Incident Postmortem Report

SEV-1

US-EAST-1 RDS Aurora Cluster Failover & API Gateway Cascade Failure

2024-11-14 03:17 UTC Duration: **2h 43m** Affected: **~47 000 users** Est. Cost: **$183 000**

Incident ID	INC-2024-0847	**Severity**	SEV-1 Critical
Service	Payment Processing API	**Status**	RESOLVED
Incident Cmdr.	Elena Vasquez	**Document Rev.**	v2.1 (2024-11-21)
Scribe	Marcus Okonkwo	**Reviewed By**	Priya Nair, James Trevino
Detection	2024-11-14 03:17 UTC	**Resolution**	2024-11-14 06:00 UTC

1 Executive Summary

On **2024-11-14** at 03:17 UTC, a misconfigured automated maintenance window triggered an unplanned failover of the primary Vertex Aurora PostgreSQL cluster (`payments-db-us-east-1`) serving the Payment Processing API. The failover propagated a cascade failure into the API Gateway layer due to hardcoded connection pool exhaustion logic. Approximately **47 000 end users** were unable to complete checkout flows for **2 hours 43 minutes**. Total estimated revenue impact is **$183 000** with an additional $24 000 in cloud cost overrun from emergency scaling actions. This document captures the full timeline, root cause analysis, blast radius, and the set of action items committed to by the team to prevent recurrence.

Key Finding

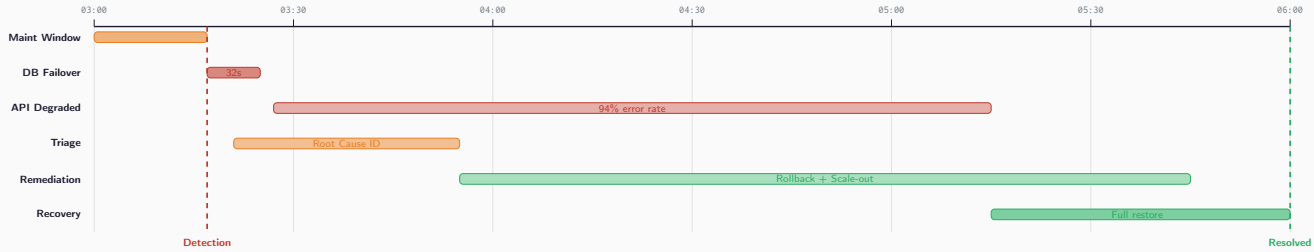
The failure was *not* caused by the database failover itself (which completed successfully in 32 s) but by the API service's connection pool refusing to re-establish connections after the new primary endpoint's DNS propagated, due to a missing `read_after_write` retry wrapper introduced in `v4.7.2`.

2 Incident Timeline

2.1. Chronological Events

Timestamp (UTC)	Actor	Event
03:00	**Automation**	Scheduled maintenance window opens (misconfigured: Prod instead of Staging)
03:12	**Aurora**	Parameter group push triggers instance reboot sequence on primary
03:17	**PagerDuty**	**ALERT**: P99 latency on `/api/v2/payments` exceeds 8 s
03:19	**Aurora**	Primary failover initiates; replica `payments-db-r2` promoted
03:21	**On-call (Elena)**	Acknowledged alert; begins triage on API Gateway metrics
03:25	**Aurora**	New primary endpoint DNS propagated; failover complete (32 s)
03:27	**API Layer**	Connection pool exhausted; `HTTP 503` rate reaches 94%
03:31	**Elena**	War room bridge opened; Marcus, Priya, James join
03:40	**James**	Identifies connection pool config bug in `app-config-v4.7.2`
03:55	**Priya**	Rolls back connection pool config via feature flag
04:10	**SRE Team**	Error rate drops to 28%; partial recovery; caches warming
04:40	**SRE Team**	Emergency pod scaling applied (32 → 96 replicas)
05:15	**Payment API**	Error rate below 1%; user-facing impact effectively resolved
05:45	**Marcus**	Full traffic restored; rollback tagged `hotfix/pool-v4.7.3`
06:00	**Elena**	Incident declared resolved; all-clear notification sent

