EPISODE 479

[INTRODUCTION]

[0:00:00.3] JM: 10 years ago, if you wanted to build software, you probably needed to know how to write code. Today, the line between technical and non-technical people is blurring. Website designers can make a living building sites for people on WordPress or Squarespace without having to know how to write code. Salesforce integration experts can help a sales team set up their complicated sales pipeline without having to know how to write code.

Shopify experts can set up an e-commerce store to your exact specifications without having to know how to write code. WordPress and Squarespace and Salesforce and Shopify, these are all fantastic services, but they're not compatible with each other. I can't install a WordPress plug-in on Salesforce.

Now, imagine this from the point of view of plug-in creators. Plug-in creators make easy ways to integrate different pieces of software together. Take PayPal as an example. PayPal wants to make it easy for software builders to integrate with their API. One way that PayPal does this is with a plug-in that is a button that says, “Pay with PayPal.”

If I'm a developer at PayPal and I'm building a button that people should easily be able to put on their webpage so that anybody can pay with PayPal by just clicking a button, I have to create a button that is compatible with WordPress and Squarespace and Wix and Weebly and GoDaddy and Blogger and all the other website builders that I might want to integrate with.

In 2014, Zack Bloom started a company called Eager. Eager was a cloud app marketplace which allowed app developers to make flexible plug-ins that non-technical users could drag and drop on to their site without any technical expertise. Unfortunately, in order for these non-technical users to add any apps from the Eager marketplace to their webpage, they had to drop in a line of JavaScript to their application and that's unfortunately a significant hurdle for a non-technical user.

Eager was a useful distribution mechanism for plug-in developers because Eager allow these plug-in developers to write a plug-in once and get distributed to multiple plug-in marketplaces. So if you are writing that PayPal button for the Eager app marketplace, you would just write it once on Eager and Eager would figure out a way to distributed onto the WordPress plug-in's site and the Squarespace plug-in platform and Wix's plug-in platform and all the different plug-in platform. So Eager actually did build a successful distribution model, but it was not widely used as a way for non-technical users to directly drag-and-drop plug-ins on to their websites.

The question was; how do you build a marketplace for non-technical users to be able to add plug-ins to any website without forcing this non-technical user to write code? How do you make editing and building a website ss easy as a WYSIWYG editor, the what you see is what you get editors, and this is kind of hard, because you look at the business of Wix and Squarespace and WordPress, these are giant businesses and the reason there're a bunch of different website builders is because they don't really play nice together and you have to create this sort of walled garden to make it easier for these non-technical users to build their websites.

But Zach really wanted to build this marketplace that would be interoperable and would be fairly open, and one way that you can actually deploy code to an app without having to insert a blob of JavaScript is to intercept the page at the CDN level. So the CDN turns out to be a perfect distribution platform for these kinds of apps like the PayPal button, this plug-in idea, because even if your non-technical user, you have to integrate with the CDN oftentimes, because the CDN is going to be the content distribution network that gives your app global scalability and reliability and responsiveness.

When these users integrate with a CDN, the CDN could do the work of inserting that blob of JavaScript that allows the plug-ins to be added to that user's webpage, and because of the opportunity for the integration between the plug-in marketplace and a CDN, Eager was acquired by Cloudflare and Eager became Cloudflare apps.

Zack Bloom joins the show today to discuss the motivations for his company, the engineering behind building a cloud app marketplace and the acquisition process for his company, Eager. It was a pretty interesting show because of the subtlety of what was difficult about making his marketplace work, and I think it just goes to show why marketplaces are so difficult to build.

I hope you enjoy this episode as much as I did.

[SPONSOR MESSAGE]

[0:05:24.0] JM: GrammaTech CodeSonar helps development teams improve code quality with static analysis. It helps flag issues early in the development process, allowing developers to release better code faster.

CodeSonar can easily be integrated into any development process. CodeSonar performs advanced static analysis of C, C++, Java, and even raw binary code. CodeSonar performs unique data flow and symbolic execution analysis to aggressively scan for problems in your code. Just like battleships uses sonar to detect objects deep underwater, engineers use CodeSonar to detect subtle problems deep within their code.

Go to go.grammatech.com/sedaily to get your free 30-day trial exclusively for Software Engineering Daily listeners. CodeSonar analyzes your code and it delivers a detailed report. The CodeSonar user interface provides all the information that you need to quickly understand the reports. Follow cross-functional paths, understand cross-references, quickly navigate between files and visualize large pieces of your code.

Go to go.grammatech.com/sedaily to get your 30-day free trial and unleash the power of advanced static analysis.

Thanks to GammaTech for being a sponsor of Software Engineering Daily.

[INTERVIEW]

[0:07:02.7] JM: Zack Bloom is an engineering manager at CloudFlare. Zack welcome to Software Engineering Daily.

[0:07:07.6] ZB: Thank you so much. I really appreciate it, Jeff.

[0:07:09.5] JM: You started a company in 2014, called Eager. The goal was to make it easy to integrate apps into websites, and then there's a lot more complexity beyond that simple mission of easily integrating apps into websites, but describe that problem in more detail. What is that actually mean integrating apps into a website?

[0:07:35.0] ZB: I was hoping you knew. I don't. No. It's a great question. The back story of it, of Eager itself and kind of how my cofounder, Adam, and I came to start that company, is we were working at HubSpot in Boston, which is a company that people kind of heard of now, but hadn't necessarily heard of at the time. When we were working at HubSpot, we started a team called the open source team and all we did on that team all day was build open source projects that we thought would be valuable inside HubSpot and that we could release and kind of improve the way that people thought of this Boston tech company in the JavaScript community predominantly.

And so we were pretty successful at that. We were able to release a lot of great projects that gave a lot of people value and they got integrated into Twitter bootstrap an tons and tons of stars on GitHub and all of that stuff that's great for developers, but we eventually realized that no matter how many people saw this on Hacker News, the vast majority of websites, 99% of websites, are not ran by technical people. They're not ran by people who have capability to copy and paste this embedded code somewhere in your HTML and they're not going to find anything that we do on GitHub.

