International Journal of Core Engineering & Management
Volume-7, Issue-03, 2022 ISSN No: 2348-9510

OPTIMIZING MULTI-CLOUD DATA STREAMING: A KAFKA AND KINESIS HYBRID ARCHITECTURE FOR REAL-TIME FINANCIAL COMPLIANCE

Tathagata Roy
Independent Researcher, Colorado, USA
tatha.roy@gmail.com

Abstract

Financial institutions face growing pressure to enforce real-time regulatory compliance across distributed environments, particularly within multi-cloud deployments spanning AWS and Azure. This paper introduces a hybrid data streaming architecture that integrates Apache Kafka with Amazon Kinesis to optimize workflows for Anti-Money Laundering (AML) and fraud detection. The architecture addresses sub-second latency and strict data consistency requirements by evaluating cross-cloud replication strategies, including partition tuning and preserving exactly-once processing semantics under network partitions. A comparative analysis of Kafka Connect and custom replication mechanisms highlights trade-offs in throughput and resilience when bridging heterogeneous infrastructures. Furthermore, the architecture demonstrates integration with Complex Event Processing (CEP) engines to detect suspicious transactions in real time across cloud boundaries. Experimental evaluations demonstrate that the proposed approach achieves secure data lineage, high throughput, and reduced vendor lock-in, enabling financial services to meet regulatory obligations in distributed cloud environments.

Keywords: Amazon Kinesis, Anti-Money Laundering (AML), Apache Kafka, Complex Event Processing (CEP), Cross-cloud Replication, Distributed Systems, Financial Compliance, Low-latency Architecture, Multi-cloud Data Streaming.

I. INTRODUCTION

The financial services industry has historically relied on on-premises data centers to maintain strict control over transaction processing and regulatory compliance. The operational paradigm has since shifted toward cloud-native architectures to achieve scalability, elasticity, and cost efficiency [7], [8]. A substantial portion of financial institutions now adopt multi-cloud strategies, leveraging distinct services from providers such as Amazon Web Services (AWS) and Microsoft Azure [1], [34]. This diversification mitigates the risks of vendor lock-in and enhances resilience by ensuring that critical banking services remain available even during outages affecting a single cloud provider. Despite these advantages, the distribution of data across

heterogeneous environments introduces significant complexity in enforcing real-time regulatory compliance [14], [15].

For compliance workflows such as Anti-Money Laundering (AML) and fraud detection, institutions must maintain sub-second latency, strict data consistency, and secure lineage across cloud boundaries [5], [13]. Traditional single-cloud streaming solutions often fail to meet these requirements when extended to multi-cloud deployments. Apache Kafka, with its distributed log architecture [8], and Amazon Kinesis, with its managed streaming capabilities [1], represent two complementary paradigms for real-time data processing. Integrating these platforms across AWS and Azure requires careful attention to replication strategies, partition management, and exactly-once processing semantics to avoid data loss or inconsistent compliance enforcement [3], [4], [21].

This paper introduces a hybrid architecture that bridges Kafka clusters deployed in Azure with Kinesis Data Streams in AWS to construct secure, resilient, and low-latency pipelines for financial compliance. The design emphasizes strategies for cross-cloud replication, latency optimization, and resilience under network partitions, while also demonstrating integration with Complex Event Processing (CEP) engines to detect suspicious transactions in real time [11], [12], [37]. Comparative evaluation of Kafka Connect and custom replication mechanisms highlights trade-offs in throughput, resilience, and operational complexity [9], [10], [18].

II. BACKGROUND AND RELATED WORK

A. Multi-Cloud Data Architectures

The adoption of multi-cloud architectures in the financial sector has evolved from a disaster recovery strategy into a primary operational model aimed at securing best-of-breed capabilities. By leveraging Azure's integration with enterprise identity systems [34] and AWS's mature analytics ecosystem [1], institutions optimize their technology stack for specific workloads. However, this heterogeneity imposes interoperability costs. Data gravity, the tendency of large datasets to attract applications and services, often leads to siloed environments where data egress becomes both a performance bottleneck and a significant cost center. Architectural patterns emerging around 2020, such as data mesh, advocate decentralized domain-oriented ownership [13]. In a multi-cloud context, this requires federated governance capable of standardizing schemas and policies across Azure Event Hubs [40] and Amazon Kinesis boundaries [1], a challenge not fully addressed by vendor-native tools.

B. Financial Compliance Requirements

Regulatory compliance operates under heightened scrutiny following the Anti-Money Laundering Act of 2020 (AMLA) in the United States [14] and the ongoing enforcement of GDPR in Europe [15]. AML frameworks require the detection of structuring or smurfing techniques, where illicit transfers are decomposed into smaller transactions to evade the $10,000 reporting threshold. Effective detection engines must maintain a global state of customer transactions across cloud environments in real time [5]. This necessitates correlation of events

across regions, such as ingress in Azure East US and a disbursement in AWS US East-1, within milliseconds. Audit standards further demand immutable lineage, requiring compliance officers to prove that inspected data is bitwise identical to the source transaction. Cryptographic verification protocols must therefore survive cross-cloud replication to ensure regulatory integrity [30], [31].

