Zapolu

Performance audit: deep dive (sample report)

> **This is a sample.** “Acme Outdoor s.r.o.” is a fictional client. The findings are composites drawn from real audits we have delivered on other stores, with identifying details and numbers changed. The structure and the level of detail are what a paying client receives.

Client	Acme Outdoor s.r.o. (fictional)
Store	acme-outdoor.example (Magento 2.4.6, Luma frontend)
Stack	Varnish 7, Redis 7, MySQL 8, 2 app nodes behind a load balancer
Data window	28 days of field data, 14 days of APM traces, one load test
Tooling	New Relic, CrUX, Lighthouse, k6, varnishstat, MySQL slow log
Author	Luboš Zápotočný, Zapolu s.r.o.
Engagement	Deep dive, 7 business days, fixed price

For a brief overview, read sections 1 and 5. The sections in between contain the supporting evidence.

1. Executive summary

The store’s slowness originates on the server side, not from underpowered hardware. Field LCP on catalog pages is **4.6 s at p75** (target: ≤ 2.5 s), driven by a **1.9 s TTFB**. The TTFB is high because the full-page cache is effectively disabled: Varnish serves only **34 %** of catalog requests, and 38 % of all server time goes into rendering product pages at 2.2 s average. During the May campaign evening the checkout p95 reached **6.1 s** with **3.4 % of requests failing**, while application CPU never exceeded 12 %.

Eleven findings follow. Six of them cost a day of work or less, and two of those six affect only configuration files. Executing rows 1-5 of the fix plan should bring catalog LCP under 2.5 s at p75 and stop the checkout failures at peak; row 6 then brings checkout p95 under 2 s during campaigns. The remaining rows are ranked by return, so the plan can be halted at any point.

In financial terms: during the three campaign hours on May 12, roughly 230 checkouts ended in an error. At the store’s average order value of 1 800 CZK (≈ €72), that is about 415 000 CZK in attempted orders that failed, not counting customers who abandoned the checkout after a six-second wait. The remaining findings are stated in milliseconds; a revenue figure for them would be speculative, so we have omitted it.

2. What we measured, and how

- **Real-user metrics:** Core Web Vitals from a 28-day CrUX query, segmented by page type (home / category / product / checkout). Synthetic Lighthouse runs were used only to reproduce findings, not to score them. Where the store stands against Google’s thresholds:

Metric (p75, field data)	Store	Threshold	Verdict
LCP	4.6 s	< 2.5 s	fail
INP	280 ms	< 200 ms	fail
CLS	0.04	< 0.1	pass
FCP	2.9 s	< 1.8 s	fail
TTFB	1.9 s	< 0.8 s	fail

- **Server-side trace:** New Relic transaction profile over 14 days, showing where the server time actually goes:

Transaction	Share of server time	Avg response
`catalog/product/view`	38 %	2.2 s
`catalog/category/view`	17 %	1.8 s
checkout REST (cart + shipping)	11 %	1.4 s (6.1 s p95 at peak)

  Plus the MySQL slow query log, the external-services profile, and Varnish hit/miss statistics per URL pattern from `varnishstat`.

- **Load test:** a replay of the May 12 campaign traffic shape (0 → 420 req/s over 20 minutes) against staging, with the same cache state as production.
- **Configuration review:** PHP and OPcache settings, Composer autoloader, `env.php`, the Varnish VCL, and MySQL settings, each against Magento's documented recommendations.
- **Bundle and asset audit:** what ships on first view of a product page, what blocks rendering, and what is unnecessary payload.

Most findings quote the exact command or query we used, partly so your team can verify our work, partly because the same probes run on any Magento store, including yours. Every number below says where it came from; where a finding's impact is an estimate, it is labeled as one.

3. Findings overview

High: customers are affected, or revenue depends on it. **Medium**: staff are affected daily, or data is silently corrupted. **Low**: minor effort that removes a real operational risk.

#	Finding	Severity	Effort	Expected impact
F1	Full-page cache is effectively disabled and cannot be invalidated	High	1–2 days	–700 ms TTFB at p75 on catalog
F2	PHP runtime and autoloader misconfigured for Magento	High	0.5 day	–400 ms on every uncached request
F3	Checkout blocks on a synchronous ERP stock call	High	3–5 days	–2.8 s checkout p95 at peak; removes timeout failures
F4	Frontend ships 2.3 MB of JavaScript, render-blocking	High	project	–1.2 s LCP (interim mitigations: –0.4 s)
F5	Product images: 1.6 MB PNG heroes, no responsive sizes	Medium	1 day	–0.6 s LCP on product pages

#	Finding	Severity	Effort	Expected impact
F6	Indexers left on “Update on Save”	Medium	0.5 day	Admin saves 30 s → ~2 s; eliminates stock drift
F7	PHP-FPM pool exhausted by session locks at peak	High	2 days	Checkout survives campaign traffic
F8	No grace mode in Varnish	Low	0.5 day	Absorbs the stampede after every cache flush
F9	Two extensions observe the same event; one is abandoned	Medium	1 day	–300 ms on add-to-cart
F10	41 GB of dead indexer temp tables in MySQL	Low	0.5 day	Backups shrink 60 %; restore time roughly halves
F11	Admin order grid filters on an unindexed column	Low	0.5 day	Order grid 12 s → under 1 s

