What's Serverless? - by Justin

Technically
What's Serverless?
Beware of marketers

Justin
May 11, 2021

The TL;DR

Serverless is a catch-all term for apps and products that **don't require you to manage your own servers**.

- Every app runs on a set of servers somewhere, and there's **high overhead** involved in managing them
- Serverless just means that your app – or the infrastructure you pay for – **doesn't require server management**
- In practice, people use the term to refer to a **function-based architecture** instead of an app-based one
- Each major cloud has a serverless platform, the most notable being **AWS's Lambda**

People mean a million different things when they talk about serverless, so we'll explore it from a few different angles. The important thing is that it's getting wildly popular, and may be how everyone ships apps in the future. The **future**.

Quick reminder: backends today

If you're a developer building your app, how do you deploy your backend today?

Remember, most apps you use have two major components:

- A **frontend** – the user interface, usually in a web browser or phone app
- A **backend** – the logic and data of the app, usually in a database and APIs on top of it

A good example is Substack, the lovely folks who help me send out these emails and posts. Substack has a frontend and a backend:

- The Substack **frontend** is the web app I use to write posts, look at analytics, and send emails. It's written in HTML, CSS, and Javascript
- The Substack **backend** stores text, recipients, and stats, as well as has APIs for sending mail and adding subscriptions. It uses a database and a backend language (which could also be Javascript, but that's neither here nor there)

The frontend and backend work together to give you what you see when you load a page in your favorite app. The backend is usually the part of the app that's the most *complex* to build and deploy, because it undergoes the most "stress" and needs to scale appropriately. If 1,000 people are using your app at the same time, your database is going to need to be able to handle 1,000 queries every few seconds – so you can see how it quickly gets out of hand.

> 🔮 **Dependencies** 🔮
>
> If you're still a bit iffy on how this works, read up on what DevOps is and how engineering teams deploy high performing apps.
>
> 🔮 **Dependencies** 🔮

When you're trying to make sure your database and APIs can keep up with your app's traffic, you have a few options:

- Scaling **vertically** – whatever server you're running your database on, make it bigger, faster, stronger
- Scaling **horizontally** – create copies of your server, and let each shoulder a bit of the load

Scaling vertically is really easy (you just tell AWS to suck out more of your money), but not fully sustainable or cost efficient; scaling horizontally, though, is notoriously difficult.

And if that devil's bargain wasn't enough, scaling isn't the only problem you'll face when running your backend. Making sure your servers are performant is highly stack dependent, but will likely involve:

- Setting up the servers in the first place, installing things, etc.
- Monitoring your app to make sure requests are running smoothly
- Monitoring your hardware to check CPU, RAM, and Disk utilization

These are just a few of the things that engineers deal with (especially if you have a DevOps focused team) when managing their own servers.

> 🚨 **Confusion Alert** 🚨
>
> Scaling needs apply to frontends too. And pieces of the stack that are glue for the frontend and backend – like if you're using a web framework like NextJS – need to run on a server somewhere. So while we're covering mostly backend here, this is a bit of an oversimplification.
>
> 🚨 **Confusion Alert** 🚨

Some definitions of serverless

So if the above is working with servers, serverless – in *theory* – should just mean anything that *doesn't* require you to use servers. But that's kind of a stupid definition, because a lot of things don't require you to interact with servers. If you're paying GitHub to manage your code repositories, is that serverless? Is Gmail serverless?

Erik Bernhardsson @fulhack

I feel like we need to raise the bar for what's "serverless". IMO any of these things disqualifies: - You have to maintain the compute nodes - Anything that includes the notion of a "cluster" - Pre-provisioning capacity - Paying for idle capacity

9:55 PM · Apr 6, 2021

66Likes5Retweets

To navigate through such murky waters, I'm going to propose 3 definitions of serverless below. The 3rd and final one is the most important, because it's what developers generally mean when they talk about it.

1. **Serverless as SaaS**

When you (i.e. developers) outsource any piece of the software stack to a third party via SaaS, that's technically serverless. A few examples:

- Using GitHub for hosting code repositories
- Using Auth0 for your app's authentication
- Using Datadog for monitoring
- Using Stripe for payments

While it *is* useful to appreciate that all of these SaaS solutions obviate the need for developers to build and maintain their own infrastructure, it's not usually what we mean when we say serverless. I call this the MBA definition – i.e. it's useless.

2. **Serverless as PaaS**

Moving further down into the infrastructure layer, outsourcing parts of your infrastructure to managed services from our favorite neighborhood cloud providers can be referred to as serverless. A few examples:

- Vercel and Netlify for frontends and very light backends

- Confluent for event busses
- S3 for object storage

This is where terms like “serverless database” or “serverless frontend” apply.

3. **Serverless as functions**

Finally, the most specific definition of serverless is what developers are talking about most of the time: it’s a method of deploying your app via **functions** instead of monolithic apps and endpoints. Each major cloud provider has their own product for it:

- AWS has Lambda
- GCP has Cloud Functions
- Azure has Functions

So in other words, you don’t just not manage servers – you **deploy your app as a series of separate functions**, which is *quite* different than how it’s normally done. In that sense, serverless is an architecture decision as much as it is a product.