C. Stream Processing Semantics

The divergence between Azure and AWS streaming abstractions complicates multi-cloud integration. Apache Kafka operates on a distributed log model, enabling long-term retention and replay through consumer-managed offsets [8]. In contrast, Amazon Kinesis Data Streams provides a shard-based throughput model with fixed ingress limits of 1 MB/sec per shard and retention windows of 24 hours, extendable to 7 days [21]. These differences challenge the preservation of exactly-once semantics. Replication tools available, such as MirrorMaker 2, is designed for Kafka-to-Kafka replication [16] and often degrade to at-least-once delivery when adapted for Kafka-to-Kinesis workflows. This introduces duplicates during network partitions or retries, which is unacceptable in financial ledgers where balance calculations must remain precise [29]. Engineers therefore implement idempotency keys and stateful deduplication logic at the application layer, increasing the complexity of compliance pipelines [3], [37].

III. SYSTEM ARCHITECTURE

A. High-Level Design

The proposed hybrid architecture integrates Apache Kafka clusters deployed in Microsoft Azure [34] with Amazon Kinesis Data Streams in AWS [1] to construct secure, resilient, and low-latency pipelines for financial compliance. The design is organized into three logical layers: ingestion, replication, and processing. Each layer is responsible for ensuring that data flows seamlessly across heterogeneous cloud environments while maintaining the strict compliance guarantees required by regulations such as AMLA [14]. The architecture segregates the operational domain from the compliance domain, ensuring that heavy analytical workloads required for fraud detection do not degrade the performance of core banking services. See Fig 1 for the hybrid architecture.

B. Ingestion and Replication Layers

The ingestion layer resides in Azure, where Kafka clusters capture raw transactional data from core banking mainframes. These events are serialized into the ISO 8583 standard format [17] and published to local Kafka topics using Java consumers, with alternative serialization frameworks such as Avro and Protocol Buffers also supported for downstream interoperability [38], [39]. Partitioning strategies are applied at this stage to balance throughput across consumer groups while preserving ordering guarantees for customer-specific transaction streams [8]. The replication layer then bridges the Azure and AWS environments. This component is responsible for reading committed offsets from the Azure Kafka partitions and securely transmitting the

payload to the AWS environment via a secure tunnel. To mitigate vendor lock-in, the architecture abstracts the replication logic, allowing the underlying engine to be swapped between standard Kafka Connect workers [9], [10] and custom Netty-based replicators [18]. This abstraction allows for protocol translation from Kafka record batches into Kinesis PutRecords requests [21] without requiring code changes in the upstream banking application.

C. Processing and Compliance Layer

The processing layer operates in AWS, where Kinesis Data Streams feed into compliance engines such as Amazon Kinesis Data Analytics or Apache Flink [4]. This decoupling allows the complex event processing (CEP) engines to scale independently of the ingestion rate. Within this layer, CEP rules are applied to detect suspicious transaction patterns, including structuring and smurfing, in real time [11], [12]. Exactly-once semantics are enforced through idempotent writes and stateful deduplication strategies [3], [37]. This ensures that compliance checks remain consistent even during replication retries. The processing layer also integrates with audit systems to maintain immutable lineage records, enabling regulators to verify that the inspected data is bitwise identical to its source [30], [31].

D. Security and Governance

Security and governance are embedded throughout the architecture to meet zero-trust requirements. Encryption in transit is achieved through mutual TLS (mTLS) across all replication tunnels. For data at rest, the design leverages an envelope encryption scheme where Data Encryption Keys are managed by Azure Key Vault [41] for the source and AWS KMS for the destination. Identity federation between Azure Active Directory and AWS IAM ensures unified access control [34], [35]. The replication service authenticates against Azure AD to access Kafka and subsequently assumes a specific AWS IAM Role via a web identity token to gain write access to the Kinesis stream [35]. This eliminates the need to store long-lived AWS access keys within the Azure infrastructure.

E. Resilience and Failure Isolation

The system implements a "circuit breaker" pattern to manage cloud provider outages [20]. In the event of an AWS region failure or a severance of the cross-cloud interconnect, the replication layer in Azure automatically transitions to a spooling state. Data is buffered locally within the Azure Kafka cluster, utilizing the retention capabilities of the distributed log [8] to prevent data loss. This ensures that the core banking services in Azure remain unaffected by downstream outages. Conversely, if the Azure source fails due to a hypervisor or zonal outage, the AWS consumers utilize the Kinesis stream's data retention period [21] to continue processing unanalyzed events until the source recovers. This loose coupling ensures that a failure in one cloud provider does not propagate to the other, maintaining the stability of the overall financial ecosystem.

