EPISODE 1168

[INTRODUCTION]

[0:00:00.3] JM: Netlify is a cloud provider for Jamstack applications. To make those applications more performant, Netlify has built out capacity for edge computing, specifically Edge Handlers. Edge Handlers can be used for a variety of used cases that need lower latency, or other edge computing functionality.

Matt Billlmann Christensen is the CEO of Netlify and joins the show to talk through the engineering behind Edge Handlers.

[INTERVIEW]

[00:00:29] JM: Matt, welcome back to the show.

[00:00:31] MBC: Thank you.

[00:00:32] JM: As you know, your company, Netlify, started mostly focused on static sites, but has expanded into more and more dynamic content. What have been the difficulties in expanding into increasingly dynamic websites?

[00:00:46] MBC: It's always been the vision since the beginning. When we first started talking about Jamstack, we were looking at the whole ecosystem spanning, of course, static site generators, but also the single-page application frameworks and all the gray areas in between. Of course, right since the beginning, app.netlify.com has always been a very dynamic application running on Netlify and being built with Netlify.

I think, of course, when AWS Lambda came around, there was one of the moments where we started seeing that there was a real opportunity for the front-end presentation layer to also own a piece of their API responses, or glue code, or any of those pieces as part of this layer of the stack that should be really interesting from the front-end web developers.

I think it's been a very gradual path to introducing more of the capabilities that makes building dynamic applications with a Jamstack approach, easier and introduces less friction for the developers. Then of course, as we keep just looking at our user base and talking to our customers and talking to enterprises and so on, we just keep looking at where is the friction, right? What is it that makes life harder than it should be for development teams?

Then we start thinking, “Okay. Where is it that we can introduce the right level of abstraction that will simplify things and make it easier for development teams to build and deploy and create great user experiences?”

[00:02:23] JM: Right. That brings us to today's topic, which is Edge Handlers. Can you explain what an Edge Handler is?

[00:02:30] MBC: Yeah. Early on when we launched Netlify, we took this early decision of building our own edge network from the beginning. We did that, because traditionally, you had traditional CDNs that were really built to sit in front of an organ, and then proxy requests through an organ web server and implement some different caching rules and so on. We wanted to fundamentally get rid of that concept of your web server. We wanted developers to just push to get and now they're front in this live on a globally distributed network.

As we started down that path, we very quickly realized that that network would need certain capabilities, because all of the things you would traditionally do in your web server, you would now need to be able to do on the edge, things like proxying and redirect and rewrite rules. Even a bit of the more complex use cases, like geolocation or role-based authentication and so on.

The reason we built our own edge network was that we knew that at scale we would really need to own the software running on the edge to solve those problems for developers. Since the beginning, our edge network had a declarative rule set, where you could use an underscore redirects file in our system to define these rules. That solves a broad set of use cases.

As we talked to larger enterprises and as we look into just in general, what's also happening in the framework, landscape and so on, of course, we can see that there are things at that edge layer that you can't just do with a declarative rule. This can be anything from things like, imagine you're a retailer and you have stores in different specific cities in specific locations and you want to make sure that if a user opened up your website and they are close to one of those stores, they automatically see the store they're close to right on the front page, because that's probably what they're interested in.

That's something that you can't define just by saying if the user is in this country, or whatever, then show this. You need to do the mapping on the fly between stores and use the location and see if there actually is a store and take a decision. That was one of the problem that people couldn't quite solve in this way. More fine-grained authentication is another case. Right now, you can put up rules based on just the role, but sometimes you have to take decisions based on the specific user and what they have access to and so on.

Fine-grained personalization, like for e-commerce and multi-variant testing, or all of these things are also something that gets more complex. Then there are framework use cases, like how could we easily integrate things like image resizing into a framework and just have the edge pick up the right query parameters and send your requests on to Cloudinary. There's a whole set of actual use cases that we saw could be really useful to solve for.

