Information technology

Sosnytskyi Oleksandr

Senior Full Stack Developer, Knot

(New York, USA)

INVESTIGATION OF THE EFFECTIVENESS OF LARAVEL-BASED WEB APPLICATIONS IN VARIOUS PHP EXECUTION ENVIRONMENTS

Summary. *The article is dedicated to the study of the efficiency of web applications built on the Laravel framework in different PHP runtime environments, such as PHP-FPM, Apache/mod_php, and Nginx. It also covers high-performance solutions based on long-lived processes, like Swoole and RoadRunner, as well as Laravel Octane. A comparative analysis of the architectural features, startup costs associated with the framework, and the impact of different server configurations on bandwidth and HTTP request processing delays was conducted. Based on the experimental data from synthetic tests and real-world cases, recommendations were formulated for selecting a runtime environment and optimizing Laravel apps for different workload scenarios.*

Key words: *Laravel, PHP-FPM, Swoole, RoadRunner, Laravel Octane, web application performancee, long-lived processes, optimization, throughput, load testing.*

Relevance of the study

The relevance of the research topic stems from the widespread use of the Laravel framework in the PHP ecosystem and the growing demand for performance and reliability in web applications. With the development of PHP versions 7.4-8.x and JIT compilation, as well as the emergence of alternative runtimes such as PHP-FPM, RoadRunner, and Swoole, developers have been faced with a challenge in choosing the optimal server-side configuration for their applications. This has led to a need for a deeper understanding of best practices and optimization techniques, which is what this research aims to address.

In practice, different execution environments can behave differently when dealing with different types of loads, architectural solutions, and methods of using caches, queues, and databases. Choosing the wrong environment can lead to an increase in response time, deterioration of user experience and increased infrastructure costs.

At the same time, there are still not many comprehensive comparative studies on the effectiveness of Laravel applications in different PHP runtime environments. Under these circumstances, it seems relevant to conduct a systematic study on the performance of Laravel applications in various PHP environment configurations, both for the theory of web system performance and the practice of developing and maintaining modern information systems. This will provide a basis for sound recommendations for developers, architects, and DevOps specialists.

The purpose of the study

The aim of this research is to assess the performance of Laravel applications in different PHP runtime environments, and to identify the impact of architectural choices and process models on performance, response times, and application stability under heavy load. Additionally, we aim to provide practical recommendations for selecting the optimal server configuration to ensure optimal performance and stability.

Materials and research methods

The study used materials from official documentation of Laravel, PHP, Swoole, RoadRunner, and Nginx, as well as the results of load tests published in open sources and data from practical performance tests (wrk and Apache Benchmark).

The research was based on a set of general scientific and specialized approaches. Theoretical analysis of scientific, technical literature, official documentation, and practical cases was conducted, as well as comparative and systematic analysis methods.

The results of the study

Laravel is a web application development framework based on PHP, with a model-view-controller (MVC) architecture and a well-designed service infrastructure. It includes a dependency injection container, a routing system, middleware, an object-relational mapping tool called Eloquent, and a template engine called Blade. The internal organization of these components determines how HTTP requests are processed and how different parts of the code execute with each request. This also determines where overhead costs occur, which can affect response time and bandwidth usage.

In terms of the query lifecycle in Laravel, the main file is public/index.php, which connects the Composer autoloader and initializes the application instance through bootstrap/app.php. After that, the request is sent to the HTTP core, where the middleware pipeline and global and route intermediaries are executed. Then, control is transferred to a controller or route handler, and the generated response passes through the middleware stack before being returned to the web server. The official Laravel documentation emphasizes the importance of understanding this sequence for targeted optimization. Delays can occur at each stage of the process, from configuration loading to view generation, due to redundant I/O operations, object reinitialization, or inefficient database queries [6].

The figure below shows the typical sequence of events when an HTTP request is processed in Laravel, starting with the startup of the application, the initialization of service providers, the routing process, and finally the generation of a response.

