EPISODE 1539

[EPISODE]

[0:00:01] JMC: Hi, Matt. Welcome to Software Engineering Daily.

[0:00:02] MB: Yeah, thanks for having me. I'm excited for today's conversation.

[0:00:06] JMC: We are here in Open Source Summit North America. Is it your first Open Source Summit?

[0:00:10] MB: Ah, no, I've been to a lot of them. This is one of my favorite conferences, because you get so many different – there’s so many different tracks here that you get so many different kinds of developers, and so many different open-source projects. It's the one conference I know of that you get the super broad cross-section of all the interesting stuff going on in open source. I rarely miss this one.

[0:00:31] JMC: It's funny, because I think I was complaining about that. Oh yeah, it's a double-edged sword, right? It's either completely overwhelming, or you got a quarter inch deep and a whole bunch of different things. I think I was talking to, I think, it was Eric Brewer from Google about this. The same happens in KubeCon + CloudNativeCon, right? Even at a bigger scale than this one. It's bonkers how many people attend KubeCon. I’ll keep going. I think they're at about 5X. The number is ridiculous.

It's a very varied and diverse and even complicated to follow ecosystem. Yet, you've got this common thing that is Kubernetes, right? Everything will run in Kubernetes. Yet, Open Source Summit has this different tracks of open source, open SSF this morning, SPDX yesterday. I'm sure there's a track for WASM, or WASM related stuff, or not. Or maybe not.

[0:01:32] MB: If not yet, there will be. If I have my way, there will be.

[0:01:35] JMC: Security. I agree with you. It's beautiful. It's great. It's fun and varied. Yet, I think that it's maybe a bit too open in the sense that I wish it had a bit of more of a certain narratives going out there, right?

[0:01:52] MB: Yeah.

[0:01:53] JMC: It's not criticism.

[0:01:58] MB: Yeah, yeah. I think when you come into this one, you have to be coming in with a perspective that cross-pollination is the goal, right? For us to hear what other people are doing, that we normally would not run into in our regular day-to-day operations, right? That is a double-edged sword, right? Because it means that in some ways, you're sitting on some of these sessions and I sat in on one yesterday on quantum computing and I walked out still having no idea what the session was.

It was a great session and everybody was really engaged and I'm like, I don't understand. This is a little too far afield for my comfort zone. Then on one hand, I wanted to dismiss that and say, okay, well there's some that you don't get anything out of. There's a part of it that's like, no, this is the thing where a seed is planted and a couple of years from now I'll go, “Oh, I remember that talk and I remember that guy.” I'm going to go look that up, because right now it's relevant. I hope that that's the thing that happens here. I heard somebody describe this conference also as the thing where this is the developers' conference, right? By that, I mean, it's really oriented toward the people who are.

[0:02:59] JMC: Exactly. You need to understand that in my complaint that I just – I come from the consumer part, so I'm a journalist and I'm trying to figure out everything. From that side, it feels a bit complicated. Now, if you have a technologist, or a developer yourself and you're here to precisely get pollinated with all these ideas, then you're in the right spot. Even if some of the conferences are too advanced, or too niche, or whatever, but you're absolutely right. This diversity that exists here and exposing yourself to these ideas that might not feel relevant now and so forth down the line will be is incredible. Is it not your first time as an entrepreneur here?

[0:03:42] MB: This is the second. We launched last year. Now I'm really, yeah. I am not a developer anymore, right? This year, it's really ringing true that I've become less and less attuned to the nuances of the conversations that I had been when I was a full-time engineer. We launched Fermyon last year at Open Source Summit in Austin, and I did one of the keynotes there. It was so exciting, because it was the first time that we came out of our stealth mode and articulated the vision and got to see people react, by anything from our silly cat game to this core proposition that WebAssembly is the next wave of cloud computing.

[0:04:16] JMC: Oh, that's bold.

[0:04:18] MB: Yeah, yeah. I only talk about that. Now it's the first year where we get to reflect and say, "So, how has this year gone for us?" Yeah.

[0:04:24] JMC: Okay, that's the next set of questions. Before we move on to that one, exactly, precisely, where does the name Fermyon come from?

