Sarcouncil Journal of Engineering and Computer Sciences

ISSN(Online): 2945-3585

Volume- 04| Issue- 09| 2025

Research Article **Received:** 27-07-2025| **Accepted:** 20-08-2025 | **Published:** 14-09-2025

Fundamentals of Real-Time Data Streaming Architectures

Pallavi Desai

Independent Researcher, USA

Abstract: The article has explored data streaming architecture in real-time and how everything in it works, including the fundamentals, mechanisms, and constructs. Starting with the disruptive nature of streaming in contemporary data systems, it forms the base of event sourcing as the architecture that records changes of immutable states. The article specifies the three essential elements - producers, brokers, and consumers that altogether bring about a faithful flow of information, and then examines the continuum of processing guarantees present in at-most-once and exactly-once delivery. Advanced material on stateful stream processing is presented, including strategies of state management, windowing strategies to approximate their relationship over measured time, and management of late-arriving data within distributed settings. The article carries to best operational practices which include required monitoring metrics, scaling approaches to include the rising data volumes, and health management techniques that will provide reliability of systems. The article has used authoritative sources to give it technical substance, whilst making it accessible to readers to be able to apply the same in a practical application in either deployment and maintenance of streaming architectures in the enterprise settings.

Keywords: Event Sourcing, Stream Processing, Real-Time Analytics, Stateful Processing, Processing Semantics.

INTRODUCTION

In the modern data-driven world, real-time data streaming is beginning to become the building block of modern analytics, operational intelligence, and the automated decision system. Streaming architectures have become a necessity as companies rely on them to process a constant stream of data and derive useful insights within the shortest period. This conceptual leap outside of the paradigm of batch-processing towards continuous streaming is a radical change in the way businesses use data assets and create value.

The real-time streaming technology infrastructure has become extremely developed, with platforms such as Apache Kafka and Apache Flink serving as de facto standards. These platforms are now providing more complex processing and stateful computation, and easy connections to legacy systems and to cloud native applications. Kai Waehner observes that this maturation has pushed streaming architectures beyond niche applications to become essential elements of enterprise data strategies across numerous sectors (Waehner, K. 2025).

Financial institutions have led streaming adoption for fraud detection, risk assessment, and trading analytics, where fractions of seconds determine market advantage. Telecommunication companies leverage streaming platforms to monitor network performance continuously, catching anomalies and service issues before customer experience suffers. Manufacturing operations implement streaming systems for equipment monitoring and predictive maintenance, dramatically cutting expensive downtime. According to Jeyhun Karimov *et al*., such implementations enable processing times in milliseconds even at scale, factors that have made capabilities like instant personalization and on-demand inventory a possibility in the e-commerce and retail industry (Karimov, J. *et al*., 2018).

Streaming solutions allow practitioners to be constantly aware of the state of their patients and use predictive analytics, which hastens the ability of a practitioner to respond and positively impact the outcomes of patients. Transportation and logistics processes are real-time data processing-based and can be used to dynamically route and allocate resources, reacting to changes in real-time. Artificial intelligence integrated with streaming architectures has gone even further, offering the possibility of constantly training models and making inferences in real time at a large scale. Kai Waehner highlights that organizations using AI-enhanced streaming solutions experience marked improvements in predictive accuracy compared to conventional batch approaches (Waehner, K. 2025). Managed streaming services have emerged as an alternative to self-managed streaming provided by cloud providers, which allows reducing the complexity but ensures a level of flexibility needed when workloads are not predictable.

Security and governance practices have been developed to support the continual flow of data, and like technical capabilities, have been evolving to respond to issues specific to constant data flows. These include schema evolution management, data quality monitoring, and comprehensive lineage tracking across complex processing networks.

***Corresponding Author:** Pallavi Desai

DOI- https://doi.org/10.5281/zenodo.17117111

Rodriguez and Nguyen stress that successful implementations incorporate robust monitoring frameworks, allowing operations teams to maintain performance and reliability during scaling (Karimov, J. *et al*., 2018).

