



Demystifying Nginx and PHP-FPM for PHP Developers



Maikel González Baile · Follow

10 min read · Mar 24, 2023



Listen



Share

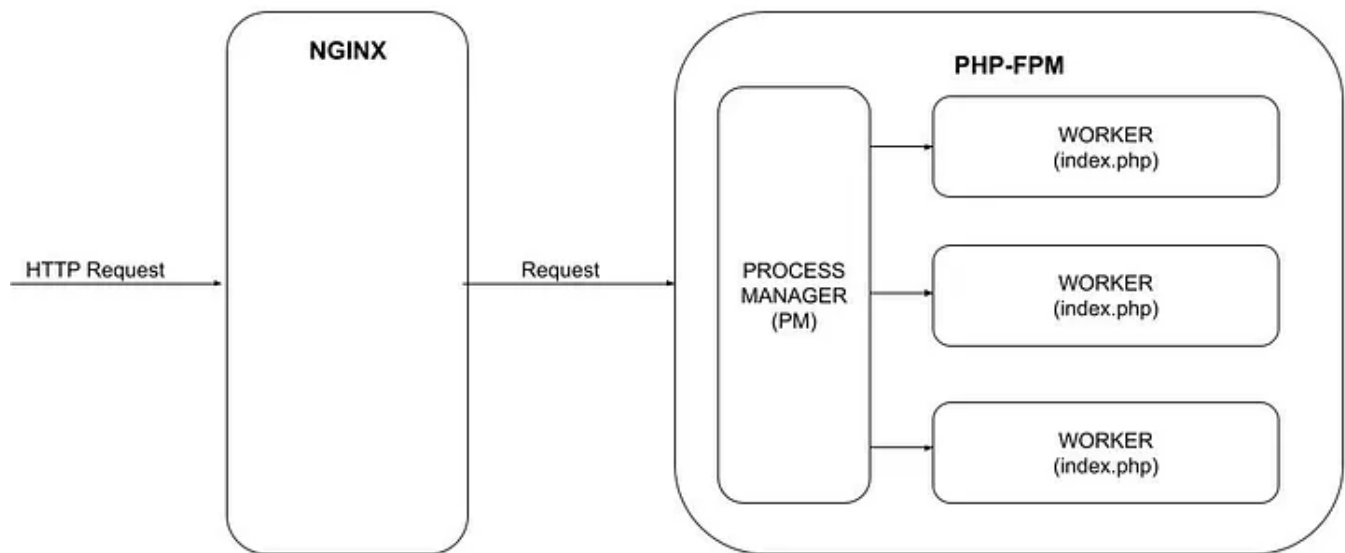
Introduction

Developers often have varying levels of experience and knowledge when it comes to the server-side execution of their code. This is attributed to the fact that developers typically focus their time on coding new features, while tasks related to deploying applications, setting up servers, and tuning the servers' configuration are often handled by SREs. Regardless of your background, gaining a deeper understanding of the standard PHP Web Architecture involving Nginx and PHP-FPM can be beneficial. In this blog post, we'll explore these server-side components that bring your applications to life and will walk through some of the key configuration parameters that can allow you to fine-tune your system for maximum performance.

Note: There are many more configuration options available for both Nginx and PHP-FPM that can help you improve your application's performance, which are not covered in this post.

PHP Web Architecture

Although the PHP core includes a built-in server that you can use for development purposes while working on your local environment, it is not suitable for production environments due to its single-threaded nature that limits it to execute one single request at a time, among others lacking capabilities related to security, caching, scaling, and so on. As a result, PHP relies on an additional component, PHP-FPM, for managing multiple processes and ultimately on Nginx to efficiently serve web applications.



PHP-FPM: The FastCGI Process Manager

PHP-FPM, or “PHP FastCGI Process Manager,” is an advanced, high-performance FastCGI process manager for PHP. It resolves the concurrency issue of PHP’s built-in server by spawning multiple workers, which can handle multiple requests simultaneously. However, PHP-FPM cannot directly manage incoming HTTP traffic, necessitating a web server to act as a reverse proxy.

