EPISODE 1235

[00:00:00] JM: The procedure that many companies follow to reach production level code is to first design the program then code and test it in different environments and then put it in a pipeline to deploy to production. Developers can make it pretty far into building a core application before inevitably having to include enterprise features and security standards such as single sign-on, Okta security and authorization for different user groups, workday integrations and other things. These requirements can put stress on a team with newer developers or limited resources. For some large projects, these standards need months to fully build out and test. The company WorkOS is dramatically shortening the time that it takes to make applications enterprise ready so that developers and IT teams can focus on the application itself. WorkOS provides developer minded tools like RESTful endpoints, JSON responses, framework native SDKs and a developer dashboard among other tools to integrate otherwise complicated enterprise standards in just a few lines of code. Their mission is to simplify building applications for enterprise users so that developers can focus on creating core features in a timely manner. In this episode we discuss the process of building enterprise applications, the challenges of modern security and administrative requirements and how WorkOS is solving those issues.

[INTERVIEW]

[00:01:19] JM: Michael, welcome to the show.

[00:01:21] MG: Thanks so much for having me.

[00:01:23] JM: You are working on WorkOS, which is involved in making apps enterprise ready. So we should discuss what is the difference between an application that is ready for a startup, let's say, versus an enterprise?

[00:01:43] MG: So there's kind of two different phases companies go through if you think about the life cycle of SaaS. First, when you get started, you're focusing on just building the core product to get what people call product market fit to get those initial users. And when you're in that phase you're really just focused on creating new experiences for customers that attract them in and creating a delightful experience and a great sign-up flow. At some point if you

succeed and your app gets more and more users you'll want to deploy that app essentially get customers at really big organizations. This is the same playbook as something like Dropbox, or Slack, or Airtable, all these companies. And when your app grows in usage and those bigger companies want to adopt it, there's a whole set of features and functionality that they need which is really not your core app experience. It's actually how that app is managed at scale within that company. And it's all these adjacent features around things like authentication, security, compliance, stuff like that. And those are the features that people usually refer to as enterprise ready features, the things that are typically in the enterprise pricing column on the SaaS apps page and things that the IT administration is using to actually manage the app within that organization. The most common one that most people are familiar with is single sign-on. So like signing in through Okta, whether you're using the identity system of the company, but there's a lot of different features like this.

[00:03:01] JM: Now you take the Okta example. Let's say I sell an app like – What's a good example here? Like Slack? Is Slack a good example?

[00:03:10] MG: Yeah, Slack's a great example.

[00:03:12] JM: So let's say I'm slack and my product is getting popular, I'm selling to enterprises and the enterprises now want me to be able to provide them a method of signing on through Okta. Isn't this kind of integration something that Okta just delivers to Slack? Like Slack can just go to the Okta webpage and figure out how to build their integration?

[00:03:36] MG: You can totally do that, and that's how people do it today. So if you have one customer saying, "Hey, we have all of our users signing with email and password, but we really need to use our enterprise identity system, Okta." The developers at Slack in that scenario would have to go to the Okta documentation and sign up for a developer key and get it connected and configured and all of that. The problem here is it's not just Okat that's out there. There're many many different vendors that actually provide these enterprise services that large organizations use. Okta is just one of them. In the identity space there's also Microsoft ADFS, Azure AD. Google actually has a SAML, Google SAML implementation, Ping Identity, OneLogin, generic open source like open source Shibboleth servers which expose SAML endpoints. And so as a developer you might be able to knock off one of them by just going and building with

Okta, but there's this tremendous ongoing effort you have to do to actually keep adding that integration work and keep making your app work with more and more customers.

And so it starts off as being like feeling like a one-off one or two things. Actually snowballs end up into being a really substantial engineering and product lift. And most of these companies, if you look at like Slack's organization today, a huge number of their engineers are actually not focused on building product features for users. They're focused on all these enterprise features because at the end of the day these are what block revenue and block deals for them going up market. So they end up getting prioritized over novel features.

[00:05:01] JM: Okay. Now let's look at it from your point of view. If you want to build a product that makes it easier for Slack to have single sign-on, what do you give them?

[00:05:15] MG: So a lot of people compare WorkOS to other software middleware essentially out there, things like Plaid or Twilio. So let's take Twilio as an example, right? So if you wanted to use Twilio, or if you wanted to send SMS messages in the previous world, you would have to go to all the independent you know SMS networks, all the different telecos and build direct integrations with all of them. That would take a long time. With Twilio you integrate once. They act as an aggregator. You send it to Twilio and then Twilio has the integration with all those different systems. WorkOS behaves very, very similar to that where a developer integrates with the WorkOS SSO feature for example, and then we've actually done all the detailed work integrating with all the different identity systems that are out there. So you don't even have to know what SAML stands for. You don't even have to read docs around ADFS or OpenID Connect or any of these things. You plug in with modern APIs and WorkOS handles it after that for you.

