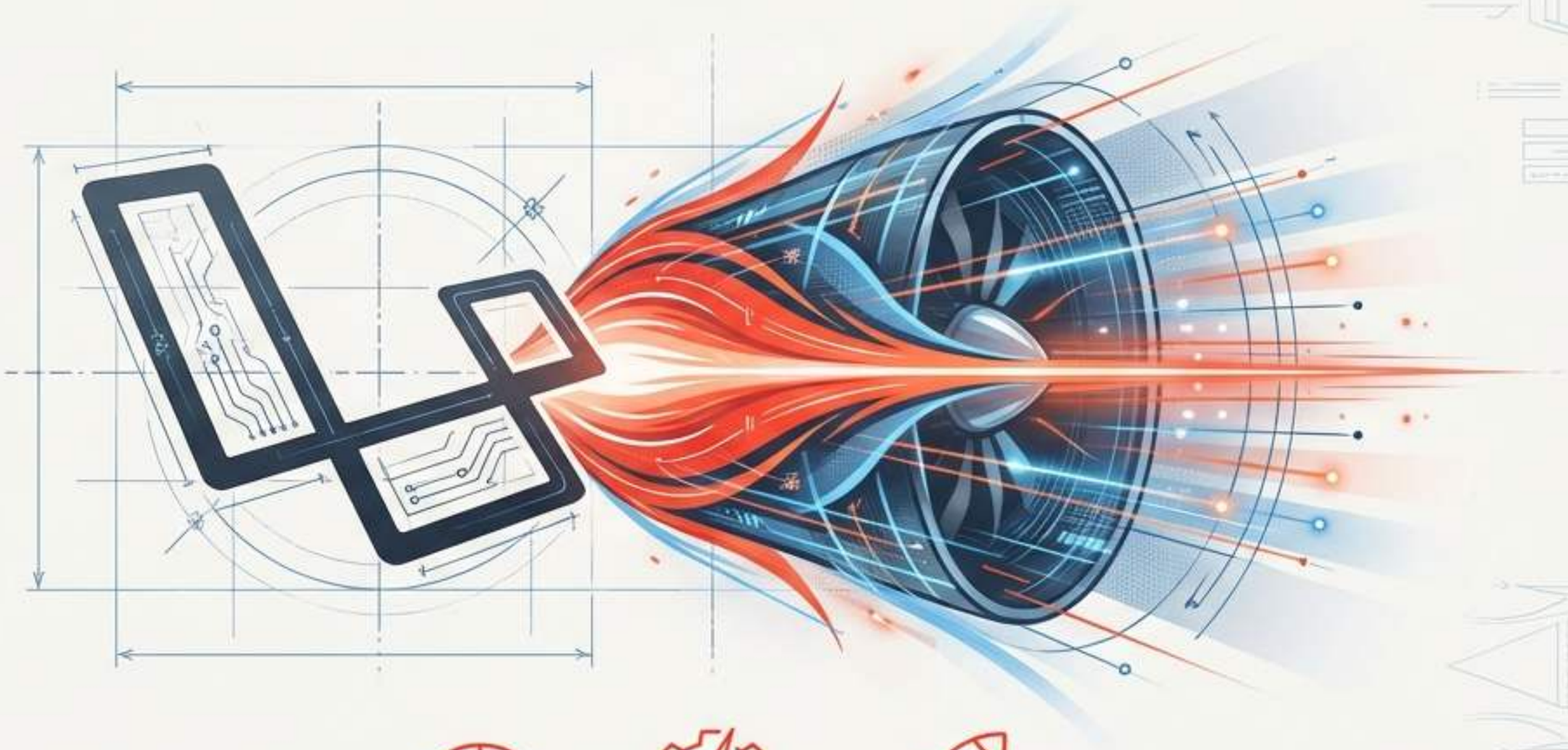
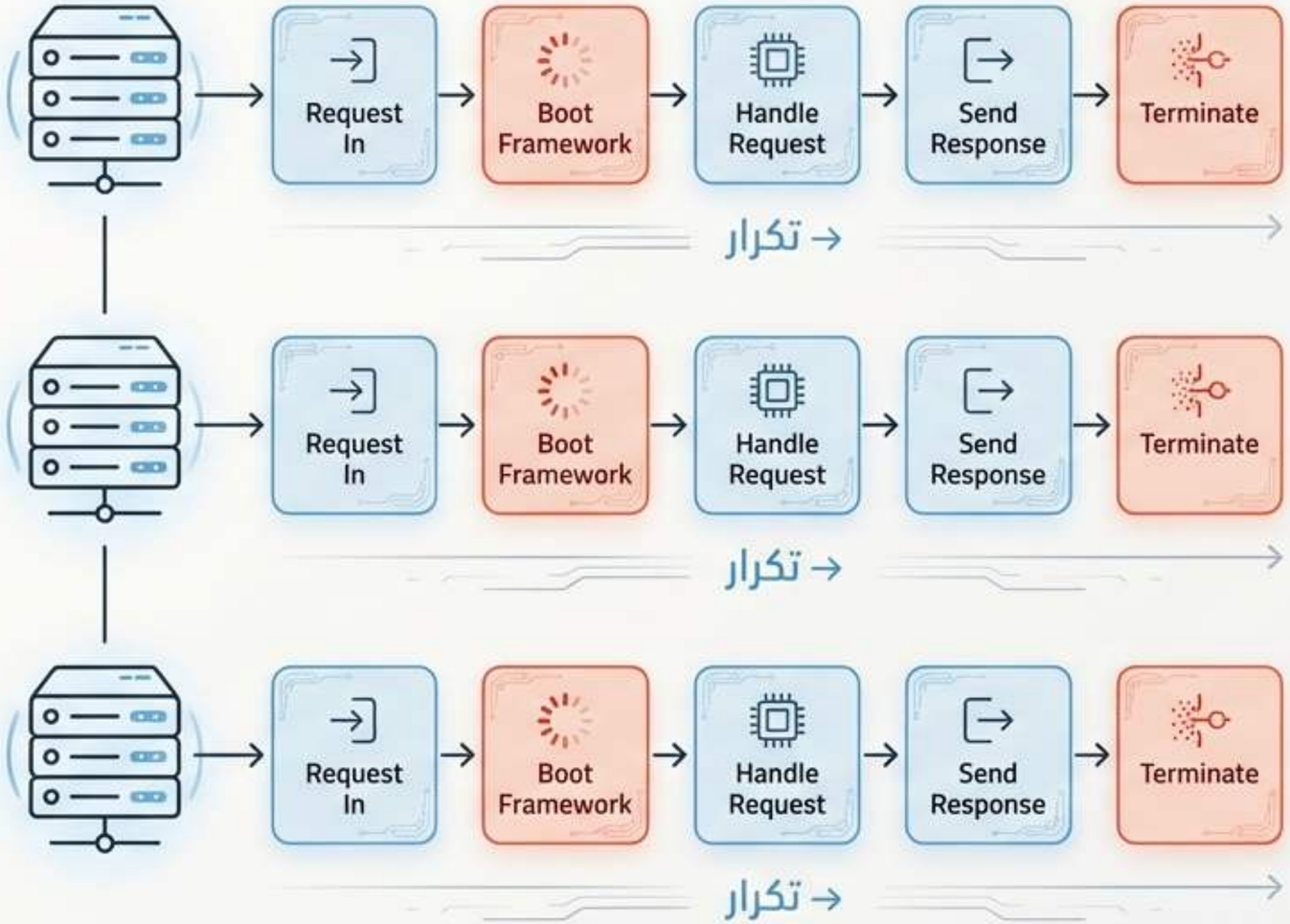
Laravel Octane: السرعة الفائقة بين يديك

