EPISODE 1291

[INTRODUCTION]

[00:00:00] JM: Continuous integration is a coding practice where engineers deliver incremental and frequent code changes to create higher quality software and collaborate more. Teams attempting to continuously integrate new code need a consistent and automated pipeline for reviewing, testing and deploying the changes. Otherwise change requests pile up in the queue, and nothing gets integrated efficiently. The company, LayerCI, is a platform built to deliver a better remote infrastructure experience. It enables engineers to preview full stack staging environments for every commit and have a centralized CI/CD stack with full end-to-end testing.

Layer CI can duplicate a fully provisioned environment so that end-to-end workflows can run parallel and alongside unit tests. The result is faster code review, testing and deployment. In this episode, we talk with Collin Chartier, CEO at LayerCI.

Our first book is coming soon. *Move Fast* is a book about how Facebook builds software. It comes out July 6, and it's something we're pretty proud of. We've spent about two and a half years on this book. And it's been a great exploration of how one of the most successful companies in the world builds software. In the process of writing *Move Fast*, I was reinforced with regard to the idea that I want to build a software company. And I have a new idea that I'm starting to build. The difference between this company and the previous software companies that I've started is I need to let go of some of the responsibilities of Software Engineering Daily. We're going to be starting to transition to having more voices on Software Engineering Daily. And in the long run, I think this will be much better for the business, because we'll have a deeper, more diverse voice about what the world of software entails.

If you are interested in becoming a host, please email me, jeff@softwareengineeringdaily.com. This is a paid opportunity. And it's also a great opportunity for learning, and access, and growing your personal brand. Speaking of personal brand, we are starting a YouTube channel as well. We'll start to air choice interviews that we've done in-person at a studio. And these are high-quality videos that we're going to be uploading to YouTube. And you can subscribe to those videos at YouTube and find the Software Daily YouTube channel.

Thank you for listening. Thank you for reading. I hope you check out Move Fast. And very soon, thanks for watching Software Daily.

[INTERVIEW]

[00:02:47] JM: Colin, welcome to show.

[00:02:49] CC: Hey, Jeff, thanks for having me.

[00:02:51] JM: You work on LayerCI, and there have been many continuous integration products throughout the years. What are you doing new with LayerCI?

[00:03:03] CC: Yeah, common question. So a continuous integration is really just building stuff every time developers push. And I guess 10 years ago, everyone is building the same sorts of things. They were making their Java applications. Npm was starting to get popular. So npm build. And then distribution was just to take the build artifacts and you send them to your customers somehow. But in those 10 years, things got pretty differentiated. So nowadays, launching a web app or an Android app is very different than launching a desktop app. So we're focusing specifically on people making websites, and we give them a bunch of stuff that they need, not just building the artifacts, but also giving preview environments and starting a bunch of copies of their database so that they can run kind of full stack tests against things. So we're focusing specifically on the niche of people building websites.

[00:03:54] JM: How does that differ from the main stays like CircleCI?

[00:04:01] CC: Yeah. I mean, CircleCI mostly just builds Docker images these days. So if you're building a website with CircleCI, or GitHub actions I guess is the big one. When a developer pushes code, it builds a Docker image, pushes it to a registry somewhere. Your cloud provider even has that that functionality built in. So the experience for us is you push code, and it keeps track of memory snapshots. So the process of setting up an environment for a full stack app is generally start the database, seed the database with some data, start the microservices, start the frontend, start the backend. And if you have to run that every time, even if you have the pre-

built Docker containers, it still takes a long time, because Docker doesn't help you run services quickly.

So the LayerCI kind of secret sauce is that we take snapshots of the whole cluster started, and then figure out how to share those across runs. So if you start your database, start your email microservice, start your API, and then make a change to the frontend, you don't have to restart those three things every time. You can just make a bunch of copies and then run several copies of your frontend against it and run some end-to-end tests or manual QA against it.

[00:05:14] JM: So is the overhead for spinning up lots of additional resources for just testing environments, is that significant?

