

PERFORMANCE AND COST OPTIMIZATION OF STREAMING ETL PIPELINES
USING AWS GLUE

Computer Science

Aditya Ranjan

FNU Independent Researcher

ABSTRACT

Streaming extract-transform-load (ETL) pipelines run continuously, so their cloud cost is driven mainly by reserved compute capacity rather than by the volume of data processed. AWS Glue runs Apache Spark Structured Streaming jobs on a serverless basis and bills them per Data Processing Unit (DPU) hour, which makes every capacity decision visible on the invoice. This paper presents a practitioner-oriented framework for jointly tuning performance and cost in Glue streaming pipelines. It identifies four control levers: worker sizing with auto scaling, micro-batch configuration, transformation efficiency, and output-layer design. It relates each lever to latency, throughput, and DPU-hours through a simple analytical cost model. A worked example using published list prices shows that, for a workload with a pronounced daily peak, auto scaling can lower monthly compute charges by roughly 56 to 65 percent compared with a fixed ten-DPU allocation. The paper also summarizes practical tuning guidance and a protocol for validating the model on production workloads. The results are model-based and indicative rather than measured benchmarks, and should be confirmed against workload-specific measurements before adoption.

KEYWORDS

AWS Glue, Streaming ETL, Cost Optimization, Spark Structured Streaming

1. INTRODUCTION

Organizations increasingly act on data within seconds of its creation, for example in fraud screening, operational monitoring, and personalization. Streaming extract-transform-load (ETL) pipelines support this by ingesting events continuously, transforming them, and delivering them to analytical stores. Unlike a batch job, which consumes resources only while it runs, a streaming job occupies capacity around the clock. Its cost therefore depends less on how much data arrives than on how much capacity is reserved for the busiest moments. The stakes are rising as streaming data increasingly feeds autonomous, agentic AI systems, which shifts the design goal from simply moving data to supporting decisions.

AWS Glue is a serverless data integration service that runs Apache Spark Structured Streaming jobs against sources such as Amazon Kinesis Data Streams and Apache Kafka (Amazon Web Services, n.d.-a). Compute is billed in DPU-hours, where one DPU provides 4 vCPUs and 16 GB of memory. The list price in the US East (N. Virginia) region is USD 0.44 per DPU-hour for Glue 5.1 and earlier, with a lower rate announced for Glue 6.0 (Amazon Web Services, 2026). Since there is no cluster to administer, tuning a Glue streaming pipeline is largely a matter of controlling DPU-hours without breaching latency targets.

This paper makes four contributions: (a) a compact model linking arrival rate, allocated DPUs, window size, and cost; (b) four optimization levers with Glue-specific settings; (c) a worked cost example using published prices; and (d) a protocol for validating the recommendations empirically.

2. Background and Related Work

Spark treats a stream as a series of small batch jobs, an approach introduced as discretized streams (Zaharia et al., 2013). It provides fault tolerance and high throughput, but latency cannot fall below the batch interval. Structured Streaming added a declarative API, replayable sources, and checkpointing that supports end-to-end exactly-once processing (Armbrust et al., 2018). Glue exposes this engine through a `foreachBatch` interface in which the `windowSize` option sets the micro-batch interval, 100 seconds by default.

Surveys of stream processing frameworks (Isah et al., 2019) and benchmarking studies (Karimov et al., 2018) report that latency and throughput depend heavily on configuration, which argues for deliberate tuning rather than reliance on defaults. Engines with record-at-a-time processing, such as Apache Flink (Carbone et al., 2015), can achieve lower latency; Glue is better suited to pipelines whose latency targets are in the range of seconds to minutes and whose operators prefer not to manage infrastructure.

3. Performance and Cost Model

Let $\lambda(t)$ be the arrival rate in events per second, μ the throughput one DPU can sustain at the target utilization, $D(t)$ the number of DPUs allocated, and r the price per DPU-hour. The pipeline is stable only if $D(t) \cdot \mu \geq \lambda(t)$. Monthly compute cost is $C = r \cdot \sum D(t) \Delta t$, summed over hours. End-to-end latency is approximately $w + p$, where w is the

window size and p is the batch processing time. The ratio $\rho = p/w$ indicates headroom; a common heuristic is to keep ρ below about 0.7 to 0.8 so that bursts do not create a growing backlog.

