EPISODE 1280

[INTRODUCTION]

[00:00:00] JM: Delivering SaaS products involves a lot more than just building the product. SaaS management involves customer relationship management, licensing, renewals, software visibility and the general management of a technology portfolio. The company, Blissfully, helps businesses manage their SaaS products from within a complete IT platform with organization, automation and security built in. The Blissfully platform offers a system of record for creating and maintaining a single source of truth for technology, as well as a workflows and automations feature for defining and executing consistent IT processes and an IT collaboration feature, as well as security and compliance. These features come together to form a comprehensive IT management platform. If that still sounds confusing, then it will be explained soon.

In this episode, we talked with Aaron White, who is a founder and CTO at Blissfully. Aaron was previously a cofounder and board member at Price Intelligently, and worked at Venrock before that. I hope you enjoy today's episode.

A few announcements before we get started. One, if you like Clubhouse, subscribe to the Club for Software Daily on Clubhouse. It's just Software Daily. And we'll be doing some interesting Clubhouse sessions within the next few weeks. And two, if you're looking for a job, we are hiring a variety of roles. We're looking for a social media manager. We're looking for a graphic designer. And we're looking for writers. If you are interested in contributing content to Software Engineering Daily, or even if you're a podcaster, and you're curious about how to get involved, we are looking for people with interesting backgrounds who can contribute to Software Engineering Daily. Again, mostly we're looking for social media help and design help. But if you're a writer or a podcaster, we'd also love to hear from you. You can send me an email with your resume, jeff@softwareengineeringdaily.com. That's jeff@softwareengineeringdaily.com.

[INTERVIEW]

[00:02:03] JM: Aaron, welcome to the show.

[00:02:05] AW: Hey, thanks for having me. I'd like this show to be much in the direction of a case study. So I think what your company is doing is quite interesting. But I'm also intrigued by your perspectives on how to build infrastructure these days. So given that we're going to be thinking of this as sort of a case study, let's start with what you actually do you run blissfully.com. Explain what blissfully does.

[00:02:32] AW: Sure. So Blissfully is a SaaS management platform. And what we do is we help IT teams figure out all the different pieces of technology in use throughout the organization, which apps people are using, what you're spending money on, what you're not using, but spending money on, who has access to what? And we create a unified system of record of all those technical relationships. And from there, that context informs just about everything IT operations might need. So whether you're hiring a new employee and trying to figure out how to rapidly get them into all the tools that their teams might imply, or if somebody's leaving and you need reverse access to any number of apps, or you're doing a security audit, or compliance process, or even just cleaning up your spend, we help IT teams figure all that out.

[00:03:16] JM: Can you give an example of the onboarding process and the day-to-day usage just so people have a better sense of what Blissfully does?

[00:03:25] AW: Sure. Yeah. So in terms of day-to-day usage, I think the first thing is just how would you set up a tool like ours. We integrate to all of your different back office systems. So GSuite, Microsoft 365, Okta, OneLogin, you name it. We plug into it. We pull the data in to create that system of record. And then from there, you can use this to answer any questions you might have. But if we're talking about employee onboarding, for instance, let's say you're bringing a new engineer into an organization, right? So they're going to be part of the all employees team. They'll be part of engineering. Maybe they're part of the SF office, or maybe they’re a remote worker. You would select those things in our tool. We would figure out all the various applications those relationships would imply. And then we would spin up a unified workflow that combines the things that humans need to do to get that person up to speed, things that new hire needs to do themselves, as well as things that automations can get done for you into one spot, and then follow up with everything until it gets done. So whether that’s provisioning people into GitHub, Figma, Slack, Zoom, all these different apps, we can track the progress of all those and make sure that the person is deployed successfully.

[00:04:35] JM: Like let's take the example of onboarding somebody and giving them the permissions into Figma. Does Figma give you all like the API hooks and stuff that you need to understand the user's progress along the way of being a being listed as an enterprise member? Or do you have to like reverse engineer some of that?

[00:04:58] AW: Oh, man. Yeah, this is a fascinating question. In fact, interestingly, to your example at Figma, I was just looking at this this morning. It is really complicated. And I think just a backup before we answer Figma specifically, the reason SaaS management platforms exist is because the number of tools that organizations use has just exploded over the last the last several years, certainly the last decade, and you're in a world where everybody in the org can kind of pick and choose what they want to use and bring that into the team, whether or not IT is even aware of it. So it's just very complicated in terms of service area. And then even as you get into per application concerns, like we're talking about with Figma here, it's extremely complicated.

