



The Pool Named NVMe Lives on Spinning Disk

How a storage name outlived its placement and stalled a SQLite-backed git forge

Paper: PT-R-2026-031 v1.0

Author: PureTensor Ltd

Date: 1 October 2026

Status: Published

Abstract

On 25 September 2026 we re-targeted a replicated Ceph pool, and the Kubernetes storage class that maps to it, from a consumer NVMe tier onto the hard-disk tier. Both the pool and the class carry "nvme" in their names. The change was deliberate and reversible. It was correctly reasoned for the workloads it considered, which were large databases whose expected cost was slower random reads. The report written that day recorded that the storage class names "no longer describe the tier they use." Nothing that consumed the name was re-evaluated. One of those consumers was the data volume of our self-hosted git forge, a Gitea instance backed by a 536 MB SQLite database running in rollback-journal mode.

On 27 September an operator-directed cleanup ran in a separate agent session. It moved about 1.28 TB into a CephFS archive directory on an HDD erasure-coded pool that shares spindles with the "nvme" pool. Archive client writes held at 224 to 245 MB/s between 06:30 and 07:45 UTC. Mean commit latency across the 12 HDD OSDs rose from 7.8 ms at 05:30 to between 71 and 99 ms, and single-OSD maxima at 10-minute samples reached 184 to 294 ms. At the forge, `git push` and `git ls-remote` hung past a 60 s timeout. Over 55 minutes the logs recorded 1,647 slow-query warnings of 16 to 60 s each, all on single-row primary-key selects. An in-pod write test with `fsync` ran at about 134 KB/s, and the liveness probe killed a cold start that needed 3 min 43 s, in a loop.

Switching SQLite to WAL mode and adding a startup probe stopped the restart loop but did not stop the stall. Database-backed requests still took about 3.5 s each, 41 slow requests per minute were logged, and warming the page cache made no difference, which placed the bottleneck on write `fsync` rather than reads. We then moved only the database file, leaving the 45 GB of repositories where they were, onto a new 4 GiB volume in the SSD pool. After the move `git ls-remote` over SSH took 0.16 s, API requests took 5 to 15 ms, and slow requests fell to 0. The remediation had its own cost. A recursive ownership change ran for more than 17 minutes and left 21 pull-mirror private keys group-readable, and every pull mirror failed until 08:32.

This is one incident on one morning, with no controlled reproduction. The before/after comparison for the SSD move is confounded by time, because the archive job ended at 07:53:22 and the move was committed at 08:09:31. Three findings survive that confound: WAL alone did not stop the stall under load, the database has stayed responsive on the SSD tier since, and under load the SSD tier averaged 12 ms commits against 82 ms on HDD. The SSD tier chosen for the fix is the same consumer tier, without power-loss protection, that stalled



under CI sync writes two days earlier. The durable fix, a metadata tier on power-loss-protected NVMe, is designed and not yet fitted.

1. A Name Chosen Once

The fleet runs a k3s Kubernetes cluster whose persistent volumes are Ceph RBD images (block devices striped across the Ceph cluster), with CephFS used for archive data. Ceph runs the squid release and spans four storage hosts. Each Kubernetes storage class maps to a Ceph pool. Each pool has a CRUSH rule, the Ceph placement rule that decides which device class, HDD or SSD, holds its data. The storage class name is fixed when the class is created, and every volume claim that uses the class refers to it by that name. The pool's placement can be changed underneath the name at any time. The name itself stays.

One replicated pool, and the storage class mapped to it, have "nvme" in their names because both were created on a consumer NVMe tier. The pool holds the cluster's latency-sensitive databases (pgvector, PostgreSQL, Neo4j, Redis) and, among other things, the data volume of the git forge. On 25 September it stored about 1.1 TB.

The forge serves git over SSH and HTTP for the fleet's repositories and runs pull mirrors of repositories hosted elsewhere. About a dozen CI runner agents (Gitea act_runners) poll it constantly, reading and updating the runner table. Its SQLite database was 536 MB and ran in rollback-journal mode. This is SQLite's default: a writer takes a lock on the whole database, and readers queue behind it until the write commits. The repository volume was 45 GB.

