EPISODE 903

[INTRODUCTION]

[00:00:00] JM: The browser has become the central application of the consumer operating system. Every piece of client software, from email, to document management has become usable through the browser. Even modern desktop softwares such as Slack is built using Electron, a tool for building client applications that essentially run inside of a browser window that does not have an address bar. One activity that still takes place largely outside of the browser is the process of writing and deploying code.

A developer often uses an IDE, such as eclipse to write their code. Then switches over to a terminal where they can build and deploy their code to a remote server running in the cloud. For a developer who has been writing code for a longtime, this process of disconnected tools feels completely intuitive. But for a new developer, it can be very disorienting.

New developers sometimes have trouble understanding the difference between a local and a remote environment, or how to use repository management software like Git. And this is all in addition to the other problems that a new developer might be dealing with, such as language installation, syntax and package management.

Repl.it is a browser-based coding environment, a computing engine and a collaborative workspace. Repl.it has found significant traction among new programmers who begin their programming journey within Repl.it and then stay in the environment even as their applications become more richly featured and complicated. Repl.it is an amazing piece of software, and the story behind it is incredible.

Amjad Masad had the idea for Repl.it many years before he started the company, but he needed to first build up the money and the confidence in order to go after this business with full force. Amjad joins the show to talk about his long journey towards building Repl.it and to discuss the thriving Repl.it platform in its current form.

This show is a real pleasure. I enjoyed talked to Amjad quite a bit, and I hope you enjoy it too.

[SPONSOR MESSAGE]

[00:02:16] MB: Continuous integration allows teams to move faster. TeamCity is the continuous integration and delivery server developed by JetBrains. I've always loved the IDEs from JetBrains. I've used WebStorm, and RubyMine, and IntelliJ, and once you start using any of the JetBrains products, you realize that this is a company that knows how to build products for developers.

TeamCity gives you continuous integration and delivery designed by JetBrains. For most teams, TeamCity is completely free as long as three build agents is enough for your project. For larger organizations, there is TeamCity Enterprise, and listeners of Software Engineering Daily can get TeamCity Enterprise with a 50% discount by going to teamcity.com/sedaily. That's T-E-A-Mcity.com/sedaily.

TeamCity supports most popular programming build tools and test automation systems, version control systems and cloud platforms. Whatever the size of your organization is, check out teamcity.com/sedaily and get started with continuous delivery. Thank you to JetBrains for being a sponsor of Software Engineering Daily. And if you want to try out JetBrains TeamCity, go to teamcity.com/sedaily.

[INTERVIEW CONTINUED]

[00:03:45] JM: Amjad Masad, welcome to Software Engineering Daily.

[00:03:47] AM: Thank you. I'm glad to be here.

[00:03:49] JM: You used to work at Facebook. Describe the tooling experience of a developer at Facebook. What do you have access to there?

[00:03:58] AM: So, one of the things that you sort of struck by when you start at Facebook is how fast they give your development environment, right? At least when I was there, there's an online IDE as well that gets you started pretty quickly. So from day one in boot camp, you're sort

of in the code, you're debugging it, you're coding, you're maybe committing and sending diffs, their version of pull requests. Overtime, that experience gets better and better.

I think one of the things that was really frustrating is I wanted to mobile there. I was really excited about Android. And part of the reason why I joined Facebook is I was excited about their interent.org sort of thing that they were doing where they're trying to bring internet to the rest of the world. And Android was a big part of that, and like the growth in 2012, 2013 of devices and I thought, "Okay. We're going to get a lot more people online. We're going to get a lot more people, computers, and this is going to be exciting." So I wanted to do some mobile dev there. And that was really frustrating. I changed one line of code and it takes 15 minutes to recompile.

That actually what got me interested in what became React Native at the time when I was at Facebook. But, yeah, basically things were really great in terms of getting people started really quickly. The website was kind of frustrating. The reload will take anywhere between 30 to like 1 minute to reload the page after you changed something.

So I started working on a lot of tools to make things faster. I'm just like a tools person. I'm always frustrated by the experience of programming, because I really like interactive programming. I really like kind of programing being more of a conversation with a computer rather than sort of a thing that I think about and then type some code out.

So I was frustrated with the JavaScript development and experience there, and started building hot reloading tools to make my job easier. Then later on I joined the React and React Native team and started working on those things fulltime.

[00:06:02] JM: With React Native, was there a kind of political struggle to get Apple to be okay with React Native being a thing? Because that added so much dynamism to this – I mean, speaking of a slow process, it is slow to recompile a mobile application. It's also much slower to deploy a mobile application.

[00:06:24] AM: Yeah. I mean, we worried about it. I think before we released, they changed their terms of service to allow – They changed something about their terms of service that made

React Native sort of not infringed on Apple service. I'm trying to remember which line was that, but something about you can ship over their JavaScript, as long as you're executing within like their JavaScript engine, JavaScript core, then you're okay with updating your application on-the-fly. I think something like that.

