ChatGPT

قابلية التوسع وإدارة الحمل

مقدمة

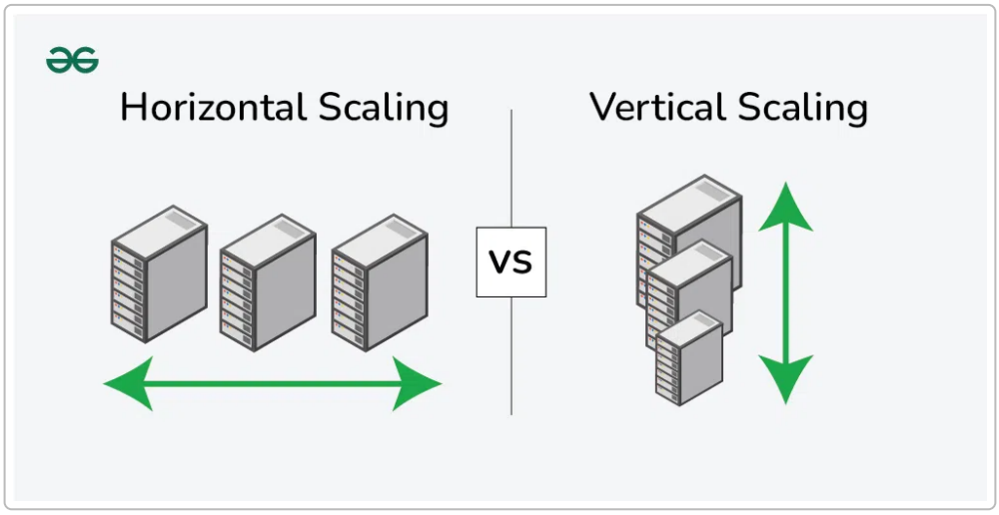
في عالم تطوير البرمجيات الحديثة، تُعتبَر **قابلية التوسع (Scalability)** وإدارة الأحمال من أهم العوامل لضمان نجاح التطبيقات. فمع نمو قاعدة المستخدمين وزيادة حجم البيانات والطلبات، يجب أن يكون النظام قادرًا على التكيف دون التضحية بالأداء أو الاستقرار [1]. يساعد فهم استراتيجيات التوسع وإدارة الحمل مهندسي البرمجيات (خصوصًا مطوري الـ Backend) على بناء أنظمة مرنة تستطيع تلبية الطلب المتزايد والحفاظ على أداء ثابت.

مفهوم قابلية التوسع (Scalability)

قابلية التوسع تعني قدرة النظام على التعامل بكفاءة مع ازدياد حجم العمل أو عدد المستخدمين أو حجم البيانات دون انخفاض في مستوى الأداء [2] [1]. بمعنى آخر، النظام القابل للتوسع يستمر في العمل بشكل جيد - أو حتى يتحسّن أداؤه - مع ارتفاع الحمل الواقع عليه. هذه الخاصية ضرورية للشركات والتطبيقات التي تتوقع نموًا مستمرًا أو زيادات مفاجئة في عدد المستخدمين. بالتخطيط الجيد لقابلية التوسع منذ البداية، يمكن تقليل مخاطر التوقف أو تدهور الأداء أثناء فترات الذروة [1].

من المهم التمييز بين نوعين رئيسيين من التوسع في الموارد الحاسوبية: **التوسعة الرأسية (Vertical Scaling)** و **التوسعة الأفقية (Horizontal Scaling)**. سنستعرض فيما يلي الفرق بينهما وكيفية الاستفادة من كل منهما.

التوسعة الرأسية مقابل التوسعة الأفقية



الشكل: مقارنة مبسّطة بين **التوسعة الرأسية** و **التوسعة الأفقية**. التوسعة الرأسية (يمين) تعني زيادة قدرات خادم واحد، بينما التوسعة الأفقية (يسار) تعني إضافة خوادم متعددة وتوزيع الحمل بينها.

- **التوسعة الرأسية (Vertical Scaling)**: تُعرَف أيضًا بـ"التحجيم الرأسي" أو "الزيادة إلى أعلى (Scale Up)". في هذا الأسلوب نقوم **بزيادة قدرات الخادم نفسه** عبر ترقية مكوناته (مثل زيادة المعالج CPU أو الذاكرة RAM أو

سعة التخزين) [3] . يظل النظام يعتمد على خادم واحد ولكن أكثر قوة. يتميز التوسع الرأسي بسهولة التنفيذ في الأنظمة الصغيرة أو الأحادية، إذ لا يتطلب تعديل هيكلية النظام بشكل كبير. من مزاياه أنه قد يكون أبسط من ناحية الإدارة (لا حاجة لإدارة عدة خوادم) وبعض الأحيان أكثر فعالية من حيث التكلفة على المدى القصير [4] . لكن من عيوبه وجود حد أعلى لهذا التوسع – فهناك سقف لقدرات أي خادم منفرد – بالإضافة إلى استمرار وجود نقطة فشل واحدة؛ فإذا تعطّل ذلك الخادم تتوقف الخدمة بالكامل [5] .

