WWW.PRESTASHOP.COM

Prestashop System Performance White Paper

Version 1.0 - 26/01/2021

Prologue

Performance is an exciting, varied, and challenging discipline.
— Brendan Gregg

You are using PrestaShop to develop your business and we are delighted to accompany you on this path. With business growing, merchants want to know more about their solution's ability to perform and scale.
PrestaShop is built on top of mature and robust technologies that are used by some of the most visited websites in the world. The solutions implemented to achieve top performance and scale on top of this technical stack are quite standard and well known. This White Paper will list which ones you can use in the context of PrestaShop and what you can expect from them.

"If you can't measure it, you can't improve it" is a famous quote by management guru Peter Drucker but it also applies to the field of software performances. That's why a large part of this White Paper will focus on the benchmark methodology itself, in order to give you the tools to evaluate your setup's performance and ensure the improvements are effective in your specific case.

This White Paper provides guidance on how to improve your PrestaShop installation by optimizing your configuration and infrastructure.

«Which infrastructure improvement to support more visitors and sell more ?»

Table of contents

Benchmark context

Benchmarking tests performance in a controlled manner, allowing choices to be compared and performance limits to be understood—before they are encountered in production.
- Brendan Gregg

PrestaShop is the well known Open Source software you can download and run with almost every hosting company. You are free to configure your shop, install extensions and modify the code in any way you want in order to answer your specific needs.

Every PrestaShop setup being unique, your mileage may vary from the benchmarks you will find in this paper.
This also means that no perfect benchmark, encompassing every possible setup, can exist.
But we had to choose one, deeming it relevant enough and we'll provide you with all the required information to reproduce the scenarii and let you check where you stand with our own setup.

a. The shop we are testing

I. Shop hosting

In our context, we decided to go with an easily reproducible environment using docker and google compute engine. Having run almost 2000 tests, ease of setup by automation was essential. This was an honest assumption, that docker would not deviate too much from a bare metal setup. You'll find more details about the hosting solution in the Annexes.

II. Shop version

We are using the latest PrestaShop version available to us at the moment of this writing: version 1.7.6.3, running with php 7.2. Using docker, we use the official ones provided by the PrestaShop company.

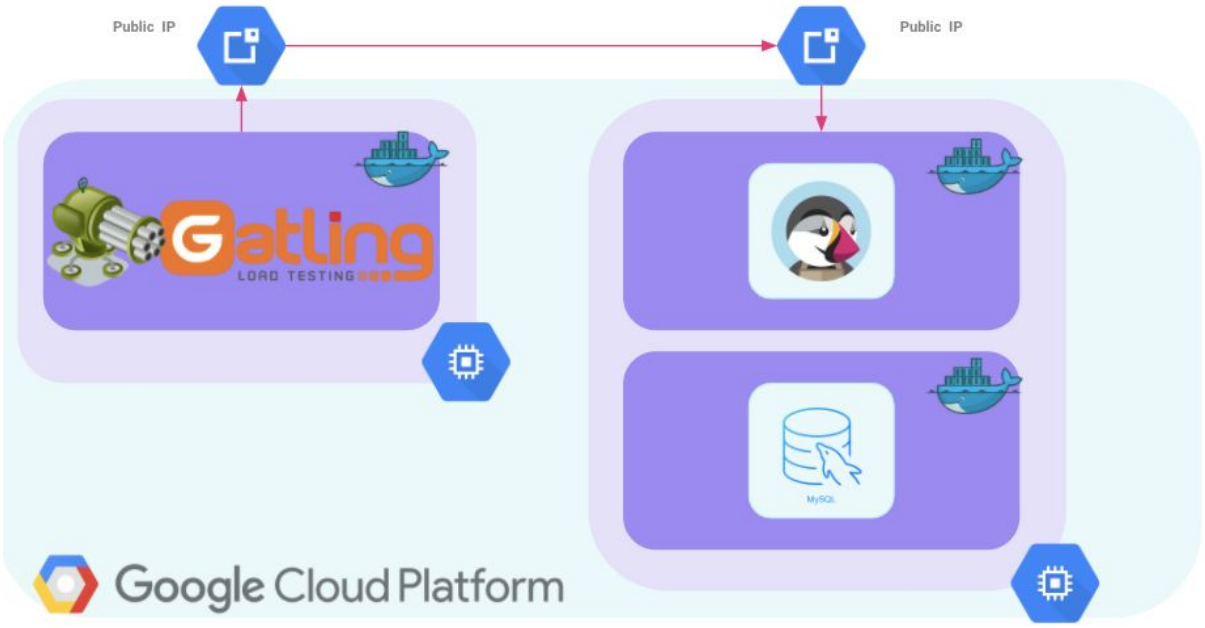
III. Shop profile

Prestashop is used in all sizes of shops around the world, from single product shop to tens of thousands of product catalogues.

We chose a medium size of content, with 1000 products, 10 suppliers, 2 languages, 5 categories, 1000 commands which result in a database size around 140Mb.

IV. Network

Depending on the visitor's terminal and network, pages load times can vary a lot. Knowing that, our goal here is to focus on the Response Time : an acceptable time to generate and download the page.
To do this in an acceptable manner, we are running our test in another GCP instance, running gatling in a docker container and we are accessing the shop through a public network, just like this:



b. Benchmark scenarios

I. Distinct scenarios

We defined several scenarii, each of them reflecting a particular use of the functionalities of an e-commerce website:

- FrontOffice crawl - Visitor Count

This is the most used scenario, with visitors browsing, opening categories and product pages. We aim to simulate a standard visitor surfing at an average speed over the different pages.
We chose the arbitrary value of opening **15 pages per session**, with a pause after each page opening varying between 0 and 5 seconds.

- FrontOffice order - Customer Count

A small part of visitors will convert into customers by adding product(s) in cart, and placing an order. We chose to add only one product in the cart. **8 pages loads by order** placement are needed (display product, add it to cart, type name, address, choose carrier...).

II. Avoiding randomness

Opening random pages would lead to non-repeatable results. That's why we use only predefined scenarios. Random page opening could be used in stress tests, which is not the scope of this study.

C. Acceptable response times

We decided this acceptable response time should be set to **300ms** for visitors browsing (product/category page), and 1.0s to checkout process (add a product to cart, place order).
Keeping this measurement below the defined threshold gives us the **business-valued KPI User per Hour**, which is the maximum supported scenario count to stay below the response time threshold.
Of course if you decide to accept higher response times it will also increase the maximum number of concurrent visitors we get as the result. We choose a low response time to make sure we don't produce false expectations. (The average human reaction time, is on the order of a quarter of a second 250 milliseconds)

I. Response time

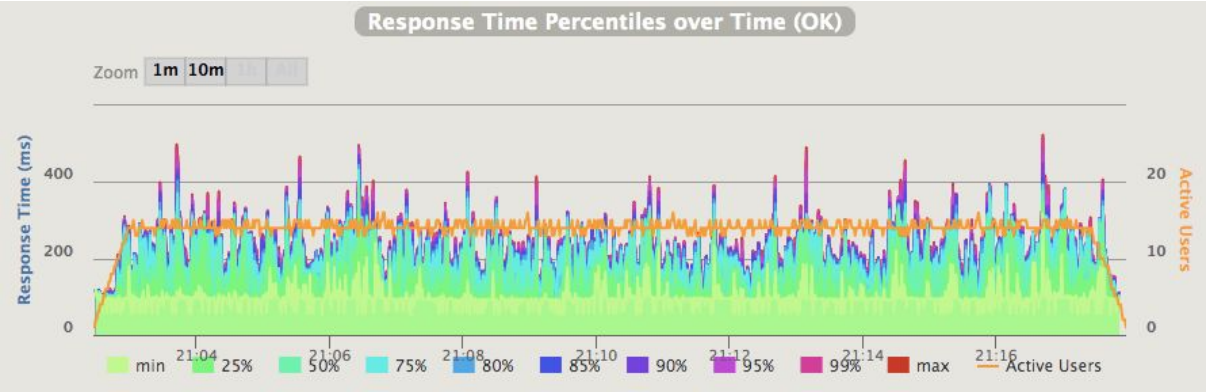
It's the response time when the system is under load.
To get reproducible results we use the "**95th percentile**" values. This excludes non-representative values, which could be caused by abnormal network latency, race-condition process or some other not reproducible states.

II. Page views per hour

Users per hour is the number of users that can crawl the shop during one hour while remaining under the acceptable response time threshold.
As already mentioned, there are 2 types of Users:

- **Visitors**, crawling the shop for products
- **Customers**, adding a product in their cart, filling in their address, etc..

To find out this KPI for each shop configuration, we ran gatling tests, with the users (either Visitors or Customers) distributed evenly during the test run.



2. Benchmark results

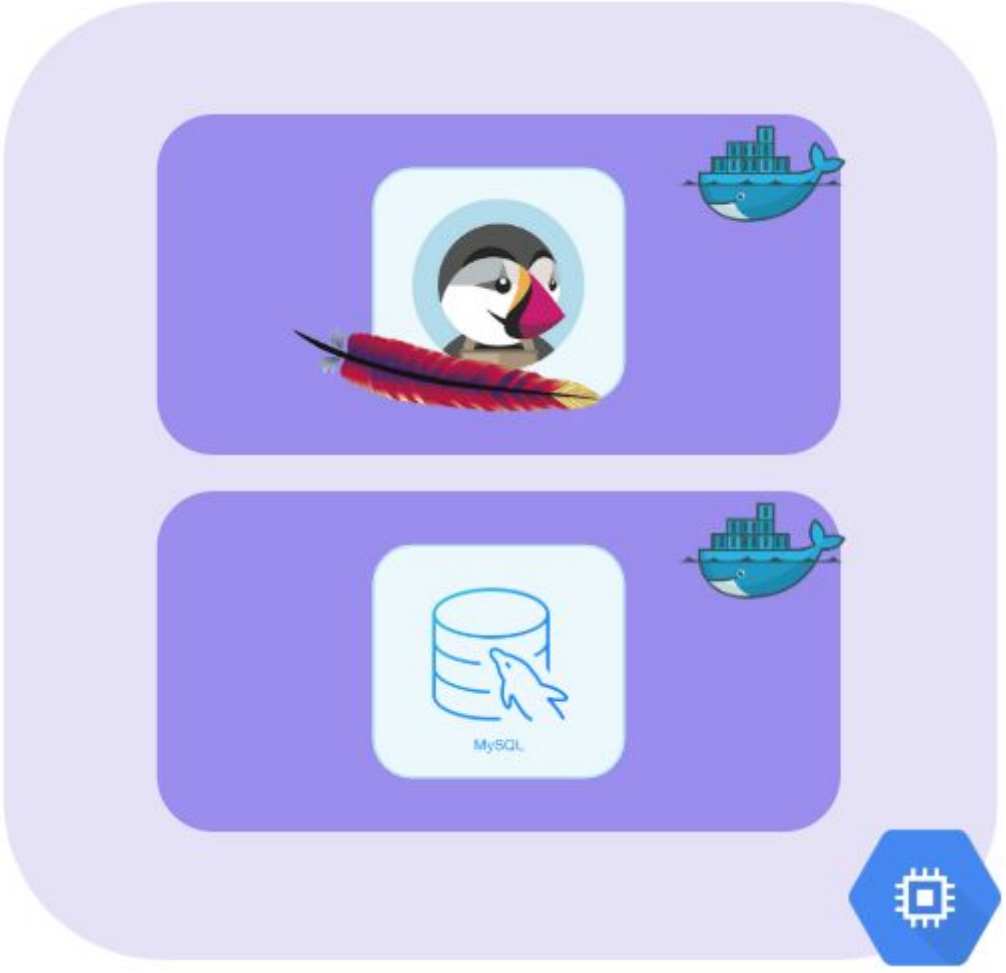
There are lies, damn lies and then there are performance measures.
— Anon et al., “A Measure of Transaction Processing Power” [Anon 85]

a. Single server architecture

In this section we’ll benchmark the most common configuration available to host a PrestaShop solution: a single server, hosting both the webserver, the php runtime and the MySQL database.
Some other configurations exist, of course, they can be less common and may add some complexity that not all setups require (especially once dealing with several frontends).

i. Single server with apache2 webserver

The most common architecture available when setting up a shop: one server running all the required services and apache2 rendering the php pages.

