BUILDING THE MODERN APPLICATION ARCHITECTURE

NGINX

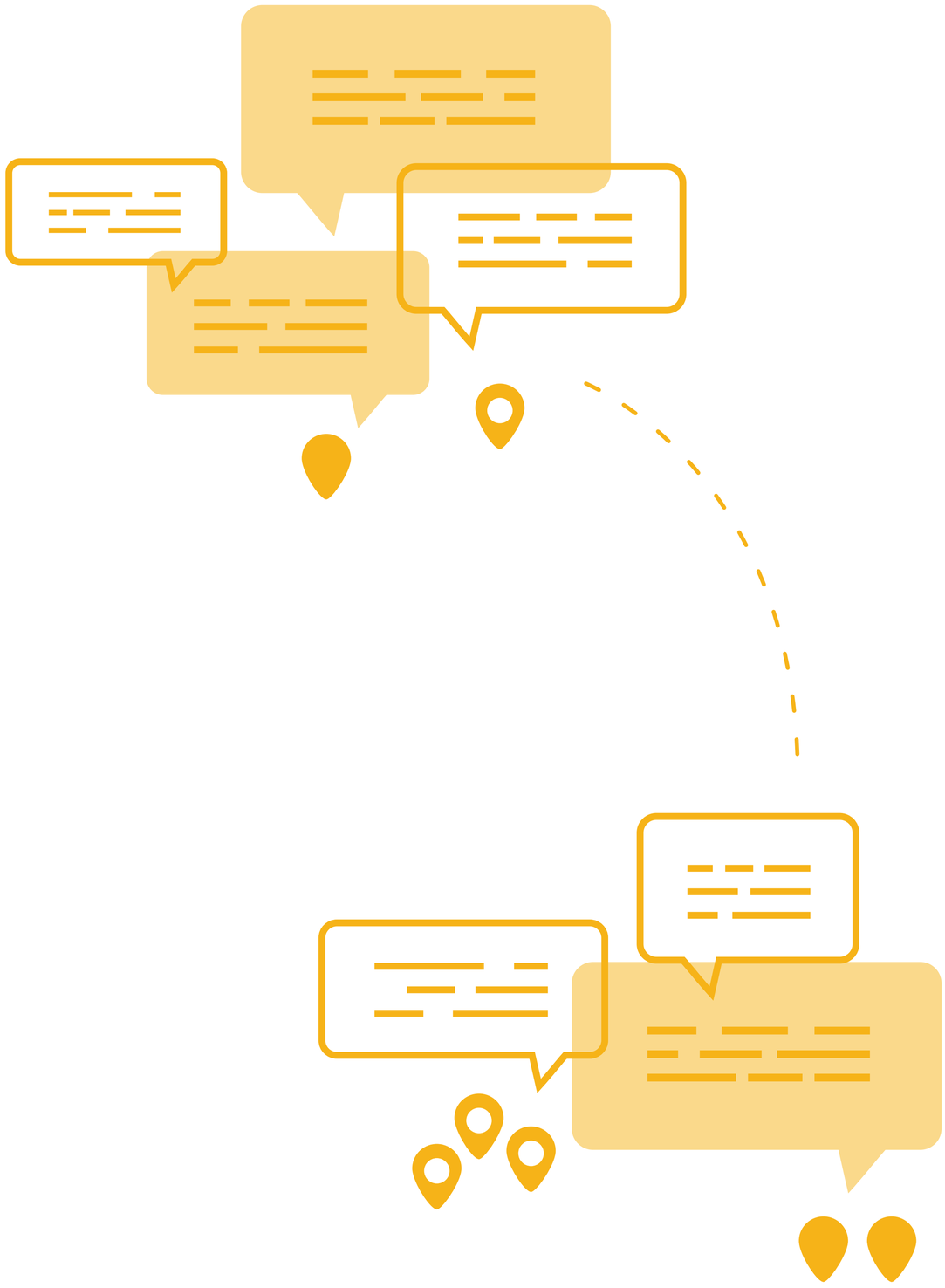
Table of Contents

Introduction

Application performance has become a hot topic in recent years—but our need for it is, quite honestly, nothing new.

