



# php[architect]

The Magazine For PHP Professionals

## Parallelize Your Code

**Introduction to Diagram-as-Code**

**Hack Your Home With a Pi**

**Teaching Through Code Review**

### ALSO INSIDE

#### **The Workshop:**

Configuring PHP-FPM  
and Apache

#### **Community Corner:**

Ben Ramsey, PHP 8.1  
Release Manager

#### **Education Station:**

Async is a Lie

#### **PHP Puzzles:**

Finding Integer Factors

#### **Security Corner:**

Getting Started with  
CyberSecurity

#### **DDD Avenue:**

When You Know The  
Pattern

#### **finally{}:**

Every Which Way But  
Loose





Learn how a Grumpy Programmer approaches improving his own codebase, including all of the tools used and why.

***The Complementary PHP Testing Tools Cookbook*** is Chris Hartjes' way to try and provide additional tools to PHP programmers who already have experience writing tests but want to improve. He believes that by learning the skills (both technical and core) surrounding testing you will be able to write tests using almost any testing framework and almost any PHP application.

**Available in Print+Digital and Digital Editions.**

**Order Your Copy**  
[\*\*phpa.me/grumpy-cookbook\*\*](http://phpa.me/grumpy-cookbook)

# CONTENTS

FEBRUARY 2022

Volume 21 - Issue 2



php[architect]

## 2 Battle-Tested PHP

Eric Van Johnson

## 3 Introduction to Diagram-as-Code

Gabriel Zerbib

## 13 Teaching Through Code Review

Derek Binkley

## 17 Hack your Home with a Raspberry Pi - Part 2

Ken Marks

## 23 Async is a Lie

Education Station

Chris Tankersley

## 27 Getting Started with Cybersecurity

Security Corner

Eric Mann

## 29 Configuring PHP-FPM & Apache

The Workshop

Joe Ferguson

## 33 Interview with PHP 8.1 Release Manager Ben Ramsey

Community Corner

Eric Van Johnson

## 36 New and Noteworthy

## 37 Finding Integer Factors

PHP Puzzles

Oscar Merida

## 41 When You Know the Pattern

DDD Alley

Edward Barnard

## 46 Every Which Way But Loose finally}}

Beth Tucker Long

Edited with Cupid's Arrow

php[architect] is published twelve times a year by:

PHP Architect, LLC  
9245 Twin Trails Dr #720503  
San Diego, CA 92129, USA

### Subscriptions

Print, digital, and corporate subscriptions are available. Visit <https://www.phparch.com/magazine> to subscribe or email [contact@phparch.com](mailto:contact@phparch.com) for more information.

### Advertising

To learn about advertising and receive the full prospectus, contact us at [ads@phparch.com](mailto:ads@phparch.com) today!

### Contact Information:

General mailbox: [contact@phparch.com](mailto:contact@phparch.com)  
Editorial: [editors@phparch.com](mailto:editors@phparch.com)

Print ISSN 1709-7169  
Digital ISSN 2375-3544

Copyright © 2022—PHP Architect, LLC  
All Rights Reserved

Although all possible care has been placed in assuring the accuracy of the contents of this magazine, including all associated source code, listings and figures, the publisher assumes no responsibilities with regards of use of the information contained herein or in all associated material.

php[architect], php[a], the php[architect] logo, PHP Architect, LLC and the PHP Architect, LLC logo are trademarks of PHP Architect, LLC.

# Battle-Tested PHP

Eric Van Johnson

There are a lot of things you can say about PHP and trust me a lot has been said about PHP, but the one thing you can not say is that it hasn't proven itself. PHP has grown with the internet itself and has the battle scars to show for it.

I've dabbled in many languages, and I have a "better than beginners" knowledge of a few of them, such as Ruby, Python, and Go. But I always come back to PHP, considering PHP my primary language for almost a quarter of a century. In that time, a new hot language would surface every few years, the next "PHP Killer." Some of these languages are from companies that build closed source proprietary languages, and others are strong open source languages. I've seen them come into the development world roaring like a lion only to quietly fall in line once they've carved enough of a community out to stay relevant or, in some cases, disappear into the ether (RIP Flash). All this time PHP and PHP developers continued to code, build the internet, adapting, and evolving its core codebase.

Sure it wasn't always smooth sailing in the PHP world, and there were some years when it was hard to hold our heads up high as PHP developers. But the volunteers who make up the core developers never stopped working on making the codebase stronger, adding the features being asked of a new language, and keeping PHP a leader when developing for the Internet.

This month's release touches on some examples that keep PHP and its community strong, relevant, and a fun language to code.

Ken Marks continues his series on using PHP and a Raspberry Pi in a

real-world example with *Raspberry Pi Part 2 - Installing the LAMP Stack on your Pi*. As developers, we live a life where we are constantly learning, and Derek Binkley helps with this by contributing an article called *Teaching Through Code Review*. We have a bonus third feature article this month in which Gabriel Zerbib introduces us to a documentation concept with his contribution *Introduction to Diagram-as-Code*.

In Eric Mann's Security Corner, he talks about how to expand your knowledge with *Getting Started with Cybersecurity*. Joe Ferguson takes time out of his busy schedule to show us some benefits to using PHP-FPM, such as running multiple versions of PHP in his The Workshop section article *Configuring PHP-FPM and Apache*. In Community Corner, I sit down and get to know our second rookie release manager in my *Interview with PHP 8.1 Release Manager Ben Ramsey*. Edward Barnard continues his new DDD Alley series with this month's installment of *When You Know the Pattern*. Oscar Merida helps us exercise our learning muscle with this month's PHP Puzzles, *Finding Integer Factors*. In this month's Education Station, Chris Tankersly gives his take on the elusive async development with PHP in his article *Async is a Lie*. Wrapping up this month's release is Beth Tucker Long finally {}, *Everything Which Way But Loose*.

## Write For Us

If you would like to contribute, contact us, and one of our editors will be happy to help you hone your idea and turn it into a beautiful article for our magazine.

Visit <https://phpa.me/write> or contact our editorial team at [write@phparch.com](mailto:write@phparch.com) and get started!

## Stay in Touch

Don't miss out on conference, book, and special announcements. Make sure you're connected with us.

- Subscribe to our list: <http://phpa.me/sub-to-updates>
- Twitter: [@phparch](https://twitter.com/phparch)
- Facebook: <http://facebook.com/phparch>

## Download the Code Archive:

[http://phpa.me/February2022\\_code](http://phpa.me/February2022_code)





# Introduction to Diagram-as-Code

Gabriel Zerbib

The documentation of source code and APIs have now moved to the source files side. There are tools that make it possible to extract the documentary blocks, to always keep the manual and the code in phase without duplicating the effort. However, a good drawing is better than a long speech, and it becomes more necessary than ever to practice Diagram as Code (DaC) to incorporate technical illustrations directly in the sources.

Today it is very easy to create a diagram explaining the overall dynamics of a system: general tools such as Excalidraw<sup>1</sup> provide for more than the imagination desires if one wants to draw the big picture of an algorithm or the interactions of a program. But then arises the question of where to keep the drawing: should we store an image in the project's Wiki? Should we add to Git the JSON file of the diagram? How do we handle the modifications made to the drawing, as neither of these options will be diff-friendly?

At the end of the 90s, there were applications<sup>2</sup> that we called “software engineering workshops,” which analyzed the source code of a project to produce UML diagrams of all kinds. They would even do the other way round by letting the designer draw visual diagrams without programming, which would then be used to automatically generate source code for their class hierarchies. Some developers may have liked this ideology, but these workshops aren't popular anymore (at least in the fields of activity where I had the chance to find myself over the last two decades). However, there is no need to give up documenting the code with drawings. But, in the era of Git repositories, of README files ever closer to the code, and of embedded comment blocks for the generation of documentation, the time has come to switch to Diagram as Code.

## 1. Principle

Given a collection of standardized diagram types (class hierarchies, sequence, entity-relations, Gantt, pie charts, etc.), the idea is to describe what must be presented graphically by textual and readable directives. We will let an engine render the best possible plot in an abstracted way (layout, formalism) to accommodate the diagram's constraints.

We can think to compare this to the Markdown approach, which produces HTML format from a collection of human-readable symbols inside the source text.

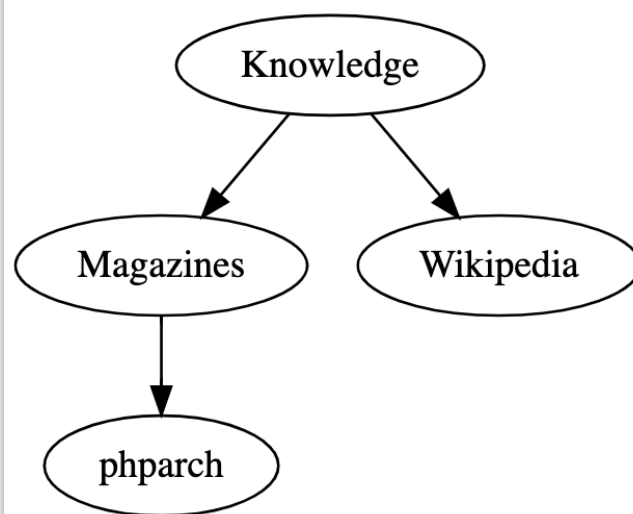
This idea is not new at all! The wonderful Graphviz<sup>3</sup> tool has been serving, for over 30 years, the purpose of plotting a graph based on some textual primitives to describe its essence: a list of states and transitions.

### 1.1 Graphviz

For example, this simple snippet is enough to make the dot executable of the Graphviz suite render the image in Figure 1.

```
digraph {
  Knowledge -> Magazines
  Knowledge -> Wikipedia
  Magazines -> phparch
}
```

Figure 1. A simple directed graph with Graphviz



One can experiment with Graphviz on the online playground<sup>4</sup>. With a local install of Graphviz, the picture is obtained by running a command line similar to:

```
$ dot -Tsvg -o fig1.svg diagram1.txt
```

As we can see, the code does not stipulate anything concerning the shapes of the nodes, the alignment, the colors, etc. With this approach, the emphasis is not on the art or the positioning of the elements. Instead, the description is symbolic, and the layout is deterministic, calculated according to the models pre-wired in the visualization engine.

<sup>1</sup> Excalidraw: <https://excalidraw.com>

<sup>2</sup> applications: <https://whatistechtarget.com/definition/Rational-Rose>

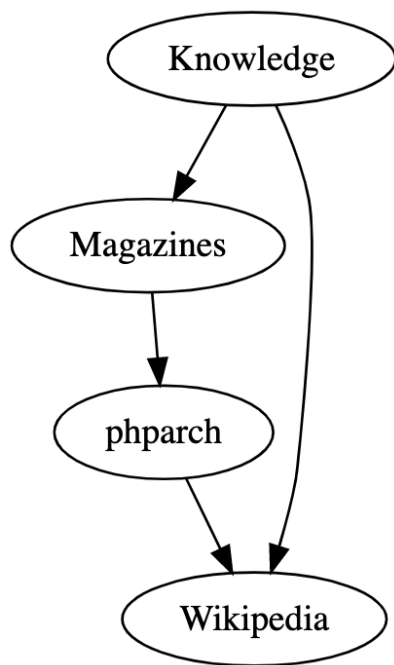
<sup>3</sup> Graphviz: <https://graphviz.org>

<sup>4</sup> online playground: <https://dreampuf.github.io/GraphvizOnline>

As proof, if we simply add a new link between two of the nodes, the engine then produces the image in Figure 2, which it thus calculates on its own to save space and minimize the crossing of arrows.

```
digraph {
  Knowledge -> Magazines
  Knowledge -> Wikipedia
  Magazines -> phparch
  phparch -> Wikipedia
}
```

Figure 2. Graphviz lays out the elements automatically



## 1.2 Customization

It is still possible to specify certain attributes of the elements, such as the color, the thickness of the lines, or the weight of the links (to tighten or lengthen the relative spacing). If these choices are important in the documentary role targeted with the diagram, they have their place in its source code. For the following example, which orients the graph from left to right, we obtain Figure 3:

```
digraph LR
  subgraph Magazines [
    shape="cylinder"
    style="filled"
    fillcolor="#ffa3fc"
  ]
  subgraph phparch [
    shape="hexagon"
    style="filled"
    fillcolor="#aeffa3"
  ]
  Magazines -->|color="#f3cc56" penwidth="6"| phparch
}
```

It is important is that, unlike other drawing approaches, even the assisted ones in terms of anchoring and symmetry, here we do not arbitrarily choose the x, y coordinates of an element or the curve of an arrow (although the syntax allows it to some extent). Instead, we rely on the engine's optimal calculation of the diagram, given our abstract declaration.

## 1.3 Uml

To document an algorithm, a family of classes, a database structure, or the relationships between various components of a distributed system, one should not count on creativity or artistic fiber. A conventional symbolism called UML (Unified Modeling Language<sup>5</sup> defines a set of diagram shapes to convey accurately the concepts involved in software engineering.

In other words, it is a series of standardized drawings (similar to what the road signs are to a driver) which make it possible to express the operation of a program or the structure of a system, using a visual codification shared by all readers.

There are several types of drawings, depending on the topic to be documented: sequence diagrams that highlight the chronological interactions between different elements of the system; entity-relationship diagrams to represent a data structure; class diagrams to articulate hierarchies of objects; etc.

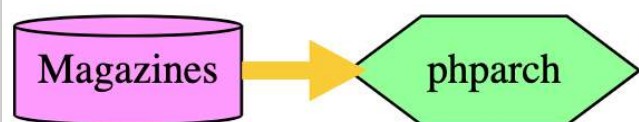
For example, in the UML visual codification, a class should always be represented as a particular rectangular shape; an asynchronous function call should be an arrow with specific stroke properties; and so on. Of course, we don't want to be required to draw all these detailed parts ourselves, with a too general language such as Graphviz. Instead, we will use dedicated tools which specialize in UML and the documenting of software systems.

## 1.4 Tooling

Thus, Diagram as Code is the era of documentary sketching as a first-class citizen, right in the codebase. It is a way of describing drawings in a DSL (Domain Specific Language) with declarative textual instructions (and sometimes even procedural!), easily incorporated in comments or a README.

An interpreter is needed to transform these instructions into an image. Each interpreter supports one or more types of diagrams in their own DSLs. These engines come in various natures: command-line binaries, web services, or even JavaScript modules that run in the browser. The variety of the available tooling allows for the visualization of Diagrams as Code in many situations, from a project's Wiki to an IDE plugin, to a PDF manual.

Figure 3. Attributes in Graphviz



<sup>5</sup> Unified Modeling Language:

[https://en.wikipedia.org/wiki/Unified\\_Modeling\\_Language](https://en.wikipedia.org/wiki/Unified_Modeling_Language)

## 2. Engines

There are a fair number of projects dedicated to the task. We will focus on two of them, PlantUML and MermaidJS, of different paradigms, and then we will present Kroki, a federating facility.

### 2.1 PlantUML

This great project by Arnaud Roques<sup>6</sup> provides an abstraction for writing UML diagrams with a high-level DSL. PlantUML<sup>7</sup> encapsulates all of the complexity of plotting the specific elements that make up the UML standard. It also represents other types of diagrams that are not part of this standard. Under the hood, for certain types of plots, PlantUML uses Graphviz.

#### 2.1.1 Usage

The engine can be downloaded in Java for use in CLI on a file, or on a whole directory of sources, like this:

```
$ java -jar **plantuml.jar** -o "docs/" -**pdf** "src/**/*.php"
```

It will detect all the diagram definitions contained in your source code's comments (or ReadMe files, etc.) and produce a PDF (or an image) for each.

But you can also try it online and get familiar with the syntax in the project's playground (accessible via the Online Server button on the official Website). The image rendered in the playground can be shared by its URL since it contains the source definition of the diagram encoded in Base 64 (it is also possible to revert the URL into a diagram snippet in clear).

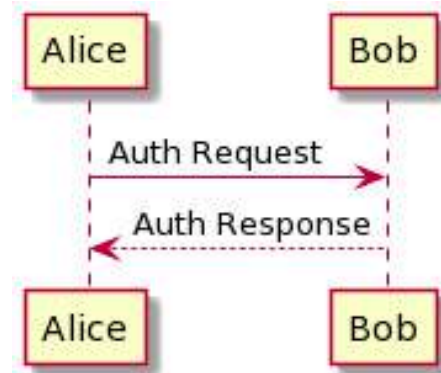
#### 2.1.2 Syntax

Let's have a look.

##### Listing 1.

```
1. @startuml
2. !theme cerulean
3.
4. autonumber
5.
6. participant Alice
7. box
8.   participant Bob
9.   participant Charlie
10. end box
11.
12. Alice -> Bob ++ : says hello
13. Alice -> Bob : asks the time it is
14. Alice <- Bob : replies hello
15. Bob -> Charlie ++ : asks the time
16. ...
17. Charlie -> Charlie : looks at wrist watch
18. Bob <- Charlie -- : replies the time it is
19. Alice <- Bob -- : gives the time
20.
21. @enduml
```

Figure 4. A sequence diagram with PlantUML

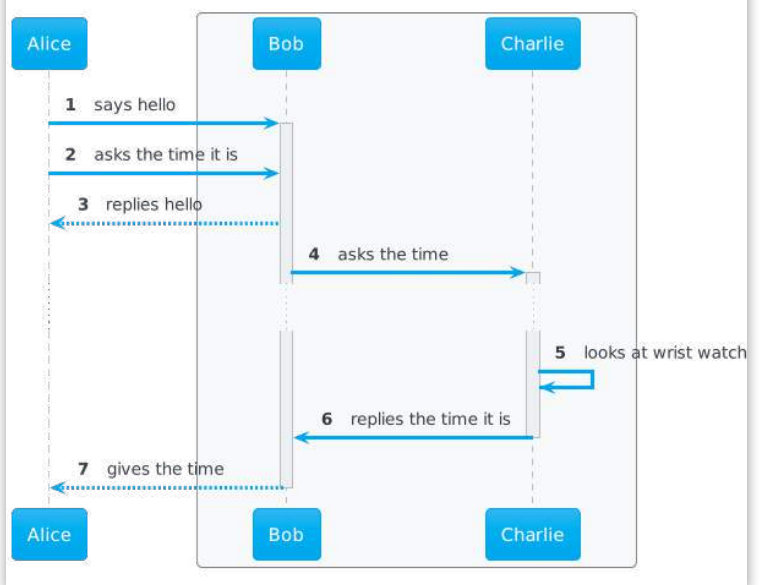


Stripped down as it may seem, the following script draws via PlantUML the sequence diagram with a completely regulatory look in Figure 4. The engine takes care of drawing the participants' boxes, timelines, and generally, everything that characterizes the UML model for a sequence diagram. We only focus on the actual content.

```
@startuml
Alice -> Bob: Auth Request
Bob --> Alice: Auth Response
@enduml
```

The capacities are vast. In Listing 1, we add some semantic elements inherent to this type of diagram and take the opportunity to change the theme (among about twenty predefined) because the default one is not to my liking.

Figure 5. An alternate theme in PlantUML



The rendering yields Figure 5. We can see that via a syntactical abstraction, the engine takes care of the UML elements such as the rectangle surrounding the responsibility group, the symbolism of a long duration on the sequence lines, the

<sup>6</sup> project by Arnaud Roques: <https://github.com/plantuml/>

<sup>7</sup> PlantUML: <https://www.plantuml.com>

numbering of the calls, and so on, without the need for us to bother plotting their visual details explicitly.

This other snippet in Listing 2, creates an Activity diagram (Figure 6): very useful to explain an algorithm or a workflow. And let's give it a handwritten touch, because, why not!

#### Listing 2.

```

1. @startuml
2. !theme cerulean
3. skinparam handwritten true
4.
5. start
6. - Open php[arch] issue
7.
8. if (Already seen?) then (yes)
9.   - Read again
10.  - Try the code examples
11. else (no)
12.   - Read now
13. endif
14.
15. - Put back to drawer
16. stop
17.
18. @enduml

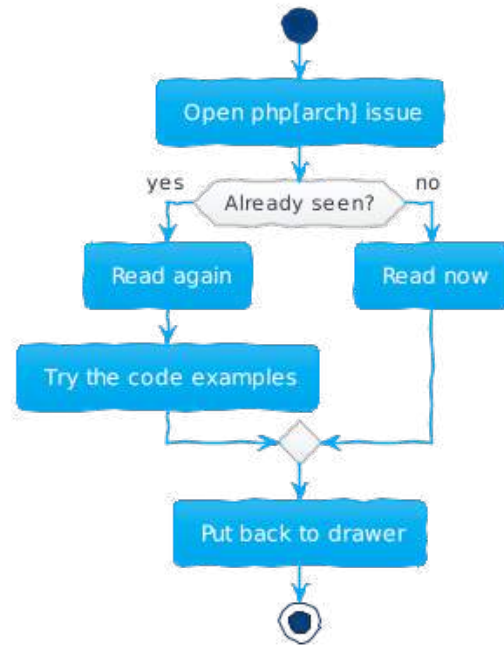
```

We can notice that the DSL infers the type of diagram from the various elements that it contains. Here, the terms **start**, **if**, and **stop**, are typical of an Activity diagram.

