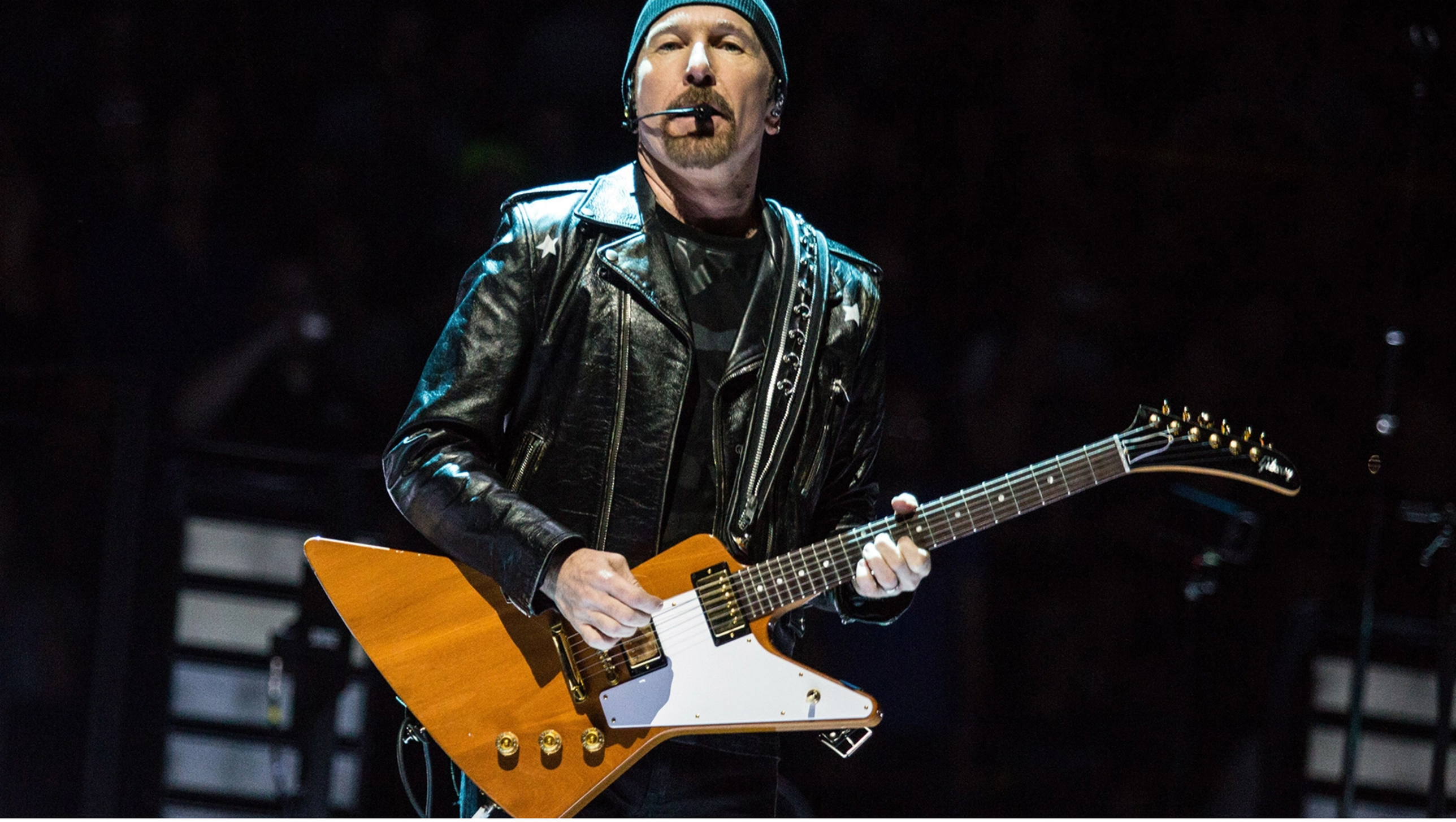
What You Need to Know About the Edge

Who am I?

- Developer Experience Engineer at LaunchDarkly
- Co-author of The Jamstack Book from Manning
- Editor of the Jamstacked newsletter from Cooper Press
- Run the CFE.dev community and Orlando Devs


LaunchDarkly

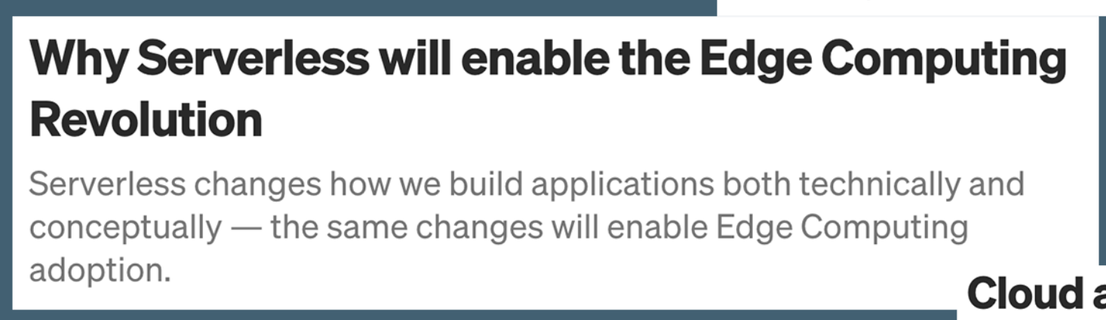
What is "the edge"?

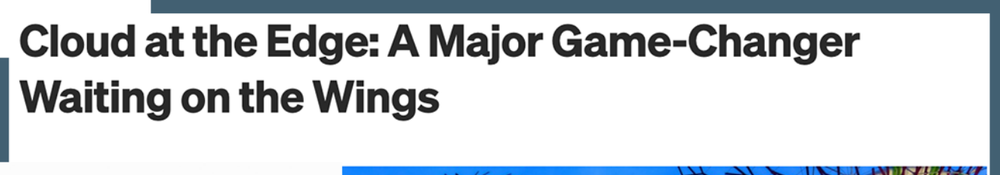


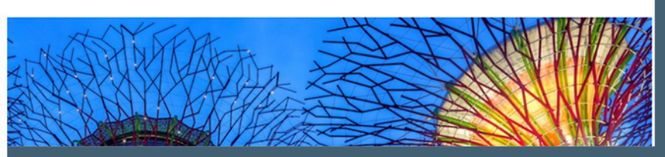
Questions?

- Twitter: @remotesynth
- Mastodon: @remotesynth@mastodon.xyz
- Email: brinaldi@launchdarkly.com

In all seriousness though, before we discuss what the edge is, I want to talk about broken promises. As developers, I think we've all gotten used to being sold a new technology only to find out later that it doesn't quite live up to the hype.







And let's bet honest, edge computing and edge serverless specifically have gotten a ton of hype lately. In many cases, the hype is coming from within companies that have a stake in the game.

