Evaluating AWS Kinesis and AWS Glue for streaming data processing

Sayani Boral | University of Washington , MSDS
Corbin Charpentier | University of Washington , MSDS

Introduction

With the demand for real-time data and insights rising among business, we are seeing a rapid adoption of event driven architectures such as stream data processing and event streaming. There are many stream data processing technologies in the market like Apache Kafka, Apache Spark, AWS Kinesis Data Streams and others. There are many use cases like IoT sensors, web clickstream , application logs, smart buildings, where data is continuously generated and stream data processing can be used to generate real-time insights or used in applications.

Let us first understand what streaming data means. Stream processing is the real-time processing of data in transit. Unlike batch processing where the data input is bounded - of a known and finite size, in stream processing the data is unbounded. In streaming terminology, an event is generated once by a *producer* (also known as a *publisher* or *sender*), and then potentially processed by multiple *consumers* (*subscribers* or *recipients*) .
In this project, we wanted to understand how scalable streaming data pipelines can be created using AWS Glue. We situate this service within a simple ETL pipeline, between Amazon Kinesis Data Streams, which generates test data and streams it to the translation step, and S3, the persistent data store.
We evaluate AWS Glue with respect to three variables:

- Ease of use: how easy is the service set up, maintain, and scale?
- Cost: how expensive is the service relative to its ease of use and performance?
- Performance: how fast does the service process large amounts of streamed data
    - To what extent does that performance depend on the amount of resources provisioned?
    - Does the complexity of the transformation make much difference?

With respect to performance, we know AWS services are designed to be horizontally scalable. Glue's ability to vertically scale (i.e. per-instance compute capacity increase), is not well documented. Therefore, we test Glue with three worker configurations, GX0.25, GX1, and GX2 (essentially small, medium, and large compute resources).

Overall, the ETL pipeline was quite easy to set up, and we expect similar ease of maintenance. Cost made sense to us, but became an obstacle for the final tier of testing, performance. For reasons mentioned later in this report, performance was challenging to measure. That said, we find that for the price, the Gx0.25 worker type is most efficient.

Overall Architecture

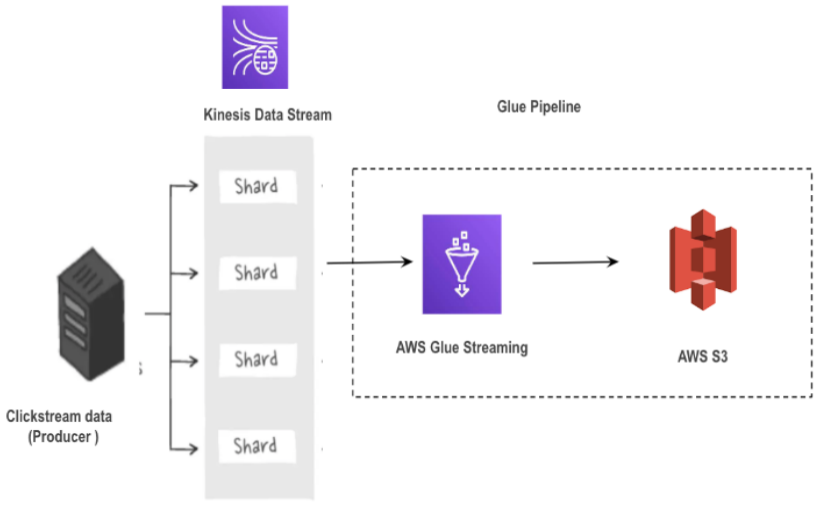


Fig 1. Overall architecture showing flow of data

Tracing data through the system is illustrative of its architecture. Dataflow flows through the ETL pipeline we setup as follows:

1. Kinesis Data Generator (KDG) generates JSON from a template we define (see section TODO)
2. KDG data is ingested and queued in Kinesis
3. Glue pulls data from Kinesis and performs a transformation we define
4. Glue saves the transformed data in S3

Let us now go through each of these components and their architecture.