The way we've solved for it is to build what we call Edge Handlers, which gives developers the full capability of programming our edge layer in JavaScript. You can essentially, completely take control over the request response cycle directly at the edge nodes and run a program that can either just quickly intercept the request and send it somewhere else, or completely take over the request response cycle and render and return a response. Or you can do response transformations. You can pass through the request to what we would normally serve and then run a streaming transformation on that response to insert personalized blocks, or anything like that. It becomes a really powerful primitive that developers can build with and that framework offers and build plugin offers and so on, can use to offer more advanced edge functionalities.

[00:06:26] JM: That programmability, is it handled by some AWS Lambda-like functionality, or what exactly is the compute layer that's serving these programmable requests?

[00:06:40] MBC: Yeah, that's been what's been a lot of work since we have built that ourselves on top of isolate. Basically, relying on V8's concept of isolate as the way we can spin up these programmable logic for very, very little cold start overhead. Again, it's also a programming layer that's a bit more – that's more restricted than what you would get in a Netlify function, for example. That Netlify function will still be the right place to go to write an API endpoint in Node.js and so on. The edge endless does come with much stricter execution timelines and so on, but also with no cold start and very fast response times.

[00:07:21] JM: We've done a show on isolates. Could you talk a little bit more about what an isolate is?

[00:07:25] MBC: Yeah. Isolates is like the V8 JavaScript engine. It's a concept they have for running a bit of JavaScript in a context that's completely locked down from the rest of the system. Of course, when you think of JavaScript, the main use case for JavaScript originally in the recent Google build V8 was to run other people's code that your browser just randomly downloads from the Internet and run it safely in your browser, without giving malicious access to your local file system, or anything like that.

Of course, this concept of isolation and security have always been really key to JavaScript as a language. That's what comes in handy once we start running multi-tenant code also at an edge network, where we need to be able to guarantee that one tenant can only get access to a certain amount of resources and that we can stop the execution if it takes more CPU time than what's allotted to it and that we can guarantee that the code running in that isolate can inspect other user's memory areas, or anything like that.

Then of course, V8, when it runs a program, well of course, first pass the JavaScript and turn it into an optimized TST and everything. Part of what we also take advantage of is that we can

distribute the result of that, rather than the raw JavaScript, so we can get this very easily resumable compute environment that can process requests and responses.

[00:09:03] JM: Could you walk me through the life cycle of a request that gets handled by an Edge Handler?

[00:09:08] MBC: Yeah. In general, the main point is when a request comes in, which is the on-request handler, where we have the headers of the request available, the path, everything and the Edge Handler can inspect all of those to potentially take decisions. At that moment, you could change part of the request. You could change the actual URL of the request, until our proxy cache. You should actually proxy this request somewhere completely new, or invoke a function, or anything like that, we could.
In that process, you could also proxy it somewhere and have the Edge Handler append headers to the request object and so on. There's first that on-request part. Then of course, there's an on-response, like once you've processed that request and done whatever you want with the request headers and the request object, our proxy cache will process the request and decide whether it needs to invoke a function, serve a static file, or a proxy somewhere.

Once that decision has been taken, you will get the on-response handler where you can do things to the response. You could append headers to the response, or you could even intercept the response and do things to the response body, either by first reading the response body and then deciding what to do based on that. Or you could do a streaming transformation of the response body, where you essentially put in a processor that as our system streams, the response body to the end-user can make changes on the fly.

[00:10:46] JM: What programming languages are supported?

[00:10:48] MBC: Right now, from the outset, we support JavaScript. Of course, there's a lot of languages by now that compiles down to JavaScript, like typescript and so on. We're also looking into the right way of supporting WebAssembly to open that broader up and we might also explore native support for typescript, rather than having to bundle typescript first.

Right now, the underlying implementation is based on Deno as a runtime. Deno supports typescript natively. Right now, we're a little more conservative and allow just JavaScript right away in the basic stage.

[00:11:23] JM: I'd like to talk a little bit more about your edge network. We've given a high-level view at this point for what Edge Handlers are, but tell me about the edge network. How many nodes are in it? What is it deployed to? Where is the edge network?

[00:11:37] MBC: Yeah. We run a couple of separated networks from an infrastructure perspective. All of our free and self-serve traffic run out of one network. The providers we use changes a bit. In total on all of our network, we are basically running across all the cloud providers, like any way from DigitalOcean, Azure, Amazon, Google, Alibaba, Yandex, Packet and so on.

