International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

Real-Time Regulatory Reporting in Banking: From 24-Hour Batch to Sub-Two-Minute Streaming with Auditable Lineage

Jeevan Krishna Paruchuri

Independent Researcher
paruchuri.g167@gmail.com

Abstract:
Regulatory reporting in retail and commercial banking has historically been built on overnight batch ETL: extract from the core banking system after end-of-day, transform overnight, submit before the morning regulatory deadline. This pattern is operationally simple and audit-friendly, and it imposes a hard floor of approximately 24 hours between when a transaction occurs and when a regulator could in principle see it. As regulatory regimes Basel IV's granularity expectations, DORA's operational resilience requirements, MiFID II's trade reporting timelines push toward shorter and more frequent reporting windows, the 24-hour floor becomes a problem. This paper presents a case study of migrating a banking regulatory reporting pipeline from a legacy overnight batch architecture to a streaming architecture with end-to-end latency under 2 minutes, while preserving and in fact strengthening the audit and lineage properties that make the legacy system defensible to regulators. The architecture is built on Attunity Replicate for mainframe CDC-based change data capture from the Oracle core banking system into per-table Kafka topics governed by a schema registry, Spark Structured Streaming with exactly-once semantics writing into Delta Lake with append-only audit semantics, and dual-region replication that drove disaster recovery time from 6 hours 20 minutes to 3 hours 45 minutes. Inter-region Kafka replication lag is under 1 minute in steady state. The pipeline handles a baseline of approximately 2 million transactions per day (~23 TPS baseline, ~230 TPS peak) with capacity for 10x spikes at month-end and during crisis scenarios. We document the regulatory requirements that drove the architecture, the technical patterns that emerged, the audit capabilities the new system provides point-in-time queries, full change history, immutable amendment records and the things that took multiple iterations to get right, most notably exactly-once semantics under Kafka offset management and schema evolution coordination across producer and consumer teams. We are honest about the operational complexity the streaming architecture introduces and about the months of work required to build compliance-team trust through hands-on demonstrations of the audit trail. The contribution is a practitioner-grounded blueprint for teams considering whether the cost of streaming regulatory reporting is justified by the regulatory and operational benefits, with explicit attention to the audit and lineage properties that compliance teams will demand.

Keywords: Regulatory reporting, Basel IV, DORA, MiFID II, Change data capture, Kafka, Spark Structured Streaming, Delta Lake, Audit trail, Disaster recovery

1. INTRODUCTION

A regulatory submission is, in the eyes of a bank examiner, a contract between the bank and the regulator. The contract says: at this date and time, here is a complete and accurate picture of the transactions, positions, exposures, and customers we are required to report. If the picture is wrong, the bank is on the hook for the error and for the explanation of how the error occurred. If the picture cannot be reconstructed after the fact, the bank is on the hook for the lineage failure. The job of the data

platform that produces these submissions is to make sure both the picture and the reconstruction story are defensible.

The traditional way to build this kind of pipeline is overnight batch ETL. The core banking system is queried after end-of-day, the relevant transactions are extracted, the data is transformed and aggregated through a series of batch jobs, the regulatory feed is generated, and the submission is sent before the morning deadline. The pattern is operationally simple, the audit trail is the batch run log, and any error in the submission can in principle be corrected by re-running the batch with the corrected input. The platform under discussion in this paper used exactly this pattern for many years, with a regulatory submission deadline of 7am that drove the entire overnight schedule.

The problem with the batch pattern is the latency floor it imposes. A transaction that occurs at 9am cannot affect any regulatory view until the following morning at the earliest, because the batch will not run until that night. For most regulatory regimes, a 24-hour delay was historically acceptable. For the regimes the bank operates under in 2025 Basel IV's granularity expectations, DORA (the Digital Operational Resilience Act), MiFID II trade reporting timelines, and the broader trajectory of regulatory expectations toward shorter reporting cycles the 24-hour floor has become an obstacle. Regulators want to see incident-level detail closer to real time. Internal compliance teams want to detect and remediate reporting errors during the day rather than the next morning. Risk teams want exposures updated continuously rather than at end-of-day snapshots.