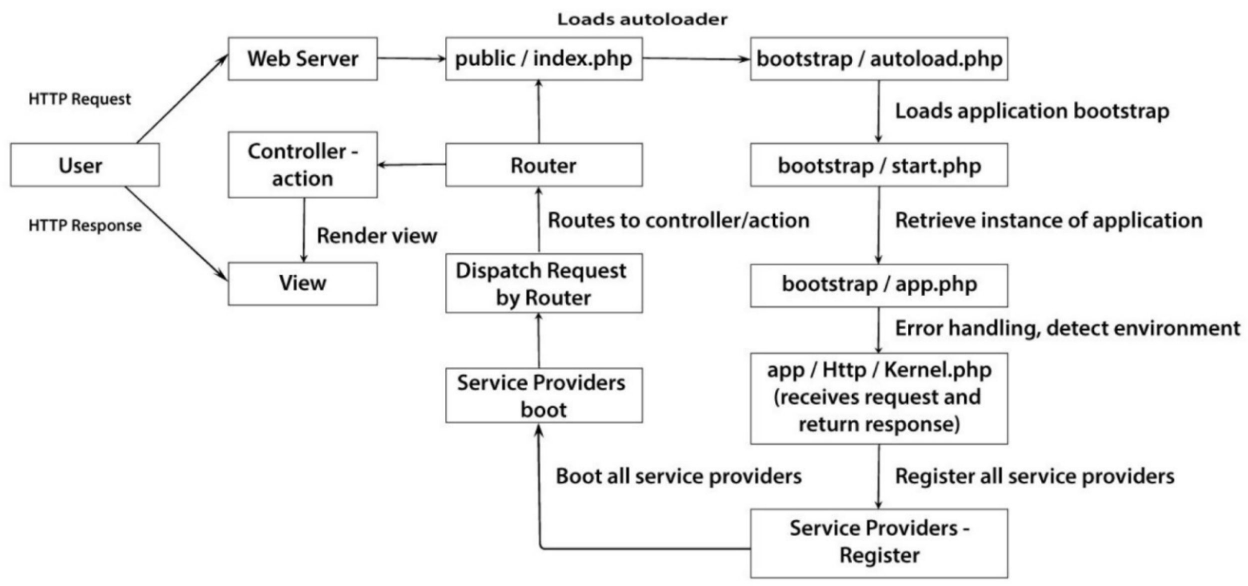
Loads autoloader
Web Server
public / index.php
bootstrap / autoload.php
HTTP Request
Loads application bootstrap
User
Controller - action
Router
bootstrap / start.php
HTTP Response
Routes to controller/action
Retrieve instance of application
Render view
View
Dispatch Request by Router
bootstrap / app.php
Error handling, detect environment
Service Providers boot
app / Http / Kernel.php (receives request and return response)
Boot all service providers
Register all service providers
Service Providers - Register

Fig. The lifecycle of an HTTP request in Laravel [5]

The execution of Laravel applications depends on the PHP interpreter and Zend engine, where the OpCache bytecode caching mechanism plays a crucial role. Starting with recent versions of PHP, OpCache allows users to store the compiled version of scripts in shared memory, avoiding re-parsing and recompilation with each request. The official PHP documentation describes certain parameters, such as opcache.enable, opcache.memory_consumption, and opcache.max_accelerated_files, as essential for reducing interpreter overhead [7].

Laravel optimization practice guides emphasize that properly configuring the OpCache and using preload scripts, if necessary; can significantly reduce the average response time without changing the application code [3].

The traditional PHP application execution model consists of Nginx or Apache with PHP-FPM as the process manager. In this setup, each request is handled by a separate workflow, which loads and initializes the framework, executes business logic, and terminates after processing is complete. While this architecture is simple to operate and reliable, it imposes, substantial overhead costs because the framework is loaded fully for each request. Analyses of PHP servers suggest that the PHP-FPM model is being gradually supplemented and replaced with new variants based on long-running processes and asynchronous I/O, such as Swoole, RoadRunner, and FrankenPHP.