We were kind of playing to a home team and we wanted to answer the question of; could we find a way to make all of these open source projects that we built and that other people built and also paid projects, SaSS projects, tools that make websites and companies better, accessible to all the people who aren't technical enough to install them?

One of the reasons that that's so cool is when you start thinking about it, you realize that the way a lot of the SaSS companies live right now, and I say right now, because it's still for the most part true, is kind of mobile apps before the iPhone. Where if you want to get something installed, you need someone who's pretty technical. They need to know, in that case, how to like use the JDK or something like that, and if you want to run one of these companies, you need a really big sales and marketing team and you need to spend years.

Imagine starting a new analytics company right now that was going to compete with Google Analytics and how many years of grind that would take, because just getting someone to get that embed code installed is such a technical task and the people you're selling to aren't necessarily technical. They're business owners.

We wanted to kind of square that circle. So we wanted to build an app store where you could install things being not technical on to your website, and you could preview them and decide that this is what you like. Configure all the options and eventually register an account and get it installed on your site without knowing any of the technical stuff that we all live in every day.

[0:10:05.6] JM: The phenomenon that you're referring to where you have people who are "non-technical", being in charge of much of the operations and even deployment of software. I saw this last week, I was at Dreamorce, and I didn't know much about the Salesforce ecosystem. But Salesforce is an example where you have people doing very complex development and integration that is at a slightly higher level of abstraction than software engineers, and you also see this with complex WordPress deployments. People configure these very unique WordPress apps out of plug-ins, and there's this is huge plug-in ecosystem.

You have this line between non-technical people and technical people that begins to blur as it becomes easier to compose different applications together without writing any lines of code. What was the level of technical expertise that you were trying to help with Eager.io.

[0:11:12.7] ZB: And I think what you describe is kind of fundamentally true and it's going to continue being true, which is the level of complexity that is possible, the things that you can accomplish without being technical has been getting continuously better. So it used to be that if you wanted to have a website you needed to know HTML, and then we live in a world with a million different CMS tools now, and we currently live in a world where if you want to use TensorFlow to do some machine learning, you need to be technical, but pretty soon we're going to be in an era where non-technical people can do that too and we need engineers who instead be thinking about the next wave of complexity.

So you do kind of want to live in a world where more and more power is pushed down to be almost a commodity so that every store you shop from can give you really good recommendations, not just amazon.com, right? We want all these technology to be accessible to everyone, and so we have to really think consciously about how we can engineer ourselves out of a job and go focus on the new things and let people experience the things that we figured out yesterday and last year on a commodity basis, on a way that they can use without having to be super technical and live in that world all the time in saying, “Our goal was always to move —” I will say downmarket, but downmarket can mean a lot of different things, “to move towards people who are less technical,” because there will always be a pyramid. There will always be more people with less knowledge than more, because every day that you show up to work and you learn something you become more knowledgeable than a big set of people that didn't have that experience that day. We’re all slowly rising up this pyramid and there’s always more people at the bottom.

So when we first launched Eager, there was kind of this fundamental flaw to it, really, which is; you had to install an embed code on to your website in order to never have to do that again, and that was not an ideal experience. That was a convenient experience for people who are sort of technical or technical enough, but we had to invest a lot of energy and effort to try to slide down that pyramid. How do we build WordPress plug-ins? How do we make it so that you can take any one of these apps that someone had built and package it for other app stores so we can leverage how easy they make it to install something to make it easy to install our thing.

Then eventually this question that I think we’ll touch on later, which is; how can we partner with platforms that make it very, very, very easy to install this stuff, because they live at a little bit of a different place in the request stack, and obviously I’m talking about Cloudflare.

[0:13:33.3] JM: Indeed. Yes, we’ll get into that. You were eventually acquired by Cloudflare and Eager became Cloudflare apps. So prior to the acquisition, it was the Eager.io app store. To help people understand what we’re talking about here, can you give a few examples of the apps from Eager that people used to integrate into their own websites?

[0:13:58.1] ZB: Absolutely. A bit category y is anything that would've traditionally you’ve installed with like a tag manager that people have heard of, so analytics. You can certainly install analytics. But you also get a lot of visual stuff on that page, so maybe you want to collect people's email addresses. You have a business and you think it would be great to be able to build an email list. Okay, great. There are many. You can put a bar at the top. You can put a bar in the corner. You can do many different things to start building that email list and growing your business. Maybe you want to make your website faster. Okay, great. You can install a tool that's going to use JavaScript, kind of like what people do with Rails, with turbolinks, to accelerate the speed that your page is going to operate at. Or maybe you want all of your images to go through CDN. Okay, great. You can install JavaScript. It's going to use mutation observers to swap out URLs for your images and point them at a CDN instead.

So a lot of it was trying to push the limits of what JavaScript was capable of in the browser, because that was kind of the sandbox that we had to plan at the time. Then a lot of it was installing existing SaSS tools, live chat, things like that that people are aware of and they're familiar with, but we just want to make incredibly easy to ourselves.

[SPONSOR MESSAGE]

[0:15:11.8] JM: You are building a cloud-native application and you need to pick a cloud service provider. Maybe you're just starting out with a new app, but you have dreams of scaling into the next giant unicorn. Maybe your business had been using on-premise servers and you want to start moving some of your infrastructure to a secure cloud provider that you can trust. Maybe you're already in the cloud, but you want to go multi-cloud for added resilience.

IBM Cloud gives you all the tools you need to build cloud-native applications. Use IBM Cloud Container Service to easily manage the deployment of your Docker containers. For serverless applications, use IBM Cloud Functions for low-cost, event-driven scalability. If you like to work with a fully-managed platform as a service, IBM Cloud Foundry gives you a cloud operating system to control your distributed application.

