EPISODE 1180

[INTRODUCTION]

[00:00:00] JM: Serverless has grown in popularity over the last five years and the space of applications that can be built entirely with serverless has increased dramatically. This is due to two factors; the growing array of serverless tools, such as edge located key value stores, and the rising number of companies with serverless offerings. One of those companies is Fastly, which originally gained adoption for its CDN solutions. Tyler McMullen is the CTO of Fastly and he joins the show to talk through how Fastly looks at edge computing today as well as serverless. This is Tyler's third appearance on the show.

[INTERVIEW]

[00:00:41] JM: Tyler, welcome back to the show.

[00:00:43] TM: Hey, Jeff. Thank you.

[00:00:46] JM: We're talking today about serverless applications. Give me an overview of the typical architecture of a “serverless application” today.

[00:00:54] TM: Yeah. No. I mean there're lots of different architectures for serverless applications, but a typical one might look a lot like actually fairly similar to what one would expect for like a typical, say, web application where you have the client side aspect of it, you have the serverless aspect of it and that serverless application may be speaking to a wide variety of different like state stores, whether that's a like a GCS or like an S3 of some kind. Or you could be talking to like a typical relational database of sorts as well. But on the other hand, then you have totally different architectures where say serverless is actually being used for post-processing of things for like batch processing for instance. But really like in my opinion like this is one of those things that like there isn't really a clear pattern at this point. Like there's a wide variety of things out there.

[00:01:43] JM: How has serverless matured in the last five years?

[00:01:47] TM: Yeah. I would say over the past year, serverless – Or the past five years rather, serverless has matured in a bunch of different ways and also not matured in a bunch of different ways as well. In some ways it's kind of interesting. It's kind of both regarding a specific point. So like for instance programming language support I think is a great example of this where, yeah, like it has matured in some ways. We have more options now than we used to. That said, it's also not matured nearly enough in my opinion.

One of the big advantages of using your own servers at this point is just the fact that you can choose your own languages. You can mix languages together. You can do all sorts of things that like serverless doesn't actually allow you to do at the moment. It can be kind of constraining, which is kind of ironic given like one of the main points of serverless is to be less constraining. But it has matured in a bunch of different ways as well. And I think a lot of the way this it has matured has been like in the use cases of it, right? Like in how people are actually able to use these things and how it's able to connect to various different services as well. Previously, like it would have been hard to use serverless for things that were distinctly stateful five years ago. Now you have like lots of different options for databases. Again, it's kind of it's kind of all over the board with this one in my opinion.

But to me what's actually more interesting are the ways that serverless is likely to mature in the next five years, right? And I would love to get into that and I think that like our work on the WebAssembly ecosystem is going to play a big role in that.

[00:03:18] JM: Tell me more about what Fastly's perspective on serverless and the tools Fastly has built over the last several years has been.

[00:03:27] TM: Yeah. So our take on serverless has been that it's been a little bit too prescriptive in my opinion. The industry as a whole has – My take on serverless, or our take on serverless rather, is that like kind of the way that the industry has gone about this so far has been a little bit constraining, right? Our perspective is that like serverless should actually be a lower level platform. It should be a platform on which you are able to build whatever you really want to build rather than being so prescriptive in the way that we are expecting it to go.

[00:04:02] JM: Can you explain to me what Fastly's perspective on serverless has been over the last five years and the tools they've built?

[00:04:10] TM: Yeah, I can definitely explain that. So our take on serverless is that, well, we have a bunch of takes on serverless, but my big thing is actually about like placement of serverless. It's about like where serverless applications can run and how they run. To us, the whole idea of having to choose a specific location for your serverless function to run in is kind of like antithesis. It's kind of the antithesis of serverless because the whole point is that like you're supposed to be able to pretend, and I'll be honest, we're kind of pretending that there isn't a server involved, right?

And here we have gone like, "Okay, you have to choose a location," which means that you're running it in a specific place, which means that the data that it also needs to be nearby that as well. So our perspective on this is that serverless needs to get a lot more flexible with the way that you can choose to run things. So a lot of the work that we have actually done has been in order to make that possible, right?

