

What Does Serverless Actually Mean?

It's actually not serverless at all



Steven PopovichFollowMay 7 · 5 min read★

🐦

📷

📘

🔖

👤



Photo by Taylor Vick on Unsplash

The word *serverless* is really misleading. It's a buzzword, for sure.

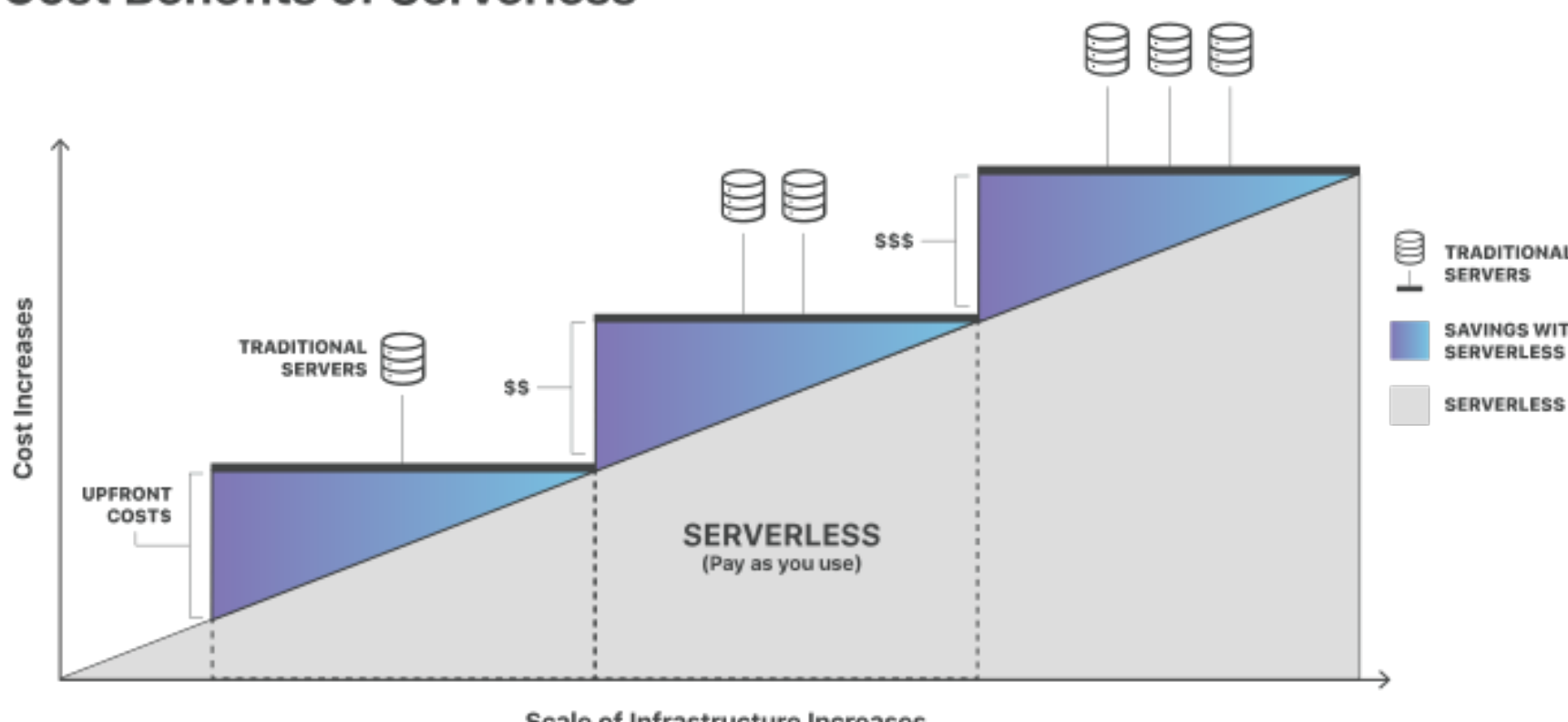
If you google “serverless,” the Wikipedia definition is actually pretty good:

“*Serverless computing is a cloud computing execution model in which the cloud provider runs the server, and dynamically manages the allocation of machine resources. Pricing is based on the actual amount of resources consumed by an application, rather than on pre-purchased units of capacity. [1] It can be a form of utility computing.*” — Wikipedia

This is all true. And if you can digest this jargony definition, you can see that serverless, in fact, involves many servers.

What serverless actually means that you don't have to worry about what size of the server box (processing power) or how many boxes you need to serve your users. The provider of the serverless service deals with the size and numbers of boxes you need, based on the amount of work you throw at it.

