1v0 Ship v1.0 Faster

How to Use Laravel Queues for Faster Performance

How to Use Laravel Queues for Faster Performance

When your Laravel app handles tasks like sending emails, generating reports, or processing uploads, running them inside the main request slows down response times. **Queues** let you offload heavy tasks to background workers so users get instant responses. In this guide, we'll configure queues, create jobs, run workers, and monitor them effectively.

1 – Configure Queue Driver

Laravel supports multiple queue backends like `database`, `redis`, and `beanstalkd`. Redis is most common in production.

.env QUEUE_CONNECTION=redis

This sets Redis as the default queue driver. Make sure Redis is installed and running on your server. For a deep dive into different cache/queue stores, see Caching Strategies in Laravel: Redis vs Database vs File.

2 – Create a Job

Jobs encapsulate the work you want to run in the background. Use Artisan to generate one:

```
php artisan make:job SendWelcomeEmail
```
Code language: Bash (bash)

This creates a new class in `app/Jobs/`. You can customize it to perform specific logic.

```
// app/Jobs/SendWelcomeEmail.php
namespace App\Jobs;

use App\Mail\WelcomeMail;
use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Foundation\Bus\Dispatchable;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Queue\SerializesModels;
use Illuminate\Support\Facades\Mail;

class SendWelcomeEmail implements ShouldQueue
{
    use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;

    public function __construct(public $user) {}

    public function handle()
    {
        Mail::to($this->user->email)
            ->send(new WelcomeMail($this->user->name));
    }
}Code language: PHP (php)
```

This job sends a welcome email. Because it implements `ShouldQueue`, Laravel automatically pushes it into the queue instead of running synchronously.

3 – Dispatch Jobs

You can dispatch jobs from controllers, events, or service classes.

```
// app/Http/Controllers/UserController.php
use App\Jobs\SendWelcomeEmail;

public function store(Request $request)
{
    $user = User::create($request->all());

    SendWelcomeEmail::dispatch($user);

    return response()->json(['message' => 'User created!']);
}Code language: PHP (php)
```

Here, when a new user registers, the welcome email job is dispatched. The controller immediately returns a response while the email is sent in the background.

4 – Run Queue Workers

Queue workers listen for jobs and process them as they come in.

```
php artisan queue:workCode language: Bash (bash)
```

This runs a worker that listens to the default connection (`redis`). In production, use a process manager like Supervisor or systemd to keep workers alive. For advanced monitoring, check How to Use Laravel Horizon for Queue Monitoring.

5 – Failed Jobs

When jobs fail (e.g., email server down), Laravel can log them for later retries.

```
php artisan queue:failed-table
php artisan migrateCode language: Bash (bash)
```

This creates a `failed_jobs` table. Workers automatically log failed jobs here. You can retry them with:

```
php artisan queue:retry allCode language: Bash (bash)
```

This is useful for handling temporary outages—retrying jobs once the external service is back online.

6 – Delayed & Chained Jobs

You can delay execution or chain multiple jobs to run in sequence.

```
// Delay by 10 minutes
SendWelcomeEmail::dispatch($user)->delay(now()->addMinutes(10));

// Chain jobs
ProcessImage::withChain([
    new ResizeImage($path),
    new UploadToS3($path),
])->dispatch();Code language: PHP (php)
```

Delays are useful for reminder emails, while chains help orchestrate multi-step workflows like image processing pipelines.

7 – Queues in High-Traffic Apps

Queues are critical in high-traffic apps to offload CPU-heavy tasks. Combine queues with caching, indexing, and Octane for best results. For a broader overview, see 10 Proven Ways to Optimize Laravel for High Traffic.

Wrapping Up

You've learned how to configure queue drivers, create jobs, dispatch them, run workers, handle failures, and chain tasks. With queues, you decouple heavy work from user-facing requests, making apps feel instant even under load. Always monitor queues and use retry logic to handle external service downtime gracefully.

What's Next

- How to Use Laravel Horizon for Queue Monitoring — manage and visualize your queue system in production.
- Caching Strategies in Laravel: Redis vs Database vs File — combine queues with efficient caching for optimal speed.
- Optimizing Laravel for High Concurrency with Octane — scale your queues and workers for thousands of requests per second.