Photo by Oscar Sutton on Unsplash

High performance PHP8 at Scale

Max Małecki

Poznań, PHPers Summit
27 May 2023

The Speed

Photo by Drew Stock on Unsplash

We all love

Photo by toine G on Unsplash

Performance

Photo by Jaxon Smith on Unsplash

We all want

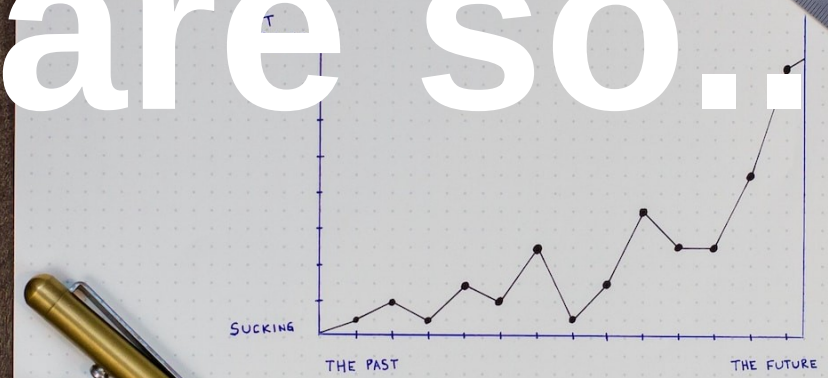
Photo by JOHN BEARBY IMAGES on Unsplash

Power

Photo by Bill Jelen on Unsplash

Source: https://opinion.al/video-sot-36-vjet-nga-vrasja-e-john-lennon/

Expectations are so...



High.

Photo by Dalton Smith on Unsplash

But results may

Photo by Dalton Smith on Unsplash

Vary

Photo by Racim Amr on Unsplash

Mediocore

Photo by David Armstrong on Unsplash

Harmfull

Photo by Conor Samuel on Unsplash

Greetings!

I’m Max

I’m from Poznań

Bitnoise_

For 17 years

As Senior Backend Engenieer

Since 2020

GitHub

emgiezet/
errbitPHP

Talked about
technical debt
and german cars.

Yet another

Boring
speaker
bullshit


Vic
@VicVijayakumar
1995: PHP is dead, learn ColdFusion
2002: PHP is dead, learn ASP.net
2003: PHP is dead, learn Django
2004: PHP is dead, learn Ruby on Rails
2010: PHP is dead, learn Flask
2011: PHP is dead, learn AngularJS
2016: PHP is dead, learn Next.js
2022: okay this is awkward
4:14 PM · Nov 1, 2022
2,822 Retweets
477 Quotes
19.9K Likes
1,122 Bookmarks


2023?
I’m Necromancer!

Desclimer:

I practise what I preach.
Example will be in Symfony 6

Sorry Tyler no Laravel in this presentation!

Today’s Agenda

0. We gonna talk about “the speed” theory;
1. How to find bottlenecks;
2. PHP Configuration tuning;
3. Every day coding good practices;
4. You will learn about benchmarks methodology;
5. We will watch some cool graphs
6. Apply performance boost in to your project.

Illustrations by Pixeltrue on icons8

The Speed

Photo by Drew Stock on Unsplash

Distance

traveled in given unit of

time

$$v = \frac{d}{t}$$

So how **fast** your application

should **respond**?

Below 1 second?

It is fast enough?

100 milisecond?

It is better?

Latency point of reference

```
L1 cache reference ......................... 0.5 ns
Branch mispredict ............................ 5 ns
L2 cache reference ........................... 7 ns
Mutex lock/unlock ........................... 25 ns
Main memory reference ...................... 100 ns
Compress 1K bytes with Zippy ............. 3,000 ns  =   3 µs
Send 2K bytes over 1 Gbps network ....... 20,000 ns  =  20 µs
SSD random read ........................ 150,000 ns  = 150 µs
Read 1 MB sequentially from memory ..... 250,000 ns  = 250 µs
Round trip within same datacenter ...... 500,000 ns  = 0.5 ms
Read 1 MB sequentially from SSD* ..... 1,000,000 ns  =   1 ms
Disk seek ........................... 10,000,000 ns  =  10 ms
Read 1 MB sequentially from disk .... 20,000,000 ns  =  20 ms
Send packet CA->Netherlands->CA .... 150,000,000 ns  = 150 ms
```

from https://gist.github.com/hellerbarde/2843375

What the speed really mean?

In web request context

Requests
per
second

$$v = \frac{r}{t}$$

That gets a **response**
with **HTTP_CODE**

2**

And **Meaningfull** content.

Ofcourse.

What the speed really mean?

in a consumer / worker context

Messages

processed per

second

Jobs

finished per

second

How to measure the speed?

Benchmarking

Photo by Fotis Fotopoulos on Unsplash

Just like a unit tests benchmarks should be organized in suites.

Can you give us a example of totally useless benchmark?


⚡Favor⚡ @heyOnuoha · 4h
So PHP is faster than Python 😂😂😂
Time taken Count to 1 billion
Language
Time taken (in seconds)
Golang
2.5s
C#
3.05s
C++
3.49s
PHP
23.51s
Python
159s
145
103
661
67.6K

PHPBench

PHPBench is a benchmark runner for PHP analogous to PHPUnit but for performance rather than correctness.

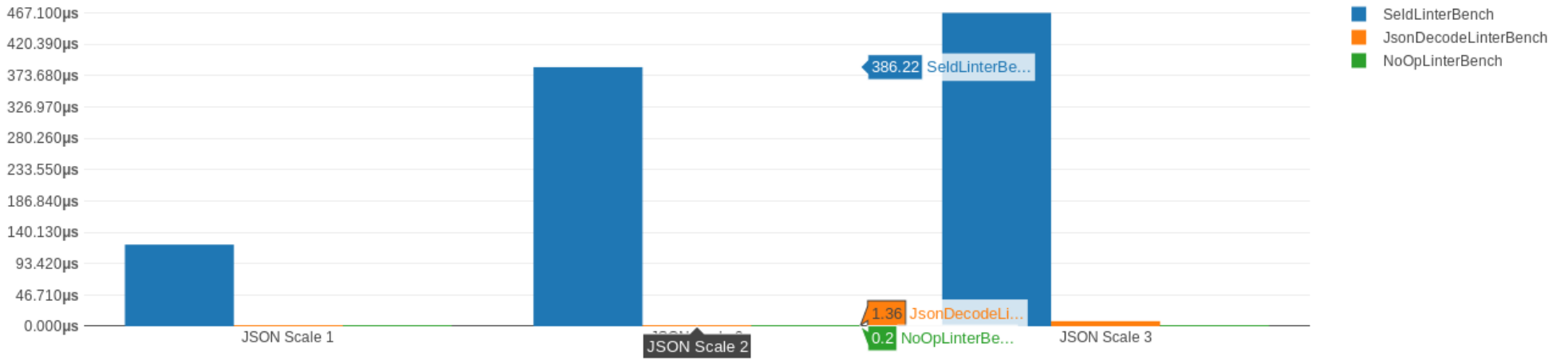
./vendor/bin/phpbench run tests/Benchmark --report=default

```
PHPBench (1.2.10) running benchmarks... #standwithukraine
with configuration file: /home/mgz/workspace/phpers/summit2023/8.1 Bench/phpbench.json
with PHP version 8.0.28, xdebug   , opcache



\emgiezet\Tests\Benchmark\TimeConsumerBench



    benchConsume.............................I4 - Mo152.444µs (±0.09%)



Subjects: 1, Assertions: 0, Failures: 0
+------+-------------------+--------------+-----+------+----------+-----------+--------------+----------------+
| iter | benchmark         | subject      | set | revs | mem_peak | time_avg  | comp_z_value | comp_deviation |
+------+-------------------+--------------+-----+------+----------+-----------+--------------+----------------+
| 0    | TimeConsumerBench | benchConsume |     | 1000 | 704,944b | 152.652µs | +0.92σ       | +0.08%         |
| 1    | TimeConsumerBench | benchConsume |     | 1000 | 704,944b | 152.713µs | +1.38σ       | +0.12%         |
| 2    | TimeConsumerBench | benchConsume |     | 1000 | 704,944b | 152.390µs | -1.05σ       | -0.09%         |
| 3    | TimeConsumerBench | benchConsume |     | 1000 | 704,944b | 152.390µs | -1.05σ       | -0.09%         |
| 4    | TimeConsumerBench | benchConsume |     | 1000 | 704,944b | 152.503µs | -0.20σ       | -0.02%         |
+------+-------------------+--------------+-----+------+----------+-----------+--------------+----------------+
```

With nice customizable HTML reports

Linter comparison

Compares different implementations of the PHPBench config linter



SeldLinterBench | JsonDecodeLinterBench | NoOpLinterBench

variant	memory	min	max	mode	rstdev	stdev
JSON Scale 1	1.593mb	0.680µs	0.680µs	0.680µs	± 0.00%	0.000µs
JSON Scale 2	1.593mb	1.360µs	1.360µs	1.360µs	± 0.00%	0.000µs
JSON Scale 3	1.593mb	7.600µs	7.600µs	7.600µs	± 0.00%	0.000µs

JsonDecodeLinterBench

What is limiting my application speed?