IBM Cloud is built on top of open-source tools and it integrates with all the third-party services that you need to build, deploy and manage your application. To start building with AI, IoT, data and mobile services today, go to softwareengineeringdaily.com/ibm and get started with countless tutorials and SDKs. You can start building apps for free and try numerous cloud services with no time restrictions. Try it out at softwareengineeringdaily.com/ibm.

Thanks again to IBM for being a new sponsor. We really appreciate it.

[INTERVIEW CONTINUED]

[0:16:48.2] JM: I think one way of is like components. If you're a React or a Vue.js or an Angular developer, there is this huge library of components where you can go to GitHub and find a component and you can install it using NPM and you can use it in your app and it's pretty cool if you're a developer. You just have this huge market of components. Well, I'm saying market, because it's all free with open source, but these are framework specific. These are only accessible to developers, and also even developers, it's kind of hard to charge for this code, because it's open source stuff.

So with your app marketplace, it seems like this is a way for non-technical people to have access to the same diversity of components that we have access to as developers with all the open source stuff, but also gives developers a way to have a marketplace for the components that they write and also have a framework agnostic marketplace where if I want to install a PayPal button on my website, I don't have to worry if it's written in React or Vue or Angular. The marketplace you built with Eager just allows people to install it a little bit more seamlessly. Am I describing things correctly?

[0:18:21.5] ZB: Yeah, you are, and you're kind of describing one of those things that we had to do in order to make it accessible to non-technical people, because it wasn't just enough that you can install this. A lot of these stuff has to be placed at a certain location on the page. So we had to write a lot of code to figure out where those locations were when the page initially loaded, figure out how you can place something in the preview, figure out how, at runtime, we're actually going to find that same location on the page, because we do want to let people do something like place a PayPal button and have it make sense to someone who isn't technical enough to understand how their Vue or React is rendering or how their regular old plane JavaScript jQuery website is rendering.

The charging for that thing I think is super, super important, at least to me, because my real dream with Eager was always that a developer in his or her basement would write the next new great tool that solves some real problem and would get it on a million websites and would make a $1 million the next day, that you could start a business, and this was an actual app store, just like the Apple App Store, that moved a $1 billion through it, that created businesses that allowed people to totally change the way that they work. Because right now if you want to do that, if you want to build tools for people to make websites better and solve business problems for people, solve accessibility problems or solve speed problems, you kind of have to start a giant company, and most of that company is going to be devoted to convincing people that what you built makes sense.

If we can do that kind of in the way that Amazon has gotten rid of a lot of that, because you have reviews and you have ratings and you have social proof and you have a consistent installation or buying experience in the analogy, we can suddenly make it, so that developers have the ability to make a living, building the stuff that makes companies better, and that would be very, very cool.

[0:20:07.0] JM: The interface on Eager, what became Cloudflare apps, it allows the user to see how components are going to appear on their website. So if I want to put a PayPal button on my page, I can put in softwaredaily.com, and softwaresdaily.com gets displayed in an I-frame on the right hand side of the page and then I can just click on the place in the I-frame page where I want that PayPal button to appear. This is similar to the WYSIWYG, the what you see is what you get editor that I've used with Squarespace or WordPress. These sites are beloved by non-technical people for building their websites.

If I am inserting a PayPal button into my page with the app building tool that you've built, what's going on there? Where is the code getting inserted and is it accessing PayPal APIs? Doesn't PayPal offer this kind of thing where I can insert a PayPal button? I guess describe using the PayPal button app as an example, how does somebody add that functionality to their page?

[0:21:30.8] ZB: Obviously when we talk about the PayPal button, what you are looking to do is you have a website, maybe it's for your soccer team, and you want a way of raising money, donations, or maybe you have a website that sells home-baked cookies you want a way of actually charging people for it that's pretty low impact. You don't need to set up all of Shopify or anything like that. You just need someone to be able to charge and you need to get an email when they pay, right? So being able to just place a PayPal button below the picture of a cookie is actually a pretty good simple solution to that.

The way you described the preview experience was really great. You have options on the left and then you have a preview of the actual user's website with this app installed on the right in an I-frame as you said, but that kind of masks a somewhat hilarious amount of complexity, because on the left, these options have to be defined by the app creator somehow. We use JSON Schema, which is an open source format for essentially defining a form or defining a data format in JSON.

Want to do that? You have to have all sorts of different fields that you support in order to support everything that apps want to do, including the ability to, for example, OAuth into an account with PayPal. You already have a PayPal account or maybe you need to register one. When you're installing this app, you need to be able to OAuth into that account. So that's a whole another input type that encodes the idea of an account and that fact that someone's logged in.

Once you have all of that on the left, you often need the ability for the app creator to manipulate those options as people pick things. For example, maybe you can log in your PayPal account and then it wants to show you a drop-down of which bank account you want the money to go to. So then you need this whole hooks feature, which is like a bidirectional web hook where we can go out your server and say, "Hey, they just fired this event. They just change this option. They just decided they want to install. Do you want to change any of these options are save anything on your end or have us save anything that will end up on the client?" That's a whole another kind of bucket of complexity.

Then on the right, when you talk about an I-frame, well you can't just I-frame most websites, because a lot of them have frame blocking JavaScript. A lot of them use X-frame options either as a header or as a meta-tag. A lot of them don't support HTTPS, and so if you want your outer frame, you want your website, you want Cloudflare apps to support HTTPS, which obviously we do, then you can't have any I-frame inside of it that isn't HTTP. So you have to be able to do that kind of rewriting.

To do that, you're rewriting all of the URLs and then you need to go through the process of figuring out how to rewrite all the URLs in such a way that all their relative links still work and all the absolute links still work, and then you remember that base tags are a thing, and then you remember that URLs can — Host names can only be 254 characters on Internet Explorer, and you go through, or at least I went through or went through, tremendous list of how can we actually make it so that you can see this preview that it will actually work in the way that we need it?