This paper describes how the platform team migrated from the overnight batch architecture to a streaming architecture with end-to-end latency under 2 minutes, while preserving the audit and lineage properties the compliance team requires. The migration was incremental, took several months, and produced a pipeline that has now been operating for more than a year. We pursue three questions. RQ1. What architectural patterns enable real-time regulatory reporting with audit and lineage properties as strong as the legacy batch system? RQ2. What does the migration cost in operational complexity, and what engineering practices keep that complexity bounded? RQ3. How do you build compliance team trust in a streaming pipeline when their mental model of correctness is built on batch?

The contribution is a concrete blueprint anchored in regulatory reality, an honest discussion of the things that took multiple iterations to get right, and a description of the audit capabilities the new system provides that the old system could not.

2. THE REGULATORY DRIVERS

Three regulatory regimes shaped the architecture and deserve explicit description.

Basel IV (more precisely, the Basel III framework finalized in 2017 and implementing reforms colloquially referred to as Basel IV) imposes capital and risk management requirements that depend on accurate and timely data about exposures, counterparties, and market conditions. The data quality and granularity expectations have tightened over successive iterations, and the regulatory expectation is that risk views are produced from the same data the front office uses, with auditable lineage between the source transactions and the reported numbers. A 24-hour gap between transaction and risk view is increasingly hard to defend.

DORA (the Digital Operational Resilience Act, EU Regulation 2022/2554) imposes operational resilience requirements on financial institutions in the European Union, including ICT risk management, incident reporting, resilience testing, and third-party risk management. For a regulatory reporting pipeline, the most consequential DORA requirements are around incident detection and reporting

timelines: significant incidents must be reported to the competent authority within tight windows, and the bank must be able to demonstrate that its detection capability is consistent with the reporting expectations. A pipeline that learns about issues 24 hours after they occur cannot meet these timelines.

MiFID II trade reporting requires that transactions in financial instruments be reported to the relevant trade repository within prescribed timeframes for many transaction types, T+1 at the latest, and for some reports closer to real time. The pipeline under discussion handles a subset of transactions that fall under this reporting obligation, and the legacy batch pattern was at risk of breaching the timeliness requirement on edge cases.

The cumulative effect of these regimes is that the batch architecture was approaching the limits of what could be defended to examiners, and a transition was inevitable. The choice was when to do it and how to manage the risk of the transition.

3. THE STREAMING ARCHITECTURE

The new architecture has five components: a CDC source layer at the Oracle core banking system, a Kafka transport layer, a Spark Structured Streaming processing layer, a Delta Lake storage layer with append-only audit semantics, and a regulatory feed generation layer that produces the immutable submissions.

3.1 CDC Source Layer

Change data capture (CDC) is the technique of capturing every change to a source database inserts, updates, deletes as a stream of events rather than as periodic full extracts. We use Attunity Replicate for mainframe CDC, an open-source CDC framework, with its Oracle connector. Attunity Replicate for mainframe CDC reads the Oracle redo log and emits a Kafka event for every committed change, with a schema that includes the before image, the after image, and the operation type.

The CDC layer is configured to emit one Kafka topic per source table. This per-table topic structure makes downstream consumption simple a consumer interested in transactions subscribes to the transactions topic and it isolates schema evolution to specific topics rather than mingling unrelated schemas. A central schema registry governs the schemas of all topics, enforces compatibility rules (backward compatibility for additive changes), and provides a single source of truth for what each topic contains.

The migration from the legacy daily extract to CDC was the single largest improvement in end-to-end latency, dropping the floor from 24 hours to under 2 minutes essentially overnight once the CDC pipeline went live.

3.2 Kafka Transport

The Kafka cluster sits between the CDC layer and the processing layer. It provides durable buffering, replay capability (essential for reprocessing during incidents), and the loose coupling that lets the producer and consumer evolve independently. The cluster is operated by the platform team and is configured for the throughput profile of the workload: a baseline of approximately 2 million transactions per day translating to roughly 25 events per second on average, with the design capacity for 10x spikes during month-end processing and crisis scenarios.

For disaster recovery, a second Kafka cluster runs in the secondary data center (DC2), and Kafka mirroring keeps it within less than 1 minute of replication lag behind the primary (DC1) in steady state. The mirrored topics are the basis for cross-region failover, described in Section 5.

3.3 Spark Structured Streaming Processing

The processing layer is Spark Structured Streaming in micro-batch mode, written in Scala. Each micro-batch reads new events from the Kafka topics, performs the necessary transformations and enrichments, and writes the result to Delta Lake. Exactly-once semantics are guaranteed end-to-end through the combination of Kafka offset tracking, Spark's checkpoint-based recovery, and Delta Lake's idempotent commit protocol.