Agent sessions working under operator direction diagnosed and repaired the whole chain. On 27 September two agent sessions were working in parallel. That fact explains most of what follows.

2. Why the Database Pool Went to Spinning Disk (25 September)

The solid-state slow-op episode

On the morning of 25 September Ceph raised a BlueStore slow-operation warning on two of the four SATA SSD OSDs. An OSD is the Ceph daemon that manages one storage device, and BlueStore is its on-disk backend. Our threshold is 100 or more slow ops within 1,800 s, so the warning indicates recent slowness, not accumulated history. The solid-state class had only 4 OSDs, one per storage host. All were consumer SATA SSDs without power-loss protection (PLP), and their RocksDB/WAL (BlueStore's metadata database and its write-ahead log) sat on a consumer M.2 NVMe drive. The HDD OSDs are enterprise drives. SMART was clean and every placement group was active+clean, so this was not a disk failure.

Measure (25 Sep, morning)	Value
BlueStore ops over 5 s, OSD A / OSD B	431 / 714
Median / max duration of those ops	about 5.7 s / 9.7 s
Dominant stall phases	kv commit (_txc_committed_kv), plus transaction throttle and submit
Burst windows	about 08:25, and 10:47 to 11:07
Solid-state replicated pool write rate, bursts / baseline	21.5 MiB/s / 0.09 MiB/s
Writer	one RBD image used as CI runner scratch, at 22.2 MiB/s



Measure (25 Sep, morning)	Value
Concurrent CI jobs	28 GitHub Actions jobs on one repository (5 more earlier in the morning)
Weakest SSD, avg aio wait / kv flush latency	13.7 ms / 15.5 ms (peers about 2.5 to 4.3 ms)

The interpretation recorded at the time was that sustained synchronous writes stall the cache flushes of SSDs without PLP. A drive without PLP must flush its volatile cache to honour each sync write, whereas a PLP drive can acknowledge from protected cache. The diagnostic recipe had three steps. We compared per-OSD BlueStore performance counters for aio-wait, kv-flush and kv-sync latency, which separates the data device, the flush and the WAL. We joined per-pool write-byte rates to pool metadata in Prometheus. Finally we read per-RBD-device written bytes on client hosts to find the image responsible.

The re-architecture

The operator's design target treats Ceph as primarily archival. It keeps the HDD tier, puts HDD RocksDB/WAL on one PLP enterprise M.2 NVMe per host, retires the consumer NVMe tier, and keeps the four SATA SSDs as a small self-contained tier for light workloads.

Step (25 Sep)	Time (UTC)	Result
CI scratch moved off Ceph to a compute node's local NVMe RAID	live copy first (407 GB, about 2.6 million files, 254 MB/s); cutover 14:12 to 14:13:21	CI down about 80 s, no job killed
SSD OSDs made self-contained (DB moved onto own device), one at a time	14:14:59 to 14:18:28	each OSD down about 40 s; 785 PGs active+clean after each
Phase A: metadata and EC pools to SSD tier, one CephFS data pool to HDD	14:19:30 to 16:52:17	online backfill, nothing degraded; SSD latency mostly 3 to 50 ms, one 1.7 s spike
Test of 3 concurrent backfills per SSD	about 16:05 to 16:14	2.3 s stall and a slow-op warning; reverted
Phase B: the "nvme"-named database pool (about 1.1 TB) to the SSD tier	after 16:52	see next table
Database pool re-targeted to the replicated HDD CRUSH rule	19:25:15	about 15,400 objects/min; HEALTH_OK from about 19:45
All PGs active+clean	21:31	785 PGs
Consumer NVMe OSDs (8) destroyed, gated on 0 PGs and safe-to-destroy	21:32 to 21:33	16 OSDs remain: 12 HDD, 4 SSD

The plan initially sent the database pool to the SSD tier. The consumer SSDs responded as follows.