Bottlenecks

Photo by Andrew Seaman on Unsplash

Bottleneck

A bottleneck is a phenomenon where the performance and/or capacity of an entire system is limited by a single or limited number of components or resources.

Examples of Bottlenecks

Photo by John Cafazza on Unsplash

Populate 5 million items into ElasticSearch Index.

Cronjob that start to overlap the time frame of next execution

Elastica update listener on entity change in Doctrine make your insert / updates slow

You need to process 15GB XML import overnight

You need to serialize big fat entity with lots of relations

You want to host mongodb with db bigger than you available RAM

Types of Bottlenecks

Photo by Dylan de Jonge on Unsplash

Hardware

Photo by Dylan de Jonge on Unsplash

CPU

CPU overload

CPU Context switches

CPU IO waits

Memory

Out of Memory and go swap

Disk

Disk: Local disk access latency

Disk: Random disk I/O -> disk seeks

Disk: Disk fragmentation

Disk: SSDs performance drop once near full capacity

Infrastructure

Photo by Dylan de Jonge on Unsplash

Network

Long resolving DNS Lookups
Dropped packets
Faulty hardware

Virtualisation

Bandwidth to and from the cloud provider
Sharing a HDD, disk seek death
Network I/O fluctuations in the cloud
Enabling high-availability without accounting for failover

Database

This slide can be separate presentation about database. Most popular:

1)Deadlocks,
2)Large joins taking up memory,
3)Long & short running queries

OS

Fsync flushing, linux buffer cache filling up,
File descriptor limits

Application

Photo by Dylan de Jonge on Unsplash

Algorithm complexity

Threads

Deadlocks

No scalability


WAIT. THERE'S
MORE BOTTLENECKS!
IT
HAS ALWAYS
BEEN
imgflip.com

How to catch them all?

Load Testing

Photo by Andrew Seaman on Unsplash

The goal is to check the performance of your system under a heavy load.

To push your system as far as it’s possible and check when it stop responding.


Your containers will
love it.
OVHcloud
Photo: Sekurak.pl

Tools

Photo by Lachlan Donald on Unsplash


siege

```
siege -c 100 -r 100 -b
<http://localhost/myapp/>
```

This will run 100 concurent sessions with 100 request each in benchmark mode, so with no delays between requests.

```
  1[|||||||||59.9%]   5[|||||||||65.2%]     9[|||||||||69.7%]  13[|||||||||60.6%]
  2[|||||||  42.0%]   6[|||||||||58.2%]    10[|||||||||62.3%]  14[|||||||||63.0%]
  3[|||||||||59.6%]   7[|||||||||55.6%]    11[|||||||||66.5%]  15[|||||||||64.8%]
  4[|||||||||56.8%]   8[|||||||||54.4%]    12[|||||||||60.2%]  16[|||||||||60.7%]
Mem[|||||||||||||||||||||||18.0G/31.3G]   Tasks: 493, 3572 thr; 9 running
Swp[|                          29.5M/33.5G]   Load average: 8.34 5.39 3.11
                                              Uptime: 05:18:23

    PID USER      PRI  NI  VIRT   RES   SHR S CPU%▽MEM%   TIME+  Command
  25813 systemd-c  20   0 4324M  426M 35988 S 163.  1.3  6:07.64 mysqld --default-authenticat
 347188 82         20   0 52008 30612  7728 S 71.4  0.1  2:05.44 php-fpm: pool www
 344978 82         20   0 52608 31212  7724 R 71.4  0.1  2:13.81 php-fpm: pool www
 347204 82         20   0 51976 30560  7724 R 70.2  0.1  2:04.72 php-fpm: pool www
 350520 82         20   0 45312 23888  7724 R 70.2  0.1  0:30.76 php-fpm: pool www
 350536 82         20   0 45248 23824  7728 R 72.0  0.1  0:30.16 php-fpm: pool www
 350535 82         20   0 42164 22368  7464 S 67.1  0.1  0:28.87 php-fpm: pool www
 348959 82         20   0 45396 25588  7460 R 67.7  0.1  1:04.39 php-fpm: pool www
 349005 82         20   0 45100 25292  7460 S 68.3  0.1  1:02.29 php-fpm: pool www
 350519 82         20   0 42360 22500  7460 R 68.9  0.1  0:30.06 php-fpm: pool www
 348822 82         20   0 45808 25964  7464 R 68.9  0.1  1:10.67 php-fpm: pool www
 344024 systemd-c  20   0 4324M  426M 35988 S 14.9  1.3  0:35.13 mysqld --default-authenticat
 187427 systemd-c  20   0 4324M  426M 35988 S 18.6  1.3  0:35.48 mysqld --default-authenticat
 344099 systemd-c  20   0 4324M  426M 35988 S 16.8  1.3  0:34.47 mysqld --default-authenticat
 187425 systemd-c  20   0 4324M  426M 35988 S 19.2  1.3  0:36.14 mysqld --default-authenticat
 187428 systemd-c  20   0 4324M  426M 35988 S 16.1  1.3  0:35.69 mysqld --default-authenticat
 187433 systemd-c  20   0 4324M  426M 35988 R 19.2  1.3  0:36.42 mysqld --default-authenticat
  31553 systemd-c  20   0 4324M  426M 35988 S 23.0  1.3  0:36.51 mysqld --default-authenticat
 187426 systemd-c  20   0 4324M  426M 35988 S 17.4  1.3  0:36.46 mysqld --default-authenticat
 187430 systemd-c  20   0 4324M  426M 35988 S 17.4  1.3  0:35.83 mysqld --default-authenticat
   3578 mgz        20   0 5885M  455M  180M S 13.7  1.4 26:51.34 /usr/bin/gnome-shell
  25385 root       20   0  172M 67236 38152 S 11.2  0.2  0:29.08 traefik traefik --web --dock
   2891 root       20   0 3083M  107M 41200 S 10.6  0.3  0:43.12 /usr/bin/dockerd -H fd:// --
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

```
{
  "transactions":                 10000,
  "availability":                 100.00,
  "elapsed_time":                 134.82,
  "data_transferred":              77.36,
  "response_time":                 1.34,
  "transaction_rate":              74.17,
  "throughput":                   0.57,
  "concurrency":                  99.57,
  "successful_transactions":      10000,
  "failed_transactions":          0,
  "longest_transaction":          1.73,
  "shortest_transaction":         0.09
}
```

Apache Bench

Apache Benchmark

ab -n 100 -c 100 http://localhost/en/blog/posts/aliquam-sodales-odio-id-eleifend-tristique

```
Benchmarking localhost (be patient).....done


Server Software:        nginx/1.17.8
Server Hostname:        localhost
Server Port:            80

Document Path:
/en/blog/posts/aliquam-sodales-odio-id-
eleifend-tristique
Document Length:        31309 bytes
```

```
Concurrency Level:      100
Time taken for tests:   0.271 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      3186100 bytes
HTML transferred:       3130900 bytes
Requests per second:    369.54 [#/sec] (mean)
Time per request:       270.604 [ms] (mean)
Time per request:       2.706 [ms] (mean,
across all concurrent requests)
Transfer rate:          11498.08 [Kbytes/sec]
received
```

Connection Times (ms)

	min	mean[+/-sd]	median	max
Connect:	0	1 0.2	1	2
Processing:	9	129 73.3	130	259
Waiting:	9	129 73.3	129	259
Total:	10	130 73.1	131	260