So there as a little bit of worry, but by the time we launched, I think there was a lot less worry about that. Obviously, a lot of people that were working in ReactNative were motivated by the idea of like how do we bring the web dev experience that we all love and used to to mobile. You see Facebook was one of the pioneers of iterative development and continuous integration and all these stuff that they worked on. When they become a mobile-first company, they just slowed down a whole lot, because they went back to this like shrink wrap software way of doing things. So there was a big push inside the company to actually change that.

I'm not sure if you had Jordan Walke on the podcast yet. Yeah, a lot of what motivates him is that. It's the idea of like more freedom in software development and things like that. He's the inventor of React and React Native and Reason, all those stuff. Definitely, there was some kind of ideological struggle and ideological drive behind these tools.

[00:08:01] JM: I think mobile and cloud are probably the two biggest shifts to the world of software development over the last decade or so. And obviously mobile has its frictions and a lot of tooling has gone into making the mobile experience better. It has gotten a lot better. Cloud, most people perceive as actually being a fairly decent experience. It's not perfect.

I was reading some of the stuff you've written, and my sense is that you think that there is a lack of understanding or a lack of usability for cloud tools or cloud products among a large set of developers in the world. Can you characterize more what you think the shortcomings of cloud usability are?

[00:08:51] AM: I've written a lot about different aspects of the programming experience. In some cases, getting worse overtime. So if you go back to the early PC, you could see computers booting up to either if it's IBM or a Microsoft-based computer. It boots up to MS-DOS. If it's something like Commodore, it boots up to the basic interpreter, and that actually invites the user to program the device.

With mobile and with cloud, the user is so far away from the programming experience. Actually, if you read a lot of the early literature that programmer wrote, computer scientists, they actually didn't have that big of a distinction between the user programmer. Sometimes they're hyphenated user-programmer. Right now when we say user, we usually mean the end user that doesn't know how to programmer.

One undeniable thing is that we've gone a lot more sophisticated in our software. So, I think one thing – One sort of informer rule that I observed about this world is like the more advanced the tools we're programming, the worst it is to the beginner experience. So, there's an inverse relationship between sophistication of our tools as programmers, end user programming, hobbyist programming, introduction to programming, people trying to get into software engineering.

You could see that in the, I think, more – I think the biggest example of this has been the JavaScript web development world, right? Where 10 years ago or even more, you opened Notepad, you save the file as index.html. You drop in a script there. You opened another Notepad with a JavaScript and you started coding JavaScript. Literally, that's all you need. You just click on the file and it will open in Netscape or Internet Explorer, then you're on your way.

Compare that with setting up React, or Angular, or Ember, or any of that stuff. That's something that's frustrated me a lot. One of the things that I'm really proud of at our work at Facebook was I was responsible for the React Native JavaScript development experience. Meaning, the React Native Packager, the React Native command line experience and all that. One thing we always kind of thought about is TTH, time to Hello World. How much time does it take for someone to get a Hello World on the screen, and we try to always get it under 5 minutes, as much for the user as possible to get it under 5 minutes.

I remember, one of the things that was really cool when we released React Native was that we had this three line bash code or shell code that we could copy and it will initiate a new application and open the emulator, the Hello World program, and people tweeted them out. They were so sort of enamored by how easy it is versus setting up X-code or any of these big things. Yeah, this is something that I've thought a lot about.

Now, when it comes to the cloud, it's that what is the boot to REPL for the cloud? What is the – In the PC, we had boot to basic using Commodore or boot to MS-DOS. What is the boot to REPL for the cloud? So, I haven't seen any, right? If we're saying the cloud is the next computing paradigm, it's where most of our computing is happening. It's getting cheaper and cheaper. Remote computing is getting better and better with Google Stadia and things like that. Even things that we always thought to be as close to the user as possible, like games, are now going to remote, going to the cloud.

So the cloud is one of those computing paradigm shifts. But how can we get to be really accessible, get it to like, "Okay, I'm on the cloud now. I'm evaluating code in the cloud. I'm writing my application really quickly." So that's one of the things that I thought a lot about, and in some ways, Repl.it is that, is boot to repel in the cloud. You go to a Python REPL.

So, open your browser, REPL.it/L/python. You get a Python REPL, and you can do Hello World in two seconds or you can start a server. You can start a Flask server and it'll detect that. It'll launch the web app for you and now you just deploy the Flask app in a few seconds. So that's kind of the philosophy of where coming towards those.

[00:13:21] JM: But if all I'm doing is Hello World, why do I need the cloud? I can do Hello World on my local machine.

[00:13:27] AM: Yeah, it's a good question. I don't think that local computing is going to be defunct anytime soon. But the cloud is certainly one of those – We went from mainframes, to microcomputers, to PCs, to mobile devices, and it seems like the cloud is another one of those shifts where a lot of the computing is happening there. So, you need a way for people to start programming the cloud in a very easy way and the same way that we were able to program microcomputers.

