

Die Shopware *Performance*-Checkliste

16 Einstellungen, die einen Shopware-6-Shop schneller machen, jeweils mit exakter Konfiguration und dem, was dabei kaputtgehen kann.

Geprüft gegen Shopware 6.7.14.0 und die Shopware-Entwicklerdokumentation, September 2026.

Prüfbogen

#	ÄNDERUNG	WO	ERLEDIGT
CACHE			
01	HTTP-Cache einschalten	Konfiguration	☐
02	App- und System-Cache nach Redis verschieben	Konfiguration	☐
03	Ausstehende Cache-Invalidierungen in Redis halten	Konfiguration	☐
04	Warenkorb und Cache mit zstd komprimieren	Server	☐
QUEUE UND CRON			
05	Admin-Worker abschalten und echte Worker betreiben	Server	☐
06	Scheduled Tasks per Cron starten	Server	☐
07	E-Mails über die Queue versenden	Konfiguration	☐
08	Mail-Variablen nicht bei jedem Versand schreiben	Konfiguration	☐
09	Sitemap per Cron erzeugen	Server	☐
KATALOG UND STOREFRONT			
10	Reduzierte Produktdaten in Listings laden	Administration	☐
11	Product-Stream-Indexer abschalten	Konfiguration	☐
12	Speculation Rules API einschalten	Administration	☐
13	Suche nicht mehr auf MySQL zurückfallen lassen	Konfiguration	☐
PHP UND DEPLOYMENT			
14	OPcache für Produktion einstellen	Server	☐
15	Umgebung nach .env.local.php dumpen	Deployment	☐
16	Composer-Classmap autoritativ machen	Deployment	☐

1. Prüfen

Haken Sie im Prüfbogen unten ab, was Ihr Shop schon hat. Die leeren Kästchen sind Ihre To-do-Liste.

2. Auf Staging testen

Ändern Sie einen Punkt nach dem anderen, nie direkt in Produktion.

3. Messen

Vergleichen Sie Antwortzeiten und Serverlast vor und nach jeder Änderung.

Cache

Mehr Anfragen ohne PHP beantworten und klein halten, was im Cache liegt.

01 HTTP-Cache einschalten ☐

KONFIGURATION

Gecachte Seiten laufen gar nicht erst durch PHP. In Shops, in denen der Cache aus ist, ist das meist der größte einzelne Gewinn.

```
# .env.local
SHOPWARE_HTTP_CACHE_ENABLED=1
```

ACHTUNG Der eingebaute Cache liegt weiter im App-Cache. Bei echtem Traffic gehört Varnish oder Fastly davor und hält die Seiten.

02 App- und System-Cache nach Redis verschieben ☐

KONFIGURATION

AB 6.5.8.3

Der Datei-Cache wird unter Last langsam und lässt sich nicht zwischen App-Servern teilen. Redis kann beides.

```
# config/packages/cache.yaml
framework:
  cache:
    app: cache.adapter.redis_tag_aware
    system: cache.adapter.redis_tag_aware
    default_redis_provider: redis://localhost
```

ACHTUNG Verwaiste Tags sammeln sich an und der Speicher in Redis wächst weiter. Planen Sie ein regelmäßiges Aufräumen ein, zum Beispiel mit `frish:redis-tag:cleanup` aus FroshTools.

03 Ausstehende Cache-Invalidierungen in Redis halten



KONFIGURATION

Standardmäßig landen sie in einer MySQL-Tabelle. In einem gut besuchten Shop mit vielen gleichzeitigen Schreibzugriffen kann diese Tabelle Deadlocks auslösen.

```
# config/packages/shopware.yaml
shopware:
  redis:
    connections:
      ephemeral:
        dsn: 'redis://localhost:6379/0'
  cache:
    invalidation:
      delay_options:
        storage: redis
        connection: 'ephemeral'
```

ACHTUNG Die Verbindung `ephemeral` muss wie oben unter `shopware.redis.connections` definiert sein. Sonst hat die Einstellung nichts, womit sie sich verbinden kann.