#### 2.1.3 Extensibility

PlantUML supports a wide range of UML models and some additional schema types, which are listed on the project's site. The themes can be completely customized by means of special `skinparam` instructions to control globally or individually the shape, stroke, and color of every element.

Figure 6. Activity Diagram in PlantUML



Finally, it is worth mentioning that the language accepts some procedural directives (variables to reuse strings and colors more easily; iterative loops). You can split a diagram into several files and use an `include` directive, and you may even create a macro for code reuse to some extent.

The `include` could bring macro definitions from an external URL (such as a GitHub raw file). There is an entire ecosystem of themes and symbols to create diagrams that comply with the visual standards of AWS, Azure, or other famous vendors.



### You're the Team Lead—Now What?

Whether you're a seasoned lead developer or have just been "promoted" to the role, this collection can help you nurture an expert programming team within your organization.

After reading this book, you'll understand what processes work for managing the tasks needed to turn a new feature or bug into deployable code.

## Order Your Copy

<http://phpa.me/devlead-book>



Listing 3 draws a picture which you can see in Figure 7. Incidentally, we can highlight that the UML grammar can be easily used to draw more general “box system” schemas, even if the drawing doesn’t strictly follow the UML standard.

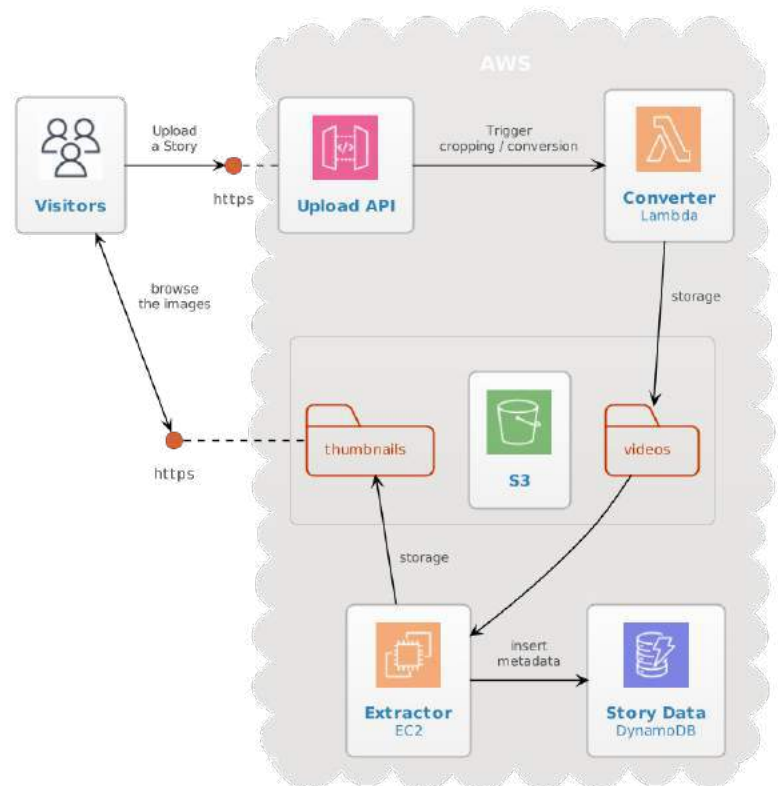
### Listing 3.

```

1. @startuml
2.
3. ' Import the libraries
4.
5. !define AWSPuml https://raw.githubusercontent.com/aws-labs/aws-icons-for-plantuml/v11.1/dist
6. !includeurl AWSPuml/AWSSimplified.puml
7. !includeurl AWSPuml/General/Users.puml
8. !includeurl AWSPuml/ApplicationIntegration/APIGateway.puml
9. !includeurl AWSPuml/Compute/Lambda.puml
10. !includeurl AWSPuml/Database/DynamoDB.puml
11. !includeurl AWSPuml/Compute/EC2.puml
12. !includeurl AWSPuml/Storage/SimpleStorageService
13.
14. ' Theme and styles
15.
16. !theme metal
17. skinparam defaultTextAlignment center
18. skinparam arrowThickness 1.5
19. skinparam arrowColor black
20.
21. ' Construction of the objects
22.
23. Users(visitors, "Visitors", "")
24.
25. cloud AWS {
26.   APIGateway(api, "Upload API", "")
27.   Lambda(lambda, "Converter\nLambda", "")
28.   DynamoDB(storyDb, "Story Data\nDynamoDB", "")
29.   EC2(ec2, "Extractor\nEC2", "")
30.
31.   rectangle {
32.     SimpleStorageService(s3, "S3", "")
33.     folder thumbnails
34.     folder videos
35.   }
36. }
37.
38. interface "https" as apiEndpoint
39. interface "https" as cdnEndpoint
40.
41. ' Bind interfaces to objects
42.
43. apiEndpoint . api
44. cdnEndpoint . thumbnails
45.
46. ' Create the arrows
47.
48. visitors -> apiEndpoint : "Upload\na Story"
49. apiEndpoint -> lambda : "Trigger\ncropping / conversion"
50. lambda --> videos : storage
51. videos --> ec2 : "insert\nmetadata"
52. ec2 --> storyDb : "insert\nmetadata"
53. ec2 --> thumbnails : storage
54. visitors <-> cdnEndpoint : "browse\nthe images"
55.
56. @enduml

```

Figure 7. Component Diagram using external libraries for custom symbols



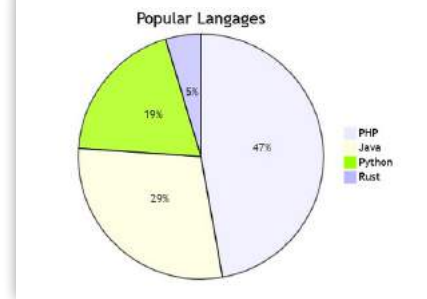
Listing 4.

```

1. <html>
2. <body>
3.   <div class="mermaid">
4.
5.     pie
6.     title Popular Languages
7.     "PHP" : 142
8.     "Java" : 87
9.     "Python" : 58
10.    "Rust" : 14
11.
12.   </div>
13.
14.   <script src="https://cdn.jsdelivr.net/npm/mermaid/dist/mermaid.min.js"></script>
15.   <script>mermaid.initialize({startOnLoad:true});</script>
16. </body>
17. </html>

```

Figure 8. Pie chart with MermaidJS



MermaidJS has a full JavaScript API to manipulate a diagram's data on the fly and to install callbacks for mouse interactions. But above all, MermaidJS is bundled natively with GitLab, whose Web platform knows to render the

## 2.2 MermaidJS

Unlike PlantUML, the MermaidJS project<sup>8</sup> is written in JavaScript and can run in a Web browser.

It emphasizes the presentation of diagrams on the client-side, with immediate rendering in SVG. Notably, there are encapsulations for jQuery, React, and other front-end frameworks. Plus, it offers a certain degree of interactivity to plug events into item clicks and hovers. It supports some UML models (sequence, activity, classes, entity-relations) and some non-UML ones. Mermaid's DSL is different from PlantUML, but it is very clear and easy to use. Everything is customizable using CSS style sheets.

Let's see how to present a pie chart on an HTML page. By default, the library detects at initialization-time all the snippets written in divs with the class mermaid (configurable) and transforms them in place into their visual drawing. Opening the HTML file (Listing 4) in a browser displays Figure 8.

You can experiment with Mermaid's DSL and its various diagram types on the project's playground<sup>9</sup>.

Listing 5.

```

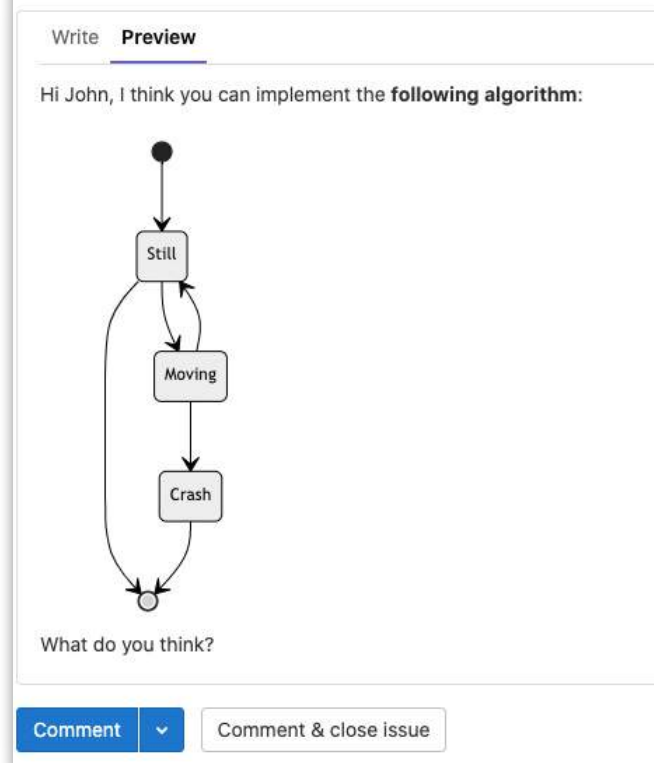
1.
2. Hi John, I think you can implement the **following algorithm**:
3. mermaid
4. stateDiagram-v2
5.   [*] --> Still
6.   Still --> [*]
7.   Still --> Moving
8.   Moving --> Still
9.   Moving --> Crash
10.  Crash --> [*]
11.
12. What do you think?

```

diagrams described in any Markdown fenced code block. In particular, you can embed a mermaid drawing in your Readme files, a Wiki page, or even an issue description or Merge Request!

Figure 9 shows how the Listing 5 example, written in an issue comment, will typically be displayed. This way, there is no reason to upload a binary image if you need to share a piece of technical documentation which could be described with a UML schema.

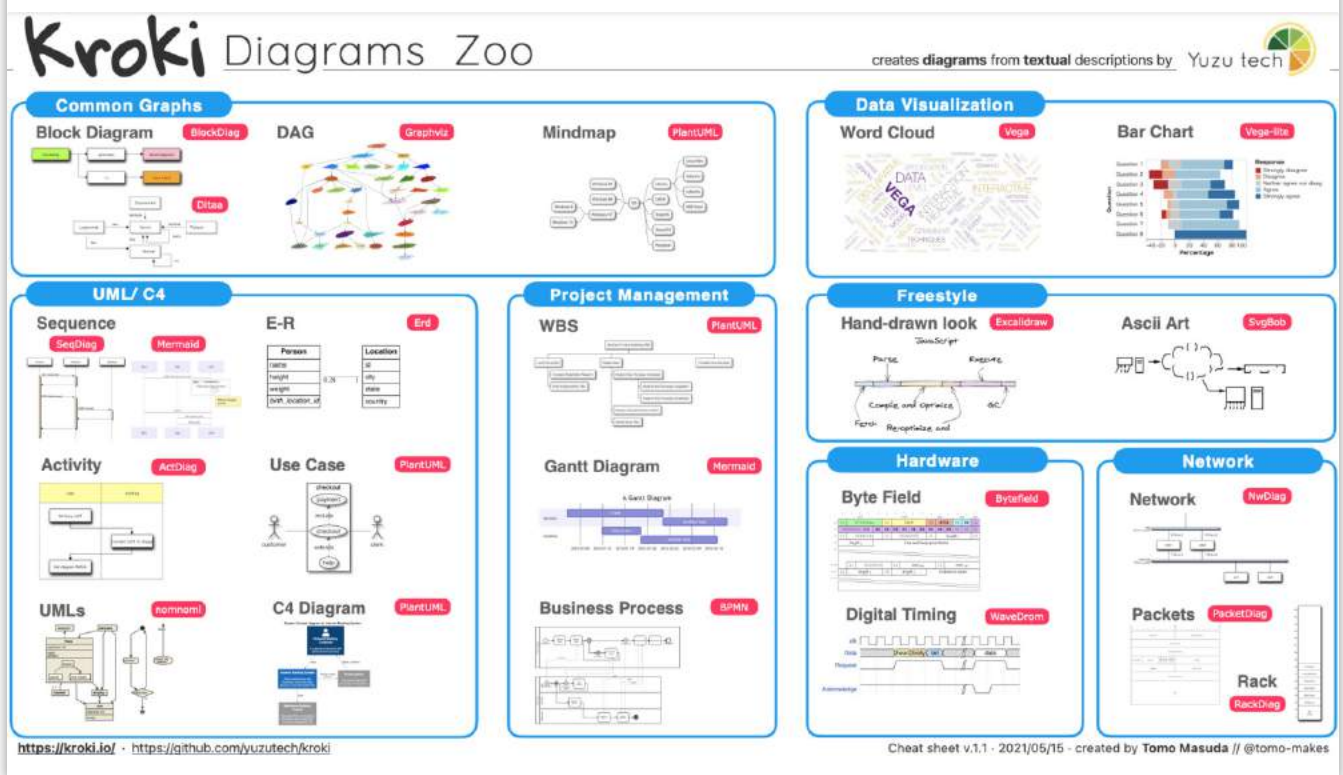
Figure 9. MermaidJS in GitLab Markdown block



<sup>8</sup> MermaidJS project: <https://mermaid-js.github.io>

<sup>9</sup> playground: <https://mermaid.live>

Figure 10. The engines supported by Kroki



## 2.3 Kroki.io

As mentioned earlier, there are a lot of projects and engines out there, which can represent many kinds of drawings — not limited to UML. To name just a few more:

- Bytefield<sup>10</sup>, for expressing byte field diagrams,
- Nomnoml<sup>11</sup>, for more UML possibilities with sleek plotting,
- WaveDrom<sup>12</sup>, for digital timing charts,
- ERD<sup>13</sup>, for elegant relational database schemas,
- ...

All these tools run on their own stack, and it would be tedious for anyone to install and maintain them separately, which is the job of Kroki.io<sup>14</sup>, an online API (also working on-premise): an HTTP request with a given snippet, responds with an image. Kroki supports more than 20 engines (including PlantUML and MermaidJS) and offers a unified invocation system. See Figure 10.

## 3. Integration

PlantUML can crawl a tree of sources and mass-produce the images of the diagrams encountered. The MermaidJS

script does not know to do this on its own since it works in direct rendering in a browser (but of course, as a JavaScript program, it is possible to run it in Node server, which is what is done in Kroki).

Yet, to use DaCs in source files, you must be equipped to write them comfortably during the development phase. Besides, to benefit fully from the possibility to write diagrams as code in the comments of our PHP files to make them self-documented, we need a way to render the pictures in a Doc Generator such as PhpDocumentor.

Let's see how to implement all this.

### 3.1 Ides

In the main development environments, plugins are available for some of the engines.

For example, with VS Code, one can install the Markdown Preview Mermaid Support<sup>15</sup> extension, to see an immediately rendered picture of any MermaidJS graph described inside a mermaid fenced block.

Regarding PlantUML, we have two options: a local binary (requires plantuml.jar and Graphviz dependencies to be installed) or rendering via a web server. The official server is the one that also acts as a playground; it is possible to install a local server very easily in a Docker container with the official plantuml/plantuml-server image. Of course, the easiest remains the remote online service.

<sup>10</sup> Bytefield: <https://bytefield-svg.deepsymmetry.org>

<sup>11</sup> Nomnoml: <https://nomnoml.com>

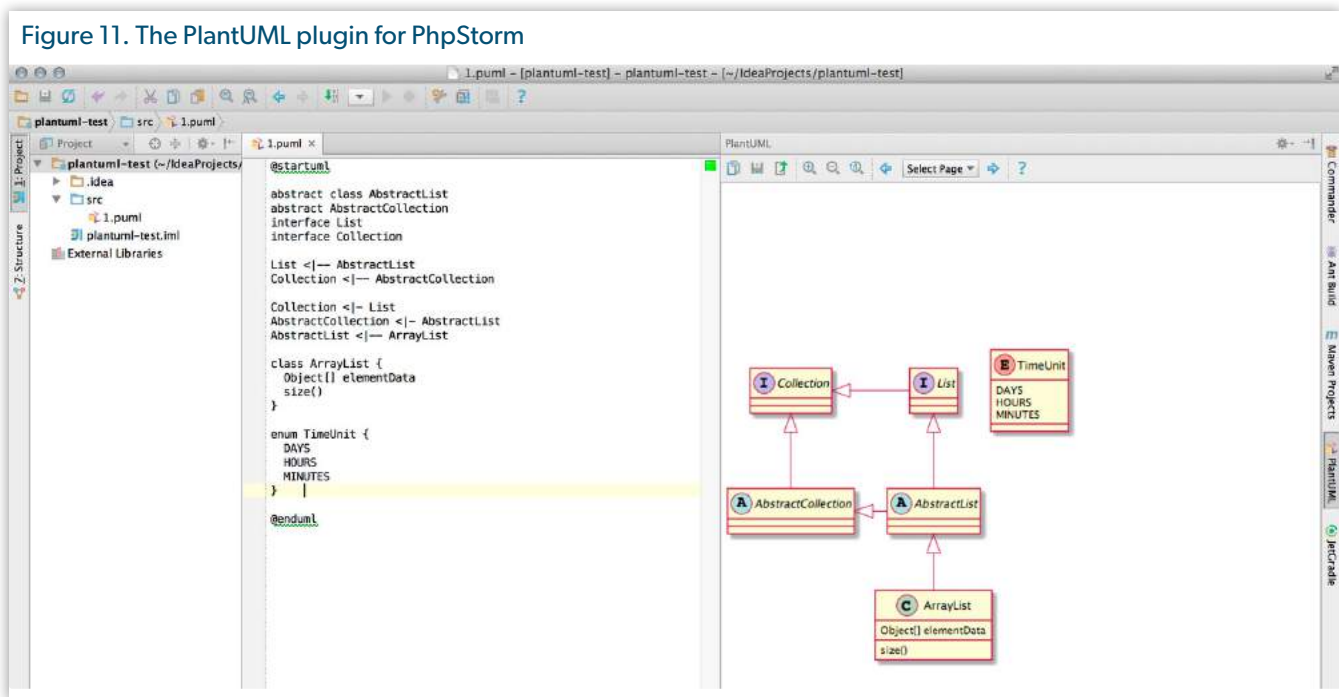
<sup>12</sup> WaveDrom: <https://wavedrom.com>

<sup>13</sup> ERD: <https://github.com/BurntSushi/erd>

<sup>14</sup> Kroki.io: <https://kroki.io>

<sup>15</sup> Markdown Preview Mermaid Support: <https://phpa.me/visualstudio-mermaid>





Either way, the PlantUML VS Code Extension<sup>16</sup> or PlantUML Plugin for PhpStorm<sup>17</sup> are very easy to deploy (See Figure 11).

### 3.2 Phpdocumentor

There is no need for a lengthy presentation of PhpDocumentor<sup>18</sup>, one of the most popular tools for generating browsable documentation for PHP projects via PhpDoc blocks on classes and functions.

We will discuss a way of rendering diagrams in our documentation by writing diagram code inside Doc Blocks. Since the output of PhpDocumentor is a collection of HTML files, we can leverage the online APIs with the help of with some JavaScript, to produce the rendering in the browser when viewing the pages.

#### 3.2.1 Foreword

There are several possible ways to use PhpDocumentor in a project. I am going to use the Docker approach for portability and simplicity.

Given a PHP project in a \$HOME/project folder, we can execute the tool with the following command:

```
docker container run --rm \
-v $HOME/project:/data \
phpdoc/phpdoc:3 \
run -d . -t docs
```

<sup>16</sup> PlantUML VS Code Extension:

<https://phpa.me/visualstudio-plantuml>

<sup>17</sup> PlantUML Plugin for PhpStorm:

<https://plugins.jetbrains.com/plugin/7017-plantuml-integration>

<sup>18</sup> PhpDocumentor: <https://www.phpdoc.org>

This command will invoke a disposable Docker container based on the official image of PhpDocumentor 3 to crawl the source directory for PHP files and create (or update) a docs/ output folder with fresh HTML documents.

