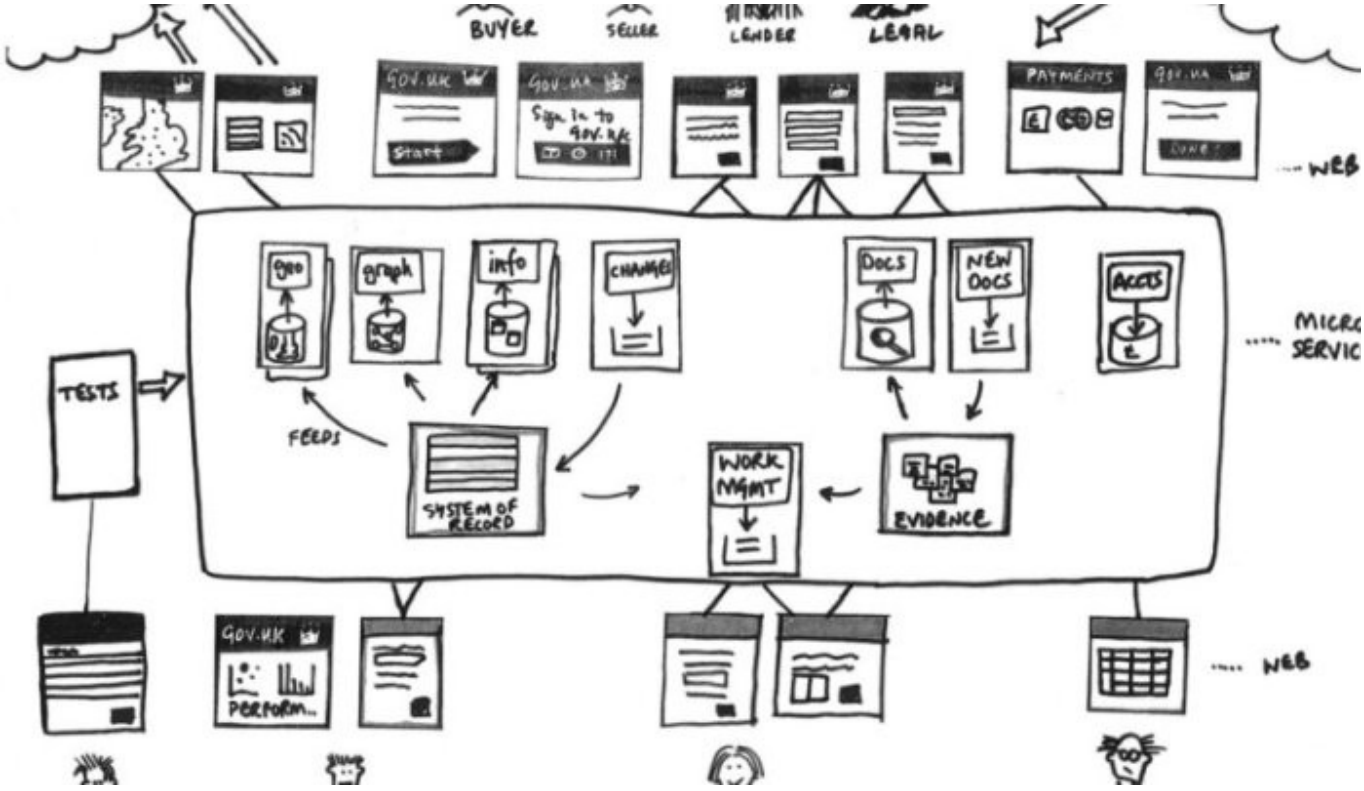
Daniel Voyce
74 Followers · About Follow

SOA vs Microservices Architecture & what this means for the future of web development?

Daniel Voyce Nov 10, 2015 · 5 min read



> SOA was not about SOAP and microservices are not about REST.
> (Even though it often seems that way!)

Service-oriented architecture (SOA) is a system design pattern that tries to abstract out the functions of a system into separate components that then provide these services to other components through a network or some other communications protocol (usually an Enterprise Service Bus (ESB), a smart "router" for messages and functions that tie these services together).

SOA has been around in some form or another for the past decade, it came about as an alternative to traditional "Monolithic" systems (single server hosting multiple services) aimed at providing better application integration and increase re-usability of service components.