دليلك الكامل لتعزيز أداء تطبيقات Laravel إلى أقصى حد.



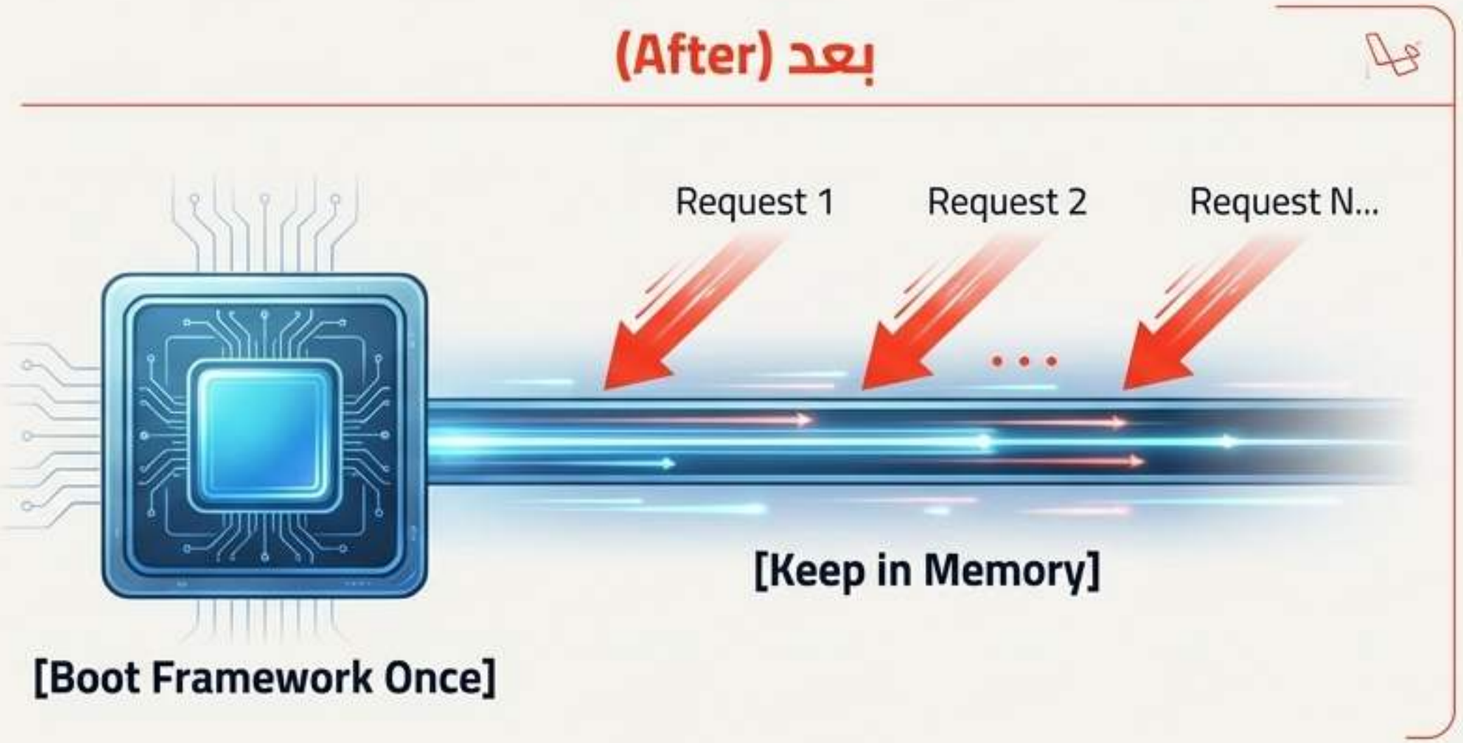
تحدي الأداء: دورة حياة طلب PHP التقليدية



- في الوضع التقليدي، مع كل طلب HTTP، يقوم Laravel بتمهيد (bootstrapping) التطبيق بالكامل من الصفر.
- هذه العملية تتضمن قراءة الإعدادات، تسجيل الـ Service Providers، وتجهيز كل شيء لمعالجة طلب واحد فقط.
- بمجرد انتهاء الطلب، يتم التخلص من كل شيء في الذاكرة. هذه العملية المتكررة تستهلك موارد قيمة وتحد من عدد الطلبات التي يمكن للتطبيق معالجتها في الثانية.