Phase B on the consumer SSDs	Value
One SSD OSD: slow ops	122 in 15 minutes
Same OSD: commit stalls	up to 4.7 s
Weakest SSD OSD: aio_submit retries	39 in 10 minutes (the recorded precursor of an earlier crash-loop on that OSD)
Backfill concurrency after cut-back	1 per SSD
Projected remaining sustained writes at that rate	about 12 hours

Three reasons were recorded for sending the pool to HDD instead. HDD placement fits the operator's design of HDD for data, PLP NVMe for metadata, and a fast tier later. An existing request already asked for heavy writers to be moved off the consumer SSDs. The change also avoided about 12 hours of load that had previously preceded a crash. If the two weak SSDs had



failed together, the SSD erasure-coded pool, which holds the monitoring stack and several small service volumes, would have stopped serving I/O. The lesson recorded was never to bulk-migrate onto the consumer SSDs.

The report named the cost in its own words: the pipeline databases "now have slower random reads until a PLP fast tier exists." It noted that the storage class names "no longer describe the tier they use. They still work," and that "one rule change reverses the deviation." The report assessed the random-read cost for the big databases. It did not assess fsync latency for small SQLite workloads, or what would happen when bulk archive writes shared the same spindles.

The end-state checks at about 21:35 found that no pool used an NVMe rule, that all database pods had run continuously since 22 September with no restarts during the migration, and that all 70 volume claims were bound. From then on the pool and storage class named for NVMe had no NVMe beneath them, and no NVMe OSDs existed in the cluster. The HDD RocksDB/WAL remained on the consumer M.2 NVMe, without PLP, on 27 September. The PLP metadata tier was designed, scripted and dry-run; the dry run exited as designed because no drive was present. It had not been fitted by the time of publication.

3. The Co-tenant (27 September)

An operator-directed cleanup freed space on a two-GPU compute node's local RAID array. It moved model weights and one disk image into a CephFS archive directory backed by an HDD erasure-coded pool. There were eight transfers, each an rsync with 8 parallel streams, and each source was deleted only after a parity check passed. All 8 passed. The items were 492 G, 192 G, 126 G, 52 G, 182 G, 25 G and 182 G, plus a 24 G tarball, about 1.28 TB in total. The job ran from 06:16:51 to 07:53:22 (1 h 36 min 31 s) and averaged about 0.2 GB/s. The transfer report notes that this was roughly half the throughput previously documented for the tier.

A different agent session ran this job from the one that later diagnosed the forge. The forge-recovery report lists the owner of the archive transfer as unidentified. The cleanup report from the same morning shows that it was operator-directed work in a parallel session. Neither session knew that the other's load was relevant to it.

We queried Prometheus for this paper. The commit latencies below are Ceph's exported per-OSD gauges.

Time (UTC)	HDD EC archive pool client writes (MB/s, 10 m rate)	HDD OSD commit latency, mean of 12 (ms)	SSD OSD commit latency, mean of 4 (ms)	Database ("nvme") pool write ops/s (10 m rate, 30 m samples)
05:30	21.9	7.8	12.3	23.7
06:00	17.3	21.3	6.5	16.9
06:15	28.0	22.2	16.5	
06:30	241.6	82.4	13.0	7.6
06:45	242.4	86.0	9.5	
07:00	245.3	87.3	16.3	10.2
07:15	229.9	99.2	40.3	
07:30	224.2	81.3	25.3	4.6
07:45	243.4	71.3	569.8	
08:00	64.6	24.6	18.0	35.9
08:15	16.9	37.3	17.0	
08:30	21.2	12.4	10.0	18.4



Time (UTC)	HDD EC archive pool client writes (MB/s, 10 m rate)	HDD OSD commit latency, mean of 12 (ms)	SSD OSD commit latency, mean of 4 (ms)	Database ("nvme") pool write ops/s (10 m rate, 30 m samples)
09:30	17.3	7.8	5.3	8.3

The HDD column follows the archive column closely. Mean commit latency sits in single digits before the job and returns to 7.8 ms by 09:30. For the whole plateau it stays between 71 and 99 ms. At 10-minute samples the worst single HDD OSD during the archive window reached 184 to 294 ms, for example 259 ms at 06:30, 273 ms at 07:20 and 294 ms at 07:30. The lulls at 06:10 to 06:20 and at 08:10 to 08:20 show 12 to 36 ms. Single-OSD spikes of 287 ms at 08:30 and 211 ms at 08:50 appear after the archive had ended, so the plateau is not the only source of tail latency on those disks.