4. Detailed findings

F1: Full-page cache is effectively disabled and cannot be invalidated (High)

What we saw. Varnish reports a 34 % hit rate on catalog URLs. For a catalog this size, with this crawler traffic, above 90 % is normal. Every category and product page carries `X-Magento-Cache-Debug: MISS`. There are two independent causes:

1. A customized `default.xml` marks a header block (a store-switcher that reads the customer session) as `cacheable="false"`. In Magento, one uncacheable block in the layout makes the **entire page** uncacheable. The block arrived in an extension update in November; the hit rate collapsed the same week.
2. Varnish is enabled in the admin (`system/full_page_cache/caching_application = 2`), but `env.php` has no `http_cache_hosts` section, so Magento cannot send purge requests to Varnish at all. The team works around it by restarting Varnish after every deploy, which is why deployments are disruptive (see F8).

How we verified it:

```
$ curl -sI https://acme-outdoor.example/tents | grep -i cache-debug
X-Magento-Cache-Debug: MISS
# same on every catalog URL we probed

$ grep -R 'cacheable="false"' app/design app/code | wc -l
1
# the store-switcher block, default.xml

$ php -r 'var_export(array_key_exists("http_cache_hosts", (include "app/etc/env.php")));'
false
# Magento cannot purge Varnish
```

Fix. Render the switcher via a private-content section (customer data JS) so the page stays cacheable, which is the documented pattern for session-dependent fragments, and add `http_cache_hosts` to `env.php` so tag-based invalidation works and the restarts can stop.

Effort / impact / risk. 1–2 days including regression tests. Expected –700 ms TTFB at p75 on catalog pages (measured: cached catalog responses serve in 80 ms vs. 1.9 s uncached). Risk: low; both changes are confined and reversible per deploy.

F2: PHP runtime and autoloader misconfigured for Magento (High)

What we saw. The codebase contains ~125 000 PHP files; OPcache is configured to hold 10 000. The Composer autoloader was never optimized: `vendor/composer/autoload_static.php` contains no class map, and the New Relic traces show whole seconds spent in `ObjectManager` factory resolution on cold paths. Current versus recommended:

Setting	Current	Recommended
`opcache.max_accelerated_files`	10 000	100 000
`opcache.memory_consumption`	128	512
`opcache.validate_timestamps`	On, revalidate every 2 s	Off; reset on deploy
`opcache.enable_cli`	Off	On (indexers and crons run PHP too)
`realpath_cache_size`	4M	32M
`realpath_cache_ttl`	120	7200
Composer autoloader	not optimized	`dump-autoload --optimize` in the build

How we verified it:

```
$ find . -type f -name '*.php' | wc -l
124862
# PHP files in the codebase

$ php -i | grep 'opcache.max_accelerated_files'
opcache.max_accelerated_files => 10000
# holds 8 % of them

$ grep -c 'classMap' vendor/composer/autoload_static.php
0
# autoloader was never optimized
```

Why. With these values, PHP re-stats and re-compiles a large part of the codebase continuously; the profiles attribute over half of uncached render time to class loading and page assembly rather than business logic.

Fix. One config change plus one build step. The only coupling to respect: with `validate_timestamps` off, the deploy pipeline must reset OPcache on release. We've documented the exact step for your current pipeline in the fix plan.

Effort / impact / risk. 0.5 day. Estimated –400 ms on every uncached request (and faster crons and indexers as a side effect). Risk: low, provided the deploy step lands together with the config.

F3: Checkout blocks on a synchronous ERP stock call (High)

What we saw. On the shipping step, the trace shows a synchronous HTTPS call to the ERP's stock endpoint with a 3-second timeout and one retry; the external-services profile attributes 92 % of all external wait time to this one endpoint. At campaign load the ERP slows down, the calls eat their full timeout, and checkout requests stack up. This is where the 6.1 s p95 and the 3.4 % failures come from. The load test reproduces it on staging.

Why. Real-time stock confirmation was added after an oversell incident last year. The intent is sound, but it puts a third-party system's latency inside every checkout.

Fix. Move stock reconciliation out of the request path: reserve against the local inventory, confirm asynchronously, and alert on divergence. Overselling stays covered, and checkout stops waiting for the ERP. (This is the same pattern described in our material on ERP/e-shop stock synchronization; a reference is available on request.)

Effort / impact / risk. 3–5 days including the divergence alert. Removes ~2.8 s from checkout p95 at peak and the timeout failures entirely (both measured in the load test with the call stubbed). Risk: medium; it needs a written sign-off on the new oversell window (estimated at < 0.1 % of campaign orders, vs. 3.4 % of checkouts failing today).