This article explores the core principles, components, and best practices underpinning robust streaming architectures, from foundational concepts to advanced implementation considerations.

EVENT SOURCING: THE FOUNDATION OF STREAMING

Event sourcing has been the core concept behind the streaming architectures, where the state of the system is fundamentally re-conceptualized. It performs snapshot isolation that records every instance of state change as a permanent, durable event, and renders a bookkeeping log of all activities. Instead of simply storing the current state, event sourcing keeps a history of all changes; it also provides a source of truth everywhere it is used during the application lifetime. This pattern is highly valuable according to the architectural guidance of Microsoft, especially in distributed systems where attempting to achieve consistency across components is very difficult. By keeping a log of events, systems will be able to achieve an eventual consistency of all states, and any operation can be traced (Microsoft,).

The immutable nature of event logs delivers several critical advantages. Complete audit trails for compliance and debugging become inherent system properties rather than bolted-on features. This characteristic proves especially valuable in regulated industries where comprehensive activity records remain mandatory. Microsoft's architectural documentation emphasizes that event sourcing naturally addresses requirements for audit logging, historical state reconstruction, and compensating transactions – capabilities that often demand additional complexity in traditional architectures (Microsoft,).

Event sourcing enables system state reconstruction at any historical point, creating powerful capabilities for analysis and recovery. By replaying events up to a specific moment, applications can regenerate exact historical states without maintaining separate snapshots. This temporal flexibility supports advanced scenarios such as point-in-time recovery, historical analysis, and temporal queries that would prove impractical with traditional state-focused architectures. Solace's analysis of event-driven patterns highlights that this capability provides significant advantages for business intelligence and compliance scenarios, allowing organizations to understand precisely what happened at any moment in operational history (Solace).

The separation between event recording and interpretation represents another crucial advantage. The same event stream can support multiple consumers with diverse processing needs and interpretations. Real-time dashboards, analytical processors, compliance monitors, and operational systems can all derive specific views from the same underlying event sequence. This characteristic enables organizational agility by allowing new consumption patterns without modifying event producers or existing consumers. Solace's architectural guidance describes this as "event-enabled independence," where teams develop and evolve services autonomously while maintaining system-wide consistency through shared event streams (Solace).

This decoupling pattern extends to system evolution as well. Microsoft's architectural documentation notes that systems built on event sourcing principles evolve more gracefully as business requirements change (Microsoft,). New functionality can be implemented by developing additional consumers that interpret existing events in novel ways, often without modifying established components. Solace emphasizes that this evolutionary flexibility represents a significant advantage for organizations operating in dynamic business environments, allowing systems to adapt incrementally rather than requiring disruptive overhauls (Solace). This characteristic contrasts sharply with traditional architectures, where state model changes frequently cascade throughout systems, requiring coordinated updates across multiple components.