The design pattern suggests that "services" are loosely coupled, unassociated (yes — it's a word!) units that provide specific functionality. Often these "services" can be external to the organisation, for example, if one integrates Salesforce through use of their API as part of their business logic, Salesforce then becomes a service.

> "If a service presents a simple interface that abstracts away its underlying complexity, then users can access independent services without knowledge of the service's platform implementation. "
> (Kishore Channabasavaiah, Kerrie Holley and Edward Tuggle, Jr. (December 16, 2003) Migrating to a service-oriented architecture, IBM DeveloperWorks.)

The idea is that each service is built as a discrete unit, so rather than exposing and defining an API, a single endpoint is exposed as the main entry point for an SOA implementation that defines the interface in terms of protocols and functionality.

> It's [SOA's] goal is in structuring multiple related applications to allow them to work together, rather than attempting to solve all of the needs in one application.

Microservices Architecture (MSA) (depending on who you ask) is either an idealistic view of SOA or (as many will have it) a rejection of SOA altogether.

I am personally in the latter of these 2, I believe that the days of architecting a system around invisible "Black-box" type services reduces the ability to understand a system, others argue it increases the simplicity, a kind of "Need to Know" basis, those people have never had to deal with sudden changes in data in a poorly designed black-box system — people rarely RTFM (if a FM was even created in the first place).

Having well architected and defined systems really helps with knowing how and where your application functions are located, traditional monolithic systems use a single database shared across multiple applications, MSA's tend to gravitate towards each service managing it's own unique database.

The advent of tools like Docker have brought MSA into the more "mainstream". The Docker philosophy is about creating services that run within their own their own container (making sense?), most of the tutorials you will see online talk about paradigms such as "service discovery" and "centralised configuration", these are central to the MSA philosophy.

Generally MSA's center around a core application with stand-alone services to accomplish other functions (for example databases, user interface & message queues to tie it all together) and communication between these items is usually through exposing and consuming some form of lightweight REST API. Each service can be developed, deployed and managed individually, and most importantly can be scaled individually.

So what does this mean for the future of web development? Should you adopt an SOA or MSA approach or stick to Monolithic?

Websites nowadays are mostly built in 2 ways, "The old way" (which is still perfectly serviceable for many types of website), where a server is commissioned that handles all of the services (UI, Database and Business Logic). For something akin to a Wordpress website there is nothing wrong with this and an MSA or SOA architecture would likely be overkill for this.

"The new way" (actually it's not that new) is probably just as familiar to many developers nowadays. An API is created based on a datasource, that then drives a front end that monitors the API data and displays changes accordingly (AngularJS / Backbone for example). This automatically provides easy code re-usability because any number of frontend instances can monitor the API (which can be driven by any number of databases, over any number of servers). As services fail (which they regularly do), there are functions that manage the pool of resources that can automatically start new services as required (this is not to say that SOA type architectures can't do this it is just ingrained into the MSA ecosystem for this to happen).

The truth is that many enterprises will probably adopt something akin to an SOA approach, simply because it has been around longer and will have more proven industry experience of it working, this then becomes the de-facto standard for building enterprise applications, it's unlikely this will change any time soon. The downside to this are heavily coupled systems that often rely heavily on one another, failure is handled through limited failover capacity rather than resource allocation as with MSA.

This kind of architecture doesn't fit with the kind of applications that are being created for and by SME's and start ups where decoupled, API centric applications that can be scaled horizontally instantly are a necessity as demand increases and decreases. It allows flexible pricing and a degree of separation that isn't possible in standard SOA implementations: Need lower latency the other side of the world? float a few containers in a UK data centre and then remove them when demand decreases, you simply cant do this on-demand with other architectures.

> The promise of SOA has become reality in microservices architecture.

For the casual user or for small business websites — there is nothing wrong with the standard monolithic architecture, VM's are cheap and many can be built with the end application in mind (for instance AWS has many Wordpress images that are ready to go), why complicate matters when your use case is small and simple? Your complications will arise should you need to think about scalability, at which point you should then consider one of the other options.

You can read further about MSA vs SOA vs Monolithic architectures from last years technology forecast from PwC:
http://www.pwc.com/us/en/technology-forecast/2014/cloud-computing/features/microservices.html

Chief Architect at Digital2Go Media Networks

Originally published at https://www.linkedin.com on November 10, 2015.

Microservices Soa Soap Rest Api