04 Warenkorb und Cache mit zstd komprimieren



SERVER AB 6.6.4.0

Kleinere Warenkorb- und Cache-Einträge brauchen weniger Speicher in Redis und bewegen pro Anfrage weniger Daten. Die Cache-Komprimierung ist schon standardmäßig an, hier ändern Sie die Methode.

```
shopware:
  cart:
    compress: true
    compression_method: zstd
  cache:
    compression_method: zstd
```

ACHTUNG Installieren Sie zuerst die PHP-Erweiterung `zstd`. Sie ist nicht in PHP enthalten. Vor 6.7.12.0 heißt der Cache-Schlüssel `cache_compression_method`. Leeren Sie nach der Umstellung den Cache.

Queue und Cron

Hintergrundarbeit gehört zu Workern und Cron, nicht zu dem, der gerade die Administration offen hat.

05 Admin-Worker abschalten und echte Worker betreiben



SERVER

Der Admin-Worker arbeitet die Queue nur ab, solange jemand die Administration offen hat. Wird der Tab geschlossen, stehen Indexierung, Mails und Importe still.

```
# config/packages/shopware.yaml
shopware:
  admin_worker:
    enable_admin_worker: false

# systemd or supervisor
bin/console messenger:consume async low_priority --time-limit=60 --memory-limit=512M
```

ACHTUNG Betreiben Sie zusätzlich einen Worker für den Transport `failed`. Behalten Sie die Zeit- und Speicherlimits bei, damit systemd oder supervisor den Prozess sauber neu startet.

06 Scheduled Tasks per Cron starten



SERVER

Aufräumjobs und wiederkehrende Indexierung starten pünktlich, statt zu warten, bis ein Worker Zeit hat.

```
*/5 * * * * php /var/www/html/bin/console scheduled-task:run --no-wait
```

ACHTUNG Richten Sie das Cron-Intervall nach Ihrem kürzesten Task-Intervall. Alle 5 Minuten ist ein vernünftiger Start.

07 E-Mails über die Queue versenden



KONFIGURATION

Der Checkout wartet nicht mehr auf den Mailserver. Die Bestellbestätigung geht Sekunden später über einen Worker raus.

```
# config/packages/prod/framework.yaml
framework:
  mailer:
    message_bus: 'messenger.default_bus'
```

ACHTUNG Erst umstellen, wenn Punkt 05 läuft. Ohne Worker bleiben Mails in der Queue liegen und kommen nie an.

08 Mail-Variablen nicht bei jedem Versand schreiben ☐

KONFIGURATION

Jede versendete Mail speichert zusätzlich Beispieldaten für den Template-Editor. Abgeschaltet fällt pro Bestellung ein Schreibzugriff weg.

```
shopware:
  mail:
    update_mail_variables_on_send: false
```

ACHTUNG Der Mail-Template-Editor in der Administration hat danach keine Beispieldaten mehr für die Autovervollständigung.

09 Sitemap per Cron erzeugen ☐

SERVER AB 6.7.1.0

Bei großen Katalogen ist die Sitemap ein schwerer Job. Lassen Sie ihn nachts laufen statt in der Scheduled-Task-Queue.

```
shopware:
  sitemap:
    scheduled_task:
      enabled: false

# crontab, nightly
0 3 * * * php /var/www/html/bin/console sitemap:generate
```

ACHTUNG Wer den Task abschaltet und den Cron-Eintrag vergisst, hat eine Sitemap, die unbemerkt nicht mehr aktualisiert wird.

Katalog und Storefront

Weniger Arbeit pro gespeichertem Produkt und pro Seite, die ein Kunde öffnet.

10 Reduzierte Produktdaten in Listings laden ☐

ADMINISTRATION AB 6.7.12.0

Listings laden nur, was die Standard-Produktbox zeigt, mit einem Klartext-Teaser von höchstens 512 Zeichen statt der vollen Beschreibung.

```
Settings > Products
"Load reduced product data in listings"
config key: core.listing.partialDataLoading
```

ACHTUNG Themes und Erweiterungen, die weitere Produktfelder im Listing lesen, zeigen dann Lücken. Neuinstallationen haben die Option an, bestehende Shops schalten sie pro Verkaufskanal ein.