So, actually, Figma, they support provisioning users via SCIM, which is a standard enterprise protocol for creating users. It typically works behind an identity provider like Okta, or OneLogin. And when you create a user in Okta or OneLogin, you can have that synchronized and create a user in Figma. Interestingly enough, even when you have that setup for Figma, it doesn't necessarily mean that the user gets the right permissions. Someone still has to go in manually afterwards and make sure that that user is set up correctly to the various teams, boards, and have the set roles that they need.

So part of our value proposition is we will automate what we can, but I think people would be shocked by how little is truly automatable. But the real value we have is that since we can track all the various app admins or app owners and kind of coordinate and orchestrate the more human element across all this, because a standard organization probably is using hundreds of apps, and you can only automate so many of them. And so you still have a lot of work to coordinate beyond that. So it's a mess, for sure. And that's one of the reasons that our platform exists, because you just really have to keep kind of constant tabs on what's really happening inside of your org from a technical perspective. You can steer me here a little bit, because I'm certainly happy to talk at depth. But it does get very complicated very quickly. On and off-boarding is just one thing that we help folks with that. But, really, it's making sure that they have one system to understand what all the apps are, what the relationships are, which ones are currently automated, which ones are going to be automated, which ones can't be automated, but who's responsible for them as an individual? It's a lot for an organization to deal with.

[00:07:22] JM: Yeah. And I think we'll dance around this subject matter. I'd love to know a little bit about the MVP for Blissfully when you when you originally went to market, because we're talking here about this platform that helps you with onboarding, and off-boarding, and user management of all these different third-party applications. That's a tall order for an initial product. So can you tell me what the initial product was when you felt comfortable going to market? And then we can talk maybe about the architecture of that first product.

[00:07:59] AW: Sure. I'd be happy to. First product – Actually, when my cofounder and I got started, originally, we were going to be an MSP, a managed service provider. So we would offer our expertise in setting up technical systems to make sure that organizations were running efficiently and getting the most leverage. And what we found was when we were trying to sell those services, either to customers we had just won because they believed our vision, or when we were trying to articulate what we would do for prospects. We ran into the same problem over and over, which is if you're going to tell someone you're going to help them with their technology, you have to know what their technology is. And so we'd ask our customers, potential customers, "What is it you're using? What are your employees using?" And, universally, the answer was, "We're not really sure." Which is kind of shocking to hear, right? Like, "We're not really sure." "What do you mean you're not sure? Of course, you need to know what software makes up your organization. How do you have continuity? How do you know your spending? And how do you manage security?"

So the very first version of the product was designed to answer those questions to tell you what you were using so that we could provide you better service. And so the very initial MVP was a Google marketplace application, which still exists today actually, although it's much more sophisticated than was back then, that you would install, and in a couple of clicks would automatically start ferreting out several pieces of information. Who are the people in your organization? What apps have they ever signed into using Google OAuth or Google SAML? And we would even find invoice and receive emails from known SaaS vendors. And in about 60 seconds, and you would watch this stream in on your screen, you will start seeing all the applications in use in your organization. Who is using them? What you are paying, when you last use them, or if the billing owner had left the company six months ago. We would surface all that for you. And so it ends up being this really impactful high value list of that technology and those relationships.

What was interesting when we launched that MVP is that I got five friendly companies to install it, give me feedback. And then the next week it was another 10 installed it. And then the week after that, it was another 20. And so we just had really rapid growth, because it turned out most companies weren't really sure what their inventory of software was. And at that point, we realized, “You know what? Forget the services that we were going to provide. Really, we're a software company. This needs to be a product that the world has access to.” And so my co-founder and I've doubled down on it. We raised a small pre seed round. We built a team, and we got to work building the real version of that software.

But the MVP took us about two and a half months to build. It was just him and I coding it. He knows a little bit of Ruby and CSS, enough to be dangerous. So mostly it was me doing the backend coding. And the initial stack was using AWS Lambda, which was very new at the time. This is about four or five years ago, as well as GraphQL and Elm. And thankfully, those particular technical choices have withstood the test of time and the platform still based on them today, although our understanding of them has improved quite a bit. But it was a really fun experience doing what I like to refer to with my team is lightning bolting a solution up. The power of an MVP is you get to just strip down everything and focus on delivering value to your user as fast as possible both from a development timeline and from a product experience. And it really allows you to focus in such an incredible way. So that was a very fun time obviously.

[00:11:22] JM: The technology choices there are a little bit surprising, particularly Elm. Going all-in on Lambda makes make sense to me. Going all-in on TypeScript makes sense. But Elm is for people, who don't know, like a scarcely used functional frontend programming language. Why would you choose Elm?