[00:05:26] CC: The big overhead is developer time. If you're using a cloud provider to start things from scratch, then like if it takes 10 minutes to provision your entire environment with building all the services, building the frontend, launching all the services, populating the database, then it's 10-minute iterations every time a developer pushes code, and they want to see if it actually builds in a production-like build. So if you can bring that down to a minute or two with a snapshot of all of the setup being done, that's a huge win for developers that don't have to go on a lunch break, and effectively an extra lunch break for work day.

[00:06:02] JM: So give me a description of what happens when I make a push to my continuous integration pipeline in Layer.

[00:06:11] CC: Yeah. So I guess the prototypical example is you're working on a team. Say you're making us open source Slack competitor. So you have a designer on the team, you have some frontend people, some backend people, someone that makes the database schema, and you push a change that changes something, be it makes a new API endpoint, or changes the color of something. So you push it. It shows up in GitHub. And GitHub, there's the files tab and the pull request tab. So in a traditional workflow, you'd look at that. And as a reviewer, like if you asked a coworker to merge your change, the only two things they could look at where did the build pass? Like does Docker build run? And did the unit tests pass? And just look at the code. So you'd get some check marks, or like it's okay, and you look at the code.

But for a lot of changes for websites, that's not enough, because people are making new widgets, and you can't really unit test widgets. And you can't test the frontend and backend are using the same API endpoints. So with LayerCI, instead of just getting the check marks, you also get a link directly in the pull request that sends you to that environment, the full front-end back-end database, and you click it, and then it spins up the environments. Like it wakes it up from hibernation. And then you can actually test that the features builds without needing to ask for a staging server. And you can actually look at the test recordings. So if you're using something like Cypress, you can set up Cypress tests that run with everything, so the frontend and the backend. And then you can get your video recordings of, “Oh, this workflow changed. The screenshot isn't the same as it used to be.” And then you can validate directly from GitHub without needing to run the tests yourself.

[00:07:54] JM: What has been the driver of the rise of these modern CI tools? Because there are some other ones that I've seen. Like I talked to Release App, for example, they've got they got acquired by Netlify. Is like the rise in better infrastructure tooling, like Kubernetes, for example, is that driving some of the creation of new CI infrastructure companies?

[00:08:21] CC: I think it's multifaceted. The reason that – I guess the big why now are everyone's using Docker. So like there's some semblance of being able to run it under a local computer. So like in 2005, everything was PHP. So if you had LAMP installed on your Windows laptop, you could run everything. And then it kind of devolved into like, “Oh, there's all of these different services and builds,” and you're using like AWS-specific things, and you can't run it locally anymore. But now we've kind of come full circle where most new products are using lots of open source tools, like they use nginx, they use Travis, they use Kubernetes, they use Postgres, instead of using the proprietary ones. And once you can run everything on an individual person's laptop, you can run it in the cloud. Like it's actually possible to run it outside of AWS.

The other thing is there're lots of opinionated platforms like Netlify that have appeared for specific niches. So Netlify is a CI effectively, but it's specifically for people using Jamstack, or people using frontend-only projects with maybe a sprinkling of backends. And that experience is really good, because if you're building a frontend, your needs are very different than the

traditional run my Java unit tests that CircleCI or Travis was built for. And I guess people expect that level of service now for their products and all sorts of verticals.

[00:09:37] JM: So is creating CI coverage for lots of different deployment situations, like if I'm deploying a Rails app, versus deploying a change to my React frontend, versus something in Kubernetes, is it a challenge to have a CI coverage of all these different environments and platforms and potential change sets that you need to cater to?

[00:10:09] CC: Well, I guess one of the big things we focus on is just like not opinionation. So the interface we show people is essentially it looks like a Docker file. So you have a linear, a series of steps. And then our promise is just that we cache the repetitive parts. So we map which files are read to like what was actually started in the step. So if you have a database provisioning script, we can tell that that was the only file read. And that means that anything you can run on a laptop, you can convert into one of these provisioning scripts, and it'll just kind of magically be cached quickly regardless of whether it's a Rails monolith, or an npm build, or if you're a microservices, you're running Docker compose build.

