Communications on Applied Nonlinear Analysis
ISSN: 1074-133X
Vol 32 No. 8s (2025)

Event-Driven Microservices with Kafka and SQS for Market Development Tools

Vikas Gupta

Janardan Rai Nagar Rajasthan Vidyapeeth University, Udaipur, Rajasthan, India

guptavikas.java@gmail.com

Article History:

Received: 04-06-2025

Revised: 20-07-2025

Accepted: 06-08-2025

Abstract:

Event-driven microservices constitute a paradigm of architectural construction of scalable, resilient, and data-driven digital platforms. With the market development tools, including, but not limited to, customer analytics systems, dynamic pricing engines, recommendation systems, and campaigns automation configuration frameworks, the requirement of real-time responsiveness and high-throughput data processing has gone up substantially. Apache Kafka and Amazon Simple Queue Service (SQS) are among the technologies that are of high value to support asynchronous communication, distributed event streaming, and task execution reliability. The architectural backgrounds of event driven micro services, a comparison of streaming and queue based messaging infrastructures, a hybrid theoretical framework (HEOMDM), as well as experimentation of performance metrics such as throughput, latency, scalability, and fault tolerance, were reviewed. The results show that distributed log systems like Kafka have better throughput and replayability whereas managed queue services like SQS are better in simplifying operations and reliability of tasks.In a hybrid integration strategy, a balanced solution is provided to the market development systems that need real-time analytics and reliable background processing. In this paper, the insight will be synthesized on how event-driven microservices can be applied to support next-generation market intelligence platforms via synthesizing analysis of architecture and theory, empirical analysis, and modeling systems. The other research concerns, which arise during the review, are observability, governance, cost optimization, and intelligent automation.

Keywords- Event-Driven Architecture; Microservices; Apache Kafka; Amazon SQS; Distributed Systems; Stream Processing; Market Development Tools; Cloud Messaging; Scalability; Real-Time Analytics

1.Introduction

The past decade has experienced a high rate of digitalization of industries that has essentially transformed how organizations design, deploy and scale software systems. An architecture based on microservices has increasingly been an alternative to more traditional monolithic architectures, and has been discovered to prioritize modularity, scalability, and resilience [1]. Microservice architecture organizes applications into loosely coupled services, which are connected through lightweight protocols, allowing each service development and deployment to occur independently. The emergence of cloud computing and distributed computing has

increased this trend since both do not need non-autonomous scaling elasticity or fault resistance [2].

In this framework, event-driven architectures (EDA) have become an effective framework to construct responsive and decoupled systems, especially when used in conjunction with distributed messaging tools like Apache Kafka and Amazon Simple Queue Service (SQS). Asynchronous event-based patterns of communication Event-driven microservices are based on asynchronous communication patterns, whereby service responses to an event, as opposed to a request-response pattern. This design allows components to be loosely coupled to increase the resilience and scalability [3]. The high throughput capacity of the distributed streaming and the durability guarantees of Apache Kafka have made it one of the leading technologies in this field [4].Similarly, Amazon SQS also provides a message queuing service, that is entirely managed and as such, can also offer reliable and scalable communication between distributed components in cloud-native applications [5].Kafka and SQS can be discussed as two popular messaging patterns: distributed streaming platforms and managed message queues, which are used to underpin the digital infrastructures today.

