АКТУАЛЬНІ ПИТАННЯ ТЕХНІЧНИХ НАУК

UDC 004.75:336.76 DOI https://doi.org/10.35546/kntu2078-4481.2025.1.2.51

D. RUMIANTSEV
Principal IT Consultant, Meng
ORCID: 0009-0007-1537-6683

DESIGNING A HIGH-THROUGHPUT MICROSERVICES PLATFORM FOR FINANCIAL RISK DATA ENRICHMENT: A CASE STUDY

Financial institutions increasingly rely on rapidly processing vast, heterogeneous data streams for effective risk management and regulatory compliance. However, integrating and enriching this data in near real-time presents significant architectural challenges, particularly at scale.

This paper details the design, implementation, and impact of a high-availability, microservices-based platform developed to automate data processing for a central Eastern European bank. The primary objective was to create a modular, scalable, and fault-tolerant system capable of managing massive data volumes while ensuring data integrity and low-latency delivery of insights. A case study methodology was employed to document the system's architecture. The platform utilizes an event-driven, asynchronous model built with Java and the Spring Boot framework. Core components include decoupled microservices for data ingestion, normalization, enrichment, and delivery, orchestrated via RabbitMQ message queues. A novel dispatcher service was implemented to manage entity-level concurrency control, preventing race conditions during parallel data processing. The system's performance and health are monitored through an integrated observability stack comprising Prometheus, Grafana, Loki, and Zipkin.

The platform successfully processes over 160 million data units daily from over 20 disparate sources. It reduced data processing latency from several days to under 15 minutes and consistently meets a critical service-level agreement (SLA) of delivering search query results in less than 10 seconds. Automation eliminated manual data handling, reduced data quality error rates by approximately 87%, and significantly enhanced the bank's ability to detect adverse financial events in near real-time.

This case study validates the efficacy of a microservices architecture combined with explicit concurrency control for building high-throughput FinTech data platforms. The presented design is a practical blueprint for engineering resilient, scalable, and observable systems to address complex data integration challenges in the financial sector and other data-intensive industries.

Key words: *microservices, fintech, risk management, data enrichment, event-driven architecture, high-throughput systems, concurrency control.*

Д. РУМЯНЦЕВ
Головний IT-консультант, MEng
ORCID: 0009-0007-1537-6683

ПРОЄКТУВАННЯ ВИСОКОПРОДУКТИВНОЇ МІКРОСЕРВІСНОЇ ПЛАТФОРМИ ДЛЯ ЗБАГАЧЕННЯ ДАНИХ ФІНАНСОВИХ РИЗИКІВ: КЕЙС-СТАДІ

Фінансові установи дедалі частіше покладаються на швидке опрацювання великих, гетерогенних потоків даних для ефективного управління ризиками та дотримання регуляторних вимог. Однак інтеграція й збагачення цих даних у режимі, наближеному до реального часу, створюють суттєві архітектурні виклики, особливо у масштабованих системах.

У статті описано проєктування, реалізацію та результати впровадження високопродуктивної платформи на основі мікросервісної архітектури, розробленої для автоматизації оброблення даних у центральноєвропейському банку. Основна мета полягала у створенні модульної, масштабованої та відмовостійкої системи, здатної керувати великими обсягами даних із забезпеченням їхньої цілісності та мінімальної затримки при формуванні аналітичних інсайтів. Методологічною основою дослідження став підхід case study, який дозволив детально задокументувати архітектуру системи. Платформа реалізована за подієво-орієнтованою (event-driven), асинхронною моделлю з використанням мови Java та фреймворку Spring Boot. Ключові компоненти включають декупльовані мікросервіси для етапів надходження, нормалізації, збагачення та доставки даних, які координуються через черги повідомлень RabbitMQ. Для забезпечення контролю конкурентного доступу на рівні сутностей впроваджено dispatcher service, що запобігає колізіям під час паралельної обробки даних. Моніторинг продуктивності та стану системи здійснюється через observability stack, який включає Prometheus, Grafana, Loki та Zipkin.