Nginx: The High-Performance Web Server and Reverse Proxy

Nginx is a powerful, open-source web server that excels in handling incoming HTTP traffic. It functions as a reverse proxy, forwarding client requests to PHP-FPM and returning the processed responses back to the clients. Additionally, Nginx provides several advantages to a web server architecture, such as load balancing, SSL termination, and serving static files, enhancing the overall performance and reliability of your web applications.

In conclusion, **understanding the relationship between Nginx and PHP-FPM** will enable you to make informed decisions about your PHP application’s architecture and deployment. By leveraging these powerful components, you can ensure your applications run efficiently and reliably, even in demanding production environments.

Tuning PHP-FPM

Process Manager Strategies: On-demand, Static, and Dynamic

Before diving into each strategy, it’s essential to understand some key concepts related to process managers, workers, and the life cycle of a request.

The life cycle of a request starts when it arrives at the web server (Nginx). The web server forwards the request to PHP-FPM which assigns an available worker to handle it. The worker processes the request, executing the PHP script, and generates a response, which is returned to the client through the web server. A worker can be in different states:

- **Idle:** In this state, the worker is not processing any request and is waiting for new requests to arrive. The worker is available to take on new tasks as they come in.
- **Active:** In this state, the worker is executing the PHP script associated with the request.
- **Terminated:** Workers can be terminated based on the process manager strategy and configuration settings. Take a look at some of the key directives you can set in the PHP-FPM config file to optimize the resources usage of your server:
 - **process_idle_timeout:** in the on-demand and dynamic strategies, idle workers are terminated after a specified idle timeout.
 - **max_requests:** a worker is replaced after reaching a maximum number of requests handled, this is key to preventing potential memory leaks.
 - **request_terminate_timeout:** an active worker can be terminated by the process manager if the maximum time to process the request is reached, this is particularly important to ensure a responsive system that always sends a response to the client in a timely manner.
 - **php_value[memory_limit]:** determines the maximum amount of memory a worker can allocate. By limiting the memory usage, you can prevent a single worker or the entire pool from exhausting all available system resources and affecting the overall performance of your server.

The process manager is responsible for controlling the pool of workers. It determines how these workers are created, maintained, and terminated based on the chosen strategy and configuration settings. When multiple requests arrive simultaneously, the process manager distributes these requests to the available workers, improving the server's capacity to handle incoming traffic.

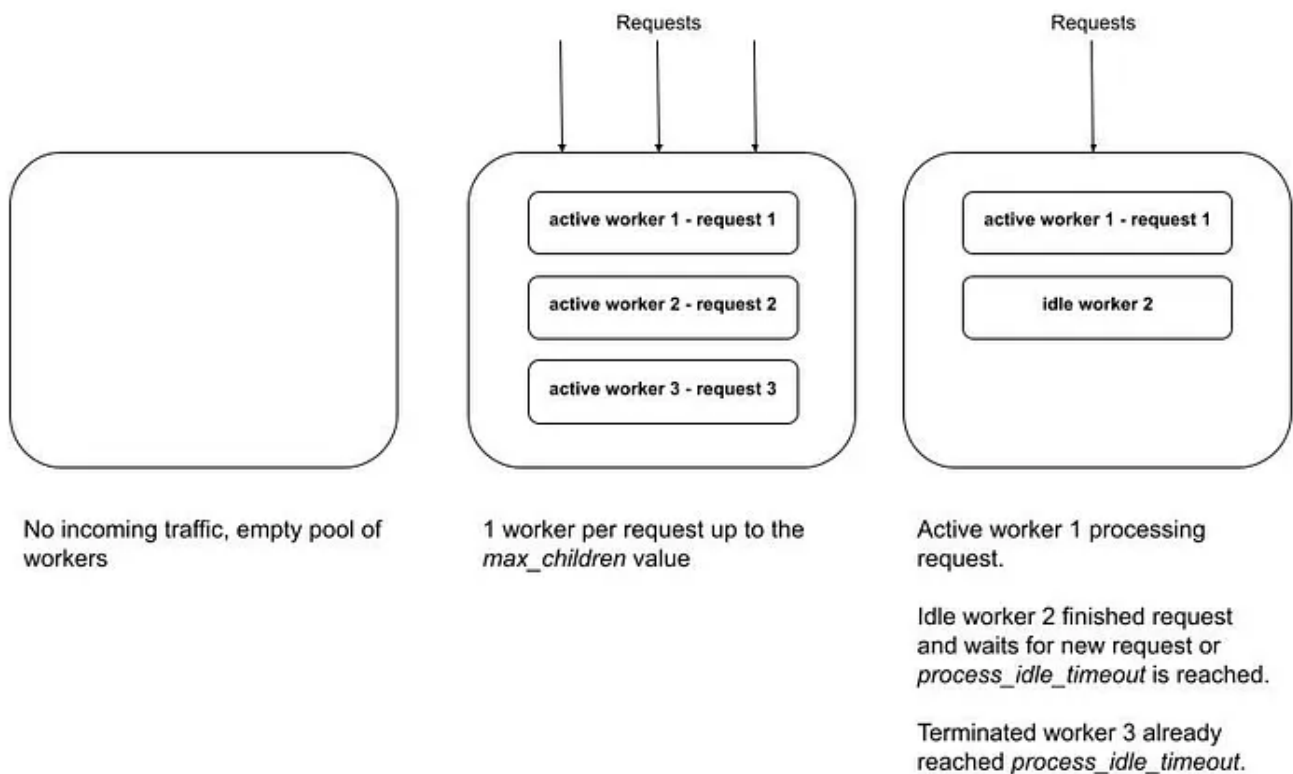
The process manager strategy impacts the potential usage of server resources such as CPU, memory, and network bandwidth. It's essential to choose a strategy that optimally utilizes available resources while maintaining a balance between responsiveness and efficiency.