11 Product-Stream-Indexer abschalten



KONFIGURATION

AB 6.6.10.0

Beim Speichern eines Produkts entfällt die Neuindexierung dynamischer Produktgruppen. Bei großen Importen summiert sich das schnell.

```
shopware:
  product_stream:
    indexing: false
```

ACHTUNG Kategorieseiten auf Basis dynamischer Produktgruppen aktualisieren sich erst, wenn der HTTP-Cache abläuft, und Regelbedingungen, die auf einen Artikel in einer dynamischen Produktgruppe prüfen, liefern immer false.

12 Speculation Rules API einschalten



ADMINISTRATION

AB 6.6.10.0

Sobald ein Kunde auf einen Link zeigt oder ihn antippt, rendert der Browser die nächste Seite vor. Der Klick landet auf einer Seite, die schon fertig ist.

```
Settings > System > Storefront
"Speculation rules API (Experimental)"
```

ACHTUNG Vorgerenderte Seiten, die nie geöffnet werden, kosten trotzdem Serverzeit und können Seitenaufrufe in Ihrer Analyse aufblähen.

13 Suche nicht mehr auf MySQL zurückfallen lassen



KONFIGURATION

Fällt OpenSearch oder Elasticsearch aus, sucht Shopware unbemerkt in MySQL weiter. Bei großen Katalogen kann dieser Fallback die Datenbank überlasten.

```
# .env.local
SHOPWARE_ES_THROW_EXCEPTION=1
```

ACHTUNG Suchausfälle werden jetzt als Fehler sichtbar. Richten Sie vorher ein Monitoring für den Such-Cluster ein.

PHP und Deployment

Einstellungen, die sich nur lohnen, wenn Ihr Deployment-Skript sie kennt.

14 OPcache für Produktion einstellen



SERVER

PHP prüft nicht mehr bei jeder Anfrage jede Datei auf Änderungen und hält mehr von Shopware kompiliert im Speicher.

```
; php.ini
opcache.validate_timestamps=0
opcache.max_accelerated_files=20000
opcache.interned_strings_buffer=20
zend.assertions=-1
realpath_cache_ttl=3600
```

ACHTUNG Ohne Timestamp-Prüfung sieht PHP neuen Code nicht von selbst. Laden Sie PHP-FPM bei jedem Deployment neu. Preloading bringt weitere 2 bis 5 %, verlangt aber nach jedem Cache-Leeren einen vollen FPM-Neustart und legt den Extension Manager lahm.

15 Umgebung nach .env.local.php dumpen



DEPLOYMENT

Shopware liest ein fertiges PHP-Array, statt bei jeder Anfrage die .env -Dateien zu parsen.

```
bin/console system:setup --dump-env
# or
bin/console dotenv:dump prod
```

ACHTUNG Die gedumpte Datei hat Vorrang vor allen .env -Dateien. Dumpen Sie bei jeder Änderung neu, am besten als Deployment-Schritt.

16 Composer-Classmap autoritativ machen



DEPLOYMENT

Der Autoloader schlägt Klassen in einer einzigen Map nach und durchsucht nie das Dateisystem.

```
// composer.json
"config": {
    "optimize-autoloader": true,
    "classmap-authoritative": true
}
```

ACHTUNG Nur, wenn alle Plugins über Composer installiert sind. Plugins in `custom/plugins`, die Composer nicht verwaltet oder die über den Extension Manager kamen, werden nicht mehr geladen.

BRAUCHEN SIE HILFE?

Wir machen diesen *Check* für Sie.

Etwa die Hälfte der Punkte dauert 5 Minuten. Der Rest, etwa Worker, Redis und OPcache, hängt von Ihrem Hosting ab und macht bei falscher Einrichtung unbemerkt etwas kaputt.

Schicken Sie uns Ihre Shop-URL und eine Zeile zu Ihrem Hosting. Wir antworten innerhalb von 24 Stunden mit dem, was wir zuerst ändern würden.

elixentdigital.com/de/kontakt

[30-Minuten-Gespräch buchen](#)

Quellen: Shopware-Entwicklerdokumentation (developer.shopware.com/docs/guides/hosting/performance/ und [/hosting/infrastructure/](https://developer.shopware.com/docs/guides/hosting/infrastructure/)) und der Quellcode von Shopware 6.7.14.0.