

[PUBLIC] Beam CDC Record Metadata

kenn@apache.org, ahmedabualsaud@apache.org, talat@apache.org

<https://s.apache.org/beam-cdc-metadata>

Overview

Beam supports change data capture (CDC) in a few different ways:

- DebeziumIO
- Spanner CDC
- IcebergIO unbounded read
- BigQuery CDC (via a Dataflow template)
- Bigtable change streams

However, there are some areas where we could improve:

- More IOs having full CDC write capability (e.g. upserts and deletions)
- Support for Beam SQL queries over CDC streams
- Creating changelog views from IO sources
- Optimization of CDC streams when upserts are possible

Proposal for Beam CDC

In summary:

- Add a "value kind" to the extended metadata associated with elements. This will be "insert", "delete", "update before", "update after"
- Add SDK and runner capability "VALUE_KIND" so we can coordinate SDKs that would be unaware of the metadata.
- Track metadata on PCollections whether non-insert ValueKind is used.
- Make all existing records automatically "insert".
- Propagate value kind automatically
- Adjust core ParDo to allow consuming ValueKind.
- Adjust core ParDo to allow producing any ValueKind.
- For now, reject aggregation over PCollections that have non-insert "ValueKind"
- Adjust key IOs to produce and consume the value kind.
 - DebeziumIO will need to unpack the Debezium records and produce insert/delete/update before/update after
 - IcebergIO will need to consume the ValueKind and perform the appropriate action on the Iceberg table

ValueKind Types

INSERT

Indicates a new record was **created** in the source system.

UPDATE_BEFORE

Indicates the state of a record immediately **before an update occurred**. This is typically used to identify the previous values of modified columns or to locate the record via its primary key.

Some sinks need this to correctly perform updates. Other sinks (that support UPSERT) can safely ignore this type of CDC record.

UPDATE_AFTER

Indicates the state of a record immediately **after an update occurred**. Represents the current, valid state of the record following the change.

DELETE

Indicates that an existing record was **removed** from the source system.

Registering CDC capability

- In `beam_runner_api.proto` we add the `VALUE_KIND` capability. We don't give it too broad a name, as it is an implementation detail that simply indicates whether a runner or SDK is capable of processing `VALUE_KIND` metadata. We need this so we don't send delete records to a party that will mistakenly interpret them as insert records, etc.
- We need to propagate the `Environment` and `ProvisionInfo` protos so that SDKs and runners correctly register their capabilities.

Encoding CDC Metadata

- **Proposed:** We put the `ValueKind` as an enum field in the new `ElementMetadata` protobuf. Even though this is heavier-weight than the required 4 bits, it is far smaller than a meaningful database `Row` or the overhead of a Debezium record.
- Alternatively: a nested message `CdcMetadata` that would be extensible to new CDC-related stuff. But this adds encoding overhead and code boilerplate and we can just inline any new piece of metadata into the main element metadata

Producing CDC Metadata in ParDo

Java

The user requests an `OutputReceiver`. This works the same for splittable DoFns for implementing CDC sources.

Java

```
class MyDoFn extends DoFn<InputType, OutputType> {
    @ProcessElement
    public void process(@Element InputType in, OutputReceiver<OutputType> out) {
        ...

        out.builder(value)
            .setValueKind(ValueKind.UPDATE_AFTER)
            .output();
    }
}
```

Python

TODO

Go

TODO

Consuming CDC Metadata in ParDo

Java

The user requests a `ValueKind` parameter

Java

```
class MyDoFn extends DoFn<InputType, OutputType> {
    @ProcessElement
    public void process(@Element InputType, ValueKind kind, OutputReceiver<OutputType> output) {
        ...

        // output normally to propagate the same ValueKind kind
        out.output(value);

        // or override and set your own ValueKind
        out.builder(value)
            .setValueKind(ValueKind.UPDATE_AFTER)
            .output();
    }
}
```

Note that if a transform is not sensitive to the `ValueKind`, for example if it is just a mapping that applies equally to all `ValueKind` then we can omit it and the transform will "just work". This is somewhat risky, so we may want to only apply this logic if the transform is declared `@Pure`.