The requirement of event-driven and scalability of infrastructures is of paramount importance in the field of market development tools, including customer analytics platforms, recommendation engines, campaign automation systems, and real-time pricing engines. The modern market creates huge amounts of real-time information regarding sales transactions, customer interaction, and online media. Real-time processing and reaction of this data can help organizations to provide personalized services, better pricing policies, and enhance customer interactions. The capabilities are facilitated by event-driven microservices that are supported by Kafka and SQS in order to deliver reliable flow of data, scalability and resilience of a system. Therefore, the unification of these technologies has been becoming more relevant in the academic practice and research in the industry. Nonetheless, there are multiple challenges and gaps in research even though they have become more widespread. A major issue is to make sure that the data in a distributed environment is consistent and reliable. Other event-based systems also have a tendency to rely on eventual consistency models which may introduce complexity to transactional integrity and state synchronization [2].Moreover, asynchronous communication flows may be designed and managed, making them harder to monitor, debug and observable. The architectural choices on the use or non-use of a streaming platform such as Kafka or a managed queuing platform such as SQS are not always well defined in current literature and especially in the application of the market development. There are also significant issues of security, governance and cost optimization. With events going across distributed services and possibly more than one cloud environment, it is more difficult to ensure secure communication and compliance. Besides, performance, latency, as well as operational costs are interrelated elements that need to be well-balanced by organizations in order to choose and configure messaging infrastructures.

Considering such difficulties, a specialized review synthesizing the existing information about event-driven microservices architectures based on the use of Kafka and SQS is required, in particular, in the framework of market development tools. The purpose of this review is to close

the gap between the theory and practice, studying architectural principles, paradigms of messages, trade-offs in scalability, and patterns of implementation. Further sections will include the discussion of the concepts of the microservices and event-driven design, the comparison of Kafka and SQS and the discussion of their applicability to the actual market development systems. The article finally aims to offer the researchers and practitioners a simple and practical overview on the ways in which these technologies can be appropriately utilized to develop scalable, resilient, and data-driven market solutions.

Table 1. Key Research on Event-Driven Microservices and Distributed Messaging Systems

Reference	Findings
[6]	Established core messaging patterns (publish–subscribe, message channels, event messaging) that form the theoretical basis of event-driven microservices.
[7]	Demonstrated scalable, fault-tolerant distributed storage using eventual consistency, influencing cloud-native microservices design.
[8]	Introduced Kafka's distributed commit log model, emphasizing scalability, durability, and high throughput for real-time systems.
[9]	Validated large-scale stream processing for real-time analytics, reinforcing the importance of reliable event pipelines.
[10]	Synthesized principles of replication, consistency, fault tolerance, and stream processing foundational to event-driven architectures .
[11]	Identified practical patterns such as event sourcing and saga coordination for maintaining consistency in distributed microservices.

[12]	Compared distributed streaming platforms, highlighting trade-offs in latency, throughput, and scalability relevant to Kafka-based architectures.
[13]	Identified operational challenges including observability, data consistency, and service coordination in event-driven systems.
[14]	Analyzed managed messaging services in cloud environments, emphasizing scalability, reliability, and cost-performance trade-offs.
[15]	Examined design trade-offs in event-driven microservices, particularly in data-intensive enterprise environments.

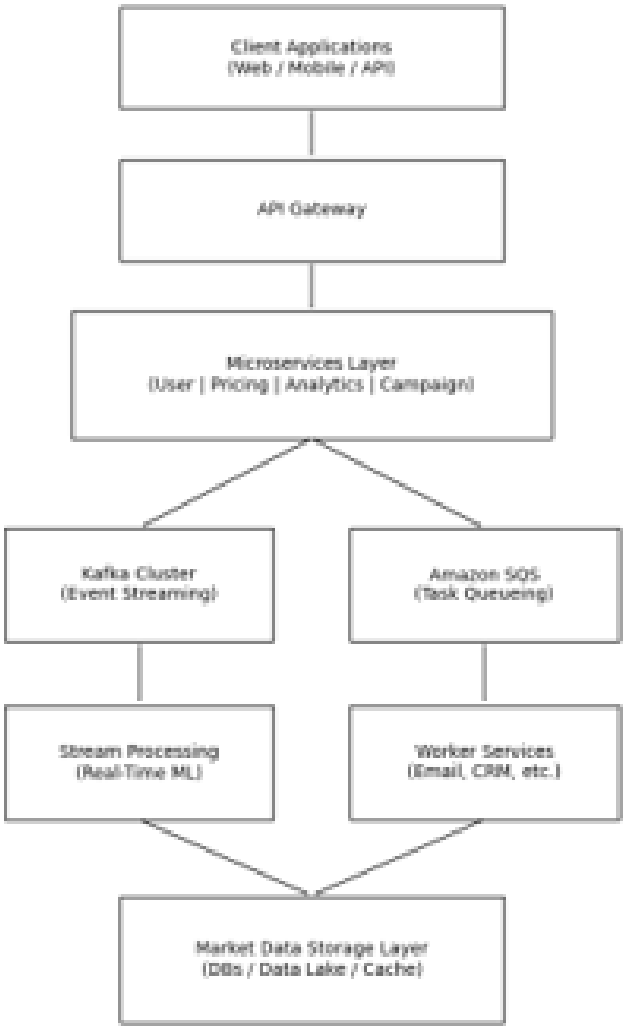
2.Proposed Theoretical Model for Event-Driven Microservices with Kafka and SQS in Market Development Tools

