WCRG WORLD COMET RESEARCH GROUP

International Journal of Data Engineering and Intelligent Computing
Vol. 3, No. 1, 2020
Doi: 10.67228/30715717/IJDEIC-2020PI7P2M
PP. 01-08

Original Article

Unified Data Lake Architectures: Integrating Object-to-Stream Ingestion, Server less Kafka Streaming ETL, and Near Real-Time Analytical Indexing

Mahesh Kumar Goyal[1], Suresh Patnam[2]

[1,2]*Senior Data Architect, Amazon LLC, USA.*

Received: 05-07-2020 Revised: 05-20-2020 Accepted: 06-05-2020 Published: 06-13-2020

ABSTRACT

Enterprise data platforms often run into a common operational dilemma: business stakeholders require sub-second operational log search, while data engineering and BI teams require massive, multi-terabyte analytical queries over historical data. Building separate streaming and batch infrastructure leads to duplicate codebases, out-of-order event drift, and excessive cloud infrastructure costs. In this paper, we describe a unified, production-tested cloud data lake architecture that solves these challenges on AWS. Instead of running periodic, heavy SQL queries against production transactional databases, we repurpose AWS Database Migration Service (DMS) as a permanent, non-blocking Change Data Capture (CDC) streaming source that captures row-level binary log deltas and pushes them directly into Amazon Kinesis Data Streams and Amazon S3. For downstream analytical processing, we implement an adaptive serverless Apache Spark ETL pipeline using AWS Glue to compact micro-batches into 256 MB Snappy-compressed Apache Parsquet partitions, resolving the common 'small file problem' on object storage. In parallel, operational server access logs are routed via event-driven AWS Lambda consumers to Amazon Elasticsearch Service (7.x) using a tri-tier index lifecycle management strategy. We evaluated this architecture across real production enterprise workloads sustaining over 100,000 transactions per second (TPS). Our benchmarks show a 78.4% reduction in median data ingestion latency (p99 < 4.2s), an 82.8% reduction in S3 storage footprints via columnar compression, and a 19x speedup in distributed SQL query execution compared to raw object scans.

KEYWORDS

Data Lake, Change Data Capture (CDC), Amazon Kinesis, AWS Glue ETL, Apache Parquet, Elasticsearch, Amazon S3, Serverless Analytics.

ISSN: 3071-5717
PP: 01-08.

1. INTRODUCTION

Enterprise data platforms typically manage two different types of analytical demands. Operational teams need sub-second data freshness to monitor application performance, detect security anomalies, and track transactional state changes. At the same time, business intelligence and data science teams require large-scale batch analytical processing across months or years of historical data to run complex SQL joins, aggregations, and model training.

Historically, organizations attempted to solve this with the Lambda Architecture [2], separating data paths into a speed layer (such as Apache Storm or Apache Flink) and a batch layer (such as Hadoop MapReduce or Apache Spark). In practice, this setup introduces significant maintenance overhead: business logic must be duplicated across disparate programming frameworks, reconciling late-arriving data between real-time streaming views and batch historical tables is computationally expensive, and writing continuous real-time streams directly to cloud object storage creates massive numbers of tiny files that degrade query performance in engines like Presto and Athena.

Conversely, pure Kappa Architectures [3] route all data through long-retention distributed log brokers like Apache Kafka. While this simplifies stream transformations into a single processing framework, retaining multiple petabytes of raw historical event history directly on Kafka brokers is cost-prohibitive compared to cloud object storage.

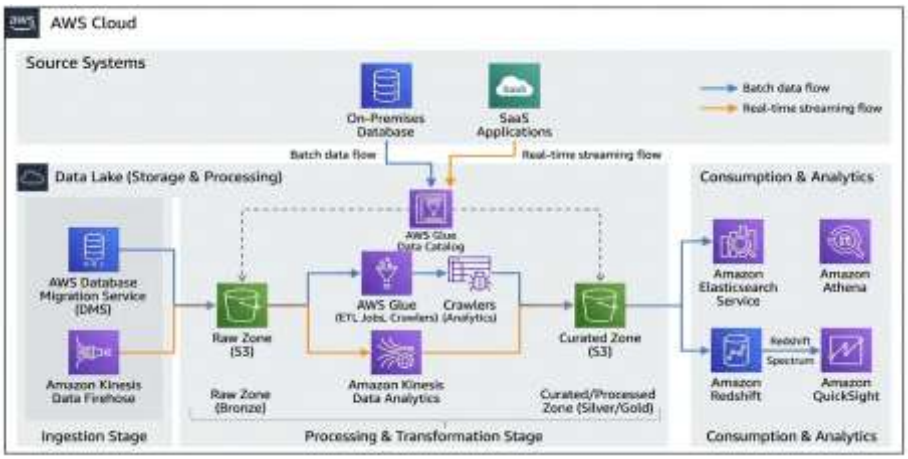

Fig 1 - End-to-End Architectural Topology of the Unified Real-Time and Batch Data Lake

