UNRAVELLING MICROSERVICES: THE HONEYMOON IS OVER

Microservices architecture has been a hot topic since the middle of the last decade with seemingly everyone looking to make the shift to it. But has it lived up to the hype? Maybe not.

Uber: Refactoring thousands of microservices

Uber is one big name pulling back from microservices, with Gergely Orosoz, who at the time was an engineering manager at the company, sharing the news on Twitter:

Kelsey Hightower: "Monoliths are the future"

If not microservices, then what? Kelsey Hightower writes on changelog.com that, "Monoliths are the future because the problem people are trying to solve with microservices doesn't really line up with reality…and I've done this before, gone from microservices to monoliths and back again."

Sam Newman: Microservices should be a last resort

Cloud consultant Sam Newman has his own take on microservices. In his presentation on Monolith Decomposition Patterns at QCon London 2020, he caid the monolith is not the enemy and microservices should not be a default choice. In his book, *Monolith to Microservices*, Newman advised people to focus on the outcome, not the technology, and to always remember the goal is independent deployability.

What's going on with microservices?

Why have so many projects become unmanageable with microservices, despite its promise of simplicity and flexibility? Are monoliths better, after all?

Microservices have always been positioned as a solution for monolithic codebases. But are monoliths necessarily a problem? According to Wikipedia's definition, a monolithic application is self-contained and independent from other computing applications. A monolith can be beneficial if

speed is more important than the perfect architecture, which is often the case for startups working with limited funding that need to start selling to survive.

Microservices could ultimately make life easier for those startups, but they come with upfront costs that require maturity to bear. To be clear, microservices do not "fix" monoliths. The real problem that microservices should solve is the inability to deliver on business goals because a system can't support exponential growth, or the business can't support unpredictable change costs.

The uncontrollable cost of change is not a property of a monolith but rather of a big ball of mud.

Big Ball of Mud

A big ball of mud is a haphazardly structured, sprawling, duct-tape-and-baling-wire, spaghetti-code jungle. These systems show unmistakable signs of unregulated growth, and repeated, expedient repair. Information is shared promiscuously among distant elements of the system, often to the point where nearly all the important information becomes global or duplicated.

Why microservices?

A good microservices architecture is elegantly simple, but it is not easy, fast, or accidental. There are many great, mature, large-scale microservices examples out there today—including Twitter, Amazon, Spotify, and LinkedIn—but there is a reason they all started out as monolithic applications and later evolved. Never let pursuit of technical perfection get in the way of business success. A pragmatic architecture means that technology choices should support the business needs, not the other way around.

We build systems, not sets of services

At BMC, we use a microservices-based architecture to optimize a system's design, not the design of individual services. Microservices cannot, and will never be, completely decoupled nor fully independent. You cannot build a system out of independent components.

So, what is the answer? Microservice? Miniservice? Macroservice? I'll discuss that in the next installment.