11.10.2021

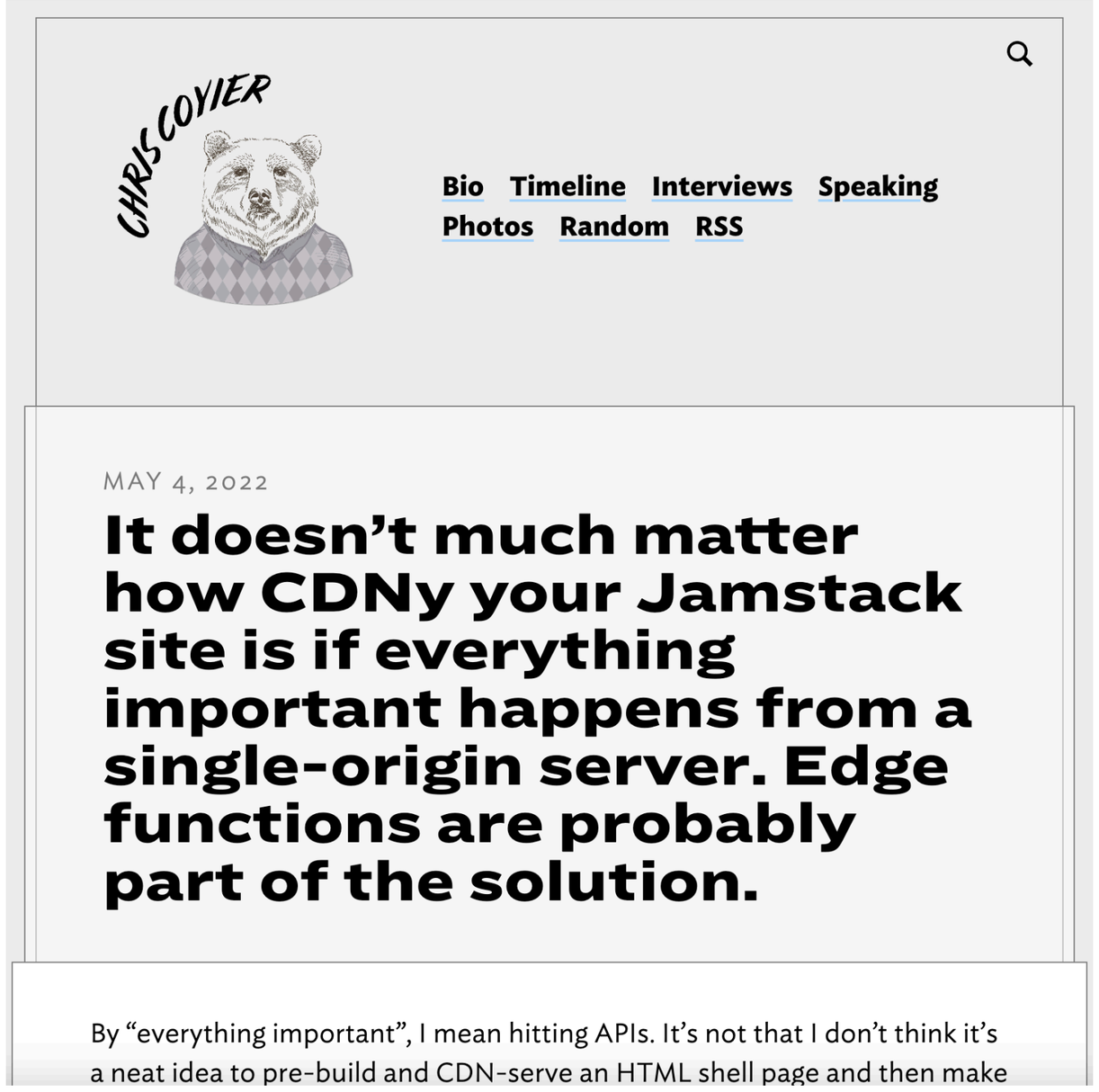
The Unfulfilled Promise of Serverless

BY COREY QUINN

I suggest that serverless computing, or “serverless” has hype that at this point has outpaced what the technology / philosophy / religion has been promising. Serverless computing arrived (debatably; please do not…

I suggest that serverless computing, or “serverless” has hype that at this point has outpaced what the technology / philosophy / religion has been promising. Serverless computing arrived (debatably; please do not email me about this whatever you do, fans of Google App Engine / CGI scripts / managed SaaS offerings /

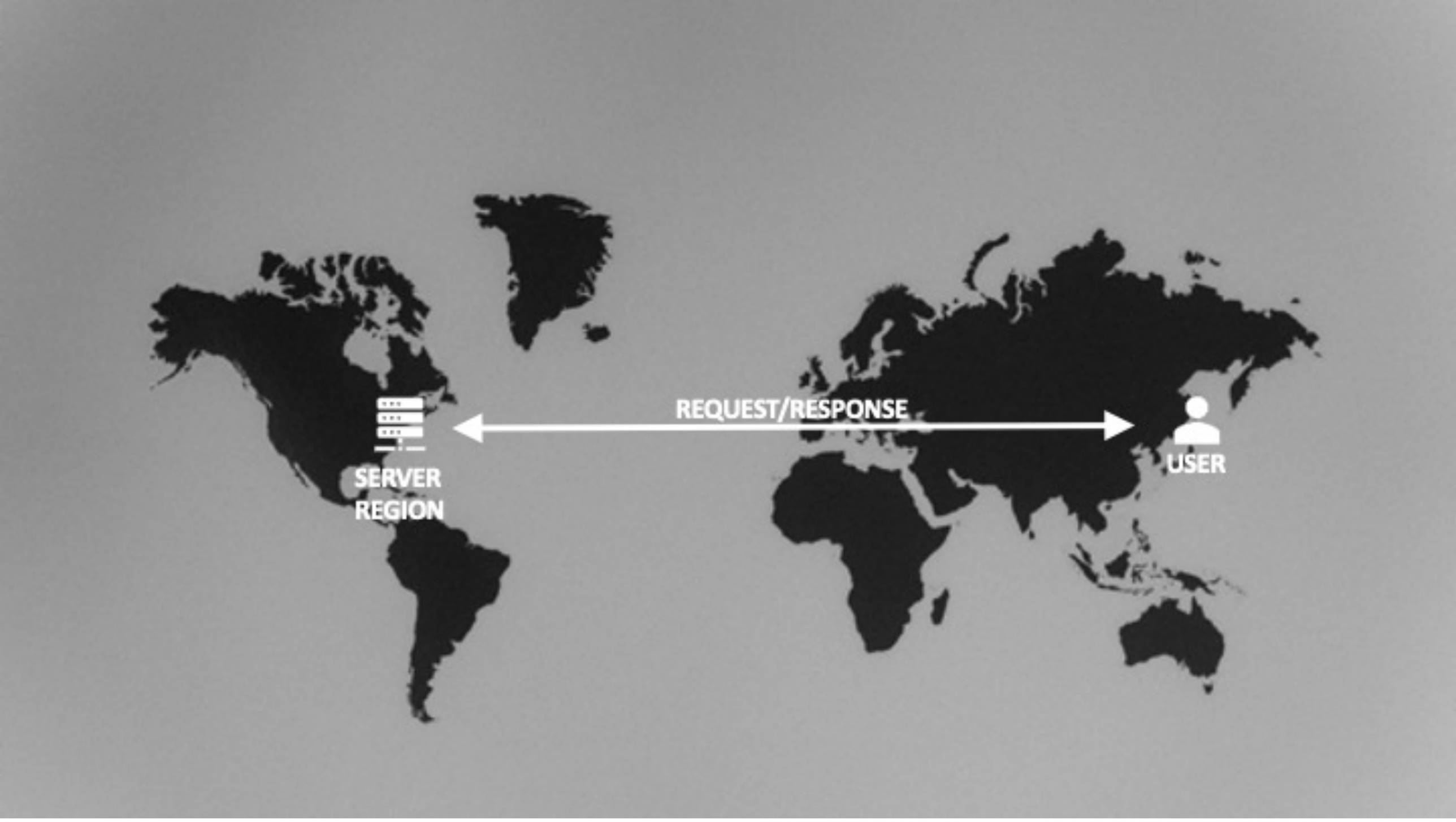
And serverless arguably has a history of broken promises. Corey Quinn, a very well known expert on AWS and serverless, wrote this in November 2021. Now I'm a big believer in serverless in general, but I won't argue that in many cases it hasn't lived up to the hype. https://www.lastweekinaws.com/blog/the-unfulfilled-promise-of-serverless/



