EPISODE 700

[INTRODUCTION]

[00:00:00] JM: Engineers can build applications faster by using tools that abstract away infrastructure. Major cloud providers offer this tooling in the form of functions as a service as well as managed services, such as Google BigQuery or Azure Container instances. The term serverless refers to these two types of serverless ideas. You have functions as a service, which are very flexible ways of deploying blobs of code and you have managed services, like BigQuery that I mentioned earlier. When you use these tools, they are serverless because you're not making calls to specific servers. You're making calls to APIs that abstract away the servers from you while guaranteeing uptime and reliability and scalability.

In previous shows, we've covered things like Heroku, and Firebase, and serverless functions, and serverless event driven application deployment, and a few startups that are built almost entirely on serverless infrastructure. This topic is endlessly fascinating.

Serverless is a way of describing these backend services that are represented by an API instead of using the addressable server notion. But what about the rest of the application stack that you use to build on top of this serverless backend concept? You still need to use JavaScript to define the custom code of your application. You still need to use HTML markup to describe the look and feel of your application.

The JAMStack is a way of building applications consisting of JavaScript, APIs and markup. Phil Hawksworth is the head of developer relations at Netlify and he joins the podcast to explain how these JAMStack applications are developed and deployed and how developers can use the JAMStack to rapidly build new systems. We also spent some time talking about Netlify, which is a fascinating company and certainly one of the companies for high productivity developers to watch closely and consider for their own projects. I hope you like this episode. It was pretty enlightening for me.

[SPONSOR MESSAGE]

[00:02:25] JM: OpenShift is a Kubernetes platform from Red Hat. OpenShift takes the Kubernetes container orchestration system and adds features that let you build software more quickly. OpenShift includes service discovery, CI/CD built-in monitoring and health management, and scalability. With OpenShift, you can avoid being locked into any of the particular large cloud providers. You can move your workloads easily between public and private cloud infrastructure as well as your own on-prim hardware.

OpenShift from Red Hat gives you Kubernetes without the complication. Security, log management, container networking, configuration management, you can focus on your application instead of complex Kubernetes issues.

OpenShift is open source technology built to enable everyone to launch their big ideas. Whether you're an engineer at a large enterprise, or a developer getting your startup off the ground, you can check out OpenShift from Red Hat by going to softwareengineeringdaily.com/redhat. That's softwareengineeringdaily.com/redhat.

I remember the earliest shows I did about Kubernetes and trying to understand its potential and what it was for, and I remember people saying that this is a platform for building platforms. So Kubernetes was not meant to be used from raw Kubernetes to have a platform as a service. It was meant as a lower level infrastructure piece to build platforms as a service on top of, which is why OpenShift came into manifestation.

So you could check it out by going to softwareengineeringdaily.com/redhat and find out about OpenShift.

[INTERVIEW]

[00:04:33] JM: Phil Hawksworth, you are the head of developer relations at Netlify. Welcome to Software Engineering Daily.

[00:04:38] PH: Thanks very much. Thanks for having me.

[00:04:39] JM: I want to talk to you today about the JAMStack, which is one of these terms that's used to describe a collection of developer tools and patterns that people use. There are examples of this in the past, like the MEAN architecture, or the LAMP stack. The JAMStack is a newer piece along that lineage. What is the JAMStack?

[00:05:07] PH: That's a great place to start. Yeah. On first glance, it does sound like a bit of a buzzword. I don't know that our industry is kind of full of terms like this, and you called out a couple of there. But JAMStack is – So the JAM in JAMStack, the J-A-M is JavaScript, APIs and markup, and really it's trying to describe pretty much as you said, like a way of approaching developments, a set of tools and processes that allow you to build sites in a slightly different way to maybe has been popular more recently. It's almost like the operating system has moved a tiny bit closer to the user in the browser. Really, this is talking about building sites in a way where we pre-render content with markups, so the content is there. Maybe we enrich that at the time at build time with access to APIs or we maybe call those APIs directly from the browser, client side, using JavaScript.

In this way, the JavaScript has kind of become the runtime of the web. You can start to bring the stack much closer to the user. So JAMStack is intended to talk about that. You could also argue, there's a bit of a convenience term to really get past some of the hang-ups that you might encounter when you're describing static sites. I often talk about static sites and the JAMStack is two sides of the same coin. Static sites, when you hear that term, you immediately think about the experience being static, but actually we're talking a little bit more about things that can be hosted on static hosting infrastructure or direct from a CDN, and that's a real virtue. But people kind of think about that as something that's going to only be experienced statically.

But if you think about it slightly different and you talk about the JAMStack, which it's JavaScript which could be served statically, but it could be used very dynamically direct from the client and then using things like APIs and, of course, sitting on top of the markup. That paints an entirely different picture of what you can create. I think that's where a lot of the kind of sites and projects that I work on, that's the direction they're going in. I hope that answers the question. It's a long answer to a short question.

