EPISODE 827

[INTRODUCTION]

[00:00:00] JM: Web Assembly is a binary instruction format for applications to run in a memory-constrained, stack-based virtual machine. The Web Assembly ecosystem consists of tools and projects that allow programs in a variety of languages to compile into Web Assembly and run in a safe, fast, sandboxed runtime environment.

Web Assembly is a transformative technology for the internet. Most users will experience Web Assembly as a set of gradual incremental improvements to their online experiences. Pages will load faster and become more dynamic. Applications will become more secure. Infrastructure will become cheaper and those cost savings will eventually reach the consumer.

For developers, Web Assembly opens a world of possibility. In today's operating systems, the user can feel a big difference between applications that need a large client side runtime, such as video editing tools, or render heavy games such as Half-Life and applications that are more lightweight and can run entirely on the web, such as Twitter.

This dichotomy may change overtime as we get Web Assembly making these super compute intensive games and render-heavy tools, like video editing software. This stuff might be able to run on the internet.

Tyler McMullen is the CTO as Fastly, and he joins the show to talk about the compilation path, the runtime and the opportunities of Web Assembly. Tyler is a great speaker and I really enjoyed having him on the show.

[SPONSOR MESSAGE]

[00:01:43] JM: When a rider calls a car using a ridesharing service, there are hundreds of backend services involved in fulfilling that request. Distributed tracing allows the developers at the ridesharing company to see how requests travel through all the stages of the network. From the frontend layer, to the application middleware, to the backend core data services, distributed tracing can be used to understand how long a complex request is taking at each of these stages so the developers can debug their complex application and improve performance issues.

LightStep is a company built around distributed tracing and modern observability. LightStep answers questions and diagnosis anomalies in mobile applications, monoliths and microservices. At lightstep.com/sedaily, you can get started with LightStep tracing and get a free t-shirt. This comfortable, well-fitting t-shirt says, “Distributed tracing is fun,” which is a quote that you may find yourself saying once you are improving the latency of your multi-service requests.

LightStep allows you to analyze every transaction that your users engage in. You can measure performance where it matters and you can find the root cause of your problems. LightStep was founded by Ben Sigleman, who is a previous guest on Software Engineering Daily. In that show he talked about his early development of distributed tracing at Google. I recommend going back and giving that episode a listen if you haven’t heard it. If you want to try distributed tracing for free, you can use LightStep and get a free t-shirt. Go to lightstep.com/sedaily.

Companies such as Lyft, Twilio and GitHub all use LightStep to observe their systems and improve their product quality.

Thanks to LightStep for being a sponsor of Software Engineering Daily, and you can support the show by going to lightstep.com/sedaily.

[INTERVIEW]

[00:03:54] JM: Tyler McMullen, you’re the CTO at Fastly. Welcome to Software Engineering Daily.

[00:03:58] TM: Thank you. Happy to be here.

[00:03:59] JM: We’ve done several shows about Web Assembly, but I think it’s important to start by just talking about why this technology is relevant to us. Why is Web Assembly important?

[00:04:12] TM: So, to me, Web Assembly is important. Honestly, the biggest reason to me is that it's kind of the first time that we have ever as a community actually agreed upon a generic safe, relatively fast, easily compilable intermediate representation. I look at Web Assembly not as a language but as an intermediate representation, as an IR. Given that it's the first time that we've done that, like I think there's a lot of stuff that we don't really know yet. Honestly, I think the closest we ever got to this before was like the JVM, which has its own set of issues.

[00:04:44] JM: No, that's okay. I think that's a fair rep description, and we'll unpack why an intermediate representation, which sounds really niche to probably a lot of the listeners. We'll get into why that is so important. But let's talk high-level. So imagine we're five years into the future. How has Web Assembly change my day-to-day life in five years?

[00:05:04] TM: Sure. Well, if everything goes to plan, it means that in five years you are theoretically writing some of the same code that you write is capable of running across many different platforms, whether that's on your local computer, in like some sort of centralized cloud setup at the edge as we are doing, in the browser, on your phone. If we can create a consistent platform that goes across all of these different types of devices, and I know that there are initiatives going on at various places like to do each of those things. I think that would change a lot of how people think about writing code.

[00:05:37] JM: The intermediate representation you alluded to and the comparison to, perhaps, the JVM or Java servlets, I think we should unpack this analogy and this description a little bit more. There's also the analogy we could draw to the LLVM, these virtual machine environments that are highly portable. If we can get our code into these intermediate representations, we can run any of these code, these types of codes that can be compiled into these intermediate representations on whatever medium can run this virtual machine. We talked a little bit about that in an episode with Steve Klabnik. But give me your perspective on that. How does the Web Assembly view of an intermediate representation compare to past intermediate representation approaches we have undergone as an industry?

[00:06:33] TM: Sure. So I think the biggest difference – So there's a bunch of differences between Web Assembly and some of the past implementations of this, but I think probably the biggest difference is the focus on safety. So, obviously, if something is running in a browser, it needs to not be able to breakout and look at other things that are going on another parts of the browser or obviously other things going on in that user's computer. It being safe was utmost importance.

The focus on safety there was like down to the point where like we have like actual formal semantics for this, like the type system for Web Assembly is like provably sound. We have what's referred to as like small step semantics for Web Assembly as well. Meaning we could symbolically execute Web Assembly and kind of see – Essentially prove things like control flow safety. So I think that's probably the biggest difference. There's also like a bunch of smaller differences as well.