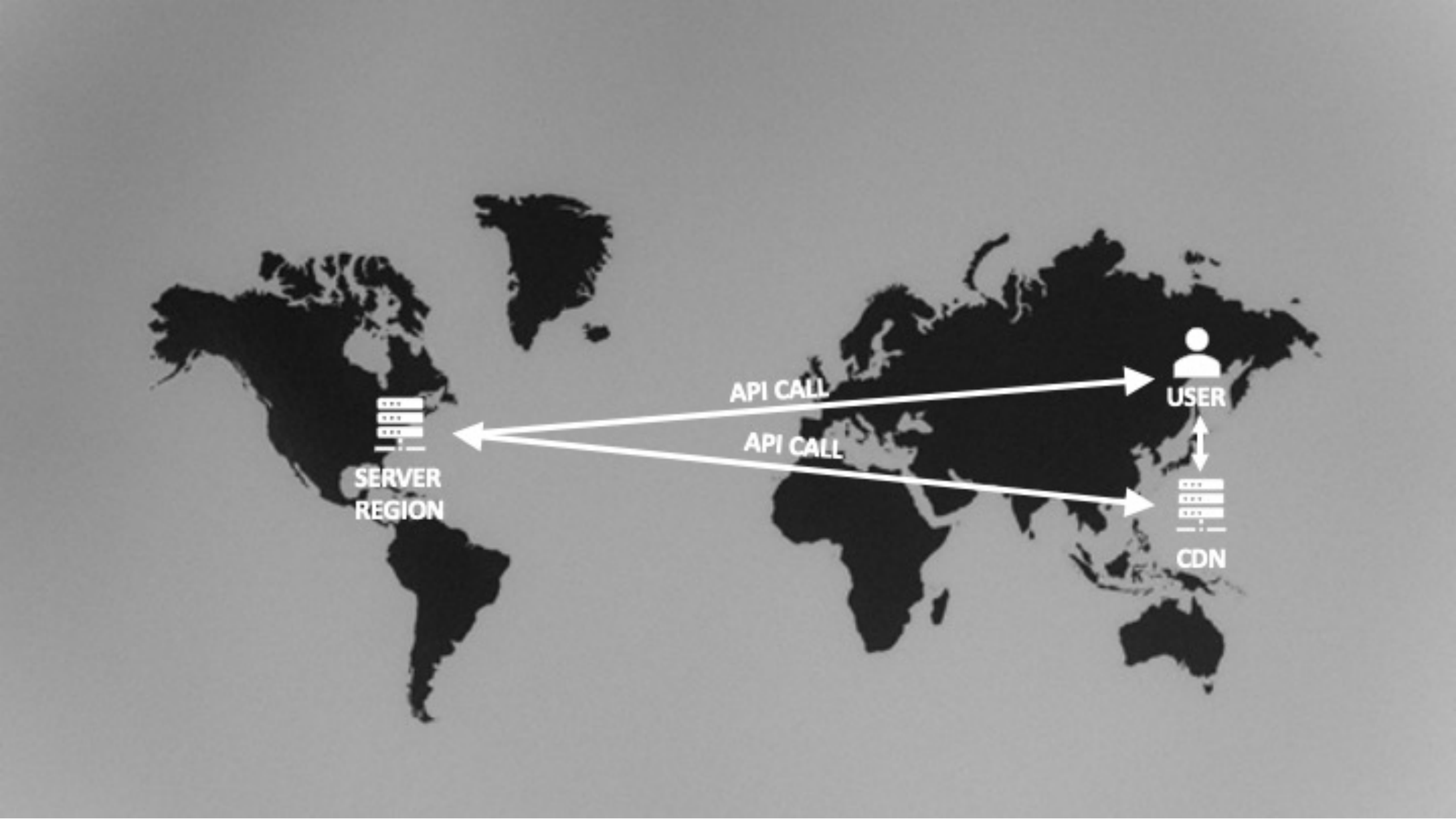
And the edge in particular already has some baggage. As you hopefully could tell from the intro, I am also a firm believer in the Jamstack architecture for building sites, which deployed site assets to the edge. But as Chris Coyier points out in this post fron May, you probably still need data and that data is still coming from a specific location or region. https://chriscoyier.net/2022/05/04/it-doesnt-much-matter-how-cdny-your-jamstack-site-is-if-everything-important-happens-from-a-single-origin-server-edge-functions-are-probably-part-of-the-solution/

Name	Code
US East (Ohio)	us-east-2
US East (N. Virginia)	us-east-1
US West (N. California)	us-west-1
US West (Oregon)	us-west-2
US GovCloud West	us-gov-west-1
US GovCloud East	us-gov-east-1
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1
Canada (Central)	ca-central-1
Europe (Frankfurt)	eu-central-1
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Europe (Stockholm)	eu-north-1
Middle East (Bahrain)	me-south-1
South America (São Paulo)	sa-east-1
China (Beijing)	cn-north-1
China (Ningxia)	cn-northwest-1

For example, if you're on AWS you'll deploy your lambda function to one of their 27 different regions across the globe.