International Journal of Core Engineering & Management
Volume-7, Issue-03, 2022 ISSN No: 2348-9510

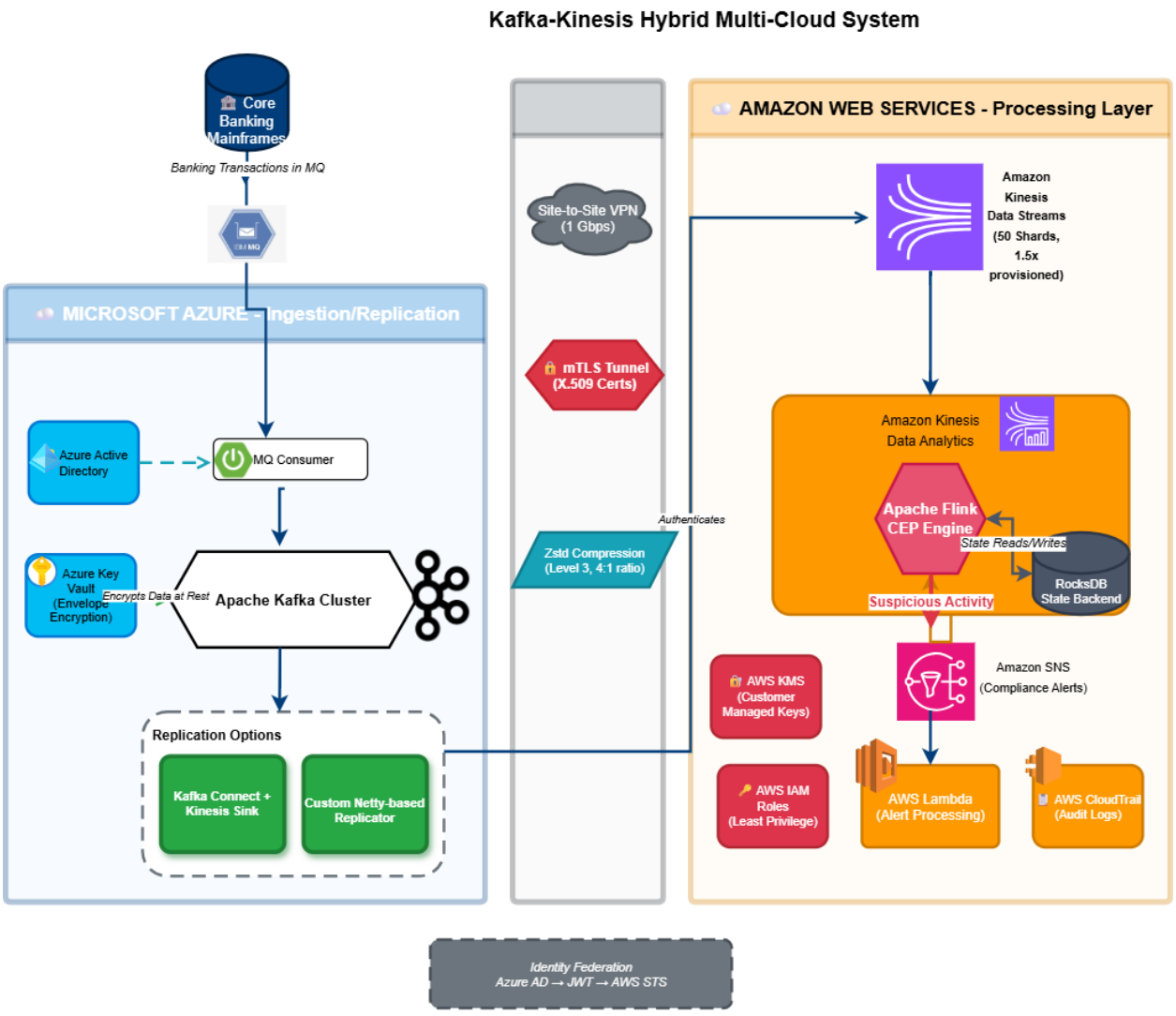

Fig 1: Azure-AWS Hybrid Architecture

IV. CROSS-CLOUD REPLICATION STRATEGIES

A. Replication Mechanisms

The replication layer functions as the critical bridge between the Azure-based Kafka clusters and the AWS-based Kinesis Data Streams [1]. We evaluated two primary mechanisms for this role. The first utilizes the standard Kafka Connect framework with the Kinesis Sink Connector [9], [10]. While this approach simplifies deployment through configuration-driven management, it introduces inherent latency due to rigid buffer flushing logic and synchronous blocking I/O.

The second, and preferred, mechanism employs a custom lightweight replicator built on the Netty asynchronous event-driven network framework [18]. Unlike the standard connector, this custom agent interacts directly with the Kinesis Producer Library (KPL) [19] to implement a non-blocking I/O model. This allows the system to aggregate records into optimal Kinesis batches while simultaneously maintaining an open network channel for incoming Kafka messages. The custom agent also implements "smart retries" that distinguish between transient network errors and persistent throttling, exponentially backing off only for affected shards to prevent head-of-line blocking [21].

