Zapolu

Performance-Audit: Deep Dive (Beispielbericht)

> **Dies ist ein Muster.** „Acme Outdoor s.r.o.“ ist ein fiktiver Kunde. Die Befunde sind aus echten Audits zusammengesetzt, die wir für andere Shops geliefert haben, mit veränderten Details und Zahlen. Struktur und Detailtiefe entsprechen dem, was ein zahlender Kunde erhält.

Kunde	Acme Outdoor s.r.o. (fiktiv)
Shop	acme-outdoor.example (Magento 2.4.6, Luma-Frontend)
Stack	Varnish 7, Redis 7, MySQL 8, 2 App-Nodes hinter einem Load Balancer
Datenfenster	28 Tage Felddaten, 14 Tage APM-Traces, ein Lasttest
Werkzeuge	New Relic, CrUX, Lighthouse, k6, varnishstat, MySQL Slow Log
Autor	Luboš Zápotočný, Zapolu s.r.o.
Engagement	Deep Dive, 7 Arbeitstage, Festpreis

Für einen kurzen Überblick lesen Sie Abschnitt 1 und Abschnitt 5. Alles dazwischen ist die Beweisführung.

1. Zusammenfassung

Die Langsamkeit des Shops entsteht serverseitig, nicht durch unterdimensionierte Hardware. Das Feld-LCP liegt auf Katalogseiten bei **4,6 s (p75)** (Ziel: ≤ 2,5 s), getrieben von einer **TTFB von 1,9 s**. Die TTFB ist hoch, weil der Full-Page-Cache faktisch deaktiviert ist: Varnish beantwortet nur **34 %** der Katalog-Requests, und 38 % der gesamten Serverzeit entfallen auf das Rendern von Produktseiten, im Schnitt 2,2 s pro Request. Am Kampagnenabend im Mai erreichte der Checkout **6,1 s (p95)** und **3,4 % der Requests schlugen fehl**, während die CPU der Applikationsserver nie über 12 % kam.

Es folgen elf Befunde. Sechs davon kosten einen Arbeitstag oder weniger, und zwei dieser sechs betreffen nur Konfigurationsdateien. Die Zeilen 1–5 des Fix-Plans sollten das Katalog-LCP unter 2,5 s (p75) bringen und die Checkout-Ausfälle in der Spitze stoppen; Zeile 6 bringt anschließend das Checkout-p95 während Kampagnen unter 2 s. Die übrigen Zeilen sind nach Ertrag sortiert; der Plan lässt sich an jeder Stelle abschneiden.

In Geld ausgedrückt: In den drei Kampagnenstunden am 12. Mai endeten rund 230 Checkouts mit einem Fehler. Beim durchschnittlichen Bestellwert von 1 800 CZK (≈ 72 €) sind das etwa 415 000 CZK an versuchten Bestellungen, die fehlschlugen, noch vor den Kunden, die den Checkout nach sechs Sekunden Wartezeit abgebrochen haben. Die übrigen Befunde geben wir in Millisekunden an; eine Umsatzzahl wäre für sie spekulativ, daher lassen wir sie weg.

2. Was wir gemessen haben, und wie

- **Real-User-Metriken:** Core Web Vitals aus einer 28-Tage- CrUX-Abfrage, segmentiert nach Seitentyp (Home / Kategorie / Produkt / Checkout). Synthetische Lighthouse-Läufe dienten nur der Reproduktion von Befunden, nicht deren Bewertung. Wo der Shop gegenüber Googles Schwellwerten steht:

Metrik (p75, Felddaten)	Shop	Schwelle	Urteil
LCP	4,6 s	< 2,5 s	fällt durch
INP	280 ms	< 200 ms	fällt durch
CLS	0,04	< 0,1	besteht
FCP	2,9 s	< 1,8 s	fällt durch
TTFB	1,9 s	< 0,8 s	fällt durch

- **Server-seitiges Tracing:** New-Relic-Transaktionsprofil über 14 Tage, das zeigt, wohin die Serverzeit wirklich geht:

Transaktion	Anteil an der Serverzeit	Ø Antwort
`catalog/product/view`	38 %	2,2 s
`catalog/category/view`	17 %	1,8 s
Checkout-REST (Warenkorb + Versand)	11 %	1,4 s (6,1 s p95 in der Spitze)

  Dazu das MySQL Slow Query Log, das Profil externer Dienste und Varnish-Hit/Miss-Statistiken je URL-Muster aus `varnishstat`.

