EPISODE 867

[INTRODUCTION]

[00:00:00] JM: Edge computing allows for faster data access and computation. When your client application makes a request, that request might be routed to the edge. Edge servers are more numerous and widely distributed than normal data centers. But an edge server might not have all of the data or the complete application logic for the backend to serve your request.

Edge servers have historically been used for content delivery networks, or CDNs. CDNs are useful for hosting and serving media files that might otherwise be slow to access over a network. More recently, applications are also using edge servers for computation as well as storage of resources, which are smaller than the movies and music and images that have traditionally been stored on an edge server.

Steve Klabnic is an engineer with Cloudflare, and he returns to the show to discuss storage at the edge. In Steve's previous appearances, we have explored Rust and WebAssembly, and we also touched on those topics in today's episode.

The SEDaily app for Android is finally ready. It is either on the App Store and will be on the App Store very soon by the time of this episode airing, and it is a well-built application to find all of our episodes. You can comment on them. You can post related links. You can find additional content around them, and we also have that app for iOS. It's been in the store for a while.

The other announcement is that the FindCollabs Hackathon is still going on. That ends later this month. FindCollabs is a company I'm building. It's a place to find collaborators and build projects. There are many people who are posting their open source projects and finding collaborators on there. So, if you're looking for a good excuse to start a project or to share a project that you already have with other people, take a look at FindCollabs, findcollabs.com, and check out the FindCollabs Open, which is our hackathon with $2,500 in prizes. You can go there by going to findcollabs.com/open.

With that, let's get on to today's show.

[INTERVIEW]

[00:02:22] JM: As a software engineer, chances are you've crossed paths with MongoDB at some point. Whether you're building an app for millions of users or just figuring out a side business. As the most popular non-relational database, MongoDB is intuitive and incredibly easy for development teams to use.

Now, with MongoDB Atlas, you can take advantage of MongoDBs flexible document data model as a fully automated cloud service. MongoDB Atlas handles all of the costly database operations and administration tasks that you'd rather not spend time on, like security, and high-availability, and data recovery, and monitoring, and elastic scaling.

Try MongoDB Atlas today for free by going to mongodb.com/se to learn more. Go to mongodb.com/se and you can learn more about MongoDB Atlas as well as support Software Engineering Daily by checking out the new MongoDB Atlas serverless solution for MongoDB. That's mongodb.com/se.

Thank you to MongoDB for being a sponsor.

[INTERVIEW]

[00:03:43] JM: Steve Klabnik, welcome to Software Engineering Daily once again.

[00:03:47] SK: Yeah, thanks so much for having me back.

[00:03:49] JM: There is a term edge computing. What does that term mean to you?

[00:03:55] SK: Yeah. So, I sort of see this as like a really interesting development in the way that you build web applications. So, first thing is we're talking about the web, and there's been a lot of change especially for those of us who've been around the web for a while. I kind of cut my teeth on some PEARL and PHP back in the day, and it became Rails, all these sort of developments.

But edge computing is sort of a shift in where the computing actually happens. So, kind of like the super rough overview as I see it is like we used to have servers and then we started renting servers from somewhere else, and edge compute is about having those servers be located all over the world instead of Northern Virginia. There's a lot of interesting things that happen due to that shift, but that's kind of like the basic idea, is like what if your compute didn't happen in one data center. What if it happens everywhere, all around the world simultaneously?

[00:04:43] JM: One way that edge computing is manifesting is in the increase in computation at the edge. We have used the edge for storage with CDNs for a long time. Why haven't we used the edge for computation?

[00:05:00] SK: So, I think it's a combination of a couple of different things. The first is, that I'm not sure anybody really have thought about this for a while. There're lots of good ideas in computing that could have existed earlier, but didn't because nobody had the idea to make it happen. But I also think that there is – It's a little more interesting when you dig in to the actual architectural constraints, because one of the reasons why we have data centers, and you have to decide where your stuff runs in the data center, it's like imagine you're on the other side of the fence. Think about if you're the person that runs these servers. Do you want everyone's code to be in all of your servers all the time? That like increases your costs significantly, because you are putting people's stuff in more places.

As you get to the outer parts of the internet, we like to think of the internet as being hyper-reliable. But once you start getting farther away from the center, things get a little dicier and it's a little harder to get things going. So, there's lot of like challenges both in actually provisioning the physical aspects of the internet. But then also the cost aspects and all these other kinds of things. I think those things have all had something to do with why this is not really a thing that's been happening until pretty recently.

[00:06:10] JM: We've got computation at the edge now, and we will talk about how and why that is implemented in a special way. But let's first talk about the applications. What kinds of applications would we want compute at the edge for?

[00:06:27] SK: I think there's sort of two major divisions here. The first is that you can build something that runs entirely at the edge, but that's using a lot of brand new technology all at once. But also a lot people are doing things where they split some parts of their stuff to the edge and some parts that are not. That's definitely – For anything that's brand new that you could call mature, it's like the more mature option.

For example, it's like authorization and authentication. If you can detect if your user is like allowed to do the thing they're trying to do sooner, then you can give them a rejection notice faster, which can also take load off of your central servers, because their request never gets past the edge of the internet. Like hybrid stuff like that that's being done.

