PURETENSOR
INTELLIGENCE DISTILLED

The Brief Blamed the Disks

Slow etcd Applies Fell 91% After We Removed 138 MB of Scanner Reports, While a Healthy Cluster Showe

Paper: PT-R-2026-069 v1.0

Author: PureTensor Ltd

Date: 8 November 2026

Status: Published

Abstract

On 8 November an internal issues brief listed ten items for our Kubernetes control plane. Item six said etcd, the consensus store behind the cluster, was logging slow applies and blamed the write path of our small boot media. We checked. The disks were healthy: write-ahead-log fsync p99 was between 3.89 and 5.07 ms across every member and window we measured, and no proposal failed. The database was the place to look. A security scanner's software-bill-of-materials (SBOM) generation was switched on, and 432 SbomReport objects held 138.25 MB of a database whose in-use size was about 177 MB (78%).

At 11:43:54 UTC we turned generation off and deleted the reports. The in-use size fell from about 177 MB to 37 MB, and slow applies (requests that took more than 100 ms to apply) fell from 591 in the 12 hours before to 52 in the 12 hours 13 minutes after, a 91% drop on a per-hour basis as well. Disk latency barely moved while that happened, which is hard to square with the disk story. Our working summary said "slow applies to zero"; the raw counters show about 16 minutes of zero, then events resumed. We also proposed that the large reads queued small lease writes for 500 to 900 ms. The log does not show that, so we report a before and after result with a plausible mechanism, not a measured one.

The same day our operator asked why our fleet console showed many yellow pods on a cluster where 189 of 189 pods were healthy. The board was wrong about meaning, not state: it judged placement by a pod's own preference after the scheduler had deliberately outranked it, graded planned reboots as churn, and counted seconds-old job pods as unhealthy. Behind the false yellow sat one real fault it could not show: a descheduler that evicted a graph database six times in 50 minutes.

The two stories share one lesson: a plausible hardware explanation and a plausible "something is wrong" colour were both wrong in detail, and reading what the system logged corrected both. This is one cluster, one day and one bundled change; Section 8 states what survives.

1. An Item That Named the Wrong Component

The control plane of our K3s cluster runs an embedded etcd on three small always-on members, carrying roughly 190 to 210 pods. Two are x86 mini-PCs with a root volume on a SATA SSD; the third is a Raspberry Pi 5 booting from NVMe.

The brief's sixth item attributed the slow applies to the eMMC or SD write path on the Pis. We had two questions: are they a disk problem and, if not, what are they; and why did the Kubernetes view of our fleet console show many yellow pods when everything was Running and Ready?

Inspection already contradicted the premise: only one member is a Pi, and it boots from NVMe, not eMMC or SD; the other two are x86 hosts with SATA SSDs. We could not read the device behind the etcd data directory, so the layout below is the root volume's.

Member	CPU architecture	Root storage	Role at 11:40 UTC
A	x86_64	SATA SSD	follower
B	x86_64	SATA SSD	follower
C	aarch64 (Raspberry Pi 5)	NVMe SSD	leader

Table 1. The three etcd members.

2. The Disks Were Healthy

We took etcd's own Prometheus histograms for the 24 hours ending 11:40 UTC (before the change described below) and the 12 hours ending 23:59 UTC (after). The p99 values are histogram quantiles over exponential buckets, so they are not exact.

Member	WAL fsync p99 before (ms)	WAL fsync p99 after (ms)	Backend commit p99 before (ms)	Backend commit p99 after (ms)	Failed proposals before / after
A	5.07	4.99	4.00	4.00	0 / 0
B	4.83	4.73	4.00	4.00	0 / 0
C	4.53	3.89	3.53	3.22	0 / 0

Table 2. Disk-side latency and proposal failures, before and after the change. WAL is the write-ahead log; fsync is the flush to stable storage.

The brief's fsync figure of 4 to 5 ms is confirmed. The numbers barely moved across a window in which slow applies fell by 91%: within 0.1 ms on A and B, and 0.6 ms lower on C. A p99 over roughly 837,000 fsyncs a day cannot see rare stalls, so we also counted them in the 24 hours to 11:40 UTC. Fsyncs slower than 64 ms numbered 9, 18 and 254 on A, B and C, and slower than 128 ms numbered 2, 11 and 23. Against the per-member slow-apply counts of 756, 206 and 323, the 128 ms fsync counts are far smaller, though member C's 254 fsyncs over 64 ms (against 323 slow applies) show the p99 alone is not the whole story. The before window is 24 hours and the after 12, so their quiet-hour mix differs. Slow-apply counts do not order by disk class either: the SATA members had the highest and lowest counts (756 and 206) and the NVMe member sat between at 323.

