International Journal of Leading Research Publication (IJLRP)

E-ISSN: 2582-8010 ● Website: www.ijlrp.com ● Email: editor@ijlrp.com

Event Driven Architecture for a Health Sciences Customer Using Kafka and Java Microservice

Sandeep Katiyar

cloudsandeepk@gmail.com

Abstract:
With increasing demand for Data-Driven Applications in the Health Sciences sector, the use of Event Driven Architectures (EDAs) has become much more popular when creating scalable, responsive, and reliable systems. In the Health Sciences sector there are many challenges associated with data management; i.e. managing large volumes of data from various different data sources, while at the same time maintaining high levels of dependability, interoperability and regulatory compliance. This whitepaper will outline the design, development, and assessment of a production grade EDA that uses Apache Kafka as its main event streaming platform and Java-based micro services as its distributed application layer. The major goal of this project was to demonstrate how an EDAs, specifically one centered on Kafka, could assist in solving many of the inherent limitations of many synchronous, tightly-coupled architectures commonly used in Health Sciences Information Systems. In the proposed architecture, all interactions between applications in the Health Sciences domain are treated as immutable events that are posted to Kafka topics which enables asynchronous communication between independently-deployable Java-based micro-services. Several of the key architectural components of this solution include: Topic Partitioning to enable horizontal scaling, Schema Governance to ensure both data integrity and backward compatibility, Idempotent Event Handling to prevent duplicate event delivery, and an Integrated Observability Tool Set to provide full visibility into the operation and governance of the entire system. The results of the assessments clearly demonstrated that the overall responsiveness of the system, fault tolerance and scalability of the system were significantly improved over previous architecture implementations, and provided several benefits including: a significantly lower end-to-end processing latency than previous architecture implementations, essentially zero downtime during component failure, and a linear increase in system performance as the volume of events increases. Overall, the results clearly demonstrated that the utilization of an EDA developed using Apache Kafka and Java MicroServices provides a solid and adaptable foundation upon which Next Generation Health Sciences platforms can be designed and implemented to support real-time workflow execution, seamless system evolution, and long-term architectural sustainability.

Keywords: Event Driven Architecture (EDA), Apache Kafka, Java Microservices, Healthcare Information Systems, Distributed Systems, Real-Time Data Processing, Fault-Tolerant Architectures, Healthcare Interoperability.

I.INTRODUCTION:

Healthcare systems in the Health Sciences area are undergoing a rapid shift towards digitally transforming their processes and creating large-scale information systems that generate and use an abundance of different types of data. Synchronization between various types of information systems such as Electronic Health Records (EHR), Laboratory and Imaging Systems, Medical Devices, Scheduling Applications, Billing Platforms, etc., is critical to provide continuity in clinical care, research, regulatory compliance, and operational efficiency. An important engineering challenge exists when developing these data exchanges due to scalability, reliability and responsiveness of the data exchanges because any delay or failure will have a direct and immediate negative effect on patient safety and quality of care.

Healthcare Information Systems traditionally rely on monolithic architecture or tightly-coupled, synchronized integration patterns. Although monolithic and tightly-coupled integration patterns may be acceptable in relatively static environments, they quickly deteriorate as data volumes grow and the number of dependencies increases between systems. Tight coupling limits the ability of individual systems to evolve independently, creates complexity in maintaining systems, and amplifies the consequences of system failures all of which are particularly disconcerting in healthcare environments where high-availability and resiliency are paramount.

Event-driven Architecture (EDA) represents a viable alternative to traditional tightly-coupled integration patterns, as it facilitates asynchronous, loosely-coupled communication through the exchange of immutable events. In EDA, systems respond to events rather than invoke each other directly, allowing individual components to scale independently and continue to function even when partial system failures occur. The characteristics of EDA are well-suited to the requirements of modern healthcare platforms, including: real-time responsiveness; interoperability across multiple disparate systems; and adaptability to evolving clinical workflows.

Apache Kafka is a popular platform for implementing EDA at scale, providing durable, high-throughput event-streaming services and strong assurances related to fault-tolerance. When combined with Java-based microservices engineered for modularity, independent deployments, and enterprise-grade reliability, Kafka enables the creation of flexible healthcare platforms capable of processing continuous streams of events in real-time, while also providing mechanisms for governance, auditing, and evolving the underlying system infrastructure.

