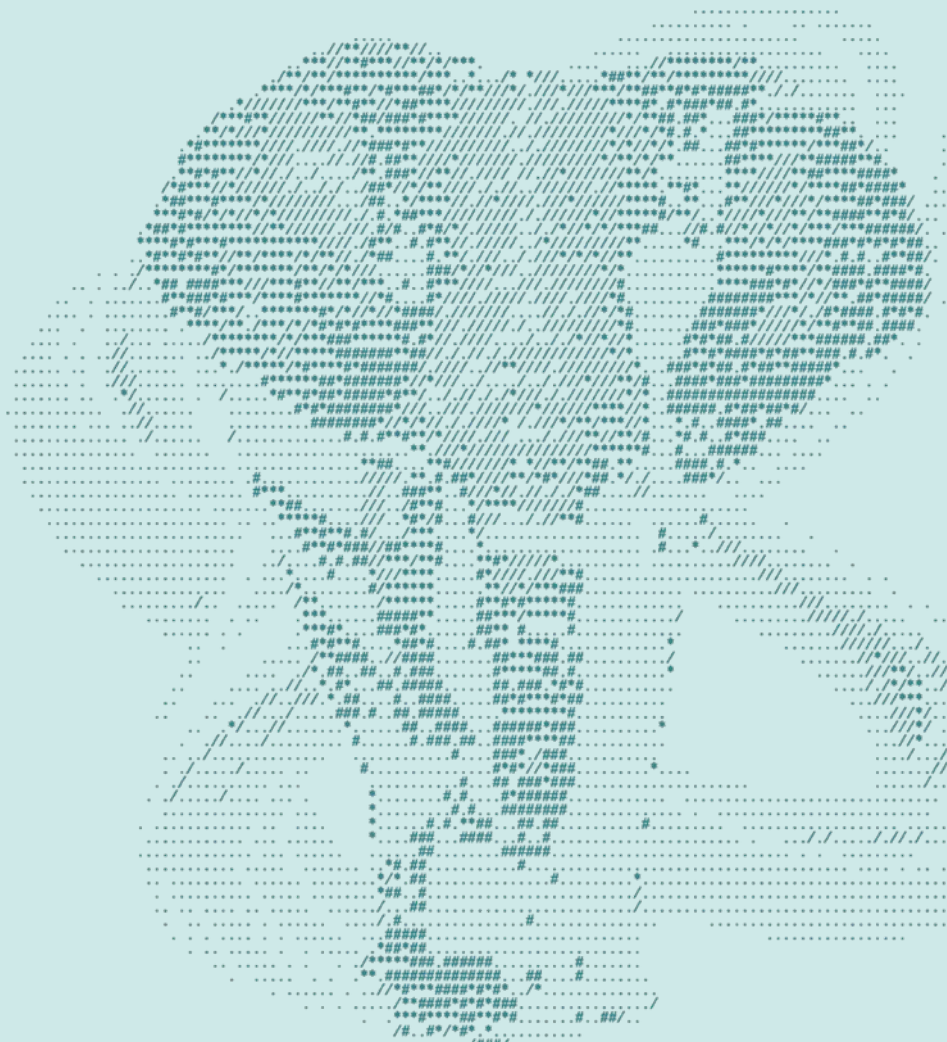
pgEdge

radar + radar-analyst: Agentless Diagnostic Snapshots for PostgreSQL



PGConf.EU 2026 · Valencia · 2026-10-22

Jimmy Angelakos
Staff Software Engineer, pgEdge

What is this talk about?

1. Why another diagnostic tool?
2. radar: the collector
3. What it collects (and what it doesn't)
4. radar-analyst: the rule pack
5. What a snapshot can catch
6. The verdict: findings first, then the LLM

© 2026 pgEdge, Inc.

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)
- ✓ Creator of pg_statviz (Postgres stats visualization)

© 2026 pgEdge, Inc.

Why another diagnostic tool?

