From zero to hero

Introduction

What is a web server and where Nginx fits

Let’s visit a website

- When you type http://www.google.com in your browser of choice, in very few seconds the Google home page appears. But within you typing the URL, pressing Enter and the page appearing lots of things happened.

- A *request* was initiated by your browser, leaving your local network, traversing the global network (the Internet) till it reaches one of the computers which are assigned with serving Google’s home page.

- That computer in return initiates a *response* containing the contents of the requested page. This response takes a similar route through the Internet till it reaches your local network and finally your computer’s browser.

- The computer that was tasked with serving you this web page is called a web server.

Please welcome the HTTP protocol

- HTTP stands for Hyper Text Transfer Protocol. A protocol in IT jargon refers to a set of rules that are agreed upon between two or parties for communication.

- When you requested www.google.com you initiated an HTTP *get* request. When the web server receives this request it knows that it should reply with a response containing the HTML, CSS, JavaScript, images and any other component that happens to be on the page. Your browser is responsible for interpreting this data and displaying it in a proper manner.

- By default, the HTTP protocol is served on port 80 of the web server (although this can be modified).

- In the following mini lab, let's examine this operation in a little more detail.

LAB: examining HTTP communication

- Open your Internet browser. You can use Mozilla Firefox or Google Chrome for this example.
- Activate the developer's panel. In Firefox this is done by pressing F12. In Chrome you can CTRL+SHIFT+I (or CMD + ALT + I on a MAC).
- Switch to the network tab.
- Type www.wordpress.org in the address bar and hit Enter.
- Observe the several lines that fill the panel. Those are HTTP requests. Each one of them has got a type, URL, a status, and other information.

So what is a web server?

- Technically, a web server is any computer that has a running daemon (service) accepting HTTP requests on some port (by default, 80), and replying with HTTP responses. This is typically HTML content.

- Serving just HTML (along with images, JS, CSS) is serving static content. That is, content that does not change from one user/condition to another. Nowadays, this is very rare!

- In an Internet dominated by *dynamic* web *applications* like social networking services, online shopping, banking portals and others, non-static (dynamic) content is generated on the fly and served.

- The web server does this by handing the requests to an application handler (like PHP, Ruby, ASP.net) that will do the necessary processing and hands the web server the final content that will be served to the client.

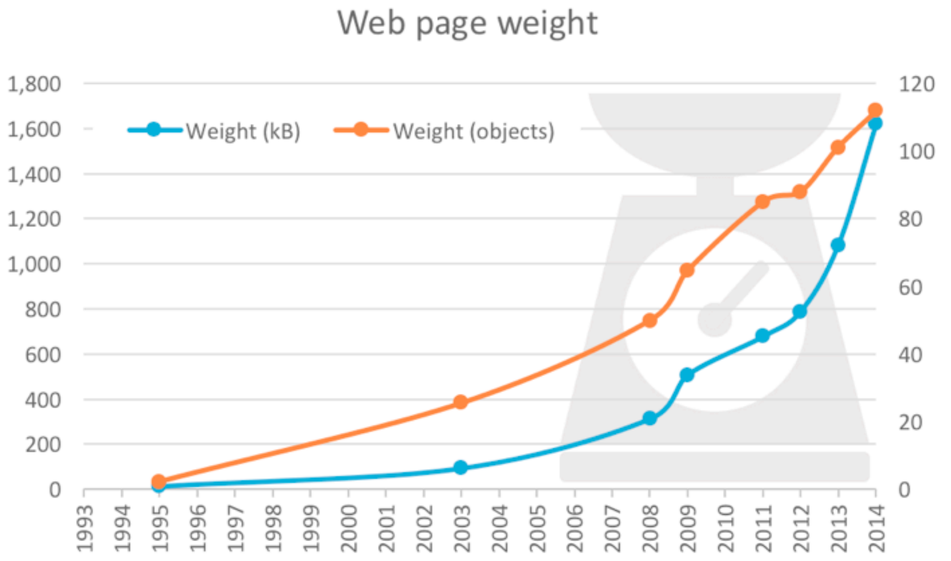
- A lot of web servers are available in the market today. Microsoft Internet Information Services (IIS), Apache, and — of course — Nginx.

