Caching Strategies in Laravel: Redis vs Database vs File

Caching Strategies in Laravel: Redis vs Database vs File

Caching is one of the fastest ways to improve performance in high-traffic Laravel apps. Laravel supports multiple cache drivers—**file**, **database**, **Redis**, and more. Choosing the right one can reduce response times dramatically. In this guide, we'll compare File, Database, and Redis caching, show you how to configure them, and explain when to use each.

1 – Configuring the Cache Driver

You can set the cache driver in your `.env` file. Common options are `file`, `database`, and `redis`.

.env CACHE_DRIVER=file # CACHE_DRIVER=database # CACHE_DRIVER=redis

This tells Laravel where to store cached data. Switch drivers by changing the `CACHE_DRIVER` value and clearing the config cache.

2 – File Cache

The default driver in Laravel is `file`. It stores cache data in the `storage/framework/cache` directory.

```
// Store value
Cache::put('homepage_posts', $posts, 300);

// Retrieve value
$posts = Cache::get('homepage_posts');Code language: PHP (php)
```

File cache is easy to set up, but reading/writing to disk becomes slow under heavy load. Suitable for small apps or low-traffic projects.

3 – Database Cache

Database caching stores key-value pairs in a database table. You'll need to create the table first:

```
php artisan cache:table
php artisan migrateCode language: Bash (bash)
```

This creates a `cache` table in your database. It's more reliable than file caching in clustered environments, but slower than memory-based caches like Redis.

```
// config/cache.php (snippet)
'default' => env('CACHE_DRIVER', 'database'),Code language: PHP (php)
```

Switch to the `database` driver in `config/cache.php` or via `.env`.

4 – Redis Cache

Redis is an in-memory data store and the fastest option for Laravel caching. Install the PHP extension and client package.

```
composer require predis/predisCode language: Bash (bash)
```

```
# .env CACHE_DRIVER=redis REDIS_HOST=127.0.0.1 REDIS_PASSWORD=null REDIS_PORT=6379
```

Redis stores cache in memory, making reads/writes extremely fast. It's ideal for high-traffic applications. For advanced usage like tagging and queues, Redis is the recommended option. For queue-specific performance, see How to Use Laravel Queues for Faster Performance.

5 – Cache Tags

With Redis or Memcached, you can group related cache entries using tags and flush them together.

```
// Store with tag
Cache::tags(['posts', 'featured'])
    ->put('post_1', $post, 600);

// Flush all "posts" cache
Cache::tags('posts')->flush();Code language: PHP (php)
```

Cache tags are useful for invalidating multiple related keys without clearing the entire cache.

6 – Cache in Controllers & APIs

You can use caching directly in controllers to speed up API responses.

```
// app/Http/Controllers/PostController.php
public function index()
{
    $posts = Cache::remember('posts.all', 300, function () {
        return Post::with('author')->latest()->take(10)->get();
    });

    return response()->json($posts);
}Code language: PHP (php)
```

This caches the query for 5 minutes. APIs can serve thousands of requests without hammering the database. For advanced query optimization, see How to Speed Up Laravel with Database Indexing.

Wrapping Up

Caching is the easiest and most effective optimization in Laravel. **File cache** is simple but limited, **database cache** works in multi-server setups but can be slower, and **Redis** provides blazing-fast performance for high-traffic apps. Choose based on scale and infrastructure. Combine caching with queues and Octane for best results.

What’s Next

- 10 Proven Ways to Optimize Laravel for High Traffic — overview of caching, queues, and scaling strategies.
- How to Use Laravel Queues for Faster Performance — offload heavy jobs to workers.
- Optimizing Laravel for High Concurrency with Octane — complement caching with in-memory server performance.