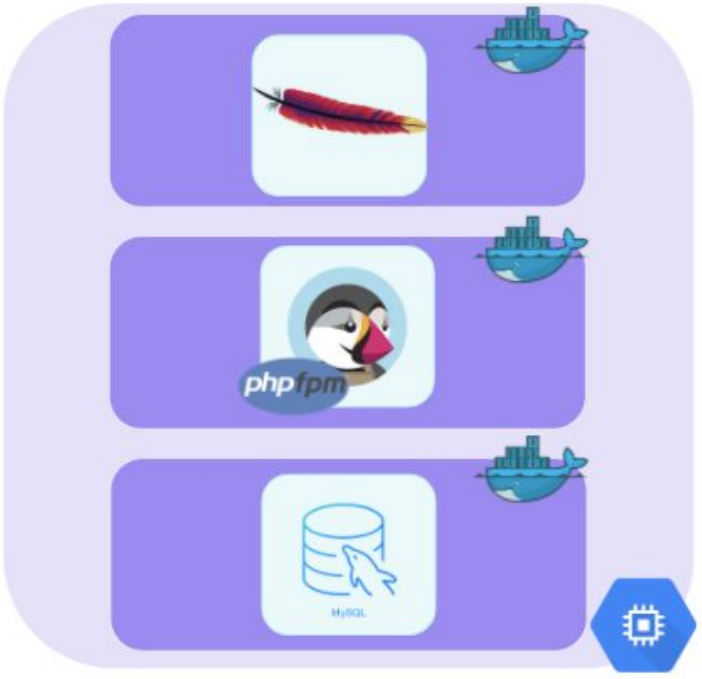


Configuration		Result
Server	**Set-up**	**Max Visitors Per Hour**
1 vCPU/ 3.75Gb RAM disk: standard	- Apache2 + php-module - MySQL - Out-of-the-box configuration	220

Well, that's our first result. It's not bad, but don't worry: that's just the first test of a long series !

ii. Single server with php-fpm

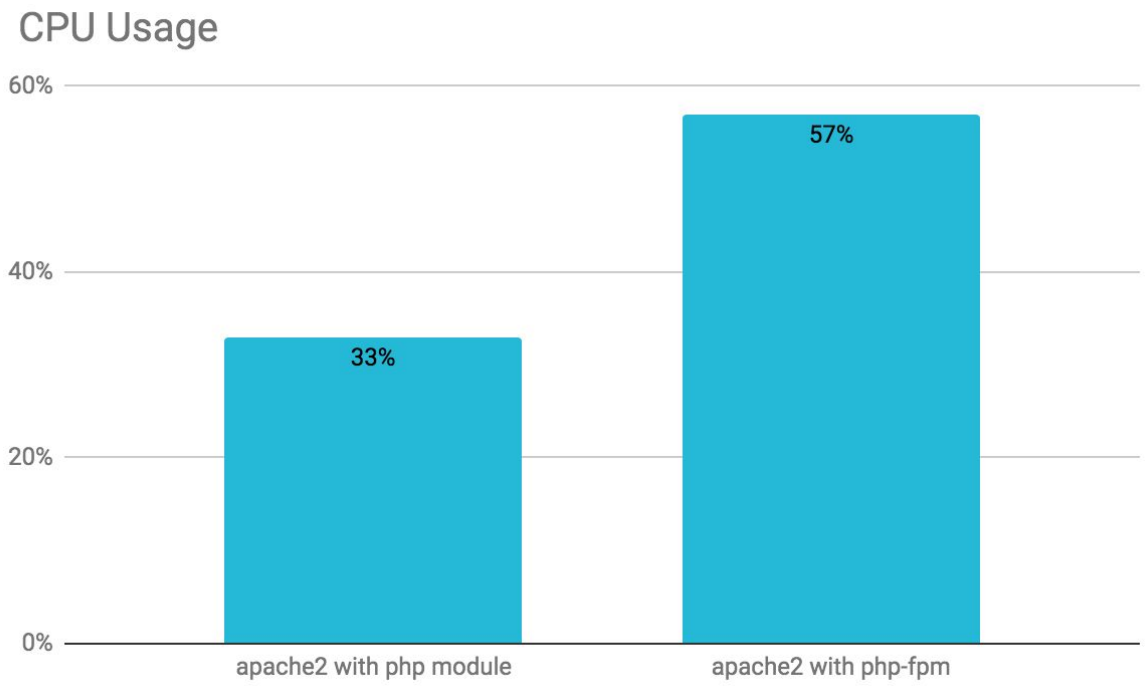
The next best thing when discussing php performance is php-fpm. Unfortunately, we rarely see any practical comparison between the two solutions' true potentials. We thought it was the perfect opportunity and ran the same test (with the exact same Visitor Count) against the php-fpm stack.
Of course, we had to update the setup a little, mostly by introducing a php-fpm runtime, but everything still runs on the same docker stack, on a single GCP Compute instance, sized the same way:



And here are the results, around 60% the number of visitors compared to the previous configuration !

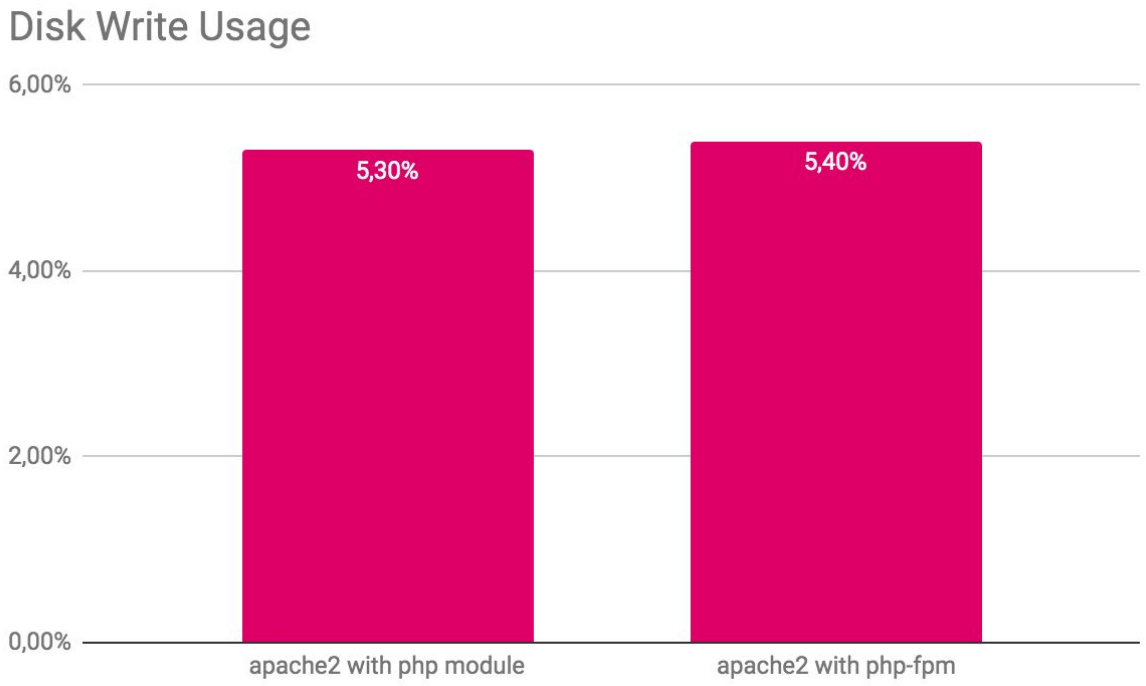
Configuration		**Result**
Server	**Set-up**	**Max Visitors Per Hour**
1 vCPU/ 3.75Gb RAM disk: standard	- **Apache2 + php module** - MySQL - Out-of-the-box configuration	220
1 vCPU/ 3.75Gb RAM disk: standard	- **Apache2 + php-fpm** - MySQL - Out-of-the-box configuration	360

Which is already an interesting number to reach in itself, just by moving from apache2 with php module to php-fpm.
Another interesting fact is that the CPU ratio is much more efficient with PHP-FPM than with apache2 module:



CPU consumption and production environment
Please note that for your production environments, you'd rather want to remain under the 40% CPU Usage, allowing any burst of activity to be managed gracefully.
As you'll see in our remaining tests, there's no reason the application would stop working efficiently when exceeding the 40% CPU usage, that's just a precaution to ensure the stability and the user experience should you experience any peak of users.

Also, looking at the disk I/O:



Disk I/O Usage

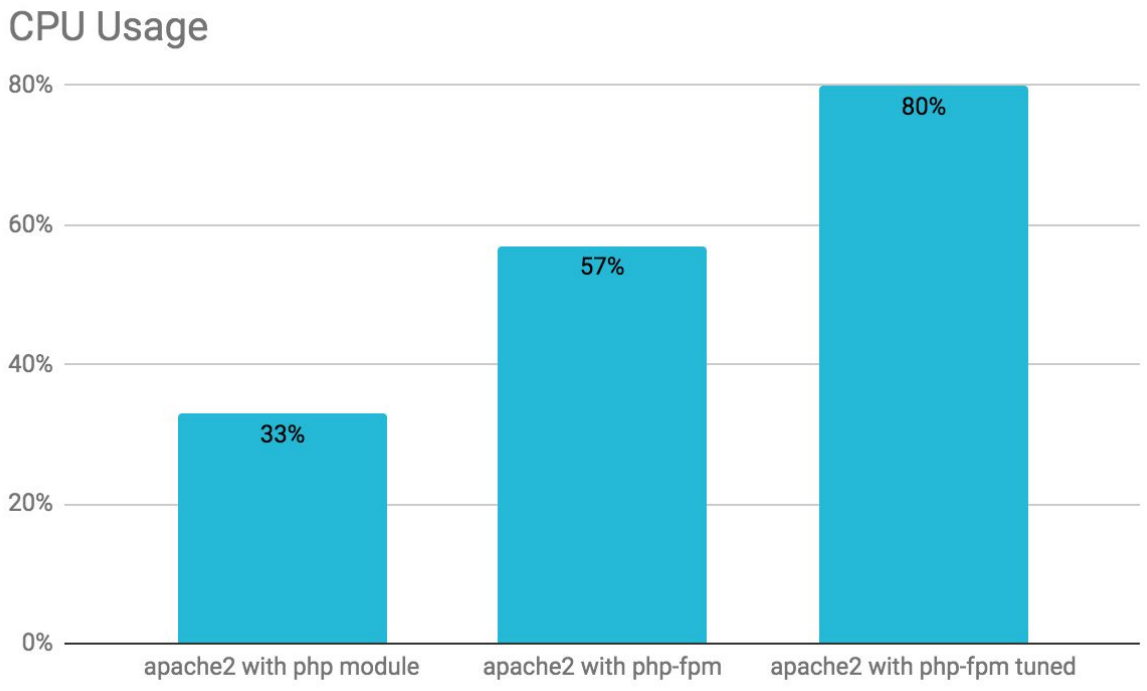
In order to check the disk's usage, we did a bit of calculus:

- First, we took the number of operations the disk could manage and multiplied it by the block size. For standard disk : 75 (number of write operations) x 4 (block size) = 300
- Then we took the number of disk writes we had, 64, and divided it by the number of allowed operations : 64 / 300 = 0,21, hence 21% of disk write usage
- Of course, an SSD disk can perform far more operation, 1500 to be exact: 1500 (number of write operations) x 4 (block size) = 6000

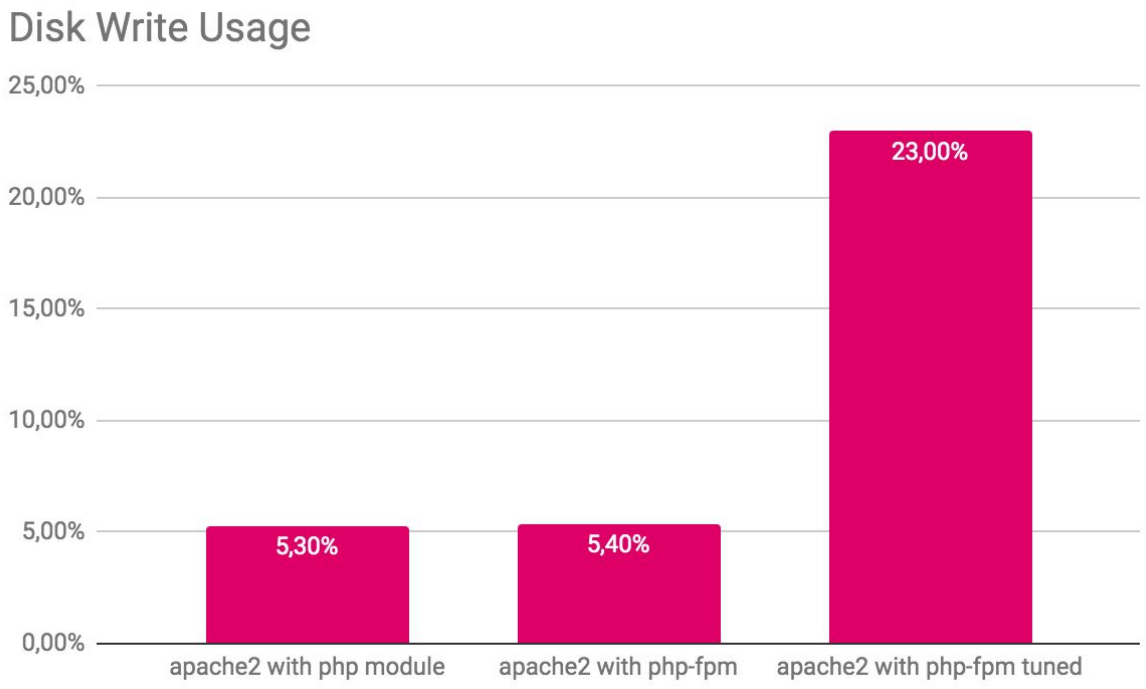
iii. Single server with php-fpm and tuning

Even then, we were sure that more could be done. That we could still get some more page views. Hence, we reviewed our configurations thoroughly and tuned them, mostly by enforcing all available caching options (either from php runtime or from mysql).
You'll find those parameters at the end of this document in the Annexes section, but without further ado, here are the results:

Configuration		**Result**
Server	**Set-up**	**Max Visitors Per Hour**
1 vCPU/ 3.75Gb RAM disk: standard	- **Apache2 + php-fpm** - **MySQL** - Out-of-the-box configuration	360
1 vCPU/ 3.75Gb RAM disk: standard	- **Apache2 + php-fpm tuned** - **MySQL tuned**	740



Also, we are seeing a big increase in Disk IO usage, which explains the contention starting to slow down the application, even after enabling caching. We'll see if that concurs with our next test !