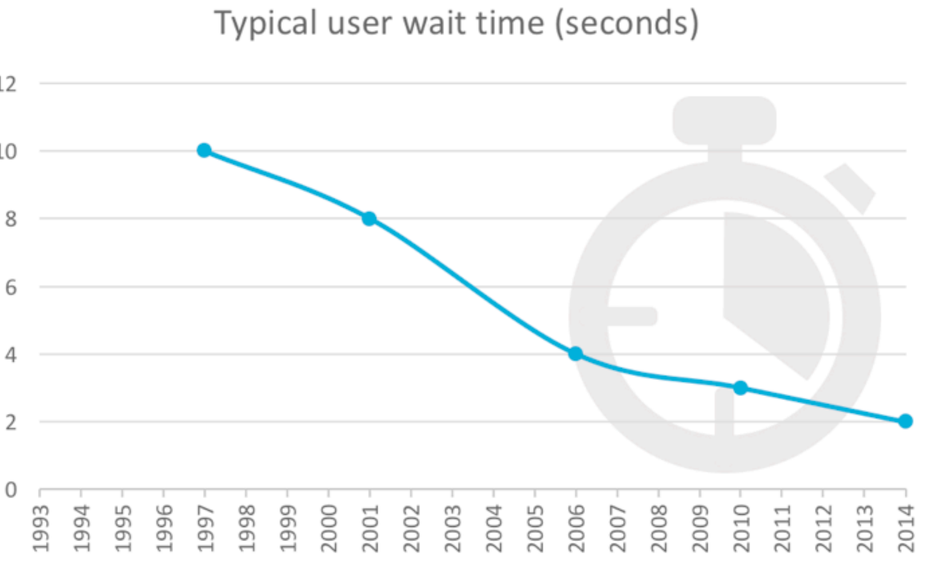
And what is Nginx?

- Nginx, pronounced eNgineEx, a web server developed by Igor Sysoev in 2002. It was officially released in 2004.
- According to the official website www.nginx.org, it is defined as an HTTP and reverse proxy server, a mail proxy server, and a generic TCP/UDP proxy server. As per Netcraft, Nginx has served or proxied more than 28% of the busiest websites in February 2017.
- It is a powerful web server that is capable of doing lots of complex stuff. Let's see why you should be using it.

Why should I be using Nginx?

- Notice how I chose “using” Nginx instead of “switching to” Nginx. That is because you don't have to drop your existing web server or to totally change your infrastructure to enjoy the power of Nginx. It can nicely fit into your setup and enhance it. Nginx provides:
  - **Speed:** Who wants a slow-loading page? Today, a user cannot wait more than two-seconds to see a fully loaded page. Otherwise you start losing visitors!
  - **Acceleration:** if you already have two or more web servers running the same application, Nginx can accelerate performance by routing traffic to those web servers in a way that enhances the overall speed.
  - **Load balancing:** a load balancer is a device/service that distributes traffic load on two or more web servers. That provides fault tolerance and increases performance. There are commercial hardware load balancers available but they are very expensive and somewhat complex to setup and service. Nginx can easily act as a robust load balancer that will cut off costs yet give you the needed load distribution.
  - **Scalable concurrent connections handling:** traditional web servers (like Apache and IIS) can serve incoming requests without a problem, until the number of concurrent requests reaches a certain limit (say 1000). Whenever this number rises, performance starts to degrade. That is because the way those servers were designed and the model by which they server web requests. Unfortunately, increasing the installed CPU cores and RAM will not give you the intended results. If 4 GB of RAM are sufficient for serving 1K of concurrent requests, 8 GB will not enable the server to handle 2K at the same level. Nginx does not suffer from this problem and it can easily handle increasing number of concurrent requests.
  - **The ability to operate on relatively cheap hardware:** Nginx can be deployed on servers with very limited hardware capabilities and still perform much better that its counterparts on the same hardware.
  - **On-the-fly upgrades:** Yes, Nginx is one of the very few systems that can patched or upgraded without having to take a downtime and disrupt your business. Personally, I would use Nginx for my environment just for that reason.
  - **Ease of installation and maintenance:** as we move on, you’ll find that Nginx is relatively easier than you think when it comes to basic (and even advanced) usage.





Source: nginx.org

What can Nginx do for me?

- If you think Nginx is just a web server that proved to be more powerful than other counterparts in the market, you're underestimating it.

- It can work alongside other web serves (like Apache) as a reverse proxy. A reverse proxy is a service that stands between the client and the web servers. It receives the request from the client and sends it on its behalf to the web servers behind it. When the response is received from the web servers, it relays it back to the requesting client. Reverse proxies are used for performance and security reasons.

