Problem Statement

On November 28, 2025, DevNest Inc. experienced a catastrophic outage that crippled their Commerce business operations during the rush of Back Friday shopping.

Timeline:

- 009:10 EST: Logins failing across all supported products
- 09:15 EST: Public status page unreachable
- 09:25 EST: 35% of requests globally return 5xx server errors
- 09:40 EST: Executives warn customers of a possible cyberattack
- 10:40 EST: Technical team confirms it is an internal issue, not a cyberattack
- 11:15 EST: Partial restoration of services
- 11:45 EST: Full recovery of services, but backlog and data delay persist for hours

DevNest's senior management and shareholders are concerned that future outages might be sufficiently extended to have a catastrophic financial and business impact.

Incident Report

November 28, 2025

08:42 EST

1. A spike in latency times in the Commerce product triggers an automated response when response times exceed the allowable threshold.
2. Dmitri, on duty in the Operations Center, and the VP of Retail Operations are sent notifications on the Ops Slack channel of the threshold limit being exceeded.
3. Because it is peak retail, the VP demands a fix and return to regular traffic within the hour, or the division will start to incur unacceptable financial losses.
4. Dmitri identifies that the cause is stale service-discovery DNS records for the Commerce API.
5. Dmitri decides to apply a "safe" internal automation runbook to refresh DNS entries.
6. The DNS automation system has a rare bug: when refreshing records for a subset of services, it writes an empty/invalid DNS record to the control plane.

Post mortem reveals:

- Dimitri was operating outside the approved change window
- Bypasses peer review because “it’s an emergency.”
- The automation tool he uses is privileged + global in scope.
- The tool has no hard guardrails to limit blast radius.

08:44 EST: Initial failure (T+2 minutes)

1. The DNS refresh runs.
2. Instead of only touching Commerce discovery, the automation unintentionally updates records for the core NoSQL datastore (DynamoDB-like) because of a bad dependency tag in config.

Result:

1. Commerce services start resolving the datastore endpoint to NULL / empty record.
2. Requests fail immediately
3. Retrying logic kicks in

08:46–09:05 EST: Cascading failure

Retry storms spread across the fleet:

1. Commerce API retries hammer the datastore with exponential backoff.
2. Connection pools thrash, causing thread exhaustion.
3. Dependent services (Identity checkout tokens, Analytics events) also retry because they share the same datastore.

Within 20 minutes:

1. Datastore error rate hits 60–80%
2. Shared service health checks fail
3. Auto-scaling adds more instances… which also retry and worsen the load

09:07 EST: Second-order collapse - edge/security layer

1. With apps failing, customers and bots begin hammering DevNest endpoints.
2. Traffic spikes 3–4× above normal, especially on login/checkout.
3. DevNest sits behind a Cloudflare-like bot mitigation layer. Under load, that layer generates a massive ruleset update (auto-tightening policy) to “protect” services.

But there’s another latent defect:

1. The bot management system can’t handle oversized config files
2. It crashes when a ruleset exceeds internal size limits

DevNest now has:

1. A broken core datastore resolution and
2. A degraded global edge layer

Which explains why the outage feels “everywhere at once.”

10:05 – 10:40: Remediation

1. 10:05 Ops team realizes DNS automation touched the shared datastore
2. 10:22 DNS Records manually repaired
3. 10:40 Datastore stabilizes, retries subside
4. 11:15: Edge layer config rollback restores traffic
5. 11:45: Full recovery, but backlog and data delay persist for hours
6. 14:04: Backlog cleared, full operation resumed

Post Incident Analysis

Root/Primary failure path

Human pressure + privileged access chain of events

1. Unreviewed change in global automation

2. Latent DNS automation bug
3. Invalid datastore DNS record
4. Service discovery failure
5. Retry storm & datastore overload
6. Fleetwide instability

Amplifier/Secondary path

Datastore errors cause a traffic surge, causing:

1. Bot/security layer auto-tightening
2. Oversized config file defect
3. Edge routing crash
4. Global outage footprint expands

Tertiary effects

1. Observability degraded because the metrics pipeline depends on the datastore.
2. Incident response slowed because the status page and dashboards sit behind the edge devices
3. Exec decision-making is impaired due to a lack of trusted telemetry.