Getting exactly-once right took multiple iterations. The naive configuration produced duplicates under failure, because the offset commit and the Delta write were not coordinated through the same transaction. The working configuration stores the Kafka offsets in the same Delta transaction that writes the data, so that recovery from a checkpoint guarantees that no event is processed twice and no event is lost. This pattern is well-documented in the Spark and Delta Lake literature, but it took us several rounds of testing under deliberate failure injection to verify that we had it implemented correctly. The honest acknowledgment is that exactly-once is harder than it looks.

3.4 Delta Lake with Append-Only Semantics

The storage layer is Delta Lake in an append-only mode. The transactional table model the regulatory snapshot of every transaction does not allow in-place updates. Corrections to a transaction take the form of an amendment record: a new row that references the original transaction, records the correction, and is itself immutable. This pattern preserves the full history of every change, which is exactly what regulators expect from an audit trail. Time-travel queries against Delta Lake can answer the question "what did the system know about this transaction at this point in time," which is the core capability the audit trail provides.

The append-only model required some retraining for the team. Engineers used to UPSERT semantics initially tried to overwrite incorrect records, and the pattern had to be enforced through both code review and an automated check that rejected any operation that would modify or delete an existing row in the regulatory tables.

3.5 Regulatory Feed Generation

A separate downstream job generates the regulatory feed itself the file or stream of records that is sent to the regulator. The feed is generated every minute, and once generated it is frozen: the feed file is treated as immutable and is preserved alongside the data it was generated from. If a correction is needed after a feed has been generated, the correction takes the form of a follow-up amendment feed rather than a regeneration of the original. This pattern matches the way regulators think about submissions: a submission is an artifact at a moment in time, and the historical record of what was submitted when is part of the audit trail.

4. AUDIT CAPABILITIES

This section describes the audit capabilities the new architecture provides, with concrete scenarios.

Point-in-time queries. A regulator or auditor can ask "what did the system know about transaction X at 14:32 on March 15," and the answer is reconstructible from Delta Lake time travel. The query takes seconds to execute regardless of how far in the past the point in time is, within the 7-year retention window. This is materially better than the legacy batch system, where the equivalent question required restoring a backup of the previous day's batch state.

Full change history. Every modification to a transaction is preserved as an amendment record, and the chain of amendments for any transaction is queryable. The legacy system had only the most recent state,

plus a paper trail of correction tickets in a separate ticketing system. The new system collapses these into a single queryable history.

Amendment traceability. Each amendment record carries the identity of the person or system that initiated the correction, the timestamp, and a free-text reason field. Compliance reviews of amendments a routine activity are now possible directly against the Delta tables rather than against a separate spreadsheet of corrections.

Sub-five-second compliance queries at scale. A representative scenario: a regulator asks for all transactions affecting customer X between January and February, with full audit trail. In the legacy system, this query required pulling backups, running a custom report, and assembling the correction history manually typically a half-day exercise. In the new system, the query runs against Delta Lake directly and returns a complete result 47 transactions in one specific case we measured in under 5 seconds. The compliance team's workflow for these queries collapsed from hours to minutes.

Spark job recovery scenario. A representative incident: a Spark Structured Streaming job crashed during a deployment due to a transient resource issue. Detection was under 1 minute via the lag-based alerting on the Kafka consumer offsets. The job auto-recovered from its checkpoint within several minutes. The post-incident investigation traced the affected transactions through the Delta history and confirmed that no records were lost or duplicated the exactly-once guarantees held under failure. The incident was logged but did not produce a regulatory issue because the recovery completed well before any submission window.

The audit capabilities are the answer to the legitimate compliance question of why the streaming architecture is more defensible than the batch architecture rather than less. The honest answer is that streaming with append-only Delta Lake provides a stronger audit trail than batch with daily snapshots, because every change is captured and preserved rather than only the end-of-day state.

5. DISASTER RECOVERY: 6H20M → 3H45M

The legacy batch architecture had a documented disaster recovery procedure that took approximately 6 hours and 20 minutes to execute end-to-end during drills. The new streaming architecture, after a period of optimization, executes the equivalent failover in 3 hours and 45 minutes an improvement attributable to a combination of technical and procedural changes.