الحل: Laravel Octane يغير قواعد اللعبة

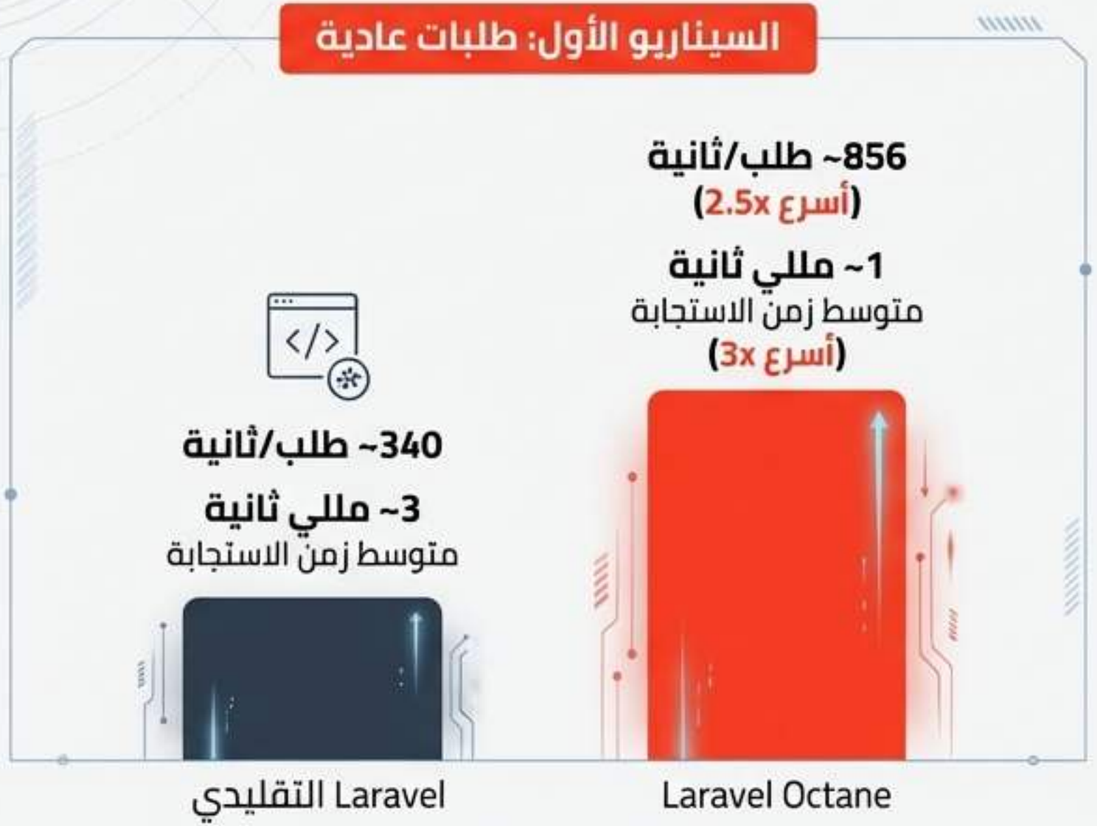
- يقوم Octane بتعزيز أداء تطبيقك بشكل هائل عن طريق تشغيله باستخدام خوادم تطبيقات عالية القوة مثل FrankenPHP, RoadRunnerg, Swooleg.
- ****المفهوم الأساسي*:** يقوم Octane بتمهيد إطار العمل *مرة واحدة فقط* ويبقيه في الذاكرة.
- عندما تصل الطلبات الجديدة, يتم تمريرها مباشرة إلى التطبيق الجاهز بالفعل, مما يلغي تمامًا وقت التمهيد المتكرر.



