Parsing Postgres logs the non-pgBadger way

Kaarel Moppel, Freelance PostgreSQL Consultant
pgConf.EU 2025, Riga

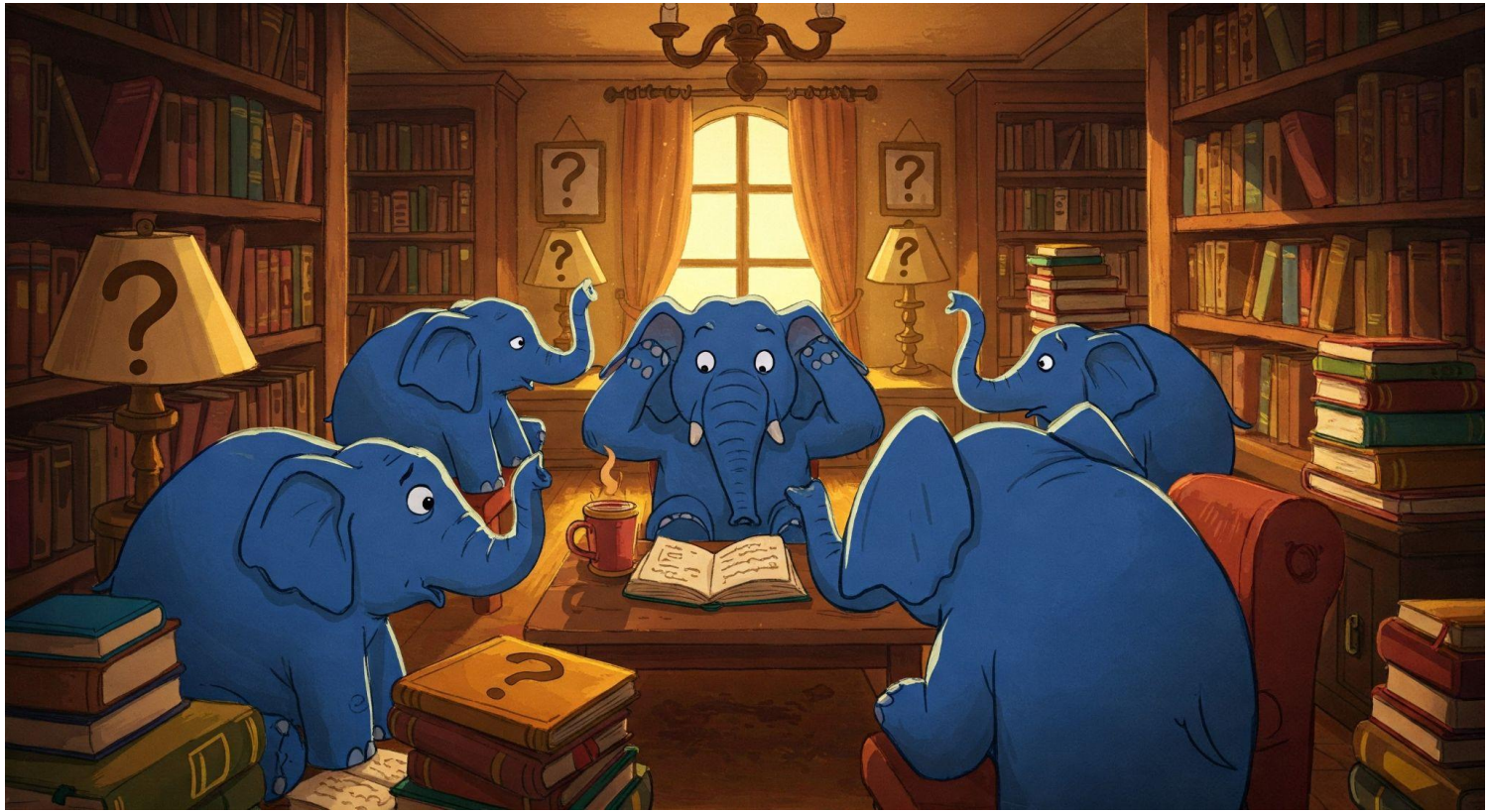
$ whoami

- Full-time “wrestling” with databases since 2007
- 20K+ hours in the Postgres ecosystem
  - Many hats along the way
  - Have developed somekind of a gut feeling on “Postgres”-y things if anything
- Freelance DBE / DBA / consultant
  - https://kmoppel.github.io/ (Postgres Blog & Services)