Розроблена платформа обробляє понад 160 мільйонів одиниць даних щодня з більш ніж 20 різнорідних джерел, скорочуючи затримку оброблення з кількох днів до менше ніж 15 хвилин. Система стабільно виконує кри-

тичні вимоги SLA, забезпечуючи результати пошукових запитів менш ніж за 10 секунд. Автоматизація процесів усунула ручну обробку, знизила рівень помилок якості даних приблизно на 87% та значно підвищила здатність банку виявляти негативні фінансові події у майже реальному часі.

Наведене дослідження підтверджує ефективність поєднання мікросервісної архітектури з явним контролем конкурентності при створенні високопродуктивних FinTech-платформ для оброблення даних. Представлений дизайн може слугувати практичною дорожньою картою для інженерії масштабованих, надійних та спостережуваних систем, здатних вирішувати складні завдання інтеграції даних у фінансовому секторі та інших галузях із високою інтенсивністю даних.

Ключові слова: *мікросервіси, фінтех, управління ризиками, збагачення даних, подієво-орієнтована архітектура, високопродуктивні системи, контроль конкурентності.*

Introduction

In the contemporary financial landscape, the velocity, volume, and variety of data have become defining factors for competitive advantage and operational resilience [1, с.112]. Risk management departments within banking institutions are particularly dependent on the timely acquisition and analysis of high-quality information to assess counterparty risk, comply with anti-money laundering (AML) regulations, and detect emergent threats [2]. This data originates from a complex ecosystem of internal legacy systems, such as data warehouses, and many external sources, including government registries, sanctions lists, and open-data portals [3]. The consolidation and processing of these heterogeneous datasets pose a substantial engineering challenge, characterized by inconsistencies in data structure, reliability, and update frequency.

This study addresses an operational challenge faced by a leading Eastern European financial institution: the need to process over 160 million data units daily from more than twenty disparate sources. The existing manual and semi-automated workflows could not handle this scale, leading to significant delays between data availability and its delivery to decision-makers. This latency impeded the bank's ability to react swiftly to adverse customer or counterparty risk profile changes. Consequently, the strategic goal was to engineer a fully automated, modular, and fault-tolerant platform capable of ingesting, validating, enriching, and distributing this data in near real-time.

The existing body of literature extensively covers the theoretical benefits of microservice and event-driven architectures [4,5]. Yet, there is a paucity of detailed, real-world case studies demonstrating their application to high-throughput, concurrent data enrichment tasks within the banking sector's stringent security and regulatory environment. This paper aims to fill that research gap by presenting a comprehensive architectural blueprint and implementation analysis of the developed platform.

The practical significance of this work lies in its detailed exposition of a solution that balances scalability, resilience, and data integrity. It provides a tangible example of decomposing a complex data pipeline into specialized microservices, managing concurrency in a distributed environment to prevent data corruption, and embedding observability as a core feature to ensure operational transparency and SLA compliance. This case study is intended to serve as a valuable resource for software architects, engineers, and information technology leaders in the financial industry, as well as an instructional tool for academics and students exploring modern FinTech system design.

Literature Review The challenges this research addresses are at the intersection of financial technology (FinTech), big data processing, and software architecture. The decision to adopt a microservices-based approach is grounded in established principles of software engineering that advocate for modularity and separation of concerns to manage complexity in large-scale systems [4]. Unlike monolithic architectures, a microservice architecture decomposes an application into a collection of loosely coupled, independently deployable services [5]. This paradigm is well-suited for enterprise environments like banking, where different business capabilities (e.g., data ingestion, enrichment, and user interface) can be developed, scaled, and maintained by separate teams, thereby increasing organizational agility [6].

