🛡️ Disaster Recovery Architecture Using Kafka/SQS/Kinesis (DR Ledger Only)

📌 Executive Summary

Modern cloud-native applications operating on AWS need strong disaster recovery (DR) strategies, especially when using globally replicated databases like **Aurora Global**.
While Aurora provides cross-region replication, it **does not eliminate data loss during regional failover** due to replication lag (typically 2–5 seconds).

To mitigate this gap, we introduce a **side-channel DR recovery ledger** using **Kafka**, driven by a **Debezium-based outbox pattern**, designed solely to detect and optionally recover lost transactions after an unplanned regional disaster.

This document evaluates the approach in detail, compares queue-based solutions, and recommends a scalable, low-intrusion strategy for **resilient but decoupled disaster recovery**.

🏗️ High-Level Architecture Overview

This architecture assumes:

- **Aurora Global** is the system of record (SOR)
- A **dedicated outbox table** captures every insert
- **Debezium** streams the outbox to **Kafka** asynchronously
- Kafka is **replicated across regions**
- Kafka is **used only for recovery, not normal processing**

The flow ensures that **every committed transaction is also recorded in Kafka** — acting as a **DR ledger**.

Component	Role
Aurora Global	Primary DB with cross-region replication
Outbox Table	Stores a durable record per transaction

Debezium	Monitors DB log and sends to Kafka
Kafka (DR Ledger)	Holds copies of committed inserts
Reconciliation Job	Finds mismatches between Kafka and DB
Replay Tool	(Optional) Replays lost records from Kafka

🎯 Purpose of Queue in This Design

Queues like **Kafka, SQS, or Kinesis** are not used for business logic but as **a secondary backup path** in case of failure. The goal is **not to prevent all failures** but to ensure **visibility and traceability** of what might be lost and provide tools to restore it.

Goal	Achieved?	Mechanism
Prevent data loss from Aurora lag	✅ Partially	Kafka holds copy of committed data
Maintain clean app architecture	✅ Yes	All logic stays in DB + Debezium
Enable after-the-fact recovery	✅ Yes	Replay from Kafka
Avoid use of queue for normal logic	✅ Yes	Kafka is read-only during DR
Work seamlessly in normal flow	✅ Yes	Asynchronous, non-blocking path
Detect and alert on loss	✅ Yes	Reconciliation job uses Kafka ledger

📊 Aurora Global Only vs. Aurora + Outbox + Kafka for DR

This comparison shows how relying solely on Aurora Global compares with using Kafka + outbox to reduce data loss during failover.

Aspect	Aurora Global Only	Aurora + Kafka Ledger
Replication Lag	2–5 seconds	Same, but dual path
Data Loss on Region Down	❌ Entire lag window lost	✅ Partial (recoverable)
Recovery Mechanism	❌ None	✅ Replay from Kafka

Detection Capability	❌ No insight	✅ Kafka-based audit
Infrastructure Complexity	✅ Simple	⚠️ Medium (Kafka + Debezium)
App Impact	✅ None	✅ None (via outbox)
Operational Cost	✅ Low	⚠️ Higher but targeted
DR Readiness	⚠️ Basic	✅ Strong & Auditable

🔍 DR Scenarios: What Happens on Failure?

To understand this solution's value, consider several failure scenarios:

Scenario	DB Lost	Kafka Lost	Kafka Helps?	Recovery Possible?
DB write committed, Kafka OK	✅ Yes	❌	✅ Recoverable	✅ Replay from Kafka
Kafka lagged, DB committed	❌	✅	⚠️ Might miss few	⚠️ Partial
DB write not committed	❌	❌	❌ Not logged	❌ No recovery
DB + Kafka down (region failure)	✅	✅	⚠️ If Kafka replication faster	⚠️ Sometimes
Kafka up, Outbox not written	❌	✅	❌ Nothing to replay	❌ Data lost
Full healthy flow	❌	❌	❌ Not needed	✅ No issue

🧪 Debezium-Based Kafka vs Direct Kafka Write from App

There are two ways to populate Kafka for DR:

1. **Debezium-based approach**, where the app only writes to DB.
2. **Direct Kafka writes** from the application in parallel to DB writes.

Dimension	Debezium-Based Kafka	Direct Kafka from App
App Simplicity	✅ No dual writes	❌ Must write Kafka + DB
Transactional Consistency	✅ Single atomic DB commit	❌ Risk of inconsistency

DR Recovery Validity	✅ Only committed events	⚠️ May include garbage
Reconciliation Ease	✅ Clean	⚠️ Conflicting states possible
Replay Safety	✅ High – matches DB	⚠️ Unreliable
Infra Overhead	⚠️ Debezium infra	✅ No Debezium
DR-only Fit	✅ Perfect	⚠️ Overkill unless used for other purposes
Monitoring Complexity	✅ Centralized	⚠️ Scattered per app

✅ **Debezium + Outbox** is the recommended option for **DR-only Kafka usage**, ensuring clean architecture and safe replay.

🧠 Reconciliation and Replay

After a DR event:

1. **Kafka is queried** for a time window of recent inserts.
2. **Aurora outbox is queried** in DR region after failover.
3. **Missing entries are identified**.
4. **Idempotent replay** logic can reinsert lost transactions into the DB.

This ensures **data loss is traceable and recoverable**, without interfering with the primary app logic.

📈 Quantifying Benefits

Let's compare a real scenario:
Application does 100 TPS inserts

Metric	Aurora Global Only	Aurora + Kafka + Debezium
Cross-region replication lag	2–5 seconds	Same
Max insert loss during region failover	**200–500 records**	**0–50 records**, recoverable
Data replay possible?	❌ No	✅ Yes

Loss detection possible?	❌ No	✅ Yes
Replay confidence	❌ Not applicable	✅ Strong (only committed)

✅ Conclusion

In DR-critical environments, relying solely on Aurora Global can lead to **silent, unrecoverable data loss** due to replication lag.
By adding a **dedicated disaster recovery ledger (Kafka)**, populated via the **Debezium outbox pattern**, teams can detect and recover lost inserts with **minimal impact on application design**.

✔️ Final Takeaways:

- ✅ **Aurora is the SOR**, Kafka is a **DR journal**
- ✅ Kafka is only used **after failure**, not during normal ops
- ✅ **Debezium-based** streaming ensures **only committed data** is logged
- ✅ Replay is **safe, consistent, and observable**
- ✅ This setup provides **auditable traceability and resilience** at modest cost