

# How to Seamlessly Upgrade Your Business to *php*8

# Table of contents

<b>1.</b>	<b>A Brief History of PHP Popularity</b>	<b>6</b>
1.1	The Current PHP Usage Timeline	8
<b>2.</b>	<b>Why You Need to Upgrade to PHP 8</b>	<b>11</b>
2.1	Convince Your Boss	15
<b>3.</b>	<b>The Upgrade Process</b>	<b>18</b>
3.1	I have PHP 5.x. What's next?	21
3.2	Choosing Migration to PHP 7.x	22
3.3	How Migrating to PHP 7.x Works	24
3.4	Choosing Migration to PHP 8.x	27
3.5	How Migrating to PHP 8.x Works	28


**So you're exploring how to upgrade your business's PHP applications to the latest major 8.x version for its powerful features and robust security measures. Great choice!**

What can you expect during a PHP 8 migration? What's the best way to prepare your code, technical teams and business operations for the biggest update in the history of PHP? What are the business risks of not upgrading?

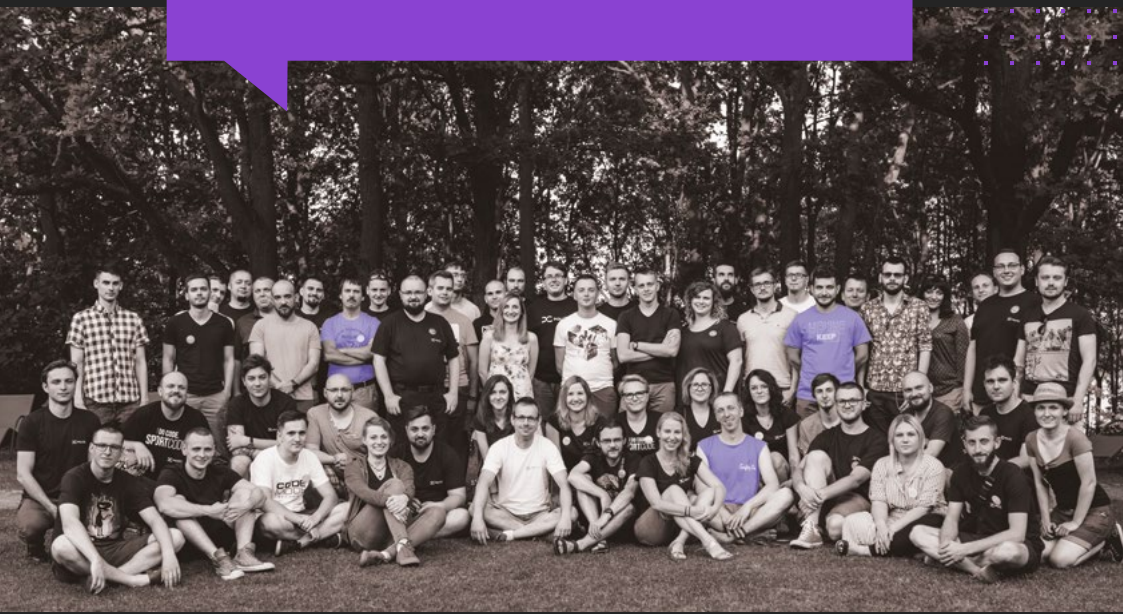
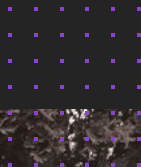
This eBook is designed to guide anyone, at any experience level, towards a seamless migration to PHP 8. It will cover why your business should consider upgrading, and what benefits and challenges await if you choose to do so. We will take you through everything you need to know and prepare beforehand, depending on which technology or version you're currently using.

If your site or business apps are running on PHP 5 or PHP 7, it is likely that you will need to refactor some code to make it PHP 8 compatible. Hopefully you'll be convinced by the end that it's worth the jump, because PHP 8.x includes significant improvements that will make your sites more efficient, reliable and secure.

