A WordPress-Shaped CMS in Rust: Latency, Memory, and Admin-Path Cost versus PHP-FPM on One Machine

Jermaine Harris
Hutts Enterprises / Ferrus Labs
2026-09-20 · CC BY 4.0
Keywords: cs.SE, cs.PF, CMS, Rust, WordPress, OxidePress
Source git 6ef9c2d · technical report

Abstract

OxidePress is a WordPress-shaped CMS: server-rendered public pages, an /admin login, posts as first-class records, health JSON. The storage engine under the lab binary is pgrust (Matt Malisper’s Postgres-in-Rust), not a hosted PostgreSQL. We compare that stack to stock WordPress (php8.3-apache plus MariaDB 11) on one workstation, using local fixtures only.

Public home TTFB (serial curl, n=20 after warmup) was 21.9 ms plus or minus 4.9 ms for OxidePress versus 34.5 ms plus or minus 2.6 ms for WordPress. A 5-second serial GET loop reached 40.6 req/s versus 23.4 req/s. GET /admin/login versus wp-login.php was 2.6 ms versus 12.3 ms (n=10). Process RSS was not a win for Rust: OxidePress about 290 MiB versus Apache about 49 MiB plus MariaDB about 125 MiB.

The engine under OxidePress is pgrust; we do not reuse pgrust’s published ClickHouse or PostgreSQL leaderboard numbers. This is an experience report on a WordPress-shaped product, not a new query engine.

1. Introduction

Replacing WordPress is a product claim, not a language claim. Operators care about public TTFB, a login form that is cheap to GET, and whether RSS fits the box. They do not care that the binary is Rust unless those numbers move.

Research question. For local fixtures on one host, how do public TTFB, a crude serial RPS, RSS, and a login-page GET compare between OxidePress plus pgrust and WordPress plus MariaDB plus PHP-FPM/Apache?

Contributions:

- C1. Same-box numbers in results/03/same_box.csv (Tables 1-4).
- C2. Explicit engine disclosure: postgres_engine=pgrust from GET /health.
- C3. A threat section on unequal post counts, Docker versus host network, and serial curl.

2. What OxidePress is (and is not)

OxidePress at ae2175ea0ab214ae833633f2f288907935bbb3f2 is an Axum SSR binary (target/release/oxidepress) listening on 127.0.0.1:3000. Health JSON on the lab run: catalog=postgres, postgres_engine=pgrust, plan=community, demo=false, ok=true. DATABASE_URL pointed at loopback pgrust 127.0.0.1:54329.

It is WordPress-shaped: public home, /health, /admin/login. It is not WordPress: no WP-CLI, no plugin directory, no wp-cron. Comparing a 50-post import to a 101-post WP-CLI generate is already a threat; we state it instead of pretending the corpora match.

pgrust is cited as related work only. ClickBench and PostgreSQL leaderboard numbers from that project are about an analytics engine, not this CMS request path.

3. System under test

WordPress: wordpress:php8.3-apache plus mariadb:11 via harness/03-oxidepress-wp/docker-compose.yml, published at 127.0.0.1:8088. wp core install then wp post generate --count=50. WP-CLI listed 101 posts (generated plus defaults).

OxidePress: 50 synthetic JSON posts imported through /admin/tools/import after a local admin session. Release binary on the host network namespace, not in Docker.

Hardware: same i5-12600KF workstation as Paper 1 (64 GiB RAM, Linux 6.17).

4. Method

Load. curl -w time_starttransfer. Three warmup GETs discarded, then 20 reps for homes and health, 10 reps for login pages. Five-second serial loops for RPS (portable; not wrk). No cookies on public GET. Admin path is unauthenticated GET of the login form only. No posted credentials in the timed loop.

RSS. ps RSS of processes matching oxidepress, apache2, mariadbd. For Apache we take the max matching RSS, which is one worker, not the sum of all workers.

What we did not do. wrk -c 200. Authenticated admin table render. CDN or nginx in front. Production vhost. Mixing pgrust ClickBench scores into Table 1.

5. Results

Table 1. Public TTFB, n=20, population stdev. Source: results/03/same_box.csv.