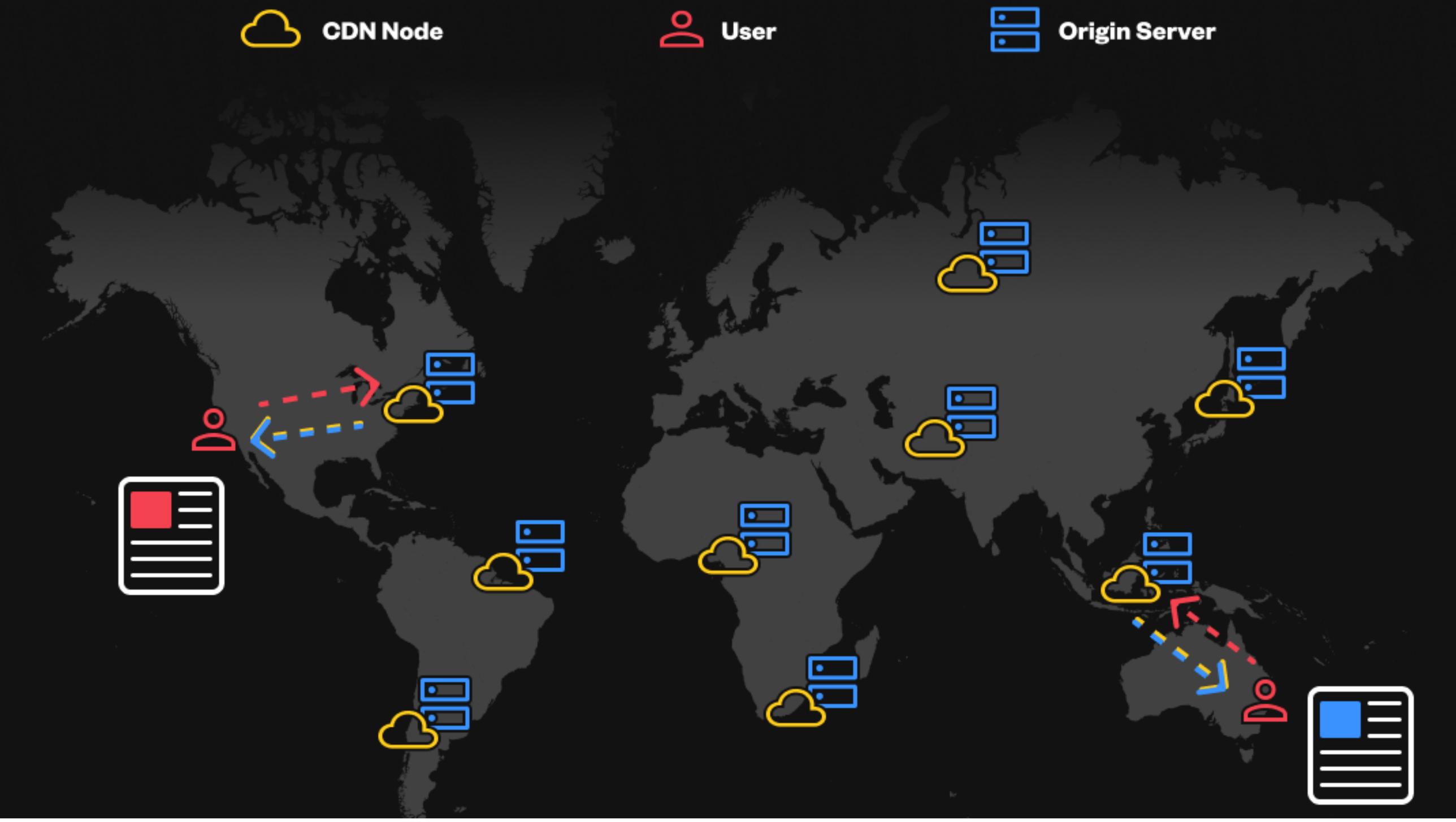


in essence we moved from a model that looked like this, where the user, regardless of their location, was served a site from a single region.



To a model like this, where the user would be served static assets from a region close to them but then all of the data which came from APIs was served from a single region. This means that the static assets load fast, but similar latency still exists populating the page data.

Edge functions aim to solve this problem, by bringing some - or all - of the backend processing up to the CDN level in proximity to the user. So it's no longer just static assets on the CDN but actual compute is being performed there as well.



In an ideal world, it might look something like this image taken from the Remix blog, where every user request has static assets fulfilled by the CDN and backend assets also fulfilled by the CDN via edge functions.

Promise: Reduced latency

- ✅ Backend calls go to the server geographically closest to the user.
- 🤔 Geographic latency may not be the critical issue in your site's performance

So that's the biggest selling point of edge functions: that they will improve the peformance of your app by reducing the latency. And they do actually achieve that promise in terms of the actual function call. However, if your edge function needs data, that data may still be region locked. And anyway, unless you are building games or IoT, that latency caused by distance, while not worth dismissing, may not be the critical aspect to improving your site's performance.

Just How Much Does Latency Affect Your Speed?

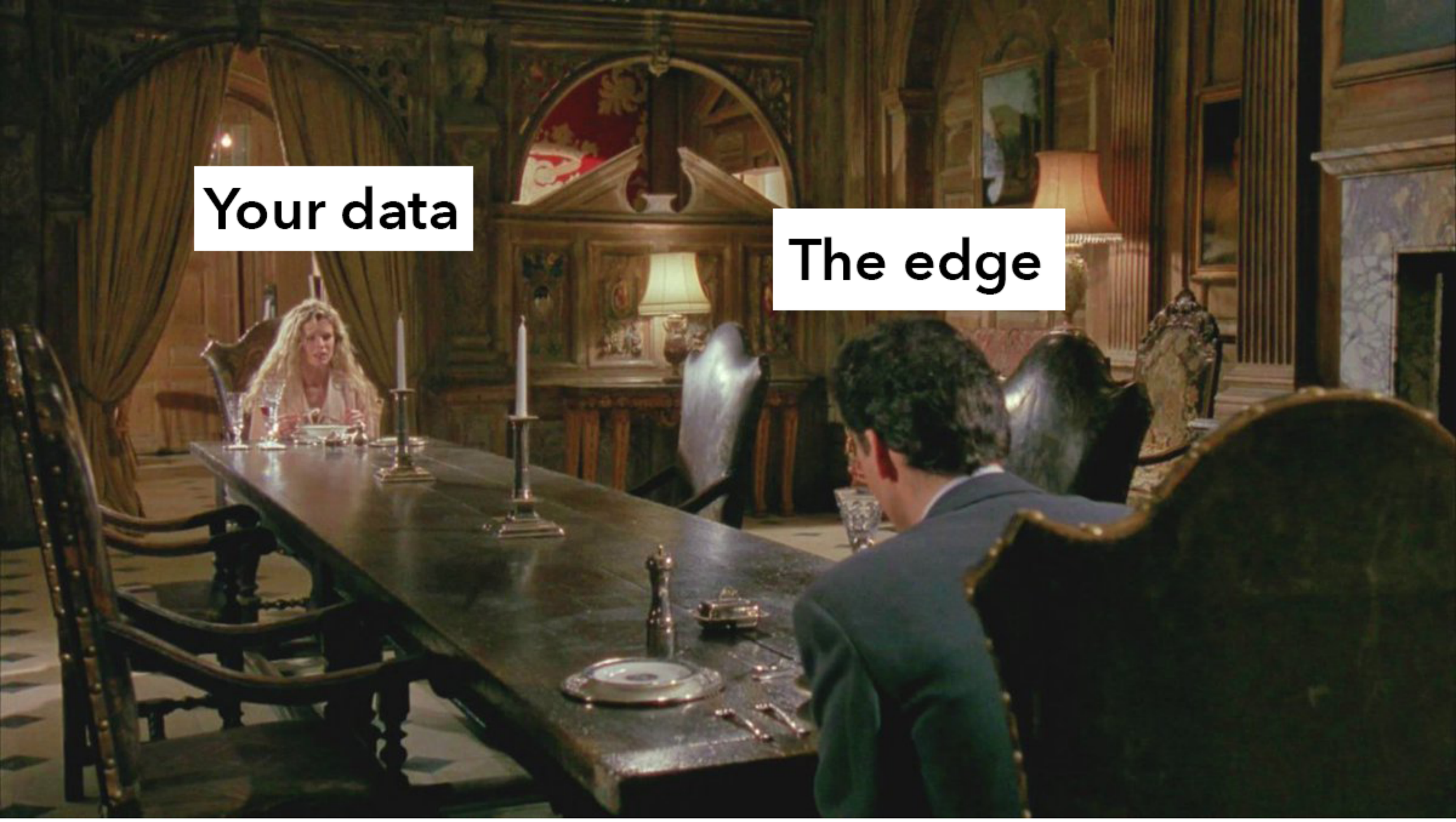
Well, let's consider the perfect example of having a high-speed fiber-optic network to every point in the globe and a speed of light that is approximately 200,000 km/second.