[00:11:48] AW: Yeah, let's call it boutique. So I am – Well, we can get into my technical worldview. I'm predisposed to functional staticallytyped programming languages. I will talk at great length if anybody gives me the opportunity, and maybe you will, on why I have that preference. But boiling it down, what it does is by removing type errors from the runtime, by removing state management, you really just eliminate entire classes of errors from the possibility space. And so that means that the code you're writing is necessarily more correct at the other end, but more meaningful as you're writing it. You're not doing a lot of extraneous stuff to protect yourself against bizarre situations that it's hard to prepare for.

For me, as a one person MVP author, I was trying to choose technologies that gave me the absolute most leverage. And here was something that could eliminate all these various problems upfront. I was going to lean all into that. And I found that it was actually highly efficient for me to produce an MVP application using Elm. Now, the question is, would that be the frontend technology choice for a company as we grew it over time? And that actually was less clear to me, right? So for an MVP, absolutely, it means that I'm not dealing with null reference exceptions, casting errors, undefined and properties not found, state management issues. All that was just gone, which is great. But are there enough developers that understand Elm in the ecosystem? Are there enough libraries that we can leverage or we have to be writing everything from scratch? At the MVP stage, that wasn't really a concern. Obviously, as we try to transition the code to a longer-lived actual stable project, that was a very real question. And I don't know that I went into it thinking that it wouldn't be answered one way or another. I think it has absolutely withstood the test of time for us. And we've been able to build a fantastically amazing team around it. But that was the calculus for the MVP. And it paid off then, and fortunately it's still paid off even after that. If you'd like, I can talk about why kind of the rubric behind some of these choices, right?

[00:14:08] JM: Please do. You mentioned real quick, you are an experienced entrepreneur. Like you built a number of companies. And so people listening who might think of this as a complete novelty choice, there must be some reasoning behind it.

[00:14:20] AW: Absolutely. Let me get into a little bit of my philosophy now. So I've worked on any number of startups. So one of the first five people at Runkeeper, helping move and scale them into the cloud. I made an animation startup way back in the day that got half a million kids using it. My cofounder and I – It's actually our second company together before that. We launched an edtech company, venture-backed. I helped start a company called Price

Intelligently, and actually wrote the first version of that product that's doing quite well and an independent company today. Blissfully, obviously. The couple of guiding philosophies that I have are, one, I mentioned lightning bolt, as fast as humanly possible to value at all costs. I can come back to that that's more of a development process and a mindset than anything else. Two is if you can eliminate entire classes of concerns from the things you do and let the machines do the things that humans would otherwise have to. You're taking a huge massive leap ahead. And, really, all technology is designed to give you that kind of leverage.

So with Elm, imagine never having to reason about whether something is null or not. It's just not possible in Elm. The language is designed in such a way as to never have that be possible in the runtime. It's sort of provably guaranteed by the compiler. That's a huge win, right? You're spending zero time doing something to mitigate pure problems. And you can focus really on what you have to do. And I would make that claim of all typed programming languages, especially the highly statically-typed ones, over dynamic language throughout a type system, right?

The difference between what you can express in a language with a type system versus something you can express in a language without a type system, you can make things that don't make sense in languages without type systems, right? Like they're both equivalent. You could write software and either of them. We know that's provably true. There's nothing you can write in one language you couldn't fundamentally write in another. But the difference is you can write nonsensical things in languages that don't have type systems. And so when you kind of look at it through that perspective, it's like, “Well, why would we do that to ourselves? Why would we allow that to be a possibility?”

And I think the historical answer is, look, programming is kind of been both a little bit of CSS, but a lot of kind of engineering and hobby creation of tools and languages. We've just watched this industry evolve for years and years going through dynamic language after dynamic language. And I don't think it's any surprise that in 2021 TypeScript is on the rise, Python is getting a type system, Ruby has flavors of it that have type systems. Certainly the .NET ecosystem has them. Java has had them forever. It's getting more sophisticated. I think the trend line has been towards letting the computer help you eliminate entire classes of problems that just shouldn't be possible at the language level. That's why I think some of these choices, they're not flippant.

They're actually highly meaningful. And the same on our backend, right? The calculus for why would you use Lambda? Well, to me, that's a really simple, straightforward one, which is think of anything that your company might do operationally that other companies are also doing, in some sense, is wasted effort, right? It's not unique to your product or your value proposition, right? It's a pure tax.