- **التوسعة الأفقية (Horizontal Scaling)** : تُعرَف أيضًا بـ"التحجيم الأفقي" أو "التمدد إلى الخارج (Scale Out)". في هذا الأسلوب نقوم **بإضافة المزيد من الخوادم** أو الأجهزة إلى النظام والعمل على توزيع الحمل بينها [3] . عادةً يترافق ذلك مع استخدام موازن حمل لتوجيه الطلبات بين هذه الخوادم (سنتحدث عنه لاحقًا). يتيح التوسع الأفقي إمكانية زيادة الطاقة الاستيعابية للنظام بشكل شبه غير محدود عبر إضافة عقد (خوادم) جديدة [6] . يتميز هذا النهج بزيادة الاعتمادية وتجنب نقطة الفشل الواحدة؛ فعند وجود عدة خوادم، فإن تعطل خادم واحد لا يعني توقف الخدمة بالكامل [7] . كما أنه أكثر مرونة على المدى الطويل للتعامل مع نمو كبير في عدد المستخدمين. من جهة أخرى، يتطلب التوسع الأفقي بنية تحتية أكثر تعقيدًا، بما فيها إدارة **موازنات الحمل** وتنسيق البيانات بين الخوادم المختلفة، كما قد تزيد تعقيدات الحفاظ على تزامن البيانات والاتصال بينها [8] .

باختصار، التوسعة الرأسية تزيد قوة الخادم الواحد، بينما التوسعة الأفقية تزيد عدد الخوادم. كثير من الأنظمة الكبيرة تعتمد مزيجًا من الطريقتين لتحقيق أفضل النتائج، فتبدأ بتحسين قدرات الخوادم الحالية رأسيًا ثم إضافة خوادم أفقيًا عند الحاجة لتحقيق مرونة وتحمل أعلى [9] .

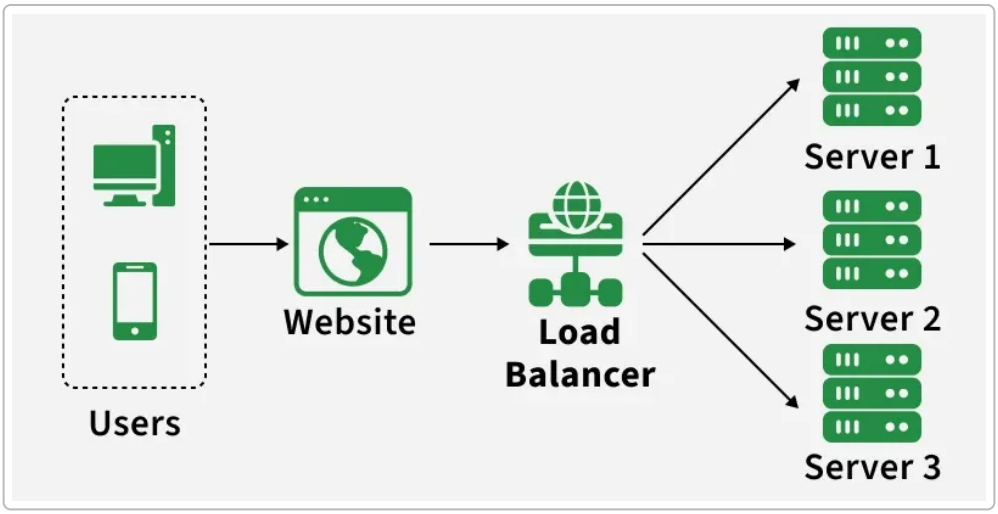
إدارة الأحمال (Load Management)

يقصد بإدارة الأحمال عملية توزيع وتنظيم عبء العمل (عدد الطلبات أو العمليات) على موارد النظام بشكل يضمن عدم استنزاف أي مورد بشكل يسبب اختناقًا أو انقطاعًا. تشمل إدارة الحمل مراقبة أداء النظام واستعمال موارده (المعالج، الذاكرة، عرض النطاق الترددي للشبكة، ...إلخ) واتخاذ إجراءات استباقية عند ارتفاع الحمل لضمان استمرار الخدمة بكفاءة.

موزانات الحمل (Load Balancers) تمثل حجر الزاوية في إدارة الأحمال للتطبيقات والخدمات واسعة النطاق. بالإضافة إلى ذلك، تشمل إدارة الأحمال استراتيجيات أخرى مثل **التوسعة التلقائية** (إضافة موارد تلقائيًا عند الحاجة) و**الفصل المعماري عبر الطوابير** (توزيع المهام على مراحل مختلفة باستخدام أنظمة الطوابير) للتعامل مع فترات الضغط العالي. سنستعرض أهم هذه العناصر فيما يلي.