**Let's get into it!**

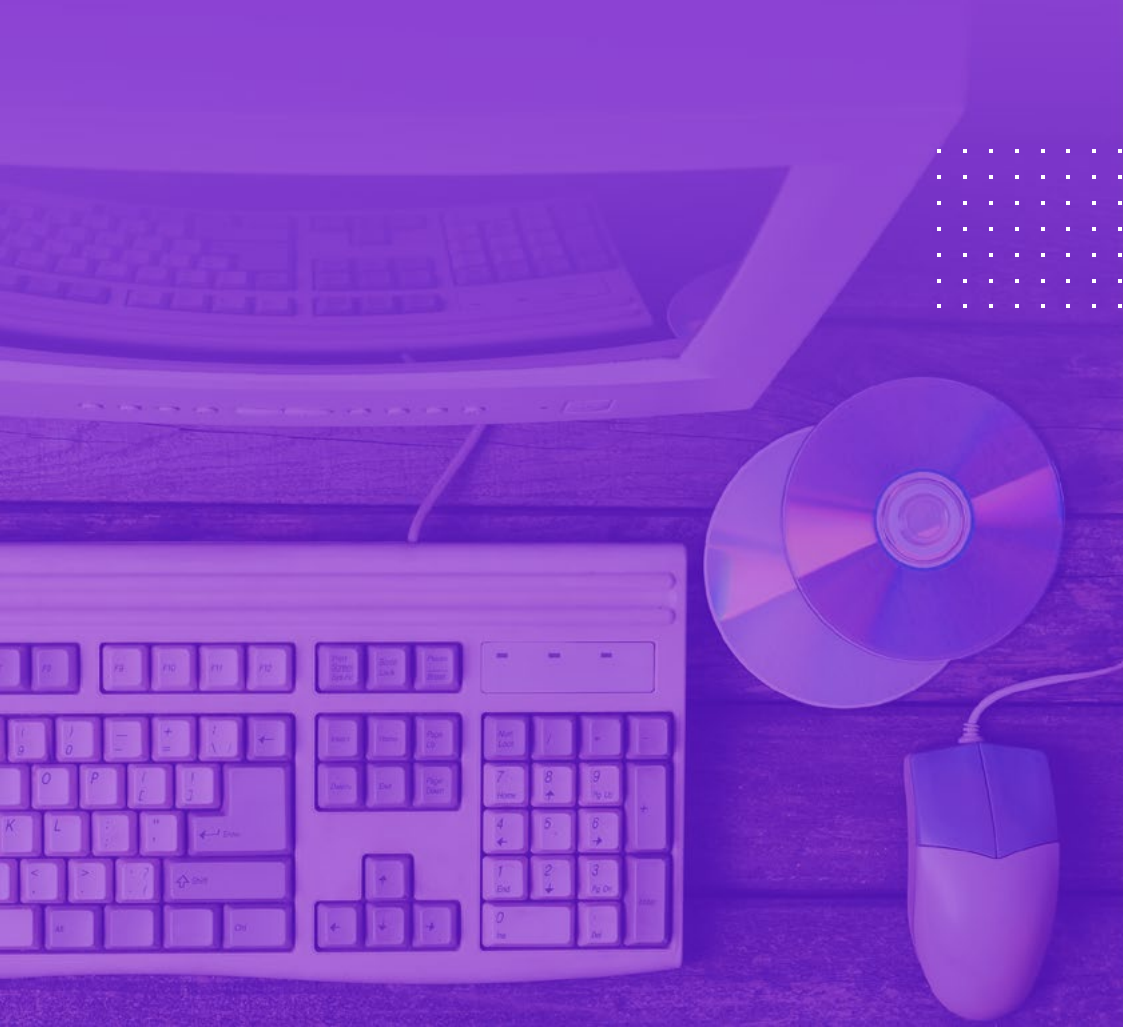


With 16 years of experience in building PHP–powered sites and web apps, the Dev Team at Polcode created this guide after conducting successful migrations for our clients worldwide. If your site runs on PHP and you’re not sure what to do next, we can audit your current site or apps and develop a plan for upgrading. Reach us at [polcode.com](https://polcode.com) to schedule a free consultation.