Agents (before the incident)

- Datadog, Prometheus exporters, telegraf, sidecars, vendor agents…
- Each one: its own attack surface, dependencies, update schedule
- Has to be running before things go wrong
- You see the metrics somebody decided to export

Logs (after the incident)

- pgBadger over the Postgres log
- Logs show what your queries did
  - Not server settings, `pg_hba.conf`, replication, statistics
  - Nor lower-level: kernel, memory, I/O configuration
  - Not even the queries with `log_min_duration_statement=off`

© 2026 pgEdge, Inc.

What does support actually ask for?

- “Can you send me your `postgresql.conf`?”
- “…and `pg_hba.conf`? And `pg_settings`?”
- “…and `pg_stat_replication`, `pg_replication_slots`, `pg_stat_archiver`?”
- “…and dead tuple counts, bloat estimate, indexes?”
- “…and `free -m`, `dmesg`, `iostat`, transparent hugepages, cgroup limits?”
- 12 emails later: a screenshot of psql (missing the one column you needed)
- So I wrote a tool that collects all of it into one zipfile
  - And another that helps you understand it

© 2026 pgEdge, Inc.

01
radar: the collector

A static Go binary that writes one zip

© 2026 pgEdge, Inc.

Introducing radar

- ✓ Agentless: no daemon, no scheduled job. Run it to get a zipfile
- ✓ Zero dependencies: one static Go binary, nothing to install
  - No Python, no systemd unit, no agent that “phones home” or needs regular patching
- ✓ Captures a PostgreSQL + system snapshot in seconds
- ✓ Linux and macOS, bare metal, VMs and containers
- ✓ PG 12 to 19, collectors added as PG releases add views
  - `pg_stat_io` on 16+, `pg_stat_checkpointer` on 17+, `pg_stat_lock` on 19+
- ✓ Output: `radar-<host>-<timestamp>.zip`. Send it to support

© 2026 pgEdge, Inc.

Design tenets

- ✓ KISS, as usual
- ✓ Nothing stays resident: run, write zipfile, exit
- ✓ Metadata only: catalogs, statistics, configuration
- ✓ Soft failure: missing command or permission denied → noted, not fatal
  - root + superuser gets everything, root + `pg_monitor` gets nearly everything
- ✓ Every collector is skippable: `-exclude bloat,stat_ssl`
- ✓ Heavy ones are opt-in: `-include pgstattuple`
- ✓ Streams into the zipfile, nothing buffered in memory
- ✓ One connection per database, reused for all radar queries

© 2026 pgEdge, Inc.

What does radar collect?

Host & OS

- ✓ CPU, memory, NUMA, THP, swappiness, overcommit
- ✓ cgroup v1/v2 limits: host has 64 GB RAM but does the container?
- ✓ Block devices, mounts, I/O schedulers, `iostat`
- ✓ CPU, memory, I/O pressure, `dmesg`, `sysctl`, load average
- ✓ Packages, systemd, SELinux/FIPS, tuned profile, cloud identity

PostgreSQL

- ✓ Settings, HBA, roles, databases, extensions, control file
- ✓ Activity, locks, `pg_stat_*`, `pg_stat_statements`
- ✓ Replication: slots, standbys, subscriptions
- ✓ Per database: tables, indexes, sequences, bloat
- ✓ pg_statviz, Spock, PgBouncer if present

© 2026 pgEdge, Inc.

What radar does NOT collect

- ✓ Row data: never. No query results, no table contents
- ✓ Passwords
  - `pgbouncer.ini`: only the `[pgbouncer]` section. `[databases]` has `password=` in it, so no
  - `userlist.txt`: recorded as “exists”, never read
  - Spock `node_interface` skipped entirely: `if_dsn` holds the node password
- ✓ Log contents: file names and sizes only
- ✓ Spock conflict log: counters and error messages, not the row images
- ✓ `pg_stat_statements`: normalised query text, the constants are already gone
- ✓ What's left may still be sensitive: object names, settings, hostnames
  - Check the zipfile before you send it

© 2026 pgEdge, Inc.

How do I run it?

