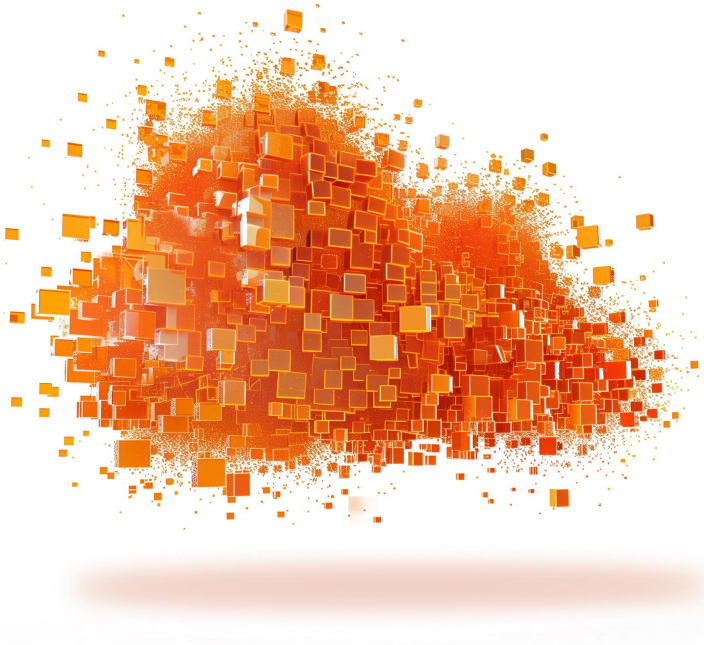
The Pragmatic Programmers

Serverless Apps on Cloudflare

Build Solutions, Not Infrastructure



Ashley Peacock

edited by Michael Swaine

This extract shows the online version of this title, and may contain features (such as hyperlinks and colors) that are not available in the print version.

For more information, or to purchase a paperback or ebook copy, please visit https://www.pragprog.com.

Copyright © The Pragmatic Programmers, LLC.

Introduction

Back when I started creating software, over fifteen years ago, things looked very different to how they do now. I started out like a lot of engineers, building a simple blog using PHP and MySQL. It was the perfect introduction to programming, with the ability to simply upload your PHP code to some cheap, shared hosting, and instantly be able to run your code.

There were no containers, and nobody had really heard of the cloud, with Amazon just starting its journey with AWS. As the cloud took off, and everyone started migrating, technologies such as Docker became increasingly popular, allowing you to run your applications locally inside containers, and deploy those same containers remotely in the cloud.

At each point, the idea has been to make building and deploying applications simpler and easier for everyone. Right now, you can go and spin up quite literally anything in the cloud, from your own instances, to databases, to machine learning tools, and everything in between.

The next big shift in how we build and deploy applications, in my opinion, is serverless.

What is Serverless?

With serverless, you don't pay for what you don't use. That's a big deal. I'll explain.

The most common way to deploy applications to the cloud today is to provision an instance, such as on AWS EC2, and then deploy your application to it. Whether you put the code on there directly, or via containers using AWS ECS, you're ultimately renting servers from the cloud provider, and paying for every hour your instance is running. Considering your websites can be accessed 24/7, that means your EC2 instances are also running 24/7, and you're paying 24/7.

• Click HERE to purchase this book now. discuss

With serverless, you package up your code, upload it to a serverless runtime, and the cloud provider handles the rest. You don't need to provision any servers, your code is simply executed when a request comes in. If your website receives zero requests in twenty-four hours, you'd pay $0 because your application didn't handle any requests.

Imagine your application hasn't received any requests for a while. Then, a single HTTP request comes in. Your application's code will be loaded into memory, the request will be executed against it, and a response returned. Imagine a second request comes in right after; it'll potentially be executed against that same instance of your serverless function, or perhaps a new instance will be created. After a period of that application receiving no requests, the code will be removed from memory, and another instance will be created when the next request comes in.

For some cloud providers, when a request comes in and a new instance of your serverless application is needed, this can sometimes be a little slow, as it has to load your application into memory. It varies depending on the language and runtime, but these initial loads can often take upwards of 500 milliseconds.

With Cloudflare, cold starts are not generally an issue, thanks to their platform being geared entirely towards serverless. When a request is made from a client, such as a browser, it must first establish a secure connection (as pretty much everything these days is over a secure connection, such as HTTPS). This process is known as the TLS handshake, and during that handshake, the client must send a "hello" message to whoever it's trying to connect to. When Cloudflare sees that hello, it knows what code to load to serve that request based on the URL. It loads that code in memory ready to go during the TLS handshake, so when the handshake completes and it needs to handle the actual request, there's no wait time for your code to load.

