A HYBRID FRAMEWORK FOR CONTEXT-AWARE AUDITING IN EVENT-DRIVEN MICROSERVICE ARCHITECTURES

https://doi.org/10.70728/tech.v3.i04.012

Izzat Madaminov
Department of Software Engineering, Urgench State University named after Abu Rayhan Biruni, Urgench, Uzbekistan
izzatbek.dev@gmail.com
Oybek Allamov
Department of Software Engineering, Urgench State University named after Abu Rayhan Biruni, Urgench, Uzbekistan
oybek.allamov@gmail.com

ABSTRACT In distributed microservice architectures, maintaining a consistent and domain-aware audit trail is challenged by database isolation and the trade-offs between context capture and system performance. This paper proposes a hybrid auditing framework that integrates Hibernate event listeners with an Apache Kafka-based asynchronous transport layer. By capturing change events at the application layer, the system preserves critical domain context—such as user identity and localized field labels—while ensuring horizontal scalability and non-blocking performance. Empirical evaluations demonstrate that the framework handles 50,000 events per minute with sub-500ms latency, offering a production-ready solution for high-integrity enterprise environments. The framework was successfully validated through a real-world deployment in a utility billing system, where it processed over 125,000 change events with 100% correctness.

Keywords: Microservices, Database Auditing, Apache Kafka, Hibernate, Event-Driven Architecture, Observability.

1. INTRODUCTION

The shift towards microservice architectures has revolutionized enterprise software development, enabling independent deployment and elastic scalability of business-critical components. However, this decentralization has introduced significant challenges in maintaining system-wide observability, particularly concerning data auditing and regulatory compliance. In monolithic systems, centralized auditing is facilitated by a unified database persistence layer; in contrast, the "database-per-service" pattern inherent to microservices renders such centralized approaches obsolete, forcing a fragmented and inconsistent approach to audit trails.

Auditing is essential in high-stakes domains such as banking, healthcare, and public utilities, where accountability and traceability are legal requirements. When a state change occurs in a distributed system, failing to track the origin, purpose, and impact of the operation can lead to severe data inconsistencies and debugging difficulties. While

open-source tools like Hibernate Envers, Javers, and Debezium attempt to address these needs, empirical literature—such as the systematic comparison conducted by Tahir and Hussain (2022)—reveals that none of these tools simultaneously satisfy the requirements of domain-context preservation, loose coupling, and asynchronous event transport.

Specifically, infrastructure-level Change Data Capture (CDC) platforms like Debezium provide excellent scalability but are "domain-blind," failing to capture necessary application-level context such as user identity. Conversely, ORM-centric tools like Hibernate Envers capture application context but introduce synchronous bottlenecks that violate microservice autonomy and scalability.

This paper proposes a novel hybrid framework that bridges this gap. By leveraging Hibernate event listeners for application-layer change detection and Apache Kafka for decoupled, asynchronous event transport, our approach allows microservices to emit enriched, context-aware audit events without blocking primary business transactions. The remainder of this paper details the architectural design, evaluates the system's performance under high transaction volumes, and demonstrates its effectiveness in maintaining a reliable audit trail across distributed environments.

2. RELATED WORK

The challenge of auditing in distributed systems has led to the development of various architectural patterns, ranging from application-level interceptors to infrastructure-level log sniffers. This section categorizes and evaluates these approaches based on their suitability for microservice environments.

2.1 Application-Layer Auditing Tools

Frameworks such as **Hibernate Envers** and **Javers** operate within the application context. Envers provides a robust revision-tracking mechanism for JPA entities by intercepting lifecycle events. However, its synchronous nature introduces significant latency, as audit writes occur within the primary business transaction. Furthermore, both tools lack native asynchronous transport, leading to tight coupling between the domain service and the audit storage.

2.2 Infrastructure-Level Change Data Capture (CDC)

Debezium represents the state-of-the-art in infrastructure-level auditing. By reading database transaction logs (e.g., PostgreSQL WAL), Debezium streams changes to a message broker like Apache Kafka . While this ensures "completeness" (capturing changes from all sources) and high scalability, it suffers from "domain-blindness" . CDC events lack the application-level semantic context, such as the authenticated user ID or the business intent behind a change, which are critical for compliance-grade audit trails.