So Web Assembly focused a lot on keeping the binary representations small, and that was primarily for the benefit of browsers. So if you're downloading these Web Assembly modules there as small as reasonably can be expected. Then I guess another important one was that there's also a focus on making it generic enough that it can be compiled across many different types of CPUs and platforms.

[00:07:50] JM: My sense is that you can boil down the difference between the Web Assembly compiler tool chain and runtime environment versus the previous environments and compiler tool chains. You can just describe – Web Assembly is cleaner. It's cleaner, it's less of a footprint. It's more efficient. We've had more time to grow as an industry. We've had more time, more time to develop relationships where large players trust each other and they can reach across the table and shake hands with other large players and agree upon standards without nefarious underhanded tactics.

[00:08:31] TM: So far, yeah. That part has gone really smoothly.

[00:08:34] JM: It has, and perhaps we can talk a little bit more about that diplomacy a little bit later on. But I think sufficed to say, I think the industry has realized just how much value there is in fundamental software and basically like it seems like we're kind of on the verge of solving some of these collective action problems that happened in open source in the past.

[00:08:54] TM: Yeah, I totally agree with that. I guess one comment though about the idea of Web Assembly being like smaller and cleaner, that is absolutely true at the moment. But of course, it's also a lot younger than all the other projects out there. So there's always a concern that like you're not careful with how this thing grows. It could grow in a way that you really don't want it to.

But so far, the way that the Web Assembly community group and working group has been setup, that collaborate nature also goes down to like deciding which things actually make it in? Multiple implementations have to exist already before a modification can get into the standard, for instance. So just the fact that like it's not controlled by one single entity means that I think that we're doing a pretty good job at preventing it from growing in like weird ways.

[00:09:41] JM: It's one of these tools where for some developers, Web Assembly is going to completely fade into the background and you're not really going to know why, but your applications are getting more secure. They're getting more performant. Things are just improving. I think about containers and Kubernetes relative to the React JS developer, it's like totally not in my purview. My deployments are getting better. Things are getting better, but I don't really care about the goings on of the Kubernetes world. So who is Web Assembly – Whose developer life is Web Assembly actually going to change?

[00:10:21] TM: That's a really interesting question. I think in a lot of ways you're exactly right. For most developers, they shouldn't actually even know that it's there. Obviously, if you're working with it, say, in a browser at the moment, like you very much know that it's there. It takes work to make Web Assembly work smoothly, especially like to interact with other parts of your JavaScript code, for instance.

That said, if you were to use something like Terrarium, a thing that we released back at the end of last year, which is kind of like a – It's kind of a tech demo of like the compiler that we had been working on. You don't really even need to know that Web Assembly is there. All you know is that you have a platform that can take a bunch of different languages and it just works.

I think that overtime we will actually see Web Assembly ideally kind of fade into the background for a lot of people kind of just as you described there. You shouldn't really know that it's there.

Most people shouldn't really be working with Web Assembly code directly, but in the same way that I'm not typically working with like X86 assembly directly.

[00:11:20] JM: Yes, the term assembly. Perhaps should be telling. Now that said, my favorite example of like what I really want to see Web Assembly change is I use a digital audio work station, and I use FL Studio, which is this – I think I'm on FL Studio 20, which is like the 20th version of this really old monolithic software that is magical to me. It's like super useful to me. It's kind of like Photoshop. Photoshop is this thing. It's like been around for a really long time, and I think they've got Photoshop maybe working in the browser now or it's like verging on working in the browser.

[00:11:58] TM: Oh, yeah. I heard that.

[00:11:59] JM: And probably Web Assembly has something to do with it, or if not, then they have very big plans for that. But there's a lot of people at Adobe and there's a lot of incentive to make Photoshop work really well in the browser. It's less the case for a digital audio work station. I mean, certainly, plenty of people use these things to make music and to do audio editing, but there's a little bit less incentive, so there's less developer resources.

So I wonder what the roadmap is for a company like that, a company like Image-Line, maybe there's like – I don't know, 15 developers working on it, if at all. Maybe there's five developers working on it, and they want to port their giant, old monolith to Web Assembly. When is their life going to be easy?

[00:12:42] TM: Right. Okay. So this is kind of a fun one, right? So they could absolutely do that today. It would be absolutely possible for them to do that today. The problem of course is like integrating it. It's like, "All right, we have all these code. We can compile it to Web Assembly," but of course that code is expecting other interfaces to exist, whether it's like syscalls into the operating system or like specific sets of libraries, for instance. It's expecting certain lower level interfaces to be there.

Right now, well, up until recently anyway, if you wanted that in Web Assembly, the problem was that you essentially had to create that syscall layer yourself in JavaScript. So this is actually kind of the point behind or like one of the points behind the WASI Initiative, Web Assembly System Interface Initiative. Kind of think of it – Like the way I think of it is basically POSIX, the POSIX syscall layer, but for Web Assembly.

The whole idea there is that if you have a normal C program, relatively basic C program. Think like starting with like printf Hello, World. Right now, or until WASI was released, the way that that would actually end up being implemented is like you would compile it and then you would have to somehow like write a print function that existed in JavaScript that would know how to look into the Web Assembly linear memory, pull out the string that you actually wanted to print and then print it in the console or whatever.

The cool thing with WASI is that it makes that syscall layer exist already. In the browser, it will create like a polyfill for it and potentially browsers will also support it directly eventually. But it kind of creates this layer that shouldn't make Web Assembly feel a lot more like a normal platform that you would normally like you just compile it and you run it and – This is actually like – I don't know, I think WASI for a while. The really cool thing with WASI to me is it creates a target that compilers can actually compile toward, right?