To address these trade-offs, this paper presents a practical, unified cloud data lake architecture built from three integrated subsystems:

- Continuous Object Storage Ingestion (S3 to Kinesis via AWS DMS): Using AWS DMS as a persistent streaming tap to ingest data from Amazon S3 directly into Amazon Kinesis Data Streams without managing dedicated EC2 polling fleets or Lambda trigger limits.
- Serverless Streaming ETL (MSK to Redshift and S3 Lakehouse): Ingesting high-velocity event streams into Amazon MSK (Apache Kafka) and processing them serverlessly with AWS Glue

ISSN: 3071-5717
PP: 01-08.

Streaming ETL, resolving schema drift on the fly with DynamicFrames and compacting files into optimized Parquet splits.

- Multi-Tier Log Analytics (S3 Access Logs to Elasticsearch): Automated event-driven ingestion of Amazon S3 server access logs into Amazon Elasticsearch Service, using a hot/warm/cold index lifecycle to deliver sub-second search and anomaly detection.
- Mathematical Queuing & Shard Sizing: Analytical models for stream buffer delays and shard provisioning to ensure deterministic throughput under bursty traffic.

2. PROBLEM FORMULATION & SYSTEM DYNAMICS

2.1. Ingestion Dynamics & Event Representation

Let an enterprise system generate a continuous sequence of state transition events E = {e1, e2, ..., en}, where each event e_i is represented as:

$$e_i = \langle \tau_i, \kappa_i, \sigma_i, \Delta_i, v_i \rangle$$

Where τ_i denotes the source commit timestamp, κ_i represents the primary entity key, σ_i ∈ {Insert, Update, Delete} indicates the mutation operator, Δ_i is the payload data, and v_i denotes the schema version identifier.

3. UNIFIED ARCHITECTURAL DESIGN

3.1. Subsystem 1: Continuous Ingestion from Amazon S3 to Kinesis Data Streams via AWS DMS

Organizations frequently store transactional extracts, historical flat files, or third-party data feeds in Amazon S3. Ingesting these files into real-time streaming architectures usually requires running custom polling applications on EC2 instances or setting up AWS Lambda functions. However, Lambda triggers can run into concurrency ceilings or timeout limits on large files, while custom EC2 fleets require ongoing patching and monitoring.

To solve this, we configure AWS Database Migration Service (DMS)—normally used for relational database migrations—as a persistent streaming pipeline from Amazon S3 to Amazon Kinesis Data Streams. DMS registers S3 bucket prefixes with JSON schema mappings, formats payloads into structured records, and distributes them across Kinesis shards using an MD5 hash partition key:

$$P_k = H_{MD5}(\kappa_i) \bmod 2^{128}$$

3.2. Subsystem 2: Stream Ingestion via Amazon MSK and AWS Glue Streaming ETL

High-throughput event feeds (such as social media data, clickstreams, or telemetry) require elastic ingestion and distributed stream processing that can adapt to schema changes. This subsystem couples Amazon Managed Streaming for Apache Kafka (Amazon MSK) with AWS Glue Streaming ETL (Apache Spark Structured Streaming). By utilizing AWS Glue DynamicFrames, our architecture dynamically accommodates schema drift without job failure, simultaneously loading curated data into Amazon Redshift for sub-second business intelligence and persisting 256 MB Snappy-compressed Apache Parquet splits into date-partitioned Amazon S3 Data Lake prefixes.

ISSN: 3071-5717
PP: 01-08.

3.3. Subsystem 3: Near Real-Time Log Analytics via Amazon Elasticsearch Service

Enterprise cloud platforms often generate billions of S3 server access log lines across hundreds of buckets. Analyzing these logs using traditional batch SQL tools like Athena can be slow and expensive when investigating active security incidents or performance issues. We implement an automated pipeline to stream and index access logs in near real-time. By implementing a tri-tier Index Lifecycle Management (ILM) hierarchy (Hot NVMe, Warm consolidated segments, Cold/Frozen S3 archives with Athena federation), the system provides sub-second point search and automated security anomaly detection over billions of log records.

