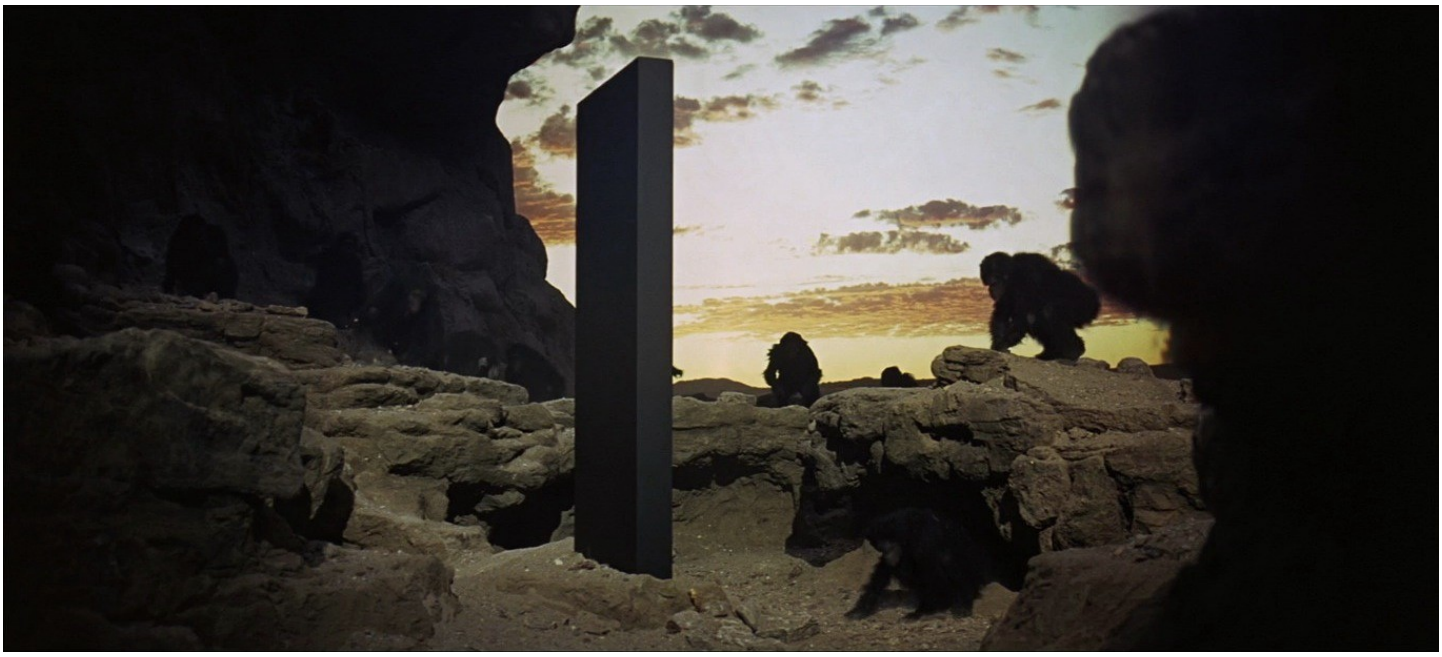
Monolith vs microservices: which architecture is right for your team?

Jake Lumetta Follow

Jan 17, 2018 · 8 min read



The iconic Monolith. Your next application?

My good friend Darby Frey recently kicked off a greenfield project after assuming his new role as Sr. Platform Engineering Lead of Gamut. Despite starting out with monolith at his previous company Belly, he discovered that—in the right circumstances—starting with a monolith isn't always the best way to go.

At Belly, Darby and his team broke down their monolith into a fairly large microservices architecture. They managed to get it to a good place, but only after months of trials and tribulations migrating to microservices.

With this experience fresh in his mind, he approached his new project at Gamut a bit more cautious of microservices.

"I was firmly a member of Team Monolith. [I thought] let's build a single application and just pull things apart later if we start to feel pain," he said.

While this was a greenfield project, Darby's team was small, and he had aggressive timelines, so on the surface, a monolith seemed like the obvious choice.