[00:07:10] JM: Well, that term static site, I think it originated when people started serving – Well, maybe it didn't originate, but the first time I heard the term static site, was the idea that you could serve a website out of an S3 bucket, for example.

[00:07:27] PH: Right. Yeah.

[00:07:28] JM: And it's simultaneously amazing, because, "Oh! I've got this S3 bucket. I'm just using for storage. Whoa! I can actually serve an entire website out of it," and that's pretty nice, because setting up an entire website is sometimes annoying or it's historically been annoying. But it was, of course, quite limiting, because you just had routes pointing to assets that were sitting around in your storage bucket, your S3 bucket or your Azure blog storage bucket. But static sites or the notion of static sites is kind of a fuzzy term. Can you unpack that a little bit more? What do people mean when they say static website?

[00:08:11] PH: That's another good question, because what do people mean indeed? I know what I mean when I talk about a static site, is a site that is capable of being served from simple web hosting infrastructure. Being served directly from the CDN. You've already called out S3 as the perfect example. It's super convenient and kind of low maintenance for you as the site's maintainer to just throw your assets on to S3 and then they'll get serve. Fabulous! Having to maintain a server that has logic running that you got to keep patch with security updates and all of those kind of things. So there's a real virtue to being able to be capable of being served in that way. As you say, it does seem just a little bit limiting, because you're just serving dumb assets.

I think a lot of times that is a hang-up that we kind of have, and that's a bit of a hangover if you like of the way that we've looked at hosting infrastructure in particular. I think a lot of people do think of static sites as being a brochure, a resume, something very, very simple. Actually, a static site can actually become much more dynamic in behavior if you're able to lower things like the friction to deploying updates and the way that you can interact and those kind of things.

A typical dynamic site that you might think of could be a blog. You think of the biggest, most popular blogging infrastructure there is, something like WordPress, which of course runs on PhP and has a dynamic backend because it's hitting a database and all of those kinds of things.

Really, the content that you create on a blog – I don't know if I would necessarily describe that as dynamic in its behavior, is the kind of thing that's written infrequently and read very, very frequently.

So do you need to be hitting a database for every request is kind of the question there, and my position is to say, "No. You don't need to do that." If you can pre-render that and then have all of the views rendered on a CDN for people to hit, that's great, because that can be served statically. It's capable of being served statically. It means it's going to perform much better. There's fewer security, kind of attack vectors, and all of those kinds of things that go along with that in terms of the complexity of the infrastructure. So I always like to think about putting as much distance as possible between the end user and the complexity of the background. So if the complexity happened to build time instead of a request time, that's kind of desirable. So that's what I think of when I think of a static site. Something where the cogs aren't turning at the time the request happens, but that's not to say that you couldn't have lots of cogs turning and all kinds of complexity at build time. I think the tooling is getting better now to make that much lower friction and easier to do complex things in that space.

[00:10:59] JM: The thing you said there that stuck out to me was these websites are written infrequently, but perhaps read frequently. So I think about my own website, softwareengineeringdaily.com. We publish a podcast episode every day. We might publish an article or two per day as well. That's not a whole lot of writes. It's just three changes to the website. It's not a big deal, and that might fall under the rubric of static websites. We do have some more dynamic content. So we've had contents or a comments plugin in the past that we don't anymore. But when we have that comments plugin, the commenting system was actually handled by a third-party API provider. It was handled by Discuss.

So this makes me think that you often see these apps where, like you said, most of it is read often, write infrequently. Maybe you write two or three times a day. But maybe you have isolated components of the website that are perhaps write-intensive, like a commenting system, but the commenting system you can get handled by a third-party provider. This to me seems like one of the beauties of our modern infrastructure that is encapsulated in the idea of the JAMStack, a JavaScript – Was it JavaScript, APIs, markup? Is that right?

[00:12:27] PH: That's exactly it. Yeah.

[00:12:28] JM: Right. So APIs, bevy of APIs that are available to people, whether you talk about Stripe, or Discuss, or name your API in the ever-expanding grocery list of APIs. That's where you can get your dynamism. Of course, they take care of the distribution and the write or read intensity. They naturally scale up to handle those things.

[00:12:54] PH: Yeah, that's just exactly the point. It's hard to be an expert in all of those things. My background was originally in web development, particularly frontend development, and I think the more time I spent in that role, the more I've realized that there was just this huge bevy of things that I didn't understand about web development. It's hard to be a very good frontend developer and a very good DBA, or DevOps person. There's a lot to take on.

I really like this notion of handing off some of the complexity and things that are either commoditized or very, very complex to other services that specialize in it. I mean, you mentioned Discuss is a great example for comments, and also Stripe is a great example for kind of payments and ecommerce. That's complex stuff to do yourself, and if there are services that you can leverage, then I think there's great advantages to do that.