# 1

## A Brief History of PHP Popularity





# A Brief History of PHP Popularity

PHP began as a simple open–source project in 1994 by Rasmus Lerdorf, as an intuitive server–side scripting language, mostly used for templating and tracking simple page activities. The scripting tools were coined “Personal Home Page Tools,” and the PHP name has been retained ever since. In 1995, the source code was made available to the public, and anyone could use the code freely, motivating many developers to contribute bug fixes to the code and improve upon it as a community. Around 15,000 sites were using PHP within the first year of its official release.

In the following decade, PHP experienced rapid changes and a steady rate of adoption across the web. Sometime after PHP 3’s release in June 1998, an estimated 10% of the world’s web servers were running PHP. By the time PHP 4 (under the ‘Zend Engine’) was released in 1999, and the core redesign was finally adopted, it became difficult to measure just how much of the internet was running on PHP.

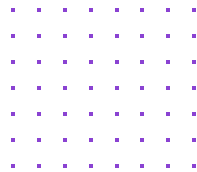
It wasn’t until 2004 (PHP 5) that we saw serious open–source development and popularity at the highest scales. PHP had just gone through

a major evolution, supporting object-oriented programming, PDO extension, and performance improvements. The world's top tech businesses at the time were predominantly using PHP to power their server-side programming, with hundreds of millions of sites potentially powered by PHP.

Today, PHP is still regarded as the open-source language that drives the modern web. By current estimates, almost 80% of the visible web is built upon PHP according to [W3techs](#). Slack, Wikipedia, WordPress and Mailchimp continue to rely on PHP. It remains an easy-to-learn language, and is backed by extremely productive frameworks like Laravel and Symfony that make it cost-effective for development.

Despite many elite tech giants using complex and intimidating tech stacks for their backend systems, PHP is still perceived as the reliable, productive, cost-efficient workhorse for most developers. It continues to be the backbone of many successful web projects.

However, problems can arise if businesses are not on the latest version of PHP 8.x. Support for the last PHP 7 version (7.4) ended on November 28, 2021 and security support will end in November 2022, potentially creating security risks.



# The Current PHP Usage Timeline

Year	Major Version	Description
1994–1996	<b>PHP/FI</b> – <b>PHP 2</b>	The earliest version of PHP was in use on at least 15,000 web sites around the world. In 1995 the source code was released publicly, allowing anyone to use it as they saw fit. Despite being open source and contributors providing bug fixes, it was still largely maintained by one single developer.
1997–1998	<b>PHP 3</b>	<p>By mid–1997, around 50,000 websites were powered by PHP, culminating in the release of PHP 3 which is the earliest version that resembles the PHP that we know today.</p> <p>Two developers led the major changes. Suraski and Andi Gutmans rewrote the core of PHP/FI 2.0 in 1997 and formed the underlying base of PHP 3, while changing the language's name to the recursive acronym PHP: Hypertext Preprocessor. Public testing of PHP 3 began and it was officially launched in June 1998.</p>
2000	<b>PHP 4</b>	PHP 4 introduced the concepts of core scalability and optimization. Under the new Zend Engine, PHP 4 could handle complex applications with more efficiency, and support integrations better than previous versions. By this time, the web was diversifying and expanding at an exponential rate, making it difficult to estimate PHP usage, but potentially millions of web servers were running PHP at the time.

continue  
on next page



---

2004–2015     **PHP 5**     By the time PHP 5 was released, it was already a world-wide phenomenon with potentially hundreds of millions of websites running PHP backend code. Dozens of core developers contributed to new code, and the PHP open source community consisted of hundreds of supporters, bug fixers and enthusiasts.