[00:07:07] JM: And then what about something like custom responses, like maybe if I want to keep a machine learning model at the edge for each of my users, or perhaps if I want to have some kind of spam detection system that's at the edge so I can process that kind of stuff faster. What about these kinds of applications?

[00:07:28] SK: Yeah. So the tricky part about this sort of stuff is that I think the spam one is slightly better than the sort of AI or like ML kind of applications, and that's because the data access part is harder. So, pure computation is much simpler of a thing. Once you start getting needing to access like on data, not just doing pure compute. That's where things get a little bit trickier.

However, it is totally true that like doing stuff like spam filtering can work. I guess another sort of similar aspects, like anytime you want to route someone to one thing or the other based on some of sort of things, like AB testing, for example. Do the cohort analysis at the edge and then point up the central server that way, or some people are doing stuff like blue-green deploys, which all sort of boils down to this are you in person set A or set B? Maybe that spam goes to something to slower and worst and like not spam goes to something faster and better, like all sorts of things like that are things we were doing.

[00:08:19] JM: You recently joined Cloudflare to work on storage at the edge. We've had CDNs for a longtime, and CDN is a kind of storage. I can store my media files at the edge, my photos, my videos. Videos are a lot of data. That's a kind of storage. Why do we need something new

for these other kinds of applications that we're putting at the edge? Why do we need new storage primitives for edge computing?

[00:08:51] SK: So, the first one is that those kinds of applications are almost purely read only. You're fetching something and you're not like really updating it. You're just consuming it. So, being able to like purely read, it's much, much easier to distribute things geographically if they're not going to change. So that's like sort of the first thing. In this way, if you think about the way the CDN works, it's like a pass or cache, right? So, like check on the edge to see if it's stored or not. If it is, then you get it superfast. If it's not, it hits your origin and it pulls in, updates the cache and like that kind of thing works.

But if you would to start being able to build applications, where like you're doing more significant things, where you're writing as well as reading. That's where stuff starts to get tricky. Because it's kind of funny. If you have your code and your data, like collocated in a data center somewhere, then like the latency between your application and your database are going to be small, because they're next to each other.

So, when you try to optimize that transfer time by moving your code closer to the person that's actually like seeing the thing, you've [inaudible 00:09:54]. But now you've introduced the same amount of latency to your database if it's still sitting in some sort of centralized location. So the trick is like how do we let people read and write from the edge as supposed to always needing to go back to some sort of like centralized store? That's like sort of where things are extremely up in the air and are much more interesting.

[00:10:14] JM: If I understand you correctly, you're saying that in the world of CDNs, the CDN is a kind of a cache where a user might request a movie. If that movie is in the CDN already, then the access time is fast. If the movie is not in the CDN, then the request will go all the way to the central servers and the user will get that movie. Then, also, the CDN will then populate the cache with that movie.

But what's different about the edge storage that we're talking about is you might want a user to never have a cache miss, which means that you've got to pre-populate your edge storage

system aggressively with the writes that are being made to that edge storage. Am I understanding you correctly?

[00:11:08] SK: Yeah. So like to maybe putting it in more concrete terms would also be helpful. So, right now, I am in London, but normally I would not be there. So maybe like, say, a coworker of mine is in Texas. So we're building an application together. Like maybe, say, we're editing a shard document, like in a Google Docs style situation. If the code is running in Austin and the code is running in London, then that still doesn't like – I mean, it helps a little bit, but it doesn't super significantly decrease the amount of time that it takes to get to the end user if we're both connecting to S3, which is stored wherever, somewhere in the states. I still have to go over an ocean. They may have to go somewhere else within the U.S.

So, the trick is like is there a way that you can make it so that we can both be editing these things where it doesn't necessarily have to go the whole way back initially and/or can we maybe pick an area that's like good for us. Whereas if, say, two other people are adding a document and one of them is in like Australia and the other one is in, say, Malaysia. They are doing their writes somewhere that's closer to them, not necessarily closer to us. So you start talking about how do you reconcile all these things and like all that kind of like trickiness, basically.

[00:12:17] JM: So there are these data types and these compute types that we want to do at the edge, but we may not get the same consistency properties from this edge computation and storage that we get out of traditional database models. Could you explain why that is and how that might affect what applications we choose to use at the edge?

[00:12:48] SK: Yeah, totally. So the first thing that Cloudflare is doing on storage is a thing called Workers KV, which I'm now the product manager of. The thing about – It basically gives you a very simple key value store that works at the edge like this. However, we've also been very upfront. It's sort of like a glorified cache. The exact same thing that you said about CDNs is basically how Workers KV works.

So, like reads are stored locally and you'll tend to get them very fast. Then writes still have to be reconciled somewhere centrally and then pushed outwards. So, this is really good for like a

read-heavy sort of user flow. But like not very great if you need to do a ton of writes. So, that is good for many applications, but it's definitely not good for like all of them.