#### 3.2.2 Custom Template

The default template that ships with PhpDocumentor cannot transform our Diagram-as-Code into a rendered image. But we can easily create a workaround by making it load a custom JavaScript file of ours, which will detect the diagrams and invoke Kroki for the display.

In the virtual file system inside the Docker image, the template that PhpDocumentor uses by default is located in: /opt/phpdoc/data/templates/default.

As we can see in the GitHub repository of the tool, this template uses a base structure<sup>19</sup> in layout.html.twig, in which several JavaScript files are loaded. We can add the loading of our file here, and, for the sake of comfort and simplicity, I will use jQuery in my Transformer code:

```
{% block javascripts %}
<!-- We can load our program here -->
<script
  src=
  "https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.0/jquery.min.js"
  >
</script>
<script src="js/kroki-transformer.js"></script>
<!-- The default scripts start here -->
<script src="..."></script>
<script src="..."></script>
{% endblock %}
```

<sup>19</sup> base structure: <https://phpa.me/github-documentor-layout>

Now, before we write the actual code for the transformer, let's see how the Doc Blocks end up in the HTML files that PhpDocumentor generates.

### 3.2.3 Diagram Markers

Let's assume that the source folder contains a file with the Doc Block on a class definition as shown in Listing 6:

Out of this source file, the PhpDoc renderer will produce an HTML document in which a tag contains our fenced code block. This tag has a specific class name, reflecting the language name specified at the fence level.

```
<section class="phpdocumentor-description">
  <p>This is a longer description paragraph, ...</p>

  <pre class="prettyprint">
    <code class="language-plantuml">
      Alice -> Bob: first message
    </code>
  </pre>
</section>
```

### 3.2.4 Fetch

All we need to do is, when the page has loaded, to capture every block of this kind, extract the specified language name, and invoke the Kroki API to obtain an image in response, which we can paste in replacement of the original code block.

Essentially, this will be the code of our `kroki-transformer.js`. See Listing 7.

### 3.2.5 Pulling the Pieces Together

Whether you are using a local installation of PhpDocumentor, or provide your own template, you will know how to make the relevant additions and modifications in the template's files to reflect the solution that we've built.

We need to take some extra steps to use the Docker container and the default template shipped with the official image. The `default/template.xml` file describes which files must be rendered into the output directory, so we need to add our JavaScript file to the list by adding an entry to the `<transformations>` tag:

```
<transformations>
  ...
  <transformation
    writer="twig"
    source="templates/default/kroki-transformer.js.twig"
    artifact="js/kroki-transformer.js"
  />
</transformations>
```

#### Listing 6.

```
1. <?php
2.
3. /**
4.  * This is a title for PhpDoc
5.  *
6.  * This is a longer description paragraph,
7.  * which can contain _Markdown notation_ including
8.  * arbitrary **Fenced Code Blocks**, with the Language
9.  * Name of our choice as follows:
10. *
11. *
12. * ````plantuml
13. * skinparam handwritten true
14. * "php[arch]" -> Reader: explain stuff
15. * Reader --> "php[arch]": thanks!
16. * ````
17. */
18. class DiscountVoucher {
19.     // ...
20. }
```

#### Listing 7.

```
1. $() => {
2.   $("code").each((i, elt) => {
3.     const languageClass = $(elt).attr("class")
4.       .split(/\s+/)
5.       .find(cls => cls.match(/^language-/));
6.     if (!languageClass) {
7.       return;
8.     }
9.
10.    const language = languageClass.match(/^language-(.+)$/)[1];
11.    const graphDefinition = $(elt).text().trim();
12.
13.    $.ajax({
14.      type: "POST",
15.      url: `https://kroki.io/${language}/svg`,
16.      data: graphDefinition,
17.      processData: false,
18.      contentType: "text/plain",
19.      success: (svgDocument) => {
20.        const svg = svgDocument.documentElement.outerHTML;
21.        const $pre = $(elt).parent();
22.        $(svg).insertBefore($pre);
23.        $pre.remove();
24.      },
25.    });
26.  });
27. });
```

## Listing 8.

```

1. cat <<EOF | docker run -i --rm \
2. -v $HOME/kroki-transformer.js:/opt/phpdoc/data/templates/default/kroki-transformer.js.twig:ro \
3. -v $HOME/project/:/data \
4. --entrypoint=sh phpdoc/phpdoc:3
5.
6. sed -i "/<\transformations>/i <transformation writer=\"twig\" \
7. source=\"templates/default/kroki-transformer.js.twig\" \
8. artifact=\"js/kroki-transformer.js\" />\" \
9. /opt/phpdoc/data/templates/default/template.xml
10. sed -i "/{% block javascripts %}/a <script \
11. src=\"https://cdn.jsdelivr.net/ajax/libs/jquery/3.6.0/jquery.min.js\"> \
12. </script> <script src=\"js/kroki-transformer.js\"></script>\" \
13. /opt/phpdoc/data/templates/default/layout.html.twig
14. /opt/phpdoc/bin/phpdoc run -d . -t docs
15. EOF

```

This is how you can launch the process. We will assume that our new JavaScript file (Listing 8) is in `$HOME/kroki-transformer.js`.

Let's break it down:

1. Instead of just executing the default command of the container's image, we need to execute a few lines of Shell script beforehand. On line 1, we pipe over to the docker command a small script that we prepare on the spot. The `-i` flag for docker means that it should pay attention to its STDIN stream. But since the shell script spans several lines, we use `cat` and the "heredoc" notation `<<`, to specify that we want to pipe everything up to the EOF symbol on line 11.
2. The command that we activate inside the container on line 4 is simply the default shell.
3. On line 6, we begin our script per se. It has only three commands: two `sed` transforms, and finally, the execution of the `phpdoc` program itself.
4. The `sed` commands on lines 6-7 and 8-9, work "in-place" with the `-i` flag; we modify the files on which we run the search. We can do this because the container is disposable (`--rm`), so we don't care about changing the contents of its filesystem.
5. On line 10, we execute the `phpdoc` command, similar to what we did in paragraph 3.2.1, except there, we simply used the embedded default entry point of the container.

Next, you can point a browser to `file:///<your-project's-path>/docs/index.html` which is the Table Of Contents, then click on your `DiscountVoucher` class, and see the Diagram As Code appear right there in the documentation, as in Figure 12!

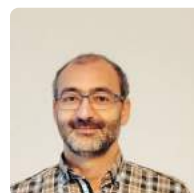
Figure 12. PhpDocumentor page with Kroki invocation



## Conclusion

We have seen how to create technical diagrams using a dedicated language instead of drawing the shapes manually, and we showed that there are many tools available in this domain. We also covered how to use these diagrams in a compatible IDE or Git online platform, and finally, we studied an approach to embedding these diagrams in our generated documentation.

I hope to have given you an entry point to start using the Diagram As Code in your works!



*Gabriel has been using PHP since 2001 to develop Websites and APIs. As an Enterprise Architect, he has had the opportunity to work with several other languages and stacks in Backend, Frontend and DevOps. When he is not at the keyboard, he likes to spend some time mentoring younger developers in the making of their own projects or startups.*  
@zzgab



# Teaching Through Code Review

Derek Binkley

In March 2020, our world shut down. Our connections with colleagues were severed, and the ways we connect with others were changed and will possibly never return to our pre-pandemic norms. We have been overworked, stressed, lonely and afraid. In this environment, we need more than ever to find ways to build each other up and succeed as a team. That is why I propose taking our code review process and changing it from a point of conflict into an opportunity for learning.

Code review is often conducted from a gatekeeper point of view. The reviewer protects their main code branch from buggy or unnecessary features. Code writers are expected to submit code 100% ready for merge, and anything less is unacceptable. While this is an effective method of ensuring bug-free code, it does not help you build a team of developers. It doesn't allow developers to learn from each other and can contribute to developers being siloed away from each other. If we allow our developers to learn from each other, we can build a cohesive and effective team and take full advantage of less experienced developers.

Let's look at Google's engineering practices guide.

*Code review can have an important function of teaching developers something new about a language, a framework, or general software design principles. It's always fine to leave comments that help a developer learn something new. Sharing knowledge is part of improving the code health of a system over time. Just keep in mind that if your comment is purely educational, but not critical to meeting the standards described in this document, prefix it with "Nit: " or otherwise indicate that it's not mandatory for the author to resolve it in this CL.*

This single paragraph<sup>1</sup> gives no advice on how to mentor another developer. In fact, it sounds to me like you are most likely to get an unwanted lecture rather than a two-way learning experience.

## Benefits

Sharing knowledge throughout our team can improve our productivity over time. We gain more developers that can work on any given area of code and share techniques in ways that are more effective than writing documentation or reading blog posts<sup>2</sup>. In today's job market, experienced developers are in high demand, yet getting into the industry can be challenging<sup>3</sup>. Cultivating young talent will allow your team to grow and flourish in the long term. It's not just good for your team but good for our industry as a whole. The tech industry

needs to find ways to build up this large pool of potential developers into a great talent pool.

To make room in our process for teaching and learning, let's start by removing some barriers that add time to our day without a lot of benefits.

## Consistency

A few years ago, I joined a new team. Passing code review was extremely important, and our code was expected to be of the highest quality. Unfortunately, coding styles varied greatly between projects, and code would need to be written in different ways based upon who was expected to be the reviewer. Do yourself and your teammates a favor and decide on what type of consistency in your coding is important, and then document. This way, if you do see inconsistent code, you can merely refer the developer to your team's documentation.

For even more time savings, automate as much as you can. Having human reviewers looking for missing semicolons is not a good use of time. These are tasks where computers beat humans hands down. It may be a challenge to agree at first, but once your team comes up with a standard linting or prettier file, you can move on to solving bigger problems.

You could also take your automation up a level by adding checks to your git process. For example, you can use the tool Husky<sup>4</sup> along with php-cs-fixer<sup>5</sup> to add a hook that runs before each commit. First, set up php-cs-fixer by running:

```
$ mkdir --parents tools/php-cs-fixer
$ composer require --working-dir=tools/php-cs-fixer friendsofphp/php-cs-fixer
```

Next, set up husky by running the appropriate command depending upon which node package manager you use.

```
$ npx husky-init && npm install # npm
$ npx husky-init && yarn # Yarn 1
$ yarn dlx husky-init --yarn2 && yarn # Yarn 2
```

Finally, add this line to the newly created .husky/pre-commit file.

```
tools/php-cs-fixer/vendor/bin/php-cs-fixer fix src
```

1 single paragraph: <https://phpa.me/github-review-standard>

2 reading blog posts: <https://phpa.me/atlassian-code-review>

3 challenging: <https://phpa.me/builtin-dev-team>

4 Husky: <https://typicode.github.io/husky/#/>

5 php-cs-fixer: <https://github.com/FriendsOfPhp/PHP-CS-Fixer>

Now, we get consistently formatted code on every commit to our repository.

## Make Time for Teaching

Generally, we don't account for the time it takes to review code in our task estimates. However, it is important to acknowledge that reviews take time to do well. Even without the learning and teaching elements, reviewing code takes time. You may get some pushback initially, especially if you are billing time to a client, but as your process begins showing results, your coworkers, bosses, and clients will start to trust your estimates. Over time, as this process makes developers more productive and bugs less common, hopefully, you'll find that you've merely shifted where you spend your time instead of adding to your burden.

Eliminating these obstacles can lead to better reviews, but what about actually teaching others through review? What does that mean, and how does it help?

A code reviewer's job is to give feedback. To make this a learning experience, the reviewer needs to be thoughtful about the presentation of their feedback. The easiest and fastest way to submit your review is to say the code is wrong and changes are required. Harvard business review performed a study about the effectiveness of negative feedback<sup>6</sup>. In this study, they found that negative feedback wasn't effective at improving performance and led to people avoiding situations where they would receive such feedback. Negative feedback makes people defensive and not in a state where they are ready to learn.

Let's take a look at a small PHP code snippet (Listing 1) and think about what we would put in a review.

An experienced PHP developer may be irritated by the mistakes made here, but it is important to remember our purpose. We want the author to learn so that they will write higher quality code in the future. We can do that by guiding them to a solution without completely solving<sup>7</sup> it for them. Take a look at this example review.

*The uasort function modifies the original array. The function could be called on a copy of the array to prevent this. Would this work better in this instance? PHP's spaceship operator could replace many lines of comparison code with a single line, e.g. `$a->count <=> $b->count`. Could that be used to make this comparison more readable?*

This review uses questions instead of demands. This phrasing starts a conversation that allows the author to explain their thought process. There may be some context beyond our awareness. Maybe our suggestions were already attempted but didn't work out for some reason. Perhaps there were system requirements we aren't aware of that affected the

Listing 1.

```
1. class CustomerCollectionDTO
2. {
3.     private $data = [];
4.
5.     public function getSortedArray()
6.     {
7.         return uasort($this->data, function($a, $b) {
8.             $comparison = 0;
9.             if ($a->count > $b->count) {
10.                 $comparison = 1;
11.             } else if ($a->count < $b->count) {
12.                 $comparison = -1;
13.             }
14.             return $comparison;
15.         });
16.     }
17. }
```

chosen solution. By learning the author's thought process, we can tailor our advice to help them learn.

Code review doesn't have to happen just when a task is complete. Many code review processes allow for reviewing preliminary code. Some teams do this as needed, and others push their daily work in progress. We can formalize this process by using tags to indicate that a branch is not ready for merging, is ready to review, needs work, or is approved. Another great learning practice for junior developers is to have their plans for development reviewed. By checking in method stubs and having those reviewed, they'll get quick feedback so they can make changes before they have invested a lot of time. These early reviews can also relieve frustration and build confidence while learning their craft.

Look at this example class.

```
Class NonDescriptiveEmployerService
{
    public function transform($emp) {}
    public function get($id) {}
}
```

A side effect of reviewing at this time is that you will be encouraged to provide descriptive names and type hints. If you leave these out, you will need to use comments to communicate to the reviewer what your intentions are. See how different this class looks in Listing 2.

Listing 2.

```
Class EmployerService
{
    public function transformToDTO(EmployerEntity $employer)
        : EmployerDTO {}

    public function findInRepository(int $employerId)
        : EmployerEntity {}
}
```

<sup>6</sup> negative feedback: <https://phpa.me/hbr-feedback>

<sup>7</sup> completely solving: <https://phpa.me/htmlallthethings-mentoring>

## Writing Code to be Read

Using descriptive method names is one way to begin writing code meant to be read by humans. Generally, when we learn how to write code, it is meant to be read by a machine. In fact, there is even a computer science competition<sup>8</sup> where the winner writes the most unreadable code. Once we start writing code that will last longer than a semester course, we need to think about another set of readers, other developers. Even without a review process, at some point, our code will need to be read so that we can verify what task it performs, modify or replace it.

In *What is Readable Code*<sup>9</sup>, Janeen Neri writes that for code to be readable, you'll need to find what you need while skimming through the code. The reader will need to "understand what the code is trying to do" and be "confident that it's doing what it claims." Doing so means having a well-organized and well-labeled set of code and unit tests. Defining how to best organize our code would take an entire article in itself, yet beginning to think of the reader as you write your code will take you a long way towards the goal of readable code.

## Reviewer Fatigue

This approach to reviewing code may seem daunting. We are already working full-time jobs or more. What prevents a code reviewer from merely skimming through and approving with LGTM (looks good to me) if no obvious bugs are found? That is where we need to be cognizant of reviewer fatigue. More frequent, smaller reviews will be easier to fit into another developer's schedule and allow the time to really understand what your change is accomplishing. We can achieve this by breaking up tasks into smaller portions or by, as mentioned earlier, reviewing code many times throughout the development process.

Another way to be respectful of both the reviewer's time and the author's time is to be explicit in what you are asking. If you are requesting a code change, briefly explain or give an example and reason why. If you are requesting a review, make sure your pull request is documented so the reviewer knows what is being changed and why. Guessing someone's intent is not a good use of time.

Knowledge sharing is another important benefit of effective code reviews. The mere act of a code review gets a second or third person involved who must understand the process(es) surrounding the code. They will learn what the code does and how it does it. Suddenly you have one more person who can help out with this project, debugging or adding new features.

## Commenting with Kindness

As stated by Software Engineer, Curtis Einsmann<sup>10</sup>

*I comment on the code, not the person. I avoid using "you" or "your." I prefix nitpicks with "nit." I phrase most comments as questions or suggestions. I leave at least one positive comment.*

In a recent code review of one of my pull requests, the engineering manager added to the end of his review. "Thanks for communicating the issue clearly in a comment." The comment wasn't necessary to the pull request, but it felt nice to read. It also reinforced how this team I had newly joined wanted to structure comments on pull requests.

Most code review advice helps us avoid unintentionally creating conflict by providing different ways to open up a conversation. Are people intentionally creating conflict, or are they letting frustrations bleed through into their comments? If so, why? There can be many reasons. Does it seem like a team member isn't pulling their weight? Does their code need lots of revisions before passing review?

It's way too easy to get frustrated at others. "Oh man, what were they thinking when they submitted this?" I'm sure we've all had thoughts like that about another developer, colleague, or user of our product. There are a lot of potential reasons for these feelings. Development is a creative discipline, and it can be hard to accept criticism of work that we are proud of. Ben Brearly, in the *Thoughtful Leader*<sup>11</sup>, cites communication and process problems as two reasons for employee frustration that may commonly occur in developer roles. There is also the idea that developers need to perform at a high level all the time, which adds pressure on the review process. No matter where this frustration is coming from, it is essential to leave it out of the code review process. We can do this by having empathy for others on our team and remembering that we are all trying to work together for similar goals. We all benefit when we bring them up and help them get to our level.

## Inclusion

Creating a learning environment can open a team up to be more inclusive. Instead of judging each other through the review process, we are collaborating, teaching, and learning. Doing so can make our process more open and safe for developers of all levels and backgrounds. By being thoughtful in our interactions during the review process, we open the team up to input and participation from all members. I strongly believe that any diversity and inclusion initiatives you have will directly benefit from such a process.

<sup>8</sup> computer science competition: <https://www.ioccc.org>

<sup>9</sup> What is Readable Code: <https://phpa.me/medium-readability>

<sup>10</sup> Curtis Einsmann: <https://twitter.com/curtiseinsmann>

<sup>11</sup> Thoughtful Leader: <https://phpa.me/thoughtfulleader-workplace>



## Communication Methods

Gauging a writer's tone when reading their text can be very difficult. Due to time constraints, we often make terse or quick comments when writing reviews. While often fine, sometimes comments like these can leave a user wondering about intentions or lead to misunderstandings. Professor Ray Birdwhistell wrote a set of essays<sup>12</sup> in 1970 where he studied how all five senses are used in human communication. He found how important cues can be missed when not speaking in person. While in-person communication is usually not feasible, we can try other methods like having a call or posting a video or image review. Most importantly, we may need to try out different options with different people to see what works best.

## Who can Participate?

Each team will have its own working styles, which will dictate who performs reviews. Some smaller teams have a leader who conducts all reviews. Some teams divide the responsibilities by content area, while others allow anybody on a team to review as long as a certain number signs off before merging code. No matter whom you choose, I would encourage you to find a way for junior developers to participate. For those who haven't been involved in code reviews in the past, it is important to set the stage, so they understand how to review effectively. Part of a good review is to learn what code is doing and why. This learning can be difficult but will

provide great benefits by exposing junior developers to expert work. Over time hopefully, you'll find that junior developers can be just as good at pointing out areas of improvement, bringing in ideas from other projects, or sharing classroom knowledge.

In addition to including developers of all levels, there is one developer that should always be included: the code author. If you are working alone, go ahead and review your own code. It won't be collaborative, but you can still catch errors by going through your code in a PR and ensuring you know what all the lines of codes you have written are doing. If you are part of a team, still go ahead and review your own code before you open a PR. I'm amazed at the things I notice when reading my own code in the GitHub review screens and outside my IDE.

## The Accidental Code Review

There was a time when a lot of us worked in offices. It was so easy to peek your head around a cubicle wall and ask somebody's opinion on where you are stuck—doing so often led to grabbing a chair and working through code together. In our remote, work from anywhere environments, please take the time to share with others when you can. A screen share works just as well, if not better than huddling around the same computer.

If you are in a position of power at your organization, make sure your employees know that asking for help benefits everybody.

## Conclusion

We've explored a new way of looking at code reviews. While some changes require restructuring of a process many changes can be made easily by individuals. By keeping your mind open to learning and teaching as your approach code review, you can make a significant impact.