[0:04:34] MB: Fermyon is one of the two kinds of particums, [inaudible 0:04:37] being the other. Fermy was the one who discovered all of this. The team with the exception of me is all end of physics. I'm a philosopher. I'm a formerly trained philosopher and PhD in philosophy, don't know a thing about physics. Yet, they come up with a name that I have to explain over and over again and my explanation gets worse every time I get it, because I really don't know. I know Fermy on sub spin, so when we created a project, we created a project called spin. I know that they're distinct from bosons and that we will never name a project boson, because that really just doesn't have that confidence building on a lot up here.

[0:05:11] JMC: Arch enemies. Bosons are arch enemies.

[0:05:15] MB: Yeah, I like it.

[0:05:17] JMC: Fermyon develops basically, not basically, at a very high level, WASM technologies. Would that be in –

[0:05:23] MB: Yeah.

[0:05:23] JMC: Okay. What is WASM? What is WebAssembly?

[0:05:25] MB: Yeah, that's a great place to start. WebAssembly, you can go online and you can read a number of different definitions of it, some of which are highly technical, and some of which are very –

[0:05:34] JMC: In contrast of it.

[0:05:36] MB: Yeah. I think really at the end of the day, the way to approach it is by looking at what came as a precursor to it and work our way from there into what this is. All right, at its core, WebAssembly is a bytecode format that you can compile languages to, okay. When we're thinking about it in the big landscape, this is not the first time we've seen a bytecode format. Java is the paradigmatic one with .NET also being another good example. Java, you take a particular language, the Java programming language, compile it to bytecode, which is a platform neutral intermediary binary representation, or a binary format that's not native –

[0:06:12] JMC: Which is not, by the way, as low level as assembly I presume. It's a bit high level.

[0:06:18] MB: Yeah. It describes what the program needs to do in a fast, executable way, but not system specifically. Then you need the JVM to execute that bytecode and manage to do the plumbing through to the underlying operation system in the underlying architecture.

[0:06:31] JMC: What's the beauty? What was the benefits of introducing that intermediate bytecode for Java, at least, would you?

[0:06:38] MB: Yeah. For Java, that original mantra was write once, run anywhere.

[0:06:42] JMC: Okay. Portability.

[0:06:43] MB: Yeah. Keeping in mind, Java was actually a language that was designed initially in its first inception to be an embedded programming language. That value proposition is very big, because we know once we get to the embedded world, that's where you really start to hit a lot of exotic operating systems, a lot of exotic architecture.

[0:07:00] JMC: Hardware.

[0:07:02] MB: Yeah. The JVM made some – and .NET as well, right? Well, maybe I should pause here and say, so we've seen about 20 years of development on the idea of a top quality, bytecode-based virtual machine, language virtual machines, right? With two pioneers leading the way, Java on one hand, and then .NET on the other. Over time, they've done a lot of innovations. They've introduced JITs, and then they had JIT compilers, just in time compilers, and then they've managed to optimize those and begin to do some really – as they've gone back and forth in the performance race and feature race.

[0:07:40] JMC: Exactly. Because the existence of JITs precisely is like moving away from, not necessarily bytecode, but promoting performance, right? Okay.

[0:07:53] MB: Because once you're running it, then you know what the system architecture is.

[0:07:55] JMC: Exactly.

[0:07:56] MB: You know what the operating system is. You can start optimizing for the specific instance that you're running on. Java and .NET, they go back and forth, honing and retooling. Well, so at some point then, a group of engineers that were comprised of people from all the major browser vendors, right? We had some Microsoft people, some Mozilla people, some Google people from the Chrome team, the Safari team. They all get together and start talking in about 2015 about building a runtime for the browser. They would borrow, they would learn from a lot of the lessons from the Java and .NET ecosystem, as far as a bytecode, an intermediary bytecode format.

One that was really, at that point, targeted toward executing in the web browser. The use case for this, by the way, was, wouldn't it be cool if we could take that old legacy C library, compile it to this format, and then access it from JavaScript in the browser. That was the original idea. Or what if we wanted to write some really high-performance number crunching code? This is the way Figma uses it when they use C++ to write their code, compile it to WebAssembly, and then they execute it in the browser and use it.

[0:09:01] JMC: Oh, okay. Yeah.

[0:09:04] MB: You got a couple of these use cases that involve really taking advantage of languages, language features that are outside of Java, or bringing in legacy code that, sorry, JavaScript. Or bringing in legacy code that's outside of JavaScript, and making it accessible within the web browser.