[SPONSOR MESSAGE]

[00:14:08] JM: When I'm building a new product, G2i is the company that I call on to help me find a developer who can build the first version of my product. G2i is a hiring platform run by

engineers that matches you with React, React Native, GraphQL and mobile engineers who you can trust. Whether you are a new company building your first product, like me, or an established company that wants additional engineering help, G2i has the talent that you need to accomplish your goals.

Go to softwareengineeringdaily.com/g2i to learn more about what G2i has to offer. We've also done several shows with the people who run G2i, Gabe Greenberg, and the rest of his team. These are engineers who know about the React ecosystem, about the mobile ecosystem, about GraphQL, React Native. They know their stuff and they run a great organization.

In my personal experience, G2i has linked me up with experienced engineers that can fit my budget, and the G2i staff are friendly and easy to work with. They know how product development works. They can help you find the perfect engineer for your stack, and you can go to softwareengineeringdaily.com/g2i to learn more about G2i.

Thank you to G2i for being a great supporter of Software Engineering Daily both as listeners and also as people who have contributed code that have helped me out in my projects. So if you want to get some additional help for your engineering projects, go to softwareengineeringdaily

[INTERVIEW CONTINUED]

[00:15:56] JM: I definitely agree with that. I guess what I'm taking issue with is whether or not that point needs to occur at Hello World. When I'm in my Hello World phase of programming, to some extent, I don't even want to think about servers and like remote stuff and like HTTP something. I'm just overwhelmed by like what's the syntax.

[00:16:23] AM: Yeah. You're sort of agnostic to that, and especially on Repl.it, you don't know where you're executing a code. A lot of people think they're doing it locally. So, it doesn't matter. You're right. But one other principle that we have is another thing that we've lost is the incremental approach to learning programming.

So, it used to be that maybe you've edited your MySpace page and then you've opened your HTML letter and you created your first website. You sort of accidentally became a programmer.

Right now, it's really hard to be accidentally a programmer. You have to go to a boot camp or a computer science university or something like that. But you need that sort of continuum. So maybe you've written Hello World and then you wrote your first programmer.

But then maybe you want to save things to a file and have that file be downloadable or share that with your friends. Maybe you want to start a server or maybe the first thing that after you learned the basic programming, you Google like how to make a website in Python. Then you go to a Stack Overflow link. Inside the Stack Overflow link is like a piece of code for a simple HTTP server in Python. You copy that. You should be able to paste it on Repl.it and it should just work.

So we make that work actually, and the way we make that work is that, say, it imports Flask. We have a package manager we call Universal Package Manager. It's an abstraction on top of all the different package managers for different languages. One of its more instinct features is what we call bear imports. So bear imports basically analyzes your code. Looks at what imports you have, and then installs them for you.

If you're someone who's just getting started with programming and you copied a piece of code from the internet and you want to do a specific thing, like start a server, or start a website, start a developer website. It should just work. So I do agree with you that you don't have to think about all that stuff. You can delay the complexity as much as possible, and we're totally onboard with that.

Now, the question is like why not do it locally then put it on the cloud. I think that incremental approach to learning makes it easy if it's all in one place and you're like coding against the same runtime, same environment and that doesn't change under your feet.

[00:18:48] JM: Yeah. The friction I agree with you on is the point at which – If I decide I want to standup a Python webserver, I can go to Stack Overflow and I can learn about how to set up that server locally. That's never been an issue for me in my recollection. Where it gets tough is like when you want to deploy it. When you actually want to deploy that server and make it accessible to people to go to www.jeffswebsite.com. Then you have to start looking, like, "What is the hosting thing? Is there a firewall rule thing?"

There is where I see the idea that if you've actually started in the cloud, then whatever cloud system you're on, can support you easily making that a universally accessible website, rather than a locally accessible website.

[00:19:44] AM: But I would actually – The first part of what you said here is that it's always been easy for you to send up a local webserver. I don't think that's true for everyone, right? So, think about everything that you need to say do a Flask, which is the most popular Python web framework, a Flask server locally.

So, think about everything that you need to say do a Flask, which is the most popular Python web framework, a Flak server, locally. Maybe you Google that. Maybe you get that code snippet, but then you need to know how to start in your project, how to create directory CD into that. You know, in the project. And then you need to learn how to use Pip. You probably don't have Pip installed.

Then you need to go figure out how to install Pip. Then after you install Pip – Well, there are some issues there. Maybe install Pip 2 instead of Pip 3, and you're trying to do a Python 3 project versus a Python 3 project. So, there're a lot of issues. Then you get into virtual env and you're getting into all these complexity. Maybe we can blame Python for that complexity. But every language has its complexity. A lot of people get stuck and a lot of people just don't know where to go after that.

In my opinion, for programing to hook people in, they need to have that feeling of programming. What it feels like to be programming. A lot of people think what programming is, is IT, is trouble shooting. It's installing and uninstalling things and linking and unlinking thing and doing all that stuff. So, I think abstracting that away early on in the process is important.

Now, the usual criticism of this is that, "Okay, you're being too magical. You're hiding that way. People are not learning properly." Actually, invariably, we see people that start with Repl.it go locally and know how to set up development environment and actually learn all of these stuff. So it's not an either or situation, but I think especially early on, you'd want to get people as fast as possible to see a result.

