

Building Observability for Microsecond Systems

Something already decided what it costs you

Lyubomir Bozhinov · Founder @ Barnett Studios

KCD Sofia · 29 September 2026

+0.10 ms

p95 latency added by an OpenTelemetry Java agent.

1.96 ms → 2.06 ms.

Is that a lot?

Source: Elastic, EDOT Java agent overhead benchmark — also +22% allocated memory, +16% GC pause time, +0.43pp CPU

+0.10 ms, against three budgets

Latency budget	What +0.10 ms costs you
100 ms · a normal web request	0.1% · free, ship it
1 ms · order processing	10% of your entire budget
50 µs · a matching engine	200% · the observer is slower than the observed

The number didn't change. The budget did.

Where it goes: 3–5 allocations per span before any of your work happens — and allocation rate is GC frequency is pause time (+22% memory, +16% GC pause, same benchmark). **And it is not fixed:** OTel's own published JMH dashboard (`SpanRecordBenchmark.threads1` , 2026-08-24) gives a small span at **317 ns** and a large one at **20,860 ns** — **the cost scales ~60x with what you hang off the span.** Sources: Elastic EDOT benchmark · opentelemetry-java `SdkSpanBuilder`

Observability isn't free.

It's a **latency decision you've already made**
— whether or not you measured it.

Something made that decision. Let's find out what.

PART ONE

Your framework decided

Logback `AsyncAppender`, out of the box

Property	Default	Consequence
<code>queueSize</code>	256	one burst of debug logging fills it
<code>discardingThreshold</code>	20% remaining	silently drops TRACE/DEBUG/INFO
<code>neverBlock</code>	false	blocks the caller

"**If false (the default)** the appender will block on appending to a full queue rather than losing the message."

Your request thread stops. During an incident, when you are logging most.

Source: Logback manual, "Async and sifting" — logback.qos.ch/manual/appenders-async-sift.html · **This is Java.** Your log shipper has a buffer, your sidecar has a queue, your ingress has a timeout. Somebody picked those numbers, and it wasn't you.

PART TWO

Your substrate decided

Three budgets, three instruments

Tier	Budget	Instrument
Hot · order entry to execution	microseconds	counters · no strings, no serialisation, no allocation
Warm · risk, health, consensus	milliseconds	structured logs that drop before they block
Cold · compliance, audit, analytics	"before the auditor asks"	everything , batched, retained for years

Losing a log line is cheaper than adding latency to a risk check.

What that bottom row costs you to **keep** — Juan Herrera Utande covered that at 15:20, two hours before this talk. This one is about what it costs you to **emit**.

We didn't invent this. Aeron did.

"Aeron tracks a lot of its internal state as AtomicCounters in a memory mapped file. This file can be read by **any external process with no significant impact on performance.**"

"Rather than take the approach of using log files, which can fill disks due to chronic issues, Aeron **records errors to a section of its CnC file as distinct errors with a count of observations** plus time of first and last observation."

Write: an increment into a mapped page. **Read:** a different process, mapping the same file. The producer does nothing extra to be observed.

Each counter slot is **128 bytes — 32 of data, 96 of padding**, so two counters can never share a cache line. **Sources:** Aeron wiki, "Monitoring and Debugging" (both quotes verbatim) · agrona `CountersReader` , `COUNTER_LENGTH = CACHE_LINE_LENGTH * 2`

The hot-path counter

```
long id = getOrCreateMetricId(name);  
long current = counters.get(id);  
counters.put(id, current + 1);    // no atomic. no CAS. no lock.
```

Not lock-free. **Lock-absent**. Aeron Cluster runs each service **on one thread**.

The cheapest concurrency primitive is the one you deleted by choosing an execution model that didn't need it.

One increment per message. Not free to export. Ours is an on-heap map, so the snapshot walks every series.

What we did not inherit is Aeron's free read: `cnc.dat` is mmap'd, so another process reads it for nothing. Most teams reach for `AtomicLong` and never ask why they have a second writer. (Our own documentation still says this is a CAS. It isn't.)

Cardinality is an availability problem

A label is not free. Untagged counter: single-digit ns. Add one dimension: ~10× that.

And you all know what unbounded label cardinality does to Prometheus. Here it's worse:

Cap breached. Exception out of the message handler. Wire that handler to terminate the node.

Not "inflates your bill." Not "degrades the dashboard." Kills the process.

So: never `"oms.order." + orderId`. Ever.

Aeron's default is to log and continue; killing the node is a choice — a replica that threw mid-message is no longer deterministic, and a non-deterministic replica is worse than a dead one. Fast-fail beats eating hundreds of megabytes of histogram until you OOM at 3am. Mitigated by a usage gauge alerted well below the cap.

The other cap fails the other way

Tag cache full. Log a warning. `return EMPTY`.

The metric keeps recording. **It silently stops having dimensions.**

Your scrape succeeds. Your dashboard stays green. It just quietly becomes less useful.

Nothing breaks. Nobody notices.

Two caps, one class, two philosophies. The memory cap **fails loud and kills the node**. The cardinality cap **fails soft and drops your labels**. I'm not sure the asymmetry was deliberate — fair thing to ask me afterwards.

PART THREE

The decision we did make

First, the honest answer

Chronicle-Logger. Sub-microsecond Java logger. SLF4J / Log4j / Logback bindings. Binary records into a memory-mapped Chronicle Queue. Apache 2.0.
Actively maintained. 2026.0 released January 2026, commits through the summer.

It's good. I'm not going to pretend otherwise.

My friend and colleague Stefan wanted to build it himself and get his hands dirty with the ring buffer.

And so did I.

