EPISODE 455

[INTRODUCTION]

[0:00:00.3] JM: Serverless architecture is software that runs without an addressable server. Serverless is made possible by two types of technology. Platform is a service providers like Auth0 and Heroku and Firebase and functions as a service like AWS Lambda.

With both of these technologies, we can program logic that runs without being deployed to a server. By using platform as a service together with functions as a service, it really becomes a bright future in terms of how easy it is to manage our infrastructure.

Functions as a service are cheap and scalable. Write your code for a serverless function and the cloud provider will cheaply deploy and execute that function on some server somewhere. The difficult part is maintaining state.

Since serverless computer instances are ephemeral, you are not dealing with a system that will keep track of your state. It's going to disappear eventually. The ephemeral nature of serverless code requires us to shift our thinking, but the dramatic cost and simplified scalability make it well worth the effort.

Serverless functions can add complexity in exchange for a lower price and excellent scalability. Serverless platform as a service offers lower complexity at a slightly higher price. A serverless database like Firebase handles database scaling and gives you a nice web interface.

A serverless machine learning platform like Google Cloud ML gives your models scalability and controlled deployment. A serverless authentication service like Auth0 manages your authentication. In addition to authentication, Auth0 has built a set of tools to allow SAS companies to extend their platforms into a sandbox code execution environment.

Bobby Johnson is an engineer at Auth0 and he joined the show to describe the toolbox that Auth0 has developed; authentication, web tasks and extensibility. We also talk about how the world of serverless architecture is evolving.

Full disclosure, Auth0 is a sponsor of Software Engineering Daily.

[SPONSOR MESSAGE]

[0:02:27.2] JM: Simplify continuous delivery with GoCD, the on-premise, open-source, continuous delivery tool by ThoughtWorks. With GoCD, you can easily model complex deployment workflows using pipelines and visualize them end to end with the value stream map. You get complete visibility into and control over your company's deployments.

At gocd.org/sedaily, find out how to bring continuous delivery to your teams. Say goodbye to deployment panic and hello to consistent predictable deliveries. Visit gocd.org/sedaily to learn more about GoCD. Commercial support and enterprise add-ons, including disaster recovery are available. Thanks to GoCD for being a continued sponsor of Software Engineering Daily.

[INTERVIEW]

[0:03:26.8] JM: Bobby Johnson works at Auth0 as a developer evangelist. Bobby, welcome to Software Engineering Daily.

[0:03:33.3] BJ: Hi, Jeff. Thanks for having me.

[0:03:34.6] JM: Let's start by talking about authentication at a high level. We're going to get into serverless and its relationship to authentication. But just talking simply about authentication, why do application need authentication?

[0:03:50.5] BJ: Applications need authentication to ensure that code is executed by people who have authorization to execute that code.

[0:03:58.4] JM: If I have a basic website with a username and a password login, how does authentication work?

[0:04:05.7] BJ: Sure. Well, that's a pretty heady question. It really depends on how you have your application setup, but typically you have a username and a password that's sent to some back-end service that validates those username and passwords, identify you uniquely within the system.

[0:04:25.6] JM: Let's go a little bit deeper. What's going on? What's the contract between the client and the server in just a basic authentication pattern with actual servers, the kind of thing that we've been dealing with for the last 20 years?

[0:04:37.8] BJ: Sure. Username and password is sent by the client to the server. That server then checks the username and password are valid.

[0:04:47.3] JM: After I'm logged in, how does an application keep track of whether I'm logged in and who I am?

[0:04:54.1] BJ: An application once the authentication request is complete, it's some form of token is passed back to the client, that you'd subsequent request passes that token back to the server. The server knows on each request who you are basically via that token.

That token can have varying levels of complexity. It can just be a simple session token that identifies a session that's open on the server, or it could be a JWT full of claims of who you are and what you're allowed to do within the system.

[0:05:33.8] JM: Tell me more about the varying of complexity. What are some different things that our authentication token can have?

[0:05:39.3] BJ: Sure. Authentication token could identify your authority, so who says you are, what you are. If I'm authenticating, like maybe using my Google or Twitter or Facebook accounts, that authority may be a portion of that token. Information about you can be encoded into that token, so your e-mail address, your name, that sorts of thing, and maybe claims related to the specific application that you're using. Are you an administrator? Are you allowed to create things within the application?

[0:06:16.5] JM: The client maintains the authentication token, is that right?

[0:06:20.0] BJ: Well, the client maintains the authentication token in terms of handing it back with each request and managing any type of maybe refresh token scheme that might be involved in authentication. Potentially, the token could be handed back to you with a expiration of three minutes. After three minutes or slightly before three minutes, you want to request a new token to keep the session live.

[0:06:48.4] JM: Tell me more about how that management of authentication tokens works and how often those tokens get invalidated by a server.

