CoffeeSprout

# Recovery, Proven

Case study: a full failover test in 49 minutes

CoffeeSprout · Version 1.0
2026

# Contents

# The question: prove it

One of our clients in a regulated industry faced an external audit of the controls around their platform. The auditors didn't ask "do you have backups?" They never do. The question was: **show that they work**. A completed backup with logs, a tested recovery procedure with a report, and recovery objectives based on something real.

That is the difference between having backups and being able to prove them. We decided to make that difference measurable: not one test, but three, on the same day, documented in the auditor's own report templates.

# The setup: three independent layers

The platform runs as a Kubernetes cluster on our own infrastructure. Protection against data loss consists of three layers designed to fail independently of each other:

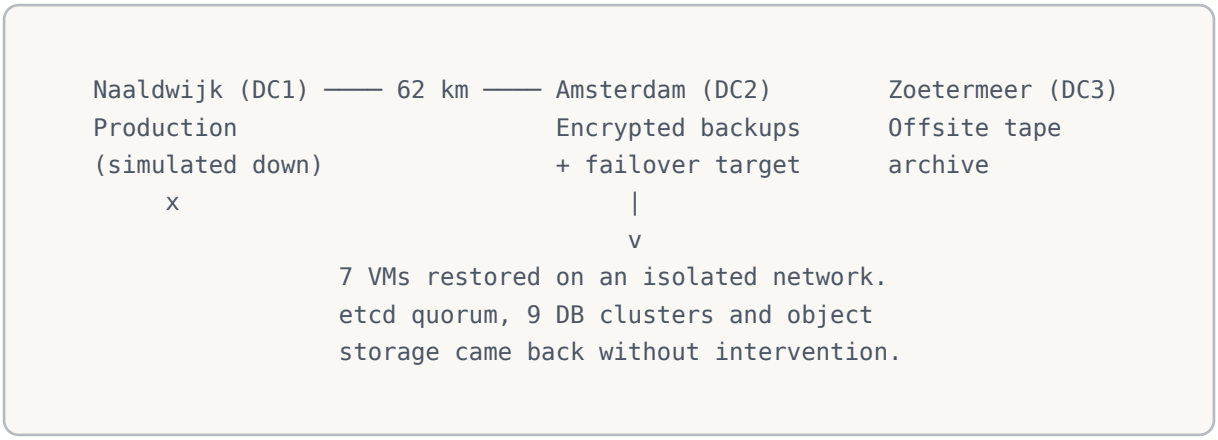
| Layer | What | Details |
|---|---|---|
| VM backups | The complete platform, twice a day | AES-256 encrypted, stored in a different datacenter, daily checksum verification, tiered retention reaching back over 6 months |
| Database backups | Application-consistent PostgreSQL backups + continuous WAL archiving | Point-in-time recovery: restore to any moment, stored outside the cluster |
| Configuration as code | All platform configuration in git | Every change traceable; the platform is reproducible |

But a green dashboard tells you very little. The only way to know these layers work is to restore and verify.

# The infrastructure: three Dutch datacenters

The platform runs on CoffeeSprout's own hardware, spread across three ISO 27001-certified Dutch datacenters. The geographic separation is what makes the failover scenario real: the backups live in a different datacenter than the workload they protect.

| Datacenter | Role in this test |
| --- | --- |
| Naaldwijk (DC1) | Production workload (source), untouched throughout |
| Amsterdam (DC2) | Encrypted backups and the failover restore target, 62 km from DC1 |
| Zoetermeer (DC3) | Offsite tape archive |



# Test 1: Database restore in 90 seconds

The platform's database was restored from the object store to a separate, new database cluster, without touching the running environment. Ninety seconds after applying the recovery manifest, a healthy database was up. Row-level verification: identical to the source, down to the last record.

# Test 2: Point-in-time recovery to the second

The strongest form of database recovery isn't "back to last night" but "back to 13:59, just before the bad change at 14:00". We demonstrated it with a marker test:

1. Write marker A and record timestamp T; six seconds later, write marker B to simulate the mistake.
2. Restore the database to exactly timestamp T.
3. Result: marker A present, marker B absent. Recovery accurate to the second, again in about 90 seconds.

# Test 3: The full failover test

The main test simulated the worst-case scenario: *the primary datacenter is completely unavailable.* All seven virtual machines of the platform, six Kubernetes nodes plus the gateway, were restored from the encrypted backups at the failover datacenter, on a fully isolated network segment. The live environment was never touched.

| Milestone | Time |
|---|---|
| Restore of all 7 VMs (parallel) | 26m23s |
| All VMs booted | +3 min |
| Kubernetes cluster: quorum restored, all nodes Ready | +18 min |
| Data verification complete (9 database clusters healthy, application data checked at row level) | +2 min |
| **Total technical recovery** | **49 minutes** |

No manual intervention was needed: the etcd quorum formed autonomously from snapshots taken minutes apart, all database clusters came up healthy, and the object storage was immediately queryable. No data manipulation, no reconfiguration.

> **Why you test**
> The test also produced two small findings: behaviour that only surfaces in full isolation. Both went straight into the recovery runbook. That's not a blemish on the test; it is why you test. A runbook that has never been held up against reality is a stack of assumptions with a staple through it.

# The result: measurements instead of estimates

> **RTO and RPO**
> **RTO** (Recovery Time Objective): how long recovery may take before the platform is back. **RPO** (Recovery Point Objective): how much data you may lose, that is, how far back the most recent usable copy sits.

| Objective | Value | Basis |
|---|---|---|
| RTO (target) | 4 hours | Including detection, decision-making and response time |
| RTO (measured) | 49 minutes | Full failover test |
| RPO (VM layer) | max 12 hours | Two backups per day |
| RPO (databases) | ≤ 5 minutes | Continuous WAL archiving |

Our client walked into the audit with measured figures instead of estimates, and with test reports in which every claim traces back to literal command output.

> **What the auditor received**
> Three deliverables, the same day: a recovery runbook, two sign-off-ready test reports in the auditor's own format, and the literal command output behind every figure (restore timings, etcd and node status, row-level data checks, bucket listings).

# Why this should be the standard

A backup only exists once the restore is proven. That's why restore testing isn't something we do only for audits. It's part of running a managed platform, with the verification regime agreed per platform. A typical regime:

| What | Cadence |
|---|---|
| Automated backup verification | Daily |
| Database restore test | Quarterly |
| Full failover test | Annually |

The audit makes it visible; the work happens all year.

> **Where this is standard**
> This recovery regime is part of our **Managed Clusters** (dedicated Kubernetes, from €2,500 per month) and our **CaffeineStacks** managed servers (from €300 per month). Same backup layers, same tested restores, on infrastructure we own.

Does your platform run on infrastructure where you can prove this? We're happy to show you, stopwatch included.