- **Lasttest:** ein Replay des Traffic-Verlaufs der Kampagne vom 12. Mai (0 → 420 req/s über 20 Minuten) gegen Staging, mit demselben Cache-Zustand wie in Produktion.
- **Konfigurations-Review:** PHP- und OPcache-Einstellungen, Composer-Autoloader, `env.php`, die Varnish-VCL und MySQL-Einstellungen, jeweils gegen Magentos dokumentierte Empfehlungen.
- **Bundle- und Asset-Audit:** was beim ersten Aufruf einer Produktseite ausgeliefert wird, was das Rendern blockiert und was unnötige Last ist.

Die meisten Befunde zitieren den exakten Befehl oder die exakte Query, die wir verwendet haben: einerseits, damit Ihr Team unsere Arbeit nachprüfen kann, andererseits, weil dieselben Prüfbefehle auf jedem Magento-Shop laufen, auch auf Ihrem. Zu jeder Zahl unten steht, woher sie kommt; wo die Wirkung eines Befunds eine Schätzung ist, ist sie so gekennzeichnet.

3. Befunde im Überblick

Hoch: Kunden sind betroffen, oder Umsatz hängt davon ab. **Mittel**: das Team ist täglich betroffen, oder Daten werden still beschädigt. **Niedrig**: kleiner Aufwand, der ein reales Betriebsrisiko beseitigt.

#	Befund	Schweregrad	Aufwand	Erwartete Wirkung
F1	Full-Page-Cache ist faktisch deaktiviert und lässt sich nicht invalidieren	Hoch	1-2 Tage	–700 ms TTFB (p75) im Katalog
F2	PHP-Runtime und Autoloader für Magento falsch konfiguriert	Hoch	0,5 Tage	–400 ms bei jedem ungecachten Request
F3	Checkout blockiert auf einem synchronen ERP-Bestandsabruf	Hoch	3–5 Tage	–2,8 s Checkout-p95 in der Spitze; keine Timeout-Fehler mehr
F4	Frontend liefert 2,3 MB JavaScript aus, renderblockierend	Hoch	Projekt	–1,2 s LCP (Zwischenlösungen: –0,4 s)

#	Befund	Schweregrad	Aufwand	Erwartete Wirkung
F5	Produktbilder: 1,6-MB-PNGs, keine responsiven Größen	Mittel	1 Tag	–0,6 s LCP auf Produktseiten
F6	Indexer auf „Update on Save“ belassen	Mittel	0,5 Tage	Admin-Speichern 30 s → ~2 s; Bestandsdrift endet
F7	PHP-FPM-Pool durch Session-Locks in der Spitze erschöpft	Hoch	2 Tage	Checkout übersteht Kampagnen-Traffic
F8	Kein Grace-Modus in Varnish	Niedrig	0,5 Tage	Fängt die Welle nach jedem Cache-Flush ab
F9	Zwei Extensions beobachten dasselbe Event; eine ist verwaist	Mittel	1 Tag	–300 ms beim In-den-Warenkorb-Legen
F10	41 GB tote Indexer-Temp-Tabellen in MySQL	Niedrig	0,5 Tage	Backups 60 % kleiner; Restore etwa halb so lang
F11	Admin-Bestellgrid filtert auf einer unindizierten Spalte	Niedrig	0,5 Tage	Bestellgrid 12 s → unter 1 s

4. Befunde im Detail

F1: Full-Page-Cache ist faktisch deaktiviert und lässt sich nicht invalidieren (Hoch)

Was wir sahen. Varnish meldet auf Katalog-URLs eine Hit-Rate von 34 %. Für einen Katalog dieser Größe, bei diesem Crawler-Anteil, sind über 90 % normal. Jede Kategorie- und Produktseite trägt `X-Magento-Cache-Debug: MISS`. Es gibt zwei unabhängige Ursachen:

1. Eine angepasste `default.xml` markiert einen Header-Block (einen Store-Switcher, der die Kunden-Session liest) als `cacheable="false"`. In Magento macht ein einziger uncachebarer Block im Layout die **gesamte Seite** uncachebar. Der Block kam mit einem Extension-Update im November; die Hit-Rate brach in derselben Woche ein.
2. Varnish ist im Admin aktiviert (`system/full_page_cache/caching_application = 2`), aber in der `env.php` fehlt die Sektion `http_cache_hosts`; Magento kann also gar keine Purge-Requests an Varnish senden. Das Team behilft sich mit einem Varnish-Neustart nach jedem Deploy, und genau deshalb sind Deploys problematisch (siehe F8).