Then on our free and self-serve network, we'll tend to prioritize more being able to offer people solid robust to edge network with affordable bandwidth prices and so on. Like on the enterprise tier, we'll optimize for the size of the network and for the performance for enough cash for more cash-based per user and higher reliability and redundancy.

[00:12:32] JM: How do you manage that edge network? That's a lot of servers in a lot of different places.

[00:12:37] MBC: It's a lot of servers with a lot of different places. Of course, we spend a lot of time building the automation infrastructure to manage all of that, from terraform plays a key role in the actual provisioning of infrastructure across all of these different providers. We're using Ansible for the initial setup control templates. We run a console cluster as well to distribute settings and so on. It's for sure a big machine to operate. Of course, also lots of tooling around monitoring, alerting, filtering all of that and a sizable SRE team that focuses on that part.

Part of our infrastructure will be for example, our DNS controller that constantly monitors every single CDN edge node. If it detects issues on any node, it'll immediately roll it out of the rotation at the CDN network and also, allows us to configure a network pattern in general. For example, as an interesting example for a while, we had issues with Russia blocking access to AWS instances and the like, that created connectivity problems for users in Russia.

We rolled out a presence inside Russia on Yandex. When we rolled that out, we started getting reports from users in Ukraine that they could no longer access their sites, because they got routed to the closest edge node, which was in Russia and Russia was blocking Ukrainian traffic. Now we had to tell our infrastructure, if you are in Ukraine, even if the closest pop might be Yandex, we need to send you somewhere else. All of that is also a lot of constant work, of course.

[00:14:22] JM: Can we talk from the outside looking in a little bit? When a request comes from Ukraine, for example, what's the series of events that happens to respond to that request?

[00:14:36] MBC: I mean, first, there'll be a DNS lookup that'll send you to the best available CDN server. As I said, if some CDN node has an issue, we'll remove it from rotation. You'll get routed to the best available CDN session. You'll open up a connection there and start a TLS handshake to verify the certificate for your domain. In our system there, since we run millions and millions of sites, we'll have a smart way of finding the right certificate and caching that at the edge as well and getting it resolved quickly.

Then you'll start the actual HCP cycle where our system will process request headers and everything and figure out what to do. If there is any rules, it’s rules defined for your site, they will get invoked directly at the edge and that could involve starting up an Edge Handler, or calling into an Edge Handler to see if that wants to do anything with the request. There'll be a whole state machine running throughout based on continuations and so on. Of course, our edge nodes needs to manage very high amounts of concurrencies, both across threats and event loops.

They will process that request. Once the request processing, which is both the edge rules, or the Edge Handlers are done with the request, we'll do a cache lookup to see if we have a response for that finally processed request already lying around in our cache. If we do, we'll just serve this response. If there's an Edge Handler in place, we might of course do a streaming response transformation, or we might have our system do broadly encoding, or things like that.

That's actually the typical case. We have a cache hit rate in the high 90s. In so in most cases, especially on the enterprise CDN where there's plenty of cache space, that's exactly what will happen. A request will come in, we'll find the right cached object and we'll serve it back within a millisecond or so.

In other cases, we might find that there's nothing in the cache, we then need to either fetch the static file and put it in the cache and serve it, or we'll detect that the request is a function invocation, where we'll send it to a little invocator that will just set all the right client context, all of that invocate in AWS Lambda and send back the response. Or we might see that it's a proxy request, like in which case, will just directly from the edge node, append some signature headers and so on, that the other organs can use to verify that it comes from us. Then just proxy the request directly to another origin. That's the typical life cycle of a Netlify request.

[00:17:24] JM: How does authentication work for these Edge Handlers?

[00:17:28] MBC: We already have a concept of authentication, even with our declarative rules sets, based on JSON web tokens. If a request comes in and it has either a cookie with a JSON web token, or an authorization header with a JSON web token, the edge node will see if that domain has some identity service configured. By default, we have our own identity service, but on higher plans, you can also put in a custom secret to verify tokens with and so on.