And because we're supporting so many different applications and so many different users, nothing for us is a one-off. We've actually obsessed around this problem and fixed all these small edge cases and details. So the experience that you get by using WorkOS is essentially as if you had an entire team working on it for several years, which is what we've had.

[00:06:27] JM: Give me a little bit of the nitty gritty detail of what's difficult to build in solving this kind of problem.

[00:06:37] MG: So there's a lot of stuff in the actual SAML spec that is if you go read the RFC not super well-specified. Sometimes RDCs use the words like should or may instead of being really tightly specified, and what this results in is actually a lot of implementations of SAML that are kind of broken around the edges are brittle. This is a big problem because it's authentication, and if there's an issue there, it's totally a prime target for people trying to hack into systems and breach security. SAML is also an XML-based protocol, and I think XML canonicalization is sort of one of those like impossibly difficult problems out there that most developers will just kind of brush past.

The major problem here is just that no company can really dedicate their primary deep resources to working in this problem space and so it ends up being something that's done really quickly or done by like an intern over a weekend, but it's then your enterprise authentication. So what we've been able to do is actually build a really rigorous test suite, work with all the different vendors, and in many cases we have to do custom work with them. There's instances where we've connected to SAML integrations with huge companies that that you would – Like household names with their SAML implementation and found issues with it where they either don't match the spec or they're doing something a little bit differently or in some cases they've only implemented like half of it. And so we have to kind of patch the holes and patch the cracks in order to provide a really consistent experience to developers on the other side. And that's not necessarily glamorous work. It's not necessarily the things that certainly at most companies you get promoted or praised for, but it's what people pay us to do. It's the reason why WorkOS is dependable piece of infrastructure is because we've handled all those edge cases and really tried to make something super, super smooth for developers.

[00:08:25] JM: Can you give me a little bit about the architecture of WorkOS or like the programming languages and just what the software architecture is?

[00:08:36] MG: Yeah, totally. My philosophy on a lot of this stuff is to try to keep your infrastructure and keep your actual software architecture as simple as possible so it's extremely easy to reason about so that when things break you kind of know where it's breaking. So we don't really use any super fancy technology or super fancy distributed systems even though I love that stuff. Like our code exists in a monorepo. We just run a handful of services. Where we

like to innovate and really like push the envelope is actually in the application layer where we're integrating with systems.

One thing that we did do early on is we've written everything in Typescript. So everything across the company from the frontend to the backend to all of our services are completely in Typescript, and the type safety and actually having that kind of strong typing declaration has had a huge lift for us internally in terms of our own development velocity just as a team engineering ergonomics. It's also meant that when new engineers join and new team members join, they only really need to be familiar with one language to be able to dive into the code, and so it democratizes the ability for people to work really fully across the stack. You don't have to change your editor. Or some companies if you're writing Rails in the backend and JavaScript on the frontend, some people are kind of in one world or the other. With WorkOS it's just this one consistent environment across end-to-end internally.

But I would say we shy away from a lot of stuff. I think I've tweeted about this. WorkOS runs on Heroku. I love Heroku. I think most people that are spending time spinning up Kubernetes early on, if you're before product market fit, you should really not focus on those areas and you should put all of your energy as much as possible into just the thin layer around the unique product features which are making your customers actually happy. Nobody buys a service because it's running on Kubernetes. They buy it because it solves a problem for them.

[00:10:21] JM: I love the use of Heroku. That's kind of I would say unfashionable to say these days, but I love Heroku.

[00:10:29] MG: We also use Vercel as well. So we have some new shiny hotness in it too I'll say. But Heroku is great. It's fantastic. It's like very stable. It's fast. It's everything you want.

[00:10:40] JM: Why do you think Heroku has become passe? Why do you think people don't like to talk about Heroku or don't use Heroku as much these days?

[00:10:48] MG: It's interesting. So Heroku today, I don't know if the sales numbers are public around their revenue, but it's just become an extraordinary business after the acquisition within Salesforce. I mean, just like unbelievable how much it's grown, and yet I think still retains a lot of

that developer love and sentiment with the developer community around good design and good services.

The main thing that I've seen Heroku do as a strategy is they really haven't hopped on yet, they might, but they haven't jumped on the new movement towards like serverless and people building around this kind of Lambda methodology for web applications. So what Vercel has done and what Netlify has done and host is doing with like cloud – If you know Cloudflare workers, if you've seen that, it's kind of similar running like edge node Lambda functions. Heroku hasn't done that. The Heroku model today is still kind of in this like push your Rails app up to the cloud and have it run.