2.3 Observability and Distributed Tracing

Modern observability stacks, including **ELK (Elasticsearch, Logstash, Kibana)** and **OpenTelemetry**, offer cross-service log aggregation. While these tools are excellent for operational monitoring, they are not designed for structured data auditing. They typically lack native support for point-in-time state comparisons or field-level "old-vs-new" value pairs, which are foundational requirements for audit integrity.

2.4 The Research Gap

The existing literature highlights a clear trade-off: application-layer tools provide rich context but poor scalability, while infrastructure-level CDC provides scalability but poor context. Tahir and Hussain (2022) confirm that a solution combining domain-context preservation with asynchronous, loosely coupled transport remains largely unaddressed in current open-source offerings. This paper addresses this gap by proposing a hybrid model that utilizes application hooks for context and Kafka for scalable, asynchronous transport.

3. METHODOLOGY AND SYSTEM ARCHITECTURE

The proposed solution is designed as a decoupled, event-driven pipeline that bridges the gap between application-level domain awareness and infrastructure-level scalability. The methodology centers on a three-tier architectural pattern: the **Domain Tier**, the **Messaging Tier**, and the **Auditing Tier**.

3.1 Architectural Design (The Listener-Broker-Consumer Pattern)

Unlike traditional synchronous auditing, our framework utilizes an asynchronous pipeline consisting of four distinct stages:

1. **Event Generation:** We utilize Hibernate 6.x event listeners (specifically PostInsertEventListener and PostUpdateEventListener) to intercept entity lifecycle events. This occurs at the point of transaction commit, ensuring that only successful data mutations are recorded.
2. **Contextual Enrichment:** Upon interception, the AuditChangeProcessor extracts the authenticated user context from the Spring Security SecurityContextHolder. This allows the audit log to include the "who" and "why" behind a change—data typically inaccessible to database-level CDC tools.
3. **Asynchronous Transport:** The enriched data is serialized into a JSON payload and published to an Apache Kafka topic. Using Kafka as a durable message broker satisfies the requirements for loose coupling (R4) and horizontal scalability (R5).
4. **Centralized Persistence:** A dedicated Audit Microservice consumes events from Kafka and persists them into a normalized PostgreSQL 15 schema. This service can be scaled independently of the domain services to handle varying transaction volumes.

The temporal decoupling between the synchronous business transaction and the asynchronous audit flow is detailed in the Sequence Diagram (see Appendix A). This ensures the primary microservice remains unblocked, achieving the zero-latency impact requirement.

3.2 Formal Logic of the Audit Pipeline

The system enforces a strict separation of concerns. The domain microservice (the Producer) is responsible only for emitting the event and does not wait for a confirmation of persistence. This non-blocking design ensures that the primary business logic maintains zero latency impact from the auditing concern. To maintain data integrity, we implement the requiresPostCommitHandling method in the Hibernate listener, ensuring

that audit events are only generated after the primary database transaction has successfully finalized.

3.3 Implementation Details and Stack

The framework is implemented using Java 17 and Spring Boot 3.x. The selection of PostgreSQL for the centralized store was driven by its support for JSONB column types, which allows for a flexible audit schema that can evolve without requiring costly migrations as new microservices with different data structures are integrated into the ecosystem.

4. EVALUATION AND RESULTS ANALYSIS

To evaluate the effectiveness of the proposed hybrid auditing framework, a series of controlled experiments were conducted focusing on functional correctness, system latency, and transaction throughput. The test environment consisted of three Spring Boot domain services and a single-node Kafka broker running on Ubuntu 22.04 with 16GB RAM.

4.1 Functional Validation and Correctness

The system was first tested for its ability to capture 100% of managed database operations. Results indicated that the Hibernate event listeners successfully intercepted all @PostInsert and @PostUpdate events without omission. Furthermore, the enrichment logic correctly appended the authenticated User ID and localized field labels to every event, successfully addressing the "domain-blindness" identified in infrastructure-level tools.