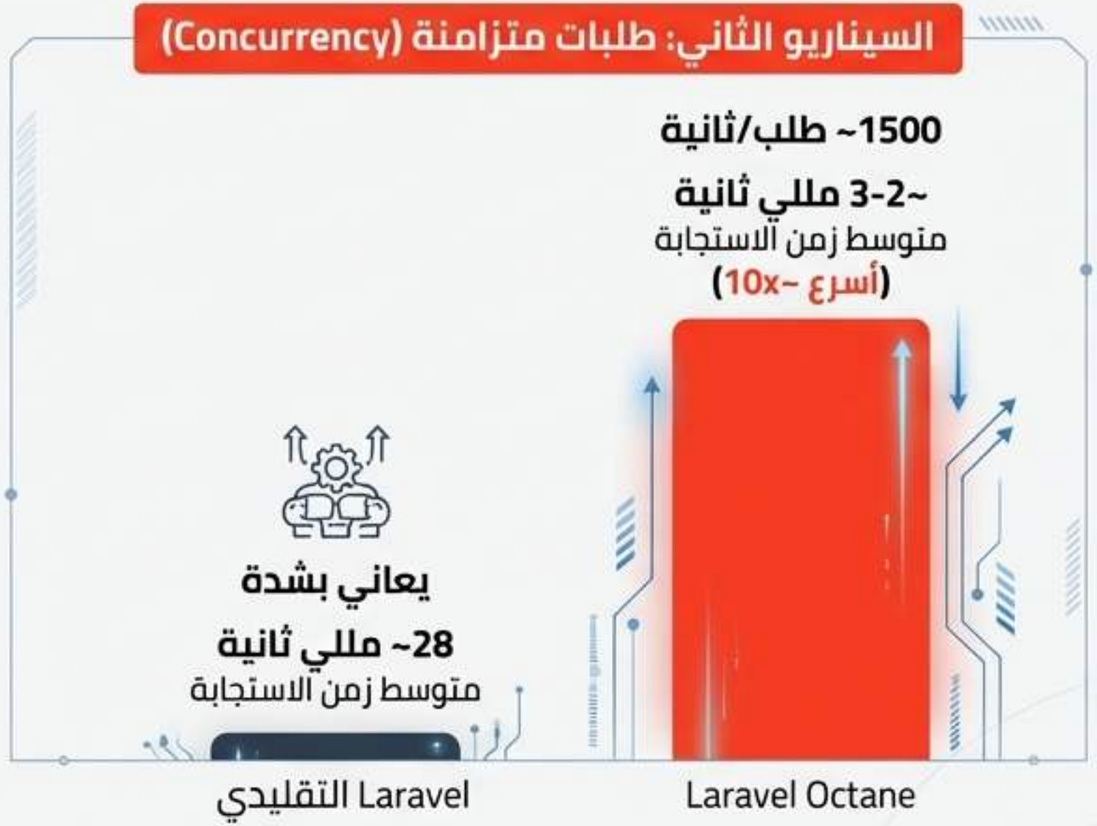
الأرقام لا تكذب: قفزة الأداء مع Octane

استنادًا إلى اختبار أداء (stress test) على تطبيق Laravel جديد, هذه هي النتائج:

السيناريو الأول: طلبات عادية



السيناريو الثاني: طلبات متزامنة (Concurrency)



"هذه الأرقام مذهلة. هذا كم هائل من الزيارات, وأزمنة الاستجابة لا تصدق."

الخطوة الأولى: التثبيت والإعداد

يمكنك البدء باستخدام Octane في دقائق. كل ما تحتاجه هو أمران في Composer و Artisan.

الخطوة 1: التثبيت عبر Composer

```
composer require laravel/octane
```

الخطوة 2: تهيئة ملف الإعدادات

سيقوم هذا الأمر بإنشاء ملف `config/octane.php` في تطبيقك.

```
php artisan octane:install
```

اختر محركك: متطلبات الخادم

يدعم Octane العديد من خوادم التطبيقات القوية. اختر ما يناسب مشروعك.