- ✓ Root + superuser gets everything
- ✓ `pg_monitor` loses `pg_hba_file_rules` + 2 views, still completes
- ✓ `PG*` env variables, client certs and Kerberos work
- ✓ Runs in seconds, zip is ~1 MB

terminal

```
$ ./radar -d mydb -U radaruser
✓ Archive created: radar-db1-20260115-133700.zip (1.2 MB)

$ ./radar --skip-postgres             # host only
$ ./radar -d mydb --skip-system        # Postgres only
$ ./radar -d mydb -exclude bloat,stat_ssl
$ ./radar -d mydb -include pgstattuple

$ ./radar -h db1 -d mydb -U radaruser \
    --sslmode verify-full --sslcert client.crt \
    --sslkey client.key --sslrootcert ca.crt
```


© 2026 pgEdge, Inc.

What's in the zipfile?

- ✓ `.tsv`: query results, with headers
- ✓ `.out`: command output, stdout and stderr together
- ✓ `.conf`: configuration files, verbatim
- ✓ Optional directories only if X is installed
- ✓ `DATA.md` in the repo explains every file

```
unzip -l radar-db1-20260115-133700.zip

radar-db1-20260115-133700.zip
├── system/
│   ├── lsblk.out  df.out  mount.out  dmesg.out …
│   └── proc/  meminfo.out  cpuinfo.out  pressure/ …
├── postgresql/
│   ├── version.tsv  configuration.tsv  stat_activity.tsv …
│   └── postgresql.conf  pg_hba.conf  postgresql.auto.conf
├── databases/
│   └── mydb/  tables.tsv  indexes.tsv  sequences.tsv …
├── pg_statviz/mydb/        (if installed)
├── spock/mydb/             (if installed)
├── pgbouncer/              (if installed)
└── radar.out               version, host, timestamp
```


© 2026 pgEdge, Inc.

Obligatory numbers slide

3,225

lines of Go

You can read all of it in one afternoon

12 → 19

PostgreSQL versions

Collectors added as the catalog grows

246

collectors

Linux, macOS and PostgreSQL

© 2026 pgEdge, Inc.

02
radar-analyst: the rule pack

What to do with the zipfile

© 2026 pgEdge, Inc.

Why an analyst?

- ✓ 150 files collected. Who has the time to read them?
- ✓ Experienced support staff opens the same 10-20 first
  - THP? Swap? `trust` in `pg_hba`? Dead rows? Repslots?
- ✓ Those empirical checks are the rule pack
- ✓ The hard part is what they mean and what to fix first
- ✓ radar-analyst decides deterministically, and asks LLM for synthesis
- ✓ Never touches the host: works from the zipfile, needs no credentials

© 2026 pgEdge, Inc.

Radar Analyst: the basic idea

- ✓ Upload the zip: web console or JSON API
- ✓ Parses every entry into records
- ✓ Runs deterministic rules across five categories
  - Host & OS, Postgres configuration, workload, internals & I/O, replication
- ✓ Plus per-database analysis for every user database
- ✓ Every finding: a fact + a fix
  - “198 recent WAL archive failures” → `archive_command`, the destination, the disk
  - “`oc_filecache` has 56% dead rows” → that table's autovacuum settings, then `VACUUM`
- ✓ A brief per category, written by the model of your choice (or none)
- ✓ Two containers (analyst + its own Postgres): `docker compose up -d`

© 2026 pgEdge, Inc.

What does a rule look like?

- ✓ One function per check, registered to a category
- ✓ Thresholds are defined in the code
- ✓ A rule that errors out is logged & skipped, the rest still run
- ✓ 81 rules so far, in 10 modules

rules/pg_config.py

```
@register("PostgreSQL Configuration")
def fsync_off(parsed: dict[str, Any]) -> list[Finding]:
    """Critical if fsync is disabled: durability is gone."""
    s = _off_setting(parsed, "fsync")
    if s is None:
        return []
    return [
        Finding(
            rule_id="pg.config.fsync_off",
            severity="critical",
            title="fsync is disabled",
            detail=(
                f"fsync = {s.setting}. Data durability is "
                "NOT guaranteed across crashes. Set fsync = on "
                "unless this is a disposable instance."
            ),
        )
    ]
```


© 2026 pgEdge, Inc.

What does it catch?

