EPISODE 513

[INTRODUCTION]

[0:00:00.2] JM: Over the last decade, computation and storage have moved from on-premise hardware into the cloud data center. Instead of having large servers on-premise, companies started to outsource their server workloads to cloud service providers. At the same time, there's been a proliferation of devices at the edge. The most common edge device is your smartphone, but there are many other smart devices that are growing in number; drones, smart cars, nest thermostats, smart refrigerators, IoT sensors, next-generation centrifuges. Each of these devices contains computational hardware.

Another class of edge device is the edge server. Edge servers are used to facilitate faster response times than your core application servers. For example, Software Engineering Daily uses a content delivery network for audiophiles. These audio files are distributed throughout the world on edge servers. The core application logic of Software Engineering Daily on a WordPress website, and that WordPress application is distributed to far fewer servers than our audiophiles, because the audiophiles are the content they get distributed through a content delivery network.

Cloud computing and edge computing both refer to computers that can serve requests. Technically, all of these things could be treated as servers. The edge is commonly used to refer to devices that are closer to the user so they will deliver faster responses. The cloud refers to big bulky servers that can do heavy duty processing workloads such as training machine learning models or issuing a large distributed map reduce query. As the volume of computation and data increases, we look for better ways to utilize our resources and we're realizing that the devices at the edge are under-utilized.

In today's episode, Kenton Varda explains how and why to deploy application logic to the edge. He works at CloudFlare on a project called CloudFlare Workers, which is a way to deploy JavaScript to edge servers. Edge servers are, for example, the hundreds of data centers around the world that are used by CloudFlare for caching.

Kenton was previously on the show to discuss protocol buffers, which is a project that he led while he was at Google, and if you want to find that episode as well as many other episodes about protocols, serverless technology, many other topics, you can download the Software Engineering Daily app for iOS or android. These apps have all 650 of our episodes in a searchable categorized format. We have recommendations, related links, discussions around the episodes. It's all free and open-source, and if you're interested in getting involved in our-open source community, we have lots of people working on the project and we do our best to be friendly and inviting to new people coming in looking for their first open-source project. You can find that project to github.com/softwareengineeringdaily. You can find our slack channel at softwareengineeringdaily.com. We would love to have you as part of the community if you're interested in working on some open-source software.

With that, let's get on with this episode.

[SPONSOR MESSAGE]

[0:03:18.0] JM: DigitalOcean is a reliable, easy-to-use cloud provider. I've used DigitalOcean for years whenever I want to get an application off the ground quickly, and I've always loved the focus on user experience, the great documentation and the simple user-interface. More and more people are finding out about DigitalOcean and realizing that DigitalOcean is perfect for their application workloads.

This year, DigitalOcean is making that even easier with new node types. A $15 flexible droplet that can mix and match different configurations of CPU and RAM to get the perfect amount of resources for your application. There are also CPU-optimized droplets, perfect for highly active front-end servers or CICD workloads, and running on the cloud can get expensive, which is why DigitalOcean makes it easy to choose the right size instance, and the prices on standard instances have gone down too. You can check out all their new deals by going to do.co/sedaily, and as a bonus to our listeners, you will get $100 in credit to use over 60 days. That's a lot of money to experiment with. You can make $100 go pretty far on DigitalOcean. You can use the credit for hosting or infrastructure and that includes load balancers, object storage. DigitalOcean Spaces is a great new product that provides object storage and, of course, computation.

Get your free $100 credit at do.co/sedaily, and thanks to DigitalOcean for being a sponsor. The cofounder of DigitalOcean, Moisey Uretsky, was one of the first people I interviewed and his interview is really inspirational for me, so I've always thought of DigitalOcean is a pretty inspirational company. So, thank you, DigitalOcean.

[INTERVIEW]

[0:05:24.0] JM: Kenton Varda, you were last on the show to discuss protocol buffers and you're back on the show. Thanks for returning.

[0:05:32.4] KV: It's good to be here.

[0:05:33.8] JM: You work at CloudFlare. On the last episode we talked about protocol buffers and then your company Sandstorm and then how you eventually wound up at CloudFlare, and today we're talking with some technologies that are related to what you work on at CloudFlare. So maybe you could briefly explain what the company does for people who do now.

[0:05:52.3] KV: Well, the way I like to explain it to people like my parents is CloudFlare runs about 10% of the internet, more specifically, its 10% or so of web requests. HTTP requests end up going through CloudFlare's edge network where some people call us the CDN, but it's a bit more than that. Let's put it this way; if you have a website and you put it behind CloudFlare, then when people visit your site, their web requests go to CloudFlare first where CloudFlare can do a number of things with our network of — We have servers in 118 locations, I think, is the number today. We can cache resources from your site so they can be served directly from that location that's much closer to the user then your servers might be and are to respond more quickly, and slew of other features, security features, blocking, denial of service requests, blocking hacking attempts. There's a long list of though. You have to go to the website to see.

[0:06:52.7] JM: Right. At a basic level, people could consider CloudFlare to be a caching service, but it's more generalizable than that. It's edge servers. So if you want to service a user's request, any times that user request can be served entirely by an edge server as opposed to your core infrastructure, which might be on Google Cloud, or AWS, or Heroku or hosted on your own servers. You would want to use CloudFlare for the majority of those requests if you could, because the distribution of all those different edge servers is going to make them closer to the user, and therefore you're going to get more performant responses sort.