[0:06:57.0] BJ: Sure. That's based on configuration of your provider and how you want that set up. It dials that you can tweak. It's getting a little maybe into the weeds on the identity product at Auth0. I am actually a evangelist for the extend and Webtask products at Auth0, so going any deeper than basically what I've explained is maybe a little outside my –

[0:07:25.3] JM: Sure, no problem. Then we'll dive into the Auth0 related stuff shortly. But I guess talk a little bit more about just basic authentication. What's preventing somebody from stealing my authentication token and just authenticating as me?

[0:07:40.5] BJ: Sure. Well, typically you want all that traffic going over HTTPS secure channel that doesn't allow any type of man in the middle to get access to that token. The token is typically also encrypted by the service itself, so only it can decrypt that token to get out the values.

[0:08:03.6] JM: When I use a external provider like Twitter or Facebook authentication to some external app, how does that work? Is that any different than the basic password login that we're talking about on authentication?

[0:08:17.9] BJ: Yes, absolutely. Typically, you are handing off a portion of your authentication flow to that provider itself to do the login portions and then provide the token back out. That communication happens both between the authentication provider, your client and your backend services. There is communication line between Twitter and the back-end service to ensure the token issuing is secure.

[0:08:46.9] JM: We have done many shows about serverless technology, but we've not focused on identity. If people are totally unfamiliar with serverless, you can go back into some previous shows. Serverless is a broad term, just to refresh people's memory.

You can be talking about a service like Heroku or Firebase where you have – I guess with Heroku you still have the notion of servers. But with Firebase, for example you'd have an abstraction that represents a database and you don't have to manage the servers that underlie that database.

It's an addressable server. It's more like a blob of storage essentially, same thing for Amazon S3, or Google cloud machine learning where you're just interfacing with this service and some opaque inpoint and you're not addressing servers.

Then on the more contemporary level, serverless means functions as a service; Amazon Lambda, Google Cloud functions, which are executable blobs of code that will be scheduled onto some machine on an Amazon or Google data center. It will be executed cheaply, but since you're not addressing a server, the underlying infrastructure might shift around.

We get these benefits out of serverless functions as a service like scability and low cost, but we have some issues, because we no longer have a server as an addressable entity. Have I described the world of serverless in a way that matches your description of it?

[0:10:32.6] BJ: I think you did a pretty good job of describing it. I think the big difference between using a service like Heroku and using a serverless service is you're deploying to Heroku an application as a unit, a set of functions that interact together. When you go to scale out on Heroku, you're scaling the entire application itself. Whereas, a function as a service platform allows you to do that scaling at a much more coarse level.

I can deploy several functions that are targeted and very narrowly targeted to do one thing and one thing well. I can individually turn the dials for each of those to scale up as necessary. With the ultimate goal of I'm only paying for the resources that I'm using, not paying for an application to be sitting out an available, whether it's processing information or not.

[0:11:32.1] JM: In the model of authentication that we explored, you've got this relationship between a client and a server, where the client sends a username and password to login to a server and the server responds with an authentication token, and they maintain a little relationship with each other.

If you have this shifting sands of underlying infrastructure, where the server that you're communicating with might essentially disappear, because the code gets scheduled away from it. It seems like we have to figure out a different way to manage the authentication. Am I explaining things correctly?

[0:12:09.8] JM: Yeah. In terms of at least Webtasks, that authentication is really kind of up to you how deep you want that to go. You can do a simple shared key type authentication where each request hitting an endpoint has to match a key that's defined within the Webtask itself, or you could rely on an external provider like Auth0 to do that key validation for you.

[0:12:36.9] JM: Can you go deeper on those two options? If I am building a service where I want to use functions as a service, these ephemeral blobs of compute, I want to take advantage of these, but I also want to have identity associated with my application. Describe those two once again in a little more detail.

[0:12:54.8] BJ: Sure. Basically, the two options I was describing is going very simply with just a shared key type of authentication scheme, where say you put a function out into the service that you want to ensure that only people who are authorized to call it are authorized to call it. You provide that person with a key that they pass with every single request.

I'm talking about this in kind of a manual process. But then you could also hook into Auth0 as an identity provider to take whatever key that is sent by the client to validate that it's a valid key in terms of Auth0's identity product saying, "Yes, this is a valid key." Am I saying that clearly enough?

[0:13:45.8] JM: Yeah. I understand. So talk in more detail about how Auth0 handles identity management, because this is essentially identity as a service, and we're going to go into a little bit more detail in a sec. But just give a overview for what the product does.

[0:14:02.8] BJ: Sure. Auth0 identity basically is a drop-in component to your application that handles all authentication and identity for you. It allows you to merge in any type of identity provider that you would like to consume within your application, like we talked previously about social logins like Facebook and Twitter, but we can also consume sample logins, traditional username and password logins, and we're managing all of that for you toward none of that data is stored in your application.

There is the old chestnut about never try to roll your own crypto. The same could be true about never trying to roll your own authentication and identity. We will handle that for you in a secure manner, and we have some of the best engineers in the world working on that to ensure it's the highest quality possible.