```
Percentage of the requests served within a
certain time (ms)
  50%    131
  66%    171
  75%    191
  80%    205
  90%    230
  95%    249
  98%    260
  99%    260
 100%    260 (longest request)
```

Gatling

https://github.com/gatling/gatling

- Written mostly in Scala
- Support load testing scenarios in scala/java/kotlin
- Have scenario recorder
- Needs own server to unleash its full capabilities
- Available also as paid SaaS: gatling.io

How to measure changes in my code performance?

Put benchmarking as additional CI step

Load testing is an overkill to put it in CI.

How to find a bottlenecks in my application?

Profiling

Photo by Fotis Fotopoulos on Unsplash

What actually profiling is?

Profiling checking the entire callstack for given part of code.
Each function is listed with: execution time and amount of calls.

XHProf

Installation

*Example can be found in the presentation repository

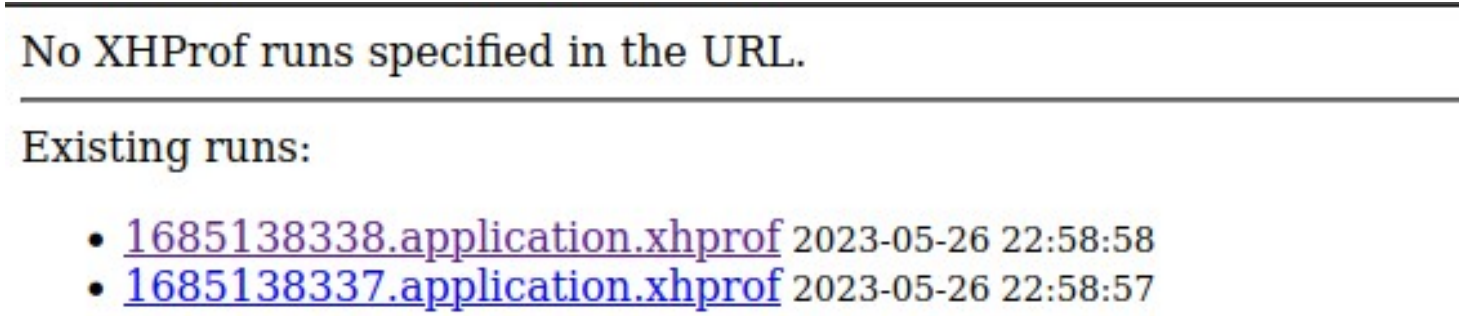
Add it to your Dockerfile

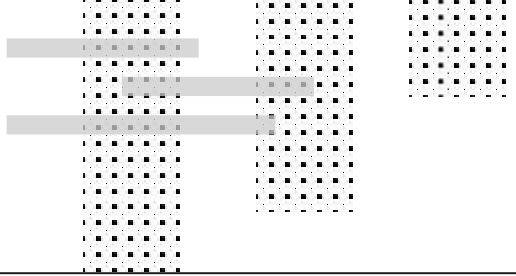
```
RUN curl "http://pecl.php.net/get/xhprof-2.3.2.tgz" -fsL -o ./xhprof-
2.3.2.tgz && \
    mkdir /var/xhprof && tar xf ./xhprof-2.3.2.tgz -C /var/xhprof && \
    cd /var/xhprof/xhprof-2.3.2/extension && \
    phpize && \
    ./configure && \
    make && \
    make install
# custom settings for xhprof
COPY ./xhprof.ini /usr/local/etc/php/conf.d/xhprof.ini
RUN docker-php-ext-enable xhprof
#folder for xhprof profiles (same as in file xhprof.ini)
RUN mkdir -m 777 /profiles
```

How to use it