*Derek Binkley is a Software Consultant with Spark Labs. While getting his start fixing the Y2K date problem in Cobol, Derek quickly moved on to spend over twenty years using PHP, JVM Languages, JavaScript, MySQL, Elasticsearch, and Oracle. He enjoys teaching others through speaking, mentoring, and writing articles. Derek encourages developers to write clean, testable code. When not in front of a computer he spends time with family, travels, makes pizza, and drinks beer. @DerekB WI*



<sup>12</sup> set of essays: <https://www.upenn.edu/pennpress/book/179.html>

# How to Hack your Home with a Raspberry Pi - Part 2 - Installing the LAMP Stack on your Pi

Ken Marks

Welcome back to another installment of 'How to Hack your Home with a Raspberry Pi.' At the end of this article, you should have a Raspberry Pi running a full LAMP stack that can serve up web pages to any browser on your home network. So grab your Raspberry Pi and a beverage of your choice so you can continue this journey with me as we install some more software!

## Introduction

Last month (in part 1), I covered the following topics of this project:

- The story behind my motivation for this project
- A list of components needed (including the Raspberry Pi)
- How to install the Operating System on the Pi
- Configuring the Pi

So please make sure you start from the beginning if you want to build this project yourself.

In this article, I will cover:

- Updating the software packages on your Pi
- Installing the LAMP Stack (consisting of):
  - Apache
  - MySQL
  - PHP

## Power Up Your Pi and Remotely Log in

Connect the power supply to your Raspberry Pi and plug it into A/C mains.

In part 1, you configured your Raspberry Pi so you can SSH into it remotely from a terminal window on your computer connected to your home WiFi.

## Ssh Into Your Pi

After a few minutes of letting the Pi power up, bring up a terminal window on your computer. Next, SSH into your Pi with the user `pi` by typing the following command into the terminal:

```
ssh pi@raspberrypi.local
```

Enter your password. You should be logged in to your Raspberry Pi if all goes well.

## Update the Packages

Before installing any software packages on our Raspberry Pi, we need to ensure all the existing software (OS, kernel, etc.) is up to date. Since we are using a Debian-based Linux distribution on the Pi, I like to use the apt package manager to keep my system's software up-to-date. Doing so is as simple as running the following two commands in the terminal window:

```
sudo apt update
sudo apt upgrade
```

After running `sudo apt update`, you will see the results and the number of packages that can be upgraded in the terminal window Figure 1. In my case,

that was 44 packages. The first time you run the `apt update` command, you could have over 100 packages that need to be updated.

After running `sudo apt upgrade`, you will be asked:

```
Do you want to continue? [Y/n]
```

Press either Y and the return key or just the return key. Depending on how many packages need upgrading, it could take anywhere between a few seconds to several minutes for the apt package manager to complete. Once the apt command finishes upgrading all the packages, we are ready to install the LAMP stack.

## Install the Lamp Stack

Our installation of the LAMP stack will consist of Linux, Apache, MySQL, and PHP. Since we already have Linux installed and updated, we can continue with the Apache web server, the MySQL database, and the PHP interpreter for our server-side coding language.

## Install Apache

We will install the tried and true Apache webserver to serve up our web

Figure 1. `sudo apt update`

```
pi@raspberrypi:~$ sudo apt update
Hit:1 http://archive.raspberrypi.org/debian bullseye InRelease
Hit:2 http://raspbian.raspberrypi.org/raspbian bullseye InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
44 packages can be upgraded. Run 'apt list --upgradable' to see them.
pi@raspberrypi:~$
```

pages. In the terminal window (assuming you are still logged into your Raspberry Pi), type the following command to install the latest Apache webserver:

```
sudo apt install apache2
```

You will be asked:

```
Do you want to continue? [Y/n]
```

Press either Y and the return key or just the return key. You will see the details of the apache installation process in your terminal window. Once the apt command finishes installing Apache, we can test it. Let's see if Apache is running correctly by running the systemctl command from the command line. Type the following command in the terminal window:

```
sudo systemctl status apache2
```

When running systemctl, I am looking for two things:

- The apache2 service is installed correctly in systemd
- apache2 is active and running

You will see the details of systemctl status of apache2 in your terminal window as shown in Figure 2.

Figure 2. systemctl status apache2 - running

```
pi@raspberrypi:~$ sudo systemctl status apache2
● apache2.service - The Apache HTTP Server
   Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2022-01-20 19:14:39 CST; 53s ago
     Docs: https://httpd.apache.org/docs/2.4/
   Main PID: 4911 (apache2)
    Tasks: 55 (limit: 415)
```

Type `ctrl+c` to exit out of the systemctl status screen.

Now that I know Apache is installed and running, bring up a browser on your computer and navigate to `http://raspberrypi.local/`. See Figure 3.

Figure 3.



### Create a New Home Page

Let's create our own home page to identify our Raspberry Pi! Our web pages will be located in the directory `/var/www/html`. Back in our terminal window (assuming you are still logged into your Pi), type the following two commands in your terminal window to change to the `/var/www/html` directory and to list out the contents. Then you should see the `index.html` file listed in your terminal window.

```
cd /var/www/html
ls -al
```

To make our home page look clean, I like to use the Bootstrap<sup>1</sup> CSS framework. To learn more about the

framework and get more comfortable using it, head to the site and click the Get started button. I'm using version 5.1 of the Bootstrap framework in this article series.

We have a couple of options for how we can create our home page: we can use one of the built-in editors on the Pi (like nano), or we can use our favorite editor on our PC or Mac and remote copy the home page to the Pi. I'm going to use an editor on my Mac and use scp to remote copy this simple `index.html` (Listing 1) page to the Pi.

### Listing 1.

```
1. <!DOCTYPE html>
2. <html>
3.   <head>
4.     <title>Lame Raspberry Pi Home Page</title>
5.     <!-- Latest compiled and minified CSS -->
6.     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
7.           rel="stylesheet"
8.           integrity="sha384-1BmE4kWBq78iYhFdvKubhFTAU6auU8tT94WlRhftJDbrcEXSU1oBoqyl2QvZ6jIW3"
9.           crossorigin="anonymous">
10.   </head>
11.   <body>
12.     <div class='card'>
13.       <div class='card-body'>
14.         <h1>This is my Less, but Still Lame Raspberry Pi Home Page</h1>
15.       </div>
16.       <div class='card-body'>
17.         <div class="alert alert-dark" role="alert">
18.           <h3 class="text-danger">Wow! If you got here, that means Apache is running. Yay!</h3>
19.         </div>
20.       </div>
21.     </div>
22.     <!-- Latest compiled and minified bundled JavaScript -->
23.     <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
24.            integrity="sha384-ka7Sk0Gln4gmtz2MlQnik11wXgys0g+OMhuP+I'L9sENBO0LRn5q+8nbTov4+1p"
25.            crossorigin="anonymous"></script>
26.   </body>
27. </html>
```

<sup>1</sup> Bootstrap:

<https://getbootstrap.com>



In the terminal window where you are logged into your Pi, create a Downloads directory in your home directory.

```
cd
mkdir Downloads
```

On your local computer, bring up another terminal window in the folder where you saved your index.html file. Then type the following scp command into this terminal window to remote copy the index.html file to the Downloads directory on your Pi:

```
scp index.html pi@raspberrypi.local:~/Downloads
```

After entering the password for your Pi login, you should see the results shown in Figure 4 from the scp command in the terminal window.

Figure 4. scp index.html to Pi

```
Kenneths-MacBook-Pro:files_on_server kemarks$
Kenneths-MacBook-Pro:files_on_server kemarks$ scp index.html pi@raspberrypi.local:~/Downloads
pi@raspberrypi.local's password:
index.html                                100% 1007  109.6KB/s  00:00
Kenneths-MacBook-Pro:files_on_server kemarks$
```

You can dismiss this window if you like. Going back to the other terminal window where you are logged into your Pi, you should be able to verify the index.html file has been successfully copied to your Downloads directory on your Pi. See Figure 5.

As superuser, copy this newly uploaded index.html file to

Figure 5. index.html file copy to Downloads directory

```
pi@raspberrypi:~$ ls -al Downloads/
total 12
drwxr-xr-x 2 pi pi 4096 Jan 20 19:27 .
drwxr-xr-x 4 pi pi 4096 Jan 20 19:25 ..
-rw-r--r-- 1 pi pi 1007 Jan 20 19:33 index.html
pi@raspberrypi:~$
```

the /var/www/html/ folder to overwrite the original default apache index.html file:

```
sudo cp ~/Downloads/index.html /var/www/html
```

Refreshing your browser on your computer that is displaying: <http://raspberrypi.local/>, we can see our new home page similar to Figure 6.

## Install MySQL

Now that Apache is running, let's install MySQL—one of the most popular open-source relational databases. We will actually be installing MariaDB<sup>2</sup> (a more efficient and fully open source clone of MySQL).

At the terminal window where you are logged into your Pi, enter the following command to install the latest version of the MariaDB server:

```
sudo apt install mariadb-server
```

2 MariaDB: <https://mariadb.org/about/>

Figure 6. Raspberry Pi home page



You will be asked: Do you want to continue? [Y/n]. Either press Y and the return key or just press the return key. You will see the details of the mariadb-server installation process in your terminal window, along with a progress bar.

## Verify MySQL is Running

The great thing about installing MariaDB is that we can use all the MySQL commands we are used to using.

Once the apt command finishes installing MariaDB, let's see if it's running correctly by running the systemctl command from the command line. Type the following command in the terminal window:

```
sudo systemctl status mysql
```

When running systemctl, I am looking for two things:

- The mariadb service is installed correctly in systemd
- mariadb is active and running

Figure 7 shows the details of systemctl status of mariadb/mysql in your terminal window.

Figure 7. sudo systemctl status mysql - running

```
mariadb.service - MariaDB 10.5.12 database server
Loaded: loaded (/lib/systemd/system/mariadb.service; enabled; vendor preset: enabled)
Active: active (running) since Fri 2022-01-21 10:39:02 CST; 1min 52s ago
    Docs: man:mariadb(8)
          https://mariadb.com/kb/en/library/systemd/
Process: 1213 ExecStartPre=/usr/bin/install -m 755 -o mysql -g root -d /var/lib/mysql
Process: 1215 ExecStartPre=/bin/sh -c systemctl unset-environment _WSREP_START_TIMESTAMP
Process: 1224 ExecStartPre=/bin/sh -c [ ! -e /usr/bin/galera_recovery ] && mv /etc/mysql/conf.d/mariadb.galera_initial_configuration.galera.conf /etc/mysql/conf.d/
Process: 1298 ExecStartPost=/bin/sh -c systemctl unset-environment _WSREP_START_TIMESTAMP
Process: 1300 ExecStartPost=/etc/mysql/debian-start (code=exited, status=0/SUCCESS)
Main PID: 1286 (mariadb)
Status: "Taking your SQL requests now..."
Tasks: 9 (limit: 415)
CPU: 8.231s
CGroup: /system.slice/mariadb.service
└─1286 /usr/sbin/mariadb

Jan 21 10:39:16 raspberrypi /etc/mysql/debian-start[1305]: information_schema
Jan 21 10:39:16 raspberrypi /etc/mysql/debian-start[1305]: mysql
Jan 21 10:39:16 raspberrypi /etc/mysql/debian-start[1305]: performance_schema
Jan 21 10:39:16 raspberrypi /etc/mysql/debian-start[1305]: Phase 6/7: Checking
Jan 21 10:39:16 raspberrypi /etc/mysql/debian-start[1305]: Processing databases
Jan 21 10:39:16 raspberrypi /etc/mysql/debian-start[1305]: information_schema
lines 1-23
```

Type `ctrl+c` to exit out of the systemctl status screen.

Now let's test the MySQL command-line interface (CLI) by logging in as superuser in your terminal window:

```
sudo mysql
```

By typing `SHOW DATABASES;` into the MySQL CLI, you should see the standard three schemas being listed as shown in Figure 8.

Figure 8. `sudo mysql, show databases`

```
pi@raspberrypi:~$ sudo mysql
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 44
Server version: 10.5.12-MariaDB-0+deb11u1 Raspbian 11

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mysql      |
| performance_schema |
+-----+
3 rows in set (0.006 sec)

MariaDB [(none)]>
```

Type `exit` to exit out of the CLI in the terminal window and return you to the shell.

## Install Php

The last packages we need to install are the PHP interpreter and PHP connector for MySQL.

In addition to the PHP interpreter, we need to install the PHP MySQL connector to communicate with our MySQL databases from our PHP scripts.

*The default version of PHP installed with this distribution of Raspberry Pi OS is 7.4. If you wanted to, you could specifically install whatever version you wanted by tacking on the version number (i.e., `sudo apt install php8.1`). However, you would first need to add a third-party repository to the apt package manager list that includes the version of PHP you are interested in installing like, `launchpad`<sup>3</sup> maintained by Ondrej Sury. For this project, the default PHP version of 7.4 will work just fine, and it is the one I tested.*

At the terminal window where you are logged into your Pi, enter the following command to install the latest version of PHP and the PHP connector for MySQL:

```
sudo apt install php php-mysql
```

You will be asked: Do you want to continue? [Y/n]. Press either `Y` and the return key or just the return key. You will see the details of the php installation process in your terminal window, along with a progress bar.

Verify the Php Installation and Allow Errors to be Displayed

Let's verify our PHP installation. In the terminal window, verify the version by typing `php -v`.

Mine is showing it is version 7.4.25

On a normal LAMP deployment, we would be good to go. However, since we are running this on our own home WiFi, and we will want to have the means to debug issues if something goes wrong, let's modify our PHP installation to display errors.

In the terminal window, bring up the `php.ini` file into the nano editor as superuser:

```
sudo nano /etc/php/7.4/apache2/php.ini
```

Search for the second occurrence of `display_errors` using the Where Is command (`ctrl+w`) and set `display_errors = On` as shown in Figure 9.

Figure 9. Enable `display_errors` in `php.ini`

```
GNU nano 5.4 /etc/php/7.4/apache2/php.ini
; For production environments, we recommend logging errors rather than
; sending them to STDOUT.
; Possible Values:
;   Off = Do not display any errors
;   stderr = Display errors to STDERR (affects only CGI/CLI binaries!)
;   On or stdout = Display errors to STDOUT
; Default Value: On
; Development Value: On
; Production Value: Off
; http://php.net/display-errors
display_errors = On

; The display of errors which occur during PHP's startup sequence are handled
; separately from display errors. PHP's default behavior is to suppress those
; errors from clients. Turning the display of startup errors on can be useful in
; debugging configuration problems. We strongly recommend you
; set this to 'off' for production servers.
; Default Value: Off
; Development Value: On
; Production Value: Off
```

Type `^O` to write out the file and `^X` to exit the nano editor.

We need to restart the webserver so that Apache can reread the `php.ini` file. Type the following command into the terminal window:

```
sudo systemctl restart apache2
```

Let's modify our home page to display today's date and time. First we need to go to the `/var/www/html` directory and change the extension of our index page from `.html` to `.php`:

```
cd /var/www/html
sudo mv index.html index.php
```

Next, open the renamed `index.php` file with the nano editor as superuser:

```
sudo nano index.php
```

And add the following Bootstrap card-body with PHP code to output today's date and time, just below the last card-body:

```
<div class='card-body'>
  <div class="alert alert-info" role="alert">
    <? date_default_timezone_set('America/Chicago'); ?>
    <h3>Today is: <?= date('l \t\h\e jS \of F Y') ?></h3>
    <h3>The time is: <?= date('h:i:s A') ?></h3><br/>
  </div>
</div>
```

3 launchpad: <https://launchpad.net/~ondrej/+archive/ubuntu/php/>

## Listing 2.

```

1. <!DOCTYPE html>
2. <html>
3.   <head>
4.     <title>Lame Raspberry Pi Home Page</title>
5.     <!-- Latest compiled and minified CSS -->
6.     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
7.           rel="stylesheet"
8.           integrity="sha384-1BmE4kWBq78iYhFdvKubhFTA6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIw3"
9.           crossorigin="anonymous">
10.   </head>
11.   <body>
12.     <div class='card'>
13.       <div class='card-body'>
14.         <h1>This is my Less, but Still Lame Raspberry Pi Home Page</h1>
15.       </div>
16.       <div class='card-body'>
17.         <div class="alert alert-dark" role="alert">
18.           <h3 class='text-danger'>Wow! If you got here, that means Apache is running. Yay!</h3>
19.         </div>
20.       </div>
21.       <div class='card-body'>
22.         <div class="alert alert-info" role="alert">
23.           <? date_default_timezone_set('America/Chicago'); ?>
24.           <h3>Today is: <?= date('l \t\h\e jS \of F Y') ?></h3>
25.           <h3>The time is: <?= date('h:i:s A') ?></h3><br/>
26.         </div>
27.       </div>
28.     </div>
29.     <!-- Latest compiled and minified bundled JavaScript -->
30.     <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
31.            integrity="sha384-ka7Sk0Gl4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBB00LrN5q+8nbTov4+1p"
32.            crossorigin="anonymous"></script>
33.   </body>
34. </html>

```

Listing 2 shows the complete listing for the index.php script. Type ^O to write out the file and ^X to exit the nano editor.

Refreshing your browser on your computer that is displaying: <http://raspberrypi.local/>, we can see our updated home page display today's date and time. See Figure 10.

We now have a complete LAMP stack installed and tested!

## Install Adminer

There is one more package I like to install for interacting with and managing our databases. Adminer<sup>4</sup> is a web-based database management tool similar to phpMyAdmin. I like Adminer because it is contained in a single PHP script and is very intuitive.

To get the latest version of Adminer, open up your browser and navigate to <https://www.adminer.org/#download> and you will see the different options you have for downloading Adminer as shown in Figure 11.

Since we are using a MySQL-based database, I suggest either the Adminer for MySQL or the Adminer for MySQL English-only versions. I'm going to download the English-only

<sup>4</sup> Adminer: <https://www.adminer.org>

Figure 10.

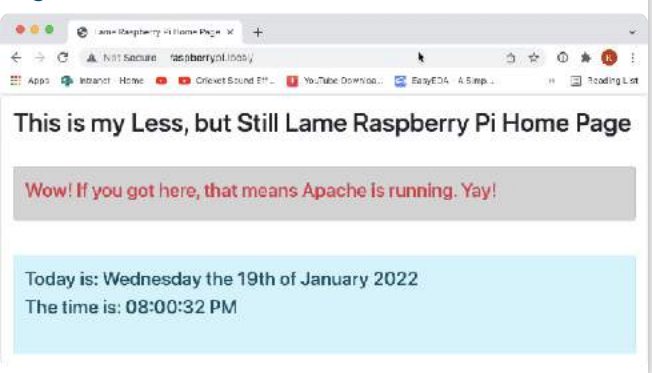


Figure 11.





version. The way I like to download this on my Raspberry Pi is to copy the link address into the terminal window and use `wget` to download it onto my Pi.

First, I'll create a `Downloads` directory in my home directory on my Pi and `cd` into it:

```
cd
mkdir Downloads
cd Downloads
```

Next, I'll use `wget` and paste the copied link right afterward to download Adminer:

```
wget https://github.com/vrana/adminer/releases/download/v4.8.1/adminer-4.8.1-mysql-en.php
```

Now change to the `/var/www/html` directory. As superuser, create a directory called `adminer`, then copy the newly downloaded Adminer script to the `adminer` directory and name it `index.php`:

```
cd /var/www/html
sudo mkdir adminer
cd adminer
sudo cp ~/Downloads/adminer-4.8.1-mysql-en.php index.php
```

In your browser on your computer, bring up another tab and navigate to: `http://raspberrypi.local/adminer`, and you will see the login form for Adminer (Figure 12).

We'll need to create a database user and grant access to it before we can use Adminer, but we'll save that for the next installment of *How to Hack your Home with a Raspberry Pi*.

## Conclusion

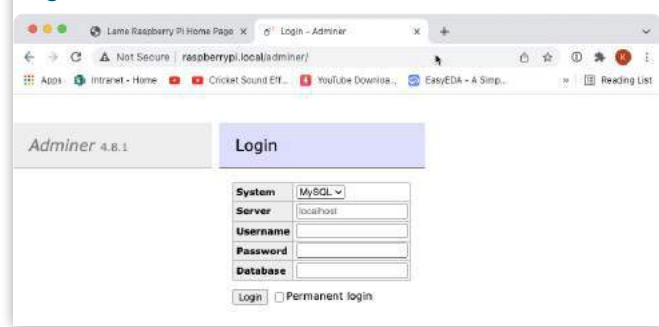
You now have a Raspberry Pi provisioned with a full LAMP stack! This concludes our second installment of *How to Hack your Home with a Raspberry Pi*.

