
6 tips to build Fast Web Applications
-
Luciano Mammino (@loige)
PHP Dublin 22/03/2016

About me

Integrations Engineer at Smartbox

Php developer since ***.php3**

I Tweet as **@loige**

I Blog on **loige.co**

Preamble

- Only 6 tips (not exhaustive)
- Focused on backend
- Many generic advices (not just Php)
- ... Tools and examples (mostly) for Php :)

Tip 1.

Avoid premature optimisation!

"Premature optimisation is the root of all evil"

— Donald Knuth

(not an excuse to write 💩 code!)

The right mindset

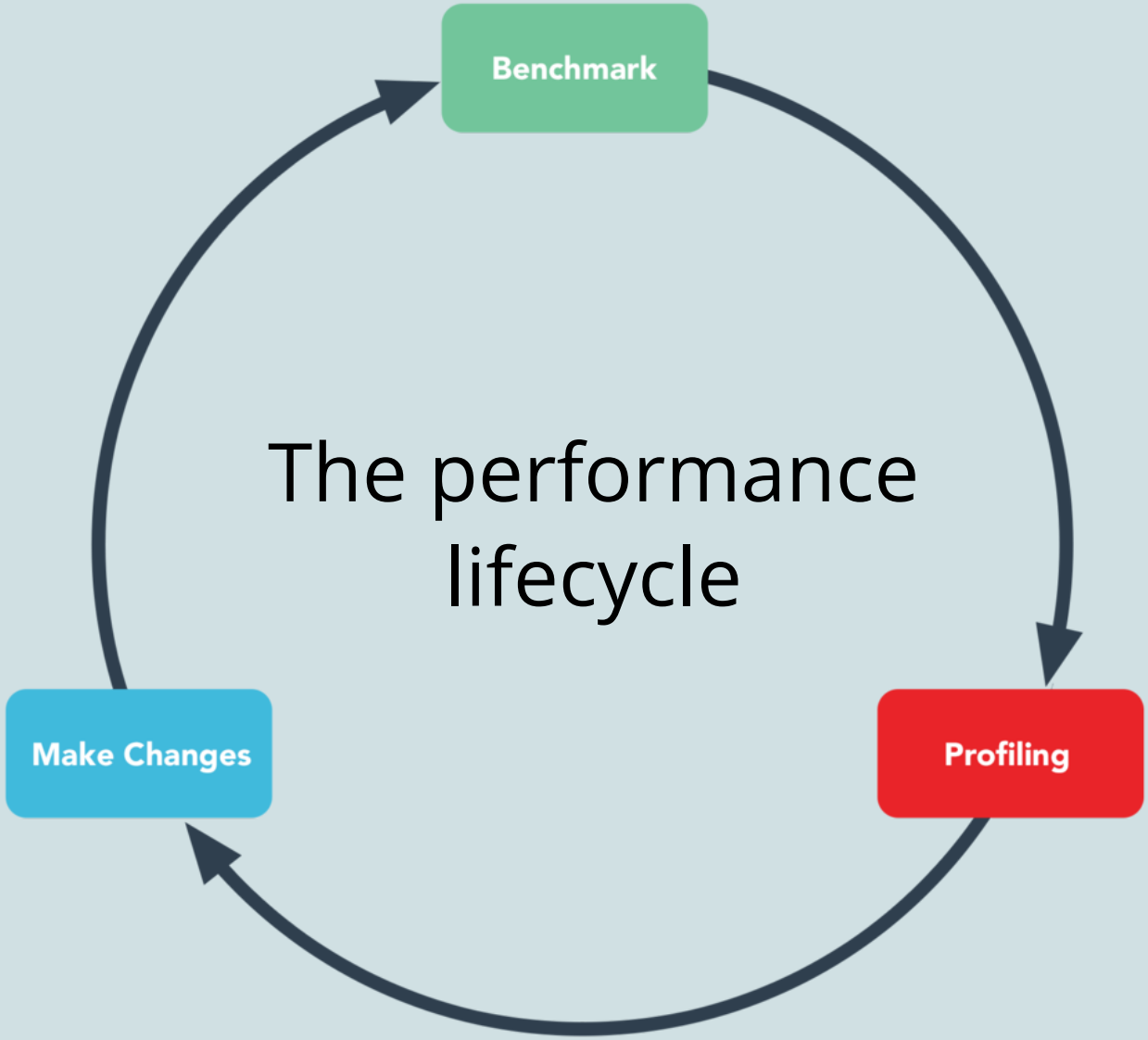
Do you really have a problem?

