The American Journal of Engineering and Technology
ISSN 2689-0984 | Open Access

TYPE Original Research
PAGE NO. 71-77
DOI 10.37547/tajet/Volume07Issue08-08

Real-time Data Streaming using Kafka, Kinesis, and RabbitMQ

Vladyslav Vodopianov
Senior Software Engineer, Wirex Kyiv, Ukraine

OPEN ACCESS

SUBMITED 17 July 2025
ACCEPTED 28 July 2025
PUBLISHED 12 August 2025
VOLUME Vol.07 Issue 08 2025

CITATION

Vladyslav Vodopianov. (2025). Real-time Data Streaming using Kafka, Kinesis, and RabbitMQ. The American Journal of Engineering and Technology, 7(8), 71–77. https://doi.org/10.37547/tajet/Volume07Issue08-08

Abstract: In the present work a comprehensive comparative analysis of the three leading platforms for organizing message streaming — Apache Kafka, Amazon Kinesis and RabbitMQ — is performed with the aim of identifying their architectural features, operational strengths and limitations under conditions of peak loads and stringent latency requirements. The study relies on a comprehensive methodological approach, including a systematic review of current scientific publications, the conduct of comparative performance measurements in laboratory settings and the synthesis of practical case studies of integrating the systems under consideration into real IT landscapes. The obtained results demonstrate that a reasoned choice of platform for stream processing depends on a multitude of interrelated factors: the volume of messages processed, the required throughput metrics and maximum response time, the preferred deployment model (on-premises solution, cloud service or their hybrid), the capabilities for seamless integration with existing services and infrastructure, as well as the project's budgetary constraints. On the basis of the conducted analysis a unified decision-making methodology is proposed for selecting tools for streaming data processing, adapted to the tasks of data engineers, distributed systems architects and researchers of high-performance information platforms. The material is of practical interest to specialists designing fault-tolerant and scalable distributed message queues, as well as to experts in real-time analytics and cloud solution developers seeking to gain a deeper understanding of the architectural schemes and methods for optimizing throughput applied in Kafka, Kinesis and RabbitMQ. In addition, the research results may be useful to scientists in the field of distributed computing and the Internet of Things, focusing on the theoretical foundations and

practical aspects of constructing reliable event-data pipelines.

Keywords: streaming data processing, Apache Kafka, Amazon Kinesis, RabbitMQ, big data, distributed systems, low latency, high bandwidth, data architecture, platform comparison.

Introduction

Modern real-time data streaming technologies have fundamentally transformed methodologies for data collection, processing, and analytics, shifting organizations from traditional batch processing to architectures of continuous monitoring and immediate decision-making based on operational data. The rapid growth in generated data volumes—particularly in areas such as the Internet of Things (IoT), high-frequency financial transactions, social media activity, and cloud-based online services—has sustained strong demand for end-to-end analytics solutions. According to [1], the global streaming analytics market is projected to grow from USD 29.53 billion in 2024 to USD 125.85 billion by 2029, representing a compound annual growth rate of 33.6 percent over the forecast period [1].

However, a scientific-methodological gap exists in the comprehensive comparative analysis of leading streaming platforms—Apache Kafka, Amazon Kinesis, and RabbitMQ—taking into account their latest functional enhancements, scalability, and performance metrics in hybrid and multi-cloud environments, as well as the specifics of integration with modern data processing pipelines (data lakes and data warehouses).

The objective of the study is to analyze the characteristics of real-time data streaming using Kafka, Kinesis, and RabbitMQ.

The scientific novelty resides in outlining the criteria for selecting streaming infrastructure components, which encompass not only key technical specifications but also operational complexity and total cost of ownership.

The study hypothesizes that the most rational choice of platform is determined by the results of a multi-criteria analysis of the specific project requirements and the characteristics of its operational environment.

Materials and methods

Literature review reveals that researchers address real-time challenges in streaming data from multiple perspectives. In the Research and Markets report [1], a quantitative forecasting methodology is applied, with market segmentation by technology—Complex Event Processing (CEP), Event Stream Processing (ESP), and data visualization—as well as by application domain, including fraud detection, asset management, and risk management. The authors construct long-term development scenarios through 2029, relying on deployment statistics and growth rates in key sectors.

