Monolith Vs Microservice Vs Serverless — The Real Winner? The Developer

Elliot Forbes Follow
Jan 13, 2018 · 5 min read

Microservices and Serverless have certainly grown a lot in popularity over the past few years. Everyone these days wants to replicate the successes of companies such as Netflix and create a system that is fault-tolerant and resilient to the nth-degree.

Many lead developers and architects are now realising some of the major advantages of both microservice and serverless architecture styles and have been able to adopt them with varying degrees of success around the globe. The one underlying theme that seems to come across from evangelicals of these newer architecture styles is that the world is black and white. If you are going to leverage a microservice architecture, **everything** has to follow this architecture style.

The Death of Microservice Madness in 2018

I came across Dave Kerr's post on his site recently that talked about "the Death of Microservice Madness in 2018". He seemed to highlight the very numerous problems that teams buying into this microservice and serverless styles seem to be hitting day after day.

The Death of Microservice Madness in 2018

Microservices became a very popular topic in over the last couple of years. 'Microservice madness'...

www.dwmkerr.com

Some of the comments found on /r/programming were, in my opinion, overly critical of these newer architecture styles. I feel that this is slightly unfair and bashing these newer approaches seems to be antithetical to what we, the developer community, want to achieve.

We are forever going to be building more and more complex systems that solve new problems in ways we could never have dreamt of 20–30 years ago. With these new architecture styles, we can approach problems in ways that would never have been thought possible way-back-when.

Understandably, this transition from the traditional monolith structure will ultimately be painful as newer practices have to be developed to try and minimize the amount of self-harm that is done. As time goes by, we will continue to see improvements in tooling around these distributed system as more companies continue to feel the pain points highlighted in Dave Kerr's post.

Serverless—The Advantages

Imagine the scenario where you have a new requirement handed down to you from upper management. This requirement comes relatively out of the blue and doesn't fit nicely into any of your current systems.

The options are:

- Shoe horn your new requirement into one of your existing systems
- Architect a new system that handles this requirement
- leverage the likes of AWS Lambda.

Now many of you may think option 1 sounds ideal here, but if you are extending the one of your key systems with loads of these new relatively left-field requirements, the likelihood for something bringing down your system and costing your company money increases.

Option 2 is also a very valid option but then you are left with the increased development time, more time worrying about the operations side of things for this newer system and more time spent in the initial design period as you have to ensure this new system is resilient.

Option 3 seems to be the favourite here. You simply write the code that will fulfil this requirement, deploy it to the likes of AWS lambda and mark the jira as complete. This may be oversimplifying the development but AWS lambda does a hell of a lot of heavy lifting for you, including things like resiliency, horizontal scaling and so on.

Microservices—The Advantages

Imagine a second scenario where you have a monolithic system that is currently performing optimally. A requirement comes from management that you need to start doing X within your monolith.

'X' could in theory become a bottleneck in terms of performance for your monolith, you may have to start deploying your system on more powerful machines or possibly across more machines thus increasing costs.

The advantages of a microservice based approach in this scenario is that you can effectively design, develop a microservice that satisfies 'X' and can be called from within your monolith through your favourite communication protocol.

This allows you:

- To independently scale 'X' to cope with variable demands on the system.
- To reduce the risk of 'X' bringing down your system in Production at 3 in the morning.

Monoliths—The Advantages

Say you are a startup, time to market is incredibly important and you need to get your product out there as soon as possible. In this situation the advantages are:

- Less time worrying about the complexities of distributed systems
- Less time worrying about the deployment of your systems
- Simplified Architecture

From the very start of our careers in University we tend to fall into this style of architecture. It's the default and the most widely used right now and the easiest to develop. Whilst it does come with a lot of issues, it does really simplify the earlier stages of a product before it matures fully.

Hybrid Approaches—The Compromise

There is absolutely nothing stopping you going for a hybrid approach and employing monolithic architecture styles when needed. The vast majority of legacy systems out there employ this architecture style and *it's part of the reason that they are still around.*

> *The key thing to note is that legacy systems are only legacy because they've been successful enough to last this long.*

One of the key things I feel we should actively avoid doing is change for the sake of change. If a legacy system has been running for the past 10 years with little or no issue, leave it running and work on providing value in other various ways as opposed to re-architecting the system into something that will ultimately provide no real value over the older style of architecture.

Conclusion

When it comes down to it, as we continue to improve, these newer architecture styles simply represent another tool that good software developers can leverage in order to get products out the door sooner.

It's up to us as developers to learn the advantages of these newer trends and try to understand how they can be best used in order to improve the systems that we are working on.

We need to provide constructive feedback and in the words of Mark Watney: *"Work the problem"*

Hopefully you found this article thought provoking! I'm always eager to hear your counter-arguments or whether or not you agree so please feel free to leave a comment in the comments section below! If you want to support my writing, as well as donate to the EFF then please feel free to check out my newest book which is currently under production:

An Introduction to Cloud Development

This book provides a gentle introduction to the world of cloud development. We will be taking a...

leanpub.com