Swoole / Open Swoole

- يتطلب تثبيت إضافة PHP خاصة (PECL extension).
- يفتح الباب أمام ميزات متقدمة مثل المهام المتزامنة والتخزين المؤقت فائق السرعة.
- للأداء الأقصى والتطبيقات المعقدة.

RoadRunner

- مدعوم بـ RoadRunner binary، مبني أيضًا بلغة Go.
- يقوم Octane أيضًا بالمساعدة في تنزيله وتثبيته.
- خيار قوي ومستقر للإنتاج.

FrankenPHP

- خادم تطبيقات PHP حديث مكتوب بلغة Go.
- سهل الإعداد: Octane يقوم بتنزيله وتثبيته تلقائيًا عند الاختيار. الاختيار.
- مثالي للبدء السريع والتطوير المحلي.

مثال عملي: إعداد Octane مع FrankenPHP و Laravel Sail

إذا كنت تستخدم Laravel Sail للتطوير المحلي، فإن إعداد Octane بسيط للغاية.

الخطوة 1: تثبيت Octane وتحديد الخادم

```
./vendor/bin/sail composer require
laravel/octane
./vendor/bin/sail artisan
octane:install --server=frankenphp
```

الخطوة 2: تحديث `docker-compose.yml`

أضف متغير البيئة `SUPERVISOR_PHP_COMMAND` ليقوم Sail بتشغيل Octane بدلاً من خادم PHP المدمج.

```
services:
  laravel.test:
    environment:
      SUPERVISOR_PHP_COMMAND: "/usr/bin/php
artisan octane:start --server=frankenphp --
host=0.0.0.0 --port=80"
```

ملاحظة: "هذه التعديلات تجعل تجربة التطوير سلسة وتعكس بيئة الإنتاج بشكل أفضل."

تشغيل المحرك: إدارة خادم Octane

بدء تشغيل الخادم

بشكل افتراضي، سيعمل الخادم على `http://localhost:8000`.

```
php artisan octane:start
```

أهم خيارات التشغيل (Options):

للأمان

إعادة تشغيل الـ worker تلقائيًا بعد عدد معين من الطلبات لمنع تسرب تسرب الذاكرة.

```
--max-requests=250
```

للإنتاج (Production)

تحديد عدد الـ Workers ليتناسب مع أنوية المعالج (CPU cores).

```
--workers=4
```

للتطوير (Development)

إعادة تشغيل الخادم تلقائيًا عند أي تغيير في الملفات.

```
--watch
```

⚠ *متطلب*:
npm install --save-dev chokidar

جاهز للإنتاج: النشر (Deployment) والإدارة المستمرة

في بيئة الإنتاج، يجب أن تضمن أن خادم Octane يعمل باستمرار.

تحديث الكود بدون توقف (Zero-Downtime Reload)

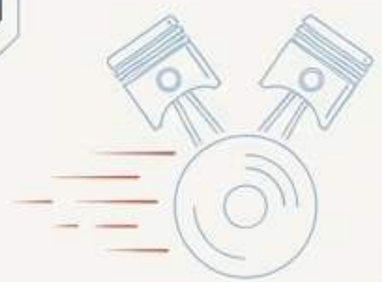
بعد نشر تحديثات جديدة، استخدم الأمر التالي لتحميل الكود الجديد في الذاكرة بدون إيقاف الخادم.

```
php artisan octane:reload
```

أوامر أخرى:

بعد نشر `octane:status`

`octane:status` و `octane:stop`



استخدام مدير عمليات (Process Manager)

ينصح بشدة باستخدام أداة مثل `Supervisor` لمراقبة عملية Octane وإعادة تشغيلها تلقائيًا في حالة فشلها.

مثال لإعداد Supervisor:

```
[program:octane]
command=php /path/to/your/project/artisan
octane:start --server=frankenphp --port=8000
autostart=true
autorestart=true
user=forge
redirect_stderr=true
stdout_logfile=/path/to/logs/octane.log
```

القاعدة الذهبية في Octane: فهم الحالة (State) الدائمة

بما أن Octane يُبقي تطبيقك في الذاكرة بين الطلبات، فإن أي شيء يتم إنشاؤه مرة واحدة (مثل الـ Singletons) سيتم مشاركته عبر طلبات متعددة.