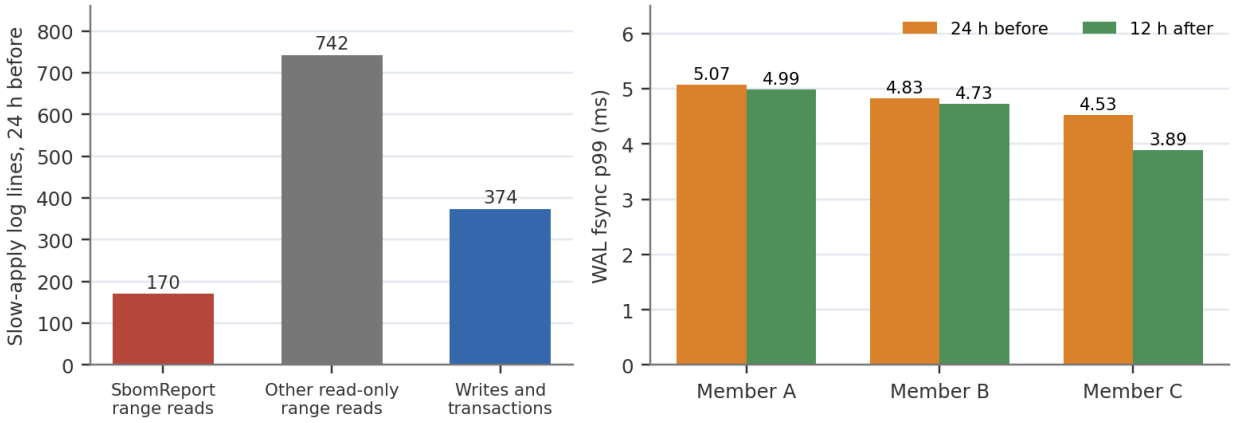


Figure 1. Left: composition of the 1,286 slow-apply log lines in the 24 hours before the change. Right: WAL fsync p99 on each member before and after. Values are in Tables 2 and 4.

3. What Was Slow, in etcd's Own Log

etcd logs the warning "apply request took too long" when a single request takes longer than 100 ms to apply. Each member also keeps a slow-apply counter. The two agree: over the 24 hours to 11:40 UTC the counters rose by 1,285, and the retained log, read to 11:46, holds 1,286 matching lines; the extra line plausibly falls in those last six minutes.

Member	Counter at 7 Nov 11:40	Counter at 8 Nov 11:40	Slow applies in 24 h	Log lines in 24 h
A	471	1,227	756	757
B	147	353	206	206
C	90	413	323	323
Total			1,285	1,286

Table 3. Slow applies in the 24 hours before the change, from the raw cumulative counters and from the log (log window ends at 11:46).

We classified the 1,286 log lines by what was being applied.

Class	Lines	Share
Whole-collection range read of the scanner's SbomReport kind	170	13.2%
Other read-only range reads (secrets, pods, replica sets, config-audit reports and others)	742	57.7%
Writes and transactions	374	29.1%

Table 4. Slow-apply log lines by request class, 24 hours before the change.

Only 13% of the slow lines were the SBOM read itself; most slow requests were other reads. Lease-keyed requests, the periodic lease renewals and leader-election writes that Kubernetes components make, were the largest single key class: 389 lines (30.2%), median 199 ms, 48 at or above 500 ms of which 44 fell between 500 and 900 ms, maximum 2,441 ms.

The SbomReport read is interesting because of its size: the logged response was 138,250,366 bytes for 432 objects, about 320 KB per report. Its slow-logged durations had a minimum of 100.07 ms, a median of 110.8 ms, 11 reads at or above 300 ms, 4 at or above 500 ms and a maximum of 1,148 ms.

Most whole reads never crossed 100 ms and were never logged: the API servers' list counter for this kind rose by about 63 to 65 an hour on each member at 11:40 UTC. The 170 logged reads are about 3.7% of an estimated 4,600 a day (64 an hour, times three members, times 24 hours), extrapolated from the rate in a single hour and not a count.