So the way we lean is that, because these teams are going to have multiple different technologies, we can't really focus on any particular ones. Like if we made npm build itself very fast, then that doesn't help much. Because for our customers, you're always going to use a different web framework for the backend. Like you might have npm build for your React frontend, and you might have some like Python, or Django, or Rails monolith for your backend. I think there are some tools that are very opinionated about like, "This is specifically for Go developers. Like we do all the Go tidy stuff." And I think there is space for that, but probably not so much at bigger companies, because they'll inevitably use multiple technologies and languages. And then if you're only focusing on one language, then they'll need to splinter their build across two services anyways.

[00:11:36] JM: What are the other canonical engineering problems that you run into in developing Layer?

[00:11:44] CC: I guess the big problems are just scaling the memory snapshotting, because if you're building a tree of snapshots – Because every time you build, essentially, you're making

snapshots that inherit. And then if your developer team is making hundreds of snapshots a day, so hundreds of pushes a day, then we have to figure out kind of which snapshots might be relevant in the future. And you can't keep all of them, because snapshots are relatively big. If you're snapshotting a 10 gigabyte VMs memory, even if it's just the difference in memory, that still ends up being several gigabytes. So we have to be very clever about which snapshots we keep around. And that's probably the hardest engineering challenge. We have lots of heuristics for which snapshots might be used. And also, the snapshots are used for the ephemeral environments. So if you click a link in a PR and it wakes up the snapshot so that you can interact with it, we can't delete the snapshots that are used for that. So we need more logic to figure out which memory snapshots might be used, which ones might be used to speed up further builds, and which ones can be safely deleted.

[00:12:50] JM: And sorry, if this is obvious, but can you explain why memory snapshotting is important here? Like why is this even a feature of CI?

[00:12:59] CC: Yeah. So I guess the big problem with things like Docker build – So Docker build itself is just the snapshots. Docker doesn't do really any memory snapshotting in the build process. So if your Docker file looked like, run start Postgres, run load the database dump, run backend, run end-to-end tests, for example, for like the npm test use case, then when you ran the frontend, or the backend, the back end would say Postgres isn't running. Because between every layer of a Docker build, everything shuts off, because it's only snapshotting the file system. So for like, I guess, full stack use cases, that's a huge limitation, because you often want your services to be running between the build steps. And that's why we care so much about memory snapshots.

[00:13:49] JM: Can you tell me more about the infrastructure stack that goes into the CI environments that you spin up? Like what does this stack look like? What's the cloud service? What's the virtualization stack look like?

[00:14:03] CC: Yeah, this is probably one of the biggest backend differences between us and like a traditional CI. So like if you are starting something that just ran builds in 2010, say, the infrastructure you think of was use a cloud provider, use spot instances. So like cloud providers give you machines quickly, and for cheap, if you're using them for an hour, which is perfect for

builds. And then like every time a customer asks you to build something, you spin up one of these machines. You load the customer's code in it, you build, and then you tear down the machine. But we can't do that, because we need very low-level access to the VMs. Like AWS doesn't let you quickly restore a snapshot of a specific VM. And in particular, doesn't let you make these trees of snapshots.

So our infrastructure at Layer is actually a Kubernetes cluster over bare metal. We, I guess, kind of made our own cloud provider by stitching together a bunch of really big servers. And then the servers have local disks with all of the memory snapshots. And then when you push, we scheduled you onto a server that has the most relevant memory snapshot to what you pushed. And then the server can very quickly bring up the VM using its own hypervisor. So it's Kubernetes over bare metal. Kubernetes is running containers. Those containers are running VMs. And then those VMs run the customer's workloads. So the customer actually ends up being very close to bare metal with the way we do things.

[00:15:32] JM: Is there a reason? Is that important?

[00:15:34] CC: The big reason is the memory snapshots. Again, if you only have the interface of, "Here's a VM," and then copy all the data into it, then it would take a long time to pre-populate it with the dependencies you need to run your services. So if you're using spot instances in AWS, the only interface you get is give me a fresh VM or give me one based on a disk snapshot. And again, that's the problem with Docker builds that we talked about earlier.

[00:16:03] JM: So the bare metal provider is where? Sorry. What's the cloud provider using?

[00:16:10] CC: We use OVH, which is a big, I guess, European bare metal provider. Our servers are actually in Montreal, because it happens to be near a hydroelectric dam. Like the data centers right next to a dam. So it's easy to get lots of big servers there.