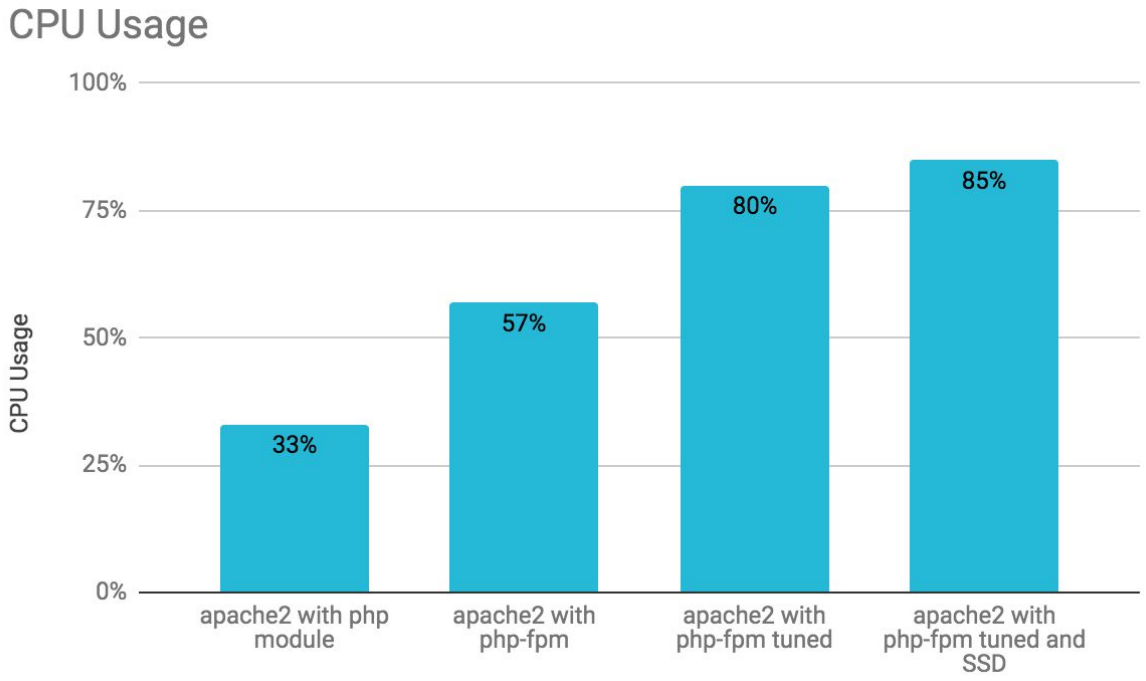
Validity of CPU Usage
Of course, CPU usage is far from being the sole indicator of an application's performance and health. Well, depending how you read it.
In our case, a high CPU usage means that most of the other bottlenecks (disks, IOs, parallel processes, etc..) have been managed and the CPU can now do its work as expected by focusing on the application itself.

iv. Single server with php-fpm and tuning and SSD disks

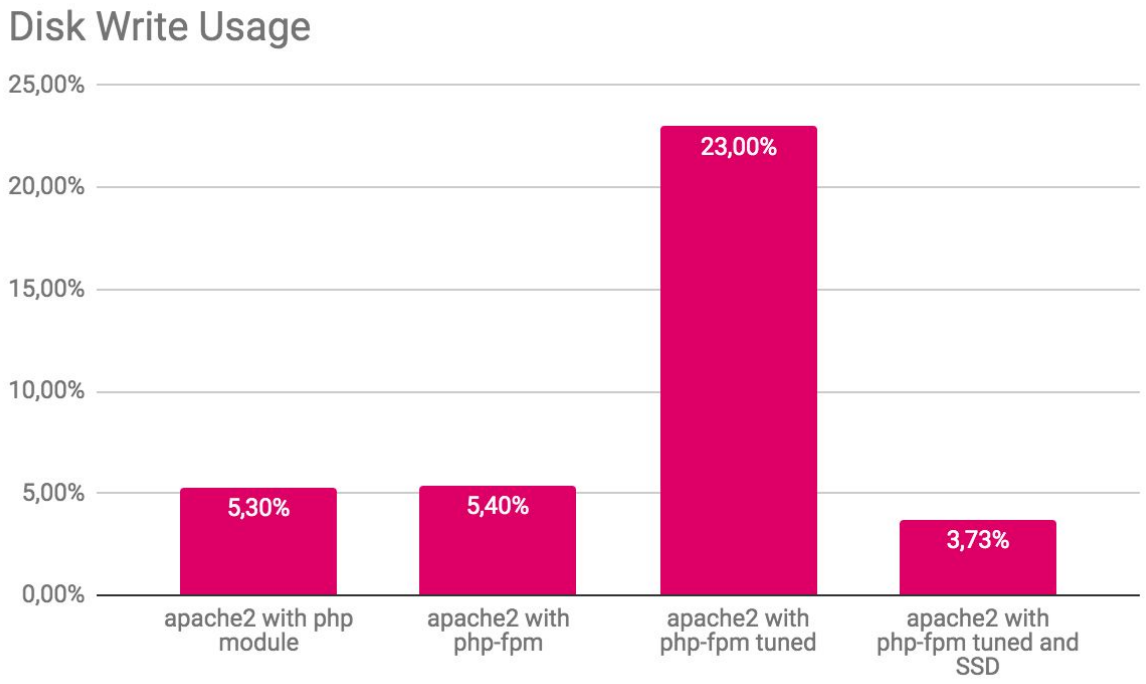
As google compute documentation puts it: *"SSD persistent disks are suited for enterprise applications and high-performance database needs that require lower latency and more IOPs than standard persistent disks provide"*. With both our database and application stored on the same server and disk, it makes perfect sense to make use of it.
And here are the results:

Configuration		**Result**
Server	**Set-up**	**Max Visitors Per Hour**
1 vCPU/ 3.75Gb RAM disk: standard	- Apache2 + php-fpm tuned - MySQL tuned	740
1 vCPU/ 3.75Gb RAM **disk: SSD**	- Apache2 + php-fpm tuned - MySQL tuned	900

We could have expected a bigger gain when moving to SSD but keep in mind that most of the configuration optimizations performed at the last step were all about caching and buffering. Also, we saw that we were *nearing* the disk capacity limit, not reaching it. With that in mind, it's still a nice performance gap: as we can see, it provides even more room to the CPU by spending even less resources time to other tasks (such as disks and IO operations) to manage the application's load:



Also, it's interesting to see that with SSD disk, we see a large decrease in disk writes operations usage:

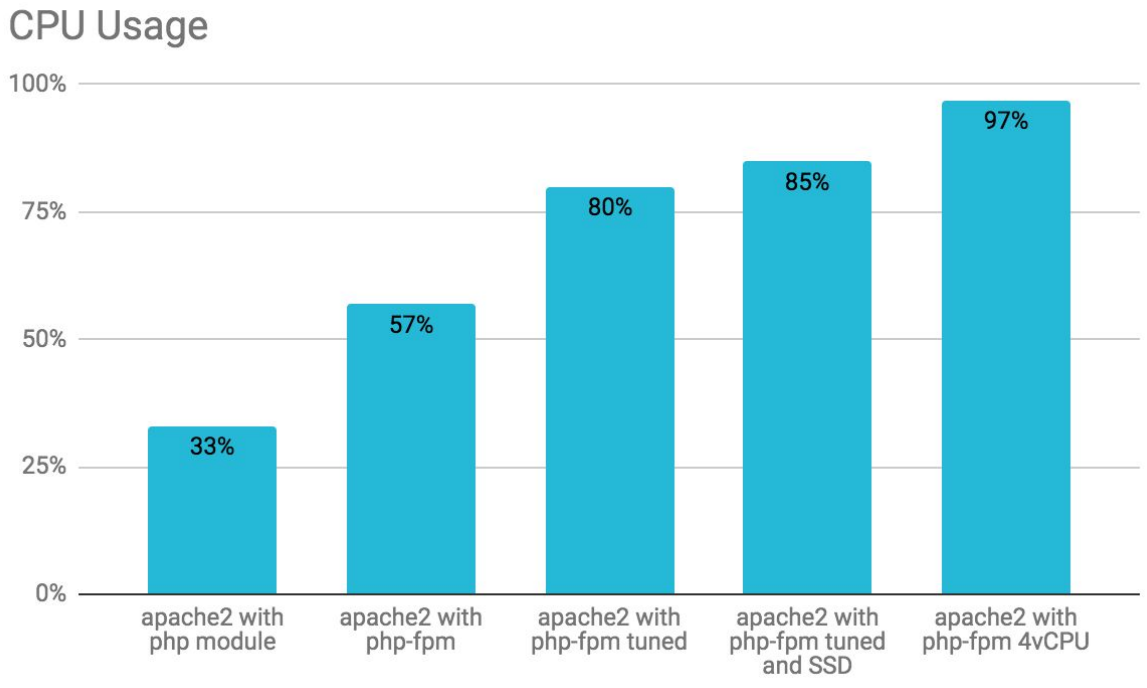


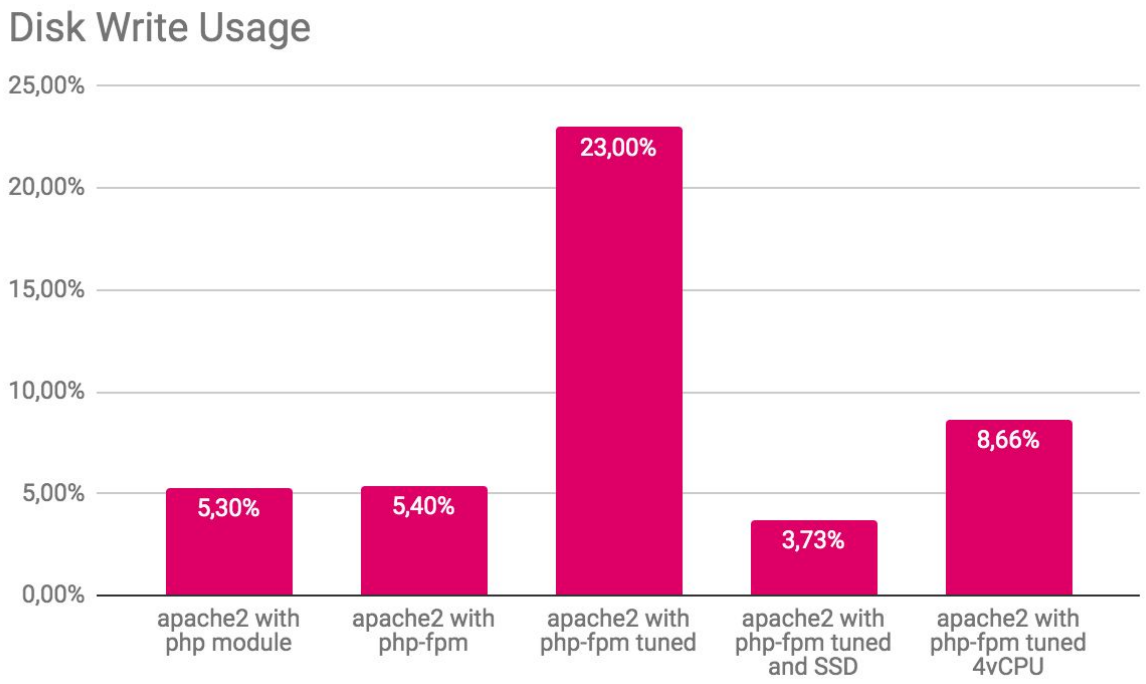
Please advise it's not because we are performing less writes, but that SSD disks can manage far more write operations than standard disks.

v. Single server with php-fpm and tuning and SSD disks scaled up !