[0:09:17] JMC: Oh, I see. I see. Who wants to touch that? Yeah, I see the motivation, okay.

[0:09:22] MB: Yeah. You end up with WebAssembly, this specification that's going to do something similar to what Java and .NET did, but in a different environment. It's got four key features, and we'll probably go back and forth on these. These key features are things that have enabled all kinds of interesting stuff. It's got to have a good security model, right? Because you essentially are saying, "I should be able to run an untrusted binary inside of the browser. Of course, I don't want that binary to be able to root my box, or even to be able to be used as an attack factor against the JavaScript sandbox."

[0:09:54] JMC: Wait, let me interject, before we move on to this precisely. WASM is then a spec, which to which companies like Fermyon build against products, right? Or products that are conformant with that? Did you guys contribute to that spec? Is that a spec evolving, and so forth?

[0:10:12] MB: Oh, yeah. This is an excellent question, because it is a specification, and that's what makes WebAssembly impervious to some of the previous attempts to embed richer programming languages in the browser. The W3C is the standard body that oversees WebAssembly. That's the same body that does HTML and CSS. They run it just like they run all their other projects, so there's a broad consortium of people who work together on that, ranging from the big browser vendors to small companies, like Fermyon, to edge providers like Fastly. Just this big cross-segment of an industry that cares about this runtime programming, this performance.

[0:10:54] JMC: We've got that. WASI, apologies. I want to move on to WASI exactly, but that's what I was thinking. WASM is a spec that is hosted in the W3C, and then an ecosystem of tools is built around it, right? Well, we'll go into the benefits of that, or WASM in a minute. What is WASI, then? Let's describe the whole ecosystem if these are equal peers.

[0:11:19] MB: All right, so WASI stands for the WebAssembly System Interface. When we think about, what is the specification that W3 has done already? It's really the bytecode format and a way to execute it. When we're talking about that, we're talking about the core compute features. But we're not talking about how that compute feature interfaces with the world around it.

[0:11:40] JMC: Exactly.

[0:11:41] MB: The original idea, again, browser focused, was, well, you can just inject functions from JavaScript into the WebAssembly runtime, and it can call out, or you can – essentially, you're just describing an RPC model, but in JavaScript and the binary. Then as we started looking at cases outside of the web browser, it became evident to the group working on the WebAssembly standards that if you were going to move it out of the browser, you needed some common system interface kinds of things, opening and closing files, reading, writing files, that kind of thing. Reading environment variables, accessing the system clock, accessing a random number generator, and so on.

Distinct from WASM, the binary format, this is the second set of standards we get emerging. This is how WebAssembly can safely and securely interface with other kinds of features, so that from the developers' perspective, they're opening and closing files when they write their code. But from the host perspective, the thing that's running it, maybe it's a real file system, maybe it's some in-memory representation of it, maybe, but that abstraction there makes it so that you can expose all the familiar idioms to the developers using it, while still reserving your ability to implement a security model that is much stricter than, or whenever much more elaborate than, or flexible than, merely passing system calls straight through to the underlying operating system.

Here again, we could actually contrast this with Java. This is one of the reasons why WebAssembly is distinct from Java. When you think about how Java's runtime works, there are a certain set of core assumptions. One of them is that by default, Java's security posture is that the code that the JVM executes is trusted. When JVM starts, when the JRE starts up and you give it a file, it says, “Okay, yeah. You’re asking for a file in the file system? Sure, knock yourself out. You want to start up a server? Knock yourself out.”

[0:13:39] JMC: No questions asked.

[0:13:40] MB: Yeah. Because that was the model that the developers were thinking of as a created Java. This is going to be a general-purpose language. The security posture of WebAssembly was the opposite in which by default, the guest code is untrusted. You assume, again, think about the browser model, right? You're assuming that you downloaded random code off the internet and you're running it, and you want to protect the environment from it. Then by default, the WebAssembly can't do anything, right? If it asks for a file by default, no, of course not, right? Or, we'll give you an empty file system and you can –

[0:14:12] JMC: Do nothing with it.

[0:14:12] MB: Do nothing with it. Yeah. Then WASI provides these ways of saying, “Okay, this is how I'm going to securely grant this binary access to these things and I'm going to present them as files.” But for the developer, it's like, your regular old POSIX-E system gowns and you're just like, open the file reading it, closing it, and then so on. WASI really is an enabling technology to allow WebAssembly to be run outside of the browser environment in a wide variety of different kinds of circumstances.