The processing of high-volume, real-time data streams necessitates a scalable and resilient architectural pattern. Event-driven architectures (EDAs), often implemented using message brokers like RabbitMQ or Apache Kafka, have emerged as a dominant solution [7]. In an EDA, services communicate asynchronously by producing and consuming event messages, which decouple them in time and space. This decoupling prevents failures in one service from cascading and impacting the entire system, allows for the absorption of workload bursts, and enables individual services to be scaled horizontally based on demand [8]. Our use of RabbitMQ for orchestrating workflows aligns directly with these established benefits.

A critical challenge in parallel data processing systems is ensuring data consistency, especially when multiple services may attempt to modify the same data entity concurrently. While distributed transaction protocols exist, they often introduce significant performance overhead and complexity (9). An alternative approach, and the one explored in this case study, is to implement explicit concurrency control mechanisms. Our design of a centralized Dispatcher service that manages entity-level locks with time-to-live (TTL) leases is a practical implementation of optimistic locking patterns adapted for a distributed, high-throughput environment. This avoids the bottlenecks of global locks while preventing the race conditions that can lead to data corruption.

Finally, the operational viability of any distributed system hinges on its observability—the ability to infer its internal state from external outputs (10). A modern observability stack typically integrates three pillars: metrics, logs, and traces

[11]. Tools such as Prometheus for time-series metrics, Loki for log aggregation, and Zipkin for distributed tracing have become industry standards for providing deep insights into system performance and behavior. This research demonstrates the integration of such a stack as a foundational architecture component, essential for troubleshooting, performance tuning, and ensuring compliance with stringent service-level agreements (SLAs). The synthesis of these concepts – microservices, event-driven communication, explicit concurrency control, and built-in observability – forms the theoretical foundation upon which our platform was constructed, addressing the specific research question of how to architect a resilient system for real-time financial data enrichment at scale.

Methodology**.** This study employs a descriptive case study methodology to provide an in-depth analysis of a bespoke data enrichment platform's design, implementation, and operational outcomes. The case is centered on a central Eastern European bank, which provides a real-world context for the technical and business constraints inherent in the financial sector. The research involved the system's architectural design, development, deployment, and performance monitoring over its operational lifecycle.

The architectural design was fundamentally shaped by the challenge of processing over 160 million data records daily from more than 20 heterogeneous sources, including legacy systems like an Oracle Data Warehouse. The system must apply complex business rules and deliver near real-time updates to end-users while enforcing strict role-based access control. These demanding functional requirements necessitated an architecture capable of immense scale and complexity.

Even more influential were the non-functional requirements for horizontal scalability, high fault tolerance, and comprehensive observability, all within the stringent security posture mandated by the financial industry. A critical service-level agreement (SLA) requiring search queries to complete in under 10 seconds made performance a first-class architectural driver. Faced with these constraints, a traditional monolithic approach was deemed untenable. Instead, the decision was made to adopt a microservices architecture, supported by literature from thought leaders like Fowler and Lewis [4], who argue it enables the decomposition of complex applications into manageable, independently deployable services. As detailed by Newman in Building Microservices [5], this approach is well-documented for enhancing scalability and fostering the organizational agility necessary in modern service organizations, a concept explored by van Oosterhout [6].

Five core principles guided this architectural choice. First, concerns were separated by isolating distinct functionalities into independent microservices. Second, asynchronous workflows were implemented using message queues, a key event-driven architecture (EDA) pattern for decoupling services and building resilient systems, as described in foundational texts like Enterprise Integration Patterns by Hohpe and Woolf [8]. Third, all data mutation operations were designed to be idempotent, ensuring they could be safely retried without causing data corruption – a crucial feature for achieving fault tolerance in large-scale systems, as emphasized by Kleppmann in Designing Data-Intensive Applications [9]. Fourth, observability by default was embedded from the project's inception by integrating monitoring, logging, and tracing. This principle is a cornerstone of Site Reliability Engineering (SRE), a discipline extensively detailed in publications by Beyer et al. [10, 11], and is essential for operating complex distributed systems and meeting SLAs. Finally, security and compliance were addressed through centralized authentication and immutable audit trails.