[0:15:06.6] JM: If I am using Twitter login though – Let's say I want to build an app that has Twitter login, why wouldn't I just use Twitter, because in that sense Twitter is being my authentication provider? How would Auth0 differ from that kind of outsourcing of an authentication service?

[0:15:30.0] BJ: Sure. It's relatively easy to setup authentication through Twitter directly in your application. But you have to wire up all those callbacks and handlers yourself just for Twitter. Now you want to implement Facebook, you have to go through the same set of tasks and boiler plate code to implement Facebook as well.

With Auth0, you can basically go in, flip a few toggle switches, add a couple keys and all of that infrastructure is in place for you already. It's separated from your application code. If six months from now you want to come in and you want to add, additionally say as your AD-based authentication. You would do that in the Auth0 dashboard, and it wouldn't necessarily affect your application in any way other than the ability to be able to login without your ID and the application continues working the way you expect it to.

[SPONSOR MESSAGE]

[0:16:35.1] JM: At Software Engineering Daily, we need to keep our metrics reliable. If a botnet started listening to all of our episodes and we have nothing to stop it, our statistics would be corrupted. We would have no way to know whether a listen came from a bot or a real user. That's why we use Incapsula, to stop attackers and improve performance.

When a listener makes a request to play an episode of Software Engineering Daily, Incapsula checks that request before it reaches our servers and it filters the bot traffic preventing it from ever reaching us. Botnets and DDoS attacks are not just a threat to podcasts, they can impact your application too. Incapsula can protect API servers and micro-services from responding to unwanted requests.

To try Incapsula for yourself, go to Incapsula.com/2017podcasts and get a free enterprise trial of Incapsula. Incapsula's API gives you control over the security and performance of your application and that's true whether you have a complex micro-services architecture or a Wordpress site, like Software Engineering Daily.

Incapsula has a global network of over 30 data centers that optimize routing and cashier content. The same network of data centers are filtering your content for attackers and they're operating as a CDN and they're speeding up your application, but doing all of these for you and you can try it today for free by going to incapsula.com/2017podcasts and you can get that free enterprise trial of Incapsula. That's Incapsula.com/2017podcasts. Check it out. Thanks again, Incapsula.

[INTERVIEW CONTINUED]

[0:18:24.2] JM: Services that have multiple authentication types, like if I have authentication with Twitter and authentication with Facebook and Google and all these other – it's a bunch of enterprise login systems that we could also explore. What is the database of record for all these authentication types? How does that look?

Do we try to create a N-to-1 mapping, where there is a bunch of different login schemes that any user can login with, or do you try to maintain a one-to-one mapping where if a user logs in with Twitter and then they log in with Facebook, you give them two different identities? How do you typically do that?

[0:19:06.6] BJ: Well, usually we need some point of reference to be able to merge identities like that in the backend. That is typically keyed on your e-mail address. When you authenticate through Twitter, or you authenticate through Facebook, one of the demands we ask in that authentication cycle is give us your e-mail address.

Then we can search through our back-end systems for other accounts typically or not accounts, but identities that have that same claim. It's up to you in the dashboard and Auth0 to decide whether you want to merge those two things together or not.

[0:19:44.5] JM: I want to go from here to talking about web tasks. I think the place to start there is with WebHooks, then we're going to get into Webtasks, then we'll talk about how that relates to serverless and authentication. This is how we can start to talk about some usability layers that are being built on top of the serverless infrastructure.

Obviously, the value of the serverless infrastructure is that it gives you on-demand, really easy scalability and much lower cost. But with that has come some usability issue. Let's start with WebHooks. Explain what a WebHook is.

[0:20:26.1] BJ: Sure. A WebHook is a very elegant solution to a very old problem. That problem is when you put a software as a service application out on the internet for other people to come consume that application, eventually they want to use your application not just in a context of that application itself.

We use several different SAS offerings on the internet within a company itself, and we want those things to be able to communicate with each other. The example of maybe Github when I have an issue that's logged in Github, I want something to be able to pop-up in Slack for me to let me know that, "Hey, an issue was just filed on Github and I need to go address it."

A WebHook allows you a very simply method to subscribe to those events and have a payload be shipped across the internet to some inpoint to handle that event happening.

[0:21:31.3] JM: Describe the client-server relationship for a WebHook in a little more detail and maybe you could give an example.

[0:21:38.5] BJ: Sure. In the example that I just mentioned, filing an issue with Github on a repository. In Github, you would go in and actually create a new WebHook provided a URL of where to send that payload. Then it’s up to you to implement wherever that URL is going to resolve to and handle the payload coming in to your infrastructure.

You might be lucky and that the other service that you’re wanting to kind of wire together or glue together already has baked-in functionality for handling Github WebHooks, but it might not as well. You’re at the mercy of the two services for communicating together, unless you want to implement it yourself and stand up an endpoint that’s available on the web, secure it yourself, maintain it and deal with any monitoring or scalability issues for servicing that WebHook call.