The technical changes are: dual Kafka clusters with cross-region mirroring keeping replication lag under 1 minute, dual Delta Lake storage in primary and secondary regions, Spark Structured Streaming jobs that can resume from checkpoint at the secondary site without manual reconfiguration, and Terraform-managed infrastructure that allows the secondary site to be brought to full capacity from a known-good baseline. Each of these changes shaved time off specific steps in the failover procedure.

The procedural changes are equally consequential: rewritten runbooks that specify each step with the exact command and the expected output, monthly DR drills that have built up team muscle memory and surfaced procedural snags before they could affect a real incident, and automation of the steps that were previously manual.

The remaining 3h45m is dominated by human decision latency rather than by technical operations. Further reductions will require either more automation of decision points or more aggressive parallelization of human and machine work, both of which have their own risks. We discuss DR in more detail in our companion work on observability-driven SRE; the relevant point here is that the streaming

architecture is at least as recoverable as the batch architecture and in fact materially more recoverable because the data is continuously replicated rather than restored from periodic backups.

6. WHAT TOOK MULTIPLE ITERATIONS

Three things in the streaming architecture took multiple iterations to get right and deserve specific honest discussion.

Exactly-once semantics under Kafka offset management. The naive Spark Structured Streaming configuration produces at-least-once guarantees by default, which means duplicates can occur under failure. Achieving exactly-once required coordinating Kafka offsets with Delta writes through a single transactional commit, validating the configuration against deliberate failure injection (kill the Spark driver mid-batch, kill an executor, restart the Kafka broker), and finding the corner cases where the documentation was ambiguous. We went through several rounds before we were confident in the guarantee, and we still re-validate it after major Spark or Delta version upgrades.

Schema evolution coordination across producer and consumer teams. Attunity Replicate for mainframe CDC captures schema changes from the source database and propagates them through Kafka via the schema registry, but the downstream consumers Spark jobs, the regulatory feed generator, the audit query layer have their own expectations about what fields exist. A schema change in the source that is backward-compatible for some consumers may not be for others. We learned this the hard way when an upstream column rename broke a downstream feed for a few hours before the discrepancy was caught. The fix was a tighter coordination process between the team that owns the source database and the platform team, with schema changes announced in advance and tested in a staging environment before reaching production. The technical mechanism was already in place; what was missing was the organizational discipline.

Building compliance team trust. The compliance team's mental model of correctness was built on batch processing, where the daily snapshot is "the answer" and any difference between consecutive snapshots represents the day's activity. Streaming breaks this mental model. When we first showed the compliance team the new architecture, the response was skepticism not because they thought streaming was wrong but because they did not yet have a way to convince themselves it was right. Building the trust took months and several specific interventions: hands-on walkthroughs of the audit trail with real (anonymized) transaction examples, side-by-side comparisons of regulatory feeds generated from the legacy and new systems, and a formal certification exercise in which an independent reviewer attested that the new architecture preserved the audit properties of the old one. The technical work was a small fraction of this effort; the social work was the bulk of it.

7. COST AND OPERATIONAL COMPLEXITY

The streaming architecture is more operationally complex than the batch architecture it replaces. The batch system was a small number of scheduled jobs with well-understood failure modes; the streaming system is a continuous pipeline with state, checkpoints, replication, and a larger surface area for things to go wrong. The honest cost accounting:

Infrastructure cost. The streaming infrastructure (Kafka cluster, Spark cluster running continuously, Delta Lake storage with full history retention, secondary region replication) costs more than the equivalent batch infrastructure. The increase is on the order of 20-30% above the batch baseline for the regulatory reporting workload specifically, though this is partially offset by the reuse of infrastructure that was already in place for other workloads.

Operational effort. The platform team spends measurable on-call attention on the streaming pipeline that the batch pipeline did not require Kafka consumer lag, schema evolution coordination, checkpoint health monitoring. The effort is bounded by the SRE practices described in our companion work, but it is real and it does not go to zero.

Compliance team learning curve. As discussed in Section 6, building compliance team trust was a months-long effort. The cost is one-time but it is not negligible, and any team contemplating this migration should budget for it.

The benefits sub-2-minute latency, stronger audit trail, faster compliance queries, better disaster recovery justify the costs in our environment. They may not justify them in environments where the regulatory pressure for low-latency reporting is weaker, where the batch system is already meeting all obligations comfortably, or where the team does not have the operational maturity to run a streaming pipeline reliably. The honest framing is that this is a worthwhile migration for organizations that need it, not a default upgrade for everyone.