Cloudflare, one provider of serverless back ends has a great graphic about how serverless can save you money and how it scales:



Taken from Cloudflare's great piece on the cost/benefit of serverless

In a traditional system, when your application gets more users or your users do more things, or really anything that causes your back end to do more processing, you have to increase the number of boxes (or the power of the boxes) as you get more demand.

Let's take an example. Let's say you operate Reddit. You have to have servers to serve the website to the user and a database to keep all your links, comments, and user profiles.

In the traditional model, you could have one box serving traffic. You might have a scaling policy that if the box has 95% CPU or more for five minutes, add another box to serve the traffic. That's the blue boxes above. You would be paying a fixed price per box, per hour. When you go from one box to two boxes, you are doubling your cost.

But what if, when you scale up to two boxes, users get off your site. Your new box doesn't even get used. You could have a scaling policy that will scale back down to one box, but the point is, if you have two boxes spun up, you are paying for them.

You can see the overhead this introduces. You are technically paying for server power you may not be using.

With serverless, however, in theory, you pay for exactly only what you use. You don't set scaling policies or pick how many servers you need. You just throw traffic at the provider and pay for how much it costs. You don't know how many boxes the provider is using, and you don't care. And the savings are passed on to you. In theory.

This is where the name *serverless* comes from. You, as a user of server power, don't have to think about the servers that are working behind the scenes. But they are still there!

...

So What's the Catch?

So why isn't everything serverless? It seems like everyone would win because we don't end up with wasted server time.

But it isn't that simple. Behind the scenes, the serverless provider is still doing the same thing as the traditional model. And it won't become magically perfectly granular.

The serverless provider — meaning a service like AWS Lambda, Cloudflare, Azure, and Google Cloud — is still scaling up and down, and allocating boxes to work that goes up and down. And sometimes they will have unused servers. And someone still has to pay for the servers, whether they are unused or not.

Now, they might be better at it. They can have advanced machine-learning algorithms and distribute your work across many boxes intelligently. Boxes can run work from different people. Since it is their main purpose to cost-effectively provide server time without wasting server time, they can probably do it more efficiently than you. But they, just as you did when you were managing your own boxes, have to deal with the scaling of boxes.

Serverless really just puts the job of scaling onto the service provider. So what if the provider isn't good at it? And remember, you are paying for the engineering of the serverless algorithms and infrastructure. This means, for some workloads, you could end up paying more.

Remember: These providers are businesses.

It is important to keep in mind that if you're using one of the providers, they are still businesses that need to make money.

And they can lose money on small projects and make all their money on big projects.

A lot of providers of developer back-end services (Google Maps API and Firebase, to name a few I have used) employ a “you'll pay when you scale” model. In that, for some small workloads, the cost is much cheaper than provisioning your own boxes. But when you scale up your workload (your business grows) the cost would be much more than your own setup. Keep this in mind and use the provider's calculators as necessary.

There is also a man behind a curtain

Furthermore, serverless has another problem that makes me nervous as a distributed system maintainer: You can't see exactly what is happening.

When you spin up your own boxes, run your own software on them, and roll your own monitoring and scaling for all the boxes you manage, you have *control*. And for large-scale applications, developers and companies want control. You want insight as to what is happening in your system.

With serverless, if you have a problem with your workload, it's much harder to tell what is going on because you can't just look at the logs on a box. The computation that is happening is hidden away from you.

Or when users start writing in that your software isn't working or is slow, you can't see exactly why your system didn't scale. The best you can do is file a ticket with your provider and hope that it was just a blip.

This means debugging your system is more difficult. It's not impossible; you still have logging and the like. But most back-end maintainers I know like having the ability to hop onto a box that is having a problem and see exactly what is going on.

There are also myriad other problems with serverless that I'm not going to get into (vendor lock-in, cold starts, security?), but they are worth considering if you're thinking about using serverless for your application.

...

But We Are Getting There

I wish serverless was perfect. I and many other people spend too much worrying about if our system will scale.

Remember, any time you can reduce the number of variables you have to worry about in a system, it's a better system. You always want to reduce complexity.

But going serverless tomorrow doesn't necessarily reduce complexity and doesn't necessarily reduce costs, either. As with any other technical or architectural decision, it's important to see if it fits your project. Take your time and do a strong analysis of what is right for you.

It also still uses servers — don't be fooled! Thanks for reading!

Thanks to Zack Shapiro.