Cost is minimized when $D(t)$ tracks $\lambda(t)/\mu$ while ρ stays under its target. Any DPU idle between peaks is waste, and any window shorter than necessary adds per-batch overhead. Figure 1 shows the reference architecture assumed throughout.

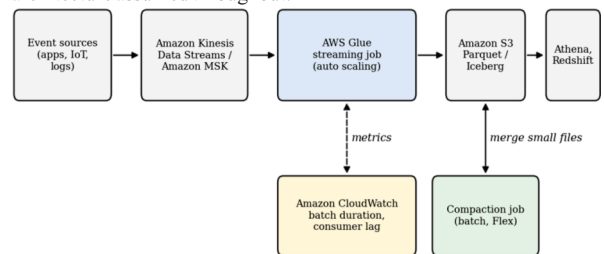


Figure 1: Reference Architecture for a Cost-optimized Glue Streaming Pipeline

4. Optimization Levers

4.1 Right-sizing and Auto Scaling

A fixed allocation sized for peak load wastes capacity at all other times. Glue auto scaling, available from Glue 3.0, adds and removes workers during a run according to demand, with the configured number of workers acting as a ceiling, and billing follows the capacity actually provisioned (Amazon Web Services, n.d.-b). Practitioners should start from a small baseline, use smaller worker types such as G.025X for low-volume streams, and raise the ceiling only when consumer-lag metrics show a persistent backlog.

4.2 Micro-batch Configuration

Longer windows amortize per-batch overhead such as query planning, checkpoint commits, and S3 writes, and they produce larger output files, which lowers DPU-hours per event at the expense of latency. The window should therefore be the largest value that satisfies the latency requirement. Intake limits such as `maxOffsetsPerTrigger` for Kafka sources prevent oversized batches after a restart. Setting `spark.sql.shuffle.partitions` close to the total core count, rather than Spark's default of 200, avoids many tiny tasks on small clusters.

4.3 Transformation Efficiency

Filtering and projecting columns early reduces shuffled data. Built-in Spark SQL functions generally outperform Python user-defined functions, and small reference tables should be broadcast rather than joined through a shuffle. Heavy transformations are usually faster on Spark DataFrames than on Glue DynamicFrames. Watermarks should bound the state of aggregations and deduplication so that memory use and checkpoint size do not grow without limit.

4.4 Output Layout

Frequent micro-batches create many small objects, which inflate S3

request counts and slow downstream queries. Output should be written in a columnar format such as Parquet, partitioned coarsely (for example by date and hour) rather than by high-cardinality keys. Open table formats supported by Glue, such as Apache Iceberg, allow periodic compaction. Because compaction is not latency sensitive, it can run as a scheduled batch job under the Flex execution class, which was priced at USD 0.29 per DPU-hour before Glue 6.0; streaming jobs cannot use Flex.

4.5 Version and Observability

Moving jobs to Glue 6.0 lowers the Spark DPU-hour rate by 30 percent to USD 0.308 in us-east-1 (Amazon Web Services, 2026), subject to compatibility testing. Amazon CloudWatch metrics for batch duration, input versus processed rows per second, and consumer lag should drive alarms, for instance when ρ exceeds 0.8.

5. Worked Cost Example

Consider a stream with a diurnal profile that needs 8 DPUs for 6 hours each day and 2 DPUs for the remaining 18 hours, over a 730-hour month at USD 0.44 per DPU-hour. Table 1 compares four configurations. Configuration D assumes that auto scaling lags demand, and therefore adds 25 percent to the ideal average allocation.