So if you look at like the way that we build out our like WebAssembly-based serverless platform has been in a way to make it possible for these WebAssembly applications to run in a variety of different locations. And so like from our perspective that means the edge, right? That means 55 different markets around the world able to run your application and able to run it in all of those locations simultaneously. But I think it actually goes beyond that where, like to me, in order to make it possible for serverless to not be like a constraining environment that you use for like tertiary parts of your applications, the whole thing needs to get a lot more flexible. It means that it's more than just about running it at the edge, in my opinion. It's also about like how you can run it at the origin. How you can run it even closer to the user's client. How you can run it in things like embedded inside of a database or embedded again inside of the client, right? To me, like the whole point, like the whole power or the potential power of serverless is actually about being able to spread your application across all of these different locations, right? And the current technology that's being used to implement these serverless platforms just doesn't actually make that feasible, right?

So a lot of the work that we have done has been going into making the technology underlying this much more flexible. If you look at our work on Lucid, the big thing that we have trumpeted about it has been how much faster startup time can be, like the cold start time problem, right? And to me it's not actually about the cold start time like a lot of the time. What it's actually about is like what the cold start time allows you to do. So on the one hand it's actually about security, right? Cold start – Like having very low or non-existent cold start time means that you can like individually isolate lots of tiny little things. It means that like you're not sharing isolates. It means you're not sharing sandboxes across requests and across different like contexts, but it's also about like where, again, it's possible to run these things. Running a container-based sort of serverless platform at the edge I think just doesn't make any sense, right? It makes a lot more sense if you can run the whole thing and get it started in a very short amount of time. I think that's also going to open up like tons of different possibilities for different locations within the network as well.

[00:07:42] JM: So these abstractions that you're providing, how do they compare to architectures or the abstractions that are offered by like AWS? Like I think of AWS is offering these more thick and well-defined abstractions, and I think Fastly is more concerned with just enabling programmability at the edge and a lot of flexibility at the edge. Do you think that's accurate depiction?

[00:08:09] TM: So that's not exactly how I think of it. I think that the difference in my opinion is actually about like the prescriptiveness of it and the flexibility again that you have in defining those abstractions, right? So like Fastly, our approach to it has been to work on the lower level abstractions for serverless to be able to provide folks the ability to do a lot more because we provide, again, that lower level abstraction. However, like that's not actually the way that people like tend to program. That's not the way that a typical application is developed.

And so the thing about like us providing that flexibility is that then we can define a much higher level and different abstractions on top of that, right? And so I think if you look at like – I think our Rust abstractions are probably the most well-defined at this point. But as we introduce more languages, we're going to see similar and also different abstractions built on top of that, again, because we have given that amount of flexibility in the system. And, again, I think once again this is just in my opinion coming back to like that flexibility, right? It means that if we can provide those lower level abstractions, we can actually make it possible to run similar serverless applications across a wide variety of different like platforms be that Fastly platforms or also non-Fastly platforms for that matter as well.

So to me like those two things aren't actually like opposites of each other. It's just that like with the flexibility that we have like put into the system it means that you can again define different and more higher level abstractions, thicker abstractions as well rather than being like beholden to exactly what like your platform provider actually says that you can do.

[00:09:53] JM: How aggressively are users adopting serverless technology? I know that the providers are building out all these tools and platforms for making serverless possible, but are users adopting it?

[00:10:04] TM: Yeah. I mean to me it looks a lot like the early cloud days where we are somewhere along that curve. Like I remember when like AWS and such first came out and I was obviously a software engineer back then, and I remember the like internal battles that were had about it inside various companies that I had worked at where the early adopter type engineers who were like, “Yeah, we should be using this immediately,” and then you had the older folks or like just the more conservative folks who are going like, “Well, let's kind of wait and see. I'm not convinced yet.” I think we're in that same position now. Like the adoption levels are definitely increasing and they're increasing rapidly, but there's definitely still resistance to it as well. And I think that it's kind of on us and like on us as like serverless platform providers to be able to clear up those concerns that folks have.

And again, like to me this comes back to like – This again comes back to like that whole flexibility question, right? Like if our platforms are not able to like address all of the different concerns that users have, then like well adoption is going to start to stall. If we can't provide like languages that people actually are used to and want to use, well, serverless is going to stall. If we, again, like end up with like vendor lock-in problems, well, it's going to stall, right?