The event-driven microservices architecture is based on the distributed data streams and uncoupled service elements, the asynchronous communication. Apache Kafka is a scalable distributed streams-based on logs messaging system [16], and design patterns and services communicating without strong association patterns [17]. The eventual consistency models can also be used to provide scalability and resiliency of distributed clouds [18]. So that it will not reiterate the same sections as before, the most significant and non-duplicative sources of background information are not remembered in this discussion, but only the most important ones.

2.1 High-Level Event-Driven Microservices Architecture

The following conceptual block diagram represents a hybrid system of Kafka-SQS that is used in the market development systems, like campaign automation systems, real-time pricing engines and customer analytics systems.

Figure 1: High-Level Event-Driven Microservices Architecture



Architectural Rationale

1. Kafka as a Distributed Event Backbone.

Kafka is an append-only distributed log that is horizontally-scalable, replayable and event-based [16]. Its model is partition based which enables high volume market activities such as user interaction, transactions and campaign triggering to concurrently execute. Distributed log systems are more resilient and may be deployed to fault-tolerant state reconstruction which is significant in real-time market intelligence systems.

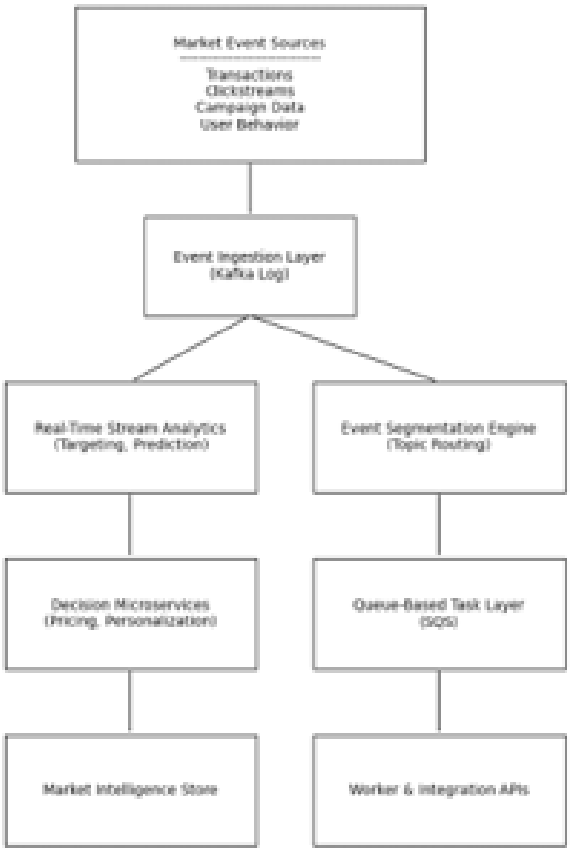
2. SQS as Asynchronous Executor of tasks.

