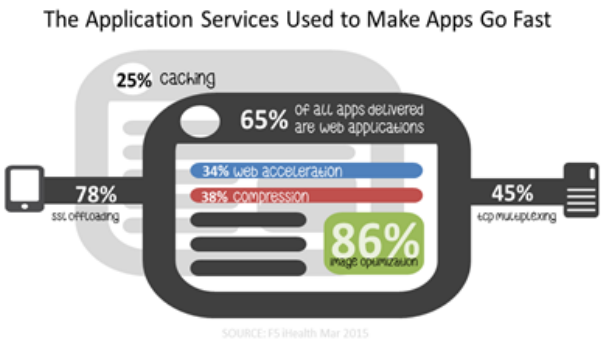
Microservices and HTTP/2

Lori MacVittie, 2015-08-06

It's all about that architecture.

There's a lot of things we do to improve the performance of web and mobile applications. We use caching. We use compression. We offload security (SSL and TLS) to a proxy with greater compute capacity.



We apply image optimization and minification to content.

We do all that because performance is king. Failure to perform can be, for many businesses, equivalent to an outage with increased abandonment rates and angry customers taking to the Internet to express their extreme displeasure.

The recently official HTTP/2 specification takes performance very seriously, and introduced a variety of key components designed specifically to address the need for speed. One of these was to base the newest version of the Internet's lingua franca on SPDY.

One of the impacts of this decision is that connections between the client (whether tethered or mobile) and the app (whether in the cloud or on big-iron) are limited to just one. One TCP connection per app. That's a huge divergence from HTTP/1 where it was typical to open 2, 4 or 6 TCP connections per site in order to take advantage of broadband. And it worked for the most part because, well, broadband. So it wouldn't be a surprise if someone interprets that ONE connection per app limitation to be a negative in terms of app performance.

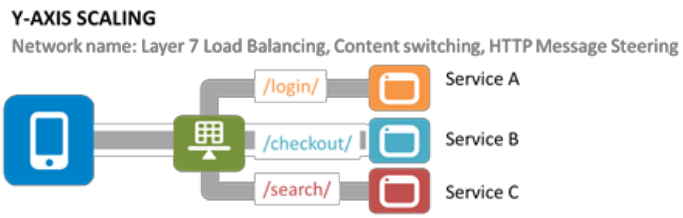
There are, of course, a number of changes in the way HTTP/2 communicates over that single connection that ultimately should counteract any potential negative impact on performance from the reduction in TCP connections. The elimination of the overhead of multiple DNS lookups (not insignificant, by the way) as well as TCP-related impacts from slow start and session setup as well as a more forgiving exchange of frames under the covers is certainly a boon in terms of application performance. The ability to just push multiple responses to the client without having to play the HTTP acknowledgement game is significant in that it eliminates one of the biggest performance inhibitors of the web: latency arising from too many round trips. We've (as in the corporate We) seen gains of 2-3 times the performance of HTTP/1 with HTTP/2 during testing. And we aren't alone; there's plenty of performance testing going on out there, on the Internets, that are showing similar improvements.

Which is why it's important (very important) that we not undo all the gains of HTTP/2 with an architecture that mimics the behavior (and performance) of HTTP/1.

Domain Sharding and Microservices

Before we jump into microservices, we should review domain sharding because the concept is important when we look at how microservices are actually consumed and delivered from an HTTP point of view.

Scalability patterns (i.e. architectures) include the notion of Y-axis scale which is a sharding-based pattern. That is, it creates individual scalability domains (or clusters, if you prefer) based on some identifiable characteristic in the request. User identification (often extricated from an HTTP cookie) and URL are commonly used information upon which to shard requests and distribute them to achieve greater scalability.



An incarnation of the Y-axis scaling pattern is domain sharding. Domain sharding, for the uninitiated, is the practice of distributing content to a variety of different host names within a domain. This technique was (and probably still is) very common to overcome connection limitations imposed by HTTP/1 and its supporting browsers. You can see evidence of domain sharding when a web site uses *images.example.com* and *scripts.example.com* and *static.example.com* to optimize page or application load time. Connection limitations were by ***host (origin server)***, not domain, so this technique was invaluable in achieving greater parallelization of data transfers that made it appear, at least, that pages were loading more quickly.

Which made everyone happy. Until mobile came along. Then we suddenly began to realize the detrimental impact of introducing all that extra latency (every connection requires a DNS lookup, a TCP handshake, and suffers the performance impacts of TCP slow start) on a device with much more limited processing (and network) capability. I'm not going to detail the impact; if you want to read about it in more detail I recommend reading some material from Steve Souder and Tom Daly or Mobify on the subject. Suffice to say, domain sharding has an impact on mobile performance, and it is rarely a positive one.

You might think, well, HTTP/2 is coming and all that's behind us now. Except it isn't. Microservice architectures in theory, if not in practice, are ultimately a sharding-based ***application*** architecture that, if we're not careful, can translate into a domain sharding-based ***network*** architecture that ultimately negates any of the performance gains realized by adopting HTTP/2.

That means the architectural approach you (that's you, ops) adopt to delivering microservices can have a profound impact on the performance of applications composed from those services.