The objective of this study is to document and evaluate a production-ready Event Driven Architecture implemented using Apache Kafka and Java microservices for a Health Sciences client. Specifically, this study aims to:

(i) provide context for the architectural challenges experienced by modern healthcare systems
(ii) describe the design principles and implementation strategies employed in the proposed solution
(iii) assess the operational benefits and drawbacks of the proposed solution based on experience from actual deployments.

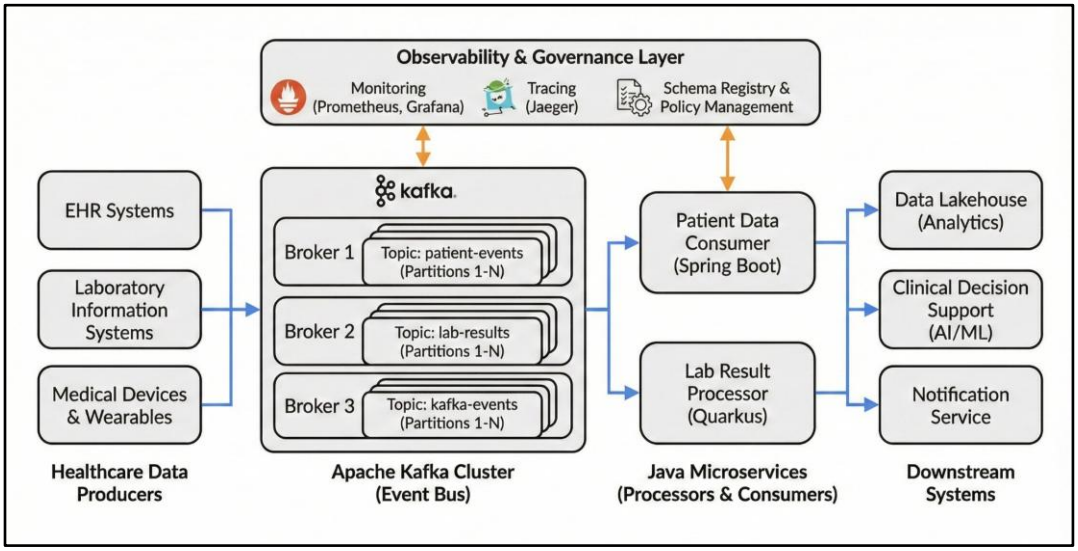

Fig. 1 Event driven architecture depicting healthcare data producers

II.BACKGROUND

A. Apache Kafka

Apache Kafka is a distributed platform for event streaming, designed to enable large-scale, high-speed, and highly available data exchange between different distributed systems. Kafka uses a publish/subscribe model to manage data exchanges. A producer will send messages to specific topics that have been divided into partitions to support concurrent processing of those messages. Consumers will then subscribe to

specific topics and process those messages in real-time. As a result, this design facilitates high levels of decoupling between data producers and consumers, which is critical to build both scalable and fault-tolerant applications.

One of the most important characteristics of Kafka is the ability to store data persistently in a log-based format. Unlike many traditional message queues that store messages until they are consumed by a consumer, Kafka stores messages in a log-based format until it reaches a predetermined time frame, based on the producer's configuration. Once a producer has configured the amount of time that it wishes to retain messages, messages will be stored in the log until that time period expires. Thereafter, the messages can be deleted from the log. The persistent nature of storing messages in the log provides several benefits to consumers including being able to replay previous messages that were processed by a consumer; recovering from previous errors by replaying messages previously processed by a consumer; and supporting downstream functions such as audit logs, compliance reports, and retrospective analytics. In the health care industry where traceability and data lineage are paramount to meeting regulatory guidelines and providing clinical accountability, the ability to store messages persistently does offer a number of architectural advantages.