Agenda

- Value of Postgres server logs
- Log management approaches
- Common tools
- Cloud / SaaS tools
- AI-assisted tooling
- A fresh take - pgweasel

Value of logs



Why look at Postgres logs ?

The Postgres Stats subsystem doesn’t cover everything by far!

- The obvious - get full details on PostgreSQL errors
- Slow queries with specific execution params
- Query plans (auto_explain)
- Detect unauthenticated access / brute force attack attempts
  - Sadly possible by default with Postgres ...
- Background processing details (e.g. autovacuum, 3rd party exts.)
- Lock wait details
- Temp file details
- Backend terminations / OOM
- ...

Log management approaches

Log management approaches

- Ad-hoc searching/grepping
  - grep, pgBadger & co
  - Exposing logfiles directly via file_fdw for SQL power
- Continuous analyzing
  - Loading into a DB (e.g. Postgres + ZFS)
  - Homegrown parsing (e.g. Cron+Python)
  - Filebeat / ELK (Elastic) pipeline
    - Sometimes via Syslog
  - Prometheus integration, e.g. pgwatch “server_log_event_counts”
  - Generic framework (Nagios etc) Postgres plugins
  - Cloud tools e.g. Loggly or full APM suites
- Hooking into the Postgres log emitting mechanism
  - Tapping into “emit_log_hook”, a la “redislog” (unmaintained sadly)
- …

Common tools

Common tools - grep & co

The daily driver for many uses cases...

```
grep -E 'ERROR|FATAL' testdata/cloudsql.log
egrep 'ERROR|FATAL' testdata/cloudsql.log
egrep -c 'ERROR|FATAL' testdata/cloudsql.log

# Alternatives like “ripgrep” can be faster / nicer ...
rg 'ERROR|FATAL' testdata/cloudsql.log
...
```

Common tools - awk

For ad-hoc analysis it’s “possible” to achieve below tasks (and DBA-s usually have something similar at their fingertips)...but not easy. LLMs can help a bit of course :)

- Find the N slowest queries
- Errors by type
- Connection attempts per user or host
- Average query duration
- Event counts by hour / minute
- ...

PS with CSV format life is somewhat nicer:

```
awk -F, '$12 == "ERROR" && $3 ~ /mydb/' postgresql.csv
```

Common tools - awk - an example (not perfect)

```
gawk '{
    # time_parts[1] will contain "HH:MM:SS", time_parts[2] will contain the milliseconds
    split($2, time_parts, ".");

    timestamp_str = $1 " " time_parts[1];

    # Replace dashes and colons with spaces for mktime() format "YYYY MM DD HH MM SS".
    gsub(/[-:]/, " ", timestamp_str);

    # Convert the string to an epoch timestamp.
    epoch = mktime(timestamp_str);

    # Format the epoch time to an hourly key (e.g., "2025-05-21 10").
    hour_key = strftime("%Y-%m-%d %H", epoch);

    hourly_counts[hour_key]++;
}
END {
    print "--- Hourly Event Summary ---";
    for (hour in hourly_counts) {
        print hour ":00:00  | Events: " hourly_counts[hour];
    }
}' testdata/cloudsql.log
```

Common tools - pgBadger

The #1 choice for many - a solid tool in principle! Main highlights:

- A HTML report with graphs by default
- Single-run and incremental modes
- Parallel processing (--jobs) support (not for CSV)
- Integrated remote access (SSH, http[s], [s]ftp)
- Ecosystem aware - RDS Cloudwatch & pgBouncer
- --exclude-query / --include-query regex object filtering
- Bind parameters “merging” for prepared statements
- JSON output
- --dump-raw-csv for loading / more structure
- ...

```
krl@e7420:~$ pgbadger --help | grep -E '^\s+\-{1,2}' | wc -l
106
```

Common techniques - loading into Postgres

To gain the power of SQL! 💪 Works OOTB with CSV only...

```
CREATE FOREIGN TABLE pglog (
  log_time timestamp(3) with time zone,
  user_name text,
  database_name text,
  process_id integer,
  connection_from text,
  session_id text,
  session_line_num bigint,
  command_tag text,
  session_start_time timestamp with time zone,
  virtual_transaction_id text,
  transaction_id bigint,
  error_severity text,
  sql_state_code text,
  message text,
  detail text,
  hint text,
  internal_query text,
  internal_query_pos integer,
  context text,
  query text,
  query_pos integer,
  location text,
  application_name text,
  backend_type text,
  leader_pid integer,
  query_id bigint
) SERVER pglog
OPTIONS ( filename 'log/pglog.csv', format 'csv' );
```

The fields / format can be best viewed from “file_fdw” documentation https://www.postgresql.org/docs/current/file-fdw.html

PS - a reminder on pgbadger --dump-raw-csv mode again for default textual logs

Cloud / SaaS tools

Cloud / SaaS tools - no shortage

The “problem space” is not new actually - quite some tools out there!

- Top managed DBaaS providers have something built-in
  - Usually with some proprietary query langage though :(
- Datadog
- AppDynamics
- Pganalyze
- Loggly
- BetterStack
- ...

PS Note that in many cases you need to adhere to their understanding of `log_line_prefix`!

Tooling - AD 2025

AI-assisted tooling ?

The big questions:

1. Can public LLM-s find the needle from a haystack?
2. Is it cost-effective?
3. Security ...

Based on my experiments:

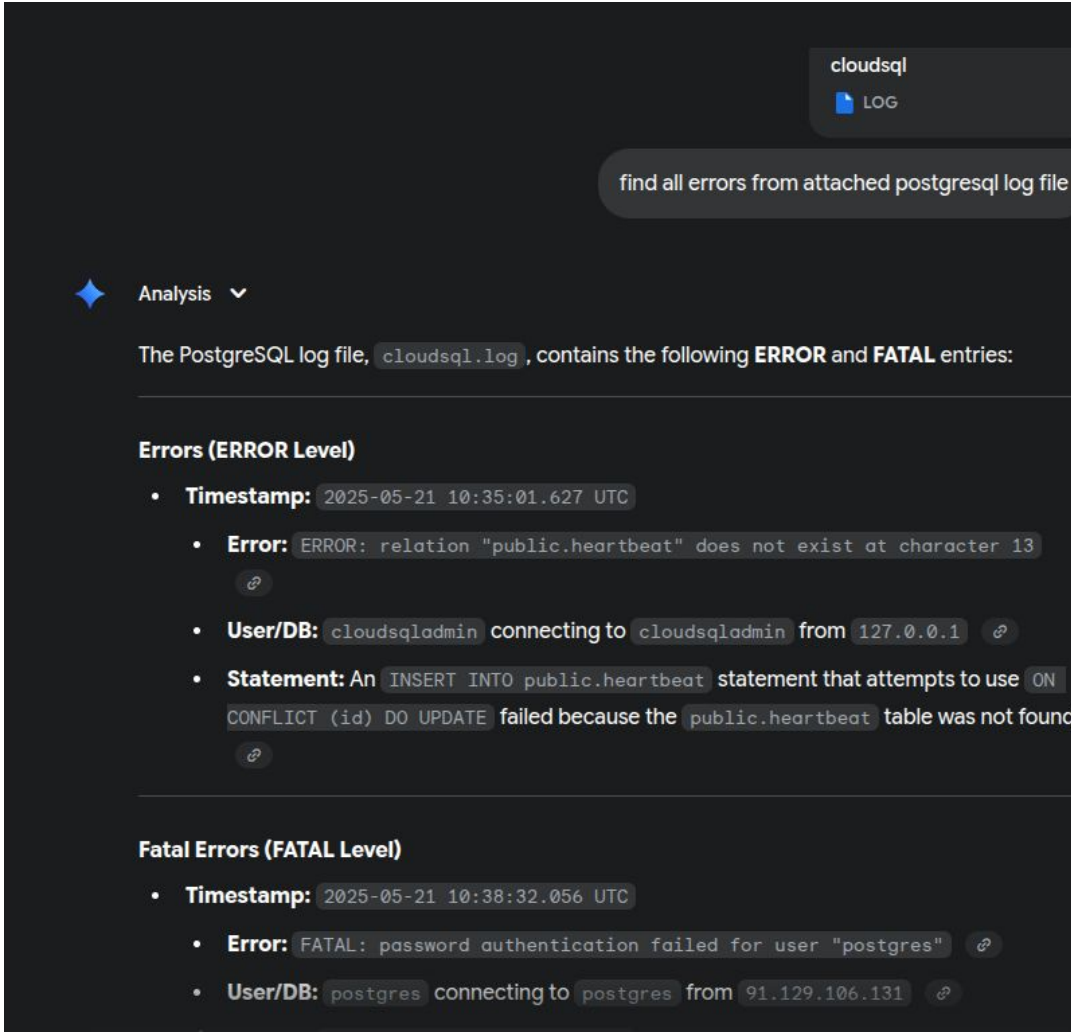
- No point to throw everything at LLMs - need to pre-filter
  - Response time
  - $$ can get prohibitive - 1GB log file ~ $1 with a fast model
  - Context window limitations / chunking
- Thus need to get hands dirty anyways - meaning direct usage loses in appeal...

AI-assisted tooling ?

Demo - a simple 400kB / 4k line file with 10 problems: https://github.com/kmoppel/pgweasel/blob/main/testdata/cloudsql.log

Result: Misses 2 simple “SELECT asd” ERROR’s + 3 FATAL’s with rarer protocol errors

