Unified Event-Sourced CQRS Architecture for Continuous Compliance Monitoring and Automated Regulatory Reporting

Sai Nitesh Palamakula

Software Engineer
Microsoft Corporation
Charlotte, NC, USA
palamakulasainitesh@gmail.com

Abstract:
Compliance audits in distributed systems have historically been reactive, relying on periodic sampling and manual correlation of disparate logs. This paper presents a unified event-sourced Command Query Responsibility Segregation (CQRS) architecture that streams both command-side state transitions and query-side access events into a dedicated compliance engine for real-time evaluation. A canonical event model, standardized event-to-rule mapping, and deterministic replay semantics are specified to transform auditability into a continuous capability. The architecture integrates immutable event storage, low latency streaming computation, and declarative policy evaluation to enable automated regulatory reporting and verifiable evidence generation. Design considerations, subsystem interactions, evaluation metrics, and operational constraints are detailed, drawing on recent advances in event-driven compliance monitoring and distributed system observability.

Keywords: Event sourcing, CQRS, compliance auditing, streaming policy evaluation, regulatory reporting, immutable logs, provenance, deterministic replay, data governance.

I. INTRODUCTION

Industries such as finance, healthcare, and telecommunications operate under stringent regulatory frameworks that demand continuous assurance of compliance. Traditional audit processes depend on coarse-grained logs and asynchronous reconciliation, introducing latency and blind spots in violation detection. Event sourcing records every state change as an immutable fact, enabling precise reconstruction of system behaviour at any point in time [1][4]. CQRS separates write-path validation from read-path projections, allowing specialized models for operational commands and analytical queries [2][3].

Recent work in event-driven architectures (EDAs) has demonstrated the feasibility of real-time compliance monitoring by streaming events into policy evaluation engines [6][7][8], yet lacks a standardized model for mapping heterogeneous events to regulatory rules.

This paper addresses that gap by defining an architecture that unifies command and query events into a compliance pipeline capable of continuous monitoring and automated reporting [5][9][12][13][16][18].

II. PURPOSE AND SCOPE

A. Purpose

The paper aims to specify a technical architecture that operationalizes event-sourced CQRS for continuous compliance monitoring. By streaming enriched command and query events into a compliance engine, regulatory rules can be evaluated at the moment of occurrence, producing signed evidence artifacts and automated reports. Deterministic replay ensures that historical evaluations can be reproduced exactly, supporting audit verification and forensic analysis [1][2][4][6][9][10][12][15][16]

B. Scope

The scope is limited to architectural design and evaluation strategy. Implementation details such as specific vendor technologies are discussed only in terms of suitability for the required guarantees. The architecture targets microservice-based distributed systems in regulated domains, assuming the existence of a compliance program and regulatory control catalog. Evaluation focuses on latency, coverage, replay determinism, and scalability, without fabricating empirical results [3][7][18].

III. RELATED WORK

Event sourcing has been formalized as a persistence pattern that replaces mutable state with append-only events, strengthening auditability and enabling time-travel reconstruction [1][4]. CQRS has been documented to decouple read and write concerns, supporting independent scaling and specialized models [2][3]. Streaming computation frameworks such as Apache Kafka Streams, Flink, and MillWheel provide primitives for ordered event delivery, stateful operators, and fault-tolerant processing [5][6][7][8]. Declarative policy languages (e.g., XACML, Rego) enable attribute- and context-based rule evaluation over event streams [12][13]. Recent industry case studies highlight challenges in compliance monitoring for EDAs, including decentralized event sources, high throughput, and sensitive data handling [6]. Provenance models in databases and distributed systems offer methods for capturing and verifying lineage, critical for audit artifact integrity [17]. However, a unified CQRS-based architecture with standardized event-to-rule mapping for continuous compliance remains underdeveloped [12][13].

IV. SYSTEM ARCHITECTURE

A. System Overview

The architecture comprises four subsystems: command side, query side, event bus and store, and compliance engine.

