Performance

For CMS developers, how to make an Exponential site fast and keep it fast. It covers measuring with exp:benchmark, what the content model and templates decide, fetch rules, caching per layer, the server side (Velocity…

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

This page is for developers responsible for how fast a site is. Most of the speed of an Exponential site is decided by three things you control: how much each page fetches, how much of it can be cached, and how the site is served. Measure first, change one thing, measure again.

Contents

1. Measure
2. The content model and templates
3. Fetches
4. Caching
5. The server
6. Images and the front end
7. Warm caches
8. A short checklist
9. When it is slow in production

Measure

Shell

```
./console exp:benchmark --allow-root-user
./console exp:cache status --allow-root-user
curl -s -o /dev/null -w '%{time_total}s\n' https://www.example.com/
```

`exp:benchmark` requests the front page, menu pages, search and the administration login many times and prints median, p90 and p99 times; it can save runs and compare against a baseline, and also measures parts of a page inside the kernel (Benchmarking Exponential). The debug output lists the SQL queries and the time of a page on a development machine. A single page load proves nothing: the first request fills caches.

The content model and templates

- A **list page** that draws fifty items, each with relations fetched one by one, makes hundreds of queries. Model so that list views need only the item's own attributes; use `load_data_map` when you read attributes of many nodes.
- **Pagelayout fetches** run on every request and are not in the view cache: wrap menus and footers in cache blocks.

- **Uncacheable parts** (the visitor's name, the basket) should be small and kept out of the view cache (Caching for CMS developers).

Fetches

Always pass `limit`; filter by class in the fetch; count with `list_count`; avoid fetches inside loops; keep attribute filters on large trees to what you measured (Fetch performance, Fetching content in templates). Code in your extensions follows the same rules (Performance in extensions).

Caching

Layer	What to do
View cache	keep it on (`ViewCaching=enabled`); keep `full` cached; set `cache_ttl` only where needed
Cache blocks	around everything in the pagelayout that is the same for many visitors, with correct keys
SQL query cache	on by default; `./console exp:cache querycache status`
HTTP cache, Velocity response cache	whole pages for anonymous visitors; check with `exp:cache status`

Every layer: Caching layers.

The server

1. **Velocity** with the `qbix` engine keeps PHP workers loaded between requests and has a response cache; it is the recommended engine (Velocity persistent worker server).
2. **OPcache** on for PHP-FPM; Velocity passes its own opcode cache options (OPcache, Velocity and the opcode cache).
3. **Workers**: `velocity.ini [ServerSettings] Workers`, raised with the CPU count and measured after each change.
4. **The database**: an engine sized for the content; see Sizing memory, CPU and disk.

The order of these steps with commands: Make it fast.

Images and the front end

- Define image aliases for the sizes you show; print `width` and `height`; lazy-load images below the fold (Images and image aliases, Image performance).
- Load stylesheets and scripts through the packer, which bundles and caches them (The packer, Theme performance).
- Measure layout shift and the largest paint on real pages (Measuring CLS and LCP).

Warm caches

After a deployment or a cache clear, the first visitors pay for building caches. `exp:velocity deploy` requests warm-up pages, and the preloader requests a site's pages as an anonymous visitor would (**Setup** > **Preload**, `php bin/php/preload.php --apply`), listing broken links on the way (Preloading caches).

A short checklist

- ☐ Benchmarked before and after each change, with the same command.
- ☐ No page makes more queries than its content explains; the debug output was checked on the heaviest pages.
- ☐ Every list fetch has a limit; no fetches in loops.
- ☐ Pagelayout menus and footers are in cache blocks with the right keys.
- ☐ The view cache is on; visitor-specific output is outside it.
- ☐ Served by Velocity (or another tuned server) with OPcache.
- ☐ Image aliases for every shown size; assets through the packer.

When it is slow in production

Follow the runbook Slow site: check the load, the cache layers, the request logs and the slowest pages before changing anything (Request logs).

Related pages

- Checklist: performance
- Benchmarking a hosted site
- Deploying a site
- 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.