In the next installment, we will:

- connect up an accelerometer sensor
- create a database to store the accelerometer data
- compile a C++ program that reads the accelerometer data from the sensor and logs it to the database
- and create a Unix Service that:
  - automatically starts the data logging when the Pi boots
  - and stops the data logging when we shut our Pi down

Until next time, I hope you have fun playing around with your Pi!

Figure 12.



*Ken Marks has been working in his dream job as a Programming Instructor at Madison College in Madison, Wisconsin, teaching PHP web development using MySQL since 2012. Prior to teaching, Ken worked as a software engineer for more than 20 years, mainly developing medical device software. Ken is actively involved in the PHP community, speaking and teaching at conferences. @FlibertiGiblets*



### Learn how to build dynamic and secure websites.

The book also walks you through building a typical Create-Read-Update-Delete (CRUD) application. Along the way, you'll get solid, practical advice on how to add authentication, handle file uploads, safely store passwords, application security, and more.

Available in Print+Digital and Digital Editions.

**Purchase Your Copy**  
<https://phpa.me/php-development-book>



# Async is a Lie

Chris Tankersley

One of the more popular programming topics over the last few years has been the idea of “async programming.” Async programming is touted as a way to speed up applications by avoiding issues that normally stall a program. When an application hits an operation that prohibits anything else from happening, this is called a “blocking operation.” A blocking operation blocks all other execution until it finishes.

A few of the most common blocking operations in a web application are reading and writing from the disk (disk I/O) and database operations. If a large file is being read or a large or complex database query is executed, the application will generally stall until that operation completes. The program is not misbehaving—it is just simply waiting. We want to return a response for web applications as soon as possible, and blocking operations can make our websites feel slow.

In the previous article, “Background Queues,” I mentioned that PHP is a single-threaded, procedural language.

*PHP is traditionally a single-threaded, procedural language, which means that when a request comes in, the engine starts at the beginning of a script and processes it line-by-line. The engine does not move on to the next line until the current line is finished. For most workloads, like rendering a JSON or HTML response, this is fine.*

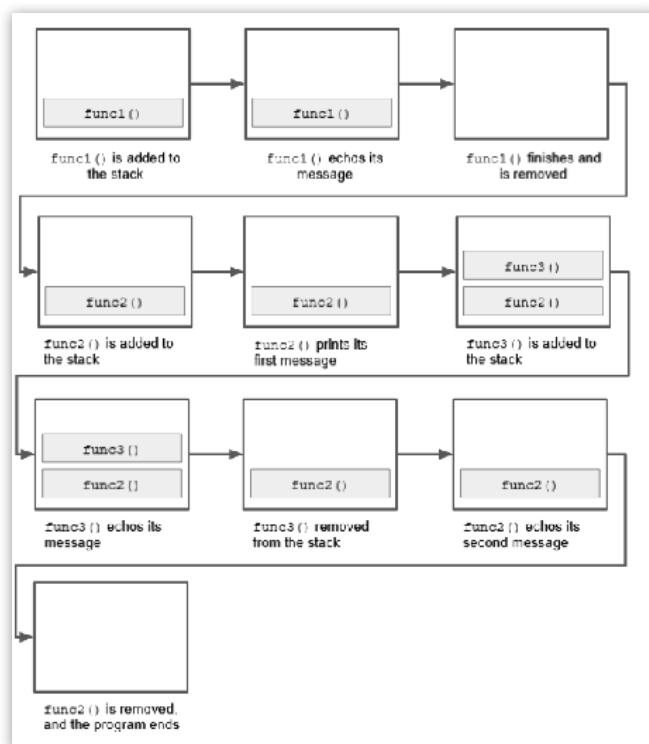
By definition, PHP is very susceptible to blocking operations. Thankfully the language itself is incredibly quick, but disk I/O and third-party connections like databases can still make our web applications feel slow. In fact, PHP itself is ever rarely the bottleneck in applications. Application slowdowns happen during these blocking operations.

## The Callstack

When a PHP script executes, what happens?

In any procedural program, a structure called a call stack is used to keep track of what functions are currently being called. A call stack is just like a Last-In-First-Out queue, where function calls are added to the queue and then executed from newest to oldest. The call stack is parsed until there is nothing left, at which point the program exits.

Let’s look at a super simple PHP script to see what happens.



### Listing 1.

```
1. function func1() {
2.     echo 'Hello World';
3. }
4.
5. function func2() {
6.     echo 'This is an example.';
7.     func3();
8.     echo 'Back in func2()';
9. }
10.
11. function func3() {
12.     echo 'Now we are here';
13. }
14.
15. func1();
16. func2();
```

This basic PHP script has three basic functions, `func1()`, `func2()`, and `func3()`. The PHP parser reads and catalogs the functions but does not immediately execute them. When it hits the first line to execute, which in this case is the line just calling `func1()`, `func1()` is added to the call stack.



All `func1()` does is echo out “Hello World” and return. The script then goes to the next line, which is an invocation of `func2()`. It adds this call to the call stack. The interpreter then jumps to the first line of `func2()`. `func2()` echos out its first line, “This is an example.”

The interpreter then moves to the next line, which is the invocation of `func3()`. `func3()` gets added to the call stack. At this point, we have `func3()` at the top of the stack and `func2()` just below it. Since a call stack is Last-In-First-Out, `func3()` is called before `func2()` finishes.

`func3()` echos out its line of “Now we are here.” `func3()` is now finished, so it returns and is removed from the call stack. The call stack then picks back up in `func2()` and echos out the line of “Back in `func2()`.” `func2()` then returns and is removed from the call stack.

Then the script exits since the call stack is empty and there are no more lines to parse.

The simplest way to think about this is that as each function is called, it is added to the call stack and immediately executed. As new functions are called, they are added to the stack in the order they are called. None of the functions already on the stack finish execution until the newer functions finish, and they always finish in the order they are on the stack. `func2()` will never finish before `func3()` finishes.

In straightforward terms, this is why blocking operations are so bad. If we stick a `sleep(10)` call in `func3()`, the entire call stack will stop until `sleep()` returns after ten seconds. Calling something like `file_get_contents()` will not allow anything else to run until the entire file is read into memory. If the disk is slow or the file is exceptionally large, this can completely stall your program until the file is finished reading.

Each thread in a program has its own call stack. Since PHP is single-threaded, there is only ever one call stack. PHP handles a single request at a time until it is finished. If that single call stack ever stalls, the entire application stalls. Async programming attempts to alleviate this issue.

## Wait, How Does Php Handle Multiple Requests?

If PHP is single-threaded, and PHP can only handle one request at a time... how do high traffic websites handle potentially thousands of requests a second? PHP is one of the most performant web application languages in use.

It's simple; PHP doesn't. It makes someone else do it.

If you are running `mod_php` under Apache, Apache spawns multiple threads. For example, if you want Apache to handle 20 concurrent requests, you configure it to spawn 20 threads. Each thread can handle a single request, and since `mod_php` basically embeds a PHP engine in each request, those requests can be parsed by a PHP engine in the thread.

If you are using FastCGI under Apache or Nginx (or some other Fast-CGI compatible server), you use a separate process manager like `php-fpm`. This process is a separate program that manages multiple PHP threads. Each request from the web server talks to `php-fpm`, and `php-fpm` passes the request to an available thread. Like with Apache, you configure `php-fpm` to

spawn a number of threads, and this number is how many requests you can handle at once.

In either case, the actual PHP engine itself is not multi-threading. An external process just spawns multiple PHP engines.

## Async in Php

If PHP is single-threaded with a single call stack, and PHP does not handle threading, how does async programming in PHP work? Well, it does not work in PHP.

Async in PHP is a lie.

There are two different types of async programming. One is traditional asynchronous programming, where various threads of code execute at the same time, but you cannot guarantee which thread of code will finish first. This programming is commonly referred to as “multi-threaded programming.”

With multi-threaded programming, the main program called the “parent” starts up with its own call stack. It then generates other threads, called “children.” These child threads each have their own call stack. The parent program watches the children and waits for them to finish their work.

In some cases, they may share resources like memory, but essentially each thread runs on its own until their individual call stacks finish. The CPU then handles which thread has priority for work, and the parent program just waits for the children to finish. This method is how Apache and `php-fpm` handle processing PHP requests.

Many will argue that this is not really “async,” as async programming deals with tasks and not workers. I would argue that more workers mean tasks can be split amongst multiple workers, and you still have the same advantages that async purports to have.

The second type of async programming is the more recent kind which is more akin to cooperative multitasking, popularized by `node.js` and javascript programming. In this paradigm, blocks of code are put onto a call stack and priority shifts between these blocks of code and various task queues. They do not execute at the same time; however, they just efficiently cede control of execution between themselves. This paradigm is considered async because there is still no guarantee as to the order of execution.

## Event Loops

The easiest way to fake async in PHP is through the use of what is called an Event Loop. An Event Loop is a mechanism that simply loops through blocks of code to check to see if they need to run (and executes them), are currently running, are waiting for some response, or need to return something to the main code. These blocks of code the loop is inspecting are generally very small and efficient.

Async in `node.js` and the browser is done via Javascript's native Event Loop. In a browser, Javascript is effectively run with two threads. One thread runs the procedural code of a javascript script. The second thread is for the WebAPIs, or the browser itself. A process known as an event loop helps move





things onto the main call stack from the WebAPI thread and queues.

In node.js, the node interpreter and the event loop itself are a single thread. Through a combination of various libraries being multi-threaded and the interpreter itself having built-in ways to handle context switching the overall system is asynchronous. The ability to handle these libraries and the context switching is baked into the interpreter itself, and the engine knows how and when to parallelize the operations.

The main thread, or the code you write, puts things into the event loop. This code, like a call using the `fetch()` API, gets added to the event loop's call stack. The event loop moves the needed code to the WebAPI for it to handle. When the WebAPI is finished, it adds calls to a task queue. The event loop constantly loops, waiting for things to get added to the task queue and adds them or their callbacks back to the main call stack.

Frameworks like ReactPHP emulate an Event Loop architecture which uses things like generators and other systems to quickly switch between small blocks of code. Each block of code can be, and are, blocking. ReactPHP just requires you to structure your code in a way that minimizes blocking. All it takes is a single SQL query to block everything, as most PHP database drivers are not "async."

The reason for this is PHP is single-threaded. The engine can only ever execute one command at a time. There is no second thread like a browser to do other work. We can emulate an event loop, but without a separate thread or better context switching, our event loop does not natively work exactly the same way.

You can build a simple and crude event loop using nothing but an array and some callable code as shown in Listing 2.

Our `EventLoop` class is very simple. It has an internal array to keep track of things, which is our "stack." We can add callable objects to that stack, like the three anonymous functions that echo out some text. If any of that code returns another callable, we add it to the stack. The event loop then just loops through everything (preferably continually, but our event loop stops when the stack is empty).

Async-aware code is structured in a way to do small amounts of work, generally one operation, and if there is more work to be done, return another callable block of code. For things like HTTP calls or SQL queries that can take time, the libraries are built to more intelligently re-add themselves to the stack while they wait for a response.

Properly written async code for ReactPHP can run incredibly fast, but much of the code in the PHP ecosystem is not async aware. For example, you cannot use Doctrine currently in ReactPHP efficiently. The code executes just fine, but technically the code will block on every query and will not be any faster than a procedural script.

Generally, this means that async PHP is not suitable for mixing long-running worker tasks and also serving general HTTP requests. It is very easy to write blocking code in PHP because the language is not very async aware.

## Listing 2.

```

1. class EventLoop {
2.     public array $stack = [];
3.
4.     public function run(): void {
5.         while (count($this->stack)) {
6.             $invokable = array_shift($this->stack);
7.             if ($invokable) {
8.                 $response = $invokable();
9.                 if (!is_null($response) && is_callable($response)) {
10.                     $this->stack[] = $response;
11.                 }
12.             }
13.         };
14.     }
15. }
16.
17. $loop = new EventLoop();
18. $loop->stack[] = function() {
19.     echo "I was added to the stack first" . PHP_EOL;
20. };
21. $loop->stack[] = function() {
22.     echo "I was added to the stack second" . PHP_EOL;
23.     return function() {
24.         echo "I was added from inside a function" . PHP_EOL;
25.     };
26. };
27. $loop->stack[] = function() {
28.     echo "I was added to the stack third" . PHP_EOL;
29. };
30. $loop->run();
31.
32. // Output
33. I was added to the stack first
34. I was added to the stack second
35. I was added to the stack third
36. I was added from inside a function

```

## Coroutines

An alternative to event loop programming is coroutines. In PHP, this can be handled using generators or fibers. Generators are blocks of code that yield back multiple responses and can effectively pause their execution while yielding. You see them used when large amounts of data are being looped over and manipulated. Listing 3 shows an example of a generator.

Generators on large amounts of data that need looping can drastically reduce memory usage. Instead of `myGenerator()`

## Listing 3.

```

1. function myGenerator(): Generator {
2.     for($i = 1; $i <= 5; $i++) {
3.         yield $i;
4.     }
5. }
6.
7. foreach (myGenerator() as $number) {
8.     echo $number . PHP_EOL;
9. }

```

## Listing 3 output.

```

1
2
3
4
5

```



returning an array of five elements, it returns each of the elements one-by-one. It does this by pausing execution on the `yield` keyword. We yield the value of `$i` back to the `foreach` loop each time the `foreach` loop loops.

Generators have a lesser-used feature in that you can pass info back into the generator. Doing so turns the generator into what is known as a coroutine. The coroutine allows itself to do some work, pause (and possibly return an intermediary value), and then later pick up the rest of the execution.

#### Listing 4.

```
3. function myCoroutine(): Generator {
4.     echo "I did some work and will now wait." . PHP_EOL;
5.     $value = yield;
6.     echo "Thanks, I will print " . $value . PHP_EOL;
7. }
8.
9. $cor = myCoroutine();
10. $cor->current(); // Execute up to the yield
11. echo "Now we wait" . PHP_EOL;
12. sleep(1);
13. $cor->send(1); // Finish execution
```

#### Listing 4 output.

```
I did some work and will now wait.
Now we wait
Thanks, I will print 1
```

Our coroutine is created, then the call to `$cor->current()` causes it to execute up to the first `yield`. This time we do not yield back a value, but by placing it on the right-hand side of the assignment operator (`=`), we can pass a value back using `$cor->send(1)`. The coroutine then finishes execution with access to the passed-back value.

Nikita Popov has an excellent write-up<sup>1</sup> on using generators to provide cooperative multitasking. Generators and coroutines do not provide true async programming but get very close as PHP is still single-threaded and only does one operation at a time.

As a note, fibers are a bit cleaner way to implement coroutines in PHP but do not add much in the way of actual async programming. The core engine of PHP is still single-threaded.

## Open Swoole

Open Swoole<sup>2</sup> takes a completely different approach to traditional PHP async by using multiple threads. Open Swoole provides ways for threads to share memory space and more easily pass information around than traditional multi-threading in PHP by handling it through an extension, making it a bit more of a challenge to run everywhere. It is not just a userland abstraction on top of the threading extension that has existed for a while.

Using Open Swoole means rewriting your code to follow their coroutine and threading setup. Open Swoole also ships with its own HTTP and database clients, which are aware of how Open Swoole handles threading. While switching to ReactPHP also tends to require some refactoring, Open Swoole requires using its library of classes to function—meaning its code is a bit more incompatible with non-Open Swoole frameworks PHP.

That does not necessarily mean you need to abandon frameworks; Laravel Octane<sup>3</sup> and Mezzio-Swoole<sup>4</sup> wrap Open Swoole to make it easier to use at the framework level. If you are interested in working with Open Swoole, both projects are a good foot in the door.

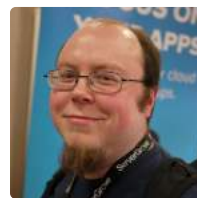
## Is Async Right for You?

It is a bit of a hot take, but async programming offers no major benefits for most general web applications. PHP is extremely quick as a language and well suited to let others handle the multi-threading. We live and program in a world where servers can routinely have hundreds of threads at a time, and CPUs are very effective at handling the context switching themselves. It is cheap to scale horizontally.

Other languages like node do not operate like PHP, and therefore async is much more suited for them. Node applications are generally single-threaded, and the web servers interact with them differently, so the language and libraries are designed with that in mind. Some languages like C# use interrupts to know when work is being finished and needs to return.

Under some workloads, async programming can be much quicker than traditional procedural programming when the libraries and language can effectively switch between tasks. It is good to understand the blockers in your applications and how to break things up, and async programming can be a very effective way to optimize some code.

As with everything technical, look at your problem before deciding on a solution. Async programming has its place, but it is not a silver bullet. Take a look at libraries like ReactPHP and Swoole and see if they solve any issues you are having. If they do, great! If they do not, traditional procedural PHP still works and serves billions of requests every day.



*Chris Tankersley is a husband, father, author, speaker, podcast host, and PHP developer. Chris has worked with many different frameworks and languages throughout his twelve years of programming but spends most of his day working in PHP and Python. He is the author of Docker for Developers and works with companies and developers for integrating containers into their workflows. @dragonmantank*

1 write-up: <https://phpa.me/npopov-multitasking>

2 Open Swoole: <https://www.swoole.co.uk>

3 Laravel Octane: <https://laravel.com/docs/8.x/octane>

4 Mezzio-Swoole: <https://docs.mezzio.dev/mezzio-swoole/>

# Getting Started with Cybersecurity

Eric Mann

Every career track starts somewhere. Cybersecurity doesn't always begin where you'd expect.

Understanding how to start a career in software development can be a challenge. Some of the best engineers complete a bachelor's degree in computer science before ever starting a job as a dev. Others finish their master's degree before getting a job. Yet others start in engineering with no formal education.

Each of these paths is equally valid. Where the path takes you, though, is your choice.

Several years ago, a colleague asked how I'd gotten started myself. Not just in engineering, but particularly in cybersecurity. Their question was less about my own path and more about seeking advice for how they could make the jump from analytics to security.

I gave them the same advice I give anyone else.

## Learn to break things

One avenue to get started in security is through tinkering. Take something apart and put it back together. This could be a Lego set or a blender. It doesn't need to be complex or even important. The goal is to learn how things are put together and find a way to un-make those things.

At first, this doesn't feel directly related to cybersecurity. It's more closely tied to thinking like an engineer.

But thinking like an engineer is the first prerequisite to starting with cybersecurity. You have to learn to figure out how things you didn't build work. The most straightforward way to do so is to take them apart and try to put them together again.

## Aim to misbehave

Once you know how things work, take time to learn how to make a system misbehave.

Submitting instructions to a program to force it to behave in unexpected ways is the most critical part for those wanting to practice cybersecurity. In fact, causing a program to misbehave is the easiest way to identify a flaw that an attacker could exploit.

Code fuzzing<sup>1</sup> is a technique in automated software testing that forces random, often invalid data into an application's inputs. The goal is to identify ways the application can crash or behave unexpectedly. These unexpected behaviors are usually the first point of ingress for a would-be attacker.

In November<sup>2</sup>, I shared an experience from my own coding history where I'd ignored a bug discovered by a researcher submitting junk data to one of my applications. That was a major mistake on my part, as the bug turned out to have a very critical impact. But it was a massive win on the part of the researcher. He'd identified a flaw in my code (though neither of us immediately knew how to exploit it) merely by tricking my program into misbehaving.

## Pick locks

Learning how a tumbler lock work is easy, and I don't use the word "easy" very often. There are hundreds of free instructional videos online, and you can buy transparent tumbler locks from merchants like Amazon. Learning how to bypass a lock is a bit harder, but the resources are the same.

Aside from having a fun skill to show off at parties, learning to bypass a lock with a set of picks concretes both of the previous two skills. You've learned how something works internally (by either taking it apart or using a transparent version), and you've figured out a way to operate the system that the designers did not intend.

Once you've mastered picking simple locks, you will never look at a locked door the same way again. What once was a nigh-impenetrable barrier is now a gateway with a minor impediment holding it closed. Knowing practically just how simple it is for someone with the skills to bypass most locks will drastically shift your perspective on physical security.

A word of caution as you start down this path: the curse of knowledge cannot be removed. Once you start developing a security-focused mindset, you will begin to see the cracks in the very thin veneer of safety we usually ascribe to the world around us. This knowledge can be distressing for many people, particularly as they're just getting started.