2.2. Gantt-Style Timeline Visualization



3 Impact Assessment

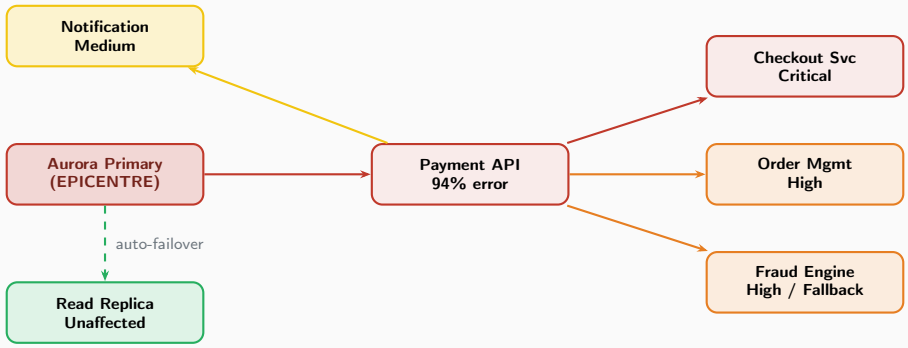
3.1. Blast Radius

Component	Severity	Details
Payment Processing API	Critical	94% of requests returning HTTP 503; $183k revenue loss
Checkout Service	Critical	Unable to initiate or complete transactions; cart abandonment spike
Order Management Svc	High	Write operations queued; 12-minute backlog at peak
Fraud Detection Engine	High	Fallback mode activated; 340 transactions processed without scoring
Notification Service	Medium	Email/SMS delays up to 18 min; no data loss
Analytics Pipeline	Medium	47-minute data gap in Redshift; backfill completed post-incident
Read Replica Queries	None	Unaffected; read traffic shifted automatically
Static Assets / CDN	None	Fully available throughout the incident

3.2. Quantified Impact

Users Affected	≈47 000 unique sessions unable to complete payment
Duration	2 hours 43 minutes (03:17 – 06:00 UTC)
Revenue Impact	$183 000 estimated (based on avg. transaction value $38.70)
Cloud Cost Overrun	$24 000 (emergency Lambda concurrency + RDS multi-AZ resync)
SLA Breach	Yes — 99.95% monthly SLA violated (actual: 99.62%)
Customer Tickets	1 247 tickets filed; 94 escalated to Tier-2

3.3. Service Dependency Blast-Radius Diagram



4 Root Cause Analysis

4.1. Five Whys Analysis

Symptom: 47 000 users unable to complete payment for 2h 43m

↓

Why 1: Payment API returned HTTP 503 at 94% rate

↓

Why 2: API connection pool fully exhausted after DB failover

↓

Why 3: Connection pool lacked retry/reconnect on new primary DNS

↓

Why 4: `read_after_write` retry wrapper removed in `v4.7.2` refactor

↓

Why 5 (Root): No integration test covering Aurora failover reconnection path; regression not caught pre-deploy

4.2. Contributing Factors

1. **Misconfigured Maintenance Window**: The automation runbook targeted `env=production` instead of `env=staging` due to a copy-paste error in the Terraform variable file committed 3 days prior.
2. **Missing Canary Test**: Deployment pipeline for `v4.7.2` had canary gates disabled for "hotfix track" releases, bypassing the integration suite that would have caught the missing retry wrapper.
3. **Insufficient Observability**: No dedicated metric for "connection pool exhaustion" existed; the signal

appeared only as generic latency, delaying diagnosis by ≈14 minutes.

4. **On-call Runbook Gap**: The DB failover runbook did not include a step to verify API-layer reconnection behaviour post-failover, leaving the on-call engineer without a clear path to diagnosis.

4.3. What Went Well

1. PagerDuty alert fired within **8 minutes** of first errors, meeting the <10-minute SLO for P99 threshold breach.
2. War room assembled and effective within 10 minutes of acknowledgement.
3. Aurora failover itself completed in **32 seconds** — well within the 60-second RTO target for the database tier.
4. Feature-flag rollback of the pool config was executed in under 3 minutes once the root cause was identified.
5. Read replica traffic remained fully available; browse and search features were unaffected throughout the incident.

4.4. Code Evidence — Missing Retry Wrapper

The following diff shows the `v4.7.2` change that inadvertently removed the Aurora reconnection wrapper:

Listing 1: `src/db/pool.py` — diff v4.7.1 to v4.7.2

```
- async def get_connection(self) -> Connection:
-     for attempt in range(self.max_retries):
-         try:
-             conn = await self._pool.acquire(timeout=2.0)
-             await conn.execute("SELECT 1")  # probe after failover
-             return conn
-         except (PoolExhaustedError, ConnectionRefusedError):
-             await asyncio.sleep(0.5 * (2 ** attempt))
-     raise DatabaseUnavailableError("Pool exhausted after retries")
+
+ async def get_connection(self) -> Connection:
+     return await self._pool.acquire()  # no retry, no probe
```

The Aurora cluster endpoint (`cluster.proxy-xxx.rds.amazonaws.com`) updated DNS within 32 seconds of failover, but in-flight pool connections cached the old primary IP. Without the probe, callers received stale connections that immediately failed, flooding the pool.