موازن الحمل (Load Balancer) وأهميته

موازن الحمل (Load Balancer) هو جهاز أو برنامج يقوم بتوزيع الطلبات الواردة على عدة خوادم خلفه لضمان عدم انهيار أي خادم تحت الضغط [10] . يلعب موازن التحميل دور "شرطي المرور" الذي يوجّه حركة البيانات: يستقبل الطلبات من العملاء، ثم يمرّر كل طلب إلى الخادم الأنسب أو الأقل انشغالًا ضمن مجموعة الخوادم [10] . بهذا يضمن أن **لا يتعرض خادم واحد لكل الضغط وحده** وأن الموارد متاحة لخدمة المستخدمين بكفاءة.



الشكل: مثال على بنية نظام تستخدم **موازن تحميل** لتوزيع حركة الطلبات بين خوادم متعددة. يقوم المستخدمون بإرسال الطلبات إلى عنوان واحد (الموقع)، فيتولى موازن الحمل توجيه كل طلب إلى أحد الخوادم المتاحة بالتساوي لضمان عدم ازدحام أي خادم.

تظهر أهمية **موازنات الحمل** بشكل جلي في تحسين أمرين أساسيين: **التوفرية والموثوقية** . فبدون موازن حمل، أي تعطل في الخادم الوحيد يعني توقف الخدمة (Single Point of Failure). أما مع وجود موازن حمل وخوادم متعددة، يستطيع الموازن استبعاد أي خادم معطل وتوجيه الطلبات إلى خوادم أخرى سليمة [11] [12] ، مما يوفر قدرًا عاليًا من الاستمرارية والخدمة دون انقطاع. كذلك يساهم التوزيع المتوازن للطلبات في **تحسين الأداء** ، حيث يتم خدمة المستخدمين بسرعة أكبر لتجنب تراكم كل العمل في خادم واحد [10] . ومن فوائد موازن الحمل أيضًا إمكانية توزيع الطلبات بناءً على معايير متقدمة (مثل أقل خادم من حيث الاتصالات النشطة أو الزمن الأسرع للاستجابة)، واستخدامه كنقطة مركزية يمكن عندها تنفيذ سياسات أمان مثل جدران الحماية أو فحص الحزم.

باختصار، يضمن موازن الحمل استغلال كل الخوادم المتاحة بكفاءة، ويحسّن قدرة النظام على **التوسع الأفقي** بسهولة، حيث يكفي إضافة خادم جديد وربطه بالموازن ليبدأ باستقبال جزء من الطلبات. هذه الآلية ضرورية خاصةً في فترات ارتفاع الحمل المفاجئ حيث تتضاعف أعداد المستخدمين أو حجم الترافيك.

استراتيجيات التعامل مع الحمل الزائد

عندما يتجاوز الضغط على النظام قدراته الحالية، لابد من اتباع استراتيجيات فعّالة لضمان استمرارية الخدمة. فيما يلي بعض أبرز الاستراتيجيات للتعامل مع الحمل الزائد والارتفاعات المفاجئة في الضغط:

- **التوسعة التلقائية (Auto-Scaling)** : وهي تقنية تتمثل في **زيادة (أو تقليل) عدد الخوادم تلقائيًا** وفقًا لمستوى الطلب. تعتمد هذه الإستراتيجية عادةً على مزوّدي الخدمات السحابية (مثل AWS أو Azure) حيث يتم رصد مؤشرات معينة (كنسبة استخدام المعالج أو عدد الطلبات) وعند تجاوز عتبة محددة تُنشئ خوادم إضافية تلقائيًا لتلبية الطلب [13] . على سبيل المثال، يمكن إعداد مجموعة خوادم تلقائية التوسع بحيث في **ساعات الذروة** (مثل مواسم التخفيضات أو الأحداث المباشرة) يتم تشغيل خوادم جديدة تلقائيًا، ثم إيقافها عند انتهاء الضغط، مما يحقق مرونة عالية وتكلفة فعّالة. تساعد هذه الآلية على تجنب بطء الاستجابة أو انهيار الخدمة أثناء التدفق العالي للمستخدمين، إذ يتم **توزيع الحمل الجديد على موارد إضافية** دون تدخل يدوي. من الأمثلة الواقعية الشهيرة: خدمة Auto Scaling من أمازون التي تقوم بتشغيل المزيد من مثيلات EC2 تلقائيًا خلال مواسم التسوق المزدحمة (مثل حدث الـBlack Friday) لاستيعاب الارتفاع الهائل في الطلب [13] . بالتزامن

مع هذا، يعمل **موازن الحمل** على إدراج هذه الخوادم الجديدة في الموازنة فور توفرها [14] لضمان توزيع فوري للحمل عليها.