Then, the real answer

	ns/op
counter increment, no tags	single digits
HdrHistogram record	3–6
counter increment, tagged	~50
Chronicle-Logger, simple message	784.761
Log4J2 file appender	2,427.177
OTel span record, small → large	317 → 20,860
Logback default async, queue full	blocks

Two orders of magnitude more than the primitive we needed. And that's about as fast as logging gets.

Source: Chronicle-Logger README, their own published benchmark. Not a criticism of Chronicle — a statement about the category. **The fastest possible log line is still a log line.**

Run it through the gate

- 1. Who maintains this?** Chronicle Software. Single vendor, Apache-2.0 core, commercial Enterprise tier for the features we'd have wanted. Not a red flag. Not nothing.
- 2. What does it drag in?** A second memory-mapped persistence subsystem on a box already running Aeron's log, archive and counters file.
- 3. Is there a smaller alternative? Yes. Zero.** The hot path doesn't log.

And binary logs need Chronicle's own tooling to read — also a cost at 3am.

The warm path **does** log. With Logback. At defaults.

There, Chronicle would have been a fine call.

But we never evaluated it. Logback was good enough.

PART FOUR

**Something decided what
"healthy" means**

The dashboard was green the entire time

A cluster node fails over. **It works exactly as designed.**

Traffic reroutes. No downtime. Every health check passes.

The failover opens a brief window where incoming data is **silently dropped.**

The system is healthy. The pipeline is broken.

Nobody finds out until someone queries the output weeks later and finds a hole.

If your observability answers "is the infrastructure healthy?" but not "is the product working?" — you're monitoring the engine without watching the road.

for: 0s

```
– alert: RiskLimitBreach  
  expr: increase(risk_limit_breaches_total[1m]) > 0  
  for: 0s          # ← not 30s. not 1m. instant.  
  labels: { severity: critical }
```

A risk breach is a **compliance event**. A fire alarm. Fire alarms don't wait to see whether the fire is sustained.

Same treatment: leader election · p95 > 1 ms warning, 5 ms critical · throughput below baseline.

Yours is checkout completion. Or payment success. The metric changes. The principle doesn't.

Note `increase()`, not a bare `> 0` — on a cumulative counter a bare comparison latches forever after the first breach. Whether you want it to latch is a real decision. Make it on purpose.

Alert on the absence, not the event

A recurring cluster timer does work, then **reschedules itself**.

Failure mode: back-pressure drops the re-arm. The chain breaks. It never fires again.

It throws nothing. Logs nothing. Errors nothing. The metrics just stop changing.

```
# the easy case – the pod is gone
absent_over_time(timers_last_scheduled_seconds[10m])

# the one that gets you – the pod is alive, the timer is frozen
(time() - timers_last_scheduled_seconds) > 2 * slowest_interval
```

absent() catches the series that disappeared. Nothing catches the live pod whose metric stopped moving.

Every cron, every queue consumer, every heartbeat, every nightly batch. How many alert when they **fail** — and how many alert when they simply, silently, **stop**?

Count your discards

Ours is configured with a fixed range. Resizing costs the speed you came for. An out-of-range value throws on the cluster thread and kills the node.

So values get **clamped**. And every clamp is **counted**: by direction, cumulative, never reset:

```
<your_metric>_clamped_total          # lost the tail → capacity review  
<your_metric>_clamped_negative_total # should never happen → page on it
```

If your instrument has to discard data to stay alive, count the discards.

Generalises well past histograms: every exporter that drops on a full queue, every sampler, every rate limiter is throwing your data away — and most don't tell you how often. A silent lossy path is worse than no path, because you'll trust the number it hands you. HdrHistogram records in **as little as** 3–6 ns, constant memory, no allocation — on circa-2012 Intel silicon (project README).

PART FIVE

When nothing decides

What's actually in our warm-path config

The launcher's async appender is beautifully configured:

`queueSize` 10× the default · `discardingThreshold` 0 · `neverBlock` true .

Three deliberate non-defaults. Somebody thought about this.

And the same file wires `ASYNC` and a synchronous `FileAppender` with `immediateFlush=true` into the same root logger.

Every INFO line: a synchronous, flushed disk write, on the calling thread.

Most of the other configs don't have an async appender at all.

Our documentation says the warm path emits structured JSON.

One config does.

I wrote that documentation. I hadn't read the config. [This is the file in the repository](#) — I haven't checked what the deployed image resolves to, so I'm not claiming it. The drift between a repo's own docs and its own config is the defect; it needs no production claim at all.

The hot path is disciplined because Aeron
forces it to be.

The warm path drifted to defaults because
nothing did.

Same engineers. The difference was never discipline — it was
what was doing the deciding.

Architectural discipline that nothing enforces
decays.

CLOSE

Choose what decides

You can't make every decision. Choose what makes them.

1. Measure what your observability costs: your service, agent on vs off. If you don't know the number, you didn't make the decision. You inherited it.
2. Open your logging config. Look at `neverBlock`. Now, on your phone, is fine.
3. Stop instrumenting every tier the same way. You have a hot path, even if it's a checkout.
4. Find one thing that fails by **stopping** rather than erroring. Write the alert that notices its silence.
5. **Check your documentation against your config.** The only item here I can promise will find something.

Every one of these is a decision something already made for you. Go and find out what.

"The recording operation cannot be allowed to be a cause of the very outliers it is trying to measure."

Gil Tene, "WriterReaderPhaser: A story about a new (?) synchronization primitive", 2014 — author of HdrHistogram. That was 2014. Logback still blocks by default.

Something already decided what your
observability costs you.

Go and find out what.

And if the answer is nothing — that's the tier that's rotting.

Thank you



LinkedIn

[in/lyubomir-bozhinov](#)



GitHub

[@lyubomir-bozhinov](#)



Blog

[barnett-studios.com](#)