Publisher: SARC Publisher

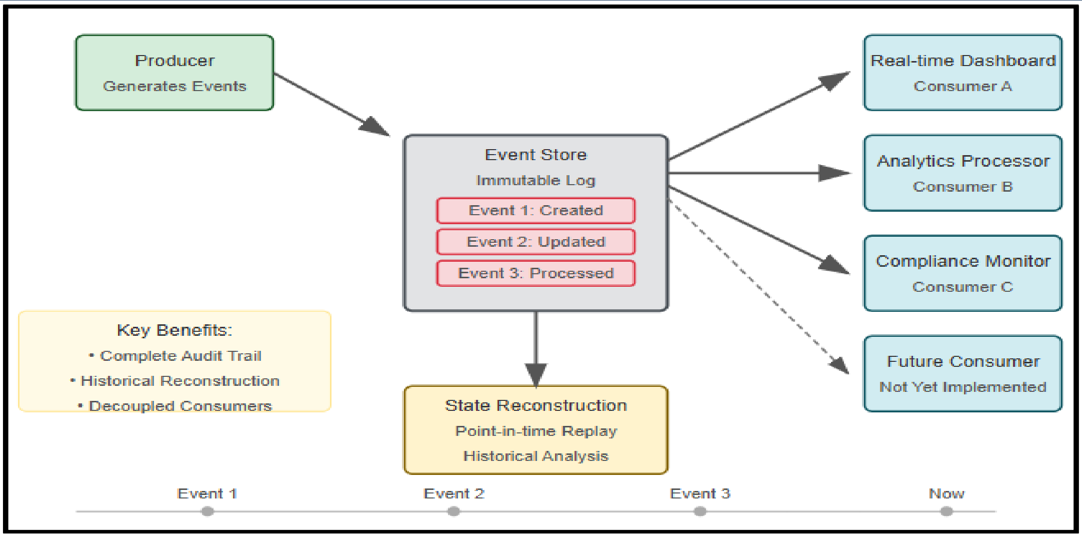

Fig 1: Event Sourcing Architecture (Microsoft,; Solace)

Core Components of Streaming Systems

Most of the contemporary streaming systems consist of three basic elements that collaborate to ensure steady and engrossed data flow: producers, brokers, and consumers. Each of the components performs very different functions and has its own protocols to uphold system integrity and optimize performance.

Producers

Producers serve as entry points for data into streaming systems, generating events and publishing them to the streaming platform. According to Apache Kafka's documentation, producers perform several critical functions beneath their seemingly simple interface (Kafka, A.).

Producers serialize event data according to predefined schemas, transforming in-memory objects into byte sequences suitable for transmission. They assign partition keys that determine event placement within topic partitions, enabling related events to maintain order while allowing horizontal scaling. This partitioning strategy directly impacts processing parallelism and message ordering guarantees.

To optimize network utilization, producers implement batching and compression mechanisms. Rather than transmitting each event individually, they accumulate events into batches based on configurable thresholds before transmission. The producer's acknowledgment protocol management represents a critical aspect of reliability guarantees, with configurable settings that balance throughput against durability.

Advanced producer implementations incorporate idempotency mechanisms to prevent duplicate event publishing during retries. As Estuary notes in their architectural overview, these mechanisms typically involve producer-assigned unique identifiers and broker-side deduplication logic to ensure consistent processing semantics even during network failures (Richman, J. 2025).

Brokers

Brokers constitute the central storage and coordination mechanism of streaming architectures. These specialized servers form clusters that collectively maintain streaming platform durability and availability guarantees (Kafka, A.).

Brokers receive events from producers and persist them to disk in a highly optimized, append-only log structure. By organizing events into partitioned logs, brokers enable horizontal scaling while preserving strict ordering guarantees within each partition. Data replication forms a cornerstone of broker operations, with each partition typically replicated across multiple brokers according to configurable replication factors.

Brokers implement configurable retention policies that determine how long events remain available. These policies can be based on time, size, or compaction strategies. According to Estuary, these retention strategies directly influence storage requirements, query capabilities, and compliance characteristics (Richman, J. 2025).

Consumer group coordination represents another vital broker function. Brokers track consumer group membership, manage partition assignments, and maintain offset metadata that enables

Publisher: SARC Publisher

consumers to resume processing from their last position after restarts or failures.

Consumers

Consumers form the final link in the streaming chain, reading and processing events to extract value from the data flow (Kafka, A.).

Consumers deserialize event data according to schema definitions, converting byte sequences back into structured data suitable for application logic. Offset management represents a critical consumer function, with each consumer tracking its position within each assigned partition to ensure reliable processing.

Sophisticated consumers implement backpressure mechanisms to handle scenarios where processing capacity cannot match incoming event rates. These mechanisms can slow consumption rates to prevent memory exhaustion while maintaining system stability. As detailed in Estuary's architectural guidance, effective backpressure implementation requires careful coordination between system components to avoid cascading failures during load spikes (Richman, J. 2025).

