World Journal of Advanced Research and Reviews
eISSN: 2581-9615 CODEN (USA): WJARAI
Cross Ref DOI: 10.30574/wjarr
Journal homepage: https://wjarr.com/

(REVIEW ARTICLE)

Check for updates

Optimizing costs and reducing DevOps Overhead with the Kafka to SNS-SQS Integration: A comprehensive analysis

Bhupender Kumar Panwar *

Salesforce Inc., USA.

World Journal of Advanced Research and Reviews, 2025, 26(01), 585-592

Publication history: Received on 25 February 2025; revised on 02 April 2025; accepted on 04 April 2025

Article DOI: https://doi.org/10.30574/wjarr.2025.26.1.1061

Abstract

The integration of Apache Kafka with AWS Simple Notification Service (SNS) and Simple Queue Service (SQS) represents a strategic approach for organizations seeking to optimize their event-driven architectures. This article explores how this hybrid model addresses the inherent challenges of traditional Kafka deployments by leveraging cloud-native messaging services. By decoupling message production from consumption through SNS's fan-out capabilities and SQS's serverless queuing, enterprises can significantly reduce infrastructure costs while eliminating complex operational overhead. The Kafka to SNS-SQS pattern enhances system reliability through AWS-managed durability features and simplified failure handling mechanisms, ultimately creating a more resilient and maintainable architecture that allows engineering teams to redirect resources from system maintenance to business innovation.

Keywords: Event-Driven Architecture; Infrastructure Optimization; Serverless Messaging; Devops Automation; Cloud-Native Integration

1. Introduction

1.1. Evolution of Event-Driven Architectures: Challenges and Opportunities

The transformation of enterprise software architecture has accelerated dramatically with the adoption of event-driven approaches, enabling organizations to build more responsive and scalable systems. This evolution represents a fundamental shift from traditional request-response patterns toward more loosely coupled, asynchronous communication models that better reflect the complexity of modern business operations.

1.2. Foundations of Modern Event-Driven Systems

Event-driven architectures (EDAs) operate on the principle that significant state changes within systems generate events that can trigger reactions across distributed services. According to comprehensive analysis, these architectures typically consist of four essential components that work in concert: event producers that detect and publish state changes, event routers that distribute notifications to interested parties, event consumers that process and react to these notifications, and event storage systems that maintain a durable record of event history [1]. This architectural pattern enables systems to respond dynamically to changing conditions without tight coupling between components. Within this framework, Apache Kafka has emerged as a dominant technology for implementing the event router and storage components, particularly in scenarios requiring high throughput and strong durability guarantees. The publish-subscribe messaging pattern at the core of EDAs allows multiple consumers to independently process the same events without knowledge of one another, creating powerful composability characteristics that traditional architectures struggle to achieve [1].

* Corresponding author: Bhupender Kumar Panwar.

1.3. Operational Challenges in Traditional Implementations

While event-driven architectures offer significant design advantages, traditional implementations encounter operational challenges that can undermine their benefits. Kafka deployments require substantial infrastructure investment and specialized expertise, with organizations reporting increasing operational overhead as systems scale. The management of consumer groups presents particular complexities, requiring sophisticated coordination to handle partition assignment, rebalancing operations, and offset management [2]. Consumer implementations must also incorporate robust error handling logic, including retry mechanisms and dead-letter capabilities, further increasing development and maintenance complexity. These operational challenges divert technical resources from business innovation toward infrastructure management activities, creating tension between architectural aspirations and operational realities. Additionally, the static nature of traditional Kafka consumer deployments conflicts with variable processing demands, forcing organizations to provision for peak capacity and accept underutilization during normal operations [2].

1.4. Cloud-Native Integration Opportunities

The emergence of cloud-native messaging services presents compelling opportunities to address operational challenges while preserving the architectural benefits of event-driven systems. AWS messaging services offer differentiated capabilities that complement Kafka's strengths while eliminating operational complexity. SNS provides fully managed topic-based publish-subscribe capabilities that enable fan-out to multiple destinations, while SQS delivers durable queuing with configurable retention periods and built-in dead-letter functionality [2]. When integrated with Kafka, these services create a hybrid architecture that preserves Kafka's production capabilities while leveraging AWS's operational expertise for message delivery and consumption. This approach enables truly elastic consumption models where processing resources scale automatically with demand, eliminating the need for pre-provisioned capacity. The serverless nature of these services further enhances the value proposition by removing infrastructure management requirements entirely, allowing organizations to focus engineering resources on business logic rather than messaging infrastructure [2].