Listing 2: CloudWatch — connection pool metric (03:17–03:55 UTC)

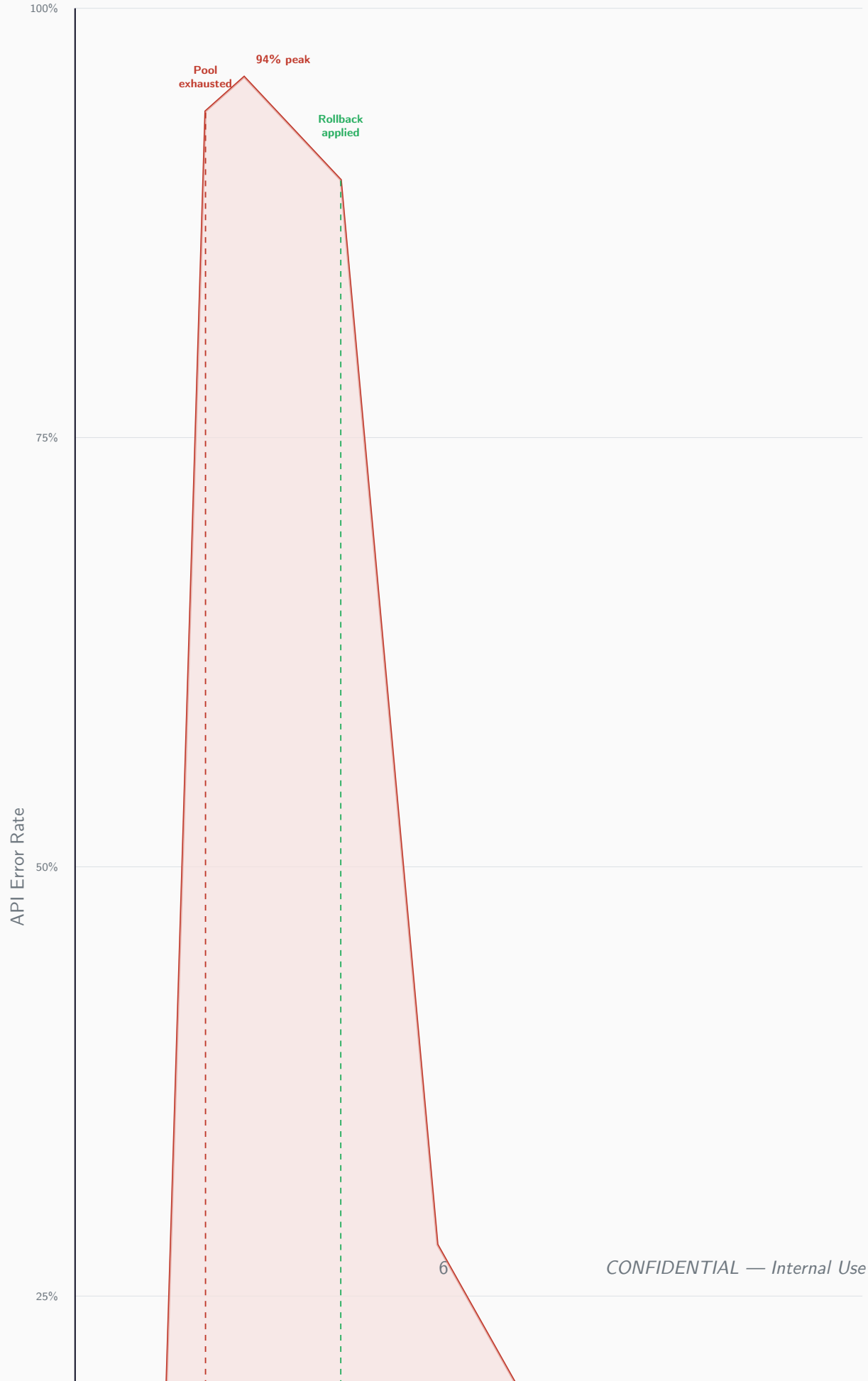
```
03:17:08  payments-api  pool.active=250  pool.idle=0  pool.waiting=847
03:19:22  payments-api  pool.active=250  pool.idle=0  pool.waiting=2341
03:25:44  payments-api  pool.active=250  pool.idle=0  pool.waiting=6109
03:31:00  payments-api  pool.active=250  pool.idle=0  pool.waiting=8820
03:55:12  payments-api  pool.active=182  pool.idle=68 pool.waiting=0  [ROLLBACK]
```

5 Detection and Resolution Metrics

5.1. Key Time Metrics

Metric	Value	Target / SLO
Time to Detect (TTD)	8 min	< 10 min ✅
Time to Acknowledge (TTA)	4 min	< 5 min ✅
Time to Mitigate (TTM)	38 min	< 30 min ❌
Time to Resolve (TTR)	2h 43m	< 1h ❌
Time to Root Cause	23 min	< 20 min ⚠

5.2. Error Rate Recovery Visualization



6 Action Items & Remediation Tracker

Tracking

All action items are tracked in Jira project RELY. Status reviewed weekly in the SRE reliability sync. DRI = Directly Responsible Individual.