But to your second point about deployment, to see result and share it with someone, right? I want to be able to share it with – A lot of people, their first user is their grandma or their friends. So you want to just share a website with them and should be able to render. We actually go beyond that on the sharing side.

If you've written a command line application, just Hello World, or what's your name, hello Jeff. We actually give you a shareable link just for the terminal output. So you can give that to your friend and they can just see that. So, another thing that we try to do is like the first experience of getting a user should be really fast. You're 9-years-old. You're building a small program. You should be able to get your first user to like play with your small, let's say, texts story game or something like that. On the deployment side, we make sharing a lot easier. We make sharing web apps and command line applications a lot easier.

[00:22:36] JM: Yeah. I mean, I'm with you that everything moves to the cloud for one reason or another, and it actually doesn't even matter what I believe, because you just look at what Google is doing. They wouldn't be investing so much in Chromebooks if this was not the future. Everything is going to the cloud. It's dumb client devices for the large part. I mean, unless you're like a car, right? Then you actually have some like real significant tradeoffs to make and decisions to make.

I'm sure this shift always goes back and forth. But like right now, you look at what the Chromebook is doing. You look at what's going on with WebAssembly. Clearly, we're pushing a lot of functionality into its – You've got a client that's kind of doing a lot of stateless processing, and then all of the stateful stuff is happening in the cloud, because why would you want to have two phase, three phase commit involving like your client device in a way where you might lose – Just like push everything to the cloud.

So, it becomes a question of like, "I wonder where we're at in the movement towards that complete cloud development experience." Specifically, what are the shortcomings or what's the division of labor today between the client device and the cloud. What developments need to occur to get that closer to the ideal? Whatever ideal you envision? Whatever is the division of labor that you envision?

[00:24:05] AM: Are we talking about computing I general or development?

[00:24:07] JM: Computing in general.

[00:24:09] AM: Yeah. So o the extreme side of decorating, you have Apple and iPhone and its very sort of edge-oriented computing. It's a very client-heavy computing. The likes Google is on the other side, where it's like very cloud computing. I think a lot of the world, a lot more of hat were seeing is going towards cloud. If you think about the next 2 billion people coming online, they're probably going to use cheap computers that will not enable them to do all the computing that they want to do locally. But then the question is like, "Are they actually getting good internet?" And then there are few things we can talk about there, like the Satellite Internet by Amazon or by SpaceX. That might happen very soon in the next couple of years.

So, if you think about these shifts, very cheap mobile devices and client computing and high-speed internet, cheap high-speed internet all over the world. Then cheap computing, like obviously all the big clouds are just like competing, computing prices, like lower, lower every year. If you look at this three shift, I'd be betting that like we're going to see more and more. Maybe cloud might not be the best thing here, but remote computing. I think there are opportunities where you can see some kind of like mesh networking, remote computing thing happen.

For example, if you have a very powerful desktop at home, maybe you could like plug it into a network and it becomes that –

[00:25:43] JM: We've heard that story before.

[00:25:45] AM: Yeah.

[00:25:47] JM: Eventually it will happen. Some involving Ethereum or something.

[00:25:50] AM: Yeah. No. I'm not going that way. But, I mean, they –

[00:25:55] JM: But eventually, why not?

[00:25:56] AM: Why not? Yeah. Yeah. Why not? You could see it trending that direction. But, with regards to development, what does it take to bring it online and what are the critiques there. Obviously, the biggest critique is that people are used to their tools, right? They're used to Emacs, then VSCode. You have like complete control over your machine. If you wreck it, it's your problem. You're going to fix it.

On the cloud, you're basically renting out a computer from someone on the – It's someone else's computer basically, and a lot of time cloud development environments actually don't have as many tools and options. It's more narrow. But I think a lot of these things are just because it's really early in that direction. I think a lot of these things could be solved. But I'm actually not against local development as well. It's not like something. It's not like we at Repl.it are pushing cloud evolvement as the end and be all of everything. Who knows? We might also be releasing like a local editor sometime in the future. We don't take like a very strong stance on that.

The strongest stance we take actually is interactive programming. We think that things, like I said earlier, is that programming is a conversation between the human and the machine.

Seymour Papert, one of the grandfathers of MIT Media Lab that does all these fantastic computing experiments, and now they're on Scratch. I think the world's most popular programming environment for kids. It has millions of kids on it. He said once that computers are match speaking machines, and that for kids to learn math, they can just talk to the computer.

I thought this was like a really good metaphor for what programming is. It's sort of like a conversation with this highly-logical entity that's kind of executing your commands. But it needs to be a little bit more interactive, because we make mistakes, because we don't have fully-formed thoughts as humans. So, it needs to be this back and forth experience with the machine.

So, this is one strong sense we take. Speed as to which people start is another strong sense we take. Incremental or adaptive IDEs. Just start at simple. Start at very easy, and then expand and show their power and allow you to do more interesting things as you get advanced in your career. That's another strong sense that we take. But as far as the cloud, that's where my

thinking is right now, but I'm open to the idea that things might change or that future might not come to fruition, etc.

