Profiling PHP code

Measuring and speeding up PHP code in Exponential - exp:benchmark for pages, kernel probes and micro benchmarks, eZDebug accumulators around your own code, the debug bar's SQL and memory figures, the OPcache profile…

Source file: `doc/eng-us/help/php-developers/profiling-php-code.md` Last changed 2026-11-04
https://doc.exponential.earth/exphelp/viewer/help/php-developers/profiling-php-code.md

This page is for PHP developers who make code faster, or who must prove a change did not make it slower. The method is always the same: measure, change one thing, measure again, under the same conditions. The application view is Profiling and performance; the full guide is Benchmarking Exponential.

Contents

1. Three levels of measurement
2. Pages: exp:benchmark
3. The kernel and micro benchmarks
4. Inside one request
5. OPcache, under any engine
6. The SQL query cache
7. Causes that recur
8. Measuring fairly

Three levels of measurement

Level	Tool	Answers
whole pages over HTTP	`exp:benchmark`	is the site faster or slower, and by how much
kernel operations in-process	`exp:benchmark kernel`	which part of a request costs (boot, render, INI, fetch, database, cache)
your code in one request	accumulators, the debug bar	where in your code the time goes

Pages: exp:benchmark

Shell

```
php bin/php/console exp:benchmark --save=var/benchmark/base.json
# change something
php bin/php/console exp:benchmark --baseline=var/benchmark/base.json --threshold=15
php bin/php/console exp:benchmark --cold --requests=50                # past the response caches
php bin/php/console exp:benchmark --compare=https://www.example.com,https://www.example.com:8080
```

It requests a set of pages many times after a warm-up and prints median, p90, p99 and more. `--baseline` with `--threshold` turns it into a regression check that fails when a page got slower

by more than the threshold (Benchmark).

The kernel and micro benchmarks

Shell

```
php bin/php/console exp:benchmark kernel --repeat=30 --save=var/benchmark/kernel.json
php bin/php/benchmark.php micro --baseline=tests/benchmark/baseline.json --all-checks
```

The kernel probes time boot, render (cold and warm), INI load, a content fetch, a database round trip and cache writes and reads in-process. The micro benchmark times pure PHP hot paths without HTTP or a database; CI's Performance workflow runs it against `tests/benchmark/baseline.json` on PHP 8.5 with OPcache off, as a warning.

Inside one request

Wrap the code you suspect in an accumulator:

PHP

```
eZDebug::accumulatorStart( 'acmereport_totals', 'acmereport_total', 'Monthly totals' );
$totals = acmeReportService::monthlyTotals( $month );
eZDebug::accumulatorStop( 'acmereport_totals' );
```

With debug output on, the bar shows:

- **Timing**: your accumulators with their total time and call count, next to the kernel's;
- **SQL**: the number of statements, their time and share of the page, slowest first (`SQLOutput=enabled`);
- **Memory**: peak and the steps that took the most;
- **Templates**: how many templates were used.

The summary turns figures yellow or red against `debugbar.ini` thresholds, which makes a new hot spot visible at a glance (The debug bar for PHP developers).

OPcache, under any engine

The kernel can write one line per request saying how the request used OPcache: hits and misses, the scripts served most, and included files that OPcache does not hold (each of those is compiled again whenever it is included). It is off unless the file `var/tmp/opcache_profile.on` exists; the lines go to `var/tmp/opcache_profile.log` (`ezpKernelWeb::writeOPcacheProfile()`). Use it to compare engines; remove the switch file afterwards (Velocity and the opcode cache).

The SQL query cache

`settings/querycache.ini` (`Mode=off` by default) caches the results of repeated read queries (in APCu where the process has it, else in files, by default). Measure with it on and off: a page that issues the same query many times gains, a page with unique queries does not. It does not apply to MongoDB (SQL query cache).

Causes that recur

Symptom	Usual cause	Fix
many queries per page	fetches in loops, `dataMap()` on every item of a list	fetch once; limit; a cache block
a list gets slower with the data	no limit, or a count of everything	page it; bounded counts
memory grows in a command	objects kept in the kernel's in-memory caches	`eZContentObject::clearCache()` between pages
a slow publish	heavy work in a workflow event or listener	a queue or a content job
slow after deploy	caches cleared	warm them (OPcache and cache preloading)
slow only under Velocity after a change	workers not restarted, or OPcache not revalidating	`exp:velocity deploy`

Measuring fairly

- Same machine, nothing else running, OPcache as in production, Xdebug off.
- Compare two servers by alternating requests, not by timing one after the other.
- Record the environment with each run; a comparison with a baseline from another setup says little.
- Keep the runs: commit a baseline file for your extension and compare against it before a release.

Related pages

- Benchmarking Exponential
- Performance in extensions
- Caching in applications
- Free documentation: this page is free to copy, translate and improve under the GNU FDL 1.3.

> **Become an Exponential Partner**
>
> Partners share their information, provide funding, contribute back and develop the products with us, and that is how Exponential grows. A partner website and dashboard are planned for mid-2027: please become a partner today to speed up our growth rate and land new features faster.
>
> Partners: start here | How to apply

Exponential Copyright © 1998-2026 Exponential Foundation and 7x
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".
This documentation is free documentation, licensed under the GNU Free Documentation License 1.3 or later:

https://www.gnu.org/licenses/fdl-1.3.html.