```
#[Route('/', name: 'blog_index', defaults: ['page' => '1', '_format' => 'html'], methods: ['GET'])]
#[Route('/rss.xml', name: 'blog_rss', defaults: ['page' => '1', '_format' => 'xml'], methods:
['GET'])]
#[Route('/page/{page<[1-9]\d{0,8}>}', name: 'blog_index_paginated', defaults: ['_format' =>
'html'], methods: ['GET'])]
#[Cache(smaxage: 10)]
public function index(Request $request, int $page, string $_format, PostRepository $posts,
TagRepository $tags): Response
{
    xhprof_enable(XHPROF_FLAGS_MEMORY + XHPROF_FLAGS_CPU);
    $tag = null;
    if ($request->query->has('tag')) {
        $tag = $tags->findOneBy(['name' => $request->query->get('tag')]);
    }
    $latestPosts = $posts->findLatest($page, $tag);
    file_put_contents('/profiles/'.time().'.application.xhprof', serialize(xhprof_disable()));
    // Every template name also has two extensions that specify the format and
    // engine for that template.
    // See https://symfony.com/doc/current/templates.html#template-naming
    return $this->render('blog/index.'.$_format.'.twig', [
        'paginator' => $latestPosts,
        'tagName' => $tag?->getName(),
    ]);
}
```

Open xhprof web





Select the run

Run Report

Run #1685138338: XHProf Run (Namespace=application)

Tip

Click a function name below to drill down.

View Top Level Run Report

Overall Summary

Total Incl. Wall Time (microsec): 11,803 microsecs
Total Incl. CPU (microsecs): 12,546 microsecs
Total Incl. MemUse (bytes): 1,217,536 bytes
Total Incl. PeakMemUse (bytes): 1,308,864 bytes
Number of Function Calls: 13,468

[View Full Callgraph]

Displaying top 100 functions: Sorted by Incl. Wall Time (microsec) [display all]

Function Name	Calls	Calls%	Incl. Wall Time (microsec)	IWall%	Excl. Wall Time (microsec)	EWall%	Incl. CPU (microsecs)	ICpu%	Excl. CPU (microsec)	ECPU%	Incl. MemUse (bytes)	IMemUse%	Excl. MemUse (bytes)	EMemUse%	Incl. PeakMemUse (bytes)	IPeakMemU
main()	1	0.0%	11,803	100.0%	12	0.1%	12,546	100.0%	0	0.0%	1,217,536	100.0%	824	0.1%	1,308,864	10
App\Repository\PostRepository::findLatest	1	0.0%	11,789	99.9%	11	0.1%	12,546	100.0%	0	0.0%	1,214,792	99.8%	440	0.0%	1,308,864	10
App\Pagination\Paginator::paginate	1	0.0%	11,362	96.3%	12	0.1%	10,358	82.6%	0	0.0%	1,068,824	87.8%	-12,944	-1.1%	1,164,600	8
Doctrine\ORM\Tools\Pagination \Paginator::getIterator	1	0.0%	9,127	77.3%	25	0.2%	7,058	56.3%	0	0.0%	1,017,544	83.6%	-14,336	-1.2%	1,139,752	8
Doctrine\ORM\AbstractQuery::execute	3	0.0%	9,039	76.6%	4	0.0%	10,358	82.6%	0	0.0%	696,624	57.2%	-9,104	-0.7%	792,280	6
Doctrine\ORM \AbstractQuery::executeIgnoreQueryCache	3	0.0%	9,035	76.5%	21	0.2%	10,358	82.6%	0	0.0%	705,728	58.0%	-3,272	-0.3%	792,280	6
Doctrine\ORM\Query::_doExecute	3	0.0%	6,197	52.5%	31	0.3%	9,083	72.4%	0	0.0%	662,584	54.4%	-42,392	-3.5%	605,880	4
Doctrine\ORM\AbstractQuery::getResult	1	0.0%	5,054	42.8%	1	0.0%	3,588	28.6%	0	0.0%	299,328	24.6%	664	0.1%	419,768	3
Doctrine\ORM\AbstractQuery::getScalarResult	2	0.0%	3,990	33.8%	4	0.0%	6,770	54.0%	0	0.0%	399,224	32.8%	1,264	0.1%	372,512	2
Doctrine\ORM\Query::parse	3	0.0%	3,894	33.0%	40	0.3%	0	0.0%	0	0.0%	327,408	26.9%	-20,640	-1.7%	290,392	2
Doctrine\ORM\Query\Parser::parse	5	0.0%	3,824	32.4%	44	0.4%	0	0.0%	0	0.0%	488,672	40.1%	3,776	0.3%	433,408	3

Select function

Run Report

Run #1685138338: XHProf Run (Namespace=application)

Tip

Click a function name below to drill down.

View Top Level Run Report

Parent/Child report for App\Repository\PostRepository::findLatest [View Callgraph]

Function Name	Calls	Calls%	Incl. Wall Time (microsec)	IWall%	Incl. CPU (microsecs)	ICpu%	Incl. MemUse (bytes)	IMemUse%	Incl. PeakMemUse (bytes)	IPeakMemUse%
Current Function										
App\Repository\PostRepository::findLatest	1	6.2%	11,789	99.9%	12,546	100.0%	1,214,792	99.8%	1,308,864	100.0%
Exclusive Metrics for Current Function			11	0.1%	0	0.0%	440	0.0%	704	0.1%
Parent function										
main()	1	100.0%	11,789	100.0%	12,546	100.0%	1,214,792	100.0%	1,308,864	100.0%
Child functions										
App\Pagination\Paginator::paginate	1	9.1%	11,362	96.4%	10,358	82.6%	1,068,824	88.0%	1,164,600	89.0%
Doctrine\ORM\EntityRepository::createQueryBuilder	1	9.1%	340	2.9%	2,188	17.4%	105,112	8.7%	103,408	7.9%
Doctrine\ORM\QueryBuilder::setParameter	1	9.1%	25	0.2%	0	0.0%	11,568	1.0%	11,568	0.9%
Doctrine\ORM\QueryBuilder::innerJoin	1	9.1%	16	0.1%	0	0.0%	12,968	1.1%	14,800	1.1%
Doctrine\ORM\QueryBuilder::where	1	9.1%	10	0.1%	0	0.0%	3,080	0.3%	3,080	0.2%
Doctrine\ORM\QueryBuilder::leftJoin	1	9.1%	8	0.1%	0	0.0%	3,904	0.3%	1,808	0.1%
Doctrine\ORM\QueryBuilder::orderBy	1	9.1%	7	0.1%	0	0.0%	4,088	0.3%	4,088	0.3%
Doctrine\ORM\QueryBuilder::addSelect	1	9.1%	5	0.0%	0	0.0%	2,808	0.2%	2,808	0.2%
DateTime::__construct	1	9.1%	2	0.0%	0	0.0%	832	0.1%	832	0.1%
Composer\Autoload\ClassLoader::loadClass	1	9.1%	2	0.0%	0	0.0%	584	0.0%	584	0.0%
App\Pagination\Paginator::__construct	1	9.1%	1	0.0%	0	0.0%	584	0.0%	584	0.0%

Pros

- Can be used with production settings.
- Can profile only couple lines of code.

Tip of the day.

- If you like your SSD life span turn off profiling while load testing ;)

Other options?

xdebug

Symfony Profiler

Symfony Profiler

search on symfony.com

GET http://localhost/en/blog/page/2

Response: 200 OK | Browse referrer URL | IP: 172.19.0.1 | Profiled on: Fri, 26 May 2023 14:05:17 +0100 | Token: 22a10e

Last 10 | Latest | Search

- Request / Response
- Performance
- Validator
- Forms
- Exception
- Logs
- Events
- Routing
- Cache
- Translation
- Security
- Twig
- HTTP Client
- Doctrine
- Debug
- E-mails
- Serializer
- Configuration

Profiler settings

Performance metrics

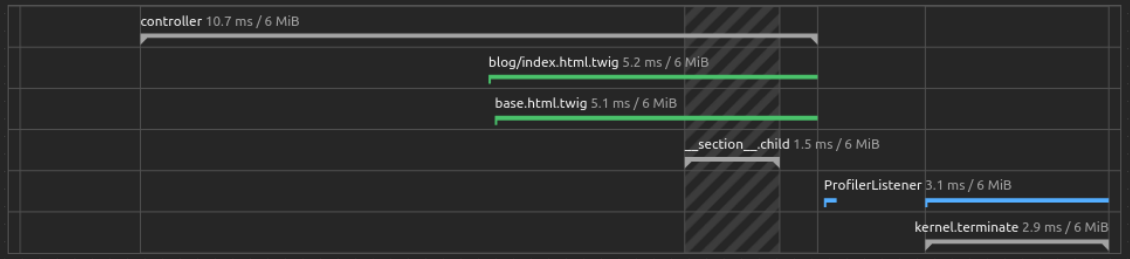
Total execution time	Symfony initialization	Peak memory usage	Sub-Request	Sub-Request time
19 ms	2 ms	6.00 MiB	1	1.5 ms

Execution timeline

Threshold 1 ms (timeline only displays events with a duration longer than this threshold)

Main Request 16.3 ms

default | section | event_listener | template | controller.argument_value_resolver



Note: sections with a striped background correspond to sub-requests.

Sub-requests (1)

Symfony\Bundle\FrameworkBundle\Controller\TemplateController::templateAction() 1.5 ms

default | section | event_listener | template | controller.argument_value_resolver

Symfony Profiler

search on symfony.com

GET http://localhost/en/blog/page/2

Response: 200 OK | Browse referrer URL | IP: 172.19.0.1 | Profiled on: Fri, 26 May 2023 14:05:17 +0100 | Token: 22a10e

Last 10 | Latest | Search

- Request / Response
- Performance
- Validator
- Forms
- Exception
- Logs
- Events
- Routing
- Cache
- Translation
- Security
- Twig
- HTTP Client
- Doctrine
- Debug
- E-mails
- Serializer
- Configuration

Profiler settings

Query Metrics

Database Queries	Different statements	Query time	Invalid entities
3	3	1.30 ms	0

Queries | Database Connections | Entity Managers | Second Level Cache | Entities Mapping

Group similar statements

#	Time	Info
1	0.57 ms	`SELECT DISTINCT s0_.id AS id_0, s0_.published_at AS published_at_1 FROM symfony_demo_post s0_ INNER JOIN symfony_demo_user s1_ ON s0_.author_id = s1_.id LEFT JOIN symfony_demo_post_tag s3_ ON s0_.id = s3_.post_id LEFT JOIN symfony_demo_tag s2_ ON s2_.id = s3_.tag_id WHERE s0_.published_at <= ? ORDER BY s0_.published_at DESC LIMIT 10 OFFSET 10` **Parameters:** [▾ "2023-05-26 14:05:17"] View formatted query | View runnable query | Explain query
2	0.42 ms	`SELECT s0_.id AS id_0, s0_.title AS title_1, s0_.slug AS slug_2, s0_.summary AS summary_3, s0_.content AS content_4, s0_.published_at AS published_at_5, s1_.id AS id_6, s1_.full_name AS full_name_7, s1_.username AS username_8, s1_.email AS email_9, s1_.password AS password_10, s1_.roles AS roles_11, s2_.id AS id_12, s2_.name AS name_13, s0_.author_id AS author_id_14 FROM symfony_demo_post s0_ INNER JOIN symfony_demo_user s1_ ON s0_.author_id = s1_.id LEFT JOIN symfony_demo_post_tag s3_ ON s0_.id = s3_.post_id LEFT JOIN symfony_demo_tag s2_ ON s2_.id = s3_.tag_id WHERE s0_.published_at <= ? AND s0_.id IN (?, ?, ?, ?, ?, ?, ?, ?, ?, ?) ORDER BY s0_.published_at DESC, s2_.name ASC` **Parameters:** [▾ "2023-05-26 14:05:17" 11 12 13 14 15 16 17

How to improve app response speed?

Caching

Photo by Fotis Fotopoulos on Unsplash

How the load will look like when app hasn’t any cache?