Let's now dive into each of the different available strategies:

On-demand Strategy

Definition

The on-demand strategy starts with an empty pool of workers and creates worker processes only when needed. After the request is processed, idle workers terminate.



Configuration

- `pm`: Set this parameter to `ondemand` to activate the ondemand strategy.
- `pm.max_children`: Defines the maximum number of worker processes that can be created. PHP-FPM will not spawn more workers than this limit.
- `pm.process_idle_timeout`: Specifies the duration in seconds after which idle workers will be terminated.

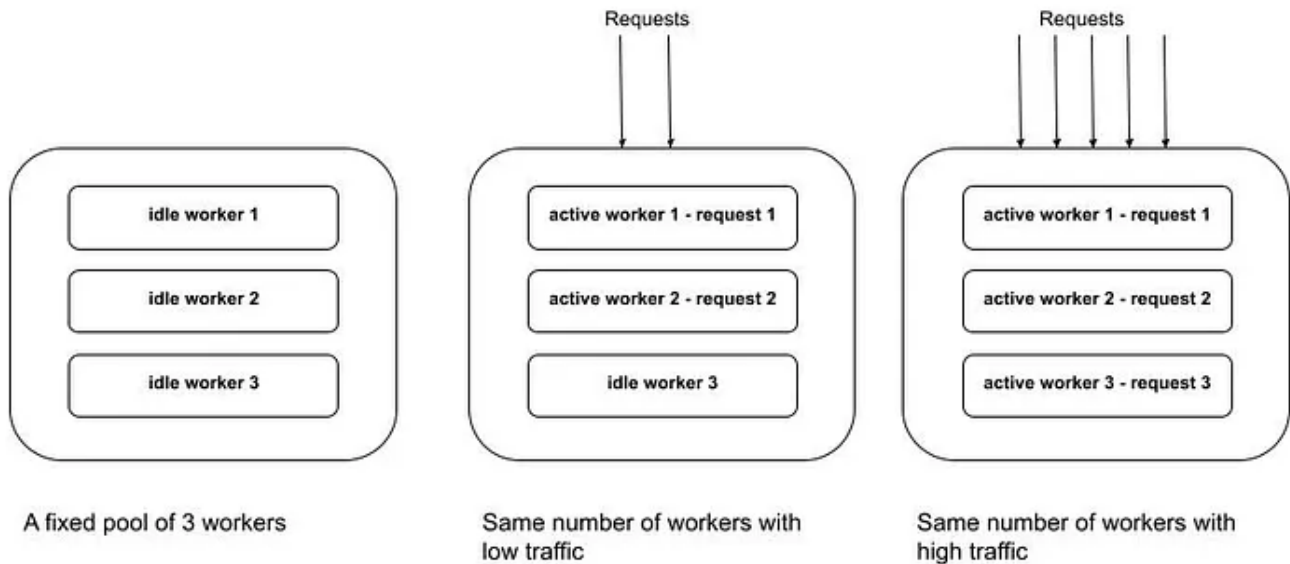
Considerations

While the on-demand strategy is resource-efficient, it can cause additional latency due to the time required for creating and terminating worker processes.