So to me we're at kind of a pivotal point in like that curve of serverless adoption where if we do this wrong, well, serverless is going to always be kind of a sideshow sort of thing. But if we do this right, well, serverless could become the dominant like way of doing computing on the internet, and that's definitely what we're trying to head towards.

[00:11:52] JM: What other kinds of outcomes could lead to a stalled serverless environment?

[00:11:58] TM: So there's been a lot of talk about this one lately, and the big points that I have seen, anyway, pointed out are, again, limited programming language support. We have vendor lock-in problems. We have performance issues, which I think everybody knows at this point, and just kind of like an inability to run entire applications on it, right? Where, again, this just ends up being a tertiary part of your actual application architecture rather than the thing on which you are actually building the whole thing. And there's lots of ways to address all of those different pieces, and I think it's actually really important that we do so.

To me, probably the biggest one is actually like this first two; programming language support and vendor lock-in. If we can't provide the programming language support that folks are looking for, then it's always going to be a thing where it's just for JavaScript developers or it's just for Rust developers, right? If you're you know using .net as millions and millions of developers are, well, like what are you going to use? You're going to be stuck with like the one platform that happens to support that.

So to me like vendor lock-in and programming language support actually kind of go hand-in-hand where you end up with either not being able to use it or only being able to use the one that you have been programming against. So that's why we've been putting so much effort into like the standards type approach to this problem. I think that we're at the point with serverless it's big enough now where we're kind of beyond the experimental stage, right? We need to start agreeing as an industry on what exactly this thing is actually going to be when it grows up. So that's why we have put a lot of effort into the WebAssembly ecosystem. That's why we have put a ton of effort into WASI, the WebAssembly System Interface, and why we have open source so much of what we have done. And we're working with a bunch of different folks in the industry to actually make that happen. Part of that is through the Bytecode Alliance, and then there are others that we're just working with directly as well.

But yeah, I think if we can't solve especially those two problems, it's never going to be the – It's never going to end up being the big thing that we all hope it will be. It kind of reminds me of the processor wars of the like 1980s, right? Like are we going to go with x86? Are we going to go with various other different processors out there? And we ended up like landing in I think a reasonable spot where like most people ended up on like an x86 type platform.

And to me like it kind of all points to the same thing, which is we need to be – We effectively need to be competing over the feature set, not over like the base architecture of the system. We need to agree upon that one, otherwise none of us grow. And so that’s to me I think the most important point. That's to me why I think it's like it's so important for us to agree upon like the basic architecture of serverless and like what the interfaces actually start to look like. And I think if we can do that, like we have quite the potential here.

[00:15:05] JM: Give me a little bit more context as to how the WebAssembly-based platform at Fastly has developed. Where are you? What can you do with WebAssembly and what can you not do?

[00:15:17] TM: So WebAssembly as a whole is in my opinion like a really good place at this point. There's not much that one would want to do with WebAssembly that you can't do. To me it's actually about the developer experience side of it. It's like how easy can we actually make those things. People approach me all the time and ask about, “Well, like what are the use cases for this thing?” And I can come up with a whole bunch of use cases, but to me it's kind of a funny question in the sense that like you're asking me what are the use cases of a computer essentially, right? Like there are so many different use cases. Like it's hard for me to even place my finger on like the one like killer use case for it because it is, again, such a like flexible platform.

But yeah, so like to get back to your question, like what can and can't you do? Fastly is in limited availability with our platform right now. We're in the process where we are building out some of these state related pieces of our platform. So key value stores and databases and so on, right? And so those will definitely like make a big difference in what is possible with it. But even without those, folks have found like just tremendous things that they're capable of doing with it. Be that like mobile offload or like page assembly or GraphQL or like manifest manipulation for video processing. There're just so many different things that are possible to do with it. And the more features we add to it, the more is going to be possible to do with it.

[00:16:51] JM: What does a key value store specifically built for edge computing or for serverless look like?