So, a lot of this work is kind of like research and development. I'm not telling you that I instantly have a super-secret amazing way to make this stuff kind of all work out. But I think that the motivation is very interesting and the problem we face is very interesting. So, that's kind of like future work to do something that has like better consistency models. Something we're sort of like actively investigating and working on and that kind of thing.

[SPONSOR MESSAGE]

[00:14:03] JM: ExpressVPN is a popular virtual private network. ExpressVPN is useful for getting a private, secure, anonymous connection for your internet browsing. It encrypts your data and it hides your public IP address. You've got easy to use apps that run seamlessly in the background on your computer, or your phone, or your table, and turning on the ExprsesVPN protection only takes a single click.

If you use ExpressVPN, you can safely surf on public Wi-Fi without being snooped on or having your personal data or your Bitcoin account information stolen. For less than $7 a month, you can get ExpressVPN protection.

ExpressVPN is the number one VPN service rated by TechRadar, and it comes with a 30-day money back guarantee. You can also support Software Engineering Daily if you check out expressvpn.com/sedaily. You would get three months free. Everybody needs at VPN at some point in their life. So if you want to get your ExpressVPN subscription for free for three months while also trying out ExpressVPN and supporting Software Engineering Daily, you can kill all those birds with one stone by going to expressvpn.com/sedaily.

Thanks to ExpressVPN for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[00:15:43] JM: This key value storage system. Go into a little bit more detail about what is new about this. If the properties are similar to storing media files in a CDN – Well, I think of media files as heavier than files that would be needed for authentication or small data transactional workloads. What's new about this system?

[00:16:11] SK: Well, I mean, in some sense, nothing is ever totally new in computing. Ultimately, there are some bytes on a hard drive somewhere and you're trying to get the bytes on a hard driver somewhere else. I'm not claiming this revolutionizes computer science or anything yet. However, it's always about the affordances that are offered, right?

So, like, the way that we've interacted with CDNs and the APIs to sort of make that happen are not really like tuned for doing this kind of thing. They don't feel like a data store. Whereas this is kind of more like a slightly different take on the actual interface and like gives you the ability to sort of manipulate these things in a way that like gets you thinking about it like a more traditional data store rather than just a CDN.

Because you could just say, "Here's this huge chunk of byte stored somewhere and I'm going to like read it and then mess with it and write it back," but you would have to implement all that logic yourself. These are not things that a CDN provides you sort of like natively. In this case, you can just think about this as like we've done all that kind of work for you. Somebody else could write this kind of system. It's just we've already done it.

So, there're those kinds of like aspects to it. I think, really, it's also about slowly enabling people to build these kind of applications and seeing where the weak points are and then improving stuff from there. It's not something where my team is going to go into the woods for a year and come out with something like amazing and wonderful. We got customers who are like, "Okay. I'm using this thing. Here's how it's awesome. Here's what I wish was better." Then you're like, "Okay. I think we can fix that problem," and kind of like move on from there, right? So, it's very much like it's partly research but it's also partly just like watching how people build stuff, because people do things you never expect.

[00:17:54] JM: Agreed. One thing that I guess a point I would make about this space of edge computing and the changes to how we want to interact with the edge is it's a space that's hard

to understand or even think about in terms of where the technology and where the use cases are at today. I think it's something that is easier to understand if you just spend time in the industry and you have a sense of – There's going to be an increased desire to do things faster. Ultimately, in order to do things faster, you need to interact with the server faster. The server needs to be closer to you, and a CDN is a server that is closer to you. But a CDN has different infrastructure constraints than a central data center. Is there some distinction we can draw between like what re the constraints of a CDN as a piece of server infrastructure versus a fully-fledged AWS data center?

[00:19:03] SK: Yeah. So, one of the things – And I guess this is sort of what you're also talking about. But a way to think about this is like the speed of light is the enemy. We like to think about the internet as being kind of like virtual and where geography doesn't matter. But if you've ever like visited Australia and tried to access a U.S. website, then you know that there's like a physical distance and you could only get bytes down the wire. We do run into physical limits when you're talking about going all over the world.

So, like one interesting thing about sort of the CDN space as supposed to having a central server is that things need to be much more lighter weight in order to fit everybody's stuff everywhere. So, like if I want to be storing people's stuff all around the world, then I need to be able to have that space all around the world. So, that requires like being more efficient about what goes where and like figuring out how these things work together.

So get in a one sort of more technical aspect of the way that like the workers runtime works is that we currently support like JavaScript and WebAssembly, because the workers runtime is built on top of V8. So, workers is like the compute part of the platform that I work on. Like I work on the database part, but the compute part is just workers.

So, part of the reason why we did that is because, basically, V8 has this concept called isolates, sort of like lightweight threads within V8 itself. So, when you're deploying to say like AWS, every application – Say you're building a bunch of node applications and you're putting them on AWS. Other people are putting node applications on AWS.

There, you have a whole lot of space and you have a whole lot of compute. Do you can afford to let everyone have their own copy of node. I'm sure there's like millions upon millions of different identical copies of node running inside of EC2, right?