And now we scale up the server's resources by 4 for both RAM and CPU !
Can our beloved PrestaShop application manage that much powerhorse ?
We think it can, but see for yourself:

Configuration		**Result**
Server	**Set-up**	**Max Visitors Per Hour**
1 vCPU/ 3.75 Gb RAM disk: SSD	- php-fpm tuned - MySQL tuned	900
4 vCPU/ 15 Gb RAM disk: SSD	- php-fpm tuned - MySQL tuned	2810





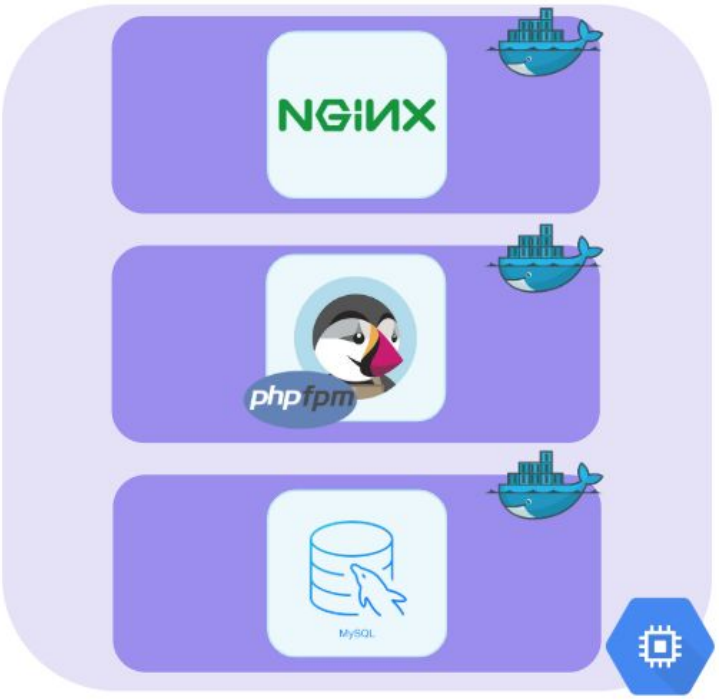
Scale up expectations
There is often an understanding that doubling the instance's resources (CPU and RAM) should double up the application's results. In real life, that's not that easy, for several reasons :

- Orchestration, run queues and context switching, almost invisible, take some kernel resources and time
- Bottlenecks may not be CPU or RAM. It *may appear* as CPU in your monitoring but that could indicate that CPU time is *spent waiting* for other resources (network, disks, and so on)

Cloud scale up
A fact that is not that well known about scaling up instances with cloud computing solutions, is that you not only add CPU and/or RAM but potentially less obvious resources: IOs, network bandwidth and so on.
Which is always a good thing to have more of.

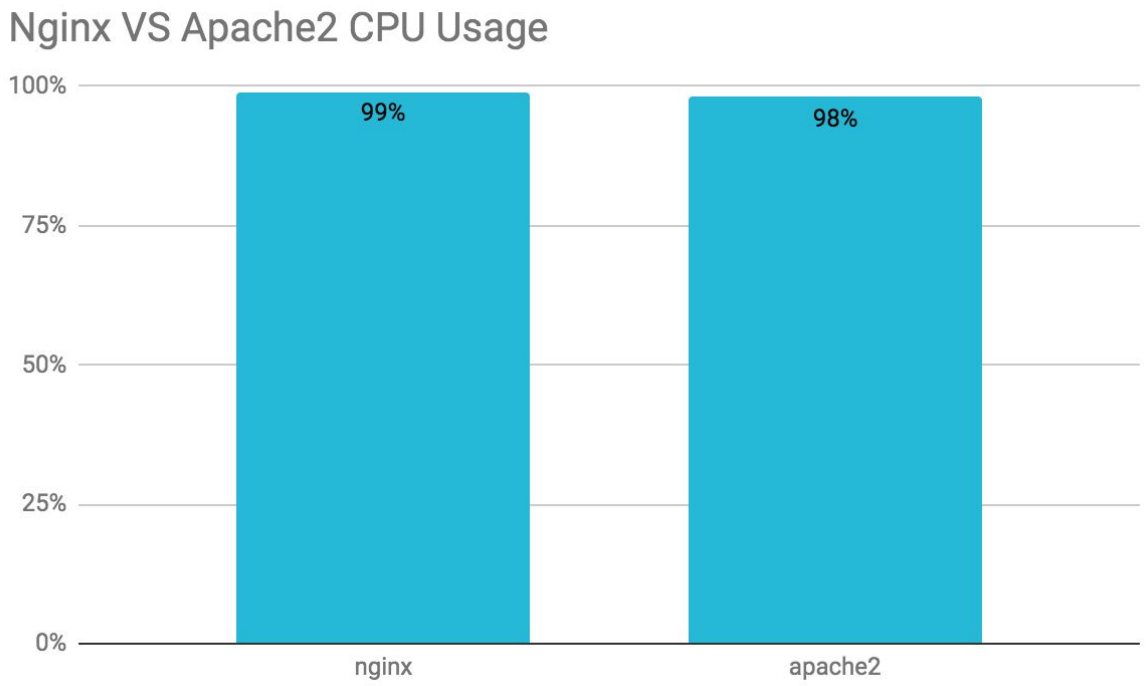
vi. Single server with nginx php-fpm and tuning and SSD disks scaled up !

We often hear debates about nginx performing better than apache2 and vice versa.
Though we have our idea about it, we thought it would be interesting to compare them and check it for ourselves.
Here is the stack we used, the exact same as previous test, just replacing apache2 with nginx:



Configuration		Result
Server	**Set-up**	**Max Visitors Per Hour**
4 vCPU/ 15 Gb RAM disk: SSD	- **apache2+php-fpm tuned** - MySQL tuned	2810
4 vCPU/ 15 Gb RAM disk: SSD	- **nginx+php-fpm tuned** - MySQL tuned	2390

Which is quite good for itself, but not exactly what apache2 provides.



Our take on Apache and Nginx
Both are top notch tools, though their inner workings are very different.
Long story short, nginx is better at handling static content and apache2 is better at handling dynamic content.
The asynchronous design of nginx adds some latency when rendering dynamic pages (event loop time) hence slightly increasing response times.
On the other hand, this asynchronous design allows it to manage higher loads and better handle static contents (such as images, css or javascript files).
Hence, there is not really a better solution than the other, it really depends on your expectations and use cases.
But nothing prevents you from using both of them, aside from a tiny technical complexity build up.

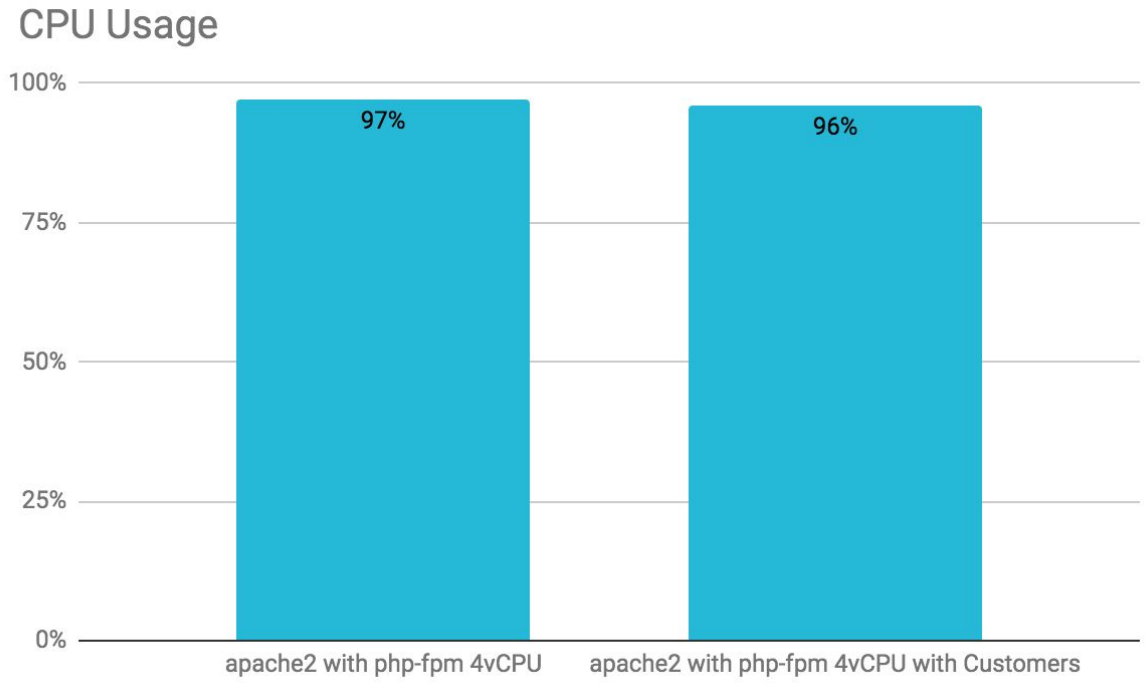
vii. Single server tuned and scaled with customers

And now, our final test for this architecture : mixing Visitors and Customers and observing what happens in this situation.

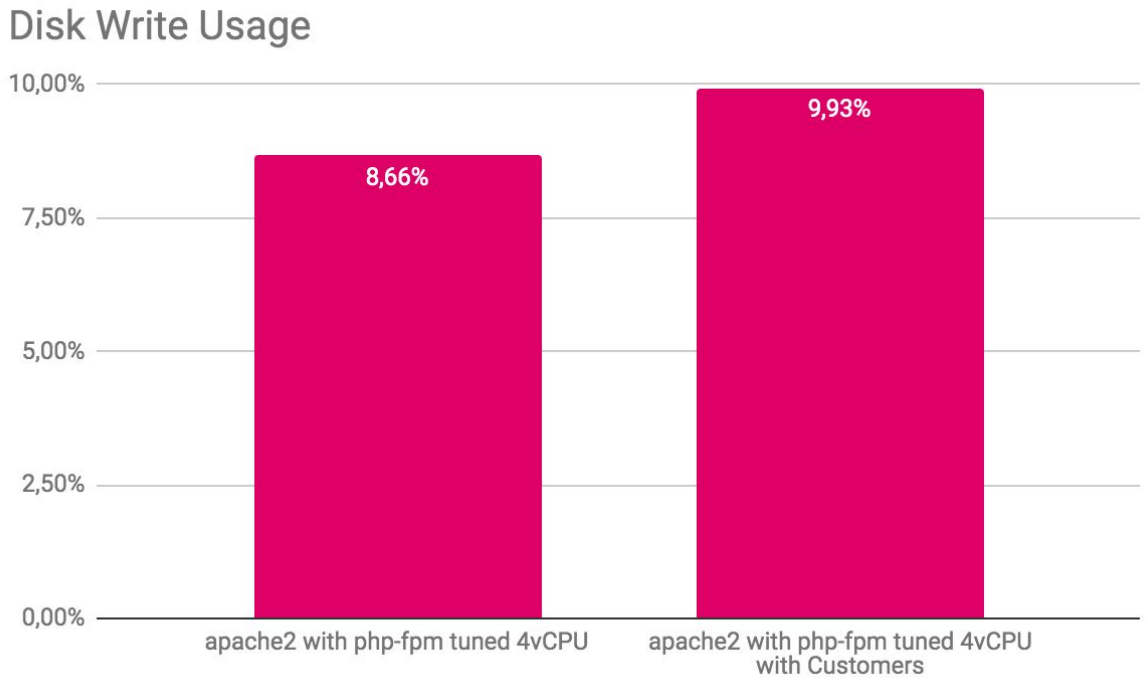
Configuration		**Result**	
Server	**Set-up**	**Max Visitors Per Hour**	**Max Customers Per Hour**
4 vCPU/ 15 Gb RAM disk: SSD	- apache2+php-fpm tuned - MySQL tuned	2700	110