So if you look at the way that say like Rust, and Clang, and Assembly Script and all these other languages that now have like Web Assembly backends to them, there was no real way to predict exactly what they were going to attempt to import from the external world. Everybody had a different way of doing this.

So the cool thing with WASI is that like we can actually start to agree upon what the interface to that outside world should actually look like, and then all these different compilers will come along and their output, it should just work.

[00:15:14] JM: WASI is Web Assembly System Interface, and my understanding is like this gives Web Assembly nice APIs for interacting with a file system?

[00:15:25] TM: Well, the file system is a part of it. It's probably the biggest part of it. That's the part that people will like see initially. But really, it's more like you have all your syscalls in Linux that allow you to interact with the outside world. So that'll be the file system, the standard in, standard out, reading and writing. There's like potentially sockets and so on. There's like timers and [inaudible 00:15:48] and all these sorts of different things that like the kernel actually provides for you on, say, a Linux box or like on an OS X box. So now we're kind of matching those same sorts of concepts at least as an interface that Web Assembly can use. Really, this basically just allows compiler developers to know what to target.

[00:16:08] JM: Syscall, is that – If I make a syscall, I'm calling from user space into kernel space?

[00:16:16] TM: Yeah, indeed. Yeah. Sorry, I probably should have explained that. This kind of comes back to – I'm using the term in a strange way in this case. So this kind of comes back to like what Web Assembly is actually emulating or simulating, like what that virtual machine is. The way that I think about it is this, like if you think about like the way a container works, like a Docker container. The Docker container is more or less like emulating the entire operating system. It's pretending that you have your own operating system that you are like running processes inside of.

Web Assembly is like one level beyond that. So if you imagine, basically a Web Assembly instance kind of emulates like a process, in the way that you would think about a process, which means that if you are running a Web Assembly instance inside of something, then the thing that you are running it inside of effectively becomes the operating system to that Web Assembly instance.

So when I say a syscall here, I'm using it in kind of a metaphorical way. Essentially, when you're calling out of that Web Assembly instance, it doesn't really know what's on the other side. It doesn't need to know what platform you're running on, what operating system and what embedding you're running in. All it knows is that on the other side is effectively the kernel. That's kind of the way that I think about it anyway.

[00:17:32] JM: So this is kind if important, because if we're talking about letting my browser – Web Assembly code in the browser can reach in into effectively kernel space, that's kind of a scary proposition, right?

[00:17:47] TM: Not quite that. Again, this is kind of a metaphorical thing, right? The thing that you are running to Web Assembly inside of effectively becomes the kernel to the Web Assembly. It doesn't know the difference. Of course, what you have done is protected the actual kernel from this. It's kind of a hard concept to explain now that I think about it. It's a weird one.

But suffice it to say. Yeah, this is still safe as long as you do it correctly. It's not actually reaching into your kernel. It's that it thinks it's reaching into the kernel, but it's really just reaching into your user space code that you provided.

[SPONSOR MESSAGE]

[00:18:26] JM: GoCD is a continuous delivery tool created by ThoughtWorks. It's open source. It's free to use, and GoCD has all the features that you need for continuous delivery. You can model your deployment pipelines without installing any plugins. You can use the value stream map to visualize your end-to-end workflow, and if you use Kubernetes, GoCD is a natural fit to add continuous delivery to your cloud native project.

With GoCD on Kubernetes, you define your build workflow. You let GoCD provision and scale your infrastructure on the fly, and GoCD agents use Kubernetes to scale as needed. Check out gocd.org/sedaily and learn how you can get started.

GoCD was built with the learnings of the ThoughWorks engineering team, and they have talked in such detail about building the product in previous episodes of Software Engineering Daily. ThoughtWorks was very early to the continuous delivery trend and they know about continuous delivery as much as almost anybody in the industry.

It's great to always see continued progress on GoCD with new features, like Kubernetes integrations, so you know that you're investing in a continuous delivery tool that is built for the long-term. You can check it out yourself at gocd.org/sedaily.

[INTERVIEW CONTINUED]

[00:19:58] JM: Okay. When I have this browser, I'm running a bunch of browsers right now. Well, a bunch of browser windows and tabs in those windows. I got Firefox doing this. I got Chrome doing that. What access levels do these browsers have to the underlying resource? What is ultimately – Give me kind of the trace of when I do something in the browser, how does that result – To what extent does that browser have privileges to to access the underlying system resources?

[00:20:33] TM: Sure. I'm definitely not a browser expert to be clear, but the browser itself – I mean, as far as I understand, has like the standard system resource access that any other process on your system would have. The point that I was trying to make is that if you're running, say, a Web Assembly instance inside of some JavaScript in your browser, the Web Assembly instance doesn't really know what's on the other side. It only has access to the exact things that you allow it to have access to. Much in the same way that a process on your computer only has access to the things that the kernel allows it to have access to.

So like metaphorically, it's kind of a similar sort of concept here. So that's effectively like what WASI is doing, is like in the same way that your kernel provides a set of syscalls to allow your process to access whatever resources, the embeder of the Web Assembly instance provides a certain set of function calls, effectively, syscalls, to allow the Web Assembly instance access to whatever resources.

[00:21:33] JM: Now, I think I'm recalling from my conversations with Lin Clark, that Web Assembly is memory-constrained. So like you have to allocate the resources that Web Assembly is going to get and then like grab those garbage, collect them manually yourself. That's what you're saying here, if I understand correctly.