Once you get all of that, the actual inclusion isn't that complicated. With the old Eager way of doing it, what we would do is bundle all of your JavaScript and all of your CSS into two files. One that we were going to load in the head of the page, and then one that it would asynchronously load in the body. So an app could say, "I have this JavaScript for the head and then I have this JavaScript CSS for the body."

That single initial JavaScript file you loaded would include the head and it would trigger the loading of the body, and we would also include a little bit of JavaScript to give you some sort of reasonable APIs and app developer that made sense.

In the Cloudflare apps world, we still do that, but we have the ability to serve all of that JavaScript from your original domain, because we control the edge. We control your actual domain. We have a lot more capability than we had before. Then we also have the capability to allow apps to a bunch of stuff that aren't just JavaScript browser, which we can definitely talk more about too.

[0:25:22.1] JM: So let's say I am a non-technical user who has built a website. There are other different ways I could build a website, and I imagine that you want to accommodate those different ones when you were building out Eager. I mean this is the obvious problem, but I'd love to hear how you solved it, is you need to basically inject code into these webpages that people have and you need to be able to inject that code into all kinds of different places, because you want to make it really easy for people to configure where they are dropping the PayPal button on their website. You want to turn everybody's website into a WYSIWYG where you can just click and drag stuff. So how do you get the code that you need injected into the users website when it's hosted on a place that you do not control?

[0:26:15.5] ZB: Yeah. A lot of this is iterative, right? You start with this implementation that maybe takes you three days and you think, "Oh great! I took care of that. That is done." Then you spend the rest of your life effectively either fixing bugs or re-implementing it. By the time you get to the third time you've done it, it maybe is halfway decent.

Initially, the way that it would work is we would use cookies to serve a different version of this if you are inside the preview. You would get a special version that we were dynamically generating that would include all your options. Then we realize, "You know what? People really want to see changes to their options much faster than it takes to reload webpage and regenerate this file."

We had to build a whole way for changes in the app to run for changes on the left side, changes in that options area to dynamically change the app inside the website. So that's a bunch of post-messaging and local storage to move between those two worlds.

Then we have this problem of, "Okay, great. You can pick a location on the page," but it turns out the structure of pages change as they load. If you're using React or you're using Vue or you're doing any kind of modification, by the time the app loads, the structure of your page may have changed, and then this is a fun one, different apps can change the structure of your page. So you could install three different apps and one could insert a DOM node right at the top of the page and change everywhere that we thought we were placing things.

We had to move to a two pass placement system, where when the page initially loads, we mark all the locations that we think are interesting, that we think we're going to place an app, and then we have those DOM nodes so that we can then actually placed apps in a deterministic way later on. That's another kind of dimension of things that has to be figured out.

In the world of Cloudflare, it's very similar, but we have the ability to control how that file gets served in the preview separate from how the actual website it get served. So in the preview, you're actually serving the website through a proxy that fixes all of these cookie issues. It fixes all of the X-frame options issues and frame blocking and HTTPS and all the things that would prevent us from successfully I-framing the site, and then it also injects a special version of the embed code that we use that we're going to dynamically generate every time with exactly what options we've mentioned.

[0:28:29.1] JM: So some of those things I didn't quite understand. So let's just say like I'm a user and I'm using Cloudflare apps for the first time and my website is hosted on, let's say, HostGator or some other random hosting site. Do I need to install like a single JavaScript blob that will give the CloudFlare apps access to my page?

[0:28:57.2] ZB: No, and this is the beauty of being a part of Cloudflare. Cloudflare sits at the edge with 180 data centers around the world for over 7 million websites. So that 7 million websites that requests travel through the Cloudflare network in order to get to the user, and that gives us the ability to give people a lot of security and a lot of reliability and protect them from attacks and handle their HTTPS for them and it also gives us the ability to add any apps that they've installed.

All of a sudden if you're one of those 7 million sites or you're interested in becoming one, you don't need to be technical at all, which is if we remember it was the original dream. It was one of the reasons that we were so excited when we got the opportunity to join Cloudflare.

[0:29:40.1] JM: When you were just Eager and you were not part of Cloudflare, was that problem a little bit harder to solve?

[0:29:48.0] ZB: It didn't make any sense. We were telling people, "Hey, you're not technical enough to do this. But by the way, install a JavaScript embed code into the head of your HTML and then deploy." It made zero sense.

Again, we tried to square that circle a bunch of different ways in the way that you do in a startup where you want to solve problems, but we never really got to the point where I could confidently say to the person running that cookie shop or that soccer club like, "Oh yeah! Eager is great. You should be using it. It's click, click, click." It didn't make any sense.

[0:30:21.0] JM: Some people I imagine were on WordPress or Squarespace or Wix, and WordPress, Squarespace, Wix, these website builders, some of them have app stores and that's part of what makes them successful and configurable is because the app store is a layer of abstraction that gives the non-technical user the ability to configure their website. But I imagine that could also be a way for you to get a toehold into that market. You could have a WordPress app that is the Eager marketplace and then that installs the JavaScript blob in a non-technical way.

[0:30:57.5] ZB: So I think you hit on two great points there. One is you can absolutely do that. You could absolutely create an app store plug-in. But one thing that we learned is that people don't install an app store. They don't know what an app store is a thing. No one's ever heard of Eager. So putting that on the store didn't do a lot.

The thing that was more valuable actually was packaging individual apps as WordPress plug-ins. So we could go to companies and say, “Hey, you're going to end up building a plug-in for WordPress and for Joomla! and for Drupal and for all these other platforms,” and that’s going to be a giant pain. Build an Eager app and then we have a tool that will generate all of these plug-ins for you programmatically, and each of those plug-ins will have that live preview that we are talking about inside Joomla! and inside Drupal and inside WordPress, which is something that those platforms don't necessarily support.