And so when I look at what it takes to manage and maintain EC2 fleets, or Kubernetes clusters, that's not specific to your product, right? That is something that every company that uses those technologies has to engage with. Well, the advantage of doing something with Lambda, and let's set aside for a second kind of the bleeding edgeness ever than not. It eliminates an entire class of operational concerns. I'm not worried about health checks, and failover, and patching of operating systems or container technologies. That's all handled by the layer of AWS now. So from a leverage standpoint, it's a massive win. Really letting our team focus on what our product is.

Now, obviously, there's still DevOps stuff that we do. But assuredly, it's much less than if we're maintaining a cluster. So I think when you're choosing technologies, all those decisions about which tech you adopt and which class of problems that tech just totally obviates is hugely meaningful for leverage. And it's a fun one to tease out each and every one of those. I have similar feelings with GraphQL versus a more typical REST-based JSON API. These choices have added up for us such that our relatively small team has been able to make a very wide and deep product. And that's just been a fantastic feeling.

To your point, this isn't my first rodeo building software companies. I have made mistakes. I have gone down paths where I over-indexed on the technology and it came back to bite me, or I under-indexed on it, and we didn't get the leverage we needed. And it takes a little bit of honing over time to figure out where those lines actually are. But if you find them, it's pretty great.

[00:19:22] JM: I'm bought in to what you're saying. Your stack is TypeScript, Elm, GraphQL, Lambda. I assume other serverless stuff, maybe step functions or things like that. And one thing I'm curious about exploring is, particularly on the backend side, the technical debt and the technical challenges that come with a serverless focus, because from my point of view, like the game is over. If you can go all-in on serverless, you absolutely should. You were introduced to me by Troy Goode over at Courier. I had this conversation with him like a little more than a year ago. Just like his company is entirely serverless. They're an infrastructure company. And basically it’s like if you can be serverless, you should. But I'd love to know, what cost does that come? Like what are the penalties for going all-in on serverless? What are the backend sources of technical debt?

[00:20:18] AW: That's great. So you're saying there're no magic bullets that just give you pure upside?

[00:20:23] JM: Apparently not.

[00:20:24] AW: Apparently not. Well, I'm still looking for them. Yes, there are costs of going serverless. We definitely paid them along the way, and even now continue to pay them. I think, I believe and I think I've seen that it's a lot less, but they are there. So in the earliest days, for us, when we were adopting serverless, the real challenge for us was understanding where to draw the lines. And I don't think this is surprising given any backend technology choice. But for serverless in particular, what is a function? Like how much should a single function do? And should it just do one thing? Or should it be parameterised and be able to do kind of multiple things? Do you divide functions up by functional area of your product by a repo, by some other set of concerns? Just figuring out where to draw the lines and split how code is developed and deployed is a non-obvious question.

So we have definitely explored all the all possible variations of this, from tiny functions that do barely anything, to monolithic functions that do almost everything, right? For instance, we have hundreds of serverless functions. Some are hyper focused. Take some JSON and statically transform it, spit JSON out, right? And then you've got something more like our API, which is largely self-contained in a single Lambda, which you could argue is just overly large. And I'd have trouble defending that. So that's one real concern, is just where do you draw the lines? And that has a lot of implications for the development process, for ops, etc.

The next set of concerns that we ran into are how do you coordinate all your various Lambdas? You need to start becoming a master of figuring out when to fan-out, when to fan-in. Which tools you use to coordinate them? So are you going to do one Lambda invokes the next Lambda? Are you going to buffer those two things with a queue? Are you going to use SNS instead maybe to trigger listeners on some more pub/sub model? Should it be a Kinesis stream? Should you be using step functions to coordinate these Lambdas? And there's no obvious answer for any of those questions either. You kind of have to look at the shape of the data, the rate of the data, the managerial pros and cons to each of those approaches to try to figure out how those things ought to get stitched together.

And I think in the earliest days of pure serverless companies, there was just very little writing out there. So there was not a lot of community knowledge to draw on to figure out how to start dividing those lines. Going forward, I think there are still a lot of these same challenges ahead of us as we start to get more sophisticated in terms of multiple teams working on different things. Making sure they can't step on each other's toes, uptime of each Lambda. You can kind of think of each as a microservice. And uptime here, I don't mean necessarily is it available? It's almost guaranteed to be at least available. But as code changes over time, you need to sunset Lambdas for one reason or another that has implications to any other number of Lambdas. So you kind of have microservices challenge all over again. So there are very real concerns.

The one kind of North Star for me, if I were to look 10 years ahead and say, “Well, how do I think this is all going to evolve on the backend?” I would look to something more like a single programming language that allows you to author a backend, and the act of compiling it figures out how to split that across a number of services. So if you imagine a programming language, like let's take TypeScript, for example, and instead of just having array.map, you have array.parallelmap, where the parallel in this case means split out each mapping operation over as many Lambdas as you need to to achieve kind of hyperscale, right?