---

2015–2020     **PHP 7**     By the time PHP turned 20 years old, (there was no release for PHP 6) the seventh version contained the most modern version of PHP ever. It offered a complete language renewal powered by the Zend Engine 3, highly regarded for its ability to power mobile, web, cloud and enterprise applications with ease. By all attempts at estimates, the claim that, “Nearly 80% of the visible web is powered by PHP” was made around this time, and despite major contributors leaving the project’s core development, PHP continued to find success and adoption across many of the world’s enterprises and small businesses alike.

---

2020–Today     **PHP 8**     Today, PHP remains the most widely used server-side programming language in the world. The latest version of PHP introduced many powerful new features, opening up new possibilities for business products, as well as streamlining code execution performance. Frequent bugs and time-sink tasks have largely been optimized for developers, decreasing the time it will take for feature-rich services to reach the market.

---

# 2

## Why You Need to Upgrade to PHP 8



# Why You Need to Upgrade to PHP 8

**Explore the top benefits of upgrading to the latest version of PHP, as well as the consequences for not regularly maintaining a healthy and updated PHP codebase.**

## 1. Security Matters

Companies left running unsupported versions of PHP risk opening themselves to the latest hacks, ransomware and malicious attacks that have become all too common over the last few years. PHP's popularity makes it even more of a target for hackers targeting vulnerabilities. The best way to avoid these security issues is to update your PHP stack as soon as possible, and as frequently as possible.

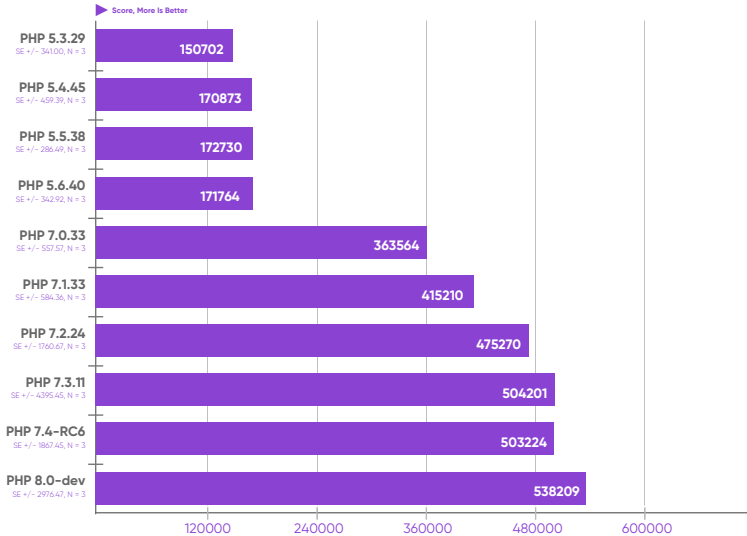
## 2. Peak Performance

As you can expect, being on the latest versions of PHP will produce better performance results overall, and the latest version of PHP 8.x is no different. Page load speeds, throughput, and utilization of resources under load have been optimized to handle the gargantuan amounts of data that the web now demands. Improvements to performance have direct benefits towards less memory usage, fewer servers, and happier customer experiences.

### PHPBench v0.8.1

PHP Benchmark Suite

ptsl  
OpenBenchmarking.org



## 3. Cost Efficient Coding

Software development has trended towards doing more with less, as business owners carefully measure between speed, quality, and cost. Upgrading to PHP 8 influences these factors almost immediately, largely

in part to its asynchronous design qualities and JIT compiler—allowing developers to quickly implement asynchronous applications that deliver continuously updated application data to users.

## 4. Developer–Talent Access

Finding talent to deal with a legacy codebase only goes in one direction as time goes on; it's harder to find, and more expensive to hire. It can be difficult to find programmers with a high level of knowledge about a legacy version, and many of your business requests will require ever–more complex 'hacks' to work around, increasing feature delivery time, frustration and budgets. Upgrading to the latest version of PHP ensures that your business can hire from a wider range of available talent pools.