But like what we need to do is save space and be able to fit everyone's stuff all over the place. So, we run like the runtime as part of our infrastructure. So you can't just run arbitrary code. You basically are running our code on top of V8. The reason that it happens is because we need to be able to share that infrastructure across all these applications in order for like the economics to work out. Because you can't afford to fit a complete full U.S. East one size data center around the world all the time. Even Amazon doesn't have that much money. So, these kind of like economic aspects also inform the technical aspects too.

[00:21:30] JM: I've had this point come up in discussions around the cold start problem for functions as a service, because the cold start problem for serverless applications occurs when you make a request to execute a serverless function. Let's say it's a node.js function. In order to load that serverless function into an environment that it can actually execute in, you need to load the node.js runtime. If you start up a server, often times a container, with the node.js runtime, that might a little bit. Then you have to load your function into it and then you actually have to execute it. Then you have this dedicated server with node.js already installed in it. The cloud providers, if they can allocate another workload to that same server, another workload that is node.js compliant that is a node.js function, then they can get to reuse that server infrastructure and they can avoid some of the cold start penalty. That's a similar thing to what you're describing here, except instead of a container, it's a WebAssembly system, I guess.

[00:22:45] SK: Yes, or JavaScript itself. Basically, I'm not going to say that things start instantly, because, of course, it never starts literally instantly. But we don't have cold starts the same way that Lambda does, because the V8 instance and all of the associated machinery are just always running. There's no need to like boot up. It's just about like instantiating your particular program. There's not like the booting of the entire world first and then instantiating your program.

So we tend to be able to start stuff up like very, very quickly, because that aspect is just running all the time. Then sort of the homogeneity of the fact that we're basically just like purely running

V8 and nothing else, if there's no container support. That's like part of what makes the architecture work.

[00:23:27] JM: We've had a show about Cloudflare workers. Can you just describe in brief how that runtime works. Are the users who are creating these functions that are going to run at the edge, are they creating all of them in JavaScript and then deploying them to Cloudflare workers. Is that the runtime model?

[00:23:49] SK: Yeah. So, either JavaScript or WASM. So, by now, this is like a pretty new thing. So it probably was not available whenever you had your last show on this, but there's this command line tool called Wrangler and it lets you basically like package up either your JavaScript program or if you're using Rust, or C, or C++. It will compile it to WebAssembly and package it up and ship it off to Cloudflare for you. Then, you have that running in a particular place. That's a way to configure what your outlet runs on and all that kind of shenanigans. But, yeah, fundamentally, you're writing either JavaScript or some things about the WASM and you're like giving to us and then we're running it.

[00:24:26] JM: In a post that you wrote, I saw you discussing the distinction between a WebAssembly runtime that supports JavaScript and one that does not. Could you describe that distinction in more detail? Why is that important?

[00:24:44] SK: All right. So, I love WebAssembly. Don't get me wrong. It's still early days as a technology. There's lot of rough edges and people are still figuring it out. It's only been a little over a year that it's even been generally available in browsers. Let alone, the long-tail of browsers that they leave. So, while I'm like psyched that we're supporting WebAssembly, because I think it's important. Ultimately, JavaScript is the most popular programming language that exists. Even if you're not a "JavaScript" programmer, you probably know a little JavaScript. You probably used it in anger from time to time.

So, you sort of have this like choice about like is it worth supporting JavaScript or not. So, if the answer is yes, which is like what we think at Cloudflare, then it makes sense to reuse an existing JavaScript runtime, because they also tend to support WebAssembly. Thanks to WebAssembly being part of a browser technology. If you don't want to support JavaScript, you

can build your own WebAssembly runtime, which maybe a little more like strictly customized to WebAssembly specifically. But there's no current way to compile JavaScript to WebAssembly. So you kind of like – Even though in theory that is a thing that could happen someday. It does not exist yet, definitely not in a production capacity.

So, what I've seen in this process of edge compute is people are basically choosing one of those two paths. If you want to support JavaScript, then it makes sense to – Google is not exactly abandoning V8 anytime soon. They pour so much time and effort and resources and they fantastic engineers working on V8 to make it awesome. So, like that means that it makes sense to go with V8. But if you don't, then like maybe you want to do fancy custom things that are like WebAssembly specific. Some people are also like choosing that route. I think it's going to be kind of interesting to see what those choices mean in the future. Maybe you don't actually need JavaScript to be successful. I found that like betting against JavaScript is generally not something you want to do personally. Even though I've never really been a person who writes or even particularly likes JavaScript, it somehow always wins in the end.

[00:26:51] JM: It does. It does have a sense of inevitability. Have you seen people use Cloudflare workers to do server-side rendering?

[00:27:01] SK: Yeah. So, hilariously, like my personal website is actually sort of this. I mean, I guess it's not in JavaScript. So it's like it depends on what specifically you mean by server-side rendering, but just like, for example, if you are willing to live with the constraints that I mentioned earlier about KV, then you can build sites where there is no origin and it runs entirely on the edge.

So, like I have the contents of my website stored in KV, and then it gets rendered entirely in a worker by pulling and doing all that kind of stuff. So, that's kind of like an interesting thing. I have seen people do like React server-side rendering things as well, and that's definitely like something that I like. I've seen some of our customers start to do that. Let's put it that way.