Make sure your home is safe - a security system helps. Ensure your computers are locked down - firewalls, antivirus software, and regularly-maintained offsite backups are a must. Most importantly, find a mentor who has been in this space a while. They'll be able to add perspective and help you balance your newfound professional paranoia.<sup>3</sup>

1 Code fuzzing: <https://en.wikipedia.org/wiki/Fuzzing>

2 November: <https://phpa.me/no-bug-too-small>

3 professional paranoia.: <https://phpa.me/think-like-attacker>





## Digital security is no different than physical security.

A career in cybersecurity starts with two things: an interest and an understanding. You have to be interested in pursuing the career first.

If you've read this far, chances are good you qualify.

You also need to cultivate an understanding of security - not just in a cyber space, but security in general. This understanding is why it's important to be a tinkerer who understands how to break things and can see ways to bypass expected access patterns.

Once you have adopted and honed those three skills, you will be able to see any software application from a new perspective. How does the logic work under the hood? Can I crash the system by submitting special characters in an input field? What's the easiest way to trick the application into giving me data<sup>4</sup> to which I shouldn't have access?

These skills put you at the very beginning of a career track in cybersecurity. Start with these basics, and you've built a solid foundation to which you can add coding, certifications, and on-the-job experience. After that, the limits will only be what you choose.

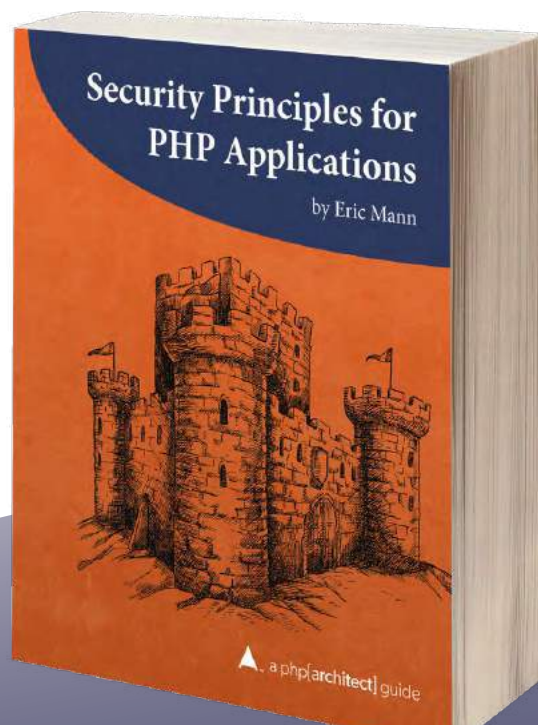


*Eric is a seasoned web developer experienced with multiple languages and platforms. He's been working with PHP for more than a decade and focuses his time on helping developers get started and learn new skills with their tech of choice. You can reach out to him directly via Twitter: [@EricMann](https://twitter.com/EricMann)*

### Related Reading

- *Security Corner: Enforcing Subresource Integrity* by Eric Mann, January 2021.  
<http://phpa.me/enforce-integrity>
- *Security Corner: Observable Security* by Eric Mann, September 2020.  
<https://phpa.me/security-sept-20>
- *Security Corner: The Pit of Success* by Eric Mann, September 2021.  
<https://phpa.me/security-pit-of-success>

<sup>4</sup> giving me data: <https://phpa.me/owasp-cheat-sheet>



**Order Your Copy**

<http://phpa.me/security-principles>



# Configuring PHP-FPM & Apache

Joe Ferguson

Last month we covered PHP and Apache and demonstrated how to get started with our custom virtual host in Apache and execute PHP via the `libapache2-mod-php` (`mod-php`) library. This month we will replace our use of `mod-php` with the Fast CGI Process Manager (FPM). Instead of bundling a PHP worker process in Apache, we'll use FPM as another running service that will execute PHP. The requests are normally handled by Apache but instead of using `mod-php`, we'll configure our virtual hosts to pass requests into the FPM service which will execute our PHP code and return the response to the Apache web server. One of the biggest benefits of using FPM with Apache is the ability to serve multiple PHP versions via the web server, which is not possible when using `mod-php`.

FPM gives us features such as process management with a graceful restart, stop and start. We're also able to start processes as different users, groups, or even in a chroot for isolating processes/applications. With `mod-php`, we're unable to restart a specific virtual host because PHP is directly attached to the Apache process. Think of virtual hosts as individual websites or applications hosted by your website. Virtual hosts are the web server configuration that describes what network interfaces to listen on, what domain names to respond to, and then how those requests are handled once a matching request is found.

If we have `paste.test` and `rfid.test` applications running on Apache via `mod-php`, we would restart them via `sudo service apache2 restart`, which would restart *both* applications. PHP-FPM extracts the execution of PHP for an application outside of the web server and we're able to restart FPM via `sudo service php-8.1-fpm` to apply PHP changes without restarting Apache. Doing so also allows us to apply changes to our Apache configuration or virtual hosts without restarting FPM.

## Diving into PHP-FPM and Apache

Getting started on a fresh Ubuntu 20.04 (the current Long Term Support version), we can install PHP, Apache, and FPM packages with the Install PHP 7.4, 8.0, PHP-FPM, and Apache2 Script. Much of this script has been taken from `laravel/settler`, which is MIT licensed.

Our script assumes you're using the fantastic Ubuntu Personal Package Archive by Ondřej Surý: PHP PPA. You can easily add the PPA via `sudo add-apt-repository ppa:ondrej/php` and run `apt update` if it's not automatically done for you to refresh the list of available packages. What makes Ondřej's PPA so powerful is it allows us to run *multiple versions* of PHP concurrently through the command line and PHP-FPM Server Application Programming Interfaces (SAPI). Doing so is how programs interact with the PHP service, whether at the command line or via a web server or other application server.

Outside of installing packages, our script will also configure several values for running Apache, FPM, and PHP as a specific user. Apache will run its processes and services as the `www-data` user and group by default. Since we're no longer using `mod-php` we can take advantage of FPM's ability to run services as individual users. For example, we replace `www-data` in the FPM configuration files for PHP 7.4 as shown in Listing 1.

### Listing 1.

```
1. sed -i "s/user = www-data/user = vagrant/" \
2.   /etc/php/7.4/fpm/pool.d/www.conf
3. sed -i "s/group = www-data/group = vagrant/" \
4.   /etc/php/7.4/fpm/pool.d/www.conf
5. sed -i "s/listen.owner.*listen.owner = vagrant/" \
6.   /etc/php/7.4/fpm/pool.d/www.conf
7. sed -i "s/listen.group.*listen.group = vagrant/" \
8.   /etc/php/7.4/fpm/pool.d/www.conf
9. sed -i "s/;listen.mode.*listen.mode = 0666/" \
10.  /etc/php/7.4/fpm/pool.d/www.conf
```

## Configure FPM

The configuration for PHP lives at `/etc/php`, and if we use `tree -L 2` (`sudo apt install tree`), we can see an overview of the folder layout in Listing 2.

Each PHP version is contained in its folder, and each sub-folder represents an individual SAPI of PHP. This configuration is how we're able to configure CLI PHP completely outside of FPM PHP as we'll typically want

### Listing 2.

```
1. /etc/php$ tree -L 2
2. .
3. └─ 7.4
4.   │   └─ cgi
5.   │   └─ cli
6.   │   └─ fpm
7.   │   └─ mods-available
8.   └─ phpdbg
9. └─ 8.0
10.   │   └─ cgi
11.   │   └─ cli
12.   │   └─ fpm
13.   │   └─ mods-available
14.   └─ phpdbg
```



```
192.168.10.10 fresh.test
192.168.10.10 paste.test
192.168.10.10 rfid.test
```

[illegible]

```

1. <VirtualHost *:80>
2.     ServerAdmin webmaster@localhost
3.     ServerName fresh.test
4.     ServerAlias www.fresh.test
5.     DocumentRoot /home/vagrant/fresh/public
6.
7.     <Directory /home/vagrant/fresh/public>
8.         AllowOverride All
9.         Require all granted
10.    </Directory>
11.
12.    <FilesMatch "\.+\.(ph(ar|p|tml))$">
13.        SetHandler "proxy:unix:/var/run/php/php8.0-fpm.sock|fcgi://localhost"
14.    </FilesMatch>
15.
16.    ErrorLog ${APACHE_LOG_DIR}/fresh.test-error.log
17.    CustomLog ${APACHE_LOG_DIR}/fresh.test-access.log combined
18. </VirtualHost>

```





## Listing 4.

```

1. ; The address on which to accept FastCGI requests.
2. ; Valid syntaxes are:
3. ;   'ip.add.re.ss:port'   - to listen on a TCP socket to a specific IPv4 address on
4. ;                           a specific port;
5. ;   '[ip:6:addr:ess]:port' - to listen on a TCP socket to a specific IPv6 address on
6. ;                           a specific port;
7. ;   'port'                 - to listen on a TCP socket to all addresses
8. ;                           (IPv6 and IPv4-mapped) on a specific port;
9. ;   '/path/to/unix/socket' - to listen on a unix socket.
10. ; Note: This value is mandatory.
11. listen = /var/run/php/php8.0-fpm.sock

```

The `<Directory /home/vagrant/fresh/public>` directive defines the webroot for our virtual host. I have used Git to clone my projects to my machine, and we point `Directory` to the webroot of our application. Typically this is `public` but could also be `web` or `public_html` depending on your application or hosting provider. We then use `AllowOverride All` to allow the virtual host to read configuration overrides from a `.htaccess` file.

The next directive in our virtual host is `FilesMatch`, which is how we match requests for file types and pass the request over to the FPM server. If a file matches our regular expression `".+\.php(ar|p|tml)$"` we use `SetHandler"proxy:unix:/var/run/php/php8.0-fpm.sock|fcgi://localhost"` to proxy the request to FPM. Our example in Listing 4 uses the Unix socket path `/var/run/php/php8.0-fpm.sock` but you can also use IP addresses if desired such as `127.0.0.1:9000`; If you would prefer to listen to a TCP socket you can change the `listen` value in `/etc/php/fpm/8.0/pool.d/www.conf`.

## Overriding Apache with .htaccess

Apache overrides via `.htaccess` are quite common. Laravel uses the method shown in Listing 5 as default when you create a new project.

The `RewriteCond` defines URL patterns to match, and if a match is found, the `RewriteRule` is applied to the condition. These are the fundamental basics that allow pretty URL rewriting in PHP applications and allow Apache to conditionally redirect users based on any number of variables. For example, if the application had a URL path such as `users/edit/profile` and a change in the application required that path to now be `users/profile/edit`, we could easily add the following to redirect users that may have bookmarked the old path to the new path without them having to do anything. We can also write a condition that rewrites any connections to port 80 (non HTTPS) to the HTTPS URL of our application.

Note: the backslashes(\) below are not valid syntax in an `.htaccess` file. It is only used here to break long lines for publication.

RewriteEngine On

```
#Redirect an old path to a new path
Redirect 301 /users/edit/profile \
http://fresh.test/users/profile/edit
```

```
#Require all connections get redirected to SSL
RewriteCond %{SERVER_PORT} 80 \
RewriteRule ^(.*)$ https://fresh.test/$1 [R,L]
```

For comparison, CakePHP 4 ships with the following `.htaccess` and has documentation on extending this functionality for your project based on the application's requirements.

```

<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteRule ^ index.php [L]
</IfModule>

```

## Listing 5.

```

1. <IfModule mod_rewrite.c>
2.   <IfModule mod_negotiation.c>
3.     Options -MultiViews -Indexes
4.   </IfModule>
5.
6.   RewriteEngine On
7.
8.   # Handle Authorization Header
9.   RewriteCond %{HTTP:Authorization} .
10.  RewriteRule .* - [E=HTTP_AUTHORIZATION:%{HTTP:Authorization}]
11.
12.  # Redirect Trailing Slashes If Not A Folder...
13.  RewriteCond %{REQUEST_FILENAME} !-d
14.  RewriteCond %{REQUEST_URI} (.+)/$
15.  RewriteRule ^ %1 [L,R=301]
16.
17.  # Send Requests To Front Controller...
18.  RewriteCond %{REQUEST_FILENAME} !-d
19.  RewriteCond %{REQUEST_FILENAME} !-f
20.  RewriteRule ^ index.php [L]
21. </IfModule>

```



## Listing 6.

```

1. <VirtualHost *:80>
2.   ServerAdmin webmaster@localhost
3.   ServerName paste.test
4.   DocumentRoot /home/vagrant/paste/public
5.
6.   <Directory /home/vagrant/paste/public>
7.     AllowOverride All
8.     Require all granted
9.   </Directory>
10.
11.   <FilesMatch "\.+\.(ph(ar|p|tml))$">
12.     SetHandler "proxy:unix:/var/run/php/php7.4-fpm.sock|fcgi://localhost"
13.   </FilesMatch>
14.
15.   ErrorLog ${APACHE_LOG_DIR}/paste.test-error.log
16.   CustomLog ${APACHE_LOG_DIR}/paste.test-access.log combined
17. </VirtualHost>

```

## Multiple Concurrent PHP versions

We installed PHP 8.0 and PHP 7.4 and configured fresh.test to use PHP 8.0. Our paste.test site hasn't been upgraded to PHP 8.0 yet so we need to use PHP 7.4 as 8.0 will cause issues. We can change the `SetHandler` "proxy:unix:/var/run/php/php8.0-fpm.sock|fcgi://localhost" directive of the virtual host configuration as shown in Listing 6.

The one-line change instructs fresh.test to use php7.4-fpm service instead of php8.0-fpm and will run both sites concurrently, something we were unable to do with mod-php. This flexibility allows you to easily run applications side by side and compare how they respond to being run by different versions of PHP.

## Conclusion

We've covered installing PHP 8.0, PHP 7.4, Apache, FPM, and configured virtual hosts to serve our local applications. We've also covered basic URL rewriting and redirection abilities which can be leveraged using `.htaccess` files. We also demonstrated how to configure multiple versions of PHP to be used by Apache via changing the `SetHandler` value to another FPM version in `SetHandler` directive. You can take these examples and expand them to your projects or use them as a starting point to experiment with PHP-FPM. If you would prefer to test Apache and FPM in a sandbox virtual machine, you can do that via Laravel Homestead<sup>1</sup> by setting your site's type to `apache` to instruct Homestead to use Apache instead of `nginx`. Happy web server configuring!



*Joe Ferguson is a PHP developer and community organizer. He is involved with many different technology related initiatives in Memphis including the Memphis PHP User group. He's been married to his extremely supportive and amazing wife for a really long time and she turned him into a crazy cat man. They live in the Memphis suburbs with their two cats. [@JoePFerguson](#)*

<sup>1</sup> Laravel Homestead: <https://docs.laravel.com/homestead>

**Using Xdebug to squash bugs, identify bottlenecks, and boost productivity?**



Become a Pro or Business supporter to help ongoing development.

Supporters get help via email and elevated issue priority.

<https://xdebug.org/support>

[support@xdebug.org](mailto:support@xdebug.org)

# Interview with PHP 8.1 Release Manager Ben Ramsey

*Eric Van Johnson*

Last month we talked with Patrick Allaert, the release manager for PHP 8.1, but Patrick is only one member of the dynamic duo known as the first-time release managers. The core team had been pairing one veteran release manager with one new release manager, but for PHP 8.1, they decided to have two new release managers. Although it may be Ben Ramsey's first time being a release manager, he is not new to PHP Internals. As long as I've been lurking on the PHP Internal Mailing List, I've seen Ben contributing to the discussions and voting on RFCs. This month we learn a little more about Ben and his journey into the PHP coding life.



## Can you let our readers know who you are?

My name is Ben Ramsey, though in most places online, I'm known only as Ramsey. I've been online since the early 90s, and I've been building websites for about as long. I created my high school's first website, in fact, working with our newly-established (at the time) Media Club.

Even though I had a lot of fun building websites, and I got pretty good at it, I didn't think it was something that could turn into a career; I was pretty naïve. So, I ended up going to college and getting a Bachelor of Arts in English. Oddly enough, I think that degree has turned out to be pretty valuable in this industry that often suffers from communication problems.

At college, I got a job as the webmaster for my school's sports information office. I kept the site up-to-date with the latest stats for all the players and games. I wrote some Perl scripts to process the sports data they sent me. Soon after, I got a job working for a friend's brother at his web development agency in Atlanta, Georgia, and the rest is history. I enjoyed my job so much that I stayed in the industry.

Along the way, I got involved in the PHP community. I founded the Atlanta PHP user group and started speaking at conferences. I've organized PHP Appalachia (2006), PHP Community Conference (2011), and PHP TestFest (2017). When I moved to Nashville, Tennessee, I started helping out as a co-organizer at the Nashville PHP user group. I've contributed to a few books on PHP development, and I've written a number of articles. I used to blog a lot, but now I just complain on Twitter.

## How did you get started with coding, and what got you interested in PHP?

I started programming when I was in the fourth grade. My parents found an Atari 400 at a garage sale, and it included several cartridges and a cassette tape deck for recording and playing back programs. One of the cartridges was Basic, and I began learning how to use it to create a Mad Libs game. When it ran, it presented the user with prompts like "enter a noun," "enter a verb ending in -ing," "enter the name of a person in the room," etc. When it finished, it printed a Mad Libs story to the screen, with all the blanks filled in.

Fast forward a decade or so, and I was building websites, using Perl for server-side scripting. Someone introduced me to Visual Basic Script, so I started building with Microsoft Active Server Pages (ASP). To cut down on the cost of purchasing COM objects for database connectivity, etc., I began looking for an alternative to ASP. That's when I found PHP.

This was around late-2000, and I decided to install PHP myself onto our production server, a co-located Cobalt RaQ 4r running Red Hat Linux 6.2. Interestingly, we were running ASP websites on a Linux server, using Chili!Soft ASP. To make a long story short, I broke all of our websites for a week. The folks at Chili!Soft were nice enough to investigate, but they couldn't figure out what was wrong. It was only after disabling PHP that our websites began working again. It turned out that I had enabled PHP for more than just .php files, and ASP tags were turned on, so PHP was trying to process all the ASP scripts!

After this fiasco, I started asking questions on the PHP general mailing list. Everyone was so helpful, and that's what hooked me. It's never been about the language for me; it's



always been about the community. That's what makes PHP great. PHP/Professional Life Questions

**What are some of your proudest accomplishments or achievements?**

One thing I'm really proud of is having a PHP package that's been installed over 215 million times. I started building ramsey/uuid in 2012 mainly to learn about Composer and how to build packages. In the past ten years, it's grown to be an important part of the ecosystem.

How much PHP coding do you do nowadays?

I spend most of my time working on the backend, and the overwhelming majority of my work is in PHP. I'll write an occasional Bash script, and sometimes I'll work in JavaScript or TypeScript.

**What challenges do you see for PHP in the coming years?**

We've had over ten great years of Composer and Packagist. One of the biggest problems I see on the horizon is a growing collection of unmaintained packages, many of which haven't been updated to support PHP 8. While Composer is one of the better package managers among programming language communities, I'm beginning to feel the weight of "dependency hell" piling up.

Along the same lines, I think the pace of PHP releases is too fast. Package maintainers spend too much time fixing issues so their packages can run on the next PHP version while continuing to work on older versions. I worry that this is stretching thin our already limited community resources. Some of this might level out as we put PHP 7 behind us and see PHP 8 mature, though.

As we look ahead, I think our community faces some growth challenges. For the past decade, JavaScript has led the web development world. Many new developers entered the space having never worked with PHP. In January 2022, for the first time since PHP appeared on the TIOBE Index, it dropped out of the top ten. Python is the language of the year for the second year in a row. If we don't want our core or userspace to stagnate, we'll need to get more folks using and contributing to PHP.

**What excites you about PHP's future?**

I think PHP still has a lot to offer in terms of quick, practical solutions to solving problems. We've been adding new features that bring better developer experiences to the language, such as match, enums, and intersection types, to name a few. But it's the performance improvements and functionality like Fibers and the JIT compiler that I'm really excited about. I can't wait to see what the community builds using these features. I think the userspace is where we'll begin to see the most innovation over the next few years.

**How long have you been involved with Internals?**

I think this depends on what you mean by "involved." I've been on the Internals mailing list since 2003. I've chimed in on occasion to take part in discussions or ask questions. I've

known and am friends with a number of internal developers, chatting with them on IRC or meeting up at conferences over the years. Now, I'm trying to be more involved by paying more attention to the mailing list and contributing code where I can.

