MicroservicePremium

13 May 2015

Martin Fowler

MICROSERVICES

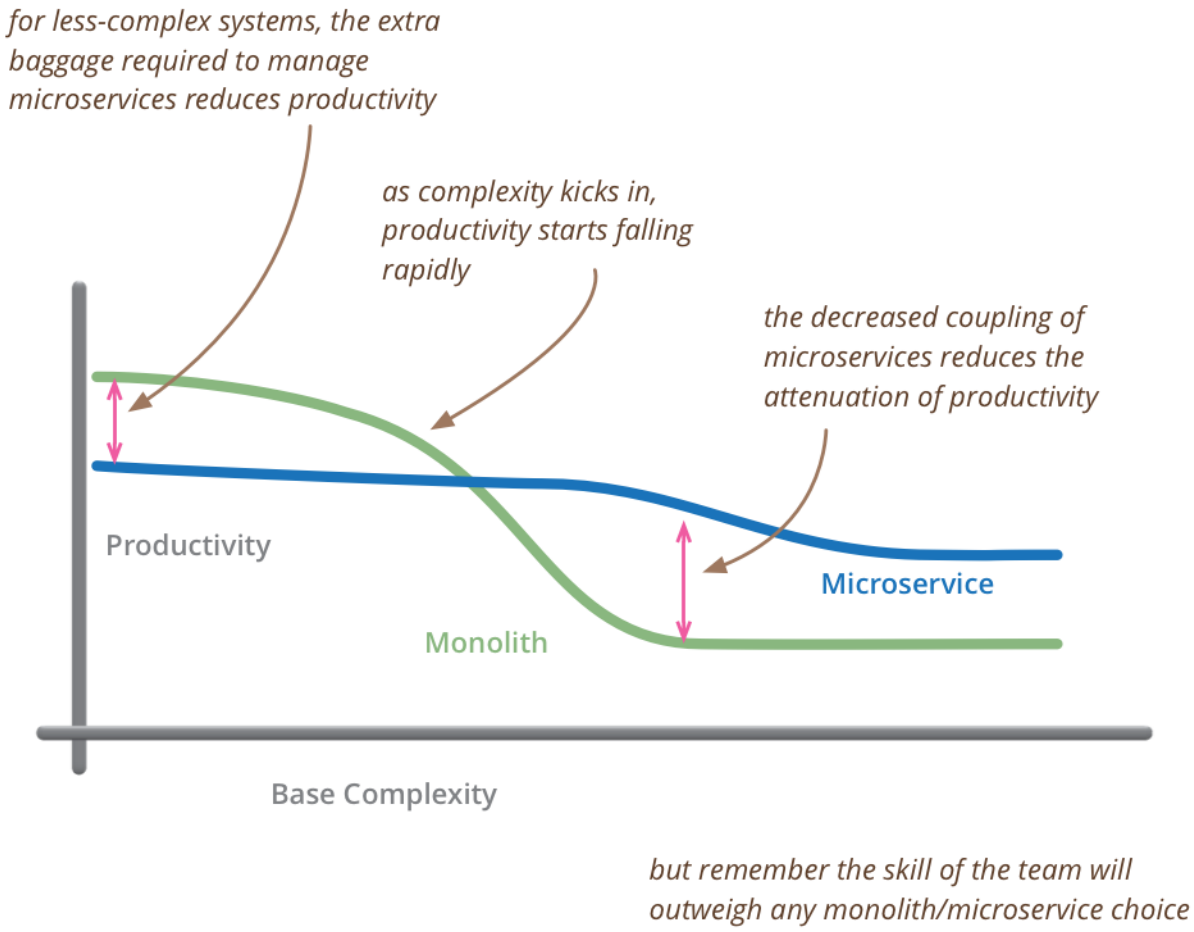
The microservices architectural style has been the hot topic over the last year. At the recent O'Reilly software architecture conference, it seemed like every session talked about microservices. Enough to get everyone's over-hyped-bullshit detector up and flashing. One of the consequences of this is that we've seen teams be too eager to embrace microservices, [1] not realizing that microservices introduce complexity on their own account. This adds a premium to a project's cost and risk - one that often gets projects into serious trouble.

While this hype around microservices is annoying, I do think it's a useful bit of terminology for a style of architecture which has been around for a while, but needed a name to make it easier to talk about. The important thing here is not how annoyed you feel about the hype, but the architectural question it raises: **is a microservice architecture a good choice for the system you're working on?**

> *any decent answer to an interesting question begins, "it depends..." -- Kent Beck*

"It depends" must start my answer, but then I must shift the focus to what factors it depends *on*. The fulcrum of whether or not to use microservices is the complexity of the system you're contemplating. The microservices approach is all about handling a complex system, but in order to do so the approach introduces its own set of complexities. When you use microservices you have to work on automated deployment, monitoring, dealing with failure, eventual consistency, and other factors

that a distributed system introduces. There are well-known ways to cope with all this, but it's extra effort, and nobody I know in software development seems to have acres of free time.



So my primary guideline would be **don't even consider microservices unless you have a system that's too complex to manage as a monolith**. The majority of software systems should be built as a single monolithic application. Do pay attention to good modularity within that monolith, but don't try to separate it into separate services.

The complexity that drives us to microservices can come from many sources including dealing with large teams [2], multi-tenancy, supporting many user interaction models, allowing different business functions to evolve independently, and scaling. But the biggest factor is that of sheer size - people finding they have a monolith that's too big to modify and deploy.

At this point I feel a certain frustration. Many of the problems ascribed to monoliths aren't essential to that style. I've heard people say that you need to use microservices because it's impossible to do ContinuousDelivery with monoliths - yet there are plenty

of organizations that succeed with a cookie-cutter deployment approach: Facebook and Etsy are two well-known examples.

I've also heard arguments that say that as a system increases in size, you have to use microservices in order to have parts that are easy to modify and replace. Yet there's no reason why you can't make a single monolith with well defined module boundaries. At least there's no reason *in theory*, in practice it seems too easy for module boundaries to be breached and monoliths to get tangled as well as large.

We should also remember that there's a substantial variation in service-size between different microservice systems. I've seen microservice systems vary from a team of 60 with 20 services to a team of 4 with 200 services. It's not clear to what degree service size affects the premium.

As size and other complexity boosters kick into a project I've seen many teams find that microservices are a better place to be. But unless you're faced with that complexity, remember that the microservices approach brings a high premium, one that can slow down your development considerably. So if you can keep your system simple enough to avoid the need for microservices: do.

Notes

1: It's a common enough problem that our recent radar called it out as Microservice Envy.

2: Conway's Law says that the structure of a system follows the organization of the people that built it. Some examples of microservice usage had organizations deliberately split themselves into small, loosely coupled groups in order to push the software into a similar modular structure - a notion that's called the Inverse Conway Maneuver.

Acknowledgements

I stole much of this thinking from my colleagues: James Lewis, Sam Newman, Thiyagu Palanisamy, and Evan Bottcher. Stefan Tilkov's comments on an earlier draft were instrumental in sharpening this post. Rob Miles, David Nelson, Brian Mason, and Scott Robinson discussed drafts of this article on our internal mailing list.