How to Use Laravel Horizon for Queue Monitoring

How to Use Laravel Horizon for Queue Monitoring

When your Laravel app uses queues to handle heavy workloads (emails, reports, notifications), monitoring those jobs becomes essential. **Laravel Horizon** provides a beautiful dashboard to manage, monitor, and scale Redis-based queues. In this guide, we'll install Horizon, configure it, run workers, and explore key features for monitoring high-performance apps.

1 – Install Horizon

Horizon works with Redis queues, so ensure your app uses `QUEUE_CONNECTION=redis`. Then install Horizon:

```
composer require laravel/horizon
php artisan horizon:install
php artisan migrateCode language: Bash (bash)
```

This installs Horizon's migrations for monitoring job batches and failed jobs. Horizon also publishes a config file for customizing queues and balancing.

2 – Running Horizon

Start Horizon just like a queue worker, but with its own dashboard and balancing features:

```
php artisan horizonCode language: Bash (bash)
```

By default, Horizon runs workers and exposes a dashboard at `/horizon`. It automatically restarts workers after code changes, unlike `queue:work`.

3 – Horizon Dashboard

Visit `/horizon` in your app to see job metrics. Horizon shows:

- Current and recent jobs
- Failed jobs
- Processing times
- Throughput and retries
- Which worker processed each job

This visibility helps diagnose bottlenecks—like a failing job blocking a queue or an overloaded worker pool. For background on queues, see How to Use Laravel Queues for Faster Performance.

4 – Configuring Supervisors

Horizon uses **supervisors** to manage how many workers are assigned to a queue. Configure them in `config/horizon.php`:

```
// config/horizon.php (snippet)
'supervisors' => [
    'app-supervisor' => [
```

```
        'connection' => 'redis',
        'queue' => ['default', 'emails'],
        'balance' => 'auto',
        'minProcesses' => 2,
        'maxProcesses' => 10,
        'tries' => 3,
    ],
],
```

Code language: PHP (php)

This supervisor auto-scales between 2–10 workers depending on queue load. The `balance` setting dynamically adjusts worker counts for optimal throughput.

5 – Monitoring Failed Jobs

Horizon makes it easy to retry failed jobs directly from the dashboard. You can also configure alerts for specific job failures.

```
// app/Providers/HorizonServiceProvider.php
namespace App\Providers;

use Illuminate\Support\ServiceProvider;
use Laravel\Horizon\Horizon;

class HorizonServiceProvider extends ServiceProvider
{
    public function boot(): void
    {
        Horizon::routeMailNotificationsTo('admin@example.com');
        Horizon::routeSlackNotificationsTo('slack-webhook-url');
    }
}
```

Code language: PHP (php)

This configures Horizon to send notifications when jobs fail repeatedly. You can use mail, Slack, or custom notification channels for alerts.

6 – Horizon and High Concurrency

In high-traffic apps, Horizon works well with Octane. Octane reduces response latency, while Horizon balances and monitors background job throughput. Together they keep both request handling and background tasks efficient.

Wrapping Up

Laravel Horizon gives you visibility and control over queued jobs. We installed Horizon, ran the dashboard, configured supervisors, and set up alerts. With Horizon, Redis queues become not only powerful but also transparent—helping you scale workloads confidently. Pair Horizon with Octane and Redis caching for maximum performance under heavy load.

What's Next

- How to Use Laravel Queues for Faster Performance — foundational guide before adding Horizon.
- Caching Strategies in Laravel: Redis vs Database vs File — Redis powers Horizon

queues and cache.
- Using Laravel Telescope to Debug Performance Issues — monitor requests and jobs alongside Horizon.