databricks

Streaming Data and CDF

LECTURE

Capturing Changed Data

In this lecture, we will review how Delta Change Data Feed captures, processes, and delivers streaming data changes compared to traditional CDC.

Streaming Data and Data Changes

Updates and Deletes in streaming data

- In Structured Streaming, a data stream is treated as a table that is being **continuously appended**. Structured Streaming expects to work with data sources that are **append only**.
- Changes in existing data (updates and deletes) breaks this expectation!
- We need a **deduplication logic** to identify updated and deleted records.

- **Note:** Delta transaction logs tracks files than rows. Updating a single row will point to a new version of the file.

Spark Structured Streaming is a great tool to implement PIIs controls by incrementally propagate updates/deletes operations across a number of tables in your pipeline to make sure users' private information is properly handled as a whole.

However, there are some limitations of structured streaming that need to be addressed.

In structured streaming, a data stream source is treated as a table that is being continuously appended, and is expected to work in data sources that are append only. Same thing with the streaming tables in Lakeflow Spark Declarative Pipelines. Changes in existing data such as updates and deletes break this exception.

We need to add deduplication logic to identify updated and deleted records. This is what the APPLY CHANGES INTO in Lakeflow Spark Declarative Pipelines helps us do much more easily and concisely than we had to do before when manually implement implementing that logic in structured streaming. That's what the previous course includes.

Solution 1: Ignore Changes

Prevent re-processing by ignoring deletes, updates and overwrites

Ignore Deletes

- Ignore transactions that delete data at partition boundaries
- No new data files are written with full partition removal
- `spark.readStream.format("delta")`
  `.option("ignoreDeletes", "true")`

Skip Change Commits

- Allows stream to be executed against Delta table with upstream changes
- Discard files changing operations completely
- Subsumes ignoreDeletes
- `spark.readStream.format("delta")`
  `.option("skipChangeCommits", "true")`

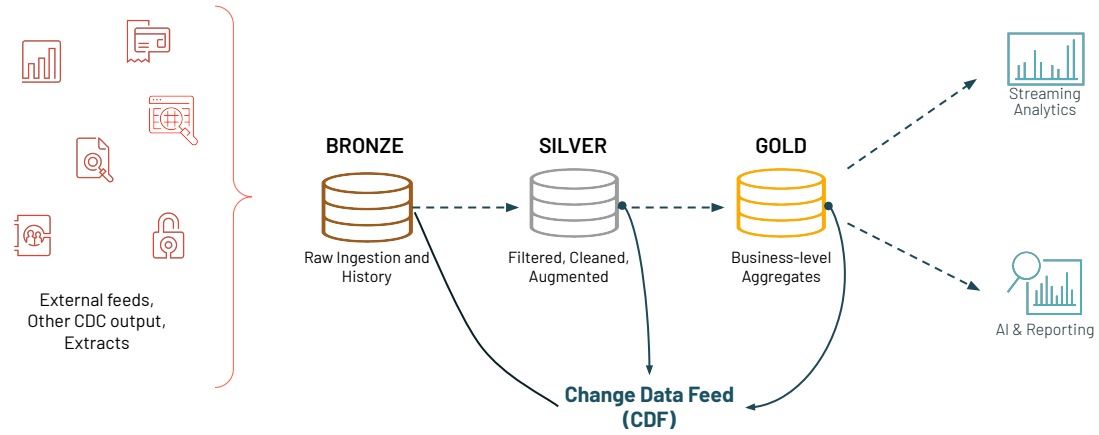
The first solution is to ignore deletes and changes; this keeps alignment with the append-only processing rule and simplifies stream processing. This can be achieved by leveraging the options:

- ignoreDeletes: that ignores transactions that delete data at partition boundaries.
- skipChangeCommits: Disregards file-changing operations entirely and only returns inserted rows, ignoring updates and deletes, and it subsumes ignoreDeletes, meaning it handles both deletions and updates to the source table
- Be aware that the option “ignorechanges,” is now being deprecated in favor of SkipChangeCommits.

A use case is when the focus is on processing new data additions, and separate logic can be implemented to handle changes if necessary.

Solution 2: Change Data Feeds (CDF)

Propage incremental changes to downstream tables



- The second option uses a Change Data Feed, or CDF, that allows tracking of row-level changes between versions of a delta table, including row data and metadata.
- To use CDF on the table level, you must manually enable it during the table creation or after it is created using the “ALTER Table” command.
- To take advantage of the Delta Change Data Feed, simply bring your external data sources to the Bronze layer and enable CDF from that point forward. This will allow you to use the Change Data Feed to move to the Silver or Gold layers or feed out to an external platform.
- Be aware that using CDF incurs some additional overhead for storing CDC-related metadata.