We're going to be talking about these edge servers. How would you define an edge server?

[0:07:41.9] KV: So there's a lot of different things people mean when they say edge. What we mean is — So our network of 118 data centers around the world, they're basically servers that are spread out all over the place that they are close to the end users, and that's basically it. Like we have servers that are within 10 milliseconds latency of 90% of the world's population, and there's lots of ways you can use that.

[0:08:07.9] JM: When you compare that to a model like AWS, or Microsoft, or Google Cloud, these other cloud providers, is their competitive advantage. Are the servers more geographically distributed than those other cloud providers?

[0:08:23.6] KV: Yeah. So usually when you use something like AWS or Google Cloud, when you start out, you have your servers and you decide one particular region that you put them in, and as you grow you might ask do multi-homing and have servers in multiple regions, but the total number of regions that AWS or Google Cloud give you is a handful. There's like a few locations in the United States, a few locations in Europe, a few in Asia, but it's not a huge number.

But then what you would do is with CloudFlare you would put CloudFlare on top of that. So you'd use these together. It's not either/or. So then you can leverage CloudFlare having servers everywhere to make your site faster and more secure and more reliable.

[0:09:05.5] JM: We've done some shows about using IoT edge servers. So we've had some conversations about the idea of connected cars, for example. Once we have all these cars on the road that have servers inside of them essentially, like the a world full of Teslas, or when you talk about all the IoT devices that we might get in the coming future, your IoT refrigerator, your IoT speakers, your IoT light switches. Who knows? then drones, for example. If we have drones buzzing around outside. All of these different devices could be used for edge computing. How do you think about edge computing on data centers versus on IoT devices?

[0:09:51.8] KV: So this is one of those other definitions of edge that some people use. I think that the definition we're using goes back a little further than this newer definition. But yes, sometimes when people say edge, they mean all desire IoT devices. Now that's not usually what we mean when we're talking about our edge network of data centers. This is just a different definition, I guess.

[0:10:15.4] JM: Yes. Okay. Many people are familiar with the idea of a CDN, and let's say you make a request for an image on the internet, and the first time you request that image, perhaps it's served by the source of where that image is. Maybe it's in the Amazon S3 bucket somewhere on the internet, and because the user has requested it, that image gets stored in a cache somewhere. Maybe it get stored in a CDN on the edge or somewhere on CloudFlare, if you want to call CloudFlare or CDN, get stored in CloudFlare perhaps, so that the next time somebody else accesses it, that access is faster. So this is the common model of just throwing storage on an edge server as a CDN. We've had CDN's for a long time, Akamai, Libsyn, the company that serves the podcasts that I have. It's served from a CDN. These are often the content delivery networks, because you have this content that's often — Netflix has tons of CDN's or has tons of — I don't know, what they use for CDN, but they have tons of content throughout CDN's. But a CDN, thinking of edge servers just for CDN purposes, is a somewhat limited idea, and I think what we're going to get to talking about is the fact that you don't just want to put storage on edge devices. You actually want to put compute logic on the edge. You don't just want to be using this is as a CDN. Why would I want to deploy logic to the edge?

[0:11:48.1] KV: Well, sometimes you want to interact with the user in a way that is not static. So with just the cache, all you can do cache static assets, like images and such that no change. But say you have a website and it's like a new site, so it contains mostly static content and that the articles don't change very often, but people log in to your site because they have subscriptions. Once they login, now you want to say at the top like, "Hello X, you are logged in," and that's a little change to your site. That now means that the pages are all different for that user than they are for everyone else. So now they can no longer share a cache effectively. I mean, you can do it in JavaScript, perhaps, but sometimes people don't want to do in JavaScript. So if you could write some code that runs on the edge and can substitute in the user's name from there, then you can return results much faster rather than going all the way back to your home server.

[0:12:47.8] JM: Right. Does this start to become something where if we start to deploy some logic to the edge, you could imagine slippery slope where, “Well, let's just put all of our servers on the edge. Let's service every single user request from the edge.” Why is that not a slippery slope? Why would we not start doing that?

[0:13:06.7] KV: We don't know yet if this is where this is going to go. For now, our goal in making this so you can write code on the edge is more around these kinds of optimizations, but over time we’ll see what happens. So, I mean, one problem though is, of course, you do still have a database somewhere usually. You have some storage that you have to talk to, and that, at least today, usually isn't decentralized. So you're going to have to make requests back to the database, and if you're making more request to the database than you are communications with the user, then maybe the code should be closer to the database. Really, you want your code to run so that is closest to whatever you're communicating with the most.

[0:13:47.7] JM: Right. Yes. So if I’ve got like the softwaredaily.com, the site that we have, when a user logs in and we want to load the episodes that the user has queued up or that the ones that they've listened to, if we serviced that request from an edge server, the edge server, if it had to make multiple database requests, it would have to make round trips between the edge server and the database, and the database is not sitting on the edge server, because part of this idea of edge computing that we’re going to get into is that you're going to want to put like lightweight things on the edge servers, because this is how these edge servers get to take advantage of economies of scale. This is why they're not big bulky. You're not throwing your entire database on there, although eventually maybe you will.