Static Strategy

Definition

The static strategy starts and maintains a fixed number of worker processes, ensuring a constant pool of workers always ready to handle incoming requests.



Configuration

- `pm`: Set this parameter to `static` to activate the static strategy.
- `pm.max_children`: Determines the fixed number of worker processes that will be maintained.

Considerations

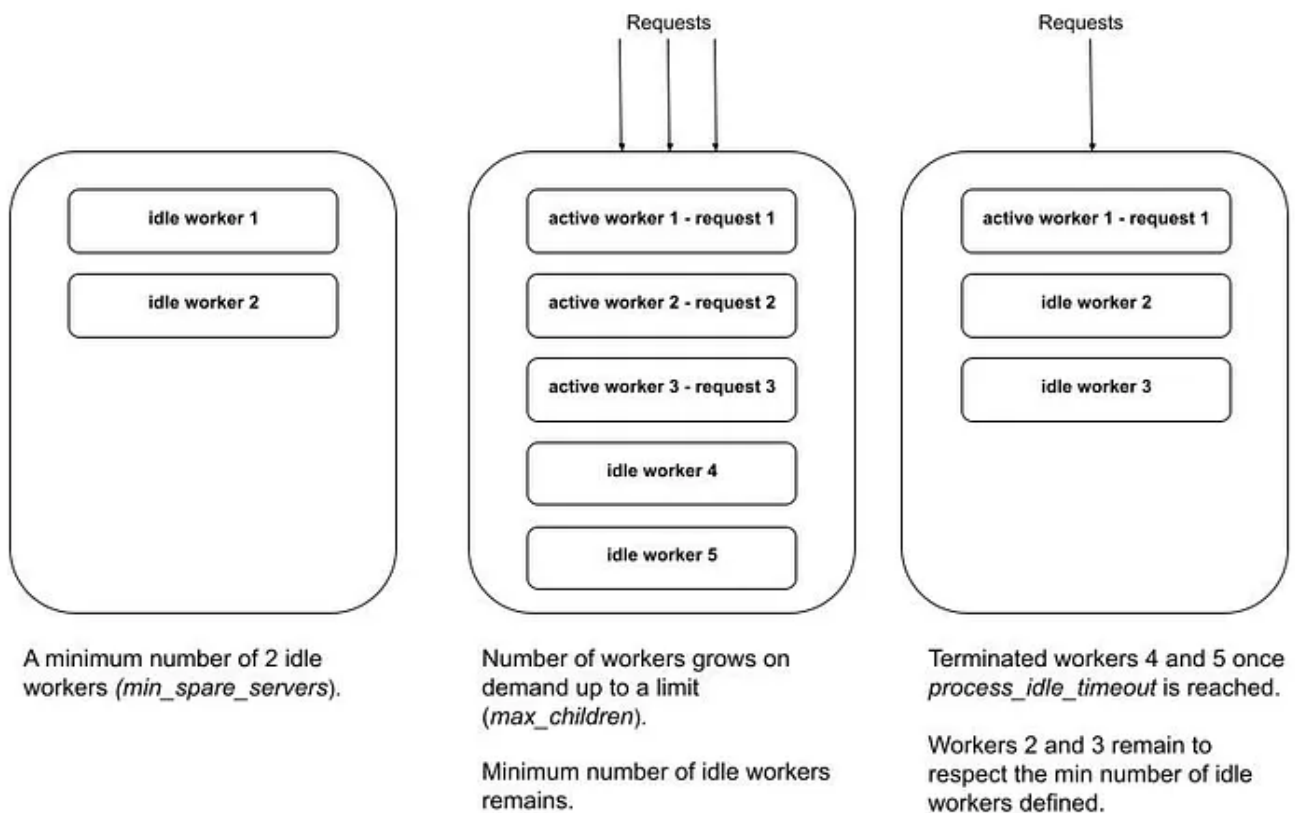
The static strategy provides consistent performance and predictable resource allocation. **The waste of resources is marginal, as idle processes consume very low resources in terms of memory and CPU.** The performance of the server is more stable and predictable in both high and low-traffic scenarios due to a fixed number of workers always ready.

