KIP-89: Allow sink connectors to decouple flush and offset commit

Status

Current state: *Accepted*

Discussion thread: *here*

JIRA: *KAFKA-4161*

Released: 0.10.2.0

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

Motivation

This KIP is with regard to sink connectors and the offset commit process managed by the Connect runtime.

Periodic offset commits (controlled with `offset.flush.interval.ms`) require knowing what offset state is safe to commit to ensure at-least-once delivery from Kafka to the sink connector.

With the current API guarantees that are available, after `SinkTask.flush()` and before any further `put()` calls the current offset state is safe to commit. So this is what the runtime relies upon, and connectors have an expectation of periodic calls to `SinkTask.flush()`.

However, it is not ideal to couple offset commits and flushes:

- `SinkTasks` have the most knowledge about their buffer state, so giving them more control over when they should flush rather than imposing it via the API makes sense.
  - Some may not buffer at all and flush to the destination system all the records provided to `put()`.
  - Some may buffer time-based, whether via the current periodic flushes, or a background thread if they need more control over it.
  - Some may buffer size-based and write records to the destination system / make them visible when temporary output or in-memory data structure reaches a certain size.
  - Many will want a combination of time and size-based - size as hard-limit and time for liveness.
- Flushing buffers unnecessarily when one of the desired conditions has not been met should be rare, as it can hurt throughput.

There are some additional use-cases that this proposal aims to address:

- Some connectors that store offset state in the destination system, may not wish for Connect to manage offset commits at all as it is wasted effort.
- To minimize the window of redelivery upon failure, commits should ideally follow soon after a flush by the sink. So it would be useful to have a mechanism for the connector to explicitly request an offset commit.

Public Interfaces and Proposed Changes

New method on **SinkTask** abstract base class:

SinkTask.java

```
/**
 * Pre-commit hook invoked prior to an offset commit.
 *
 * The default implementation simply invokes {@link #flush(Map)} and is thus able to assume all {@code
currentOffsets} are committable.
 *
 * @param currentOffsets the current offset state as of the last call to {@link #put(Collection)}},
 *                        provided for convenience but could also be determined by tracking all offsets included
in the {@link SinkRecord}s
 *                        passed to {@link #put}.
 *
 * @return an empty map if Connect-managed offset commits are not desired, otherwise a map of committable
offsets by topic-partition.
 */
public Map<TopicPartition, OffsetAndMetadata> preCommit(Map<TopicPartition, OffsetAndMetadata> currentOffsets) {
    flush(currentOffsets);
    return currentOffsets;
}
```

New method on **SinkTaskContext** interface:

SinkTaskContext.java

```
/**
 * Request an offset commit. Sink tasks can use this to minimize the potential for redelivery
 * by requesting an offset commit as soon as they flush data to the destination system.
 *
 * This is a hint to the runtime and no timing guarantee should be assumed.
 */
void requestCommit();
```

Semantics

- When checking whether an offset commit is 'due' as per `offset.flush.interval.ms`, any pending commit request from the connector via `SinkTaskContext.requestCommit()` is also taken into consideration. Starting a commit clears any such pending request.
- Instead of invoking `SinkTask.flush()` as part of the offset commit process, `SinkTask.preCommit()` is invoked and the returned offset state committed.

The motivating use-cases are met as follows:

- Connectors that need to keep relying on periodic flushes don't need to do anything as `preCommit()` will invoke `flush()` by default, however connectors that would like to flush data based on custom policies like number of records or serialized size can do so by overriding `preCommit()`. They would need to maintain 'committable' offset state in a `Map` internally that they can return from `preCommit()`. They are free to optionally rely on the periodic calls to `preCommit()` for flushing data when e.g. a liveness timeout is hit.
- Connectors that benefit from disabling Connect-managed offset commits can override `preCommit()` and return an empty map.
- Connectors that want to minimize redelivery after failures can `requestCommit()` after a flush.

Compatibility, Deprecation, and Migration Plan

While `preCommit()` is a new method on the `SinkTask` API, it has a default implementation that calls `flush()`, so there are no compatibility issues.

There are no deprecations and no migration is needed, however some connectors can consider overriding `preCommit()` to avoid unnecessary flushes or disable offset commits.

Rejected Alternatives

- Deprecate `flush()` altogether: if we were starting from a clean slate, `flush()` is not strictly necessary. Connectors could choose to perform flushes from `SinkTask.preCommit()` and also upon `SinkTask.close()`.
    - However, `SinkTask.flush()` is now implemented by a number of connectors and can be expected to be invoked periodically as well as prior to `SinkTask.close()`.

- Avoid adding the new `preCommit()` method on the `SinkTask` API, but have `SinkTaskContext.requestCommit()` accept the offset state as an argument. Then could allow runtime to just use whatever offsets it has been given and skip `flush()` as long as the connector has been providing them.
    - Offset commits are asynchronous to `requestCommit()`, and it is desirable to use the most-recent committable offset state.
    - The implication of disabling periodic flushes based on offset commit requests from the connector is not very obvious.
    - Not clear how the use-case of disabling offset commits should be addressed.