```
  1[|||||||||59.9%]   5[|||||||||65.2%]     9[|||||||||69.7%]  13[|||||||||60.6%]
  2[|||||||  42.0%]   6[|||||||||58.2%]    10[|||||||||62.3%]  14[|||||||||63.0%]
  3[|||||||||59.6%]   7[|||||||||55.6%]    11[|||||||||66.5%]  15[|||||||||64.8%]
  4[|||||||||56.8%]   8[|||||||||54.4%]    12[|||||||||60.2%]  16[|||||||||60.7%]
Mem[|||||||||||||||||||||||18.0G/31.3G]   Tasks: 493, 3572 thr; 9 running
Swp[|                          29.5M/33.5G]   Load average: 8.34 5.39 3.11
                                              Uptime: 05:18:23

    PID USER      PRI  NI  VIRT   RES   SHR S CPU%▽MEM%   TIME+  Command
  25813 systemd-c  20   0 4324M  426M 35988 S 163.  1.3  6:07.64 mysqld --default-authenticat
 347188 82         20   0 52008 30612  7728 S 71.4  0.1  2:05.44 php-fpm: pool www
 344978 82         20   0 52608 31212  7724 R 71.4  0.1  2:13.81 php-fpm: pool www
 347204 82         20   0 51976 30560  7724 R 70.2  0.1  2:04.72 php-fpm: pool www
 350520 82         20   0 45312 23888  7724 R 70.2  0.1  0:30.76 php-fpm: pool www
 350536 82         20   0 45248 23824  7728 R 72.0  0.1  0:30.16 php-fpm: pool www
 350535 82         20   0 42164 22368  7464 S 67.1  0.1  0:28.87 php-fpm: pool www
 348959 82         20   0 45396 25588  7460 R 67.7  0.1  1:04.39 php-fpm: pool www
 349005 82         20   0 45100 25292  7460 S 68.3  0.1  1:02.29 php-fpm: pool www
 350519 82         20   0 42360 22500  7460 R 68.9  0.1  0:30.06 php-fpm: pool www
 348822 82         20   0 45808 25964  7464 R 68.9  0.1  1:10.67 php-fpm: pool www
 344024 systemd-c  20   0 4324M  426M 35988 S 14.9  1.3  0:35.13 mysqld --default-authenticat
 187427 systemd-c  20   0 4324M  426M 35988 S 18.6  1.3  0:35.48 mysqld --default-authenticat
 344099 systemd-c  20   0 4324M  426M 35988 S 16.8  1.3  0:34.47 mysqld --default-authenticat
 187425 systemd-c  20   0 4324M  426M 35988 S 19.2  1.3  0:36.14 mysqld --default-authenticat
 187428 systemd-c  20   0 4324M  426M 35988 S 16.1  1.3  0:35.69 mysqld --default-authenticat
 187433 systemd-c  20   0 4324M  426M 35988 R 19.2  1.3  0:36.42 mysqld --default-authenticat
  31553 systemd-c  20   0 4324M  426M 35988 S 23.0  1.3  0:36.51 mysqld --default-authenticat
 187426 systemd-c  20   0 4324M  426M 35988 S 17.4  1.3  0:36.46 mysqld --default-authenticat
 187430 systemd-c  20   0 4324M  426M 35988 S 17.4  1.3  0:35.83 mysqld --default-authenticat
   3578 mgz        20   0 5885M  455M  180M S 13.7  1.4 26:51.34 /usr/bin/gnome-shell
  25385 root       20   0  172M 67236 38152 S 11.2  0.2  0:29.08 traefik traefik --web --dock
   2891 root       20   0 3083M  107M 41200 S 10.6  0.3  0:43.12 /usr/bin/dockerd -H fd:// --
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

2 Goals:

1. Response cache: should prevent duplicated requests to reach the php-fpm.

2. Result cache:

should prevent cached queries to reach the database.

If the cache entry has

valid TTL.

Cache keys

should have

semantic naming

Because

There are only two hard things in Computer Science: cache invalidation and naming things.
- Phil Karlton

Check your headers!

`Cache-Control: max-age=604800, must-revalidate`

REST API GET endpoint responses can be cached

Database is last thing that should be called.

Prioritize endpoints caching by load they generate.

Photo by Kelly Sikkema on Unsplash

It’s time for the main course

Photo by Kyle Mackie on Unsplash

or for the

tofu course

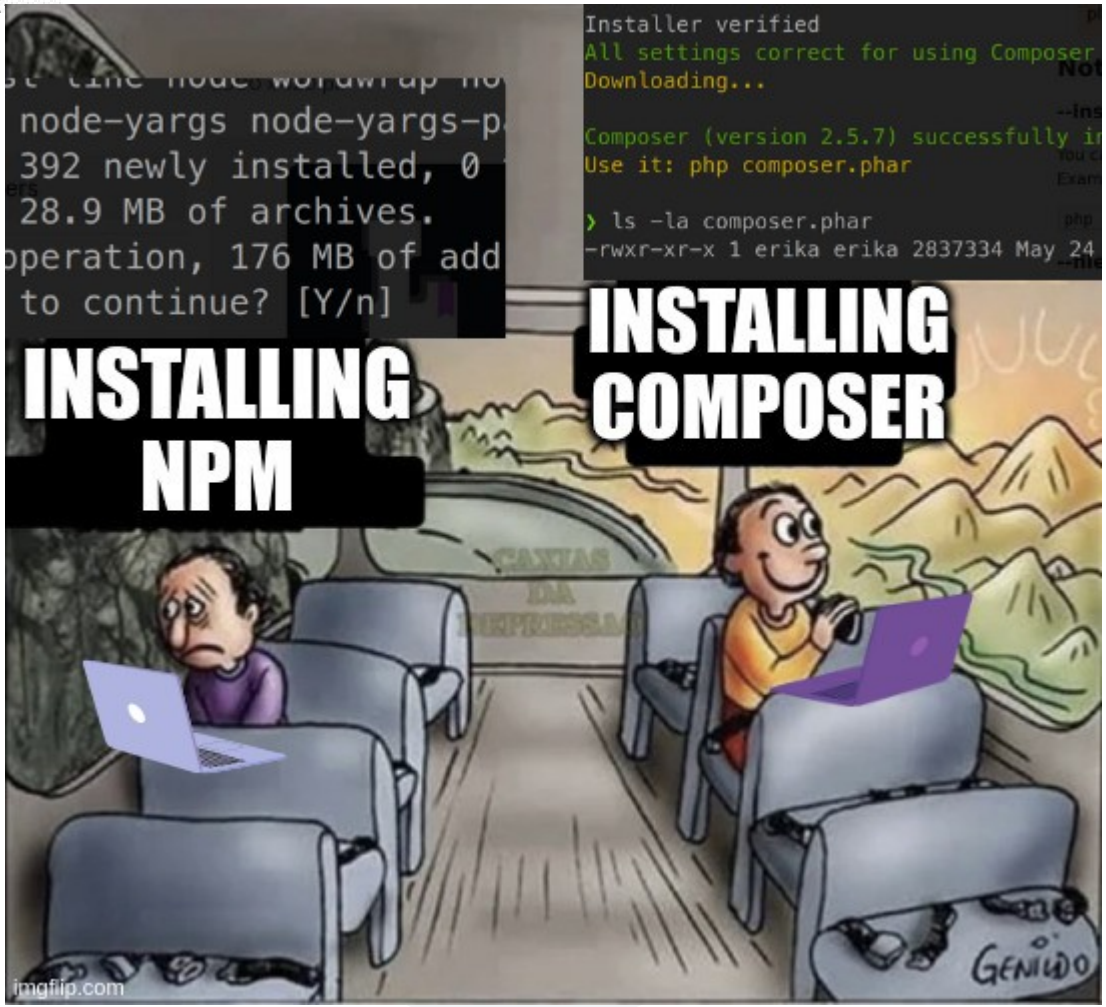
To don’t offend anybody today.