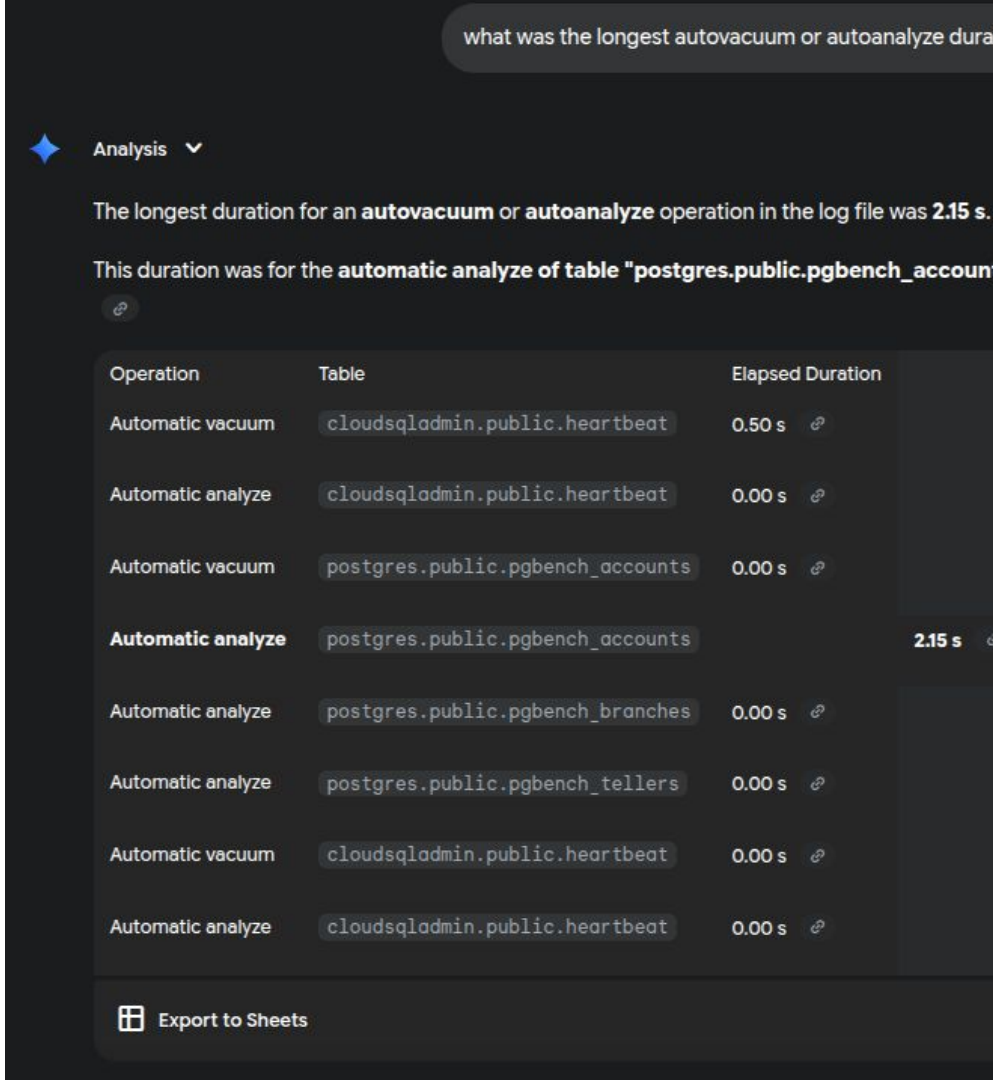
Model: Gemini 2.5 Flash



AI-assisted tooling ?

But better already with autovacuum analysis!

As there are a gazillion of evolving models out there, maybe one day good enough 🤞 ...



AI-assisted tooling ?

Some “specialized” progress is being made of course, e.g.:

https://github.com/salesforce/logai

...but it’s harder than you think!

Tooling - summary

I’ve personally tried out A LOT of tools...but mostly concluded:

- Not that they create as many problems as they solve, but every tool brings some limitations as usual
  - Which tend to surface only when past some first simpler milestones
- Sadly common pretty to hit bugs or performance / parallelism walls
- The tools keep changing...creating random backlog

Consider if you really need all those features!

Starting with something super-simple, but reliable, and fully under your control, is never a bad idea!

pgweasel

pgweasel - to complement pgBadger

A new project, aiming to gain community attention / action!

The tool tries to be different from pgBadger, focusing on the “Pareto” DBA flow of **“on the server” CLI usage** for humans :)

- Much simple to handle - command + subcommand approach
  - Less features - “less is more” philosophy
- Way faster! Currently in Golang
  - Rust re-write started (in “rust-rewrite” branch)
- A single binary - Perl can get prohibitive on containers
- Zero-config, always works (no --log-line-prefix)
- User-friendly - relative time inputs, command aliases, auto-detect logs from standard locations, automatic multi-core usage
- ...

pgweasel - looking for contributors / co-owner!

- One person has quite a narrow view on things...
- Plus the typical side-project time issue
- Also...to make it work really good under all conditions a lot of server log examples would be needed!

In case you have some large real-life logfiles without security risks, please send me privately some S3 etc link or just open an Github issue!

1GB+ would be especially nice! 🙏

https://github.com/kmoppel/pgweasel

Licence: PostgreSQL license

All kinds of feedback, feature ideas, pull requests very much appreciated!
Or just a ⭐ :)

DEMO time!