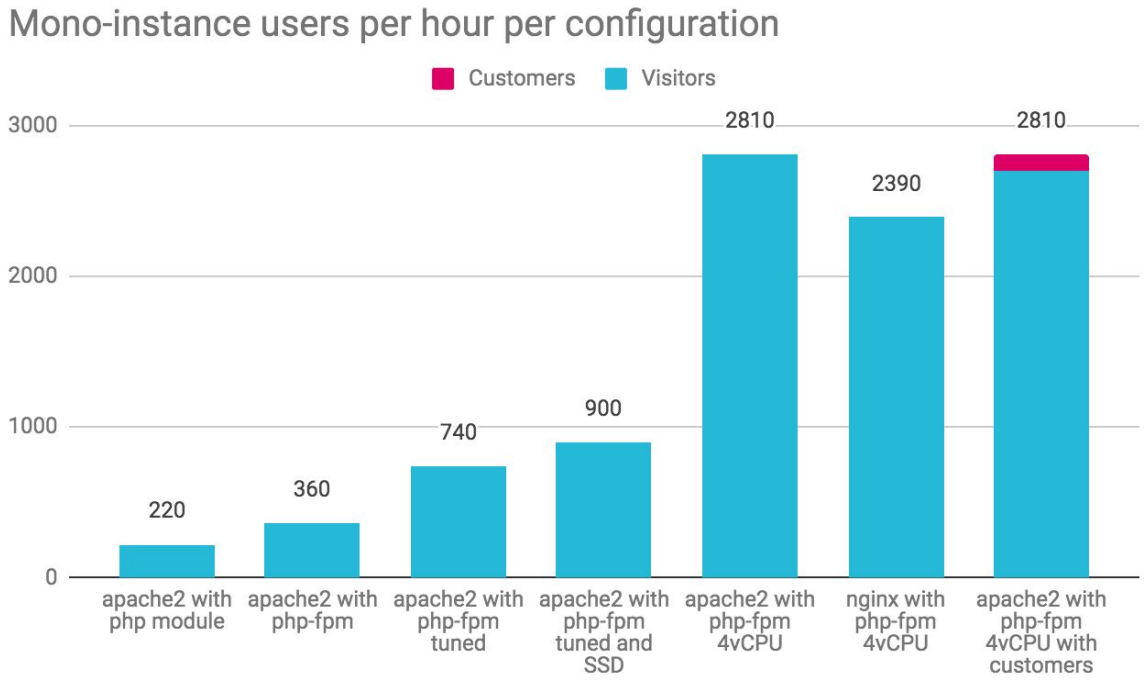
As you can see, we've decreased a tiny bit the Visitor Count, let us explain why:

- During the previous test, we were close to the maximum response time allowed for this benchmark - it does not mean we've reached the system's limit, far from it, just the test's parameters
- So, if we just added some Customers, we would have crossed the 300ms threshold for sure
- The accepted ratio between Visitors and Customers being 4%, we added 110 Customers (2740 x 0.04 = 109.6)
- Hence, we removed 110 Visitors and added 110 Customers
- As you can see, for this given test and environment, we have an interesting ratio of almost 1 Customer per Visitor - we'll see later if that concurs in other scenarii



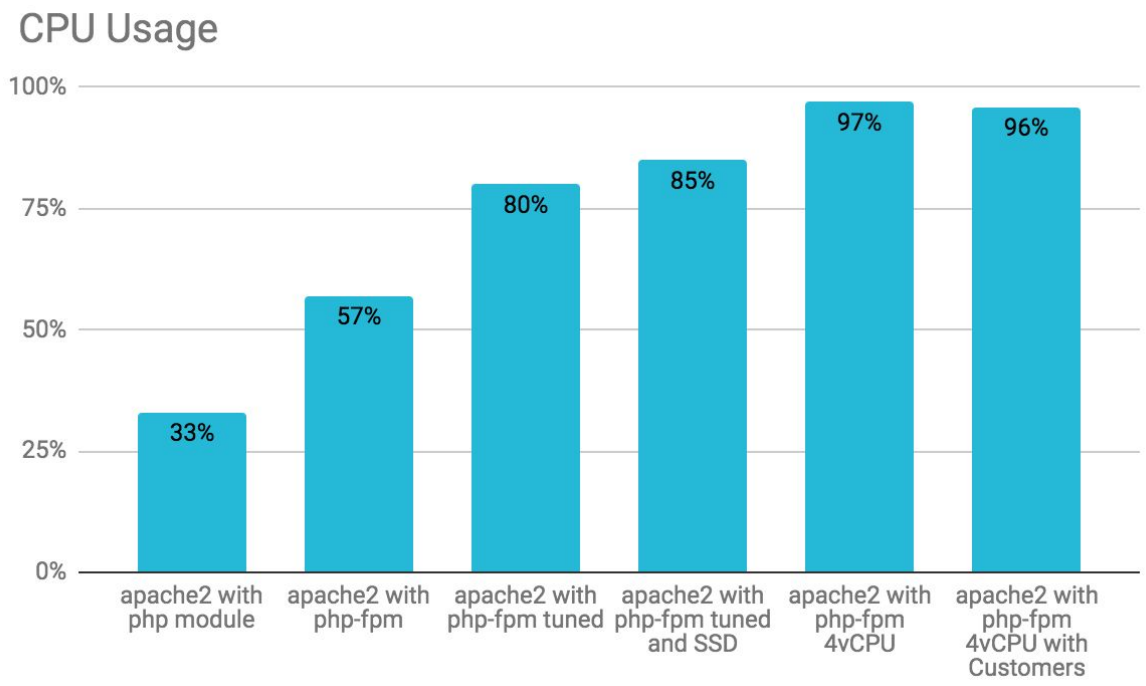
It is interesting to see a slight increase of disk write operations. This is easily explained as Customer operations involve more DB writes than Visitors' operations:




Mono-instance users per hour per configuration
Customers
Visitors
3000
2000
1000
0
220
360
740
900
2810
2390
2810
apache2 with php module
apache2 with php-fpm
apache2 with php-fpm tuned
apache2 with php-fpm tuned and SSD
apache2 with php-fpm 4vCPU
nginx with php-fpm 4vCPU
apache2 with php-fpm 4vCPU with customers

Reaching the limits

As you can see in this graphics, we are reaching the limits of this architecture:



Of course, the application can still work as expected and should not breakdown if the threshold is attained, but:

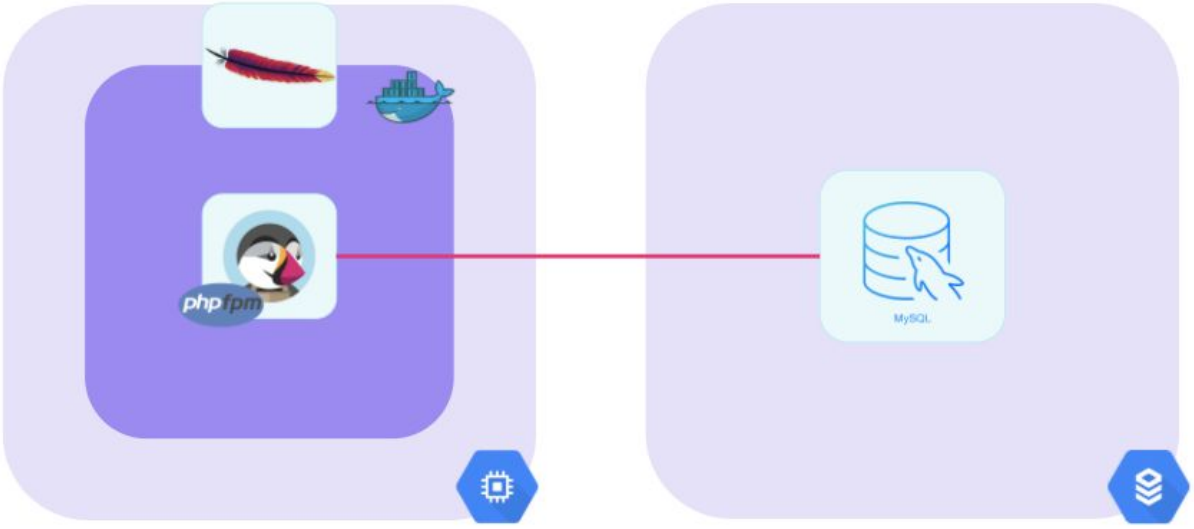
- Any additional load (either external from visitors or internal from employees) will considerably increase the response time
- Any burst of activity could be ill managed by the server, resulting in poor perceived performance by any visitor or customer

For your production environments, you'd rather want to remain under the 40% CPU Usage, allowing any peak of activity to be managed gracefully.

b. Externalized DB Architecture

In this architecture, we externalized the MySQL database to the Cloud SQL service, with dedicated resources.
What could be expected is that:

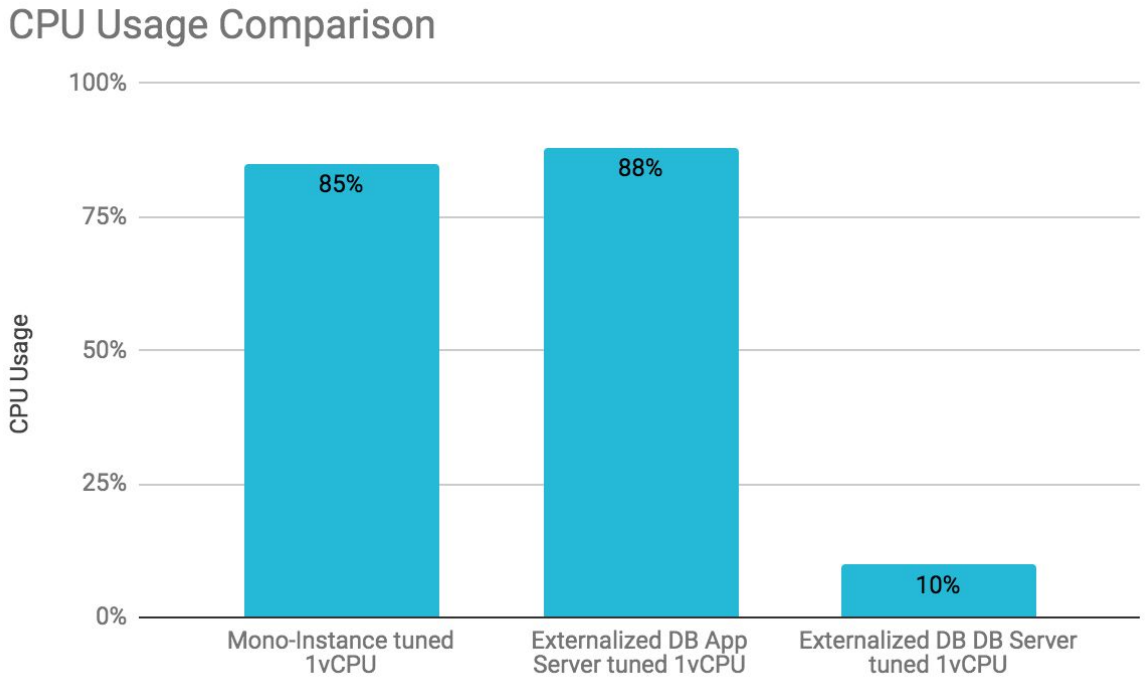
- More resources should be available to both the application and the DB
- Application and infrastructure management is easier (backups, migrations, etc..)



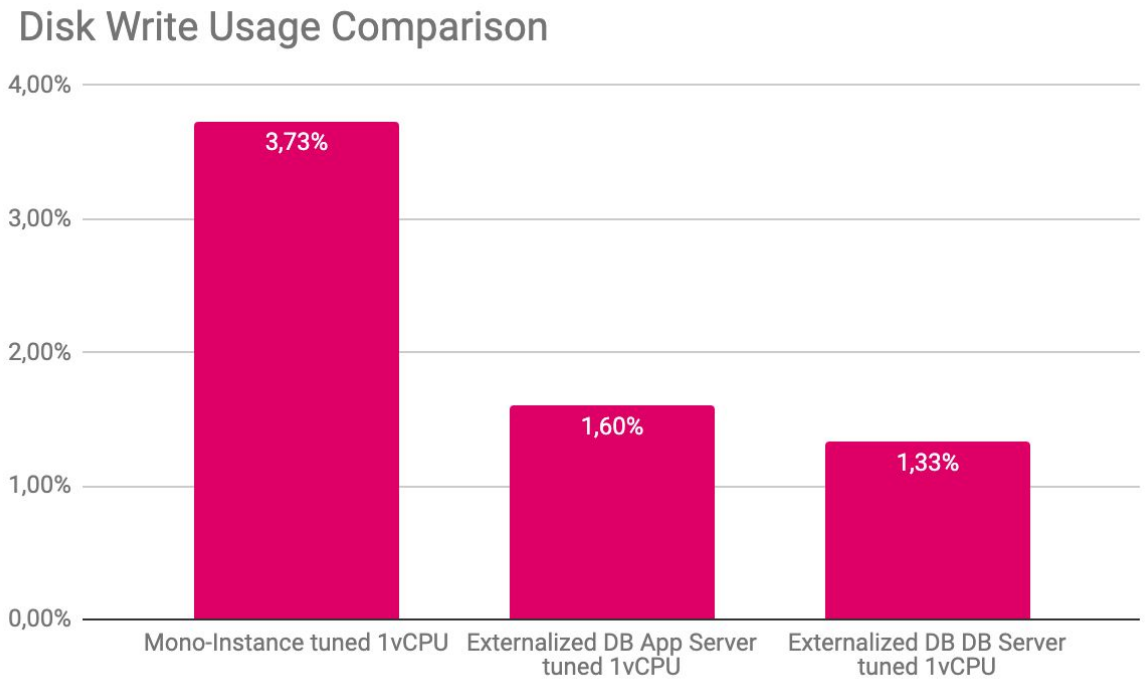
i. Externalized DB server with php-fpm and tuning and SSD disks

Since we've already experienced the different tuning configurations available to us, we directly start with them included in this architecture.
And here are the results:

Configuration		Result
Servers	**Set-up**	**Max Visitors Per Hour**
Web 1 vCPU/ 3.75 Gb RAM disk: SSD	- apache2+php-fpm tuned - MySQL tuned - **Externalized MySQL**	940
DB 1 vCPU/ 3.75 Gb RAM disk: SSD		
Web & DB 1 vCPU/ 3.75Gb RAM disk: SSD	- Apache2 + php-fpm tuned - MySQL tuned - **Mysql with Apache**	900



As you can see, the DB CPU usage is quite low.
Though, keep in mind that back office operations, not included in the current tests, can be very intense for the DB server, especially with some modules activated, which is another good reason to externalize the DB server.