Amazon Kinesis Data Generator (KDG)

When building a streaming data solution, customers will want to test it with production level streaming data. Generating this data and streaming it to your solution can often be the most tedious task in testing the solution. Amazon Kinesis Data Generator simplifies this task by generating fake data in a template that you specify. It can generate CSV, json and also unstructured data.

The KDG uses Faker.js under the hood. Faker.js is an open source library to generate fake data. All this data can be generated using a web browser. We have noticed it is compatible with Chrome but not with Safari.

Amazon Kinesis Data Streams (KDS)

Specifically, we use Amazon Kinesis Data Streams , a simple streaming service offered by Amazon Kinesis. Generally, KDS is useful when multiple stream consumers need to consume the same data, the data needs to be retained for small amounts of time, and small amounts of data-loss is acceptable.

Operating KDS was quite cheap; between all the services we utilized, KDS incurred the smallest expense.

Our setup used most of the default settings, including 24 hour retention and on-demand shard scaling. This allows roughly 200Mi/s (200,0000 records/second) throughput of writes and 400Mi/s throughput of reads, which is well beyond the needs of our test.

KDG was deployed via a CloudFormation script executed by Lambda. Costs here were trivial as well, even for large amounts of data (10,000 objects per second). KDG simulate click-stream data in the following format:

```
{
    "SessionID": {{random.number(50)}},
    "IP": {{internet.ip}},
    "OrderID": {{random.number(50)}},
    "EventDate": {{date.now}},
    "PageNumber": {{random.number(
        {
            "min":1,
            "max":6
        }
    )}},
    "LocationinPage": {{random.number(
        {
            "min":1,
            "max":15
        }
    )}},
    "Country": "{{random.arrayElement(
        ["US","IN","AU","BD","BE","IE","HU","GR","CH","CN"]
    )}}"
}
```

Typically when a user visits a website and clicks through various components of a webpage for navigation, each click gets captured in a detailed log. This log is the clickstream data. It is extremely valuable to capture this data real-time to understand a user’s website browsing pattern and also purchasing habits. The clickstream data is continuous and high volume, hence the need for a stream ingestion tool like Amazon Kinesis.

Glue Pipeline

Amazon Glue provides managed ETL abstraction over and between other Amazon services. ETL jobs are executed in a streaming Spark environment. For ETL jobs, the user simply needs

to specify the input service (for streaming, Apache Kafka or Amazon Kinesis), define the transformation, and specify the output datastore (in our case, S3).

The Glue job was also configured with its AWS defaults. We allocate G1X workers (4vCPU and 16Gb RAM).

The job is visually defined in Figure 2. The “ApplyMapping” step outputs a Collection type, which necessitates the “Select From Collection” step since S3 does not accept inputs of type Collection.

We feed Kinesis with two levels of streaming, 100 records per second and 1000 records per second.

Fig1. Glue Job Confuration

Problem Statement and Method

We examine various aspects of Amazon Glue as it relates to ETL pipelines in the great AWS ecosystem. Specifically, we analyze ease-of-use, cost, and data throughput. We assessed ease-of-use as we went, noting what was easy or difficult along the way. The analysis of cost was more subjective: after noting the costs, we decided if the performance we observed was congruent to the amount we were expected to pay. Lastly, measuring performance was more involved. TODO

Because of budget limitations, instead of exhaustively testing every combination of Glue worker type (there are three: small, medium, and large) and number of workers allocated, we simply test against each worker type with a constant number of workers (two).

Results

Ease-of-use

Setting up Glue through the browser console was extremely easy. We did not even need to consult any documentation. We simply defined a Glue job, graphically, using Glue Studio. The job consists of an upstream data source, Kinesis in our case, a transformation which we define, and a downstream data dump, S3. Beyond that, we only need to assign Glue the relevant roles, granting Glue permission to interact with Kinesis and S3.

Moreover, since Glue is serverless, there’s nothing to maintain, and it plays extremely nicely with the adjacent AWS services it's intended to interact with (of which, we’ve only tested Kinesis and S3).