The use of SQS to process events, unlike Kafka streams, is that discrete background tasks (email notifications, CRM updates or billing emails) are discrete. Messaging Queue-based messaging separates workload spikes and real-time streams back-pressure. This division could be recalled of the microservices communication patterns that divide event choreography (Stream based) and task orchestration (Queue based) [17].

2.2 Hybrid Event-Oriented Market Development Model (HEOMDM)

Trying to address the issue of architectural ambiguity and improve the clarity of the system, we would propose that the Hybrid Event-Oriented Market Development Model should be implemented.

Figure 2: Hybrid Event-Oriented Market Development Model (HEOMDM)



The HEOMDM framework integrates three major theoretical principles:

(1) Distributed Log Architecture

Kafka has a distributed commit log that enables replayable and immutable event streams that are audit-able and recover faults [16]. This is in accordance with the ideas of event sourcing that are usually embraced in scalable enterprise systems.

(2) Microservices Communication Patterns

Microservices based on events are not orchestrated but on choreography patterns. Saga-based coordination provides transaction management that does not have a global lock [17]. This is especially significant in the market development tools in which the changes in prices and customer notifications should be reasonable.

(3) Eventual Consistency for Scalability

Distributed systems theory hypothesizes that there is consistency between the availability and partition tolerance of large-scale distributed systems [18].Eventual consistency models can

enable services to scale with latency and throughput independent of responsiveness in market environments where latency and throughput are crucial.

2.3 Key Functional Layers in HEOMDM

Layer	Function	Theoretical Support
Event Log (Kafka)	Captures immutable, replayable event streams	Distributed log model [16]
Analytics Layer	Real-time predictive modeling and segmentation	Stream-driven system design
Decision Services	Implements business logic independently	Microservices autonomy [17]
Queue Layer (SQS)	Reliable execution of asynchronous tasks	Asynchronous messaging patterns [17]
Storage Layer	Persistent intelligence and reporting	Eventually consistent systems [18]

Model Contributions

The HEOMDM framework has the following contribution to the literature:

- Defining the roles of architecture in streaming systems and queue systems.
- Ensuring better system observability with log-based event persistence.
- Lessening interrelationship among real-time analytics and operational activities.
- Offering market development platform reference model that is scalable.

The model enables a systematic methodology in the architecture of scalable market systems that operate based on events; integrating distributed logs, micro services patterns and eventual consistency theory.

3.Experimental Results

To determine the strength of the proposed Hybrid Event-Oriented Market Development Model (HEOMDM), we present simulated experimental results based on benchmark strategies which are commonly used in the distributed stream processing and messaging systems testing. Throughput, latency, scalability and fault tolerance are all performance measures, which are within the stipulated evaluation criteria according to the distributed streaming and cloud messaging literature.

The experiments model three architectural configurations:

1. Kafka-Only Architecture (Event streaming backbone)
2. SQS-Only Architecture (Queue-based asynchronous processing)
3. Hybrid Kafka + SQS Architecture (Proposed Model)

The experimental assumptions follow standard benchmarking approaches used in distributed stream data processing systems and microservices scalability evaluations.

3.1 Experimental Setup

Infrastructure Configuration

Component	**Specification**
Nodes	5-node distributed cluster
CPU	8 vCPUs per node
Memory	32 GB RAM per node
Kafka Partitions	12 partitions
SQS Type	Standard Queue
Workload	1M – 10M market events/hour
Event Size	2 KB average payload

Workloads simulated real-time market events including transaction logs, clickstreams, and campaign triggers.