The `max_requests` directive is essential in the static strategy to make sure that worker processes are restarted regularly and therefore prevent potential memory leaks.

Dynamic Strategy

Definition

The dynamic strategy provides a balance between the on-demand and static strategies. It sets a minimum number of idle workers and adjusts the pool size based on demand. It offers a compromise between resource usage and responsiveness.



Configuration

- **pm:** Set this parameter to `dynamic` to activate the dynamic strategy.
- **pm.start_servers:** Determines the initial number of worker processes to be created at startup.
- **pm.max_children:** Defines the maximum number of worker processes that can be created.
- **pm.min_spare_servers:** Sets the minimum number of idle worker processes that should be always maintained. If the number of idle workers falls below this value, PHP-FPM will create additional processes.
- **pm.max_spare_servers:** Specifies the maximum number of idle worker processes allowed. If the number of idle workers exceeds this value, PHP-FPM will terminate excess processes.

- `pm.process_idle_timeout`: Indicates the duration after which idle worker processes will be terminated. This helps maintain an optimal number of idle processes, balancing resource usage and responsiveness.

Considerations

The dynamic strategy is adaptable and can be suitable for a wide range of scenarios. However, **fine-tuning the configuration parameters is crucial for optimal performance, and can be error-prone**. Balancing the number of idle workers against system resources and traffic levels is essential, but you can face adverse situations during traffic peaks due to a wrong or non-optimal configuration. In such cases, **the static strategy might be a more appropriate choice**, as it provides more stable and predictable performance.

Tuning Nginx

Understanding how Nginx handles HTTP requests and manages workers and connections is key for optimizing your web server's performance.

Nginx handles incoming HTTP requests from clients, forwards these requests to PHP-FPM, and returns the appropriate responses back to the client. To efficiently manage these requests, Nginx utilizes workers and connections.

A worker in Nginx is a separate process responsible for handling incoming client connections and processing requests. Each worker process can manage multiple connections, allowing Nginx to handle many simultaneous requests with a relatively small number of worker processes.

A connection in Nginx refers to an individual client connection to the server. Each connection represents a single client making a request to your web application. Nginx is designed to handle a large number of connections efficiently, thanks to its event-driven architecture.

Worker Processes

The `worker_processes` directive specifies the number of Nginx worker processes that will be spawned to handle a pool of connections. Generally, it is recommended to set this value equal to the number of CPU cores available on your server. This setting ensures optimal performance by maximizing the utilization of available CPU resources. **Setting the value to auto** allows Nginx to automatically detect and use the optimal number of worker processes based on the number of available CPU cores.

Worker Connections

The `worker_connections` directive determines the maximum number of simultaneous connections that can be handled by each worker process. This value directly affects the total number of clients your server can serve simultaneously. To adjust the value, consider your server's capacity and the expected number of concurrent connections.

By understanding the roles of workers and connections in Nginx and fine-tuning the `worker_processes` and `worker_connections` directives, you can optimize your web server's performance. Keep in mind that Nginx has many other capabilities for enhancing performance and system behavior, which are not covered in this post but are crucial for web applications — learn more about Nginx optimization techniques in [this talk](#).

Monitoring and Troubleshooting

To maintain the performance and stability of your PHP applications, it's crucial to monitor and troubleshoot issues that may arise in your server environment.

Monitoring PHP-FPM

PHP-FPM provides a built-in status page that can help you monitor the performance and health of your PHP-FPM pools. By enabling the status page in your PHP-FPM configuration, you can access real-time information about your PHP-FPM pools, such as the number of active processes, idle processes, and requests in the queue.