At that insanely fast speed, it will take around **150ms to reach from the US West Coast to central Asia**. This theoretical scenario assumes that every point on the internet is a fiber-optic network, which we know is not the case for every location in the world, and ignores the impact of routing devices and the numerous hops that traffic will need to make along the way. **In reality, such a trip across the globe would take several seconds to complete.**

However, even if we assume the fastest scenario of 150ms to connect to a website across the globe, a user will start to notice the impact. As data needs to move both there and back, it will mean around a 300ms delay in data getting back to the user.

300ms might not sound like a noticeable delay when surfing a website if we think of it as a one-off delay. But we are not sending data only once. The reality is that there is a constant stream of messages going back and forth to provide relevant content and to ensure the reliability and security of the network. That 300ms delay affects every one of those communications.

I'm not trying to dismiss the cost of geographic latency as it can add up. As this article points out, data going from US West to Central Asia, for example, could take up to a few seconds. So this is a legitimate selling point that edge functions can live up to when used properly. https://www.snapt.net/blog/how-geographic-distance-affects-latency



What do I mean by "used properly"? Well that gets at the biggest potential drawback to edge functions when it comes to reducing latency.

Promise: Reduced latency

- ✅ Backend calls go to the server geographically closest to the user.
- 🤔 Geographic latency may not be the critical issue in your site's performance
- ❌ Your database or other backend APIs may still be region locked

And that's that your data in the form of APIs, third-party services or a database may still exist in specific region. There are solutions that can help mitigate this such as edge enabled databases but the key thing to remember is that, if you're going to access data in your edge function, you need to have a plan or you may not see the performance improvement from reduced latency you'd hoped for.

Promise: It's easy

- ✅ It's just JavaScript, so there's a low barrier to entry and lots of framework support
- 🤔 Runtimes are often not Node-based, so you may not have all your favorite tools
- ❌ Every provider is unique and choosing one isn't easy

Another promise you may hear about edge functions is that it's super easy to get started. And there's truth to this. In every case you can just use JavaScript, though some providers do support other languages as well. And lots of frameworks are available to help you build them. But the runtimes are mostly not Node-based, so you can't always bring your npm-based tools with you. And each provider implements them differently so it can be tough to choose and difficult to learn their offering and quirks. And your code is definitely not easily portable.