Now, I would just kind of offer a slight counterpoint as well, that you need to be a tiny bit cautious about what kind of experience that's going to create for you. It used to be that you might just drop in like a JavaScript plugin and that would create the entire experience for you and you got what you got, particularly with things like ecommerce and kind of payments journeys. I think it's important to us as people who are publishing things for a brand, trying to create a brand identity and experience to have that control where possible. That's where it's really nice.

I think now that the wealth of APIs that exist and the wealth of services that exist now allow you to tell your own experience on top of their infrastructure and on top of their APIs. It just unlocks all of these great possibilities. So I think we're seeing more and more examples of that out in the wild now, and I think that what makes it kind of an exciting time.

[00:14:40] JM: I think we're in a place right now where we have yet to see any gigantic scale JAMStack style apps. You can correct me if I'm wrong, but I think it is mostly today used for more narrow kind of content-y websites. There's less of dynamic – There's no social networks built on the JAMStack I don't think. What's the most complex app you have – By the way, that's not to say that we won't see these things proliferate. I think we will, but what's the most complex JAMStack app you've seen?

[00:15:17] PH: Well, I could take the easy way out and talk about Netlify itself, because Netlify is a JAMStack app. The entire thing, we very much live and breathe the model we talk about. Netlify is a service that does this kind of this type of hosting and this type of infrastructure, but also there's a fairly complex admin interface. So anyone who hosts a site there, they can add a site, connect it to their Git repository, do various configurations there. See all of the deploys that have happened, roll forward and bad, and it's a fairly complex admin console on there. That's a React app, and that is sitting on top of a JAMStack architecture. This is JavaScript, which has bundled a build time. It's served from a CDN and it's hitting APIs in the background. That's pretty complex. That's pretty rich. It's the kind of thing that you would maybe not typically have thought you would serve as a "static site." But that's certainly what we do.

It's kind of a nice side effect from that. One of the benefits from that is that since every deployment that has happened is just a bunch of static assets that live out on a CDN, it means that we can step back and forward through time for previous deploys, because they're all like this immutable atomic deploys. It's a bunch of assets that just lives out on a CDN that's talking to our APIs.

On my first week at Netlify, we had a meeting where it's like an all-hands meeting, and there was a nice moment where we're kind of just doing a bit of a retrospective of where Netlify come from. One of the team members showed the Netlify admin console as it is today, as it was six months ago, as it was a year ago. The way that he did that was he just showed a different version of the site deployed to stepping back to different versions of the admin UI deployed, because it's just a bunch of dumb static assets.

But really, that's still talking to the same APIs in the background. So still usable and workable. That's a very extreme example of that, because you wouldn't necessarily roll back your app a year at a time. But knowing that you can roll back any deploy instantly for something that you've just updated, that's incredibly liberating. So this is an example of a very complex admin UI that's built on the JAMStack. But as I said, that's kind of maybe taking the easy way out and talking about something very close to home.

Another good example, and we do use this reference quite often, is smashing magazines. So I don't know if you're familiar or your listeners are familiar with smashing magazine, but a very large publishing sites specializing web development content. That's a site that was once hosted across a whole host of different services. There was WordPress. There was a CMS. There was a shop, a custom shop on another platform. There's a job board. These were all on different platforms and they moved across to the JAMStack. We did a bit of a project with them to do that.

Yes, it's a very big publishing site that has lots of content posted, but it also has comments. It also has ecommerce ticketing. Still a job board. This is all on the JAMStack as well, and actually it's the kind of thing that you would absolutely not associate with this notion of a static site. It's all possible, and these kind of APIs that enable that are out there now and they're getting more and more traction and those are becoming more readily available.

As you said a moment ago, I think the examples that we're going to start to see, they're going to get more and more prolific and more and more advanced and I think they're going to really start to prove out that, yes, a blog is a very obvious place to go with this, but it's far from the limits. You can really apply this to most of the things that you might be able to imagine.

[00:19:07] JM: Right. Even I think about something like Netflix, or something – Like Netflix I think is actually a good example. So there's a company, a fairly new company called Mux. They're a sponsor. They've been a sponsor of the show in the past, and I've had them on the show a couple of times. They're really interesting to me, because they take care of video hosting and delivery and easy embedding of video and it's all through an API and it's a very simple API. I think it's an API that fewer developers know about than they should, because I think a lot of developers are afraid to build applications that heavily involve video, like video sharing apps, for example. They just assume it's really hard and complicated, but that's an example of something that's becoming easier.