[0:22:39.6] JM: How does a WebHook compare to a system where I would make a remote call from my client to the server and then just pull the server? Because that seems like an alternative to this making a request and then waiting for a call back.

[0:22:57.7] BJ: Could you state the question again? I didn’t –

[0:23:00.9] JM: Basically, there is two ways of getting information from a server that takes some time, so that you could have your client make a remote call to the server and then pull the server and just keep asking it like have you finished the task yet? Have you finished the task yet?

Or you could make a call to a server that has a WebHook, and then once the WebHook finishes processing, it can call back to the client. Am I kind of describing two different ways that you could have this kind of interaction? Is that the problem that the WebHook solves is the pulling problem?

[0:23:36.6] BJ: Yeah, I believe so. Instead of constantly hitting Github to say what issues are there, are there any nuance you can actually have Github notify you of new issues coming in, and that's fundamentally easier to work with. It takes less resources.

[0:23:55.0] JM: What are the challenges for the provider of a service with ooks? Is there anything difficult about setting up WebHooks on your server?

[0:24:04.6] BJ: Provided like say for Github does WebHooks present a quandary for them?

[0:24:10.8] JM: Yeah, like a manage – is there any difficult management issue?

[0:24:13.1] BJ: I wouldn't think there would be difficult management issues for the WebHook issuer, being able to fire off a WebHook. It solves a problem for them of my customers want to be able to customize the service that I'm offering, so I give them a way of being able to respond to events within my system. But all of the burden is placed on the customer themselves of how to actually do that response.

[0:24:43.5] JM: Okay. Are there any usability issues with WebHooks that are worth discussing before we dive into Webtasks? Like the shortcomings of WebHooks that perhaps the Webtasks ended up solving.

[0:24:56.3] BJ: Sure. I think what I just said is the major red flag for me, WebHooks are brilliant solution like I said to a problem that we've been dealing with since the 90s or maybe even before. That's customizing a software offering for individual clients who want to be able to consume your application.

But WebHooks really solves the problem for the SAS itself. It doesn't necessarily solve the complete problem for the customer that's using the product. The end-user has to consider how they're going to consume the WebHook calls in their own infrastructure, maintain that infrastructure and any cost associated with doing that, including internal organizational cost of developing new functionality of responding to the WebHooks. Am I saying that clear enough?

[0:25:54.6] JM: Yeah, that makes sense. To explain to people further, a Webtask is a WebHook with a server included. It's essentially a serverless WebHook. Explain what that means.

[0:26:06.1] BJ: Sure. We like to use the term serverless extensibility, and it's a term that we coined to described a pattern where SAS providers take advantage of a serverless platform, like as your functions Lambda to securely execute for the purposes of allowing customers to extend functionality of the SAS product.

Some really good examples of that are Twilio functions and Auth0 rules. The ultimate goal of serverless extensibility is to remove that burden that I've been talking about that WebHooks place on your customers to implement and maintain application needed to process the WebHook calls. Instead, you take that burden on for your customers by providing a secure execution platform directly as a part of your product.

[0:26:59.4] JM: Can you give an example of that? Explain that in more detail.

[0:27:02.5] BJ: Sure. For instance, Twilio functions offers – or Twilio if you're familiar with it is a service that allows you to basically buy a phone number and then offers an API to be able to programmatically respond to phone calls or text or that sort of thing.

They offered WebHook functionality, just like Github does. When a call comes in, a WebHook is called with information about that phone call and whatever you choose to implement that the other end of that WebHook call can respond to the phone calls or text messages and you can build pretty complicated kind of phone tree sort of systems.

Brilliant product. Twilio is an amazing product. But it put that burden on you to be able to host the application yourself that's going to be doing the interaction with the application. They created this new feature offering called functions, which basically allows you right within their dashboard to wire up the call back and response system running in their server list environment.

[0:28:13.5] JM: What's an example of an application that I would deploy on a Twilio function?

[0:28:18.5] BJ: Sure. Chatbot over SMS. I send a SMS message to a specific phone number, Twilio gets that message and inside their infrastructure bundles up just like you would with a WebHook, but sends it off to a function you predefined to execute whatever logic you'd like to execute and then respond back to the text message with another SMS message.

[0:28:47.9] JM: I see. In contrast, if I were setting up my – Basically I'm like, I want to have a chatbot service where I can order a pizza via text message and if I wanted to build that application without Twilio functions, I would be building that – deploying it to a server somewhere on AWS, or Heroku, or posted up my own server and basically I could do – use Twilio for the authentication of phone numbers and have my users send text to a number that I have assigned on –

I've set up a phone number on Twilio's infrastructure, my users send a message to that phone number to order a pizza, and Twilio manages the request response of the phone number, but then I have to handle those messages with my own server and in contrast with – I was using Twilio functions, they would be hosting the code and then they would scale up or down the code for actually processing the pizza order.