[00:21:58] TM: Kind of, yeah. It's really more about like the interface. Web Assembly, if you run it without providing it any, say, import functions, any functions that it's allowed to import, it only has access to like itself. It has no access to the outside world. So that is actually one of the like most important things to me about like how Web Assembly works, is that that interface is entirely customizable. So WASI is like an attempt at providing a standard around that.

But it’s an interesting point about Web Assembly and like it's your need for garbage collecting things explicitly inside of it, and it's like general like resource restrictions. That's also like a really important part of safety in this case.

[00:22:41] JM: I want to approach this topic from another angle. In the JavaScript world, there are multiple runtimes. So we have the V8 engine. We have Chakra. We have SpiderMonkey. SpiderMonkey I think came out of Mozilla. Chakra came out of Microsoft, and these different JavaScript runtimes will perform kind of different lower-level implementations of higher-level JavaScript code depending on what browser they are running in. This is one of the reasons why you sometimes have like kind of browser issues from application to application.

So this world of JavaScript, multiple JavaScript runtimes – Ignoring Web Assembly for second, what were the pros and cons of this world where we had multiple engines that could run JavaScript in different ways?

[00:23:30] TM: Pros and cons, right? Of course we have – I guess I’d start with cons. To me, the biggest con is the fact that – Or even if different implementations are following a standard, different implementations are going to have different bugs, which obviously like makes things a little bit bothersome when that happens to developers.

But I think, in my opinion, like the pros far outweigh that. It means that not only not just one entity is controlling where the evolution of the language goes. I mean, to me, having that competition in browser technology, it’s super important. I really don't want to go back to the days where it was just IE5. So I don't know. To me, it's definitely like clearly on the pro side of things. But, I mean, we also see the fact that like browser engines and JavaScript engines in particular have gotten like dramatically faster over the last several years, and like that can’t all just be chalked up to like improvements in hardware.

Just the fact that we have multiple different teams that are trying different ways of optimizing these JavaScript engines has clearly like improved the state of JavaScript in the world –

[00:24:40] JM: How does that compare to the evolution we’re seeing with Web Assembly where we’re seeing multiple different Web Assembly runtimes?

[00:24:49] TM: Oh, yeah. No, I think that it is very similar in that. I also think that we're still in like the relatively early days of this. Web Assembly performance can get a lot better than it is now. I know that a lot of the implementations are effectively like – They're more less transliterating Web Assembly into like the intermediates that are used for JavaScript as well, which have like pretty different performance needs, I guess, and different like PaaS that one can take for optimization.

For instance, as a counterexample to that, engine that will eventually, hopefully, end up in Firefox is something called Crane Lift. So Crane Lift is a project that we've been like working with Mozilla on for a while, like Mozilla has done the vast majority of the work on that, of course, but like we at Fastly have been contributing to that as well. So Crane Lift will eventually be the git backend for Firefox and it's also the ahead of time backend for our compiler that we open sourced just recently actually.

[00:25:49] JM: Right. Let's go a little bit deeper into that. So this tool that Fastly built – Well, you've built a compiler and a runtime for Web Assembly. Why would you do that?

[00:26:04] TM: That's a very reasonable question. We certainly could have just like taken V8 and use that instead, but we thought that we could get some really interesting new performance, especially performance related speedups if we did it ourselves, when we did it in a pretty different way than what's been done out there.

So what we did was we worked together with Mozilla on Crane Lift. Basically, what we did is we build our own compiler that uses Crane Lift as a backend. So what it does is we take Web Assembly in the frontend of it. What comes out the backend is actually like a fully compiled, like normal executable object that it's specifically meant for like Linux systems that run on Intel at the moment anyway. We'll see if how we expand that in the future.

But the biggest reason for it was that we realized that, especially at Fastly, we have this problem that I guess maybe it's not super common. What we need to be able to do is run many, many thousands of requests per second, and what I want to be able to do is isolate each of them individually. I want to make each of these run inside of their own sandbox effectively. But at the same time, I also want to provide people access to higher-level languages that they can run at the edge of our network.

So what we realized was that using Web Assembly, we can actually get these startup time of these sandboxes down to on the order of like 15 to 20 microseconds and run like thousands of them in parallel, which isn't really a thing that you can do with any other technology that's out there right now.

All of that really comes down to some of the decisions that were made about how Web Assembly works at a low level, right? So the way that it does linear memory and the way that it handles control flow, for instance, have made it possible that we don't need to do – Basically, what we can do is we can turn the safety requirements that we need into this collaboration between the compiler and the runtime. So the compiler generates code, that given that it is run in a particular runtime, in a particular environment, we can prove to be safe. That's the main reason behind it, is that like there were other technologies out there, they had too much overhead for what we needed, and we realize that we could do it safely ourselves.

[00:28:22] JM: You mentioned V8 there. So V8 today can run Web Assembly?

[00:28:27] TM: Yeah, absolutely.

[00:28:28] JM: Okay. To my mind, your answer – Well, you illustrated two things. So one, there’s kind of just value in kind of going down your own path, and in this kind of circumstance, much like in the – To some extent, it just makes complete sense for Mozilla. Does Mozilla continue to develop their own JavaScript engine? Do they continue to work on SpiderMonkey?

[00:28:52] TM: Oh, absolutely. Yeah, they’re working on all sorts of exciting things over there.