I think the main reason is just as they commercialized it, most apps in the world are using that previous model and there're fewer apps that are using the Jamstack and using like Lambda style programming models. But as that world moves towards that, I would expect Heroku to do something there.

The thing that Heroku also has added, looping back to enterprise readiness, is a lot of these features. So Heroku did add things like single sign-on, audit logs, they're SOC 2 compliant and ISO compliant and HIPAA compliant, and this means that if you are a developer actually at a big company like Bank of America or something, you can actually use Heroku today. It's democratizing access to the stuff. So even though it might seem maybe passe in the tech Twitter world or like less fashionable I guess amongst new developers, today Heroku is still pretty groundbreaking to people in constrained enterprise environments that are just looking for really good developer ergonomics and a good platform. To them Heroku has only been available probably for a couple years because those enterprise features just got added recently.

[00:12:41] JM: And what do you use Vercel for?

[00:12:44] MG: We run our developer dashboard through Vercel and some marketing stuff and actually a bunch of experimental stuff. We like to build on it. It's just so fast to deploy and have used it. I think I was one of the really earliest users when it used to be super, super buggy. But I've played around with like a lot of the other stuff. And we also use Cloudflare workers and Cloudflare for quite a bit of things too.

[00:13:04] JM: What do you think about Vercel versus Netlify?

[00:13:08] MG: Both are great. So for disclosure, two of the Netlify, or I guess the two Netlify cofounders are actually angel investors in WorkOS. So they're obviously fans of what we're working on. I've used both and played around with both. Really, like I think I just like discovered Vercel years ago before Netlify, which is why we're using it, but I've been really impressed with the speed at which both teams have been shipping. I feel like they're sort of like neck and neck competing with each other, and that's leading to this industry just like moving forward so much faster, right? It's sort of like two car companies trying to build the fastest car versus just one alone. It's funny seeing like a new innovative feature pop up on one and then you kind of know like a few months later it might pop on the other.

[00:13:54] JM: Is there some differentiating force between them or you think they're basically asymptoting towards the same thing? Just better Jamstack development?

[00:14:03] MG: It's hard to say from like a development perspective when you're looking at it, and this is sort of a tabs versus spaces type of argument. Like you can get the same thing done with both, it's more just your personal philosophy, or Vim versus Emacs or whatever, if that's even relevant anymore with VS Code. But I think there's a slightly different philosophy if you look at the two companies focus and kind of where they pour their time. So Vercel spends an incredible amount of time on the next.js platform and evangelizing that open source and building that and trying to just make Vercel the easiest place to deploy those apps, but it's a separate project.

I think Netlify has spent a little bit more time thinking about commercialization and revenue and building up some additional add-ons, and I believe Netlify also has like an add-on marketplace to really easily drop in integrations. Vercel might have this today actually. I'm not totally sure, but I know at least in talking with the Netlify founders who angel invested in WorkOS, they totally understand this whole problem of moving up market and enterprise readiness and they were like, "Hey, we would love to potentially use WorkOS down the line." I think they're playing around with it for some stuff. I don't know if we're live with them yet, but potentially use WorkOS, and then also just help get as many apps as possible enterprise ready. We're really, really

excited as we grow to partner with them or maybe the Heroku's and AWS's of the world such that any app built on any platform can become enterprise ready literally with just the check of a box. That would be the ideal state to get into.

I think the Vercel folks are a little bit more on the open source side and maybe the Netlify folks are more on the commercial side, but both are fine strategies. They're incredible companies built both directions. So it's hard to say which one's ahead.

[00:15:41] JM: All right. So you also mentioned Cloudflare workers. What are you using those for?

[00:15:46] MG: We use it for some kind of like frontline routing for our public website. So one thing that's kind of interesting is Webflow. If you know Webflow, which is a web authoring tool. It's sort of like a really nice WYSIWYG design tool for web apps. Webflow is actually a WorkOS customer. So all enterprise users signing in to Webflow are actually signing in through WorkOS. They were an early customer of ours. We actually decided to adopt Webflow for our own website to sort of like eat your own dog food in that way. And so we run our www site. If you go to workos.com, what you're looking at there is actually being hosted by Wwebflow.

Now if you go to workos.com/docs, which is our documentation, that is a completely custom built next.js application. So when we were building our documentation, we just obsess over this. I think we've rewritten our docs like three or four times. We wanted to create this extraordinary experience for developers with search and injecting API keys and like this incredible layout and make it mobile responsive and stuff and we just couldn't find any service that actually could host documentation in this way up to our level of design polish and the experience that we wanted. And so we built a complete custom next.js React app for our documentation and we're able to run this on the same domain as our Webflow site by using Cloudflare workers to actually split the traffic. And so we have a couple routes that are running the backend that actually will go request the content from those pages. Like I think that docs app is deployed on Vercel, but you don't see the Vercel URL. It's actually proxying through Cloudflare workers. And because that's done at the edge through their edge node system, it's extremely, extremely fast for customers. And we do that with like a handful of other things around the site. So even though it looks like it's running with one domain, it's actually a bunch of different services being combined together.

