highscalability.com

One Team at Uber is Moving from Microservices to Macroservices - High Scalability -

10-13 minutes

There may be an undiscovered tribe deep in some jungle somewhere that hasn't made up their mind on microservices, but I doubt it. People love microservices or love to hate microservices. There's not much in between.

So it means something when even a team at a company like Uber announces a change away from microservices to something else. What? Macroservices. But we'll get to that. Think what you want about Uber the company, but from a software perspective Uber has been a good citizen.

Gergely Orosz, an Engineering Manager on the Payments Experience Platform at Uber, in a tweet signaled a change in architectural direction:

> @GergelyOrosz: For the record, at Uber, we're moving many of our microservices to what @copyconstruct calls macroservices (well-sized services).
>
> Exactly b/c testing and maintaining thousands of microservices is not only hard - it can cause more trouble long-term than it solves the short-term.
>
> - Microservices _do_ help teams move fast early on.
>
> - By the time you realize fewer services would be great, it's too late. You need to solve the "hard" part of many services.
>
> - We keep adding more services, but also retiring, and putting more thoughtfulness in new ones.

> @GergelyOrosz:
>
> Yes, we're doing this and the approach touches on a pain point of many microservices.
>
> Every service needs to support tenancies, including many stateless ones.
>
> We also need to retrofit much of this work with the existing services. For new services, we just add it from the start.

> @GergelyOrosz:
>
> 1. Microservices helped (and still help) move fast, be autonomous & experiment.
>
> 2. The more mature an area becomes, the more sense / easier to reason "well-sized services" are.
>
> I'll have to write up thoughts in longer form later.

> @GergelyOrosz:

> I'm probably overdue with a post on the hard-learned downsides of microservices. There's lots of talk of the blissful honeymoon period, but people rarely talk how they later get into nasty fights with microservices, then make up but change a few things. For good.

> Gregdoesit:
>
> I wrote that tweet that is making the rounds. Not many things fit in 280 characters and Twitter being immutable, there's not many things you can go back to clarify. So let me give some more details on this forum.

1. I speak for my experience, for my team, not for all of Uber. Heck, we have hundreds of teams, 95% of whom I don't know. And teams are autonomous and decide how and what they do (including following guidelines or ignoring them partially or fully) - even if I wanted to, I couldn't make sweeping statements.
2. Uber has - and still has - thousands of microservices. Last I checked it was [around 4,000]. (https://eng.uber.com/optimizing-observability/). And, to be very clear: this number is (and will keep) growing.
3. I've been working here for almost 4 years and see some trends in my org / area (payments). Back in the day, we'd spin up a microservice that did one, small thing just like that. We had a bunch of small services built and maintained by one person. This was great for autonomy, iteration speed, learning and making devops a no-brainer. You could spin up a service anytime: but you'd be oncall for it.
4. Now, as my area is maturing and it's easier to look ahead, as we create new platforms, we're doing far more thoughtful planning on new services. These services don't just do one thing: they serve one business function. They are built and maintained by a team (5-10 engineers). They are more resilient and get far more investment development and maintenance-wise than some of

those early microservceis. Cindy called these macroservices and I said we're doing something similar. The only difference in what we do is a service is owned by one team, not multiple teams.

5. While many microservices are evolving like this, the majority, frankly, stays as is. Thousands of microservices bring a lot of problems that need to be solved. Monitoring. Testing. CI/CD, SLAs. Library versions across all of them (security, timezone issues). And so on. There are good initiatives we keep doing - and sharing what works and open sourcing some of the tools we build to deal with the problems, as they pop up. Like testing microservices with a multi-tenancy approach. Distributed tracing across the services. All of this is a lot of investment. Only do microservices at scale if you're ready to make this investment.

   So no, Uber is not going no-microservices like I'm seeing many people interpret it. It's not even going to less microservices. And when I said "we're moving", that was not exact phrasing. New microservices are more thoughtfully created in my team and in my org. These are "larger" services than some of the early, small, focused, microservices.

   Microservices worked well at Uber in many ways and keep helping in others areas. There are problems, of course, and you deal with the problems as you go. This would be the same with e.g. a monolith with thousands of developers, SOA with thousands of developers or {you name whatever you do} with thousands of developers. The number of services is still growing, as a whole, as the business grows - though in some orgs, like mine, they are at level, or even going down a bit (though this is not the goal itself). But not all microservices are equal any more. The critical ones look less like your classic microservice - or at least what I called microservices years back.

   On another note: everyone interprets the name "microservice"

> differently. I'll write a post summarizing my experiences with the ups and downs of microservices at scale. For now, hopefully this gives some more color.