[00:28:56] JM: Yeah. So I mean, that continues to be a really nice check on kind of Google's control – Well, I don’t know to what extent they control V8. I guess it could always be forked, but their oversight and their knowledge of V8, it's really nice to have SpiderMonkey in there. I think similarly here, like Fastly obviously charting a future for Fastly. It's very easy to imagine Web Assembly playing quite a huge role there. So you probably wouldn't want to take like

Cloudflares's Web Assembly stuff off the shelf. You want to build a core competency and you want to kind of vie for influence perhaps and earned influence in this space.

The other thing you kind of illustrated there is that different players are going to have different priorities. If you think about V8, V8 does so much. Every node application is using V8 I believe. So that's kind of a really wide spectrum of different applications that are using this same JavaScript engine.

You're building a CDN. I mean, the vision for what is CDN is is expanding by the day, and I think the aspirational asymptote you're going for is this idea of the edge cloud, which is really cool, but is a little bit more tightly scoped than like every single Node.js application in existence. So with that tighter scope, you get to make different lower-level tradeoffs. So what are –

[00:30:28] TM: I think that's exactly –

[00:30:29] JM: Tell me about those tradeoffs.

[00:30:30] TM: Yeah. Yeah. So I guess from the way that I think about this problem is that if you think about like the way that the network actually works, like if you're – I don't know, providing some service to your users, you're using a website of some kind, or a mobile app of some kind, you have multiple different stops along the path between your origin and your end user where like different types of computation can be done. I definitely don't want to like try to say that like the edge is the place where everything should be done, because it's not. It's really a question of like efficiency.

So there are many things that it's actually like way more efficient for you to do at your origin server. Computing at the edge is always going to be like we have servers all over the world. We have them in places that you probably wouldn't normally want to put a server like your origin because it's expensive. It's expensive real estate and like more expensive transit and so on, of course.

But there are certain types of problems where it just makes a lot more sense to do them close to the user. I think we'll see more and more of those like specific use cases come out over the next few years. But to me, when I think about like where should I run this computation, it's not really a question that you've realistically been able to just like decide before. You haven't been able to decide, like, “Oh, this actually is better at like this point in the network,” or “This one is better at my origin server.” Really, all you had before was like you have the client and you had your server, and that was it.

So what I'm excited about with this is the ability for us to eventually be able to say like, “Actually, this particular computation belongs closer to the user. This one belongs somewhere in the middle. This one belongs of the origin. This one over here should actually be run on the client itself.” It's like it's a tradeoff we haven’t been able to make in the past. So I'm pretty excited about that.

[00:32:19] JM: I don't want to overuse this analogy. So if I’m going too far, stop me. But like in the JavaScript world, it's like you choose either V8 or SpiderMonkey for a given application. To my knowledge, you’re never going through V8 and then like going again through SpiderMonkey. Are you describing a world where you might use like one Web Assembly runtime for one application and then like another Web Assembly runtime might actually be more performant for another area of your application?

[00:32:50] TM: Not exactly. So where I’m going with this is that you shouldn't really have to care. So what I'm saying is like an average user would say like write their code and then be able to decide where it runs. So if you are running it at your origin server, like maybe you don't need to use Web Assembly at all, maybe you do, or you could take that Web Assembly module you've compiled and you could run it on the browser, or you can take it and like throw it over to Fastly and have us run it on the edge. That's kind of where I'm going with it. It really comes down to a question of like where it's most efficient to run it. Where it makes sense to run this particular computation?

[00:33:28] JM: Indeed, but would the run times, like the –Will the execution layer be written differently in those different environments?

[00:33:37] TM: So when you say the execution layer, what do you mean?

[00:33:40] JM: Well, I guess I mean like, okay, I'm shipping my JavaScript code. So like let's say if I run my JavaScript code in the browser, in Chrome, it's running on V8. So it's getting executed in the same way that it's getting executed on the server, because if I have my Node.js backend on a server somewhere, it's also running the V8 JavaScript engine.

I could imagine a world where there would be a JavaScript engine that would actually work better on a server than it would in the browser. I mean, maybe I'm completely off here.

[00:34:14] TM: Oh, yeah. Of course. No, that makes perfect sense. Yeah, totally. I mean that a lot of the reason why we decided to build our own web assembly like compiler and runtime, because we don't have the same restrictions in some ways as browsers, and we have different sets of restrictions. Really, it's kind of the point. I don't need to – I have the ability to compile code beforehand. So I don't run a git inside of our servers. Instead, we do a like ahead of time compilation of it. Browsers don't have that ability. They download the code and then they compile it as they go.

At the same time, browsers also don't necessarily care quite as much about startup time. A couple milliseconds is probably perfectly reasonable there. Whereas if you're running tens of thousands of these a second, getting it down to the microsecond level is totally worthwhile.

[00:35:02] JM: Now, that said. You could still have the same runtime be deployed everywhere, but maybe you have some like config flag that says, "Okay. Now we're on a server." So that means that you should execute the code. You should use the – If you're on a server, like if on server, use the ahead of time compiler. If on client, use the just-in-time compiler. Would that make sense or do we just literally want different runtimes, like totally different code stacks?

[00:35:29] TM: Yeah. I mean that might make some sense. I mean, if you look at the way that LLVM works, it's not totally dissimilar from that. If you have LLVM bit code, you can choose to git that as you're executing it, or you can choose to like compile it ahead of time.

Yeah, I mean that's reasonable. But again, I also think there's totally value in having like entirely different approaches to this. As long as they meet the standards, like the end user shouldn't really – Like the average programmer shouldn't really need to worry about it. Hopefully, where we're going.