Photo by shri on Unsplash

PHP Configuration

```
memory_limit=1024M

opcache.enable=1
opcache.revalidate_freq=10
opcache.validate_timestamps=1
opcache.max_accelerated_files=10000
opcache.memory_consumption=192
opcache.max_wasted_percentage=10
opcache.interned_strings_buffer=1
opcache.fast_shutdown=1

opcache.jit_buffer_size=256M
opcache.jit=1255
opcache.jit=tracing

upload_max_filesize = 20M
post_max_size = 20M


pm = ondemand
pm.max_children = 500
pm.max_requests = 500
```

Composer


node-yargs node-yargs-p
392 newly installed, 0
28.9 MB of archives.
operation, 176 MB of add
to continue? [Y/n]
INSTALLING NPM
Installer verified
All settings correct for using Composer
Downloading...
Composer (version 2.5.7) successfully in
Use it: php composer.phar
❯ ls -la composer.phar
-rwxr-xr-x 1 erika erika 2837334 May 24
INSTALLING COMPOSER
imgflip.com

```
/usr/bin/composer install \
        --no-progress \
        --no-interaction \
        --no-dev \
        --optimize-autoloader \
        --prefer-dist
```

```
composer dump-autoload --classmap-authoritative
```

Or put this to APCu

```
composer dump-autoload --apcu
```

OPCache

Installation

```
apt-get install php-opcache
```

Configuration

```
#/etc/php8.1/fpm/php.ini

opcache.enable=1
opcache.memory_consumption=128
opcache.max_accelerated_files=10000
opcache.revalidate_freq=200
```

Restart php-fpm

Add some extra preloading on fpm startup

```
composer require
practically/preloader
```

practically/preloader hooks after install and generates preload.php executable file.

update php.ini

```
;/etc/php8.1/fpm/php.ini
opcache.enable=1
opcache.enable_cli=1
opcache.memory_consumption=128
opcache.max_accelerated_files=10000
opcache.revalidate_freq=200
opcache.preload=/path/to/preload.php
opcache.preload_user=user
```

JIT Compiler

```
opcache.jit_buffer_size=256M
opcache.jit=1255
opcache.jit=tracing
```

JIT Modes

- opcache.jit = 1205 - all code is JIT compiled
- opcache.jit = 1235 - only selected code portions (based on their relative use) are passed to the JIT compilation
- opcache.jit = 1255 - application code is tracked for compilation by JIT and selected parts of the code are transferred to the compiler

Every day coding good practices

Photo by Fotis Fotopoulos on Unsplash

Use isset() to check that array is initialized

\engiezet\Tests\Benchmark\IssetBench

    benchIsset.....................................I0 - Mo0.012µs (±0.00%)
    benchSizeof....................................I0 - Mo0.021µs (±0.00%)
    benchCount.....................................I0 - Mo0.021µs (±0.00%)

Stick with single quotes

```
\emgiezet\Tests\Benchmark\QuotesBench

    benchSingle.............................I0 - Mo0.445µs (±0.00%)
    benchDouble.............................I0 - Mo2.498µs (±0.00%)
```

RegExp’s are sexy but you need to pay

```
\emgiezet\Tests\Benchmark\PregBench

    benchPregReplace.........................I0 - Mo3.906µs (±0.00%)
    benchStrReplace..........................I0 - Mo2.832µs (±0.00%)
```

Try identical operator instead of equals

\emgiezet\Tests\Benchmark\EqualsBench

    benchEqual.....................................I0 - Mo0.305µs (±0.00%)
    benchIdentical.................................I0 - Mo0.301µs (±0.00%)

When you need precision math use BCMath but be aware that it costs double

```
\emgiezet\Tests\Benchmark\MathBench

    benchBcmul.....................................I0 - Mo0.492µs (±0.00%)
    benchMultiply..................................I0 - Mo0.258µs (±0.00%)
```

Forget about other loops than foreach()

\engiezet\Tests\Benchmark\LoopsBench

    benchFor.....................................I0 - Mo7.168ms (±0.00%)
    benchForeach.................................I0 - Mo6.791ms (±0.00%)
    benchWhile...................................I0 - Mo7.153ms (±0.00%)
    benchDoWhile.................................I0 - Mo7.056ms (±0.00%)

If you want more benchmarks like those go:

https://phpbench.com/

If you want to run my benchmarks yourself go:

https://github.com/emgiezet/phpers-summit-2023

Find balance between Lazy & Eager loading in Doctrine

Use Generators & Iterators

Trust me.

Check @**alexdaubois** presentation at **13:45**

Remember of batch processing

Partition your cli commands

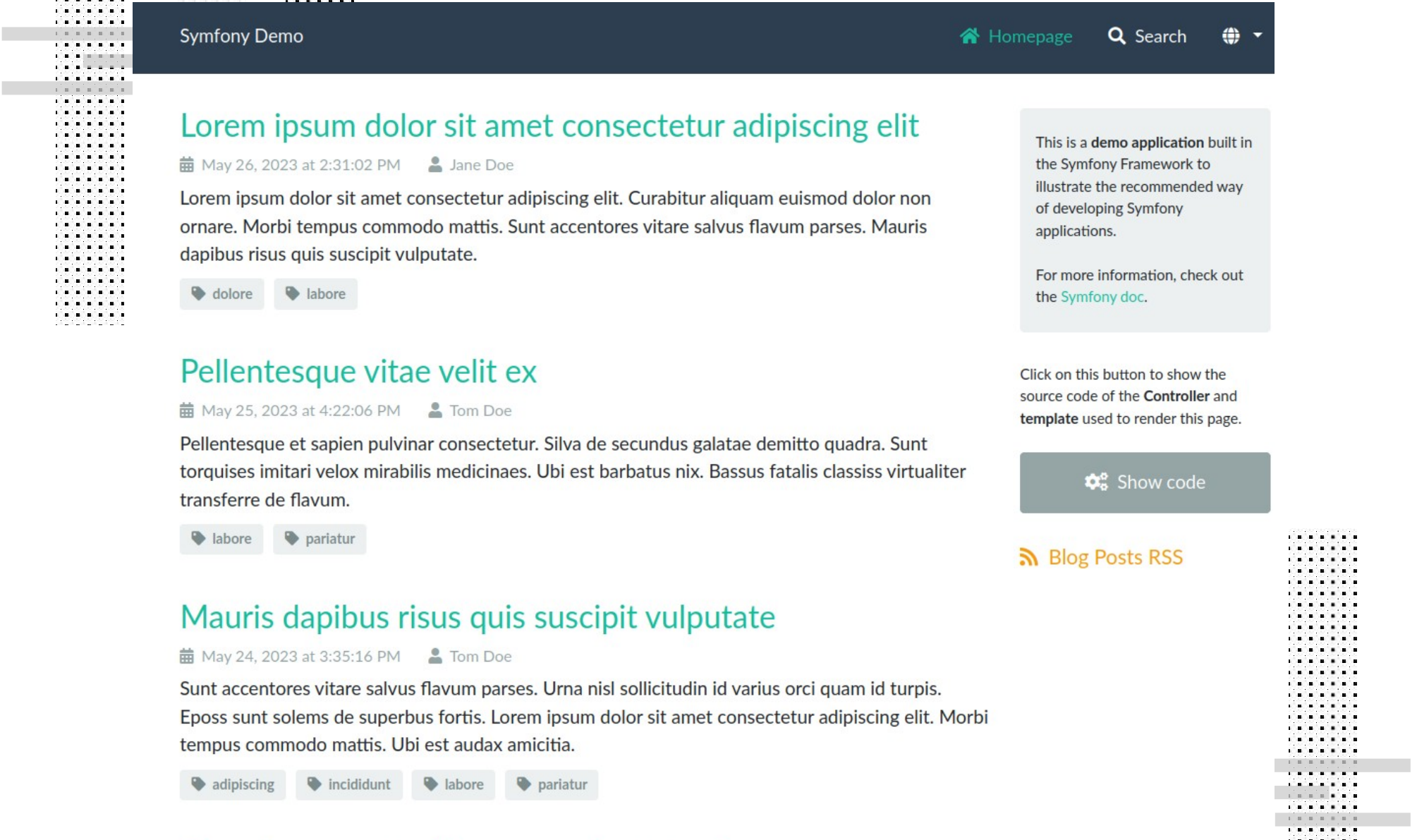
Never try to load XML’s bigger than 50 MB to DOMObject

Don’t spam logs. Each echo() each print() costs.