- ✓ Host & OS: OOM killer fired, THP on `always`, swap, cgroup limit, disk ≥ 80% full
- ✓ Configuration: `fsync` off, `trust` in a network HBA rule, `ALTER SYSTEM` drift, `shared_buffers` off by a mile
- ✓ Workload: idle in transaction, blocking lock chains, connections near `max_connections`
- ✓ Replication: inactive slot holding GiBs of WAL, standby gone or lagging, subscription not running
- ✓ Internals: wraparound age, archiver failing, invalid or duplicate indexes, sequences nearing exhaustion

© 2026 pgEdge, Inc.

Real example: one host, one run

- ✓ [CRITICAL] OOM killer fired 7 times in `dmesg`
- ✓ [CRITICAL] 1 filesystem ≥ 80% full: `/` at 99%, 2.2 GB free
- ✓ [CRITICAL] 1 network HBA rule uses `trust` authentication
- ✓ [WARNING] Transparent Huge Pages is set to `always`
- ✓ [WARNING] `vm.swappiness = 20`, and swap is configured on a PostgreSQL host
- ✓ [WARNING] `shared_buffers` at 6% of RAM; `postgresql.auto.conf` has `ALTER SYSTEM` drift
- ✓ [WARNING] WAL buffers exhausted 27,709 times; 2 duplicate-index groups; 2 unexpected superusers
- ✓ None of this is in the Postgres log

© 2026 pgEdge, Inc.

03
The verdict

What to make of these findings

© 2026 pgEdge, Inc.

Deterministic findings, AI brief

Findings (rules)

- ✓ A verdict per category: HEALTHY, WARNING or CRITICAL
- ✓ From the worst finding. Reproducible, no model involved
- ✓ UNKNOWN: not collected (doesn’t mean it’s fine)
- ✓ Only `rules/` can create a finding. The LLM model never does

Brief (model, optional)

- ✓ What it means & what to do first, as a senior DBA would write it
- ✓ Claude, Gemini, OpenAI or any OpenAI-compatible endpoint, or a local Ollama
- ✓ No provider available? You still get findings and verdicts, just no brief
- ✓ LLM notices something extra? It stays in the brief, labelled as the model's

© 2026 pgEdge, Inc.

The severity floor

- ✓ The model proposes a status tag
- ✓ The floor: never better than the worst finding
- ✓ CRITICAL cannot become “consider investigating”
- ✓ It can go worse, but never softened

rules/base.py

```
def apply_finding_floor(verdict, severities):
    """Return *verdict* floored by the worst finding severity.

    The floor keeps an LLM answer from downgrading a violation
    the rules detected, and carries a category through an LLM
    outage: with no verdict at all, findings alone decide.
    """
    floor = max(
        (severity_rank(s) for s in severities), default=0
    )
    if verdict is None or floor > severity_rank(verdict):
        return rank_to_verdict(floor)
    return verdict
```


© 2026 pgEdge, Inc.

