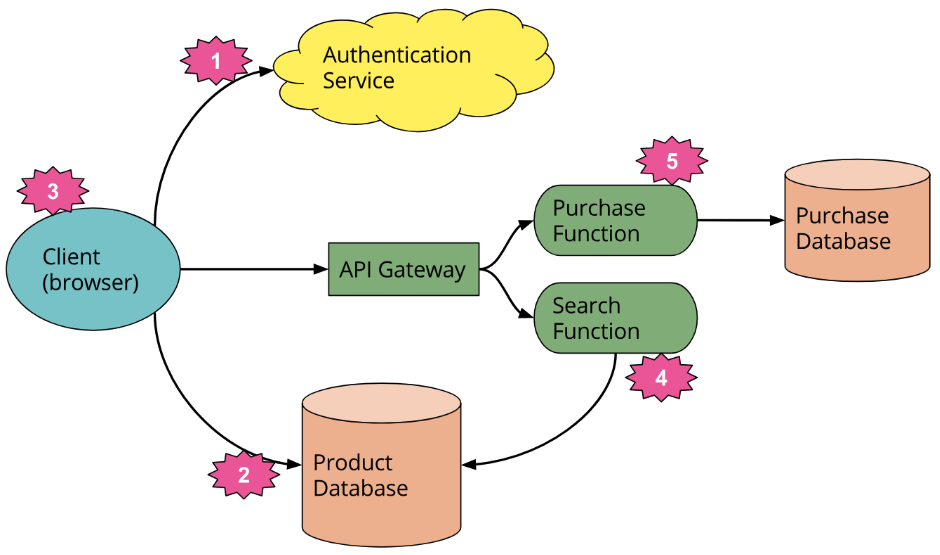
Revisiting “Serverless Architectures”



I started writing “Serverless Architectures” in May 2016. At that point I’d recently finished up at my previous job, had just been to the first Serverlessconf, hadn’t done much writing in a while, had some time on my hands, and so decided to put a few ideas together. I thought a few folks might be interested. Of course publishing it on Martin Fowler’s site was always going to get it to a wider audience (thanks Martin!), but I didn’t realize quite *how* wide.

Fast forward to two years later and the article has had more than half a million visits, regularly appears in the top five Google search results for “Serverless”, and helped launched Symphonia—my company with John Chapin. But it was also starting to show its age—Serverless is a highly dynamic area and two years is a lifetime in this world. That, and a few typos, convinced me I needed to go back and revisit it.

The fundamental structure of the article hasn't changed—clearly people like it and I didn't want to mess that up. But here's a few points that I added or updated based on advances over the last couple of years.

What is Serverless?

Two years ago I defined Serverless as the combination of BaaS and FaaS, and I'm pretty happy with that. I was slightly worried that it would come to mean "just FaaS", but lots of folk have helped steer it away from that. The earliest reference I could find to FaaS no longer exists, unfortunately.

In the first example section I added a couple of paragraphs on "choreography over orchestration". While this idea exists in the Microservices world too, the extent to which that can be taken, with great effect, with Serverless is astonishing to me. Gojko Adzic has done some great speaking and writing on his experience here, and I included the link to his talk from late 2017.

In the "Unpacking FaaS" section I explicitly describe that with FaaS the vendor handles all resource provisioning and allocation. While "containers as a service" is catching up in this regard I still think this lack of concern of resource management is a big benefit of FaaS.

I rewrote the "State" section—I was a little restricted in my thinking the first time around and I've come to see FaaS as something *not quite* stateless, since caching state in a Lambda instance that might stick around for 5 hours is a perfectly reasonable idea.

I also rewrote the section on Startup Latency since Cold Starts are one of the big "FUD" areas of Serverless. I also wrote an article on just this area last year.

Tooling has vastly improved since the first version of this article, and I was glad to tone down my dire warnings a little in this regard. Also there's been a lot of open source updates, including from Amazon and Microsoft. It's a brave new world we're living in.

Under "What isn't Serverless" I considered possibly needing to redefine precisely what I think counts as a Serverless service. But I've already done that elsewhere, with John, so I left it with adding a link to our definition. Whether containers are Serverless or not has become even more complex over the last couple of years. I thought it might do—I mentioned Kubernetes pod autoscaling in the original version—and products like AWS Fargate have made this distinction murkier still. I think there's definitely a place for both models for now, as I describe.