[00:16:24] JM: Do you feel like you're missing out from having the AWS ecosystem of API's?

[00:16:31] CC: I mean, it certainly wouldn't have been possible before Kubernetes. Things like access roles, and like reverse proxies, and floating IPS. Like all of the, I guess, traditional cloud

provider things are really important if you want high uptime. But we can circumvent that by using lots of Kubernetes and using a few additional SaaS providers. So instead of AWS' CDN, we use CloudFlare, which is free actually, I guess amazingly. So it goes our users hit CloudFlare. So the page load times are still fast as if we were using like an AWS load balancer geo replication setup. And then once it hits Kubernetes, it uses Kubernetes built-in Ingress management system. So it's easy enough to do routing and do high-availability and service discovery.

So because of Kubernetes, we can kind of avoid all of the downsides. I think we'll actually be seeing a lot more of that in future companies. Cloud providers, I guess, traditionally provided VMs and CDNs were the two big things they provided. Or containers now, containers, VMs, and CDNs. But even things like serverless, you can run on a Kubernetes cluster. I guess there're lots of providers that just run your Firecracker VMs the same way Lambda would. You can really run anything in a Kubernetes cluster these days.

[00:17:46] JM: Can you elaborate on the Firecracker stuff. I mean, I know what Firecracker is. We've done a show on it. I know AWS knows how to run Firecracker. But are you saying that you can run – Like other people are running Firecracker on their data centers?

[00:18:00] CC: Yeah, there're services like open fast OpenFaaS, open Functions as a Service, and Kubevirt. All of these services that run in a Kubernetes cluster. And they let you make the same interface. So you can declare your Lambdas as like a JavaScript blob. You put that as a resource in your Kubernetes cluster. And you say, "When someone visits this endpoint, wake up that Lamda. Send the request to it, and then turn off the Lambda, or like hibernate it, or whatever you're going to do."

So even if you're using cloud provider specific tools, you can still usually find some equivalent in a Kubernetes cluster. Then past a certain point, the Kubernetes clusters are nice, because you have full control of replication and where the servers are and how they're interacting with your users. And you might have some big servers for batch processing. And you can label those as the batch processing servers and attach them to an existing cluster. So we have some servers for US customers. And it was as simple as buying servers in the states so that they don't – like the data doesn't leave the states, and then attaching them to our like Montreal-based cluster. And it was effectively just running 10 commands to connect those worker nodes in the US to the

big cluster in Canada. And then we can make sure that data stays local to whatever country the user cares about. But you couldn't really do that easily in a traditional cloud provider, because like cross-world connections are much more difficult there.

[00:19:27] JM: Do you have any other predictions for how data center infrastructure might evolve?

[00:19:35] CC: The developer tools world is becoming very, like opinionated, almost like vertical-focused. So I guess, again, like 10 or 15 years ago, everyone was just building their Java app, like their spring hibernate app. And then all they'd have to do is build the JAR and then they'd like push the JAR wherever. Nowadays, there's like no code is essentially totally removing the need for code for something companies. Like different verticals have very different needs. And lots of things are moving to the web. So like maybe the traditional model for like a UX designer's tool is like Creative Suite. So like Adobe would make these applications. And they bundle them and ship them to designers. And the designers pay some fixed price.

But in 2021, everything is SaaS, and everything's in the web browser, because it's easy to distribute and upgrade. So Figma is taking over that industry. It wouldn't surprise me if Figma got acquired by Adobe, and Adobe started putting their weight behind online subscription-based like UX editing. And the same happens all over the place. Like, we've been seeing lots of tools that do like video editing in the browser, or like pushing your browser to a data center so that you have very powerful processing in your browser so that you could run like 3D model builds.

[00:21:00] JM: You're talking about Mighty.

[00:21:01] CC: Yeah, I think, Mighty. We have a friend that works there. But like all of these technologies are basically like conglomerating everything into the browser, joining the browser into like the core workflow, and then facilitating making browser-based companies. So like the traditional kind of publishing-based websites, like the newspapers or whatever, like those publishing platforms like Medium that require no programming. You can become a publisher without needing any code on the internet, on specifically websites. Things like games. There's Unity for web. So you can make games that fit in the browser without needing any browser programming. You just make the game. You publish it to your Unity web store. People play it.