In the context of Laravel, a significant development was the introduction of Laravel Octane. This provides support for long-running workflows based on Swoole or RoadRunner. As described in the official documentation and practical articles, Octane initializes the dependency container and application core once, keeping them in memory. This allows multiple requests to be served without having to reload the framework, reducing initialization overhead and allowing resources to focus on executing business logic and interacting with the database. Public tests have shown that configurations using Octane can lead to a significant increase in throughput compared to traditional PHP-FPM setups, while average and median response times are noticeably reduced. In one of the tests, Octane based on Swoole showed a 3-5 times improvement in throughput and a 70% reduction in median latency compared to FPM under typical load. RoadRunner also demonstrated similar results with slightly lower memory consumption [2]. Table 1 provides an example of the comparative throughput performance of a Laravel application in various execution environments (based on synthetic API endpoint testing).

Table 1

An example of comparative throughput performance of a Laravel application in different execution environments

Execution environment	**Approximate range of RPS Relative**	**Relative performance**
Nginx + PHP-FPM	200-400	1×
Laravel Octane (RoadRunner)	4 000-8 000+	≈ 10-20×
Laravel Octane (Swoole)	3-5× higher FPM in bandwidth, comparable to RoadRunner in some tests	3-5× and more

Source: author's development on based [8]

These values were obtained under controlled conditions and are indicative, but they demonstrate a general trend: migrating a Laravel application from the classic FastCGI Process Manager (FPM) model to long-lived Octane processes significantly reduces the overhead of loading the framework and improves query processing efficiency.

It is important to note the differences between the characteristics of various environments used for Laravel development. Swoole, for example, is an extension to PHP that provides low-level asynchronous input/output (I/O), coroutines, timers, and other features that allow developers to build highly scalable systems. However, it requires compilation and a more complex setup process. On the other hand, RoadRunner is an independent server written in the Go programming language that acts as a replacement for the PHP-FastCGI Process Manager, without requiring modifications to the PHP interpreter. This makes it easier to deploy and integrate into existing infrastructure.

The comparative characteristics of PHP-FPM, RoadRunner, and Swoole, when used in conjunction with Laravel, are presented in Table 2.

Table 2

Comparative characteristics of PHP-FPM, RoadRunner and Swoole when used with Laravel

Wednesday	**Model Type**	**Implementation features**	**Main advantages**
PHP-FPM	Short-lived processes	Standard Linux Distribution Packages	Easy to set up, predictable behavior
RoadRunner	Long-lived workers	A separate binary server on Go	Increased productivity, easier than Swoole
Swoole	PHP Extension	Requires compilation and specific assembly	Maximum performance, asynchronous I/O

Source: author's development on based [1]

Thus, already at the level of choosing a runtime environment, the basic limitations of the effectiveness of a Laravel application are set. At the same time, architectural and software optimization tools integrated into Laravel itself are a prerequisite for achieving sustainable results.

Publicly available results from load testing of Laravel applications show that the choice of runtime environment has a significant impact on performance and latency. In synthetic "Hello, World" tests using wrk and minimal business logic; authors of several articles on Laravel Octane obtained the following data: Laravel running on Octane processed 2,667 requests (266 RPS) in 10 seconds, while when running on Apache, 1,210 requests (121 RPS) were processed and the built-in PHP server achieved 705 requests (70 RPS).

Table 3 presents a comparison of Laravel start-up modes based on synthetic wrk testing results.

Table 3

Comparison of Laravel startup modes based on the results of the synthetic wrk test

Laravel operating mode	**Requests processed in 10 seconds**	**Requests per Second**
Laravel with Octane	2667	266 RPS
Laravel via Apache	1210	121 RPS
Laravel on an embedded server	705	70 RPS

Source: author's development on based [9]

Even taking into account that this test doesn't include database access and reflects the potential of servers rather than actual load, it's clear that using Octane as an add-on on high-performance servers results in at least a twofold increase in RPS (requests per second) compared to the classic setup with Apache.

A comparison of PHP-FPM, Nginx Unit, and Laravel Octane on Habr shows that Nginx Unit provides almost a tenfold reduction in response time when using Laravel compared to PHP-FPM. Octane, based on Swoole, further reduces delays but imposes strict requirements for application configuration and memory. In practical cases, developers have achieved more than 2,000 RPS using Laravel Octane with Swoole under strict logging control. However, adding intensive logging can reduce throughput to hundreds of RPS, emphasizing the importance of optimizing the logging system.