[0:29:55.6] BJ: Exactly. Yeah, you said it very well. WebHooks, you know we've already said is a very good solution to a problem. The problem that we're trying to solve with serverless extensibility is the other side of the WebHook. So once the WebHook calls something, what is that something you're calling? Can we bake that thing directly into our platform, so that it doesn't put the burden on you as a customer to have to go and create AWS account, or create a small application to service those WebHook calls

[0:30:30.0] JM: The Twilio example – Twilio is not actually using Webtasks, right? They're using basically something that is similar to a Webtask, but you're giving that as an example for the kind of functionality where you would want to couple the API response with the provider. I guess what I should ask is are there direct examples of people that are using Webtasks themselves to give a more direct example of Webtasks?

[0:31:07.0] BJ: Yeah, sure. Webtasks in Auth0 actually grew out of our own identity product in terms of Auth0 rules. When a person authenticates using the identity product, a lot of our customers wanted some type of WebHook-like functionality to be able to execute some bit of logic on every single login.

Similar to what Twilio did, they had a similar problem. We built in-house a platform for being able to create these customizations and execute them securely. These customizations being custom untrusting code and being able to keep them in isolation away from other customer instances.

The rules was incredibly popular for Auth0 and really helped us get a lot of success with our users. So we thought if this was a benefit to our customers, we could carve it out and make it actually a product into itself, and that product that I work on the team, myself and my team work on is called Auth0 extend.

[0:32:19.9] JM: Fascinating. Okay. The idea is you had these Auth0 rules where every time somebody authenticated using Auth0 – just to remind people, Auth0 is basically the service that takes the issues of authentication out of your purview and it makes it quite simple and Auth0 maintains your identity service and gives you lots of different ways that your users can authenticate, whether it's Twitter or Facebook or these other enterprise login services office 365, whatever.

When your users log in, maybe you want to do something like, "Let's gather telemetry data on a user." Let's say, after they authenticated for the first time, we want to also send a ping back to – we want to take their IP address and run it through some check and get more telemetry data on where they are.

Or we want to send a message to some other logging server that we have to increase the user counts that have logged in today. We just want to have messaging or notifications or updates that are tightly coupled with people authenticating.

[0:33:43.6] BJ: Yeah, definitely. The use cases are unlimited, right? The things that our cus would want to do at the time of a login attempt are unique to each individual customer. It's customization specifically for them. You could imagine scenarios like wanting to white list domains for e-mail addresses where we're only going to allow authentication through for example.com, or we want to black list certain API ranges, or we want to have a Slack message be posted every time someone a new sign-up happens in our application, or a person authenticates within our application.

Maybe that's a silly example, but there are customers out there who want to be able to do those sorts of things. The question is do you bake each one of those request into your application directly, or do you give an extensibility point to your customers to be able to do that themselves specific to their tenant basically within Auth0, and make it as easy as possible for them to go in and write that code for whatever specific use case they have and execute it securely so that it doesn't affect any other client within Auth0 as well.

[SPONSOR MESSAGE]

[0:35:13.2] JM: DigitalOcean Spaces gives you simple object storage with a beautiful user interface. You need an easy way to host objects like images and videos. Your users need to upload objects like PDFs and music files. DigitalOcean built spaces, because every application uses objects storage. Spaces simplifies object storage with automatic scalability, reliability and low cost. But the user interface takes it over the top.

I've built a lot of web applications and I always use some kind of object storage. The other object storage dashboards that I've used are confusing, they're painful, and they feel like they were built 10 years ago. DigitalOcean Spaces is modern object storage with a modern UI that you will love to use. It's like the UI for Dropbox, but with the pricing of a raw object storage. I almost want to use it like a consumer product.

To try DigitalOcean Spaces, go to do.co/sedaily and get two months of spaces plus a $10 credit to use on any other DigitalOcean products. You get this credit, even if you have been with DigitalOcean for a while. You could spend it on spaces or you could spend it on anything else in DigitalOcean. It's a nice added bonus just for trying out spaces.

The pricing is simple. $5 per month, which includes 250 gigabytes of storage and 1 terabyte of outbound bandwidth. There are no cost per request and additional storage is priced at the lowest rate available; just a cent per gigabyte transferred and 2 cents per gigabyte stored. There won't be any surprises on your bill.

DigitalOcean simplifies the Cloud. They look for every opportunity to remove friction from a developer's experience. I'm already using DigitalOcean Spaces to host music and video files for a product that I'm building, and I love it. I think you will too. Check it out at do.co/sedaily and get that free $10 credit in addition to two months of spaces for free. That's do.co/sedaily.

[INTERVIEW CONTINUED]

[0:37:32.6] JM: How does that differ from a raw function as a service? If I'm deploying code to run in a Webtask, well let's say to run in Auth0 rules for example, which is you're basically – Auth0 rules was this thing that was the first Webtask essentially and you use that as a sort of validation that this more general idea of code that you would want to run in response to an Auth0 login.