## 5. Compatibility & Capability

Early on during a major version update, there are a few areas which are not code–compatible on the newer versions of PHP. For example, at the launch of PHP 8, some WordPress sites and themes were incompatible at launch. However, this is typically only temporary, and the real reward is about achieving new capabilities with rich feature sets, as each PHP version builds upon its predecessors. For instance, reclassifying and retyping error handling has relieved developers of the most distressing part of their jobs. Simple pleasures such as the @–operator no longer silencing fatal errors have saved many a headache on PHP 8, while dozens of other changes which can be found here: <https://stitcher.io/blog/new-in-php-8>, if you're curious.

## 6. Lower Maintenance Costs

Letting your PHP version fall behind the most current version is not just a security risk—it's a path towards technical debt, where it eventually becomes insurmountable to expedite the delivery of features and changes to your business. Even though your business may not have the resources or time available to upgrade to the latest version every single time, it is important to keep maintenance at a regular pace.

## 7. Faster Time to Market

And finally, the latest version of PHP optimizes the developer experience and code execution performance, decreasing the major bottlenecks of time (and frustration) that previous versions held over developers in the past. Most notably the JIT Compiler gives web developers a real performance boost in complex calculations, paving the way for data-heavy tasks; perhaps easing the creation of features centered around Artificial Intelligence and Machine Learning.



# Convince Your Boss

In our experience, making the switch to PHP 8 often comes with convincing a higher-up that software upgrades are not a luxury, but rather a necessity. If you or your team are worried about convincing a decision-maker to adopt PHP 8, then take a moment to follow these tips before making your ask.

Gaining support from your management is invaluable and a PHP developer is best suited to perform this task. Compliance with the latest version makes developer life easier almost instantly, and can lead to countless benefits on the business end, not to mention greater compliance with the latest security standards

## Do Your Homework

In addition to citing the information in this eBook 🤔 it's always a good tactic to gather data on your current usage statistics. It's likely that you have some way to track how a web application behaves in certain scenarios, and identify which parts are underperforming.



## Be Ready to do a Cost–Benefit Analysis

If your boss or manager is more of a numbers person, present information that they can understand. Get specific in terms of profitability, business and potential loss. How might PHP 8 increase workplace productivity? What are the costs of not upgrading? Which risks can be mitigated by upgrading? These questions should be answered in the form of numbers. Quantifying your argument will have a stronger impact on a role that is focused on analytics, not developer jargon. Think about the short–term and long–term goals that can be achieved by performing a PHP 8 upgrade.



## Suggest Ways to Avoid Risk

Risk aversion is top of mind whenever a major upgrade is proposed. Outline or list the best courses of action to eliminate risk. If you're not sure of the risks involved, there are plenty of PHP experts to consult, like our [PHP Developer Team](#). After 16 years of closely following PHP upgrades and conducting dozens of migrations for our clients, we like to think we know a thing or two about successful PHP updates!



# 3

## The Upgrade Process



# The Upgrade Process


So you're ready to kick off your PHP upgrade process. Now what? Before starting any migration process to a new version, there are two steps that form a solid foundation before any code refactoring comes into play. The following preparations will lay the groundwork for a successful migration:

- ✓ **Deploy tools to catch errors or exception monitors**
- ✓ **Perform application tests**

These two actions form a solid foundation that should not be missed before performing any major application change, update, performance improvement, or rewrite.

## Deploy Error / Exception Monitoring

Many popular PHP frameworks such as Symfony and Laravel have built-in error and exception reporting modules. In addition to the convenience offered by saving error reports to files or databases, they also offer the ability (natively or via external package), to push these changes to specially prepared tools that help manage, view or solve reported problems.