[00:27:44] JM: The reason I asked you about server-side rendering, is I did a show a couple of weeks ago with a company that has a lot of user-generated content. The way that they use server-side rendering, is there's a very late binding between what the data model in a user's page request is and when that page gets rendered by React. The reason this is important, is because the way that React works to the extent that I understand it, is you have this templating language, and the templating language accepts some data and together with the templating language and the data, there is interpretation step where that React model gets turned into HTML that the user can actually display on the page. If you do that on the server, then you can just send the fully computed HTML page to the user and then the user can just render that page. Whereas otherwise you might be sending the – If you did client-side rendering, you might be sending the React file together with some JavaScript packages that cause the user to do all these computation in their browser and then they might have to do data fetching as well. It can be much slower.

So you can really accelerate this by putting it on the server, and this is especially useful for a system with all these user-generated content. Because users are updating stuff a lot. So then you might want those updates to be pushed out to a key-value store at the edge. So, I just find this to be a great subtle – When the person on the show told me about this, this was an episode about gaming, if people are curious. It's called Gaming with Eli Brown is the title of the episode. But it was something I hadn't heard before. I had heard these examples, like the ones you mentioned, like the authentication example, or perhaps I'm doing like AB testing, or maybe I want different users to see different versions of ecommerce website. So they're more tempted to buy their shoes or buy a dress, depending on who they are. But I hadn't heard the server-side rendering use case. I thought it was a good example of how these technologies often they get built with some set of use cases in mind. But more generally, a fundamental, like you said, the speed of light thing, fundamental notion of computing that's going to be important. Then we see perhaps even more applicable, more relevant use cases that we didn't expect.

[00:30:27] SK: Definitely. I think this also is like part of – Okay. As you said, you've heard of some of these examples before, and that's because I think they are the ones that people have talked about and they're a little easier to get people's heads around. But, your example reminded me of another interesting related example, because in some sense, it is a server-side rendering and sometimes it's not.

But I've seen people do a thing where if you request a page in an older browser, it will include the polyfills and the JavaScript [inaudible 00:30:52] sends back to the client. But if you make the request in a newer browser, they will serve the smaller file that only includes not the polyfills. Therefore, you're able to support – Like we think about like, "Okay, I want to support old browsers and new browsers."

So the new browser people end up paying the cost of transferring and interpreting and compiling all that JavaScript. But if you've been dynamically sort of serve – Again, it's sort of like dynamic and sort of static. Because if you have both of those – Say, you pre-render like, "Okay, I have a bundle that supports everything back to IE6 or whatever and I have a bundle that only supports latest edge Chrome and I have them – They're pre-rendered and they're both sitting there in said store and I can serve them dynamically to people based on what kind of browser they're accessing it with. Your cutting edge users get a much faster experience and your laggard, or slower uptake customers gets something that still works for them. I think there's all sorts of little interesting bits like this, is like part of why I'm interested and excited about this space. There's just like so many little ways, and I don't think we've really like figured out the best way to do any of these things yet.

One thing that makes DigitalOcean special is they're really interested in long-term developer productivity, and I remember one particular example of this when I found a tutorial in DigitalOcean about how to get started on a different cloud provider. I thought that really stood

for a sense of confidence, and an attention to just getting developers off the ground faster, and they've continued to do that with DigitalOcean today. All their services are easy to use and have simple interfaces.

Try it out at do.co/sedaily. That's the D-O.C-O/sedaily. You will get started for free with some free credits. Thanks to DigitalOcean for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[00:34:04] JM: We did a show pretty recently with Till Schneidereit from Mozille, where you used to work.

[00:34:11] SK: Totally. I know Till.

[00:34:12] JM: Okay, right. You know Till. Okay. Great. One thing that was really cool about that episode was I've been doing some shows about WebAssembly for a while, and it's one of those topics where it's hard for me to exactly understand where we're at in the evolution of the technology and what the bottlenecks are. He co-wrote an article that just laid out the landscape of all the tooling that we need in order to make WebAssembly really good. It shows in granular detail, here's why it's still immature. You can draw these – You can say, "Okay. Once we get this next iterative improvement, it's going to be this much better. Once we get this other iterative improvements, it's going to get this much better."

It makes it more concrete that this is going to improve. It's going to get better overtime. When you're looking at the intersection of WebAssembly and the edge, what are the biggest handicaps in the ecosystem today? What are the most important areas of improvement what will get to in the near future?

[00:35:22] SK: Biggest one by far is a combination of file size and the fact that WASM modules are sort of statically linked. These two things interact with each other in a way that is complicated. So, for example, we have a limit currently, where you can only upload something that's one megabyte in size at Maximum, and that's for all of these reasons before about

needing to be able to afford to store everything everywhere and all those kind of constraints. So, there's sort of this like file system limit that we currently have.

The problem is, is that when you use a language that's not Rust or C or C++ and compile it to WebAssembly, you need to compile the entire runtime of your programming language to WebAssembly as well and include it in the buddle.

For example, I would say that our number one requested language support is Go. Right now, if you compile Go to WASM, hello world is about 1.1 megabytes. I swear these limits are picked before I was involved, but it make specifically – There's like a joke about being on the Rust core team and have like Rust work and Go not work or something.