[0:14:40.1] KV: Yeah, at least for now. I have some ideas around this.

[0:14:43.8] JM: Okay! Then we should probably hurry up and get to this design. But anyway, just continue my example, if you stored that compute logic for — The business logic for servicing the core user request, you would have to make the round-trip between that edge server and the database, or multiple round trips, if you had to make multiple database requests to compose the user’s information, like the episodes they've listened and the episodes they’ve downloaded and then whatnot. And so what you are saying is you would want to keep that kind of business logic closer to the actual database.

[0:15:20.0] KV: Depending on how you do it, but, yeah.

[0:15:22.2] JM: Right. Depending on how you do it. Okay. So I think we've given people a good overview for logic at the edge versus where you would want to put logic on your core server infrastructure. You work on this project called CloudFlare Workers, which is basically the idea is you put this logic at the edge. Can you give an outline for what these CloudFlare Workers do?

[0:15:46.9] KV: Yeah. So the name CloudFlare Workers is derived from web workers, which is a standard feature that browsers have of basically JavaScript that runs in the background, and specifically there's a is a kind called service workers, which is some JavaScript that runs in the background and can intercept all requests from the browser that go back to your server and can like rewrite them and such.

What we did is we said, "Well, we want people to be able to intercept requests to their server when they reach CloudFlare and rewrite them there." But that's basically, coding-wise, the same concept. So we took the service workers standard that already exists in browsers and made it so you can deploy service workers that run on CloudFlare servers instead. So basically you're writing JavaScript that runs on the server, intercepts web requests, can respond to them directly , can call back to the origin server, call back to other servers. Basically do whatever you want, and then eventually reply.

[SPONSOR MESSAGE]

[0:16:55.3] JM: If you are building a product for software engineers or you are hiring software engineers, Software Engineering Daily is accepting sponsorships for 2018. Send me an email, jeff@softwareengineeringdaily.com if you're interested.

With 23,000 people listening Monday through Friday and the content being fairly selective for a technical listener, Software Engineering Daily is a great way to reach top engineers. I know that the listeners of Software Engineering Daily are great engineers because I talked to them all the time. I hear from CTOs, CEOs, directors of engineering who listen to the show regularly. I also hear about many newer hungry software engineers who are looking to level up quickly and prove themselves, and to find out more about sponsoring the show, you can send me an email or tell your marketing director to send me an email, jeff@softwareengineering.com.

If you're listening to the show, thank you so much for supporting it through your audienceship. That is quite enough, but if you're interested in taking your support of the show to the next level, then look at sponsoring the show through your company. So send me an email at jeff@softwarengineeringdaily.com.

Thank you.

[INTERVIEW CONTINUED]

[0:18:22.7] JM: Service workers intercept web requests that are destined for your servers. Explain how service workers are traditionally used.

[0:18:33.5] KV: So one of the major goals of the service workers standard originally was to be able to serve off-line applications. So the idea was that you have like a document editor app or something, think of Google Docs. A lot of this work was done at Google. You want to be able to go on an airplane and keep editing your document, and since it's a web app, traditionally, you can’t do. So what a service worker can do is pre-download all of the resources for that web app and store them in the local cache and then be able to serve them out of that cache, because it intercepts all web requests going to that server. So it can act as like a stand-in server that runs locally. Now, the use case for CloudFlare workers is actually very different, but the code ends up looking similar.

[0:19:23.4] JM: So describe how service workers are used in these edge workers.

[0:19:29.9] KV: Edge workers is actually internal name, but the official brand name is CloudFlare Workers.

[0:19:35.5] JM: Oh, yes. I'm sorry. Okay.

[0:19:37.8] KV: I believe one of our competitors has a product they're calling Edge Workers.

[0:19:41.7] JM: No. Okay.

[0:19:42.8] KV: So in CloudFlare workers, I mean, you write your code in the same way as if you're writing a service worker. So the basic structure of your code is you register an event handler that will be called whenever a web request comes in, and then the event handler essentially returns a promise for a future response and then it can do some I/O. It can make other request, outbound requests in order to build that response and eventually return that response.

[0:20:11.9] JM: Can you give a simple example for how somebody would want to use one of these workers? How they would deploy their code to it? What they would be deploying it for and how those requests would be serviced?

[0:20:25.4] KV: Right. So I gave an example earlier of you want to substitute in the user’s name into some HTML, and you want to do that while being able to cache most of the content to that HTML between users. So you'd write some code that basically the first thing it does is it requests the same — Well, so it receives this request for some URL, some page, and it makes the same request on to your origin server, and so it receives back this HTML that isn't personalized for the user and then it can have some code that reads in the HTML content and does a search and replace for some string where that's a place where you’re supposed to insert the username, and then writes that out.

Other use cases though, there's tons of use cases for this. Other things you might do are like if you want a custom load-balancing policy, so you want to send different requests to different servers depending on some arbitrary signal or you want to serve — Say, you have some resources that are in an S3 bucket, but you have other parts to your site that are dynamically generated. So for the dynamic resources, you want to send it on to your server running somewhere, but for the static ones, you want to just pull them straight out of S3 without bothering your home server. These are all things you can do with this.

