KIP-910: Update Source offsets for Source Connectors without producing records

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

Source Connectors in Kafka Connect poll for data from sources like databases etc, convert them into a SourceRecord and push them to a Kafka topic. In this process, offsets are written periodically by the connect framework to an offsets topic (in distributed mode). These offsets are critical for source connectors since these act as checkpoints signifying the point from where to start on the source side when a connector is restarted. Offsets could be things like id from a specific database table or some identifier in the change logs of databases for CDC based connectors. Keeping this offsets updated is important for 2 reasons:

- It allows the number of events that need to be reprocessed on connector restarts to remain as minimal as possible.
- For CDC based connectors in particular, if the offsets are updated regularly then the database doesn't need to retain change logs for a higher duration. Because in the event that the offset(like BinlogPosition in a BinlogFile in case of MySQL or LSN in Postgres WAL) from which the connector wants to start doesn't exist anymore due to a shorter retention time of change logs on the database, then the connector can't proceed without dataloss or would need to start capturing from the beginning again.

The catch to all of this is that for the source connector offsets to be able to updated regularly, the connector should keep producing data regularly. However a lot of times, the source from which the change events are being captured mayn't be receiving changes regularly. Taking the example of databases again, if the database is being used for batch workloads where in a huge volume of data is ingested for 2-3 hours and then there's no activity for the rest of the day. Another possible scenario where this might be a problem is when there are a lot of updates in the source but the connector is tracking changes from a subset of the entities in the source which aren't getting a lot of updates. In such cases, if the offset gets purged from the source side then we could land into issues similar to point number 2 above.

Public Interfaces

`SourceTask` class would be enhanced to add a new method called `updateOffsets(Map<Map<String, Object>, Map<String, Object> offsets)`.

SourceTask#updateOffsets

```
/**
     * Hook to update the offsets for source partitions before offsets are committed. Source tasks can use this
     * hook to update the offsets for any source partition which isn't part of the offsets about to be
committed.
     * If any source partition is dropped, then it has no effect on the offsets committed. However, if the
offset
     * is set to a null value for any source partition, a tombstone record would be written to the offsets topic
     * for that source partition.
     * @param offsets the offsets that are about to be committed
     *
     * @return A map consisting of offsets that the task wants to update. The default implementation returns
{@code null}
     * which means there would not be any changes to the offsets about to be committed.
     * It is important to note that tasks shouldn't modify the original about to be committed offsets map
passed as an argument.
 */
     public Map<Map<String, Object>, Map<String, Object>> updateOffsets(Map<Map<String, Object>, Map<String,
Object>> offsets) {
        return null;
    }
```

Proposed Changes

This KIP aims to address the problem described in the motivation section by providing the connector with an option within the framework to be able to update offsets without it needing to necessarily produce actual source data. The aim of the KIP is to provide this capability in a way which is non-invasive to the connector's core functionality, backward compatible, easy to toggle on/off.

To achieve this, this KIP proposes to add a new method `updateOffsets(Map<Map<String, Object>, Map<String, Object> offsets)` in `SourceTask` which would provide an opportunity to the tasks to update their source offsets. The offsets passed to the `updateOffsets` method would be the offset from the latest source record amongst all source records per partition. This way, if the source offset for a given source partition is updated, that offset is the one that gets committed for the source partition. Note that the latest offsets are the ones that are about to be committed and not the latest offsets returned from `SourceTask::poll` so far. Tasks can return a map of offsets which it wants to update. By default this method would return null meaning no offsets would be updated. It is important to note that tasks shouldn't modify the original about to be committed offsets map passed as an argument.

With the offsets map that the task can return as an output, a few cases arise:

1. If a source partition is missing in the offsets map, the tasks can add that source partition along with it's offsets that the task thinks it should commit to the offsets in the output map.
2. It is also possible that a task might choose to send a tombstone record as an offset. In such cases, a tombstone record would be written to the offsets topic. A connector can use this mechanism to remove offsets for partitions which it knows won't need to be processed anymore. This logic would be very specific to the connector and is not a supplement to KIP-875: First-class offsets support in Kafka Connect which allows removal of offsets by Cluster Admins. However, it can be very useful in certain connectors (like File or Object based connectors when a file once processed won't need to be reprocessed and can be safely removed from offsets topic) and hence this feature is being added as part of this KIP.
3. If a source partition which was present in the original to be committed offsets map but not present in the returned map, it won't have any impact on the committed offsets i.e the offsets of the dropped source partition would be committed nonetheless.
4. If a task returns an empty map, the behaviour would would be disabled.