Python
TODO

Go
TODO

Overview of other systems

Debezium

TODO (A debezium record has before and after records, plus more metadata.)

BigQuery

Beam has a BigQuery CDC sink. It has UPSERT and DELETE mutation types:

<https://github.com/apache/beam/blob/cbf57c4c608a6c900b38c22027b2fb6d5ff4959a/sdks/java/io/google-cloud-platform/src/main/java/org/apache/beam/sdk/io/gcp/bigquery/RowMutationInformation.java#L32>

This is an example of a sink that can ignore ValueKind.UPDATE_BEFORE, because it doesn't need to know the prior state of an update. The mapping can look like this:

ValueKind.INSERT	UPSERT
ValueKind.UPDATE_AFTER	UPSERT
ValueKind.DELETE	DELETE
ValueKind.UPDATE_BEFORE	ignored

Of course, just passing ValueKind with records is not enough to utilize the sink. Beam does not preserve ordering, so the source would need to provide sequence numbers for the sink to commit changes in the correct order.

Flink

Flink models CDC record metadata on rows only, via their `RowKind` type. It can be "insert", "delete", "update before", or "update after". A debezium-style before/after record is modeled as two rows in Flink with the types "update before" and "update after". When records arrive at a filter or grouping, the "update before" and "update after" may get separated and become "insert" and "delete" types. By analysis of the graph, some collections do not require the "update before" record and it can be omitted.

In Beam:

- Since all records already have metadata in their `WindowedValue` wrapper, we can support this model on all records, not just rows.

- I propose we still use "update before" and "update after" records and follow Flink's learnings

See Also

- <https://issues.apache.org/jira/browse/FLINK-6047> - Jira tracking Flink SQL support for RowKind / retractions (note sub-jiras)
 - <https://docs.google.com/document/d/18XIGPcfsGbnPSApRipJDLPg5IFNGTQjnz7emkVpZlkw/edit?tab=t.0#heading=h.8ushfd8olrj6>
 - <https://github.com/apache/flink/pull/3733/> first implementation of retractions
 - <https://github.com/apache/flink/pull/12152/files#diff-de1caf21f44566412fa88087502c14317c48e7c71a41b4463bfd928bc6f0c909R122> how Debezium is adapted to Flink rows (<https://issues.apache.org/jira/browse/FLINK-17149> / <https://cwiki.apache.org/confluence/pages/viewpage.action?pageId=147427289>)
 - <https://cwiki.apache.org/confluence/display/FLINK/FLIP-95%3A+New+TableSource+and+TableSink+interfaces> Flink table interface proposal, w/ how tables declare the set of RowKind they contain. (<https://issues.apache.org/jira/browse/FLINK-16987> / <https://github.com/apache/flink/pull/11646>)
 - <https://www.ververica.com/blog/flink-sql-secrets-mastering-the-art-of-changelog-event-out-of-orderness>
- <https://debezium.io/documentation/reference/stable/connectors/spanner.html> Debezium + Spanner
 - <https://www.youtube.com/watch?v=SP2Ufaa6UBk>
 - <https://github.com/GoogleCloudPlatform/DataflowTemplates/blob/HEAD/v2/googlecloud-to-googlecloud/src/main/java/com/google/cloud/teleport/v2/templates/SpannerChangeStreamsToPubSub.java>
 - <https://cloud.google.com/dataflow/docs/guides/templates/provided/cloud-spanner-change-streams-to-pubsub>
- DebeziumIO
 - <https://2021.beamsummit.org/sessions/real-time-change-data-debeziumio/>
 - <https://github.com/apache/beam/pull/13983> (original DebeziumIO CDC)
 - <https://github.com/apache/beam/pull/24629/files> (schema transform)