In the empirical research section, emphasis is placed on performance measurement and throughput optimization.Amilineni K., Krishnan R., Goyal S., Rao S. V. N.[2] employ test streams with varying packet sizes and configurable batching parameters to identify optimal settings for minimizing latency and maximizing throughput in real-world applications. Padmanaban K., Balaji R. V., Baskar S., Sharma V. [3] focus on tuning Apache Kafka clusters—adjusting the number of partitions and replication factors—and demonstrate how these adjustments influence latency and event propagation speed when scaling to thousands of messages per second. Velickovska M., Gusev M. [4] present a case study of streaming electrocardiogram data via AWS Kinesis and Firehose, evaluating latency, packet loss, and infrastructure load. Bux R., Shenoy G. S. [6] compare RESTful services with RabbitMQ in a microservices architecture, showing that the message broker maintains stable throughput under peak loads but exhibits higher latency for small messages. The official Confluent guide “Apache Kafka® Performance” [10] provides recommendations for tuning the Java Virtual Machine, network buffers, and producer and consumer settings.

Comparative reviews by Vyas S., Jain P., Sharma S., Soni P. [7] and by Dingorkar S., Singh S., Ghosh S., Roy R.[9] provide a comprehensive overview of the data-transmission ecosystem. George J. [5] outlines the design of a scalable AWS pipeline using Amazon Kinesis for ingestion, AWS Lambda for processing, Amazon S3 and Redshift for storage, and Amazon QuickSight for visualization. Chen F., Yan Z., Gu L. [8] propose a low-latency infrastructure based on Sangfor, combining Apache Kafka with zero-copy and RDMA-optimized network stacks, and integrating Apache Storm and Spark Streaming for hybrid processing. The Microsoft [11] and AWS [12] technical guides offer recommendations on selecting streaming platforms and message brokers in serverless environments, emphasizing specific use cases such as log management, real-time analytics and ETL.

These evaluations reveal divergent assessments of

latency: some researchers assert Kafka's superiority under heavy loads [3, 10], whereas others report low end-to-end latency for RabbitMQ with small message sizes [6]. Findings regarding the Kappa architecture for IoT are similarly contradictory: Dingorkar S., Singh S., Ghosh S. and Roy R. [9] endorse it, while Chen F., Yan Z. and Gu L. [8] underline the merits of hybrid approaches. Security and real-time encryption, resilience to network failures and split-brain scenarios in distributed brokers remain underexplored. Likewise, mechanisms for automatic scaling in multi-cluster and multi-cloud deployments, and the integration of advanced complex-event-processing (CEP) engines with machine-learning models for adaptive analytics, are insufficiently developed.

Results and discussions

The analysis of Apache Kafka, Amazon Kinesis, and RabbitMQ reveals fundamental differences in their architectures, delivery models, performance characteristics, and intended use cases. Each system was engineered for a distinct purpose: Kafka to handle event streams at LinkedIn, Kinesis as AWS's cloud-based streaming analytics platform, and RabbitMQ as a reliable broker conforming to the AMQP standard.

