PostgreSQL Outage Report – Dashsoft Infrastru

author: Julian R.date: 2025-11-26project: Dashsoft Platform Engineering

Overview

This week revolved around diagnosing and resolving a PostgreSQL outage that affected several internal services at Dashsoft. The interruption surfaced suddenly and cascaded across the platform, forcing us to shift focus from planned development toward recovery, root-cause analysis, and infrastructure hardening.

Main Activities

The first signs of trouble appeared when multiple services began timing out during routine database operations. Because PostgreSQL sits at the center of several Dashsoft systems, even small disruptions propagate quickly, and we had to work deliberately to separate symptoms from the root cause. The immediate priority was to confirm whether the issue came from network instability, disk saturation, connection pooling misbehavior, or the database engine itself.

After stabilizing the environment by redirecting traffic through fallback service paths, we inspected the PostgreSQL instance and found that its connection count had grown far beyond the configured maximum. This was traced to a combination of long-running idle transactions and an unexpected spike in parallel request patterns that overwhelmed the connection pool. Once we understood that the database was essentially suffocating under a backlog of unclosed sessions, fixing the outage became a matter of clearing out stuck connections, restarting the affected services in a staggered fashion, and re-establishing a predictable load profile.

Challenges and Solutions

The largest obstacle was the difficulty in identifying the original trigger. Metrics suggested performance degradation before the outage became visible, but none of them alone told the full story. We eventually determined that a recently deployed service had begun opening connections without closing them reliably, and over time this unruly behavior snowballed. The solution involved more than restarting the database: we added targeted fixes to the offending service, implemented additional connection lifetime rules in the pooler, and improved log visibility so that misbehaving components reveal themselves earlier.

Moreover, vacuum processes had been taking unusually long, which amplified the pressure on the server. By reorganizing the table statistics and manually triggering a full auto-vacuum run after recovery, we were able to restore performance to its expected baseline.

Achievements and Progress

By the end of the incident, we restored normal PostgreSQL performance, ensured that no user data was lost, and verified that all services dependent on the database were functioning correctly. The connection leaks were patched, the application pool settings were revised, and the infrastructure regained its stability.

We also documented the entire event, collected relevant metrics snapshots, and produced clear internal guidance that will help other teams avoid similar patterns in future deployments.

Learning and Reflection

The outage reinforced the uncomfortable truth that databases rarely fail without warning; trouble accumulates in the background until it finally becomes loud enough to force attention. The experience deepened our understanding of connection pooling behavior, reiterated the importance of strict transaction boundaries, and highlighted the need for better pre-deployment observation when rolling out backend changes.

Across the team, there was a renewed appreciation for proactive monitoring and the odd balancing act of keeping systems both flexible and guarded.

Plans for Next Week

Next week's efforts will focus on building automated alerts around idle transactions, improving our PostgreSQL dashboards, and formalizing a recurring database-health review. We also plan to implement stricter CI checks on ORM usage patterns, ensuring that connection leaks are caught before they ever reach production. With these steps, we expect to shift from reactive firefighting toward a more predictable and resilient database ecosystem that can keep pace with Dashsoft's platform growth.Here is a revised report that uses every feature described:✓ Metadata✓ Headings✓ Narrative sections✓ Lists✓ Code blocks✓ Tables✓ Professional Markdown suitable for PDF generation

All wrapped inside a top-level code block so you can feed it directly into your generator.

The results confirmed dozens of idle transactions holding locks far longer than intended.

With the cause identified, we proceeded to clean up problematic sessions, restart affected services, and restore a stable connection pattern. After recovery, we performed a controlled VACUUM ANALYZE to normalize table statistics, since vacuum lag had amplified the slowdown.

Challenges and Solutions

The biggest challenge was the subtle and gradual nature of the failure. At first, metrics showed minor slowdowns that did not yet indicate an emergency. Because the connection leak grew slowly, the system appeared fine until it buckled. Tracking the responsible service required combining database logs, application logs, and time-correlated performance graphs.

Once the leaking service was isolated, we patched the offending code. The essence of the fix was ensuring that transactions always closed, even in edge-case failure branches. A simplified pseudocode sketch of the corrected logic is shown here for documentation clarity:

This pattern guarantees clean session closure regardless of unexpected runtime behavior.

Achievements and Progress

The database was restored without data loss, and all dependent services returned to regular operation. We added new monitoring panels, improved log visibility, and introduced connection lifetime limits in our PostgreSQL pool configuration.

To document the outcome of the incident, we assembled a concise internal summary. A short excerpt of the key metrics is provided below:

Post-Incident Metrics Snapshot

Metric	Before Outage	During Outage	After Fix
Active Connections	~120	420+ (stalled)	~95
Idle Transactions	<10	180+	<5
Average Query Latency	12 ms	480+ ms	10 ms
Vacuum Lag	Moderate	High	Low

These values gave the team confidence that the database returned to a healthy baseline.

Learning and Reflection

The outage highlighted the deceptive nature of long-running idle transactions. They accumulate silently and only reveal their long-term cost when it's inconvenient. We also learned that vacuum pressure should be treated as a first-class metric, not as background maintenance. The week served as a reminder that reliability is not achieved through a single tool, but through the interplay of careful coding, disciplined connection management, and proactive monitoring.

Plans for Next Week

Our next steps focus on preventing repeat incidents. We will introduce automated alerts around idle session spikes, strengthen dashboard visibility for connection usage, and require new backend services to pass connection-handling checks in CI. We also plan to add a small test harness that simulates surge-loan traffic and validates pool exhaustion behavior:

Collectively, these measures aim to transform the outage from a disruption into an investment in long-term reliability. The platform becomes sturdier, and our operational understanding deepens in the process.