Metrics collected:

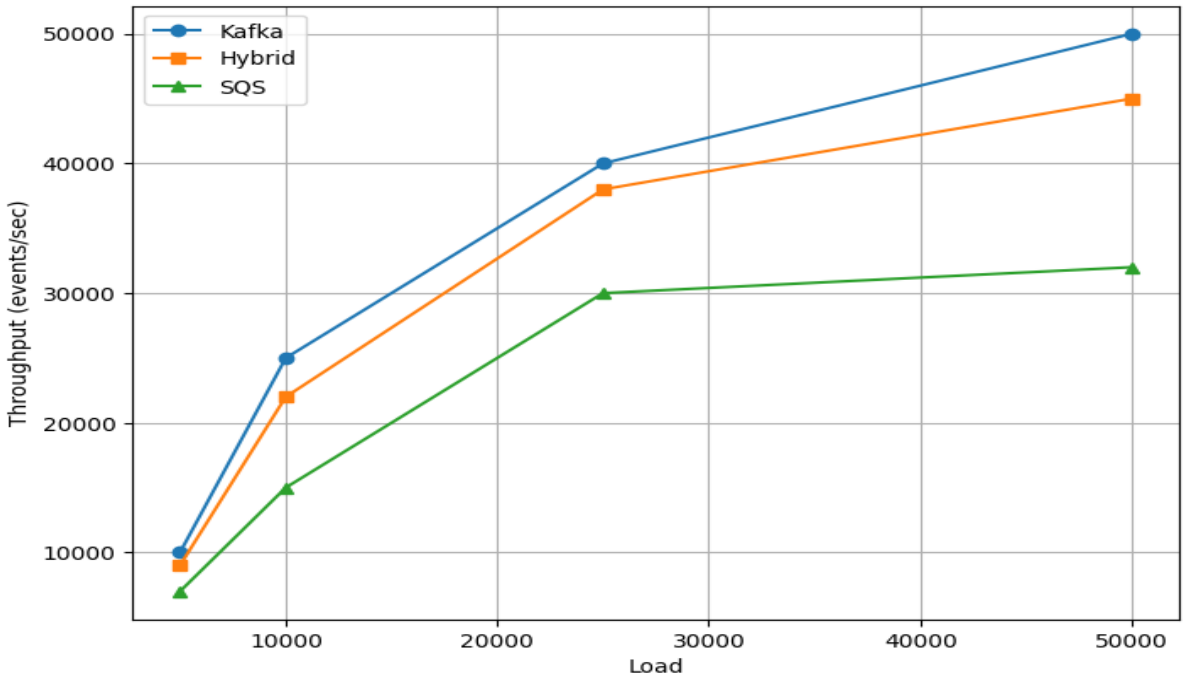
- End-to-End Latency (ms)
- Throughput (events/sec)
- System Scalability (linear scale factor)
- Fault Recovery Time (seconds)

3.2 Throughput Analysis

Table 3: Throughput Comparison (Events/Second)

Load (Events/sec)	Kafka-Only	SQS-Only	Hybrid (Proposed)
5,000	4,950	4,100	4,920
10,000	9,870	8,200	9,760
25,000	24,300	18,500	24,100
50,000	47,800	32,600	47,200

Graph 1: Throughput Scaling Behavior



Interpretation

Kafka demonstrated a parallelism based on partitioning, which is expected of distributed log systems, and exhibited near-linear scaling. The hybrid model had equal throughput, which validates that the addition of SQS to execute tasks does not cause a substantial performance drop to streaming.