[0:21:49.4] JM: Just to set some context, what's the penalty for people who are not using this kind of edge logic today? Are we just talking about latency or does it result in additional costs? What are the savings that you get from deploying more logic to the edge?

[0:22:07.6] KV: So it’s is highly dependent on the use case, but, yeah, in a lot of cases it's latency. The latency of going all the way back to your origin server for something that could have been done with some simple logic at the edge. Some people might use this for — Well, so in the case I mentioned of like I want to serve some resources from S3 instead of from my origin server, you’re saving serving costs, because normally the way you'd implement that is you would have your origin server go fetch the things from S3 when it recognizes that this is a resource that has to come from S3, and that means a bunch more traffic is going through your origin server that really doesn't need to be and it’s going all the way back over the internet to your origin server instead of S3 is replicated all over the world already. So fetching it directly would be faster. So there're a lot of things you can save.

[0:22:57.0] JM: Right. So if this gives programmable logic to these edge servers that might have previously just been used for declarative logic, like you declare that you want your CDN to be used versus the imperative model of actually having executable code that's going to be deployed to these edge servers. Does that change the kind of — Like, for you as the cloud service provider, as CloudFlare, does that change what kinds of things you need to deploy to all these servers that you have throughout the world? All of these edge servers that you have available?

[0:23:39.3] KV: Yeah. We have to deploy the worker runtime, the JavaScript runtime, to all of our machines, all of our edge servers around the world. This is the main project that I work on, is building this core runtime. It’s basically just another background process that runs on all these machines that when we receive a web request that we know has a worker script configured, that it gets forwarded to this daemon. If the script does sub-requests, this outbound HTTP request, then those get forwarded back, back in the front door, basically.

[0:24:16.2] JM: Yeah. This code they gets deployed to your edge servers for logic, it’s in JavaScript.

[0:24:24.7] KV: Correction. So the code I was talking about is actually a C++ program that embeds V8, the JavaScript engine. I was talking about our code. The customer's code, if you're a CloudFlare customer and you're using this, you go into the CloudFlare configuration dashboard and there's a place where you can enter your script and you can preview what the results will look like. But once you deploy that, then that script gets also sent out to all of the edge machines. And so then my code on the edge will load that script as needed and use it.

[0:24:57.1] JM: Okay. Right. To give people a little more clarity, V8 is what's sitting in your Chrome browser, if use Google Chrome. I think it's basically — It's an execution engine for JavaScript, like what is it? It compiles your JavaScript down to byte code and then executes the JavaScript byte code. Is that how it works?

[0:25:19.8] KV: Yeah. It can do a variety of things. So usually it will parse the JavaScript code and it will start out just basically parsing it into some internal structures, which it will run the code like an interpreter would going through each line and doing what it says. But then if someone code is run particularly often, then it invokes the just-in-time compiler, the JIT, which turns it into assembly, or not assembly, but machine code at that point. That's typically how this work. V8 is a little bit more complicated than that, but that's the basics.

[0:25:52.4] JM: I really need to do a show on V8. People have asked for it. Maybe if you some V8 expert, we can figure out who I should interview after this show. I'm sure, since you worked for Google for a while, you probably know a few V8 experts.

[0:26:07.6] KV: You know, I don't know if I met them at Google, but we're talking to them now. So, yeah.

[0:26:12.1] JM: Okay. Very interesting. Tell me a little bit about that space, like the V8 space. Does node - Node must use V8, right?

[0:26:20.1] KV: Node was one of the first interesting uses of V8 outside of Chrome. They said, "Yeah, let's make a server-side JavaScript environment. Let's use V8 as a starting point." But interestingly, node is not designed to run code that's not trusted, whereas the browser is, and CloudFlare Workers is. Like we need to protect ourselves against customers who might want to run code that's malicious. So we do lots of things to prevent that from being a problem that isn’t present in Node.js.

[0:26:54.8] JM: So just to give people more clarity, this is why you don't have Node.js running for this execution environment of the edge logic. You have your own worker environment that's built off of V8. Was it custom-built? You built it yourself?

[0:27:11.1] KV: Yup.

[0:27:11.8] JM: Okay. Could you maybe contrast it with Node.js, because you're describing a server-side execution environment for JavaScript, but it's Node.

[0:27:21.8] KV: Right. Node.js was designed to allow you to write traditional servers in JavaScript. Now in a traditional server, you're writing a bunch of code that belongs to you. You know where it came from. You wrote most of it, and you're running it on your private machine, and so there is no security concern in which you're worried that part of that code might be malicious. Node.js really isn't designed to protect against that, so the code running in Node.js can access the file system, can make arbitrary network requests, can do whatever it wants. Whereas in CloudFlare Workers, we are running code from potentially multiple different customers on the same machine, so we have do a lot of things to make sure that the code, those different customer’s codes can't interfere with each other in any way. That includes like they shouldn't be able to obviously read each other's memory. They shouldn't be able to affect the outcome of each other, and they also shouldn't be able to consume resources to the point where the others get starved. These are all different interesting problems to solve.