This could be abstracted into code that you would want to run alongside an API service, like Auth0 or Twilio or Salesforce, these other popular SAS platforms that have turned into APIs, or they started out as APIs for developers in the case of Twilio. When you want to couple this functionality that's going to run in response to something that SAS provider, that platform and the service provider is doing and you're talking about its abstraction Webtask. How does that differ from just deploying that code on raw functions and service like AWS Lambda?

[0:38:39.3] BJ: Sure. Deploying that code on something like Lambda requires you to know how to do that. Whereas, the rules system within Auth0's dashboard itself is baked right into the UI, right there in your account. You get an editor right in your browser where you define the logic in javascript right there, and we do all the management of deployment scaling, maintenance and those things for you.

It's like I said, removes a lot of burden on the customer from using our service, which makes them happy and they tend to stay with our service, The benefit for SAS providers is that Webtask and extend both provide more stickiness within your platform than traditional WebHooks.

[0:39:28.1] JM: Do Webtasks have the cold start problem that the functions as a service have?

[0:39:34.6] BJ: Well, Webtasks, the Auth0 Webtask technology that we've developed is very focused on execution with high fidelity, HP fidelity. We go to great lengths to reduce the cold start execution.

[0:39:50.7] JM: Explain the cold start problem, for you who don't know, who are not familiar with the serverless cold start problem.

[0:39:55.6] BJ: Sure. Cold start, what we mean by that is that we don't – a request comes in to an HTTP endpoint and there isn't necessarily something they're waiting to respond directly to that HTTP request. We have to set up some form of infrastructure to be able to properly handle the request. With that time is can't take – I'm explaining this horribly.

[0:40:27.2] JM: Let me see if I can word it correctly. I know this stuff can get muddy. So AWS Lambda for example, I'm going to deploy my code to AWS Lambda, and what that means is that whenever an event triggers the Lambda function to run, my code is going to get loaded onto a container somewhere on Amazon's infrastructure and it's going to get executed.

I don't know where that container is, I don't have an ability to address it, I don't know how long it's going to be standing up, but it's going to address the response, it's going to address my request and it's going to send me a response, and then it's going to be around for some time and I can ping it with more requests.

But that first ping to the function as a service has a cold start problem, because the code that I've loaded into a Lambda function has to get – I mean, the code that is basically hosted on Amazon's servers at that point has to actually get loaded into a running container in order to execute, because the difference with functions of the service versus actual addressable servers is that the code is not always in memory waiting to get executed for you.

It's like a function as a service. It's a function you can call on demand. This cold start problem could be an issue if you've got – if you want to run stuff that's super responsive on functions as a service.

[0:41:57.2] BJ: Very well put. Yeah, that as a great explanation of the cold start problem. The way we go about solving that is that we've always got containers that are spun up and ready to accept a call. If a container is not set up to handle a specific function already, we have a method to quickly deploy that code directly to one of those hot containers and respond to the call quickly.

Because our focus is on execution in an HTTP scope or request, we need to keep that cold start functionality very low latency. We try to make that cold start functionality happen in hundreds of milliseconds, instead of potentially a couple minutes.

[0:42:51.0] JM: It's pretty interesting. You think about serverless and just how dramatic the cost reductions are. When we're talking about that raw serverless, like AWS Lambda or GCP, and in that huge margin of cost reductions, there's this opportunity for all these other providers such as Auth0 to stand up a little more domain specific serverless functionality.

Basically, you can offer serverless at a slightly higher mark-up to raw AWS Lambda or Google Cloud functions, but it's still dramatically less than what it would cost to run on your own servers. I think it's an interesting market that is cropping up between the actual compute and the raw serverless functions.

[0:43:52.6] BJ: Well, while Webtask can be used to do similar things as other fast platforms like you're saying, we like to think that we are fundamentally trying to solve a different problem. In that problem is the serverless extensibility concept that we were talking about earlier.

[0:44:09.4] JM: Okay. Describe that in more detail, because I know you work on Auth0 extend. What is that serverless extensibility you're talking about?

[0:44:16.2] BJ: Sure. Serverless extensibility is a term we coined to describe a pattern where a SAS provider takes advantage of a serverless platform to securely execute code for the purpose of allowing their customers to extend the functionality of their SAS product.

[0:44:33.6] JM: It almost seems like a sidecar. It's like I'm Salesforce, today I have an API that you can use to build applications on top of. But if I give serverless extensibility, it takes more burden – it gives Salesforce the ability accomplish more than on the Salesforce side of things than in the current model where the client has to perhaps process the API request with more – Am I describing that correctly?

[0:45:13.0] BJ: Yeah. The way I tend to look at it is that you're lowering the barrier of entry to customizing your platform or your SAS product, to the point that you don't necessarily need developers in the backend developing applications that process those WebHooks.