I know AWS also has some APIs around making video a lot easier. But you can imagine a very rich serverless Netflix style thing, or I think a machine learning heavy, data heavy app. Somebody might be afraid to build a static site or an infrastructure involving that kind of thing. But we just did a show a while ago with a company that's a real estate, a machine learning company. This is one of the most interesting shows that I had the pleasure to do recently about this company that just runs large scale machine learning stuff around housing and real estate and they've got an infrastructure that's mostly serverless, and I see the serverless stuff as kind of the same thing as the JAMStack. You have maybe an admin panel kind of thing, or a dashboard kind of thing. It is framed in a static site, kind of JAMStack UI layer, but the dynamism of the app might be described more in terms of the serverless parlance. These are sort of two sites of the same coin. Maybe JAMStack is the frontend. Serverless is the backend.

[00:21:16] PH: That's a really good way of putting it. Certainly, I know that serverless is such a kind of contentious term. I know that people kind of respond to it quite viscerally saying "[inaudible 00:21:24] servers," and absolutely [inaudible 00:21:26] servers. But it's not a server you ever need to concern yourself with. It's a server that you're never going to touch the infrastructure and you don't need to even think about it in those terms.

But we certainly see – At Netlify, we certainly think that serverless is a great companion to the JAMStack. It's a great way of extending the capabilities and extending the kind of power of a site that's hosted as a set of static assets. One of the things that we're particularly excited about actually is building things out so that for some roots on a website, that you absolutely can't pre-render that at build time. You need to have some kind of call to API that's maybe that involves secrets that you can't send through the client, or maybe that performs a tiny bit of logic at request time. Serverless functions are a great way of doing that. If you can just identify the few roots on your experience that need that, augmenting a JAMStack site with these serverless functions is a fantastic way of getting that logic at request time that you don't really want to have to build and maintain and secure an entire server infrastructure for. Yes, certainly, kind of a good marriage of those two.

[SPONSOR MESSAGE]

[00:22:48] JM: For all of the advances in data science and machine learning over the last few years, most teams are still stuck trying to deploy their machine learning models manually. That is tedious. It's resource-intensive and it often ends in various forms of failure. The Algorithmia AI layer deploys your models automatically in minutes, empowering data scientists and machine learning engineers to productionize their work with ease. Algorithmia's AI tooling optimizes hardware usage and GPU acceleration and works with all popular languages and frameworks.

Deploy ML models the smart way and hardware, and head to algorithmia.com to get started and upload your pre-trained models. If you use the code SWEDaily, they will give you 50,000 credits free, which is pretty sweet. That's code SWEDaily.

The expert engineers at Algorithmia are always available to help your team successfully deploy your models to production with the AI layer tooling, and you can also listen to a couple of episodes I've done with the CEO of Algorithmia. If you want to hear more about their pretty sweet set of software platforms, go to algorithmia.com and use the code SWEDaily to try it out or, of course, listen to those episodes.

[INTERVIEW CONTINUED]

[00:24:23] JM: What are some of the best practices of the JAMStack?

[00:24:29] PH: It's such a broad thing, but I've been building things for the web for a long time and you might kind of describe me as a bit of a web hippy, and although I'm kind of pushing something that's got the word JavaScript at the very beginning of it. I'm always a little bit cautious about bundling everything into JavaScript and leaving everything just to JavaScript.

So one of the best practices I would say is pre-rendering as much as you possibly can. It's great to use progressive enhancements to augment these things, but wherever possible having the highest possible watermark for where you start from, where you start to progressively enhance from. That's just a really good practice. It's good not just for things like performance, the speed of the render for the user, but just also in terms of making sure that as much content is delivered as quickly as possible and you got a really kind of robust starting point.

There are a lots of types of sites that could be described as JAMstack that don't need any JavaScript augmentation in the frontend. It's purely pre-rendered, but there are also lots that do require that and do benefit from that. But I would say pre-rendering as much as possible and then using progressive enhancement to then add those other layers. That's a good practice for every kind of site and that extends to the JAMStack as well.

Another thing that I think is probably worth calling out is a really nice pattern for using static site generation and using the JAMStack, and that is in harvesting content from lots of different APIs or the appreciate APIs at build time. Often, there's this nice pattern of a static site generated that's run in a local build environment and a build step and having the first thing that it does to say, "Okay. Well, I'm going to go and get the content I need from," for example, "the Twitter API. My content API for maybe a headless CMS, maybe a pricing engine. I'll get that data, gather together. They'll run my static site generator and create this deployable asset," and then that gets published out on to the CDN.

So having that kind of multiple step of gather the assets, generate the site, perform the optimizations and deploy. That's a really kind of nice pattern as a place to start from.

When I described it that way, it sounds like, "Well, you're doing quite a lot," but that's where build automation and some of the tools like Netlify really come into their own, because that's where we can use kind of scripting and automation to make these processes repeatable and very easy to perform many, many more times than you might traditionally think about doing a deployment. Just as by way of a kind of ridiculous example. I know you asked me about best practice, and I'm about to give you a ridiculous example.