[00:16:59] TM: That's a really fun question. So like I've mentioned this a couple times in the past, but like our version, like our path down the serverless rabbit hole actually started quite a few years ago. But like the initial thing that we were starting on when we when we first ended up going down this path was not actually a serverless platform. It started with my desire to actually build out a global distributed data store and then me realizing, "Well, in order for that to be useful, I guess I'm first going to have to build a platform for people to run like more advanced programs on to make that useful." But like over the last several years I think that our view on this has actually gotten a lot more nuanced.

So, to me, like my initial view of this was, "Okay, cool. I'm going to build out like a massively intelligent, globally distributed, eventually consistent, like strongly eventually consistent data store, and that will solve everyone's problems." And to me that's just kind of a – In retrospect, that's kind of a silly view to have, right? Because it turns out that what people actually want a lot of the time is access to their own data. And the way that people access that data is not necessarily in a way that is like appropriate for eventually consistent applications. It turns out that like the use cases kind of dictate what exactly is going to be useful for this.

So there are certainly some things where a great key value store for an edge-based serverless platform is going to be an eventually consistent one. It's going to be one where you can make updates from across the world and you have to kind of expect that they're not all going to be in sync all of the time. But our realization is that that's actually not nearly enough, right? And it's certainly possible at this point for you to say like make an HTTP request and send that request back to your own database that is in your own like data center somewhere else across the world. But doing that repeatedly from, say, let's say your database is in Virginia and the request that we're processing is in, say, Melbourne, Australia, right? Doing multiple database queries from across the world like that is just not going to be an efficient thing to do unless of course we have that built into the platform itself, right?

And so that is I think a lot of the direction that we're going is kind of a multi-pronged approach to this, where we want to build out a key value store. We want to build out a state story that addresses all of these different types of use cases that folks have where the ones where folks can actually do the updates and do the reads from the edge directly where those are supported and extremely fast for that matter as well, as well as being able to support the idea of being able to query your own data, being able to query strongly consistent data from the edge as well and make that possible to be fast as well.

[00:20:00] JM: So with all that being said, can you make queries at the edge strongly consistent?

[00:20:08] TM: So my answer that question is yes. It is totally possible. However, at first glance, it may not seem obvious how, right? Because I think if you try to do this in a way that sounds a lot like, that looks a lot like the way that we have typically done, that we have typically done querying of like strongly consistent databases from the edge, it's going to be unreasonably slow. It's not going to work well.

However, I think that we have – Because of our like flexible approach to this whole serverless problem, we actually have some options that may not be obvious and may not be available to others as well. So from my perspective, this really comes down to the concept of like how do you break up a program to make it run most efficiently, right? So if you think about the network as this big interconnected thing, but you may actually think about the network in this case as rather like a line with a bunch of different stops on it going from your origin. Like the most central location in your architecture out to the client that's actually talking to you. And no that line, at each of those stops, there is different data available, right?

So, to me, the whole thing with serverless and the whole thing with making something like this feasible actually comes down to where does your program run. And that answer to that shouldn't actually be one location on that line. It should actually be most of the locations on that line. And so I know I'm speaking very like philosophically about this, but essentially like our answer to this question is going to come down to like how do you break up this program to make it feasible for you to query and use the types of data you need all the way across the network?

So, for instance, if you have a program that does a lot of things that are stateless and then does a few things that need to like actually talk to the database directly, those probably shouldn't run in the same place, right? And so our plan with this, our goal for this is actually to figure out ways to like build patterns into our architecture to help users to spread their application across multiple locations like this. And I know that's a really vague, like big idea, like large brain sort of like approach. Sorry. Excuse me. Galaxy brain sort of approach to answering this question. But I think this is the direction it's actually going to need to go. My big problem with the way that serverless works today is, again, it comes back to that like location question. We shouldn't have to specify a location. This should actually happen automatically if we can figure out a way to do it. And I'm looking forward to like figuring out the answers to those and figuring out the ways to do that over the coming years.

[00:22:59] JM: And so tell me more about what kinds of applications want state management at the edge. Like is it only used for caching or am I using this for like my actual application data?