The database pool's write operations fell from about 17 to 24 per second before the archive to about 5 to 10 per second during it, and rebounded to about 36 per second at 08:00. That rebound probably combines a backlog with recovery work; we have not decomposed it. The SSD tier shows two single-OSD commit spikes inside the window, 3,472 ms at 06:50 and 2,826 ms at 07:20, and a high 15-minute mean at 07:45. We have not established their cause. The CephFS metadata pool lives on the SSD tier, which makes it a candidate, but that is a hypothesis only.

4. What the Forge Saw

Measure	Value
<code>git push</code> / <code>git ls-remote</code>	hung for 60 s or more (60 s timeout)
Health endpoint	timed out
Log errors	"Unable to get repository owner: ... context canceled"; later "Unable to get public key: context canceled"
Slow-query warnings	1,647 in 55 minutes, each 16 to 60 s
Tables in slow queries	runner table, public keys, users, access tokens: single-row primary-key selects
In-pod write test	1.2 MB written with fsync in 8.9 s, about 134 KB/s
Cold start on the loaded tier	3 min 43 s (database ping about 60 s)
Liveness probe budget	60 s initial delay plus 3 x 30 s: killed the starting pod in a loop

The shape of the failure follows from rollback-journal mode. A write holds a lock on the whole database until its fsync returns. The runner agents write to the runner table continuously, so a writer is almost always present. When each fsync takes as long as the loaded spindles need, every read queues behind the lock, including the single-row lookup of a public key that authenticates an SSH connection. The "context canceled" errors are those reads giving up. The same mechanism turned a slow start into a restart loop. The database ping at start took about 60 s, and the liveness probe (the Kubernetes check that restarts a container it judges dead) allowed a 60 s initial delay plus three 30 s failures before killing the pod.

Time (UTC, 27 Sep)	Event
06:16:51	Archive job starts (the forge incident note says 06:07; the transfer log and Prometheus support about 06:16 to 06:30)
06:30 to 07:45	Archive pool writes 224 to 245 MB/s; HDD commit mean 71 to 99 ms



Time (UTC, 27 Sep)	Event
(time not recorded)	Forge git operations time out; slow-query warnings accumulate
07:28:49	WAL plus startupProbe committed (applied live)
07:53:22	Archive job ends, 8 of 8 parity passes
08:09:31	Database-to-SSD move committed
08:32	Pull-mirror key modes restored; 19 mirrors re-sync

5. The Diagnosis Chain

The diagnosing session worked from the application down, with one cheap test at each step:

1. Forge logs, with slow SQL grouped by table. The slow queries were tiny primary-key selects, which ruled out a query problem.
2. A `dd` with `fsync` inside the pod, which ran at 134 KB/s. The volume was slow, not the application.
3. `ceph osd perf` and per-pool statistics. HDD OSDs showed commits in the 100 ms class, and one pool was writing about 240 MB/s.
4. The volume's pool and that pool's CRUSH rule. The "nvme" pool was on the HDD rule.
5. Per-host network transmit, then the process on the transmitting host, which was the archive job.
6. A page-cache warm test after WAL was enabled. It made no difference, so the workload was `fsync`-bound and not read-bound.

Step 4 is the only step in which the answer had to be read from configuration and could not have been assumed. A reader who took the pool name at its word would have excluded the storage layer after step 3, on the grounds that the forge's data lived on NVMe and the slow OSDs were spinning disks.

6. Two Fixes, and What the First One Showed

Fix 1: WAL mode and a startup probe (committed 07:28:49)

We set SQLite's journal mode to WAL (write-ahead log) through Gitea's environment configuration. In WAL mode readers proceed while a write is in progress, and commits need fewer `fsync`s. We also added a Kubernetes `startupProbe` on the version endpoint, 15 s x 40, which gives a 10-minute start budget during which the liveness probe is suspended. The restart loop stopped. The stall did not. Git over SSH still timed out, database requests took about 3.5 s each, 535 goroutines were waiting, and 41 slow requests per minute were logged. We then read the whole database file to warm the page cache, and nothing changed. If reads had been the constraint, a warm cache would have helped. It did not help, and WAL had already removed reader blocking, so the remaining cost was write `fsync` latency itself.

