StreamSets Integration with Delta Lake using Standalone

A while ago we talked to a prospect that wanted to stream their IoT data into Delta Lake without a Spark (Databricks) cluster running continuously. While streaming data into Delta Lake is supported by Streamsets, the second part of this requirement made me raise an eyebrow. The prospect wanted the pipeline to run 24/7, and running a Databricks cluster can be potentially expensive. Here I'm going to explore the potential solution to this requirement using Delta Standalone integration.

A Delta Lake is a set of data (parquet) files and the accompanying Log. While data files can be read/written as just that, files, it's the Log that provides the benefits of Delta: ACID compliance, versioning, etc. Data files are immutable; to change data in a Delta table, one needs to add new files with changed data, and mark some existing files as obsolete.

Delta Standalone is a Delta Log client. It provides, for a reading scenario, a way to read only the required files, given selections, and, for a writing scenario, a way to update Delta Log *after data files have been added*. This is an important point: Delta Standalone doesn't provide tools to change data files like I described above; it's the responsibility of a client. It's simple enough just to insert new data (i.e. append a new file) into a Delta table, but, in deleting/updating/merging operations, multiple files should be read and rewritten. That may quickly become a complex and resource-consuming thing to do, exactly why an Apache Spark cluster would be handy.

With that in mind, how can Streamsets help? While the Streamsets Data Collector Engine (SDC) provides out of the box seamless integration with various sources and targets of data, including on-prem/cloud, batch and streaming, is also extendable, as it has freehand coding evaluators for JavaScript, Jython and, what's most interesting here, Java (in the form of Groovy).

Here was the plan:

1. Gather Java libraries related to Delta Standalone functionality, and their compile dependencies, and upload them to Streamsets.
2. Code a couple of Groovy evaluators:
    - inserting into a Delta table stored in an Azure Data Lake Gen2 (ADLS2) directory;
    - reading from a Delta table;

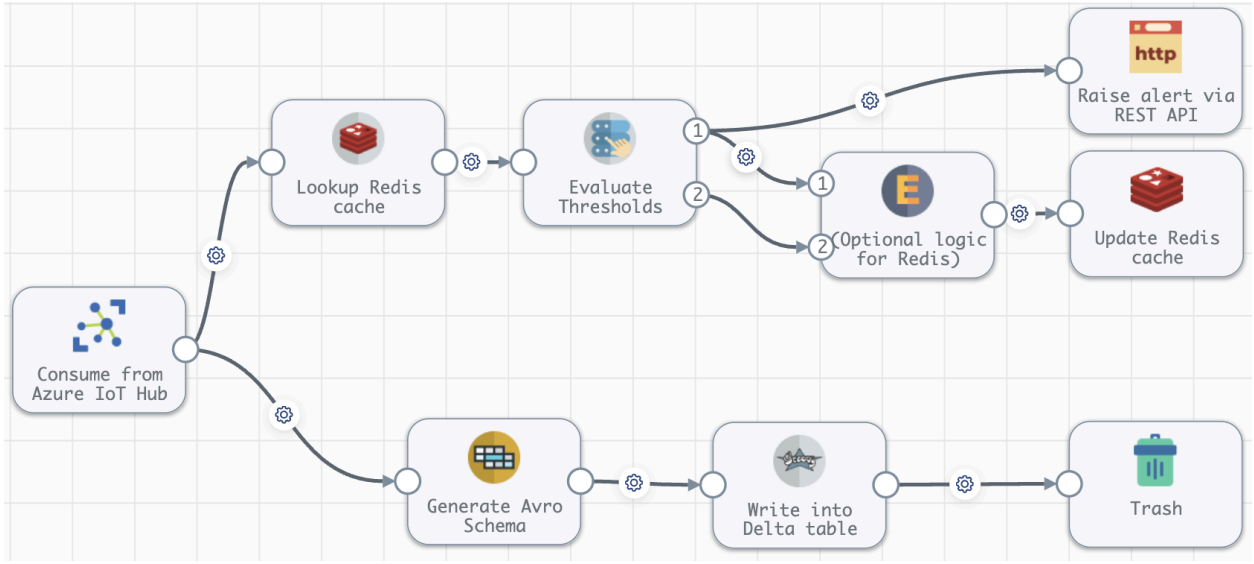
After that, the engine would take care of the rest: integration with other sources/destinations, packaging data, providing schema for writing (I just used an out of the box Schema Generator), job orchestration and monitoring.

I have published the code for Microsoft Azure-based implementation of Delta Standalone with Groovy in a github repo, along with brief instructions on how to set it up.

A pipeline design option might be like this:

- consume sensor readings from an Azure IoT Hub,
- optionally perform any logic that requires “live” data, e.g., check if measurements delivered from the sensor are within some thresholds for alerting, run REST calls to specific APIs, or even evaluate data against a machine learning model
- write the batch into a Delta table of choice on ADLS2

Here’s an example design of such a pipeline in Streamsets canvas.



Captions on pipeline stages are pretty self-explanatory, the idea is to apply lightweight logic on “live” data, involving a fast-performing Redis cache, while capturing incoming raw data in a Delta table, where it becomes available for further data transformations which could be done with a StreamSets Transformer for Spark Engine (using a Spark cluster of choice).

As for the Delta reader part, I mostly developed it for the completeness of the picture. Use cases for Delta reading in StreamSets would be scarce, as you rarely extract data from your data warehouse. Rather, you just report on that data; even if such extraction is required (e.g., from Gold data, following Databricks terminology), it would be faster to do via file export. I could imagine, however, a need for a lookup against a small Gold level Delta table as a part of a pipeline processing some other input.

Conclusion:

StreamSets can help you overcome the central struggle of all data engineering: balancing performance with cost effectiveness. The solution described above lets you stream data into Delta Lake without a Databricks cluster running continuously, allowing you to leverage the power of Delta Lake and keep down the cost of standing up a cluster. You get the best of both worlds and the added benefit of being able to transform data in flight before it lands in Delta Lake. A clean, streamlined data solution of the kind that StreamSets excels at. Let us help you find your balance.