I think what's going to happen is you're going to see more of the infrastructure and code keep merging until you get to a single language that manages all these questions. And at that point, using the right patterns becomes a lot easier, because you're talking about code and infrastructure in the same breadth, as opposed to two sort of totally separate things. And there are some bold companies going down that path today. Like I don't know if you've gotten a chance to see Unisonweb, but that's essentially what they're trying to build, is a single programming language that also encompasses infrastructure and abstract –

[00:24:49] JM: And Dark. Dark, also, right?

[00:24:50] AW: And Dark. Yeah, exactly, right? Dark has even bolder vision, which is to not just do the backend, but the entire deployment model, the coding environment, the language. But yeah, those are probably the two products and companies that I think, are looking towards more of the end game, while the rest of us kind of inch there. But if I look at my own codebase through that lens, that kind of helps maybe answer some of these questions around how you might think about splitting your code and when it makes sense and when it might not.

[00:25:21] JM: What are the infrastructure choices that you've made outside of just Lambda? What are the other pieces of infrastructure you build off or the other monitoring tools? Whatever other tools you can share?

[00:25:33] AW: So I think we've got a fairly – An otherwise fairly Vanilla AWS application. So I think what I'm about to mention to you will not shock anybody, but we have an Aurora RDS cluster. We use copious SQSQs. We use S3 to store temporary data. We have a Redshift cluster to handle analytics. We use step functions, which are just fantastic for coordinating kind of long-lived operations using Lambdas. I think that's all great and fantastic.

In terms of monitoring, we lean heavy into CloudWatch and CloudMetrics. There are, of course, entire companies dedicated to providing better UIs to your logging and metrics than what you get from AWS, and some of them are fantastic. We also use Datadog to do some of our application metrics. But otherwise I think it's fairly, fairly Vanilla in terms of our infrastructure.

[00:26:29] JM: Revisiting the company side of things, what has been your strategic expansion into supporting different enterprise use cases? Like take me from your MVP today and how you chose the product expansion.

[00:26:50] AW: That's a great question. So the MVP was very much focused on telling you what you didn't know, right? You didn't know these things. We're going to tell them to you. And as we started getting that into the hands of more teams, more IT folks, around the ecosystem, you start learning about the other problems they have. Like why do you care that there are things you didn't know about, right? Sort of the next part of the question naturally. Like you only care to uncover the software because you're trying to do X. Well, what is X? X can be I need to manage costs. It can be I need to understand my security footprint. I need to understand my compliance posture, or my GDPR, my ability to even execute on GDPR if a customer would ask me. Or I need to understand what these things are so I can efficiently get people in and out of them or maintain operational continuity. There all these different reasons people want that data. So once we started giving them the data, one wonderful thing about b2b, and I've done consumer, I've done education, I've done some b2b. A wonderful thing about b2b is that your customers and your potential customers are very clear about what it is they want. So we started hearing all those use cases, "I want to be compliant. I want to save money. I want a more efficiently on and off-board employees."

And from there, I think what you do is you look at, "Well, where can we be most helpful?" right? Where does this data set really shine and how can our software do more for those folks? And, strategically, for us, what we discovered as we went was there's going to be three layers to our product, the system of record, getting all the data in one spot and making sense of it, de-duping it, merging it, presenting it in a really helpful way. A workflow layer for a variety of use cases, everything I just described, you're trying to accomplish. If our data can help you accomplish it better, then our data ought to be able to set the project plan for you dynamically. So we knew we were going to have a workflow layer. And then the last thing that we also knew is that, fundamentally, for all these various concerns, given how decentralized IT decisions are, how apps are adopted, there's no future in which an IT product does not involve everybody in the company in some capacity. So we need an employee portal experience, which is to say, if we need to ask you, "Hey, which of these tools you're still using?" We need an ability to contact each employee in your organization to get those data points from them to help IT do their job better. Again, because not all of it can be automated, certainly not sentiment, but even the long tail of tools can't be automated. So we started building through that stack of those three layers, adding the most value that we couldn't kind of prioritize in that way.

And so, for us, I could get into the specifics of each of those things that we did, but the overall strategy was deliver people the best source of truth they have never had before. Then use that truth to dynamically help them accomplish the projects at-hand and involve the company as necessary, because it's just something you have to do in today's day and age.

[00:30:03] JM: On the competitive front, I don't know this area as well as a lot of other infrastructure areas. I think one of the competitors that comes to mind is Rippling. Am I right about that? Is that the domain that you're playing in?