The platform was developed using Java and the Spring Boot framework, a mature ecosystem for building production-grade microservices. As illustrated in Figure 1, the architecture consists of specialized services – such as the Importer, Enricher, and Dispatcher – that communicate asynchronously via RabbitMQ. This decomposition allows for independent scaling; for instance, the Enricher instances can be increased to handle high processing loads without affecting other system parts. The multi-stage, event-driven pipeline ensures an orderly and fault-tolerant flow of data. A critical innovation was the Dispatcher service, which manages concurrency by providing entity-level locks with a time-to-live (TTL) lease. This is a pragmatic approach to preventing race conditions and ensuring data integrity in a high-throughput environment, avoiding the complexity and performance overhead of distributed transactions often discussed in data system design literature.

As a single-case study, the findings are specific to this context and are not intended to be universally generalizable. The study describes a successfully implemented system rather than comparing multiple architectures through controlled experiments.

Results

Implementing the microservices-based platform yielded significant, measurable improvements across operational performance, strategic outcomes, and resource efficiency. The results demonstrate the achievement of the project's primary objectives.

The platform successfully met its primary throughput objective, consistently processing over 160 million records daily from more than 20 heterogeneous sources without data loss or integrity issues. The most significant impact was observed in the reduction of data processing latency. The automation of the data pipeline reduced the end-to-end time required to make a newly available dataset accessible to analysts from a baseline of 1-3 days to under 15 minutes. This represents a performance improvement of over 95%. This magnitude of improvement is a practical validation of the theoretical benefits of scalability and parallelization promised by modern event-driven and microservice architectures in data-intensive environments [5, 7, 9].

Furthermore, the system consistently met the critical service-level agreement (SLA) for user-facing queries. Despite high concurrent usage, over 99% of all search queries returned results in under 10 seconds, providing analysts with the near real-time data access required for effective risk monitoring. The key performance metrics are summarized below.

Fig. 1. System Architecture Overview

Table 1

Summary of Key Performance Improvements

Metric	Before Implementation	After Implementation	Improvement
Data Processing Time	1–3 days	< 15 minutes	>95% reduction
Daily Processing Capacity	< 10 million records	160+ million records	16× increase
Search Query Response Time	30–60 seconds	< 10 seconds (p95)	3–6× faster
Data Quality Error Rate	~8%	< 1%	~87% reduction

A key performance indicator for the system was the user-facing search query response time. Through a combination of optimized database indexing (including B-tree and GIN indexes), query profiling, and the use of materialized views for frequently accessed data, the platform consistently met its SLA of <10 seconds for search responses, a 3-6x improvement over the previous system's 30-60 second response times.

The platform significantly enhanced the bank's risk management capabilities. The system expanded the breadth of risk monitoring coverage by facilitating the rapid integration of new external data sources, such as government open-data portals and commercial registries. The near-real-time processing pipeline reduced the latency for detecting adverse events, such as a counterparty's inclusion in a sanctions list, enabling analysts to take immediate action.

Furthermore, the automated data validation and rule-based enrichment engine led to a marked improvement in data quality. The system automatically corrected common formatting errors, deduplicated records, and augmented entities

with supplementary information, reducing the estimated data quality error rate from approximately 8% to less than 1%. Implementing the Dispatcher service for entity-level concurrency control successfully eliminated the risk of data corruption due to conflicting updates, a critical requirement for maintaining data integrity in the bank's core risk database.

Beyond performance metrics, the platform delivered substantial strategic benefits. The automated data validation and enrichment processes markedly improved data quality and consistency, reducing the risk of errors in downstream analytical models. This enhanced data reliability empowered the bank's risk management teams to make faster, more confident decisions.