Practical recommendations for optimizing the performance of Laravel applications should take into account the results of experiments and the official documentation. The most important thing is to correctly configure the framework: use the php artisan commands config:cache, route:cache, and view:cache to pre-cache configuration, routes, and templates. It is also important to select a high-performance cache driver, such as Redis or Memcached, and disable debugging panels in production.

Database optimization involves eliminating the N+1 problem by loading links greedily, adding indexes to frequently used columns, using aggregate queries, and using the Query Builder in bottleneck situations. This is confirmed by practical guidelines for optimizing queries in Laravel.

Recommended runtime options and basic optimization measures for different types of Laravel applications are presented in Table 4, based on the general conclusions of relevant articles and documentation.

Table 4

Recommended runtime options and basic optimization measures for different classes of Laravel applications

Application Type	**Recommended execution environment**	**Basic optimization measures**
Small websites, internal services	Nginx + PHP-FPM	Cache of configs and routes, disabling debug, basic MySQL tuning
Medium-loaded Internet services	Nginx + PHP-FPM or Nginx Unit	Redis cache, queues for background tasks, optimization of database queries
Highly loaded APIs and stores	Laravel Octane (Swoole, RoadRunner, FrankenPHP)	Long-lived workers, aggressive caching, load sharing, health monitoring and logging

Source: author's development

Special attention should be given to logging, queue management, and background task processing. In one of the requests made to the Laravel Octane repository, a developer noted a drop in throughput from over 2,000 requests per second (RPS) to less than 100 RPS after adding an entry to the log for each request. This demonstrates the need to limit synchronous logging and move heavy operations into asynchronous queues [4].

Conclusions. The research conducted has shown that the performance of Laravel applications depends significantly on the chosen PHP runtime environment. The traditional PHP-FPM model provides stable performance for moderate loads, but it has a significant overhead associated with reinitializing the framework with each request.

High-performance environments based on long-lived processes, such as Swoole, RoadRunner, and FrankenPHP (used in conjunction with Laravel Octane); demonstrate a multiple increase in throughput (3-20 times, depending on the scenario) and a significant reduction in latency by keeping the Laravel core in memory.

For synthetic tests and real-world loads on large applications, Swoole and RoadRunner configurations provide the best results, ensuring an optimal balance between RPS, CPU, and RAM usage. Nginx Unit and Apache/mod_php show significantly lower performance and are less preferable for high-load projects.

The study confirms the need for comprehensive optimization, including pre-caching configuration and routes, controlling logs, eliminating the N+1 effect, configuring databases correctly, queuing long-term operations, and using Redis as the primary cache driver. Based on the analysis of the obtained data, recommendations are provided for the use of different execution environments depending on the level of load and response requirements.

References

1. Benchmarking Laravel Helpers: A Performance Comparison. URL: https://medium.com/%40mirceacojocaru90/benchmarking-laravel-helpers-a-performance-comparison-e5643e55df47.
2. Laravel Octane vs RoadRunner: Benchmarking PHP for Real-Time Apps. URL: https://valtry.co.uk/laravel-octane-vs-roadrunner-benchmarking-php-for-real-time-apps/.
3. Laravel Performance Boost with OPcache Preloading. URL: https://medium.com/insiderengineering/laravel-performance-boost-with-opcache-preloading-62878b3f3769.
4. Logging in requests is very slow. URL: https://github.com/laravel/octane/issues/724.
5. Php Register Shutdown Function Nedir? – Ahmet İmamoğlu. URL: https://ahmeti.com.tr/php-register-shutdown-function-nedir.
6. Request Lifecycle. URL: https://laravel.com/docs/12.x/lifecycle.
7. Runtime Configuration. URL: https://www.php.net/manual/en/opcache.configuration.php.
8. Swoole vs Roadrunner for Laravel Octane. URL: https://chriswhite.blog/coding/swoole-vs-roadrunner-for-laravel-octane/.
9. What is Laravel Octane? – Things To Know &amp; Benchmarks. URL: https://redberry.international/what-is-octane-in-laravel-things-to-know/.