People used to do this.Like I remember I interned at Dropbox a long time ago, like 10 years ago, and I think we were using like HAProxy for that and putting in rules kind of right there at the edge in terms of load balancers even before it hit nginx. And what's interesting is you can do that in today's world with something like Cloudflare at this kind of higher level API and it's just – I mean, the Cloudfare worker stuff for folks that haven't checked it out, it's really extraordinary. This like globally distributed, fault tolerant, like key value store and be able to write Lambdas that work with that. I mean, it's just so much fun. I think it's sort of like the future of maybe where things will go in 10 to 15 years, but for a lot of our workflow, it doesn't actually match to the application programming model. So we only use it for kind of limited use case today, largely routing like I described.

[00:18:15] JM: Can you say more about that? Like why do you see it as a futuristic programming model?

[00:18:21] MG: Yeah. So if you think about the way that applications have been built, like web applications have been built, first of all web apps are not that old. Like you can squint and say like maybe late 90s like was the beginning, like something like Amazon, which I think was written in like C or C++, cgi-bin for those of you that remember. Seeing that URL, like Perl scripting. But kind of like the modern web application like PHP or like a Rails app that's connecting to a database and there's routes and there's like data transformations happening and all of that. That's relatively new. And I think what happened in that first era is people built these sort of monolithic applications where a whole app runs in a single process or a single service and you horizontally scale that out by adding just more workers, more app workers and connect to the same database, kind of very common web architecture.

Then what people started doing was breaking these into different microservices, right? So saying, “Hey –” And usually you do this when something gets slow. So you say, “Hey, the feed site –” Say you're building a social network and you're like the feed generation, the news feed needs to be its own service because it's getting tons of traffic and we need to optimize it. So you break it out into different service and then you can scale that horizontally however you want and you kind of just keep decomposing your application into many, many different services. At some

point you get something where it's like hundreds of services and you can't even really run your application on one machine anymore because it's so distributed.

There's a middle step I think, which is kind of interesting and it's emerging now, is where if you think about your web app, you have a certain set of URLs, like routes, that do certain things, and that move to Lambda and this idea of like decomposing it into those services. It's almost to say like what if every single URL on your site was its own service? Like what if every single route actually was a micro service, like micro-micro service, like pico service, right? Like just going down the actual prefixes? And you said every single URL, so you have like thousands of these things. If you did that, you can just scale up a single endpoint. You can say, "Hey, this one endpoint needs a ton of traffic on it, and so that thing is actually going to be – We're going to have 10,000 Lambda processes actually doing it. And this other thing that's a URL that's like maybe only ever requested once a month is like dormant, right? It's essentially like put on ice and maybe not even active, not even have resources on it. And so it changes this model where instead of having to decompose your service over time into all these different services being managed by different teams and all that, from the get-go when you're developing your application it's actually already decomposed and you can start off by running it on a single service or run it on your laptop, but as it scales and as the load changes around it, it can dynamically adapt because you've partitioned the sort of quanta around work into all these different URLs. I think that to me is like the biggest shift around this Lambda programming model is to think about no longer your application as like a monolith, or even if the code is in one code base, it feels like a unified application that way. But the way it's being deployed actually is sort of infinitely shardable and you've sort of already decomposed the application down to these small components that are the smallest you could go, these like atomic primitives.

And then if you marry that with infrastructure that can scale, whether it's something like Netlify or Vercel or Cloudflare's edge node workers, you can also deploy it at the edge. So it's really, really close to your customer. And you can imagine a scenario in which only some of those Lambdas are deployed to the edge, maybe the ones that are kind of the most used. And then for endpoints or Lambdas that are less frequently used, maybe that has to call back home or go back to your main applications server running in some region. And I think this is going to be the future programming model, the future model for how people think about building web applications for like global scale, is that you say, "Hey, let's use the Lambda model and how

we're designing the app to be able to have really good developer ergonomics around this. So we're still developing in like one code base and it feels like a Rails app, that old model. But when we actually deploy this, we can get the same benefits of running these like edge node services where like Facebook does this for CDN caching. They have this like incredible like photos cache that runs at the edge and keeps stuff in memory and whatever. But mere mortals like you and me like can't build that. But if you build your app using this like worker model, it just naturally adapts to that workload and can scale up and down.

