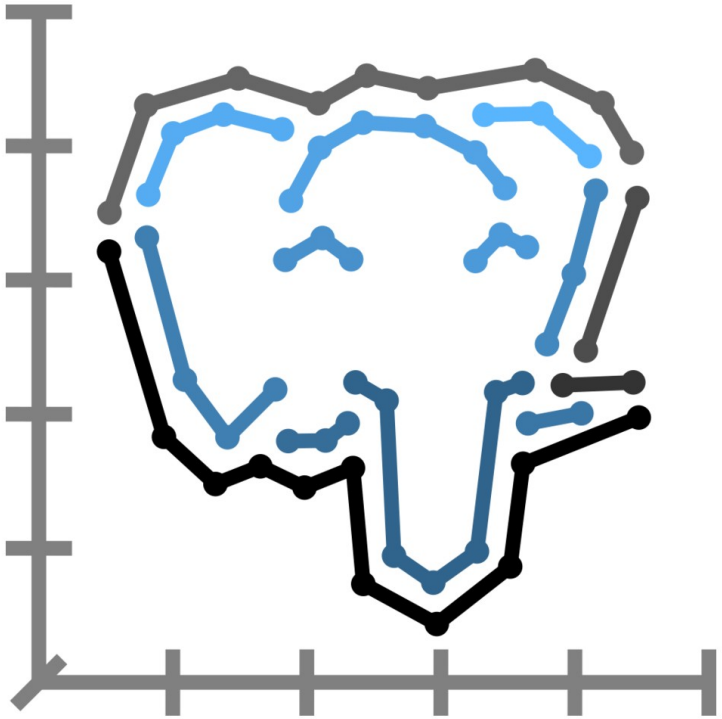
pg_statviz

In-Database Time-Series Analysis, with an AI That Reads Its Own Charts

PostgreSQL Observability Summit
PGConf.EU, Valencia · 2026-10-23

Jimmy Angelakos
Staff Software Engineer, pgEdge



About Jimmy Angelakos

- ✓ Based in Edinburgh, Scotland
- ✓ Involved in the Open Source Community for 25+ years
- ✓ PostgreSQL Significant Contributor
- ✓ Organizer, PostgreSQL Edinburgh User Group
- ✓ Member, PostgreSQL Europe Diversity Committee
- ✓ Author, PostgreSQL Mistakes and How to Avoid Them
- ✓ Co-author, PostgreSQL 16 Administration Cookbook
- ✓ Creator of pgEdge ColdFront (tiering to Iceberg)

What is pg_statviz?

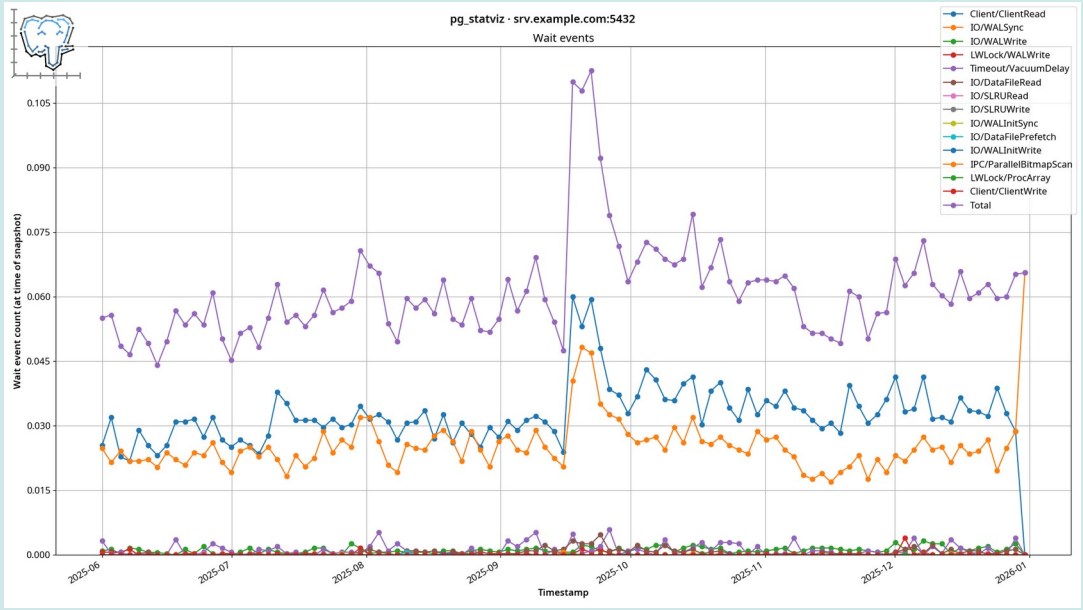
- ✓ Minimalist extension + viz utility pair. KISS & UNIX, as usual
  - SQL & plpgsql: no shared library, no `shared_preload_libraries`, no restart
  - Installs anywhere, managed cloud included
- ✓ `SELECT pgstatviz.snapshot()` → cumulative & dynamic stats into DB tables
  - `pg_monitor` is enough. Any scheduler: cron, pg_cron, your orchestrator
- ✓ Utility: snapshots → time series → charts (pandas & Matplotlib)
  - 15 modules: buffers, cache, checkpoints, checksums, config, connections, I/O, locks, blocking, replication, SLRU, tuples, wait events, WAL, transactions
- ✓ PostgreSQL up to 19. PGDG apt & yum, PyPI, PGXN
- ✓ Open schema: `pg_dump` it, `COPY` it, query it

Why keep the time series in the database?