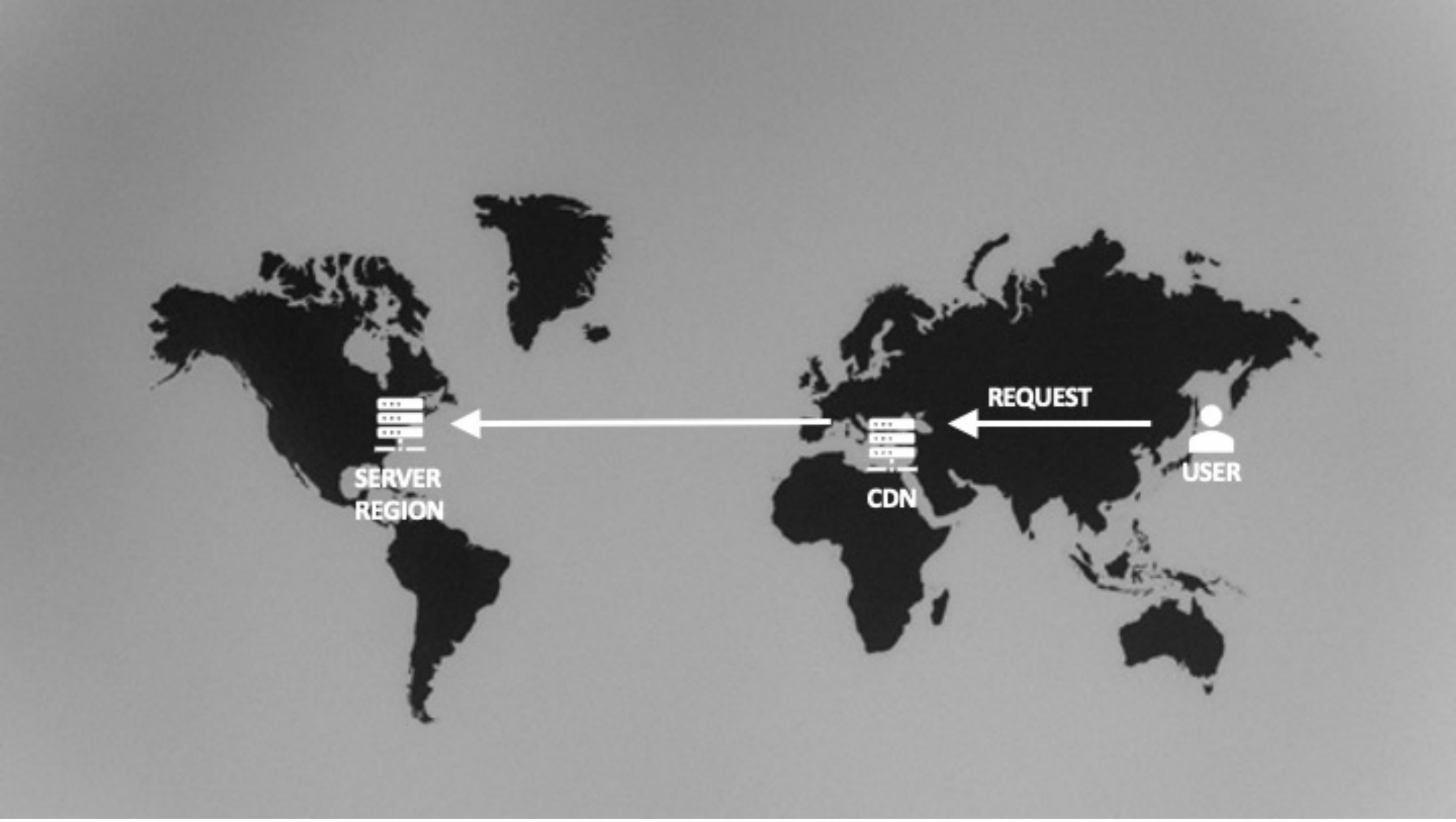
Promise: Additional capabilities

- ✅ Edge functions can intercept the request or response
- 🤔 Um...actually, this is just cool
- ❌ I got nuthin

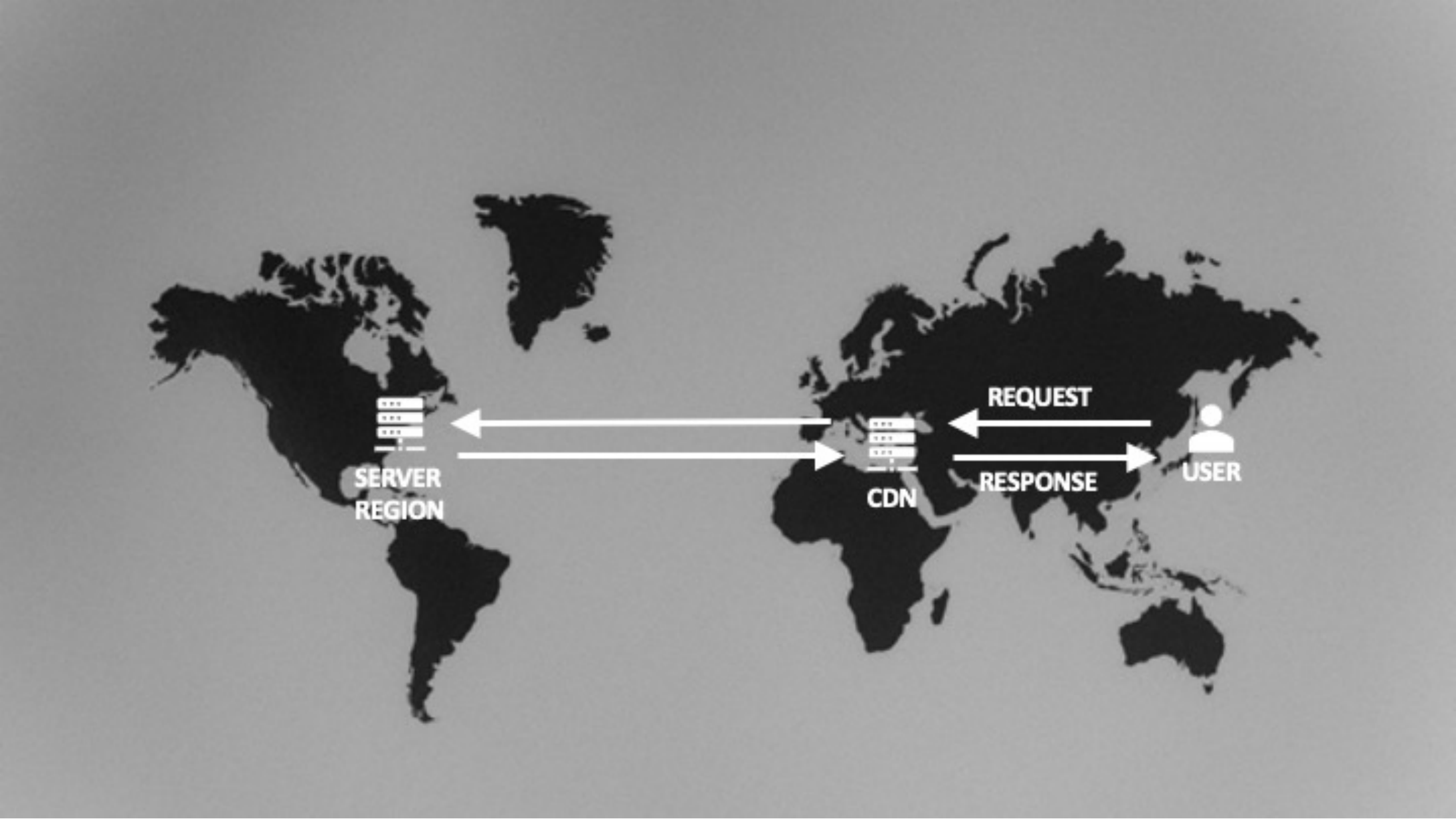
Where I get more excited about edge functions from a developer standpoint is where it can actually allow me to do things that previously were much more difficult. And that is where the ability to intercept the request and response comes in.

Essentially, your edge function – or middleware as some frameworks call it – will be called when the request comes in before it hits the origin server. It can modify that request, redirect it - so no need for slower server-side or client-side redirects. You can even handle access and authentication here.

Or it can even fulfill the response itself. For example, if you have static assets like HTML, CSS, images and more cached locally, you can simply respond without hitting the origin server. This is actually how Cloudflare Pages, which we'll talk about, works.



If you haven't redirected or responded directly, the request continues to the origin server to be fulfilled.



When that response comes back, your edge function can intercept it again. In this case, you can do things like modify the response. For example, you might add headers that have user specific information or you may even modify the HTML of the response.

Add personalization to static HTML with Netlify Edge Functions — no browser JavaScript required

by Salma Alam-Naylor

11 Aug 2022

8 min read

Table of contents

Personalization — the process of creating customized experiences for visitors to a website — is a hot topic on the web in 2022. In a crowded world of digital experiences clambering for our attention, businesses are turning to personalization to connect more deeply with their customers, and ultimately, generate more sales and growth.

And shipping less client-side JavaScript is an even hotter topic. New and emerging front-end frameworks such as Astro and Eleventy are pushing the boundaries of the JavaScript framework era of the late 2010s, making it their core mission to ship less and less JavaScript *by default*, and bring developers' focus back to a lean, native and performant web.