[00:30:17] AW: Yeah, it's interesting. I think we definitely have direct competitors. So they're the Intelo's, Tory's, Productive's, Zylo's. There're a number of folks who have sprung up to do SaaS management. There are some IT companies, for instance, Bettercloud that provides a lot of IT automations that had started trying to offer SaaS management as well. And then there are adjacent companies who I think our vision is just very similar to like Rippling, right? And so I think maybe your audience hopefully knows Rippling, but if not, they are company started designed to do HR plus IT. It's actually started by the founder of Zenefits. This is his second go around at this. So there're a lot of similarities there. And their essential view is if you know the truth of who's in the company and what the team layout looks like, then onboarding a new employee from payroll, through benefits, through software, and even devices, it kind of all flows naturally. Rippling tends to serve companies smaller than us, where a single decision to address HR sets of concerns alongside IT concerns make sense. But as you get into larger companies that breaks down, it's very hard to imagine one product that both perfectly satisfies the needs of IT and perfectly satisfies the needs of HR. And so we tend to play more upmarket than Rippling.

At the super scale end, you maybe have a company like ServiceNow, which is one of those like dramatically large companies very few people seem to have heard of outside of the industry, but they're massive. They serve giant companies of the world. I'm not sure these exact customers, but think that Walmarts, Toyota's, and been doing it for decades, and it's very similar in a way, right? They have one data model, one platform, many use cases. That's one of their kinds of slogans. Because managing this stuff is just very complicated.

And I think kind of we're all sort of siblings in this in that it's all very complicated, whether you're a sub-100 person size Rippling consumer. Someone using Blissfully in sort of the midmarket. Maybe you're using ServiceNow at the largest end. All these companies have similar problems, because the rise of software has just been so dramatic and powerful that it just creates the need to have these products that help you wrangle it all.

[00:32:41] JM: And is this like a technology category that basically every large company wants?

[00:32:50] AW: Well, I'm biased? I hope so. I hope that the value we're offering is something that people want. I think the answer is going to have to be yes, for a couple of reasons. One is, all the data we have and that other players have suggests that the rise of software is not going to slow down. And in fact, here's an interesting stat for you. The top three expenditures for companies traditionally are payroll, your lease, and software. And in the post-COVID age, number two and three there just flipped, right? With software taking sort of the second position for many companies at this point. It's probably not hard to imagine a future where this starts rivaling even payroll, right? But that's a different conversation.

So when you look at the rise of software and the explosion of niche tools, so much of your company's dollars are going to software. And just the sheer number of applications is going up. That if you care at all about leverage, efficiency, all these things, you're going to need to have a great understanding. And it's not something a human can deliver on. And no sleight to my fellow people, it's just if you're trying to maintain a spreadsheet of this, yes, you may be able to, for a brief moment in time, nail the exact vendors that you use by laboriously serving all various tools, data points, people. But the moment you complete that list, it's immediately out of date, which means it wasn't a great use of your time. And that doesn't even get into, "Well, who are all the users of those applications? How much do we spend on that?" It's far too much data for people to manually maintain.

And then on the other end of pressures, you've got the rise of various security standards, right? So let's take SOC 2 is kind of the compliance craze that's sweeping the b2b nation. Every company wants to know that their vendors are some minimum level of responsible with their data, their security practices, etc., so that there won't be some massive leak or some discontinuity of service. And so we've all kind of picked a few of these standards that are becoming, frankly, table stakes for b2b companies. If you're looking at one vendor or another and you've got one that has clearly invested in security and compliance, it makes it a lot easier to go with that vendor because you know that they care about the same things you do. And that sort of spreads virally, right? Like moment you start caring about it, then other vendors who aren't running your business, they need to care about it. Then in order for them to deliver on it, their vendors have to care about it as well. So that's spreading rapidly through the b2b ecosystem. I don't think that's going to slow down at all. I think it's going to accelerate. Other things like data privacy for consumers. I mentioned GDPR earlier. That's another one of those things that matter. And so understanding your vendors and their investments and then managing how you collect that data, or if you use Blissfully, we'll bring that to the table. We already have all that data on most of your vendors. This kind of management, I think, becomes table stakes for companies that use a lot of software. And the data suggests that all companies are going to use a lot of software.

[00:35:55] JM: The number of integrations that you must have to build is just going to grow and grow and grow and grow over time. What's your strategy for managing those in a sustainable fashion?