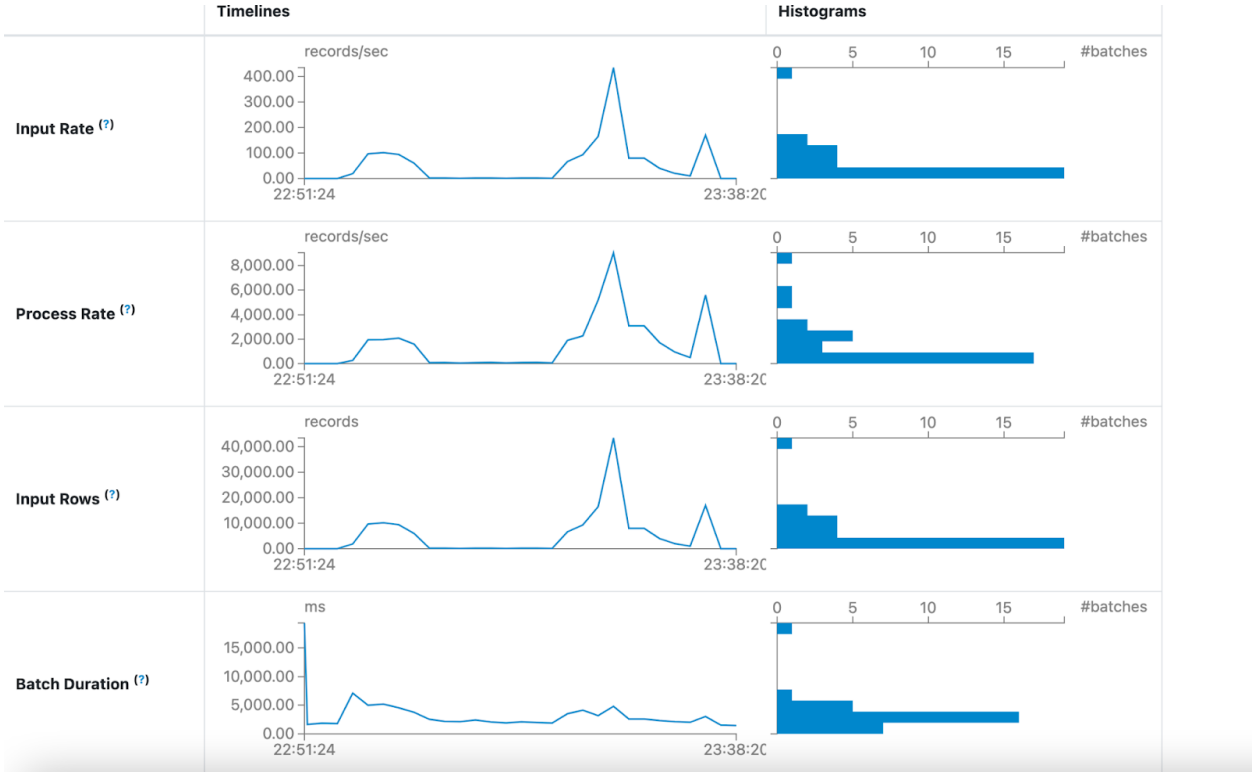
Cost

We think the amount changed was fair for the service offered. Glue workers incur costs by the hour. The price of a blue worker goes up linearly with the compute capability of that worker. So, a GX0.25 worker size, the smallest available, costs $0.11 cents an hour. From the AWS website:

- $0.44 per DPU-Hour for each **Apache Spark** or **Spark Streaming job**, billed per second with a 1-minute minimum (Glue version 2.0 and later) or 10-minute minimum (Glue version 0.9/1.0)
- $0.29 per DPU-Hour for each **Apache Spark job with flexible execution**, billed per second with a 1-minute minimum (Glue version 3.0 and later)
- $0.44 per DPU-Hour for each **Python Shell** job, billed per second, with a 1-minute minimum
- $0.44 per DPU-Hour for each provisioned **Development Endpoint**, billed per second with a 10-minute minimum
- $0.44 per DPU-Hour for each **Interactive Session** billed per second with a 1-minute minimum.
- $0.44 per DPU-Hour for each **AWS Glue Studio data preview session**, billed in 30-minute units and invoiced as development endpoints

Performance

This aspect of our assessment was harder to make sense of.