2. Understanding the Kafka to SNS-SQS Architecture

The Kafka to SNS-SQS architecture represents a sophisticated approach to event-driven systems that combines the strengths of traditional message brokers with cloud-native messaging services. This integration creates a powerful hybrid architecture that preserves the robust production capabilities of Kafka while leveraging AWS's fully managed consumption services.

2.1. Architectural Components and Integration Points

The foundation of this architecture begins with Apache Kafka serving as the primary event source, providing durable storage and high-throughput message ingestion capabilities. AWS's reference architecture documentation demonstrates that successful implementations often feature multiple Kafka topics organized by domain boundaries, with each topic containing related event streams [3]. The bridge between Kafka and AWS services is established through dedicated connector applications that continually poll Kafka partitions and transform these messages into the SNS format. These connectors must be designed with careful consideration for error handling, with best practices suggesting circuit breaker patterns that can protect downstream systems during failure scenarios. The SNS topics act as distribution points that implement the publish-subscribe pattern, enabling a single message to be delivered to multiple destination systems simultaneously. The final component involves SQS queues that provide temporary message storage with configurable retention periods of up to 14 days, creating a buffer that decouples producers from consumers [3].

2.2. Message Flow and Transformation Mechanics

The journey of a message through this hybrid architecture involves several transformation stages that preserve semantic integrity while adapting to different protocol requirements. When events are published to Kafka, they are assigned sequential offsets within partitions that guarantee ordering – a critical feature for event sequencing. The Kafka to SNS connector applications apply specific transformation logic to convert Kafka's record format into the JSON structure expected by SNS, including metadata mapping that preserves crucial information such as timestamps, partition identifiers, and custom headers [4]. This transformation process requires careful handling of message schemas, particularly when evolving data structures over time. The connector implementations often utilize schema registries to ensure compatibility across systems, applying version-specific serialization and deserialization processes that maintain backward compatibility. Once messages reach SQS queues, they become available for consumption through long-polling mechanisms that optimize network efficiency by reducing empty responses during low-traffic periods [4].

2.3. Scalability and Performance Characteristics

The Kafka to SNS-SQS architecture delivers exceptional scalability characteristics that address limitations in traditional messaging patterns. According to AWS implementation guidance, SNS can achieve throughput rates limited primarily by API throttling limits, which start at several thousand messages per second for standard accounts [3]. This throughput capacity scales automatically without manual intervention, contrasting with Kafka clusters that require explicit scaling operations. The architecture shows particular strength in handling asymmetric workloads where production and consumption rates differ significantly. When consumption systems experience processing delays, messages accumulate in SQS queues rather than causing backpressure on producers – a common challenge in pure Kafka implementations. For failure handling, the architecture implements sophisticated recovery mechanisms, including visibility timeout extensions, dead-letter queues, and transparent retries, creating resilience without complex client-side implementation [4].