[0:28:29.1] JM: Yes, and you did write about this in an article that I’ll include in the show notes called Introducing CloudFlare Workers, and you talked about the design decision not to use containers, and this is interesting to me, because I've done all these shows recently about Kubernetes, and containerization, and scheduling, and serverless, and these things are all related. Like serverless, many of the implementations — Well, actually, the main implementation that I've seen is this idea where you have code that’ sitting in a database somewhere, and it's sitting in a database on a cloud provider, and when a user makes a request for that code, it gets dynamically loaded on to a container and then invoked from that container, and from the outside looking in, somebody might look at the CloudFlare Worker model and think, “Oh! This is serverless. They’re just using edge servers and implementing serverless, and they're probably using containers,” but that's not what you're using. So give a little more explanation for the design decision not to use containers.

[0:29:33.2] KV: Right. As you know, I previously worked on something called Sandstorm, which actually included a custom container engine which I wrote. So, initially, when we started the workers project, that was my first inclination, was, “Well, we’ll give people containers, and that they can run arbitrary Linux programs in that container.”

The problem with that is even though containers are known for being far more efficient than VM's, they’re still not efficient enough, because we have millions of customers and thousands of machines around the world, and we want like every customer’s codes to be able to run in every location all at once.

The overhead of, if each one of those customers had their own process, their own program running, that just — It takes too much memory on each machine and the overhead of switching between processes for potentially every request going to different origins would just be too high. What we need to be able to do is have them all in one process, all on the same address space while still having the strong security, which is what V8 does, which is what V8 was built to do.

So in Chrome, for instance, when you have an I-frame inside of a page and that I-frame belongs to a completely different site, they’re still running in the same process and they’re relying on V8 to make it so that the page in the I-frame can’t attack the page that's framing it or vice versa.

[0:31:07.2] JM: It’s partly security, partly noisy neighbor issues.

[0:31:11.9] KV: Well, the reason for using V8 instead of using containers is performance, is resource usage, is scalability. Well, that and I would say that containers have had some security issues as well, but that's another story, but mostly it's the performance issues of like this way we can mostly implementation underneath each of these scripts. Most of the CloudFlare worker’s implementation can be shared between all of them instead of loading separate copies and things like that.

[0:31:42.8] JM: So why is it that all of these cloud providers — Well, as far as I know, I mean they haven't talked much publicly about how their serverless systems are actually implemented, but I know that this common model is one that uses the container-based scheduling, or at least that's what IBM OpenWhisk is. Actually, that's the only one that I actually can definitively say that's how it works, is this container scheduling model, but assuming that that's how the other ones work, assuming AWS does use containers. That's at least the rumor, I suppose, that I've heard from other people. Are they crazy? I assume that the advantage that they get out of these serverless platforms, like AWS Lambda is, they get these economies of scale where they have all these users running compute, and then so they have these little blobs of compute that they can allocate efficiently, and they do so using containers. Why wouldn't they just do that with V8? What's the difference between the design of a serverless system like AWS Lambda and a serverless system like CloudFlare Workers?

[0:32:47.5] KV: The big limitation obviously is that V8 runs JavaScript. It doesn't run arbitrary code and arbitrary languages. Now, there is web assembly, which opens that up to C and Go and so on, but it's not native code. But to answer this in a different way, and I should caution that like I don't work for these companies, so I don't know exactly what their motivations are, but I think that a lot of the developments in the server infrastructure space over the years have started from bulky server technology that is bloated, that uses lots of resources, lots of memory, takes forever to start up, things like that, all these things that didn't traditionally matter when you had monolithic servers.

It used to be, you have your gigantic server written in Java and it takes 30 seconds or more just to start, and it uses gigabytes of memory, and that was fine. No one cared, because you have all that available to you and it doesn't matter how long it takes to start. But we've been taking these server-side technologies and trying to fit them into smaller containers, if you will, so that we can deploy more instances of them and have them running in more locations at once, and they haven't scaled down very well. It's really hard to take something bloated and then make it leaner.

Now, what we're doing with CloudFlare workers is we're actually starting from the other end of the stack, the browser side technology, and scaling it up. So browsers, for a long time, have the problem that they download some JavaScript and they need to have it running within milliseconds to satisfy the user who wants to see this webpage, and the users is there literally waiting.

So V8 has been designed for a long time to start fast, and that helps us with workers, because it means that we can get your code deployed worldwide within literally a couple of seconds, and we can load code. If a request comes in and your code isn't running yet, we can have it running within a couple of milliseconds, and that's just makes the whole system so much more scalable and it allows us to run your code in more locations at once. So with lambda, for instance, normally — Like, yeah, you create your lambda function, but it's not going to run in a lot of different places in the world at once. It will run in a few locations. It's not going to run in hundreds, unless you get — I mean, maybe if you get enough traffic to warrant it, but [inaudible 0:35:17.7], and we can actually do that.

[0:35:20.0] JM: The other thing about these serverless systems, like AWS lambda, is that they have this cold start issue because the code has to be loaded on to the container and maybe you even have to install some packages and stuff on the container before it can actually service the request. It has to get that the code itself out of the database. If people find this totally confusing, we've done some shows previously about this you can check out. But there is this cold start time.

So with the CloudFlare Worker model where you've got just JavaScript that runs on V8, when a user deploys their code, like if I deploy some custom logic to render a template differently on the fly, for example, some custom CloudFlare worker logic, does that code sit in a place where it can execute immediately or does it sit in a database where it has to be loaded from the database on to the edge worker before it executes?