Software DevelopmentDistributed SystemsServerlessAWSProgramming



217 claps


🐦

📷

📘

🔖

👤



WRITTEN BY
Steven Popovich
Software Engineer | Full-stack | GameChanger | DICK's Sporting Goods | Trying my best

Follow



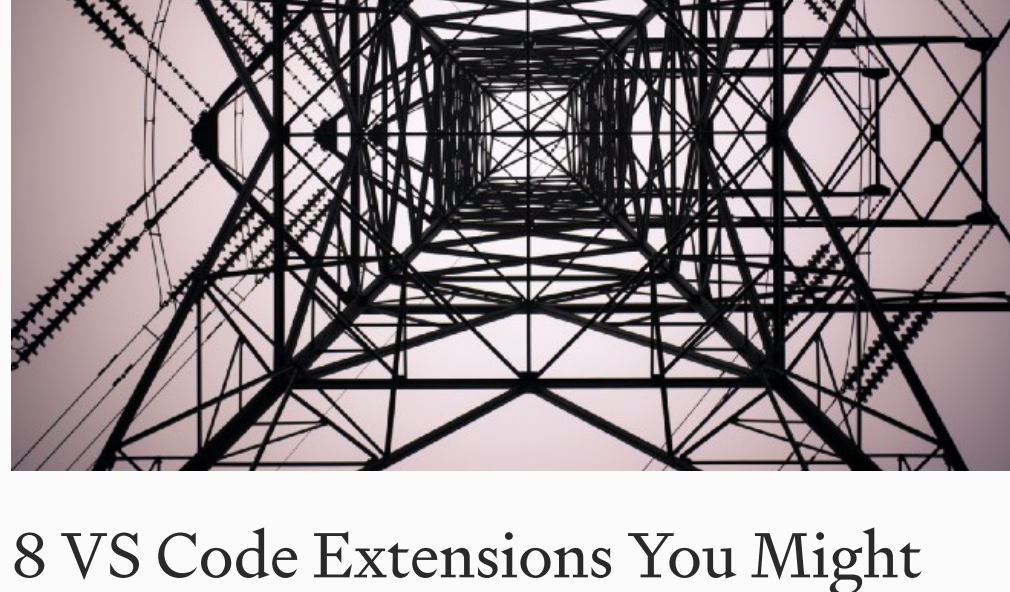
Better Programming
Advice for programmers.

Follow

Write the first response

More From Medium


More from Better Programming



8 VS Code Extensions You Might Love

Marvin Wendt in B... ProgrammingMay 4 · 3 min read★

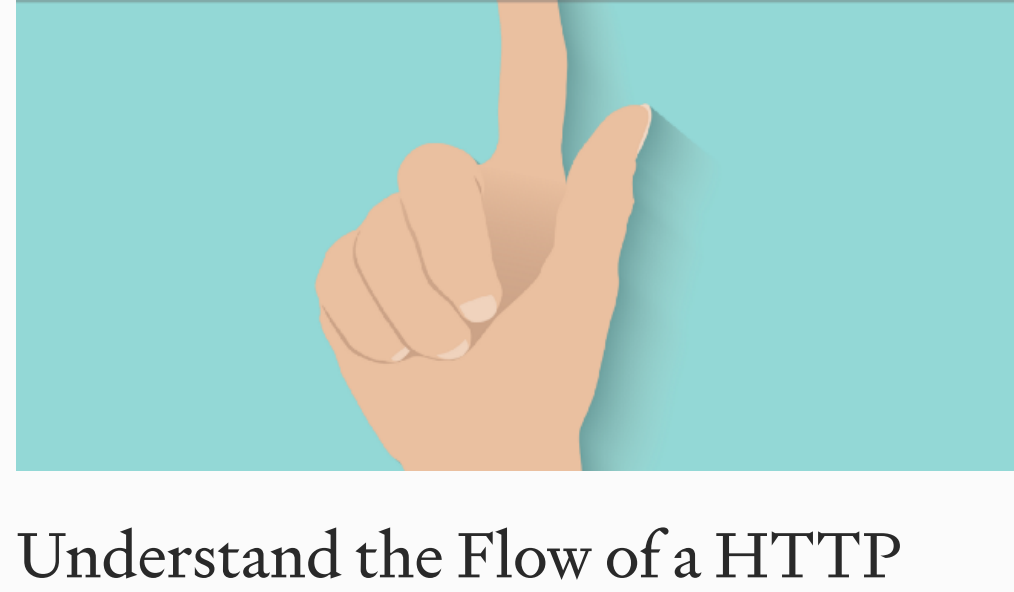
More from Better Programming



Don't Use Boolean Arguments, Use Enums

Anupam Chugh in ... ProgrammingApr 28 · 6 min read

More from Better Programming



Understand the Flow of a HTTP Request

Aakash Yadav in ... ProgrammingMay 1 · 6 min read★

Discover Medium

Welcome to a place where words matter. On Medium, smart voices and original ideas take center stage - with no ads in sight. Watch

Make Medium yours

Follow all the topics you care about, and we'll deliver the best stories for you to your homepage and inbox. Explore

Explore your membership

Thank you for being a member of Medium. You get unlimited access to insightful stories from amazing thinkers and storytellers. Browse