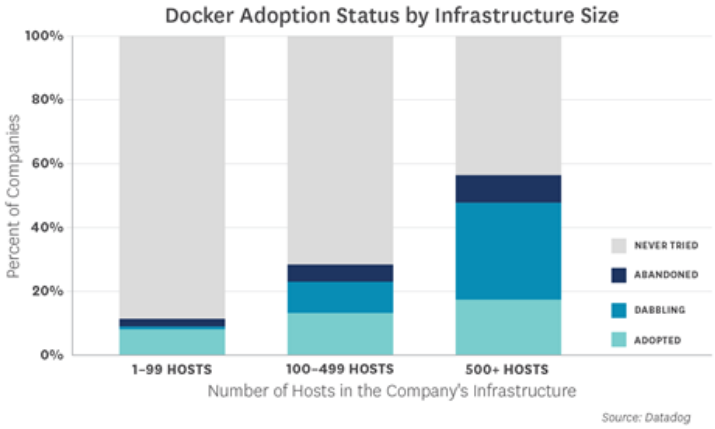
On monoliths versus microservices

Lori MacVittie, 2016-04-01

Microservices (and their oft-referenced BFF, containers) are beginning to take over the hearts and minds of developers across the board. It's not just startups that are adopting the loosely coupled, API-only, granular design principles of microservices; large enterprises are also getting in the game.

With increasing adoption (Datadog, a cloud infrastructure monitoring provider, has seen almost 5x growth in the 12 months since September 2014) comes the very vocal cries declaring the death of monolithic applications and their wholly inappropriate architecture as being not just outdated, but plain old bad.



But as a recent article on Gigaom noted, there are tradeoffs that need to be considered before pulling out the jackhammer and busting every monolithic into a hundred different microservices. This is not a new concept, by the way. The father of microservices, Martin Fowler, wrote about these trade-offs long ago and has cautioned what can only be deemed "blind adoption" of microservices. In fact, the Gigaom article generally makes the same points as Fowler, if in a more concise format.

If you're looking for the TL;DR on both, basically it comes down to this: microservices adds operational complexity and can negatively impact the application experience in terms of performance. Both are undesirable – and often unintended – consequences that need to be understood up front, before that guy over there in your data center starts with the allegorical jackhammer.

All that said, why the heck does Lori care? After all, neither she nor F5 is in the business of building or designing applications. F5 is going to deliver those apps whether they're monoliths or microservices or next app architecture here>.

All true. But we are about the business of building and deploying the application services that deliver those applications and recent movements like DevOps and microservices (and container fever) bring the same questions to our domain. Namely, should you decompose the app services typically deployed on an ADC platform into their more granular, application-affine services in a model that more closely aligns with microservices architecture?

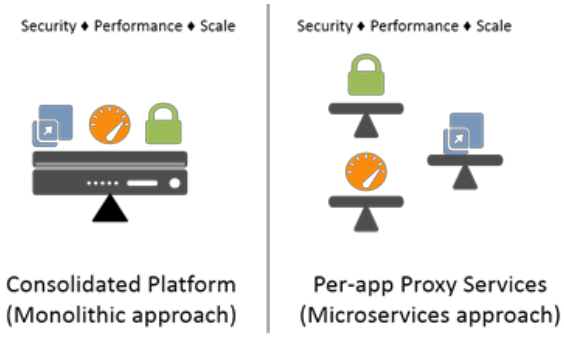
Still About Trade-offs

Regardless of whether we're talking about app architecture or app service architectures, the answer remains one of understanding the tradeoffs involved before making such a decision.

The Platform (Monolithic) Approach

This is the traditional approach to delivering the app services required to secure, scale, and optimize applications of all types. Services are deployed on a single, shared platform. Because of the architecture of the underlying proxy, this approach has the advantage of improving performance. That's because all requests (and responses) can traverse the

required services without leaving the same environment. This means no additional network hops (and the associated latency) or connections (resources, latency) are necessary. Each service can still be scaled and managed individually, but they are all dependent on a single, shared piece of hardware (COTS or custom). That means the shared hardware is a single-point of failure that will impact not one but many services.



The Per-app Proxy (Microservices) Approach

This model more closely aligns with DevOps and emerging application architectural practices. Each service is individually deployed, managed, and scaled. While this incurs additional administrative costs (there are more instances to manage, after all) some of those costs are mitigated if each service is deployed on the same platform, but does offer the ability to "mix and match" services from different providers. The advantages to this approach is that the services can be more closely associated with – and therefore included as part of – the application architecture, including integration with popular automation frameworks.

The same drawbacks – namely that of performance and increasing complexity – are associated with the decomposition of application delivery into its composite application services. Conversely, the same reasons why developers are embracing microservices and avoiding monoliths – namely a desire for agility, diversity, and modularity – are also true for application delivery.

I'm going to simply quote Martin Fowler: "Many development teams have found the microservices architectural style to be a superior approach to a monolithic architecture. But other teams have found them to be a productivity-sapping burden. Like any architectural style, microservices bring costs and benefits. To make a sensible choice you have to understand these and apply them to your specific context."

This statement applies equally well to application services. Both a traditional (monolithic) and modern (microservices) approach have costs and benefits which need to be considered within the context of the application they are going to be delivering, securing, and optimizing.

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