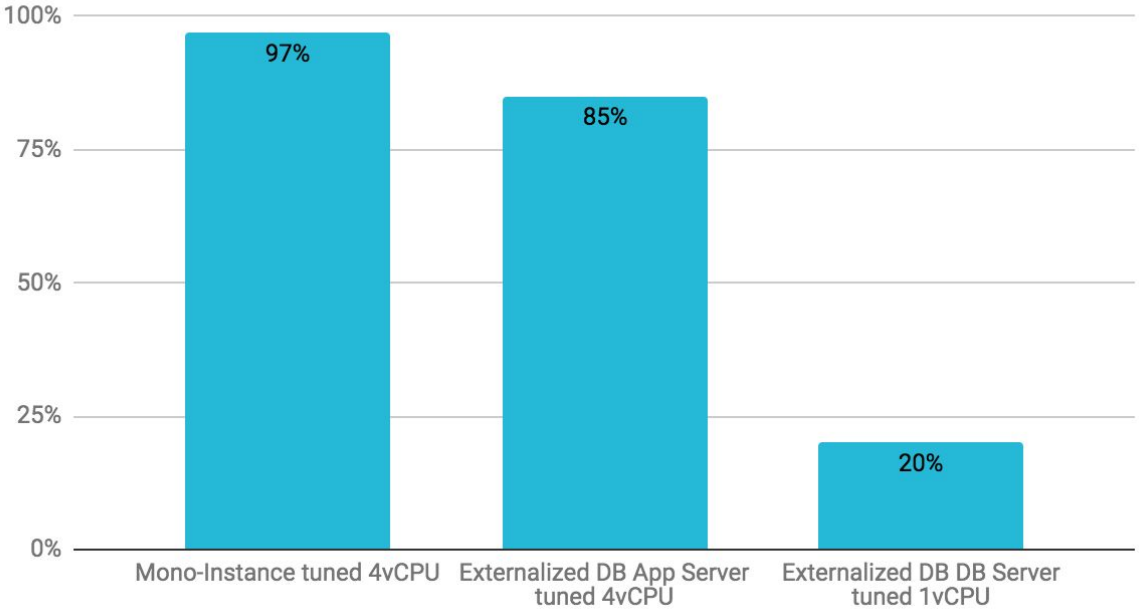


iii. Externalized DB server with php-fpm and tuning and SSD disks scaled up !

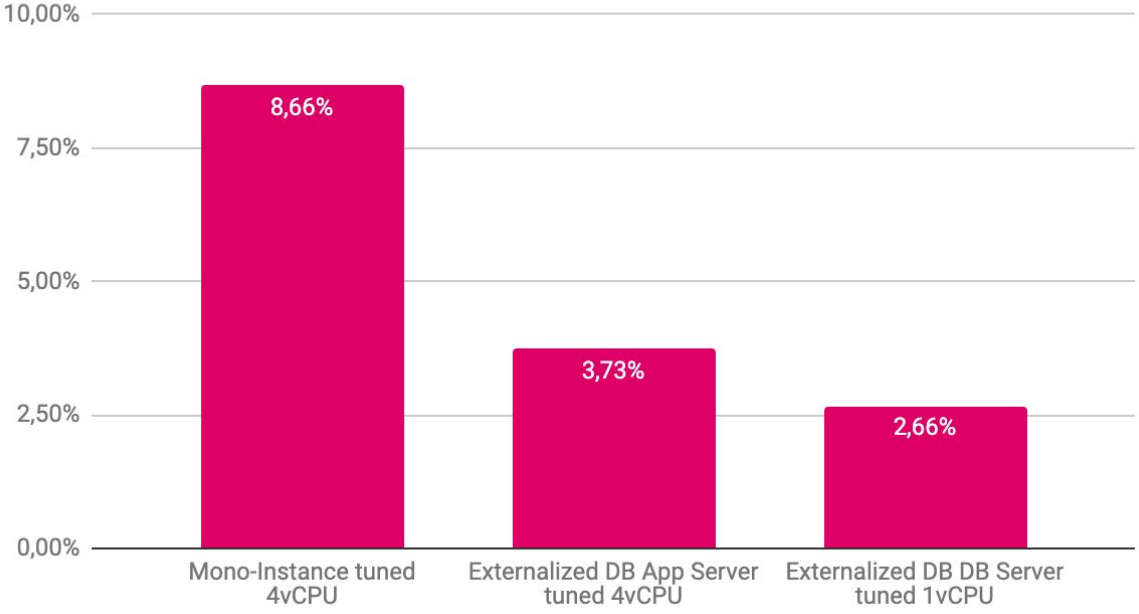
Here, we just scaled up the application's instance, quadrupling its RAM and CPU. As we've seen the DB resources are far from reaching their limits, we're not updating it, keeping its current setup:

<table>
<tr><th colspan="2">Configuration</th><th>Result</th></tr>
<tr><th>Servers</th><th>Set-up</th><th>Max visitor Count</th></tr>
<tr><td>Web
1 vCPU/
3.75 Gb RAM
disk: SSD</td><td rowspan="2">- php-fpm tuned
- Tuned externalized MySQL</td><td rowspan="2">900</td></tr>
<tr><td>DB
1 vCPU/
3.75 Gb RAM
disk: SSD</td></tr>
<tr><td>Web
4 vCPU/
15 Gb RAM
disk: SSD</td><td rowspan="2">- php-fpm tuned
- Tuned externalized MySQL</td><td rowspan="2">2550</td></tr>
<tr><td>DB
1 vCPU/
3.75 Gb RAM
disk: SSD</td></tr>
</table>

CPU Usage Comparison



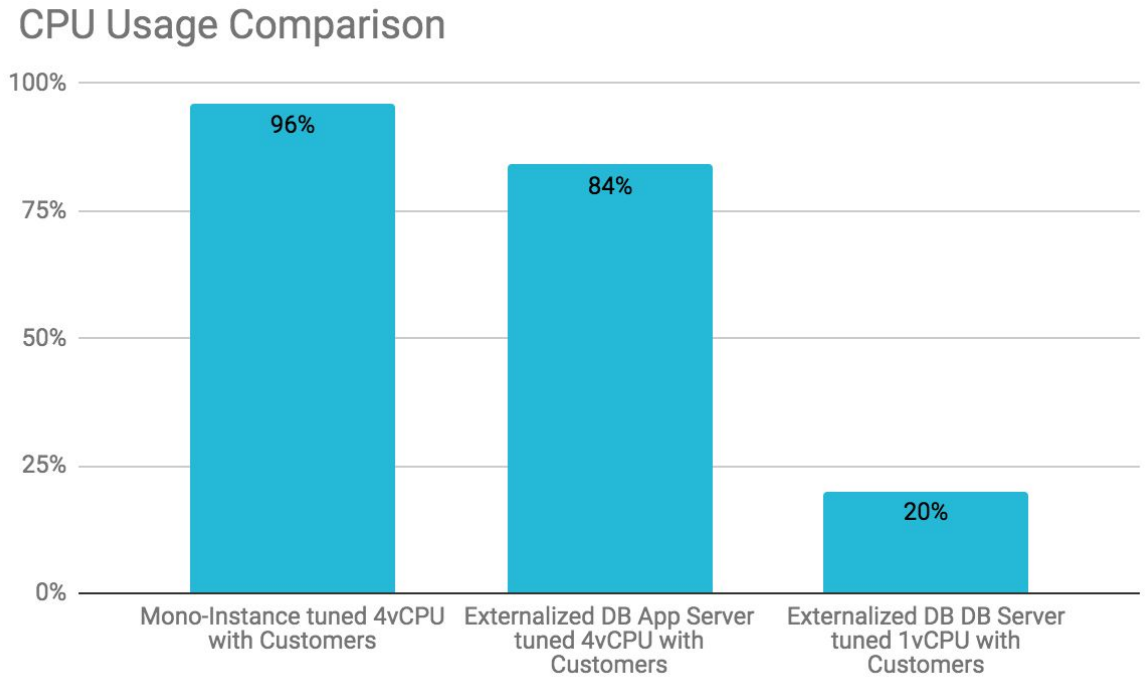
Disk Write Usage Comparison



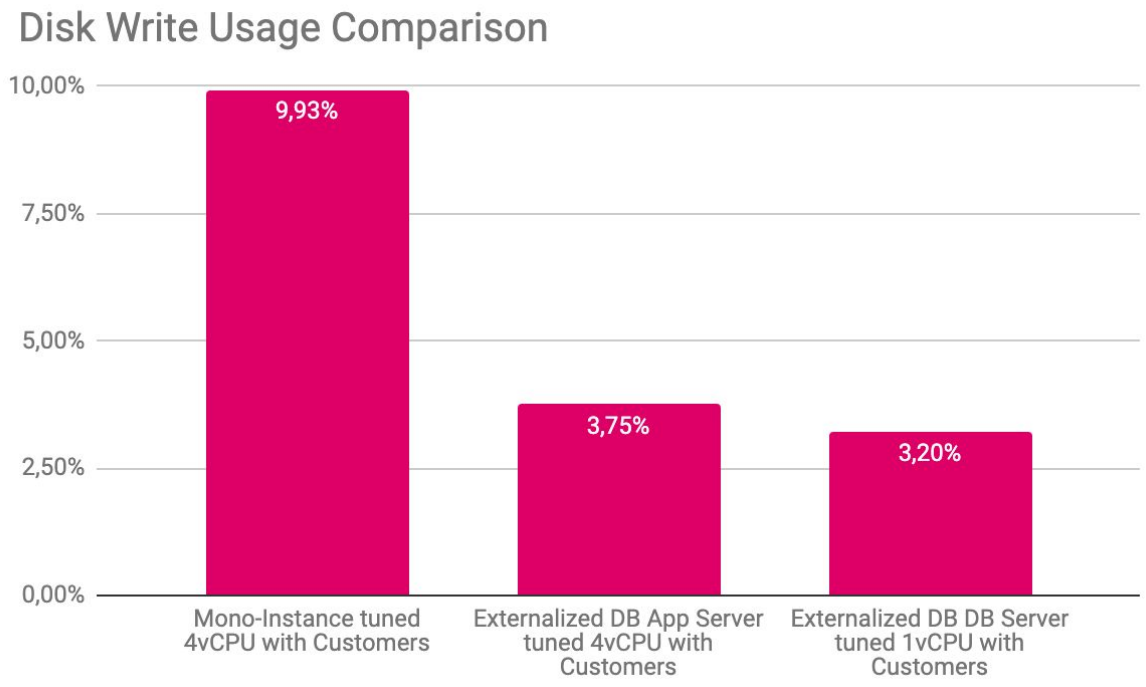
iv. Externalized DB server with php-fpm and tuning and SSD disks scaled up with Customers !