So haben wir es verifiziert:

```
$ curl -sI https://acme-outdoor.example/zelte | grep -i cache-debug
X-Magento-Cache-Debug: MISS
# auf jeder geprüften Katalog-URL

$ grep -R 'cacheable="false"' app/design app/code | wc -l
1
# der Store-Switcher-Block, default.xml

$ php -r 'var_export(array_key_exists("http_cache_hosts", (include "app/etc/env.php")));'
false
# Magento kann Varnish nicht purgen
```

Fix. Den Switcher über eine Private-Content-Sektion rendern (Customer-Data-JS), damit die Seite cachebar bleibt (das dokumentierte Muster für session-abhängige Fragmente), und `http_cache_hosts` in `env.php` ergänzen, damit die Tag-basierte Invalidierung funktioniert und die Neustarts enden können.

Aufwand / Wirkung / Risiko. 1-2 Tage inklusive Regressionstests. Erwartet: –700 ms TTFB (p75) auf Katalogseiten (gemessen: gecachte Katalogantworten in 80 ms vs. 1,9 s ungecacht). Risiko: gering; beide Änderungen sind begrenzt und pro Deploy reversibel.

F2: PHP-Runtime und Autoloader für Magento falsch konfiguriert (Hoch)

Was wir sahen. Die Codebase umfasst ~125 000 PHP-Dateien; OPcache ist auf 10 000 konfiguriert. Der Composer-Autoloader wurde nie optimiert: `vendor/composer/autoload_static.php` enthält keine Class-Map, und die New-Relic-Traces zeigen ganze Sekunden in der `ObjectManager`-Factory-Auflösung auf kalten Pfaden. Aktuell versus empfohlen:

Einstellung	Aktuell	Empfohlen
`opcache.max_accelerated_files`	10 000	100 000
`opcache.memory_consumption`	128	512
`opcache.validate_timestamps`	On, Revalidierung alle 2 s	Off; Reset beim Deploy
`opcache.enable_cli`	Off	On (Indexer und Crons sind auch PHP)
`realpath_cache_size`	4M	32M
`realpath_cache_ttl`	120	7200
Composer-Autoloader	nicht optimiert	`dump-autoload --optimize` im Build

So haben wir es verifiziert:

```
$ find . -type f -name '*.php' | wc -l
124862
# PHP-Dateien in der Codebase

$ php -i | grep 'opcache.max_accelerated_files'
opcache.max_accelerated_files => 10000
# fasst 8 % davon

$ grep -c 'classMap' vendor/composer/autoload_static.php
0
# Autoloader wurde nie optimiert
```

Warum. Mit diesen Werten prüft und kompiliert PHP laufend einen großen Teil der Codebase neu; die Profile schreiben über die Hälfte der ungecachten Renderzeit dem Klassenladen und Seitenaufbau zu, nicht der Geschäftslogik.

Fix. Eine Konfigurationsänderung plus ein Build-Schritt. Die einzige Kopplung, die zu respektieren ist: Mit abgeschaltetem `validate_timestamps` muss die Deploy-Pipeline beim Release den OPcache zurücksetzen. Den genauen Schritt für Ihre aktuelle Pipeline haben wir im Fix-Plan dokumentiert.

Aufwand / Wirkung / Risiko. 0,5 Tage. Geschätzt –400 ms bei jedem ungecachten Request (und schnellere Crons und Indexer als Nebeneffekt). Risiko: gering, sofern der Deploy-Schritt zusammen mit der Konfiguration landet.

F3: Checkout blockiert auf einem synchronen ERP-Bestandsabruf (Hoch)

Was wir sahen. Im Versandschritt zeigt der Trace einen synchronen HTTPS-Call auf den Bestands-Endpoint des ERP mit 3-Sekunden-Timeout und einem Retry; das Profil externer Dienste schreibt 92 % der gesamten externen Wartezeit diesem einen Endpoint zu. Unter Kampagnenlast wird das ERP langsam, die Calls schöpfen ihren Timeout voll aus, und Checkout-Requests stauen sich. Genau daher kommen die 6,1 s p95 und die 3,4 % Fehler. Der Lasttest reproduziert es auf Staging.

