StreamKernel

95.5 Million Documents. 163K docs/sec. The MongoDB Write Ceiling.

MongoDB insert baseline · MONGO_INSERT plugin · Pure insertMany · No upsert · No embedding · No vector overhead

By Steven Lopez, StreamKernel · March 22, 2026 · 15:36–15:47 UTC

95,487,000	163K docs/s	256K docs/s	1.39ms	Zero
Documents Written	Avg Throughput	Peak Throughput	Avg MAX Latency	Data Loss

Hardware: Intel i9-8950HK · 6C/12T · 32GB RAM · JVM: 8GB heap · G1GC · 5 GC threads · MongoDB 7.0.28 · WiredTiger · Docker · local

ACT 1 OF 4 · THE NEW PANEL · MONGODB INSERT SINK

The Write Rate Is Now Live in Grafana.

The MONGO_INSERT plugin emits its own Prometheus metrics under the streamkernel.mongo.insert.sink.* namespace. A new Grafana panel — MongoDB Insert Sink — Write Rate — visualizes insert writes per second and flush rate in real time, alongside the existing MongoVectorSink panel. For the first time, both MongoDB write paths are visible on the same dashboard.

The chart ramps from zero at 11:35, climbs through 100K at 11:38, reaches **200K+ ops/sec by 11:40**, and sustains between 150K and 200K for the remainder of the run. The flush rate line (yellow) sits close to zero on the same scale — each flush call handles 500 documents (one pipeline batch), so the flush rate is approximately 1/500th of the write rate. Both lines are clean — no error events, no drops.

Image 1 — MongoDB Insert Sink — Write Rate panel (new). insert writes/s peaks above 200K, flush rate/s tracks at 1/500th. Zero error events throughout the run.

ACT 2 OF 4 · EXECUTIVE DASHBOARD

55.3K ops/sec Snapshot. 100% Integrity. P99: 0.002ms.

The executive summary captures a 30-second rate snapshot mid-run. PROC EPS, OUT EPS, and IN EPS all read 55.3K — the three values are identical, confirming no internal buffering gap between source admission, processing, and MongoDB delivery. Integrity is 100%, loss rate is zero.

The throughput chart shows the ramp characteristic of a cold JVM on a freshly rebooted machine: the first third averages 137K ops/sec as the JIT compiler warms up, the middle third peaks at 179K as the JIT reaches steady state, and the final third settles at 172K. This is the expected profile — not instability, but the JVM's tiered compilation working correctly. The cumulative total rises continuously from zero to 95M with no flat sections.

The latency panel shows P50 at 0.001ms, P95 at 0.001ms, P99 at 0.002ms, and **P99.9 at 0.016ms**. The MAX chart ceiling is 15ms with only three windows exceeding 10ms — all corresponding to G1GC young-generation collections. This is sub-millisecond per-record latency at 163K documents per second, delivered to a persistent document store.

INFLIGHT is 6K — well within the 18,000 ceiling. LOAD % is 100% — MongoDB is the constraint, not the pipeline orchestrator. StreamKernel is saturating WiredTiger's journal write budget.

Image 2 — Executive: 55.3K ops/s snapshot (30s rate). 100% integrity, 0 loss. P50/P95: 0.001ms. P99: 0.002ms. P99.9: 0.016ms. INFLIGHT 6K. LOAD 100%.

ACT 3 OF 4 · SECURITY, INTEGRITY & JVM

Zero Errors. Zero Loss. 91 GC Pauses. Zero Full GC.

The security panel shows auth decisions at 55.3K/sec with zero denials and zero auth errors. The Security Events chart traces the full run arc: a clean ramp from 30K at 11:35 to a sustained plateau between 150K and 200K from 11:40 onward, ending at a terminal drop at 11:47 as the run completes. The admitted/s chart in Source Starvation Diagnosis mirrors this shape exactly — the source and the security layer are moving in lockstep.

The Source vs Pipeline Totals chart reaches 95M at completion with a smooth linear slope. All three lines — processed, out, in — overlap as a single trace. The error rate panels and record loss panels are flat at zero for the entire run duration.