What Delta Change Data Feed Does for You

Benefits and use cases of CDF

Improve ETL Pipelines

Process less data during ETL to increase efficiency of your pipelines by processing only row-level changes

Unify batch and streaming

Common change format for batch and streaming updates, appends, and deletes

BI on your data lake

Incrementally update the data supporting your BI tool of choice

Meet regulatory needs

Full history available of changes made to the data, including deleted information

Silver & Gold Tables

- Using Delta Change Data Feed outputs for changes in the Silver and Gold layers ensure that all changes are reflected with substantially less processing cost

Materialized Views

- In many cases there's a need to capture an aggregated view of the gold level data for a dashboard or real-time application. Relying on the change data feed can eliminate the need for costly re-aggregations off of full tables while ensuring that changes are reflected appropriately.

Transmit Changes

- Outputting data from Delta to other systems can help solve specific needs and support other applications. For platforms that can ingest Change Data Output, this creates a means of incrementally updating databases, applications, and other systems with minimal overhead.

Audit Trail Table

- Compliance and audit typically need to be able to identify when, where, and how data has been changed. Change Data Feed outputs stored in a Delta table provide a quick, queryable means of finding exactly what has happened either to a specific record, to sets of data, or to the entire table.

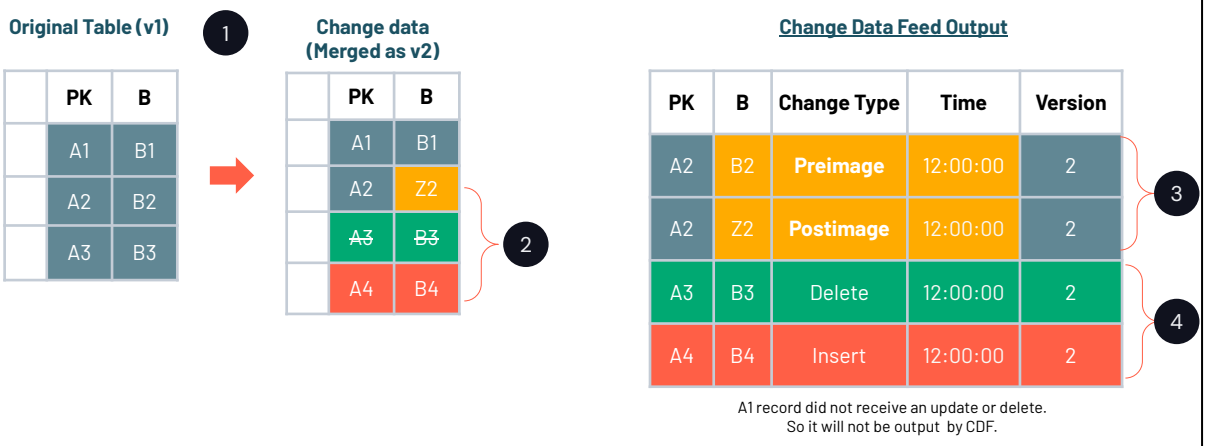
Comparison CDF vs CDC

Feature	Change Data Feed (CDF)	Change Data Capture (CDC)
Scope	Specific to **Delta Lake tables**	General concept applicable across systems
Functionality	Tracks row-level changes within Delta tables	Captures data changes for synchronization across systems
Implementation	Enabled on Delta tables; uses `**_change_data**` folder and `**table_changes**` function.	Implemented using Spark Declarative Pipelines and APIs like **APPLY CHANGES**
Efficiency	Processes only changed rows for operations.	Synchronizes incremental changes from source databases
Use Case	Tracking changes within Databricks	Capturing changes from external sources