[0:14:40] JMC: Does any example come to mind that describes the use of WASI specifically, or?

[0:14:46] MB: Yeah. I think there are kind of – well, so the first version of WASI, WASI is an evolving spec. Their preview release one just added files and environment variables and system clocking, those kinds of things. Preview release two will add networking. Preview release three adds concurrency. But what that enabled us to do was start writing standalone programs that could be run through an interpreter, like WASM Time.

When one of the things that we at Fermyon got started with was saying, well we see a lot of potential for WebAssembly running serverless workloads, and we can get into that in a little bit, but I'll just describe very quickly right here why WASI was germane to that one, right? We wanted to say, okay, if we wanted to just expose a minimal surface that we could build a thing in, could we reimplement CGI, the common gateway to interface revenue, the late 90s, the first way we all did web development, could we reimplement that with WASI and basically provide people with a very interesting idea, “Oh, I see. You can take WebAssembly, you can use this old standard and you can write applications the way we used to.”

It was a fun starting game, because it was went through [inaudible 0:15:58], but it opened up people to this idea that now we could run CGI, which was notoriously insecure as the way it originally worked was it just, the web server shelled out to the operating system and ran something. No guardrails, whatsoever. But here, we can run it in a secure and sequestered runtime and make a very secure version of CGI. It was a fun first experiment. Actually, got us inspired to what we built later on. But that's a good example of how WASM has to do something outside of the browser.

[0:16:25] JMC: Because security is probably the – I was hesitant to say, the main driver. But if not the main driver, one of the main ones. What is it connatural to WASM? Should we use, by the way, WASM to call the overarching technology, to include WASI and WASM? I think it's fair to.

[0:16:45] MB: It's point to do that. There's no real distinguishing the technology and a particular specification.

[0:16:51] JMC: Then, would you agree? If so, or if you disagree, what are the reasons for security being the driver behind the adoption of, in just at least, in WASM?

[0:17:03] MB: Yeah, and I think in the browser model, it made sense, right? Security needed to be a core feature of this, if you were going to run untrusted user code. When you think about the number of environments, particularly now with the rise of cloud over the last decade or so, a core principle that we think of, when we think about the way we architect things is, being able to rent somebody else's server and execute my workload there.

Now, take the server providers. We’re going to take AWS's perspective, or Azure's perspective, right? They're running gobs and gobs of code that they didn’t audit, that they didn't write, that they've never looked into. Of course, they're not going to trust that, right? They want to keep one customer from attacking another customer. They want to protect anybody from, keep anybody from attacking Amazon itself. That security model suddenly becomes very important.

You can also think about plugins in the way plugin architecture or two, where you want to be able to allow somebody to extend your application, but you don't necessarily want that to become a vehicle for a security, for an attack against your system. That core security posture, it was the distinguishing factor of WebAssembly in the browser over against Java and .NET. That has a broad set of applications, where a number of others of us were looking at that model and going, “Yes, that's what I want. That solves a problem for me.”

[0:18:23] JMC: Okay. Another, I was hesitant before to say that security was the main one, because then, another contestant to this is portability, the ability to run everywhere. Could you explain why is it so portable? I mean, you've already explained it, but – yeah, and elaborate a bit more.

[0:18:40] MB: Yeah, the portability story is really important. Again, we can talk about why it was important in the web browser and how that ripples out, right? Think about the web browser in back in the 90s. It was not uncommon, unfortunately, to load a web page and have it say, “Oh, you can't run this website, because it's IE only. It has to run on Windows, or it's Linux only and Netscape.” That’s the word I was looking for. Netscape only. For a while, that was tolerated, but we all didn't like it, right?

As the web matured, that story became untenable. Everybody had to say, it's got to run anywhere. Operating system can't be a determining factor as ARM arose again as a prominent desktop and system architecture. Of course, then we wanted to make sure it was cross architecture, as well a cross processor. That became a necessary feature for the world of the browser.

When you think about the way that we're doing a lot of cloud services, the rise of ARM in the data center, when I was at Microsoft, you can imagine there, two dimensions to this, right? First of all, Microsoft created Windows. Of course, it's important for Azure to be able to support Windows workloads. Also, Linux is the standard platform for the cloud. They had two operating systems they need to make sure has excellent coverage. Then they had Intel architectures and then ARM in the data center starts showing up.