4.2 Latency and Throughput Analysis

A primary objective of this research (RQ3) was to minimize the performance impact on the domain microservices. We measured the "Audit Lag"—the time elapsed from the database commit to the final persistence in the centralized store.

Load Level (Events/min)	Average Latency (ms)	Main Transaction Impact
Low (1,000)	140 ms	0 ms (Non-blocking)
Medium (10,000)	210 ms	0 ms (Non-blocking)
High (50,000)	330 ms	0 ms (Non-blocking)

Table 1: Performance analysis - Latency

The data shows that even under high load, the average latency remains well below the 500ms threshold. Most critically, because the Kafka producer is asynchronous, the primary business transaction experienced zero additional latency, confirming the framework's suitability for performance-sensitive environments.

4.3 Comparison with Existing Solutions

As summarized in Table 2, the custom tool strikes a unique balance between existing technologies. While Debezium offers high scalability, it lacks the user identity capture

provided by our solution. Conversely, Hibernate Envers provides context but lacks the decoupled, Kafka-based transport necessary for independent service scaling.

Feature	Custom Tool	Envers	Javers	Debezium
Captures UPDATE, INSERT	+	+	+	+
Captures DELETE	?	+	+	+
Decoupled from main DB	+	-	-	+
Captures user identity	+	-	?	-
Built-in Kafka support	+	-	-	+
Easy to customize	+	-	?	-
Field-level comparison	+	-	+	+

Table 2: Comparison Matrix

5. DISCUSSION AND CONCLUSION

5.1 Discussion of Findings

The results of this study demonstrate that the proposed hybrid auditing framework successfully addresses the "Context-Performance Trade-off" that characterizes existing distributed auditing tools. By utilizing application-layer hooks, the system captures rich semantic data—such as user roles and localized field labels—which is traditionally lost in database-level Change Data Capture (CDC) systems like Debezium. Furthermore, the transition from synchronous ORM interceptors to an asynchronous Kafka-based transport layer ensures that the domain service's transaction integrity and performance remain unaffected.

A significant advantage observed during the real-world deployment in a utility billing system was the tool's ability to aid in rapid forensic debugging. By providing a human-readable history of meter status and payment updates, the system allowed developers to identify the root cause of a status-mismatch bug within hours—a task that previously required extensive manual log parsing.

5.2 Limitations and Future Work

While the framework is production-ready for standard CRUD operations, certain limitations remain. Currently, the system does not natively support the auditing of **DELETE** operations due to the loss of entity state during the removal process in Hibernate . Future research will explore the implementation of "Soft-Delete" patterns or the use of specific database triggers to bridge this gap. Additionally, while the current use of JSON for event serialization is flexible, integrating a schema registry (e.g., Avro) would enhance long-term data compatibility and governance in larger enterprise ecosystems.

5.3 Conclusion

This dissertation has presented a scalable, context-aware auditing tool designed specifically for microservice architectures. By striking a balance between application-level awareness and infrastructure-level reliability, the framework provides a robust solution for industries requiring high levels of accountability and transparency . As microservices continue to dominate the enterprise landscape, decoupled and extensible observability tools like the one proposed here will be essential for maintaining trust and compliance in distributed environments.

REFERENCES

1. **Amir-Mohammadian, S., & Yousefi Zowj, A. (2021).** Towards Concurrent Audit Logging in Microservices. *In 2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)* (pp. 1357-1362). IEEE.
2. **Barabanov, A., et al. (2021).** Security Audit Logging in Microservice-Based Systems: Survey of Architecture Patterns. *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 605-609. IEEE.
3. **Kleppmann, M. (2017).** *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems.* O'Reilly Media.
4. **Richardson, C. (2019).** *Microservices Patterns: With Examples in Java.* Shelter Island, NY: Manning Publications.
5. **Tahir, A., & Hussain, F. (2022).** Design and Implementation of a Generic Audit Logging Framework for Java-Based Microservices. *International Journal of Computer Science and Software Engineering*, 12(3), 42–50.