In fact, for as long as we've used computer systems in the workplace, application performance has always mattered to businesses. From restaurants to banks, manufacturers to hospitals, almost every industry—indeed, almost every department of every business in every industry—has relied on a variety of applications to be productive and competitive for decades. So why is application performance such a big deal now? And, more importantly, how can we work to ensure that we achieve optimal performance today?

In this eBook, we'll look at why the way we deliver applications has dramatically upped the ante on our performance expectations and demands. We'll also explore how the right infrastructure architecture can be a definitive game-changer when it comes to optimizing the performance of today's most business-critical applications.



HTTP: The Rise to Dominance

A decade or so ago, we looked to the Internet primarily as a way to disseminate information using HTML, while applications were cumbersome, proprietary entities that lived offline. Today, however, the Internet connects a number of systems together on a backbone of web-based applications. And enterprises are beginning to invest substantial time and effort in rewriting proprietary software using more extensible, web-friendly languages that take advantage of HTTP and Web APIs.

These are huge changes in the way we develop and deliver applications. But in order to understand how and why they occurred, we need to backtrack a bit.

Let's start with how applications talk to each other. HTTP has become a lingua franca of application communication—but it wasn't always that way. In the years before software developers started to commonly use HTTP, practically every networking application had its own way of speaking. Imagine, then, how difficult it was when applications needed to share information, and how urgently businesses wanted to address that difficulty, especially as client/server applications began to dominate the enterprise software landscape.

Before long, we saw an explosion of middleware designed to integrate legacy systems and newer applications—which, in turn, created a number of smaller components that also needed to communicate. Bottom line: Developers needed a standardized way for all of these elements to exchange information in order for applications to be both accessible and useful.

Because it is based on the use of lowest common denominators—syntax, semantics, and timing—HTTP became seen as an effective solution. And, as it gathered momentum, HTTP slowly but radically democratized software development by giving programmers the common communication protocol they needed to develop their applications.

Through this steady migration over the past 10 years, HTTP has proven to be simple and versatile enough to become the common language of the Internet, while also becoming capable of encapsulating legacy enterprise systems, integrating newer developments with already existing systems, and successfully uniting very different building blocks of modern software applications.

But HTTP also afforded software developers something else. This simple but powerful protocol provided applications with an entirely new delivery paradigm—the Web.

"The modern Internet is made of compose-able and interoperable components all speaking HTTP."

HTTP as a Disrupting Force

What do companies like Netflix and Amazon have in common? They developed software platforms powered by HTTP to disrupt their respective industries. Where Netflix utilized HTTP to stream video subscriptions at HD quality over best-effort delivery networks of the Internet, Amazon revolutionized the retail environment with product recommendations and searching. Both of these companies recognized that software communication driven by HTTP could ultimately enable them to integrate systems faster, simplify development (due to the abundance of tools and libraries), experiment more flexibly, and accelerate the rollout of production features into market.

Applications Move to the Web

While HTTP gave programmers a common communication protocol by which to make their software extensible, it was the Web that became the model for the modern application, thanks to three trends:

- **Agility.** Part of software's migration to the Web stemmed from the ability of developers to easily separate the interface layer from business logic layer. In this way, web applications presented programmers with a powerful framework that enabled them to code more efficiently and quickly. Changes to functionality could be made without disrupting the front-end.

- **Mobility.** More and more workers began using laptops and tablets to perform their job functions. Traditional client/server software just doesn't accomplish business tasks well when the workforce is moving outside of the corporate network.

- **Functionality.** Ultimately, the Web provided a means for users to access software through an ordinary web browser—which meant nothing for the user to install or for IT to maintain.