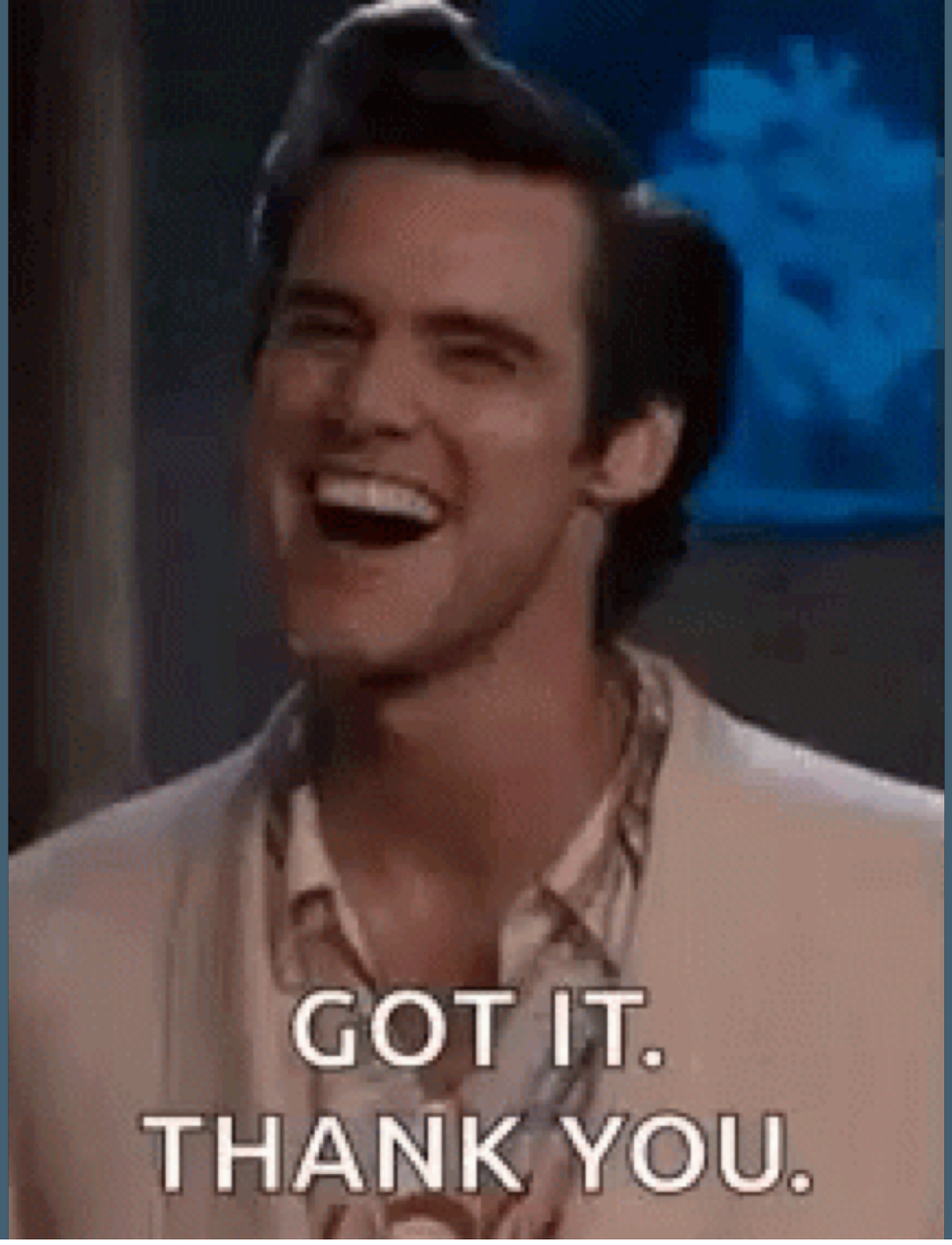
The great news is that with Netlify Edge Functions you can achieve dynamic personalization with *no* client-side JavaScript — resulting in a great developer

From a front-end developer stanpoint, this enables you to do some really cool things such as make a dynamic, user-specific response based upon a completely static HTML asset. Salma Alam-Naylor has a number of examples doing just that using Netlify's edge functions. https://whitep4nth3r.com/blog/add-personalization-to-static-html-with-edge-functions-no-browser-javascript/

Use Cases

- A/B Testing
- Updating/Modifying/Custom HTTP headers
- Conditional routing
- SEO
- User authentication/authorization
- IoT/Gaming
- Personalization
- Compliance

With all that in mind, here are some of the things you might consider doing using edge functions. You can replace out content for A/B tests without any flash of rendering. You can add custom http headers. You can route user's to a new location. You can serve a different version of your site for search crawlers. You can check for authentication and redirect otherwise. Remove critical latency on games or IoT apps. Personalize the output, for example show login state without flash of rendering. Serve up different versions depending on the user location to meet location specific compliance issues.



Hopefully that gave you the gist of what edge functions are and where they can be especially useful. Let's move on to the various options for implementing them.

Core Edge Serverless

- Cloudflare Workers
- Deno Deploy
- AWS (Lambda@Edge, Cloudfront Functions)
- Fastly Compute@Edge
- Stackpath
- Azion

When I think about edge functions there is basically two different types of offerings. The first I'm calling core edge serverless. These are largely CDN companies who have added compute offerings to their products. This isn't a complete list but some of the more well known players.

Edge Platforms

- Netlify Edge Functions (Deno)
- Vercel Edge Functions (Cloudflare)
- Supabase Edge Functions (Deno)
- Cloudflare Pages Functions (Workers)

Then there are what I am calling edge platforms. These are edge function offerings that are built on top of those core edge serverless offerings but also provide improved developer expeirence and integrated continuous deployment.

Framework Support

- Next.js
- Astro (Example the Netlify SSR Integration)
- SvelteKit (See examples from Vercel and Netlify)
- Eleventy (Using the Eleventy Edge plugin that is Netlify only)
- Remix (Using adapters)
- Fresh (Built by Deno)

There are also the front-end frameworks that help you build applications using edge functions usually by adding conventions that hide the underlying implementation and complexity from you.

Edge Enabled Databases

- Fauna – A serverless database with built in global replication and strong consistency.
- Planetscale Portals (public beta) – A serverless MySQL database with support for replicating to read-only regions.
- Cloudflare DI (still in private beta) – A SQLite based relational database that has globally replicated reads.

When it comes to moving your data to the edge, there is a growing list of offerings that are full databases - rather than just cache or KV stores - that are edge enabled. In most cases, the reads are replicated but the writes are region locked, which is likely sufficient for your needs. It's worth keeping in mind tht most of these are eventually consistent and there may be a lag.

Cloudflare Workers

Let's start our more detailed look with Cloudflare Workers. In this area, Cloudflare has arguable one of the most mature products. They also have an extensive network of 275 CDN locations where your function gets repicated to.

