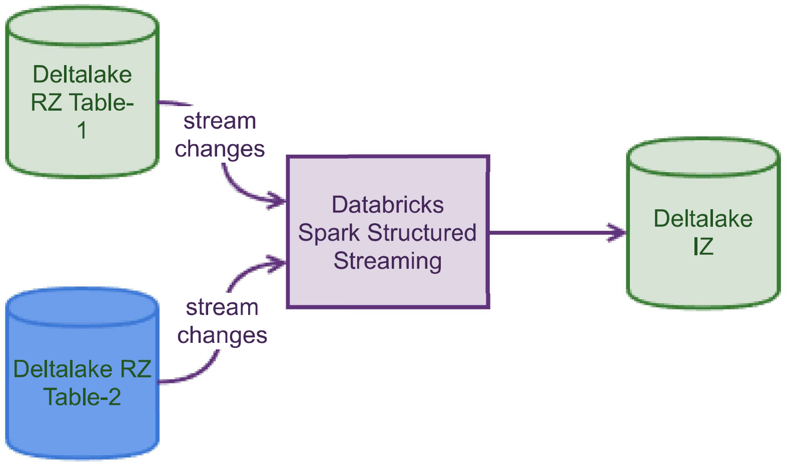
07. Setting up CDC on DeltaLake

Created by HariharaSubramanian, Narayanan, last modified on Dec 20, 2021

Introduction

DeltaLake can be used as a streaming source and sink. This will enable spark structured streaming applications to stream the changes from DeltaLake and write to it after processing.



One caveat with this is when using Delta as the streaming source, structured streaming does not handle updates to the existing records so if an existing record is updated due to an UPDATE or MERGE INTO operation, the streaming will fail. This effectively handles only inserts to the source delta tables and works in append-only mode.

To overcome this, in the latest version of **Databricks runtime 8.4 -** there is a way to configure CDC on the Deltalake tables that can streaming changes such as INSERT, UPDATE, DELETE.

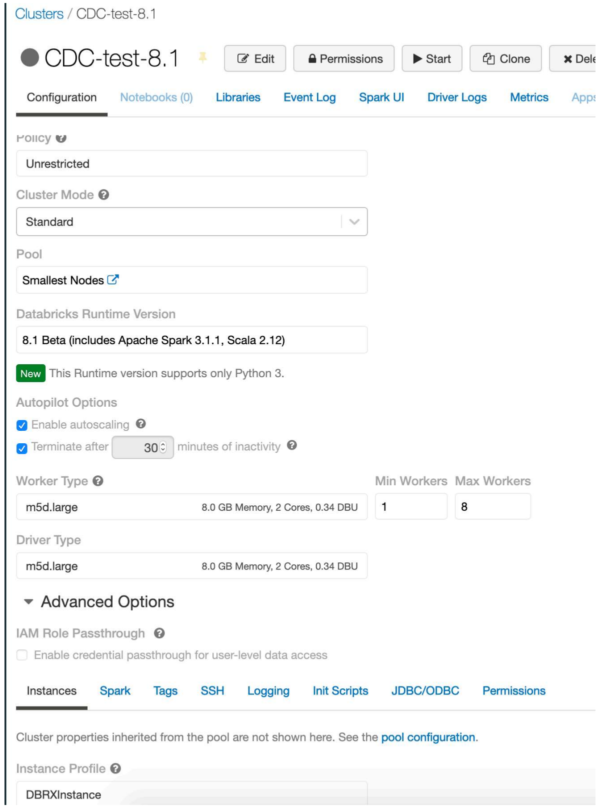
Change Data Feed on DeltaLake: https://docs.databricks.com/delta/delta-change-data-feed.html

Set up

Note that, CDC on Deltalake requires Databricks runtime 8.4 or more.

Create new interactive cluster (for testing purpose only) with 8.4 version with the right instance profile and spark config parameters.

For real-life workloads, use job clusters instead and they can be created with 8.4 runtime.



Create a delta table with CDC enabled

SQL

```
--To Create a new table with CDC enabled
CREATE TABLE student (
  id INT,
  name STRING, age INT
```

```
) USING DELTA
TBLPROPERTIES (delta.enableChangeDataFeed = true)

-- To modify an existing table to enable CDC
ALTER TABLE Customer SET TBLPROPERTIES (delta.enableChangeDataFeed = true);
```

Dataframe

dataframe

```
spark.conf.set('spark.databricks.delta.properties.defaults.enableChangeDataFeed',True)

df
  .write
  .format("delta")
  .saveAsTable("Customer")
```

Querying the change data

When CDC is enabled, the resultant dataframe will contain 2 additional columns __cdc_type and __log_version along with all the other additional columns on the original dataframe itself.