> Gregdoesit:
>
> Uber went from monolith to SOA in 2015. This SOA followed a microservice-based architecture. And we've been sharing what we learned along the way: the steps it usually takes to build a microservice, addressing testing problems with a multi-tenancy approach or how and why we use distributed tracing. We also open sourced some of our tools like Jaeger, which is part of the Cloud Native Foundation's graduated projects, alongside Kubernetes and Prometheus...All of these can serve as inspiration: but the end of the day, you need to make decisions in your environment that you think will work best. Anyone copying the likes of Google, Uber, Shopify, Stack Overflow or others when they're not even similar in setup will be disappointed.

> @copyconstruct:
>
> - Microservices are hard.
>
> - Building reliable and testable microservices is a lot harder than most folks think
>
> - Effectively *testing* microservices requires a ton of tooling and foresight.
>
> - A Netflix/Uber style microservices isn't required by many (most?) orgs.
>
> - Macroservices?
>
> Macroservice:
>
> - not a monolith
>
> - Has no more than 20 devs/3 teams working on the service (5 pizza rule?)

> - may or may not have/need monorepo. Dependency management becomes a lot easier (though still non-trivial) the fewer the services/repos
>
> - better observability, debugging

The world will of course go crazy that we have a new semi-branded term like macroservice.

How is a macroservice different from the plain old service we've known for decades? Who cares. Names are a product of their times. Names simply serve as a scaffolding for discussions. This isn't magic. A name isn't a secret symbol that must be kept safe lest a dark arts practitioner turn you into their meat puppet. A name is just a gathering place, a marker, that helps us find our way. Take a deep breath, let any name induced angst go.

As you might imagine there has been a lot of response. Much of it gleefully extolling the much deserved end to the microservices scourge. Microservices have always been a bad idea is the general consensus of the dissenterati.

> @sandofsky:
>
> Uber in 2016: "We have thousands of microservices."
>
> Everyone: "That sounds insane."
>
> Uber in 2020: "It turns out that was insane."

> @dhh:
>
> The amount of pain that's been inflicted by the overeager adoption of microservices is immense.
>
> In addition to the Majestic Monolith, someone should write up the pattern of The Citadel: A single Majestic Monolith captures the majority mass of the app, with a few auxiliary outpost apps for highly specialized and divergent needs.

But it's not all negative.

> @saikishore001:
>
> We have found quite a bit of success at Bayer with microservices. For us maintaining a large monolith was a nightmare.. now, much much better with microservices architecture.

> @Carnage4Life:
>
> There are 2 key lessons from Uber moving away from microservices after being a big proponent of them in 2016
>
> 1. The trade offs that big companies at large scale make may not be right for your startup
>
> 2. Big companies also make poor architectural choices so beware cargo culting

> @adam_zethraeus:
>
> Uber only really did that in order to avoid coordination cost. Building without concern for reuse or consolidation was, roughly speaking, explicitly encouraged. e.g. Uber's China team reproduced a bunch of tertiary architecture to move faster. (Worked in the short term!)

There's an economic argument for the architectural hype cycle that's also worth considering:

> @ridingwithrails
>
> During Internet crashes and recessions monoliths always win. Folks realize it is hard to keep ten teams that use ten different systems around ... bye Felicia!:

> @sandofsky
>
> Every tech talk should have to disclose their venture capital burn rate. You can get away with just about anything when throwing someone else’s money at your problems.

The glee in the potential downfall of microservices is not a great

look for the industry. We need to focus on getting things right rather than being right. I know, being right is at the very heart of competitive nerdom.

Change is how we progress and adding friction to the change process doesn't help anyone. Uber maturing, learning, refactoring is a good thing. It's not an admission of failure or even evidence of poorly made decisions early on. When you have a lot of money and the BHAG of taking over the world, microservices make a lot of sense. As does retrenching and consolidating when that phase of your growth comes to an end. What do you do when facts on the ground change?

Face it, we have next to no idea of how to build software. I'm convinced part of the reason microservices took off as they did is because it at least gave programmers a coherent theory of how to build programs.

Everyone gives their own pet alternative to microservices, but there is no consensus, we have no systematic theory. Witness this whole discussion as evidence.

Software is a mess.

Related Articles

- Why We Leverage Multi-tenancy in Uber's Microservice Architecture
- Distributed architecture concepts I learned while building a large payments system
- Operating a Large, Distributed System in a Reliable Way: Practices I Learned
- Continuous Testing at Scale
- Service-Oriented Architecture: Scaling the Uber Engineering

Codebase As We Grow