Guides & Tutorials
Operations & Observability
Engineering Culture

When (and why) not to go serverless

Andrea Passwater Mar 21, 2018

There are a lot of people out there championing the serverless movement. Serverless lowers administrative overhead. It takes server maintenance off developers' plates forever and cuts server costs. The benefits are real.

But so are the drawbacks. If you're considering serverless, read on.

Observability is more difficult

It's probably the biggest critique of serverless right now: you just lose some amount of critical insight into your functions.

Serverless encourages event-based architectures, which a lot of people aren't familiar with. Add to that, that serverless is a new enough space that the available tooling is relatively immature. It can be hard to do things as simple as stack traces.

> The observability talks have not just been practically useful, but also somewhat reassuring that there are still problems to solve with microservice/serverless architectures and it's not just me missing something obvious!
>
> — Matthew Jones (@matt_rhys_jones) March 6, 2018

In the past year, logging and monitoring platforms such as Dashbird, IOpipe, and X-ray have vastly improved their options. Within the next one or two years, serverless observability should be much closer to parity. But there may always be the caveat that, by their very design, serverless functions are stateless. It makes them hard to debug in production by using anything except logs.

While there is tooling that keeps developers from flying blind, there is a lot of room for improvement in the serverless observability space.

Latency

Serverless functions mean you'll be dealing with cold starts.

Small caveat to say that there *is* a fairly simple workaround that many serverless developers use: keeping functions warm by hitting them at regular intervals.

But this is mostly effective for smaller functions. Things get a lot more complicated when you have larger functions or relatively complicated workflows.

To minimize cold start times, here are some things you should keep in mind:

- Application architecture: keep your serverless functions small and focused; cold start times increase linearly with memory and code size
- Choice of language: Python & Go can considerably lower cold start times, whereas C# & Java notoriously have the highest cold start times.
- VPCs: cold start times increase due to extra overhead of provisioning networking resources

Heavier reliance on vendor ecosystems

With serverless, you don't manage the server. That also means you lose control over server hardware, runtimes and runtime updates (at the time of writing, Node.js 8 is out but AWS is still on Node.js 6). The provider also imposes concurrency and resource limits.

The specifics of your application architecture can suddenly become determined by the provider you're using. If you go serverless with AWS Lambda, for example, the only serverlessesque databases you can use are DynamoDB or Serverless Aurora. (Though you can also, say, attach a Lambda to a VPC and access other databases like RDS, ElastiCache, and ElasticSearch instances inside it.*)

We're talking here about vendor lock-in. There are a lot of discussions out there about the long-term impacts of going all-in on a single provider, with a wide disparity in opinions:

> Instead of trying to avoid vendor lock-in, concentrate on switching cost. How easy is a solution to adopt now; and migrate away from later?
>
> — Kelsey Hightower (@kelseyhightower) April 24, 2017

The CNCF is also actively working to initiate standardization across platforms, in order to make it easier to migrate applications and mitigate vendor lock-in in general.

It's harder to hire

A lot of developers don't know what severless is. And even if they do, it's a hazy enough concept that applicants can have a hard time imagining what their job would entail.

Having 'serverless' in a job title has a real chance of shrinking the size of your candidate pool, in a market where finding qualified people is already hard enough. Even if you're willing to take developers without specific serverless experience, they may be too intimidated to apply.

On the flip side—to a smaller group of experimenters and fast-paced environment lovers, up-and-coming technology stacks are a huge selling point.

All that said—why use serverless?

If there are drawbacks to serverless, then why are people using it?

Well, overall it can add a lot of efficiency into application development and workflow.

These are the four main reasons people switch to serverless:

- it scales with demand automatically
- it significantly reduces server cost (70-90%), because you don't pay for idle
- it eliminates server maintenance
- it frees up developer resources to take on projects that directly drive business value (versus spending that time on maintenance)

> I have had *every* argument thrown at me. I then throw back: "I hardly have to manage anything and it scales and costs a lot less". #win

> — 🦄Paul Johnston 🦄(@PaulDJohnston) August 14, 2017

There are some use cases for serverless which, despite any possible downsides, are especially hard to argue against. Serverless APIs are workhorses.

Along those lines, the number of digital businesses not just utilizing, but going *fully serverless* is increasing:

> As of today @bustle has fully adopted serverless. We're down to 15 ec2 instances mostly comprised of self-managed HA Redis. We serve upwards of a billion requests to 80 million people using SSR preact and react a month. We are a thriving example of modern JavaScript at scale.
>
> — Tyler Love (@tyleralove) March 2, 2018

Our own website is a static, serverless site built using Lambda, the Serverless Framework, and Netlify. It's never gone down and we spend zero hours a week maintaining it.

TL;DR

As with all things in life, there are tradeoffs. Serverless means you gain efficiency, and trade some control & visibility.

Further reading on serverless architectures

- Serverless architectures primer
- How we migrated our startup to serverless
- Why we switched from Docker to Serverless
- Serverless (FaaS) vs. Containers - when to pick which?

Thanks to* *@hotzgaspacho*** *for adding this to the post.*