Azure Stream Analytics Vs AWS Kinesis Data Analytics services

Query

To start, let's check the **query composition**. The query in **Azure Stream Analytics** is composed of 3 parts: input, transformation and output. The service executes the query as a job on top of a parallel data processing layer. Among the inputs, you will find pure native streaming services like IoT Hubs or Event Hubs, but you can also use static storage with Azure Blob Storage or Data Lake Gen 2.

An interesting concept present in Stream Analytics is the **reference data input**, so a static or slowly changing dataset often used to enrich the streaming input. It can come from SQL Database table, Blob Storage or Data Lake Storage Gen 2. One point to notice, though. The bigger the reference dataset is, the longer the query can take to execute.

Regarding the transformation logic, you define it with a SQL-like language and extend it with custom functions written in JavaScript or C#.

Every query can have more than 1 input or output. However, the max number of them is limited by the service and, as of this writing, is equal to 60.

And what about the queries in **AWS Kinesis Data Analytics**? Here the querying part is a bit more complex. It's still composed of an input, an output, and a transformation, with a few subtleties, though! The input is called an in-application stream.

You can use it also as an intermediary step. To send some data to an in-application stream, you use another operator called pump. This description applies only to the SQL-based processing.

Because yes, you can use a programming language to define the processing query! If you do, you will have to implement it with Apache Flink API, and in that scenario, Kinesis Data Analytics becomes a compute environment. You can also leverage the SQL-based query by adding an extra preprocessing step calling a Lambda function.

But despite these differences in the data processing part, Kinesis Data Analytics shares some features with Stream Analytics. It has a similar reference dataset feature as Stream Analytics which, as of this writing, supports only the datasets stored on S3.

Regarding the input data sources, Kinesis Data Analytics supports AWS' streaming services like Kinesis Firehose and Kinesis Data Streams. Of course, I mention here only the SQL API because if you rely on Apache Flink, you can use its data sources.

I've just seen that I haven't mentioned the output yet! Here, if we stay with Kinesis Data Analytics SQL, Stream Analytics offers a wider range of possibilities. You can write the data to streaming systems (Event Hubs, IoT Hubs, Service Bus) and more static stores like

CosmosDB, Blob Storage, or even PowerBI! Kinesis Data Analytics SQL supports only Kinesis Firehose and Kinesis Data Streams. Once again, you can rely on Apache Flink API and the supported sources, for from the flexibility standpoint, AWS offers more possibilities.

Processing semantics

Both services work with an **at-least once delivery guarantee** which strongly depends on the type of the output data store. If you write your data to a key-based data store like Table Storage or CosmosDB, and your processing logic is idempotent, then we can consider that the service provides an exactly-once guarantee. The same applies to Kinesis Data Analytics.

And what about **late data management**? Stream Analytics deals with late data by a feature called **late arrival policy** and it's based on the comparison between the event time of the event and its arrival time to the pipeline. If this difference is bigger than the allowed lateness, you can decide to drop it or adjust its event time to the maximum late arrival tolerance time.

Stream Analytics also has a configuration for **early arriving events**, which are the events with the arrival time bigger precedes the arrival time. Regarding Kinesis Data Analytics SQL, I didn't find any specific feature to deal with late events, but once again, you can rely on Apache Flink's watermark for that.

Stream Analytics also has something specific to manage the **out-of-order events**. It's called **out-of-order policy** and an allowed interval to handle out-of-order events. Put another way, if the difference between the arrival time and the event time is smaller or equal to the out-of-order interval, the service integrates this event to the processing window and puts it in a corresponding place - so before some of the already processed events.

Think about it like about a window where some events are accumulated and reordered before emitting the final output. Kinesis Data Analytics supports events ordering with the **ORDER BY** clause in SQL, or windows in Apache Flink API.

Scaling

And to terminate, let's see how both services scale. After all, we're dealing here with continuously arriving data that don't always have the same load. The concept in Stream Analytics responsible for scaling is **Streaming Unit**.

It represents the compute (CPU, memory) resource associated with the job. To detect whether you should scale or not, you have to monitor the SU usage and "Backlogged events". If these values are high, you can consider starting the scaling action. The full capacity of a single computing node is 6 SU.

Kinesis Data Analytics uses a similar concept to describe the compute layer. It's called **Kinesis Processing Unit** and each unit comes with 1 core, 4GB of memory, and 50GB for Apache Flink-based jobs. These jobs also support auto-scaling based on the hardware usage - CPU >= 75% for 15 minutes for scaling up, CPU < 10% for 6 hours for the scaling down action.

To control the level of parallelism in Kinesis Data Analytics, you can use the **Parallelism** property that will divide the partitions of the input. In other words, if you set the Parallelism to 5 for a Kinesis Data Stream having 10 shards, every in-stream application will process 2 shards.

Stream Analytics supports parallelism with the **partitioning** requiring a partitioned input and partition-based processing. When these criteria are met, the service will process every partition in a separate process. A recommended strategy to parallelise the processing for Stream Analytics is to make the number of nodes (1 node = 6 SU) a divisor of the number of input partitions. For example, an input with 4 partitions could be processed by 2 nodes (12 SU) or 4 nodes (24 SU).

What will be your conclusion? For me, if we stop the comparison on Kinesis Data Analytics SQL, both services are quite similar. They support SQL, distributed computing, and various streaming semantics. But I wouldn't say they're the same.

My feeling is that Kinesis Data Analytics counts a lot on Apache Flink compute model and provides "only" the serverless compute environment for it. Which is a great and flexible idea but requires mastering another tool. Stream Analytics doesn't have such flexibility but on the other hand, can be easier to approach thanks to the SQL support and service-managed policies for late and out-of-order events.