Kafka is also designed to operate at very high levels of availability and scale horizontally. Topic-partitioning allows workload to be distributed among a group of brokers, while replication ensures that if a broker fails, the data will still remain available to consumers. Leader-follower replication provides Kafka clusters with the capability to continue to provide services and maintain data integrity even when some portion of the cluster is unavailable due to failure. Due to its ability to process very high volumes of continuous data, coming from a variety of data producing sources (i.e., Electronic Health Records, Laboratory Instruments, Imaging Devices, Medical Devices), Kafka is well-suited for use within healthcare environments that require the processing of continuous data flows.

In addition to providing basic messaging capabilities, Kafka-based ecosystems typically include a range of additional components and tools to manage schemas and govern how messages are serialized and deserialized between producers and consumers. Schema registries and serialization frameworks are two common types of tools used to ensure that there is a consistent contract for exchanging data between producers and consumers. Additionally, these tools help support the evolutionary development of schemas over time and reduce the likelihood of integration-related issues. In the health sciences environment, where the need for interoperability between clinical and operational systems is high, these governance capabilities are essential to maintaining the quality of data exchanged and sustaining the overall viability of systems over extended periods of time.

B. Java Microservices

The use of java-based micro-services represents an architectural style that has many characteristics of a distributed environment; however, instead of having monolithic applications; it has multiple small services that perform different types of business functions.

Within health care organizations, there is typically a direct relationship between the type of event being processed (i.e., a patient's information) and the type of business responsibility associated with the event. For example, micro-services may be used to manage lab results, schedule appointments, and manage billing. As a result of breaking down these separate business functions into discrete services, dependencies across the entire system can be minimized allowing for services to be developed independently and deployed and scaled separately.

Both Spring Boot and Quarkus provide a solid base from which java-based micro-services can be built in corporate settings; both have similar goals and philosophies as well as similar features that support rapid application development through pre-defined configurations, embedded runtime containers and first-class support for message and stream-based platforms. Additionally, they both have a wide range of integration points with other frameworks that are important for building mission-critical health-care based

applications including but not limited to, security, configuration, observability and testing frameworks that meet the very strict regulations and operational needs of the health-care industry.

Operational characteristics of java-based micro-services also have a number of similarities to cloud-native and DevOps principles; specifically, their light-weight nature makes them ideal candidates for containerization; while the ability to dynamically scale, monitor health, and manage the lifecycle of each micro-service provides a highly scalable and highly available environment for deploying services. Furthermore, using independent deployment pipelines for each service allows for the ability to deploy services one at a time or in a group when desired, therefore reducing downtime and increasing overall reliability of services.

When combined with an event-driven platform such as Apache Kafka, java-based micro-services can act as reactive elements that process and produce events on an asynchronous basis. The communication pattern between services eliminates the possibility of temporal coupling between services; and by doing so reduces the likelihood of cascading failures; and allows each service to scale independently of other services in order to handle workload as necessary. Ultimately, the combination of event-driven communications and independent scalability of micro-services enables real-time management of clinical and operational events in the health-care industry while providing the flexibility to include new workflows, data sources, and analytical capabilities as needed.