And I think that more and more, all of the products that people commonly use will be moved to websites where they can easily be upgraded and distributed and shared across computers. And services that facilitate that will be more and more opinionated. So the blogs and the CMS's will get specific tools for people working with them. The full stack sites, the ones that need a backend that are doing some like blazing cutting edge stuff, we'll have tools like LayerCI that focus on them. And like people building artifacts that need to be distributed, like Android apps and iOS apps will have CIs and build tools and developer tools that focus for them. So I think that's the way the world is going. The no-code is kind of eating the traditional things you could do with a website. And then things you can do with websites are getting more powerful. So like with Canvas, and WebGL, and all of those new technologies, it's possible to make all sorts of new amazing things like Figma in the browser that wouldn't have been possible 10 years ago.

[00:22:46] JM: Side note on Mighty, since you mentioned it, and I've been following this company. Do you think that company has a viable go-to-market strategy? Like the technology is so cool, but it just seems really hard to get people to switch their browser.

[00:23:03] CC: I mean, I think there are people that need something like that. So it's almost the same argument as like who buys the Mac Pro. This is like a ridiculous $6,000 tower. And like who needs a prebuilt $6,000 Mac? Oh, it's the musicians. It's the CAD creators. Like there're all of these different industries that need a super powerful single Mac to do the professional work on. And so like despite the Mac Pro not being a common ubiquitous thing, it still makes a lot of money for Apple, because people, like professionals need it for their specific verticals. And I think Mighty is going to be very similar. Like you'll have the cutting edge companies that are making like Canvas-based video games, or Canvas- based CAD software, or Canvas-based like video editing software, and all of those need a very powerful browser to be able to run or need a very powerful computer. But then you can basically turn your laptop into a thin client if you do that. So if you have a browser-based app that does something powerful, like animation, and you have a weak computer, like you have a MacBook your choices are you can either buy a bigger MacBook and then hope that it doesn't happen again. Or you can just shuffle it off to Mighty. So I think there are definitely some people in the world that need it. And there'll be a lot more people that need it if there's bigger apps pushed into the browser. If there's more of creative

suite like Photoshop that's pushed into the browser, then people will pay to not wait for things to load or move around.

[00:24:39] JM: So it's like the same argument as Superhuman, basically, power users.

[00:24:44] CC: Yeah. I mean, you have to find your niche, right? Like Mighty's go-to-market I assume starts with power users given how much they're charging. And then once the power users are using it, you can backstop and get more geographies, geographies.

[00:24:57] JM: How much are they charging? I haven't seen it.

[00:25:00] CC: It's $30 a month or something? It's pretty high for a browser. I mean, most people wouldn't pay $30 a month. But if you're doing video editing and you'd like to do it on your Chromebook because it has 20 hours of battery life and you want to work from the beach, it might be something you consider.

[00:25:14] JM: Okay. Coming back to LayerCI, I think one way of looking at Layer is I think CI is actually one of these things like logging, and monitoring, and name your sort of infrastructures thing, where it's a category that's so deep and so perennial that new companies just always are going to be created for it. And it comes down to this sort of like how can you capture the flavor of the month and build a big enough market share given where you are chronologically. So maybe you can tell me whether that's true and how you find that niche and attack it?

[00:26:02] CC: Yeah. I mean, like there're lots of recent examples of companies that have done really well with kind of like maybe not – They don't call themselves the CI companies necessarily. But, again, things like Vercel and Netlify that focus entirely on frontend developers, and their go-to-market is, "Are you a frontend developer that's building sites and wants to share them with your team before you publish them? Then use Netlify preview environments. Or use Verce preview environments." If you're making Android apps, there's like specifically for Android and iOS. Like get the latest version of Xcode, because GitHub actions doesn't have the latest version. So if you find the target user that's using the general solution, it's usually relatively easy to better serve them. Like a lot of users migrate from GitHub actions to us because they need some specific thing that the general platform won't build and has no attention to build. So they

need like a debug terminal that they can enter the debug terminal for the failing build and Docker logs the failing container. It's like if you don't have that functionality, and you're doing Docker-based stuff, then you're going to waste a lot of time. So that's a relatively easy sell.