[00:36:10] AW: This is a great question. We actually went through a major project last year to ensure that as we keep delivering integrations to our customers, our ability to maintain them is only getting better and better. And so here I would point out a few things. Number one, we are big believers in type systems, as I mentioned before, which means we've got a lot of great interface definitions under the hood that we know that if we build to that interface, and the functionality is correct, that integration will seamlessly fit into all the downstream value that our product can leverage that data to provide. So we take a lot of time to get our domain modeling right so that we know exactly, when we're looking at a new app to integrate, all the various interfaces we want it to support. And then if we build that, we feel confident all the downstream systems work. So standardizing how different data sources or automation targets seamlessly fit into your infrastructure so that you're only making decisions on the edges is incredibly important. So we spend a lot of time doing that.

Next, you get into actually offering those. So we've spent a lot of time with things like smart cogeneration, and really isolated testing environments, so that we can very rapidly spin up these new integrations and test them in isolation before exposing our system to them. And then thirdly, there's an operational concern to all this, which is once you've got all these integrations running, how do you manage them? How do you report errors both to your dev team, or your ops team, to your customers, when they can't be solved automatically by our system? And there, again, we've taken standard interfaces, this time in the form of step functions, such that all of our integrations are managed by the exact same set of operational code so that all that error handling, recovery, time of day, rate limiting is wholly handled by one system. And so we've invested over and over and over in our leverage to be able to author integration after integration after integration precisely because it's going to take hundreds. And I don't think we'll ever stop building integrations. It's sort of the name of the game to service our customers in the best way. So yeah, we've invested a lot to give our small team just an incredible amount of leverage.

[00:38:27] JM: How many people you got working for you now?

[00:38:30] AW: So, on the engineering team, we're 10 people. And I'm both proud to share that, because I think this team has done a lot, and also because we're hiring. So if anybody's listening, of course, please reach out to me if any of this sounds interesting. The split is about two-thirds backend engineers, one-third front end engineers. And it's a fully distributed team. We actually started sort of 5050 on-prem and remote. And over the course of just finding great talent to work with, and certainly the pandemic accelerating some of that, we've leaned into a fully remote engineering culture. So we have folks in Europe, we have folks in the West Coast, the Midwest, the Northeast. I don't think we have the Southwest yet, but we could. So it's been a small team, but it's been a highly efficient team. And I'm very proud of the work they've done.

[00:39:20] JM: So what's the division of labor across the team?

[00:39:23] AW: Yeah. So, it's funny. When I hire, I tend to try to hire for a couple of different attributes. Number one is product thinking. My assertion is that engineers or product people with a much kind of more technically narrow focus. But if you're dealing with the domain and you're trying to generate outcomes for customers, you are in fact a product person. So I tend to try to hire people that have product sense whether they're going to do frontend or backend. All of it matters. You're making decisions constantly. And if you're looking for folks like that, you actually tend to find more full stack curious, if not outright capable folks, because they're interested in delivering value to customers. And they follow that value creation, whether it takes them from database management through API design through the frontend.

So while the team split, maybe two-thirds backend engineers and one-third frontend engineers, people cross that boundary all the time in pursuit of delivering the feature that they're responsible for, their pod is responsible for. And so there're no really hard divisions. There're more preferences and areas of expertise. But it's been exciting to watch people kind of push their own technical skill set in order to accomplish something unique.

[00:40:42] JM: Let's zoom out a bit. Tell me about your vision for how the company unfolds over the next five to 10 years.

[00:40:49] AGH: Great question. Well, we serve hundreds of customers today, which I'm very proud of, and I hope that's thousands of customers over the next couple of years. So just from a number of companies were able to help, certainly, that's up there. In terms of the impact that I'd like this organization to have, I would like folks to realize that if you have a platform for IT that can fundamentally help you accomplish your goals and turn how we think about IT away from being a cost center into a point of organization-wide leverage, that would be a huge spiritual win.

So, I think classically, if you look at how some companies at least have viewed IT, its cost center, right? Its cost of doing businesses to tax. You need people to help manage the technology, but they're just there to kind of keep the engine greased and working maybe with some sort of bare minimum level of responsibility. And I think that's a little unfortunate. That kind of forgets the fact that, ultimately, every person in the organization that you hire is providing leverage. And technology provides each of those folks leverage in turn, like that's the whole point of it. And so what a real win for this company over the long haul would be helping folks really realize and demonstrate and get great value from the fact that a well-run IT organization empowers everybody throughout the company to provide your organization just far more leverage and impact on the world, whatever your mission is. That's how Blissfully succeeds, is we help companies empower people through technology in a very real way, not as a catchphrase, but day-to-day, you always have access to what helps you do your job better. And we're helping companies realize that.

[00:42:43] JM: What are the biggest technical challenges that stand in your way? And, I guess, business challenges as well? Give me a perspective for what stands in your way.