4. What Filled the Database: Reports Stored in the Consensus Store

The scanner is the Trivy Operator (Helm chart 0.33.2, application 0.31.2), and its SBOM generation was switched on. Its SbomReport objects are custom resources, so they live in etcd. The metrics show 432 of them, plus 3 cluster-scoped SBOM reports, on every API server through 11:40 UTC. Nothing in this cluster consumed them (only RBAC rules named the kind).

Quantity	Value
SbomReport objects	432
Bytes in a whole read	138,250,366
In-use size at 11:40 UTC (A / B / C)	176.95 / 177.03 / 176.99 MB
Share of in-use database held by reports	78% (138.25 of 177.0 MB)
Allocated etcd file size	232.7 / 230.2 / 233.0 MB, unchanged all day

Table 5. The size of the reports against the database that stored them.

This is our retention and sizing choice, not a defect in the scanner: putting 432 reports of about 320 KB each into the consensus store was ours to prevent, and the Trivy Operator supports an alternative report store.

5. The Change and the Result

At 11:43:54 UTC one Helm upgrade set SBOM generation off, leaving every other live value unchanged. We then deleted the existing SbomReports and the cluster-scoped ones. The object-count metric read 432 on all members at 11:45 (one already at 103) and 0 on all at 11:50. Vulnerability reports, which feed our CVE alerting, were not touched.

Time (UTC)	In-use size, member A (MB)	SbomReport objects
11:00	177.09	
11:10	177.18	
11:20	177.80	
11:30	176.93	432
11:40	176.95	432
11:48	177.51	
11:50	178.43	0
11:51	180.61	
11:52	181.15	
11:53	37.34	
12:00	38.23	0
12:20	39.11	

Time (UTC)	In-use size, member A (MB)	SbomReport objects
12:30	38.27	
13:00	37.97	

Table 6. In-use database size on member A through the change. The 11:35 and 11:45 samples also read 432 objects; the 11:55 sample read 0.

The size peaked at 181.15 MB at 11:52, then fell to 37.34 MB at the first compaction after the deletion. The fall was about 139.6 MB, close to the 138.25 MB payload plus the cluster-scoped reports. The allocated file did not shrink (we did not defragment).

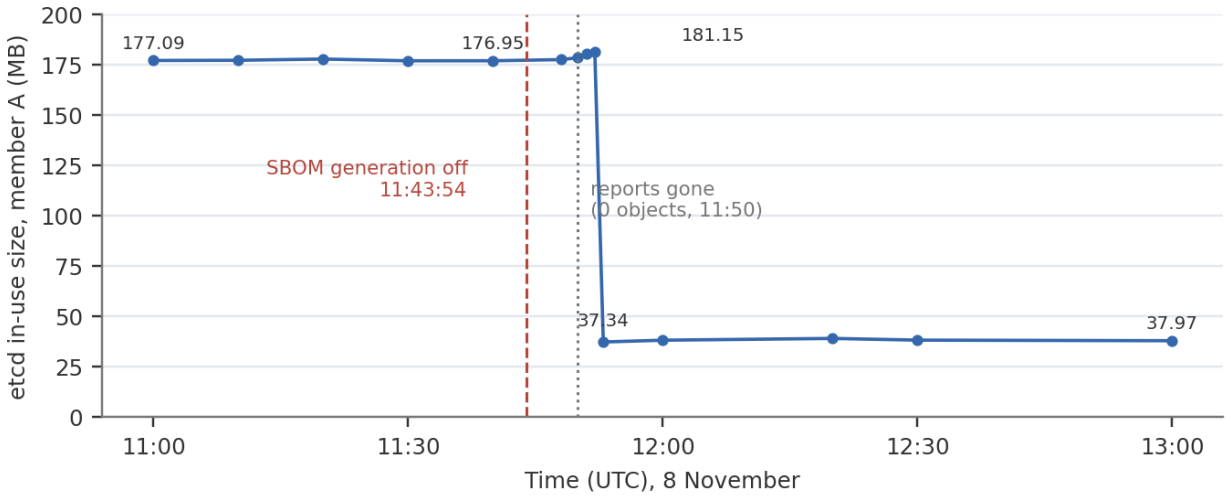


Figure 2. etcd in-use size on member A across the change. Values are in Table 6.