[SPONSOR MESSAGE]

[00:29:01] JM: At the beginning of 2019, we had problems with Software Daily, which is our custom-built website and mobile app set. The website was not engineered properly, and our iOS app was buggy. Everything needed a redesign. To help us refactor our cross-platform application, we brought in Altology. Altology a full stack software engineering firm that helps innovators build worthwhile products.

Altology will help you get your project or your company to where you want them to be. They can rescue your project. They can augment your team. They can help you get a new version of your product out the door. If you're building a brand-new product from scratch, Altology can also design and develop web and mobile products that are brand-new. The Altology team is entrepreneurial. They're design-focused, and they're able to work across the stack. To get help with your engineering projects, check out Altology today by going to all altology.com. That's altology.com.

Thank you to the Altology team for helping us get Software Daily to where it is today, and for being continued friends of the show. If you need help with your application, check out altology.com.

[INTERVIEW CONTINUED]

[00:30:28] JM: Let's get into the engineering behind Repl.it. So, for people who don't know, it's a system where you can just open up Repl.it and it's like you have a terminal in the cloud. There have been a number of efforts at this kind of thing over the years. Is there some specific engineering breakthrough that happened more recently that has made it easier to build Repl.it, or did you do anything marketly different? Tell me about the engineering behind it and kind of the why now question. Because this thing has been tried. You've got Cloud9 and a bunch of these other things that have been tried over the years. Why has it gotten better today?

[00:31:16] AM: There are a number of things that made it better today. Let me answer this question. So, actually, the technology behind Repl.it, the original idea has been around for a longtime. So, 2009, 2010, I was going to a university back in Jordan where I'm from, and one of my frustration has been every time I go to computer science class, someone with no laptop. I couldn't afford one.

So I would go to class, and every time before we start coding, we have to set up the development environment for the different languages we're using. That's very frustrating, because you're in the computer lab. You might not have the necessary software, or it might have the software, but it's configured differently or the versions are wrong. So you could see the professor or the assistant kind of like going around and trying to fix all that development environments.

So, all these frustrations let me kind of dream about the world where I can ctrl+t and open a new tab and start coding. So that was original idea behind it. And I started working on it. At the time, 2009, Chrome had just come out like a year or two earlier. V8 had just come out. JavaScript was getting faster. It was obvious to me that the browser is going to become the primary application development or application distribution runtime environment.

So I was like, "Okay. Now we have docs and like the whole Google Suite online. Probably a lot of the things that we do locally will probably move to the browser as well, that programming should be one of them." I looked around and there wasn't a lot out there. I'm not sure if I'm mixing my timeline, but I think the only thing that I saw, the only experiment that I saw was Mozilla Bespin, it was called, which became later on Ace, which was the editor that was powering Cloud9.

But there wasn't really a lot of people. There was a few JavaScript REPLs where you can execute JavaScript like JS Bin style, things like that. There wasn't a REPL that could run any language. There wasn't a REPL that you can both like edit and like do interactive programming in. So I wanted to do that.

My friends and I were setting SICP, Structure and Interpretation of Computer Programs, which is a classic MIT introduction to programming course. That was originally – The language for it was Scheme. Right now I think they switched to Python, but it was Scheme as the introduction and there's a lovely YouTube series on it where the original authors just give the lectures and it's one of the best computer science courses out there. It's elementary, but they do it in such an elegant way.

So, when my friends and I were setting it outside of school. It wasn't a school program or anything like that. We just decided to do it, because we loved it. I put up a Scheme REPL for my friend. So I wrote a Scheme interpreter and put it up and we started using it, my friends and I, and we're like, "Okay. This is really cool." We really loved this experience. So, "Okay. Maybe this idea has legs. Okay. Let's do Python afterwards."

So we started writing a Python interpreter in JavaScript, because naturally that's how you want to do it. Two, three months in and it's like, "Okay. This is going to take 10 years. This project – Just to like get all the languages in."

At the time, Mozilla had mscripton that they were working on, a researcher was working on. Then I think they had like a rudimentary version of Python compiled to JavaScript. So we thought, "Okay. This is interesting. Let's try this out." We tried it out and evaluated basic Python. But then there was a lot of things missing for it. So we started contributing to the project.

I think we implemented the original kind of POSICs emulation layer, like the file system and things like that for mscripton. Then we got Ruby. We got Python. We got Lua. We got a few other languages then. So we released the first, basically, production-ready mscripton app. The first production mscripton app was Repl.it.

[00:35:21] JM: By the way, this was like long before you were even at Facebook, right? So you've been thinking about this for a long time.

[00:35:27] AM: Yeah, this was like literally 10 years ago. So, we released JS REPL, was the thing that released the open source, the open source environment for building JavaScript REPLs for any language. Then Repl.it was basically the demo application on top of that.