Arm	URL	mean TTFB (ms)	stdev (ms)	errors
OxidePress home	127.0.0.1:3000/	21.93	4.87	0
OxidePress health	127.0.0.1:3000/health	0.30	0.08	0
WordPress home	127.0.0.1:8088/	34.49	2.64	0

Table 2. Serial GET throughput, 5 s.

URL	requests	errors	rps
OxidePress /	203	0	40.55
WordPress /	117	0	23.38

Table 3. Login page GET, n=10.

Arm	mean TTFB (ms)	stdev (ms)
OxidePress /admin/login	2.62	0.76
WordPress /wp-login.php	12.35	2.21

Table 4. RSS (KiB, ps).

Process	RSS KiB	MiB
oxidepress	297332	290
apache2 (max match)	49852	49
mariadbd	127676	125
Apache + MariaDB	177528	173

Home TTFB ratio WordPress / OxidePress is about 1.57. Serial rps ratio OxidePress / WordPress is about 1.73. Login GET ratio is about 4.7. RSS ratio OxidePress / (Apache+MariaDB) is about 1.67 against OxidePress.

Health is 0.30 ms: the process is up. Home is 22 ms: SSR plus catalog. WordPress home is tighter (stdev 2.6 ms vs 4.9 ms) even though slower; OxidePress home has more run-to-run jitter on this box.

6. Interpretation

OxidePress home is faster on this box under a polite serial client. That does not predict production at the far side of nginx and Cloudflare, and it does not predict wrk -c 200. Serial curl is closer to “an operator hitting reload” than to a load test.

RSS is higher for the single OxidePress process than for Apache alone. Adding MariaDB still leaves WordPress’s two-process footprint smaller than OxidePress RSS here (173 MiB vs 290 MiB). Debug-versus-release is not a factor (release binary). Docker networking for WordPress (localhost published port) may add a small constant versus OxidePress on the host network namespace; we did not isolate that constant.

Login GET is the cleanest language-plus-stack comparison in the set: both are unauthenticated HTML forms. WordPress still has to boot enough of PHP to render wp-login.php. OxidePress serves a small Axum template. Neither number includes a successful password POST or a session cookie round-trip.

7. Discussion and threats

Treat pgrust as the storage engine under test, not as a claimed analytics database champion. Health JSON already names it. Hiding that would make the CMS look like “Rust plus PostgreSQL” when the lab engine is neither stock Postgres nor production-ready per upstream (pgrust 0.2).

Threats. (1) Unequal post counts (50 vs 101). (2) Serial curl, not concurrent. (3) WordPress in Docker, OxidePress on the host. (4) No authenticated admin table render. (5) No CDN. (6) Apache RSS is max worker, not sum. (7) Fixtures are synthetic JSON / WP-CLI lorem, not a cloned production database.

8. Related work

We cite pgrust's own narrative and do not copy ClickBench scores. WordPress remains the CMS baseline. Other Rust CMS experiments exist; this report is one same-box pairing with a disclosed engine, not a survey.

9. Ethics

No production vhost. Compose passwords are local fixture strings and are not used in this PDF. Demo passwords that appear in OxidePress HTML templates are not repeated here.

10. Conclusion

On one machine with small fixtures, OxidePress was faster on public home TTFB and serial rps, used more RSS, and served a cheaper login GET. The honest reading is a WordPress-shaped SSR binary on a lab Postgres-in-Rust engine, not a drop-in host for a 200-site fleet.

Appendix A. Artifact

```
cd harness/03-oxidepress-wp && docker compose up -d
python3 harness/03-oxidepress-wp/run_bench.py
```

Start pgrust and the OxidePress release binary before the Python driver. See the ZIP supplement.

References

- Malisper, M. pgrust: rebuilding Postgres in Rust with AI. https://malisper.me/pgrust-rebuilding-postgres-in-rust-with-ai/
- WordPress Plugin Handbook. https://developer.wordpress.org/plugins/
- oxidepress (pinned). git SHA ae2175ea0ab214ae833633f2f288907935bbb3f2