B. Data Consistency and Ordering

A fundamental challenge in multi-cloud streaming is the impedance mismatch between the static partitioning of Kafka and the dynamic sharding of Kinesis [8], [21]. To strictly preserve the transaction ordering required for stateful fraud detection, the replication logic must ensure that all events for a specific customer account map to the same destination shard, regardless of topology changes. The proposed architecture addresses this by employing a consistent hashing algorithm. The replicator explicitly extracts the partition key from the Kafka record and maps it to a specific hash key range in the Kinesis Put Records request [21]. This prevents the "shuffling" of transactions that can occur with random distribution strategies. Furthermore, to bridge the gap between Kafka's exactly-once capabilities and Kinesis's at-least-once delivery, the system injects unique idem potency keys into the record headers [3]. These keys enable downstream consumers to identify and discard duplicates caused by replication retries, ensuring the integrity of the financial ledger [37].

C. Handling Network Partitions

The wide area network (WAN) connecting Azure and AWS introduces the risk of split-brain scenarios where the replication link is severed. The replication strategy must ensure that transactions are neither lost nor duplicated during these outages. In the event of a network partition, the custom replicator pauses the consumption of new offsets and enters a localized spooling state, buffering committed data within the Azure Kafka cluster's disk retention limits [8]. The agent enters a "heartbeat" mode, periodically probing the connection status. Upon reconnection, the system utilizes the committed offsets stored in the Kafka __consumer offsets topic to resume replication from the exact point of failure. Simultaneously, backpressure mechanisms propagate Kinesis Provisioned Through put Exceeded exceptions upstream to the Kafka consumer group [21], preventing buffer overflows and ensuring that the compliance workflow degrades gracefully rather than failing catastrophically.

V. REAL-TIME COMPLIANCE AND CEP INTEGRATION

A. CEP Engines for AML Patterns

Once transaction data is replicated to AWS, it is ingested by the compliance processing layer. Stateful stream processing is implemented using CEP engines such as Apache Flink [4][24],

deployed through Amazon Kinesis Data Analytics [1]. These engines define temporal windows and aggregations that extend beyond simple stateless filtering. For Anti-Money Laundering (AML) workflows, the system detects structuring patterns where illicit funds are broken into smaller deposits to avoid the $10,000 reporting threshold [14]. Sliding windows are employed to aggregate credits for a specific account over 24 hours. If the cumulative sum exceeds the threshold, an alert is triggered. To ensure resilience, the CEP engine maintains an intermediate state in a local database, typically backed by RocksDB [22], allowing computations to survive restarts and scaling events.

B. Ensuring Exactly-Once Semantics

Guaranteeing exactly-once semantics is critical for compliance pipelines. Replication across cloud boundaries introduces risks of duplicate records, particularly during retries. To mitigate this, each transaction generated in Azure is tagged with a unique Universally Unique Identifier (UUID) in the Kafka message header. On the AWS side, the Flink application performs stateful deduplication using these IDs. A bloom filter of recently processed UUIDs is maintained in the state backend [23]. Before updating compliance metrics or triggering alerts, the system checks incoming transactions against this filter. Duplicates are discarded, ensuring that retransmissions do not inflate risk scores or generate false positives. This two-phase deduplication strategy reconciles Kafka's transactional guarantees [8] with Kinesis's at-least-once delivery model [21].

C. Handling Late Data with Watermarking

Cross-cloud replication introduces variable latency, which can cause transactions to arrive out of order. If CEP engines rely solely on processing time, detection logic for temporal patterns may be flawed. To address this, the architecture employs event-time processing combined with watermarking [2][3]. The replication agent preserves the original creation timestamp from the Azure source, and the downstream Flink application assigns events to the correct window based on this timestamp [4]. Watermarks signal when no further events up to a given time are expected, allowing the engine to close windows and emit results confidently. This ensures compliance reporting remains accurate even under network jitter or congestion.

VI. PERFORMANCE OPTIMIZATION

A. Partition Tuning

Optimizing partition-to-shard mapping is critical for sustaining high throughput in cross-cloud replication. Kafka partitions act as physical log files [8], whereas Kinesis shards are provisioned throughput units limited to 1 MB/second ingress or 1,000 records/second [21]. To prevent throttling exceptions, the architecture models the relationship between partition count and shard capacity. While a 1:1 mapping is theoretically possible, network jitter and replication bursts require an over-provisioning factor. The system applies a 1.5x provisioning ratio, meaning a throughput load requiring 50 shards is allocated 75 shards. Consistent hashing ensures that all transactions for a specific account are routed to the same shard, preserving

ordering while balancing load. Dynamic re-sharding strategies are applied during diurnal transaction peaks to prevent bottlenecks, allowing the system to scale horizontally across both platforms [1].