Fix 2: the database file alone moves to the SSD tier (committed 08:09:31)

We created a new 4 GiB RBD volume in the replicated SSD pool that holds only the SQLite database, and pointed Gitea's database path at it. The 45 GB repository volume stayed on the HDD-backed "nvme" pool, and WAL and the startup probe were kept. The backup job now mounts the new volume. The migration went as follows: scale the forge to 0, then have a helper pod copy the database together with its `-wal` and `-shm` files. The WAL had not been



checkpointed at shutdown, so those three files had to move together. We confirmed that sha256 digests matched on both sides, renamed the originals and kept them as a rollback, and applied the change.

Measure	Before (HDD pool, WAL on)	After (DB on SSD tier)
Time to ready	more than 2.5 min (3 min 43 s on the first loaded start)	40 s
<code>git ls-remote</code> over SSH	60 s timeout	0.16 s (162 ms)
API request latency	about 3.5 s per DB-backed request	5 to 15 ms
Slow requests per minute	41	0
Average OSD commit under load (tier comparison)	HDD 82 ms	SSD 12 ms

The final row is the comparison we used to make the decision, and it matches the Prometheus HDD mean of 82.4 ms at 06:30. The incident notes also describe HDD commit latency under load as "100 to 240 ms" in one place and "80 to 200 ms" in another. Section 9 discusses those ranges.

7. The Remediation's Own Blast Radius

The first migration pod mounted the 45 GB repository volume with an fsGroup (a pod setting that makes the kubelet set a group owner on the volume's files) but without fsGroupChangePolicy: OnRootMismatch. The kubelet therefore changed group ownership of every file on the HDD tier recursively. The pod sat in VolumePermissionChangeInProgress for more than 17 minutes, and we could not interrupt it. The workaround was a second pod pinned to the same node, since a ReadWriteOnce volume can be shared by pods on one node.

The same recursive change set every pull-mirror private key to mode 0660, 21 keys in all. SSH refuses group-readable private keys ("Permissions 0660 ... too open", "bad permissions"), so every pull mirror failed. At 08:32 we restored private keys to 0600, public keys to 0644 and directories to 0700. Nineteen mirrors then re-synced cleanly. The remaining 2 keys are stale and belong to deleted repositories. The forge's JWT signing keys had also been loosened to 0660 and were restored to 0600. A scan for other group-readable or world-readable private keys found none. The deployment now sets OnRootMismatch.

During the outage one repository diverged between the forge and its GitHub counterpart, with 2 commits on each side that overlapped only in a version file. We reconciled it with a merge.

8. Reading the Chain

A storage class name records a decision made once. After 25 September the pool and class named for NVMe had no NVMe under them. The change was reversible, correctly reasoned for the workloads it considered, and written down. The step it lacked was a pass over the consumers of the name. A SQLite database, whose performance depends almost entirely on fsync latency, was still requesting "nvme" and was getting spinning disk with its RocksDB/WAL on a consumer NVMe without PLP.

Slow disks alone did not cause the failure; it also needed a co-tenant. Before the archive job, HDD commits averaged about 8 ms. The sources record no forge problem during the roughly 33 hours the forge had already spent on the HDD-backed pool. A bulk write of about 240 MB/s into a different pool on the same spindles raised mean HDD commit latency about tenfold (7.8 ms,



then 82 ms, then 99 ms at the plateau's worst sample), with single-OSD peaks near 300 ms. Measured fsync throughput at the forge fell to about 134 KB/s.

Rollback-journal SQLite turns fsync latency into total unavailability, because one slow writer holds the whole-database lock and every read queues behind it. Liveness probing then turns a slow start into a restart loop. WAL and the startup probe removed the second mechanism but not the first. Together with the null result from the page-cache test, that is the evidence that the constraint was write fsync latency.

