Event Streaming Approaches from Segment to Redshift

Action	Author	Date	Purpose
Document created	Vasista Polali	10/12/2018	To recommend designs for building a real-time streaming application to stream events from Segment to Redshift

Approaches to Streaming data from Segment to Redshift

Kinesis Rate limits:

- Refer to https://docs.aws.amazon.com/streams/latest/dev/service-sizes-and-limits.html for Data Stream limits.

1.) Kinesis Firehose

Kinesis Firehose → AWS Firehose-Redshift Delivery Streams →Redshift Staging → Deduplication → Main Table

Implementation:

- Using AWS Firehose-Redshift Delivery Streams.

Benefits:

- Simple and minimal development effort. Only needs configuration on AWS.
- Near real time, takes around 60s for the whole process end-to-end into staging table in redshift.
- Auto scaling taken care by AWS.
- Can make copies of data in s3 as well for usage by other systems.

Restrictions:

- No custom processing capability, if any transformations have to be done on the messages in the future then the pipeline becomes redundant.
- Only works with S3 and Redshift.

2.) Kinesis streams and Lambda

Kinesis streams → Lambda → S3 → AWS Redshift Database Loader →Redshift Staging → Deduplication → Main Table

Implementation:

- AWS lambda configured with Stream Based model to continuously ingest messages in batches from kinesis streams.
- The messages are stored as objects in S3 as Redshift Database loader from AWS can only process events of S3.
- The messages are loaded into a Redshift staging table using Redshift Database loader which uses lambda's triggered by S3 events and the processed files are recorded in Dynamo DB to enable only incremental upload.(This is out-of-the-box)

- Deduplication will be done while loading from staging into the main table.

Benefits:

- Direct, on-demand and event based. Doesn't have to be up and running all the time.
- Near real time as it is event based. Latency estimate is under a second end-to end, into staging table in redshift under ideal conditions.
- Modest development effort, since being serverless, less configuration and maintenance and easy operational management.
- The functions can be easily modified, in the future, to change the logic or connect to other Data sources and Sinks.
- Auto scalable and myriad choice of programming languages.

Restrictions:

- Have to use two layers of lambda's , one to put data into S3 and load to Redshift thereafter introducing a little sense of complexity.
- Subject to vendor rate limits.
- May prove to be expensive than running a Spark Streaming app, in case of higher throughputs.
- No control over Environment which may affect latency sometimes as the resource sharing is being controlled by the Vendor. But with continuous flow of events cold start may not kick in there by mitigating this restriction.

3.) Spark Streaming

Spark Streaming→ Kinesis receiver → Micro Batch-DStream → DataFrame → Spark-Redshift connector →Redshift Staging → Deduplication → Main Table

Spark Structure Streaming → 3 rd Party Kinesis connector→Only updated records(deduplication handled by Spark) → DataFrame → Custom Kinesis Write stream or Spark-Redshift connector or Custom Python function mapping →Redshift Staging → Main Table

Implementation:

- A Spark streaming application developed with Kinesis connector will receive messages as micro batch every couple of seconds.
- The Dstream is converted into a DataFrame and ingested into Redshift using the Spark-Redshift connector
- The connector uses a temporary S3 dir as staging which is out-of the box.
- Spark Streaming supports atleast-once-processing per message.Hence. deduplication will be done while loading from staging into the main table.

Structure Streaming:
This would be the cleanest approach of all if materialized.

- Only-once-per message processing can be handled by Spark Structured streaming avoiding separate deduplication process at later statges in the flow. But..
- No in built receiver for kinesis. Have to leverage a 3rd party Qubole-Kinesis-Sql connector.
- No direct write stream to Kinesis as well, have to experiment with writing custom classes extending Spark streaming-sink interface or trying to use the Foreach sink and Redshift connector or in case of Pyspark mapping a df.tosql(SQL Alchemy)function to load the data directly into Redshift ,on each row in the underlying Dataset.

Benefits:

- Easy to handle complex transformations and loads with higher processing speeds.
- Granular control over the implementation as the various components of the streaming pipeline and their interaction with the underlying Spark environment have to be coded in along with the processing logic.
- Easy to customize and maintain as the processing needs change or become more complex or making it work with other Data sources or Sinks.
- Near real time. Latency estimate is under a second end-to-end, into staging table in redshift.
- Once developed the application can be run on any Spark cluster irrespective of the hosting vendor.
- Optimal and cost effective for high throughputs as the Application has to be up and running 24/7.

Restrictions:

- Overhead of running the infrastructure 24/7.Optimnal for high throughputs.
- By far needs the most development effort.
- May have to compete for resources with other applications running on the cluster.