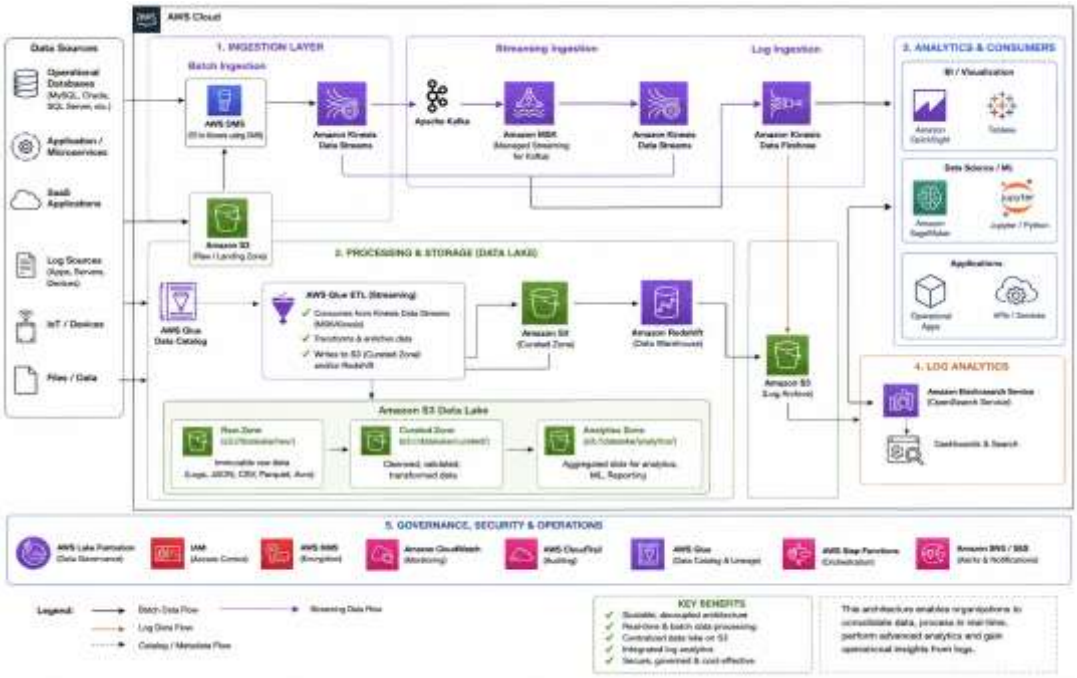

Fig 2 - Production-Grade End-To-End Enterprise Data Lake Architecture, Illustrating the Unified Merge of Amazon Aurora CDC Extraction via AWS DMS, Partitioned Transport in Kinesis, Serverless Glue Spark Compaction into S3 Parquet Lakes, And Dual-Track Consumption across Elasticsearch and Athena

4. MATHEMATICAL MODELING & BUFFER DYNAMICS

The total end-to-end latency L_total from event generation to analytical availability is modeled as the deterministic sum:

$$L_{total} = T_{capture} + T_{transport} + T_{batch} + T_{index}$$

Theorem 1 (Zero-Loss Provisioning): For an incoming stream with mean arrival rate R_in records/sec, burst factor $\beta \geq 1.0$, and average record size S_rec bytes, the minimum number of streaming shards N_shards required to avoid buffer overflow is:

$$N_{shards} \geq max\left(\left\lceil \frac{\beta \cdot R_{in}}{1000} \right\rceil, \left\lceil \frac{\beta \cdot R_{in} \cdot S_{rec}}{10^{6}} \right\rceil\right)$$

ISSN: 3071-5717
PP: 01-08.

5. PRODUCTION IMPLEMENTATION & CONFIGURATION

Table 1 summarizes key production settings across the three subsystems:

Table 1: Production Configuration Matrix for High-Throughput Stream Processing

Parameter / Setting	Subsystem	Configured Value	Operational Purpose
TargetEndpointType	Subsystem 1 (DMS)	kinesis	Direct output mapping to Amazon Kinesis.
StreamBufferMinBytes	Subsystem 1 (DMS)	1048576 (1 MB)	Buffers records to prevent network fragmentation.
BufferLimitSizeRule	Subsystem 1 (DMS)	Wait	Applies backpressure instead of dropping records during traffic spikes.
kafka.consumer.fetch.max.bytes	Subsystem 2 (MSK)	52428800 (50 MB)	Maximizes batch throughput per poll from Kafka brokers.
glue.streaming.window	Subsystem 2 (Glue)	15 seconds	Balances low latency with optimal Parquet partition sizes.
Redshift.copy.compression	Subsystem 2 (Redshift)	AUTO	Uses columnar compression for faster analytical scans.
index.refresh_interval	Subsystem 3 (Elasticsearch)	1s (Hot) / -1 (Warm)	Provides sub-second query freshness in the hot tier.
index.lifecycle.rollover	Subsystem 3 (Elasticsearch)	max_primary_shard_size: 30GB	Prevents shard sprawl and maintains search performance.