So, like this is problem though, because it's like just a little too big. So, we could up that limit and see if that works out in all the various ways. But the real problem here is that like with JavaScript, as I said before, we're sharing V8 across all these instances. Because WASM is kind of like statically linked to put everything into the module, there's no way for me to say compile the Go runtime to WASM and then share that runtime across all of the individual programs that are written in WASM, that are compiled that way.

So, that's like a combination of like Go's preference for statically gained WASM, preference for statically gained kind of like turning into this sort of situation. So, the C# folks are working on this kind of thing a lot with the Blazor project, where they kind of have like the C# runtime kind of dynamically linked into the actual WASM application code. I'm not sure exactly how that works. I haven't had some time to look into it. But like in order for the extra economics of the edge to work out, WASM improving its dynamic linking story is definitely going to be one of the most important parts of being able to bring things, any kind of programming language to the edge.

[00:37:28] JM: Just in case that was a little bit too in the weeds for people. Could you describe static linking versus dynamic linking and maybe give an example?

[00:37:40] SK: All right. So, this is like an age-old tradeoff in programing that was actually motivated by this sort of like main frame days in some ways. So, linking is how you put different pieces of your code together to produce a final executable. This stuff has a very strong like C

kind of heritage to it. But, basically, the idea is that like a statically linked thing means that everything needed to run your program is included in the binary of your program. So, like I send you a binary and everything is in there and you can run it and that's totally find.

But there's also this option called dynamic linking. You maybe remember .dll, those files [inaudible 00:38:17] dynamically linked to library and the associated dll hell that goes along with it is basically like if I give you a program that's dynamically linked, then it sort of has this like holes in it, and you need to have the software that fills in the holes as well as the thing I actually gave you. So you're like, "That sounds strictly worst. Why would I care?"

Well, the advantage is that you're able to share the part that fills those holes with everything running on your system. So like only one copy of whatever runtime you're using and it's shared across all these programs. So that means that the file that you're transferring over is smaller and it means that you're saving space on the server and there's like memory and all that kind of thing that's going on. So, static linking is nice, because it just means it's definitely going to work, but it's a tradeoff of binary size for basically portability in some ways. So, yeah, I hope that's maybe a little higher-level explanation.

[00:39:10] JM: No. That's perfect. Can you just give a brief example of why that's relevant to the cloud world and the edge computing world?

[00:39:17] SK: Yeah. So, right now, if five people want to upload a Go program to the edge, then we're going to have five copies of the Go runtime running simultaneously. That's taking up five runtimes worth of space, and it's all the same code anyway. So, it'd be nice if we can instead run one copy ourselves and share them amongst all those programs, then the total amount of memory size will go down and things would be faster and all that kind of stuff.

Because, also, back to our discussion earlier about cold starts. If everyone has their own individual instance, it also needs to be started and stopped individually. Whereas if it's shared across everyone, start it up once and then used by everyone. So, it's also like faster in that fashion.

[00:39:58] JM: That word runtime, what does that actually mean? What goes into a runtime?

[00:40:04] SK: Words are hard.

[00:40:05] JM: Yes.

[00:40:05] SK: So, almost everyone uses a term slightly incorrectly to be honest. But strictly speaking, a runtime is all the stuff that's in your program that like you didn't write. So, for example, in many languages, that includes things like the garbage collector, or this kind of thing. But it turns out that, strictly speaking, every language other than assembly languages has a runtime. Even C and C++ and Rust have a runtime, but they're just so small. C's runtime is on the order of like 200 bytes or something.

So, people colloquially use the term runtime to mean language with a big runtime and no runtime to mean a language with a tiny runtime. So, it gets really like complicated when you're trying to get to the actual bottom of it. Also, because of virtual machine like is a kind of runtime. Some people use those indistinguishably as well. So, it can be really messy, honestly. This is one of the terms that I like find most confusing when trying to talk to people, because you can never be sure which way in which they're particularly using it. Oh, well.

[00:41:09] JM: Well, it's so complicated. I think about JavaScript. When I have a JavaScript file, and that JavaScript file is going to execute in the browser, that has to be interpreted, or it's turned into bytecode. The bytecode gets executed. Some of the bytecode execution might get put into a format that's like faster access for the hot code path. Then you have different areas of the code that are in different states of execution speed. Then if you add in WebAssembly into the mix and then you have WebAssembly modules that are sitting in one place and you have JavaScript modules that are sitting in another space. Then you have tooling that is loaded into your – Whatever, if you're talking about the server or the browser. You have tooling that's sitting somewhere, and the tooling is called in order to execute these different modules. It's like what areas of this is code? What areas of this is data? What areas of this is "runtime"? It's very hard for me to understand from the vocabulary.

[00:42:20] SK: Definitely. A lot of the vocabulary comes from the days when this was much more cleanly separated, which is why it's like so difficult today. For example, I'm looking up to

call some of the people on the V8 team, my friends, and I'll occasionally joke with them about how like types are compile time concepts. They're like, "What happens if your compiler happens at runtime? Because we write a JIT, and like a JIT is a compiler that runs during your runtime, not during compile time." All these terms are like – It's sort of like everything does everything now. It's just really, really difficult. Because a lot of these has this heritage that goes back to simpler days.