ID	Action Item	DRI	Deadline	Status
RELY-441	Restore `read_after_write` retry wrapper; add Aurora failover integration test to CI pipeline	**James T.**	2024-11-21	DONE
RELY-442	Add connection pool exhaustion CloudWatch alarm (threshold: idle < 10) with PagerDuty routing	**Marcus O.**	2024-11-21	DONE
RELY-443	Fix Terraform variable isolation: separate env maps for prod/staging maintenance windows	**Priya N.**	2024-11-28	IN PROG
RELY-444	Re-enable canary integration gates for hotfix track releases; update SDLC policy doc	**Elena V.**	2024-11-28	IN PROG
RELY-445	Update DB failover runbook: add step to validate API connection pool health post-failover	**Marcus O.**	2024-12-05	IN PROG
RELY-446	Implement RDS Proxy for Payment API to absorb future failovers transparently	**Infra Team**	2024-12-19	OPEN
RELY-447	Conduct chaos engineering exercise: simulate Aurora failover in staging with full load	**SRE Team**	2025-01-10	OPEN
RELY-448	FinOps review: auto-scaling cost guardrails to cap emergency Lambda concurrency spend	**FinOps / Priya**	2025-01-10	OPEN

7 Preventive Measures & Long-Term Improvements

7.1. Immediate (Complete ≤ 30 days)

- ☑ **Aurora failover integration test** added to `pytest-asyncio` suite; CI pipeline blocks merge on failure (RELY-441 — *complete*).
- ☑ **CloudWatch composite alarm** created: `PoolExhaustion` + `HighLatency` triggers a dedicated PagerDuty policy with the correct runbook URL attached (RELY-442 — *complete*).
- ⏳ **Terraform environment isolation**: introduce per-environment Terraform workspaces and a `prevent_destroy` lifecycle rule on production RDS maintenance resources (RELY-443).
- ⏳ **Hotfix track canary re-enabled**: gating removed from hotfix track was a deliberate speed tradeoff; policy updated to require at least a smoke-test pass and explicit SRE sign-off (RELY-444).

7.2. Medium-Term (30–90 days)

- **RDS Proxy adoption** (RELY-446): RDS Proxy will absorb connection churn during failover at the network layer, making the application-level retry wrapper a belt-and-suspenders safety net rather than the sole recovery mechanism.
- **Chaos Engineering Programme** (RELY-447): Quarterly Aurora failover drills in staging with production-level traffic simulation using AWS Fault Injection Simulator (FIS). Results published to SRE reliability dashboard.

7.3. FinOps / Cost Controls

- **Emergency scaling guardrails** (RELY-448): Set AWS Budgets action to notify and optionally block Lambda concurrency above 1500 units during an incident. Review IAM break-glass permissions for emergency overrides.
- **Cost attribution tagging**: all emergency resources deployed during the incident tagged `incident=INC-2024-0847` for FinOps chargeback reporting.

8 Lessons Learned

Lesson 1 **Failover ≠ Transparent**: A database failover completing in 32 s does not guarantee application-tier transparency. Each service must be tested independently against failover events.

Lesson 2 **Hotfix Shortcuts Backfire**: Disabling canary gates for speed during hotfix releases introduces exactly the kind of regression that requires a longer incident response. The tradeoff is almost never worth it.

Lesson 3 **Observability Gaps Kill Diagnosis Speed**: 14 minutes were lost because there was no metric distinguishing "connection pool exhaustion" from generic latency. Instrument what you need to triage, not just what you think will fail.

Lesson 4 **Runbooks Age Poorly**: The DB failover runbook was 18 months old and did not reflect the current API architecture. All Tier-1 runbooks should be reviewed quarterly and tested against actual chaos drills.

9 Sign-off & Review Record

Name	Role	Date	Outcome
Elena Vasquez	Incident Commander	2024-11-20	APPROVED
Priya Nair	Staff SRE	2024-11-20	APPROVED
James Trevino	Senior Engineer	2024-11-21	APPROVED
Marcus Okonkwo	Reliability Scribe	2024-11-21	APPROVED
Anita Desai	Engineering Director	2024-11-22	APPROVED

Document revision history tracked in Git: `git log docs/postmortems/INC-2024-0847.tex`