2011, we had the same domain name and we put up the application. It was basically a list of languages. You select one language and you just like – There's an editor. There's a console. That's it. And there's a big play button, and you could run that thing. And we released it. It was number in Hacker News for a few days. Then later on was really surreal what happened.

[00:36:03] JM: Wait. That was 2011?

[00:36:04] AM: That's 2011.

[00:36:05] JM: What years were you at Facebook again?

[00:36:06] AM: 2013 to 2016.

[00:36:09] JM: Oh my God! Okay.

[00:36:10] AM: Yeah. 2011, released this, top of Hacker News. We started getting all the companies interested in what we're doing. We're like two or three kids from Jordan. I recruited some of my friends. My now wife did the logo design for us. And Udacity started using it. We got an email from Peter Norvig saying like, "Hey, we're really interested in what you're doing. Can we use it for Udacity?"

A bunch of other companies started using a lot of online coding schools. Some are still alive. Some are dead. Then Codeacademy started using it. Codeacademy had launched in 2011 and they used a lot of the infrastructure that we've built to build their website.

[00:36:51] JM: Oh my God! So that's how you wound up at Codeacademy.

[00:36:53] AM: Yup.

[00:36:55] JM: Oh my God!

[00:36:56] AM: So the founder CEO, Zach Sims, wanted to hire me and I'm like, "No. No. I want to make this a startup. I want to build this." I resisted for a while and I tried to raise money on Jordan, but there was like a handful of VCs and they're like, "Look, son. We only fund copies of American companies. We let globalism do our work. Why would we innovate?" Which is an irrational thing to do, I think. It's gone a lot better now.

So I reluctantly kind of agreed to start talking to the Codeacdemy folks and did some consulting for them and worked on more languages for Codeacademy. Then founder CEO came to Jordan, flew to Jordan to recruit me, and we went to the Dead Sea and we hang out for a while, and afterwards I kept saying no and he kept giving me a bigger offer. Then I was taking him back to the airport, because I just said no. Then he gave me a bigger offer at the airport, and we literally in the car signed the offer.

[00:38:01] JM: Do you think if you would have just done Repl.it back then it would have worked?

[00:38:06] AM: Yeah.

[00:38:06] JM: Wow! That is hilarious.

[00:38:10] AM: I think I should have just like – Yahoo had acquired a Jordanian company. The first Arabic email company called Maktoob. So I was working at Yahoo. That was my day job after graduating, and hated it. Yahoo was like the beginning of the decline, 2011, and it was just awful. They treated engineers really badly, because they were thinking, "Oh, we're a media company. Engineering is a cost that we shouldn't have."

So I really hated my job and I would go home and work on this at night and weekends and things like that. That's the sort of the path and life where like you just go zigzag and go back to the original idea. It took me basically from launch to the starting the company five years, and had abandoned the project like after I worked at Codeacademy and then Facebook. I abandoned the project for five years.

Then the person who took the initiative to revive the project was actually my wife, Haya, which is my cofounder. Now, designer, Repl.it's designer. She's like work on side projects all the time, and she was struggling to find a job in design at the time, I think, New York. We're in New York. And she wanted more things for portfolio, and we thought it'd be fun to work on her project. She's like, "Whatever happened to that like Repl.it project that we worked on?"

So, I opened up Google Analytics and sure enough there was like, 5,000, 6,000 monthly active users. I'm like, "Okay. It's really buggy and we haven't updated it in a really longtime. Let's see what happens." So I write – What became our infrastructure, we call goval now. It's written in Go. It's an evaluation service, like goval. So I built this Docker-based container orchestration and evaluation system sort of pre- Kubernetes. I probably would have used Kubernetes then. And released it and we added more languages and we saw users grow. Then we added user accounts and saving your projects. But that didn't exist. We saw more users.

So, by the time my role at Facebook was kind of naturally coming to an end, I had started the JavaScript infrastructure team, which maintained Babel and Jest and a lot of these tools. They wanted to just wind that down to team, and I thought, "Okay. Maybe we could finally make this thing a company." It was really growing to the point that it was like causing me a big chunk of my salary just to keep it up.

So I quit my job in April 2016 and went fulltime on this. We're able to raise a round of funding, or on the same time, not long afterwards, from Bloomberg Beta.

[00:40:56] JM: I just want to pause there and say, "I don't think it's that. It's neither common nor uncommon for people to have these projects that they tinker with for like a decade, and like think about for a decade until they kind of –" It's like some combination of the project reaching maturity and just like looking around in the market and saying to yourself enough times, "Why isn't anybody doing this?" Not to insert personal anecdote.

But before I started this podcast, I was so sure somebody else would do it. I was like, "Why even try? Somebody else is just going to do it." Which is it's such a weird internal – I don't know if you were playing those internal psychological games, but you sort of just like assume you're going to get scooped, or you assumed there's some reason not to do it. You assumed like the

efficient market is going to take care of it or it should have already taken care of it by now. Therefore, it means that it doesn't work, right?

It's just weird how the psychology sometimes keeps you from doing what you should have done. Until 10 years later, you finally have the – Whatever. Self-motivation or self-determination, confidence, whatever you want to call it to actually go through it.