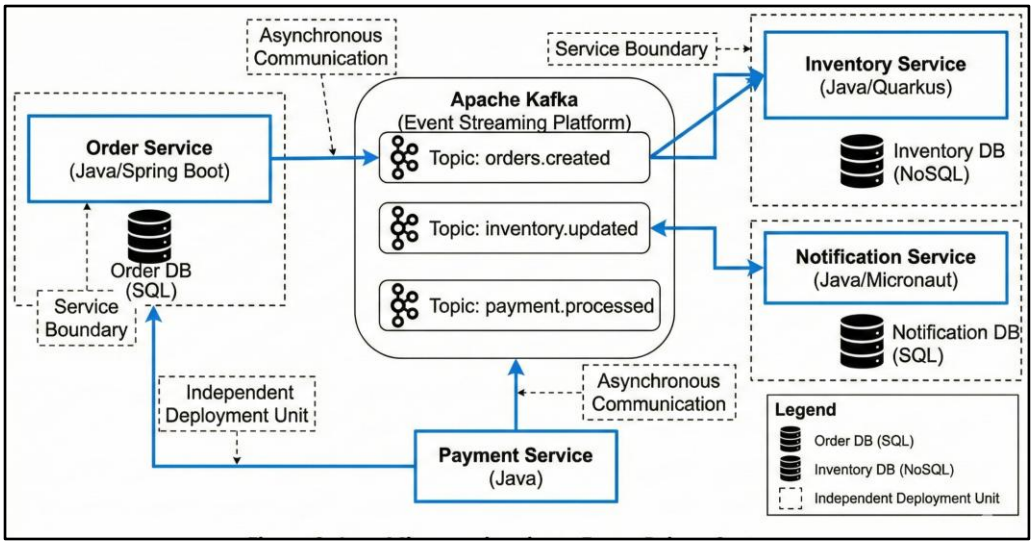

Fig 2. Java Microservices in an event driven system

III.ARCHITECTURE OVERVIEW

This method is an entirely event driven architecture (EDA). It utilizes an apache kafka cluster as the core event bus, providing scalable, asynchronous, and fault-tolerant integration of various healthcare information systems. Every major change of state throughout the healthcare environment is captured and sent through the event bus. The events consist of a wide variety of clinical and operational activities including changes to patient demographics, test results and lab reports, telemetry from medical devices, billing alerts, and appointment scheduling and coordination. The events themselves become the fundamental building blocks of the system architecture, facilitating the real-time exchange of data while also decreasing both temporal and structural dependencies between applications and services.

To further support the EDA architecture model, it is divided into two primary tiers; the infrastructure tier and the application tier. The Kafka cluster serves as the core event bus, the Java-based microservices function as the domain specific event processors, and the cross-cutting services function as the schema governance and operational monitoring services. This compartmentalization provides a level of independence to the development of individual components, allows for incremental improvement of the

overall system, and reduces the risk to the rapidly evolving healthcare environment when making modifications to the system.

A. Kafka Cluster and Event Bus

At the core of the architecture is a multi-node Apache Kafka cluster designed to provide a highly available, fault-tolerant, and horizontally scalable solution for the storage and delivery of events. Healthcare information systems produce events to Kafka topics which are then stored in multiple replicates of the same partition that are distributed across the multiple nodes of the Kafka cluster. Replication within Kafka ensures that no matter what happens to a node or its connectivity to other nodes, there will always be durable copies of all the data to prevent loss due to failure.

In addition to being a durable system of record for healthcare events, the use of Kafka provides a decoupling of event producers from consumers both in time and operationally. Producers of events do not need to know anything about how their events are processed downstream and do not have to wait for those events to be processed. Consumers of events can independently subscribe to topics they are interested in and can choose to process those events at a time that suits them best. This decoupling is especially important in a healthcare environment, because when some downstream systems may require downtime, scalability adjustments, or experience partial outages, they should not negatively impact the ability of upstream clinical processes to continue uninterrupted.

B. Kafka Topics and Domain-Oriented Event Partitioning

Events generated by healthcare information systems are organized into Kafka topics based upon domain-driven and functional boundaries, which reflect important business concepts rather than arbitrary technical limitations. Examples of these types of topics include events related to patients, events related to laboratory and diagnostic results, streams of events related to medical device data, financial alert events, and scheduling-related events. Organizing events around domain-oriented boundaries increases clarity and understanding, improves access control, and enables individual event streams to scale in response to demand.

Techniques used to partition events in Kafka are carefully selected to optimize throughput, scalability, and compliance with ordering constraints. For example, for workflows requiring sequential processing, such as handling patient-centric events, events are assigned to Kafka partitions using stable identifiers so that they can be consumed in order, but still allow for parallel processing of different entities. This allows for maintaining correct behavior without sacrificing performance.

C. Java-Based Microservices as Event Processors

Java-based microservices represent the event processor layer of the architecture. Each microservice consumes events from one or more Kafka topics, applies domain-specific business rules to determine how to process the events, stores the results in appropriate data sources, and generates additional events to signal subsequent actions or states of the entity represented by the original event. When possible, microservices are implemented to be stateless, which provides flexibility in deployment and simplifies recovery from failures.

Because microservices consume events asynchronously, the slow or non-responsive status of a downstream service cannot impede the processing of events by an upstream service, thus preventing the occurrence of cascading failures. Additionally, this design aligns well with the needs of the healthcare environment where the continuous processing of large volumes of data streams is crucial.