This wasn't necessarily best practice, but this illustrates the growing confidence and how frequently you could perhaps run a deployment. I responded to a slight challenge on Twitter recently where someone suggested, "Well, here's an idea for you. How about a static site hosted on Netlify that redeploys every minute to tell you the current time? Building a statically generated clock."

I made that, and actually it works beautifully. Of course, that's a ridiculous example. A site that regenerates and redeploys every minute to tell you the correct time, but it's just this demonstration that the confidence in doing a deployment and the friction to do a deployment is reduced and the confidence has gone up through the roof. So it means that you can start to have these build processes where you're doing quite a lot, but you're happy to do it frequently. That's kind of a silly example. Yeah, I think going back to your original question, pre-rendering as much as possible using progressive enhancement. Those are real kind of key lookouts I would say.

[00:28:16] JM: For people who want to build with more dynamism, they'll get to a point where they need some third-party API, the Stripes, or the Discuss, or other APIs, and there's been a rise of some API marketplaces. So there's Rapid API, there's Manifold, there are some other places where you can find large collections of APIs. Do API marketplaces fit into any of the workflows that you've seen with JAMStack developers?

[00:28:51] PH: I'm not sure if I'd say they fit in with the workflows. I think they're fantastic resources. I think finding the right services and finding the right APIs that you can leverage is of course a big step to architecting any sites, but particularly JAMStack sites. I don't know I would describe it as a key part of the workflow. But, yeah, certainly it's an important resource.

Also, having the opportunity to assess those APIs and assess the capabilities of the services that are providing is another important step. Feeling confident that the services that you're building on top off are going to exist for long enough for you to use them. That's always been a consideration when you're architecting any kind of site that's going to leverage a third-party service, and that's just as much the case on the JAMStack.

As you mentioned, these kind of marketplaces are starting to evolve and they're kind of API economy if you like. It feels like it's getting more mature and more established as a viable business for these API providers to exist in. I think confidence is just generally growing across the board for those kind of things. Yeah, things like those API marketplaces are a big help for that.

[00:30:10] JM: You work at Netlify. I want to get into what Netlify. Netlify started by being widely known as a static site hosting company, but just as static site hosting is a fuzzy term as we've already discussed. Netlify has grown to encompass more than that definition. How would you describe Netlify today?

[00:30:37] PH: Yeah. Apologies. I started talking a little bit about Netlify before we'd really introduced it. Yeah, you're quite right. Its background does come from static hosting. It kind of grew out of a company called BitBalloon, which was doing exactly that. It was just purely kind of, "Give us your directory and we'll get it to CDN." It was just reducing the friction and deploying it, and that's largely where it stopped.

When it grew into Netlify and we started to add more capabilities in there, I think the best way to describe what we are now is a set of modern tools for web developers to get their sites deployed as rapidly as possible on to a global CDN, and then we include build automation and other tooling associated with that. The kind of things that we'll do is we'll say, "Okay. You want to create a new site? Tell us the git repository where your code for that is housed and then we'll set up some git hooks without repository, and every time you push code to that repo, we'll spot that's happened. We'll pull your code. We'll run your build. So we'll run it in our internal – In a container, in like a CI container, and we'll perform that build for you. At the output of that build, we'll propagate to a global CDN on your behalf and we'll handle everything like that cache headers and the cache expiration, all of the kind of the stuff that is frankly painful and very easy to get wrong." We're trying to commoditize that and say, "You shouldn't need to worry about that. We'll do that for you." So that's the kind of the very basis of what we do.

But then on top of that we also have some other kind of tooling. I already mentioned that these deploy are immutable and atomic. So in other words, whenever we do a new deploy, it's not a mutation of what's there already. We're not updating some files and some assets over deploy. We're creating an entirely new instance of your site and then we propagate that to the CDN globally, and once everything is in place, then we flip it over and the traffic starts immediately going to that. That allows us to do things like these rollbacks from one version to any other. You can step back anywhere in history, those kind of things. That's kind of nice.

But then there's a few other obvious tools that we'd add to that to start to raise the ceiling of what you can do with a site, which is host it statically on a CDN. So for instance, an obvious barrier that you would run into when building a site like this is – Everything is basically a static view, but I need a form on there. I need to be able to just allow someone to register or give me their details. Just submit some data. I need somebody to post to. Of course, there's no server running in here, because we've tried to take the origin out of it. We're trying to take the service out of the equation.

So what Netlify do is instead of giving you a server where you can write your form handlers, what we'll do is as we run your build, if we see in your HTML that there is a form element and if you decorated it with a Netlify attribute, we'll say, "Okay. We'll look at the form. See what form fields are in there and we'll create for you on the fly an API that that will post to for you automatically." So that's not adding like another third-party or a JavaScript dependency. That's just purely giving you a form handler.