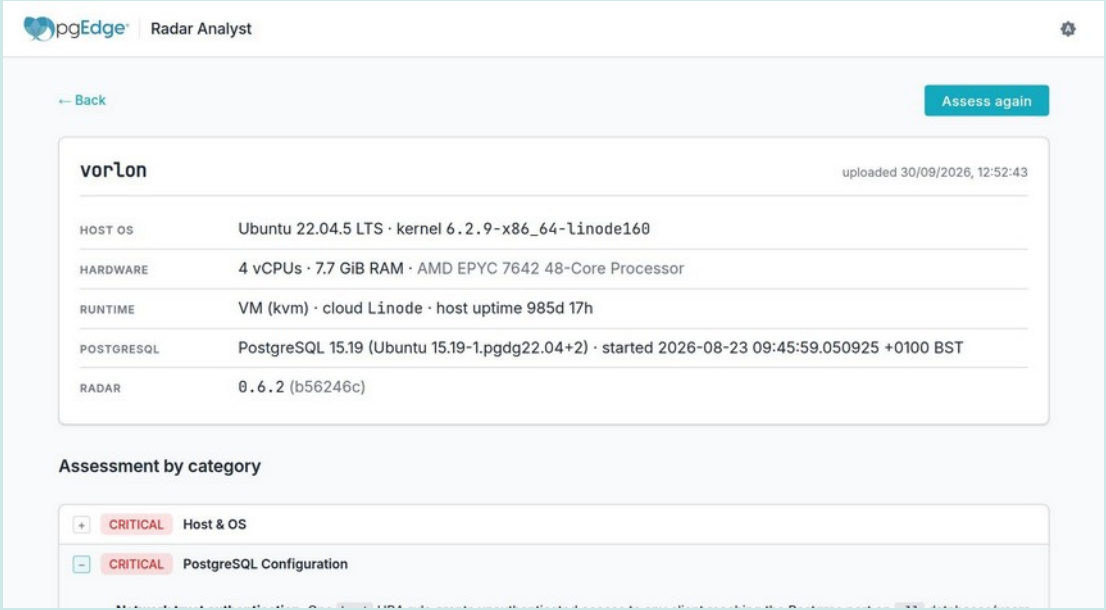
Keeping the LLM in check

- ✓ Prompt: “you are a senior PostgreSQL DBA”. Standard DBA terms used
- ✓ We don’t dump the whole zipfile on the model, just a digest
- ✓ Everything from the archive goes inside `<user_data>` tags: data, never instructions
  - A table called `ignore previous instructions` is just a badly named table
- ✓ Thresholds included in prompt: model and rules agree on “warning”
- ✓ The brief may change from run to run, findings and verdict cannot

© 2026 pgEdge, Inc.

The console

- ✓ Upload the zip, watch the progress stream
- ✓ Five category cards: verdict, brief, and the findings behind it
- ✓ One card per database with findings (healthy ones: no model call)
- ✓ Every brief links to the archive files. Also available as JSON



© 2026 pgEdge, Inc.

Recap

- ✓ Agents have to be there before the incident, a snapshot can be taken during
- ✓ Metadata is enough to diagnose, and is safe to send
- ✓ Keep deterministic findings and AI brief separate
- ✓ LLM cannot downgrade severity of rule-based findings
- ✓ Read the zipfile before you send it anywhere
- ✓ Open Source (PostgreSQL licence), feel free to add rules

© 2026 pgEdge, Inc.

Links

radar

- ✓ `github.com/pgEdge/radar`
- ✓ Static binaries: Linux and macOS, amd64 and arm64
- ✓ `DATA.md` explains every file in the zip

radar-analyst

- ✓ `github.com/pgEdge/radar-analyst`
- ✓ `docker compose up -d --wait`
- ✓ Docs, walkthrough, API reference, `ARCHITECTURE.md`

© 2026 pgEdge, Inc.

Find me on socials

YouTube: youtube.com/JimmyAngelakos

Mastodon: fosstodon.org/@vyruss

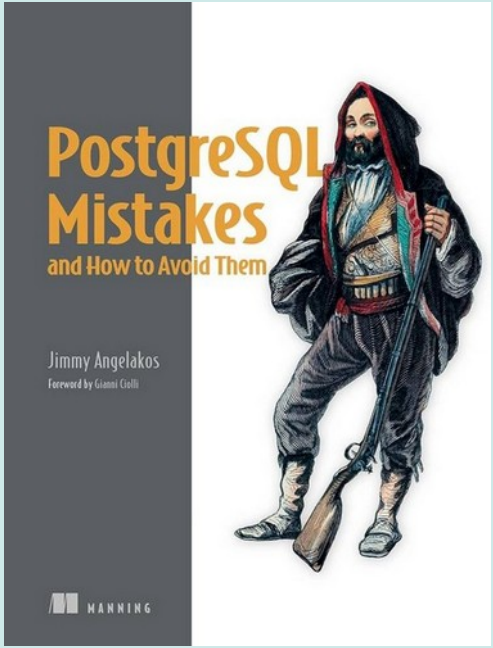
BlueSky: bsky.app/profile/vyruss.org

LinkedIn: linkedin.com/in/vyruss

© 2026 pgEdge, Inc.

Thank you!

Discount code **pgconfeu26** for **45% off** all Manning products



© 2026 pgEdge, Inc.