[00:42:52] AW: Yeah. Well, I think from a business/technical challenge, and we talked about this earlier, is just that the number of applications and the diversity of sophistication of applications is all over the map. There are thousands and thousands of apps. New ones are getting created all the time for every niche. And they're not all created equal. Some have API's. Some don't. Some support single sign on, some don't. Some support automatic provisioning. Others won't. When you start looking at all the various mechanisms these tools have to allow themselves to be managed, or that make it resistant to manage them, it is just very hard to wrangle that chaotic and that diverse of an ecosystem into a single platform. So I think this is both a real challenge we have. It's certainly a very real challenge for our customers in a world where a Blissfully doesn't exist. And kind of like interestingly, I mean, maybe intuitively, or counterintuitively, depending on your perspective, the harder it is, sort of the more meaningful our work becomes, right? Because as an individual company, you would have no hope. And so you need a single organization to invest in getting all of that under control. That makes our job very tough from a technical perspective, a business relationship to all these various different entities as we try to do our best. It's just very complicated. And I think we are committed to the mission. We enjoy our success when we have it. We get frustrated when we can't go fast enough or when we suffer a setback trying to deliver something unique. But I think that diversity of the ecosystem technically is the same reason people value us and it's the same thing that makes doing what we do just insanely challenging.

[00:44:43] JM: Can you tell me any other unconventional infrastructure decisions you've made?

[00:44:48] AW: Well, there are a few things that we've built internally that we get a lot of use from that are powerful. And so I don't know if they're unconventional, but they're more things that I hope people would give a go. So we practice continuous testing, continuous deployment. But we've tried to bring the robots in even earlier into our development process. So I mentioned earlier – Actually, there’s a great quote by Agent Smith in the Matrix, if anybody's a fan, which is, “Never send a human to do a machine's job.” It's actually a phrase we've used internally. Our chief architect, Jacob, clued me into it. But, for instance, if you're ever doing a PR review and you are watching your team give the same feedback over any number of PRs. Why are we having humans provide that feedback? You have to ask yourself, “Is that useful? Is that a good use of their time? Or should they be focusing on higher level issues of code quality and organizational strategy?”

So we use a great tool called Danger JS, if you're familiar with it, which allows you to code up rules all your PRs are subject to. So whether it's splitting out a migration from a code change, and that's one of our policies here. You can't ship the two simultaneously. They need to be separated. We're not unique in that, but we have the bots enforce that. Or if you fail to check for certain conditions on the backend, or certain database schema changes are being proposed, the bots will automatically flag past operational issues to you in that moment in time. And we don't always have those issues serviced in the PRs. There's not like a static checklist. People get fatigued. They stop paying attention. So we have our bots strategically guide us as we go. And we're constantly adding that library.

We do the same thing on the front end using Elm Review. And, in fact, Elm, again, it's just a powerful language. The bots can propose the changes outright, and you can just accept them, which has been nice. So we leverage a lot of code gen, automated code review, code manipulation, to give us more leverage. So that's been great. And two other techniques that we use are maybe a little bit more social. I'm a big fan. I think the team is a fan. And I'd be curious if other organizations do this. One is we've instituted a policy of rubber ducking in Slack channels. So every engineer has essentially what amounts to a public engineer's notebook. So RD, rubber ducking/aaron as an example. And day-to-day I just sort of vocalize what it is I'm working on as I'm working on it.

And the reason we do this is, because in a distributed organization across time zones, across concerns, we still get together for stand out in various meetings to kind of discuss what we're working on. But if I am working on something that's pretty technically involved and I get to a place where I'm stuck and I want to pull someone in. Well, I don't need to spend time bringing them up to speed synchronously on what it is I've been working on. I can just add them at that point in my rubber ducking channel. And either right then in there, or at their own convenience, depending on time zones and their workload, can get up to speed and provide the answers to unstick me. So we try to get this culture of developing in the open air publicly, because it accelerates collaboration when we need it. And we even taken this a step further where every story that gets created in our issue trackers, we use Clubhouse, automatically gets a Slack room spun up around it. Automatically synchronizes data from Clubhouse into it as changes are made, pulls participants in and out of that channel. So people can have very freeform long discussions about the code about the feature in a space that doesn't pollute the story, that doesn't pollute popular channel like dev, that allows people really to keep establishing a shared context. So we lean very heavily into this kind of sharing your work in public to aid collaboration and make for better products.

[00:48:30] JM: Aaron, thank you so much for coming on the show. It’s been a real pleasure talking to you.

[00:48:33] AW: Yeah, thank you. It's been mine.

[END]