**What got you involved with PHP Internals?**

I got involved with internals through relationships I built up over the years with internal developers. I saw what they were doing and how they were contributing, and I wanted to do the same. Seeing their contributions was and is inspiring, and I wanted to be part of something that has a lasting impact on the lives of many developers around the world.

**What is it like being a release manager for PHP? What are the responsibilities?**

The release manager role probably sounds more important or grandiose than it is. I don't have a lot of authority or decision-making power; it's grunt work. The release managers are responsible for tagging releases, packaging the code for the release into compressed tarballs, and announcing the release on the website and mailing lists.

**What is involved with releasing PHP 8.1?**

When releasing PHP, I use the command line exclusively. I edit a few files to bump the PHP version numbers. Use Git to create branches and tags. I build PHP locally and run its test suite. There's a script I run to create the source tarballs, which I sign with my GPG key. Then, I add the tarballs and signatures to a distribution repository. Last, I announce the release by updating the website and sending emails to several mailing lists.

The full release process is described in detail in the release-process.md document<sup>1</sup> in the PHP source repository.

**How has it been working with Joe Watkins through this process? This is the first time there have been 2 "new" release managers. How has it been learning this process with Patrick Allaert?**

Joe has been a great mentor, as Patrick and I have both learned the release process. For the first few releases, we joined Google Hangouts together, and he walked us through the process and helped us with any questions or concerns we had as we began to handle the releases on our own. Now, Patrick and I trade off the releases, and Joe provides support. I think this has worked out very well, and down the road, Patrick and I will be well-equipped to mentor other release managers.

**What do you see are some of the benefits of the PHP Community?**

The biggest benefit for me is in finding new friends and building relationships. Sometimes these relationships are personal, sometimes professional, oftentimes both. Through

---

<sup>1</sup> release-process.md document:

<https://github.com/php/php-src/blob/master/docs/release-process.md>





my involvement with the community, I feel like I have direct access to the individuals who maintain many of the tools I use on a daily basis, and I can regularly approach them for help. Since 2004, every job I've had came through connections in the community.

**Are there any memorable experiences you can share that you experienced in the PHP Community?**

Over the last 20 years, there are many, many awesome experiences I've been fortunate enough to have through my involvement in the PHP community. Perhaps one that sticks out to me was a two-week, seven-city conference tour that php[architect] did back in 2009. I had the privilege of speaking in all seven cities and traveling with the group for the full two weeks. We went from San Francisco to Los Angeles to Dallas, Atlanta, Miami, DC, and New York. It was a lot of fun, and I'd do it again at the drop of a hat.



*Eric Van Johnson is the CTO of DiegoDev Group, LLC. A group of passionate and talented developers that strive to provide outstanding services. Organizers of San Diego PHP (SDPHP) and podcaster with php[podcast], PHPUgly, and PHPRoundtable. A husband, father, and enjoyer of scotch and baseball. @shocm*

**Related Reading**

- *Community Corner: Let's Talk Xdebug* by Eric Van Johnson, June 2020.  
<http://phpa.me/cc-xdebug>
- *Community Corner: PHP 8 Release Managers: Interview with Sara Golemon and Gabriel Caruso, Part 1* by Eric Van Johnson, July 2020.  
<http://phpa.me/php7-release-managers>

## Web Apps • Mobile Apps • E-Commerce

# A DIFFERENT DEVELOPMENT EXPERIENCE

**Developers who care about the code they create, the communities they build, and the solutions they implement.**



DIEGODEV.COM  
(406) PHP-CODE or (406) 747-2633

# New and Noteworthy

## PHP Releases

PHP 8.1.2 Released!:

<https://www.php.net/archive/2022.php#2022-01-21-1>

PHP 8.0.15 (Bug Fix Release):

<https://www.php.net/archive/2022.php#2022-01-20-1>

## News

### PHP Foundation Update

The PHP Foundation has nearly 1200 contributors and over \$300,000USD.

<https://opencollective.com/phpfoundation>

### Dealing with Dependencies

Brent shares his thoughts on dealing with dependencies (including PHP itself).

<https://stitcher.io/blog/dealing-with-dependencies>

### PHP in 2021 (video)

Brent shares a video with his roundup of 2021 in the php ecosystem.

<https://stitcher.io/blog/php-in-2021-video>

### PHP in 7 Minutes (video)

What kind of language is PHP in 2022? Find out in 7 minutes!

<https://www.youtube.com/watch?v=IfcFQxYPTxo>

### 8 Annoying Things That Can Make Any Programmer Go Crazy

Programmers are generally peaceful creatures. So, here are the top things that can make a programmer go crazy.

<http://phpa.me/8-annoying>

### Best Tools to Work Smarter

Claire shares simple ways to impress your manager by being more efficient and not spending extra work hours to show that you are the next model employee of the company.

<http://phpa.me/work-smarter>

### PHP Annotated by JetBrains

JetBrains shares lots of information from the PHP ecosystem.

<http://phpa.me/annotated-01-2022>



turnoff.us | Daniel Stori  
Shared with permission from the artist

# Finding Integer Factors

Oscar Merida

This article looks at solutions for finding integer factors for a given integer. Another way to put this problem is to decompose an integer into a multiple of two smaller integers. Besides hearkening back to Algebra classes of yore, we'll discuss applications of this technique.

## Applications

Have you asked, “is 288 divisible by 9” or some variation of it? When we want to know if one integer divides cleanly—that is, without a remainder—into another integer, we’re asking if it is a factor of the larger number.

Cryptography is the most well-known application for integer factorization. Modern cryptographic algorithms rely on factorization being non-trivial. A fast method for integer factorization would break algorithms like RSA encryption. A user’s public key is based on two large prime numbers. To maintain security, we must keep the primes secret. And, for the same reason, they should be very, very large because part of the public key is based on the product of these two primes. If it were feasible to factor the public key, an attacker could recreate the private key. This problem is known as the RSA problem<sup>1</sup>. By using large integers, measured by the number of bits needed to store them, we rely on the fact that it would take significant computing resources and time to crack.

*In cryptography, the RSA problem summarizes the task of performing an RSA private-key operation given only the public key. The RSA algorithm raises a message to an exponent, modulo a composite number  $N$  whose factors are not known. Thus, the task can be neatly described as finding the  $e$ th roots of an arbitrary number, modulo  $N$ . For large RSA key sizes (in excess*

*of 1024 bits), no efficient method for solving this problem is known; if an efficient method is ever developed, it would threaten the current or eventual security of RSA-based cryptosystems—both for public-key encryption and digital signatures.*

We’ll need new encryption algorithms with Quantum computing on the horizon. That search is on “Quantum computers could crack today’s encrypted messages<sup>2</sup>. That’s a problem.”

*In 1994, Peter Shor, a professor at MIT, figured out that quantum computers could find the prime factors of numbers through a technique now named after him. Shor’s algorithm was the spark that ignited quantum computing interest from companies, academics, and intelligence agencies, says Seth Lloyd, another MIT professor and a pioneer of the field.*

## Recap

Given a positive integer, write a PHP script that finds and outputs the pairs of positive integers that can be multiplied, in any order, to equal that integer.

For example, given the integer 24, your script should output:

```
1,24
2,12
3,8
4,6
```

**Bonus:** For any positive or negative integer, output the pairs of integers that can be multiplied to equal that integer.

## A Naive Loop

From last month’s FizzBuzz puzzle, we know we can use the modulus operator % to test if an integer divides evenly into another integer. We can use a naive loop that tests every integer from 2 to our \$product and output pairs with no remainder. See Listing 1.

Listing 1.

```
1. $product = 24;
2.
3. // every integer can be multiplied by 1
4. echo "1, $product" . PHP_EOL;
5.
6. for ($i = 2; $i <= $product; $i++) {
7.     if ($product % $i == 0) {
8.         $factor = (int) $product / $i;
9.         echo "$i, $factor" . PHP_EOL;
10.    }
11. }
```

My first attempt is close but repeats factor pairs. 2 × 12 is the same as 12 × 2, and it’s redundant to include it here.

```
1, 24
2, 12
3, 8
4, 6
6, 4
8, 3
12, 2
24, 1
```

<sup>1</sup> RSA problem:  
[https://en.wikipedia.org/wiki/RSA\\_problem](https://en.wikipedia.org/wiki/RSA_problem)

<sup>2</sup> crack today’s encrypted messages:  
<https://phpa.me/cnet-quantum-crack>



## Refining the Loop

We could use arrays to track our pairs and then filter out identical pairs. However, that sounds like a lot of work and memory to juggle. An insight that may help us here is that somewhere around  $4 \times 6$ , our pairs flip but don't change. Let's look at the factors for 36:

```
1, 36
2, 18
3, 12
4, 9
6, 6
9, 4
12, 3
18, 2
36, 1
```

Do you see it? The flip happens at  $6 \times 6$ , and 6 is the square root of 36. We only need to test factors from 2 to  $\sqrt{\$product}$ . We can update the loop as shown in Listing 2.

Listing 2.

```
1. $product = 24;
2.
3. // every integer can be multiplied by 1
4. echo "1, $product" . PHP_EOL;
5.
6. $limit = sqrt($product);
7. for ($i = 2; $i <= $limit; $i++) {
8.     if ($product % $i == 0) {
9.         $factor = (int) $product / $i;
10.        echo "$i, $factor" . PHP_EOL;
11.    }
12. }
```

This update loop gives us the output below and runs twice as fast since it tests half as many integers.

```
1, 24
2, 12
3, 8
4, 6
```

## The Functional Approach

We need to test integers between 1 and the square root of our target number. Instead of looping, we can use a functional approach to find a solution. See Listing 3, which finds the factors of 96.

Running it shows:

```
Factors of 96
1 x 96
2 x 48
3 x 32
4 x 24
6 x 16
8 x 12
```

Listing 3.

```
1. // integer to factor
2. $product = 96;
3.
4. // initialize an array with all possible integer factors
5. $factors = range(1, (int) sqrt($product));
6.
7. // keep the ones that divide by zero
8. $factors = array_filter($factors,
9.     function($factor) use ($product) {
10.         return ($product % $factor == 0);
11.     }
12. );
13.
14. // now output the pairs
15. echo "Factors of $product\n";
16. foreach ($factors as $factor) {
17.     echo $factor . ' x ' . ($product / $factor) . "\n";
18. }
19.
20. die();
21.
22. // every integer can be multiplied by 1
23. echo "1, $product" . PHP_EOL;
24.
25. for ($i = 2; $i <= $product; $i++) {
26.     if ($product % $i == 0) {
27.         $factor = (int) $product / $i;
28.         echo "$i, $factor" . PHP_EOL;
29.     }
30. }
```

Listing 4.

```
1. // integer to factor
2. $product = 96;
3.
4. // initialize an array with all possible integer factors
5. $factors = range(1, (int) sqrt($product));
6.
7. // keep the ones that divide by zero
8. $factors = array_map(
9.     function($factor) use ($product) {
10.         if ($product % $factor == 0) {
11.             return [$factor, $product / $factor];
12.         }
13.     }, $factors
14. );
15.
16. $factors = array_filter($factors);
17.
18. // now output the pairs
19. echo "Factors of $product\n";
20. foreach ($factors as $pair) {
21.     echo $pair[0] . ' x ' . $pair[1] . "\n";
22. }
```

It works for outputting the pairs of factors but what if we want to store the pairs for further processing? We can use `array_map()`<sup>3</sup> to generate a new array. Then, we can keep the non-empty values as shown in Listing 4.

3 `array_map()`: [https://php.net/array\\_map](https://php.net/array_map)





For later re-use, we can turn it into a function as shown in Listing 5.

Listing 5.

```
1. function findFactors(int $product) : array {
2.     $factors = range(1, (int) sqrt($product));
3.     $factors = array_map(
4.         function($factor) use ($product) {
5.             if ($product % $factor == 0) {
6.                 return [$factor, $product / $factor];
7.             }
8.         }, $factors
9.     );
10.
11.     return array_filter($factors);
12. }
```

## Non-positive Factors

Once we have the set of positive factors, we can find the rest. Remember how signed integer multiplication works:

- The product of two negative integers will be positive
- The product of a positive and a negative integer will be negative.

We use our function to find the positive factors first. Then, depending on the sign of our integer, we can generate the other valid pairs. See Listing 6. Keep in mind that if we're looking for factors of a negative integer, we don't return pairs where the sign of both factors are the same. That is, multiplying two positive or two negative integers never gives a negative product.

Listing 6.

```
1. function findAllFactors(int $product) : array {
2.     // get the positive factors
3.     $pairs = findFactors(abs($product));
4.     $negatives = [];
5.
6.     if ($product > 0) {
7.         // also add pairs of negative integers
8.         foreach ($pairs as $pair) {
9.             $negatives[] = [$pair[0] * -1, $pair[1] * -1];
10.        }
11.
12.        return array_merge($pairs, $negatives);
13.    } elseif ($product < 0) {
14.        foreach ($pairs as $pair) {
15.            $negatives[] = [$pair[0] * -1, $pair[1]];
16.            $negatives[] = [$pair[0], $pair[1] * -1];
17.        }
18.
19.        return $negatives;
20.    }
21.
22.    return [];
23. }
```

We can get the factors for 60:

```
$pairs = findAllFactors(60);
echo "All factors of 60:\n";
foreach ($pairs as $pair) {
    echo $pair[0] . ' x ' . $pair[1] . "\n";
}
```

Which produces this output. Note the second set where both factors are negative.

All factors of 60:

```
1 x 60
2 x 30
3 x 20
4 x 15
5 x 12
6 x 10
-1 x -60
-2 x -30
-3 x -20
-4 x -15
-5 x -12
6 x -10
```

We can get all the factors for -32:

```
$pairs = findAllFactors(-32);
echo "All factors of -32:\n";
foreach ($pairs as $pair) {
    echo $pair[0] . ' x ' . $pair[1] . "\n";
}
```

Which produces the following. In this output, only one integer in each pair can be negative.

All factors of -32:

```
-1 x 32
1 x -32
-2 x 16
2 x -16
-4 x 8
4 x -8
```

## Next Month's Challenge: Prime Factors

Now that we can find the integer factors of a number let's find all the prime factors and see if a number is prime or not.

Write a script that will list all the prime factors of an integer. Also, indicate if the integer is prime.

For example, if the number is 24, your code should list the prime factors as:

2 x 2 x 2 x 3

or

2^3 \* 3



### Some Guidelines And Tips

The puzzles can be solved with pure PHP. No frameworks or libraries are required.

- Each solution is encapsulated in a function or a class, and I'll give sample output to test your solution against.
- You're encouraged to make your first attempt at solving the problem without using the web for clues.
- Refactoring is encouraged.
- I'm not looking for speed, cleverness, or elegance in the solutions. I'm looking for solutions that work.
- Go ahead and try many solutions if you like.
- PHP's interactive shell (php -a at the command line) or 3rd party tools like PsySH<sup>4</sup> can be helpful when working on your solution.
- To keep solutions brief, we'll omit the handling of out-of-range inputs or exception checking.

4 PsySH: <https://psysh.org>



Oscar Merida has been working with PHP since version 4 was released and is constantly learning new things about it and still remembers installing an early version on Apache. When not coding or writing, he enjoys RPGs, soccer, and drawing. [@omerida](#)

### Related Reading

- *PHP Puzzles: Factorials* by Sherri Wheeler, May 2020. <https://phpa.me/puzzles-may-20>
- *PHP Puzzles: Calculating Fibonacci Sequences* by Sherri Wheeler, June 2020. <https://phpa.me/puzzles-june-20>
- *PHP Puzzles: Destination Point* by Sherri Wheeler, November 2020. <https://phpa.me/puzzles-nov20>



## Listen to Episode 69

### Domain-Driven Resolutions

In this month's podcast, we get started with "Introducing FilterIterators" by Mauro Chojrin. This feature article will get you looking at your projects in a whole new way.



**Hosted By:**  
Eric Van Johnson and John Congdon

<https://phpa.me/podcast-ep-69>



# When You Know the Pattern

Edward Barnard

Software design patterns provide a “voice of experience” that can help solve the problem you’re trying to solve. However, it’s often not obvious how to implement any such design pattern within our modern PHP ecosystem. Here’s an example solution. We are implementing a memoizing<sup>1</sup> Registry that’s outside my CakePHP framework code but uses CakePHP’s database layer. In July 2020’s php[architect], I wrote a most excellent rant, “We Got Robbed.”<sup>2</sup>

## Rant

In July 2020’s php[architect], I wrote a most excellent rant, “We Got Robbed.” And so we did.

What’s the problem, and how does that problem affect our choice to turn to Domain-Driven Design?

Too often, modern software design falls to the side in favor of executing the current “agile” backlog. That situation has been perfectly fine within our PHP ecosystem in many cases. In other words, the standard PHP frameworks and open source projects have proven well up to the task of running most of the world’s websites. That’s why, historically, PHP projects often favored launching quickly, adding new features, and fighting today’s fires instead of design and planning, paying down technical debt, and formal automated testing.

Haven’t I just described Agile software development? No. On the contrary, Martin Fowler and Ron Jeffries, both signers of the original *Manifesto for Agile Software Development*<sup>3</sup> at Snowbird, call this “Dark Agile” or “Dark Scrum.”

Fowler says it’s just the name “agile” with none of the practices and values in place:

*This is actually even worse than just pretending to do agile, it’s actively using the name “agile” against the basic principles of what we were trying to do, when we talked about doing this kind of work in the late 90s and at Snowbird.*

We’re not here to solve, or even define, “agile.” We’re here to figure out how to turn to Domain-Driven Design (DDD) in response to “hitting the wall” with technical debt. When it gets harder and harder to add any new feature, we need to look at different ways of doing things.

I mention the Agile Manifesto because modern “agile” projects are often more in the nature of Dark Agile. We, collectively, have often moved away from the Agile Manifesto’s vision, with the result that we eventually “hit the wall.” It’s often become too painful to touch or change anything in the existing codebase.

## Design Patterns

We have a similar situation with software design patterns. The concept took hold in roughly the same time frame as Agile, and both concepts have been well-known for 20+ years. Both date from literally a generation ago!

Software design patterns, by definition, represent the voice of experience. I have found that solutions to various design and structural problems do exist and are well documented. The trouble is that few of these solutions have examples within our modern PHP ecosystem.

We do have books with design patterns implemented in PHP. But there’s still a problem. Those patterns tend to be shown in isolation rather than as part of a typical PHP framework application. Or, conversely, the example is so closely tied to a framework that it’s of little use to someone using a different framework.

PHP-based development tends to be biased in favor of getting a website up and running quickly. That’s where PHP projects shine; this is PHP’s sweet spot and has been since Rasmus Lerdorf created a set of “Personal Home Page Tools” (“PHP Tools”) circa 1994.

Many PHP frameworks promote this same bias. This is not a bad thing! It helps to have instructions to, for example, put database-related code in the “model” area, rendering and templates in the “view” area, and any decision-making or business logic in the “controller” area. We have a visible separation of concerns and a quickly-released website.

You know where this is going, so I won’t belabor the point! We add more code in the appropriate places each agile sprint and, after years have passed, each piece of code now interacts with so many other pieces of code that it’s difficult to know how or where to touch anything.

Meanwhile, software design patterns may well provide known solutions to the problem of the moment. The trouble is that it’s not at all obvious how to implement a design pattern within whatever PHP framework is in use at the moment.

1 memoizing: <https://en.wikipedia.org/wiki/Memoization>

2 We Got Robbed.: <https://www.phparch.com/article/sustainable-php-we-got-robbed/>

3 Manifesto for Agile Software Development: <https://agilemanifesto.org/>



## Clean Architecture

Robert C. Martin, also one of the original signers of the Agile Manifesto, advocates a comprehensive solution in his 2018 book *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. He has good solid ideas (which makes sense given that he created the SOLID acronym). The trouble is that he steps outside any framework or ecosystem.

We most certainly don't want to start from scratch without using any PHP framework to our advantage. We don't want to ignore and re-invent the decades of framework developers' experience running most of the world's websites.

What we can do, however, is take a careful look at whatever code we're writing. Note which areas can best use the framework and which areas require little or no framework support at all. We're laying out a separation of concerns.

I can best explain with an example.

## Season Registration

"Turning to Domain-Driven Design<sup>4</sup>" in January 2022 `php[architect]` explained that our project includes two workflows involving high school athletes:

- Create an online account
- Register to compete during a specific season

While implementing the second workflow, a pattern emerged. It has to do with the database design. When the athlete creates an online account, we insert a record in the `users` table with login information. However, an athlete competing on a high school team could later compete on a college team, become a coach, and so on.