[00:35:58] JM: Right. Okay, we're really getting into the weeds here. So I want to pop out a little bit. Define the term sandbox. How does that word apply to this conversation?

[00:36:08] TM: Yeah. That is a great question. People use it in so many different ways and to apply to so many different technologies. I think that sandbox is it's a reasonable term. But really, what it's describing is like isolation of some kind. Some level of isolation, which in different ways of using sandbox actually means different things.

So I guess I can't really speak to what everyone means when they say about – What I mean when I say it is that this is – I'm going to execute this code inside of something that will prevent that code from reaching out other than in well-defined ways that I have allowed it to, and if that code crashes, the rest of the system should be entirely unaware and like not have to worry about it. It doesn't affect anything except itself is really what it comes down to.

I think that if you like look at the way that we've approached the idea of sandbox, it's really weird. Depending on how you look at it, you could say that like a process running on a computer is also a sandbox. The whole concept of like virtual memory on computers is to sandbox every individual process into believing that it has its own computer that it's running on and not know about and to not know about the other processes that are running.

To me, really, it comes down to like these concepts of like resource isolation, meaning that it doesn't need to – It doesn't worry about what other processes are accessing. Fault isolation, meaning that it doesn't like – A crashing web assembly instance or a crashing sandbox shouldn't be able to affect the rest of the system either.

[00:37:43] JM: What are some different tools that we use for this sandboxing in modern infrastructure?

[00:37:49] TM: So V8 certainly is one of those. Containers are a different type of sandbox. Again, processes are a different type of sandbox. Arguably, like VMs, like VMware and Zen and all the different hypervisors are also sandboxes. Of course, Web Assembly itself is also a sandbox. Again, this is a super generic term, which is like – I don't know. I feel it's almost like reaching the end of its useful life because like it can mean so many different things at this point.

[00:38:16] JM: From a security standpoint, to some extent, we want as many sandboxes as possible. We want as many layers between our application runtime and the bare metal underlying resources that are so sensitive and so vulnerable as possible. I guess, what are the downsides of just layering on these sandboxes?

[00:38:40] TM: Yeah. So the downside of all of this sandbox is that they all have performance overhead of some kind. Obviously, if we're compiling something to Web Assembling and then compiling it to native code, most of the time, anyway, it's not going to be as efficient as just compiling that code to native code to start with. There are certain micro-benchmarks that we found were actually like Web Assembly goes faster than native code, which is fun, but most of them are not that way. They always have some level of overhead.

If you then take that and run it inside of a VM of some kind, VM also has overhead. [inaudible 00:39:13] container in there. Of course, you also have overhead, different types of overhead in that case. But still, it all comes down to overhead.

So I guess our goal is to eventually get to the point where something like Lucet, that we open sourced reasonably, would be capable of like sandboxing things entirely itself. Of course, like [inaudible 00:39:34] is a bit of a wrench into their, which is very irritating.

For instance, with our Terrarium demo that we put out there. That was a tech demo of the compiler and runtime that we had been working on. So of course, we didn't fully trust it yet. So those actually run inside of a Web Assembly instance that is running inside of a container that is running inside of Kubernetes that is heavily monitored to look for breakouts. That was kind of the point of that whole tech demo was to like get some production experience with what we had built, and also like see if people can break it.

Eventually, once we get to the point where we have a lot of trust in that runtime, we would like to be able to do that without that, because it would dramatically reduce the amount of overhead that is actually required to run any of these sandboxes. I'm fairly sure that like at this point, Lucet is like one of the lightest weight sandboxes that has ever been made. Of course, that comes with its own tradeoffs.

[SPONSOR MESSAGE]

[00:40:34] JM: When I was working fulltime as a software engineer, I was always trying to start projects within the companies that I was working, and those projects were often crazy. They were sometimes unrelated to the core business of the company I was working at. But I assume this would be desirable for the companies, because this is how companies disrupt themselves. This is how companies develop new products, and it always struck me that that was so hard to do in any of these organizations. It was really hard to get projects started and it was really hard to find other people to work on those projects with me.

I have been thinking about that problem ever since I left the software industry to start Software Engineering Daily, and now I have a product that I am working on to solve that problem, which is FindCollabs.

FindCollabs is a place to find collaborators and to build projects. FindCollabs has an open network and it also has a closed enterprise system. In the open network, you can find collaborators to work on your projects with from all across the world. In the closed network enterprise offering, you can just create closed projects for people within your company to collaborate on with, and all of it is free right now. You can sign up for the enterprise offering by just logging in with your corporate email address, your corporate Google email address, like if it's @softwareengineeringdaily.com. We run on Gmail, so we login and we can create projects within our company.

We actually use FindCollabs to create and manage collaborative projects within Software Engineering Daily. So we are dog fooding this product. This is something that I really think lots of companies could find useful, because you have innovative engineers within your organization and they've probably got some awesome ideas. So FindCollabs is a place for them to create ideas and find each other.

Also, if you're in the mood to run a hackathon, FindCollabs is a great place to run your corporate hackathon. If you have any feedback on FindCollabs, I would love to hear it. Just send me an email, jeff@softwareengineeringtothe.com. This is really something I want to exist in the world, and that's why I'm building it.

So thanks for being a listener to Software Engineering Daily, and I hope you check out FindCollabs at findcollabs.com.

