codesphere

Who enjoys waiting

DEEP DIVE PAGE SPEED FOR ONLINE BUSINESSES

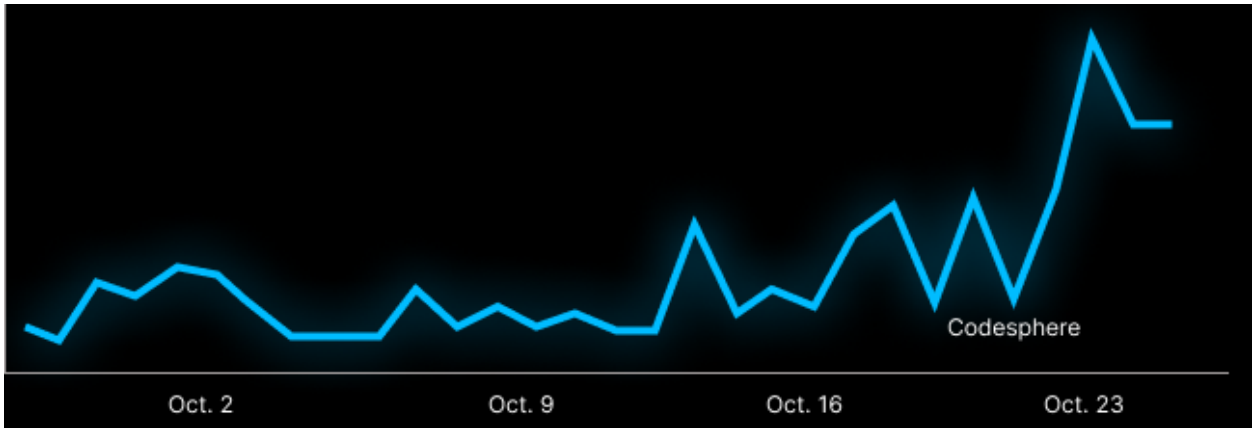
Abstract:
Page speed is key for any online business, this whitepaper explains in detail what kind of issues arise from slow page speed, where these issues originate technically and what possible solutions entail. We focus on e-commerce applications because the issues are most relevant for them and the potential upsides of improving speed are most noticeable and directly measurable. However, the issues discussed here also apply to other online businesses.
Many of these issues cannot be addressed by typical (single server) hosting providers. The underlying technology frameworks have limitations on how many requests can be handled in parallel. The solution is seemingly simple but has been hard to implement until recently. Load balancing, sometimes also called multi-server hosting, addresses the issue by dividing incoming requests and sending portions of the traffic to different servers. Setting your infrastructure up this way used to require technical infrastructure expertise usually handled by DevOps teams which was not readily available or affordable for a majority of e-commerce businesses. With the advancement of accessible cloud providers this is now in reach for anyone.

Problem outline:
PHP based applications still make up a majority of all websites out there (77.4% in 2022 [1]). PHP really only scales with more hardware - the number of server cores directly determines how many processes your server can handle in parallel. With a four core processor on your server your PHP site can run exactly four processes in parallel. Typically for an e-commerce application a single user visiting the website already triggers multiple requests, the frontend call to your html and css files and on top any number of additional tasks i.e. tracking and analytics, special offer pop ups or chatbot widgets.
If all cores are currently processing requests (they are “busy”), additional requests get queued in line and processed once a core becomes available. How long your server takes to process each request is determined by the CPU frequency and other limiting factors like write/read speed of the underlying disk space. So in theory there are at least two types of speed issues, the first type being the servers take too long to process requests (either because the server’s CPU is insufficient for the tasks or the tasks are overly complicated). The other type of issue arises from load, meaning there are a lot of requests in parallel and the server's capacity limitations lead to

requests being queued. The real life illustration of this would be a packed bakery on a Sunday morning, depending on the number of people in line in front of you and how long each customer takes to finish their order the wait time will differ. We will keep using this bakery analogy throughout the paper, so keep this setting in mind.

Example numbers:
Let's take a look at what this queuing behavior actually means for customers arriving at any webshop. Say you are hosting your shop on a typical 4 core (v)CPU with 1-2 GB of RAM server and per user arriving on the page there are four processes triggered. In bakery terms this means there are four (average speed) cashiers working and every customer wants four interactions initially. On a website of course once the user starts navigating around your page that triggers additional requests, but for simplicity let's put that aside. If there is one user only the initial requests take around 50 milliseconds to finish. The second user arriving already waits 100 milliseconds, 50 ms for the requests triggered by the first user and another 50 ms for his own requests. If there are 100 users arriving at the same time, user number 100 already waits 5 full seconds for the page to load. The users don't necessarily need to arrive exactly at the same time either, as long as there is a queue their request will wait for a core to become available. Users continuing to interact with the server after the page has loaded of course worsens the issue and lowers the number of parallel arrivals needed to create a lag in load time.