Table 1: Estimated Monthly Glue Compute Cost by Configuration

Configuration	Avg. DPUs	DPU-hours	Cost (USD)	Saving vs. A
A. Fixed 10 DPUs	10.000	7,300.00	3,212.00	–
B. Fixed 8 DPUs (peak-sized)	8.000	5,840.00	2,569.60	20.0%
C. Auto scaling, ideal	3.500	2,555.00	1,124.20	65.0%
D. Auto scaling, 25% overhead	4.375	3,193.75	1,405.25	56.3%

Source: Author calculations from list price of USD 0.44 per DPU-hour (Amazon Web Services, 2026). Excludes Kinesis/MSK, S3, and CloudWatch charges.

Auto scaling reduces cost by 56.3 to 65.0 percent against the over-provisioned baseline, and by 45.3 percent even against a configuration sized exactly to the peak. At the Glue 6.0 rate of USD 0.308, configuration D would cost about USD 983.68 per month, roughly 69 percent below configuration A. These savings follow directly from the assumed profile. A flat workload would gain little from auto scaling, and its savings would come instead from window, worker, and transformation tuning.

6. Validation Protocol

Because the model is analytical, the following protocol is proposed for empirical confirmation. First, a synthetic producer replays the diurnal profile into Kinesis or MSK. Second, configurations A to D are run with window sizes of 30, 60, 100, and 300 seconds. Third, the study records median and 95th-percentile end-to-end latency, throughput per DPU (μ), DPU-hours from the AWS Cost and Usage Report, and files per partition. Fourth, each run is repeated at least three times, and means and variation are reported. A configuration is acceptable only if its 95th-percentile latency stays within the service-level objective.

7. Limitations

The findings are derived from a model and list prices, not from measured benchmarks. Prices differ by region and change over time, and μ varies widely with workload. Ancillary services are excluded. Pipelines needing sub-second latency may be better served by record-at-a-time engines, and sustained high-throughput workloads may be cheaper on self-managed clusters.

8. CONCLUSION

For Glue streaming pipelines, cost is a function of reserved capacity over time. Aligning allocated DPUs with demand through auto scaling, choosing the longest acceptable window, writing compact columnar output, and monitoring headroom together offer substantial savings without sacrificing latency targets. The worked example indicates savings of more than half for a peaked workload. Future work should validate these estimates experimentally across varied workloads, compare Glue with alternative engines, and examine how cost-aware streaming design can support the decision pipelines that agentic AI requires.

REFERENCES

1. Amazon Web Services. (n.d.-a). *Streaming ETL jobs in AWS Glue*. AWS Glue Developer

Guide. <https://docs.aws.amazon.com/glue/latest/dg/add-job-streaming.html>
2. Amazon Web Services. (n.d.-b). *Using auto scaling for AWS Glue*. AWS Glue Developer Guide. <https://docs.aws.amazon.com/glue/latest/dg/auto-scaling.html>
3. Amazon Web Services. (2026). *AWS Glue pricing*. <https://aws.amazon.com/glue/pricing/>
4. Armbrust, M., Das, T., Torres, J., Yavuz, B., Zhu, S., Xin, R., Ghodsi, A., Stoica, I., & Zaharia, M. (2018). Structured Streaming: A declarative API for real-time applications in Apache Spark. *Proceedings of the 2018 International Conference on Management of Data* (pp. 601–613). ACM.
5. Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., & Tzoumas, K. (2015). Apache Flink: Stream and batch processing in a single engine. *IEEE Data Engineering Bulletin*, 38(4), 28–38.
6. Isah, H., Abughofa, T., Mahfuz, S., Ajerla, D., Zulkernine, F., & Khan, S. (2019). A survey of distributed data stream processing frameworks. *IEEE Access*, 7, 154300–154316.
7. Karimov, J., Rabl, T., Katsifodimos, A., Samarev, R., Heiskanen, H., & Markl, V. (2018). Benchmarking distributed stream data processing systems. *2018 IEEE 34th International Conference on Data Engineering* (pp. 1507–1518). IEEE.
8. Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., & Stoica, I. (2013). Discretized streams: Fault-tolerant streaming computation at scale. *Proceedings of the 24th ACM Symposium on Operating Systems Principles* (pp. 423–438). ACM.