You could potentially have, I don't want to say lower-skilled people, but not necessarily a developer going and customize how your platform behaves to certain events, right? We're lowering the barrier of entry by removing some of the burdens that might require more costly solution.

[0:45:53.9] JM: Can we give another example? Because I think the – this is kind of a hard topic for me to understand personally, so I'm going to assume that there is some difficulty for the listeners to understand it.

Is there another example we can give? We gave the example of the pizza delivery service where basically you take some of the effort out of the client and put it on to the server. You basically did the same thing with the example with Auth0's – what the trigger? The thing that became Webtask, it was the original rules?

[0:46:26.7] BJ: Auth0 rules.

[0:46:27.5] JM: Right. Okay, so that was another example.

[0:46:29.4] BJ: Well, you brought up Salesforce, right? Salesforce is a customer relationship management kind of platform application. You could imagine if you were Salesforce and you were trying to sell this software to Fortune 500 companies or other customers, those customers start to come up with the requirements of things they would like Salesforce to do for you.

Each customer would have their own unique set of requirements, of functionality that they would like to happen. If you're a CRM, let me give you a couple examples of things you might want to do. You're adding a lead into the system and customer A would like to have some logic execute when the lead is added that if the value of that lead is over say $50,000 we'd like a message to be sent directly into Slack.

But customer B, what they would like to do when a lead is put in to the system is to maybe call out to another service to gather or enrich some data about that particular data. Maybe all you have is a name and an e-mail address, so you'd like to reach out to some service that would maybe based on that e-mail address give you a Twitter handle, a Facebook URL, potentially any public address or something like that associated with the e-mail address.

Have that enriched into your lead data model, and then stored within the system itself. Webtasks work really well for that type of glue code between services to get that business done. Similar to maybe you're familiar with Zapier, where you can use Zapier to bridge or glue two different services together through a visual means, Webtask or a more developer-focused offering than Zapier. That's just one potential use case, writing that gluco between services and enriching data.

[0:48:36.4] JM: If I'm just running all of that stuff, if I'm managing all of that code myself instead of putting into some kind of sidecar, or put again into a Zapier service – just zoom in a little bit more on why that's harder to do. If I'm the developer and I don't have access to these serverless types of management things, why is that harder?

[0:49:00.9] BJ: Sure. Well, I mean it's the context of your business. Do you want to spend resources and time doing these types of integration? Or do you want to focus on your core business and devote all of your resources to that?

[0:49:17.6] JM: The Auth0 extend product, so that's built on Webtasks, right?

[0:49:23.9] BJ: Yes. Extend is our premium server list extensibility product based on Auth0 Webtasks. It consists of a backend infrastructure that securely handles execution of your custom untrusted code in isolation, but it also offers a white-labeled embeddable editor that is fully customizable and themable, as well as any monitoring or maintenance that might be needed for that backend infrastructure. We handle all of that for you.

[0:49:55.6] JM: Okay. I see. For a SAS company, what you're really getting out of this is the – I think that white-labeled editor, that sounds pretty cool, because like if I'm Salesforce, I can just now have a thing where people can drop in code that can execute in a way that is – I mean, it's flexible because it's any code can execute.

I assume Salesforce would be able to tailor it to respond in ways that are domain-specific to Salesforce. Salesforce could define a name space or something that a request could respond to.

[0:50:34.3] BJ: Sure. Yeah. I like to think of – we talked about Twilio functions earlier, I like to think that if extend was in existence when Twilio was working on that, we could've made a play to say, "Twilio, why don't you use our platform instead of building all that yourself inhouse." That's primarily the market we're going after with extend.

The companies that have recognized this problem within their own platform and are making the choice of do we solve it by dedicating our resources to building it, or do we look at something like extend that is a drop-in serverless extensibility platform?

[0:51:12.6] JM: It's a brilliant product. It seems nascent. I'm very interested in this product, but are there a lot of other people that are interested in it at this point, or does it feel like, "Oh, we're a little bit early and this is a market that's going to develop," or is the market already popped in?

[0:51:29.9] BJ: Sure. We've got a nice funnel of interested parties and a handful of very engaged customers. But you're right, serverless extensibility is a new concept that is starting to emerge out there with Twilio functions being a primary example, but there are other examples out there.

We've talked about Auth0 rules. There is also stand play, which is doing a similar type of serverless extensibility within their product offering. So companies are recognizing that they need to go beyond WebHooks to entice more customers to come to their platform and use it.

WebHooks is like we said, a brilliant solution to an existing problem we've all been working with. We're attempting to make WebHooks better and go beyond that to like I said, lower the bar for people who are wanting to engage with these services.

[0:52:29.8] JM: When you go to a company like Github, and I don't know if you've talked to Github or not. You would say, "Hey, we've got this Auth0 extend product and we can give you a execution environment where programmers can spin up their own code. Doesn't that sound more appealing than WebHooks?"