That was a pretty powerful pitch, and then of course once they installed that first Eager app, the first Eager plug-in, it put a link on their side bar that then took them to the app store. It then gave them the opportunity to install more things, and that was powerful. Another thing that is powerful actually is the fact that not every CMS has an app store. You're right that the Wix’s of the world kind of have this problem. If not solved, they at least have a solution that they're pretty happy with, but there's always an opportunity in the hundreds and thousands of other app stores — These are not app stores. Sorry. Hundreds of thousands of others — Hundreds or thousands of other CMS's that exist in the world that don't have the infrastructure to build an app store, and so they would be interested in partnering with an Eager to suddenly get access to that without having to do all of the boots on the ground work. So that's another way that those CMSs become apps.

[SPONSOR MESSAGE]

[0:32:48.1] JM: Do you want the flexibility of a non-relational, key-value store, together with the query capabilities of SQL? Take a look at c-treeACE by FairCom. C-treeACE is a non-relational key-value store that offers ACID transactions complemented by a full SQL engine. C-treeACE offers simultaneous access to the data through non-relational and relational APIs.

Company's use c-treeACE to process ACID transactions through non-relational APIs for extreme performance while using the SQL APIs to connect third part aps or query the data for reports or business intelligence. C-treeACE is platform and hardware-agnostic and it's capable of being embedded, deployed on premises, or in the cloud.

FairCom has been around for decades powering applications that most people use daily. Whether you are listening to NRP, shipping a package through UPS, paying for gas at the pump, or swiping your VISA card in Europe, FairCom is powering through your day. Software Engineering Daily listeners can download an evaluation version of c-treeACE for free by going to softwareengineeringdaily.com/faircom.

Thanks to FairCom c-treeACE for being a new sponsor of Software Engineering Daily, and you can go to softwareengineeringdaily.com/faircom to check it out and support the show.

[INTERVIEW CONTINUED]

[0:34:28.6] JM: Right. You're talking about becoming like a market as a service to other CMS platforms. I could imagine when you're talking about Salesforce earlier, you could imagine in a world where a Salesforce does not have 100 million apps in the Salesforce ecosystem. Let's say some new app is starting to pop — Let say Airbnb. Let's say Airbnb decides that they want to make it easier for hosts to set up their own website. Like within Airbnb you have a configurable website and then they started to say. "Okay. But we also want to give these hosts the ability to add PayPal buttons to their Airbnb host pages because maybe they want to take money for their own services that we don't offer on Airbnb. Well, we'll just go to Eager and we'll build our Airbnb. We'll build an Airbnb app store by bootstrapping with Eager." Is that the vision that you were kind of like leaning towards?

[0:35:29.1] ZB: Exactly. So we had done a lot of the work to not just build the infrastructure, but also to collect the apps and nurture the app community. So that meant we had something that these services would find valuable, the services that aren't the handful of giant CMSs that have enough reach that they could consider building their own. That was actually — This is a little bit of inside baseball. That was actually how we started talking to Cloudflare. It wasn't originally, "Hey, we would love to be acquired, because we were running a startup and it was going fine, and like every startup, it had its ups and its downs, but we were having a good time." We want to Cloudflare and said, "You have 7 million sites. Would you be interested in having our app store integrated into Cloudflare?"

Then we started having those conversations and it turned out if we were a part of Cloudflare we could make this app store do a whole lot more than we could if we didn't have that edge network, and that kind of changed everything.

[0:36:23.7] JM: Yeah, and Cloudflare had actually been building their own apps marketplace, but I guess they were encountering their own set of problems. What was the Cloudflare apps world like before they met Eager?

[0:36:39.5] ZB: Yeah. Obviously, people at Cloudflare pretty smart and they recognized all the way back in 2011, which is very early in the history of CloudFlare, that it would be valuable if people could install extra stuff on to their websites. We have is edge network. It can do anything to these websites essentially. Why can't we let people install all sorts of valuable stuff?

So they built a version of this app store, but the problem with CloudFlare is it's super successful, and it's not just successful and that we have a few hundred or thousand enterprise customers we have to keep happy. We have 7 million sites and we deal with over a trillion requests a month. So there isn't a lot of available bandwidth within that company as it's scaling up to those numbers to continue iterating on this app store. So that 2011 app store that did launch kind of stuck around and didn't get a ton of love or attention, because the company was trying to figure out how do we process two petabytes of log files a day?

That was why it kind of made sense all the way in 2016 to say “Oh, okay. Maybe now that we have a little bit more resources. Now that we have a little bit more breathing room, why don't we take another pass at this?” Like I was saying, sometimes it takes the second or third approach at building something before it actually gets to a point that it starts working.

[0:37:58.4] JM: Yeah, totally. As we've described, CloudFlare took an interest in Eager enough to acquire you. So what was that acquisition process like?

[0:38:11.1] ZB: I think that’s a really good question, because I don't think it's something that a lot of people — You can read press releases and stuff like that, but you don't get a lot of visibility into how and acquisition works, and I've only ever been involved in one. So I have limited visibility myself, but the biggest thing that I would say is it’s meeting people and talking to them and finding out if there's a meeting of the minds and if you can get along with these people and if they get along with you and you like the vision of the company and the people that you meet. So it's very similar to taking a job, because you fundamentally are taking a job. You're taking a job at this new company and you're not just doing it for yourself, but you’re doing it for all of the work that we had put three years of work into this.

The fact that we were able to continue working on the same thing made a big difference. The fact that the people that we were going to be working with totally shared our vision, actually had more aggressive vision than we did made a huge difference. It's kind of a gradual courting process over multiple months where you figure out, “Okay. I like these people and maybe I want to work with these people and maybe working there isn't going to be horrible. It's going to be great.” In our case, fortunately, that's how it turned out. Obviously, everyone is not that lucky.

[0:39:23.3] JM: When you are part of an acquisition, does the acquirer typically make some kind of offer where the value of your acquisition price is going to be related to how your acquired company performs under the acquirer post-acquisition?

[0:39:45.3] ZB: Well, I don't think I can talk about the specifics, I think.