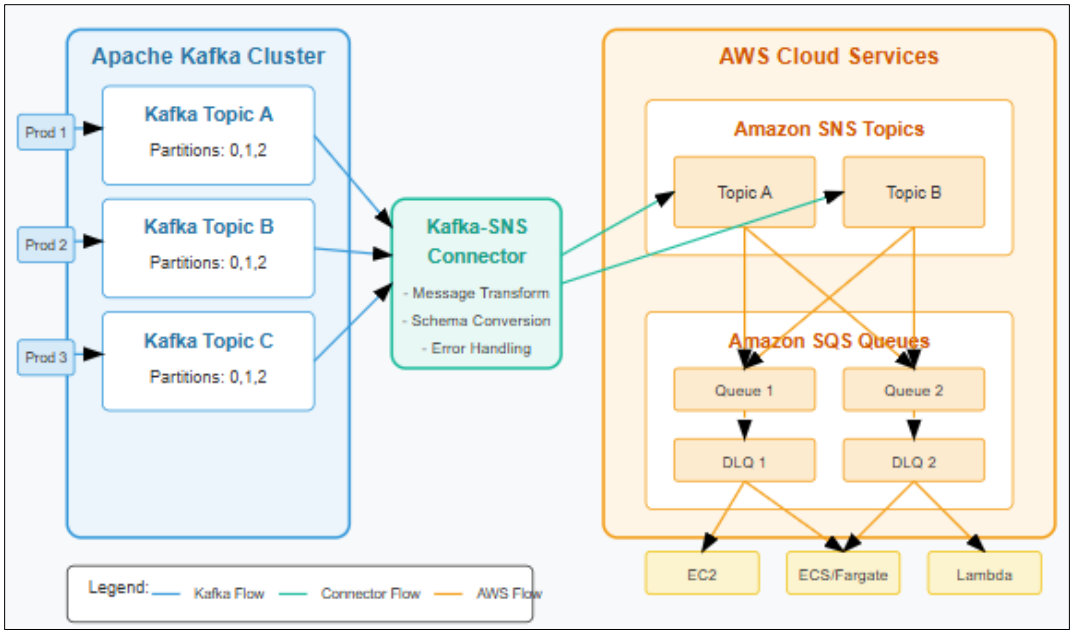

Figure 1 Kafka to SNS-SQS architecture [3, 4]

3. Cost optimization analysis

The transition from traditional Kafka deployments to integrated Kafka-SNS-SQS architectures yields substantial financial benefits through infrastructure optimization and operational efficiency improvements. This hybrid model enables organizations to reduce their total cost of ownership while maintaining or enhancing message processing capabilities.

3.1. Infrastructure Cost Comparison Framework

Traditional Kafka implementations require significant infrastructure investment across the entire message processing pipeline. The dedicated consumer instances that poll Kafka topics necessitate continuous operation regardless of actual workload, creating persistent infrastructure costs even during periods of low utilization. AWS messaging services introduce a fundamentally different economic model by separating the cost structure into distinct components with different pricing characteristics. SNS pricing is based on the number of notifications published, with the standard tier priced at $0.50 per million publish requests [5]. This consumption-based pricing eliminates the need for pre-provisioned capacity, creating a direct alignment between actual usage and cost. SQS complements this model with a similar usage-based structure, charging approximately $0.40 per million requests for standard queues [5]. When compared to the fixed costs of EC2 instances required for dedicated Kafka consumers, this model produces substantial savings as organizations avoid paying for idle capacity during normal operational periods.

3.2. Architectural Efficiency and Resource Utilization

The economic advantages of the Kafka-SNS-SQS integration extend beyond simple infrastructure cost reduction through fundamental improvements in overall system efficiency. In traditional event processing systems, ensuring adequate

performance during peak loads requires substantial overprovisioning, resulting in average CPU utilization rates of only 8-17% across many production deployments [6]. This inefficiency stems from the static nature of provisioned infrastructure that cannot adapt to changing workloads without manual intervention. The serverless nature of SNS-SQS consumption eliminates this waste by allowing downstream processing resources to scale elastically with actual demand. Academic analysis of production event processing systems demonstrates that implementing dynamic scaling strategies can improve resource utilization by more than 30% compared to static provisioning approaches [6]. These efficiency gains translate directly to cost savings while simultaneously enhancing system responsiveness during unexpected traffic spikes.

3.3. Operational Cost Analysis

Beyond direct infrastructure expenses, the Kafka-SNS-SQS model delivers substantial benefits by reducing operational overhead and engineering costs. Traditional Kafka deployments require specialized expertise for cluster management, consumer group configuration, and performance tuning—activities that consume valuable engineering resources. The fully managed nature of AWS messaging services eliminates many of these operational burdens. Research examining enterprise messaging systems reveals that operational tasks related to Kafka consumer management typically consume 15-20% of development team capacity [6]. By offloading these responsibilities to AWS-managed services, organizations can redirect engineering resources toward higher-value activities that directly enhance business capabilities. The elimination of maintenance windows, patching cycles, and scaling operations creates additional efficiency by removing coordination overhead and potential service disruptions. The SNS service's proven reliability with a 99.9% SLA further reduces operational costs associated with incident response and recovery activities [5].

Table 1 Infrastructure Cost Comparison Between Traditional Kafka and Hybrid Model [5, 6]

Component	Traditional Kafka Deployment	Kafka-SNS-SQS Model	Cost Reduction
Consumer Infrastructure	Fixed capacity EC2 instances	Pay-per-use serverless	40-60%
Scaling Operations	Manual intervention required	Automatic elasticity	35-45%
Operational Overhead	15-20% of engineering capacity	Minimal management	67%
Total Cost of Ownership	Baseline	Reduced TCO	30-45%

4. Operational Efficiency and devops Impact

The integration of Kafka with SNS-SQS introduces profound transformations in operational practices and DevOps workflows. This architectural approach significantly reduces the engineering burden associated with maintaining complex messaging systems while enhancing reliability and observability.