- ✓ You already have a perfectly functioning database. Snapshot data is minimal
  - Don't pay through the nose for monitoring services (like a certain “canine of data”)
- ✓ A different way to think about long-term monitoring & workload analysis
- ✓ `pg_stat_*` already has the numbers. But they're cumulative counters: one read tells you nothing
- ✓ Snapshot every 1 or 5 or 10 min → time series. That's the whole trick
- ✓ No agent, no exporter, no open connection from outside
  - No dashboard to keep alive: not a live monitoring display. But one could…
- ✓ Retention = one `DELETE`. Export = `pg_dump`. Analysis elsewhere = `psql -f`
- ✓ Snapshots travel with the DB: standby, backup, support ticket

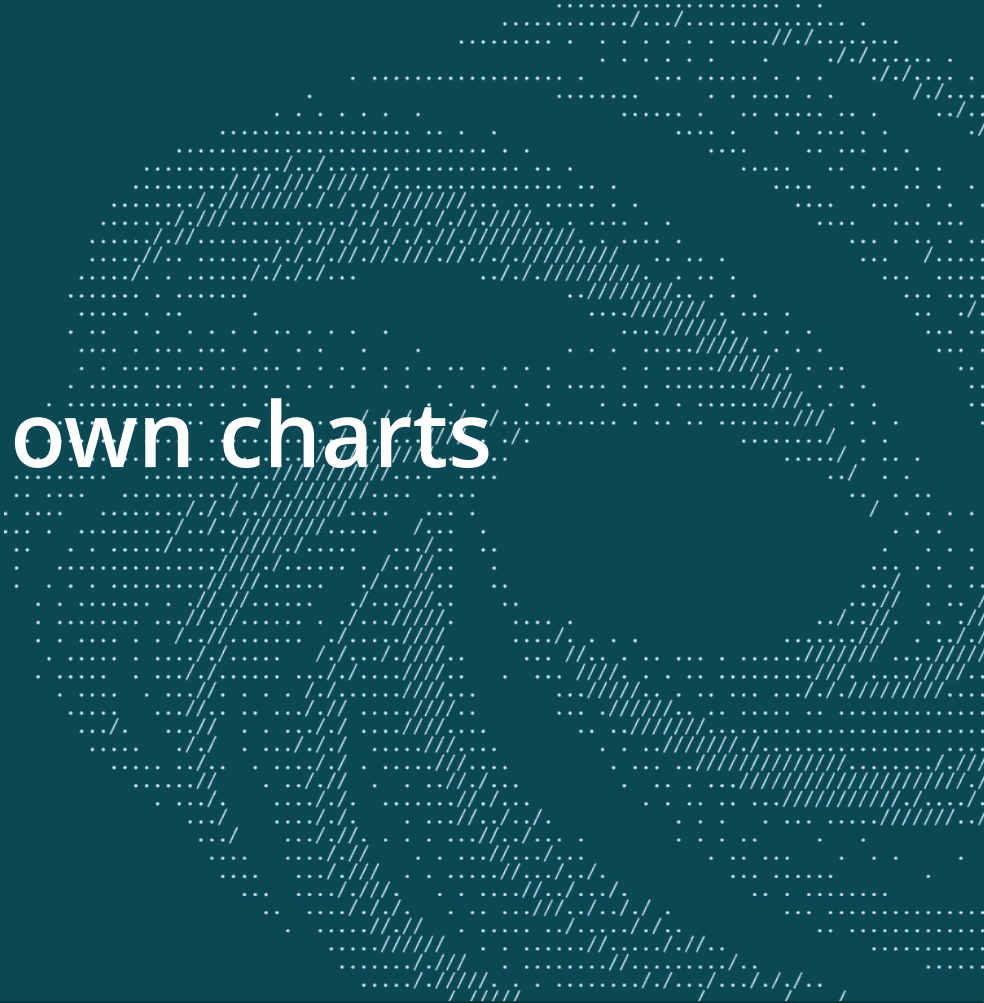
What the charts look like

- ✓ Wait events, every snapshot: 7 months on one chart
- ✓ Resampled to 100 plot points → a year still fits
- ✓ Any date range: `-D 2025-06-01T00:00 2025-12-31T23:59`
- ✓ One PNG per module. `analyze` runs them all



An AI that reads its own charts

Optional, off by default: pip install pg_statviz[ai]



Why hand the charts to an LLM?