- **الفصل عبر نظام الطوابير (Queue-Based Decoupling)** : هذه الإستراتيجية تعتمد على **تفكيك العمل إلى مهام منفصلة** وتخزين المهام في طابور (Queue) ليتم معالجتها بشكل غير متزامن بواسطة عمال (Workers) في الخلفية. بدلاً من أن يقوم التطبيق بتنفيذ كل المهام الثقيلة بشكل فوري ومباشر ضمن طلب المستخدم – مما قد يطيل زمن الاستجابة – يتم وضع تلك المهام في **صف انتظار** لتُعالَج بالتوازي خلف الكواليس [15] . بهذه الطريقة **يتحرر التطبيق للتجاوب سريعًا مع المستخدم** (مثلاً تأكيد استلام الطلب في الحال)، بينما تستمر معالجة المهام الإضافية (مثل إرسال بريد تأكيد أو توليد تقارير أو معالجة صور) على مهل عبر نظام الطوابير [15] . هذه الإستراتيجية تحقق **عزلًا (Decoupling)** بين جزء النظام الذي يتعامل مع المستخدم بشكل آنٍ وبين الأجزاء التي تقوم بالأعمال المرهقة زمنيًا. النتيجة هي قدرة أفضل على تحمل الأعباء العالية: يمكن إضافة المزيد من عمال الطابور لمعالجة المهام المتراكمة عند ارتفاع الضغط، دون التأثير على سرعة استجابة التطبيق الأساسي. تستخدم الكثير من الأطر العمل (مثل Laravel) نظام الطوابير كوسيلة لتحسين قابلية التوسع؛ حيث تدعم إنشاء **عمال متعددين** وتنظيمهم وربما موازنتهم تلقائيًا (كما سنرى في Horizon) للتعامل مع عدد هائل من الوظائف في الخلفية. إن فصل المهام بهذه الطريقة **يحول الحمل الزائد من كونه عبئًا على عملية واحدة إلى عبء موزع عبر عدة عمليات**، مما يحمي الجزء الأساسي من التطبيق من الانهيار تحت الضغط.

بالإضافة إلى ما سبق، هناك استراتيجيات أخرى مساعدة: مثل **استخدام التخزين المؤقت (Caching)** لتقليل الضغط على قواعد البيانات عبر تخزين النتائج المتكررة، أو **توزيع القراءات على قواعد بيانات ثانوية** (Read Replicas) لتخفيف الحمل عن قاعدة البيانات الرئيسية. كل هذه الأساليب تكمل بعضها لتعزيز قدرة النظام على التعامل مع أحمال عالية بشكل انسيابي.

استخدام نظام الصف (Queues) في Laravel وإعداد Horizon

توفر إطار عمل Laravel دعمًا قويًا لنظام الطوابير، مما يسهل تطبيق استراتيجية الفصل بالصفوف في تطبيقات الـBackend. سنستعرض مثالًا عمليًا لكيفية إعداد **Worker** لمعالجة المهام في الخلفية باستخدام Laravel، بالإضافة إلى استخدام أداة **Horizon** لمراقبة وإدارة العمال (Workers) بشكل فعال.

في Laravel، يتم عادةً تعريف المهام التي ستنفذ في الخلفية ضمن **فئات Jobs** تقوم بتنفيذ واجهة `ShouldQueue` . عند تنفيذ أي Job من هذا النوع، يمكن وضعه تلقائيًا في طابور بدلاً من تنفيذه حالًا. فيما يلي مثال مبسط لتعريف **Job** لإرسال بريد إلكتروني لتأكيد طلب، بحيث يتم إرساله في الخلفية بدلاً من الانتظار في الطلب الأساسي:

```php
<?php

use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Queue\SerializesModels;
use App\Models\Order;
use App\Mail\OrderConfirmationMail;
use Mail;

class SendOrderConfirmationEmail implements ShouldQueue
{
```

```
    use InteractsWithQueue, Queueable, SerializesModels;

    protected $order;

    // نمرر البيانات المطلوبة لتنفيذ المهمة (هنا تمرير الطلب الذي سنرسل بريده) في الـ constructor
    public function __construct(Order $order)
    {
        $this->order = $order;
    }

    public function handle()
    {
        // إرسال بريد التأكيد بشكل غير متزامن في الخلفية
        Mail::to($this->order->user->email)
            ->send(new OrderConfirmationMail($this->order));
        // ملاحظة: تنفيذ هذه المهمة في الخلفية يحسن قابلية التوسع بتخفيف الحمل عن طلب الويب الأساسي
    }
}
```

في المثال أعلاه، عرفنا مهمة (Job) باسم `SendOrderConfirmationEmail` تقوم بإرسال بريد إلكتروني عند استدعاء الدالة `handle()` . هذه المهمة سيتم وضعها في **Queue** (طابور) تلقائيًا لأن الكلاس implement واجهة `ShouldQueue` . استخدمنا facade `Mail` لإرسال رسالة تأكيد الطلب – وهو إجراء يستغرق وقتًا (إرسال بريد) – وبالتالي جعله في الخلفية بدلاً من إبطاء استجابة التطبيق الأساسية.