So I think that shift is huge. It's not totally realized yet. Most people don't build apps in that mindset or in that that way, but I think over time we'll see it start chipping away especially when people want like high performance. And the cool thing is it's actually not harder to do. It's actually easier in some way, like that new model of building is kind of easier and more fun. And to me that's what people are attracted to Vercel or Netflix with today is like, “Hey, this is so easy to build on top of. It's so easy to ship your app and you just ship constantly,’ but I think there's this underlying structural change that's happening around how web apps are built and how they can be decomposed, and that to me is like super, super exciting. I think that's – I would say like if you look forward to like 2030, I think then in 2030 we'll all be building apps like this in this new model. Sorry, it's a bit of a rant. Yeah. Yeah. We're not doing a lot of stuff like this at WorkOS, but this is what I think about in the shower.

[00:23:45] JM: Yeah. Just take you further down that tangent, I think you mentioned or maybe you're aware of the workers KV thing, the key value store system that can use with the Cloudflare workers? Do you use that for anything?

[00:23:59] MG: I think we are using it for some stuff. We've used it for some experimental stuff. I’m not sure if we're using it for production use today. WorkOS is a little bit of an interesting duck in that we have very specific requirements both from a compliance perspective, but also our internal requirements or how we deal with data, because we end up dealing with like a lot of pretty sensitive data regarding customers. And so because we do that, I’m not saying KV is not secure or something like that, but we just try to keep our data in a very kind of like singular consistent area so we can reason about it in a very strong way. And so I think in that regard we're probably not using KV for anything related to the production API service today, but it's

really cool. I know it's really early. The team that they have working on that is really strong. I think it's just like an incredibly cool idea.

And going back to like my roots, like I studied CM at MIT and like my favorite classes were these things where you're building these like globally distributed, like fault tolerant eventually consistent systems, like how you design that, and it seems like they're doing pretty incredible work already with that.

[00:25:01] JM: So coming back to WorkOS and I guess the core of the business. So we only really talked about one facet of what you're building, which is the single sign-on functionality stuff, but there's a host of other features that an enterprise wants out of a given SaaS application. Tell me about some of those, or give me another example of something that's very difficult for a software company like if a Slack – Again, the problem being if Slack wanted to sell to an enterprise, give me something else that Slack would have to implement.

[00:25:39] MG: Yeah. So there's a whole long list of these things under the umbrella around enterprise ready features, and actually about a year and a half ago I gave a talk about all this stuff. It's called crossing the enterprise chasm. It's on my YouTube. And really like SSO is just the starting point. It's the front door product. It oftentimes will let you land your first set of enterprise customers and it really is a deal breaker. So you got to have it in your app.

A fast follow from that something people start asking for is directory integration, and this often comes up in the term SCIM provisioning, SCIM.Iit's another acronym. SCIM is kind of like SAML but for directories. It's a little bit newer. Unfortunately it's not as standardized. It's still pretty fragmented and not everyone supports SCIM yet. And so what we've built at WorkOS is an API that integrates with all these different directory systems that are out there. So Google directory, SCIM endpoints, Okta directory, all these different – Active Directory, AzureAD, but we have also integrated with the HR systems because those often end up being the directory for customers. So today we have integrations with Gusto and Rippling, BambooHR, Workday, PeopleSoft, there's like in Hibob. There's like a bunch of them. And what WorkOS lets you do here is easily connect your application and then receive web hooks when users change in the directory. So a new employee is hired, someone leaves the company, or their group membership changes. So they change teams or their membership inside of a group and the organization changes.

What this lets you do as a developer if you're building a SaaS app is automatically provision users in the app. So automatically onboard people when they're hired. You're able to keep in sync with changes from the directory system. So say you're using their groups in order to do some kind of access control on top of it, or there's like an admin group or a payroll group. You can pull that in and do some work with that. And then very specifically you can deactivate them when they leave the company. And so this is really what IT admins care about. They want to say, “Hey, as soon as I deactivate a user in Workday, we're letting somebody go. It's Friday at five o'clock. I want right at five o'clock, 5:01, all of their access to be revoked across their entire system.” And in order to do that you have to tie into the directory to get that notification. And so WorkOS would send your app a webhook and say, “Hey, this user's been suspended.” And then inside of your app you can revoke access tokens, revoke that session cookie, transfer data if you need to, do whatever action you want. So WorkOS helps unify the fragmented directory systems in the same way we're unifying identity systems, but it's this other use case around provisioning and deep provisioning. And we keep adding providers to that actually, like every month we ship more and more of them.

[00:28:26] JM: Let's talk to the engineering of one of these features. So is there another one that was particularly difficult to engineer that we could discuss?