F4: Frontend ships 2.3 MB of JavaScript (High)

What we saw. First view of a product page loads 2.3 MB of JavaScript across 61 files; main-thread blocking time is 2.9 s on a mid-range phone. Four of the fourteen marketing tags no longer have a working account behind them.

Why. Partly the Luma frontend architecture, partly eight years of accumulated tags. No single tag dominates the total.

Fix, short term (this quarter). Remove the four dead tags, defer the analytics bundle, lazy-load the reviews widget below the fold. Estimated −0.4 s LCP, 2 days.

Fix, real (separate decision). A storefront rebuild on Hyvä. On comparable catalogs this takes LCP into the green at p75 and cuts frontend maintenance cost; it is a 6–10 week project and deserves its own scoped proposal, not a line in a fix list. We mention it here so that the short-term estimate above is not read as the maximum achievable.

F7: PHP-FPM pool exhausted by session locks at peak (High)

What we saw. During the load test, failures begin at ~280 req/s while CPU sits at 12 %. The FPM status page shows all workers busy; the traces show them waiting on Redis session locks: customers polling the checkout success page hold locks that queue every other request from the same session.

Fix. Disable session locking for the polling endpoints, raise the worker count to what the memory headroom actually allows (the current number dates from a smaller instance type), and add an FPM saturation alert; this outage was undetected because the CPU graphs appeared healthy.

Effort / impact / risk. 2 days including a re-run of the load test. With F3 done as well, staging survives 420 req/s with checkout p95 at 1.9 s. Risk: low.

F5, F6, F8, F9, F10, F11 in brief

- **F5 Images:** product heroes are 1.6 MB PNGs served at one size to every device. WebP conversion + responsive sizes via the CDN: 1 day, −0.6 s LCP on product pages (measured on a converted sample).
- **F6 Indexers:** switched to “Update on Save” during a bulk-import incident in February and never switched back. Admin product saves take 30 s, and the mismatch between storefront stock and warehouse reality traces to reindex races. Restore scheduled indexing: 0.5 day.
- **F8 Varnish grace:** every cache flush currently sends the full traffic wave to PHP. Grace mode serves stale objects while the cache refills: 0.5 day.
- **F9 Duplicate observers:** two extensions subscribe to the same checkout event; one has been abandoned by its vendor since 2023 and re-fetches the quote on every call. Remove it (its feature is unused): 1 day including verification, −300 ms on add-to-cart.

- **F10 Database housekeeping:** the database carries 41 GB of `catalogrule_product__temp*` tables (leftovers of interrupted indexer runs on an older Magento version, which never cleaned them up) plus a 12-million-row custom queue table with no retention. One `information_schema` query finds them:

```
mysql> SELECT COUNT(*) tables, ROUND(SUM(data_length+index_length)
    -> /1024/1024/1024, 1) gb FROM information_schema.tables
    -> WHERE table_name LIKE 'catalogrule\_product\_\_temp%';
+--------+------+
| tables | gb   |
+--------+------+
|     96 | 41.2 |
+--------+------+
```

  Dropping the leftovers and adding pruning: 0.5 day. Backups shrink from 68 GB to 27 GB and restore time roughly halves, which matters most on the day a restore is actually needed.

- **F11 Admin order grid:** a custom column added for the fulfillment team filters on an unindexed attribute; the grid takes 12 s to load and occupies a worker each time. One index migration: 0.5 day.

5. The plan, sorted by return on effort

Order	Items	Effort	What you get
1	F2 + F8	1 day	Every uncached request –400 ms; cache flushes no longer cause a load spike
2	F1	1–2 days	Catalog TTFB –700 ms; deploys stop requiring a Varnish restart
3	F6 + F10 + F11	1.5 days	Admin usable again; stock drift stops; backups halve
4	F7	2 days	Checkout no longer fails at peak
5	F5 + F9	2 days	Product-page LCP –0.9 s combined
6	F3	3–5 days	Checkout p95 under 2 s during campaigns
7	F4 short-term	2 days	LCP –0.4 s while the storefront decision is made
8	F4 rebuild	separate proposal	LCP permanently in the green range

Rows 1–6 total 10–14 engineering days and address every failure a customer can feel. Stop after any row and the work up to it stands on its own. Row 6 is the only one that touches order-path architecture; whoever executes it, have the design reviewed by a senior engineer.

Re-measure after row 6 with the same field-data query this report started from. If the numbers do not move as predicted, contact us and we will determine the cause.

6. What we did not review

Security posture, SEO, accessibility, and code quality outside the profiled hot paths are out of scope for this engagement. The audit covers the default store view; the B2B store view shares the stack but was not separately measured. One access limitation: our audit user could not read the Varnish admin socket during the window, so Varnish runtime counters come from `varnishstat` snapshots exported by the hosting provider rather than from live inspection.

7. Next steps

The fix plan can be executed by any competent team, including your own: every finding names the change, not just the symptom. If you want us to execute it, rows 1–6 fit a two-to-three-week fixed-price engagement under our standard engagement terms.

Questions about any finding: info@zapolu.com, or book a call and bring your own numbers; we will advise during the call whether an audit is warranted.