Is that kind of thing enticing to – Because I'm sure – whether you've talked to Github or maybe we could talk about some other enterprises. I mean, what's their response? Are they like, "Yes, let's absolutely do this"? Or, "It's a little too early for us," or, "Our WebHooks are working just fine." How are they responding to that pitch?

[0:53:07.8] BJ: It really depends on the customer that we're speaking with. We have had people approach us, because they've seen our marketing out in the marketplace and they have this problem. They're feeling that pain. We also approached companies who offer WebHooks and made the pitch to them, and they generally seem receptive and we typically do some type of prototyping with them, proof of concept sort of thing to prove out the model. So far, we've had great success with it.

[0:53:41.8] JM: Okay. Let's talk a little bit about the future. You're at one of the companies that is betting the most on serverless. Give me a picture for what infrastructure is going to look like in five or 10 years, the most cutting-edge infrastructure from your standpoint of where serverless is today and where things are going.

[0:54:06.0] BJ: Serverless is really allowing you to focus in a very granular way of just the logic that you want to execute. I could really see organizations being able to deploy entire applications out and serverless infrastructure without having to consider things like docker images, or VMs, or those sorts of things and get very far.

You mentioned earlier that I attended Serverlessconf in NYC this last week. One of the compelling stories that came out of that was A Cloud Guru, which is a online training sort of website. They were able to use serverless technology including Webtasks to stand up their first iteration of their application offering over a weekend and gained considerable amount of attraction. So that speed the market for them was definitely a boon that allowed them to be successful early on and iterate on their product quickly.

[0:55:11.4] JM: There is no logging in to an EC2 server, there is no logging in to docker containers. They built their entire infrastructure on services basically.

[0:55:21.4] BJ: Yeah. Basically serving up HTML via s3 and then making any kind of API calls back out to simple serverless endpoints.

[0:55:33.0] JM: Who still needs to maintain servers? Like in five or 10 years, what are the businesses that still need to maintain AWS instances?

[0:55:43.3] BJ: Hopefully AWS and everybody has moved to serverless. It's where they can focus on their core business and not be as concerned with infrastructure.

[0:55:54.2] JM: Is there a resistance from anybody who are nervous about deploying their stuff to serverless?

[0:56:02.7] BJ: Well, there is always resistance to new ways of thinking and new ways of doing things, right? My background over the past few years, I've done a lot of contracting for state agencies, and there are resistance within those state agencies to even considering using containers in the cloud. They want to maintain their own servers and that sort of thing. You're always going to encounter that, but the longer a technology stays around and gets proven out that that resistance deteriorates.

[0:56:35.9] JM: Yeah, because it seems like switching costs are not super high anymore. Even if I start building infrastructure on any one of these services, most of them are just like very small isolated pieces of infrastructure. I think people maybe got an allergy to this when certain database providers or certain operating system providers really leveraged their relationships with companies that build on top of them and set up an antagonistic relationship.

Nowadays, your infrastructure spread across a bagillion different providers or in the serverless world across so many different providers that nobody has – nobody is irreplaceable, it kind of feels like.

[0:57:29.2] BJ: Yeah, absolutely. In the serverless space, you can look at projects like the serverless CLIL that are attempting to abstract a way the serverless provider aspect of your code base, right? I can choose to write my functions and use the serverless CLI to deploy that function to Lambda, to Webtask, to Azure functions, GCP, and have a single code base deployed out to all of those providers.

[0:58:00.0] JM: What was the coolest talk you saw at Serverlessconf?

[0:58:03.2] BJ: Well, I really enjoyed Glen's talk on serverless extensibility.

[0:58:07.4] JM: Okay. All right. Let's remove a little bit of bias from this.

[0:58:11.6] BJ: Okay. Yeah. Sam Kroonenburg's talk on his effort of launching a cloud guru using purely serverless – not serverless extensibility, but serverless technologies before that term was even coined was really eye-opening.

I was amazed at just how simple his initial application structure was. Was really just based on S3, Auth0 identity, Webtasks and Firebase. He was able to get something out there without considering servers, or VMs, or containers at all. It was pretty amazing.

[0:58:47.4] JM: I love Firebase. I mean, I love those other things you mentioned too, but I Firebase. I've had great experience with it. Okay, well maybe I'll check out that talk.

Bobby, it's been great talking to you.

[0:58:58.5] BJ: All right, Jeff. I really appreciate it. Thanks for having me on.

[0:59:01.6] JM: Yes, absolutely. I think we covered a lot of great ground.

[END OF INTERVIEW]

[0:59:06.9] JM: Thanks to Symphono for sponsoring Software Engineering Daily. Symphono is a custom engineering shop where senior engineers tackle big tech challenges while learning from each other. Check it out at symphono.com/sedaily. That's S-Y-M-P-H-O-N-O.com/sedaily.

Thanks again to Symphono for being a sponsor of Software Engineering Daily for almost a year now. Your continued support allows us to deliver content to the listeners on a regular basis.

[END]