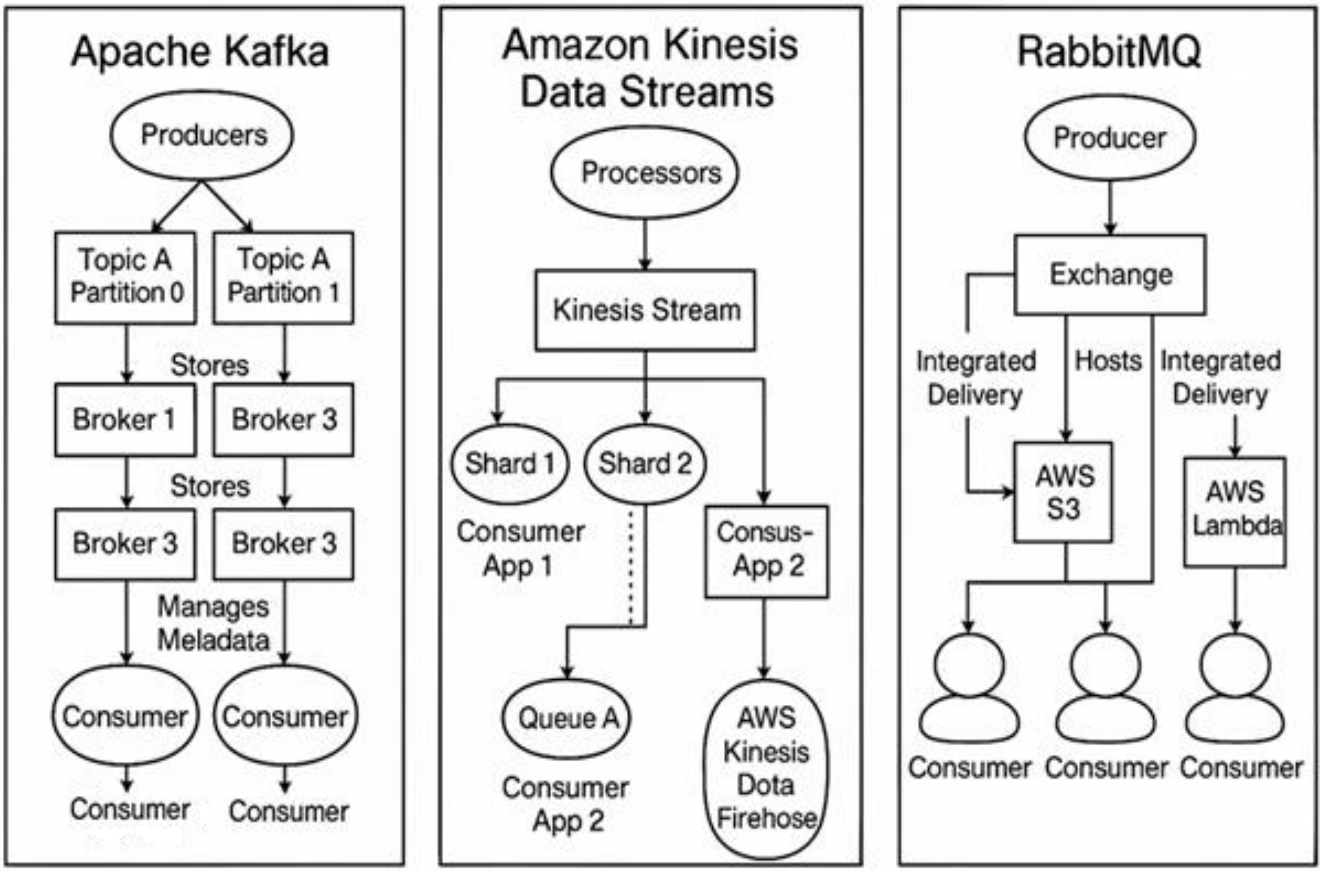
Apache Kafka is based on an immutable commit log: events are appended sequentially to topics partitioned across the cluster, enabling linear scalability through parallel processing [2, 3]. Messages are stored on disk with configurable retention policies, and consumer offsets permit arbitrary navigation through the event history. Optimizations for sequential write and read operations deliver throughput of up to millions of messages per second per cluster, and long-term storage of streaming data extends Kafka's functionality beyond simple message delivery [7]. Partition replication across brokers ensures fault tolerance; however, cluster configuration and ZooKeeper management (though simplified in recent releases) demand operational expertise [9].

Amazon Kinesis is a fully managed AWS service for ingesting and processing data streams. It comprises Kinesis Data Streams and Kinesis Data Firehose. Data Streams provides low latency and automatic scaling via sharding: each shard offers a dedicated throughput unit, and adjusting the shard count enables dynamic adaptation to workload fluctuations [4]. Deep integration with Lambda, S3, DynamoDB, Redshift, and other AWS services simplifies the construction of end-to-end analytics pipelines [5]. Data Firehose automates event delivery and transformation into target storage systems and analytics tools, freeing developers from custom ETL coding. Its primary limitation is the dependency on the AWS ecosystem, which may be unsuitable for hybrid or multi-cloud architectures.

RabbitMQ implements the AMQP standard and supports MQTT, STOMP, and other protocols, focusing on flexible routing through exchanges and queues [6]. Bindings between exchanges and queues enable the configuration of complex message-delivery topologies, a critical capability for microservices and IoT scenarios. Delivery guarantees include at-most-once, at-least-once, and exactly-once, with optional message persistence. Under extreme workloads, however, RabbitMQ's throughput generally falls short of Kafka's, and its storage model does not provide a long-lived streaming log [8].

Figure 1 schematically shows the basic components of the three systems.

Figure 1 - Architectures of Apache Kafka, Amazon Kinesis Data Streams and RabbitMQ [2, 4, 6].



In the comparative analysis of distributed messaging systems, the key metrics are throughput and latency. Since these parameters vary according to hardware platform characteristics, software configuration settings, and workload profile, aggregated results of empirical studies [2, 8, 10] are often employed. For single-message transmission, Apache Kafka in typical industrial scenarios is capable of processing a high number of messages per second on a scalable cluster while maintaining minimal delay.

