CSCI E-97
Extra Credit
Microservices
Essay

Gerald Trotman
12/17/2015

Microservices might not be done justice of its summary if this didn't start by Martin Fowler's reinforcing of the rather obvious which is that: "this concept is nether novel nor innovative, its roots go back as far as Unix…" That however is not to be mistaken with the sentiment of those who feel that this is simply the rebranding of the Service-Oriented-Architecture; Fowler, Lewis, an many users who subscribe to implementation of Microservices will assure you that it is not. So what exactly are Microservices? One way to answer this is to first look at its more used counter part for Application Development. All too common, applications are made using a rigid hexagonal architecture were all of the core business logic lies at its center of its design surrounded by the APIs. When it comes time for implementation, it gets deployed as one big Monolith. It's approach is simple and proven effective. Ideally, the Application still holds true the *Design Principles* that we have taken away from this class. The application stays true to Divide and Conquer and Separation of Concerns. Occam's Razor, and the notion of "keeping things simple but no more". It was coded to interfaces and not implementations. It identifies what is changed versus what stays the same. It has modularity that's even loosely coupled and cohesive. All of the moniker that SOLID represents is being observed in all of its Monolithic glory. The Application becomes successful… But then what? You've accounted for scale of course to some extent but what happens when the needs of the Application evolve? Well one thing for sure is that like anything, as it grows, it has this natural inclination to become more complex. Then what? Will it easily adapt to the newer technological solutions that might not have been there from inception? At what point should I become concerned that all of my code is riding on one data set? Or the fact that one bug could systematically undue literally millions of lines of code of my application.

It's these questions that Microservices tries to answer. Martin Fowler and his experienced practitioner colleague James Lewis are credited with spearheading its formality. They would say that:

> "In short, the micro service architecture stye is an approach to develop a single application as a suite of small services each running in its **own process** and communicating with lightweight mechanisms, often an HTTP resource API. These services are **built around business capabilities** and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage techniques"

As loaded as this assessment of microservices may be, it provides a very wholistic sense of what micro services provide that the current implementation of how Applications are made might not. If you extract out this assessment of microservices, you can see that there is are principles of their own that Microservices brings to the table. At the heart of Microservices is this concept that you break down a traditional monolith into components via services. Organizing around Business Capabilities is something that Fowler points out is not only important to Microservice design, but fundamental to approaching Applications in general. Microservices reinforces this notion of what Fowler calls "Smart endpoints and dumb pipes". Microservices intend to decentralize governance and data management. Microservices address designing for failure. And last but not least, Microservices builds in anticipation of evolutionary design. Running through this list of attributes, one can't help but get the sense that there is this striking similarity between the characteristics that Martin Fowler uses to describe Microservices and that of the Design Principles Patterns and Projects. At their core, they both emphasize the importance of Divide and Conquer and to Separate Concerns. The whole point of Microservices is to deliver componentization via services, to decentralize governance and data management. In Fowlers explanation of Microservices contribution of "Smart endpoints and dumb pipes" he explicitly mentions that "Microservices aim to be as decoupled and as cohesive as possible", and that "They own their own domain logic".

The NGINX site does an excellent job explaining Microservices in using their Taxi Cab Service (very similar to our smart car service. So similar in fact that I actually wish I used Microservices in my design). This answered the question of viability of building applications with Microservices. It starts by taking a Monolithic approach and then breaking it down into services and their corresponding REST APIs which are inherently light weight. There are a couple of things to note here. NGINX is emphasizing the fact that the backend service management of the REST APIs to each other is done by an API Gateway

that does the actual load balancing caching, access control, API metering, and monitoring (which, coincidentally is a product that they provide). This is very important in understanding just exactly how Microservices talk/coordinate wth each other. Another thing to comment on is the fact that this is one of the more celebrated differences between Microservices and SOA because SOA is associated with complex more proprietary protocols such as WS- or BPEL which is a central tool which Microservice advocates feel defeats the point. It's also worth mentioning that Microservices have data base storage per service while SOA still believes in enterprising one data storage service for the entire Application. It might produce redundant data (which isn't necessarily a bad thing) but it allows for versatility of hardware and the data persists in a "polyglot" fashion.

The advantages and disadvantages to Microservices versus Monoliths certainly favor Microservices more. But it's was very interesting taking a look at Martin Fowler's graphical analysis of productivity versus base complexity of both the Monolith and Microservices methods. Fowler coins this term "MicroservicesPremium" which means that there is this initial dip in productivity in learning Microservices before the benefits kick in. This of course depends on the team. But it concludes that the long lasting effects of productivity as complexity increases are steady. The Monolith decreases productivity as complexity increases which is where the need for other approaches like Microservices are in need in the first place. Most compelling though is the conclusion which was that Monoliths are still somewhat a necessity as far as understanding them is concerned and might be a good starting point before implementing Microservices (though Microservices advocates disagree).

In conclusion, Microservices as Martin Fowler points out draws is philosophy from the roots of philosophy as the Design Principles and Patterns. It is an approach to Application Development that sprung out of necessity to tackle the challenges of Monolithic Application Development despite the fact that its concepts date as far back as Unix. The viability of Microservices being used in applications has already been proven successful in industry with companies like Netflix and Google but the success greatly depends on the capabilities of the business.