Suddenly, you're telling your developers, you need to write a version of your Docker container that runs on each of these permutations, right? I need a Windows ARM, I need a Windows Intel, I need a Linux ARM, and developers do not like spending time doing this thing.

[0:20:19] JMC: Don't mention risk five. Don't bring that.

[0:20:22] MB: Yeah. That is an excellent point. This is not a problem that's going away.

[0:20:26] JMC: Oh, no.

[0:20:27] MB: This is a problem that we're going to see a richer set of complexities that we start to be able to do things in better and better ways.

[0:20:33] JMC: Exactly. We would be optimizing for sustainability, for performance and the variety of architectures for that is it's not going away.

[0:20:40] MB: Yeah. When the Fermyon team, we're looking at WebAssembly as a candidate to provide us an ultra-lightweight runtime for the cloud. To give a little context, right? When we think about cloud computing, we think right now of two big categories. There are virtual machines, which go from operating in system and drivers, all the way up, with all the bells and whistles up to the application you're running. They're the powerhouse of the cloud, right? They're never going away, because they are so powerful. But they're also slow, difficult to move the images around, difficult to build the images, slow to start up and it takes minutes to start one up.

We needed a second compute and Docker containers came on the scene and we were like, “Oh, look at this. We don't have to ship the kernel and the drivers anymore. We can just take one application and its dependencies and the supporting tools we needed and we could build a long running process that ran inside of a Docker container and we could run our servers that way.” That was the landscape we were looking at when we started hearing from developers, “You know what we really like? We this whole Lambda E-like thing. We like these serverless functions, because all I have to do is write a very small program that takes a request, processes it, returns a response and shuts down.”

[0:21:52] JMC: No packaging. No dependency.

[0:21:54] MB: Yeah. We're suddenly talking about a thing whose lifetime is milliseconds to maybe a few minutes, compared to a container that's designed to be run for days, or months, or so on. And virtual machines which are designed to run basically, indefinitely, right? We're looking at that and the limitations that the current serverless models have had. These limitations are based on the fact that that style of startup, run, shut down in seconds was being executed on platforms that were built either to run from kernel, the virtual machine, from kernel to top of the stack, and that's inefficient. Or run long-running processes, like servers and that's inefficient.

We said, what if there's a third kind of computing runtime that would be really secure and would be cross-platform and cross-architecture? But could really be optimized for this super-fast startup, execute, run to completion, shut down. That's what got us interested in WebAssembly, because all of those characteristics that were good in the web browser, the security model, the execution model, the cross-platform, cross-architecture story, those were exactly the set of features we were looking for.

Fermyon built a couple of tools around this, a spin, a developer tool. That was a bit source, developer tool. Spin basically helps you get from the blinking cursor to the scaffolded application, to the compiled web assembly binary and then to the locally running instance. Then you need a place to deploy your application, and so you can push it up into Fermyon Cloud, which is our hosted version, or you can set up your own Fermyon platform and run it on your own infrastructure, or you can use Azure's AKS offering, they have spin support. Docker desktop has spin support. There's all these different places you can deploy these kinds of applications. But it's really optimized for that millisecond to seconds, maybe minutes timeframe for executing this application.

[0:23:42] JMC: Apologies for the ignorant question. But then orchestrating these processes, functions, whatever they are, does require a Kubernetes-like orchestration scheduler?

[0:23:55] MB: I think in most cases you really want one. Fermyon Cloud, we run Nomad for our orchestration scheduler, which works great for us for the use case we want. We're really trying to optimize for very quick scalability. Also, Kubernetes is another environment in which you can run these. The container deep project has a a sub-project called Run WASI, and that is basically the little shim layer that allows you to plug in something like spin, or something like WASM Time and even these kinds of WASI, WebAssembly WASI run times and be able to execute those kinds of workloads inside of Kubernetes as if they were containers. Use the same pod manifest and describe your applications in the same way you can use deployments and replica sets and all of these kinds of things.

The binaries, instead of being container images are WebAssembly and they get scheduled down to the WebAssembly. But you're right. Your intuition there is 100% correct. You still want to get the most out of this, in most use cases, you still want to be able to schedule this out to run across a cluster, and so you want and orchestrate.