The AWS Kinesis Data Streams architecture exhibits comparable performance: by elastically increasing the number of shards, it achieves stable throughput with end-to-end latency on the order of single-digit milliseconds [4, 5].

RabbitMQ, when optimally configured and applied to workflows involving numerous small messages and complex routing, also delivers high message-processing rates with low latency; however, its horizontal scalability under extreme peak loads is inferior to that of Kafka [6].

Table 1 summarizes the key characteristics of the platforms under consideration.

Table 1 - Comparative characteristics of Apache Kafka, Amazon Kinesis and RabbitMQ [2, 5, 6, 7, 11, 12]).

Characteristic	**Apache Kafka**	**Amazon Kinesis Data Streams**	**RabbitMQ**
Core paradigm	Distributed commit log	Shard-based data streams	Message broker (AMQP, MQTT, STOMP)
Throughput	Very high	High; horizontally scalable	Moderate to high
Latency	Low	Low	Very low in specific scenarios

Message retention	Long-term, configurable	Up to 7 days (extendable to 365 days)	Short-term by default; persistence optional
Deployment model	On-premises, cloud, hybrid	AWS managed service	On-premises, cloud, hybrid
Operational complexity	Moderate to high (requires expertise)	Low (managed service)	Moderate
Scalability	Horizontal (adding brokers or partitions)	Horizontal (adding or merging shards)	Horizontal (clustering) and vertical
Delivery guarantees	At least once; exactly once (since v0.11)	At least once	At most once; at least once; exactly once with transactions
Ecosystem / integrations	Extensive (Spark, Flink, Storm, connectors)	Deep integration with AWS services	Broad client support; pluggable architecture
Primary use cases	Big-data analytics, event sourcing, log aggregation, streaming ETL	Real-time applications on AWS, IoT, mobile data	Microservices, task queues, notifications, IoT
Cost model	Open-source software (infrastructure and support costs)	Pay-as-you-go pricing (throughput and storage)	Open-source software (infrastructure and support costs); commercial editions available

Next, table 2 will describe the advantages, disadvantages, and trends of using Apache Kafka, Amazon Kinesis, and RabbitMQ in real-time data streaming.

Table 2 - Advantages, disadvantages, and trends of using Apache Kafka, Amazon Kinesis, and RabbitMQ in real-time data streaming [2, 5, 7].

Technology	Advantages	Disadvantages	Future trends
Apache Kafka	- High bandwidth and low latency- Horizontal scaling (sharding via topic-partition) - Delivery guarantees (at least-once, exactly-once) - - Large ecosystem (Kafka Streams, hsqldb, Connect)	- The complexity of the initial setup and operation - High resource requirements (disk I/O, memory) - Difficulties with security and integration into corporate networks - The need to manage your own cluster	- Transition to cloud-based (Managed Kafka: Confluent Cloud, AWS MSK) - Active development of streamSQL (ksqlDB) and integration with ML/AI - Unification of event-sourcing and CQRS-patterns - Improvement of Operator approaches for Kubernetes

Amazon Kinesis	- Fully managed AWS service- Auto-scaling and high availability out-of-the-box - Deep integration with the AWS ecosystem (Lambda, S3, Redshift) - AWS IAM-level security and data encryption	- Vendor lock-in (AWS only) - Cost increases with data volume and retention - Bandwidth limits per shard (single shard) - Less flexibility for non-standard scenarios	- Development of enhanced fanout and HTTP/2 connections to reduce delays- Tight integration with ML services (Salemaker, Rekognition) - The emergence of Kinesis Data Streams for IoT and edge cases - Automatic scaling of shards based on load
RabbitMQ	- Easy to install and configure - Support for multiple protocols (AMQP, MQTT, STOMP) - Flexible routing and reliable queuing mechanism - Lightweight and intuitive web UI for administration	- Limited horizontal scalability (clustering is more difficult) - Delays increase with very large volumes of messages - There is no native support for stream-processing	- Operator development for Kubernetes (RabbitMQ Operator) - Expansion of cloud-based managed offerings (CloudAMQP, AWS MQ) - - Integration with stream-processing frameworks (Flunk, Aka Streams)

Thus, in the context of enterprises deeply integrated into the AWS cloud ecosystem, Kinesis is often regarded as the preferred solution due to its tight integration with all AWS services, simplified cluster management, and built-in scalability and monitoring mechanisms. In contrast, organizations prioritizing maximum autonomy and avoidance of vendor lock-in frequently opt to deploy Kafka or RabbitMQ directly within their own data centers or on virtual infrastructure in a cloud environment of their choice.