Finally in this first overall section it was good to be able to talk about AWS SAM and SAR—"Serverless Application modeling" is absolutely something that's being tackled.

Benefits

This was the section that changed the least—the benefits still hold, over all. One area that I did add extra emphasis to was time-to-market, and the idea I now refer to as "Continuous Experimentation" . I personally think this rapid "zero-to-production" capability is Serverless' biggest advantage.

I've been disappointed that the section on "Greener" computing hasn't received too much attention since the original. I still think this is important.

Drawbacks

Drawbacks is divided into two—inherent and implementation.

On the inherent side things didn’t change too much, unsurprisingly. I refined my section on vendor lock-in, I added another bullet point to security, and I edited the section on State, in line with the changes I made to the similar section earlier in the article.

Implementation drawbacks though got some big edits, thanks to all the advances from vendors and open source projects over the last two years!

Lambda has configuration now and it has reserved capacity to help you avoid DoS’ing yourself. Execution duration is still limited to 5 minutes, and cold starts are still a factor, although they have improved.

In testing we have seen some advances in tools, and in mindset. Lambda and Azure functions both now offer some amount of local-integration testing. On the other hand I’m more convinced that for automated testing we should just be using real cloud environments.

Debugging has improved massively in the Azure world … and somewhat in the Lambda world—I mention SAM CLI as an interesting addition to Amazon’s tooling, especially for rapid feedback for developing a Lambda-backed HTTP API application.

Deployment and packaging has also got much better.

Discovery is an area that I wanted to mention in the first version since it’s such a hot topic in the Microservices community. But it’s something that gets very little discussion in Serverless. I'm not certain why that is, but I added some thoughts.

The monitoring section got renamed "Monitoring and Observability". Because of course it did. I think function-level monitoring has got better, despite the environments, but I still think distributed monitoring is a ways off of where it needs to be.

AWS API Gateway, and its associated tooling, has got much, much better. However there's still the question of how much behavior should go in API GW vs. in Lambda code—this is something even John and I don't completely agree on. That said, API GW can be completely controlled from version controllable configuration, and tested in an isolated environment, so I'm not too worried.

The Future of Serverless

There was a lot I could have added to this section but I reigned myself in a little. The epic article was already growing enough. I still think most of the ideas I described are to come to pass. Maybe in another two years?

While tooling in general is still a concern, as I mention several times earlier in the article, there have been definite advances. I was glad to be able to talk about Amazon's automated traffic shifting / canary releases. And this also went well with an idea that Nat Pryce described to me a while ago about the idea of 'mixing desk' releases.

I also added under tooling the need for better tools for 'meta operations'—how to more effectively look after hundreds or thousands of FaaS functions, configured services, etc.

On the patterns side we have seen some ideas starting here (I'm looking forward to Tim, Yochay and Peter's book) . Particularly interesting to me since two years ago is the discussion around Serverless + event-thinking, and also 'micro-apps', as made real by Amazon's Serverless Application Repo.

I added a section on "Globally distributed architectures" … take a read and see what you think. :)

In the testing section here I added explicit reference to testing-in-production and monitoring driven development—two ideas that I think are very useful when we're using technology as rapidly deployable as Serverless ones.

Finally, of course, there's the community section. The original said:

> *I fully expect to see a huge growth of the Serverless community. Right now there's been one conference and there are a handful of meetups around the world. I expect that we'll start seeing something more like the massive communities that exist around Docker and Spring—many conferences, many communities, and more online fora than you can possibly keep track of.*

I might not get all my predictions right, but this one I'm happy has come to pass.

And that's that. I hope that in another year or two I'll feel the desire to revisit this article again. As long as people keep reading it, and as long as Serverless keeps evolving, I hope it can continue to be useful both as a primer and reference. And in the mean time, if you need some help wrapping your head around this stuff, feel free to drop me a line.

Serverless Cloud Computing Blog Architecture

Software Architecture

Like what you read? Give Mike Roberts a round of applause.

From a quick cheer to a standing ovation, clap to show how much you enjoyed this story.