Consumer group coordination enables horizontal scaling by distributing partitions across multiple consumer instances, ensuring efficient workload distribution while maintaining processing guarantees.

Table 1: Component Responsibilities in Streaming Architectures (Kafka, A.; Richman, J. 2025)

Component	**Primary Function**	**Key Capabilities**	**Technical Implementation**
Producers	Generate and publish events	Serialization, Partition keys, Batching, Idempotency	Schema-based transformation, Configurable thresholds, Unique identifiers
Brokers	Store and coordinate data flow	Persistence, Replication, Retention, Consumer coordination	Append-only logs, Partition management, Time/size/compaction policies, Group tracking
Consumers	Process and extract value	Deserialization, Offset management, Backpressure, Horizontal scaling	Schema-based conversion, Position tracking, Rate limitation, Partition distribution

PROCESSING SEMANTICS AND GUARANTEES

Streaming systems offer varying delivery guarantees, each balancing distinct trade-offs that system architects must carefully evaluate against specific project requirements. These processing semantics profoundly shape how applications handle failures, recovery scenarios, and consistency challenges across distributed environments.

At-Most-Once Delivery

This approach places speed and minimal delay above guaranteed delivery. Kleppmann, writing about data-intensive applications, characterizes at-most-once as the weakest delivery guarantee where messages might vanish but duplication never occurs (Kleppmann, M. 2019). This pattern emerges naturally when systems favor minimal overhead above absolute reliability.

Implementation typically strips away safeguards: no confirmation mechanism from broker to producer, no lasting record of consumer positions, and straightforward fire-and-forget messaging patterns. This stripped-down approach cuts overhead tied to persistence and retry mechanisms, maximizing throughput while slashing processing delays.

At-most-once delivery makes sense for scenarios where occasional gaps prove harmless, like metrics gathering, status notifications, or monitoring systems where scattered data points matter little. As Bryant explains while dissecting Kafka processing guarantees, such applications tolerate missing information because they typically handle high-volume, time-series data where broader patterns remain clear despite occasional holes (Bryant, A. 2019).

At-Least-Once Delivery

Most streaming platforms default to at-least-once semantics, guaranteeing message processing while accepting potential duplication during recovery scenarios. This middle-ground balances reliability against performance, preventing data loss while acknowledging duplicate possibilities.

Key implementation details include persistent producer retries until receiving acknowledgment, durable tracking of consumer positions, and clear commit protocols for position advancement.

Publisher: SARC Publisher

Bryant explains that Kafka's standard configuration follows precisely this model, where producers retry message publication while consumers save position markers only after completing processing tasks (Bryant, A. 2019).

Applications must build idempotent operations to manage potential duplicate processing. Kleppmann stresses that idempotence patterns become essential when working with at-least-once systems, allowing applications to handle identical messages repeatedly without adverse consequences (Kleppmann, M. 2019). Practical idempotence approaches include natural keys, message identifier deduplication, and mathematical operations that remain stable across multiple applications.

Exactly-Once Delivery

The pinnacle standard for mission-critical applications, exactly-once semantics guarantee messages are processed precisely once, even amid failures. Bryant clarifies that achieving this semantics across distributed systems demands coordination between numerous components and typically introduces performance compromises (Bryant, A. 2019).

Current systems achieve exactly-once processing through transactional interfaces coordinating writes across multiple partitions, two-phase commit protocols linking processors with state stores, and producer idempotence combined with consumer transaction backing. Kleppmann points out that these mechanisms aim to provide complete atomicity throughout processing pipelines, guaranteeing that either all related operations succeed together or none proceed (Kleppmann, M. 2019).

Exactly-once processing carries performance costs but remains essential for financial transfers, medical records, order management, and similar domains where accuracy trumps all other concerns. The implementation complexity and speed impact find justification through eliminating expensive reconciliation processes while guaranteeing data integrity across entire processing chains.

