KSQL and Slowly Changing Dimension Data

Some traditionally useful ETL tools:

SQL – Declarative and understandable by the business.
Spark – Flexible ETL workhorse, can SQL
Kafka – Proven platform for fast data messaging*

*circular buffer, not a queue

Kafka KStreams

- External processor that reads one or more input Kafka topics and outputs to a Kafka topic
- Similar to a Java Spring or Spark Streaming job that just reads and writes Kafka Topics
- Handles filters, aggregations*, joins**
- Tighter Kafka integration
- KTables! An abstraction of the table/changelog duality

* e.g. time-windowed

** must be co-partitioned

KSQL: A SQL layer on top of KStreams

- Simpler than Java KStreams API, less code => less bugs
- Easier to understand by business
- Typical SQL functions (e.g. UCASE) plus UDF/UDAF
- Still have to reason about partitioning though, but they provide topic repartition commands

Streaming Joins

Some kinds of streaming joins:

- **Stream-Table**: static (lookup code table 78702→ Austin)
- **Stream-Stream**: Time windowed (order.custid=click.custid)
- **Stream-Stream as KTable**: Dynamic dimensions (order.currencyid=exchange_rate.currencyid)

Difficult piece - SCD

- “Slowly” changing dimension (SCD) data use case.
- Typical example:
  - 10,000-100,000 rec/sec fact data stream (clicks, IOT sensors)
  - joined to **multiple** 10-100 rec/sec dimensions by various keys (customer status, device configuration) data stream.
- Need accurate and fast current value for keys.
- Keys might not have changed in years, or maybe a second ago.

Spark Streaming Challenges

- Mini batch latency.
- Immutable RDDs.
- Reloading dimension data from external source every N batches (don’t).
- Even with Continuous Streaming still need to localize partitions/keys with shuffles.

KTables

- Get only the latest Value for a Key backed by a keyed compressed topic, unlike with time series data.
- Ideal for SCD star joins
- Currently must join one table at a time
- Cannot join across partitions, but therefore can get partition-local optimization advantages
- Partitioning is usually by hash of the topic key (or concatenation of multi-part logical key)
- Need a way to save and load bootstrap/backup/checkpoint data.
- Not ideal for some use cases where you need to aggregate across partitions or with a lot of skew

Caveats

- Basics: auto.offset.reset setting and understand partitioning first.
- Should have good comprehensive Kafka monitoring in place, e.g. consumer group and replication lags.
- KSQL creates a lot of internal topics each multiplying storage requirements.
- To reduce storage, may want to reduce default Kafka topic retention from a week to hours or less.
- Can still turn off topic auto-create because KSQL uses admin access to do it.
- May want to turn off the marketing phone-home-with-your statistics setting.
- The went off Apache license to their own for KSQL in 5.1.

References

- https://en.wikipedia.org/wiki/Slowly_changing_dimension
- https://docs.confluent.io/current/ksql/docs/tutorials/examples.html#ksql-examples

Other similar, but hosted technologies:

- https://docs.aws.amazon.com/kinesisanalytics/latest/sqlref/sql-reference-join-clause.html
- https://docs.microsoft.com/en-us/stream-analytics-query/join-azure-stream-analytics
- https://cloud.google.com/blog/products/gcp/apache-kafka-for-gcp-users-connectors-for-pubsub-dataflow-and-bigquery