[00:23:10] TM: Oh yeah. Again, there's just a wide variety of different types of applications for this. So yeah, I mean part of it is caching, for sure. Part of it is like partial caching. Being able to cache certain parts of pages, for instance, and not other parts of them. Certain parts of APIs and not other parts of them. But you can also use it for application level data as well. Like if we go back to like the earliest like Dynamo papers from Amazon, right? Like those are a great example of like how you build – Like it was all kind of based around the whole concept of a shopping cart, right? Like how do you build a shopping cart out in a like eventually consistent manner? And, to me, like when you see eventually consistent manner, what you're actually saying there is how can I do this at the edge, right?

And being able to build out, say, a shopping cart or something like that is totally feasible at the edge as well. But there are other like I think weirder applications of it as well. So for instance, one of the ideas that has come up a lot recently for us is actually mobile offload. And so I guess for those who may not be like aware of that one, the whole idea is, well, a lot of us walk around with basically mini-super computers in our pockets called iPhones and powerful android devices. But most of the world doesn't, right? And so if we're building out applications that require like a massive amount of computation on these devices, you're kind of cutting off a large swath of the world from using it.

And so mobile offload is an approach to solving that problem where if the device that a computation is trying to run on isn't powerful enough or doesn't have access to the things that it needs, you can actually take that computation and push it out to the edge instead and run it there and feed the information back.

So the way this ties into state is I think interesting because if you are having to pass the entire state of the application back and forth every time that you do one of those computations, it's not going to be nearly as efficient as if you said, "Okay. Hey, here is the state and I would like you to run multiple different computations on it over time." And so being able to hold applications state at the edge as well for like mobile offload I think is like going to end up being one of the bigger use cases as well.

[00:25:31] JM: What kinds of programming languages are you using to build this infrastructure at the edge? Like to build serverless functionality and to build database functionality at the edge?

[00:25:41] TM: Yeah. So Fastly has kind of landed mostly on Rust and some Go at this point to like answer your question directly. And the reason that we chose Rust in particular really comes down to two things, and it's like performance and security, and performance and safety even. Rust gets us close to or just as good as or in some cases like better than the performance that we had previously gotten when writing C for our server applications, but so much safer. It also allows us like a lot more flexibility in like the ways that we build applications. That means that we can hire engineers who can make reasonable abstractions rather than having to work in like the lowest levels of languages.

So yeah, Rust is my primary answer to that question, and I'm very pleased with that choice. I was kind of pushed into that choice a number of years ago. I didn't think it was ready at the time, but listen to the engineers in your team. They probably know what they're talking about. And I'm really glad that we made that choice. I think that it has definitely borne fruit in terms of like being able to build on and like continue iterating on the applications that we build or rather the platforms that we build at the edge safely, right?

[00:26:53] JM: And what are some of the big engineering struggles you've had in building out that serverless functionality?

[00:26:59] TM: So I think the most interesting – We've had all sorts of challenges and opportunities and wins and losses here. But, to me, probably the most interesting one relates to fairness. Fairness and like what I actually described as scheduling, right? When you have like such a wide variety of different types of applications running on a platform where you have some that may be done in like under a millisecond and you have others that take 200 milliseconds to run, we're talking like multiple orders of magnitude variation in the types of workloads that are running on top of this thing. And, to me, one of the trickiest problems and one of the things that I think we spent a lot of time on trying to like nail it down really well is how you maintain fairness and as close to real-time nature in a system like that. And it turns out that there is just tremendous amounts of like academic research into this area. How you run heterogeneous workloads across systems like this.

I think we've come up with some interesting approaches, and some of that's going to end up being open source as part of Lucid and Wasmtime now as well. But, to me, that's really the trickiest thing when building out a platform like this, is how do you handle like so many different users with so many different needs and so many different types of applications? It's particularly tricky. And of course there's also all the things that typically go into building out a system like this, which is maintaining safety as well as maintaining performance across the whole thing as well, right? And so we put a ton of effort into this stuff. That's where most of our effort goes, I would say, is like making this stuff completely bulletproof. And I think we're getting there. We've done a pretty good job of this.

[00:28:49] JM: How do you expect to see serverless evolving over the next five years?