Column Name	Type	Possible values	Description
_change_type	String	insert, update_preimage, update_postimage or delete	update_preimage is the before change data and update_postimage is the after change data in the case of MERGE operation on the source Deltalake table
_commit_version	Long	Deltalake log version number on that table	Every commits creates a new version on delta and this columns denotes that.

SQL

Using Spark SQL to query data from Delta CDC with __log_versions playing a role.

sql

```
SELECT … FROM table_changes ('tableName', startingVersion)
-- version as ints or longs
SELECT … FROM table_changes ('tableName', startingVersion, endingVersion)
-- timestamp as string formatted timestamps
SELECT … FROM table_changes ('tableName', 'startingTimestamp', 'endingTimestamp')
-- database/schema names inside the string for table name, with backticks for escaping dots and special characters
SELECT … FROM table_changes ('dbName.`dotted.tableName`', 'startingTimestamp','endingTimestamp')

-- path based tables
SELECT … FROM table_changes_by_path ('\path', 'startingTimestamp', 'endingTimestamp')
```

Dataframe

sql

```
account_df = (spark.read
              .format("delta")
              .option("readChangeData", "true")
              .option("startingVersion", 1)
              .table("gwy1.account") # table is available on the hive metastore (glue catalog)
              .filter("_change_type IN ('INSERT', 'UPDATE_POSTIMAGE')")
             )
display(account_df)
```

Streaming using Dataframe

CDC Enabled delta tables can also be accessed using Spark Structured Streaming to get realtime changes that will be available to push to the downstream systems. For example - from Raw Zone to Integrated Zone.

streaming consumer

```
from pyspark.sql.functions import col, expr, max

account_df = (spark.readStream
```

```
              .format("delta")
              .option("readChangeData", "true")
              .option("startingVersion", 0)
              .table("gwy1.account")
              .filter("_change_type IN ('insert', 'update_postimage')")
             )
def batch_func(change_df, batch_id):
  count = (change_df
           .count()
          )
  print(f"Number of change records received={count}")

(account_df
   .writeStream
   .option("checkpointLocation", "s3://da-datastore-client.dev-cignasplithorizon/checkpoint/delta-join-test")
   .foreachBatch(batch_func)
   .trigger(once=True) # Trigger once is enabled but can be disabled for 24 x 7 streaming
   .start()
)
```

Limitations

Multiple stream aggregations

Spark structured streaming has limitations for consuming multiple datasets in a streaming manner with "aggregations running on each stream". This is not supported right now and will need some rethinking to implement if the usecase warrants it.

For example, the below process won't work because there are group-by happening on both the source streams to pick up the most recent change for every record if there are multiple updates to the source tables.

multiple stream aggregates

```
account_df = (spark.readStream
              .format("delta")
              .option("readChangeData", "true")
              .option("startingVersion", 1)
```

```
              .table("gwy1.account")
              .filter("_change_type IN ('insert', 'update_postimage')")
              .select("_commit_version", expr("struct(_commit_version as mx, *) as r"))
              .groupBy("_commit_version")
              .agg(max("r").alias("r"))
              .select("r.*")
              .drop("mx")
             )
client_df = (spark.readStream
              .format("delta")
              .option("readChangeData", "true")
              .option("startingVersion", 1)
              .table("gwy1.client")
              .filter("_change_type IN ('insert', 'update_postimage')")
              .select("_commit_version", expr("struct(_commit_version as mx, *) as r"))
              .groupBy("_commit_version")
              .agg(max("r").alias("r"))
              .select("r.*")
              .drop("mx")
             )
key_df = account_df.select(col("CLIENT_ID")).union(client_df.select(col("CLIENT_ID")))
def batch_func(change_df, batch_id):
  count = (change_df
           .count()
          )
  print(f"Number of unique client_ids changed={count}")

(key_df
   .writeStream
   .option("checkpointLocation", "s3://da-datastore-client.dev-cignasplithorizon/checkpoint/delta-join-test")
   .foreachBatch(batch_func)
   .trigger(once=True) # Trigger once is enabled but can be disabled for 24 x 7 streaming
   .start()
)
```

Conclusion

The gap on propagating changes from source to downstream applications is getting narrower with Deltalake CDC feature. This can effectively work to process changes from sources to Kafka to raw zone to integrated and for-purpose zones in change only manner easily without a lot of hurdles.

No labels