- **Command Side** validates invariants and authorization, executes domain logic, and emits append-only events into an immutable store.
- **Query Side** derives materialized views for analytics and emits access events to capture read-path compliance contexts.
- **Event Bus** transports both write-path and read-path events to downstream consumers with ordered semantics and backpressure control.
- **Compliance Engine** ingests, normalizes, evaluates, and emits signed evidence artifacts and alerts.

It is visualized in the Fig. 1.

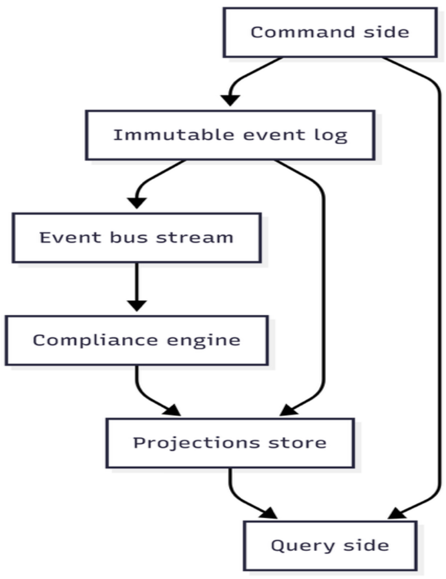

Fig. 1. High Level Architecture

B. Canonical Event Model

Events include type, schema version, aggregate ID, sequence number, causation/correlation IDs, actor identity, authorization context, event-time timestamp, clock-source metadata, and payload hash. Query events capture access attempts, projection IDs, and caller context. Schema registries enforce versioning and compatibility [2][15][18].

C. Compliance Engine Data Path

Ingestion validates signatures and schemas. Normalization maps events to a domain ontology and enriches with computed attributes. Rule evaluation applies declarative policies over attributes, temporal windows, and sequences. Evidence emission persists signed assertions, alerts, and attestations [12][13][17][18]. It is shown in Fig. 2

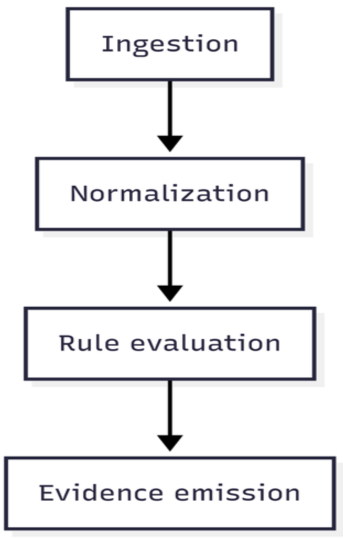

Fig. 2. Compliance Engine Flow

V. IMPLEMENTATION

The implementation of the unified event-sourced CQRS architecture for continuous compliance monitoring is organized into five integrated layers:

- Event Ingestion and Persistence
- Streaming and Delivery
- Projection and Query Handling
- Compliance Rule Evaluation
- Evidence Management and Reporting.

A. Event Ingestion and Persistence

The command side validates incoming commands against domain invariants and authorization policies before applying state changes. Once validated, commands are executed on aggregates, producing domain events that are serialized and appended to the immutable event store. The design elements include:

- **Immutable Event Store**: Implemented using a distributed log (e.g., Apache Kafka, EventStoreDB) with ordered partitions to guarantee per-aggregate sequencing [5][15].
- **Transactional Producers**: Ensure atomicity between event publication and state changes
- **Schema Registry Integration**: Enforces event schema versioning and compatibility.
- **Metadata Enrichment**: Events tagged with causation/correlation IDs, actor identity, authorization context, and event-time timestamps.

This layer ensures that every state change is captured as a verifiable, append-only record, forming the foundation for compliance evaluation.

B. Streaming and Delivery

The event bus transports both command-side domain events and query-side access events to downstream consumers, including the compliance engine and projection builders. The design elements include:

- **Low-Latency Streaming Platform**: Configured for sub-second delivery to compliance consumers.
- **Topic Partitioning Strategy**: Separates compliance-critical events from operational telemetry.
- **Exactly-Once or Effectively-Once Semantics**: Achieved through idempotent consumers and transactional offsets.
- **Security Controls**: TLS encryption in transit, topic-level ACLs, and signed payloads.