If that's in place, the edge node will check that the integrity of that token. If it is a verifiable token, it's then exposed both to the rules engine, where you can use rules based around claims in the token. Or if it's a function invocation, we'll have all those claims available in the

client context for the function. The same if it's an Edge Handler, you'll simply just have a context in the Edge Handler, where you can look up and see if there's already a verified JSON web token, and then you can easily use those claims in your custom logic to take decision-based on what do we want to do with this user.

[00:18:41] JM: Okay. We've spoken more about how Edge Handlers work than their actual applications at this point. Could you give me some examples for why edge handling would be useful?

[00:18:50] MBC: Yeah. I mean, there are many different kinds of them, ranging from things you could do at the framework level, to things you could do for enterprises and so on. A really common use case for them is around personalization, especially if you're building – we hear that especially from e-commerce customers, where being able to do personalization on the fly can be really important. That's one part where just being able to do that as for example, as a response transformation, or just deciding when the request comes in, whether to show one pre-built page, or another. That can be really powerful.

More advanced proxying, or almost API gateway-like functionality can be really powerful as well. When a request comes in you, can decide at different origin, you can rewrite to a different site, or different project, or you can rewrite to a completely different service. You can append the right headers if you want to proxy to a secured API. You can verify a web token, decide based on the claims of that token to set certain headers in an API request to an external service.

I mentioned the retailer concept. That's when we've heard often that using more advanced geo-localization features. We did a small demo of that at the Jamstack conference around, I think we called it Votelify, where as soon as you go to the page, we'll detect what's the closest voter registration location based on your location and we'll show that directly on the page. Then there's also a framework like features. For example, imagine you want to build a component in React for images that can be resized on the fly as they're served. You might simply have a little component that where you set different image transformation options and

then it appends those in some format to the image URL as query parameters. Now you could easily write a little Edge Handler for Netlify that'll detect those query parameters and see if those are present and maybe if they're assigned in the right way, then it can rewrite the request object to point to Cloudinary with the right settings for Cloudinary.

Now Cloudinary can do an image transformation, return the transformed image and we can cache it on our edge servers and send it back to you. Edge Handlers would be a really easy way to accomplish something like that without us specifically having to build it as a feature. You could imagine the similar things, if you wanted to experiment with a bundler outputting device specific bundles for bundle size optimization, for example.

You could then write an Edge Handler that when you request the JavaScript bundle, will check the user agent header and everything. Then based on those characteristics, return the right bundle for the device that made the request. A lot of these things become programmable in a way that can be really interesting for framework authors to take advantage of.

[00:21:58] JM: Tell me about some of the engineering challenges that emerged when you were building Edge Handlers.

[00:22:03] MBC: The biggest challenge is running the Edge Handlers in context of a proxy cache, where we wanted to be able to do things, like response transformations. We wanted the Edge Handler to be able to interact with the caching software as well. Giving access to storing and getting objects in a space of the cache specific to their genders and running transform, like running response transformations and so on.

Because it's proxy caches tend to be pretty advanced pieces of software, handling very high levels of concurrency with lots of IO going on and so on. Figuring out how we could take that layer and build a bridge between the code running in an isolate and the actual proxy cache code. That's been one of the big challenges, where we had to before even really building the Edge Handler, start rebuilding a lot of the code that actually runs our rules and in everything. We completely rewrote from scratch in Rust, migrating from C++ to make it more easier for the

team to work on and a better developer experience in our end. That's been a really big project starting at the start of this year for us.

[00:23:22] JM: Are Edge Handlers useful for doing something like AB testing?

[00:23:27] MBC: Yeah, absolutely. That's one of the popular use cases we see around it. You can imagine doing it in many ways. If you essentially have some way of bucketing the users into experiments and that can be something an Edge Handler can simply do by. If a request comes in and it has no cookie set for the experiments it's looking for, can assign some random number to a cookie and set it on the response.

Then it can either pick the response, like imagine that you just have for example, let's say you have your pricing page. You want to run an experiment with three different versions of the pricing page and see which one performs best. You could do that simply by having like, when you build the site, building three different pricing pages, pricing page A, B and C. Then you can have the Edge Handler look for the cookie. If there's no cookie, assign a random value. Then based on that value, pick either A, B and C, rewrite the request and then just let the cache serve the right page.