لإرسال هذه المهمة إلى الطابور، نستدعيها من مكان مناسب في تطبيق Laravel (مثلاً داخل **Controller** بعد إنشاء الطلب). بدلاً من استدعاء عملية إرسال البريد مباشرة، سنقوم بإرسال الـJob إلى الـQueue باستخدام الدالة `dispatch` المساعدة:

```
use App\Jobs\SendOrderConfirmationEmail;

class OrderController extends Controller
{
    public function placeOrder(Request $request)
    {
        // إنشاء طلب جديد (عملية افتراضية)
        $order = Order::create($request->all());
        // ... معالجة منطق إنشاء الطلب ...

        // إرسال مهمة إرسال البريد إلى نظام الطوابير بدلاً من تنفيذها الآن
        SendOrderConfirmationEmail::dispatch($order);

        // إعادة استجابة فورية للمستخدم دون انتظار انتهاء إرسال البريد
        return response()->json(['status' => 'Order placed successfully']);
```

```
    }
}
```

في هذا الكود داخل دالة `placeOrder` , بعد إتمام منطق إنشاء الطلب، قمنا بوضع مهمة إرسال البريد في الطابور بواسطة `SendOrderConfirmationEmail::dispatch($order)` . هذا يعني أن المستخدم سيتلقى استجابة **فورية** بتأكيد نجاح طلبه، بينما تتم عملية إرسال البريد في الخلفية عبر العامل (Worker) المنفذ للمهام في الـQueue [15] . بهذه الطريقة، لو ارتفع عدد الطلبات بشكل كبير، لن تتكدّس عمليات إرسال البريد ضمن طلب المستخدم نفسه، بل ستُدار بشكل منفصل في الخلفية، مما **يحسن زمن استجابة الـAPI بشكل ملحوظ** ويساعد على تحمل عدد أكبر من الطلبات المتزامنة.

Laravel Horizon هي أداة لوحة تحكم قوية توفرها Laravel لإدارة ومراقبة نظام الطوابير المستند إلى Redis. بدلاً من تشغيل العمال يدويًا باستخدام أمر `php artisan queue:work` ، يتيح Horizon تشغيل العمال كخدمات مستقرة ومراقبتهم عبر واجهة رسومية، مع دعم استراتيجيات موازنة تلقائية للعمال. في ملف الإعدادات `config/horizon.php` يمكن تعريف إعدادات **المشرفين (Supervisors)** الذين يديرون العمال. على سبيل المثال، نستطيع تحديد عدد العمال الأدنى والأقصى لكل صف، وتفعيل الموازنة التلقائية *(auto balancing)* للعمال بين الطوابير المختلفة. يوضح المقطع التالي جزءًا من إعدادات Horizon في بيئة الإنتاج، يعرّف مشرفًا يقوم بإدارة صفّين (`default` و `emails`) باستخدام إستراتيجية الموازنة التلقائية:

```
/* مقتطف من ملف config/horizon.php */
'environments' => [
    'production' => [
        'supervisor-1' => [
            'connection' => 'redis',
            'queue'      => ['default', 'emails'],
            'balance'    => 'auto',    // تفعيل الموازنة التلقائية بين الطوابير
            'minProcesses'    => 1,    // عدد العمال الأدنى لكل صف
            'maxProcesses'    => 10,   // العدد الكلي الأقصى للعمال لهذا المشرف
            'balanceMaxShift' => 1,    // أقصى زيادة أو نقصان في عدد العمال كل فترة
            'balanceCooldown' => 3,    // فترة التحقق (بالثواني) لتعديل عدد العمال
        ],
    ],
],
```

في هذا الإعداد حددنا أن Horizon سيستخدم **إستراتيجية Auto Balancing** للموازنة بين صفَّي `default` و `emails` . معنى ذلك أن Horizon سيقوم **بتخصيص عدد العمال تلقائيًا لكل صف بناءً على حجم العمل الحالي** في الصف [16] . مثلًا، إذا تراكمت 1000 مهمة في صف `emails` بينما صف `default` فارغ، فسيتم **تحويل المزيد من العمال للعمل على صف** `emails` إلى حين إنهاء المهام المتراكمة [17] . يساعدنا ذلك في الاستغلال الأمثل للموارد: العمال لا يبقون عاطلين في صف فارغ بينما صف آخر مزدحم بالمهام. أيضًا قمنا بتقييد العدد الأقصى للعمال بـ10 لضمان عدم استنزاف موارد الخادم بشكل مفرط، وتحديد زيادة أو نقصان بمقدار عامل واحد كل 3 ثوانٍ (`balanceMaxShift` و `balanceCooldown`) حتى تتم عملية **توسيع العمال أو تقليصهم تدريجيًا** وليس بشكل مفاجئ. هذه المرونة في ضبط العمال تلقائيًا حسب الحمل تجعل نظام الصف في Laravel قادرًا على مواكبة ارتفاع الضغط بشكل ديناميكي دون تدخل يدوي.

