EPISODE 785

[INTRODUCTION]

[00:00:00] JM: Cloud computing started to become popular in 2006 with the release of Amazon EC2, a system for deploying applications to virtual machines sitting on remote data center infrastructure. With cloud computing, application developers no longer needed to purchase expensive server hardware. Creating an application for the internet became easier, cheaper and simpler. As the cloud has become popular, new ways of deploying applications have emerged. A developer with a web app today has so many different options. You can host your app on an Amazon EC2 server, which will require you to manage cloud infrastructure in case your server crashes. You can deploy your app to a platform as a service, like Heroku, which gives your cloud deployment better uptime guarantees for a higher price than Amazon EC2, or you can use Linode, or Microsoft Azure, or Google Cloud. There's such a large market for cloud computing that the world of cloud providers servers more niches every year.

In past episodes, we have explored a variety of different cloud providers and the markets that they target. Pivotal Cloud Foundry is for managing complex distributed systems applications, typically ones with large teams. Firebase is a cloud provider that simplifies the developer experience for applications with small teams. Spotinst is a cloud provider that emphasizes low-cost. ZEIT is a cloud provider that's built to manage applications through serverless functions as a service, like AWS Lambda.

In today's episode, we explore another niche platform as a service, infrastructure as a service hosting tool with Netlify. Mathias Biilmann Christensen is the CEO of Netlify and he joins the show. Netlify is a cloud provider that was built for modern web projects. Netlify represents the convergence of several trends in software development. You have static site deployment, serverless functions, the desire to have no ops development with minimal management, and the rise of newer tools, like GraphQL and Gatsby.

Mathias explores these trends in detail and explores the technical challenges of building Netlify. Mathias was a great quest. He was capable of talking about difficult backend problems that require writing C++ as well as the frontend world of JavaScript frameworks, and he learned musicology in college, so he is quite a diverse thinker and we went to a variety of interesting fringe areas in this conversation. So I really enjoyed it and I think you will as well.

By the way, I want to mention, my voice is not quite good recording this preamble, but it's much better I the interview. So when the interview turns on after this first ad break, my voice will be much better.

Thanks for listening.

[SPONSOR MESSAGE]

[00:03:08] JM: HPE OneView is a foundation for building a software-defined data center. HPE OneView integrates compute, storage and networking resources across your data center and leverages a unified API to enable IT to manage infrastructure as code. Deploy infrastructure faster. Simplify lifecycle maintenance for your servers. Give IT the ability to deliver infrastructure to developers as a service, like the public cloud.

Go to softwareengineeringdaily.com/HPE to learn about how HPE OneView can improve your infrastructure operations. HPE OneView has easy integrations with Terraform, Kubernetes, Docker and more than 30 other infrastructure management tools. HPE OneView was recently named as CRN's Enterprise Software Product of the Year. To learn more about how HPE OneView can help you simplify your hybrid operations, go to softwareengineering daily.com/HPE to learn more and support Software Engineering Daily.

Thanks to HPE for being a sponsor of Software Engineering Daily. We appreciate the support.

[INTERVIEW]

[00:04:30] JM: Matt Biilmann, you are the CEO of Netlify. Welcome to Software Engineering Daily.

[00:04:34] MBC: Thank you so much. Happy to be here.

[00:04:36] JM: Netlify represents a set of fundamental changes to trends in web development that are occurring. What are the fundamental changes to web development that Netlify is built around?

[00:04:51] MBC: Yeah, I think – I mean, when we started out building Netlify, we sort of did it because we saw a set of things changing the way especially frontend developers works and even changing what it meant to be a frontend developer. So 5 or 10 years ago, a frontend developer was typically someone who got a PhD from a designer and then sort of sliced it into HTML and CSS and then handed it over to a backend developer that would actually turn it into a real application by integrating it in some large monolithic system, like a workforce site or Rails application or Drupal app or anything like that, right?

We started seeing sort of a whole set of things happening. One sort of happened around through emergence of git and GitHub that really popularized not just the idea of version control, but this workflow around version control, this whole git-centric workflow where before that, especially I the world of frontend development, like all the frontend developers I worked with before that was doing version control with endless folders called like version four or final for real this time and so on, right?

Suddenly they started it up, this different kind of much more methodical software architecture of like committing and pull requests and so on and started getting the expectation of that sort of triggering the built workflows and the deployment workflows and so on. At the same time, the browser underwent this drastic revolution that probably really happened when the IE6 finally died and we suddenly had like Chrome and Firefox and Safari really innovating and the browser essentially turned from a document viewer into an operating system running JavaScript and today even WebAssembly. That marked like this whole change where suddenly you had like a client running in a browser talking to all these different services. Some were your services, but some were services like Stripe or discussed early on or any of these services that you could just like pull in to the browser.

Then the third thing we started seeing happening was probably triggered by sort of the emergence of Node.js, where these frontend developers are building up the scaling in JavaScript and so on also suddenly started like using it for all kinds of other things and started compiling stuff. Suddenly , frontend developers again like just underwent the massive transition from mainly just slicing PhDs into HTML and CSS to compiling complex applications with a real software application working through version control and publishing them to a browser that's now like really an operating system.

So we sort of saw those trends emerge and saw that that would mean – That would start meaning as shift in the architecture of the web where you would really go from like a traditional architecture of a large monolithic application with the sort of traditional [inaudible 00:07:41] architecture where you have a request coming in, it talks to a web server, that talks to an application server, that talks to a database, that sends back some data, you take some template, you build out an HTML and then you send it back to the client and you do that for every single request and you build a whole ecosystem around that.