[INTERVIEW CONTINUED]

[00:43:24] JM: To thrust us back into the world of fairly niche, somewhat I guess advanced topics, the modern way that functions as a service get run is I have some code, I want to execute on the cloud provider. It's a function. I give it to the cloud provider. That code upon getting event triggered gets loaded into a container. You have to do the compilation step. You've got to load all the packages. Hopefully the container has the necessary tool chain to run whatever kind of code is there in your function.

Eventually, after some time, your function will run within a container. This is sandboxed and it's also on demand computation has scale up. It has scaled down. Basically, everything we want out of a cloud provider, but there is some of this cold start. To my mind, Web Assembly seems like a better medium for the spin up and execution of these kind of functions as a service. Would you agree with that?

[00:44:29] TM: I absolutely agree with that, and I think that based on what I've heard from others in the industry, like that is the direction that a lot of us are heading in right now. That's effectively the direction we're heading in as well.

[00:44:41] JM: What needs to be done to have that be a reality?

[00:44:46] TM: Right. So obviously we're going to need more testing, of course, for like the safety properties of these different runtimes. But I think that's actually a relatively small part at this point. We've had a decent amount of production experience in lot of the Web Assembly runtimes that are out there now.

To me, the bigger problem right now is language support. So I love writing in REST. I love writing in C. I really like TypeScript as well. A lot of people don't go, and like they don't necessarily fit the problem space that lot of people are working in. So like I think that, realistically, for us to get to the point where Web Assembly can take over a lot of these use cases, we need more support from the developers of those languages, of course, to actually like produce high-quality Web Assembly code. But we need some changes in Web Assembly itself to be able to support higher-level languages. That's certainly on its way, but we're a bit of a ways off from that still.

[00:45:42] JM: Since we're mentioning this function as a service paradigm, I've love to get your perspective for how the usage of functions as a service will evolve. I think this is the edge cloud vision to some extent. How does the edge cloud look to you? What is that aspirational term mean and how does it relate to functions as a service?

[00:46:06] TM: Sure. Okay. So I different it a little bit from the usual concept of functions as a service, because normally when you're describing a functions as a service service, what you're really describing is kind of – I guess you actually said this a little bit ago. You're describing a function that runs inside of a container that is located in some location. It is in some individual space. Whether that's in like US East, or US West, or EU West, I differentiate that from what we're planning for the edge cloud, because when we say that we are going to run something, run a function on our platform, what we're really describing is that we will deploy that function everywhere. We have a large number of servers spread around a large number of locations around the world. When you actually like go and click deploy on this thing, I want it to go to all of them.

So the whole idea of like needing to spin up and spin down your computations, like in our case, shouldn't actually be a thing. If you have users all around the world and you have spikes in there, like in their viewership and so on, our platform should already be ready for it. So there shouldn't be that whole concept of cold start.

To be fair, that matters a lot for us. It doesn't matter a lot for all use cases. There are certainly going to be things where it makes more sense to run these in a like centralized functions as a service sort of setup. But on the other hand, there are also problems where you want to be able to respond to your users like really, really rapidly with small pieces of capitation.

So I think at that point, having those sandboxes effectively ready to go on every server around the world and ready to respond to those users that are close to them I think is really valuable. That's the primary way that I differentiate it. It's kind of a different execution and deployment model, fundamentally anyway.

[00:47:57] JM: When I started doing podcasts about different companies like 3-1/2 years ago, when I started this podcast, I did some shows with CDN, and around that time, I mean, there is enough complexity end enough complexity to be worth discussing just the content delivery network, which is what Fastly started as.

A content delivery network historically is just used for like static assets, just like an image, or a movie. These things that you need kind of at the edge, but you want to leave them on the servers so you don't want to – Yeah, Anyway. We're going from this place where we used content delivery networks just as caches for assets, for storage, for static assets, to a place where we are caching computation. That to me seems like a pretty fundamental change from their perspective of a CDN.

[00:48:55] TM: Oh, yeah. No, I totally agree. It's a fundamental change and it's also a really exciting change, because honestly, like cascing static assets is not all that exciting of a problem. It is still a problem and it's a hard problem when you do it at scale, but it's not all that exciting. So I'm pretty jazzed about the direction that this is all going in.

But I guess one thing I would say about that is that it is a fundamental shift, but I don't think it's a sudden shift. When we launched about eight years ago, it was primarily to different – We a problem that we were specifically trying to solve that other CDN's couldn't solve for us. Artur, Artur is our CEO, and I both came from your running really large websites with user-generated content. So Artur was the CTO of Wikia for instance and he had approached the various CDNs and said, "Hey, I know that you are primarily for caching static content. Have you consider the fact that like a wiki is actually static until someone goes and changes it?"

So like the use case specifically was – If you imagine like – I don't know, Game of Thrones wiki or something like that. There aren't that many changes when the show is not on. Then a new episode comes out and suddenly you have this like flurry of changes. So the problem was like he wanted to be able to cache the entire HTML content of all these pages. The problem of course was that when he tried to do this, the actual users revolted, because in order for them to view the changes that they had made, they had to wait for a purge to happen. They had to wait for that content actually be replicated around the world, which at the time was taking 10 or 15 minutes depending on which CDN you went with.

So the thing that we differentiated on the very beginning was being able to purge content really, really rapidly. So our purging system is still like I think the fastest out there. It's about 150 milliseconds around the world, which, again, fundamentally changed what you could cache at the edge, like what you could cache in CDN. But it was also kind of, in my opinion, like kind of the first step toward being able to do these like more dynamic things at the edge of the network. It suddenly wasn't just about static content itself. It was about like, "Oh! Well, this content can change. If I can somehow integrate updating my CDN with my backend, my application server, like suddenly I can do like a lot more interesting things." So I think it has been like kind of a gradual but fundamental change.