Warum. Die Echtzeit-Bestandsbestätigung kam nach einem Überverkaufs-Vorfall im letzten Jahr. Die Absicht ist richtig, nur legt sie die Latenz eines Drittsystems in jeden einzelnen Checkout.

Fix. Den Bestandsabgleich aus dem Request-Pfad nehmen: gegen den lokalen Bestand reservieren, asynchron bestätigen, bei Abweichung alarmieren. Der Schutz vor Überverkauf bleibt erhalten, und der Checkout wartet nicht mehr auf das ERP. (Es ist dasselbe Muster, das wir in unserem Text über den ERP/Shop-Bestandsabgleich beschreiben; eine Referenz stellen wir auf Anfrage bereit.)

Aufwand / Wirkung / Risiko. 3–5 Tage inklusive Abweichungs-Alarm. Entfernt ~2,8 s aus dem Checkout-p95 in der Spitze und die Timeout-Fehler vollständig (beides im Lasttest mit gestubbtem Call gemessen). Risiko: mittel; es braucht eine schriftliche Freigabe des neuen Überverkaufs-Fensters (geschätzt < 0,1 % der Kampagnenbestellungen, gegenüber heute 3,4 % fehlschlagender Checkouts).

F4: Frontend liefert 2,3 MB JavaScript aus (Hoch)

Was wir sahen. Der erste Aufruf einer Produktseite lädt 2,3 MB JavaScript in 61 Dateien; die Main-Thread-Blockierung liegt auf einem Mittelklasse-Telefon bei 2,9 s. Hinter vier der vierzehn Marketing-Tags steht kein funktionierendes Konto mehr.

Warum. Teils die Luma-Frontend-Architektur, teils acht Jahre Tag-Zuwachs; kein einzelner Tag dominiert die Summe.

Fix, kurzfristig (dieses Quartal). Die vier toten Tags entfernen, das Analytics-Bundle verzögern, das Bewertungs-Widget unterhalb des Folds lazy laden. Geschätzt –0,4 s LCP, 2 Tage.

Der richtige Fix (separate Entscheidung). Ein Storefront-Neubau auf Hyvä. Auf vergleichbaren Katalogen bringt das das LCP auf p75 in den grünen Bereich und senkt die Frontend-Wartungskosten; es ist ein Projekt von 6–10 Wochen und verdient ein eigenes, sauber geschnittenes Angebot, keine Zeile in einer Fix-Liste. Wir nennen es hier, damit die kurzfristige Schätzung oben nicht als das Maximum gelesen wird, das erreichbar ist.

F7: PHP-FPM-Pool durch Session-Locks in der Spitze erschöpft (Hoch)

Was wir sahen. Im Lasttest beginnen die Fehler bei ~280 req/s, während die CPU bei 12 % sitzt. Die FPM-Statusseite zeigt alle Worker belegt; die Traces zeigen Warten auf Redis-Session-Locks: Kunden, die die Bestellbestätigungsseite pollen, halten Locks, die jeden weiteren Request derselben Session in die Warteschlange zwingen.

Fix. Session-Locking für die Polling-Endpoints deaktivieren, die Worker-Zahl auf das anheben, was der Speicher-Headroom real erlaubt (die aktuelle Zahl stammt von einem kleineren Instanztyp), und einen FPM-Sättigungsalarm ergänzen; dieser Ausfall blieb unbemerkt, weil die CPU-Graphen unauffällig aussahen.

Aufwand / Wirkung / Risiko. 2 Tage inklusive Wiederholung des Lasttests. Zusammen mit F3 übersteht Staging 420 req/s mit einem Checkout-p95 von 1,9 s. Risiko: gering.