I'm going to focus on the Cloudflare serverless platform in this book, but the serverless concept is agnostic of any cloud provider. Every cloud provider has a serverless offering, so you can apply what you learn in this book to any of those platforms. Although the implementation details will differ, the underlying serverless approach you'll learn here will work anywhere.

I'm a serverless fan, and as I see it, serverless benefits you in four ways: pay for what you use, scalability, high availability, and no server maintenance. I'll discuss each of these with Cloudflare as my example, but the benefits apply to any serverless solution.

• Click HERE to purchase this book now. discuss

Pay for What You Use

With the major cloud providers, you effectively pay per request handled. Each request handled by your serverless function is called an invocation. That might sound expensive, and it can be if you get a lot of requests, but for the vast majority of websites and applications, your costs won't get anywhere near what they would if you were provisioning your own instances.

Take Cloudflare, for example; you can sign up for a free account, and instantly get access to 100,000 free requests per day. If your website becomes popular enough to hit that limit, you'll only pay $0.50 per million invocations after that. Considering the cheapest EC2 instance from AWS, at the time of writing, is around $50 a month, you'd need your EC2 instance to serve a lot of requests before it would be more cost-effective to run on EC2. In fact, you'd need to handle over a hundred million requests through Cloudflare before you reach the same spend as a single EC2 instance on AWS.

However, if you were receiving enough requests to make an EC2 instance more cost-effective than serverless, I doubt the smallest EC2 instance would handle that. In the example above, assuming the hundred million requests were spread out evenly, your single EC2 instance would have to handle thirty-eight requests per second, which is a tall ask for a single EC2 instance with limited resources.

That brings us nicely to the next principle of serverless: built-in scalability.

Scalability

As your application grows and becomes more popular, you'll receive more traffic to it. That will raise the amount of CPU and memory your application uses, and at some point, you won't have enough to handle all the requests. You can extract a lot from a single EC2 instance, especially if you use concurrency, but it will only go so far.

In a traditional cloud setup, you have two options in order to handle more traffic: vertical or horizontal scaling. Vertical scaling involves increasing the resources on your instances, whereas horizontal scaling increases the number of instances you have running.

In both cases, there's naturally a cost increase. Increasing the instance type on your EC2 instance, even from the smallest to the second smallest, doubles your monthly cost. Each time you increase the size of your instance in AWS, the cost roughly doubles. If you need more memory, you have no real choice but to increase the instance type (unless you can optimize your application's

• Click HERE to purchase this book now. discuss

memory use). Along the same lines, adding more instances and horizontally scaling increases costs too. And don't forget about the additional cost of load balancers.

If we compare that to serverless, a lot of that complexity is handled for you. You might still need to pick how much memory you want to allocate, or how much CPU, but that's about it. With Cloudflare, there are set limits for CPU and memory. Each instance of your serverless function has 128MB of memory to work with, and as much CPU as you could realistically need.

Even on the free plan, you get 10ms of CPU time per request. It might not sound like a lot, but in terms of CPU time, the vast majority of requests to Cloudflare use less than 1ms of CPU time. That doesn't mean your application has to respond in 10ms, as a lot of what an application might do isn't CPU-related. For example, waiting for a response from an API call or writing data to a database. I've yet to hit any of these limits when developing applications on Cloudflare.

As we're now just packaging up code and giving it to the cloud to handle, we don't worry about servers, and your serverless applications will automatically scale to meet demand, without any work from you. Additionally, in the case of Cloudflare, you don't need to worry about load balancers either, or even how to expose your application; it's all handled for you, including DNS and SSL certificates.

That means if your application gets a surge of traffic, which can sometimes unexpectedly happen, your application will scale to meet demand. There's no configuration needed; unlike when you horizontally scale your own instances, serverless has auto-scaling built-in.

In the traditional cloud setup, there's actually a second reason to run multiple containers besides scalability, and that brings us on to the third principle of serverless: high availability.

High Availability

High availability and scalability are closely linked, as they are both key to ensuring your application is always ready to meet the needs of your users. Whereas scalability is all about ensuring your application can deal with increases in traffic, high availability is ensuring you can deal with unexpected failures.

I described scaling the number of server instances of your application to meet demand, but running more instances of your application also makes it more

• Click HERE to purchase this book now. discuss

highly available. For example, if you are running three instances, and one of those instances unexpectedly crashes, you have two more that are still able to serve traffic.