So I guess the niche of customers we were initially targeting was people using Docker Compose, because it's easy enough to get started if you're using Docker Compose, and that have like a built frontend of some sort. Because if you have a frontend, then it's even more compelling, because you get the ephemeral environments. And I think if people want to start similar companies, then they should just find a common niche like that. Like the intersection of people using Docker Compose and like React is already enough to get $100 million company probably. And then from there, you can branch-off into other technologies. Do you use Kubernetes? Do use non-build frontends? Are you a web app that is primarily an API that mobile apps connect to? Those are all things we can support longer term.

And, yeah, I think there is a big torrent of developer tools that needs to be built in the next 10 years to facilitate all of these new companies, the virtual event companies, the virtual office companies, the Figma competitors. And those niches are all underserved right now. And people are still using Jenkins. They're so underserved that people are building their own things with engineering teams internally using Jenkins, which of course is ridiculous, but such is life.

[00:28:24] JM: And humor me, why is that? What do you think that's ridiculous? So, I mean, I would personally say like, "Okay, Jenkins dominated for a period of time," and it basically built its dominant presence around the kind of the rise of the Java monolith basically, or the period of time where you had Java monoliths moving to Java microservices and service-oriented architecture, and it was a toolset built for kind of a specific time and place still serves that niche. I mean, it still makes sense for those companies to be using Jenkins to a certain extent, right?

[00:29:05] CC: Well, I guess when I say ridiculous, I mean, the people that aren't building Java model monoliths. Like we see people using Jenkins with Node.js, or Jenkins with their new startups that has no users yet. They set up a Jenkins configuration, because it's what they're comfortable with. But just the experience, like if you set up Jenkins now in five years, your developers will have a much harder time building features than if you started with like some tool that was specifically made for your use case. Like if you're a frontend developer making a

frontend-only product with Jamstack, and you choose Jenkins instead of like Netlify, that's going to be a huge pain for your team, and you're going to have much worse features, and you're going to have much more maintenance costs despite all the plugins, because it's just not meant for that specific persona. So I guess when I say ridiculous, I mean, it's ridiculous for people not building Spring Boot monoliths.

[00:29:57] JM: So you characterize your customer profile as sort of this more modern Jamstack type character.

[00:30:06] CC: Our customer profile is people that have like a frontend framework. So they're building some sort of like SaaS or interactive web app, like ecommerce, something along the lines of you have a frontend and you have a backend. And you'd like people to be able to develop both of those things without setting up a staging server or setting up a screen sharing session to show their work. And in particular, you'd like reviewers to be able to tell that the description of the ticket is actually solved. So if someone says, “I fixed this bug in our full stack project.” You might not be able to write a unit test, because there're lots of interactions between the components, but you can at least manually check that the interactions have been bought through. Like does the frontend talk to the right API endpoints, for example? That's the sort of customer persona we're dealing with. Whereas if you're frontend only, there's always things like Netlify or Vercel that you can choose. I mean, you can always use us with frontend only as well, but I think the others are more established in those niches.

[00:31:07] JM: What's the nature of the name LayerCI? What does the Layer refer to?

[00:31:12] CC: It talks about like the build layers. So starting the database is a layer. And then from that layer, you can have like multiple different services. So if you have three backends, for example, you can have your base configuration that sets up the common parts. Maybe installs the right version of Rails, or installs the right version of Docker Compose. And then from that layer, you can have a tree of dependencies. Each of the dependencies builds on that common base, and it forks the state of everything that's running. So for example, you might install Docker Compose, run your Docker Compose start for your microservices, and then make three different VMs for three different micro services so that you can test them in isolation.

[00:31:56] JM: What are some other ways that you can differentiate over time? I assume you thought about this problem deeply enough to have a lot of strategic plans for things you could build off of kind of the CI infrastructure that you've built thus far?