Slow applies, over the 12 hours before the first full post-change sample at 11:46 (from 23:46) and the 12 hours 13 minutes after it, to 23:59:

Member	12 h before	12 h after	Change
A	411	37	-91%
B	108	15	-86%
C	72	0	-100%
Total	591	52	-91%

Table 7. Slow applies in the windows either side of the change. The "after" figure is the raw counter delta over 12 hours 13 minutes; the retained log returns the same 52 lines. Per hour the totals are 49.3 and 4.25, still a 91% fall.

Member A went from about 34 an hour to about 3. Post-change durations had a median of 121 ms and a maximum of 557 ms, against a maximum of 2,465 ms before (a range read of the pod-templates collection).

The two windows cover different parts of the day, and the rate varies with it (member A logged 50, 41 and 60 an hour in the small hours and morning, and 0 to 9 in the evening). The raw counters give a same-time-of-day control: from 11:40 to 23:59 on 7 November the members rose by 347, 115 and 251, a total of 713, against 52 over the same hours on 8 November. That answers the diurnal objection, though it is still one day against one day.

6. Two Claims That Did Not Survive Our Own Data

Our working summary said slow applies went to zero. They did not: all three members were at zero for about 16 minutes (11:46 to 12:02) and member A for about 27 (to 12:13), when the first events came on B and A respectively. The "first 10 minutes" result in our working summary was true and short. Of the remaining 52 events, 37 fell in these classes: reads of the secrets collection (11), config-audit reports (5), config maps (5) and replica sets (4), plus lease writes in two namespaces (12).

The second claim was the mechanism: that the 138 MB reads occupied etcd and queued small lease writes behind them for 500 to 900 ms. The lease writes of that length exist (44 of them). But we tested proximity in the 24 hours of pre-change logs and found no link. Of 1,116 non-SBOM slow lines, 0 fell within 1, 2 or 5 seconds after a slow SbomReport read on the same member. A baseline that shuffled the event times gave means of 0.8, 2.1 and 4.9 lines (maxima 4, 6 and 14 over 200 shuffles). Chance alone would give a few such lines, and we saw none, so the log offers no queueing signature. (The baseline is a single seed of 200 shuffles, uniform over the window.)

The test has limits. The "took" field is the apply time of that request, not time spent waiting, and etcd applies requests serially, so we tested proximity, not overlap. Requests faster than 100 ms are not logged. And 118 of the non-SBOM slow lines were reads of collections over 1 MB, so large reads are not unique to the scanner.

What we can support is a before and after result with a plausible mechanism: a 138 MB read about once a minute on every member, against a database 4.8 times larger than it needed to be (177 MB against 37 MB). The log does not prove how.

7. A Healthy Cluster and a Yellow Board

The second question had a separate answer. On 8 November the Kubernetes view of our fleet console reported many yellow pods on a cluster with 189 of 189 pods Running and Ready. The trigger was a planned reboot of one storage host from 08:52 to 09:06 UTC, which drained 19 pods onto three other nodes where they stayed. Before our fixes the board showed 9 pods flagged off-preference and 28 restart badges at warning or critical (20 yellow and 8 red in our check notes; an earlier note says 26, with 6 red).

We found five causes; only the third is a real fault.

First, two placement policies disagreed. Since 4 November the scheduler adds its own preferred affinity to every pod (weight 100 for the main pool, 50 for a second sub-pool, with the NodeAffinity score plugin weighted 3 against a default of 2). Twenty-three workloads keep their own storage-tier preference (weight 80 to 100) to stay near their data, but the scheduler's policy outranks it, so every reschedule put those pods on the main pool. The console judged placement only by the pod's own preference and painted 9 pods amber.

Second, planned reboots were graded as churn. Each reboot adds one restart per container to every pod on the node. Daemon-set pods crossed the console's warning bar (10 or more restarts in 7 days): 20 yellow badges, plus red ones on the rebooted host.

Third, and this is the real fault, the descheduler's return-to-preferred profile also judged only the pod's own preference. Its retained log shows the graph database pod evicted at 16:20:06, 16:30:04, 16:40:01, 16:50:14, 17:00:07 and 17:10:06 UTC on 6 November: six restarts in 50 minutes, because the scheduler put it back on the main pool each time. The board never showed it.

Fourth, cron job waves. Jobs run every 5 minutes, and about 10 pods start at :00 and :30. Seconds-old job pods are Pending or ContainerCreating, and the console counted each as unhealthy. At 10:30 UTC the board read 197 of 207 healthy with 10 Pending job pods counted against it. This is the likeliest source of the yellow our operator saw.