Now, that HTML has somewhere that it will post data to, and then you can access that data either through the admin UI or we give you an API. We give you an API effectively for everything. So then you can get that data back and either use it in another build or you can notify other things and those kind of things. So that's kind of an obvious kind of next step that we'd add as tooling. There's a bunch of other things like we'll do branch builds for you. I don't if you've ever kind of lived through the pain of working on a big project where you've had to create a production environment, a staging environment, a dev environment, a testing environmental, all manner of environments which are meant to be kind of perfect facsimiles of each other. So you have confidence as you build your site out and progress. It can be quite painful in creating all the infrastructure.

The way that we do is that we just treat every bit of infrastructure just exactly the same. Everything is effectively production infrastructure. It's kind of the thing that we're trying to commoditize and let you build on top of. So if you decide that, "Well, I want to be able to have a staged version of my site as well as the production one." You do that using git conventions that exist already. So you'll create a branch and you – Whatever. You name that branch. We'll create an instance of your site with that name prefixed in the URL as well for you. So it means that you create a dev branch, you push that. We'll also build that and we'll propagate that to dev.yoursize.netlify.com or what have you. It means that you can start to create these different environments that are all based on branches and the kind of git infrastructure on workflows that developers are already kind of used to.

The key is that these things are all kind of as production. They're all on the same global CDN. So there is no loss in performance between them. They're really kind of indicative of what the experience is truly going to be. So that's kind of quite empowering.

I'll give you another word in a second. I'm sorry. I'm kind of rambling, but this kind of leads me straight on to the next thing, which is once you've branch deploys where you can serve different versions from different branches of your sites at the same level on the CDN, we can start to add things like A-B testing to that.

Now, A-B testing is one of those things, which is often seem to be very powerful, but quite hard to do very, very well. Lots of popular approaches to this, we'll may be used JavaScript to let you serve different versions of your site. So maybe kind of a small iteration to either some language or some UI, and that's often done with JavaScript. Difficulty there of course is we know that performance has a huge impact on experience. So if we're serving two versions of the site where the performance is impacted by needing to call another JavaScript API somewhere else to change the UI in the first place, that can have an impact.

We're in a nice position where, since we are able to server all of your sites from all of the versions of your sites on their different branches directly from the CDN, we can start to run A-B test directly from the CDN as well. So it will say, "Okay. We'll set up a test that runs comparing – Splits of half the traffic to go to the master branch and then half to my new purchase button branch." That traffic switching happens at the CDN level. So both the experiences are being served to the end user exactly as they would be in production. There's nothing standing between those. That can be really powerful in terms of a way to do things like A-B testing. It ends up being really trivial to set that up, and I've lived through much more complex setups in different types environment where actually creating full A-B tests can be can be quite difficult. Those are just a few things.

[00:37:28] JM: The A-B testing element is pretty interesting, because there are all these A-B testing systems that really do penalize you on performance, but if you have different versions of your site getting built at build time – Like you said, you have this build system where I just put in a GitHub repo and the GitHub repo gets pulled on to a container and gets built on that container and then served to a CDN. Well, if you built multiple versions of the site right there, then it's like each of those versions of the site, when actually get served to the user, you don't have any latency from the A-B testing framework doing some switching statement based off of what this person's geo is, or who they are. I guess you could have that work being pushed to the CDN, like maybe if you want it to be doing an A-B test that was switching on the user's gender, for example, then you could have the CDN do the switching right there.

[00:38:36] PH: I guess so. Yeah. I mean, you need to be able to detect and then choose to route from one thing to the other. So deciding how you capture the gender, you could certainly do that. For instance, the CDNs, the edge nodes that serve the sites, we've build some logic in there as well to do things like detects your language settings or your local. So things like localization, that also become something that you can do in pre-rendered fashion as well. A static site generator that builds out a version of your site for all of the languages you want to support, those are all then pushed out to the CDN. Then we're just rooting to different paths in the site at the CDN level. Again, kind of switching content for you in that that way.

It sounds like it's the kind of thing that would be potentially quite complex to do elsewise, but just having that little bit of logic at the CDN level that we just give you access to just so that you can root things in different places. That turns out to be just really quite powerful.

[SPONSOR MESSAGE]

[00:39:50] JM: Azure Container Service simplifies the deployment, management and operations of Kubernetes. Eliminate the complicated planning and deployment of fully orchestrated containerized applications with Kubernetes. You can quickly provision clusters to be up and running in no time while simplifying your monitoring and cluster management through auto upgrades and a built-in operations console. Avoid being locked into any one vendor or resource. You can continue to work with the tools that you already know, such as Helm and move applications to any Kubernetes deployment.