[0:24:54] JMC: The last question I had about this and we can move on to success stories. You've mentioned before we started recording, the BBC. I play a bit happy. You mentioned another one in a way, but you had one bullet point yesterday in your talk that says, literally, ideally can support any language.

[0:25:13] MB: Oh, yeah.

[0:25:14] JMC: What does that ideally mean? Why did you preface it with that word?

[0:25:20] MB: Let's imagine what the biggest risk of WebAssembly is going to be. You defined a really cool bytecode format that has all these great security features and all of these, but it's any language – the language there, right? Any language should be able to compile to the WebAssembly binary format. Who has to do the work of making sure languages compile to these formats? That's an open-ended question, right? The technology itself cannot succeed, unless languages start supporting it.

When I was looking back on what the big things were that caused fear and anxiety in me a couple of years ago was what if C and Rust are the only two languages, the two proof of concept languages that ever get big support in WebAssembly? That's not a compelling use case, if it only got those two languages. What you really need, and then we keyed into the red monk top 20 languages, but you could pick any of those kinds of big language rankings, you really want that top 20 to be supported, or as many of them as possible. Yeah.

What the big risk was, was that those communities wouldn't move. What's happened really, over the last two years more than anything, we started to see .NET moving. By Microsoft, Microsoft started investing. Swift started moving. The community built a Swift to WASM compiler and is working on upstreaming it into the mainline Swift. Then we saw Python go, and Ruby, and JavaScript and TypeScript. And suddenly, now as of 2023, of the top 20 languages, about 17 of them are now at least in progress toward adding WebAssembly as one of the compile targets, or one of the runtime targets.

[0:27:01] JMC: Of course, C++ probably did not –

[0:27:05] MB: Actually, C++, each one of them largely, because if you can compile C and you can pre-process something, you can get C++.

[0:27:11] JMC: Yeah, exactly.

[0:27:13] MB: That's how Figma writes most of them.

[0:27:14] JMC: Well, yeah. Exactly.

[0:27:15] MB: In C++. It's been interesting to see this bigtime momentum moving. In some language communities, like C Python and C Ruby are both core projects for Python and Ruby. That's where the development happened, right? The core teams worked on this as core part of the language ecosystem.

[0:27:34] JMC: Wow.

[0:27:35] MB: Swift in contrast, the community built it and now is working with Apple to mainline that into the Swift language.

[0:27:41] JMC: Oh, I didn't know that about the model of Swift. Apple came to approve any upstream merger? Is it not an open-source project, Swift?

[0:27:51] MB: Oh, it is open source. They basically, the community wanted to fork it, build the WebAssembly compiler and then have that upstream back in.

[0:28:00] JMC: Does it need Apple's approval in a way?

[0:28:03] MB: I imagine so. I got to say, that's not a community I know the mechanics, but I imagine –

[0:28:07] JMC: It needs to see.

[0:28:08] MB: It does speak to the richness of the dilemma here.

[0:28:11] JMC: Yeah, yeah, indeed.

[0:28:12] MB: Even you have some, like Java, where Oracle is not particularly fast moving on things like this. But there are all kinds of alternate Java virtual machines out there like TVM and GraalVM and all of those. A lot of those are moving very quickly toward WebAssembly. It'll be interesting to see how a really big rich language ecosystem like Java can move along, even if the central player isn't necessarily going to be the leader in this case.

[0:28:39] JMC: Because on the other end is, well, success, I guess. Several examples. I mean, I mentioned the BBC player, because you mentioned it and it sound fascinated. You've mentioned those with Figma. Do you want to describe any of those in particular, or any other that you know well that applies Fermyon technology to take the most out of WASM?

[0:29:00] MB: Yeah, and we talked about Figma a little bit, which was a really good browser case. Another funny browser case. This could be mythical, right? Even though I heard this story at Microsoft.

[0:29:09] JMC: I know a company in Spain called Graphics that does it very well. Is it that one the one that you're thinking of?

[0:29:15] MB: No, this one's about Microsoft itself. There is a myth and possibly true, but I don't know. That in Office 365, the web browser version, there was some particularly finicky code that had been in Excel for since the dawn of the universe, right? Nobody touched this code. It was necessary to have this library's behavior in the spreadsheets, so that the Office 365 version is a 100% compatible with the desktop version.