يتم تنفيذ دالتي `register` و `boot` في الـ Service Providers ***مرة واحدة فقط*** عند بدء تشغيل الـ worker.

هذا يعني أن حقن (injecting) كائنات مثل حاوية الخدمات (Service Container) أو الطلب (Request) مباشرة في مُنشئ (constructor) كائن Singleton قد يؤدي إلى مشاكل خطيرة، حيث سيحتفظ الكائن بنسخة قديمة وغير محدّثة.

انتبه: طريقة تفكيرك في بناء الخدمات يجب أن تتغير.

تجنب المزالق الشائعة: حقن الاعتماديات (Dependency Injection)

كن حذرًا عند حقن الكائنات التي تتغير حالتها بين الطلبات في الـ Singletons.

المشكلة: حقن الـ Config

الكود الخاطئ

```
// This service won't see config changes
$this->app->singleton(Service::class, function ($app) {
    return new Service($app->make('config'));
});
```

الحل

استخدم الدالة المساعدة `config()` التي تضمن دائمًا الحصول على أحدث القيم.

```
// Correct: Use the config() helper or a closure
$this->app->singleton(Service::class, function ($app) {
    // This is also correct, as config() resolves fresh
    return new Service(config());
});
```

المشكلة: حقن الـ Request

الكود الخاطئ

```
// This service will hold a stale request object
$this->app->singleton(Service::class, function ($app) {
    return new Service($app['request']);
});
```

الحل

استخدم Closure لإعادة جلب نسخة الطلب الحالية عند الحاجة، أو استخدم الدالة المساعدة `request()`.

```
// Correct: Resolve the request inside a closure
$this->app->singleton(Service::class, function ($app) {
    return new Service(fn () => $app['request']);
});
```

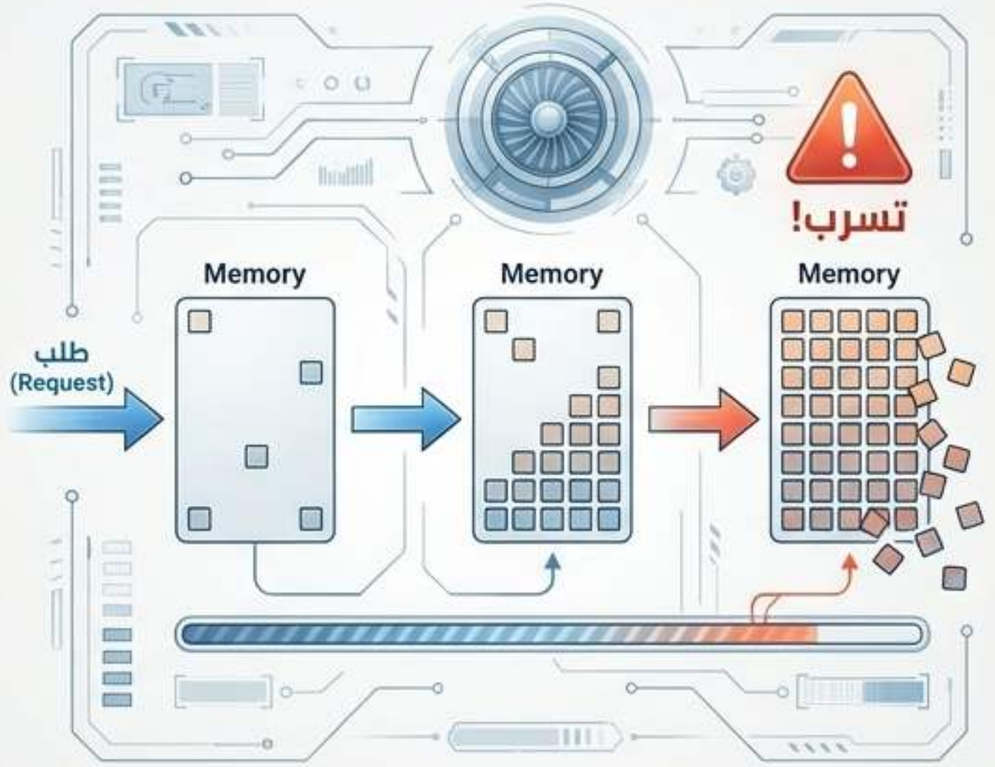
إدارة الذاكرة: كيفية منع التسرب (Memory Leaks)