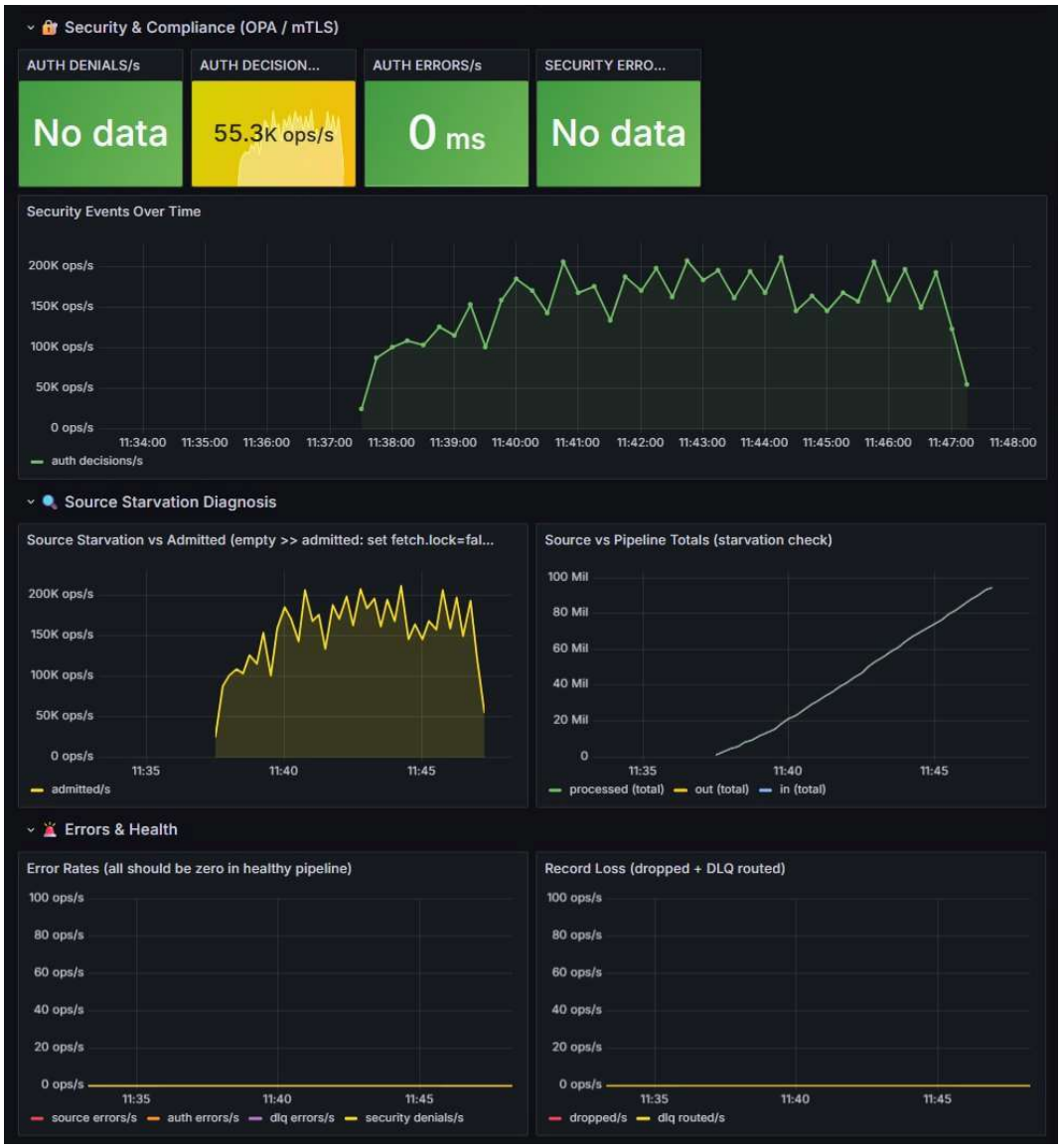

Image 3 — Auth decisions: 55.3K/sec. Zero denials, zero errors. Source vs Pipeline Totals: single overlapping line through 95M. Error rates and record loss: flat zero throughout.

The pipeline integrity gauges confirm 100%, 0% drop rate, 0% DLQ rate. The JVM heap cycles cleanly between 2GiB and 7.7GiB — G1 collecting on schedule with no accumulation toward the 8GiB ceiling. The thread count stabilizes at 27 live threads after JVM warmup.

GC log analysis: 91 pause events, average 23.1ms, maximum 89.0ms, **zero Full GC, zero GCLocker events**. The 18 windows with pauses above 50ms correspond to the larger mixed collections that G1 runs when reclaiming old-generation regions. None produced observable throughput drops — the pipeline sustained 150K+ docs/sec through every GC event.

Image 4 — Integrity: 100%. Drop: 0%. DLQ: 0%. Heap: 7.36GiB at snapshot, max 8GiB. GC: 0.0375 ops/s, 337µs (Grafana 30s rate). GC log: 91 pauses, 23.1ms avg, 89.0ms max. Zero Full GC. 27 threads.

ACT 4 OF 4 · DEMO SCRIPTS · BEFORE & AFTER

Verified Clean Start. 95.5M Documents Confirmed in MongoDB.

The demo_before_mongo_insert.ps1 script ran before the benchmark to confirm a clean collection state. It found 71,697,500 documents remaining from a previous run and cleared them automatically, confirmed zero documents, showed the collection list with sk_insert_baseline marked as the target, verified that only the _id index was present (no secondary indexes, no vector index), and reported the storage engine as WiredTiger / MongoDB 7.0.28.