[0:29:41] JMC: Yes.

[0:29:41] MB: The story goes that they actually compiled that library, that C library to WebAssembly, so that they could get this one-to-one behavior. I would love, if any of you ever know whether this is true, I mean, I heard this story at Microsoft. But fables do have a way of being more attractive than reality. I like those kinds of stories that say, look, sometimes you've got this code that needs new life, but nobody's going to actually go in and edit the code. This is a way to do that.

You mentioned BBC, and BBC is a really cool edge. BBC, Amazon Prime's player, and supposedly, also Disney Plus's player, all use WebAssembly. The reasoning behind this is really cool. It is in BBC's and Amazon Prime's best interest, they have their players on as many of the devices and TVs and sticks and game-playing things as they can.

[0:30:33] JMC: Maybe in the Tesla.

[0:30:35] MB: Yeah, yeah. Well, yeah. Depending on your driving style. Yeah. Think about the typical language for IoT is C. Writing C, that's multi-platform across. BBC said, they support 9,000 devices, right? I mean, it boggles my mind on what that would look like. It would be really cool if you could write a really small hardware-specific shim that had a WebAssembly interpreter in it, and then put the bulk of the shared code in the WebAssembly level. That's basically the approach to these players.

[0:31:06] JMC: Oh, fantastic.

[0:31:07] MB: That's really cool. I think that is such a cool way of showing the value of a technology built for the browser in an environment that is very different from the browser.

[0:31:15] JMC: Indeed.

[0:31:17] MB: The IoT, again, being this one where every byte counts when you're there. You end up dealing with some very exotic hardware configurations. That's a really cool application.

[0:31:27] JMC: It’s all compute.

[0:31:29] MB: Yeah. Then, because I can't help plug Fermyon, right? I mean, what we've really looked at trying to tackle is serverless environment, and how can we build a better serverless functions? When we went into it, we said, okay, now we think that at looking at the data, a lot of people really want to be able to write very simple web applications, or multi-function web applications that just live in that Lambda-E world, but Lambda is too slow to execute those. Takes a couple hundred milliseconds to half a second just to close out.

[0:31:59] JMC: The problem that you reckon has stalled a bit, the momentum that, okay.

[0:32:04] MB: Yeah, that and the developer experience. Two things that we've heard that about. We went, okay, well, WebAssembly’s start up time, we could get it all the way down with using basically, JIT and ahead of time compiling techniques. We could get a cold start of a WebAssembly application down to one millisecond. 199 to 499 milliseconds faster than what we were seeing with AWS Lambda. That's a really compelling thing. You could tell a good developer story on top of that, which is something we have worked diligently to do. Then suddenly, we can offer a really compelling case and say, “Look, you can write web application backends, microservice style things in WebAssembly. You're in the language of your choice.” Because the WebAssembly part isn't even a part that's necessarily visible to the developer. Write it in Rust, write it in JavaScript, whatever. It gets compiled, pushed up somewhere and executed blazingly fast.

[0:32:55] JMC: Is spin providing that developer experience?

[0:32:57] MB: Yeah.

[0:32:57] JMC: Okay. Any design decisions that you feel particularly proud of? Any constraints that actually help in that sense to make functions as a service, or service as more easily consumable and delightful?

[0:33:13] MB: I mean, our core user story for the entire 2022 year, we use user stories to make sure we're guiding our feature roadmap to actually meet the needs of the user. Our core one, the one we just absolutely were inflexible on was as a developer, I can go from a blinking cursor to a deployed application in two minutes, or less.

[0:33:34] JMC: Wow.

[0:33:35] MB: We went, okay, that's ambitious. Yeah. But it should reflect. This is going to be the hello world style application, right? The idea is, it can't be the thing that's going to be cumbersome for the developer to pick up. You really want this – for us, it was like, imagine yourself on a Friday afternoon, you're winding down on your last hour of a day, you can't leave early, you don't – you just want something to try and get you excited about your job again. How can they come to Fermyon and try this thing out, experience success, and go away for the weekend going, “That was fun. Maybe next week I'll try something bigger with it.” That was the story we told ourselves, that we really would love to be able to satisfy a developer there. We've worked really hard to tell that story.