So you have WordPress, where you have that flow inside that monolithic application. You have a WordPress plugin ecosystem where you get everything in the kitchen sink. You do it in Rails, with RubyGems, like where inside that flow you have all these libraries that you invoke and so on, right?

We moved for a long time to watch these more and more complex backend applications and suddenly I started seeing this potential shift happening to sort of a decoupled architecture, where the frontend presentation layer would get its own pipeline, its own workflow, its own set of developers and reality, and where you would take that frontend, prebuild as much of it as you could and then put it directly on a globally distributed network, because if it's just a frontend, like you just want it to be available for an end user to load from the browser with the lowest latency possible, right?

So you end up with an architecture where you decouple the frontend, you put it on a CDN and then that frontend talks no longer to one specific backend, but to all these different microservices where some tends to be your own and some tends to be other people's services, like I mentioned Stripe before, was like an early example that just completely changed like so many times before that. I've been part of implementing like complex payment gateways and so on. Suddenly just because it became a question of like, "Oh! Just make an API call to Stripe and you do the payments, right?" and more and more logic around that just moved to the frontend layer.

So that was sort of the overall trend, and with Netlify, we saw that happening and then we saw that that also kind of meant that the existing platforms that were built around these monolithic applications and around that way of working, they were no longer really tailored to this whole new world of frontend development. So we saw a big opportunity of really sort of drawing a circle around this git-centric workflow, this modern frontend build tools and this way of talking to microservices as serverless functions and then building this like really coherent and tightly integrated pipeline with its own CI/CD system connected to what we call an application delivery network, which is sort of like traditional content delivery networks were always to sit in front of something else.

If you look at any of the CDNs, the idea was that you put it in front of something, or maybe you put like very specific static assets on them, but typically they sit in front of an origin, right? We built sort of our own application delivery network that's meant to replace the origin, like to just get away from this idea of having a webserver that all your requests goes through and just having this frontend directly distributed on a network.

So our key first insight was just that if we could make that way of working like really simple and really intuitive, then this new architecture that we started to see emerging from really adaptors and things like the work that they did around the Obama campaign with like superstar team but building in this way and showing how it could scale and so on, we could sort of take that tendency and that architecture and make it viable for pretty much every frontend developer out there.

[00:11:16] JM: I acknowledge all those trends that you mentioned, and you didn't even discuss tools like Gatsby. There's also changes to middleware, GraphQL.

[00:11:28] MBC: Absolutely.

[00:11:28] JM: You've got the backend serving layer, AWS Lambda changing. So there're all these changes. So before we get into Netlify in detail, you've been building businesses for a while. You are deeply into the web frontend, or it's not even fair to call it a frontend anymore, but the changing world. There are a lot of people who feel this constant sense of change and they're afraid that they're not building the right skills or they're missing out on building web assembly. How should the modern developer react to all of these – I just mentioned another dramatic changing tool. How should the modern developer react to all these change? What they should focus on?

[00:12:15] MBC: Always, as a developer we, in general – Development is still such a relatively new field. Sometimes I use to think about it like as this state where surgery we're at in like the start of the 19th century or something, right? Like we have some 50 years of doing it behind us and [inaudible 00:12:32] the same time, like a lot of the time the patient dies when we try doing things. It's reasonable to expect that things are still out of – Like we're still only like 50 years into all of these stuff, right? Like 60 years or something, and everything is still going to change a lot.

So as developers, we have to – I mean, if you get into a development and you really don't like learning new things, you'll have to find a very specific niche, like there are still banks out there using cobalt and all of the ways people build systems today will still be alive like 50 years from now in some weird niche, but everything will keep changing a lot. A lot of us are trying to do and what we're trying to do with Netlify is to say like, "Okay. For example, if we look at frontend development, that's exploded in complexity compared to 10 years ago."

In sort of the early days of frontend development, it was really document-centric. It was relatively few set of standards. You could learn HTML and CSS and that was it, right? Today, we really have real software architectures, like Redux introduced sort of the whole light year of functional programming view of the world together with the React paradigm and so on and we started having like ideas of concepts like [inaudible 00:13:53] and so on that are more complex state management solutions and so on that's emerged.

Of course that means that the whole area has more complexity also, because you can do much more, which is the other side of it. Today, you can, as a beginner developer, you can grab a Gatsby starter template and in a few minutes you'll have like a globally distributed site that can scale through billions of visits with incredible performance and with a platform that you can quickly iterate on and drive on.

So the complexity is coming for a reason. It's coming because we're actually empowered to do more things, but it also means that companies like our or companies like Gatsby or companies that are innovated in this field has a job to take the complexity from somewhere else and reduce it. So in our case, we are trying to take all of the complexity around the operations and the pipelines and the tooling and the infrastructure away from developers so they can focus on learning the complexity of the emerging fields of building web presentation layers and building dynamic applications in the browser.

[00:15:10] JM: So there's clearly a big vision for Netlify, but let's talk about the product surface area as it stands today. So the product is mostly knows as a static site hosting tool. Describe what static site hosting is and how that contrasts with more fully involved serverfull deployments like AWS EC2.