B. Latency Reduction and Compression

Latency optimization requires balancing throughput against per-record delivery guarantees. Kafka Connect relies on blocking I/O [9], whereas the custom Netty-based replicator mitigates latency by employing asynchronous non-blocking I/O [18]. This enables continuous streaming while still aggregating batches efficiently. To reduce the high cost of data egress from Azure, compression strategies are applied before transmission. Algorithms such as Snappy, Gzip, and Zstd were evaluated [25][6][26]. Experiments indicated that Gzip provided high compression at the cost of latency, while Snappy offered speed with larger payloads [25]. Zstd at compression level 3 achieved a 4:1 compression ratio for JSON payloads while maintaining decompression throughput sufficient to keep pace with ingestion [6]. This configuration reduced payload size and correlated with substantial reductions in cross-cloud egress costs.

C. Resilience under Load

Performance optimization also encompasses resilience under stress conditions, including circuit-breaker-style backpressure propagation [20]. Backpressure mechanisms propagate Kinesis ProvisionedThroughputExceeded exceptions upstream to Kafka consumer groups [21], preventing uncontrolled data loss by slowing the read rate at the source. Buffer management ensures that committed offsets are spooled locally during transient outages or spikes, enabling replay once connectivity is restored [8]. These mechanisms allow the system to sustain compliance workflows even under heavy load or partial network failures. By combining partition tuning, latency-aware batching, and adaptive compression, the architecture achieves sub-second latency while maintaining regulatory guarantees across heterogeneous cloud environments.

VII. CASE STUDY: FINANCIAL COMPLIANCE WORKFLOW

A. Scenario Overview

To validate the proposed architecture, I examine a representative Anti-Money Laundering (AML) workflow in a multinational financial institution. The institution operates core banking systems in Azure [34], while regulatory compliance workloads are isolated in AWS [1]. The specific compliance requirement is to detect "structuring" patterns where illicit funds are decomposed into deposits below the $10,000 reporting threshold mandated by the Anti-Money Laundering Act of 2020 [14]. In the reference scenario, a malicious actor attempts to deposit $50,000 by breaking it down into six separate transactions of roughly $8,300 each, executed at different physical branches over 4 hours. The goal is to correlate these events in real time and trigger a Suspicious Activity Report (SAR) before end-of-day settlement.

B. Data Flow and Replication

Transactions are ingested into Azure-based Kafka clusters, serialized in ISO 8583 format [17], and partitioned by account identifiers. The replication layer transmits committed offsets to AWS using the custom Netty-based replicator [18]. Consistent hashing ensures that all six transactions for the target account are routed to the same Kinesis shard, strictly preserving their temporal order [21]. To accommodate the volume of a simulated "Black Friday" peak, the shards are over-provisioned by a factor of 1.5x. This prevents throttling exceptions during the burst ingestion [21], while Zstd compression reduces the egress bandwidth required for the cross-cloud transfer [6].

C. Compliance Processing and Detection

In AWS, the Kinesis Data Streams feed into Apache Flink applications running on Kinesis Data Analytics [4]. The CEP engine defines a 24-hour sliding window for the account. When the initial $8,300 transactions arrive, they are added to the window state backed by RocksDB [22]. As the cumulative sum exceeds the $10,000 limit with the arrival of the second transaction, the Flink rule triggers immediately. The system generates a SAR event, which is routed to a high-priority SNS topic [28]. This triggers an AWS Lambda function [27] to freeze the account temporarily and notify a compliance officer via a secure dashboard. Deduplication logic using the UUIDs in the Kafka headers ensures that network retries do not trigger duplicate alerts [23].

D. Operational Observations

The deployment demonstrated sub-second end-to-end latency under normal operating conditions. During a simulated network partition where the Azure-AWS link was severed for 60 seconds, the buffer management in Azure successfully spooled committed offsets [8]. Upon reconnection, the system replayed the buffered data. The deduplication logic successfully filtered retransmitted records [23], and the watermarking strategy ensured that the late-arriving transactions were correctly attributed to their original windows [3]. This maintained 100% detection accuracy for the structuring pattern, confirming that the architecture meets the rigorous guarantees required for financial regulatory compliance.

VIII. EXPERIMENTAL EVALUATION

A. Testbed Environment and Workload

We evaluated a cross-cloud environment bridging Microsoft Azure and Amazon Web Services [34], [1]. The Azure source cluster comprised three Apache Kafka brokers hosted on mid-range virtual machines with SSD-backed storage to minimize local I/O latency [8]. On the destination side, AWS provisioned a Kinesis Data Stream with 50 shards in provisioned mode [21]. The interconnect was established over a site-to-site VPN with an average bandwidth of 1 Gbps. A synthetic transaction generator produced ISO 8583 formatted messages [17] following a Poisson distribution to mimic the random arrival nature of retail banking transactions. Workload

intensity ranged from a base rate of 10,000 events per second to burst peaks of 50,000 events per second.