Tools that can help in aggregation and problem solving include: Sentry, Rollbar, or CloudWatch – a tool built into the AWS infrastructure. Each one has official or community libraries that allow the application to be integrated with the tool, in either a regular PHP application or a website based on Symfony or Laravel.

These integrations allow you to catch application errors, including their exact origin code, and the reasons for their origin. Even with extensive tests on local developers' environments, if any error appears in the production application, it will be easy to spot and correct it very quickly. Such applications provide a number of possible methods and channels of notifications: from a simple email message, through messaging notifications (Slack, Teams), to SMS messages.

## Perform Application Tests

In an ideal situation, tests are written alongside software development. However, it is often known that brilliant ideas or innovative solutions in a fast-paced, dynamic market cannot wait. In these situations, the MVP of the application is typically built with manual tests and basic functional tests for the most sensitive elements of the application.

When preparing for a migration, writing tests are a worthwhile investment of time. Tests are a kind of programming documentation of business logic. They are invaluable at describing what needs to work, what data is to be returned, how to process it and/or what the results of given queries are to be. Thanks to such documentation, the programmer can see what was the idea behind the operation of a given module or part of the page.

If you decide to perform tests, we recommend three types of tests that will make life easier for the future of refactoring the application's codebase.



- **Functional Tests**

These are the so-called black box tests. In these kinds of tests, the programmer or tester writing them need not be aware of how functionalities are implemented in the system. They know what data the application should provide and what data it should return. Thanks to this, tests can be written in ways that actually check the functions and functioning of the application, but not how the data is processed, which can be helpful for third-party programmers, or protecting need-to-know proprietary secrets.

- **Unit Tests**

Unit tests are tests that validate one specific small part of the application. Let's say we have an ecommerce store. Our assumption is that the name of the products should not be repeated in the store, so we have the functionality to check whether the name of the item provided by the user already exists. The given functionality should return some form of response that there is no product with that name, or that it already exists. And it is precisely because of this small aspect of an application that is aptly named a unit test.

- **Performance**

These types of tests are exactly as the name states. Given a performance threshold, they test the application to ensure it exceeds the minimum limitations expected. Let's say we have a blog containing articles. A performance test may be written that assumes page load speeds should not exceed, for example, 400ms and not require more than 5MB of RAM. After refactoring and changes to the application, we can run performance tests to ensure that our previous criteria for performance have still been maintained.

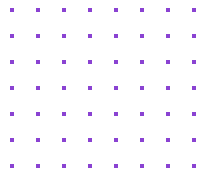
With these types of tests in our arsenal, developers can verifiably prove how the upgrade process changed a variety of factors, from version to version. The test results will show any losses or gains in performance, as well as catch any errors that may have arisen from changes in the codebase.

# I have PHP 5.x. What's next?

## Skip past PHP 6. Choose 7.x or 8.x

Version 6 of PHP was eventually scrapped, as many of its planned features were introduced in the releases of versions up to PHP 5.6. Soon afterwards, PHP 7 was released.

PHP 6 was scheduled to make major changes to Unicode support, improvements to the technology's object model, and more. Work on this version was prolonged so much that some of the functionalities planned for the sixth version were released as 5.3. Then it was realized that there was no point in releasing the remaining changes under the number "6" and it was decided that they would be included as a version of PHP 5.4. After that, the next versions, up to 5.6, were created, and then work on the seventh version was started.