The transition of applications to the Web then was a convergence of factors—developers discovering the power of HTTP, the workforce becoming more mobile, and people using the Web more for everyday needs. But with this transition came a growing expectation for reliability and performance. The more we used web-based applications, it seemed, the more we depended on them. Users lost patience for long load times, which spelled competitive trouble for businesses who couldn't keep pace. In order to accommodate the scale, flexibility, and performance required of HTTP-based web applications (and demanded by users), a new architecture was needed as well. And that meant getting rid of an old paradigm: the vertical stack.

The Rise of the Digital Business

As programmers were discovering a new way to develop enterprise level software, the enterprise itself was undergoing a transformation thanks to HTTP: the rise of the digital business. It didn't take long for organizations to see the importance of being online as companies like Amazon, eBay, Dropbox, and Facebook demonstrated the pervasiveness of the digital world and, more importantly, how well HTTP-based applications could scale and function to deliver new services. Riding on the wave of HTTP, these new businesses brought their products and services to market far quicker than the incumbents.

The Web Application Architecture

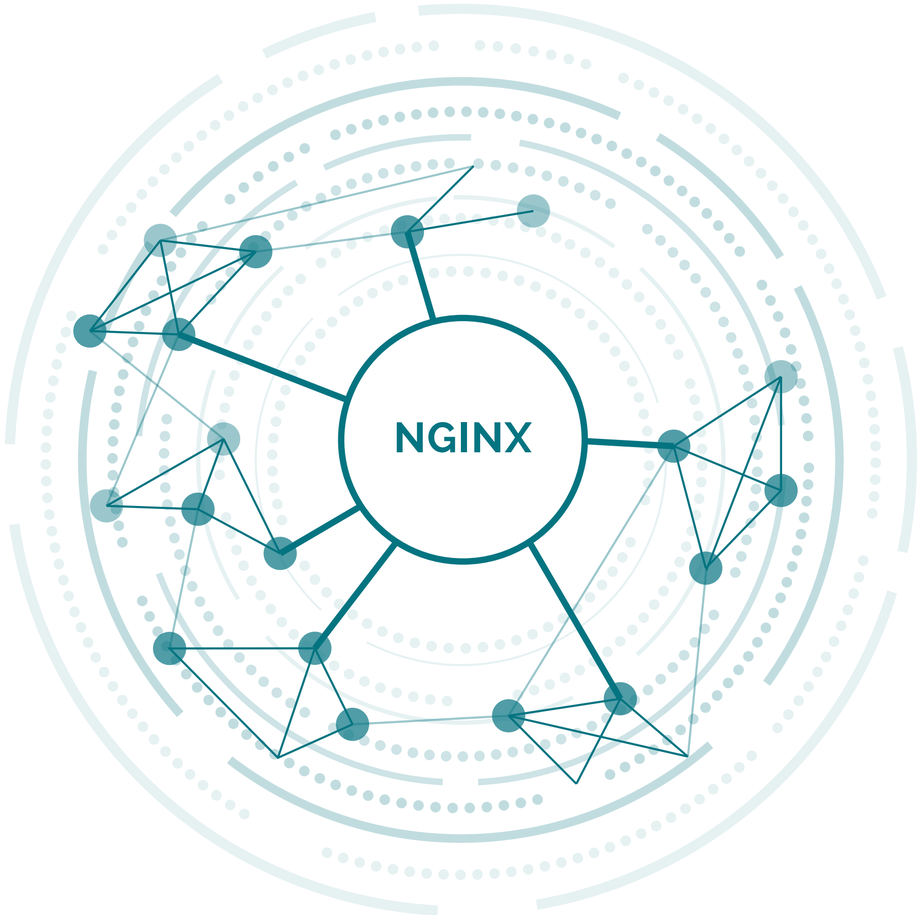
The Old Way

In the early days of the transition to HTTP, architectures were mainly vertical—where operating systems and development platforms (i.e., Sun Sparc, Solaris OS, iPlanet etc.) were tied to specific hardware. Combined with other proprietary hardware/software combinations such as load balancers, application delivery controllers, proxies, and caches, developers were essentially locked in to a specific framework, with little to no flexibility. If the framework didn't offer the functionality or performance a developer needed, they were out of luck. This environment, then, was terribly ineffective at meeting the growing needs of organizations to be "digital." Accommodating user demand meant achieving scalability, which meant procuring and configuring new stacks of both hardware and software.