[00:32:17] CC: Yeah, I mean, a lot of times people treat CI as like an afterthought for their product. So like Netlify's original pitch was like hosting for Jamstack. And then they realized for people to host things, they needed to build it. So they kind of just like threw together the npm run build and upload the files somewhere as a service. And you see this time and time again, where people want to make Heroku. So they want the money of hosting production, but the best way to do that is to be like an all-in one PaaS where, "Oh, you host your production with us. But we also run your builds."

I guess we're almost going the opposite direction of that, where we're saying, "You're building a full stack web app. Of course, you're going to host it in the cloud provider, or Kubernetes, or GCP, or like some production hosting provider, because you don't want to get fired when it goes down." So like our roadmap is really just building more and more general features for that class of people. Like how do you set up a deployment to – Like how do you use Spinnaker to deploy with a rolling deployment? How do you test your Lambdas? How do you test your individual AWS microservices? And then we can provide more and more of those interfaces so that people don't need to use their cloud providers builds and do get all the benefits of having specific service specifically for reviewing things and pull requests without needing to host their production with us.

[00:33:43] JM: And as far as the sales process, is it kind of a process of just going through developers and having kind of a bottoms up gradual winning over a large customer? Or do you have to do any kind of bigger like outbound or speaking with CTO kind of stuff?

[00:34:08] CC: It depends on the team. I guess the main decision maker for installing LayerCI as a team lead. So usually it's like there're 10 developers working on a specific web app. And people are miserable, because like bugs are making it to users like, "Oh, the Jest tests passed, but I typo'd the API name. So it didn't work in production." Or like, "My boss has yelled at me a bunch of times, because things broke. And so we had this set up like Cypress tests, but they're flaky, because it doesn't – We're using a shared database, and sometimes things change in the

shared database.” So the people that use us or choose to install us generally know why they're installing us. It's either like our hack together like a full stack testing is flaky and hard to maintain, or we need somewhere to run our Cypress tests, or screen sharing sessions are really annoying and we only have three staging servers. So people keep getting confused about the state of things, and people are stepping on each other's feet. And then like, I guess, it’s obvious throughout the team that something's broken in the tooling side. And then either a team member will hear about us and bring it up to the team lead, or the team lead will directly seek out solutions to these problems. And then, yeah, like we often install alongside an existing CI process. So if you're using GitHub and GitHub Actions for your Jest tests, you can just install us as a third check for an ephemeral environment to avoid needing to set up Zoom calls, for example. And then once people have us installed along their existing CI, overtime, they'll put everything under the same umbrella to avoid double paying and needing all the extra work.

[00:35:52] JM: As we begin to wind down the conversation, I'd like to know a little bit more about how the business is evolving right now and the challenges you're facing. Like what are the biggest things you're focused on? Is it hiring? Is it specific technical challenges? Go-to-market sales stuff? What's at the top of mind?

[00:36:14] CC: Right now it's mostly sales enablement and top-of-funnel. So like our persona, like developers that are in pain, because they have bad tooling for their full stack web apps, they don't usually know what to Google for. Like there's a few keywords like ephemeral environments that people might have heard of, and they might Google for it as a solution to their pain. But oftentimes, people just kind of ask in Reddit forums, or ask, like they think, “Is this DevOps?” And then they read about DevOps. They're like, “Well, I don't know. It's unclear to me.”

So a lot of what we're doing is just education, like teaching people, “Oh, if you are someone working on a full stack application, then here's all the different aspects of DevOps for your company.” Like there's pull request automation, which is kind of what we work with. It's like a developer proposes changes, and they want someone else on the team to verify that the changes aren't bad. And then there's deployment automation, which is like once changes aren't bad, how do you show them to users to kind of like piece together some feedback in a way that doesn't cause downtime?” And then there's performance management. So once it's deployed, how do you make sure it keeps working? And like people generally have a broad overview of

that, but they don't necessarily think about it so formally. So a lot of what we're doing is making educational material, landing pages, like one-pagers, and disseminating those.

[00:37:38] JM: And give me some broader perspective on where developer tooling is going. I mean, you're selling into the world of developers. Can you give me some interesting tidbits for what you think is going on? But anything else you can share, because there's so much change going on right now. And I'd love to get your perspective on cloud providers, or just general developer tooling trends that you have in mind that I may have overlooked.

