RELATED

Navigating the Tech Landscape: Reflections on 2023 and Predictions for 2024

Dynatrace Perform: Day Two

The Differences Between a Service Catalog, Internal Developer Platform, and PaaS

Provision Cloud Infrastructure Using Google Duet AI

Serverless Is Great; Serverless Sucks - Making Sense of the Serverless Landscape

by Navot Volk · Dec. 19, 19 · Opinion

Everyone has heard the famous line, "It was the best of times, it was the worst of times." This famous quote feels like it could perfectly describe the current state of serverless technologies and the way developers are currently building applications for those platforms.

On one hand, serverless is a boon to any company (or developer) building web applications. The idea that one can simply hand off the demands of managing infrastructure to someone else is an alluring prospect for CTOs, managers, or developers building independent projects.

While serverless platforms are by their very nature more restrictive, once you accept those restrictions it can make life much easier. Security, backups, servers — all of these infrastructure challenges are simply handed off to the serverless provider.

Serverless is even more attractive when we take into account all of the demands we place on developers. For most developers, we are asking way too much and making their lives way too hard. They are tasked with developing for the frontend and backend, working according to the constraints of DevOps teams, or independently handling infrastructure challenges. This doesn't even begin to account for the demands of device and platform fragmentation.

Of course, serverless isn't a magic bullet. It isn't even that radical of an idea. Many say that serverless is just a fancy way of saying someone else's server. In some ways, that's true. But it also misses the revolution that serverless ushers in by allowing you to buy compute and storage on demand without ever knowing how many servers were involved. It might sound simple, but in that simplicity, there is a technology that can dramatically lower the demands we place on already taxed developers.

So, in this context, serverless could be a godsend. However, as I hinted at when I started this post, the current state of serverless development is a bit mixed (to put it charitably). You just need to look at the rate of adoption of serverless platforms to see there is some weakness in this much-promised development utopia.

At this point, you're looking at this weakness and thinking, "so, is this where these crappy parts you mentioned come in?"

Serverless Development Challenges and a (Potential) Solution

Well, if you've ever had to build a web application using online development tools provided by serverless providers, you may already know where I'm headed. First, serverless or online development environments can lack capabilities and are often slower or less responsive than other environments, especially if you're not on first-world internet. There's also potential for a lack of flexibility in integrating with other tools, vendor lock-in, problems with development velocity, etc.

This introduces latency issues, debugging issues, and other problems when you need to rapidly iterate on your code. While every platform or development environment requires using specific tools or techniques, the perceived restrictions of serverless have dissuaded many developers from looking at this technology as a potential solution for their next project.

Developers, rightly so, want to use the tools that they know, that they're comfortable with, and that allow them to develop quickly and across their team. Fortunately, serverless platforms are advancing rapidly and increasingly offer more flexibility and integration with third-party technologies. Additionally, more serverless providers are offering cloud-local, hybrid development tools, which can combine the flexibility and speed of local development with the benefits provided by serverless environments in the cloud.

We have implemented this in our own serverless platform — Corvid — as we recently rolled out new ways to use third-party local development tools and collaborate with teams. With Corvid, we aimed to give a local development experience when working with the IDE and files on a local machine, but the test run environment and the design environment is on the cloud. This allows us to provide a close to a zero-configuration solution that provides a production-grade environment where everything is up to date, fully managed, and where you can't run out of resources.

Now, to actually deliver this experience, we had to solve a big problem. We looked at traffic across our own service, especially at low traffic sites, which make up a huge part of the industry. We saw that most instances that are running right now are idle over 86 percent of the time and not handling even a single user request.

So, we are paying incredible amounts of money for computers that are doing nothing. Now, there's a good reason for that. When a request arrives, you need to be able to respond fast. If it takes two minutes or even two seconds to start an instance, no one is going to wait. So, you need to have the instance up and running beforehand.

We had to find a way to manage our grid and infrastructure in such a way that a user's service is up when it's needed and down when it's not needed. This means there is zero latency for cold starts. With other environments, you need to start an environment, wait for it to be up and running, and only then can you start working with that environment.

This was a difficult challenge for us to solve, and there are other challenges that serverless providers still face. But, the industry is pushing through them at a rapid pace.

Next Steps

What I want to emphasize here, is that from the perspective of your company's CTO, or even from the perspective of a startup founder, they are likely already considering, in one way or another, using a serverless solution to streamline processes and minimize time spent managing infrastructure.

Even if you're on an enterprise-level team there, is a good chance that you will work with serverless technologies because of the way it removes the infrastructure burden and enables teams to build new categories of solutions. So the time to get ready is … not now, it was probably a few months ago or longer.

What developers can do at this stage, is look at the solutions that are out there, and start thinking about what pain points they can solve in their workflows. Finding a serverless solution, if it's the right fit for your project or business, doesn't have to be a top-down decision from management. So, find a solution that makes sense for you and push for it.

Everything Will Be Okay

Perhaps you believe me and are ready to embrace our future serverless overlords. Or, perhaps, you're not convinced that serverless development will ever be the seamless experience developers have come to expect from their current local setups.

Even though I've spent some time here raising serious questions about serverless development, at the end of the day, I really do believe this technology is the future and that it will dramatically improve the productivity of developers.

As a former founder, I'm happy to push startup CEOs in this direction while their companies are still young and flexible. And now that I work in the corporate world, I can also feel confident encouraging enterprises to move towards serverless.

But, at the end of the day, the people who need to be convinced are the developers who are going to spend countless hours working on these platforms, building the solutions we all use. So I'll end with the pitch I use to sell developers on serverless.

This technology, at its most basic level, is all about letting developers focus on code and building applications. It takes all the other infrastructure concerns and simply hands it off to some other poor halfwit (they're actually really smart people and not at all halfwits, but their problems don't need to be your problems).

And at the end of the day, I would ask developers whether they want to spend their time coding or worrying about uptime, cost management, or a host of other infrastructure challenges. To give you an analogy, when you step into a kitchen, you want to start cooking right away. You don't want to have to build the kitchen every time you cook dinner. There are professionals, carpenters, and contractors who can build you an amazing kitchen, and then you can whip up something delicious.

Serverless is, in many ways, the same. There's an incredible platform already waiting for you. Go make something amazing.

Further Reading

- Understanding Serverless Architecture Advantages and Limitations.
- Experiments in Integration (Part 1): The Serverless Framework.
- Going Serverless? Compare Your FaaS Options.

Dev | Infrastructure | Landscape (Software)

TRENDING

The Perfect Trio: GitOps, Kubernetes, and Platform Engineering

Enhancing Resiliency: Implementing the Circuit Breaker Pattern for Strong Serverless Architecture on AWS

How to Build a Data Foundation for Generative AI

Remote Control With Node.js, React.js, and Raspberry Pi