Invoking `updateOffsets`

Source connectors run in 2 modes: Exactly Once Support enabled and disabled. Based upon which mode is being run, the updateOffsets method would be invoked from different parts in the framework code. Without dwelling too much into the details, let's look at them one by one.

Exactly Once Support- Disabled

In this mode, the connector uses the `offset.flush.interval.ms` config to try committing offsets in a scheduled manner. Based on the changes proposed in this KIP, `updateOffsets` would be invoked with the to-be-committed-offsets.

One important aspect about the to-be-committed-offsets computation is that a source partition is eligible for offset commit only if one of the records belonging to it has been acknowledged by Kafka Producer. This acknowledgment is done if either the record failed to be sent or was sent successfully. But the point is, only source partitions having acknowledged messages are considered committable. The number of acknowledged messages are also tracked in a synchronised manner by the framework to avoid double ack-ing of messages. Keeping all these things in mind, the offsets returned by the `updateOffsets` method would be:

- Considered acknowledged automatically so that whenever offset flush is about to happen, these offsets would get committed.

Note that in this mode, `updateOffsets` would be invoked only if:

- The last poll invocation didn't return any records or
- All the records returned by the previous poll invocation got processed successfully
- First iteration of task.

Exactly Once Support- Enabled

EOS mode for source connectors has 3 different mechanisms for committing transactions defined by `transaction.boundary` config with possible values: (POLL, INTERVAL and CONNECTOR). These modes define when should a producer transaction be started and committed. What is relevant for this KIP is the fact that the offsets written to the offsets topic would also be done in the same transaction. Also, the offsets committed are written to a worker's global offsets topic using a non transactional producer and Worker principal.

What the above means is that, the invocation of the `updateOffsets` method should also happen in a way that the transactional boundaries are honoured. Since the offsets supplied should honour the transaction boundary, the `updateOffsets` method would be invoked just before an offset flush is attempted. The partition-offsets supplied to `updateOffsets` would still be the ones that are about to be committed and are unflushed but in the current transactional context. The updated offsets would be part of the flush that would take place later on during transaction commit.

Note that there is a very specific case when the connector is configured to use a connector specified transaction boundary and it is possible that we don't call SourceTask::updateOffsets until there are actual records that are also being returned through poll. In such cases, `updateOffsets` won't be invoked which would defeat the purpose of this KIP. In such cases, source tasks should occasionally request for a transaction commit via their transaction context if they want offsets to be committed without producing records.

Migration Plan and Compatibility

The changes are totally backward compatible since the changes are off by default. Only if a connector chooses to override the `updateOffsets` is when the connector becomes eligible for updating offsets. None of the existing connectors should be impacted even when this feature is available and deployed. Also, connectors which implement the new method will still be compatible with older Connect runtimes where the method will simply not be invoked.

Rejected Alternatives

- Unable to render Jira issues macro, execution error. has been open for a long time and discusses a lot of ideas. Some of the ideas are:
  - **a SourceRecord object with a null (or special) topic, or a null key and value**
    - Didn't want to rely on null semantics to avoid confusion.
    - Another topic adds to the operational burden.
  - **a SourceRecord object with a null key and value**
    - Same, didn't want to rely on null semantics.
  - **a subclass of SourceRecord that denotes this particular kind of request.**
    - This could have been passed as part of output of poll() method but as such SourceRecord don't have sub-classes and yet is specialised. Extending it doesn't seem the right thing to do in this case. This would have also needed to throw a `ClassNotFoundException` for older runtimes which is already being done but it won't be needed if the approach from the KIP is followed.
  - **Changing the signature of `poll()` to accept a `SourceRecordReceiver` parameter**
    - This breaks backward compatibility
  - **Creating a dedicated topic where records are being send over periodic time intervals.**
    - This was the original approach prescribed in the KIP but it was shelved because:
      - Another topic adds to the operational burden.
      - Since the mechanism was akin to heartbeat records, the general consensus was to have heartbeat mechanism by not needing to produce records explictly.
  - That ticket has a lot of comments/suggestions and some open PRs as well. This KIP chose to use the option which seemed the most non-invasive.