[00:28:36] MG: Yeah, let me talk about the admin portal. This is kind of interesting and not something that's actually in that talk I gave on that's on YouTube. So as we were building WorkOS, the first conceptualization of WorkOS was just an integration platform. Just under the hood, backend, taking a single API for that developers integrate with and giving them access to all these different systems, right? Kind of like a fan out integration. What we found after we launched was that it only solved half the problem. So for something like single sign-on, we were able to really quickly connect a developer's app to Okta and OneLogin and Ping Identity and all these things, but there's this big overhead of actually setting that up and the whole UX and IT admin needs in order to set up a SAML connection.

So the steady state is fine, but that first step, the kind of cold start problem around setting up SSO, if you're an IT admin, you need to configure something in Okta. If that's your identity provider, you need to copy and paste these ACSUri’s. You need to upload an x509 certificate or

email it to like a sales engineer to get it set up. There's this whole configuration step. And most companies don't do this well. Like if you've ever tried to set up SAML with a common SaaS app they usually say, “Contact us,” because it's this whole manual process and maybe they have some internal tool.

And the problem with that contact us is it's not really solving the problem, right? If your just underlying connection is fine, it still requires the team to do something manual. Like imagine you're trying to sell a product online and you couldn't accept credit cards unless they called you over the phone. It's kind of like that. And so we built this thing called the admin portal, and what this is is it's the entire UX for setting up WorkOS for the IT admin. So as a developer all you have to do is create a really easy to create session link for an organization. So you say, “Here's the organization ID, generate a session,” and then you redirect the IT admin to that or send them to it in some way and then they're presented with a complete web app for connecting their identity system, connecting their directory system, debugging stuff, configuring it, setting it up, kind of what you would build if you were in year 10. This is the stuff that like Dropbox eventually built or Asana eventually built at scale, but we wanted even the smallest developers to have that great setup experience and great workflow for their first enterprise customer. So it's not done manually.

And so what we found is we actually had to break out of the model of just providing the API and start providing these UI components essentially. This runs as its own web app. It's something where it's free. It just comes with WorkOS. The base version runs on a workos.com domain and we just have a little Powered by WorkOS in the corner, and this is something that we're actually able to customize and white label for people in the same way you can get like Kirkland signature, like ketchup or something. You can get Kirkland signature, admin portal set up.

So the way we do this is you can create a CNAME domain, you point this to our DNS server, we registered that in our system so it's picked up at the application routing. And then when users come in, we look at actually host parameter and plus the actual session that's being generated, we're able to dynamically change the UI for a customer. And the things we allow people to compose today are pretty limited. We allow them to, like I said, change the URL. They can put their logo in the application, that little app. So instead of saying Powered by WorkOS, it says like

FooCorp or whatever app is yours, your logo. We also allow people to change the primary button color to match their brand.

And what's pretty cool as well is we actually do the same CNAMing aliasing for our API. And so those URLs that might be copied into Okta that normally say auth.workos.com/blah-blah-blah, like a bunch of identifiers, that stuff can say sso.yourapp.com and can be fully aliased, and that's essentially just an alias for our API. And so it's this interesting thing. We're sort of thinly applying this glaze over this product that we built so it feels totally native to the IT admin that's actually logging in and setting stuff up.

We don't let people customize it beyond that today, like you can't change like the background color or the layout or things, because it's pretty dialed in from a UX perspective. The experience, you can actually try it right now if you go to demo.workos.com, you can click around and you can set up a test SSO connection and see what this is like. But we might allow people to customize it or change it more in the future, and it's interesting because in this way we're giving people sort of a mostly fully built application where it can feel at home inside of their own app experience and yet handle this really, really, really complex UX problem.

I mean, I can just tell you like setting up Microsoft ADFS, which is like runs on a windows pc probably in the basement somewhere, that's your identity service, and like getting the screenshots for configuring setting that up. We have an amazing workflow for that because we totally polished it and it's presented to the IT admin in a way that they totally, totally understand. That product, sometimes I kind of describe it kind of like TurboTax, like walks you through step by step by step, but it's for setting up connections to WorkOS. And I think that programming model of like offering this interface and this workflow that is sort of running separate from your core app but is still tightly integrated is more and more common. So we modeled a lot of this after what Plaid has done with the Plaid Link product, which is how you connect your bank. In their case it's an iframe that pops up and you put your bank credentials into it.

We actually initially did this with an iframe and experimented with that, but found from like a security perspective it actually wasn't as strong as we needed because you're uploading certificates and configuring authentication. And iframes, while being amazing and great, they're

still weird edge cases depending on browsers and stuff. And so we actually run this as a separate service just completely behind us.