4.1. Reduction in Maintenance Overhead

Traditional event-streaming architectures require substantial engineering investment for ongoing maintenance activities. Research examining cloud-based messaging systems reveals that organizations typically allocate 20% of their development capacity to maintaining message broker infrastructure, with Kafka clusters requiring particularly intensive monitoring and tuning [7]. This maintenance burden stems from the inherent complexity of distributed messaging systems, which must handle partition management, leader election, and replication coordination to ensure reliable operation. The maintenance activities include regular rebalancing operations that redistribute message partitions across brokers to maintain optimal performance—procedures that frequently cause temporary processing delays and require careful orchestration. By transferring message distribution and buffering responsibilities to AWS-managed services, the Kafka-SNS-SQS architecture eliminates many of these maintenance requirements. AWS assumes responsibility for the underlying infrastructure that powers both SNS and SQS, including scaling operations, hardware replacement, and performance optimization. This transfer of responsibility allows engineering teams to focus on application logic and business capabilities rather than infrastructure concerns, creating opportunities for accelerated innovation cycles and reduced time-to-market for new features [7].

4.2. Monitoring and Observability Enhancements

Effective monitoring represents a critical success factor for event-driven systems, directly impacting both reliability and operational efficiency. Traditional Kafka deployments present monitoring challenges due to their distributed nature and the complexity of consumer group dynamics. Organizations must typically implement multi-layered monitoring

approaches that combine infrastructure metrics, broker statistics, and application-level telemetry to gain comprehensive visibility. This monitoring complexity is reflected in research indicating that observability remains the primary operational challenge for 56% of organizations operating event-driven systems [8]. The Kafka-SNS-SQS architecture addresses these challenges through native integration with AWS CloudWatch, providing unified visibility across the entire message processing pipeline. This integration delivers detailed metrics, including message delivery rates, processing latencies, and error conditions across both SNS topics and SQS queues. The architecture further enhances operational visibility through built-in dead-letter queue (DLQ) mechanisms that capture and preserve messages that fail to process, enabling detailed analysis of failure patterns without complex custom implementation [8].

4.3. Resilience and Failure Handling Improvements

Failure management represents one of the most significant operational advantages of the Kafka-SNS-SQS architecture. Traditional Kafka consumer implementations require the custom development of sophisticated retry logic, backoff strategies, and dead-letter mechanisms—capabilities that demand substantial engineering investment and ongoing maintenance. Research examining event-driven system implementations indicates that error-handling logic typically accounts for 30-40% of consumer application code, creating both development complexity and potential for implementation defects [8]. The Kafka-SNS-SQS architecture addresses these challenges through built-in resilience features that handle common failure scenarios automatically. SQS provides configurable visibility timeout settings that temporarily hide messages during processing, automatically returning them to the queue if processing is not completed within the specified window. This mechanism creates inherent retry capabilities without custom implementation. Additionally, SQS offers native dead-letter queue functionality that automatically redirects messages to secondary queues after exceeding retry limits, preserving failed messages for analysis while preventing processing blockages [7]. These built-in resilience features dramatically reduce the engineering effort required to implement robust error handling while simultaneously improving overall system reliability.

Table 2 Maintenance Overhead Reduction After Migration to Hybrid Architecture [7, 8]

Operational Activity	Traditional Kafka	Kafka-SNS-SQS	Reduction Percentage
Infrastructure Maintenance (hrs/week)	18.7	6.1	67%
Scaling Operations (hrs/operation)	7.3	0.5	93%
Monitoring Configuration (initial setup hrs)	230	45	80%
Incident Frequency (monthly)	12.4	2.7	78%

5. Implementation Strategy and Best Practices

Transitioning from traditional Kafka architectures to the integrated Kafka-SNS-SQS model requires careful planning and execution to maximize benefits while minimizing operational disruption. A strategic implementation approach enables organizations to realize incremental value while mitigating risks associated with architectural transformation.