Monitoring Nginx

Nginx also includes a built-in status module that can help you monitor the performance and health of your Nginx server. By enabling the status module in your Nginx configuration, you can access real-time information about your Nginx server, such as the number of active connections, accepted connections, and handled requests.

Troubleshooting

When issues arise, it's essential to know where to look for relevant information. Both PHP-FPM and Nginx generate log files that can help you identify and resolve problems.

PHP-FPM Log Files

PHP-FPM log files contain valuable information about errors and warnings related to your PHP applications. By default, PHP-FPM logs can be found in the `/var/log/php-fpm` directory or specified in your PHP-FPM configuration file.

Analyzing these logs can help you identify issues with your PHP code or configuration settings.

Nginx Log Files

Nginx log files, including access and error logs, can be found in the `/var/log/nginx` directory or specified in your Nginx configuration file. Access logs record all incoming requests, while error logs contain information about issues related to the Nginx server. Analyzing these logs can help you identify issues with your server configuration, resource usage, or performance bottlenecks.

By monitoring and troubleshooting your PHP-FPM and Nginx configurations, you can ensure the smooth operation of your PHP applications and maintain a high-performance server environment.

Conclusion

As a developer, building a certain level of awareness about how your code is executed and the challenges that arise in concurrent production environments is crucial. Understanding the roles of Nginx and PHP-FPM, along with their configurations and management strategies, can help you optimize your web applications and ensure their stability, responsiveness, and efficiency.

By diving deeper into the server-side components and learning how to fine-tune various settings, you can create a better experience for your users and maintain a high-performance server environment. This knowledge not only allows you to address potential issues more effectively but also fosters better collaboration between Product and SRE teams, ultimately leading to more robust and scalable applications.

Interactive Playground

To help you better understand and experiment with the concepts discussed in this post, I have created an interactive playground on GitHub. This repository allows you to easily set up the Nginx and PHP-FPM architecture and provides sample configurations that you can modify and test.

The playground includes a docker-compose setup, making it simple to get started and experiment with different settings for both Nginx and PHP-FPM. Additionally, I've included a README file containing instructions for setting up the environment and exploring various testing scenarios to help you gain hands-on experience with the configurations.

Feel free to try out different configuration values and observe their impact on your server's performance and resource usage. This hands-on approach will further strengthen your understanding of Nginx and PHP-FPM and enable you to better optimize your PHP applications.

You can access the interactive playground by visiting the following GitHub repository:

<https://github.com/mgonzalezbaile/nginx-phpfm-playground>

Don't hesitate to share your feedback, insights, or any issues you encounter while using the playground. I'll be glad to hear from you and learn from your experiences. Happy experimenting!