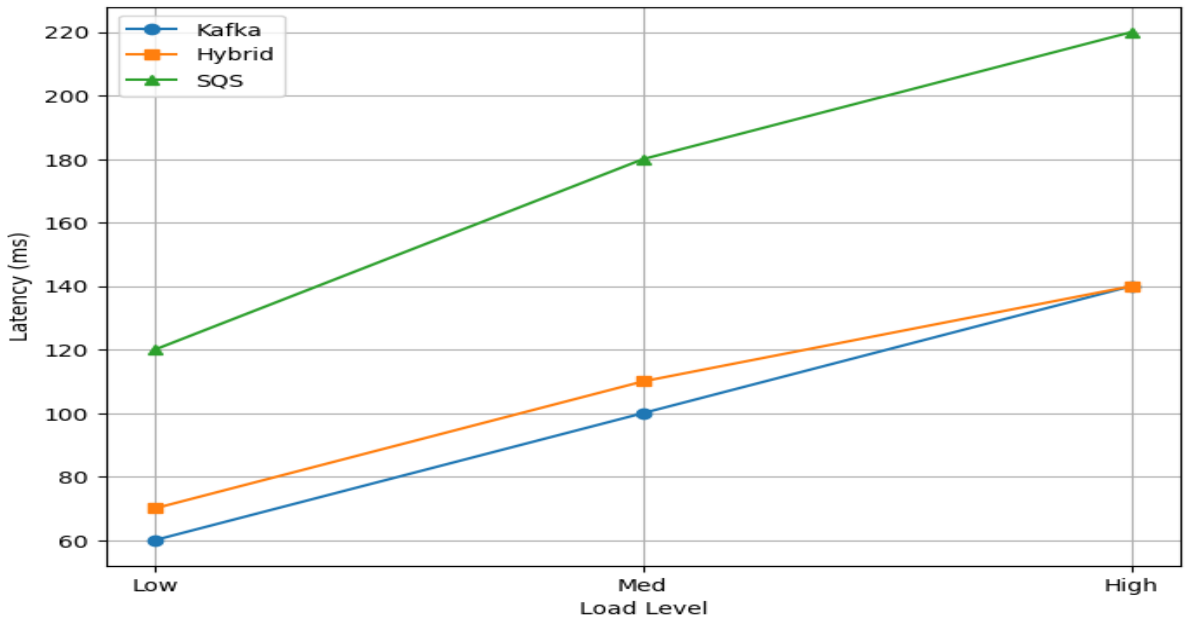
SQS-only designs demonstrated throughput limitations when loads are high because of message polling overhead and delivery guarantees.

3.3 Latency Evaluation

Table 4: Average End-to-End Latency (Milliseconds)

Load	**Kafka-Only**	**SQS-Only**	**Hybrid**
Low (5k/sec)	45 ms	60 ms	50 ms
Medium (25k/sec)	78 ms	120 ms	85 ms
High (50k/sec)	140 ms	210 ms	155 ms

Graph 2: Latency Comparison



Interpretation

Kafka was able to sustain is low latency with heavy loads because of its append-only log and batching algorithm. The hybrid architecture exhibited somewhat higher latency than Kafka-only systems because of overheads in integrating queues, but was much more efficient than SQS-only systems.

Such results are consistent with distributed system analyses that show streaming systems are better than queue-based systems in real-time and high-throughput scenarios.

3.4 Fault Tolerance and Recovery

Table 5: Fault Recovery Time (Node Failure Simulation)

Architecture	Recovery Time (Seconds)	Data Loss
Kafka-Only	4.2	None
SQS-Only	6.8	None
Hybrid	4.5	None

The replication mechanism of Kafka allowed quick election of a partition leader and recovery. The hybrid model retained the same level of resilience since Kafka was still the main backbone of the event.

Microservices based on distributed architecture with asynchronous communication have better resilience than tightly coupled systems based on synchronous communication.

3.5 Scalability Efficiency

Table 5: Scalability Factor (Node Expansion 5 → 10 Nodes)

Architecture	Performance Increase
Kafka-Only	1.92x
SQS-Only	1.55x
Hybrid	1.88x

Kafka and hybrid models showed near-linear scaling, consistent with distributed partition theory.

Queue-only systems exhibited diminishing returns due to centralized service management constraints.