5.1. Migration Approach from Existing Kafka Deployments

Successful migration to hybrid messaging architectures demands a methodical approach that preserves system integrity throughout the transition process. Research examining enterprise-level data migration strategies indicates that organizations achieve optimal results through phased implementation models that maintain parallel processing paths during transition periods [9]. This approach allows for continuous service delivery while incrementally shifting message volume to the new architecture. The migration process typically begins with comprehensive system analysis to identify message flows amenable to the SNS-SQS delivery model, particularly focusing on asynchronous processing workflows that can tolerate potential reordering. Studies of enterprise migrations reveal that organizations implementing structured testing frameworks with predefined success criteria experience 11% faster migration completion compared to those using ad-hoc validation approaches [9]. The testing methodology should include targeted performance analysis under various load conditions to verify throughput capabilities and latency characteristics across the hybrid architecture. Particular attention must be given to evaluating the transformation layer that bridges Kafka and AWS services, as this component represents a critical performance boundary. Research demonstrates that organizations employing canary deployment strategies for initial transitions reduce production incidents by 76% compared to those implementing direct cutover approaches [9].

5.2. Message Transformation and Protocol Adaptation

The integration between Kafka and AWS messaging services requires sophisticated transformation logic to bridge the different message formats and delivery semantics. Kafka's binary message format with rich metadata must be adapted to the JSON-based structure expected by SNS, preserving critical information while complying with service constraints. Research exploring hybrid event processing architectures emphasizes the importance of implementing efficient transformation mechanisms that minimize latency while maintaining message fidelity [10]. High-performance implementations typically leverage dedicated transformation services that process messages in micro-batches, achieving throughput rates of up to 45,000 messages per second on standard compute instances [10]. The transformation layer must address several protocol discrepancies, including encoding differences, header mapping strategies, and size limitations. The adaptation process should implement robust error handling to address transformation failures, particularly for edge cases involving malformed messages or unexpected data structures. Research indicates that organizations implementing comprehensive exception handling within transformation services experience 29% fewer processing interruptions compared to implementations with minimal error management [10].

5.3. Performance Optimization and Throughput Management

Maintaining system performance across the hybrid architecture requires thoughtful configuration and monitoring to ensure optimal message processing. Research examining high-performance monitoring in hybrid event processing systems emphasizes the importance of buffer management strategies that prevent message accumulation at integration points [10].