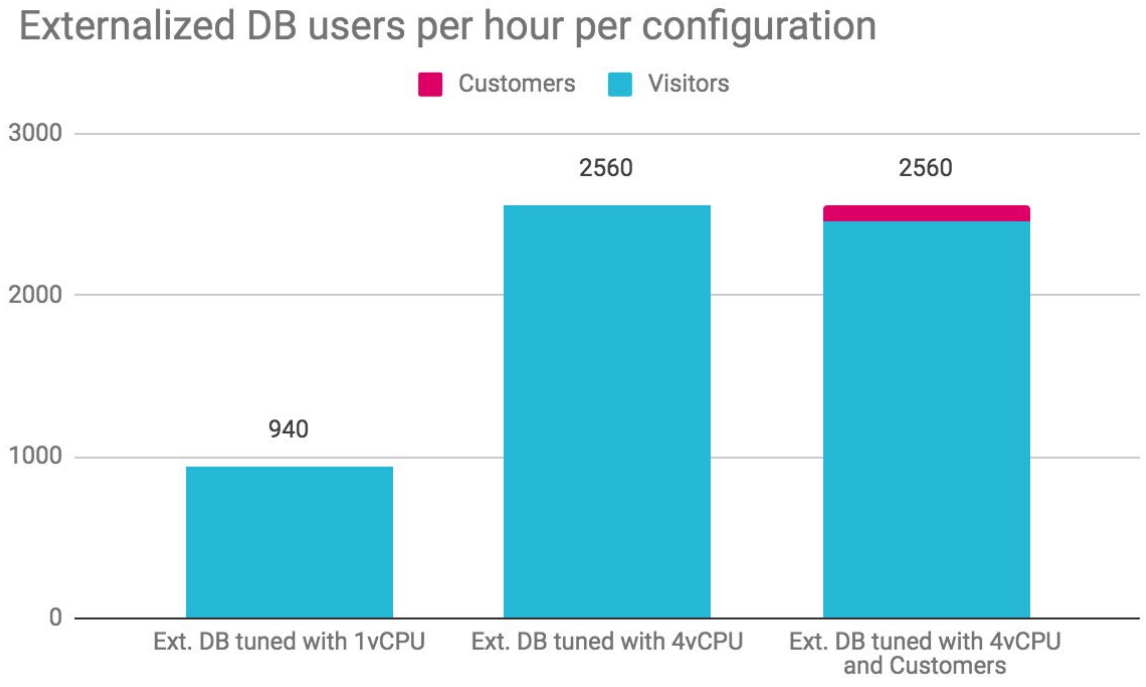
And now we add some Customers to the pools of users and see how the stack reacts:

Configuration		Result	
Server	**Set-up**	**Max Visitors Per Hour**	**Max Customers Per Hour**
Web 4 vCPU/ 15 Gb RAM disk: SSD	- php-fpm tuned - Tuned externalized MySQL	2410	100
DB 1 vCPU/ 3.75 Gb RAM disk: SSD			



Again, a slight increase of disk write operation on the DB side, explained by the Customers scenario involving more writes:

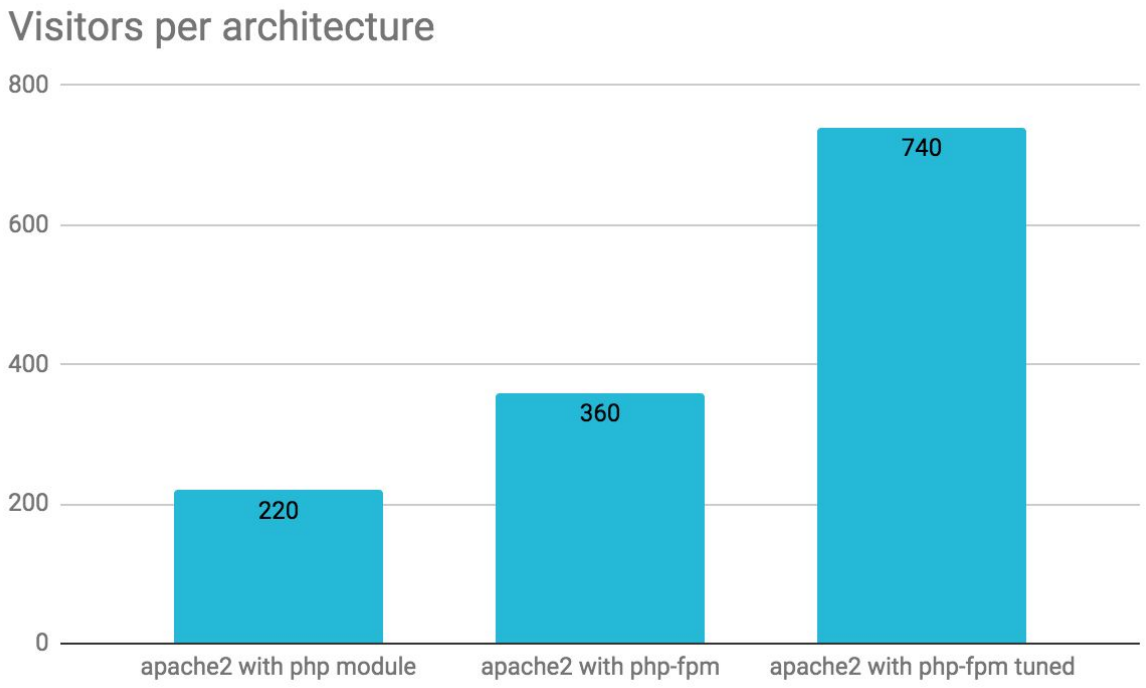




1. Insights

Now that we've tested the PrestaShop application with several configurations and architectures, here are some insights that we would like to share with you:

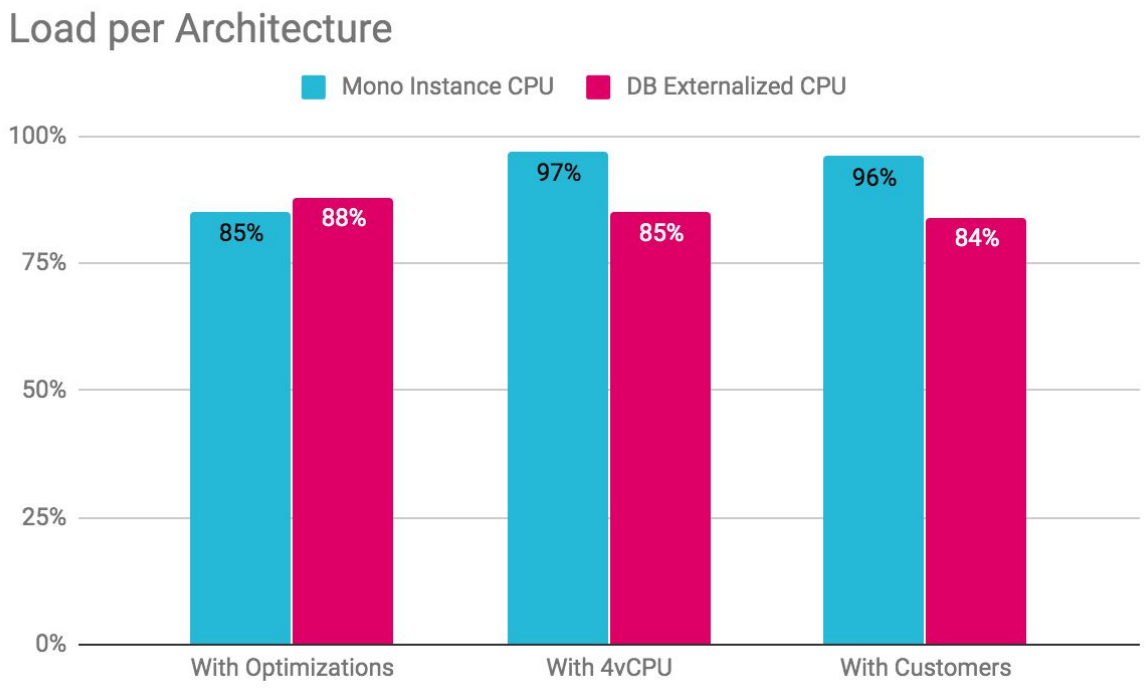
- Even for a mono-instance architecture, the configuration can make a huge difference. It may seem obvious for most, but it's still something that needs to be said and said again. Here, almost 200% visitors per hour can be gained with some love and configuration.



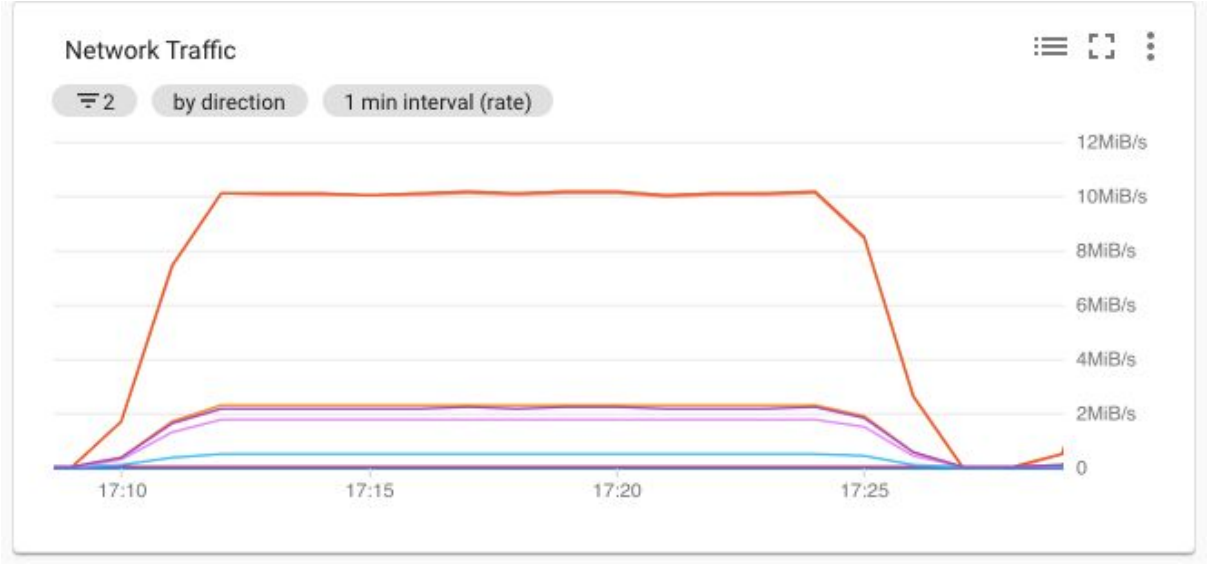
- We apparently did not see much improvements by externalizing the DB during the tests, which can be a fair assessment. Though, this behaviour can be expected by any horizontal scaling, hence adding network overhead to the platform's architecture.

  Also, by looking at the servers load and their resources usage, we can assume that:
  - The mono-instance architecture reached its limit sooner, mostly CPU
  - In a real world scenario, the externalized architecture would better handle any traffic surge - it can be appreciated by disk usage that is lower when looking at the externalized DB metrics
  - It is *always* good practice to isolate elements: this way an application load won't impact the DB and vice versa

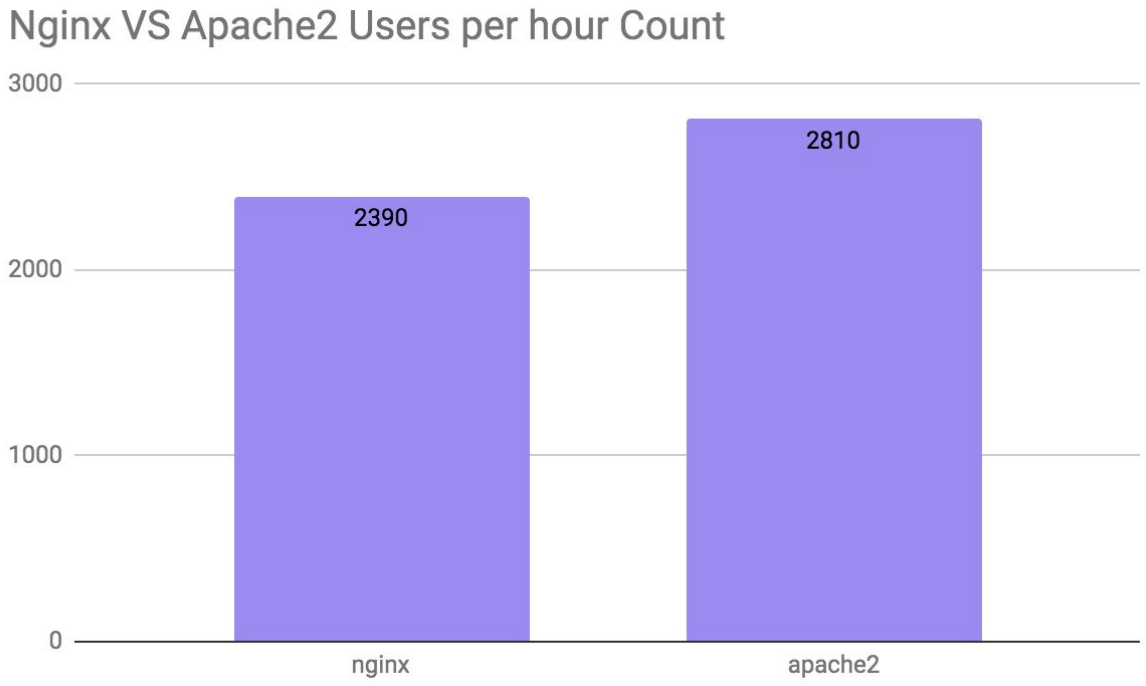
- Which includes, as already mentioned, some backoffice operations that can be pretty heavy on the database