[0:39:46.5] JM: Okay. That’s why I asked in generality.

[0:39:50.9] ZB: Yeah, I know. I think there are options. The way that I've heard this describe before is that it can be very dangerous to make deals like that, because there are always differences and perspective on whether something succeeded or not and sometimes the company can change its mind about what it wants to do.

It's dangerous for a billion dollar company to be locked into these agreements where it's either kind of committed to accomplishing the specific thing or supporting this person and accomplishing this specific thing so that they can get personal money, or the company has to bow out of that agreement and kind of put this person in a bad place.

So if I was making suggestions to someone, I would suggest that you can do it based on time. You say like, "You have to be committed to this company and we want you stick around." And you can decide how much stock is this person getting so that they really feel invested in the success of the company. But I personally wouldn't recommend to someone that they kind of a lock down in writing that in two years this company still going to be interested in accomplishing on a specific thing, and if it isn't, I, the human being person, end up in a bad place." That can be kind of [inaudible 0:40:56.9].

[0:40:58.1] JM: Yeah, okay. The performance-based, having to put yourself in a situation where your acquisition is going to perform well and that's the only scenario where you're going to get compensated for that acquisition is a dangerous thing to do because the intentions of the company might change, other things might change, and circumstances might occur where the acquisition doesn't end up being positive-sum, but you don't want to take the brunt of that because you give up your baby. You give up the acquisition that you've worked hard for, and you want to guarantee some level of compensation for that. That's a useful piece of advice.

[0:41:40.9] ZB: And at the end of the day, you don't really want a whole bunch of people running around this company only interested in their own, their own deal, right? You want all these people to care about the success of the entire company, and that to me is easier to do when they don't have a huge like payday or something tied to their particular little thing going a particular certain way if it turns out that's not actually not what's best for this whole company in general.

[0:42:06.0] JM: Before you were acquired and you were on your own as a startup and you were kind of encountering these marketplace problems, and marketplace problems in a variety of ways. One; you are having trouble identifying who is the customer of Eager. You want to have this app store for non-technical users, but you had an issue of overcoming the technical barrier of inserting that first blob of JavaScript that allows you to use the Eager app store. I imagine that also led to it being somewhat difficult to sell the marketplace to developers to build on top of Eager, because those developers will only be interested in getting distribution for an app store that was usable to people, but I guess you were able to start solving those problems with the ideas around, okay, becoming this plug-and-play marketplace for CMSs that didn't have their own marketplace.

Can you talk about some of the difficulties of building a marketplace for developers?

[0:43:10.4] ZB: I mean, isn't it a great question? Because you do have two sides. The situation we're in right now where we get to go to developers, "Hey, we have 7 million sites," is pretty good. The developer here that and they go, "Okay. I can build something here and someone's going to install. Hopefully a lot of people are going install it." If what I build is good, it's going to get traction.

But when you have hundreds of sites and then thousands of sites, there just is so much less volume, so many less eyes, so many less dollars that are available, and so how do you convince a developer that they want to build this?

One way that we answered that is to give them a lot of these other extractor capabilities, "Okay. You can actually have a WordPress plug-in," and, "Oh, that live preview, you can now send anyone to like preview you want with any URL and they're just going to see your service, and we built an entire way for you to take that live preview and embed it into your site." So it's, "Oh, okay. You're going to build an Eager app? Well, now your site, your landing page gets live preview." And you have install buttons that anyone can click and get dropped into an Eager preview and they can try live on their site before they install anything."

It's trying to run around and do a lot, build a lot. One of the reasons that we built a lot is because my cofounder and I are engineers, and there's always going to be this tendency to use the hammer that you’re comfortable with. So in general I think we tried to solve these problems with engineering. I think there are also equally good solutions, maybe better solutions with marketing and with developer evangelism and going to conferences and going to meet ups and explaining exactly why this is a good idea and why people should get involved. We did some of that as well, obviously, but that is another really powerful way.

One of the kind of fundamental things that I think I learned in doing this was there's 100 different ways to be successful and you only need one, and the one that you picked is usually going to be defined more by your resources than by what it perfect or ideal or optimal. So if you're really good at writing, then the solution is probably going to be the right, and if you're really good at engineering and you can probably find decent solutions that touch on engineering.

Another answer, because we only talked — I know I'm talking a mile a minute, but that was only one side, which was the developer side. On the user side, we really had to lean into the idea that people don't install an app store. They install an app. So we had to make it so that the view pages of apps were landing pages were it did a really good job of explaining what this app was and why it was valuable and took you through the entire flow of previewing the app and letting you pick all of your options and customize it exactly the way that you want it before we asked you to actually install anything.

The original Eager homepage actually was, “Here, copy-paste this embed code into your enter HTML,” and it was a superfund little engineering challenge because we were generating keys on the client and then we were signing them with an HMAC so we could trust that it was really you later, and then you were able to grab that code immediately and just drop into your site without even registering and we would detect that it was a new code we hadn't seen and help you register then. It was all very cool, but it was kind of useless, because that's not what people actually wanted to do. They didn't want to install an app store. What they wanted was, “Oh, I want pace. Progress bar for my website, so people feel like my website is faster.” “Okay, great. Let's show you the preview of pace. Let’s let you get it customized. Let's get it exactly how you want on your live website, and then maybe we'll start talking about dropping an embed code somewhere or installing a plug-in.”

[0:46:47.1] JM: You have this this app store that now has liquidity. It now has adoption under the auspices of CloudFlare, and so I imagine because more of an issue to control that distribution, because you don't want somebody to launch an app on the CloudFlare app store that is a well-disguised piece of malware on their website. You don't want somebody to think they're installing a PayPal button, but it's actually a PayPal button that mines Bitcoin in the background. How do you vet the apps that people are distributing on the CloudFlare app marketplace?

[0:47:34.7] ZB: Yeah, and you're absolutely right that this became much more important post-acquisition, because CloudFlare is a security company and it takes the security of its users very, very, very, very, very seriously, I can tell you.