Then you would probably make sure, either to bring that cookie all the way to the client side, or to inject a little code in the response to track which bucket the user fall into and pass that along to whatever analytic system you're using, so you can start seeing what was the behavior of users that saw pricing page version A, B or C. That's one quite robust way of doing experiments on a page-by-page basis.

You could also imagine, even again, building your own little AB testing framework, where if you have a component, you might have an experiment component that gives the experiment a name and then supplies for that specific component three different HTML variations you want to test against. Then you can have an Edge Handler, again, doing a similar trick as assigning a cookie, deciding how to bucket the user and then doing a response, streaming response transformation, where as you're streaming the response, you'll ignore the components that's

marked as experiments but not part of this user bucket and you'll include the components that are marked as in this user bucket.

Now you can even do fine-grained multi-variant experience, where you could potentially run multiple experiments on the same page. This is something I see potentially being really useful for the ecosystem, both in terms of people being able to build experimentation frameworks around it, or just quickly building page-based experiments and running those.

[00:26:09] JM: What's the process of deploying an Edge Handler?

[00:26:13] MBC: Yeah. This is one of the really cool pieces of it. Because of course, we've seen other edge platforms and so on. What we've heard over and over again is that if you have one team that operates the edge layer, often the networking team and so on. Then you have a different team that actually builds the web layer and deploys that, then you end up with completely different deployment processes for each of those two layers and often with different teams owning them that are not very good at communicating with each other.

If you're imagining building things like I just described the pricing page experiment and so on, you're starting to couple those two layers really tightly together. If those two teams doesn't work well together, or if the deployment mechanics are completely different and separate, that often leads to really costly roll out errors, or box, or communication errors and a very cumbersome development process, that in many cases, just makes people give up on the approach.

With Netlify, what's really essential is that all of this just lives in your git repository. You have a git repository with your normal build tool framework, whatever, that can output the front-end layer. You can have your typical folder with Netlify functions, cell deployed into AWS Lambda and you can have a folder called Edge Handlers that will deploy as Edge Handlers. If you spin up Netlify dev locally, we'll give you a dev server that includes all of this, so you can directly locally work with Edge Handlers with functions, with your normal single-page application framework, or site generator, or whatever.

Whenever you open a pull request in git, we'll build and bundle everything. We'll deploy it to a unique deploy preview URL, where you can see everything together in the full production setting, make sure it works. You can even make build plugins that can run integration test suites against the deployed results before it goes live, everything like that. As you merge something into master, we'll take it live and give you all the normal capabilities of Netlify, including immutable deployments with instant rollbacks and all of that.

[00:28:29] JM: Could you say more about the engineering differences between Netlify Edge Handlers and Netlify functions?

[00:28:36] MBC: Yeah. Netlify functions are implemented – our implementation runs on top of AWS Lambda, because they have the best cold start and are easy for us to work with. Functions have a full node environment, where you can use any NPM modules and so on. The current functions we have are always like request response. They have to return a full response you can't from a lambda function, you can't do streaming operations and so on, but you have a fully functioning modern node environment and you can do anything from a quick response to a response that would take a couple of seconds to run.

It's a great way to build microservices and API endpoints and talk to databases, talk to APIs, or services, glue together different services and so on. We also just now are introducing background functions, which is just like Netlify functions, but instead of being part of the request response cycle, you just trigger a function and then it runs in the background and can run for up to 15 minutes, so you can program processing tasks, or workflow steps, or anything like that.

Then Edge Handlers, again, the big advantage they have is that they're part of the full request response cycle of the front-end assets. They kick in before the actual front-end asset is served, then you can do request manipulation, you can do response transformation, you can proxy to other places. On the other hand, it's a very restricted environment. You should only do things that from a compute perspective can be done in milliseconds.