Originally, JavaScript is just purely interpreted and it's very straightforward and there's no big deal. Now, it's exactly as you say. It's sort of pre-compiled to bytecode, and then that bytecode gets interpreted for a little while, but then eventually compiled. It's awesome, but definitely not simple.

[00:43:12] JM: Taking a step back an thinking about Cloudflare and the CDN market more generally. CDNs, they were long regarded as a complete commodity business. It's basically this big caching infrastructure. All that really matters is how much are you paying for it, because they're all pretty fast. Why have Cloudflare and Fastly been able to develop these highly-differentiated businesses with a reputation for innovation in this market that we originally thought was a commodity?

[00:43:49] SK: Yeah, I think – So it's kind of funny, because when I interviewed, I was like, "Yeah, you guys are the CDN company," and almost everyone was like, "Oh, we're not a CDN company." As it turns out, as a web developer, I mostly knew about Cloudflare as like a CDN service. Cloudflare does a lot of stuff. I've been working here a couple of months now and I'm still pretty sure I don't know what all the things that it does.

So, they sort of talk about it as like internet infrastructure. Anything that makes the bytes get from point A to point B faster and better is a thing that they do. So there's like DDoS mitigation and like detecting bots and spam traffic automatically, and like smart routing between different data centers. There's like VPNs, all sorts of shenanigans.

So, I think with that, part of the reason why these companies are getting into this space is because it's like you sort of alluded to it earlier, like in order to build up a CDN, you need to build these servers all around the world and then you're like [inaudible 00:44:45] servers all around

the world, what can I do with them? Why are they just doing – Like as you said, there is some compute that needs to happen to make CDN stuff happen. So, it's like why not make it so that people could do more interesting things with it?

A lot of the early users for workers were sort of like customizing the Cloudflare flatform, like maybe making a different API call based on whatever. Maybe just redirecting some URLs from one place to another. It kind of grew originally from this almost like hyper configuration tool into like sort of – At some point you realize like I'm using a Turing complete language for this. I can just do anything. So, I think people just kind of have those realization like sort of all at the same time, which is a thing that happens every once in a while in history.

Yeah, I think that's fundamentally it. It's like when you look at all these data centers around the world and you're like, "How can we use them even more efficiently for people?" You just sort of eventually arrive at compute, because it's more general. Like I said, you could do anything. So, that's a lot of stuff.

[00:45:44] JM: I don't know if you know the answer to this, but do CDNs, do they actually own their data centers or do they rent from colos?

[00:45:51] SK: I think that it entirely depends. But I'm actually not 100% sure to be honest with you.

[00:45:56] JM: Okay. Yeah.

[00:45:58] SK: I'm sure there's also hybrid set ups too, right? I'm sure some places like own the most important ones and least the other ones out, because, yeah – They're definitely like – Then there's also like do you own a hardware, but you're renting space in a data center that already exists as supposed to building a data center. There's got to be so many options there. Honestly, since that's not part of the business I work in, I don't know that works at Cloudflare. But I'm just assuming that everyone does some sort of hodgepodge combination of everything.

[00:46:24] JM: Yeah. You know what? It's funny, because AWS started as a cloud provided, and then they expanded into the CDN business. I could imagine the inverse happening. I could

imagine Cloudflare, which I get it, it's not a CDN company. Kind of started as a CDN company, at least reputably. But there's no reason why they couldn't expand into a fully-fledged cloud provider. The cloud market is growing so quickly.

I also see changing preferences in how developers want to interact with a cloud provider. Just the AWS console for like a new developer. If I'm a new developer coming out college or even in college, I have no desire to interact with the AWS developer console. I might as well be writing assembly.

[00:47:13] SK: I've been programming most of my life and I have no idea how [inaudible 00:47:15] responsible for owning services. I was like, "You know? I 'm just going to like use either a VPN service or Heroku, because like AWS is like amazingly powerful, but there are so many switches and dials and like it's just not my area of expertise." That's why we have ops people, because that's the area that they know really well. But it's definitely its own full profession, for sure.

[00:47:37] JM: Here's a question. Why is it that Heroku has remained so dominant and so popular despite being first to market or nearly first to market as the usable hipster platform as a service. I'm not saying I'm still deploying my new applications to Heroku despite the fact that there are so many other – I think there's a bunch of other hipster platform as a – I mean, I use Firebase too. Firebase is pretty cool. But what explains the continued success of Heroku?

[00:48:09] SK: I think Heroku cares a lot about developer experience, and you could always charge more – The sort of like classic advice for pricing is you charge based on value, not based on costs, right? So, if you want something cheaper that's definitely out there, but you're going to be dealing with all that manual configuration stuff. I think if you like take a hard look at what does it cost to not only like – Are you really saving multiple ops people dollars' worth of salaries by going with a lower, lower provider and building this stuff on top, or like do you just pay a little bit more and type git-push?

So, like the ultra-simplicity of Heroku I think is why they're still super, super going strong, because it's just like really very straightforward and there will always be a market for like I a paying you to think about this problem. I don't want to have to think about this problem.