[00:15:35] MBC: We at an early point, very early on, we were talking a lot about static and we found that it tended to confuse people a lot, because people associate static with like a brochure or something like that, and the things people are building with Netlify – If you go to app.netlify.com, that's a Netlify application, like running on Netlify. Obviously, this is not about static in the sense of non-moving patch.

The main characteristic here is an architectural decision of saying, "Let's decouple the frontend from the backend and let's not mix the two together," and that we've talked about that as the gem stack approach, like the JavaScript API and markup where you say, "Okay. We ship, we prebuild whatever markup we can. We ship it directly to a content delivery network. We use JavaScript as the main runtime in the browser and then we talk from the browser to all these different APIs and microservices."

From the beginning, what we've sort of said is that we didn't want to care about the tools. People should be free to innovate in the space of Gatsby, or Hugo, or Gridsome, or create React app, or [inaudible 00:16:43], or anything like that. The tools should be free for the developers to pick, but our idea was that if we were sort of religious around the architecture and say, "If you use this decoupled architecture, splitting the backend from the frontend, then we can take all the operational best practices, all of the CI/CD best practices, the whole infrastructure setup and automate it completely and just say, “This is just going to be best in class. So it’s going to be distributed across all the different cloud providers. It’s going to be this high-performance, this high-uptime as you’ll get. It’s going to be tightly integrated into your git workflow and you just don’t have to worry about it.”

But that really comes from being more prescriptive in that area than AWS or Google compute or anything like that that’s for building anything in anyway. So they can’t come and tell you like, “This is the architecture and our power,” and of course, that’s the limitation is to say like, “Okay. We picked this architecture. If you’re building with this architecture, then we’re just going to give you the best possible setup you can get without any extra work.”

Then that architecture really goes beyond of course just the static asset. It’s more this idea of having an application delivery network for your frontend that’s no longer [inaudible 00:18:01] sitting in front of your origin, but that’s a replacement for your origin. That includes of course like the pipeline for publishing your static application or your static files for your Gatsby side, or your Hugo side, or your React application, but it also includes the whole routing layer for talking to the different microservices. So we have a very flexible rules engine that’s controlled with a little underscore [inaudible 00:18:26] file in your repository or with a Netlify [inaudible 00:18:30] file in your repository and that goes through all the same CI/CD pipeline workflows that works with the deploy requests and so on.

But it can drive things like – In a really basic case saying like anything under your API should go to this API. But in the complicate cases, can pull into things like our serverless functions integrations where you can simply have a folder with your Lambda functions. We’ll deploy them together with your frontend and then we’ll completely sidestep the whole API gateways side of things and so on and just use our globally distributed routing engine to invoke those functions for you and allow you to tie in the routing to those functions to our [inaudible 00:19:12] engine and so on. We have our identity service, for example, that was another sort of step of just seeing like what are these patterns that emerge when you’re working with this architecture.

When you have all these different microservice and you have a frontend, one of them is the idea of stateless authentication where Auth0 pioneered sort of the idea of JSON web tokens, right?

As a solution to that problem of how do we solve when we have all these uncoordinated services that still each of them needs to know who are the user, right?

So we build that into this application delivery network layer where you can set up rules saying like, "If you have someone that's locked in with a JSON web token that we can verify and we see that that user has a role called admin, then show this content instead of that content, or allow routing to this API instead of blocking that API endpoint and so on," and then we launch our own little identity service that can issue JSON web tokens, but as a pluggable service where any service, whether it's Auth0 that can issue JSON web tokens could be used instead. But all build around this architecture, you have like what are the best practices. How should the tooling look like if you take this architecture and build with that?

[SPONSOR MESSAGE]

[00:20:41] JM: Today's sponsor is Datadog, a cloud monitoring platform for dynamic infrastructure, distributed tracing and logging. The latest AWS X-ray integration from Datadog allows you to visualize request as they travel across your serverless architecture, and you can use Datadog's Lambda layer to collect business critical metrics like customer logins or purchases.

To see how Datadog can help you get a handle on your applications, your infrastructure and your serverless functions, sign up for a free trial today and get a free t-shirt. Visit softwareengineeringdaily.com/datadog to get started and get that free t-shirt. That's softwareengineeringdaily.com/datadog.

Thanks to Datadog for being a continued sponsor.

[INTERVIEW CONTINUED]

[00:21:39] JM: One way to categorize Netlify is as a platform as a service, and we have this lineage of platforms as a service. We have Cloud Foundry, Google App Engine, Firebase, Heroku. How would you differentiate your philosophy from the previous platform as a service companies?

[00:22:01] MBC: Yeah. I think the one that kind of came closest to what we're doing, early on we were sometimes talking about Netlify as a Heroku for a new stack, and I think early on what we saw from sort of a business perspective with a lot of the platform as a service with especially ones like Firebase and [inaudible 00:22:20] that got very strong developer adaption but failed to build like really big businesses around it, was that when they were very prescriptive on the tooling layer, especially around like saying you use it with our database, proprietary database, our managed database. That tended to put a ceiling in where they gave developers very great hello world experience and they became incredibly loved jewels for like prototyping and quick iteration, but then once you were building things in like an enterprise settings, there was just no way that like the existing infrastructure, security team, like database team, any of those would be like, "Yeah, let's just put all our company's data in that provider."

[00:23:08] JM: Is that because of cost, or what exactly?

[00:23:10] MBC: I think it's more – Almost regardless of cost, it's more about corporate structures and policies and which teams are you selling into. I think it's hard to get a frontend team to go to a core platform database team and say, "Hey, we just put all our stuff in this other thing."