[0:36:20.3] KV: So there's two steps. There's distributing the code the edge, and then there's loading it for execution. We distribute the code to the edge the same way we distribute any other configuration that it gets worldwide within a couple of seconds. Now, the code is on all of the machines ready to be run when needed. When we actually want to run that code, load it up from disk and parse it and execut it. That takes a couple of milliseconds, because V8 is designed for this. Basically, with CloudFlare Workers, there is no cold start time. It's just ready to go.

[0:36:51.6] JM: I see. You mentioned web assembly. With web assembly, you could actually run any language, because web assembly is this idea that — Web assembly or asm.js, where you have a subset of JavaScript and every — Like any language could theoretically compile down to this subset of JavaScript and then the subset of JavaScript runs extremely effectively on V8. Could you just remind us what web assembly is?

[0:37:18.7] KV: Yeah. What you were just describing was asm.js, which was a previous attempt at a similar idea. Web assembly is actually a new binary format that they said, "Okay. This whole idea of like encoding assembly in a specially constructed JavaScript turned out to be kind of weird." So they went ahead and defined a format is like a machine code except it's not specific to any other machine, and it's designed to load within the V8 sandbox or within any JavaScript sandbox. You can target a traditional compiler at these, so you can compile C++ code to web assembly and you get, instead of an executable binary that runs on your system, you get a web assembly binary that you can load into a browser, and other compiled languages, like Rust or like Go can all target web assembly.

[SPONSOR MESSAGE]

[0:38:20.7] 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 off-line. 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.

Check out the Azure Container Service at aka.ms/acs. That's aka.ms/acs, and the link is in the show notes.

Thank you to Azure Container Service for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:39:47.8] JM: So some people are really excited about web assembly and I have trouble identifying what the process of web assembly becoming big or, I guess, having a noticeable impact on the way that we consume the internet will unfold. But, for example, my friend Pete Hunt, he tweeted something a while ago about just all of the things that you can do on web assembly and I just didn't really understand it. It made me want to do more shows on it, but could you give me some context for why people are excited about web assembly? Where we are with it? How it can change things in the future, or is it an open question as to whether it's going to change things?

[0:40:30.5] KV: I can give you my take, but people will disagree. I think web assembly is mainly going to be important to people who need to write extremely high performance code like game, things that where you can't have garbage collection pauses every now and then because it will harm your frame rate, or use cases where you have existing code that's written in C++ that you need to be able to run in these environments. These are all use cases where web assembly makes a big difference.

I don't see a future where everything is written in web assembly, because, frankly, JavaScript is good enough for most application bubble use cases. I think there're a lot of people who are excited about web assembly, because they hate JavaScript, and my take is, yeah, the languages is — It has its warts, but it has improved a lot over the years, and this point and given that everyone knows JavaScript and it works well for most use cases, I think it's here to stay and most things will continue to be written in JavaScript with web assembly for optimization, basically.

[0:41:43.8] JM: So if you want to use web assembly, what do you have to do? Do you have to write some specific stuff for it? Like if I want to write — If I want my C++ application to run on the web, what will do I have to do for that?

[0:41:59.1] KV: Yeah. So first you have to compile it with the compiler that supports targeting web assembly so that you get the web assembly binaries and output. The other thing is you need to code against the different set of APIs. You don't have your traditional operating system. You don't have like a file system that you can open files from any more. In fact, you have almost nothing to code against and you have to often — I should caution that I don't have a whole lot of experience spreading things in web assembly, but my understanding is you have to implement a shim in JavaScript that like provides the functionality that's needed so that you can then call that from web assembly.

The HTML DOM API, how you would normally manipulate the website from JavaScript, mostly is not available directly from web assembly, because taking that whole API and importing it into a different language and creating those findings would just be actually very complicated. Usually, instead [inaudible 0:42:58.9] assemblies use for is like heavy number-crunching or I believe you can call OpenGL graphics rendering from it, because obviously the gaming use case needs that.

[0:43:11.5] JM: Okay. Let’s, I guess, get back to the main topic from web assembly. Now, that gives me two shows I need to do related to JavaScript. So let's talk a little bit more about these use cases for these CloudFlare Workers, and then we'll talk a little bit about the future, because you mentioned that you've got some plans or some intentions or ideas at least. But to reframe this product/idea for people, can you describe some of the other use cases?

So we’ve talked about expanding HTML templates dynamically. We’ve talked about some caching related things, but I know there's also, for example, parallelization. A user request could come in and your service worker that your CloudFlare Worker, service worker, is positioned to take the request and intercepted it and then parallelize it. So maybe you could talk about parallelization.

[0:44:06.2] KV: I think what you're referring to there is like if you're your page has several different components that need to be loaded and assembled together. Is that right?

[0:44:16.1] JM: I was actually just talking about something that is saw referred to in your blog post. I wasn't exactly sure what you were referring to with the parallelization, but I could see what you just described as being that's parallelizable.

[0:44:28.1] KV: Yeah. That might've been in the — I don't remember exactly what you're referring to. But it might've been in the section comparing against. There are some other frameworks for programmable proxies where they don't really allow you to do several sub requests, we call them, in parallel, so that those are the outgoing requests back to your origin server or maybe to S3 or maybe to some other API.