Integrate with your choice of container registry, including Azure container registry. Also, quickly and efficiently scale to maximize your resource utilization without having to take your applications offline. Isolate your application from infrastructure failures and transparently scale the underlying infrastructure to meet growing demands, all while increasing the security, reliability and availability of critical business workloads with Azure.

To learn more about Azure Container Service and other Azure services as well as receive a free e-book by Brendan Burns, go to aka.ms/sedaily. Brendan Burns is the creator of Kubernetes and his e-book is about some of the distributed systems design lessons that he has learned building Kubernetes. That e-book is available at aka.ms/sedaily.

[INTERVIEW CONTINUED]

[00:41:26] JM: I have spent a lot of time using Firebase. I like Firebase a lot. What's nice about Firebase is it simplifies working with large datasets and heavy database related applications as well as these lighter weights static sites based applications. Now, that's not to say these things are mutually exclusive, but do you see potential for building products within Netlify that allow for more data rich applications?

[00:42:00] PH: Yes, totally, and that's a really good example actually. Mentioning Firebase, that's a really good example, because I know a lot of people build static sites that are hosted on Netlify and then some of the roots take you to views that are fulfilled by either Firebase or something that queries Firebase in the background.

We mentioned serverless functions a little bit earlier on, and one of the things that certainly you can do with the serverless function is you can be hitting a data source and then fulfilling a template and then returning that, or indeed creating APIs that you're serving from something like Firebase or something along those lines.

That is an API that you consume from a static sites wherever that's hosted. Whether that's Netlify, or Firebase, or Dropbox, or GitHub pages or what have you. At Netlify, we don't think we're going to get into the game of giving you a database within our environment ourselves, but we're very keen on making it as simple as possible for you to access database services wherever else they might exist.

We know that different people have different tastes and different constraints about what they can and can't use. So really trying to create this kind of glue layer that allows you to build out sites that can make use of database services through suitable APIs wherever they may be. We think that’s really powerfully and we think that's quite robust. I think it's one of those things where this sensible abstraction of some of the services means that you can weather the storm if one of these services goes away and you need to use one thing instead of another rather than putting all of your eggs in one basket. So we think this kind of abstractions is really powerful, and certainly I think sites that are built on the JAMStack can leverage APIs and leverage data services wherever they may live.

[00:43:56] JM: I've done a lot of web app deployments on Heroku also. I am a big fan of Heroku. Heroku I think I think was somewhere in between the cloud serverfull world and the serverless, if you want to call it a revolution, but shift, because Heroku really did simplify a lot of the deployment issues that people had, but you still have the server that you get up and running and it’s still in that kind of paradigm. So I think the costs can be a little more pricey with Heroku as much as I love it. There's also a Gatsby. So Gatsby is –I'm just thinking of other things that are kind of in this space.

So Gatsby, we did a show with. That I believe is an open source static site generator and they’re building, I believe, a business around that or maybe they already launched a business. How does Netlify compare to Gatsby?

[00:44:58] PH: Yeah. As you say, Gatsby is a static site generator. So it’s a tool that will take templates and markup and contents, effectively run a build and then generate a deployable site. One of the things that Gatsby does really nicely is that they have a build, which is reasonably opinionated in the way that it delivers your site, your frontend. So they bundled in lots of really nice optimizations and creates almost like a progressive web app so that you can have things like service workers implemented for you automatically and give you things like offline access, what have you, so you can do a very perform rendering and also be able to withstand the sever, the connection to the network rather and kind of going away while you're on the site. So that’s really good. But some, they don’t provide hosting. So they are a build tool that you can use locally or on some kind of continuous integration server, or indeed within Netlify, to take your code and generate a site that you can then host somewhere.

Another thing that they do really nicely is that they are starting to explore interesting ways to expose the data. So when we're talking about the content APIs that might drive a site so that you're pulling content from something like a headless CMS or whatever content sources at build time, they build a lot on top of GraphQL to give you quite a nice rich way of querying the different data structures that exist in kind of the API ecosystem that you live in so that you can have very rich access to this content while you compile your template. So that's rather nice.

Again, this is something that you run at build time and then the output is something that you then host, which is – So we work very closely with – Or rather we serve a lot of sites that are built using Gatsby. There are a few services like that that kind of seem to be very, very popular at the moment, kind of capture the zeitgeist and Gatsby certainly is one of those.

[00:46:57] JM: How do you see GraphQL fitting into the JAMStack in the near future?