[00:23:27] JM: Then why doesn't an AWS database have that problem?

[00:23:30] MBC: So I think once you have like a certain scale, like AWS and Google and so on, you can start doing it, and they did have that problem in the beginning. It took a lot of convincing for enterprises to move any data out of the on-prem data centers and into partners like AWS and Google. So if you're trying to do that as a very small company, you have a far longer uphill battle.

What we saw was that on the other hand, of course, when you talk on-prem, no one will go to Akamai, "Hey, can we have Akamai on-prem?" because the whole idea is the distribution. You want your content when you put it on Akamai to live in as many places around the world as possible, right?

[00:24:12] JM: And you're not really locked-in.

[00:24:13] MBC: And you're not really locked-in. So we saw the same early on, again, with being un-prescriptive around the tooling and just taking the architecture. We could see this whole layer that would scale in the same way where you would want the best practices, whether you're like a lone developer putting up your personal block, or whether you're teletron.com building all of your web properties with Netlify.

We saw that there was like this area of the presentation layer and the frontend layer where you want the best possible global distribution, where there's a set of best practices for the whole CI/CD pipeline that we can really optimize and so on, and that we initially saw this can scale all the way from a great hello world experience for individual developers, but also up to real corporate enterprise use cases. That was one of the – It's still one of the reasons today that contrary to what Firebase did and what Parse did and what many others did, we haven't launched like our own database product. In that way, we tend to just work with providers, like FaunaDB or with people putting their own APIs inside AWS of things like that, because we think there'll be so much more of a hurdle to take over the – That's where companies have really hardcore specialized operations teams and performance tuning teams and so on around those data layers. It's harder to just say, "Here's a set of best practices that will work for anybody." It's more typical. When you reach a scaling point that's sort of the opposite, like as your data complexity really grows and so on and you start moving into more and more specialized data solutions and you have more and more people working on tweaking indexes and analyzing query paths and so on. That's not really where we want to be at.

So in contrast to those early generations of platforms as a service, I hope we've been able to strike a bit of balance of what can we actually provide as a service that will scale all the way from a developer to an enterprise and then leave out the parts of the platforms where we think these parts people will have to build teams around and will have to build expertise around and will have to scale into the [inaudible 00:26:27].

[00:26:27] JM: What you said about lock-in and propriety special systems that are data-intensive, or API specific, or proprietary in some way or another, you did mention you have an identity platform. You also have like a function as a service platform. Do those have a sense of proprietary lock-in to them?

[00:26:51] MBC: So, I mean, anytime you use a service, you'll tie yourself in some ways to the service. We've been very explicit with our Netlify functions that we are simply exposing AWS Lambda and we're taking away all the complexity of working with it, but it's not like our own function runtime or anything like that. It's very clearly Lambda. We're sort of betting and starting to see this verify that Lambda is such a winner in the functions as a service-based that whatever they do will kind of become the standard, and we're starting to see people building Lambda compatible –

[00:27:24] JM: Why is that? What makes Lambda so special? Why isn't that the same as Google Cloud Functions and Azure functions?

[00:27:29] MBC: I think the core thing is that they've done such a great job with a cold start, where any other services is still behind in that aspect.

[00:27:38] JM: So it's literally latency.

[00:27:39] MBC: I think that latency is such a big part of why Lambda has like initially rocked the boat so much, that the other sort of has still haven't caught up in that aspect. Once you have – Unless people use it to tie very deeply into other services, then there is like this [inaudible 00:27:56] aspect of the function runtime, where whoever runs it fastest and boots it up faster, it's where I'm going to put it right.

So as long as AWS has a headstart there and is doing a better job there, then there's much higher hurdle for Google or Azure to convince people to integrate there. Of course on our end it's always something we're looking into, like adding Google functions integrations and Azure functions integrations, and we are interested in it and so on. But I still think that right at this moment, Lambda is to such a big degree sitting the pace for the functions as a service base. I think it's likely to lead to things like we've seen like people building on top of things like KNative compatible layers with Lambda that are sort of starting to work and so on, and that I think really strong from an idea of like let's break the lock-in and –

[00:28:52] JM: Who's doing that? I haven't seen that?

[00:28:53] MBC: I've seen some open source projects around that. I can't remember the names off the bat of my head. It's like –

[00:28:59] JM: Well, that's cool. I mean, anyway, it's cool. I mean, we just did a show on KNative. For people who don't know, that's like Google built an open source serverless on top of Kubernetes framework, and what you're saying is that the first thing somebody is doing is building AWS Lambda on top of it.

[00:29:14] MBC: Precisely. It might not be the best abstraction, because right now KNative feels to me still more [inaudible 00:29:20] towards running longer running sort of HTTP services and so on.

[00:29:24] JM: Well, they're trying to do both.

[00:29:25] MBC: And they're trying to do both, but it feels to me having played around with it a bit that that's still like the more of a sweet spot. I mean, I think the KNative strategy is really fascinating, because the Kubernetes strategy from Google has been so powerful. Personally, I really am a big believer in having open shared standards and as little vendor lock-in as possible. So I am really hoping for that sort of – For AWS Lambda to be not a monopoly, but being just like something that gets copied widely enough that we can work with all kinds of different provider and I really hope there'll be robust, solid, deployable on-prem solutions as well for this. I'm pretty we'll see that happening overtime. I just think that AWS really pioneered the field and have really been setting the pace for serverless functions with Lambda.