باختصار، الجمع بين **نظام الطوابير** و **Horizon** في Laravel يوفر حلاً عمليًا قويًا للتعامل مع المهام الكثيفة في الخلفية. فهو يضمن عدم تأثر واجهة المستخدم الأمامية ببطء العمليات الثقيلة، وفي نفس الوقت يتيح إدارة تلك العمليات الخلفية بشكل قابل للتوسع والتحكم. وهذا يقودنا إلى رؤية كيف يُطبَّق كل ما سبق في سيناريو واقعي.

سيناريو عملي: توسيع نظام Laravel أثناء عروض تسويقية ضخمة

لنتخيل سيناريو حقيقي يواجهه مهندسو الـBackend: لدينا تطبيق Laravel يقدم **خدمة API لتسجيل الطلبات** في متجر إلكتروني. في الأيام العادية، يتعامل النظام مع مئات من الطلبات في الساعة. ولكن عند حدوث **عرض ترويجي ضخم** (مثل تخفيضات موسمية أو حملة إعلانية كبيرة)، قد يقفز عدد الطلبات إلى **عشرات الآلاف في الدقيقة** . هذا الضغط الهائل المفاجئ يشكّل تحديًا كبيرًا على بنية النظام. كيف نتأكد أن نظامنا سيصمد ويعمل بسلاسة في ظل هذا الحمل غير الاعتيادي؟

بدون تطبيق مبادئ قابلية التوسع وإدارة الحمل، ربما يعتمد النظام على خادم واحد لمعالجة كل الطلبات وإجراء كل العمليات (كتسجيل الطلب في قاعدة البيانات، إرسال تأكيد عبر البريد الإلكتروني، تحديث المخزون، ... إلخ). في حالة هجوم طلبات كثيف، **سيتجاوز الحمل طاقة ذلك الخادم بسرعة** مما يؤدي إلى بطء شديد في الاستجابة أو حتى انهيار الخدمة بالكامل. وقد يحدث **انقطاع تام (Downtime)** يخسر فيه المتجر طلبات وزبائن (وقد ينتقل العملاء إلى منافس سريع الاستجابة) [14] .

لحسن الحظ، بتطبيق ما تعلمناه، يمكن تصميم النظام للتعامل مع هذا السيناريو كما يلي:

- **استخدام التوسعة الأفقية وموازن الحمل** : قبل بدء الحملة الترويجية، نتأكد أن التطبيق يعمل على عدة خوادم (عدة نسخ من تطبيق Laravel) بدلًا من خادم واحد. يتم وضع **موازن حمل أمامي** يستقبل كل طلبات الـAPI ويوزعها على هذه الخوادم بشكل متوازن [14] . مثلًا، إذا كان لدينا 5 خوادم تطبيق، سيتولى كل منها جزءًا من الطلبات الواردة بحيث لا ينهار أي منها منفردًا. أثناء ذروة الضغط، يمكن للبنية السحابية **إطلاق خوادم جديدة تلقائيًا (Auto-Scaling)** استجابةً لمؤشرات مثل ارتفاع معدل طلبات التسجيل أو ازدياد استهلاك المعالج. موازن الحمل سيكتشف هذه الخوادم الجديدة ويبدأ توجيه جزء من الترافيك إليها فورًا [14] . وبهذا نضمن أن طاقة المعالجة الإجمالية للنظام تزداد بما يتلاءم مع عدد المستخدمين المتصلين.

- **تقسيم المهام باستخدام الطوابير** : عند استلام أي طلب شراء عبر الـAPI في مثل هذه الظروف، من المهم **تقليل ما يتم إنجازه في اللحظة نفسها ضمن ذلك الطلب** قدر الإمكان. لذا، نطبق مبدأ **Queue-based decoupling** : فبمجرد إنشاء الطلب وتخزينه في قاعدة البيانات، نقوم مباشرةً **بإرسال رد التأكيد للمستخدم** (مثلاً رسالة بنجاح عملية الشراء) دون تأخير. أما العمليات الأخرى التي يمكن تأجيلها قليلًا – مثل إرسال بريد إلكتروني بتفاصيل الفاتورة، أو تحديث نظام المخزون الداخلي، أو معالجة عملية الدفع مع بوابة خارجية – فهذه **توضع كمهام في صفوف انتظار** ليتم تنفيذها تباعًا في الخلفية. في لحظات الذروة، قد تتراكم آلاف المهام في الطوابير، لكن هذا لا يؤثر على سرعة الـAPI في إرجاع الاستجابات الأساسية للمستخدمين. يتم تشغيل عدة **عمّال (Workers)** بالتوازي via Laravel Horizon مثلًا لمعالجة هذه الطوابير: يمكن زيادة عدد العمال بسهولة (يدويًا أو تلقائيًا) بحيث يعمل 20 أو 30 عاملًا في نفس الوقت لمعالجة المهام المتراكمة بسرعة. Horizon سيقوم بموازنة هؤلاء العمال بين المهام المختلفة؛ فمثلًا إذا كان معظم الضغط على مهام إرسال البريد، سيتم تخصيص غالبية العمال على صف البريد لضمان إرسال أكبر عدد من الرسائل في أقصر وقت [16] .