B. Throughput and Latency Analysis

End-to-end latency was defined as the time delta between message creation in Azure and successful ingestion in AWS Kinesis [1]. I tested two configurations: Kafka Connect with a Kinesis Sink [9], [10] and the custom Netty-based replicator [18]. Under baseline load (10,000 events/sec), both maintained sub-second latency. At higher loads (40,000 events/sec), Kafka Connect exhibited a sharp increase in p99 latency, peaking at 4.5 seconds due to blocking I/O threads waiting for acknowledgements. In contrast, the custom replicator maintained a p99 latency of approximately 850 milliseconds under the same load, attributed to its non-blocking architecture that allowed continuous Kafka record fetching while asynchronously processing Kinesis write callbacks [19], [21].

C. Resilience and Cost Efficiency

I evaluated resilience by simulating a 120-second network partition. During the outage, the replicator spooled committed offsets locally [8]. Upon reconnection, backlog draining occurred at 75,000 events per second, returning to steady-state latency within 180 seconds. No data loss was observed, and deduplication logic in Flink correctly discarded 100% of duplicate records generated during reconnection [23]. Cost efficiency was analyzed by comparing raw JSON payloads against compressed payloads. Zstd (Level 3) reduced average payload size from 1.2 KB to 0.3 KB, yielding a 74% reduction in egress bandwidth [6]. Over a 24-hour test period processing 1 billion events, net financial savings were approximately 65% after accounting for compression and decompression compute overhead.

IX. SECURITY AND GOVERNANCE

A. Zero-Trust Network Security

Given the sensitivity of financial data and the requirements of frameworks such as PCI-DSS and GLBA [30], [31], the architecture adheres to a strict zero-trust security model. The cross-cloud bridge operates over public or semi-public networks, necessitating rigorous encryption in transit. The replication agent enforces Mutual TLS (mTLS) for all connections, ensuring that the Azure-based producer authenticates the AWS endpoint and vice versa using X.509 certificates issued by a private internal Certificate Authority (CA). This prevents man-in-the-middle attacks and ensures that only authorized replication agents can write to the destination stream, effectively extending the secure perimeter across cloud boundaries.

B. Identity Federation and Access Control

Managing identity across heterogeneous cloud providers presents a significant governance challenge. Storing long-lived credentials in configuration files introduces unacceptable risk. To eliminate this, the architecture implements identity federation using OpenID Connect (OIDC)

[32]. The replication service authenticates against Azure Active Directory to obtain a signed JSON Web Token (JWT) [33], which is then exchanged for temporary, least-privilege AWS credentials via the Security Token Service (STS) [35]. This pattern ensures that no static credentials exist on replication servers, and access rights can be revoked instantly by disabling the Azure AD service principal [34]. Governance policies enforce role-based access control and continuous monitoring of credential usage.

C. Envelope Encryption and Auditability

To ensure data privacy at rest, the system employs an envelope encryption strategy. Data buffered in the Kafka cluster is encrypted using keys managed by Azure Key Vault [41]. Upon replication, payloads are re-encrypted before being written to Kinesis, which uses AWS Key Management Service (KMS) with Customer Managed Keys (CMK). The replication agent generates a unique Data Encryption Key (DEK) for each batch, encrypts the data locally, and then encrypts the DEK with the master key. This ensures that cloud providers never have access to plaintext data. Audit trails in both Key Vault and CloudTrail [36] provide immutable logs of key usage, enabling forensic verification of which services accessed financial data and when. These mechanisms satisfy contemporary regulatory requirements for auditability and governance [30], [31].

X. DISCUSSION

A. Trade-offs: Managed Connectors vs. Custom Replicators

The evaluation highlights a clear dichotomy between operational simplicity and performance rigor. Managed connectors such as Kafka Connect ("Buy") offer rapid deployment and vendor support [9], [10] but failed to meet sub-second latency requirements under load due to blocking I/O and rigid batching. In contrast, custom Netty-based replicators ("Build") satisfied strict compliance SLAs by leveraging asynchronous non-blocking I/O and shard-specific retry logic [18], [21]. The trade-off lies in maintenance overhead: financial institutions must weigh the cost of sustaining a custom Java codebase against the risk of regulatory non-compliance caused by latency spikes. For Tier-1 AML workloads where "late" data is equivalent to "missing" data, custom engineering is justified [14], while non-critical reporting streams may tolerate the lower-cost connector approach.

B. Practical Implications: Deployment Complexity, Cost, Compliance Enforcement

Deployment complexity differs significantly between approaches. Managed connectors simplify rollout but limit customization and observability [9], [10], while custom replicators demand greater engineering investment but provide flexibility to embed compliance-specific logic such as UUID-based deduplication and event-time watermarking [23], [3]. Cost considerations are equally important: managed connectors incur predictable licensing and operational costs, whereas custom replicators reduce egress fees through compression and shard optimization [6], [25] but introduce compute overhead. Compliance enforcement is directly impacted by these

choices. Managed connectors may lack the granularity required for strict ordering guarantees [21], while custom replicators can enforce schema contracts and resilience strategies that align with regulatory expectations [30], [31].