Again, longwinded way of saying like we try to not be like a very proprietary platform, but simply saying like, "Okay. AWS Lambda is obviously becoming the standard here. Let's just make it completely frictionless to work with and make sure you don't have to worry about like how does my frontend know where my Lambdas live. Where is the CI/CD pipeline for the Lambda? How do I need to configure API gateways for? What are my endpoints and so on and just like, "We should just write the code and the rest should work as you would expect."

The same with our identity service again, right? The idea was to really look at the standards, right? Say, "Okay. The actual trend is just start using stateless authentication through JSON web token." We can make an open source service called – We made like an open source microservice called Go True that can handle like user registration sign ups and so on and issue JSON web tokens. Then we're just running a managed version of that, but all the code behind this is open source and the standard it's build in is an open standard, which means that as I mentioned, you can swap out our identity service quite easily for Auth0 opt to any other provider.

[00:31:27] JM: We're circling through all these areas, and there's a distinction we could draw between the "backend" area of all of these stuff where you've got serverless and Kubernetes and Istio and all of these backend development going on. Then the frontend open source stuff with like React and the serving layer and GraphQL and Gatsby and all these stuff, but from the point of view of being CEO and looking at both all the trends that are going on and the fact that you have to architect your own platform, you want it to run economically. So maybe you're running your own Kubernetes stuff. Do you pay attention to all of these different areas or do you try to focus on just the specific frontend layer?

[00:32:13] MBC: I've always had the sort of special characteristic of being kind of equally interested in all of those areas and I like having the sense of understanding how things work throughout the stack. So in what I've been building, yeah, I've been switching from like – I've wrote the first C++ plugins for Apache traffic server to power our actual CDN network, but I also wrote the first CSS for our application UI and set up the initial React infrastructure and so on.

Now, of course, like I'm only writing codes sort of at the fringes of what we do always making sure that I'm not part of like a critical path for any feature to go out and so on, but more exploring on what's next for Netlify. I think what the fascinating thing of building Netlify is that, of course, our end users are the frontend developers and we have to build a very strong empathy to web developers and what are their changes and what complexity can we reduce and what tools can we give them that makes them more powerful? But we typically have to do it by building very complex infrastructure products. So we have to build on top of like our own globally distributed network of [inaudible 00:33:31] nodes and our own Kubernetes clusters to run builds and manage microservices and we have to handle all their operations and devops and so on.

So I think for me personally I like putting that kind of like very nitty-gritty deep infrastructure work in the service of a very enjoyable developer experience and trying to build, like figure out always how we can build the connection between these complex internal moving parts to frontend developers that generally have enough with the growing complexity of the frontend development universe and like to dig in there. Personally, of course, you always have some tradeoffs. My tradeoff might be to not go deepest into any of the areas, but understanding sort of all the different parts of the stack.

[00:34:28] JM: Do you run Kubernetes clusters in multiple cloud providers?

[00:34:34] MBC: Yeah, we do, because we currently only run Kubernetes in two cloud providers, and that's for our actual origin lawyer. When we sell to developers, of course, it's all about the ease of use and the enjoyment of the development experience and like how fast they can work about the productivity. When we sell to enterprises, it's still to a large degree about like 10X-ing your developer's productivity, but it's also about performance and redundancy and uptime and scalability and so on.

One of the things we've build is that even our origin servers are multi-cloud between two cloud providers, where if all of Google compute went down tomorrow, we would simply trigger our failover switch and we would start serving out of an AWS data center instead, which has its complexity, right? Which is another part of this story I was telling earlier, that if we can really say, "Okay. This is the architecture," then we can do sort of all the massively hard infrastructure work to just make sure that as a client, you get all the best practices out of the box and all of the redundancy and all of the multi-cloud capabilities that otherwise would probably increase your own internal development time with like 4X or something like that at least.

[00:35:51] JM: I want to understand how your perspective in business has evolved with Netlify, because you've been building some businesses for a while. You have an open source – I'm sorry, you have a set of things that you've built in the past. You actually built a CMS in the past.

You also built a previous hosting platform. I think it was – What? It was called Push Bubble Pop or something? Bubble –

[00:36:14] MBC: Web Pop was like the CMS – Like a cloud-hosted CMS platform.

[00:36:20] JM: I had it written down. I couldn't remember. So Netlify is doing tremendously well. I presume those other businesses had some customers, but Netlify is your best business so far. How has your perspective on business and pricing and go-to-market strategy changed with Netlify?

[00:36:38] MBC: One of the fascinating things of just doing this whole journey is how much you have to learn all the time and how much your learning evolve. I think if I look back at the version of me that built my first startup and so on, like I would probably find that I was very naïve in terms of businesses compared to what I've learned since then.

So the feeling you get when you hit the business that where you really see the product getting like the kind of product market fit is very enjoyable and it can take very, very longtime to get there of building and iterating and talking to users and being wrong about what you've thought they wanted and figure out the space and so on. There has to be a very – You probably generally have to be really, really stubborn to just keep going and keep trying and keep figuring out the right thing, and it takes a bit from figuring out the right thing to figuring out that people also want it and so on. But once you start seeing it, you also see a very big sort of difference in how people are receiving your product and so on.