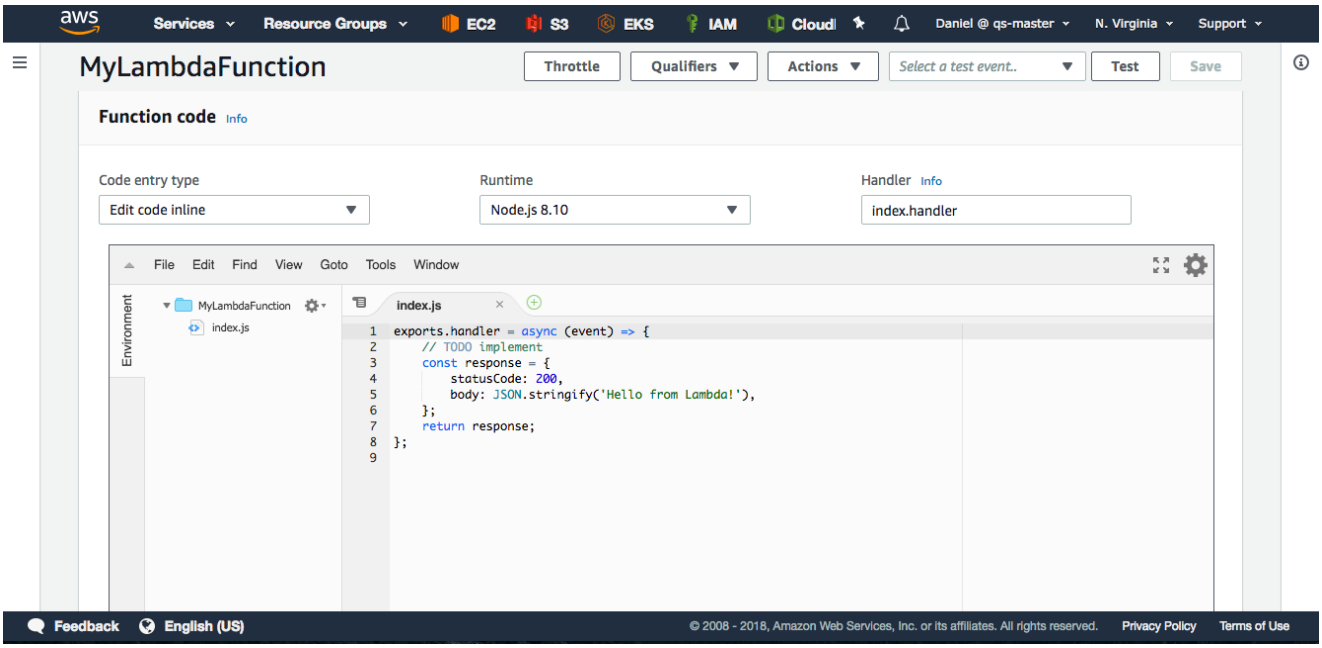
So basically to sum up here, when you see the word “serverless” it’s all about context. People also try and co-opt the term to sound more sophisticated – but as we’ve seen, plenty of “serverless” products don’t necessarily *replace* what you *would* have used a server for. So be skeptical 🙂

AWS Lambda and its malcontents

Platforms like Lambda are what developers usually mean when they say serverless. So what does Lambda (and the other equivalent offerings from cloud homies) do?

- In Lambda, you deploy your code as separate, logically independent **functions**
- You can **trigger** those functions manually, or via defined **events** in the AWS ecosystem

You might have a single function in your app that inserts a new user into the database. Instead of running this, along with all of your other functions, on a central server, you can upload them all separately to Lambda, and trigger them from your frontend. Literally, the Lambda UI lets you just paste code in there as a function:



(h/t to Daniel for the screenshot)

Once you trigger that Lambda function, AWS takes care of spinning up a server or allocating it to an existing one. Hence, serverless.

But it's deeply ironic that this is what we've come to define serverless as, because Lambda is *barely* actually serverless. That's because:

1. **Lambda is not a database**. You still need a database, and need to worry about it scaling, etc.

2. **You need to size Lambda functions**. You have to allocate a specific amount of RAM to Lambda functions in advance. Idk, sounds like a server…

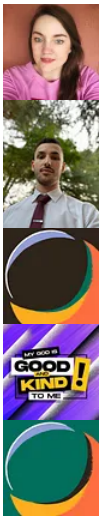
This is why in practice, most uses of Lambda and its siblings tend to be for **non-core parts of the app**. Most of the customer case studies on the site talk about things like image pipelines, analytics, or general machine learning. High growth startups are usually not running their core app logic on Lambda (until I'm proven wrong, at least).

> 🔍 **Deeper Look** 🔍
>
> Literally, AWS built a proxy service for their managed database service, RDS, so it can connect to functions running via Lambda. So on their features page for their "serverless" product, they're telling you about how to work with the "server-full" parts of your app. Which is kind of weird?

> 🔍 **Deeper Look** 🔍

This is a problem with function-driven architectures these days – the developer experience isn't quite there, especially as it relates to developing functions locally and then uploading them to the cloud. Personally, I was working on a project using GCP Cloud Functions, and it was *infuriating* – I needed to test passing a specific data type, and there's basically no tooling to mock or run that on my laptop. That's why we're seeing frameworks like SST (Serverless Stack) try to improve DevEx for platforms like Lambda.

Meanwhile, we're seeing a lot more interesting stuff in the database space. Fauna purports to be serverless, and charges you exclusively for usage (not based on server size or anything like that). BigQuery, a popular data warehouse, is definitely serverless. Consensus is that we're still on the cusp here, but it's definitely something to keep an eye out for.



50 Likes

.

1 Restack