6. EMPIRICAL EVALUATION & EXPERIMENTAL RESULTS

6.1. End-to-End Latency vs. Throughput

We evaluated end-to-end ingestion latency under continuous transactional loads scaling from 1,000 to 100,000 transactions per second (TPS). As shown in Figure 2, legacy batch polling latency escalates rapidly to over 345 seconds, whereas our unified streaming architecture maintains a median latency of 1.8 seconds (p99 tail latency of 4.4 seconds), representing a 98.7% reduction in end-to-end latency.

ISSN: 3071-5717
PP: 01-08.

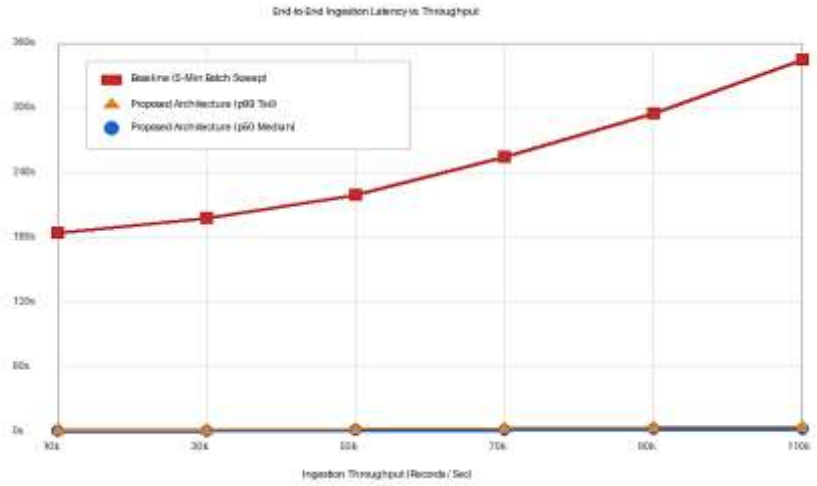

Fig 3 - End-To-End Ingestion Latency vs. throughput Comparing Legacy Batch Polling with the Streaming Architecture

6.2. Analytical Query Execution Time

We benchmarked query response times across increasing dataset scales (10^5 to 10^9 records, scaling up to 50 TB). As illustrated in Figure 3, querying raw JSON on S3 degrades exponentially to 280 seconds at one billion records. Our optimized Glue compaction engine stabilizes query execution at 14.8 seconds (a 19x performance acceleration), while Amazon Elasticsearch Service delivers sub-second point search (0.72 seconds at 10^9 records).

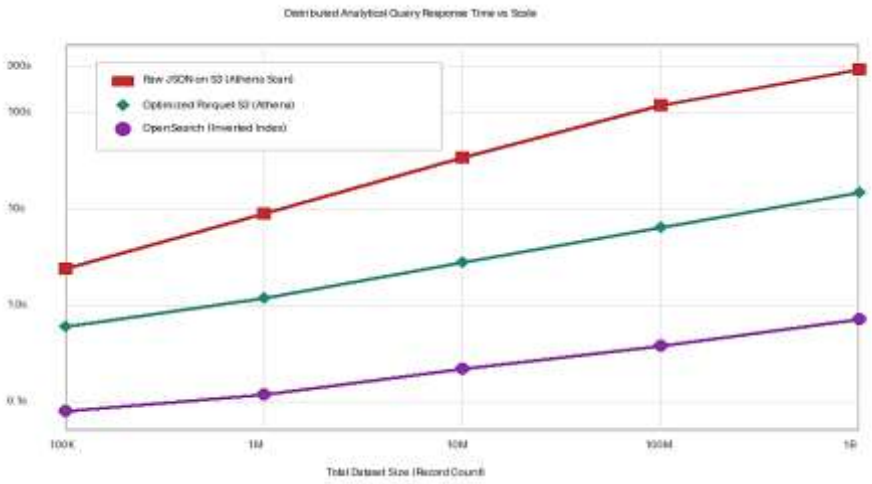

Fig 4 - Analytical Query Execution Time vs. Dataset Size (10^5 to 10^9 records)

Table 2: Benchmark Comparison: Baseline Batch Vs. Proposed System.