PHP

DevOps

Web Development



Follow

Written by Maikel González Baile

70 Followers

Senior Software Engineer | Agile, DDD & Microservices

More from Maikel González Baile



 Maikel González Baile

Implementing a use case (II)-Command Pattern

See the code here

6 min read · Jul 19, 2018



113



 Maikel González Baile

Implementing a use case (III) — Command Bus

Enhancing your use cases with extra powers easily

5 min read · Jul 26, 2018



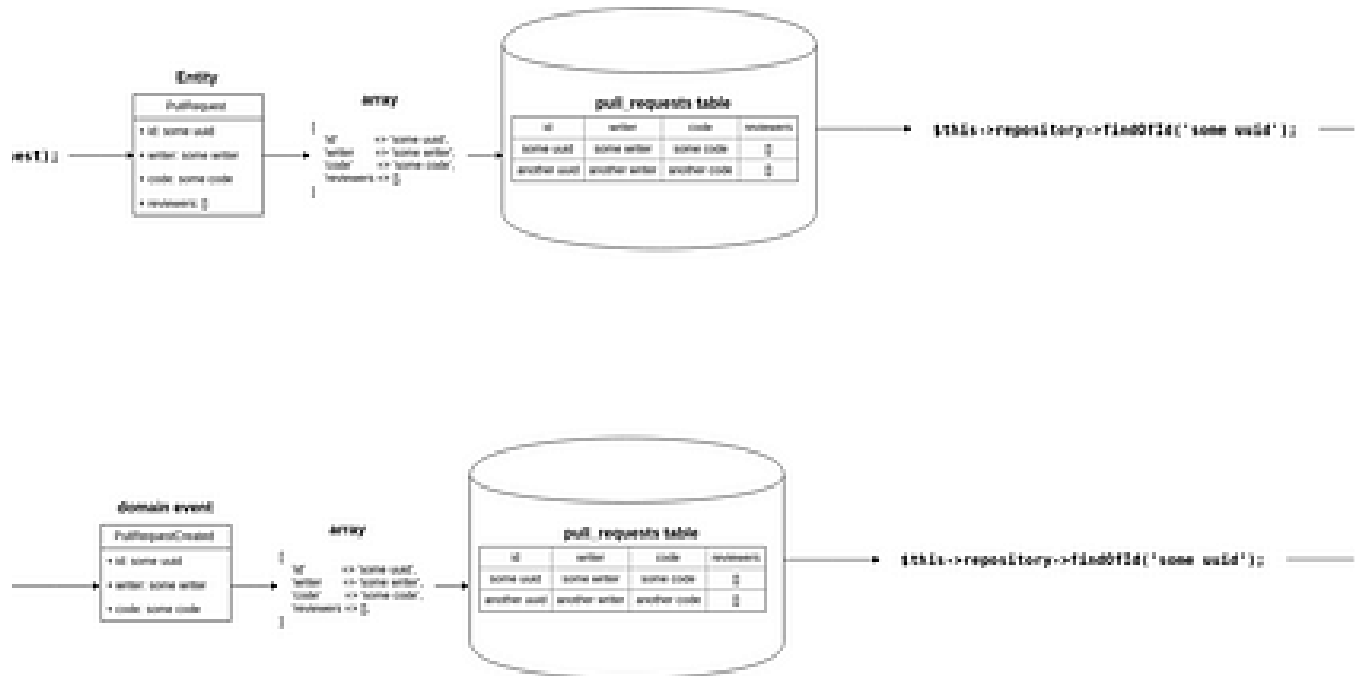
 Maikel González Baile


Implementing a use case (I) - Intro

Use cases implementation made easy

4 min read · Jul 19, 2018







Maikel González Baile

Implementing a use case (IV)—Domain Events (II)

This post is part of Implementing a Use Case, a series of posts where I share my learnings on designing, architecting and implementing use...

4 min read · Oct 5, 2018

 106





See all from Maikel González Baile

Recommended from Medium



 Oliver Samuel

Top 10 Laravel Packages You Should Know in 2024

Laravel, the sleek and expressive PHP framework, continues to enable developers to create sophisticated online apps quickly and...

4 min read · Dec 18, 2023




241



3



 Laravel Pro Tips

Handling Money in Laravel/PHP: Essential Tips

Managing finances in Laravel projects, like setting product prices or finalizing invoices, is vital. This guide offers best practices and...

6 min read · Dec 27, 2023



80



1



Lists



General Coding Knowledge

20 stories · 740 saves



Coding & Development

11 stories · 355 saves



Tech & Tools

15 stories · 117 saves



Stories to Help You Grow as a Software Developer

19 stories · 678 saves



Martin Tonev



Writing Code like a Senior Developer in Laravel

Laravel has emerged as one of the prominent PHP frameworks for building elegant applications

4 min read · Dec 22, 2023



Top 20 PHP 8.3 Exciting Features You Can't Afford to Miss

BY: TARA PRASAD ROUTRAY



Tara Prasad Routray in Level Up Coding

Top 20 PHP 8.3 Exciting Features You Can't Afford to Miss

Learn about the latest enhancements and improvements in PHP 8.3 that revolutionise web development and programming.

21 min read · Nov 27, 2023





Peter Hrobar

Serverless functions with PHP

More and more providers offer solutions to run your code in a serverless manner.

2 min read · Jul 25, 2023



11



Osvaldo Garcia

Integrating Laravel and RabbitMQ with Docker and PHP 8.1

In this tutorial, we will guide you through the process of setting up a Laravel application integrated with RabbitMQ using Docker and PHP...

3 min read · Jul 21, 2023



6



1



See more recommendations