignored
     * @throws InterruptedException
     */
    public void commitRecord(SourceRecord record, RecordMetadata metadata)
            throws InterruptedException {
        // by default, just call other method for backwards compatibility
        commitRecord(record);
    }
```

Other Changes

The behavior of errors.tolerance will change with this update! Current connectors should expect to be able to recover from producer exceptions if they are set to all. See compatibility and migration section below.

WorkerSourceTask.java

```
                    producer.send(
                producerRecord,
                (recordMetadata, e) -> {
                    if (e != null) {
                        log.error("{} failed to send record to {}: ", WorkerSourceTask.this, topic, e);
                        log.trace("{} Failed record: {}", WorkerSourceTask.this, preTransformRecord);
                        if (retryWithToleranceOperator.getErrorToleranceType().equals(ToleranceType.ALL)) {
                             commitTaskRecord(preTransformRecord, null);
                        }
                        else {
                            producerSendException.compareAndSet(null, e);
                        }
```

RetryWithToleranceOperator.java

```
    // For source connectors that want to skip kafka producer errors.
    // They cannot use withinToleranceLimits() as no failure may have actually occurred prior to the producer
failing
    // to write to kafka.
    public synchronized ToleranceType getErrorToleranceType() {
        return errorToleranceType;
    }
```

Proposed Changes

A getter method will be added to RetryWithToleranceOperator that the WorkerSourceTask will check before allowing producer write errors to kafka to be skipped and acked by the source connector.

On kafka producer failure, WorkerSourceTask will check

```
retryWithToleranceOperator.getErrorToleranceType().equals(ToleranceType.ALL)
```

to see if it should “ignore” this exception, call commitRecord(SourceRecord, RecordMetadata) to allow the connector to ack their source system, and continue processing. Otherwise it will set the producerSendException and the current behavior applies.

Setting "errors.tolerance" to "all" enables the functionality to skip records on producer write failure.

If exactly once semantics are enabled, the task will still be failed unconditionally.

Compatibility, Deprecation, and Migration Plan

Existing source connectors that do not set errors.tolerance to all will be unaffected. Default behavior is to still kill the task in the event of a producer write failure.

(Hypothetical use case): Existing source connectors that enable errors.tolerance all and expect to die on producer failure will need to update their connector to handle commitRecord with a null Metadata record. Not sure how much of these are in the wild, because it would require human intervention on the source system to fix whatever problem was causing the producer to die on kafka write in the first place.

Rejected Alternatives

Adding a new connect worker configuration item:

Existing configuration could be used and setting this at the connect worker level takes the choice out of the hands of the connector developers on how to handle kafka write errors.

`WorkerConfig.java`

```
    public static final String IGNORE_PRODUCER_EXCEPTIONS_CONFIG = "errors.producer.ignore";
    public static final boolean IGNORE_PRODUCER_EXCEPTIONS_DEFAULT = false;
    private static final String IGNORE_PRODUCER_EXCEPTIONS_DOC = "If true, the connector will ignore producer
exceptions. "
            + "The SourceRecord that failed to be written will be handed to commitRecord for handling by the
connector.";

...
                .define(IGNORE_PRODUCER_EXCEPTIONS_CONFIG, Type.BOOLEAN, IGNORE_PRODUCER_EXCEPTIONS_DEFAULT,
                        Importance.MEDIUM, IGNORE_PRODUCER_EXCEPTIONS_DOC)
...
```

SourceTask.java

```
    /**
     * <p>
     * Commit an individual {@link SourceRecord} when the callback from the producer client is received. This
method is
     * also called when a record is filtered by a transformation, and thus will never be ACK'd by a broker. It
is also
     * called when {@link WorkerConfig} "errors.producer.ignore" is set to true.
     * In both cases {@code metadata} will be null.
     * </p>
     * <p>
     * SourceTasks are not required to implement this functionality; Kafka Connect will record offsets
     * automatically. This hook is provided for systems that also need to store offsets internally
     * in their own system.
     * </p>
     * <p>
     * The default implementation just calls {@link #commitRecord(SourceRecord)}, which is a nop by default. It
is
     * not necessary to implement both methods.
     * </p>
     *
     * @param record {@link SourceRecord} that was successfully sent via the producer, filtered by a
transformation, or dropped on producer exception
     * @param metadata {@link RecordMetadata} record metadata returned from the broker, or null if the record
was filtered or if producer exceptions are ignored
     * @throws InterruptedException
     */
    public void commitRecord(SourceRecord record, RecordMetadata metadata)
            throws InterruptedException {
        // by default, just call other method for backwards compatibility
        commitRecord(record);
    }
```

WorkerSourceTask.java

```
                            producer.send(
                    producerRecord,
                    (recordMetadata, e) -> {
                        if (e != null) {
                            log.error("{} failed to send record to {}: ", WorkerSourceTask.this, topic, e);
                            log.trace("{} Failed record: {}", WorkerSourceTask.this, preTransformRecord);
                            if (workerConfig.getBoolean(WorkerConfig.IGNORE_PRODUCER_EXCEPTIONS_CONFIG)) {
                                commitTaskRecord(preTransformRecord, null);
                            }
                            else {
                                producerSendException.compareAndSet(null, e);
                            }
```

New interface method to SourceTask:

The original design has some pitfalls as outlined on the mailing list discussion above.

SourceTask.java:

```
   /**
     * <p>
     * This function gives SourceTasks an option to handle Producer Exceptions rather than dying.
     * See {@link Callback} for examples of exceptions.
     * </p>
     * <p>
     * SourceTasks are not required to implement this functionality; Kafka Connect will still kill the connector
     * by default if this function is not overridden by subclasses.
     * </p>
     * <p>
     * This function is executed in a different thread than poll().
     * </p>
     *
     * @param sourceRecord {@link SourceRecord} Pre transformation SourceRecord given to Kafka from the
Connector
     * @param producerRecord {@link ProducerRecord} Post transformation representation of the actual record
Kafka failed to write
     * @param e {@link Exception} exception that was thrown when the producer attempted to write the
ProducerRecord to Kafka
     * @return boolean whether or not Connect should ignore this exception
     */
    public boolean ignoreNonRetriableProducerException(SourceRecord sourceRecord,
                                                       ProducerRecord<byte[], byte[]> producerRecord, Exception
e) {
        return false;
    }
```

WorkerSourceTask.java:

```
        ...
                producer.send(
                    producerRecord,
                    (recordMetadata, e) -> {
                        if (e != null) {
                            log.error("{} failed to send record to {}: ", WorkerSourceTask.this, topic, e);
                            log.trace("{} Failed record: {}", WorkerSourceTask.this, preTransformRecord);
                            if (!task.ignoreNonRetriableProducerException(preTransformRecord, producerRecord,
e)) {
                                producerSendException.compareAndSet(null, e);
                            }
                        } else {
        ...
```

Passing only the SourceRecord back to the connector:

It may be beneficial for some users to have the byte[] ProducerRecord Kafka tried to send.

The connector could get access to the worker properties and do some validation:

> The message could then be intercepted before a write is attempted to Kafka. This is a lot of a priori knowledge to configure in advance and does not allow any leeway if an edge case is missed.

Keep current behavior:

> There are cases where we want to note somewhere in Kafka that we failed to ingest a record. Writing a DLQ back to the source system may not be advisable or possible. In the particular case of a record that is too large, we have no means to sample some metadata from the SourceRecord or even throw it away with some log messages.