Is a page load of a few seconds really that bad?
Well it depends of course a bit on the specific case, for example users might be fine waiting a few seconds for something major to happen in the background but an initial page load for an e-commerce website is not one of these occasions.
Studies show that the first five seconds of page load have the highest impact on conversion rates [2]. The numbers suggest that an e-commerce page with a 1 second load time has a 2.5x higher conversion rate than one that has a load time of 5 seconds. That has real direct financial implications for any business.
There are other factors which might indirectly affect e-commerce KPIs, Google for example has been known to use page load speed as a direct factor when ranking the search results. Recently they also outlined their practice to account for page load speed with any ads run via their platform [3] other major platforms like Meta also rank ads on the perceived quality. Page load speed therefore directly affects your return on ad spend (ROAS). We have seen this for our clients as well, switching to modern hosting basically increased their return on ad spend overnight.



Going back to our bakery example we can illustrate another important issue. Waiting in line is usually not a great experience and if there is a comparable bakery around with consistently lower wait times more people tend to purchase there instead. This issue is most relevant during peak times, when lots of people want to go to a bakery, say Sunday morning or during a public holiday. You could say that wait times are to be expected then but from a business perspective these times are where a majority of the revenue comes from. Consistently long wait times will have a major impact on how much is sold during these hours. This is no different for an e-commerce business, the most prominent example would be Black Friday but even the daily peak times in the evenings follow this logic. If there is another store where I can get similar products with less waiting people will go there instead.

On mobile the issues tend to be even more pronounced as mobile network connections add to load times. Google estimates that 53% of mobile users leave a site when it takes more than 3 seconds to load. The average load time on mobile was a lot slower than this at around 15 seconds [4]. The good news is that in this crowd standing out positively is rather easy.

Improving your user's experience not only decreases initial bounce rates it also makes them much more likely to return and tell their friends about it. (“Hey Mike, did you try the new bakery that opened on the corner of main and university avenue? Their bread is delicious and they have literally no lines because of a walk in walk out store concept”)

Speed up your page load speed

Alright now that we have explored the benefits of faster page load speed, what can you do to improve yours? There are a couple of things you can do on your side, making sure your pictures are optimized (i.e. in a webp format), unnecessarily large media files are avoided, javascript is set up properly and any loading that can be deferred is set up to load asynchronously. This would be analogous to removing any obstacles to make sure the bakery cashiers have a straightforward process to serve customers.

Coming back to the previous discussion on parallel requests in PHP though there is little to be done from the website’s side. How well your page performs under load i.e. when multiple people are online in parallel depends on the server hardware your site is running on. Traditional hosting providers usually offer different price tiers with an e-commerce option somewhere on the upper

end. Basically what is changing as one moves up the tiers is the underlying hardware gets a bit stronger. In the background there is still a single server where your application resides and all incoming requests are processed. This approach is however limited and there are smarter options available. For instance you can train your bakery cashiers to improve their capabilities but the potential for improvements is limited. Instead it might be smarter to add parallel cashiers to split the work.

Why single server hosting is not the way to go:
The fastest/largest single server CPU currently available on AWS has 48 cores [5]. It's not hard to tell how many parallel user requests even this machine could handle before something gets queued. The majority of hosting options out there have a lot less cores on their servers. In any case making the individual server more powerful is (very) costly and from a speed improvement option not actually the most desirable result.
Instead of using a single more powerful server, large applications use so-called load balancing. This load balancing analyzes incoming traffic, and distributes it (smartly or evenly) across multiple servers. The technology has been around for quite some time but setting this up from scratch on one of the large cloud providers like Amazon or Google is a complex development project out of reach for a majority of e-commerce sites. Even for startups with the technological know-how setting this up can be a time-consuming hassle. Luckily a variety of providers have begun offering similar solutions wrapped inside a user interface that makes it easier to set up and use.
Allow us one more bakery analogy, hiring and training more cashiers manually might be really time consuming and difficult but imagine there was a solution that made it super simple to add new or reduce cashiers (say an affordable, easy to install walk-in walk-out retail solution).