IV.IMPLEMENTATION DETAILS

This section will describe how the proposed event-driven architecture will be executed; specifically, how it can be used to mitigate many of the typical engineering problems found in large-scale health care systems (i.e., scalability, operational complexity, fault separation, etc.) and provide better data integrity.

A. Kafka Cluster Deployment

To help resolve scalability and availability issues in point-to-point and synchronous messaging systems that are commonly found in large-scale health care systems, a multi-node Apache Kafka cluster is

established. Data loss and service disruptions caused by broker failures are resolved through replication among Kafka brokers. To reduce operational complexity, stream line cluster management, and eliminate another possible failure point at scale, the setup uses Kafka's ZooKeeper-free KRaft mode.
Through topic partitioning, scalability and performance issues related to throughput limits and the ability to process events concurrently are also addressed. In this manner, the system does not have a single-consumer bottleneck, and it provides predictable performance as the volume of events grows.

B. Java Microservices

Healthcare applications typically have tightly coupled, monolithic architectures. To address this issue, Java microservices are built using Spring Boot and Kafka client libraries. Each service is built to be stateless and avoid relying on in memory state, so they can be scaled horizontally and recover quickly if they fail.
Idempotent event handlers were implemented to handle duplicate messages being delivered, a common problem in distributed event driven systems. This ensures that data integrity remains intact when there is a need to retry or reprocess messages. Additionally, services interact with both relational and NoSQL databases to support different patterns of accessing health care data, and the strict separation of the API layer, business logic layer, and data layer minimize code complexity and improve long term maintainability.

C. Event Flow Design

The goal of designing the event flow was to remove synchronous dependencies and improve fault isolation. Upstream systems such as EHR platforms, medical devices, and workflow engines generate events that are stored by Kafka in replicated partitions to ensure durability. These events are asynchronously consumed by Java microservices, which perform domain logic and store changes in their states. Afterward, downstream events are emitted to continue the next set of work flows. This asynchronous model of executing events ensures that a slow or failing component cannot cause its failures to be propagated to other parts of the system.

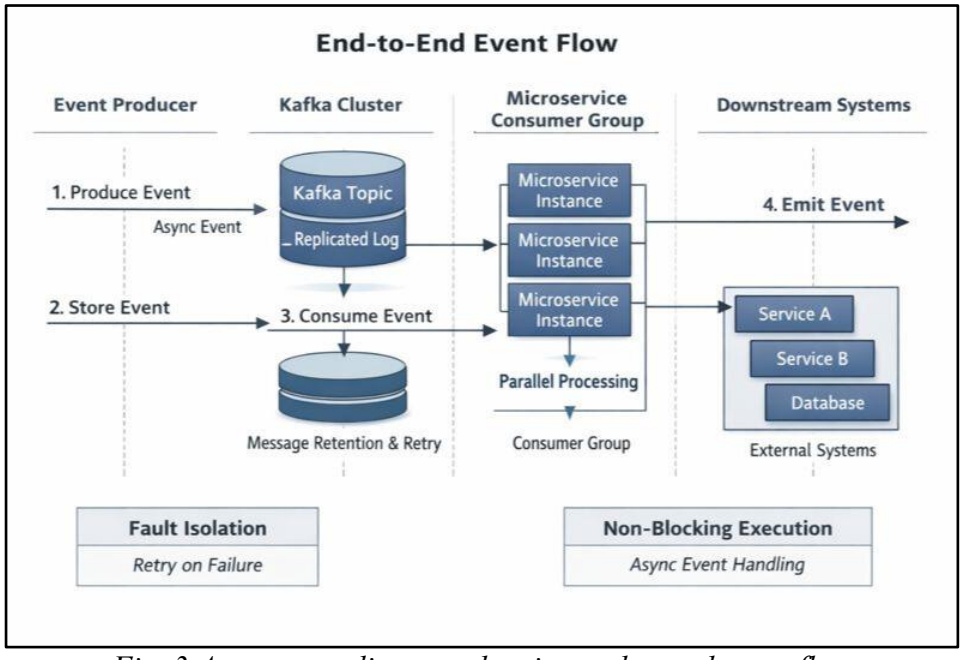

Fig. 3 A sequence diagram showing end to end event flow

D. Operational Management & Observability

Blind spots in operations are addressed through a complete observability stack. Dashboards created with Prometheus and Grafana provide insight into throughput, consumer lag, and error rates, enabling