Thus a given user can take on one of several different roles. Only one of those roles is active at any given time. The active `user_roles` record links that user to a specific team and defines that user's role on that team.

Do you see the pattern in this workflow? Each use case in the workflow begins with the user role and hops from table to table to assemble what's needed for fulfilling that use case. For example, since this is a season registration, we hop from the user role to the team, from the team up to its parent league, and from the league to the current season.

The Dragon Wrangling Pattern<sup>5</sup> (October 2021 `php[architect]`) enforces a careful separation of use cases. It also encourages a separation of concerns in that business logic is in one folder structure, and database-related logic is in a separate folder structure.

What I found was that I was looking up the same information over and over. The entire workflow needed the season id because it's registering the athlete for that particular season for that particular team. But to find the season id, we need the league, and for the league, we need the team, and we know the team from the user role.

Thus, I realized I was doing many "multi-hop" lookups (possibly as multiple table joins) because each part of the workflow needed the same few pieces of information. Furthermore, these lookups happen at one of our highest-traffic portions of the year (the last few hours of season registration).

There's a design pattern for this situation.

## Registry

On page 480 of *Patterns of Enterprise Application Architecture*, Martin Fowler introduces Registry as "a well-known object that other objects can use to find common objects and services."

*When you want to find an object you usually start with another object that has an association to it and use the association to navigate to it. Thus, if you want to find all the orders for a customer, you start with the customer object and use a method on it to get the orders.*

That does sound like our situation. Let's continue.

*However, in some cases, you won't have an appropriate object to start with. You may know the customer's ID number but not have a reference. In this case, you need some kind of lookup method—a finder—but the question remains: How do you get to the finder?*

*A Registry is essentially a global object, or at least it looks like one—even if it isn't as global as it may appear.*

After several pages of design discussion, Fowler explains:

*Despite the encapsulation of a method, a Registry is still global data and as such is something I'm uncomfortable using. I almost always see some form of Registry in an application, but I always try to access objects through regular inter-object references instead. Basically, you should only use Registry as the last resort.*

This sort of advice, by the way, is why it's always worthwhile to look up any design patterns that might apply to the problem at hand!

*There are alternatives to using a Registry. One is to pass around any widely needed data in parameters. The problem with this is that parameters are added to method calls where they aren't needed by the called method but only by some other method that's called several layers deep in the call tree. Passing a parameter around when it's not needed 90 percent of the time is what leads me to use a Registry instead.*

<sup>4</sup> Turning to Domain-Driven Design: <https://www.phparch.com/article/ddd-alley-turning-to-domain-driven-design/>

<sup>5</sup> The Dragon Wrangling Pattern: <https://www.phparch.com/article/design-patterns-by-moonlight-the-dragon-wrangling-pattern/>





## Memoization

As my “multi-hop lookup” pattern emerged during feature development, that led me to also consider the Memoization<sup>6</sup> pattern:

*In computing, memoization or memoisation is an optimization technique used primarily to speed up computer programs by storing the results of expensive function calls and returning the cached result when the same inputs occur again. Although related to caching, memoization refers to a specific case of optimization, distinguishing it from forms of caching such as buffering or page replacement.*

## System-wide Cache

As we consider our implementation, let’s step back and look at the larger picture. We’ll be looking up (and memoizing) user roles, team records, league records, season records, and so on.

- The user role is specific to a single user, and thus there’s little reason for a global cache containing this record.
- The team record applies to all team members, and if several members of the same team are all registering at the same time, it might be worthwhile to have a cached copy for more-rapid access (e.g., via Memcached or Redis).
- The league record applies to all teams in a U.S. state, and consequently everyone registering for all teams in that league.
- The season record applies to all members of all teams of all leagues making up the high school leagues.

As you can see, we have caching possibilities. Since the season record, for example, is very small (containing names such as “2022 Spring”), it might be worthwhile to cache all active rows at once.

However, as Dr. Donald Knuth has famously pointed out, Sir Tony Hoare noted that “premature optimization is the root of all evil.” That’s our reminder that we have no reason—yet—to consider system-wide caching of these specific lookups. However, by keeping this possibility in mind, we can easily prepare for that possibility should the need arise.

Let’s look at an implementation.

## Role-based Lookups

First, here’s a code fragment that uses our new RoleBasedLookup class.

```
public function loadInvoice(UserRole $userRole): Invoice
{
    $lookup = RoleBasedLookup::getInstance($userRole);
    $seasonId = $lookup->seasonId();
}
```

<sup>6</sup> Memoization: <https://en.wikipedia.org/wiki/Memoization>

You will recall that we discovered a pattern emerging: Given the user role, find something in the database. But we need more than just the user role. We also need the relevant record for the current season, or, specifically, that record’s id (\$seasonId). Rather than pass \$seasonId around from method to method to method, we look up the season id as needed.

## Array of Singletons

To get started, Listing 1 shows the constructor and how we build the Singleton.

Listing 1.

```
1. class RoleBasedLookup
2. {
3.     use AlertTrait;
4.     use ParticipantTrait;
5.     use SeasonTrait;
6.     private static array $instances = [];
7.     private ?League $league = null;
8.     private ?Participant $participant = null;
9.     private ?Season $season = null;
10.    private ?SeasonFee $seasonFee = null;
11.    private ?SeasonParticipantConfirmation
12.        $seasonParticipantConfirmation = null;
13.    private ?SeasonSchedule $seasonSchedule = null;
14.    private ?Team $team = null;
15.    private ?TeamSeasonProfile $teamSeasonProfile = null;
16.    #[Immutable(Immutable::CONSTRUCTOR_WRITE_SCOPE)]
17.    private UserRole $userRole;
18.    private function __construct(UserRole $userRole)
19.    {
20.        $this->userRole = $userRole;
21.        $this->loadModels();
22.    }
23.    private function loadModels(): void
24.    {
25.        $this->loadAlertMessagesTable();
26.        $this->loadAlertsTable();
27.        $this->loadLeaguesTable();
28.        $this->loadParticipantsTable();
29.        $this->loadSeasonFeesTable();
30.        $this->loadSeasonParticipantConfirmationsTable();
31.        $this->loadSeasonSchedulesTable();
32.        $this->loadSeasonsTable();
33.        $this->loadTeamSeasonProfilesTable();
34.        $this->loadTeamsTable();
35.    }
36.    public static function getInstance(UserRole $userRole)
37.        : RoleBasedLookup
38.    {
39.        $unique = implode(':',
40.            [$userRole->user_id,
41.            $userRole->role_id,
42.            $userRole->team_id]);
43.        if (!array_key_exists($unique, self::$instances)) {
44.            self::$instances[$unique] = new self($userRole);
45.        }
46.        return self::$instances[$unique];
47.    }
48. }
```



## Model Access As Trait

One thing that I implied, but didn't quite mention, is that `RoleBasedLookup` is outside my PHP framework's code structure. It doesn't actually matter where it is so long as it's easily found (as Fowler explained with the Registry pattern).

However, the whole point of the role-based lookups is to read records from the database. We are therefore closely tied to the PHP framework's data access layer. The CakePHP framework provides automatic access to its database layer, but since we're working outside that framework, we need to do a bit extra to gain that access. I separated that "extra work" into a set of traits, with one trait for each cluster of database tables.

Listing 2 is the start of `AlertTrait`, which provides access to a cluster of five tables. The code is CakePHP-specific, using the CakePHP-specific `LocatorAwareTrait`.

When using the trait, each table model becomes a protected property of that class. Since there's no need to run through the relatively slow initialization for every table model on every web request, I expect the class to call the `loadXXX()` method for those table models it will use—specifically, the `loadModels()` method of `RoleBasedLookup`.

It's not ideal to be loading the table models inside the constructor, but that seems the most straightforward way to make everything happen via `getInstance()`.

Note that we are potentially creating an array of Singletons, one for each user role. Do we really need to, given that we're only using one user role at a time? The answer is yes because users have the possibility of switching roles. We need to do some role-based lookups as part of the new role.

This approach is not suitable for a long-running process that might involve switching between hundreds of user roles. In that case, placing a limit of (for example) ten Singletons would protect us from unlimited memory use.

## Load Season

Here is how we implement obtaining the season id.

```
public function seasonId(): int
{
    $this->loadSeason();
    return $this->season->id ?? 0;
}
```

It's possible that the season record does not exist, in which case we return zero as the season id. That should, of course, never happen!

Listing 3 shows how we load the season record.

We first check to see if our season object is not null. That's the memoization pattern. If it's been loaded once, there's no need to load it again—which is why we designed the Registry as an array of Singletons. We have, in effect, global storage for the season record.

If we do indeed need to load the season record, we first need to load the league record. The league record, in turn,

### Listing 2.

```
1. trait AlertTrait
2. {
3.     use LocatorAwareTrait;
4.     protected AlertLevelsTable $alertLevelsTable;
5.     protected AlertMessagesTable $alertMessagesTable;
6.     protected AlertPeriodsTable $alertPeriodsTable;
7.     protected AlertTypesTable $alertTypesTable;
8.     protected AlertsTable $alertsTable;
9.     protected function loadAlertLevelsTable(): void
10.    {
11.        $this->alertLevelsTable = $this->alertLevelsTable();
12.    }
13.     protected function alertLevelsTable()
14.        : AlertLevelsTable
15.    {
16.        /** @noinspection PhpUnnecessaryLocalVariableInspection */
17.        /** @var AlertLevelsTable $table */
18.        $table = $this->getTableLocator()->get('AlertLevels');
19.        return $table;
20.    }
21. }
```

### Listing 3.

```
1. private function loadSeason(): void
2. {
3.     if (null !== $this->season) {
4.         return;
5.     }
6.     $this->loadLeague();
7.     if (null !== $this->league) {
8.         $where = [
9.             Season::FIELD_LEAGUE_TYPE_ID
10.            => $this->league->league_type_id,
11.             Season::FIELD_IS_ACTIVE => true,
12.         ];
13.         $order = [Season::FIELD_ID => 'DESC'];
14.         /** @var Season $entity */
15.         $entity = $this->seasonsTable
16.            ->find()
17.            ->where($where)
18.            ->order($order)
19.            ->first();
20.         if ($entity instanceof Season) {
21.             $this->season = $entity;
22.             return;
23.         }
24.     }
25.     $this->reportMissingRecord('season');
26. }
```

depends on having loaded the team record (not shown). In other words, memoization takes care of our "multi-hop" situation.

Finally, if the season record was not found, we fall through to report the missing record—an application of December



2021's "Designing for MySQL Transaction Failures" in php[architect]. If a team (or in this case, season) record got missed in setting up the current season, we'd be notified via the exception report.

## Opportunity for Premature Optimization

RoleBasedLookup, as shown here, caches (memoizes) up to eight database records. They all take the same program flow. Thus we can easily expand the class for anything else that needs to be determined based on the current user role.

We could also insert a cache-based lookup (e.g., via Memcached or Redis) before the database lookup. However, there's no reason to add the complexity without actual performance measurements demonstrating the need.

## Summary

Even when you know the design pattern, it's not always clear how or where to implement it. Your PHP framework of choice may well be getting in the way—obscuring your view of the code, so to speak.

Take a close look at what you're doing. Make a separation between what code is framework-related and what isn't. Identify and clearly show the pattern or patterns you're implementing.


We studied up on the Registry pattern. Fowler says to use it as a last resort. But Fowler also closely described our exact situation, in effect saying our situation is well suited to Registry.

I know from prior years that season registration is a period of peak usage. It, therefore, makes sense to *prepare* for later optimization. It equally makes sense to avoid over-engineering that aspect by leaving possible optimization for later.



*Ed Barnard had a front-row seat when the Morris Worm took down the Internet, November 1988. He was teaching CRAY-1 supercomputer operating system internals to analysts as they were being directly hit by the Worm. It was a busy week! Ed continues to indulge his interests in computer security and teaching software concepts to others.*  
[@ewbarnard](https://twitter.com/ewbarnard)

7 Designing for MySQL Transaction Failures: <https://www.phparch.com/article/designing-for-mysql-transaction-failures/>



# From Capone to Cray

WHERE COMPUTERS REALLY CAME FROM

by Edward Barnard

Why computers? Churchill destroyed the first ones to keep the secret. What was it like? Ed shares the answers!

<https://leanpub.com/capone>





# Every Which Way But Loose

Beth Tucker Long

I recently saw yet another discussion online about PHP's infamous naming convention exceptions and needle/haystack rotations. All the typical vitriol was there from the PHP haters, but I was very happy to see several new programmers add to the conversation or ask for clarifications and receive very supportive and educational responses from the thread. PHP is nothing if not consistent in its status as a recipient of the internet's hatred. However, I am forever proud of our community's ability to ignore the hate and still welcome and encourage newcomers.

## But do the haters have a point about our inconsistencies?

I mean, they are not wrong that sometimes it is "needle/haystack" and other times it is "haystack/needle." Sometimes it's "in\_" and other times it's "contains\_." I freely admit to needing to check the docs or use tooltips in an IDE when using functions I don't use very often because I can't remember which way it will go.

**Is it `phpinfo()` or `php_info()`? `var_dump()` or `var_dump()`? Has anyone thought of just fixing this?**

Most definitely. In fact, PHP RFC Consistent Function Names (`consistent_function_names`) will turn seven years old next month. This RFC proposes renaming any inconsistent functions and keeping their old names as aliases. No backward compatibility issues. Everyone can keep using whichever version they want to. It seems like a proposal that would satisfy the people who want everything renamed and those who do not want the names changed. So what's its status? You guessed it. After all these years, it is still listed as "Under Discussion."

**Why is it so difficult to make a decision on something that garners PHP so much hate? Especially when there is seemingly a perfect solution to make everyone happy?**

Well, that's the thing. Despite how it looks on the surface, renaming and aliasing is not a perfect solution. PHP has had these function names for so long they are expected to be a part of the code.

**Does this mean we can't change them because we don't like change?**

Absolutely not. What it means is that other libraries and codebases have been working around those function names for well over a decade or two. For many codebases, it means they have wrapper functions using the proposed standardized versions of the function names because they prefer standardization. Or they have built custom functions and used the proposed standardized function names because those weren't reserved. Standardizing the function names and order of variables means adding a lot of new reserved function names that could break a lot of code.

## Don't we add new functions to PHP all the time, which does the same thing?

True, but in those cases, the BC breaks come with the benefit of new functionality. In this case, we would be introducing a lot of BC breaks with no new functionality.

## Isn't a better dev experience worth it?

Maybe. Consider, though, the huge list of amazing new features, security patches, bug fixes, and everything else on the horizon for PHP. Do we really want to spend our limited core developer time on something that only improves the dev experience for some devs. It is really hard to justify this when it means losing time to work on a new feature or fix that could improve things for all devs. When dealing with limited volunteer resources, there are lots of tough decisions to make.

## But what about all the haters who say PHP is ugly because of this?

As the saying goes, haters gonna hate. In PHP's decades of life, there have always been haters. No matter what we do, there will still be haters. Look at the amazing modernization strides that PHP has made in the last five years! The perception that PHP is old, clunky, and ugly is not based on legitimate facts but rather long-held biases. Those biases aren't going to go away even if we made naming conventions perfect, which I'd venture to say that no language could truly achieve given the ever-changing nature of programming.

## So is this a dead issue?

Definitely not! There is merit to the concern and potential in the suggested solution. It just isn't the right time for this kind of a change, and that's why as we draw near to this RFC's 7th birthday, this continues to be "Under Discussion."

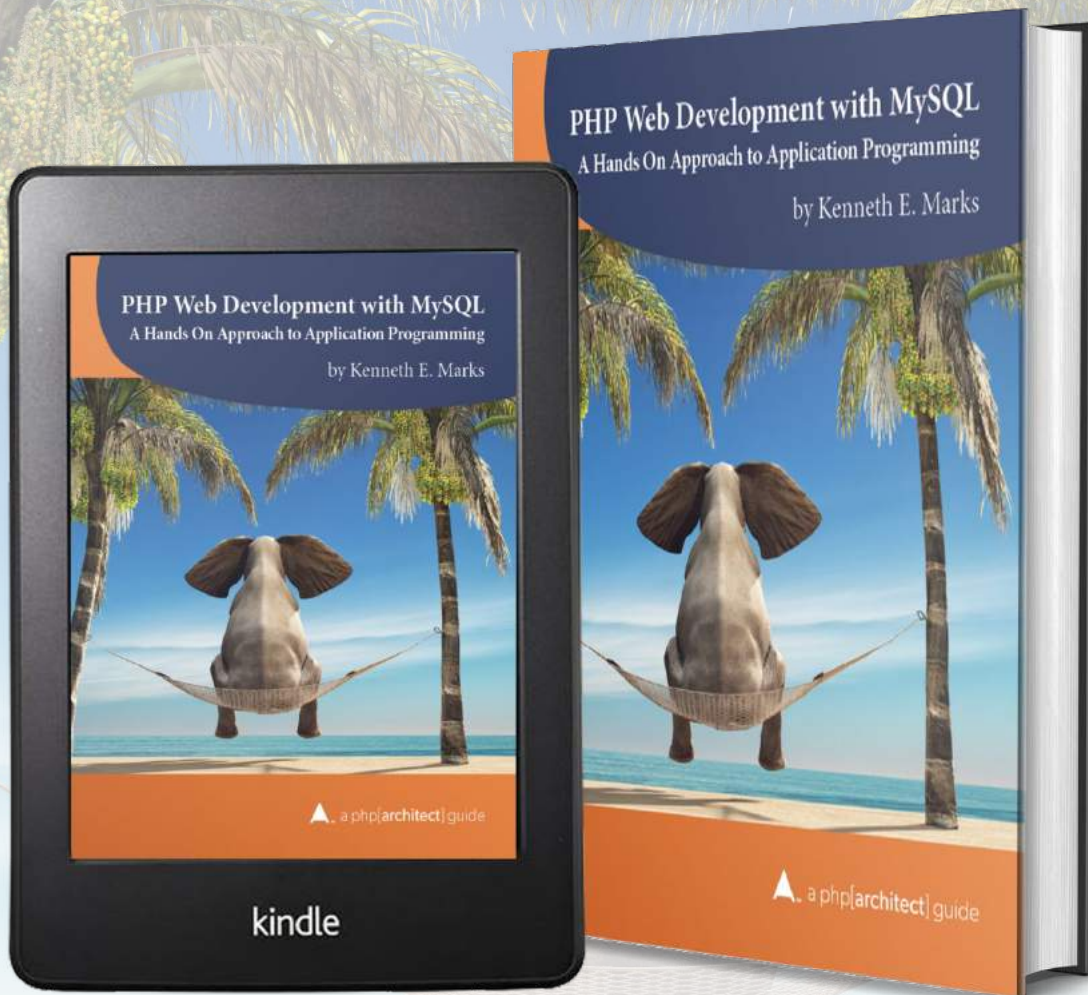


Beth Tucker Long is a developer and owner at Treeline Design, a web development company, and runs Exploricon, a gaming convention, along with her husband, Chris. She leads the Madison Web Design & Development and Full Stack Madison user groups. You can find her on her blog (<http://www.alittleofboth.com>) or on Twitter [@e3BethT](https://twitter.com/e3BethT)

## Related Reading

- [PHP RFC: Consistent Function Names](https://wiki.php.net/rfc/consistent_function_names)  
[https://wiki.php.net/rfc/consistent\\_function\\_names](https://wiki.php.net/rfc/consistent_function_names)





## Learn how to build dynamic and secure websites.

The book also walks you through building a typical Create-Read-Update-Delete (CRUD) application. Along the way, you'll get solid, practical advice on how to add authentication, handle file uploads, safely store passwords, application security, and more.

**Available in Print+Digital and Digital Editions.**

**Purchase Your Copy**  
<https://phpa.me/php-development-book>



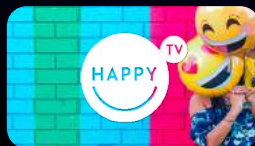
ATMOSPHERE

# Build the future of TV streaming with Atmosphere.

Help us become the global leader in entertainment for business with our modern tech stack.

Laravel 8 | PHP 8 | Vue.js | SQL | tvOS and iOS

Revolutionizing the TV Experience for Businesses



Employee-First Culture | Creative Office Space | Open Vacation Policy | Full Benefits | Daily Lunches | Company Outings



See our job openings at  
[atmosphere.tv/careers](https://atmosphere.tv/careers)