8. CONCLUSION

This paper has described the migration of a banking regulatory reporting pipeline from a 24-hour overnight batch architecture to a streaming architecture with sub-2-minute end-to-end latency, built on Attunity Replicate for mainframe CDC CDC, Kafka, Spark Structured Streaming, and Delta Lake with append-only audit semantics. The pipeline handles 2 million transactions per day with capacity for 10x spikes, runs with exactly-once guarantees, replicates across regions with under 1 minute of Kafka lag, and supports disaster recovery in 3 hours 45 minutes versus the legacy 6 hours 20 minutes. The audit capabilities the new system provides point-in-time queries, full change history, sub-5-second compliance queries are stronger than what the legacy batch system offered, despite the streaming pattern being initially less familiar to the compliance team.

The principal lessons are four. Streaming regulatory reporting is feasible and audit-defensible when the architecture is designed with audit as a first-class requirement rather than as an afterthought. Exactly-once semantics are harder than they look and deserve dedicated testing and re-validation after upgrades. Schema evolution is an organizational problem as much as a technical one and needs coordination between source and consumer teams. Building compliance team trust takes longer than building the system itself and should be budgeted accordingly.

Future work should focus on the broader question of how regulatory expectations will evolve as more banks adopt streaming reporting. The current state is that streaming is the exception and batch is the default; as the exception becomes more common, regulators are likely to adjust their expectations, which may in turn drive further architectural changes. The pattern described here is what we built for the regulatory environment of today; it may need to evolve as the environment evolves. The contribution of this paper is to make one team's design and lessons available to peers facing the same decision, with the honest acknowledgment that the right answer for any specific organization depends on its specific regulatory exposure, its operational maturity, and its tolerance for the complexity that streaming introduces.

REFERENCES:

1. Regulation (EU) 2022/2554 (Digital Operational Resilience Act, DORA). Official Journal of the European Union.
2. Basel Committee on Banking Supervision, "Basel III: Finalising post-crisis reforms," Bank for International Settlements, 2017.

3. Directive 2014/65/EU (Markets in Financial Instruments Directive II, MiFID II). Official Journal of the European Union.
4. Attunity Replicate for mainframe CDC Documentation. https://debezium.io/documentation/
5. J. Kreps, N. Narkhede, and J. Rao, "Kafka: A Distributed Messaging System for Log Processing," in Proc. NetDB, 2011.
6. Apache Kafka Documentation. https://kafka.apache.org/documentation/
7. M. Armbrust et al., "Structured Streaming: A Declarative API for Real-Time Applications in Apache Spark," in Proc. SIGMOD, 2018.
8. M. Armbrust et al., "Delta Lake: ACID table capabilities for cloud object stores," Proc. VLDB Endowment, 2020.
9. M. Zaharia et al., "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing," in Proc. NSDI, 2012.
10. T. Akidau et al., "The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing," Proc. VLDB Endowment, 2015.
11. N. Marz and J. Warren, Big Data: Principles and Best Practices of Scalable Realtime Data Systems. Manning, 2015.
12. Apache Spark Documentation. https://spark.apache.org/docs/latest/
13. HashiCorp Terraform Documentation. https://www.terraform.io/docs
14. Sarbanes-Oxley Act of 2002, Public Law 107-204, 116 Stat. 745.
15. Regulation (EU) 2016/679 (General Data Protection Regulation, GDPR).
16. B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, Eds., Site Reliability Engineering: How Google Runs Production Systems. O'Reilly, 2016.
17. Board of Governors of the Federal Reserve System and OCC, "Supervisory Guidance on Model Risk Management," SR Letter 11-7, 2011.
18. D. Sculley et al., "Hidden Technical Debt in Machine Learning Systems," in Proc. NeurIPS, 2015.
19. Apache Hive LLAP Documentation. https://kyuubi.apache.org/docs/latest/
20. J. Kreps, "Questioning the Lambda Architecture," O'Reilly Radar, 2014.
21. Databricks (2023). Unity Catalog: Unified Governance for Data and AI. Technical Report.
22. Apache Software Foundation (2024). Apache Iceberg Table Format Specification v2. Technical Documentation.
23. Shankar, S., et al. (2024). Operationalizing Machine Learning: Challenges and Best Practices. IEEE Software, 41(2), pp. 42-51.