Some of the things that most product people and engineers need to learn early on is just like this whole dictum that a product is not a company. One thing is having a product, but another thing is actually building how are people adapting the product. How are people figuring out about it? How are you going to – Not even selling and monetizing it, but how are you going to build a whole company around what you're doing. It's very different from building a product in itself. I think for a lot of product people, that takes a lot of learning that there'll be a tendency for engineers to just focus on like –

[00:38:22] JM: Is that something you failed at in previous businesses? You were not able to learn about people?

[00:38:26] MBC: I wouldn't say about people also, because my background is sort of weird in that way. I studied humanities and studied culture and people first of all, and for many, many years, programming was just a hobby for me. So I think it wasn't too much about learning people as learning business in a certain way, and learning about how to integrate the moment of taking a product to a market into the product itself and figuring out those aspects. That takes a lot of work.

Also, just learning, like no matter how much we think we're doing it, learning how to find the shortest path to get something valuable into the hands of some users, and then its rating from there has been really important. When I think about like product development in this part now, I always think that the way we approach it now is like figuring out a really, really big ambitious vision. Then sort of architecting down what's the smallest step to watch that vision we could take that we can get into the hands of users that would be a step towards that vision, just a small one. If people don't care about that little step, then we might have to revise the whole vision in some way, like there should be this connection of every part of it and you should be very honest with yourself about like if you have that vision and you put up these steps and people don't care, well then maybe you have to go back and figure out, "Why don't they care if this vision is so good?"

[00:39:56] JM: You studied musicology. What's the difference between being a composer and being a tech CEO?

[00:40:04] MBC: I would say the fun thing is that classical composers were in some ways some of the first programmers. They actually had to write down these programs that a whole orchestra had to execute and follow [inaudible 00:40:18].

[00:40:18] JM: It's like a punch card level pain.

[00:40:20] MBC: It's like punch card level programming, right? It's kind of fascinating that I do think there are some overlap in learning that way of thinking of abstract structures overtime and of thinking how well they actually interact with people when they're executed that translates well to the programming thing.

From the business side, I think at least like musicology also teach you things like getting up and conducting a choir and things like that that at the time I wasn't super aware of like –

[00:40:49] JM: Did you do that? Did you conduct a choir?

[00:40:51] MBC: I totally had to. It's part of my education. I had to get up there and conduct a choir. We even got examinated in conducting choirs or leading bands, like stuff like that was part of musicology as well, right? Those kinds of things of course are useful skills to have when you have to get a group of people to sort of follow the same motion and go in the same direction and believe in the same thing. Apart from that, everything is different.

[00:41:22] JM: Well, I wonder if it's more similar if you actually become a conductor and you are managing those musicians.

[00:41:31] MBC: Oh, yeah. I think that's another thing that I've always been extremely fascinating by by seeing some of the best conductors in the world and seeing how – Because I mean that's really – A conductor is to a large degree a management role, but it's a very –

[00:41:44] JM: Real-time.

[00:41:45] MBC: Yeah, yeah. It's real-time, but it's also the rehearsals. It's the building in and they're selling a vision. Because if you have a conductor that goes up – Like imagine you're a conductor and you go in front of like the Berliner Philharmoniker, you have like a group of hundreds of the most skilled, most like respected musicians in the whole world sitting there and you're taking a piece of music that all of those musicians knows completely by heart and can play it hundreds of times. Now you have to come as a conductor and convince all of those hundreds of people at the same time that your interpretation of this piece of music is like the one they should really put their heart into.

Seeing the very best conductors when they can do that, that's so incredibly fascinating, because it's like it almost seems impossible.

[00:42:38] JM: Not only that, but you see them managing the emotions of the instrumentalists in real-time. They're like making eye contact with that person that they know messed up a note and they're like giving sympathy while managing all 99 other musicians.

[00:42:51] MBC: Precisely. Then they also – Like the best of them are also extremely good at giving space. They also know that, "Okay. Now there's this solo and there's this incredible musician and I can try to make him play this solo my way completely," but that will probably not be the best expression. I can try to instead instill a vision of how that solo should fit into this whole orchestral piece and then give that musician the right space to lift it and interpret it and follow it.

So I could talk a lot about that. One of the really fun experiences I had was seeing a master class with a conductor called Kurt Sanderling, an old conductor who lived in East Germany at the time and was like one of these fantastic old maestros, like really schooled and old tradition and so on. He was doing a master class with like 8 young conductors with the [inaudible 00:43:48] Radio Symphony Orchestra instructing them.

I remember this one young Italian conductor going up conducting part of Tchaikovsky's 5th symphony for the orchestra and he put in so much moment and he was like really pushing and moving his arms and [inaudible 00:44:06] and trying to get him to follow his idea and so on. In the end, Sanderling said, "This passage, you don't need to do that much," and so on.

He goes up and he stands in front of this old man, in front of the orchestra, the young conductor [inaudible 00:44:22] and said, "Go again." Then he takes like one movement, but he just like – He instills like this instantly as he starts it, right? You can just see the attitude of every single musician in the orchestra change from being there, being at work, like following this conductor [inaudible 00:44:39] to suddenly like everyone's vision just changes. Everyone's gaze changes a little and he just makes like this slow movement with his hand like gearing through and it's just a question of his posture and how he represents that movement in one single gesture, and the orchestra sounded completely different. Everyone just came together. The whole thing was different. You could see that young guy standing behind and sort of shaking his head an laughing a little, because like, “How the hell am I ever going to do that?”