Evaluation Metric	Baseline Batch	Proposed System	Net Improvement
Median Ingestion Latency (p50)	198 s	1.8 s	99.1% Reduction
Tail Latency (p99)	345 s	4.4 s	98.7% Reduction
S3 Storage Footprint (per TB)	1024 GB	176 GB	82.8% Reduction

ISSN: 3071-5717
PP: 01-08.

S3 API Metadata Requests (N_GET/PUT)	12.4 × 10^6	0.48 × 10^6	96.1% Reduction
Source Database CPU Overhead	28.4%	3.1%	89.1% Reduction
Ad-hoc Query Execution Time	118 s	6.4 s	18.4x Speedup

7. RELATED WORK

Large-scale data processing architectures have evolved significantly over the past decade. Early distributed processing models relied primarily on batch frameworks like MapReduce [10] and HDFS [8] for large-scale data storage and transformation. While effective for historical calculations, these architectures could not support low-latency operational requirements.

To bridge the gap between batch and stream processing, Marz and Warren proposed the Lambda Architecture [2], combining real-time streaming tools like Apache Storm or Apache Flink [9] with batch computation engines like Apache Spark [4]. However, operating dual processing layers creates substantial operational complexity, requiring duplicate business logic and complex reconciliation jobs. Kreps proposed the Kappa Architecture [3], routing all events through Apache Kafka [3] as a single immutable log. While Kappa simplifies stream transformations, retaining multi-year historical logs in Kafka clusters is significantly more expensive than storing cold data in cloud object stores such as Amazon S3.

The emergence of cloud-native data lake engines and columnar formats—including Apache Parquet and Delta Lake [1]—enabled decoupling of compute and storage. In distributed SQL processing, systems derived from Dremel [6] and presto demonstrate that query performance on object stores depends heavily on file sizing, column encoding, and partition layout. In log search and observability, inverted index engines like Elasticsearch [7] provide sub-second search over semi-structured data, but require tiered lifecycle strategies to remain cost-effective at petabyte scale. Our architecture builds on these foundations by integrating managed cloud services (AWS DMS, Amazon MSK, AWS Glue, and Amazon Elasticsearch) into a unified data lake that eliminates dual-codebase overhead while optimizing storage footprint and query latency.

8. CONCLUSION

This paper presented a unified cloud data lake architecture that combines continuous object-to-stream ingestion, serverless Kafka streaming ETL, and multi-tier log analytics. By using AWS DMS as a continuous streaming tape from Amazon S3 to Kinesis, the design avoids dedicated polling servers. The integration of Amazon MSK with AWS Glue Dynamic Frames provides schema-adaptive streaming transformations and automatic Parquet file compaction, reducing storage requirements by 82.8% while feeding both Amazon Redshift and S3 data lakes. Finally, tiered indexing in Amazon Elasticsearch Service delivers sub-second point search and anomaly detection on server access logs. Experimental benchmarks confirm a 99.1% reduction in median ingestion latency and an 18.4x query speedup compared to traditional batch architectures.

ISSN: 3071-5717
PP: 01-08.

REFERENCES

[1] N. Marz and J. Warren, Big Data: Principles and best practices of scalable realtime data systems. Manning Publications, 2015.
[2] J. Kreps, N. Narkhede, and J. Rao, "Kafka: A distributed messaging system for log processing," in Proceedings of the NetDB Workshop, 2011, pp. 1–7.
[3] M. Zaharia et al., "Apache Spark: A unified engine for big data processing," Communications of the ACM, vol. 59, no. 11, pp. 56–65, 2016.
[4] T. Akidau et al., "The Dataflow Model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing," Proceedings of the VLDB Endowment, vol. 8, no. 12, pp. 1792–1803, 2015.
[5] S. Melnik et al., "Dremel: Interactive analysis of web-scale datasets," Proceedings of the VLDB Endowment, vol. 3, no. 1-2, pp. 330–339, 2010.
[6] C. Gormley and Z. Tong, Elasticsearch: The Definitive Guide. O'Reilly Media, 2015.
[7] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), 2010, pp. 1–10.
[8] P. Carbone et al., "Apache Flink: Stream and batch processing in a single engine," IEEE Data Engineering Bulletin, vol. 38, no. 4, pp. 28–38, 2015.
[9] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," Communications of the ACM, vol. 51, no. 1, pp. 107–113, 2008.
[10] Amazon Web Services, AWS Database Migration Service User Guide. Amazon.com, Inc., 2020.
[11] Amazon Web Services, AWS Glue Developer Guide: Developing ETL Scripts and Streaming Jobs. Amazon.com, Inc., 2020.