[00:47:04] PH: I think it's incredibly valuable. I think it's a very interesting prospect, because, again, going back to my slight web hippie days, I'm just a huge fan of Rest and Restful services. I think that's really powerful fundamentals of the web, because this kind of – It slightly disrupts that a little bit and it's a different way of querying APIs and interacting with APIs. But I think the flexibility that it brings and the efficiency that it brings can be really empowering. I've certainly built out and help to architect applications in the past that had this very nice delineation between the API layer and the UI layer, and designing the UI layer to be able to get the data it needed from the API that, again, would be designed and implemented along the way to give you just the right calls to populate your sites and do the mutations and what have you. That is quite prescriptive. You need to be able to design those things in concept.

With GraphQL, it gives you much greater freedom to be able to say, "These are the data structures that I need in my UI, and as long as the underlying data is going to be accessible through this GraphQL API layer, I can shape the request any way I like to be able to give me just the data I need just in a single request. I think that's great for things like performance and maintainability of the UI and the robustness of the API layer. I think it's a natural fit for the JAMStack and I think it will get more and more popular.

The one thing I would say, and this is just a standard word of caution. I've seen lots of sites using GraphQl that didn't need to, just because they're actually way simpler than that. It could've been done at a much simpler level. So sometimes it can add complexity sooner that it needs to, but I think that's just the kind of technology siren that exists for all of us building stuff on the web. We like seeing new things. We like exploring the new ways of building things and that's just kind of the natural temptation.

Certainly, GraphQL is incredibly powerful, incredibly liberating for lots of things that we might build that are more complex, but I wouldn't necessarily rush to use it on everything that's going to do a request to a feed or something like that.

[00:49:24] JM: We've done a pretty good job of exploring the trends of serverless and JAMStack here I think, and for people who haven't really tried to build apps in this way, I personally recommend it. I find it really fun and lightweight and frankly easier. I came out of college and was working on these big old legacy Java applications and it's just less fast and less fun to get stuff to production. I just – I will never go back to that world, thankfully.

I think the cost is actually lower. You get generally an easier app to deal with and the cost is lower, because just the trends in cloud computing and the way that these services get stitched together really amortizes your costs, or it amortizes each of these different services that you're using. They are amortized. They have economies of scale, and so things just seem overall cheaper. Is that your sense as well? Just app development is much cheaper than it used to be?

[00:50:31] PH: Oh, 100%. Absolutely. I think this kind of trend and this kind of shift towards like a JAMStack model, one of the key things it's doing is it's taking whole chunks of complexity out of your architecture. The architecture that you have to be responsible for, but also the architecture that's in play. If we take away service that needs to respond to every request that comes in, performs some logic, hit a database which needs to exist somewhere as well. Maybe go through load balances, go through all of kind of redundancy planning and all those kind of things. If you take a lot of that away and just say, "You know what? We're serving this directly from the CDN," you're removing a lot of moving parts and you're removing a lot of need for things like redundancy and all of those kind of things. You're building on top of some of the things that are commoditized and then starting to use services where their business is purely in building and providing services in a timely and performant way. They themselves have economies of scale.

I totally agree that it's lowering the barrier to entry to building some of these things and it's certainly lowering the cost. Another place that the cost can be reduced that is less obvious is in the skills that you need to build some of these things. The breath of expertise that you need in a team to build some of these very complex enterprise grade, which is a term – I'm sorry. I didn't mean to use that term, but it's often one that you hear. This kind of very high performant grade application, you don't need the same diversities of skills that you may have needed in the past, because you're outsourcing the need to be a DevOps expert, or a caching expert, or all of these different things that you would normally have to take care of yourself. Your team now can focus on client-sides technologies, frontend development technologies, API design, and it's a much smaller set of skills that you need to have on the team, but you're kind of leveraging the skills of other people that are kind of in other places. So, yes, it's faster, it's cheaper, and I agree, it's just more fun. I feel more productive than I've ever felt before. It's nice.

[00:52:48] JM: Are there any other developer trends that you expect to see in the near future that we haven't covered?

[00:52:53] PH: That's a million dollar question. I mean, from my position, someone who kind of looks a lot at things like frameworks and static site generators, they move so fast and there's so many out there. I do watch quite closely which frameworks seem to be getting popular and which static site generators and getting popular, and I think things like ViewJS, particularly interesting at the moment, which is another JavaScript framework, which can be used as static site generator or it can do server renderer as well. I think that's getting a lot of momentum at the moments and it's starting to kind of compete a little bit with things like React in terms of how people are using it and its popularity. That's not looking very far ahead. I think that's already gaining in popularity right now. I know that we'll be surprised. I think the browser capability is getting more and more interesting and more and more rich and we're starting to see people use things in more and more interesting ways. So I don't really know what's around the corner. It usually surprises me.

[00:53:54] JM: Phil Hawksworth, thank you for coming on Software Engineering Daily. It's been great talking.

[00:53:57] PH: Thanks, Jeff. Thanks for having me. It's been a lot of fun. Thanks.

[END OF INTERVIEW]

[00:54:03] 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.

[END]