"[But with this new project], I was anxious to not repeat the mistakes of the past."

And with that, he found himself faced with a decision we all struggle with: should we start with a monolith or microservices, and how do we decide?

Conventional Wisdom About Monoliths

Many startups begin with a monolith, because its very structure lends itself well to small teams and offers other advantages like less operational overhead. Also, monoliths often just have one massive code base. Server side application logic, front end client side logic, background jobs, etc, are all defined in the same code base. This means that if developers want to make any changes or updates, they need to build and deploy the entire stack all at once.

A monolith isn't a dated architecture that we need to leave in the past. In certain circumstances, a monolith is ideal. I spoke to Steven Czerwinski, Head of Engineering at Scaylr and former Google employee, to better understand this.

"Even though we had had these positive experiences of using microservices at Google, we [at Scaylr] went [for a monolith] route because having one monolithic server means less work for us as two engineers," he explained. This was back in the early beginnings of Scalyr.

In other words, because his team was small, a unified application was more manageable in comparison to splitting everything up into microservices.

The Alternative of Microservices

Zachary Crockett, CTO of Particle, told me this during an interview:

> *"When discussing microservices, people tend to focus on one end of that spectrum: many tiny applications passing too many messages to each other. At the*

> *other end of the spectrum you have a giant monolith doing too many things. For any real system, there are many possible service oriented architectures between those two extremes."*

The microservice architectural style is an approach to developing a single application as a suite of small services. Each runs in its own process and communicates with lightweight mechanisms, often an HTTP resource API.

These services are built around business capabilities and are 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 may use different data storage technologies.

Microservices do have distinct advantages:

- **Better Organization:** Microservice architectures are typically better organized. Each microservice has a very specific job, and it is not concerned with the jobs of other components.
- **Decoupled:** Decoupled services are also easier to recompose and reconfigure to serve the purposes of different apps (for example, serving both the web clients and public API). They also allow for fast, independent delivery of individual parts within a larger, integrated system.
- **Performance:** Under the right circumstances, microservices can also have performance advantages depending on how they're organized. It's possible to isolate hot services and scale them independently of the rest of the app.
- **Fewer Mistakes:** Microservices enable parallel development by establishing a hard-to-cross boundary between different parts of your system. By doing this, you make it hard—or at least harder—to do the wrong thing: namely, connecting parts that shouldn't be connected, and coupling too tightly those that need to be connected.

But they also create unique challenges:

- **Cross-cutting Concerns Across Each Service:** As you're building a new microservice architecture, you're likely to discover lots of

cross-cutting concerns that you did not anticipate at design time. You'll either need to incur the overhead of separate modules for each cross-cutting concern (i.e. testing), or encapsulate cross-cutting concerns in another service layer that all traffic gets routed through. Eventually, even monolithic architectures tend to route traffic through an outer service layer for cross-cutting concerns, but with a monolithic architecture, it's possible to delay the cost of that work until the project is much more mature.

- **Higher Operational Overhead:** Microservices are frequently deployed on their own virtual machines or containers, causing a proliferation of VM wrangling work. These tasks are frequently automated with container fleet management tools.

Which Model Is Right For You?

Interviews with dozens of CTOs taught us that one approach does not fit all. Their experiences serve as a useful rubric for you to decide between a monolith and microservices. Answering these questions should guide you in the right direction:

Are You In Familiar Territory?

Darby and his team at Gamut were able to delve directly into Microservices because he had experience with eCommerce platforms, and his company had a wealth of knowledge concerning the needs and demands of their customers. If he had been traveling down an unknown path, a monolith may have actually been the safer option.

Similarly, startups are often born out of pains experienced at previous companies. In those scenarios, sometimes it's quite clear that scaling is going to be a primary requirement, especially in infrastructure-based services like cloud log management.

Is Your Team Prepared?

Does your team have experience with microservices? What if you quadruple the size of your team within the next year—are microservices ideal for that situation? Evaluating these dimensions of your team is crucial to the success of your project.