So the other thing we looked and modeled this after is the Stripe billing portal. I think it's called the customer portal, which is a very, very similar idea, where from your app you can create a session, send the user to this billing experience. And then inside of that it's not just a pop-up, it's actually the whole interface. The whole experience is being provided by Stripe. So same thing exists, but instead of putting in your payment details for a credit card, you're putting in your credentials to set up SSO and connect Okta or Azure or Google or whatever service that you're using. So I think we'll see actually more and more of that. Especially with developer platforms and developer tools, it turns out just providing the underlying API is often not quite enough to fully nail the whole end-to-end experience soup to nuts for a developer, and you need to be able to expose some UI, and this is a pretty powerful way to do it. And if you're able to do a little bit of customization and do that CNAME so it matches their brand, you're able to kind of really, really quickly get to the end state and solve the problem for a developer totally end-to-end for them.

[00:35:19] JM: We've talked through some of the tools that you use, Typescript, Vercel, Heroku. Are there any other tools, SaaS products, that have surprised you and how useful they've been?

[00:35:34] MG: We tend to explore new tools quite a bit, but like I mentioned keep the stack pretty simple. One thing that's kind of interesting that maybe is worth saying, which is actually a tool that we built in-house, because we didn't find anything, is for how we do support. So today we have a really large number of shared Slack channels with other organizations. I think we're like almost at 200. So it's a lot with other companies, and we offer these pretty liberally to people that are integrating because developers don't really read their email. As a developer I don't like reading my email, and our team doesn't like reading their email. So email isn't great for support. So we create this shared Slack channel where we're essentially embedded in their organization helping them do their integration or answer questions that they have.

This is a huge pain to manage, because if you imagine hundreds of channels, how do you split that up? How do you deal with it? And how do you make sure you're not dropping the ball on conversations? And so we've actually built a bunch of stuff internally that looks at conversations in these shared channels and does aggregation. It'll pipe them all into one channel so you can

really easily go in and just scan the latest messages from customers. We have some bots that will help us remember to follow up if there's a dangling message we haven't followed up with. We're able to track SLAs around this kind of stuff. And so this is actually an area where typically companies will just adopt something like Zendesk and use that for support, but we wanted to be able to go a few steps further and create this like totally magic experience for developers.

I mean, imagine getting support with a product that you're trying to use and it feels like just slacking the engineer sitting next to your co-worker. That's the kind of interaction we have with customers. So in that sense like there hasn't been a product we've been able to find to do it. We've built a lot of stuff. I think we're sort of an extreme use case of Slack shared channels in that regard. But it's one of the reasons why I think developers love the product and have had such a good experience using it.

I'm trying to think other stuff that we've used. We use Semaphore for running our tests, which is pretty sweet. I actually was just on a podcast with the founder of Semaphore, which I think is coming out soon. So we can chat about that. But we use a lot of different services and managed infrastructure for the platform. We don't use Kubernetes. Not yet anyway. I think I would be very proud if we could get to millions in revenue before we use something like Kubernetes. I think it's a sign that we're spending all of our resources solving the hard problems that our customers care about today.

[00:38:04] JM: Yeah. I mean, if you're using Heroku, Heroku basically simulates what you would get out of Kubernetes, right?

[00:38:12] MG: Yeah, pretty much. Yeah. Yeah. I think it also depends on the workflow. So my previous company, Nylas, which I founded and ran for like five years started off as like a Python app as we were like building it on my dining table and eventually was syncing like hundreds of terabytes of data. And in a scenario like that where you have this really extraordinarily diverse workload and like pretty heterogeneous type of application workload that it needs to process, being able to have something like Kubernetes is actually a huge asset. I don't think it was around actually back then, like we're using Docker and stuff for some stuff.

But the WorkOS workload is actually very different. For us, correctness is a lot more important. We have less API volume than what most people would expect. Kind of similar to Stripe, right? If you think about it, like Stripe probably in absolute terms early days had a fewer number of API requests, but those requests can't fail. So we've really focused on like reliability and robustness of the system to failure. And one of the coolest features that I love about Heroku is this ability for when you start up a new dyno is what they call it for a new version of your app. You can actually get it to warm up and start the new application and then route the traffic from the old one to the new one so that requests are never dropped, right? It just keeps the connection open. Waits for all the old requests to finish and then flips it over. And this is a way that we can do like deploys constantly and not drop any requests ever for any users. Not even have a blip of down time, because that blip of downtime could result in somebody not logging in. That's not acceptable to us. And it's cool because in Heroku that's just a checkbox you turn on, right? You don't have to do anything else. You don't have to like configure this instead of HAProxy or whatever or enable it elsewhere. It's literally just a check box. It's built into the deploy system. And so there're a lot of small things like that that we do get and we're able to leverage and just really not think about. It's like every hour, every second that we're not thinking about that part of the system is an hour that we're thinking about the next feature we're going to build for a user or like the next way that we can help a developer get to production faster for their own app.