Fifth, failed vulnerability-scan pods. Two images the scanner could not pull produced failed scan pods that stayed amber for the 10-minute job TTL. In our configuration the per-annotation skip did not stop the scan jobs on scanner version 0.31.2; we did not investigate why, and the image-exclusion list worked.

The board did not lie about pod state; it was wrong about meaning. It is the inverse of the false-green class: the observer went amber on a healthy system while the one real fault stayed invisible.

Change	When (UTC)	Effect
Console mirrors the scheduler's two affinity terms; matching pods get a quiet info icon	commit at 09:49:55	off-preference pods 9 to 0
A restart is quiet when the sandbox was re-created after pod start, termination falls in the window around it and the pod is Ready (60 s, 600 s, 30 s); a crash after boot keeps normal grading	same commit	red restart badges 8 to 0; yellow 20 to 2
180 s startup grace: a young, never-restarted pod that is Pending, ContainerCreating, PodInitializing or not ready counts as starting	commit at 10:46:00	headline reads, for example, 197/207 healthy, 10 starting, and stays green
Descheduler selector excludes the graph database, the relational database and the cache workload from return-to-preferred	about 10:10	first run under it: 0 evictions
Scanner image-exclusion list for the two failing images	11:57:12	0 scan jobs and 0 errors in the 7 minutes observed after restart
Detector for drift between scheduler policy, console mirror and descheduler, on a 30-minute timer, with 4 alerts	afternoon	the pre-fix descheduler policy flags exactly the 3 databases

Table 8. Changes to the console, the descheduler and the scanner, with the observed effect.

The 2 remaining yellow badges were real crashes after the drain, cleared by rollouts. Verification came from our own check notes: a job wave at 12:30 UTC (59 samples, up to 20 pods starting, 0 unhealthy). These board counts come from observation during the fix, not from retained raw frames.

8. What These Numbers Will Not Carry

This is one cluster, one event and one day, with no controlled arm. We made two changes together (stop generating, delete existing), the database also shrank by compaction, and no arm changed only one thing, so we cannot say which part mattered. Host journals had rotated away, so we used the fleet's log store, whose counts match the counters to within one. Slow-apply logging records only requests over 100 ms. We did not examine slow reads before 7 November 11:40 or time reads of different sizes separately, and the post-change window has no pre-registered comparison. The scan-job observation is one scanner version, not reproduced in isolation.

Unexplained: member A had 3.7 times member B's count on the same hardware class, and the in-use size had been rising for a week (about 172 MB to 177 MB) while the reports' size was stable. We use only the 12 hours after the change, so we make no claim that the low residual rate persists.

What survives: the disks were not the cause (latency barely moved, no failed proposals, counts not ordered by disk class, and far fewer fsyncs over 128 ms than slow applies on every member); the reports held 78% of the in-use database; and slow applies fell 91% after we removed them, against a same-time-of-day control of 713 events. What does not survive: zero slow applies, a measured queueing mechanism, and any claim that the scanner caused most

slow applies, since only 13% of the lines were its reads.

The board-side results are weaker still: the counts come from our own check notes and a few browser checks, we did not re-derive 189 of 189 from a raw pod list, and the eviction loop is a single six-event episode. Each console number is a one-day before and after.

9. What We Changed in Practice

Our working rule is that scanner reports of this size do not belong in the consensus store; the scanner's alternative report store is the place to look. Table 8 lists the other changes we made.

10. Methods and Artefacts

We ran Prometheus queries for the etcd slow-apply counter, in-use database size, WAL fsync and backend commit histograms, failed proposals, and the API server's stored-object and list counters. We queried the log store for the "apply request took too long" warning and parsed duration, key, prefix and response size. The proximity test used a single-seed 200-shuffle baseline. We also read the Helm history, the scheduler configuration, the descheduler log and the console commit timestamps. All queries were capped at 23:59 UTC on 8 November. The cluster runs K3s with embedded etcd (Table 1) and the scanner as Helm chart 0.33.2 (application 0.31.2). The slow-apply lines (1,286 and 52 events) were queried from the retained log store with the filter on the warning text, and the metrics remain in the metrics store.

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-2026-069. The measurements were taken on 8 November 2026 from retained metrics and logs; the underlying metrics and logs are retained.