With CloudFlare Workers, you can do as many [inaudible 0:44:54.0] requests as you want. You can do them in parallel, fire off several of them, then wait for them to come back and then assemble your result from them.

Like one use case that we've seen is someone has a GraphQL query that comes in from a client, and GraphQL queries is often a bundle of smaller queries, and in their case each of these smaller queries is independently very cacheable, like often requesting the same things over and over, but the bundle together often contains different things depending on which client is asking. So the overall request is not very cacheable, but the components are. So what they want to do is split it up and do these requests in parallel and have each one come back directly from cache so that they can get a much better cache hit rate.

[0:45:42.2] JM: Right. Yeah. I was going to say this sounds exactly like — It sounds very related to GraphQL, which is this system for federating requests to multiple different data sources. If you have a GraphQL server in the typical model of GraphQL that interprets a high level request that requires several different resources and its federates that request to all the different servers that might need. You've got like a user request that requires the user's favorites and the user's likes and the user's comments and the user's friends, all of those things may be in different databases, and so GraphQL queries might contain a query for all four of those different things and then your GraphQL server can federate those requests out to those different database servers such that the person who's programming against GraphQL doesn't have to think about hitting all of those individual servers. They can just think of one GraphQL query.

[0:46:48.7] KV: Right, and graphical federation server, especially if all of the separate databases being queried are in different locations around the world, having that federation points be something that runs on the edge makes a lot of sense.

[0:47:02.7] JM: You're saying you can do that with — I guess because you can just — People just run these things in JavaScript, so you might as well keep your GraphQL server on the edge.

[0:47:12.0] KV: Yeah.

[0:47:13.4] JM: That's great. That sounds very useful. Do you know of people who are doing that in production?

[0:47:18.5] KV: Well, so this is all very new. I don't know if anyone's doing specifically that in production yet, but we do have a customer who wants to do something to that effect. By the way, CloudFlare Workers is — Currently, our state, is we have a small number of beta customers testing it out and we plan to open it up to more people in a month or two if things go well, but we have to — This is something we have to roll out very slowly, because like if we just open the doors and let everyone in on day one, like this is a big change star network. So we have to be careful, but we're getting there.

Just to comment on other use cases. Basically, we found the most interesting use cases are things that we didn't anticipate. We come up with these simple cacheability use cases and such, but then we keep seeing customers who come to us with really interesting things that we never would've imagined, and I can't actually like go into a lot of detail about them, because that's private to those customers, but the possibilities are basically infinite, and we're just really excited to see what developers come up with.

[0:48:23.9] JM: I'll explore one thing that I would be surprised if your customers are not coming to you with this use cases, the machine learning model edge deployment, because we've had a number of different shows about this where people have commented on how cool this could be or how cool this is, where basically you keep your machine learning models closer to the edge, because the user requests the model. Actually, machine learning is data intensive, but the data intensive and compute intensive part is the training process, not necessarily the serving process. Also, the model, you can have models that don't take up a huge footprint. So somebody could be having their machine learning model sitting at the edge, serving user requests, and then it just gets updated over time periodically, because the computation is running elsewhere. The computation of retraining the model is certainly not happening at the edge. Are you optimistic about the machine learning applications?

[0:49:21.6] KV: Yeah. That’s one of many things the we’ve thought about and we’re excited about. Haven’t done a whole lot of experimentation with it yet, but it looks promising.

[0:49:30.5] JM: Yes. So earlier in our conversation you mentioned that you saw a potential for more stuff being pushed to the edge, and when you said this, I was thinking of — This is not entirely related, but this serverless database that Amazon came out with recently. I don’t know if you saw it. Did you see this thing? The serverless Aurora database from Amazon?

[0:49:54.0] KV: I haven’t looked at it. No.

[0:49:55.4] JM: Okay. Well, it was — I don’t understand a whole lot about it. Basically, the idea is just that it's a database that scales up and down, and I think the reason people are excited about it is it's a database that can take up a small footprint. If you've got some service that occasionally gets very spiky workloads and needs to request a lot of stuff from the database, you want your database to be able to scale up, but if that only happens for one hour during the day, you don't want to be paying for all the extra database nodes that it scales up to. You just want to pay for like the minimum single database instance or something like that, and you could imagine having that kind of serverless database on the edge where it takes up a small footprint and then maybe you keep replicas away from the edge. I think this is essentially what we’re talking about. We’re talking about what are the limitations of the edge servers, is like can you get your database to the edge? Am I portraying things properly?

[0:50:56.4] KV: Yeah. The really interesting problem here is — As I said, CloudFlare has hundreds of locations today. We’re planning to have, few years, thousands of locations around the world, but we don't want developers to have to think about what locations their code is running in.

When you have a database, you don't think about where it is. You don't think about what data is stored where. You just have your data. If you're querying that potentially from thousands of locations around the world, there's some synchronization costs that goes into that, and most database designs are fairly centralized where you're going to end up doing your database requests all the way to the central location where it can process transactions relative to all the other requests coming in, but I don't think that design takes proper advantage of the network that we’re building.

