03 oxidepress wp

git a4aed0a | 2026-09-20 | Jermaine Harris, Hutts Enterprises / Ferrus Labs

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

Jermaine Harris

Hutts Enterprises / Ferrus Labs

date: "2026-09-20"

license: "CC BY 4.0"

keywords: "cs.SE, cs.PF, CMS, Rust, WordPress, OxidePress"

Abstract

We compare OxidePress (Axum SSR, pgrust) to stock WordPress (php8.3-apache, MariaDB 11) on one workstation, using local fixtures only. Public home TTFB (serial `curl`, n=20 after warmup) was 21.9 ms ± 4.9 ms for OxidePress versus 34.5 ms ± 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 ≈ 290 MiB versus Apache ≈ 49 MiB plus MariaDB ≈ 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.

Keywords

cs.SE; CMS; WordPress; Rust; TTFB

1. Introduction

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

Contributions:

- C1. Same-box numbers in `results/03/same_box.csv` (Table 1–3).
- C2. Explicit engine disclosure: `postgres_engine=pgrust` from `GET /health`.
- C3. A threat section on unequal post counts and serial curl.

2. System under test

OxidePress `ae2175ea0ab214ae833633f2f288907935bbb3f2`, binary `target/release/oxidepress`, listen `127.0.0.1:3000`, `DATABASE_URL` on loopback pgrust `127.0.0.1:54329`. Health JSON: `catalog=postgres`, `postgres_engine=pgrust`.

WordPress `wordpress:php8.3-apache` + `mariadb:11` via `harness/03-oxidepress-wp/docker-compose.yml`, `127.0.0.1:8088`. `wp core install` then `wp post generate --count=50`.

pgrust is Matt Malisper's Postgres-in-Rust project [malisper-pgrust]. Related work only.

3. Method

Hardware. Same i5-12600KF workstation as Paper 1.

Content. OxidePress: 50 synthetic JSON posts imported through `/admin/tools/import` after a local admin session. WordPress: 50 generated posts plus default content (101 posts listed by WP-CLI). Not byte-identical corpora.

Load. `curl -w time_starttransfer`, 3 warmup + 20 reps for homes; 10 reps for login pages; 5 s serial loops for RPS. No `wrk`. Cookies were not sent 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`.

4. Results

Table 1. Public TTFB, n=20. `results/03/same_box.csv`.

Arm	URL	mean TTFB (s)	stdev (s)	HTTP errors
OxidePress home	http://127.0.0.1:3000/	0.02193	0.00487	0
OxidePress health	http://127.0.0.1:3000/health	0.00030	0.00008	0
WordPress home	http://127.0.0.1:8088/	0.03449	0.00264	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 (s)
OxidePress `/admin/login`	0.00262
WordPress `/wp-login.php`	0.01235

Table 4. RSS (KiB, `ps`).

Process	RSS KiB
oxidepress	297332
apache2 (max match)	49852
mariadbd	127676

Home TTFB ratio WP/OxidePress ≈ 1.57. Serial rps ratio OxidePress/WP ≈ 1.73.

5. Discussion and threats

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`. 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. Debug-vs-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.

Threats. Unequal post counts (50 vs 101). Serial curl. WordPress in Docker, OxidePress on the host. No authenticated admin table render. No CDN. pgrust version 0.2 is not production-ready per upstream; this is a lab engine.

6. Related work

We cite pgrust's own narrative [malisper-pgrust] and do not copy ClickBench scores. WordPress remains the CMS baseline [wp-plugin-api].

7. 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.

8. 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. Treat pgrust as the storage engine under test, not as a claimed analytics database champion.

9. Artifact appendix

```
cd harness/03-oxidepress-wp && docker compose up -d
```

wp core install + post generate (see campaign notes)

start pgrust + oxidepress release binary

```
python3 harness/03-oxidepress-wp/run_bench.py
```

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/