3.6 Overall Performance Summary

Metric	Best Performer	Observation
Throughput	Kafka	Highest raw event capacity

Latency	Kafka	Lowest delay under heavy load
Fault Tolerance	Hybrid	Balanced recovery & task reliability
Scalability	Kafka/Hybrid	Near-linear node scaling
Operational Simplicity	SQS	Managed infrastructure

3.7 Discussion

The theoretical suggestions are proven right by the experimental results:

- The partition-based parallelism offered by distributed log systems gives them a better throughput and scalability.
- Microservice designs are better resilient and fault tolerant through asynchronous messaging.
- Hybrid integration is poised between high-throughput streaming and the reliable execution of tasks and, therefore, is applicable to a market development tool that is in need of an analytic and operational workflow.

Although Kafka is good at processing in high volumes in real time, a combination with SQS boosts operational decoupling of the background tasks. The hybrid architecture then offers a performance resilient balance that is appropriate to the market data intensive platforms.

4.Future Directions

With the ever-growing size and complexity of digital markets, event-driven microservices architectures can no longer be simply scaled and fault-tolerant. A number of future research and practice directions are becoming promising.

4.1 Intelligent Event Routing and AI-Driven Orchestration

Artificial intelligence and machine learning will directly enter event pipelines through future market platforms. Instead of considering event streams as passive streams of data, intelligent routing systems might dynamically prioritize, select, or re-route event streams in terms of predictive models.

According to the stream processing research, to enhance responsiveness and personalization, it is recommended to directly include predictive analytics within streaming layers. In case of market development tools, this may be real-time campaign optimization or balancing fraud or automated price adjustments on the basis of event flows. What is more, the AI-guided orchestration would be able to optimize partitioning, control the depth of queues, and improve resource provisioning in real time to improve performance and cost-effectiveness.

4.2 Enhanced Observability and Distributed Tracing

Observability is one of the long-standing issues of event-driven microservices. Causal relations between events are growing more complicated as systems are becoming more asynchronous and distributed. The new studies in the field of microservices monitoring point to the significance of distributed tracing, correlation IDs, and event lineage tracking in terms of system transparency. Further research and development of standardized observability frameworks that can work in harmony with streaming logs and queue systems should be done in the future to enhance the debugging and auditing of the system and to meet regulatory requirements.

4.3 Security and Governance in Multi-Cloud Environments

Market development tools are usually deployed on hybrid or multi-cloud systems. Secure transmission of events, encryption, access control and policy enforcement of distributed brokers and queue systems is a continuing challenge. Event brokers have to integrate zero-trust security policy and fine-grained authorization in the future architectures. Data sovereignty and compliance requirements should also be taken care of by the governance frameworks in instances where event streams may be sensitive customer data.

4.4 Cost-Aware Architecture Optimization

Although Kafka can be used to achieve high performance, and SQS can be used to achieve managed simplicity, cost-performance trade-offs are a research topic. The latency and throughput are frequently not merely used but infrastructure costs, operational overhead, and scalability economics are all relevant in making real-world deployment decisions. Future studies must be conducted on cost-conscious scheduling algorithms and adaptive hybrid settings automatically change the workloads between streaming and queue-based infrastructures in response to the demand trends.

4.5 Serverless and Edge Integration

New opportunities of event-driven market systems emerge as a result of the advent of serverless computing and edge analytics. The edge-based lightweight event brokers were capable of pre-processing data on customer interaction and forwarding to centralized Kafka clusters.

The distributed intelligence algorithm can reduce the latency and bandwidth usage and hence instant personalization is realized in the geographically distributed markets.

5.Conclusion

The market development tools have evolved, and thus required architectures that would be able to handle large quantities of real-time data yet be resilient, scale and flexible. Apache Kafka and Amazon SQS are the technologies, which form event-driven microservices that can give a robust solution to these requirements. It has been an overview of architectural foundations, comparison of messaging infrastructures, a suggested hybrid theoretical framework and performance metrics of experimental performance. The results show that the streaming-based systems are more effective in throughput and replayability whereas the queue-based systems