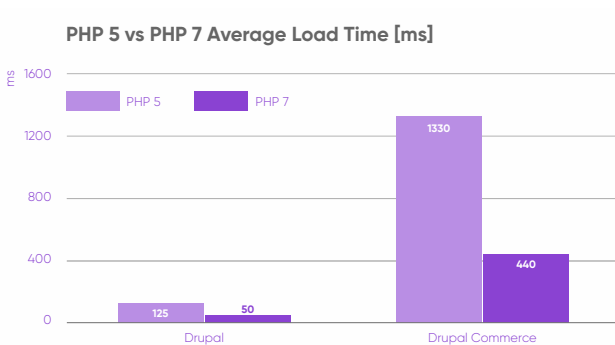


# Choosing Migration to PHP 7.x

## PHP 7.x: Safe and Performant

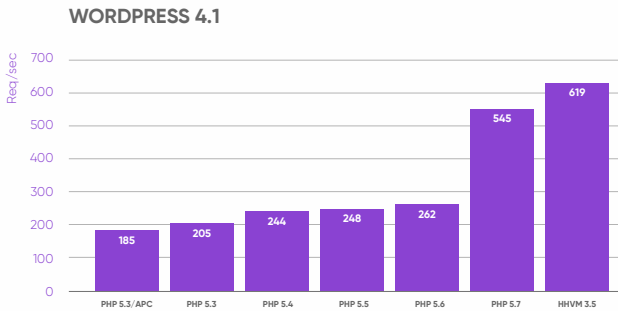
For any brand–new projects, we recommend jumping straight to PHP 8. However, many existing projects with large codebases, that already run in production, may simply find the value they need in upgrading from, say, PHP 5, to PHP 7. This is especially true for businesses which value stability over bleeding–edge features.

PHP version 7 introduced key changes in both the structured and object–oriented programming model. It improved performance, speed of script launch and many new amenities that help programmers. During the subsequent releases of 7.1 and 7.2, PHP had significantly improved the comfort of using the language, as well as the speed of processing scripts, which translated into higher speed of loading pages based on this technology. The latest version of PHP has also introduced a number of improvements in terms of application security and the security of the language itself.



PHP offers greater stability, language uniformity, security and more advanced object-oriented and typing mechanisms than its previous versions. These qualities directly translate into how stable, secure and reliable applications written in this language can be. Well-established businesses rely on these expectations and a strong application architecture, as they offer fewer errors, mitigate risks, and shorten time of software development.

Many tests performed by programmers or application owners have shown that their websites based on PHP 7 are faster than the same applications based on version 5 by up to 400%. The loading times on the language side were reduced by up to a few seconds. In today's world when milliseconds make or break user experiences, these performance gains make all the difference. Increasing performance also means that resources allocated to maintaining the infrastructure can be efficiently optimized, consuming less server resources alongside improving the end-user experience.



Apart from many new features, PHP 7 also removed some obsolete solutions and functionalities. As a result, the transition from PHP 5.6 to PHP 7 must be done with some caution and care in order to eliminate or replace the already unsupported aspects of the older release before carrying it out.

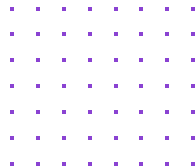
# How Migrating to PHP 7.x Works

The range of language functionalities that have changed and that should be taken into account when upgrading to version 7 are:

- handling exceptions and errors,
- support for variables, functions and indirect methods,
- changes in the functioning of functions ( e.g. `list()` ),
- automatic sorting of tables,
- foreach activity,
- and many others smaller topics.

The entire list of changes that may lead to incompatibility issues after upgrading to PHP 7 can be found here:

- [Backward incompatible changes—Manual](#)
- [Deprecated features in PHP 7.0.x—Manual](#)





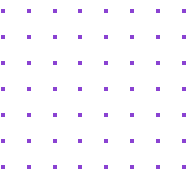
**Zero downtime deployment is a scenario** used for making the migration process run smoothly. One good way to do this involves:

- isolating a copy of the application to a new location (e.g. a folder on the same server),
- connecting all the necessary tools, such as database, queuing services or cache, to have a perfect representation of the production environment. Of course, these tools are to be attached as in production, but with test access, so that the tests on the application copy do not affect the working already version of the website,
- checking the compliance of installed libraries with the new version in theory,
- raising the language version of the application copy to the desired release. The latest version of PHP 7 is PHP 7.4,
- checking the compliance of installed libraries with the new version in practice,
- launching previously written automatic tests,
- analyzing the obtained results on the previously attached error / exception handler and repair.