But it was like one of these magic moments of what it means when someone can instill not just try to micromanage [inaudible 00:45:22] of people to do this, “This is what I want you guys to do all the time.” But when someone can get people to believe in something and instill a vision, how it can change what a group of people can do together.

[SPONSOR MESSAGE]

[00:45:43] JM: This podcast is brought to you by wix.com. Build your website quickly with Wix. Wix code unites design features with advanced code capabilities, so you can build data-driven websites and professional web apps very quickly. You can store and manage unlimited data, you can create hundreds of dynamic pages, you can add repeating layouts, make custom forms, call external APIs and take full control of your sites functionality using Wix Code APIs and your own JavaScript. You don't need HTML or CSS.

With Wix codes, built-in database and IDE, you've got one click deployment that instantly updates all the content on your site and everything is SEO friendly. What about security and hosting and maintenance? Wix has you covered, so you can spend more time focusing on yourself and your clients.

If you're not a developer, it's not a problem. There's plenty that you can do without writing a lot of code, although of course if you are a developer, then you can do much more. You can explore all the resources on the Wix Code’s site to learn more about web development wherever you are in your developer career. You can discover video tutorials, articles, code snippets, API references and a lively forum where you can get advanced tips from Wix Code experts.

Check it out for yourself at wicks.com/sed. That's wix.com/sed. You can get 10% off your premium plan while developing a website quickly for the web. To get that 10% off the premium plan and support Software Engineering Daily, go to wix.com/sed and see what you can do with Wix Code today.

[INTERVIEW CONTINUED]

[00:47:42] JM: Now, when you think about yourself as an entrepreneur or an artist, you want – To some degree, I mean it depends on your maximization function, but you want to maximize the impact you're having. If you're somebody like Hans Zimmer, you're having massive impact on millions of people. They're watching Interstellar, they're listening to the music. They're watching Inception, they're listening to the music. It's sitting with them. It's helping them process emotions. They're going through advances in their lives because of his music. So you have the option to do that, or you have the option to build something like Netlify, enable developers. You have to make some deliberate tradeoff why did you wind up making the deliberate tradeoff being a tech CEO.

[00:48:27] MBC: So much of it is – There's always an element of random change in all of these things. But I had always like this – Since I first encountered a computer, like I had this joy of the basic idea that you can write something that in a way becomes an interactive universe. You can write a little code people can interact with. It could do stuff. So it's like what's this life behind the screen once you started writing code.

Then during the things I happen to be building in my career, I sort of stumbled into this area of building tools for developers, and frontend developers surely and designers and so on. I think there's an amazing feeling in building the tools that other people build things, because you get to see what people are building with your tools. It's extremely fascinating once you start seeing some other developers being able to build more than they otherwise would have built because you built something.

I think that's the same feeling that drives people to write amazing open source libraries and that dries that whole open source experience of sharing code, right? It gives you such joy when you build something and you see that it's somehow works as a multiplier for what other people can build. That's still like one of the great experiences with Netlify from all the levels, from seeing like big enterprises, like big projects and drawing lots of terabytes out of our system and pushing like tons of requests, but also seeing like we had this – We organize this jam stack hackathon together with Free Code Camp, and I love the whole – Like everything that free code camp does.. It's amazing.

[00:50:12] JM: Same here. Quincy is a long-time friend.

[00:50:14] MBC: Yeah, Quincy is amazing. It was so fun at that hackathon to see like this group that's more a beginner audience, of course. But seeing what people could build in a weekend with this kind of tooling from a relatively fresh starting point in their careers and so on, that was super inspiring and really enjoyable.

[00:50:35] JM: Better compound annual growth rate perhaps than music. Well, maybe. We'll see how many babies come from Hans Zimmer's music.

[00:50:43] MBC: Yeah.

[00:50:43] JM: So back to business, or Netlify. I want to know the hardest problem you've had to solve, whether it's an engineering problem, specific engineering problem, charting the future, management, fundraising, what is the hardest problem that you've had to solve building this business?

[00:50:59] MBC: That's a tough question, because like every step of building –

[00:51:02] JM: Okay. Then let's focus on engineering. Let's scope it, just engineering problems.

[00:51:07] MBC: Engineering problems at Netlify is a very big distributed platform, like running across. Currently, I think we're using 7 different cloud providers with like data centers all over the world. Of course, like the core piece of nailing the architecture of how do we build an architecture that can scale from what I can build as a single developer into what a huge team can work out is probably also one of the problems I've started to most proud of getting very right. Our architecture today is fundamentally the same architecture as when I built the initial version of Netlify just scaled by many, many times. Solving that core problem was a lot of work and was hard.

Again, as I said, probably continually the most challenging problem we have to solve over and over again is like how do we build really nitty-gritty infrastructure solutions with a large distributed system as a means to make the DX experience of developers working on the web really enjoyable. That's like really hard, because it goes all the way from understanding the concepts of distributed systems and infrastructure and optimizations and so on to understanding the empathy with developers.

To tying together, figuring out how can we make that connection between like design teams that thinks in and one way and platform teams that tends to think in a very different way and API teams and so on and get them together build a platform that's really oriented to what the end user's base in understanding what we can do from an infrastructure perspective. That's probably the single sort of like greatest engineering challenge in building Netlify.

[00:52:59] JM: So synchronizing your distributed data sources and then synchronizing the development teams that are working on those distributed data sources.