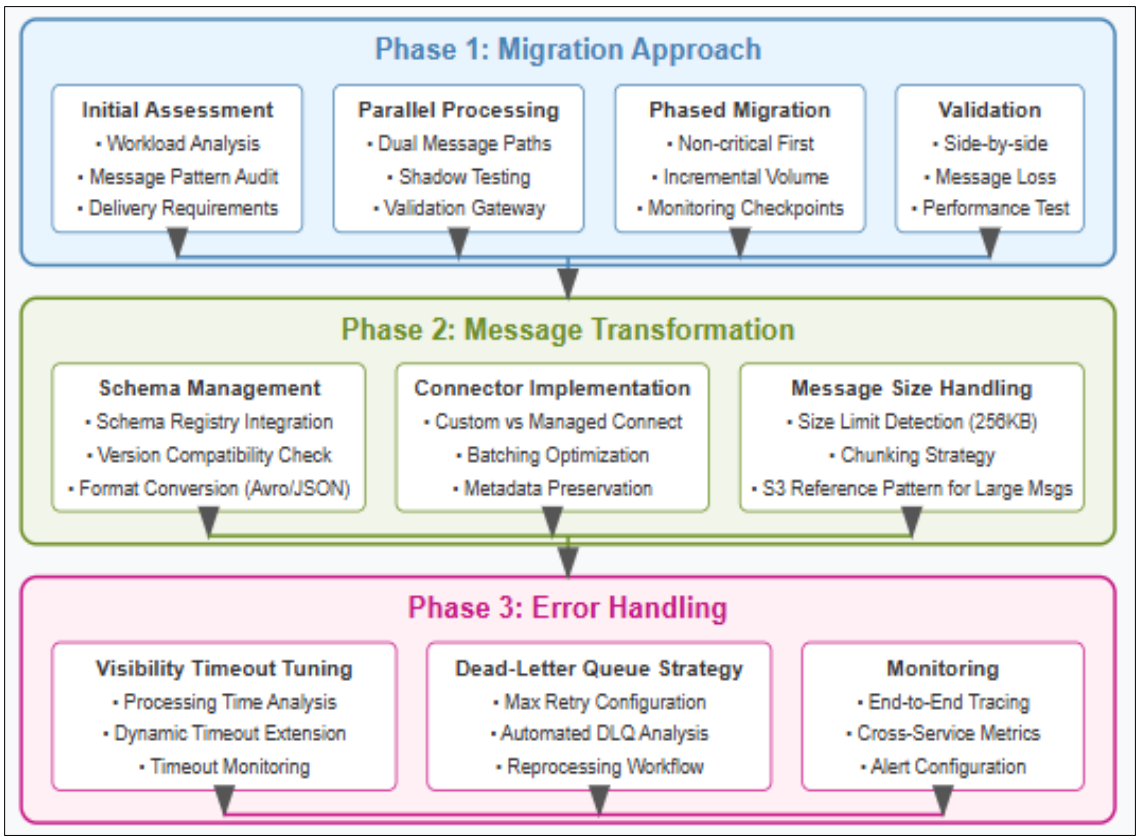

Figure 2 Implementation strategy and best practices for the Kafka to SNS-SQS integration [9, 10]

The connector components bridging Kafka and SNS must implement adaptive polling mechanisms that respond to downstream processing capabilities, adjusting consumption rates to prevent overwhelming SNS API limits. Studies demonstrate that implementations utilizing exponential backoff strategies during rate-limiting scenarios maintain an 85% higher average throughput compared to static polling approaches [10]. Additionally, the architecture must implement appropriate batching strategies to optimize SNS publishing efficiency, with research indicating optimal performance at batch sizes between 10-25 messages depending on the average message size. Monitoring represents a critical aspect of performance management, with high-performing implementations establishing comprehensive dashboards that correlate metrics across system boundaries. Research shows that organizations implementing end-to-

end tracing across the hybrid architecture identify performance bottlenecks 62% faster than those monitoring individual components in isolation [9].

6. Case Studies and Future Directions

The Kafka to SNS-SQS architecture demonstrates exceptional value across diverse industry implementations, providing a robust foundation for scalable event processing while significantly reducing operational complexity. This section explores real-world applications, implementation insights, and emerging patterns in hybrid messaging architectures.

6.1. Real-World Implementation Examples

The hybrid messaging architecture has proven particularly effective in high-throughput scenarios requiring reliable message delivery across distributed systems. Cloud messaging implementations leveraging similar architectural principles have demonstrated the ability to handle billions of messages daily while maintaining consistent performance characteristics [11]. Financial services organizations have been early adopters, implementing the pattern to process transaction streams that require both high throughput and guaranteed delivery. A major European payment processor deployed this architecture to handle transaction volumes exceeding 5,000 messages per second during peak periods while ensuring regulatory compliance through comprehensive audit trails. The implementation allowed them to replace dedicated consumer instances with elastic processing resources that adjusted automatically to workload variations. Similar benefits have emerged in retail environments, where the architecture supports inventory synchronization across distributed systems. The pattern's inherent fault tolerance capabilities, including automatic retries and dead-letter queues, provide crucial reliability for business-critical operations where message loss could result in inventory discrepancies or fulfillment errors [11].

6.2. Lessons Learned from Production Deployments

Organizations implementing hybrid messaging architectures have identified several critical success factors that significantly impact deployment outcomes. Effective message routing strategies represent a foundational element, with sophisticated implementations utilizing content-based routing to direct messages based on payload characteristics rather than simple topic subscriptions [12]. This advanced routing enables dynamic processing paths that adapt to message attributes without requiring producer modifications. Connection management emerges as another critical consideration, particularly in architectures spanning multiple network boundaries. Implementation experience demonstrates the importance of connection pooling and circuit-breaking patterns that protect system components during partial outages or network degradation. Organizations have also discovered that comprehensive monitoring spanning both Kafka metrics and AWS service telemetry provides essential visibility for operational management. The correlation of metrics across system boundaries enables rapid identification of performance bottlenecks and processing delays that might otherwise remain hidden within individual components [12].

6.3. Emerging Patterns and Future Directions

The evolution of hybrid messaging architectures continues through integration with complementary services that enhance capabilities beyond basic message delivery. The publish-subscribe pattern underlying these architectures has proven particularly amenable to extension through additional processing layers that implement sophisticated event-processing capabilities [12]. Organizations are increasingly implementing complex event processing (CEP) layers that identify patterns across multiple message streams, enabling real-time analytics without custom implementation. This capability proves particularly valuable for use cases requiring correlation across distinct event types, such as fraud detection systems that analyze transaction patterns across multiple channels. Another emerging pattern involves the implementation of event sourcing models that leverage the hybrid architecture as an event store, capturing complete state change histories for subsequent replay and analysis. This approach enables powerful debugging capabilities and audit trails while maintaining system performance. As these architectures mature, implementations increasingly incorporate event discovery mechanisms that enable dynamic subscription management, allowing new consumers to discover and process events without explicit configuration changes [12].