[00:40:24] JM: Give me a vision for what the next two years looks like, two to five years looks like for WorkOS.

[00:40:33] MG: So I think on the two-year time front, we pretty clearly have our work cut out for us. We really clearly know the features that are needed by enterprise it admins, and you can kind of guess what they are if you go to like Slack's page or Box's pricing page and go look at the enterprise column. Like we're wanting to help developers add all of those types of features to their apps. There's a lot of stuff. On the longer, longer term scale I think we have much bigger ambitions than most people expect. And I'll give kind of a parallel to this. So there used to be a platform that was very popular that gave developers an incredible developer experience. So you had APIs that were super well-designed, an incredible developer experience for writing code, good conferences and stuff. Probably the most developer-centric company of all time I would say.

And then on the other side, that product that the company built also gave IT admins all of the bells and whistles and all of the functionality that they would need to actually take an app and deploy it and administrate it and update it and do all the stuff and it was compliant and it was kind of checked all the boxes for IT and it was beloved by them. And this platform enabled developers just to focus only on building their app and getting their app built, and that was Microsoft Windows.

So in the 90s, late 90s, early 2000s, if you wanted to build an app, you would just like literally go to the Microsoft conference, learn how to write some code on their system, build your app, put it on a CD, before thumb drives, put on a CD, give it to the IT admin and you were done. You collect your paycheck, your check and you go home. And that was this incredible moment for software because there was a consistent target that you could hit. You could develop against it and you could know with certainty that your buyer was going to be able to use your app. That doesn't exist today.

So in today's world when you want to build an app for the workplace, you'll build it as a web app, but you have to tie it together with – You're running in GCP or AWS, the IT admin, the organization, your customer is using Okta and AzureAD and things like JumpCloud, or MDM solutions like Fleetsmith and like all. There's this huge fragmentation problem around IT. And if you're a developer, it's fragmented on your side and it's also fragmented on their side and it's just this huge energy overhead to deal with that. And what results in is fewer apps getting built and less time being spent on the interesting creative parts of building software. And I think that is the structurally – It's really sad. It's the structurally biggest problem in my mind today that's holding back people building new software in new products is this chasm that exists between just getting your app built and getting it in the hands of users. And what we don't have today is something like Microsoft Windows, an application development environment or platform that's built for app development

If you think about something like React. React is probably the biggest innovation in like UI programming last 10 years. It's the tooling exhaust from a social network. Like Facebook came up with it. Like Facebook isn't even trying to build apps. They're building Facebook. And so we don't really have anyone thinking about how to build new platforms and new systems to allow developers to creatively express the experience that they want.

And when I look at this problem and I try to like peel away the layers of the onion and think about like what is at the core of this thing? It turns out that there's a set of features and functionality that is required by every app. It's not unique to any app. And those are the things that really are the building blocks of modern SaaS and modern web apps. There are many, many different things around that that people are working on. We're trying to focus initially on these enterprise features because they unlock value, like immediate revenue for companies. It's a great place to start building a business, but there's a ton of other stuff. There's a ton of other stuff in that umbrella.

And those things that are under the hood, those components that actually allow you to build your app, what are those? Like what would you call those? Well, they're the hard things under the hood that every app needs but it's not specific to any application that's being built. That's what the operating system handles. That's the role of the operating system. And so when we call the company WorkOS, that's where we think about where we sit in the stack. We want to solve all the hard problems that every developer needs in their apps solved but it's not unique to any developer building any product, and that will enable more people to build software, more people to ship faster, more impact to actually happen in those large organizations. And ideally at the end of the day get the most powerful tools into the hands of the most people, right? That's the end goal for any of this.

And I think like the place that we're starting today with things like enterprise SSO and SAML, it's really early. WorkOS is like barely two years old. We have a small staff, but our ambition is to continue growing and just keep chipping away at this problem and just I think a lot of times about like creating a lever, like leverage, right? How can you make the biggest lever possible so that even the smallest groups of people, even the three people in east L.A. working on some idea or like a developer in Indonesia or something, someone like that can build an app that's as powerful and as structurally ready for the market as even something like Box or Dropbox? Like that's the place we want to get to. And I think we're on to something. Like I said, we're not the only ones doing this, but I think the place that we're starting is kind of a not obvious, but super important first step. And so if anyone is listening, and I just have to plug this, if anyone's listening and wants to work with us, please reach out if this resonates with you because there's a huge, huge ambition behind this company and we need a lot more people to be part of it.

[00:45:57] JM: All right. Well, thanks, Michael. Sounds like a good place to close off.

[00:46:00] MG: Thanks so much for having me. This is a lot of fun.

[END]