How big the problem is?

Measure before questioning

Set (realistic) goals

Improve only where needed!



The performance lifecycle

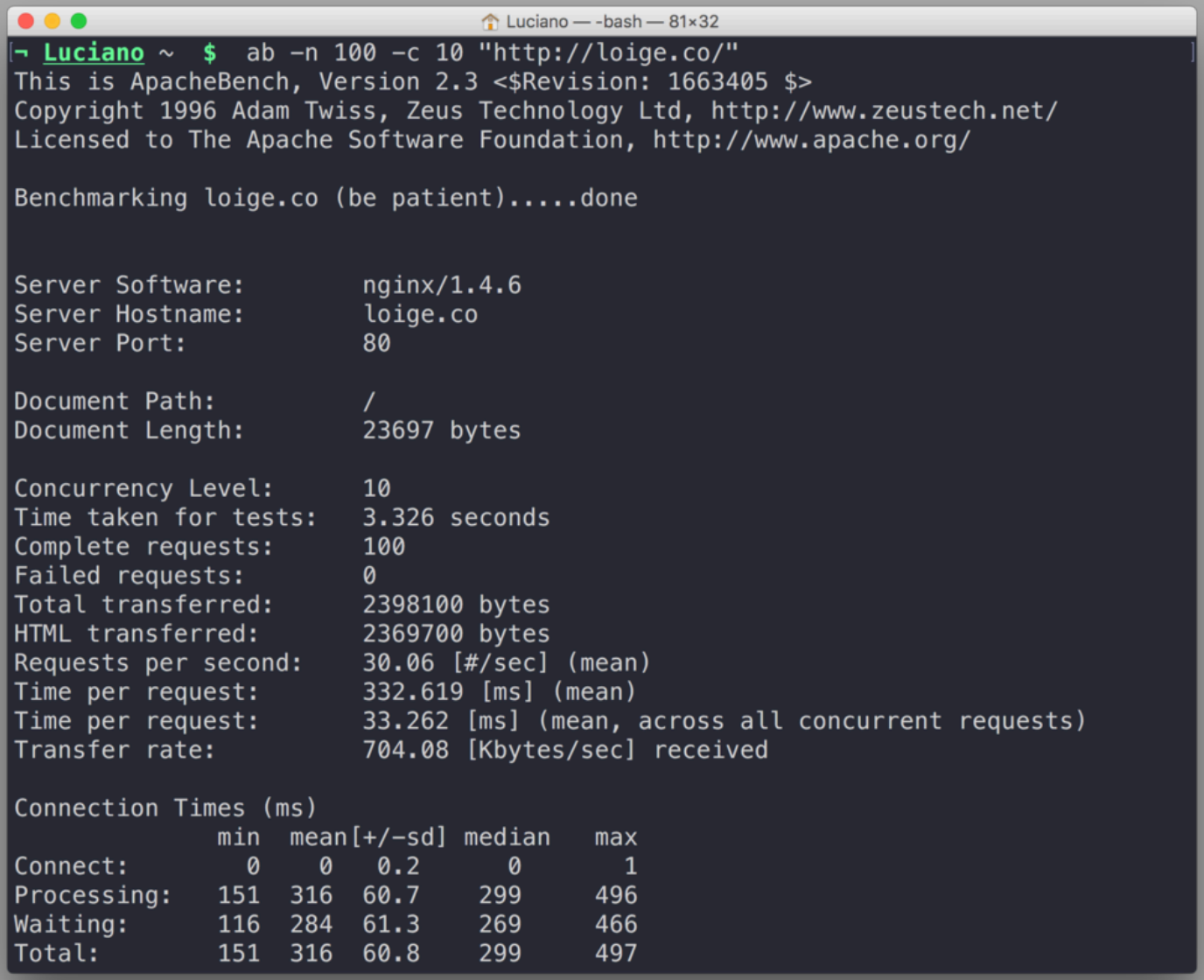
Benchmarking

Command line:

- Apache Bench (ab)
- Siege

Graphical:

- Apache JMeter
- Soap UI (for APIs)


Luciano — -bash — 81×32
¬ Luciano ~ $ ab -n 100 -c 10 "http://loige.co/"
This is ApacheBench, Version 2.3 <$Revision: 1663405 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/
Benchmarking loige.co (be patient).....done
Server Software: nginx/1.4.6
Server Hostname: loige.co
Server Port: 80
Document Path: /
Document Length: 23697 bytes
Concurrency Level: 10
Time taken for tests: 3.326 seconds
Complete requests: 100
Failed requests: 0
Total transferred: 2398100 bytes
HTML transferred: 2369700 bytes
Requests per second: 30.06 [#/sec] (mean)
Time per request: 332.619 [ms] (mean)
Time per request: 33.262 [ms] (mean, across all concurrent requests)
Transfer rate: 704.08 [Kbytes/sec] received
Connection Times (ms)
min mean[+/-sd] median max
Connect: 0 0 0.2 0 1
Processing: 151 316 60.7 299 496
Waiting: 116 284 61.3 269 466
Total: 151 316 60.8 299 497

Profiling

Active or **Passive**?

Xdebug (Active)

- Gives a lot of data
- Slows down the execution
- Good in development
- Easy to integrate with IDEs
- Also a debugger

xhprof (Passive)

- Minimal impact
- Less data
- Good in production
- Dedicated Web GUI (xhgui)
- Supported by Facebook...

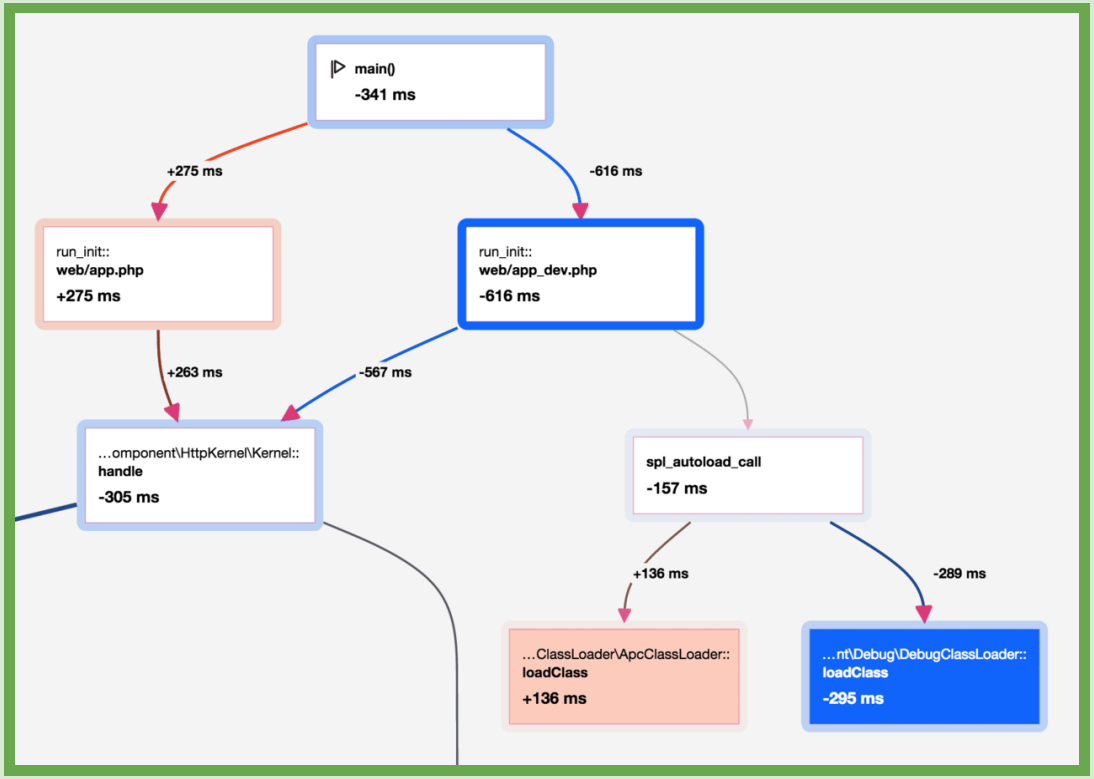
Libraries

Symfony stopwatch component

```php
<?php

use Symfony\Component\Stopwatch\Stopwatch;

$stopwatch = new Stopwatch();

// Start event named 'eventName'

$stopwatch->start('eventName');

// ... some code goes here

$event = $stopwatch->stop('eventName');

echo $event->getDuration(); // xxx milliseconds
```

Cloud tools

Blackfire.io & Tideways



Tip 2.