Table 2: Performance vs. Reliability Trade-offs in Streaming Delivery Semantics (Kleppmann, M. 2019; Bryant, A. 2019)

Delivery Semantic	Reliability	Performance	Implementation Approach
At-Most-Once	Low	High	No acknowledgments, no offset tracking, Fire-and-forget messaging
At-Least-Once	Medium	Medium	Producer retries, Persistent offset tracking, Explicit commit protocols
Exactly-Once	High	Low	Transactional APIs, Two-phase commits, Combined producer/consumer guarantees

STATEFUL STREAM PROCESSING

Beyond basic message transport, modern streaming architectures support sophisticated stateful processing, enabling complex analytical operations across continuous data flows. These capabilities transform streaming platforms from simple data movement mechanisms into comprehensive computational frameworks supporting real-time analytics and decision-making.

State Management

Stateful processors preserve contextual information spanning multiple events, allowing applications to construct and refresh aggregates, monitor patterns, and execute complex business logic across individual messages. Akidau's groundbreaking work on streaming systems identifies state management as among the toughest challenges facing stream processing design, demanding careful consideration around durability, accessibility, and recovery (Akidau, T. *et al.*, 2018).

Implementation approaches include local state stores backed by changelog topics, where processors keep state in memory or local storage while recording modifications to separate streams, enabling fault tolerance. This technique balances performance against recoverability, allowing quick state access while providing solid recovery options. Distributed key-value stores offer alternative approaches for shared state management, enabling multiple processors to reach and modify common state elements.

Checkpointing mechanisms capture periodic state snapshots, establishing recovery points for resuming processing after failures. Estuary's technical overview explains that these checkpoints create consistent recovery markers across distributed processing elements, allowing systems

Publisher: SARC Publisher

to resume operation following failures without losing or corrupting data (Richman, J. 2025). These mechanisms collectively enable fault-tolerant stateful processing despite individual processor failures, maintaining application accuracy despite infrastructure problems.

Windowing Operations

Time-based analysis demands windowing strategies to group events across specific intervals, transforming endless streams into bounded sets processable through traditional aggregation techniques. Akidau emphasizes that effective windowing strategies remain fundamental for meaningful analytics across continuous data streams (Akidau, T. *et al*., 2018).

Tumbling windows split time into fixed, non-overlapping segments, establishing clear boundaries for aggregations and analyses. These windows create distinct, sequential chunks, simplifying result interpretation. Sliding windows maintain moving time ranges with fixed lengths, updating continuously as time advances, providing smoother results with more frequent updates.

Session windows cluster events based on activity periods using timeout-based boundaries, dynamically adjusting window sizes according to data patterns. Estuary's technical documentation highlights that this approach excels at capturing natural interaction patterns, particularly within user behavior analysis contexts (Richman, J. 2025). Each windowing strategy delivers distinct advantages for different analytical scenarios, letting system architects select approaches that match specific requirements.

Handling Late Data

Across distributed systems, events frequently arrive disordered or delayed thanks to network issues, source system lags, or processing bottlenecks. Streaming platforms tackle this challenge through several complementary mechanisms.

Watermark tracking provides system-wide estimates regarding event time completeness, signaling when events up to specific timestamps have likely arrived. Akidau describes how watermarks enable systems to progress despite inherent uncertainty surrounding distributed event generation (Akidau, T. *et al*., 2018).

Allowed lateness parameters define grace periods for late events after watermarks pass, creating windows for stragglers. This mechanism lets system architects explicitly balance result completeness against processing speed. Side output collections capture extremely delayed events arriving after the allowed lateness windows close.

Estuary stresses that these late-data handling mechanisms remain essential for maintaining analytical integrity across real-world deployments where network conditions, source system behavior, and processing backlogs introduce timing variations (Richman, J. 2025). Together, these approaches help streaming systems produce accurate results despite timing challenges inherent within distributed environments.