[00:28:55] TM: So this is going to end up going back to the things we were talking about about like how to prevent stalling in the serverless like universe. Because I think, honestly, that if we don't address those things, well, we're kind of just going to end up being stuck if the industry as a whole doesn't address those problems. So I'm going to go back to those again and say like, “Look, in the next five years we've got to see much wider programming language support for one thing.” To me, if we have to keep talking about the programming language support five years from now, we have lost the game, right? The industry as a whole has lost the server game if we're still talking about like how we can support more programming languages five years from now. So that's one thing that has got to be fixed. That's one thing that we definitely have to see evolve.

I think that like that goes hand in hand with vendor lock-in, right? If every vendor is supporting different sets of languages, well, there's always going to be vendor lock-in of some sort. Again, like this also comes down to the interfaces, the APIs that are available within that. We need to agree on these things. I think the other thing that we're going to see is like addressing the performance problems, right? I think that's going to be a big change, because those who have good performance, those who don't make your users wait 20 seconds for cold start time startup, you're like they're going to win. Like if we can do that, well, we're going to win. And so like that's one thing that I think is going to have to evolve, right? It's going to evolve along because that's what users actually care about. They care about, again, like going back. Like programming language support, they care about vendor lock-in, they care about performance.

And I think the other thing that's going to change is we as an industry are going to push more and more features out there as well. Serverless is still in some ways kind of a low-level environment. You're still kind of like wiring things together yourself when you actually start to develop applications especially if you're trying to develop large applications at the edge, or not even at the edge, on just serverless in general. You're kind of like, "Okay, picking and choosing. Well, here, I need to use this, this component and that component. And how do I break up my application so it's multiple functions?" I think that the big evolution there is going to look like improved user experience for this whole thing. I think that those like three things are going to end up being like the big evolutionary changes over the next five years, is like programming language and vendor lock-in as well as performance improvements. And I think like the user experience side of this as well. That's definitely like what we're heading toward at the very least. That's something I say to our engineers on a regular basis, is that like you have to imagine how engineers are actually going to use this thing, right? Like how do we make this feel natural? How do we make this feel as if like you're running this on your own laptop as opposed to running it across the entire world simultaneously? It's all going to have to come down to user experience.

[00:31:59] JM: How is vendor lock-in avoidable for serverless platforms?

[00:32:03] TM: So I think the answer to this is going to have to come down to standards, right? We're putting a ton of effort into this already. We have like cross company, like cross industry support. Lots of people working together on things like quasi, for instance. But standards by themselves don't actually solve the problem entirely. Like standards need adoption for it to actually matter, right? Like you actually have to go out and talk to everyone and get everyone behind it, right? I think that's a lesson the computing industry has learned pretty hard over the last like 40 years. There are lots of standards out there that just no one actually supports. No one actually uses them. And I think that's kind of the fault of like the folks who implemented those standards, right? You can't just say like, “Oh, this is the standard,” right? You have to actually go out and talk to people and convince them that this is the right way to do it and listen to their concerns, right?

And so if you look at the way that, for instance, like the WASI standard is evolving over time, it's taking time, right? We've been talking about this for a couple of years now, and there's a decent amount of support. But like the reason it has to go slowly is because we need folks to actually support it. We need people to be behind it for it to actually matter. And I think that's a lesson that like a lot of the folks who are working on it have learned over the years. But yeah, I think that's the answer, right? It's got to be standards and it's got to be like widespread industry support of those standards. And I think that folks are now like kind of seeing that that is going to be necessary. Like those base level APIs, those interfaces that we provide, like that's not the thing to compete over. That's not the thing that users like are going to want to – The users are actually going to want us to be competing over, right?

To me this is one of those like a rising tide raises all ships sort of questions, where if we can create the baseline for users to work off of, for us all to work off of, then great. We can go compete over the features, over the actual like tangible things that users are using to put together their applications. But, again, that base layer infrastructure, that's not the place to compete, right? So I think standards are going to play a role in this and just industry collaboration is going to play a role in this. And to me it's totally feasible. It's just a matter of getting folks going in the same direction.

[00:34:23] JM: All right, Tyler. Well, anything else you want to discuss or explore around serverless?

[00:34:27] TM: I think I'm good for today.

[00:34:28] JM: Cool. Well, thanks for coming on the show. It's been a real pleasure talking to you.

[00:34:31] TM: Thanks a lot. I appreciate being here.

[END]