Do just what you need to

I JUST

DO THINGS

A.k.a.

"Don't do things that you don't need to"

E.g.

- Load classes that you are not using
- Open a connection to a database for every request
- Parse a huge XML config file for every request

Autoloading

~~Syntetic Namespaces~~, ~~Classmaps~~, ~~PSR-0~~, **PSR-4**

You get it for **free** with **Composer**!

```php
<?php

require __DIR__ . '/vendor/autoload.php'; // generated by Composer

use Mario\Characters\Mario;
use Mario\Powerups\Mushroom;

// Mario & Mushroom classes not loaded yet...

$mario = new Mario();

// Mario class is loaded!

$mushroom = new Mushroom();

// Mushroom class is loaded!

$mario->eat($mushroom); // Tasty!
```

Other patterns

Dependency Injection

(& Dependency Injection Container)

Simplifies the loading of classes and the resolution of dependencies

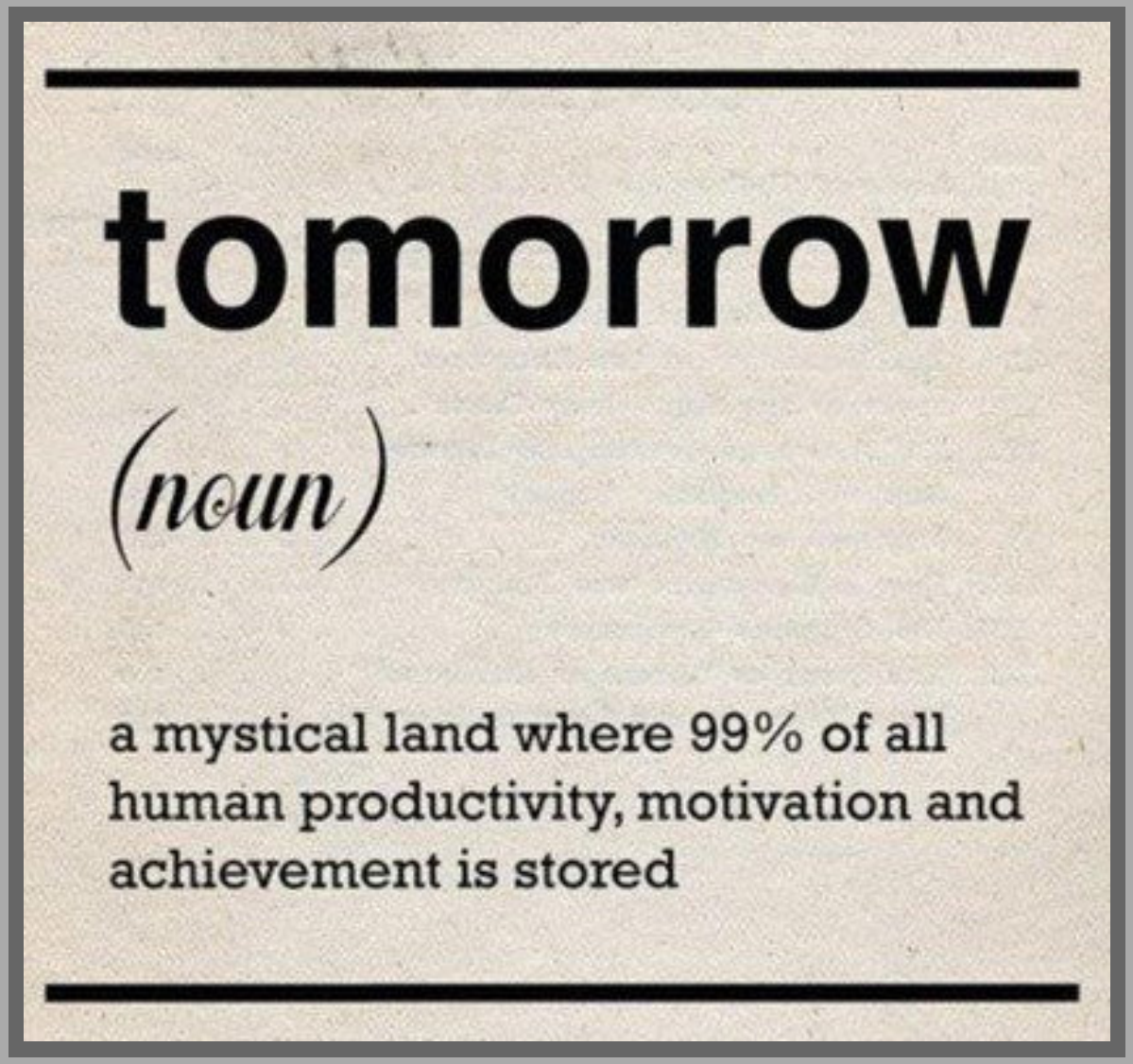
Lazy loading & Proxy

Access resources only when you are about to use them

Tip 3.

If you really need to do it,

do it **tomorrow**!


tomorrow
(noun)
a mystical land where 99% of all human productivity, motivation and achievement is stored

Defer tasks when possible

E.g.

- Send an email to a user to confirm an order
- Resize an image after the upload
- Aggregate metrics

(Serve the response as soon as it's ready!)

Queues to the rescue

Resque

A PHP library (Redis as data store).

Laravel Queues

Laravel/Lumen built-in solution (multiple data storages).

Gearman

Generic job server that supports many languages.

Beanstalkd

Work queue originally written to speed up Facebook

Tip 4.

Use cache to avoid repetitive computations and roundtrips

Different Caching layers:

- **Byte code Cache** (APC, OpCache)
- **Application Cache** (Redis, Memcache, Gibson)
- **HTTP Cache** (E-tag, Cache-control headers)
- **Proxy Cache** (Varnish, Squid, Nginx)