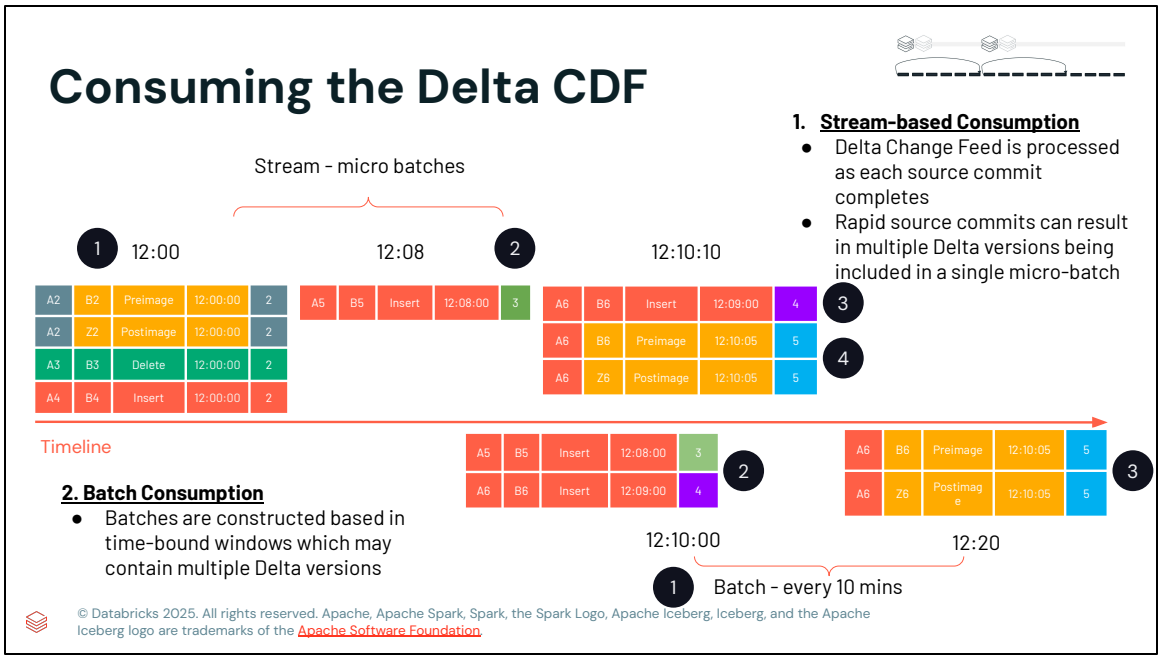
Primary comparison between these is the use of the Delta Lake Tables and the implementation using "_change_data" folder, and the changes can be queried using the function "table_changes" function, while CDC is implemented via the "Apply Changes" Syntax.

How Does Delta CDF Work?

Sample CDF data schema



1. We start with the original table, which has three records, A1 through A3, and two fields (PK and B)
2. We then receive an updated version of this table that shows a change to field B in record A2, the removal of record A3, and the addition of record A4. a. As this gets processed, the Delta Change Data Feed captures only records for which there was a change. This allows Delta Change Data Feed to speed up ETL pipelines as less data is being touched. b. Note that record A1 is not reflected in the Change Data Feed output, as no changes were made to that record.
3. In the case of updates, the output contains what the record looked like before the change, called the preimage, and what it held after the change, called the postimage. This can be particularly helpful when producing aggregated facts or materialized views, as appropriate updates could be made to individual records without reprocessing all of the data underlying the table. This allows changes to be reflected more quickly in the data used for BI and visualization.
4. For deletes and inserts, the affected record indicates whether it is being added or removed.
5. Additionally, the Delta version is noted to maintain logs of what happened to the data. This allows greater granularity for regulatory and audit purposes as needed. We also captured the timestamp of these commits and will show them in a few minutes.
6. While this shows a batch process, updates could also come from a stream, producing the same Change Data Feed output.