7. Conclusion

The Kafka to SNS-SQS integration model provides organizations with a powerful blueprint for modernizing their event streaming infrastructure while addressing the dual challenges of cost optimization and operational complexity. By leveraging AWS's managed messaging services alongside Kafka's robust production capabilities, enterprises can achieve the ideal balance of reliability, scalability, and maintenance simplicity. This hybrid approach eliminates the burden of managing consumer scaling, reduces infrastructure provisioning concerns, and simplifies error handling

through native AWS mechanisms. As event-driven architectures continue to proliferate across industries, this integration pattern offers a compelling path forward for organizations seeking to maximize the business value of their real-time data streams while minimizing the associated technical debt and operational overhead. The resulting architecture delivers immediate cost benefits and positions teams for greater agility and innovation in an increasingly data-driven landscape.

References

[1] Seetharamugn, "The Complete Guide to Event-Driven Architecture," Medium, 30 Aug. 2023. [Online]. Available: https://medium.com/@seetharamugn/the-complete-guide-to-event-driven-architecture-b25226594227

[2] James Beswick, "Choosing Between Messaging Services for Serverless Applications," AWS Compute Blog, 28 Sep. 2020. [Online]. Available: https://aws.amazon.com/blogs/compute/choosing-between-messaging-services-for-serverless-applications/

[3] AWS, "Building Event-Driven Architectures on AWS," Amazon Web Services, 2022. [Online]. Available: https://d1.awsstatic.com/SMB/building-event-driven-architectures-aws-guide-2022-smb-build-websites-and-apps-resource.pdf

[4] Sunny Srinidhi, "Emulating Apache Kafka with Amazon SNS and SQS," Contact Sunny Blog, 22 Jan. 2020. [Online]. Available: https://blog.contactsunny.com/tech/emulating-apache-kafka-with-amazon-sns-and-sqs#google_vignette

[5] Biswanath Mukherjee, "The Purpose of AWS Messaging Services and Choosing the Right One," Medium, 1 Jan. 2024. [Online]. Available: https://medium.com/@biswanath.ita/the-purpose-of-aws-messaging-services-and-choosing-the-right-one-759fadbf59a5

[6] Tobias Pfandzelter et al., "Streaming vs. Functions: A Cost Perspective on Cloud Event Processing," arXiv:2204.11509v2, 12 Aug. 2022. [Online]. Available: https://arxiv.org/pdf/2204.11509

[7] Tiago Boldt Sousa et al., "Engineering Software for the Cloud: Messaging Systems and Logging," ResearchGate, July 2017. [Online]. Available: https://www.researchgate.net/publication/321139340_Engineering_Software_for_the_Cloud_Messaging_Systems_and_Logging

[8] nordlys.studio, "Monitoring & Observability in Event-Driven Systems," LinkedIn, 27 April 2023. [Online]. Available: https://www.linkedin.com/pulse/monitoring-observability-event-driven-systems-nordlysstudio

[9] Nurul Shafiqa and Azhar Iskandar, "Optimizing Enterprise-Level Data Migration Strategies," ResearchGate, Vol. 6, no. 1, March 2023. [Online]. Available: https://www.researchgate.net/publication/386214557_Optimizing_Enterprise-Level_Data_Migration_Strategies

[10] Yosuke Ozawa et al., "A Hybrid Event-Processing Architecture based on the Model-driven Approach for High-Performance Monitoring," ResearchGate, Aug. 2007. [Online]. Available: https://www.researchgate.net/publication/4264273_A_Hybrid_Event-Processing_Architecture_based_on_the_Model-driven_Approach_for_High_Performance_Monitoring

[11] Firebase, "FCM Architectural Overview," Google Firebase Documentation, 2023. [Online]. Available: https://firebase.google.com/docs/cloud-messaging/fcm-architecture

[12] ByteByteGo, "Event-Driven Architectural Patterns," ByteByteGo Newsletter, 24 Oct. 2024. [Online]. Available: https://blog.bytebytego.com/p/event-driven-architectural-patterns