We become a much bigger target intrinsically when people see that there is more volume. It's not really a lot of fun to put malware on to an app store that is not on that many websites, but all of a sudden when it's available in seven people websites, people want to attack that.

One part of this problem is technical, where; how can we sandbox apps? How can we try to limit the exposure that they have or detect when they're doing something that's wrong? How can we validate statically that this code is not — It's very possible for us to get a general idea of what DOM API something is using, for example.

Then a big part of it is human. One of the things that we did — You asked about the kind of logistics of an acquisition, and one of the funny parts about an acquisition is you have this kind of lull where you know you're probably not going to keep operating as an individual company, but you haven't signed any paperwork yet.

During that lull, what we ended up doing was actually building a app moderation tool, which is a lot like GitHub where you are app moderators get a queue of apps that get assigned to them and they can look at each diff and they can see how the code is changed and they can look at the preview and make sure that it makes sense, and we look at every line of code, and we rebuild any project that wants to distribute, minified or packaged code. We also minify the code ourselves, but if someone wants to obfuscate that code in some way. Then we have to validate that that bill actually starts with the code that we think it starts from.

So it's similar to the Apple App Store, actually, which obviously is an inspiration for us. It's a combination of limiting the exposure through controlling the APIs, statically validating and dynamically validating that the app is doing what it's supposed to be doing, and then having human beings actually look at every line of code that every app is trying to do and look at it in an un-minified, un-obfuscated state.

[0:49:44.0] JM: The inspiration regarding the iOS app store, I'd love to hear more about that. How would you contrast the iOS app store and the android App Store in terms of their governance?

[0:49:56.7] ZB: That's an interesting question. I think there is an obvious answer to this, which is people who use android generally tend to think of things being more free. So the bias is towards everything being available, letting the community decide what it wants, and even letting individuals decide what they want. If you have an android phone and you decide that you want the title bar to be a different color, you have the control to do that. Whereas in iOS, you're meant to respect the bespoke experience that Johnny Ive has decided you should have, and that extends to the app store a little bit, because iOS obviously has a lot more editorial control over what apps show up in whether they're "good". They want every after that you install to meet some baseline level of quality. Whereas on android, they're more content be the security police and let that the market kind of decide what it wants to do.

We had a question in doing this, as you can imagine, of where we want to live on that spectrum. The thing we generally decided is the community should be able to make decisions about what apps are good and what are not assuming two things. One; that the app is secure and safe and it fundamentally does what it claims that it's going to do. If you want to install an app that mines Bitcoin and it's called that, maybe we would think about letting you create a Bitcoin mine app. I don't know if we would actually, but maybe a bad example, but it has to say that it's doing that. So the app absolutely cannot do anything that is not claiming to do.

Then two; to the live preview has to be accurate, and that's one of the most powerful things that we have that's different than these other app stores. We can actually show you what this thing is going to be doing on your site before you install it. So that buys us a lot actually, because it means that you're allowed to say carpe diem a little bit more — Or not carpe diem, sorry. You're allowed to say like a buyer beware and also carpe diem. Sure, why not? But more importantly you're allowed to say, "This person has seen a live preview of this on their actual website. They know what it's doing. They are, therefore, probably even more qualified than we would be to decide whether it's of a high-quality or not. Then, obviously, we also had to build rating and reviews and we also had to build tools to give feedback to app developers, because one of the things that we learned is app developers want to make their apps really good, but if they're not given feedback on what is wrong with them, they just don't do it, right? They don't have anything to do this. They just kind of sit and hope the money will flow, and we hope the money will flow too. Like every building any sort of business, it's a process and a back and forth.

We had to build tools to funnel feedback from app installer and uninstallers in particular back to the app developer so that they could run their own app improvement cycle and turn their app into something that was really good as well.

[0:52:39.9] JM: In the iOS app store, they have this constraint where if you do e-commerce payments, Apple takes like 30% or something of all purchases that they go through the iPhone. It's kind of shameless, but not shameless. It creates an environment that I know if people take it seriously, people take e-commerce very seriously on there, but also cripples the user experience of things like Audible and Spotify. Amazon —

[0:53:13.7] ZB: I think Spotify just charges you more.

[0:53:16.6] JM: Spotify just charges you more. Okay. So you can sign up, it just charges you more on iOS.

[0:53:21.7] ZB: Yeah, I think so.

[0:53:22.6] JM: What about Amazon? You can buy stuff on Amazon and it's just more expensive on the iOS app?

[0:53:27.7] ZB: I'm not the world scholar in iOS, but I believe the way it works is buying physical products is not charged at 30% tax, but buying digital products is.

[0:53:37.6] JM: Fascinating. So when you look at the economics of the iOS app marketplace and you think about the economics of the CloudFlare app marketplace, obviously you have the ability to let people make these apps that are $5 a month for example. If you've got some app that users can buy for $5 a month that gives them no random pictures on their page, it’s like, “Okay. Maybe I want to play for them.”

[0:54:07.2] ZB: That’s a good one. You should build that.

[0:54:09.4] JM: I will build that.

[0:54:11.4] ZB: We actually already have that app.

[0:54:12.7] JM: Okay. All right, that's great. You're way ahead of me.

[0:54:15.6] ZB: There was a point where you could go to meow.voyage. This is, by the way, one of the many — I told you, our marketing strategy as a company was engineering. One of the things that we did was actually register meow.voyage and woof.voyage, W-O-O-F, and it would let you browse the Internet with all the pictures or place with cats or dogs and show you which was more successful, and you could tweet that page with that modification it made.

[0:54:43.0] JM: Awesome.

[0:54:44.5] ZB: Okay, I’m sorry.

[0:54:45.2] JM: No. That's sounds great, I'm sure you had that the core competency to do that after building that the app viewer thing, the I-frame thing.

[0:54:54.4] ZB: Yeah, it’s all the same terminology, just a more ridiculous purpose.