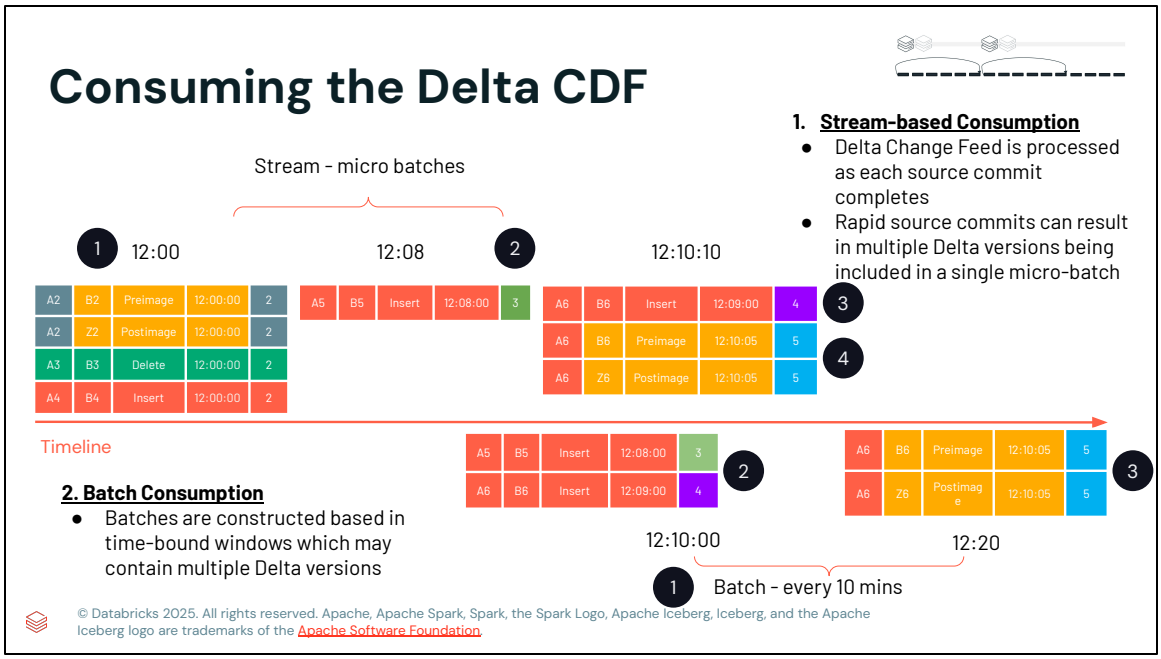


There are two ways to consume the Delta Change Data Feed in downstream processing - stream and batch.

Stream Mode.

In the stream mode scenario shown above the timeline, you use Delta structured streaming to process the Delta Change Feed as it comes in. This pattern allows micro-batches to be consumed based on the latest checkpoint without waiting for a predetermined time interval. The stream will simply process whatever has come in since the last checkpoint.

1. In the timeline shown here, let us assume that the large upsert from our previous example comes through at noon.
2. Our next insert (Delta version 3) commits at 12:08. The stream has already completed processing the first group and immediately picks up our 12:08 insert. It does not have to wait for a scheduled time to kick off.
3. We then received the insert for Delta version 4 at 12:09. The streaming process for the last commit has not yet been completed.
4. Delta version 5, an update to the same record inserted in Delta version 4, comes in at 12:10:05. As soon as the prior stream micro-batch completes, the next kicks off and picks up Delta versions 4 and 5.
5. As mentioned, this functions as a micro-batch. You now have both an insert and an update to a record in the same batch. You must add handling to choose the latest data to insert from this batch.



Batch Mode

For Batch mode, shown below the timeline. The Delta Change Feed is processed every X minutes together. You must then add logic to identify the current high watermark, the latest Delta version or timestamp previously processed, and pick up any changes from that point forward.

1. Following our previous example, let us assume that this batch process will run every 10 minutes. When it kicks off at 12:00, it processes the large upsert.
2. There is a pause until 12:10, when the process starts again, establishes that the high water mark is Delta version 2, and picks up everything since then. In this case, both version 3 and version 4 are included.
3. The batch then pauses again and does not pick up version 5 until 12:20, despite arriving shortly after the last batch was triggered.

CDF Configuration

Important notes for CDF configuration

- CDF is **not** enabled by default. It can be enabled;
  - At table level: `ALTER TABLE myDeltaTable SET TBLPROPERTIES (delta.enableChangeDataFeed = true)`
  - For all new tables: `set spark.databricks.delta.properties.defaults.enableChangeDataFeed = true;`
- Change feed can be read by;
  - Version
  - Timestamp

- Just to recap, CDF configuration is not enabled by default. If you want to apply it, allow it to use the delta.enableChangeDataFeed property in your table, whether from a altered table or by adding it into your table definition
- Also, you can change the data feed with the History version or be very specific using a timestamp.

Change Tables Function

Tracks row-level changes between versions of a Delta table

- It returns a log of changes to a Delta Lake table with Change Data Feed enabled, including inserts, updates, and deletes
- Leverages metadata columns:
  - ***_change_type***: Specifies the type of change (insert, delete, update_preimage, update_postimage
  - ***_commit_version***: The commit version associated with the change
  - ***_commit_timestamp***: The timestamp of the change
- Syntax:

```
table_changes ( table_str, start [, end ] )
```


As previously mentioned, there are two ways to collect the changes, and we'll see them both in the following demo:

1. The first option is reading the stream, which leverages the “Change_data” folder, located along with your data and metadata.
2. The second is for batch cases where you can query the “Table_Changes” function using the table name and specifying the “start” and “end” versions of the table's history.
3. Be aware that working with the Table Changes function will return the data in the specific version range, including three extra columns:
   a. Change_type to indicate what type of change it is, which could be insert, delete, update_preimage for the previous value, and update_postimage for the updated value.
   b. Commit_version provides the number of versions associated with the change
   c. Commit_timestamp that provides the specific time of the change.

Thank you for completing this lesson and continuing your journey to develop your skills with us.