[00:49:02] JM: Moving fully into the op-ed section of this podcast and away from the Steve Klabnic talking about Cloudflare workers and the dynamics of edge storage. You've written a bit about pen source, and we've had a number of conversations in this podcast recently about the changes to open source licenses that some independent software vendors have made in order to improve their business relative to the cloud provider offerings.

[00:49:33] SK: Definitely.

[00:49:33] JM: And the open source software vendors, the independent software vendors, when they change their license, they're often making the argument that they're only doing this because the cloud providers have violated the spirit of open source. But, to me it seems like a strange – It's a strange thing to say when you yourself are changing the license of your open source business, or open source project. So, I'm wondering, from your perspective. Who, if anyone, has moral authority in this debate?

[00:50:10] SK: Yeah, it's tough. Because I find myself being very old and cynical about all these shenanigans. Everyone said the GPL was horrible, because it put all these restrictions on you. Then, it turns out that when you let no restrictions, people are going to do what's in their best interest, which is arguably was not in your best interest. Then you're going to be left out in the cold and you're like, "Why did you do that?" It's like, "Well, you said I could do that. So I did it."

I don't think the GPL is like the answer either. But I think that for a while, there was like this – I want to say party, but it's kind of like everyone was having a good time, getting everybody stuff for free, because it helped build their business. Then it turns out that like once your competitors also get your stuff for free and they out-execute you, then you're like, "Wait a minute. No. This is actually bad." It's like, "You asked for this in the first place."

The biggest change with these new licenses is they're not literally open source, because [inaudible 00:51:10] place what's called a use restriction. So they say like, "You're only slowed to CockroadDB," for example. It's like the most recent change. They made thing that's basically like you're not allowed to use in a cloud service or some sort of like there's a business. If you're

doing certain kinds of business, you're not allowed to use the open source version for like three years, and then it reverts back to like a more traditional open source license.

That was like one of the pillars that open source was originally founded on, was this like you're not allowed to say how I can and can't use the thing. There's good reason for that. But this is also the other side of the thing. So, I feel like we're seeing the flipside. There's – I don't want to say bubble, exactly, but it's sort of similar to a bubble, where it's like everyone was getting along and doing real great until some people started getting squeezed out. Now they're like, "Oh my God! What have we done?"

So, I mean, I feel this way in some senses too. I have contributed so much open source software to the world that's made other people super rich, and I'm doing okay these days. But for a lot of my career, I was very extremely underpaid, and that sucked. But I was like, "What can I do? Because that's what the license says."

So, I think that we don't know how to fix this problem yet, because the blog post you probably read that you alluded to, I think licenses are not actually the answer. I don't think licenses are possible to fix this problem. I don't know what is possible to fix this problem, but I don't think licenses are the answer.

[00:52:32] JM: We need to have conversations around what the norms. I guess the conversations are being had right now, but what I didn't like, what I will pass judgment on. Like you, I think I'm withholding judgment on much of this, because this is fairly new territory. But what I didn't really like is there has been some indignation from these independent software vendors towards the cloud providers.

You look at what the cloud providers have done for the software industry and it's just tremendously positive. So, I don't like the assumed moral authority of the independent software vendors. I would appreciate more of a stance of, "Look. We didn't see this coming, and we understand this is controversial, but we kind of want to do it just to expand our business." Maybe I'm miss-characterizing the independent software vendors, and I realize there's multiple software vendors. So, perhaps, I shouldn't be general I think.

[00:53:20] SK: It's definitely tough. You're not wrong, but I don't know if I would fully agree, but it's complicated, for sure, and that's exactly the discussion that we're all kind of collectibly having as you mentioned.

[00:53:30] JM: Steve Kalbnic, thanks for coming on the show. It's been great talking once again.

[00:53:33] SK: Yeah. Thanks so much for having me.

[END OF INTERVIEW]

[00:53:38] JM: Commercial open source software businesses build their business model an open source software project. Software businesses built around open source software operate differently than those built around proprietary software.

The Open Core Summit is a conference before commercial open source software. If you are building a business around open source software, check out the Open Core Summit, September 19th and 20th at the Palace of Fine Arts in San Francisco. Go to opencoresummit.com to register.

At Open Core Summit, we'll discuss the engineering, business strategy and investment landscape of commercial open source software businesses. Speakers will include people from HashiCorp, GitLab, Confluent, MongoDB and Docker. I will be emceeing the event, and I'm hoping to do some on-stage podcast-style dialogues.

I am excited about the Open Core Summit, because open source software is the future. Most businesses don't gain that much by having their software be proprietary. As it becomes easier to build secure software, there will be even fewer reasons not to open source your code.

I love commercial open source businesses because there are so many interesting technical problems. You got governance issues. You got a strange business model. I am looking forward to exploring these curiosities at the Open Core Summit, and I hope to see you there. If you want to attend, check out opencoresummit.com. The conference is September 19th and 20th in San Francisco.

Open source is changing the world of software and it's changing the world that we live in. Check out the Open Core Summit by going to opencoresummit.com.

[END]