[0:54:56.4] JM: — same technology. Yeah. So how are you thinking about the economics of motivating people to build these apps? Does CloudFlare take a cut from the apps?

[0:55:07.6] ZB: Yeah. CloudFlare does, and the kind of logic behind it is talking to SaSS companies and talking to developers about how they actually make money now, because it turns out that the way these businesses work is you do a bunch of engineering and you build this product and your cost of goods sold, the amount that you'd actually cost you to serve a customer is usually pretty low, because they're just installing something on a website. Software is cheap, because we get computers to run it for us.

The thing that cost a company a lot of money and really determines its economics are user acquisition. How much money am I paying to acquire customer versus what is their lifetime value? So the thing that we've tried to do conceptually with CloudFlare thorax is make that cost of customer acquisition zero, because you don't have to pay for marketing. We are here to help you do marketing and help you get on our blog and help you get on our Twitter account and make sure that your landing page is really great and get a non-product time and do all of these things. Then you don't have to pay for a sales team generally unless you're building some enterprise app that's in multiple thousands of dollars a month, because people are able to find and install this thing.

So when you do that, the economics of building this stuff changes so much that we just started with that same cut that you were talking about, 70- 30, and we didn't know if we were going to be able to make that work, those people were going to be on board, and it isn't a problem, because it turns out these companies are paying a lot more than 30% of their of their revenue to acquire a customer as is.

[0:56:41.0] JM: All right. I'd like to begin to wrap up about discussing the future of this stuff. This is kind of a serverless world that we're moving towards, and CloudFlare is definitely a representative of some of the serverless architectural changes. We were talking a little bit about CloudFlare workers before the show, which is a way to get computation at the edge. I mean CloudFlare is this big system of caches, and with CloudFlare workers, you've got this ability to deploy code that executes at the edge on those caches. So for example you could have a system where you route specific versions of a page from a CDN to specific types of users, and you have this programmable logic that will just be deployed to the edge, but not in a way where you're going to have to manage these servers.

What's the overlap between the serverless vision that comes with the CloudFlare worker's side of things and the CloudFlare apps' side of things that you're working on?

[0:57:55.1] ZB: Yeah, and that's a great question. I think serverless is exactly the right word, and I'll kind of attack it from both sides, which is CloudFlare service workers are exactly what you described. It's taking a service worker that you could write and run in the browser and instead running it on our edge. So that means you can intercept any request, intercept any response, change the response, change the headers. You can make sub requests to other resources that'll come through our cache. Those data files are HTML files will be in the cache everywhere, and make literally any change to a request or response that he could possibly imagine all written in JavaScript.

These service workers are run on our edge 180 places around the world on every request. Whether you get one request a second or you get a million request a second. That's incredibly powerful, and even that I think is kind of underselling it though, because the truth is right now if you want to deploy something on AWS, you have to think about availability zones and you have to think about regions and you have to decide, "Do I want to take the time to spin this up in Australia or not? I think that's kind of an outdated model. I don't think people want to have to worry about where their code runs. I think everyone would kind of rather their code runs as close to the users as possible.

My personal vision is that people build entire applications in these service workers and they just can trust that these applications are going to run as close to the user as possible and we make the economics cheap enough that that make sense, that you obviously are going to pick to run something in a CloudFlare service worker, because it's going be faster. It's going to be closer by the speed of light distance to your user.

On the other side, you get apps, and the kind of mission of apps is to make all of CloudFlare platform. So adding JavaScript and CSS to a page is great, but right now we're delivering the ability to add DNS records to page, so you could add a subdomain for someone. Then once we integrate workers, which is something that I think is going to happen at the beginning of next year, so it's something that we're starting to talk to interested app creators about right now, you're going to be able to install these apps to make any kind of modification to the user's page that you could imagine.

So A-B testing. You want to deliver different variants to the page, change the HTML based on which test this person is in. All of a sudden you can do that on the edge without any sort of sub-request. That super positive. That's super exciting. If you want to do that for SEO, you have to modify the HTML. It's the only way to do it. All of a sudden you can build that. Literally, any type of app that you can imagine that wants to modify a page or add something to a page or modify a request, security. Maybe you want to block certain requests that look like bots. Okay, you can build that as long as you can write it in JavaScript. Under a megabyte of code or some ridiculous limit, you can run that on our edge and you can package it as an app and sell it to people or give it away for free.

[1:00:54.8] JM: All right. Zack, it's been great talking to you. I'm really fascinated to see where CloudFlare goes with all the new developments, and I wish you the best of luck.

[1:01:04.9] ZB: Great. Thank you so much. I really appreciated this.

[1:01:06.3] JM: Okay.

[1:01:06.9] ZB: This is a great experience.

[1:01:07.5] JM: Wonderful.

[1:01:07.7] ZB: Thank you.

[END OF INTERVIEW]

[1:01:10.9] JM: Every second your cloud servers are running, they are costing you money. Stop paying for idle cloud instances and VM's. Control the cost of your cloud with ParkMyCloud. ParkMyCloud automatically turns off cloud resources when you don't need them. Whether you're on AWS, Azure or Google Cloud, it's easy to start saving money with ParkMyCloud. You sign up for ParkMyCloud, you connect to your cloud provider and ParkMyCloud gives you a dashboard of all your resources including their costs.

From the dashboard, you can automatically schedule when your different cloud instances get turned on or off saving you 65% or more. Additionally, you can manage databases, auto scaling groups and you can set up logical groups of servers to turn off during nights and weekends when you don't need them, and you could see how much money you are saving.

Go to ParkMyCloud.com/sedaily to get $100 in free credit for ParkMyCloud for SEdaily listeners. ParkMyCloud is used by corporations like McDonald's, Capital One and Fox and it saves customers tens of thousands of dollars every month. Go to parkmycloud.com/sedaily and cut the cost of your cloud today. That parkmycloud.com/sedaily.

[END]