So we’re thinking about what kind of storage would allow your code to have fast storage access no matter where it's running, or like be able to move the data around the world so that it's already there and ready when it's requested in a particular location. We’re still at like — This is still in the sort of thinking phase, but it’s this really interesting big problem, and what I'd like to see in the long run is people don't think about where their code runs or where their database is. They write the code and it automatically moves to be close to whatever it is that it's interacting with. If you have code that’s interacting with the user a lot, that code and the data, maybe the data that belongs to that user all automatically migrates to be in the closest CloudFlare location to the user, and if you have a code that interacting with, say, the Stripe API a lot, then it should automatically move to be sitting in the same AWS data center that Stripe runs in. I think they use AWS. I’m not sure. So that it's right next to those servers and you get fast access to that, things like that. As an application developer, you really have no reason to think about having one central server of your own. It's just your code runs wherever it makes most sense to run.

[0:53:09.1] JM: Do you have a vision for how long it'll take to get there or how much progress has been made towards that vision? Because that’s a lot of complicated scheduling that would have to go on.

[0:53:20.5] KV: Yeah. Well, as I said, we’re pretty early in thinking about this, but we have we few promising ideas, but it would be really hard to say at this point how long that's going to take to pan out.

[0:53:32.2] JM: Fair enough. Well, I think that's a pretty good place to wrap up. Is there anything else you're excited about right now? Like maybe using CloudFlare Workers for block chain stuff or something. Any other closing thoughts on where this space is going or your craziest ideas?

[0:53:50.9] KV: Working on a sandstorm previously, a problem we were trying to solve was this data privacy and data locality and dealing with regulations that exists around the world that, say, like your data must remain within these national borders or must not enter these national borders, and I think this is a problem that has been largely ignored by cloud services, software as a service industry up until this point.

I think a lot of people have just assumed that like these kinds of rules and these kinds of concerns would just go away at some point as everyone realized, “Oh! Cloud services are so great. Do I really care where my data is? Do I really need physical control over my data?” I don't think that's actually going to happen. I think these laws and these concerns all exist for very legitimate reasons and the exact rules might change a little bit to account for technology, like account for the fact that if you're encrypting your data, then what really matters is the location of the key, not the location of the encrypted bytes. But I don't think the fundamental concept is going to go away.

What I really like to do is make it easy for a startup. Like Google has servers all over the world. If Google needs to guarantee that your data doesn't leave the borders of Germany, it can actually do that, but if you’re startup in San Francisco and you're just a few people, this is like an intractable problem and you end up saying, “No. Sorry. If you have these rules, then we can’t sell to you.” What if we could make this easy for users to be able to say, “I want my data to remain inside my country, or I want my data to remain actually on this device that I buy from CloudFlare or someone else that runs — That takes the code from the cloud and brings it here and then runs it here so that my data stays here even though I get the experience of using a cloud service and never having to update and so on?”

This is a problem that really interests me and that I hope we’re going to be looking into.

[0:56:08.4] JM: This might sound boring to some people, but I've had a few conversations about GDPR recently. What is it? General Data Privacy Regulation or something, the whole UK thing, and I need to do shows on this as well, but this is like if you do not adhere to the policies of the UK in accordance with data privacy, then it can cost you like a quarter of your revenue in the UK or something. They have some insane fine that they charge you if a customer can prove that you violated the privacy rules, and those privacy rules are really, really hard for a cloud service provider to implement, especially if they've been operating for years and years and years and years without paying attention to GDPR. Now, all of a sudden, they've got to implement all these different things in six months or else they could potentially lose a quarter of their revenue for the year in the UK. I can see why you would want to focus on this.

[0:57:08.6] KV: Yeah. My understanding is that's not just UK, but general European regulation. I might be wrong. I don't know the exact details, but this is exactly my point, is that like Europe has always had a different opinion about privacy than a lot of the United States had had, and they're not just going to change their mind. In fact, the regulations are — We see them getting stricter. We can solve this problem through technology that makes it easy to keep data where the users want it to be kept instead of just burying our heads in the sand and saying, "Well, the users will come around and eventually let us do whatever." Yeah, I hope to see more this happening and I hope to build some of it.

[0:57:54.7] JM: All right, Kenton. Great talking to you, as usual, and your last episode was really popular. It was fantastic, so this was no surprise that this one was a good one as well. So thanks for coming on the show.

[0:58:06.2] KV: Thanks for having me.

[END OF INTERVIEW]

[0:58:11.2] JM: GoCD is an open source continuous delivery server built by ThoughtWorks. GoCD provides continuous delivery out of the box with its built-in pipelines, advanced traceability and value stream visualization. With GoCD you can easily model, orchestrate and visualize complex workflows from end-to-end. GoCD supports modern infrastructure with elastic, on-demand agents and cloud deployments. The plugin ecosystem ensures that GoCD will work well within your own unique environment.

To learn more about GoCD, visit gocd.org/sedaily. That's gocd.org/sedaily. It's free to use and there's professional support and enterprise add-ons that are available from ThoughtWorks. You can find it at gocd.org/sedaily.

If you want to hear more about GoCD and the other projects that ThoughtWorks is working on, listen back to our old episodes with the ThoughtWorks team who have built the product. You can search for ThoughtWorks on Software Engineering Daily.

Thanks to ThoughtWorks for continuing to sponsor Software Engineering Daily and for building GoCD.

[END]