F5, F6, F8, F9, F10, F11 kurz

- **F5 Bilder:** die Produkt-Heroes sind 1,6-MB-PNGs, in einer einzigen Größe an jedes Gerät ausgeliefert. WebP-Konvertierung + responsive Größen über das CDN: 1 Tag, –0,6 s LCP auf Produktseiten (an einer konvertierten Stichprobe gemessen).
- **F6 Indexer:** beim Massenimport im Februar auf „Update on Save“ umgestellt und nie zurück. Admin-Produktspeichern dauert 30 s, und die Abweichung zwischen Shop-Bestand und Lagerrealität geht auf Reindex-Races zurück. Geplantes Indexieren wiederherstellen: 0,5 Tage.
- **F8 Varnish Grace:** jeder Cache-Flush schickt heute die volle Traffic-Welle auf PHP. Der Grace-Modus liefert veraltete Objekte aus, bis der Cache wieder gefüllt ist: 0,5 Tage.
- **F9 Doppelte Observer:** zwei Extensions abonnieren dasselbe Checkout-Event; eine ist seit 2023 vom Anbieter verwaist und lädt bei jedem Aufruf das Quote neu. Entfernen (ihr Feature ist ungenutzt): 1 Tag inklusive Verifikation, –300 ms beim In-den-Warenkorb-Legen.
- **F10 Datenbank-Hausputz:** die Datenbank trägt 41 GB `catalogrule_product__temp*`-Tabellen (Überbleibsel abgebrochener Indexer-Läufe aus einer älteren Magento-Version, die nie aufräumte) plus eine eigene Queue-Tabelle mit 12 Millionen Zeilen ohne Retention. Eine `information_schema`-Query findet sie:

  ```
  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 |
  +--------+------+
  ```

  Überbleibsel löschen und Pruning ergänzen: 0,5 Tage. Backups schrumpfen von 68 GB auf 27 GB, die Restore-Zeit halbiert sich ungefähr, was vor allem an dem Tag zählt, an dem ein Restore wirklich gebraucht wird.
- **F11 Admin-Bestellgrid:** eine für das Fulfillment-Team ergänzte Spalte filtert auf einem unindizierten Attribut; das Grid lädt 12 s und bindet dabei jedes Mal einen Worker. Eine Index-Migration: 0,5 Tage.

5. Der Plan, sortiert nach Ertrag pro Aufwand

Reihenfolge	Punkte	Aufwand	Was Sie bekommen
1	F2 + F8	1 Tag	Jeder ungecachte Request –400 ms; Cache-Flushes verursachen keine Lastspitze mehr
2	F1	1–2 Tage	Katalog-TTFB –700 ms; Deploys brauchen keinen Varnish-Neustart mehr
3	F6 + F10 + F11	1,5 Tage	Admin wieder benutzbar; Bestandsdrift endet; Backups halbiert
4	F7	2 Tage	Checkout fällt in der Spitze nicht mehr aus
5	F5 + F9	2 Tage	Produktseiten-LCP –0,9 s zusammen
6	F3	3–5 Tage	Checkout-p95 unter 2 s während Kampagnen

Reihenfolge	Punkte	Aufwand	Was Sie bekommen
7	F4 kurzfristig	2 Tage	LCP −0,4 s, bis die Storefront-Entscheidung fällt
8	F4 Neubau	separates Angebot	LCP dauerhaft im grünen Bereich

Die Zeilen 1–6 ergeben zusammen 10–14 Engineering-Tage und beheben jeden Ausfall, den ein Kunde spüren kann. Sie können nach jeder Zeile aufhören; die Arbeit bis dahin steht für sich. Zeile 6 ist die einzige, die die Architektur des Bestellpfads berührt. Wer auch immer sie umsetzt: Lassen Sie das Design von einem erfahrenen Engineer prüfen.

Messen Sie nach Zeile 6 mit derselben Felddaten-Abfrage nach, mit der dieser Bericht begonnen hat. Bewegen sich die Zahlen nicht wie vorhergesagt, kontaktieren Sie uns, und wir ermitteln die Ursache.

6. Was wir nicht geprüft haben

Sicherheitslage, SEO, Barrierefreiheit und Codequalität außerhalb der profilierten heißen Pfade liegen außerhalb des Umfangs dieses Engagements. Das Audit deckt die Standard-Store-View ab; die B2B-Store-View teilt den Stack, wurde aber nicht separat vermessen. Eine Zugriffs-Einschränkung: Unser Audit-Benutzer konnte im Messfenster den Varnish-Admin-Socket nicht lesen; die Varnish-Laufzeitzähler stammen daher aus `varnishstat`-Snapshots des Hosters statt aus Live-Inspektion.

7. Nächste Schritte

Den Fix-Plan kann jedes kompetente Team umsetzen, auch Ihr eigenes: jeder Befund benennt die Änderung, nicht nur das Symptom. Wenn wir ihn umsetzen sollen: Die Zeilen 1–6 passen in ein zwei- bis dreiwöchiges Festpreis-Engagement zu unseren üblichen Engagement-Bedingungen.

Fragen zu einzelnen Befunden: info@zapolu.com, oder vereinbaren Sie ein Gespräch und bringen Sie Ihre eigenen Zahlen mit; im Gespräch sagen wir Ihnen, ob sich ein Audit lohnt.