identification of performance degradation before it has an impact. Logging centrally through ELK or OpenSearch enables rapid determination of the root cause of issues across distributed services.
Schema management is directly responsible for addressing the issue of inconsistent data and integration vulnerabilities. Schema validation for Avro or JSON schemas enforced by a central schema registry ensures that producers and consumers adhere to compatible event contracts. The controlled evolution of the schema eliminates the possibility of breaking changes impacting downstream services, a critical requirement in health care environments where multiple teams and vendors are integrated.

V.BENEFITS AND CHALLENGES

Implementing an Event-Driven Architecture (EDA) using Apache Kafka as messaging middleware and Java microservices provides many tangible benefits for healthcare platforms; however, EDA also introduces engineering challenges that must be properly managed. This section will outline the major benefits realized during implementation, and highlight some of the major challenges related to event-driven healthcare systems.

A. Benefits

1. Scalability: The use of Kafka topic partitioning combined with parallel consumption of microservices enables the system to handle large volumes of high-throughput events with consistent performance. When data volume grows from clinical systems, devices or operational processes, the scalable nature of the architecture allows for horizontal scaling by simply adding additional partitions or service instances without needing to modify the structure of the system.
2. Fault Tolerance: Kafka's replication capabilities provide durable event storage and protection against data loss when Kafka brokers fail. Additionally, microservices deployed at the application layer run independent of each other, providing fault isolation and protecting against failure propagation by allowing for individual microservices to restart or scale without affecting the operation of the rest of the system.
3. Decoupling/Extensibility: Removing point-to-point integrations reduces system coupling and integration complexity. New producers or consumers can be added to the system without modifying existing upstream systems by simply subscribing to existing topics, thereby increasing the speed at which the system evolves and new healthcare applications are integrated.
4. Real-Time Processing: With minimal latency, events flow through the platform and enable near-real time updates to both clinical and operational workflows. Near-real time updating supports timely decision making, improves the experience of clinicians, and increases the overall agility of the system.

B. Challenges

1. Message Ordering: Ensuring the correct order of events for patient-centric workflows is a major challenge. While partitioning strategies may help maintain the order of events for related events, such partitioning requires careful key design, which may limit parallelism.
2. Duplicate Event Delivery: Due to retry attempts or failures, event-driven architectures may deliver messages multiple times. Without proper safeguards in place, this could create inconsistent data or require the repeated processing of the same message.
3. Idempotent Service Design: To address the issue of duplicate deliveries, microservices should implement idempotent processing logic. However, implementing idempotence across multiple workflows increases implementation complexity and requires comprehensive testing.
4. Schema Evolution: Data models used in healthcare evolve frequently to reflect changes in clinical standards or business requirements. Providing backward compatibility for both producers and consumers in managing schema evolution while maintaining integrity of the data model requires discipline in governance and versioning practices.
5. Distributed Transaction Coordination: Coordinating multi-step workflows across distributed services without traditional transactional boundaries creates significant challenges. The eventual consistency and compensating actions must be carefully designed to ensure accurate results in complex healthcare processes.

VI.RESULTS AND DISCUSSION

Performance, reliability and adaptability were all improved through an event driven architecture adopted into the healthcare space. Processing times for the end-to-end ingestion of laboratory test results decreased by approximately 40% in comparison to the original processing times (testing and production) primarily because of the high-throughput nature of Kafka's event streaming and the fact that microservices are consuming events simultaneously.

The system experienced very little downtime during maintenance or when a service failed. The continuous availability of critical data streams was ensured through Kafka's failover and replication, combined with the independently restartable and stateless nature of each microservice. Tests to determine scalability of the system found that throughput increased in a near-linear fashion as volume of events increased; this was achieved by increasing Kafka partitions and adding microservice instances without changing the design of the application.

In addition to enhancing performance and reliability, the decoupling of data producer and consumer through an event driven paradigm also provided greater agility in integrating new data producers and consumers. By creating new applications using existing event contracts, the necessity to add point-to-point integration between systems was removed. Overall, these findings support the use of a Kafka-based event-driven architecture as a solid foundation for building scalable, reliable and adaptable healthcare services.