[00:42:10] AM: Also money. That's the other thing.

[00:42:13] JM: Okay. Yeah, money.

[00:42:14] AM: Yeah. For most of my life I didn't have much money. So, in 2016 I had like some Facebook stocks that vested. I had some savings to stand up on my feet. Put in like 20k, 30K in the company before we raised money. I wouldn't been able to do that, to take that risk without some kind of – Even leaving Codeacademy I was broke. I had startup stocks, but actually exercising those and paying the taxes on them was a huge burden.

When we were doing the Facebook onboarding, I have to fly from New York to California. I didn't have money to pay for the ticket. They reimbursed it afterwards, but I didn't have money to pay for the ticket. What I did was I put out our home in New York, our apartment up on Airbnb and got the money for Airbnb to fly over to California. So, money was a big aspect in that.

But I do agree that also self-doubt and the idea that there're so many people in the world, 7 billion people. On the internet, there's like 2 or 3 billion people. But how come no one is making software engineering, really in-depth software engineering podcasts? It's like the first time I met you, I think I said that. I love this podcast. I listen to it. But when I was looking for one, there wasn't much. You kind of like go around different podcasts and nothing is really interesting. But this one was really interesting. I still think there should be more.

[00:43:56] JM: I totally agree. I totally agree.

[00:43:59] AM: So it's weird how that works. I don't know the market is really efficient, actually. But, yeah, it took a longtime. I wish there were more venture in Jordan. I wish there were more VCs taking changes no people on different parts of the world.

[00:44:16] JM: I think that Pioneer thing is pretty cool.

[00:44:18] AM: Yeah, Pioneer is really cool. There's like a lot more work now. YC is going global. So that's really cool. A lot of people are comfortable funding people online and without meeting them in person. So that's getting better. So I'm really excited about that future.

[00:44:32] JM: This is like pretty much a proof that the efficient market hypothesis doesn't work. We don't have the money going to the entrepreneurial people. Man! We've really ran up a lot of time here talking about random stuff. I guess we should talk some about – A little bit more about engineering. So now that you've been – I guess one thing I want to ask is like of all these languages, for building a web IDE, is there one that was like particularly hard? I don't know, Java or something, to actually virtualize in the cloud, or are they all – Actually, no. Here's a better question. This is a more succinct question. When I spin up a REPL, when I go to REPL, what is happening on the backend?

[00:45:12] AM: Yup, that's a great question. So, the first thing that we do is we – The webserver serves you the JavaScript and the application, the editor, everything, and then gives you a token and then you connect to an external server, which is eval.replit, and that server is where the magic happens basically. So the first thing you connect to is a service that we call conman, container manager. This is kind of like what it says. It will check if the project has an existing container. If it has an existing container, it routes you to that container. If you're connecting to a multiplayer session, our version of collaboration, it connects you – It also finds that container somewhere in our cluster. It connects you to that container.

But if it's a new project, we will spin up a new container for you. Inside that container, we put in a program called pitone, which is our version of the init process. So that sort of ZooDoo edit process. Then basically your browser is talking directly to that, to that server. Then we have a protocol that resembles SSH. So you open channels and you have – For every servers that you want, you sort of do some kind of handshake and open a bidirectional channel.

So in Repl.it we have different services. So the service is evaluating code. So that's one channel. When that channel starts, what we do is we open a process for the REPL. So, for interactive languages, we actually spin up the actual REPL for Python, Node, whatever. We have actually an open source project called Prybar. We call it Universal REPL Interface. It basically creates the same interface for all the languages that have interactive programming environment.

So Ruby, Python, JavaScript, they all behave the same way. So there're papers over that complexity, and this is a reoccurring pattern that you'll see in our infrastructure, is like it's all about papering over language complexities. So it starts that Prybar. Attaches a PTY and then streams it back to the – We use Xterm.js. So now you're actually in the REPL, in the actual REPL in the code.

Another service is files. So, when you're typing, you're actually writing to the file. So we open that file protocol, and we're sending operational transform events. Because Repl.it is going to a direction where it's collaborative at heart. So operation transform is the protocol that was invented by Google when they were doing Google Wave to do collaboration, and that's the standard for text collaboration. So we use that as the primitive for writing files. Actually, we just sent OT commands back and forth.

Say, you want to do a package installation. Whether that magically bear imports, package installation, or we have on the sidebar the packager, where you can search for packages and solve packages. That's some other service that we start. That's service, actually, we're about to open source maybe by the time this episode is out. We will have it open. A new project called UPM, Universal Package Manager.

Universal Package Manager, same as Prybar, wants the paper over the different complexities and workflows of different languages on how to do package management. So there are slight differences between NPM and bundler. There are slight differences between a Pip and NPM. So, we want to make it so that you're using the same command line interface and can install packages in any language you'd like and has that feature of guessing what kind of imports

you're trying to import. So that's some of the service that we start. So you could see all these services that could start.