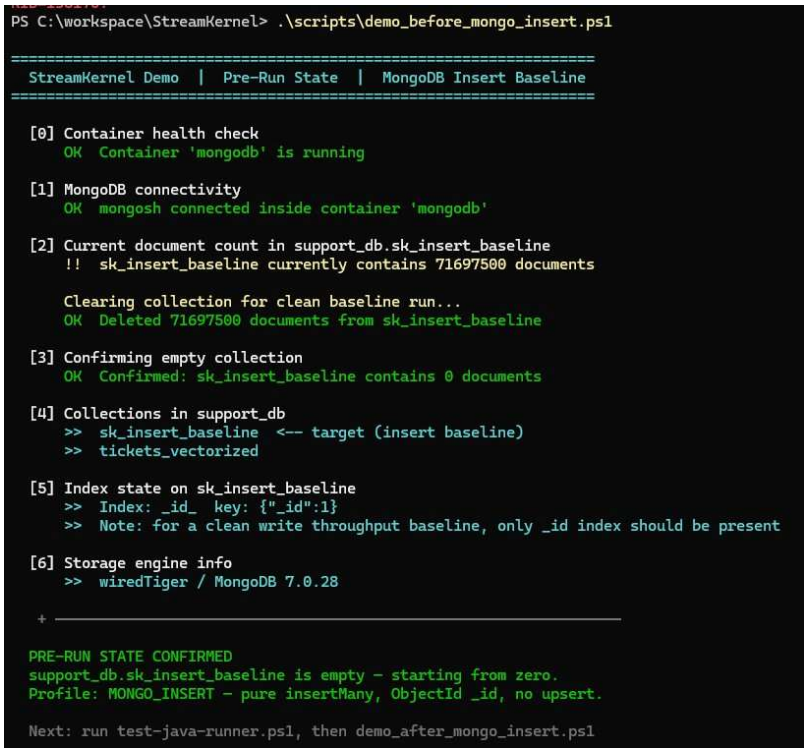

Image 5 — demo_before_mongo_insert.ps1: cleared 71.7M documents from prior run, confirmed 0 documents, showed _id-only index, WiredTiger / MongoDB 7.0.28.

The demo_after_mongo_insert.ps1 script ran immediately after completion. The results box shows 95,487,000 documents written at 159,145 docs/sec average over 10 minutes. Two sample documents show the exact schema written by MONGO_INSERT: ObjectId _id, key string, bytes integer (512), empty headers object, and a ts object. The field inventory confirms no vector or embedding field is present — this is the pure write baseline.

The storage stats panel shows 6,465.51 MB data size, 1,002.92 MB on disk after WiredTiger compression — a **6.45x compression ratio**. Average document size is 71 bytes. The _id index is the only index present. The benchmark context table at the bottom of the after-script confirms the position of this run in the cost stack.

Image 6 — demo_after_mongo_insert.ps1: 95,487,000 documents, 159,145 docs/sec. Schema: ObjectId _id, key, bytes=512, headers, ts. Storage: 6,465MB data / 1,003MB on disk (6.45x compression). _id index only.

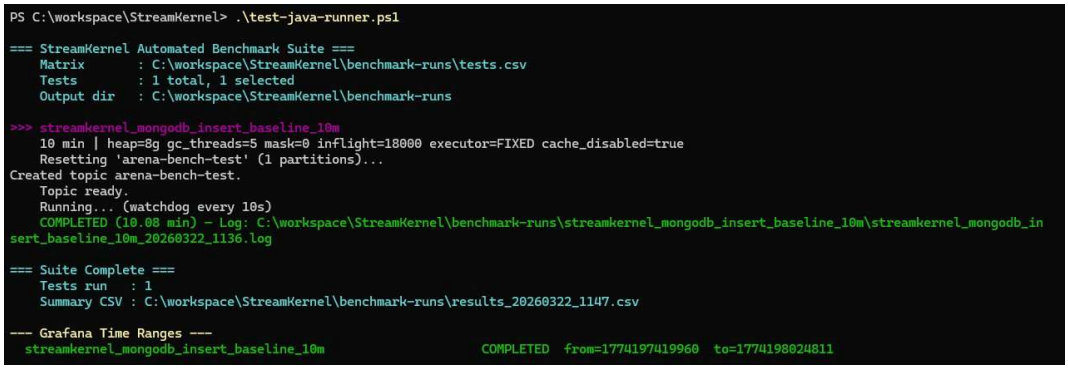

Image 7 — test-java-runner.ps1: COMPLETED (10.08 min). Grafana range: from=1774197419960 to=1774198024811.

SUMMARY · WHAT THIS NUMBER MEANS

163K docs/sec. The Floor Before Inference.

95,487,000 documents. 163,007 docs/sec average. 256,413 docs/sec peak. Zero errors. Zero data loss. 100% integrity. This is StreamKernel's raw MongoDB write throughput ceiling — the WiredTiger journal write budget on Docker with a single unindexed collection, measured from a single JVM on a laptop, over 10 continuous minutes.

Why this number matters

The next profile — streamkernel_mongodb-vector — adds ONNX inference (MiniLM-L6-v2, 384 dimensions), float[] to List<Double> boxing, and a ReplaceOneModel upsert per record. The difference between that number and 163K is the pure cost of in-process AI inference at stream speed: no model server, no network hop, no serialization overhead. That delta is the answer to the question StreamKernel was built to answer.

Profile	Avg docs/s	Documents	Plugin	What it measures
MongoDB Insert (this run)	**163,007**	**95,487,000**	MONGO_INSERT	Pure insertMany write ceiling
MongoDB Vector (pending)	TBD	TBD	MONGO_VECTOR	With ONNX inference + upsert