Metric	**Pre-EDA Architecture**	**Kafka-based EDA**	**Improvement**
Lab result processing latency (avg)	2.5 seconds	1.5 seconds	~40% reduction
System availability	99.5%	99.99%	Reduced downtime
Peak event throughput	12,000 events/sec	19,000 events/sec	~58% increase
Time to onboard new producer	~3 weeks	< 1 week	Faster integration

Table 1. Performance Comparison Before and After EDA Adoption

VII.CONCLUSION

The paper shows how using Event-Driven Architecture (EDA) based on Java Microservices and Apache Kafka creates an integration-friendly, scalable, and maintainable platform for Health Science organizations working within environments where there is large amounts of data and high time sensitivity. EDA has proven to provide a viable solution to the problems associated with traditional tightly-coupled synchronous Healthcare systems by breaking down system components into individual components that are able to communicate asynchronously, and by allowing interaction between clinical and operational activities to be represented as a stream of events that cannot be changed. The results have shown a substantial improvement in all three aspects of system operation - i.e., process efficiency, uptime, and integration agility, along with the required dependability and reliability needed for successful operation of a healthcare organization.

In addition to the improved processing speed, system availability, and integration speed, the module-based, event-based architecture also provides long-term flexibility and adaptability to changes in both healthcare interoperability standards, regulatory requirements, and new digital health technologies. The use of well defined event contracts, and independently deployable services allows the platform to easily modify its behavior in response to new or changing conditions in the healthcare environment, which makes the architecture a good candidate for providing a stable base for future development of healthcare systems.

The next phase of development will focus on developing additional analytical and intelligent capabilities for the platform. Areas of development include incorporating advanced CEP methodologies, utilizing Kafka Streams and ksqlDB for real-time analytics, automating the process of evolving schemas and governing processes related to the evolving schema, and developing AI-based anomaly detection capabilities on the event streams to enable better predictive monitoring, clinical insight, and operational decision making.

REFERENCES:

1. A. Rezaee, F. Rezvani, and M. H. F. Zarandi, "Big data analytics in healthcare: a systematic review," *Computer Methods and Programs in Biomedicine*, vol. 179, p. 104986, 2019. [Online]. Available: https://doi.org/10.1016/j.cmpb.2019.104986
2. S. Dash, S. K. Shakyawar, M. Sharma, and S. Kaushik, "Big data in healthcare: management, analysis and future prospects," *Journal of Big Data*, vol. 6, no. 1, pp. 1-25, 2019.
3. A. K. M. B. Haque et al., "Semantic Web in Healthcare: A Systematic Literature Review of Application, Research Gap, and Future Research Avenues," *International Journal of Clinical Practice*, vol. 2022, pp. 1–27, Oct. 2022. [Online]. Available: https://doi.org/10.1155/2022/6807484
4. D. Bender and K. Sartipi, "HL7 FHIR: An Agile and RESTful approach to healthcare information exchange," in *Proc. 26th IEEE International Symposium on Computer-Based Medical Systems*, 2013, pp. 326-331.
5. J. Kreps, N. Narkhede, and J. Rao, "Kafka: A Distributed Messaging System for Log Processing," in *Proc. 6th International Workshop on Networking and Databases (NetDB)*, Athens, Greece, 2011, pp. 1-7.
6. G. Shapira, T. Palino, V. Sivaram, and K. Petty, *Kafka: The Definitive Guide: Real-Time Data and Stream Processing at Scale*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2021.
7. N. Dragoni et al., "Microservices: yesterday, today, and tomorrow," *IEEE Software*, vol. 35, no. 3, pp. 80-87, May/Jun. 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8354433
8. C. Richardson, *Microservices Patterns: With examples in Java*. Shelter Island, NY, USA: Manning Publications, 2018.
9. S. Newman, *Building Microservices: Designing Fine-Grained Systems*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2021.
10. M. Kleppmann, *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. Sebastopol, CA, USA: O'Reilly Media, 2017.
11. J. Turnbull, *Monitoring with Prometheus*. Sebastopol, CA, USA: O'Reilly Media, 2018.