That was the last straw. Not only was this prospect a horrendous expense, it was also a configuration nightmare. And so, as organizations began to realize the failure of their legacy vertical stacks in meeting the needs of the new digital world, they finally zeroed in on the root of the problem: application performance.

The issue wasn't so much that improving performance required more proprietary hardware/software solutions in the vertical stack—it was that it exacerbated a bigger situation. The same developers that were responsible for building, maintaining, and iterating the application weren't in control of any of the pieces required to deliver it. Other teams controlled the delivery. The time it took for requested performance changes to be implemented (days, and sometimes even weeks) was often more than the end user was willing to wait. In fact, when application performance suffered, these users simply turned elsewhere to meet their needs. It became clear that the most cost-effective and efficient way to ensure high application performance was to allow developers themselves to control the entire ecosystem from code to configuration.

And that required a new architecture.



The New Way

Fortunately, thanks to the growing variety of tools and changing software development paradigms, web architects started to move away from vertical stacks to distributed web architectures. New languages and application frameworks appeared (i.e., PHP, Python, and Ruby, Node.js and others) that were popularized quickly because they enabled developers to code and deploy rapidly on commodity hardware and software. These frameworks prompted developers to adopt distributed, loosely-coupled application architectures that provided a more flexible approach than their predecessors allowed. This flexibility was the key to decreasing time to market and rapid iteration.

Vertical was out. Horizontal was in. And with this new way to architect for web applications, developers were freed from the entanglements of proprietary hardware and software stacks. They were firmly in charge.

A lot of popular services like Twitter, Dropbox, Instagram, and LinkedIn wouldn't exist today if a number of very important initiatives, like open source software, didn't succeed 8-10 years ago.

Understanding the Roadblocks

We've come a long way—but there are lingering challenges. Despite the flexibility and efficiency afforded by HTTP and horizontal application architectures, developers and organizations still face some fundamental issues:

- **Scale.** When HTTP-based applications like dynamic websites are exposed to the public, there's no telling exactly how much traffic they can or will generate. Sudden bursts may require unforeseen increases in hardware, for example—but if other teams are in charge of those changes, developers can hardly design apps for scalability.

- **Time.** Often, there's just not enough time to focus on the details of the infrastructure elements that are in-between "the network" and "the application" once the application is exposed online. Although many developers have started to understand the importance of handling app performance end-to-end themselves (including HTTP performance and scalability), they can't afford the time to spend tuning and optimizing the "in-between" hardware—nor can they wait for network engineers.

- **Complexity.** Most application frameworks do not offer a way to deal with the hidden complexities inherent in HTTP. For example, a developer may quickly construct an app from a number of building blocks provided by the programming language, the framework, the libraries, and a bit of third-party code found on GitHub. However, she might then discover that, when combined, all those building blocks still don't allow her to handle HTTP concurrency, or security, caching frequent content, or maybe just header manipulation.

- **Focus.** Just as the infrastructure required to deliver today's complex web applications has grown, so has the level of disconnection. All of these individual pieces require more specialized software to become connected into a single, cohesive application stack—which creates an even greater management nightmare. Not only does an organization need people to handle the different aspects of the modern web application (i.e., data specialists, systems and automation specialists, web programmers, etc.) and the infrastructure hardware, but they may also need people to cobble the systems together—engineers whose sole purpose is to make sure all the pieces talk to one another.

- **Control.** Rapid prototyping, agile development, experimentation—all of these are the hallmarks of today's application developer. Yet despite the shift from vertical to horizontal architectures, many developers still aren't fully in control of their application environments, with some infrastructure parts still completely outside of their control. In order to encourage and capitalize on these traits, organizations must give control of the stack components—the "in-between" elements—to the developers.