The project also drove significant cost efficiencies. The automation of tasks previously requiring manual intervention freed up an estimated 15-20 hours of skilled analyst labor per week, allowing personnel to be reallocated to higher-value analytical and strategic activities. On the infrastructure side, the platform's containerized design and auto-scaling capabilities led to a 30% reduction in cloud computing costs compared to the previous, statically provisioned system. This demonstrates the platform's ability to optimize resource utilization by dynamically adjusting to workload demands, achieving a strong return on investment.

The results presented in this case study offer significant insights into the practical application of modern software architecture principles to solve complex data processing challenges in the financial sector. The successful deployment of this platform validates the hypothesis that a well-designed microservices architecture can meet the dual requirements of high throughput and stringent data integrity.

The 16-fold increase in daily processing capacity and the dramatic reduction of processing latency from days to minutes directly result from the architectural choice to parallelize workloads across stateless, independently scalable services (Importer, Enricher). This finding aligns with the theoretical benefits of microservices espoused by Fowler and Lewis (4), demonstrating that the pattern's advantages in scalability and resilience are achievable in a real-world, mission-critical environment. Using an asynchronous, message-driven communication model via RabbitMQ was instrumental in achieving this outcome by decoupling services and creating a resilient data pipeline to buffer load spikes and tolerate transient failures of individual components.

One of the most novel contributions of this work is implementing the centralized Dispatcher service for managing concurrency. In many distributed systems, ensuring data consistency is often addressed with complex distributed transaction protocols or by sacrificing parallelism. Our explicit, lease-based entity-level locking approach provides a pragmatic and highly effective alternative. It successfully prevented race conditions without creating the performance bottlenecks associated with more pessimistic locking strategies, a crucial factor in maintaining high enrichment throughput. This finding suggests that a dedicated coordination service can offer a superior trade-off between consistency and performance for entity-centric data processing workflows compared to more generic solutions (9).

The consistent achievement of the sub-10-second search SLA underscores the importance of designing for performance from the outset rather than treating it as an afterthought. This was not merely a result of using a robust database but a deliberate strategy involving schema design, selective denormalization, aggressive indexing with PostgreSQL's GIN indexes, and materialized views. This reinforces the principle that SLAs must be treated as first-class architectural drivers. Furthermore, integrating a comprehensive observability stack (Prometheus, Grafana, Loki, Zipkin) proved invaluable. It provided the feedback loop necessary for continuous performance tuning and rapid incident resolution, reducing the Mean Time to Resolution (MTTR) for operational issues. This aligns with contemporary DevOps and Site Reliability Engineering (SRE) practices, which posit that a system's operability is as critical as its functional features [10,11].

Despite its success, the study has limitations. The solution is tailored to a specific set of legacy systems and data sources, and the bank's infrastructure constraints dictated the choice of an on-premises deployment. A cloud-native implementation might leverage different technologies (e.g., serverless functions, managed message queues) that could offer alternative cost and operational complexity trade-offs. Future research could compare different concurrency control patterns (e.g., optimistic locking with versioning vs. the lease-based model) in a similar high-throughput context. Additionally, the rule-based enrichment engine could be augmented with machine learning models for predictive risk scoring and anomaly detection, representing a promising avenue for further enhancement of the platform's capabilities.

Conclusion

This paper has presented a detailed case study on the architecture and implementation of a high-throughput data enrichment platform for a significant financial institution. By adopting a microservices architecture, event-driven communication, and an explicit concurrency control mechanism, the developed system successfully addressed the formidable challenge of processing over 160 million daily records from many heterogeneous sources in near real-time. The platform achieved significant quantitative improvements in processing speed, data quality, and system scalability and delivered substantial strategic value by enhancing the bank's risk detection capabilities and regulatory compliance posture.

This work's main contribution is providing a practical, evidence-based blueprint for engineering resilient and performant data processing systems in the demanding FinTech domain. It demonstrates that the theoretical advantages of modular design and asynchronous communication can be translated into tangible operational success. The novel Dispatcher service for entity-level locking, in particular, offers a valuable pattern for managing concurrency in distributed data-intensive applications.