Photo by Fotis Fotopoulos on Unsplash

Benchmark App

- Symfony Demo App
  - In a docker container
  - Ngnix & php-fpm
  - In prod mode
  - Whitout xdebug
  - Mysql isntead of sqlite
  - It has symfony.cache enabled (#[Cache(smaxage: 10)])


Symfony Demo
Homepage
Search
Lorem ipsum dolor sit amet consectetur adipiscing elit
May 26, 2023 at 2:31:02 PM
Jane Doe
Lorem ipsum dolor sit amet consectetur adipiscing elit. Curabitur aliquam euismod dolor non ornare. Morbi tempus commodo mattis. Sunt accentores vitare salvus flavum parses. Mauris dapibus risus quis suscipit vulputate.
dolore
labore
Pellentesque vitae velit ex
May 25, 2023 at 4:22:06 PM
Tom Doe
Pellentesque et sapien pulvinar consectetur. Silva de secundus galatae demitto quadra. Sunt torquises imitari velox mirabilis medicinaes. Ubi est barbatus nix. Bassus fatalis classiss virtualiter transferre de flavum.
labore
pariatur
Mauris dapibus risus quis suscipit vulputate
May 24, 2023 at 3:35:16 PM
Tom Doe
Sunt accentores vitare salvus flavum parses. Urna nisl sollicitudin id varius orci quam id turpis. Eposs sunt solems de superbus fortis. Lorem ipsum dolor sit amet consectetur adipiscing elit. Morbi tempus commodo mattis. Ubi est audax amicitia.
adipiscing
incididunt
labore
pariatur
This is a demo application built in the Symfony Framework to illustrate the recommended way of developing Symfony applications.
For more information, check out the Symfony doc.
Click on this button to show the source code of the Controller and template used to render this page.
Show code
Blog Posts RSS

What we gonna benchmark?

- 8.1
- 8.1 Opcache
- 8.1 Opcache + JIT compiler enabled

How we gonna benchmark?

- Siege on multiple urls

```
siege -c 100 -r 100 -b -i -f urls.txt
```

PHP 8.1 – no opcache


```
{
    "transactions":                  66730,
    "availability":                 100.00,
    "elapsed_time":                 149.54,
    "data_transferred":            5511.54,
    "response_time":                  0.22,
    "transaction_rate":             446.24,
    "throughput":                    36.86,
    "concurrency":                   99.45,
    "successful_transactions":       66730,
    "failed_transactions":               0,
    "longest_transaction":            1.75,
    "shortest_transaction":           0.00
}
```

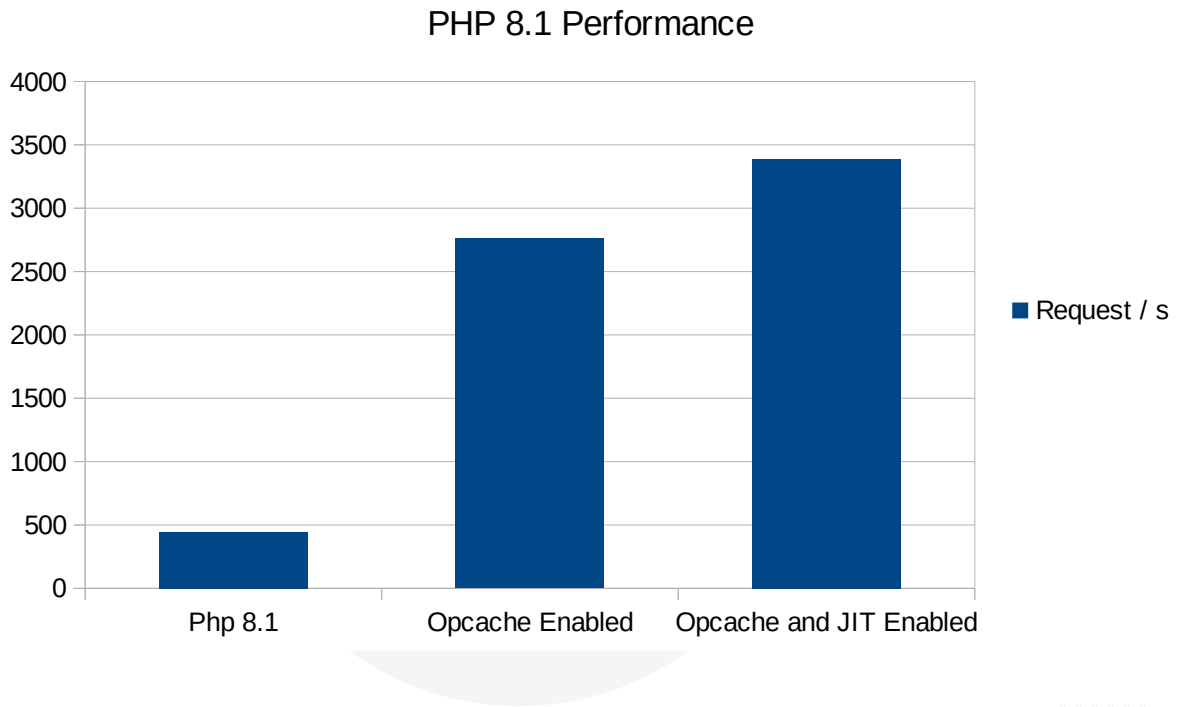
PHP 8.1 – opcache


```
{
        "transactions":                  66988,
        "availability":                 100.00,
        "elapsed_time":                  24.26,
        "data_transferred":            5535.86,
        "response_time":                  0.04,
        "transaction_rate":            2761.25,
        "throughput":                   228.19,
        "concurrency":                   99.38,
        "successful_transactions":       66988,
        "failed_transactions":               0,
        "longest_transaction":            0.40,
        "shortest_transaction":           0.00
}
```

PHP 8.1 – opcache & jit

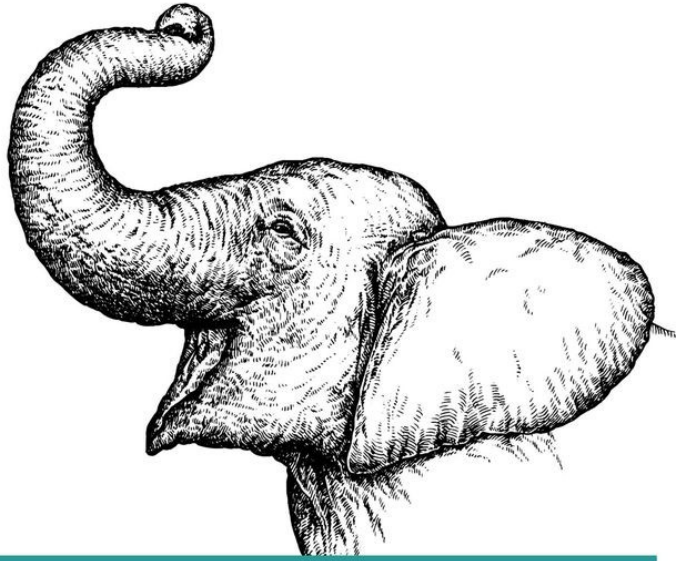

```
{
        "transactions":                  66724,
        "availability":                 100.00,
        "elapsed_time":                  19.69,
        "data_transferred":            5510.57,
        "response_time":                  0.03,
        "transaction_rate":            3388.73,
        "throughput":                   279.87,
        "concurrency":                   99.36,
        "successful_transactions":       66724,
        "failed_transactions":               0,
        "longest_transaction":            0.45,
        "shortest_transaction":           0.00
}
```

Req/s boost


PHP 8.1 Performance
4000
3500
3000
2500
2000
1500
1000
500
0
Php 8.1
Opcache Enabled
Opcache and JIT Enabled
Request / s


What you can apply to your project?
Photo by Brands&People on Unsplash

The answer to every programming question ever conceived



It Depends

The Definitive Guide

O RLY?