Another one is language server protocol. So when you're in Python, or Node or whatever, we actually have really autocomplete, and that's because we start an LSB server and that goes through the same protocol. LSB is the language server protocol. The thing that Microsoft invented to make it easy for IDEs to get really good intel sense. So VScode I think as the first to implement that, but you can see LSB now is used in every editor basically as implementing LSB. Every language is implementing LSB. That's a universal protocol that we didn't invent, and we're glad that Microsoft did. So that's another piece of technology that we really care about as just like universal protocols, universal sort of interfaces and things like that.

So now you're talking from your REPL IDE. You're talking to Pitone back and forth. You're installing packages, whatever. If you start a multiplayer session and start coding with someone else, they connect to your container, and you're both on the same container using the same services, and everything that's happening is centralized inside that container. We think the technology that we built for that is really interesting. It's an entire development suite that fits inside a single container with all its different services.

We started thinking, earlier I said we might go local in the future. That thing allows us to go local, because it's sort of agnostic as to where it's running, but you drop it anywhere, it creates this development experience.

[00:50:29] JM: I mean, I could imagine a lot of interesting shared state applications potentially being built on that kind of thing. Okay. Well, there's ton of engineering depth we go into there, but we obviously are up against time. So I'm just going to ask a few more like high-level questions. I'm sure we can – I'm definitely down in doing another show at some point. But I want to hear about your container orchestration woes.

Has the experience of growing up kind of capital constraint, has that affected how the approaches you take to the company, company building?

[00:51:04] AM: Yeah, I would say so. We have a really long horizon. We didn't get a chance to talk about this, but a big part of – We have a very diverse user base in all different axes as diver actually. One interesting part about it is the most enthusiastic users that we have are actually teenagers and young people, college students. What we've seen is that the people that are learning programing Repl.it actually become life-time users and they use it to build projects. They use it collaborate. They meet friends there.

So we think we're really building – We're building the programming service that people are growing up with. So, we're building a lot of equity that in the future will materialize. So, we try to take a lean approach to building the company. For a longtime, we're just – 2016, 2017, we were just three people. Mostly family basically. We're paying ourselves very little. Working outside, out of home.

We raise the round from Andreessen Horowitz last year, and most people would have like blown up the company afterwards and done all of that, but we didn't really. We're hiring and if anyone is interested, we have a really interesting jobs page that you should look at, replit/jobs. I'll leave it to the audience to see it. But we've been very careful in growing the team and we're trying to build the company more sustainably. We're not trying to get on the sort of VC hamster wheel. I think maybe that's been influenced by the kind of growing up capital constraint, like I said, but I just think that it's the rational thing to do.

[00:52:51] JM: And I mean, and you've been working on it for 10 – Whatever, 10, 11 years already. You're kind of have a moat already to some extent. You've been thinking about it for so long. Why give away a bunch of the company before a lot of that vision is fulfilled? You've got a lot of white space to fill in, and if you know how to do it with giving away only a minimal amount of capital, then that's awesome.

Last question, what's the overlap between musicians and programmers?

[00:53:21] AM: I think the one thing that's really interesting goes back to what I said about the conversation aspect of programming, and maybe that's not every musician, but especially improvisation. As you're programming, especially a REPL interactive environment, there's a lot of improvisation.

When I'm starting a new program, a new application, I actually don't know really where I'm headed. A lot of time, I know where the end state maybe should be, but that also changes along the way. So that's why I really like interactive programming, because people compare program to different creative aspects, and it's' all true. Programs, hackers and painters and there are people that compare it to being a chef or whatever. So I think it's the same with musicians, the way they compose new music. The way they do these like jam sessions, and I think it's the same way with programmers, that a lot of creativity just comes out of the improvisation or comes out of the circumstance of it.

[00:54:21] JM: Okay. Amjad, thanks for coming on the show. It's been really great talking.

[00:54:24] AM: My pleasure.

[END OF INTERVIEW]

[00:54:34] JM: Podsheets is open source podcast hosting platform. We are building Podsheets with the learnings from Software Engineering Daily, and our goal is to be the best place to host and monetize your podcast.

If you've been thinking about starting a podcast, check out podsheets.com. We believe the best solution to podcasting will be open source, and we had a previous episode of Software Engineering Daily where we discussed the open source vision for Podsheets.

We're in the early days of podcasting, and there's never been a better time to start a podcast. We will help you through the hurdles of starting a podcast on Podsheets. We're already working on tools to help you with the complex process of finding advertisers for your podcast and working with the ads in your podcast. These are problems that we have encountered in Software Engineering Daily. We know them intimately, and we would love to help you get started with your podcast.

You can check out podsheets.com to get started as a podcaster today. Podcasting is as easy as blogging. If you've written a blog post, you can start a podcast. We'll help you through the

process, and you can reach us at any time by emailing help at podsheets.com. We also have multiple other ways of getting in touch on Podsheets.

Podsheets is an open source podcast hosting platform, and I hope you start a podcast, because I am still running out of content to listen to. Start a podcast on podsheets.com.

[END]