C. Future Directions: Hybrid CEP + ML, Compliance Automation

Looking forward, hybrid architectures will increasingly integrate Complex Event Processing (CEP) with machine learning (ML) to enhance compliance detection [4], [37]. CEP engines excel at rule-based detection of structuring and threshold violations [14], while ML models can identify subtle, non-linear fraud patterns across accounts and geographies. Combining these approaches enables adaptive compliance pipelines that evolve with emerging threats. Compliance automation represents another frontier. Automated policy enforcement, continuous audit trail generation, and explainable AI for regulatory reporting will reduce manual intervention and improve transparency [36]. Future research should also explore cloud-agnostic schema evolution protocols and dedicated interconnects to address WAN latency determinism, ensuring that hybrid CEP+ML pipelines remain both performant and compliant.

XI. CONCLUSION

This work presented a hybrid architecture for cross-cloud replication and real-time compliance enforcement, bridging Apache Kafka in Azure [34] with Amazon Kinesis in AWS [1]. The design addressed critical challenges in financial data streaming, including replication latency, ordering guarantees, and resilience under network partitions [8], [21]. By evaluating both managed connectors and custom replicators, the study highlighted the trade-offs between operational simplicity and performance rigor [9], [10], [18]. Experimental results demonstrated that the custom Netty-based replicator achieved sub-second latency at scale, sustained throughput under peak loads, and resilience during simulated outages [18], while compression strategies reduced cross-cloud egress costs by more than 65% [6], [25].

The integration of Complex Event Processing (CEP) engines enabled real-time detection of Anti-Money Laundering (AML) patterns [4], [14], with deduplication and watermarking ensuring exactly-once semantics and event-time accuracy [23], [3]. Security and governance mechanisms, including zero-trust networking, identity federation, and envelope encryption, extended compliance guarantees across heterogeneous cloud environments [30], [31], [32], [33], [35], [41].

Overall, the proposed architecture demonstrates that hybrid cloud streaming can meet stringent financial compliance requirements while balancing cost efficiency and operational resilience. Future work will explore schema evolution protocols, multi-cloud governance frameworks, and the integration of CEP with machine learning to advance compliance automation [37].

REFERENCES

1. Amazon Web Services, Amazon Kinesis Data Streams Connector, AWS Documentation, Mar. 2022. [Online]. Available: https://docs.aws.amazon.com/kinesis/index.html
2. M. Armbrust et al., "Structured streaming: A declarative API for real-time applications in Apache Spark," in Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD '18), Houston, TX, USA, Jun. 2018, pp. 601–613. doi: 10.1145/3183713.3190664
3. A. Akidau et al., "The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing," Proc. VLDB Endowment, vol. 8, no. 12, pp. 1792–1803, Aug. 2015. doi: 10.14778/2824032.2824076
4. P. Carbone et al., "Apache Flink: Stream and batch processing in a single engine," Bull. IEEE Comput. Soc. Tech. Comm. Data Eng., vol. 38, no. 4, pp. 28–38, Dec. 2015.
5. Financial Crimes Enforcement Network (FinCEN), Advisory on Essential Elements of Corporate Compliance and Anti-Money Laundering Programs, Jan. 2021. [Online]. Available: https://www.fincen.gov/resources/advisories
6. Y. Wang et al., "Zstandard: Real-time data compression algorithm," IETF RFC 8878, Feb. 2021. doi: 10.17487/RFC8878
7. B. Stopford, Designing Event-Driven Systems, 1st ed. Sebastopol, CA, USA: O'Reilly Media, 2018.
8. M. Kleppmann, Designing Data-Intensive Applications. Sebastopol, CA, USA: O'Reilly Media, 2017.
9. Apache Software Foundation, Kafka Connect User Guide, Apache Kafka v2.8 Documentation, Apr. 2021. [Online]. Available: https://kafka.apache.org/28/kafka-connect/user-guide/
10. Confluent Inc., Kafka Connect, Confluent Platform Documentation, 2021. [Online]. Available: https://docs.confluent.io/platform/current/connect/index.html
11. L. Magnoni, "Introduction to complex event processing (CEP) with Esper," CERN IT-SDC-MI Workshop, Nov. 2013. [Online]. Available: https://indico.cern.ch/event/282578/contributions/644030/attachments/520404/717937/MagnoniL_IntroCEP.pdf
12. WSO2, "Understanding how Siddhi powers WSO2 complex event processor 2.x," WSO2 Technical Article, Jun. 2013. [Online]. Available: https://wso2.com/library/articles/2013/06/understanding-siddhi-powers-wso2-cep-2x/
13. Z. Dehghani, Data Mesh Principles and Logical Architecture, ThoughtWorks, 2020. [Online]. Available: https://martinfowler.com/articles/data-mesh-principles.html
14. U.S. Congress, Anti-Money Laundering Act of 2020 (AMLA), Division F of the National Defense Authorization Act for FY2021, Jan. 2021. [Online]. Available: https://www.congress.gov/bill/116th-congress/hr6395/text