The fix was placement by access pattern at the level of a single file. The 536 MB database needs low fsync latency, and the 45 GB of repositories evidently tolerate the HDD tier. Moving only the database removed the symptom without moving the bulk. The tier we moved it to is the same consumer, non-PLP SSD tier that stalled for 5 to 10 s under CI sync-write bursts two days earlier. A 536 MB database with light write volume is the kind of light workload that tier was kept for. A 1.1 TB database pool was not. Both conclusions are consistent with what the tier did on 25 September and on 27 September.

9. What These Numbers Will Not Carry

This is one incident, affecting one forge, on one morning. There was no controlled reproduction. We did not re-run the stall with and without WAL, or with and without the archive load, under matched conditions.

The before/after comparison for the SSD move is confounded by time. The archive job ended at 07:53:22 and the move was committed at 08:09:31. Mean HDD commit latency had already fallen to about 25 to 37 ms by 08:00 to 08:15, although single-OSD spikes of 287 ms still occurred at 08:30. The sources do not record exactly when the "about 3.5 s per request, 41 slow per minute" measurement was taken. Some of the improvement we attribute to the move may therefore be the load ending. Three claims survive this confound. WAL alone did not stop the stall while the archive load was present. The database has stayed responsive on the SSD tier since the move. The SSD tier's average commit under load was 12 ms against 82 ms on HDD. The claim that the move by itself reduced API latency from about 3.5 s to 5 to 15 ms does not survive as a clean causal attribution.

The commit latency figures come from Ceph's exported per-OSD gauges, sampled at 10- to 15-minute steps, so short spikes are under-sampled. The incident notes' ranges of "100 to 240 ms" and "80 to 200 ms" and the Prometheus 10-minute maxima of 184 to 294 ms disagree at the edges. They should be read as ranges; none of them is a reliable single peak.

The 134 KB/s fsync figure comes from a single dd of 1.2 MB with fsync inside the pod. The count of 1,647 slow queries in 55 minutes is reliable, but the sources do not give the start and end of that window. For the start of the archive job, the incident note says 06:07 and the transfer report says 06:16:51, with cache deletions just before. We have used the artefacts' time.

The 25 September slow-op figures are counts from two OSD logs on one day under one CI burst pattern. They establish that these particular consumer SSDs stall under sustained sync writes. They do not establish a general rate for the drive class. Our reading at the time attributed the stalls to the drive class as a whole; whether one drive dominates the tier's tail is a separate question we are now examining, and it does not change the account of the forge given here. We have not established the cause of the SSD-tier commit spikes of 3,472 ms and 2,826 ms on 27 September, or why the archive ran at about half the tier's previously documented throughput.

Nothing here measures the pgvector, PostgreSQL, Neo4j or Redis databases that share the HDD-backed pool. Their exposure to bulk co-tenant writes is inferred from the mechanism and has not been observed. We have no purchase figures. The PLP metadata tier's effect is unmeasured because the tier does not yet exist.



10. What We Changed, and What Comes Next

The forge now runs SQLite in WAL mode with a startup probe, and its database file lives alone on a 4 GiB volume in the replicated SSD pool. Its repositories remain on the HDD-backed pool. The deployment sets `fsGroupChangePolicy: OnRootMismatch`, key modes have been restored, and the scan for other loosened private keys came back clean.

These are containment measures. The durable fix is the PLP NVMe metadata tier, which is designed, scripted and dry-run but not fitted. Once it is fitted, the obvious experiment is the one this incident lacks: a controlled bulk write into the HDD archive pool, run with and without the new metadata tier, measuring `fsync` latency from inside a pod on the "nvme" pool and from the co-tenant databases on that pool. Moving the forge to PostgreSQL is an alternative, but the PostgreSQL volumes currently sit on the same HDD-backed pool, so on its own it would change the database engine and leave the underlying disks as they are.

PureTensor operates a sovereign AI infrastructure fleet (on-premises GPU compute, storage, and serving) run day-to-day by autonomous agents under human direction. This paper is PT-R-2026-031. The measurements were taken on 25 September 2026 (the solid-state tier slow-op episode and the Ceph re-architecture) and 27 September 2026 (the forge stall, its diagnosis and fix), and the Prometheus series were queried for this paper; raw artefacts, including transfer logs, incident notes and metric exports, are retained.