If you're doing anything that could take seconds or so, and then it's just not the right layer of the stack to do that in. Again, they're typically really useful for taking routing decisions, authentication decisions, personalization decisions, or for doing streaming response transformations, or even just from deciding on the fly to stream a response from a different origin, or service. All of those things are really powerful there, but they would not necessarily be a great place to build a full micro-service, or do server-side rendering, or things like that, because the environment is much more constrained.

Apart from that, I would also say that typically from just from an architecture perspective, whenever you're dealing with an API, or a database, or anything like that, typically, you will want the code that interacts with that to run close to the API of the database, rather than running close to the end-user, because you might have a couple of round trips. The end result will be much faster if the code runs where the data is, rather than where the user is.

On the other hand, if you're doing anything like modifying a pre-built page, or proxying anywhere, or doing API gateway functionality, or taking authentication decisions, that code you want to run as close to the end-user as possible for performance.

[00:31:46] JM: What are some of the current limitations of Edge Handlers that you'd like to work on?

[00:31:50] MBC: We are still in the early access stage of the product, really working with customers to figure out the developer experience around it and how to make it best to work with, how to look into their specific use cases and so on. We have to see to which degree over time, does it become interesting to have apart from just access to a cache? To which degree does it become interesting to have access to some distributed key value store, or is it better for us to figure out easy ways to use existing distributed databases, like phone or dynamo and so on?

It's more question of really doing the customer discovery and figuring out, when you're building real world projects with this technology, how can we make it as frictionless for developers to work with as possible?

[00:32:45] JM: Are developers adopting the Edge Handlers in the ways that you expect it, or has it been difficult to get people to try out new functionality?

[00:32:55] MBC: I would say, it's still too early for me to say exactly how that will play out. I'm just really excited to see what people do with it that I might not have thought about, because that's always the interesting parts when you launch these primitives to people to build upon, that there'll be a bunch of use cases that we know these will be really useful for and where it's – where people are very actively requesting access, because they are looking to solve specific problems that we know that this technology can give them solutions for.

We know those areas and we know that that there's real demand for solving those problems. What I'm curious to see is all the things that will emerge organically, that creative developers will come up with when you give them a new compute primitive that they didn't have before.
[00:33:49] JM: I'd like to talk a little bit about state management. I think oftentimes, these Edge Handlers are stateless, but there are times when you want state management. We did a show with Cloudflare a while ago about their Cloudflare key value store, a key value store that they push to the edge. Is there statefulness in these Edge Handlers, or are they completely stateless?

[00:34:12] MBC: There's a cache API that is a bit similar to a key value store, but it's not persistent. Whatever you put there should be used as a casual. If you need permanent state, you'll need somewhere else to talk to where you can fetch that from and put it in the cache and then use the cached object and so on.

We don't have a built-in data solutions and I'm still not sure if that's the right approach. Just because very often, you see very different applications requires very different types of contracts with their persistent layers. I'm not sure that it's necessarily a one size fits all, but

that's one of the learnings we'll go through as we go through the early access program and later through a public release.

[00:35:03] JM: With these edge handling caching layer, if I make a change to a cache at one end-point, does the cache have to get replicated to all the different other endpoints?

[00:35:14] MBC: Right now, the cache is end-point specific. If you store something in a specific location, it's specific to that location and we don't push it out to the other locations at this time.

[00:35:26] JM: Okay. Right. This is why you need some persistent store to push that data to. I could for example, write to the cache and then have the cache write to the database and have the database push eagerly out to all the other caches, right?

[00:35:41] MBC: You could do things like that. You could use something like FaunaDB, which is globally distributed and talks HTTP as your persistence layer and then use the individual edge nodes to make sure to cache for performance, but use Fauna as the source of truth, for example.

[00:35:58] JM: Bringing up FaunaDB, FaunaDB often gets mentioned in the same sentence as the Jamstack. I've always had trouble understanding, what is the connection between FaunaDB and the Jamstack?

[00:36:10] MBC: It's an interesting database to me, because it's built with a lot of the same capabilities that you often think of when you think of applications built with the Jamstack. It's fully managed, but not just in the sense that with some managed databases, like the typical experiences that you go in and then you say like, "I want you to run this database cluster for me with these replication setting and these instant sizes in this region." Then you can later go in and say, "I want you to scale them up, but I wanted to scale them down and so on." You're essentially still working at the abstraction of working with servers, where FaunaDB feels like an

actual serverless database. You just say, you just Fauna, “I want the database.” Then you start reading and writing data.