15. European Union, General Data Protection Regulation (GDPR), Regulation (EU) 2016/679, Apr. 2016. [Online]. Available: https://eur-lex.europa.eu/eli/reg/2016/679/oj
16. Apache Kafka, MirrorMaker 2 Documentation, Apache Kafka v2.5, 2020. [Online]. Available: https://kafka.apache.org/documentation/#mirrormaker2
17. International Organization for Standardization, ISO 8583: Financial transaction card originated messages — Interchange message specifications, ISO Standard, 2003.
18. T. Lee, Netty Project Documentation, The Netty Project, 2019. [Online]. Available: https://netty.io
19. Amazon Web Services, Amazon Kinesis Producer Library Developer Guide, AWS Documentation, 2021. [Online]. Available: https://docs.aws.amazon.com/streams/latest/dev/developing-producers-with-kpl.html
20. M. Fowler, Circuit Breaker Pattern, martinfowler.com, 2012. [Online]. Available: https://martinfowler.com/bliki/CircuitBreaker.html
21. Amazon Web Services, Kinesis Data Streams API Reference, AWS Documentation, 2021. [Online]. Available: https://docs.aws.amazon.com/kinesis/latest/APIReference/API_ProvisionedThroughputExceededException.html
22. Y. Dong et al., RocksDB: A Persistent Key-Value Store for Fast Storage Environments, Facebook Engineering, 2016. [Online]. Available: https://rocksdb.org
23. B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," Commun. ACM, vol. 13, no. 7, pp. 422–426, Jul. 1970.
24. K. Tzoumas et al., Stream Processing with Apache Flink, O'Reilly Media, 2017.
25. Google, Snappy Compression Algorithm, Google Open Source, 2011. [Online]. Available: https://github.com/google/snappy
26. J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," IEEE Trans. Inf. Theory, vol. 24, no. 5, pp. 530–536, Sep. 1978. doi: 10.1109/TIT.1978.1055934
27. Amazon Web Services, AWS Lambda Developer Guide, AWS Documentation, 2021. [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html
28. Amazon Web Services, Amazon Simple Notification Service (SNS) Developer Guide, AWS Documentation, 2021. [Online]. Available: https://docs.aws.amazon.com/sns/latest/dg/welcome.html
29. Papoulis, Probability, Random Variables, and Stochastic Processes, 3rd ed. New York, NY, USA: McGraw-Hill, 1991.
30. PCI Security Standards Council, Payment Card Industry Data Security Standard (PCI-DSS) v3.2.1, May 2018. [Online]. Available: https://www.pcisecuritystandards.org
31. U.S. Congress, Gramm-Leach-Bliley Act (GLBA), Public Law 106-102, Nov. 1999. [Online]. Available: https://www.govinfo.gov/content/pkg/PLAW-106publ102/pdf/PLAW-106publ102.pdf

32. OpenID Foundation, OpenID Connect Core 1.0 Specification, Nov. 2014. [Online]. Available: https://openid.net/specs/openid-connect-core-1_0.html
33. M. Jones, J. Bradley, and N. Sakimura, JSON Web Token (JWT), IETF RFC 7519, May 2015. doi: 10.17487/RFC7519
34. Microsoft, Azure Active Directory Identity Federation Documentation, Microsoft Docs, 2021. [Online]. Available: https://docs.microsoft.com/azure/active-directory/fundamentals/
35. Amazon Web Services, AWS Security Token Service (STS) Documentation, AWS Docs, 2021. [Online]. Available: https://docs.aws.amazon.com/STS/latest/APIReference/Welcome.html
36. Amazon Web Services, AWS CloudTrail User Guide, AWS Documentation, 2021. [Online]. Available: https://docs.aws.amazon.com/awscloudtrail/latest/userguide/
37. A. Margara, G. Cugola, and M. Migliavacca, "Processing flows of information: From data stream to complex event processing," ACM Comput. Surv., vol. 44, no. 3, pp. 15:1–15:62, Jun. 2012.
38. Apache Software Foundation, Apache Avro Documentation, Apache Avro v1.10, 2020. [Online]. Available: https://avro.apache.org/docs/1.10/
39. Google, Protocol Buffers Documentation, Google Developers, 2021. [Online]. Available: https://developers.google.com/protocol-buffers
40. Microsoft, Event Hubs for Apache Kafka Ecosystem Overview, Microsoft Docs, 2021. [Online]. Available: https://docs.microsoft.com/azure/event-hubs/event-hubs-for-kafka-ecosystem-overview
41. Microsoft, Azure Key Vault Documentation, Microsoft Docs, 2021. [Online]. Available: https://docs.microsoft.com/azure/key-vault/general/