**An application with well-written automatic tests** (unit tests, e2e tests or functional tests), will conveniently show you where it's necessary to make corrections or changes. Automated testing concepts also apply to libraries or frameworks used in the application. If the application does not have precise tests, the team will have to manually test all the functionalities offered by the system, increasing workload significantly. Even if the application has such tests, it will be a good idea to go through the scenarios of using it manually, which are critical for the application.

**After the application copy has been thoroughly analyzed**, relevant bugs have been fixed and the incompatible or outdated language functions have been changed, it is time to launch the latest version of the website. Zero downtime deployment in this case will consist of connecting access to tools, database, etc. to the copy of the production application, then performing the last manual tests on the application copy, and then switching the domain / server to the appropriate location containing the new version of the application.

**Of course, the process described here greatly simplifies the steps** and aspects that must be taken care of by the developer team approaching and struggling to upgrade the language version to a newer one. There are many differences, functions and decisions that can completely change the process described above. It all depends on the technologies, frameworks, packages, external tools, API and many other factors used.



# Choosing Migration to PHP 8.x

## PHP 8: Powerful, Secure, Cutting-Edge Features

For any new software projects, there's really no reason to avoid starting with PHP 8. The decision to migrate to PHP 8 from an existing application, however, is usually driven by the need for access to a ton of improved features, as well as speed, security and programming facilities that make the process of software development and maintenance much easier.

Another key advantage of PHP 8 is to reduce unnecessary code written by programmers by adding to the language the solutions they use in a simpler, more accessible version.

The most important list of changes that were introduced in PHP 8 include:

- **attributes,**
- **function and method argument names,**
- **unions,**
- **the safe null operator,**
- **the JIT compiler.**

You can read the entire list of changes in PHP 8.0: [New Features—Manual](#) including subsequent versions.



# How Migrating to PHP 8.x Works

The process of migrating from version 7.4 to 8.0 requires you to rethink the aspects that may make the code incompatible, as well as those that have been changed or abandoned in the new version. Key considerations to make include:

- comparing text to numbers—a very dangerous change that may not give any signs of error directly, but hide under logical conditions,
- word matching reservations,
- removing functions like `create_function` or `each`,
- changes in the interpretation of implicit grouping of activities by parentheses,
- new `pgsql` extension function names,
- changes in the `ZIP` extension,
- and a lot more!

These and other issues can be easily found on the page:

- [PHP 8.0: Backward Incompatible Changes—Manual](#)
- [PHP 8.0: Deprecated Features—Manual](#)

Apart from the differences of the changed or withdrawn functions, the migration process to the next version will be the same as for the previous version. There are no differences in the preparation or steps to be taken here.

# Ready to Upgrade?

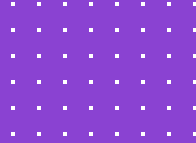
Hopefully we've convinced you that upgrading your PHP version can bring enormous benefit to your organization:

- ✓ greater application security,
- ✓ improved software stability,
- ✓ faster scripts and page loading,
- ✓ stable architecture which translates into fast software delivery,
- ✓ cutting-edge technologies to encourage and motivate the team.

The Polcode team has migrated our clients to newer versions of PHP over the last 16 years, enabling them to expand, optimize and improve the ever-changing codebase that powers their business.

Useful links to the PHP community and the largest frameworks:

- <https://laravelversions.com/en>
- <https://laravel.com>
- <https://symfony.com/releases>
- <https://www.php.net/supported-versions.php>
- <https://kinsta.com/blog/php-benchmarks>



# Let's talk!

We will audit your current site or apps and develop a plan for upgrading.

---

## Schedule a free consultation:

[sales@polcode.com](mailto:sales@polcode.com)

[polcode.com](http://polcode.com)