It can be almost painful to see the way an unprepared application copes with real-world traffic.

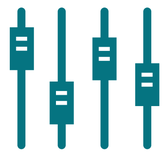
Open Source Software: Overcoming the Roadblocks

Many of the innovations to web application architectures have come about as a result of open-source initiatives such as Apache, HAProxy, Varnish, Squid, and NGINX. These initiatives paved the way for developers to solve many of those challenges:

Scale. Because these solutions are based on open-source platforms, they can be installed on commodity hardware that's able to rapidly scale when an application calls for it.

Time. Developers can deploy and control what was once network functionality (i.e., an application delivery controller) easily through a software package.

Focus. Many of the open-source building blocks of horizontal web applications are built to facilitate the communication via HTTP, giving developers the opportunity to make system-level changes using protocols and languages with which they are familiar.



Control. Because they are now creating software solutions, developers are in complete control, capable of making minor configurations to full-scale rebuilds without the involvement of network engineers or other teams.

But despite the advancements afforded through these open-source software platforms, they are still separate. Varnish doesn't do what Apache does. Apache doesn't do what HAProxy does. Ultimately, even horizontal web architectures still require multiple components that can impact the biggest challenge of all: speed.

NGINX: A Superior Web Server

As an open-source alternative to Apache and other web servers, NGINX immediately became popular among web engineers and architects for a number of reasons. First, it was designed with web acceleration in mind. It's not just a web server—it's a "web server, reinvented." It was also meant to provide developers with a way to fine-tune the performance of their applications without having to use a separate piece of software or other hardware appliance. This, in turn, makes it simple for developers to integrate NGINX into the application architecture without having to ask network administrators for assistance. Developers can set it up themselves and modify the configuration to meet their needs for rapid application development. And, lastly, NGINX is scalable, secure and resilient.

Speed: The Ultimate Challenge

End users want applications to operate quicker. Developers know that their applications need to perform faster. Businesses know that they need to get their applications into the market as quickly as possible to be competitive. But the answer isn't just to throw more hardware and software at the problem. Really solving the speed challenge requires a different approach:

- **Consolidation.** The more systems that are involved, the more opportunity for latency, as HTTP requests must pass through multiple systems en route to the application.

- **Integration.** Components in the web architecture must be integrated. Being closer to the actual application increases the opportunity to remove latency.

- **Packaging.** Elements within the web architecture must be packaged together for rapid deployment. Latency isn't just about "chatty protocols." It's also about how fast developers can scale the application when needed.

Recognizing the opportunity to consolidate typical web architecture functionality like load balancing, media delivery, and caching into a single software package, NGINX Plus was born.

Say Hello to NGINX Plus

The only way to intertwine performance and scalability to solve all of the challenges and complexities of HTTP applications is to have a very capable and versatile software-only product, running on a generic server, on a generic Linux OS. This delivers both infrastructure (performance, scalability, reliability, security at less cost) and operational benefits—which is why we created NGINX Plus, an application delivery platform like no other. Combining the power of a software load balancer, proxy, media server, and web server, NGINX Plus solves the challenges facing developers and organizations today in looking for a single, integrated solution to deliver high-performance web applications.

There are sites running NGINX today that are consistently delivering content to around a million concurrent users on a single server. That can't realistically be done with any other solution without hard-tuning.

Purpose-built from the start

NGINX originated from the world of application software with a very specific goal: to make web infrastructure operate faster. It has never been a networking tool, or firmware ripped out of a box. Over the past few years, NGINX evolved to be just the right tool for application developers and application system engineers who want a single, proven system for web delivery and acceleration.

Learn More

Curious about your ability to develop and deliver high-performance
web applications? Want to take NGINX for a test drive?
Visit www.nginx.com/free-trial-nginx-plus/ for a free trial today.