لأن تطبيقك يعمل بشكل مستمر، فإن إضافة بيانات إلى مصفوفات ثابتة (static arrays) أو خصائص كائن مشترك (shared object properties) سيؤدي حتمًا إلى تسرب في الذاكرة.

مثال على تسرب الذاكرة

```php
class SomeService
{
    public static array $data = [];

    public function handle()
    {
        // This array will grow with every request, causing a memory leak!
        self::$data[] = 'new data';
    }
}
```



الوقاية والحماية

- **القاعدة:** تجنب تخزين أي بيانات خاصة بالطلب (request-specific data) في حالة (state) مشتركة أو ثابتة.
- **شبكة الأمان:** استخدم خيار `--max-requests` لإعادة تشغيل الـ workers بشكل دوري كإجراء وقائي.

إطلاق العنان للقدرات الخارقة (يتطلب Swoole)

عند استخدام خادم Swoole, يمنحك Octane مجموعة من الأدوات القوية للتطبيقات عالية الأداء.

المهام الدورية (Ticks & Intervals)

قم بتشغيل كود معين بشكل دوري كل N ثانية.

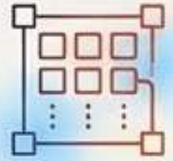
```
Octane::tick('heartbeat', fn() => info('…'))->seconds(10);
```

المهام المتزامنة (Concurrent Tasks)

قم بتنفيذ عمليات متعددة في نفس الوقت واستقبل نتائجها معًا.

```
[$users, $servers] = Octane::concurrently([
    fn () => User::all(),
    fn () => Server::all(),
]);
```

جداول Swoole Tables



وصول مباشر وعالي الأداء للبيانات المشتركة في الذاكرة.

```
Octane::table('example')->get('uuid');
```

ذاكرة التخزين المؤقت Octane Cache

ذاكرة تخزين مؤقت فائقة السرعة (تصل إلى 2 مليون عملية/ثانية) مشتركة بين جميع الـ workers.

```
Cache::store('octane')->put('key', 'value', 30);
```

لماذا تختار Octane؟ المزايا في نقاط

سرعة فائقة (Supersonic Speed)

تقليل زمن الاستجابة بشكل كبير وزيادة هائلة في عدد الطلبات التي يمكن معالجتها في الثانية.

كفاءة الموارد (Resource Efficiency)

تقليل الحمل على الـ CPU والذاكرة عن طريق إلغاء عملية التمهيد المتكررة.

جاهز للإنتاج (Production Ready)

أدوات مدمجة للإدارة والنشر السلس، ودعم لـ Supervisor g Nginx.

قدرات متقدمة (Advanced Capabilities)

مع Swoole، يمكنك بناء تطبيقات متزامنة وفي الوقت الفعلي بسهولة.

تجربة Laravel التي تحبها (The Laravel Experience You Love)

كل هذه القوة مع الحفاظ على أناقة وسهولة استخدام إطار عمل Laravel.

"الأداء الفائق لم يعد حكرًا على النخبة. إنه للجميع."

يمكن لتطبيق Laravel الخاص بك الآن منافسة أطر العمل التي تم تصميمها خصيصًا للسرعة، كل ذلك مع الحفاظ على تجربة المطور التي تعشقها.

هل أنت مستعد للانطلاق بسرعة أكبر؟

تصفح التوثيق الرسمي لـ Laravel Octane لتبدأ رحلتك.