Table 3: Comparative Analysis of Stateful Stream Processing Components (Akidau, T. *et al*., 2018; Richman, J. 2025)

Component	Purpose	Implementation Approaches	Key Benefits
State Management	Store contextual information across events	Local state stores with changelog topics, distributed key-value stores, and Checkpointing mechanisms	Enables aggregations, Pattern detection, and Complex business logic
Windowing Operations	Transform unbounded streams into finite sets	Tumbling windows (fixed, non-overlapping), Sliding windows (moving time range), Session windows (activity-based)	Time-based analysis, Traditional aggregation techniques, Natural interaction pattern detection
Late Data Handling	Process out-of-order or delayed events	Watermark tracking, Allowed lateness parameters, Side output collections	Maintains analytical integrity, Balances completeness vs. speed, Handles real-world timing variations

OPERATIONAL BEST PRACTICES

Monitoring and controlling streaming production machines in real-time present inherent difficulties and costs in terms of establishing sufficient and acceptable performance, reliability, and scalability. The need to deliver insight in real-time is also making operational excellence as crucial as architectural design, as data volumes continue to

Publisher: SARC Publisher

grow and firms depend on real-time insight to drive their operations.

Key Metrics

Critical indicators reveal streaming system health and performance. According to Shapira's definitive guide covering Kafka operations, monitoring should focus on metrics directly impacting business outcomes and user experiences (Shapira, G. *et al*., 2021).

End-to-end latency from event creation through processing represents perhaps the most crucial metric, directly affecting insight timeliness and action speed. This measurement captures the total elapsed time between event occurrence within source systems until processing completion and availability to downstream consumers.

Throughput measurements across producers, brokers, and consumers reveal system capacity and utilization patterns. These metrics require monitoring across multiple pipeline segments to spot bottlenecks while ensuring balanced operation.

Consumer lag—the offset gap between the latest produced messages and current consumer positions—acts as an early warning system flagging processing bottlenecks. Growing lag signals consumers failing to match production rates, potentially causing delayed insights or increased memory pressure.

Resource consumption across CPU, memory, storage, and network infrastructure provides foundation information supporting capacity planning and performance troubleshooting. Shapira notes that correlating infrastructure metrics alongside application-level indicators helps operations teams understand relationships between resource consumption and streaming performance (Shapira, G. *et al*., 2021).

Scaling Strategies

Streaming data systems need to scale up in volume so as not to erode performance or reliability. The appropriate scaling approaches enable horizontal growth and configuration optimization, as well as workload division.

Partition count increases serve as primary mechanisms for improving parallelism within streaming architectures. By splitting topics into additional partitions, systems distribute processing loads across expanded resources. However, Confluent's post-deployment documentation cautions against careless partition increases, noting each partition consumes resources while adding system complexity (Confluent,).

Broker additions distribute storage and processing loads across additional servers, boosting overall system capacity. When adding brokers, operations teams must carefully orchestrate data rebalancing to minimize production workload disruption.

Consumer instance expansion boosts processing capacity through workload distribution across additional application servers. This approach typically avoids broker-side changes, making it among the most straightforward scaling techniques.

Health Monitoring

Proactive health checks must verify system integrity beyond basic metrics, focusing on architectural components and processing guarantees. Shapira recommends implementing comprehensive monitoring spanning infrastructure, platform components, and application-level indicators (Shapira, G. *et al*., 2021).

Broker cluster status and leader distribution monitoring ensure balanced workloads while identifying potential hotspots potentially degrade performance. Uneven leader distribution causes resource contention on specific brokers, creating localized performance problems.

Consumer group balance and processing rates reveal workload distribution effectiveness across consumer instances. Imbalances stem from uneven data distribution, resource constraints, or application issues.

Dead letter queues capturing unprocessable messages provide visibility into data quality problems and application processing failures. By segregating events that resist normal processing, these queues help operations teams address schema violations, corrupt data, or application bugs without losing events.