Julien Lemoine, CTO at Algolia, chimed in on this point:

> *" We have always started with a microservices approach. The main goal was to be able to use*

> *different technology to build our service, for two big reasons:*
>
> *1) We want to use the best tool for each service. Our search API is highly optimized at the lowest level and C++ is the perfect language for that. That said, using C++ for everything is a waste of productivity, especially to build a dashboard!*
>
> *2) The want the best talents and using only one technology would limit our options. This is why we have different languages in the company, Go is less perfect than C++ when you want to optimize everything at the millisecond level but is the perfect language when performance is still key (processing of logs where we process several terabytes of logs per day, using ruby or python would be a waste of CPU).*"

If your team is prepared, starting with microservices is wise as it allows you to get used to the rhythm of developing in a microservice environment, right from the start.

How's Your Infrastructure?

In reality, you're going to need cloud-based infrastructure to make microservices work for your project. David Strauss, CTO of Pantheon, explained it thus:

> *"[Previously], you would want to start with a monolith because you wanted to deploy one database server. The idea of having to set up a database server for every single microservice and then scale out was a mammoth task. Only a huge, tech-savvy organization could do that.*
>
> *Whereas today with services like Google Cloud and Amazon AWS, you have many options for deploying*

> *tiny things without needing to own the persistence layer for each one."*

Evaluate The Business Risk

You may think microservices is the "right" way to go as a tech-savvy startup with high ambitions. But microservices pose a business risk. David Strauss explained:

> *"A lot of teams overbuild their project initially; everyone wants to think their startup will be the next unicorn and that they should, therefore, build everything with microservices or some other hyper-scalable infrastructure. But that's usually wrong, almost all the time."*

One example of this from his early days at Pantheon was a system that was limited to a single VM. They thought it would be a month or two until they'd be forced to scale it. It ended up being over a year—and they ended up scaling it a completely different way than they had anticipated.

He went on to say that, in these cases, the areas that you thought you needed to scale are probably not the parts that will need to scale first. This results in misplaced effort even for the systems that will need to scale.

Context is Key—Rules of Thumb For Choosing an Architecture

We've taken hours of interviews with successful CTO's and distilled them into general guidelines to follow when choosing your service architecture.

When To Start With A Monolith

Here are some scenarios that indicate that you should start your next project using monolithic architecture.

- **Your Team Is At Founding Stage:** Your team is small, between 2–5 members, and is thus unable to tackle a broader and high-overhead microservices architecture.

- **You're Building An Unproven Product or Proof of Concept:** Are you building an unproven product in the market? If it's a new idea, it's likely going to pivot and evolve over time, so a monolith is ideal to allow for rapid product iteration. Same applies to a proof of concept where your goal is just to learn as much as possible as quickly as possible, even if you end up throwing it away.
- **You Have No Microservices Experience:** If your team has no prior experience with microservices, unless you can justify taking the risk of learning "on the fly" at such an early stage, it's likely another sign you should stick to a monolith to start.

When To Start With Microservices

Here are some scenarios that indicate that you should start your next project using microservices:

- **You Need Quick, Independent Service Delivery:** Microservices allow for fast, independent delivery of individual parts within a larger, integrated system. Note that, depending on your team size, it can take time to see service delivery gains versus starting with monolith.
- **A Piece of Your Platform Needs to Be Extremely Efficient:** If your business is doing intensive processing of petabytes of log volume, you'll likely want to build that service out in a very efficient language (i.e. C++) while your user dashboard may be built in Ruby on Rails.
- **You Plan To Grow Your Team:** Starting with microservices gets your team used to developing in separate small services from the beginning. And having teams separated by service boundaries makes it much easier to scale up your team when you need to without introducing exponential complexity.

Choose the Architecture You Can Comfortably Live With

Microservices has become an increasingly popular service architecture, but it's essential to understand whether it's the best fit for your project. Your own context, evaluated against the above considerations, is the key to deciding if you should start with monolith or microservices.

Jake is the CEO of ButterCMS. For more content like this, follow @ButterCMS on Twitter and subscribe to our blog.