High availability isn't just about running more servers though, or at least not without some further conditions. While uncommon, it's possible for cloud providers to experience widespread issues on their network. For example, there might be routing or DNS issues in one of their data centers. If your application is hosted exclusively in one data center, such as us-east-1 on AWS, that means your application is going to be unavailable.

To cater for this case, you can deploy to different regions and/or availability zones. For example, you might deploy some instances to a data center on the East Coast of the US, and some to the West Coast of the US. This increases the availability of your service, as should one data center go down, you still have servers in another that can serve traffic.

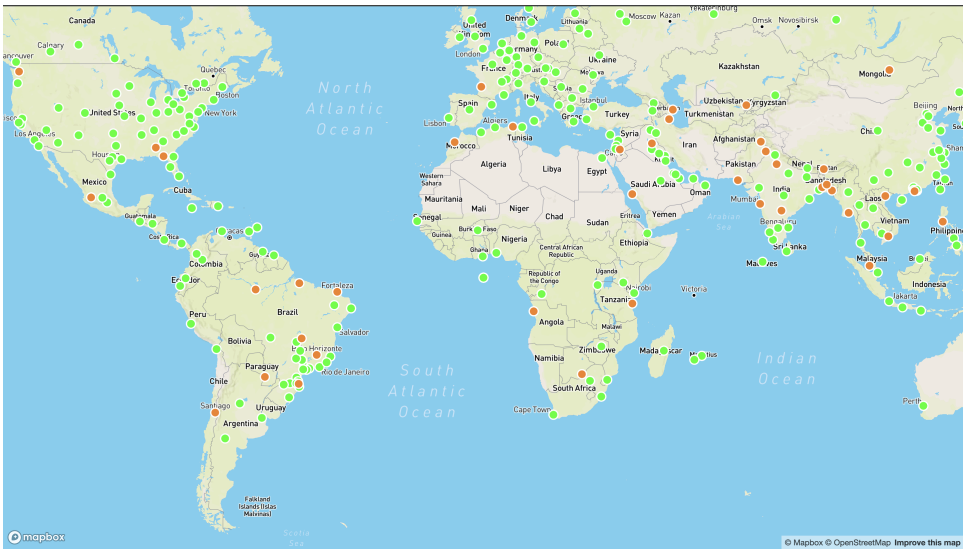
This does add to your costs and complexity though, as you now need to ensure your servers are running in different regions and/or availability zones, and if you combine that with auto-scaling, you need to ensure you can scale up and down in both regions too.

These cost and complexity considerations are taken care of with serverless. Much like scalability, high availability is just baked into serverless. Because you're not spinning up your own servers, and the cloud provider is handling how and where to host your application, they also take care of ensuring it's highly available.

That's because you're not tied to a specific set of servers, regions, or availability zones. Even if there was a catastrophic failure, and all the data centers in the US went down, your serverless application would simply be served from another country instead. Latency to your end users would increase a little, but that's a small price to pay for maintaining your application's services.

If you want to get an idea of the scale at which Cloudflare operates its network, I recommend checking out https://statusmap.cloudflare.community/. It's a community-maintained project, but it shows you all of the data centers Cloudflare operates, and just how much coverage they provide worldwide. Green dots indicate the data center is available, orange indicates it's undergoing maintenance or is being rerouted to another data center. The following is a snapshot, just to give you an idea:

• Click HERE to purchase this book now. discuss



In short, serverless applications built on Cloudflare are global by default.

We've talked a lot about the manual work required for the likes of AWS, which brings us nicely on to the final principle: no server maintenance.

No Server Maintenance

Finally, with serverless, there should never be a need to maintain any servers because with serverless, you don't spin up your own servers. No upgrading the version of the programming language you're using, no upgrading the operating system, nothing like that. If you deploy your application to an EC2 instance, you'll need to upgrade the language version you're using yourself.

You're still responsible for upgrading the dependencies you introduce, such as packages and libraries, but besides that, there should be no dependency or server management whatsoever. If you want to change the version of the language you're using, or even change languages entirely, that should be a painless process and a simple configuration change.

This is one of the primary reasons why I enjoy developing serverless applications. I can focus on what I enjoy most, and that's writing code.

Ready to Go Serverless?

So that's the three-dollar tour of serverless. Now it's time to write some code. As you go through the book, you'll master building serverless applications through hands-on projects. You'll build several different applications to showcase specific serverless concepts.

Should you get stuck at any point or simply want to view the finished projects without making them yourself, you can do so on GitHub.[1] This repository contains all the completed projects.

Let's build some serverless applications!

1. https://github.com/apeacock1991/serverless-apps-on-cloudflare/