The plot above shows metrics for the internal Spark job Glue setup. Despite feeding Kinesis a constant stream of data from KDG, we see enormous spikes where Spark processes many more records per second than we generate (process rate). We think the spikes occur because Kinesis stores all data it's received in the last 24 hours, and each new Glue job that comes online creates a new cursor within the Kinesis queue—at the beginning. We're not sure why Glue waits ~30 minute before ingesting all of this data, nor do we understand why Input Rate is a full order of magnitude below Process Rate and Input Rows, though we suspect Input Rate counts batches, not individual JSON blobs.

Regardless, for the very simple transformation we used, we could not stress even the GX0.25 worker type, at least not on our budget (Kinesis is pay-per-stream). GX1 and GX2 did not perform any better than GX0.25. The volume of incoming records from KDG did not matter. We

imagine increasing the complexity of the transformation could necessitate more expensive worker types.

Input records/second from Kinesis	Glue Job & Worker Type	No. of workers	Avg Input Rate/second	Avg Processing rate/ second	Data Processing Units (DPUs)
100 records/second	Small (G 0.25X)	2	53.35	1305	0.5
	Medium (G 1 X)	2	30.57	1007	2
	Large (G 2 X)	2	33.13	1122	4
1000 records/second	Small (G 0.25X)	2	53	1443	0.5
	Medium (G 1 X)	2	64	2061	2
	Large (G 2 X)	2	65	2291	4

Conclusion and Discussion

Overall, we were impressed with the Glue across all fronts. The ETL architecture we landed on, Kinesis → Glue → S3, was extremely easy to set up and use, cost felt fair, and performance scaled as well as we could have imagined. With respect to Glue's vertical scaling, we recommend users start with the GX0.25 work type (the smallest) and scale up from there as needed.

Roadblocks Encountered

Chiefly, we were throttled by AWSLab's restrictive IAM policies. For example, Kinesis Data Generator (KDG) requires creation of a Cognito user, which is not permitted in AWSLab(this is the free tier provided by AWS to the student community). We could not use the AWS CDK for similar reasons because IAM users, groups, and roles cannot be created on AWSLab; therefore, we could not generate user credentials for remote AWS access, which is required by the AWS CLI. We also tried generating the CloudFormation YAML template and uploading it through the CloudFormation web console. However, the CloudFormation web console does not support

many of the features we require. Consequent troubleshooting consumed the bulk of the effort expended so far on this project. Therefore, we decided to use our personal AWS accounts and pay out of pocket.

Additionally, we had originally intended to test Glue performance against Amazon Elastic Map Reduce (EMR). EMR, however, proved difficult to set up and configure, requiring a VPC and multiple supporting AWS services. So, while we couldn't compare cost and performance between Glue and EDR, Glue dominated the ease-of-use category.

The KDG caused another hiccup. KDG setup involves invoking a Lambda function that pulls and executes a CloudFormation template from an S3 bucket owned by Amazon. This spins up the KDG service, associates it with a Kinesis deployment, and creates a Cognito user. However, some parameters encoded in the generated URL used to access KDG were not read correctly by Safari. Launching KDG in Chrome solved the problem.

References

- https://aws.amazon.com/about-aws/whats-new/2020/04/aws-glue-now-supports-serverless-streaming-etl/
- https://aws.amazon.com/blogs/big-data/test-your-streaming-data-solution-with-the-new-amazon-kinesis-data-generator/
- https://ably.com/blog/a-look-at-8-top-stream-processing-platforms
- https://aws.amazon.com/blogs/big-data/best-practices-to-optimize-cost-and-performance-for-aws-glue-streaming-etl-jobs/
- https://aws.amazon.com/blogs/big-data/interactively-develop-your-aws-glue-streaming-etl-jobs-using-aws-glue-studio-notebooks/
- https://docs.aws.amazon.com/streams/latest/dev/introduction.html
- Designing Data Intensive Applications : Chapter 11 Stream processing