It's also globally distributed and using this Kelvin algorithm to find the right balance between consistency guarantees, while giving you a high-availability, globally distributed database, which is an interesting characteristics for parts of the stack, like its handlers and so on that are also globally distributed, like where you can then start having a data layer that's close to those Edge Handlers as well, rather than always having to go to a round trip to one specific data center in a central region.

Then it speaks HTTP. You can talk to it from any part of the stack that can send an HTTP request, where traditional databases are often very tied to the idea of opening a pool of connections and keeping them running, which tends to be a little trickier from layers, like serverless functions, or Edge Handlers, or anything like that.

It also means that you can technically speak to Fauna directly from the browser and it does have an authentication concept. It's obviously still early stage for that database and that experience. I’m sure there'll be a lot of interesting players in that space. I think, what's interesting with Fauna is really that it's one of the few – it's one of the only databases right now that really feels it has those characteristics of what you could call a serverless database.

[00:38:24] JM: Okay. It seems worth zooming out here and talking even more about the Jamstack. We last spoke a couple years ago. How has your perspective on the Jamstack changed since then?

[00:38:35] MBC: Our vision since the beginning has always been that the Jamstack is centered around the decoupling of the front-end web layer from the back-end layer and the back-end layer itself splitting into all these different APIs and services, where some are your own, but a lot of them are also other people's services. That we've just really seen playing out more and more in the ecosystem. I think just a few years ago, it was way more something that we had to go convince people that this was a viable approach, that it would be a thing, that you would

want to do it and so on. Now there's just way more pull from the whole ecosystem with big providers adopting it and lots of enterprises looking in this direction.

It's starting to feel more that it's a given that this is going to be a very big part of the future architecture of the web. Then are all these interesting parts of pieces of it that has grown stronger over time. When we launched Netlify functions, we were really the first to build with that idea of just a folder with your functions together with your front-end and they all deploy together and so on. Now we're starting to see that abstraction becoming more common in the space and more something that people are seeing, this is like a key piece of the architecture.

The emergence of serverless functions as an architectural piece have been interesting. I'm now thinking really about that whole compute layer of the Jamstack, going from Edge Handlers that can play an active role in the request response cycle, but best for things that runs in milliseconds, functions that can be used to building microservices, API endpoints, anything like that and can run for a couple of seconds. Then background functions that runs asynchronously and can be used for any task, where you just want to trigger it and say to the user, now it's running, and then give a response later, or send out e-mails, or do API requests, but without any direct interaction.

In that way, we for sure evolved more around the whole serverless compute space and how that plays into the overall Jamstack architecture. The other place that's become clear and clear to me is this interesting way where we also really moving from a world where traditionally, you really had your application with your database and now we're going more and more to a world where you have your application and then you have data, but in many different places.
If you use Netlify identity, or Auth0, or Octa, you'll have your user data there. It's a database, but it's specific to a user and it has both the user data and also, the special purpose logic to deal with users around login, sign-in, remember password, authentication rules, all of that. You might have a different – you might use Stripe for plans and subscriptions and now you essentially, have a different database with Stripe, but that's not a general purpose database and that also comes with a special purpose compute layer around subscriptions and upgrades and downgrades and pricing rules and all of that.

You end up maybe with a smaller general purpose data layer that's more of a glue layer between all these different data sources that you work with. That's been another interesting thing to really start seeing in practice and start thinking through what does it mean when you no longer have just one application and one database, but an application that talks to a lot of different places that has part of your data.

Of course, we're seeing services like History Cloud, or Apollo's Federation service, or take shapes, a content mesh layer, anything like that starting to also give solutions to how can you as a developer easily tie those different data sources together.

[00:42:41] JM: Well, that's a exciting place to close off. Thank you, Matt. Thanks for coming on the show.

[00:42:45] MBC: Thank you so much. Always a pleasure.

[END]