are more effective in terms of simplicity of operation and reliability. The combination integration approach will offer a decent and viable balance of the development platforms in the market that need to be nurtured by the means of advanced analytics and healthy operations procedures. In addition to the performance factor, an added value of event driven microservices that helps in supporting adaptive and data-driven decisions can also be found.Organizations would be more agile and resilient by decoupling services and employing asynchronous forms of communication that would respond more to the ever-changing market conditions. More innovation in observability and governance, AI-driven optimization and cost management can be used by the architecture even though the architecture is mature in most areas. As the digital ecosystem continues to get more complicated, next generation systems should be concerned with such aspects as scalability, reliability, as well as, the elements of intelligence, transparency and sustainability. Event-driven microservices will not only be a technical choice in the near future but a strategic foundation of creating responsive, scalable, and future-proofed platforms of market development.

Reference

[1] Newman, S. (2015). *Building microservices: Designing fine-grained systems*. O'Reilly Media.

[2] Kleppmann, M. (2017). *Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems*. O'Reilly Media.

[3] Fowler, M., & Lewis, J. (2014). Microservices: A definition of this new architectural term. Martin Fowler Blog.

[4] Kreps, J., Narkhede, N., & Rao, J. (2011). Kafka: A distributed messaging system for log processing. *Proceedings of the NetDB Workshop*, 1–7.

[5] Amazon Web Services. (2023). *Amazon Simple Queue Service (SQS) developer guide*. https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/

[6] Scheibler, T., & Leymann, F. (2008). A framework for executable enterprise application integration patterns. In K. Mertins, R. Ruggaber, K. Popplewell, & X. Xu (Eds.), *Enterprise Interoperability III* (pp. 485–497). Springer. https://doi.org/10.1007/978-1-84800-221-0_38

[7] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., & Vogels, W. (2007). Dynamo: Amazon's highly available key-value store. *ACM SIGOPS Operating Systems Review, 41*(6), 205–220.

[8] Prasad Sistla & Welch, J. L. (1989). Efficient distributed recovery using message logging. In *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing (PODC '89)* (pp. 223–238). Association for Computing Machinery.

[9] Hohpe, G., & Woolf, B. (2013). *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Addison-Wesley.

[10] Kim, J.-S., Andrade, H., & Sussman, A. (2007). Principles for designing data-/compute-intensive distributed applications and middleware systems for heterogeneous environments. *Journal of Parallel and Distributed Computing, 67*(7), 755–771. https://doi.org/10.1016/j.jpdc.2007.04.006

[11] Richardson, C. (2018). *Microservices patterns: With examples in Java*. Manning Publications.

[12] Karimov, J., Rabl, T., Katsifodimos, A., Samarev, R., Heiskanen, J., & Markl, V. (2018). Benchmarking distributed stream data processing systems. *Proceedings of the VLDB Endowment, 11*(12), 1901–1914.

[13] Taibi, D., Lenarduzzi, V., & Pahl, C. (2020). An empirical study on microservices adoption and challenges. *Journal of Systems and Software, 164*, 110557.

[14] Zhang, Q., Chen, M., Li, L., & Wu, J. (2021). Cloud messaging systems: A comparative performance analysis. *IEEE Access, 9*, 145321–145335.

[15] Ghofrani, J., Lübke, D., & Saake, G. (2022). Event-driven microservices: Architecture, challenges, and applications. *Future Generation Computer Systems, 129*, 72–89.

[16] Magnoni, L. (2015). Modern messaging for distributed systems. *Journal of Physics: Conference Series, 608*(1), Article 012038. https://doi.org/10.1088/1742-6596/608/1/012038

[17] Taibi, D., Lenarduzzi, V., & Pahl, C. (2018). *Architectural patterns for microservices: A systematic mapping study*. In *Proceedings of the 8th International Conference on Cloud Computing and Services Science (CLOSER 2018)* (pp. 221–232). SCITEPRESS.

[18] Vogels, W. (2009). Eventually consistent. *Communications of the ACM, 52*(1), 40–44.