Conclusion

The comparative evaluation indicates that Apache Kafka outperforms alternative messaging systems when tasked with ingesting and processing massive, continuous data flows: it achieves sub-millisecond end-to-end latencies while preserving messages indefinitely, a combination that has cemented its role in large-scale analytics and event-sourcing frameworks. In contrast, Amazon Kinesis—leveraging its native integration within the AWS ecosystem—provides frictionless, fully managed scalability and provisioning, making it especially attractive for enterprises already committed to Amazon's cloud platform and seeking rapid, infrastructure-light deployment. Meanwhile, RabbitMQ retains its competitive edge through a highly adaptable routing topology and support for a variety of messaging patterns; this versatility proves particularly useful in microservice environments and distributed work-queue scenarios where throughput requirements fall below the scale that would justify a Kafka-based solution.

Looking forward, it is imperative to investigate composite architectures that harness the complementary advantages of these platforms—such as coupling Kafka's high-throughput buffering with Kinesis's serverless elasticity or RabbitMQ's sophisticated exchange mechanisms. Additionally, a rigorous assessment of their performance and cost-efficiency within emerging paradigms like serverless stream processing and geographically distributed (edge) computing environments will provide critical guidance for designing resilient, low-latency data pipelines in heterogeneous deployment contexts.

References

1. Research and Markets. (n.d.). Streaming analytics market by technology (real-time data processing, complex event processing, data visualization & reporting, event stream processing), application (fraud detection, predictive asset management, risk management) - Global forecast to 2029. https://www.researchandmarkets.com/report/streaming-analytics (accessed June 6, 2025)
2. Amilineni, K., Krishnan, R., Goyal, S., & Rao, S. V. N. (2022). Optimizing data stream throughput for real-time applications. In S. K. Bhoi, S. Patnaik, S. P. Mohanty, & B. K. Tripathy (Eds.), International conference on big data intelligence and computing, 410-417.

3. Padmanaban, K., Balaji, R. V., Baskar, S., & Sharma, V. (2024). Apache Kafka on big data event streaming for enhanced data flows. In 2024 8th International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC), 977-983. https://doi.org/10.1109/I-SMAC61858.2024.10714884

4. Velickovska, M., & Gusev, M. (2022). Comparing AWS streaming services: A use case on ECG data streams. In 2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO),1387-1392. https://doi.org/10.23919/MIPRO55190.2022.9803359

5. George, J. (2024). Build a realtime data pipeline: Scalable application data analytics on Amazon Web Services (AWS). SSRN, 1-9. http://dx.doi.org/10.2139/ssrn.4963387

6. Bux, R., & Shenoy, G. S. (2024). Performance analysis of RESTful web services and RabbitMQ for microservices based systems on cloud environment. In 2024 3rd International Conference for Innovation in Technology (INOCON),1-6. https://doi.org/10.1109/INOCON60754.2024.10511747

7. Vyas, S., Jain, P., Sharma, S., & Soni, P. (2021). Literature review: A comparative study of real time streaming technologies and Apache Kafka. In 2021 Fourth International Conference on Computational Intelligence and Communication Technologies (CCICT), 146-153. https://doi.org/10.1109/CCICT53244.2021.00038

8. Chen, F., Yan, Z., & Gu, L. (2022). Towards low-latency big data infrastructure at Sangfor. In A. K. Das, P. K. Singh, & H. Ghayvat (Eds.), International symposium on emerging information security and applications, 37-54.

9. Dingorkar, S., Singh, S., Ghosh, S., & Roy, R. (2024). Real-time data processing architectures for IoT applications: A comprehensive review. In 2024 First International Conference on Technological Innovations and Advance Computing (TIACOMP), 507-513. https://doi.org/10.1109/TIACOMP64125.2024.00090

10. Confluent. (n.d.). Apache Kafka® performance. https://developer.confluent.io/learn/kafka-performance/ (accessed May 22, 2025)

11. Microsoft. (n.d.). Choose a stream processing technology in Azure. https://learn.microsoft.com/en-us/azure/architecture/data-guide/technology-choices/stream-processing (accessed June 07, 2025)

12. Amazon Web Services. (n.d.). Choosing between messaging services for serverless applications.https://aws.amazon.com/ru/blogs/compute/choosing-between-messaging-services-for-serverless-applications/ (accessed June 07, 2025)