[00:51:24] JM: So let's go a little bit deeper there. So let's take like Netflix as an example. So far I am on Netflix, I go to netflix.com, and for the most part my experience as I browse around the website, my application is not changing that much. I'm like clicking around and I've got like low low mode, the list of list of movies, and most of the stuff that's changing are the assets that I'm viewing. I click into a list of movies. I click on an individual movie, and most of the code is probably already in my browser for loading that experience. What my computer needs to request is the assets. It needs to go out to the CDN and request a static asset, which is one of these movies.

So that's the state of affairs for most applications. That applies to Facebook too. Why do I need to cache any of these compute?

[00:52:17] TM: Well, so what I would say is that, well, imagine a world where instead of just having the static assets, like the picture of the cover image there, you actually also had, say, like the list of movies that would even be on your homepage cached as well there. Okay, the problem with that of course is that now you have like if Netflix has millions of users, now you have millions of copies of different personalized content at the edge there.

So what I think computation would give us at the edge is the ability to say instead of caching all of that repeated content, all those millions of individual like personalized things, which is difficult enough as it is. What if we can just generate those things at the edge? What if like I had my list of movies and I had your tastes there and I had like a model for what your particular case would prefer, and I could can just generate the list of movies that you're going to like right there on the spot. I think that would be – I don’t know. That's a pretty exciting prospect to me. Maybe that one specifically doesn't work quite as well as some others, but like the whole concept there is it's a lot more efficient to do it that way. Rather than going all the way back to some central server and then just like getting the assets locally, the work of actually like generating a page isn't the assets. It’s the logic that goes into it, right?

[00:53:40] JM: Amazing example. Thanks for running with me on the Netflix example. That’s perfect. We’re running out of time. We alluded to the kind of diplomatic fracas earlier. Tell me about the, I guess, subtle or perhaps not-so-subtle battle, or perhaps not battle between the big players for control of this ecosystem.

[00:54:06] TM: Sure. I mean, honestly, there isn't really a battle, which is like very surprising to me. I started attending the Web Assembly community group and working group a few years ago, a couple of years ago now, and I definitely expected there to be like some level of [inaudible 00:54:20] in the room, and it really wasn't there. It's just a bunch of people who – So, A, lot of like really fantastic compiler developers first of all, but like a group of people who all just want to do a reasonable thing, a thing that will work for the users. So I was really surprised by that, and I gotten used to it. But now that you pointed out, yeah, that is kind of an unusual amazing thing now that I think about it, because it definitely wouldn't of been that way a decade or two ago. It’s a very different world with about browser wars going on.

[00:54:54] JM: It's a charming element of the web assembly community. Did you pay attention to the container orchestration wars at all?

[00:55:02] TM: Not all that much. I managed to avoid that one, thankfully.

[00:55:05] JM: It was brutal. It was brutal. I was a journalist and I got a bullet whizzed past my ear and I got some severe scratches from that. That's how vitriolic it was. Well, I mean not vitriolic, but it was a dogfight, because people thought this was like the directly commercializable technology. I guess to some extent it was, but like what ultimately ended happening is like the company who outstretched its arms most widely was able to kind of encircle the space with its generosity, and that was Google with Kubernetes.

[00:55:43] TM: My thought on that is like Fastly spends a lot of time doing standards work. To me, my take on this is like we should have like those dogfights obviously. It’s like a kind of fundamental part of running a business with competition. But like it shouldn't be over the fundamental stuff. Let's agree on basically like a watermark where it’s like, “Yeah. Okay, all those fundamental stuff, let's get that out of the way and then let's fight over the higher-level stuff above that. Let's fight over the products above that.” Trying to turn these fundamental technologies into products I think just end up hurting programmers and end-users. No one enjoys that, and at the end of the day, it all ends up being – That stuff all ends up being commoditized anyway. So let’s agree on the fundamental technologies and then like fight over the higher level products. That's my hot take on this.

[00:56:33] JM: All right, last question, and it's another hot take. Let's fast-forward 10 years. What percentage of Fastly’s CDN transactions are related to decentralized smart contracts.

[00:56:45] TM: Hopefully as few as possible. I’m sorry. Did you want me to have like an actual response to that? I really don't have one. I’m sorry.

[00:56:51] JM: Really. Are you a total blockchain nonbeliever?

[00:56:56] TM: I’m a blockchain nonbeliever.

[00:56:59] JM: Oh my God! If you're looking for an opportunity to differentiate it from Cloudrlare, you can start to take it more seriously.

[00:57:05] TM: All right. Maybe I should be looking into it more closely.

[00:57:08] JM: Okay. Well, Tyler McMullen, thank you for coming on the show. It's been real fun talking to you.

[00:57:12] TM: It's been great. Thank you so much.

[END OF INTERVIEW]

[00:57:17] JM: This podcast is brought to you by wix.com. Build your website quickly with Wix. Wix code unites design features with advanced code capabilities, so you can build data-driven websites and professional web apps very quickly. You can store and manage unlimited data, you can create hundreds of dynamic pages, you can add repeating layouts, make custom forms, call external APIs and take full control of your sites functionality using Wix Code APIs and your own JavaScript. You don't need HTML or CSS.

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

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

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

[END]