- Every feature can be activated/deactivated using *modules*. This makes it easier for the administrator to work with the required functionality only. Again, this provides security and performance enhancements.

- It uses the asynchronous way of serving requests. In traditional web servers (like IIS or Apache), a new thread is created for every request received. So if you have 1000 requests at the same time, that's 1000 threads working concurrently. If the maximum number of threads the server can handle is 1000, any further requests will have to wait till one of the previous requests has finished before it can get served. That means increasing the wait time. With Nginx, only one thread is used but it is based on *events* (more on that later) which greatly improves speed.

- It allows you to serve many protocols not just HTTP and HTTPS. You can use it to serve IMAP, POP3 and SMTP for mail protocols. Also the latest full-duplex WebSocket protocol is supported.

- When using SSL, an extra overhead is required by the web server to decrypt/encrypt the content it receives and sends. With Nginx acting as a reverse proxy, you can offload this SSL work to be done on Nginx. The backend web server no longer has to work with SSL encrypted data; as reverse proxy will do that.

- Nginx excels when it comes to serving large files or streaming media. As mentioned, other web servers create a thread for each new request. What is the request was to serve a 1 hour long video or download a several-gigabytes file? That will certainly block the thread till the download/stream is complete. Several thousands similar requests are enough to make further requests wait in a queue. However, because of the asynchronous nature of Nginx, this is never a problem.

Nginx vs. Apache

- A common misconception is that Nginx and Apache are totally interchangeable. That Nginx is just an enhanced web server so once you learn it you should drop Apache and start using Nginx instead. This couldn't be more wrong! Most to the time you'll see Nginx working side-by-side with Apache. Sometimes you have to use Nginx solely in areas where only Nginx can operate, some other times you have to use Apache. It all depends on the type of project you are working on and what you are trying to accomplish.

- In this part, we have a quick look at the technical and historical differences between Nginx and Apache.

- Apache was developed in the mid-nineties and is still having the larger market share compared to other web servers. Nginx was introduced in the early 2000's and since 2008 it's gradually taking away the market share from Apache.

- Apache creates a separate child process for each request. Each process uses a blocking thread. That means Apache falls short in performance when it comes to serving a considerably huge number of concurrent connections. Nginx on the other hand avoids this problem by creating multiple, non-blocking worker threads, each is able to serve thousands of concurrent connections.

- Because of the way Nginx is designed, it requires fewer hardware resources than Apache.

- Apache is more popular on operating systems than Nginx. For example, all Linux flavors have Apache available in their officials repositories (software packages directory think of it *roughly* as the Microsoft Market or Apple Store). On the other hand, and while it is easy to download and install, Nginx still requires some tweaking on the OS side before it can be installed using package management. In all cases - of course - you can compile the software from source.

- Nginx can act as a load balancer and/or a reverse proxy. Having multiple Apache web servers in the backend and using one or more Nginx servers in front of them as a reverse proxy will give you the best of both worlds. Apache alone does not have this capability.

- Apache has a long history of supporting dynamic content languages like PHP, Ruby, Python and others. Nginx, while it does support PHP, will require some extra effort from your side, the administrator, to get things working.

- There are subtle differences between both web servers when it comes to configuration. Apache translates the URL to a file path on the underlying OS. For example: www.example.com/plants/fruits/apples/red.html would be interpreted as requesting a file called red.html located in /var/www/html/plants/fruits/apples, assuming that the web root directory is /var/www/html. On the other hand, Nginx does not work like this; it parses the URL in a different way. That makes placing an .htaccess file, for example, in the plants directory, let's say to deny users from hot linking image files, is totally useless to Nginx.

- There is a huge difference between both web servers when dealing with modules. Modules are a way to add extra functionality to the server. For example, the rewrite module in Apache enables the server to interpret a request like www.example.com/users/johndoe as www.example.com/index.php?users/johdoe. This is very popular in content management systems like Wordpress to server *clean* URLs. Activating a module in Apache is as simple as opening the configuration file, uncommenting/adding the appropriate line(s) and restoring the service. On the other hand, to use a module in Nginx you have to re-compile it from source, adding whatever modules you need as command-line switches.

- I know that last point may sound a little intimidating but it's worth the effort and here's why: compiling Nginx with only the modules the you are going to use in the project will ensure that the binary does not contain any unneeded code. Extra code requires extra processing and memory from the infrastructure side, this means potentially less performance because of functionalities you don't need. Additionally, from a security perspective, you should only install/enable needed features to decrease the risk of attacks.