This layer ensures timely delivery of enriched events to the compliance engine without compromising ordering or integrity.

C. Projection and Query Handling

The query side builds materialized views from the event store to serve operational queries and compliance dashboards. The design elements include:

- **Projection Builders**: Stream processors transform raw events into query-optimized views.
- **Access Event Emission**: Every query request generates an access event capturing projection ID, caller identity, and purpose-of-use metadata.
- **Compliance-Aware Views**: Views expose compliance-relevant attributes alongside operational data.
- **Versioned View Definitions**: Ensures historical queries can be reconstructed exactly during replay.

This layer captures read-path behaviours that are often overlooked in compliance audits.

D. Compliance Rule Evaluation

The compliance engine consumes both command and query events, normalizes them, and applies declarative rules to detect violations or generate attestations. The design elements include:

- **Normalization Pipeline**: Maps heterogeneous events to a unified domain ontology, enriching with computed attributes.
- **Declarative Policy Language**: Rules expressed in XACML or Rego, referencing event attributes, temporal sequences, and external registries.
- **Temporal and Sequence Evaluation**: Supports sliding windows, sequence constraints, and causal chains.
- **Rule Version Pinning**: Ensures replayed events are evaluated against the same rule set used originally.
- **Alert and Attestation Generation**: Produces signed compliance assertions and violation alerts.

This layer transforms raw event streams into actionable compliance intelligence in real time.

E. Evidence Management and Reporting

The evidence ledger stores compliance assertions, alerts, and attestations in an append-only, tamper-evident repository. The design elements include:

- **Cryptographic Integrity**: Each artifact hashed and linked in a signature chain [17].
- **Immutable Ledger Storage:** Append-only design prevents alteration or deletion.
- **Automated Regulatory Reporting:** Evidence artifacts aggregated into machine-readable reports aligned with regulatory formats.
- **Auditor Access Controls:** Attribute-based access control governs retrieval.
- **Replay Verification:** Auditors can request deterministic replays to confirm historical outputs.

This layer ensures continuous audit readiness and verifiable evidence.

VI. EVALUATION STRATEGY

A comprehensive evaluation strategy addresses technical, operational, and user-outcome metrics. Table I provides an overview of the key evaluation metrics

TABLE I. EVALUATION METRICS

Metric	Description
Event-to-Alert Latency	Time from event commit to compliance alert or assertion.
Rule Coverage Ratio	Share of regulatory rules actively evaluated in real time.
Replay Determinism Rate	Percentage of events producing identical outputs under replay.
Throughput Scalability	Sustained event processing rate without correctness loss.

Each metric is continuously monitored using cloud-native observability dashboards (e.g., Azure Monitor, Amazon CloudWatch, Grafana, Prometheus), ensuring performance tracking and real-time system reporting.

VII. TECHNICAL CONSIDERATIONS

The architecture must enforce strict schema governance through a registry-backed validation process at event ingress to prevent drift and ensure downstream compatibility. Causation and correlation identifiers should be propagated consistently across workflows so that multi-step transactions, compensating actions, and sagas can be evaluated holistically. Security controls require encryption in transit and at rest, hardware-backed key management, and fine-grained authorization for rule authoring, deployment, and evidence retrieval. Privacy protections should incorporate data minimization, selective field-level encryption, and jurisdiction-aware handling to meet mandates such as GDPR and HIPAA, with de-identification policies applied for non-production replay. Operational reliability depends on idempotent consumers, transactional producers, bounded state sizes, and checkpointing aligned with streaming system guarantees, while compaction and archival strategies must be defined for long-lived event logs and evidence ledgers to balance storage efficiency with auditability [15][16][18].

VIII. CHALLENGES AND LIMITATIONS

Despite the architectural rigor, inevitable challenges and limitations affect real-world rollouts.

A. High-Throughput Event Stream Management

Large-scale systems may produce millions of events per second, requiring normalization, enrichment, and evaluation without introducing latency. Without careful partitioning and horizontal scaling, bottlenecks can compromise real-time guarantees.