- **تحسينات بنيوية أخرى** : في الخلفية، قد نكون قمنا أيضًا **بتحسين قاعدة البيانات** للتعامل مع هذا الضغط عبر وسائل متعددة. على سبيل المثال: استخدام **نسخ للقراءة فقط (Read Replicas)** لتوزيع استعلامات القراءة المكثفة عبر عدة قواعد بيانات، أو **تفعيل التخزين المؤقت (Caching)** لنتائج بعض الاستعلامات الشائعة لتقليل الضغط على قاعدة البيانات أثناء الحمل العالي. هذه التحسينات تزيد من قدرة التحمل الكلية للنظام في أوقات الذروة.

نتيجةً لهذا التصميم، عند بدء العرض الضخم وتوافد الآلاف من المستخدمين في نفس الوقت، يبقى نظامنا مستجيبًا وقادرًا على الخدمة. **موازن الحمل** يوزع السيل الكبير من طلبات الـAPI على مجموعة الخوادم، والتي يمكنها مجتمعةً معالجة عدد هائل من الطلبات المتوازية. **الطوابير** تمنع المهام غير الحرجة من إبطاء معالجة الطلبات الأساسية، و**العمال** في الخلفية يعتنون بتلك المهام بوتيرة تناسب حجمها. إذا زاد الضغط أكثر من المتوقع، تقوم آلية **التوسعة التلقائية** بإطلاق خوادم Laravel إضافية وربما زيادة عدد عمليات العمال أيضًا، مما يمنحنا قدرة معالجة إضافية في الوقت المناسب. وبانتهاء فترة الضغط، يمكن تقليص عدد الخوادم تلقائيًا لتوفير التكاليف.

هذا السيناريو الواقعي يُظهر كيف تؤدي قابلية التوسع وإدارة الحمل إلى **نظام مستقر وفعّال حتى تحت أصعب الظروف** . فبدلاً من انهيار الموقع أثناء حملة النجاح – وهو كابوس لأي عمل تجاري – يستطيع النظام خدمة العملاء بسلاسة، مما يزيد من ثقتهم وولائهم. وقد أشارت إحدى الدراسات التقنية إلى مثال لشركة تذاكر حفلات واجهت موقفًا مشابهًا؛ حيث بدون موازن حمل كانت القدرة محدودة جدًا، ولكن مع التخطيط للتوسعة الأفقية عبر موازن حمل أمكن استيعاب التدفق الهائل من الطلبات بحيث **تمكن عدد أكبر من العملاء من الحصول على طلباتهم بنجاح** [14] .

المصطلحات الأساسية (إنجليزي – عربي)

فيما يلي قائمة بأهم المصطلحات التي تم تناولها في هذه المحاضرة، مع شرح مبسط لها:

- **Scalability (قابلية التوسع)** : قدرة النظام على **النمو والتكيف** عند زيادة الحمل أو عدد المستخدمين **دون تدهور في الأداء** [2] . نظام قابل للتوسع يستمر بالعمل بكفاءة (وأحيانًا يتحسن أداؤه) مع زيادة workloads أو حجم البيانات.

- **Horizontal Scaling (التوسعة الأفقية)**: **زيادة قدرة النظام بإضافة خوادم/عقد جديدة** وتوزيع الحمل عليها [3] . مثال ذلك إضافة خوادم تطبيق إضافية خلف موازن حمل لخدمة المزيد من المستخدمين. تساهم التوسعة الأفقية في تحسين التوفرية وإزالة نقطة الفشل الواحدة، لأنها لا تعتمد على جهاز واحد.

- **Vertical Scaling (التوسعة الرأسية)**: **زيادة قدرة النظام بترقية موارد الخادم الموجود** [18] . أي استخدام جهاز أقوى (معالج أسرع، ذاكرة أكبر، تخزين أكبر) لاستيعاب حمل أكبر على نفس الخادم. هذا النهج بسيط التنفيذ ولكنه محدود بالسقف المادي لمواصفات الجهاز، ويبقي النظام معتمدًا على خادم واحد.

- **Load Balancer (موازن الحمل)** : جهاز أو برنامج يعمل على **توزيع الحركة الواردة** (طلبات المستخدمين) عبر عدة خوادم خلفه [10] . يهدف إلى ضمان عدم اكتظاظ خادم واحد، وتحسين سرعة الاستجابة والتوفرية. يقوم موازن الحمل بمراقبة صحة الخوادم وتوجيه الطلبات فقط إلى الخوادم السليمة، مما يساعد في تجاوز أعطال الخوادم الفردية بشكل شفاف للمستخدم.

- **Auto-Scaling (التوسعة التلقائية)** : خاصية (غالبًا في البنى السحابية) تقوم **بزيادة أو تقليص عدد الموارد تلقائيًا** وفقًا لمستوى الطلب الحالي. مثلًا، زيادة عدد الخوادم تلقائيًا عند ارتفاع الضغط وتقليصها عند انخفاضه [13] . الهدف هو امتلاك **موارد مرنة** "تتمدد وتتقلص" حسب الحاجة، مما يحقق كفاءة في التكلفة وضمان استعداد الخدمة للذروة.