[00:53:09] MBC: Yeah, and then scale. Obviously, just constantly being one step ahead on the kind of – I mean, we've had to scale our system massively and –

[00:53:19] JM: Why isn't that a solved problem? Why don't auto-scaling groups and auto-scaling Kubernetes clusters and all that kind of stuff –

[00:53:25] MBC: Because it's all of the platform. It's your log monitoring system. It's like suddenly you start throwing 13 billion requests of locks-in to some system and that's not behaving well, and like each of the step like this, a new system that starts breaking and you're like, "We have to reinvent that part of it."

It's because when you have these massively distributed systems, there's so many different components and each of these components need to scale. I mean, by now, we're reaching something like 100 million unique users every month from our application delivery network, and that's gone from like being two people bootstrapping in March 2015. So that kind of infrastructure scaling problem is interesting.

[00:54:14] JM: You've obviously shown a capability of building stuff. You've also shown the capability of buying stuff, because you're built on top of cloud providers.

[00:54:23] MBC: Yeah, absolutely.

[00:54:24] JM: What is a mistake you've made in a build versus buy decision?

[00:54:29] MBC: It's always hard to say. I think in general, I like the buy whenever possible, and I'm very aligned with our CTO in that aspect. We always – If we can void building something, we should. Anytime we can avoid building something, we can build something else instead that apparently doesn't exist. So that seems to add more value to the world than building something that already exists.

Then there are places that goes back and forth. Right now we're using a third-party provider for the lock streaming of our built locks to end users, and that's something that we are looking to have to rebuild internally, because we're running into issues there and we get scalability issues in that area that the provider doesn't seem to be really get to and where we now can't do anything about them, because they're not ours. So that's one area where we, on the one hand, you could say you took [inaudible 00:55:28]. On the other hand, I would have to go back and say, "If we had built that in ourselves, what should we have built it instead of?"

All of these decisions are typically really tricky tradeoffs in that way that every part of your system you'll have parts where you think, "I wish we had done this in a slightly different way, or I wish we have maybe had time to build that in-house," but a lot of these decisions are also reflections on like what did we choose to do instead at that time.

Initially, we had some failures in building. Initially, we build our own log processing and log aggregation system and then just quickly found out that scaling that system at the same time as scaling the rest of our system was massively painful, because the number of locks we ingested just kept growing exponentially. So that was one point where we initially spent quite a while building it ourselves and then ended up going and buying it to not being like, "Let's just not deal with that." The same happened with metrics. Today, we're using Datadog for our alerts and monitoring and system level metrics. That's another area where that was totally worth it versus when we were for a long time trying to run our own platform.

It's the platform that needs to tell you if you are up and down and if you both have to manage the opt time and scalability of the platform that tells you are up and down and the platform that you are trying to verify whether it's up or down. That's like [inaudible 00:57:02] the work so that then we felt that even if the build is high, it was worth it for us to [inaudible 00:57:08].

[00:57:09] JM: We won't need to run a Datadog ad on this episode. Last question, I know you're out of time. This is my favorite question for musician engineers. So we have this world where you have thousands of engineers that contribute to open source projects, and yet in the world of music, most songs are written by one, two, three, four people, maybe. Why don't we have better music collaboration? Why don't we have scaled music collaboration?

[00:57:39] MBC: It depends how you look at it, because maybe most songs are written by one, two, three, a few people, right? But none of those songs stands like in a vacuum. Most compositions and most songs and most music is very much based on other people's music, right? There's always these subtle lineages of inspiration and of like riffing like most song composers whenever will learn that trait by learning and studying other people's music and by reading the music, by playing the music and adapting it a bit and finding their own voice based on that.

I think in that way it's actually quite collaborative by nature and it's always sort of been – Even if the source code is not available, it's not so hard to [inaudible 00:58:24] music and get the source code yourself and people will do that. Like in the jazz world, you always had like all of these classic, the real standard books with like the chord progressions and scores, like all of the jazz standards that all of the jazz musicians would like get and learn and figure out and make their own variations of and so on right?

In the classical world, all the composers which study each other's scores and go through those and so on. I think in a way, music has always had that kind of like implicitly collaborative way of like learning from each other and understanding each other's code. I think open source has some of the same effect on software that all of us learn a lot by simply starting by looking at other people's code and understanding it and using their libraries and then start adapting them and start contributing to them and so on, right? In reality, I don't feel those two movements are so different.

[00:59:23] JM: Okay, Matt. Thanks for coming on the show. It's been great talking.

[00:59:25] MBC: Yeah, great talking.

[END OF INTERVIEW]

[00:59:30] JM: GoCD is a continuous delivery tool created by ThoughtWorks. It's open source and free to use, and GoCD has all the features you need for continuous delivery. Model your deployment pipelines without installing any plug-ins. Use the value stream map to visualize your end-to-end workflow, and if you use Kubernetes, GoCD is a natural fit to add continuous delivery to your project.

With GoCD running on Kubernetes, you define your build workflow and let GoCD provision and scale your infrastructure on-the-fly. GoCD agents use Kubernetes to scale as needed. Check out gocd.org/sedaily and learn about how you can get started. GoCD was built with the learnings of the ThoughtWorks engineering team who have talked about building the product in previous episodes of Software Engineering Daily, and it's great to see the continued progress on GoCD with the new Kubernetes integrations. You can check it out for yourself at gocd.org/sedaily.

Thank you so much to ThoughtWorks for being a longtime sponsor of Software Engineering Daily. We are proud to have ThoughtWorks and GoCD as sponsors of the show.

[END]