B. Time Semantics and Event Ordering

Clock skew, network delays, and out-of-order arrivals can affect rules dependent on precise temporal relationships. Implementing global ordering mechanisms can mitigate ambiguity but may introduce trade-offs in latency and availability.

C. Integration with Legacy Systems

Legacy applications may lack structured event emission or CQRS compatibility, requiring adapters or batch ingestion methods that weaken real-time monitoring and increase maintenance complexity.

CONCLUSION

The unified event-sourced CQRS architecture described in this paper provides a structured approach to achieving continuous compliance monitoring and automated regulatory reporting in distributed systems. By streaming both command-side state changes and query-side access events into a dedicated compliance engine, regulatory rules can be evaluated in real time, producing signed evidence artifacts and enabling deterministic replay for audit verification. The design emphasizes immutable event storage, schema governance, low-latency streaming, and declarative policy evaluation, ensuring that compliance intelligence is embedded directly into operational workflows. Evaluation metrics for latency, coverage, determinism, and scalability offer a framework for assessing readiness without presupposing empirical results. While challenges such as complex rule encoding, high-throughput event management, and integration with legacy systems remain, the architecture establishes a foundation for transforming compliance from a reactive process into a proactive, continuous capability that preserves privacy, integrity, and verifiability in regulated environments.

REFERENCES:

[1] M. Fowler, "Event sourcing," 2005, martinfowler.com.. [Online]. Available: https://martinfowler.com/eaaDev/EventSourcing.html

[2] M. Fowler, "CQRS," 2011, martinfowler.com.. [Online]. Available: https://martinfowler.com/bliki/CQRS.html

[3] E. Evans, Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston, MA: Addison-Wesley, 2003.

[4] M. Kleppmann, Designing Data-Intensive Applications. Sebastopol, CA: O'Reilly Media, 2017.

[5] J. Kreps, N. Narkhede, and J. Rao, "Kafka: A distributed messaging system for log processing," Proc. NetDB, 2011.

[6] T. Akidau, A. Balikov, K. Bekiroğlu, A. Cherkassky, R. McKelvey, et al., "MillWheel: Fault-tolerant stream processing at Internet scale," Proc. VLDB, vol. 6, no. 11, pp. 1033–1044, 2013.

[7] T. Akidau, S. Chernyak, and R. Lax, Streaming Systems: The What, Where, When, and How of Large-Scale Data Processing. Sebastopol, CA: O'Reilly Media, 2018.

[8] M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica, "Discretized streams: Fault-tolerant streaming computation at scale," Proc. 24th ACM Symp. Operating Systems Principles (SOSP), 2013, pp. 423–438.

[9] National Institute of Standards and Technology, "Security and privacy controls for information systems and organizations," NIST SP 800-53 Rev. 5, Dec. 2020.

[10] J. C. Corbett, J. Dean, M. Epstein, A. Ghemawat, et al., "Spanner: Google's globally-distributed database," Proc. OSDI, 2012, pp. 251–264.

[11] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," Commun. ACM, vol. 21, no. 7, pp. 558–565, 1978.

[12] T. Moses, "Extensible access control markup language (XACML) version 3.0," OASIS Standard, Jan. 2013. [Online]. Available: https://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html

[13] Open Policy Agent, "Policy-based control for cloud-native environments," 2024, CNCF Project. [Online]. Available: https://www.openpolicyagent.org

[14] H. Garcia-Molina and K. Salem, "Sagas," ACM SIGMOD Rec., vol. 16, no. 3, pp. 249–259, 1987.

[15] EventStore Ltd., "EventStoreDB architecture," 2024. [Online]. Available: https://www.eventstore.com/docs

[16] European Parliament and Council, "Regulation (EU) 2016/679 (General Data Protection Regulation)," Official Journal of the European Union, 2016.

[17] J. Cheney, L. Chiticariu, and W.-C. Tan, "Provenance in databases: Why, how, and where," Found. Trends in Databases, vol. 1, no. 4, pp. 379–474, 2009.

[18] S. Chen, G. Xu, and W. Wang, "Tamper-evident logging with cryptographic hash chains," IEEE Trans. Dependable and Secure Computing, vol. 15, no. 5, pp. 847–860, 2018.