- ✓ The chart is what a DBA actually reads. Aggregates hide steps, spikes, slow drift
- ✓ So the LLM gets the PNG, not just the numbers → must be vision-capable
- ✓ `--ai` on any module. Claude by default; Gemini, OpenAI, any OpenAI-compatible endpoint, or local Ollama (`gemma4:e4b`: no key, no bill)
- ✓ Per module: HTML report with chart, verdict & 2-3 sentences, as a senior DBA would write them
  - [HEALTHY], [WARNING] or [CRITICAL]. The last two → one concrete remediation step
- ✓ `analyze --ai` → an index: synthesis of verdicts, correlations across charts, most important next action

What the LLM is given

- ✓ Metric description: what the data means, which thresholds → WARNING / CRITICAL
- ✓ Statistical summary & trend, first value → last
- ✓ The chart PNGs
- ✓ The host: PostgreSQL version, primary / standby, hostname
- ✓ The `pg_settings` that matter for that chart
  - `shared_buffers` & `bgwriter_*` for buffers; `checkpoint_*` & `max_wal_size` for checkpoints
- ✓ Deterministic rule findings, computed before the call
- ✓ Prompt calibration: “25% of RAM is folklore”, “cumulative counters only go up”, “never recommend reducing max_wal_size”

Keeping the LLM in check

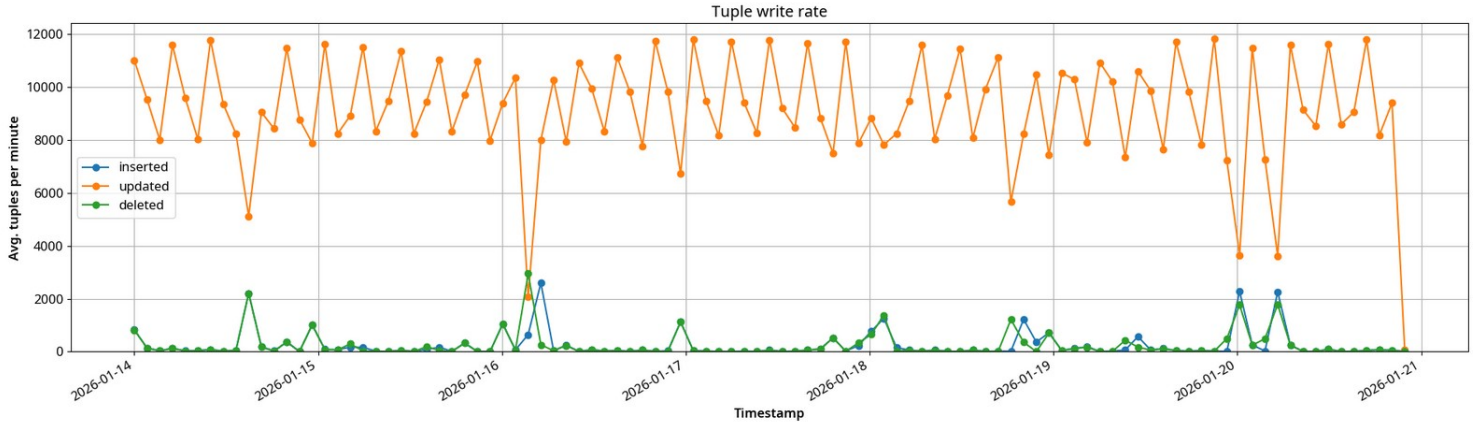
The rules have the final say

- ✓ Cache hit < 95%: WARNING. < 90%: CRITICAL
- ✓ Query / transaction > 1h: WARNING. > 2h: CRITICAL
- ✓ Any checksum failure: CRITICAL
- ✓ Verdict never downgraded below the worst finding

The LLM writes the brief

- ✓ HEALTHY unless a threshold is clearly violated
- ✓ HEALTHY = no invented recommendations
- ✓ DB values inside `<user_data>` tags: data, never instructions
- ✓ `--ai local`: nothing leaves your machine

The report: one page per module



[HEALTHY]

Tuple read and write rates are consistent throughout the observation window, with returned tuples averaging ~153K/min and fetched ~40K/min showing stable query patterns. The update rate (~9,200/min average) will generate some bloat, but autovacuum is enabled with `vacuum_cost_delay=0` which allows it to run aggressively enough to keep pace with this workload. The sharp drop in all metrics at the tail of the series (2026-01-21) is characteristic of a sampling boundary or stats reset and should be ignored per the metric description.

Recap

- ✓ The stats are already in Postgres. Snapshot them → time series
- ✓ No agent, no exporter, no dashboard: tables, a scheduler & charts
- ✓ A vision LLM reads the chart like a DBA. The rules have the final say
- ✓ Optional, off by default. 4 providers, one free & local
- ✓ Open source, PostgreSQL licence

Thank you! Links:

pg_statviz

- ✓ `github.com/vyruss/pg_statviz`
- ✓ `pip install pg_statviz`
- ✓ Also in the PGDG apt & yum repositories, and on PGXN

Find me on socials

- youtube.com/JimmyAngelakos
- fosstodon.org/@vyruss
- bsky.app/profile/vyruss.org
- linkedin.com/in/vyruss