The danger is not that each service will be its on (isolated and localized) "domain", because that's the whole point of microservices in the first place. The danger is that those isolated domains will be ***presented to the outside world*** as individual, isolated domains, each requiring their own personal, private connection by clients.

Even if we assume there are load balancing services in front of each service (a good assumption at this point) that still means direct connections between the client and each of the services used by the client application because the load balancing service acts as a virtual service, but does not eliminate the isolation. Each one is still its own "domain" in the sense that it requires a separate, dedicated TCP connection.

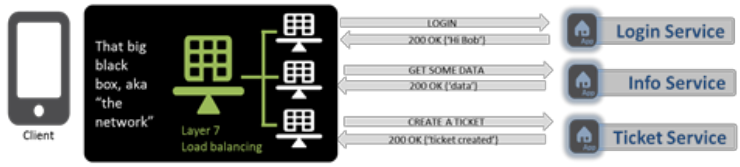
This is essentially the same thing as domain sharding as each host requires its own IP address to which the client can connect, and its behavior is counterproductive to HTTP/2*.

What we need to do to continue the benefits of a single, optimized TCP connection while being able to shard the back end is to architect a different solution in the "big black box" that is the network. To be precise, we need to take advantage of the advanced capabilities of a proxy-based load balancing service rather than a simple load balancer.

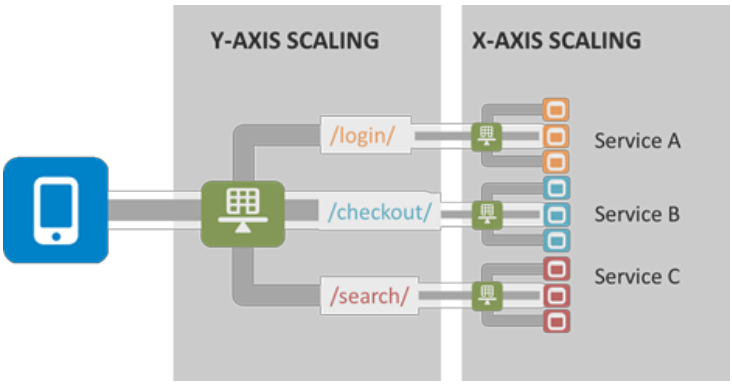
An HTTP/2 Enabling Network Architecture for Microservices

That means we need to enable a single connection between the client and the server and then utilize capabilities like Y-axis sharding (content switching, L7 load balancing, etc...) in "the network" to maintain the performance benefits of HTTP/2 to the client while enabling all the operational and development benefits of a microservices architecture.

What we can do is insert a layer 7 load balancer between the client and the local microservice load balancers. The connection on the client side maintains a single connection in the manner specified (and preferred) by HTTP/2 and requires only a single DNS lookup, one TCP session start up, and incurs the penalties from TCP slow start only once. On the service side, the layer 7 load balancer also maintains persistent connections to the local, domain load balancing services which also reduces the impact of session management on performance. Each of the local, domain load balancing services can be optimized to best distribute requests for each service. Each maintains its own algorithm and monitoring configurations which are unique to the service to ensure optimal performance.



This architecture is only minimally different from the default, but the insertion of a layer 7 load balancer capable of routing application requests based on a variety of HTTP variables (such as the cookies used for persistence or to extract user IDs or the unique verb or noun associated with a service from the URL of a RESTful API call) results in a network architecture that closely maintains the intention of HTTP/2 without requiring significant changes to a microservice based application architecture.



Essentially, we're combining X- and Y-axis scalability patterns to architect a collaborative operational architecture capable of scaling and supporting microservices without compromising on the technical aspects of HTTP/2 that were introduced to improve performance, particularly for mobile applications.

Technically speaking we're still doing sharding, but we're doing it *inside* the network and without breaking the one TCP connection per app specified by HTTP/2. Which means you get the best of both worlds - performance and efficiency.

Why DevOps Matters

The impact of new architectures - like microservices - on the network and the resources (infrastructure) that deliver those services is not always evident to developers or even ops. That's one of the reasons DevOps as a cultural force within IT is critical; because it engenders a breaking down of the isolated silos between ops groups that exist (all four of them) and enables greater collaboration that leads to more efficient deployment, yes, but also more efficient implementations. Implementations that don't necessarily cause performance problems that require disruptive modification to applications or services.

Collaboration in the design and architectural phases will go along way towards improving not only the efficacy of the deployment pipeline but the performance and efficiency of applications across the entire operational spectrum.

* It's not good for HTTP/1, either, as in this scenario there is essentially no difference** between HTTP/1 and HTTP/2.

** In terms of network impact. HTTP/2 still receives benefits from its native header compression and other performance benefits.

F5 Networks, Inc. | 401 Elliot Avenue West, Seattle, WA 98119 | 888-882-4447 | f5.com

F5 Networks, Inc.
Corporate Headquarters
info@f5.com

F5 Networks
Asia-Pacific
apacinfo@f5.com

F5 Networks Ltd.
Europe/Middle-East/Africa
emeainfo@f5.com

F5 Networks
Japan K.K.
f5j-info@f5.com