Cloudflare Workers

Workers can be deployed via the CLI tool called Wrangler. They run in the Cloudflare Worker Runtime based on V8 that supports JavaScript/Wasm.[1]

[1] The core Workers runtime was recently open-sourced under the name Workerd (meaning you can self-host Workers).

Workers use non-Node runtime basd upon V8 that allows you to build functions using JavaScript. They also recently open sourced this runtime under the unusual name of Workerd, which allows you to self-host Workers if you choose to.

Cloudflare Workers

A default worker::

```
addEventListener('fetch', event => {
    event.respondWith(handleRequest(event.request));
});

async function handleRequest(request) {
    return new Response('Hello worker!', { status: 200 });
}
```

The `fetch` event is any incoming HTTP request (ex. `*.workers.dev` or a custom origin). In most cases you will send a response, and if you are using a workers.dev it's required. This can be a custom response like the one above or the result of a fetch to return resources.

Cloudflare Pages Functions

- Automatically deploy Workers from a Cloudflare Pages site by placing Worker files in the `functions` directory (source).
- Currently in beta but is coming out of beta any day now and billed the same as Workers.

You can also easily deploy workers as part of a project hosted on Cloudflare Pages by simply placing your Workers files in a functions directory.

Cloudflare Pages Functions

```
├── ...
├── functions
│   └── api
│       ├── [[path]].ts
│       ├── [username]
│       │   └── profile.ts
│       ├── time.ts
│       └── todos
│           ├── [[path]].ts
│           ├── [id].ts
│           └── index.ts
└── ...
```

This allows for things like dynamic routes and catchall routes for your Worker-based application API. You can also add middleware in any or all of these directories which will run on every call.

Data at the Edge with Cloudflare

1. **KV** - A simple key value store that replicates to the edge. Eventually consistent.
2. **Durable objects** - Object storage (the stored objects are an instance of a class in JavaScript). Objects can have storage attached and exist in one location at any one time, giving them strong transactional consistency. This is available only on paid accounts.
3. **R2** - This is S3 compatible object storage. Available free with a pay-as-you-go option if you exceed the free limits.
4. **D1** - A SQLite-based SQL database with read-only replication to the edge. Currently only available in private beta.

Cloudflare has a ton of edge data options depending on your needs. They offer everything from a key value store to various degrees of caching and even a edge enabled SQL database that is currently still in private beta.

Cloudflare + LaunchDarkly

Gist: https://gist.github.com/remotesynth/c308dff55252bbc4e98500eff9d3433a

Before we move on, I wanted to show one more quick example because it illustrates two things, Cloudflare's HTMLRewriter that makes it easy to modify the response HTML and LaunchDarkly's edge integration for Cloudflare. This example uses the HTMLRewriter combined with LaunchDarkly's edge SDK that works with Cloudflare to modify headers and run A/B tests using a Worker.

Deno Deploy

- Supports 34 Regions
- No local KV/data support but you can use external services or *read* cached files on the file system (there's no support for writing to the file system from an edge function however). Support for the cache API is on the roadmap
- Support JSX (and TSX) out of the box which can be useful for SSR using the `renderSSR` function (source)
- Also not Node-based but a decent number of npm modules have been ported to Deno.

Deno has 34 regions currently on their edge. One big drawback is the lack of support for ata or a cache API on the edge, however Deno does support outbound TCP and TLS connections to connect to databases like PostgreSQL, SQLite or MongoDB. There's good examples of connecting Deno to edge data services this way. They support JSX out of the box, which can be useful for SSR React apps. While Deno is not Node-based, there are a number ported and you can use npm modules that aren't ported provided they use ES Modules format, though this means a lot of existing modules may not work.

Deno Deploy

Gist: https://gist.github.com/remotesynth/97c69486ab8defd0baa17c238c99cae2

This is a simple example that shows me calling the LaunchDarkly API to make changes to a simple HTML response. Note that I can't use the existing Node SDK because it won't work in Deno currently. (In the Deno dash https://dash.deno.com/playground/salty-donkey-32)

AWS Lambda@Edge and CloudFront Functions

- Lambda@Edge works just like Lambda but is replicated across 13 regional edge caches.
- CloudFront Functions are replicated across 218 edge locations but are currently limited to view triggers and have limited capabilities.
- No edge databases

I want to cover AWS quickly even though it's still the giant when it comes to serverless, but it's edge offerings are pretty standard and, to be honest, are a little confusing. Lambda@Edge lets you deploy a regular Lambda to of it's 13 regional edge caches. If you need more edge nodes, CloudFront Functions offer that but you are extremely limited in what you can do. They don't have any specific edge data offerings that I am aware of, so your data is still probably region locked - though they seem to release a new service daily.

Honestly, I find the differences between Lambda@Edge and Cloudfront functions difficult to understand but this article was the best explanation I've seen. Basically Cloudfront functions are great for things like programmatic redirects or auth. Most everything else you'll use Lambda@Edge. https://www.honeybadger.io/blog/aws-cloudfront-functions/

Netlify Edge Functions

- Add a `netlify/edge-functions` directory at the root of your project.
- Edge functions provide geolocation information in the `context.geo` variable.
- Netlify also has a Next.js Advanced Middleware package that includes things like modifying headers and content. source.

Netlify makes deploying edge functions to Deno for your project pretty easy and integrated with your project code. The easiest way is to just add a netlify/edge-functions directory to your project, though this is configurable. One cool thing is that the edge function automatically provides geolocation data. They also have an advanced middleware offering that does some cool stuff for Next.js to make it easier to modify the source. Otherwise, it's the same as Deno in that you can't use many npm modules and there isn't an edge data or caching offering.

Netlify Edge Functions

```
import { Context } from "https://edge.netlify.com";
import { DOMParser } from "https://esm.sh/linkedom";

export default async (request: Request, context: Context) => {
  const url = new URL(request.url);

  // Look for the query parameter, and return if we don't find it
  if (url.searchParams.get("method") !== "transform") {
    return;
  }

  const response = await context.next();
  const text = await response.text();

  const document = new DOMParser().parseFromString(text);
  const h1 = document.querySelector("h1");
  h1.innerHTML = "Thanks for visiting from " + context.geo.city + "!";
  return new Response(document.toString(), response);
};
```

This simple example Netlify Edge Function uses the geolocation information and and a DOM parser written for Deno to modify the HTML response.

Vercel Middleware and Edge Functions

- Powered by Cloudflare Workers (though only 20 regions supported)
- Edge functions are built like regular API functions but with the `experimental-edge` runtime defined
- You can't access the KV but cache is available in Edge Functions
- Middeware runs on every request for applicable paths before hitting the origin while edge functions are an endpoint
- Include built-in geolocation and IP information

Vercel's edge functions run on Cloudflare, but currently support 20 regions. While you don't need to use Next.js to use them, there are a ton of features built into Next.js that integrate directly into Verce's offerings.

Questions?

- Twitter: @remotesynth
- Mastodon: @remotesynth@mastodon.xyz
- Email: brinaldi@launchdarkly.com