@ThePracticalDev


Greenfield
Photo by Azin Javadzadeh on Unsplash

Manage the performance expectations

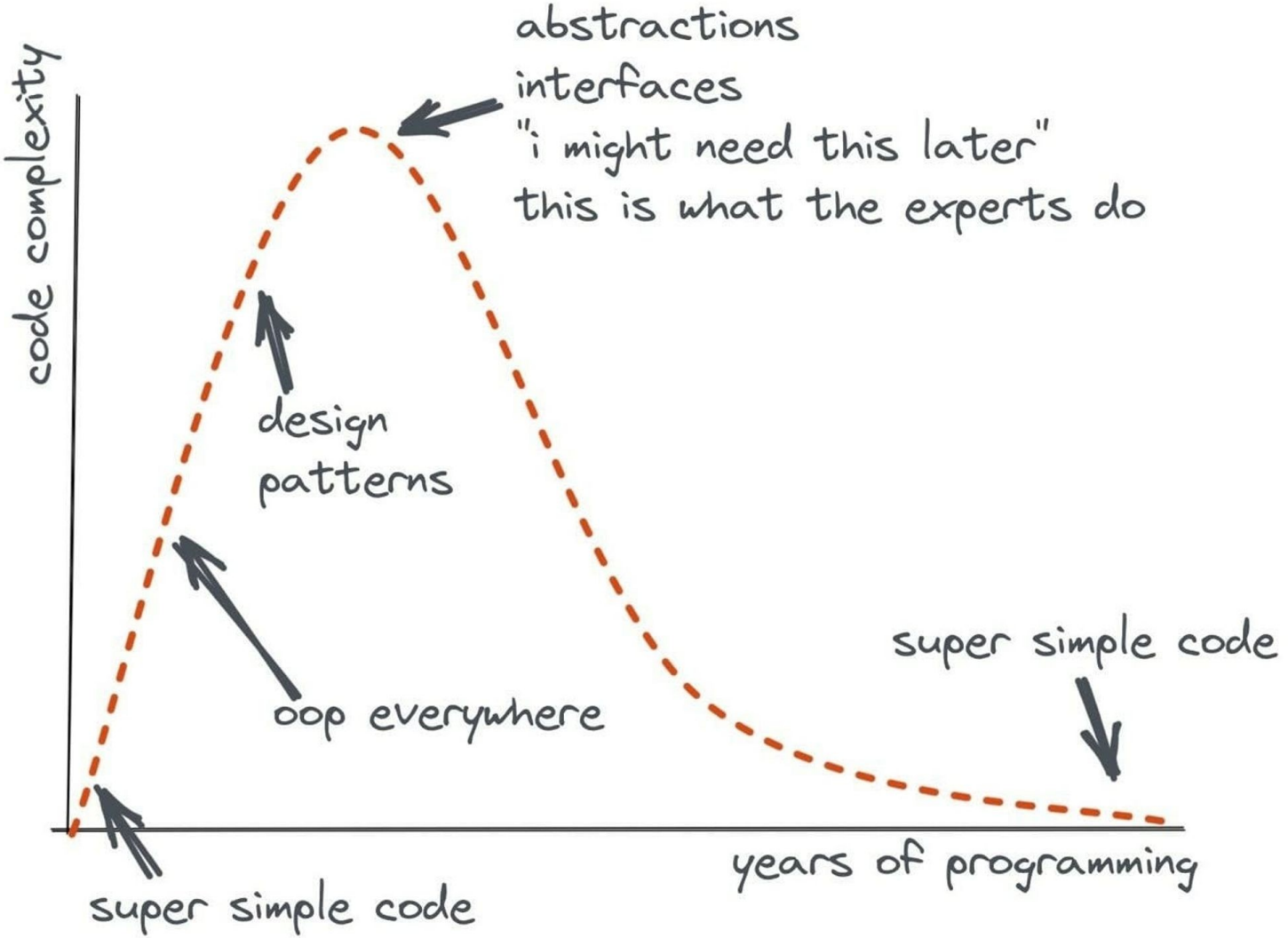
Run loadtests after each delivery cycle.

Add benchmarks to your CI pipeline.

Try to align your benchmark scenarios with KPI that businesses wants.

Take baby steps

Don’t over-engineer


abstractions
interfaces
"i might need this later"
this is what the experts do
code complexity
design
patterns
oop everywhere
super simple code
super simple code
years of programming

Add static code analysis. With performance rules check.

Always consider performance impact on feature planning.

Always consider performance impact on code review.

First implementation is always rubbish.

Legacy

Photo by Brett Jordan on Unsplash

Set up performance goals.

Find bottlenecks.

Prioritize them with businesses.

Decisions should be supported by real data.

Plan them. Apply one change at the time and then benchmark!

And...
Repeat from the first step.

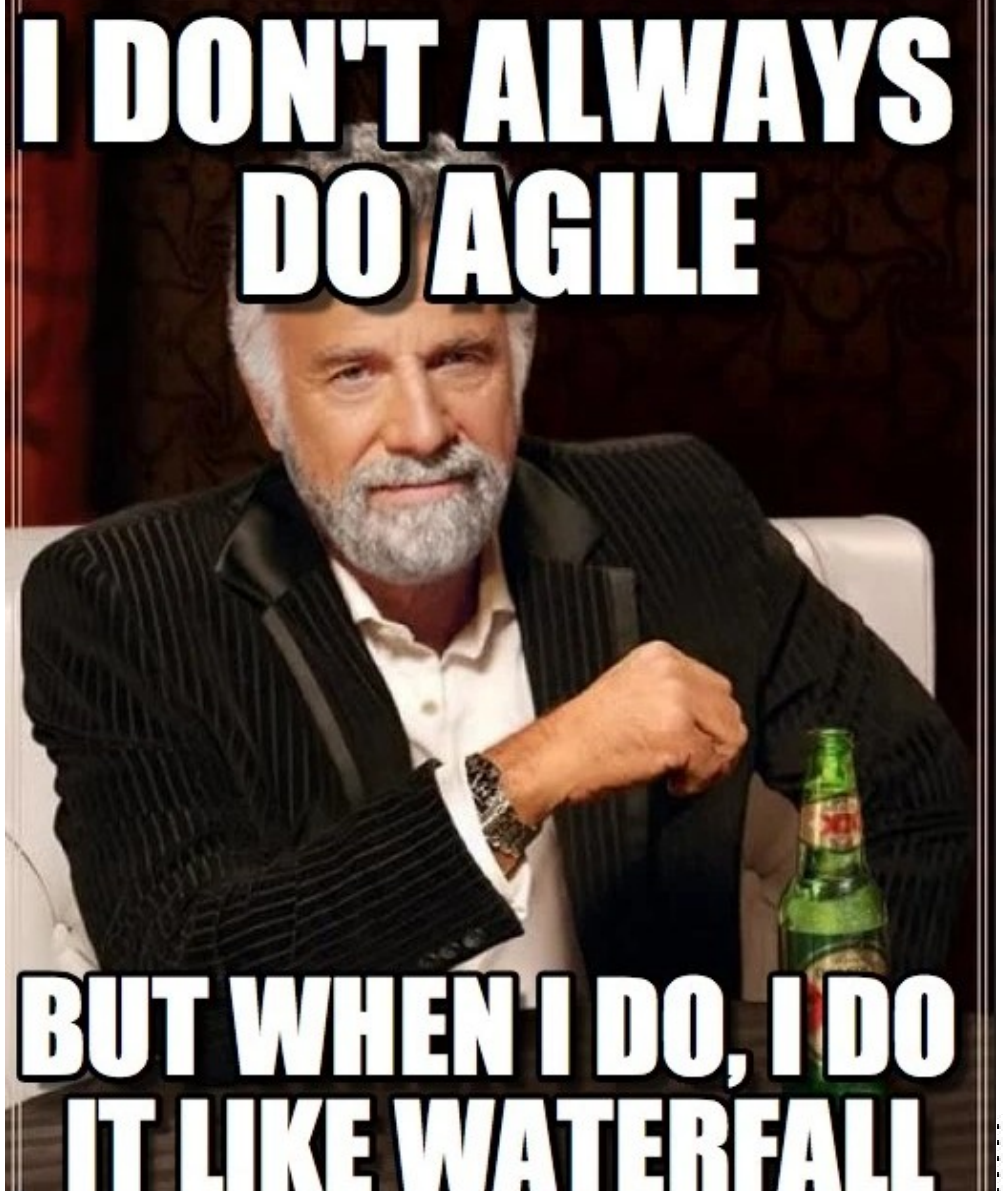

I DON'T ALWAYS
DO AGILE
BUT WHEN I DO, I DO
IT LIKE WATERFALL

Photo by Fotis Fotopoulos on Unsplash

Catch me!

- Code from presentation: https://github.com/emgiezet/phpers-summit-2023
- Twitter: @**mgz**

Photo by Kelly Sikkema on Unsplash