- **Queue / Queue-Based Decoupling (الصف / الفصل عبر الطوابير)**: **الطابور** في الحوسبة هو آلية تنتظر فيها المهام ليتم معالجتها بشكل غير متزامن بواسطة عمال خلفيين. أما **الفصل باستخدام الطوابير** فهو نمط تصميمي نقوم فيه **بفصل المهام الثقيلة عن مسار التنفيذ الرئيسي** بوضعها في صف لمعالجتها لاحقًا [15] . هذا الأسلوب يمنع المهام البطيئة (مثل إرسال بريد أو معالجة ملف) من تعطيل استجابة المستخدم اللحظية، مما **يحسّن قدرة النظام على تحمل أعداد كبيرة من الطلبات** عن طريق توزيع العمل بمرور الوقت وعلى موارد مختلفة.

- **Load Management (إدارة الأحمال)** : مصطلح يشير إلى **مجموعة الممارسات والتقنيات** المستخدمة لمراقبة توزّع الحمل (الطلبات أو العمليات) عبر النظام والتدخل لضبطه بشكل **يمنع حدوث اختناقات أو انهيارات** . يشمل ذلك استخدام موازنات الحمل، والطوابير، والتوسعة التلقائية، وتوزيع موارد قاعدة البيانات، وغيرها لضمان أن كل جزء من النظام يعمل ضمن طاقته ولا يتجاوزها.

الملاحظات والتقييم الذاتي

هذه الخانة مخصصة لتدوين أي ملاحظات شخصية من قبلك كمُطوِّر، وكذلك لتقييم فهمك للمفاهيم المطروحة. يمكنك مثلاً طرح أسئلة على نفسك ومحاولة الإجابة عليها للتحقق من استيعاب المادة:

- ما الفرق الجوهري بين التوسعة الأفقية والتوسعة الرأسية، وما مزايا وعيوب كل منهما في سياق نظام حقيقي قمت بالعمل عليه؟
- كيف يساهم موازن التحميل في تحسين توافرية النظام وفي قدرته على التوسع؟ حاول ذكر سيناريو قصير يوضح ذلك.
- قم بشرح خطوات ما يحدث عند إرسال طلب إلى نظام يحتوي على موازن حمل وعدة خوادم معالجة (من لحظة دخول الطلب إلى وصول الاستجابة للعميل).
- كيف تستفيد من نظام الـQueue في Laravel لضمان عدم انهيار التطبيق تحت ضغط آلاف المهام المفاجئة؟ وما الدور الذي تلعبه أداة Horizon في هذا السياق؟

إجابتك على هذه الأسئلة ومناقشة أفكارك مع زملائك ستعزز فهمك لقابلية التوسع وإدارة الحمل، وتساعدك على تطبيق هذه المفاهيم عمليًا في مشاريعك الحالية والمستقبلية. حافظ على هذه الملاحظات وراجعها دوريًا لتتأكد من جاهزيتك للتعامل مع تحديات النمو والضغط على الأنظمة البرمجية.

---

1 Developer Nation Community
/https://www.developernation.net/blog/scaling-laravel-applications-handling-high-traffic-and-performance-challenges

2 قابلية التوسع - ويكيبيديا
/https://ar.wikipedia.org/wiki
D9%82%D8%A7%D8%A8%D9%84%D9%8A%D8%A9_%D8%A7%D9%84%D8%AA%D9%88%D8%B3%D8%B9%

3 18 الفرق بين المرونة وقابلية التوسع في الحوسبة السحابية - The Codest
-https://thecodest.co/ar/blog/%D8%A7%D9%84%D9%81%D8%B1%D9%82-%D8%A8%D9%8A%D9%86
-D8%A7%D9%84%D9%85%D8%B1%D9%88%D9%86%D8%A9-%D9%88%D9%82%D8%A7%D8%A8%D9%84%D9%8A%D8%A9%
/D8%A7%D9%84%D8%AA%D9%88%D8%B3%D8%B9-%D9%81%D9%8A-%D8%A7%D9%84%

4 5 6 7 Vertical vs. horizontal scaling: What's the difference and which is better?
/https://www.cockroachlabs.com/blog/vertical-scaling-vs-horizontal-scaling

8 9 13 Horizontal and Vertical Scaling | System Design - GeeksforGeeks
/https://www.geeksforgeeks.org/system-design/system-design-horizontal-and-vertical-scaling

10 11 12 Introduction to Load Balancer - GeeksforGeeks
/https://www.geeksforgeeks.org/system-design/what-is-load-balancer-system-design

14 What Is a Load Balancer? | F5
https://www.f5.com/glossary/load-balancer

15 Laravel Queues in High-Traffic Applications: Setup & Cost - Abbacus Technologies
/https://www.abbacustechnologies.com/laravel-queues-in-high-traffic-applications-setup-cost