[00:38:07] CC: I mean, I think it's really like stratifying into the like enterprise and the like personal projects, bubbles. Like everything is almost splitting into those two. So like for enterprise, like once you're past a certain size, you care about like rolling deployments and feature flags. And so like there's this large group of companies that are making like open source feature flag software, or open source deployment software, or wrappers around Spinnaker. Spinnaker is Netflix's deployment tool. I think it's Netflix. And then on the low-end, there's like, "We'll do everything for you and handhold you and give you all the features with no configurations, because you don't have time to do tooling before you have product market fit." And I guess that makes sense. Like the small companies don't want to think about things. So they want something that's very low configuration. And the big companies all have very different deployment processes and technologies used, because they've been iterating for 30 years with technology. So they all have different stacks and like legacy software that all needs to be supported.

So they care more about extensibility and, I guess, the benefits of Jenkins, which are you can write plugins that run arbitrary code. And it's all very extensible and easy to understand, but maybe not easy to understand in case of Jenkins. But that's where I see developer tools going. It's a lot of like personal project stuff and a lot of enterprise stuff. And then there's not so many companies making things between the two. You have 10 developers in the team and you want tooling that is both configurable, but not going to take a consultant to setup.

[00:39:48] JM: Cool. Well, it's been a real pleasure talking to you. Do you have anything else you want to add or discuss before we close off?

[00:39:55] CC: What are the coolest companies you've seen in the built space? I guess as a podcast host, you've probably seen lots.

[00:40:02] JM: I thought Release App was really cool. It doesn't surprise me that they got acquired by Netlify. And it wouldn't surprise me if you guys got acquired by somebody. Maybe Vercel will acquire you. But as far as infrastructure goes, as far as these next generation cloud providers go, I invested in Render. I really like Render. I wouldn't be surprised if they do something – I mean I have no privileged information. But I wouldn't be surprised if they did something cool around builds. I still love Heroku. I still love the Heroku experience. I haven't used GitHub Actions. So I can't speak to that. But like I said, I think this is one of these just perennial things, builds. Nobody likes building software. Nobody likes the release process. Nobody likes doing like unit tests and stuff. People hate this stuff. So it's always going to be a pain point. It's always going to be a market. It's always going to have fresh ideas that you can bring to the table. So yeah, I follow it pretty closely. Does that make sense to you?

[00:41:01] CC: When you said Release App was acquired by Netlify, are you sure you're not thinking of FeaturePeek?

[00:41:06] JM: Oh, my God! I think you're right. I think I got those confused. You know what happened is I did those interviews pretty close to one another, and I did confuse them. So Release App hasn't got acquired. That's right. Okay.

[00:41:17] CC: No. Release App just announced a seed round. FeaturePeed was acquired.

[00:41:20] JM: Oh my God! This is a sign I'm doing too many of these interviews. Too many build tools. Okay. Yeah, FeaturePeek. That makes more sense, because Netlify has their own release stuff. Okay.
Alright. I'm losing my mind. So, cool. Yeah, I like Release App though.

[00:41:35] CC: What's the innovation in the world, which is good? I mean, like there're clearly problems around this. If you talk to people building the software, they're always –

[00:41:44] JM: How's your philosophy different from the Release App strategy?

[00:41:49] CC: It's different. They're much more opinionated. They're focused entirely on people using Docker. Ours is more like build a VM, build a VM quickly. Like the VM might have different tool chains. It might be make building Docker containers, but like whatever your existing processes is, you can just stuff it in a VM for LayerCI. Release App is more about you can figure how to set up Docker containers, assuming everything already runs on Docker. And then if you're already using everything in Docker, then it's a good experience, because it's opinionated. It's like a maybe even tighter niche than what we're building for, but similar sort of results. We're also often used for end-to-end tests. And I don't think that's a niche that Release App is really going for. They're going really entirely for human QA and release environments, or preview environments.

[00:42:38] JM: Gotcha. Cool. Well, real pleasure talking to you. Anything else?

[00:42:43] CC: That's it.

[00:42:43] JM: Cool. Well, Colin, thanks for coming on the show.

[00:42:46] CC: Yeah, thanks for having me.

[END]