The prospects for further research and development are rich. Future iterations of the platform will explore the integration of machine learning algorithms for more sophisticated, predictive enrichment and anomaly detection. Further optimization of the data pipeline and resource management could be achieved through adaptive concurrency policies that respond dynamically to workload patterns. Ultimately, this project is a compelling testament to the power of disciplined software engineering and modern architectural principles in transforming complex business challenges into robust, scalable, and operationally transparent technological solutions.

Список використаної літератури

1. Abbasi A., Sarker S., Chiang R. H. L. Big data research in information systems: Toward an inclusive research agenda. Journal of the Association for Information Systems. 2016. Vol. 17, No. 2. P. i–xxxii.
2. Site reliability engineering: How Google runs production systems / Beyer B., Jones C., Petoff J., Murphy N. R. (Eds.). 1st ed. O'Reilly Media, 2016.
3. The site reliability workbook: Practical ways to implement SRE / Beyer B., Murphy N. R., Rensin D. K., Kawahara K., Thorne S. (Eds.). 1st ed. O'Reilly Media, 2018.
4. Fowler M., Lewis J. Microservices: A definition of this new architectural term. 2014. URL: https://martinfowler.com/articles/microservices.html (дата звернення: 04.11.2025).
5. Hohpe G., Woolf B. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. 1st ed. Addison-Wesley Professional, 2003.
6. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. 1st ed. O'Reilly Media, 2017.
7. McAfee A., Brynjolfsson E. Big data: The management revolution. Harvard Business Review. 2012. Vol. 90, No. 10. P. 60–68, 128.
8. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. O'Reilly Media, 2021.
9. Philippon T. The FinTech Opportunity. Report No. 22476. National Bureau of Economic Research, 2016. DOI: https://doi.org/10.3386/w22476
10. Richards M., Ford N. Fundamentals of Software Architecture: A Modern Engineering Approach. 2nd ed. O'Reilly Media, 2025.
11. van Oosterhout M. P. A. Business Agility and Information Technology in Service Organizations. Doctoral dissertation. Erasmus University Rotterdam, 2010.

REFERENCES

1. Abbasi, A., Sarker, S., & Chiang, R. H. L. (2016). Big data research in information systems: Toward an inclusive research agenda. Journal of the Association for Information Systems, 17(2), i–xxxii.
2. Beyer, B., Jones, C., Petoff, J., & Murphy, N. R. (Eds.). (2016). Site reliability engineering: How Google runs production systems (1st ed.). O'Reilly Media.
3. Beyer, B., Murphy, N. R., Rensin, D. K., Kawahara, K., & Thorne, S. (Eds.). (2018). The site reliability workbook: Practical ways to implement SRE (1st ed.). O'Reilly Media.
4. Fowler, M., & Lewis, J. (2014, March 25). Microservices: A definition of this new architectural term. https://martinfowler.com/articles/microservices.html
5. Hohpe, G., & Woolf, B. (2003). Enterprise integration patterns: Designing, building, and deploying messaging solutions(1st ed.). Addison-Wesley Professional.
6. Kleppmann, M. (2017). Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems (1st ed.). O'Reilly Media.
7. McAfee, A., & Brynjolfsson, E. (2012, October). Big data: The management revolution. Harvard Business Review, 90(10), 60–68, 128.
8. Newman, S. (2021). Building microservices: Designing fine-grained systems (2nd ed.). O'Reilly Media.
9. Philippon, T. (2016). The FinTech opportunity (Report No. 22476). National Bureau of Economic Research. https://doi.org/10.3386/w22476
10. Richards, M., & Ford, N. (2025). Fundamentals of software architecture: A modern engineering approach (2nd ed.). O'Reilly Media.
11. van Oosterhout, M. P. A. (2010). Business agility and information technology in service organizations [Doctoral dissertation, Erasmus University Rotterdam].