Originally, again, the big risk for us was, what if languages don't move, right? We've seen languages start moving, in fact, faster than we can integrate them with and that's been, and that's awesome. But it's been this environment where we're just really excited, because we hear from developers, “Yeah. Yeah, this is the experience I want.” Now they're saying, “Hey, you know what would be great? If I didn't have to manage a database when I needed a database.” WebAssembly is uniquely suited to be able to abstract away from that network layer thing.

When we were at KubeCon in Amsterdam a couple of weeks ago, our big announcement was we announced key value storage that is built in to the platform, so that you don't have to manage the service at all. That's ops work, right? When I'm building an application locally, I should be able to call just as if it were an API level call, hey, get store, or whatever. I shouldn't manage connections. I should never even see a username, password connection string, or anything like that. Locally, that should be just some local storage. We actually use SQL, SQLite to do that locally.

Then when you deploy it, you want that same set of features, but you want a real database behind it, right? I don't mean that pejoratively, it does SQLite, right? I mean, it's [inaudible 0:35:26]. It's going to be up all the time and it's going to get automatically backed up. But the developer doesn't want to manage the ops for that. We said, all right, well, in Fermyon Cloud, we can build it in. Because again, this is outside of the guest code, the developer’s writing. It's in the runtime itself.

We made it so that when you deploy, if you need the key value storage, it's just there, right? Again, no connection management, no username and password. We were surprised to see that Dino and Vercelle both released similar things for their JavaScript platforms within a couple of weeks of us and out –

[0:35:57] JMC: Which I guess is validation of it.

[0:35:59] MB: Yeah. This is something developers really crave, because it's an easier story. We've been pushing them with Kubernetes closer and closer to having to know more and more about the ops layer. Here, we're swinging the pendulum the other way and saying, “You know what? We know you need key value storage, but we know you're going to need relational database and we're going to add that thing next, right? How can we remove every single speed bump along the way?"

[0:36:21] JMC: Wow. This is the last question that I have for you, actually. What is next for WASM, for the existing – What is next for Fermyon? Yeah, what's the next stage for adoption?

[0:36:35] MB: On WebAssembly, the core, we got a couple of specifications that are moving through that are really important. There's a memory management one that'll help bring along several different languages like Kotlin and Dart. We've got the remainder of the WASI work, preview two, preview three, and then the final release. Then we've got this thing called the component model. This is super cool. This is the thing I am actually really – the worst name for a technology.

[0:37:01] JMC: It sounds very dull. I will say that.

[0:37:04] MB: Yeah. Any time I have to say it's super cool after saying the name, that the name was undersell, the name undersells it, right? Here you got a neutral bytecode format. Inherent in it is this ability to call in and out of the environment outside of it. What would be really cool is if we could say, "Hey, this WebAssembly library was built in Rust, this one in Java, this one in Python." I'm writing a piece of JavaScript. I can just import all of those as if they were JavaScript libraries. Or another way to say it is, I should be able to link to them without carrying in all one language they were written in. What we really need as a specification that says, these are the functions I export, these are the functions I import. This is the library I export, this is the library I import. The component model is a way to do that.

You can think of it as taking a lot of those ideas that were latent in technologies like RPC, and then GRPC, and even rewinding the clock way back to Dcom, and Corba. But bringing them into relevance in this new and fascinating environment. If that works out, then finally, we will get to that point where we don't have to write a YAML parser in every language under the sun, and a network library, a database driver in every language under the sun, and we'll be able to share those. That's coming along, too. That's coming along this year.

We've been prototyping some of it in spin, and it is fun. I mean, it's just such a jolt of adrenaline to go. It doesn't matter what language this host library came from. I can just start calling functions. We've been playing around with that a little bit. The specification is still in flight, but should come out later on this year. That's the exciting stuff on the standard side.

On Fermyon outside, we're just excited to continue telling this great story about how easy it is to develop serverless applications that are going to be super-fast, that can run anywhere from Fermyon cloud to Docker, to you run on Kubernetes cluster, to a Nomad cluster. That, I think, to us is really exciting. We'll keep rolling out these data services and keep rolling out language support.

[0:38:58] JMC: Well, that was a fabulous conversation. Thanks so much, Matt, and –

[0:39:02] MB: Yeah. Thanks for having me. This was fun.

[0:39:03] JMC: My pleasure. I wish you all the best.

[0:39:05] MB: Thank you. Thank you very much.

[END]