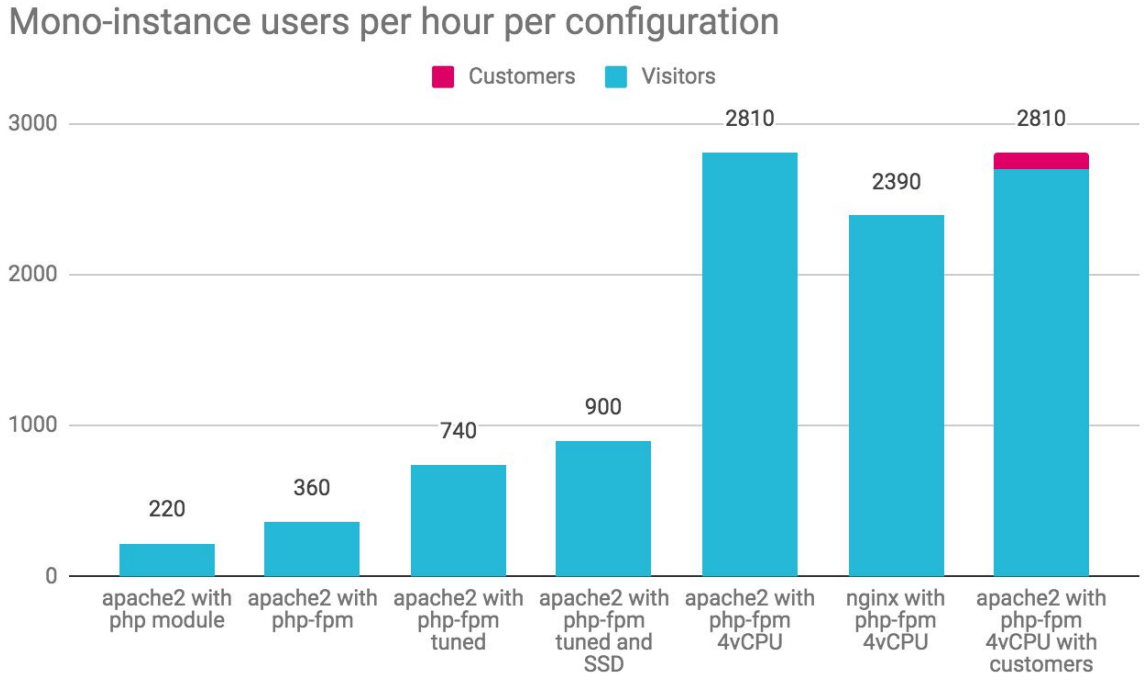


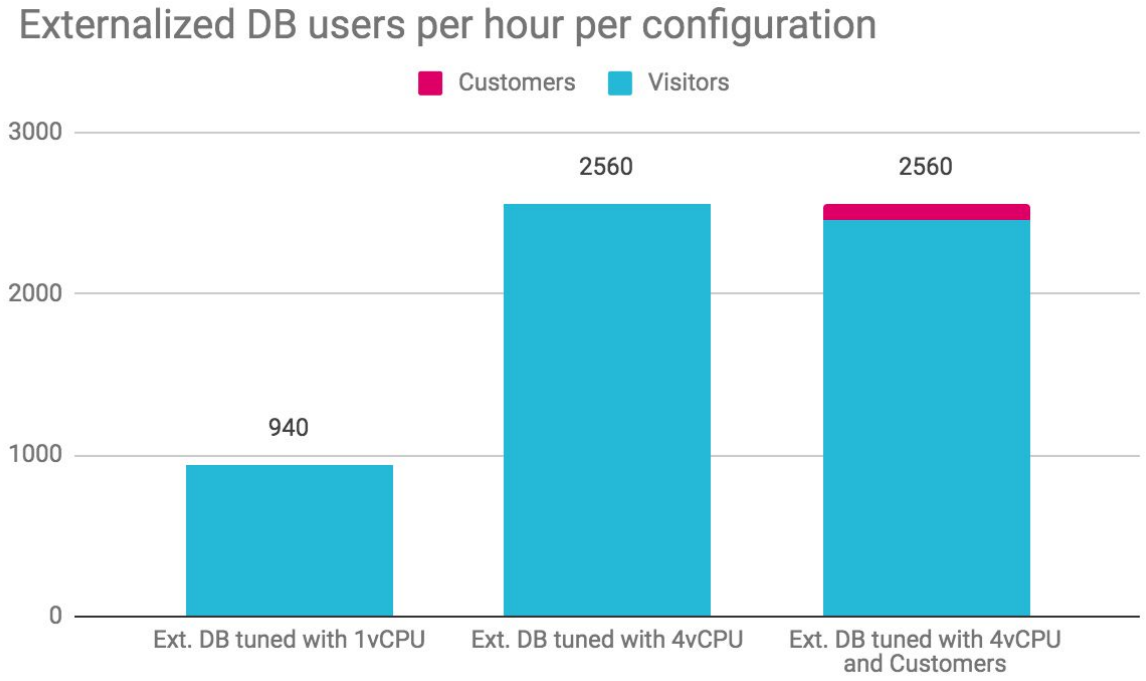
- As we have seen during this tests series, network load is NOT to be underestimated when working on your architecture, and even a good network capacity won't prevent you from performance loss:



- Another insight is that, though very different in their designs, *in our case scenario*, apache2 is more suitable than nginx:






Externalized DB users per hour per configuration
Customers
Visitors
3000
2000
1000
0
940
2560
2560
Ext. DB tuned with 1vCPU
Ext. DB tuned with 4vCPU
Ext. DB tuned with 4vCPU and Customers

2. Recommendations

We have provided you with plenty of data and it's up to you to decide which ones are relevant to your case.
Still, we have a few words left to share with you about what we consider to be best practices.

First, a good server configuration will bring you plenty of performances. We will keep sharing them with you through our devdocs website.

Also, yes, the mono-instance is *very* efficient.
And yes, the mono-instance is *very* easy to set up.

But there are plenty of areas where the mono-instance architecture won't fit your needs and those performance tests won't show them to you.

First, the mono-instance will NOT manage high availability: having only one instance of your beloved PrestaShop, any hardware failure, server restart or any configuration change can generate downtime, which can be perilous for your business.

Also, speaking of performance, you need dedicated instances to fine tune your application : apache does not require the same configuration as MySQL (overcommit, queue schedulers, etc..). This kind of fine tuning can only be achieved by separating your application's part to dedicated servers.

So, though a little less efficient (which, again, is to be expected by adding servers and network overhead), we *do* recommend separating applications' parts to different servers.

This will also provide you a better platform management, either by managing backups efficiently, adding MySQL slaves for safety, and distributing the resources were needed - maybe your database does not need as much CPU as your frontend, or maybe it requires more, depending on your needs, but that's something you can manage only when splitting your architecture.

And this *will* prepare you for high availability configurations that will become vital when your store grows and you require better uptime and better response time. Concepts that we hope you'll become familiar with and that we will be presenting to you in future documents if that's not already the case.

Finally, it will allow you to better manage your visitors and customers in the end, either bursts and surges or just some backoffice SQL requests that will definitely less impact your frontend activity.

3.Next

The configuration and infrastructure improvements that were presented in this white paper are fairly simple to implement and can provide interesting performance gains.
In a next white paper we will explore more advanced set-ups that could prove appropriate if you are already reaching the limits of those ones.

Caching: use of a reverse proxy such as varnish to vastly improve read performances.
Horizontal scaling: setting up more than one application server behind a load balancer to improve performance and availability.

Annexes

External Resources

PrestaShop

Of course, we couldn't have a resources section without mentioning the PrestaShop's github repository ! This is where most of the magic is happening and where most of the people involved in the project can be found, either to ask for their help or provide them with some :
https://github.com/PrestaShop/PrestaShop

PrestaShop Devdocs

You will find plenty of useful resources here, ranging from developing the solution to hosting it and even testing it.
Do not hesitate to pay a visit, there are good chances you'll find something you were looking for : http://devdocs.prestashop.com/

PrestaShop Docker

This is where the PrestaShop docker's images sources are worked on, to then be made available from docker hub. The same ones we used for this paper.
https://github.com/PrestaShop/docker
https://hub.docker.com/r/prestashop/prestashop

PrestaShop Docker Templates

A small repository where we provide some templates and architecture's solution for your docker hosted PrestaShop. Same ones that we used here.
https://github.com/PrestaShop/docker-templates

Prestashop Performance Project

Last but not least, a set of tests to benchmark your PrestaShop instance from docker image building with products to run the gatling tests. Same one as we used here.
https://github.com/PrestaShop/performance-project

Shop Hosting

Here are some details about the hosting solutions we used during those tests.

Cloud hosting

Cloud infrastructures provide out-of-the-box tools for applications deployment and monitoring. The overhead of virtualisation is an accepted cost to get easy-to-change infrastructure and play those scenarii fast enough.

Cloud SQL

As google puts it, "*Cloud SQL is a fully-managed database service that makes it easy to set up, maintain, manage, and administer your relational databases on Google Cloud Platform.*"
We are using it for the Externalized DB architecture.

Docker

In the same way, using docker allows reusable configurations, not only defining the code (shop installation, fixtures, etc...) but also the softwares (php version and configuration, apache and php-fpm, etc...).

Configurations tuning

In order for you to either reproduce our tests or get inspiration from our configurations, we are providing them to you as we used them.
Please advise those configuration parameters may not be suitable for your environment as is and may require some extra work for a real production environment.
For example: setting **opcache *validate_timestamps*** to **0** may not suit you as any application code change would require your webserver to be restarted to be taken into account.

Also, we're fully aware we could have adapted the configurations for each setup and go even further into the tuning part (such as adjusting php-fpm's process manager, tuning the operating system's scheduler, the IO scheduler, etc..), but:

- It is not our intention to go *this* far in *this* document
- It would have increased the number of test cases
- Doing this, it would have been more difficult to compare each solution between them
- Those configuration options rarely are available in most hosting solutions

PHP Tuning

As we have seen, tuning PHP is very important for your application's performance, whether you're running PrestaShop or any other PHP software. Especially if you're dealing with filesystem performance and NFS.
Here are the options we tweaked:

Realpath
Each time a file is included to the executed code, PHP needs to look it up and parse it, which can rapidly become heavy for the filesystem (and not even count as IO, it's just lookup), performing *lstat* commands continuously.
So, activating the realpath caching is *always* a good idea, especially with a shared filesystem (such as NFS).

```
> realpath_cache_size = 4096k
> realpath_cache_ttl = 600
```

Memory
Though PrestaShop is not that heavy on memory consumption, some backoffice pages can require more memory. As it is recommended by default, we're doing it with pleasure.

```
> memory_limit = 512M
```

OPCache

As you may know, PHP is not a compiled language, meaning every time you execute it, the system compiles it to bytecode for execution. And here comes OPCache !

It compiles and stores the code into live memory in order to improve execution time.

And it works greatly, as you may have seen it.

Just keep in mind that some configuration may require you to change the way you do things usually in your own environment. For example, ***opcache.validate_timestamps=0*** means OPCache will never update your code *unless* you let it know explicitly (either through internal functions or by restarting the webserver).

```
> opcache.enable=1
> opcache.enable_cli=0
> opcache.memory_consumption=256
> opcache.interned_strings_buffer=32
> opcache.max_accelerated_files=16229
> opcache.max_wasted_percentage=10
> opcache.validate_timestamps=0
> opcache.revalidate_path=0
> opcache.fast_shutdown=1
> opcache.enable_file_override=0
> opcache.max_file_size=0
```

MySQL Tuning

Though we're discussing it second, MySQL tuning is just as important.

Our intention here was to optimize the instance's throughput by adding caches. As for PHP, it allows the service to work as much as possible in memory and avoid disk accesses, hence reducing latency.

Caching

As mentioned, those parameters allow better cache information for further reuse, first by enabling it, then by increasing its sizes. Again, the idea is to keep the query results in memory rather than looking them up to the (higher latency) hard drive. As always, those values should be adapted to your own environment, you probably won't need a host_cache_size of 10000.

```
> query_cache_limit = 128K
> query_cache_size = 32M
> query_cache_type = ON
> table_open_cache = 1000
> thread_cache_size = 80
> host_cache_size=10000
```

Buffering
Buffering is almost another word for caching. So we work here with the memory area that holds cached data for InnoDB tables, indexes, and other auxiliary buffers, etc..
Again, those options should be adapted to your shop. We set-up the innodb_buffer_pool_size to 3G just to make sure the whole stack would be cached in memory, but make sure you have enough memory according to the value you set up.

```
> read_buffer_size = 2M
> read_rnd_buffer_size = 1M
> join_buffer_size = 2M
> sort_buffer_size = 2M
> innodb_buffer_pool_size = 3G
```

Other parameters
Some other parameters to increase the MySQL performance, such as disabling the performance schema (used for monitoring), memory tables and enhancing GROUP BY queries.

```
> performance_schema = OFF
> max_heap_table_size = 32M
> tmp_table_size = 32M
```

WWW.PRESTASHOP.COM