What accessible cloud providers can offer:
Setting up a sophisticated infrastructure to optimize speed does not require deep technical knowledge anymore. Service providers like Cloudways or Codesphere make this setup rather simple with immense benefits for your site. Any e-commerce site or solopreneur venture can take advantage of the technologies powering modern cloud native startups. You can set up load balancing with just a few clicks, it becomes super easy to set up staging environments where continuous updates can be tried out. Things like A/B testing are equally easy because connecting different codebases to one of multiple servers is just a couple of clicks away.

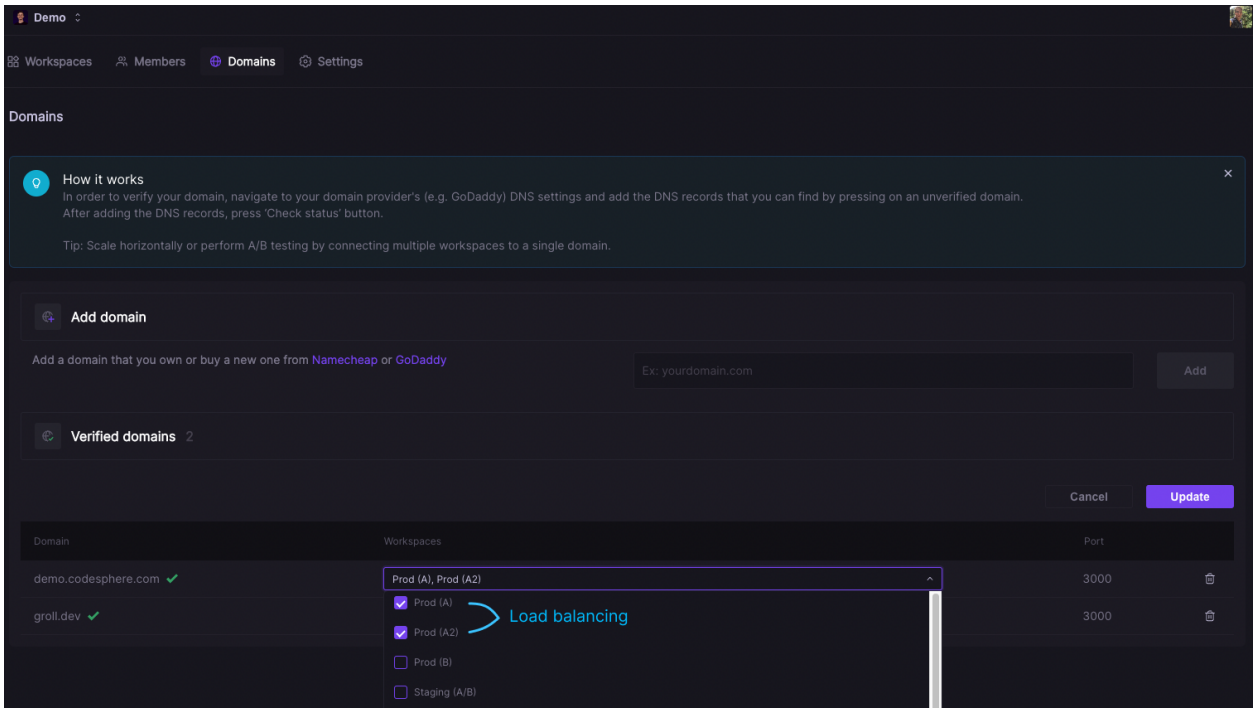
Let's do a short excerpt into how easy this can be achieved with Codesphere.

The simple steps are:

1. Clone your code onto the server either via Github or SSH (for bigger e-commerce sites our engineers take care of the entire process)
2. Connect your domain
3. Select the workspaces to run the domain (multiple workspaces -> load balancing, different code versions in workspaces -> a/b testing)

4. Done - Any workspace not connected to your domain gets an internal URL to test things (i.e. for staging)

More backgrounds and advanced usages can be found in the Codesphere documentation.



Getting set up on a cloud native infrastructure has never been easier. Interested? Book a free demo now

References:

Php usage statistics - w3techs.com [1]
Page speed issue stats - Portent.com [2]
Ad ranking for Google - developer.chrome.com [3]
Mobile load time stats - thinkwithgoogle.com [4]
Fastest single server on AWS - aws.amazon.com [5]