Alert thresholds for abnormal conditions should reflect business requirements rather than technical specifications. The operational advice provided by Confluent points out that a well-designed alerting system is a critical step in ensuring the good health of the streaming systems, with emphasis put on providing actionable alerts that require human intervention.

When these operational practices are implemented, streaming systems demonstrate the continued ability to perform the same consistently and

Publisher: SARC Publisher

provide reliability when it comes to scaling to fit higher data volumes and more complex processing demands.

CONCLUSION

Immediate patterns in data streams are a paradigm change in data processing and value extraction strategies deployed by organizations. These systems allow unprecedented real-time analytics and decision-making, based upon extension of the concepts of event sourcing and partitioned logs, and stateful processing. With the shift in batch to streaming processing, enterprises are rethinking the way they react to events, optimize operations, and innovate in their experiences across the board or industries, including financial services, healthcare, retail, manufacturing, and more. Because streaming platforms are maturing, they are increasingly becoming part of artificial intelligence, cloud services, and enterprise data ecosystems, and will continue to do so, increasing their applicability and lowering implementation overheads. Innovations in semantics processing, state maintenance, and operational tooling have raised streaming to the status of a key pillar in modern data strategies, where semantics processing, state maintenance, and operational tooling are key components of modern data strategies. Mastering these underlying architectures, implementation patterns, and best practices of how these systems operate becomes critically important to data engineers, architects, and technical leaders who are leading digital transformation and operationalizing data within a more real-time business environment.

REFERENCES

1. Waehner, K. "Top Trends for Data Streaming with Apache Kafka and Flink in 2025." *Medium*, (2025). (https://kai-waehner.medium.com/top-trends-for-data-streaming-with-apache-kafka-and-flink-in-2025-636583892b2d)
2. Karimov, J., Rabl, T., Katsifodimos, A., Samarev, R., Heiskanen, H., & Markl, V. "Benchmarking distributed stream data processing systems." *2018 IEEE 34th international conference on data engineering (ICDE)*. IEEE, (2018).
3. Microsoft, "Event Sourcing Pattern". (https://learn.microsoft.com/en-us/azure/architecture/patterns/event-sourcing)
4. Solace, "The Ultimate Guide to Event-Driven Architecture Patterns". (https://solace.com/event-driven-architecture-patterns/)
5. Kafka, A. "Documentation." (https://kafka.apache.org/documentation/)
6. Richman, J. "Data Streaming Architecture: Components, Process, & Diagrams." *Estuary*, (2025). (https://estuary.dev/blog/data-streaming-architecture/)
7. Kleppmann, M. "Designing data-intensive applications." (2019).
8. Bryant, A. "Processing guarantees in Kafka." *Medium*, (2019). (https://medium.com/@andy.bryant/processing-guarantees-in-kafka-12dd2e30be0e)
9. Akidau, T., Chernyak, S. and Lax, R. "Streaming Systems." *O'Reilly Media*, (2018). (https://www.oreilly.com/library/view/streaming-systems/9781491983867/)
10. Richman, J. "Stateful Stream Processing: Concepts, Tools, & Challenges." *Estuary*, (2025). (https://estuary.dev/blog/stateful-stream-processing/)
11. Shapira, G., Palino, T., Sivaram, R., & Petty, K. “Kafka: the definitive guide”. " *O'Reilly Media*, Inc.", (2021).
12. Confluent, "Best Practices for Kafka Production Deployments in Confluent Platform." (https://docs.confluent.io/platform/current/kafka/post-deployment.html)

Source of support: Nil; **Conflict of interest:** Nil.

Cite this article as:
Desai, P. “Fundamentals of Real-Time Data Streaming Architectures” *Sarcouncil Journal of Engineering and Computer Sciences* 4.9 (2025): pp 348-355.

Publisher: SARC Publisher