Optimizing Laravel for High Concurrency with Octane

Optimizing Laravel for High Concurrency with Octane

By default, Laravel boots the entire framework on every request. This design is flexible but adds overhead. **Laravel Octane** removes that overhead by keeping the app in memory between requests using **Swoole** or **RoadRunner**. The result? Apps that handle thousands of requests per second with minimal latency. In this guide, we'll install Octane, configure workers, run benchmarks, and compare results.

1 – Install Octane

Install Octane and choose a server engine. Swoole is the most feature-rich option.

```
composer require laravel/octane
php artisan octane:install
Code language: Bash (bash)
```

This installs Octane's service provider and config. You'll be asked to choose Swoole or RoadRunner. Swoole provides task workers, coroutines, and better concurrency support.

2 – Running Octane

You can run Octane with Artisan. By default, it uses port 8000.

```
php artisan octane:start --server=swoole --port=8000
```
Code language: Bash (bash)

This boots Laravel once, then keeps it in memory across requests. Startup overhead (autoloading, config, service providers) is eliminated on subsequent requests.

3 – Worker Configuration

Octane uses workers to handle requests. Configure workers in `config/octane.php`.

```
// config/octane.php (snippet)
'workers' => 8,
'task_workers' => 4,
'max_requests' => 1000,
```
Code language: PHP (php)

`workers` are processes handling HTTP requests. `task_workers` run async tasks like broadcasting or sending emails. `max_requests` controls how many requests a worker handles before restarting (to free memory leaks).

4 – Benchmarking Octane

Let's compare performance using `ab` (Apache Bench) or `wrk`. First, test the default PHP-FPM setup:

```
# Default PHP-FPM (without Octane)
ab -n 1000 -c 50 http://127.0.0.1:8000/
```
Code language: Bash (bash)

Typical result: ~200 requests/sec with ~50ms latency.

```
# With Octane + Swoole
ab -n 1000 -c 50 http://127.0.0.1:8000/
Code language: Bash (bash)
```

Typical result: ~1,500 requests/sec with ~5ms latency. That's a **7x performance boost** just by running under Octane.

5 – Octane Tasks

Swoole's task workers let you run async jobs without queues. This is useful for lightweight background tasks.

```
// routes/web.php
use Laravel\Octane\Facades\Octane;

Route::get('/report', function () {
    Octane::concurrently([
        fn () => DB::table('users')->count(),
        fn () => DB::table('orders')->count(),
    ]);
});
Code language: PHP (php)
```

This runs multiple DB queries concurrently inside Octane. Great for speeding up dashboards and reports. For larger workloads (like sending thousands of emails), stick to queues.

6 – Combining Octane with Caching

Octane boosts raw performance, but combining it with Redis caching makes apps both fast and efficient. Cached queries served from memory under Swoole can drop response times to under 2ms.

7 – Monitoring Octane Performance

Octane removes Laravel's per-request boot, which means memory leaks or long-running tasks can degrade performance. Monitor with Telescope or external tools like New Relic and Blackfire.

Wrapping Up

Laravel Octane is a game-changer for high concurrency. By keeping the app in memory, Octane delivers massive performance boosts with minimal changes to your code. We installed Octane, configured workers, benchmarked performance, ran concurrent tasks, and combined it with caching. With proper monitoring and queues, Octane lets Laravel handle enterprise-scale traffic with ease.

1v0 Ship v1.0 Faster

What’s Next

- 10 Proven Ways to Optimize Laravel for High Traffic — Octane is just one piece of the optimization puzzle.
- Caching Strategies in Laravel: Redis vs Database vs File — combine Octane with effective caching strategies.
- Using Laravel Telescope to Debug Performance Issues — monitor queries, requests, and memory leaks in Octane apps.