"There are only two hard things in Computer Science: cache invalidation and naming things"

– Phil Karlton

Tip 5.

Beware of the N+1 query problem!

```php
<?php

function getUsers() {
  //... retrieve the users from the database (1 query)
  return $users;
}

function loadLastLoginsForUsers($users) {
  foreach ($users as $user) {
    $lastLogins = ... // load the last logins
                // for the user (1 query, executed n times)
    $user->setLastLogins($lastLogins);
  }

  return $users;
}

$users = getUsers();
loadLastLoginsForUsers($users);
```

N+1 Queries! 😓

```
SELECT id FROM Users; -- ids: 1, 2, 3, 4, 5, 6...
SELECT * FROM Logins WHERE user_id = 1;
SELECT * FROM Logins WHERE user_id = 2;
SELECT * FROM Logins WHERE user_id = 3;
SELECT * FROM Logins WHERE user_id = 4;
SELECT * FROM Logins WHERE user_id = 5;
SELECT * FROM Logins WHERE user_id = 6;
-- ...
```

Better solution

```
SELECT id FROM Users; -- ids: 1, 2, 3, 4, 5, 6...
SELECT * FROM Logins
  WHERE user_id IN (1, 2, 3, 4, 5, 6, ...);
```

or use JOINS...

Don't trust the ORM

Check the queries generated by your code!

Tip 6.

Plan for horizontal scalability

Build your app to be replicated across many servers

Sessions:

- Don't use the default file based engine
- Be as much stateless as possible

User files:

- CDN, Cloud Storage (S3, Cloudfiles), Sync (NFS, Gluster)

Consider **Microservices**...

BONUS Tip.

Update to PHP7

php.net/migration70

(Rasmus approves)

Let's Recap

1. Avoid premature optimisation!
2. Do just what you need to
3. If you really need to do it, do it tomorrow!
4. Cache me if you can
5. Beware of the N+1 query problem!
6. Plan for horizontal scalability
7. Bonus: Update to Php 7
8. Super Secret Bonus: bit.ly/php-perf

Thank you!

Let's keep in touch:

http://loige.co - @loige