**EPISODE 1095**

[INTRODUCTION]

**[00:00:00] JM:** The modern release workflow involves multiple stakeholders; engineers, management, designers and product managers. It's a collaborative process that's often held together with brittle workflows. A developer deploys a new build to an ad hoc staging environment and pastes that link to the environment into Slack. Other stakeholders click on that link and then send messages to each other in Slack, or they make comments on the pull request in GitHub.

This workflow is far from ideal. Collaborating around pull requests can be made easier with a dedicated set of tools for sharing and discussing those pull requests, and that's the goal of FeaturePeek, which is a system for spinning up dedicated pull requests environments, creating screenshots and comments and reimagining the lifecycle of the release workflow.

Eric Silverman is a cofounder of FeaturePeek and he joins the show to discuss release management, the interactions between different stakeholders and the development of his company. Much like the previous show about Postman, in which we explored how API management has become a ripe space for collaboration, the same is true of pull requests. It's also a ripe for collaboration.

Before we get started, I want to mention that we are looking for writers. If you are interested in writing for Software Engineering Daily, send me an email, jeff@softwareengineeringdaily.com. And also, I am personally looking for companies to invest in. If you are building something for engineers or something that requires heavy engineering, send me an email, jeff@softwareengineeringdaily.com. I'd love to look at what you're building.

[SPONSOR MESSAGE]

**[00:01:41] JM:** Your code is going to have errors, even code written by an amazing developer. When bad things happen, it's nice to know that Honeybadger has your back. Honeybadger

 

combines error monitoring, uptime monitoring and cron monitoring into a single, easy to use monitoring platform for less cost than you're probably paying right now.

Honeybadger monitors and send error alerts in real-time with all the context needed to see what's causing the error and where it's hiding in your code so that you can quickly fix it and get on with your day. The included uptime monitoring and cron monitoring also lets you know when your external services are having issues or your background jobs are going AWOL or silently failing.

Honeybadger.io is 100% bootstrapped. It's a monitoring solution that's bootstrapped, and this is important, because a self-funded business means that they're priced simply. They operate the business simply and they answer to you, not investors. Software Engineering Daily listeners can get 30% off for 6 months of Honeybadger, and you can simply mention Software Engineering Daily when signing up. They'll apply the discount to your account, and you don't need a credit card.

Thanks for listening and thanks to Honeybadger for being a sponsor.

[INTERVIEW]

**[00:03:03] JM:** Eric, welcome to the show.

**[00:03:04] ES:** Thanks for having me.

**[00:03:06] JM:** You work on FeaturePeek, and in order to explain what FeaturePeek is, we need to talk about the modern release process and how the modern release process involves not just developers, but designers and product managers. Describe the modern release process to me.

**[00:03:22] ES:** The modern release process at this point has become a lot more complicated than you would really think. There are a lot more hands in the pot. There are a lot of more people touching every step of the way in terms of getting code that was written on your desktop or your laptop on to the server deployed out to your customers. The most basic way that this

has kind of evolved is it goes from your local dev environment. You push some things to staging, you push some things to prod. QA happens probably under staging environment.

What's happened more is that there are a lot of different stakeholders who are really involved in or should be more involved in every step of this process, because either frontend engineers are working way closer with design, which is something that we've really seen, and that a lot of people need to sign off on changes that can affect lots of different parts of the company. So as engineers are moving to more DevOp, or incorporating more DevOps technologies, there are a lot more opportunity to really optimize some of the release flow where it's not as linear as dev staging prod, which is what I'd say a lot of people just by default end up adding to their workflow as they're building up a product, as they're starting a company, as they're adding some formal release process to their team. Pushing to dev, QA running on staging and it going to production is well-worn territory at this point. Or they have maybe an AB canary setup, something like that.

What we're trying to do is really use the idea of deployment previews. Some people call them feature environments. Some people call them on-demand staging environments to not only disrupt that flow a little bit and bring some of the validation and verification of new features and new work earlier in the release cycle, but really make sure that all stakeholders are involved in the release process as they need to be way before any sort of critical release point.

**[00:05:15] JM:** This CI process has really become a place for collaboration as you're alluding to. This one-off feature environments, they can be collaborated around. What is the ideal collaborative workflow?

**[00:05:32] ES:** Well, I think the ideal collaborative workflow is, number one, making features available to verify test and run as soon as possible. Two, providing the right kind of tooling on top to actually keep nontechnical members of your team or distribute in members of your team able to – Putting tools in front of them that can allow them to give really actionable and quality feedback. So that means things like screenshots, bug templates, browser metadata, all the stuff that as an engineer makes me go crazy when you receive a bunch of bug reports and you have to then follow up and say, "Was this on Safari or Chrome? Was this on mobile? What's your setup?" all the kind of stuff that really drives engineers crazy. It's really being able to abstract that away and make sure that happens for free, and incentive people to participate in this

process. I think gone are the days of throwing designs over the fence or saying, “All right. Well, I handed it off to engineering.” We really think the whole developer hand-off idea is pretty antiquated. It made a lot more sense when you had more of a linear deployment model. It made more sense when teams had kind of bigger boundaries between them. But what we’ve seen is that teams – A lot of people work on everything. Designers work really close to the frontend engineers. Everyone is working together. There are a lot more stakeholders. So you need some of that ownership. All the stakeholders need some ownership in the process.

**[00:06:55] JM:** I think one analogy I could draw, and this is maybe a generous analogy, but the way that Figma has opened up the design process to a lot of stakeholders, I see you as trying to do something similar with the release process and try to allow other stakeholders to collaborate in this place that was in the past sort of like I spin up a feature environment and then I send you an instant message with the feature environment and you go look at it and then you respond back to me, and it’s really formalizing that process.

**[00:07:34] ES:** Yeah. And I think Figma – I mean, Figma is an amazing product, and we’ve obviously taken a lot of inspiration from them. If you even go back before Figma, it’s something that envisioned it really well, right? Being able to markup a prototype, annotate, leave comments there, and then Figma obviously pushed it a level further, really being able to design there.

What we see is the other side of that. We see that there’re a lot of really great tools for designers that are made for designers to work better with their teams. We see FeaturePeek as one of the first real engineering-led tools to change that process. Collaborating back with design, back with product, instead of just tools for the other side of the hand-off from design to engineering.

**[00:08:16] JM:** As we get closer to kind of explaining what you’re doing, I would like to get your perspective on the Jamstack and how this modern release process is reflected in the Jamstack or how the Jamstack has changed these releases and just the modern collaborative workflow.

**[00:08:38] ES:** Sure. I think the Jamstack works really well to move really quickly, and I think that what it’s done is given frontend developers a lot of power, a lot of power to move really quickly to build really powerful single-page applications to kind of abstract away some of the

content pieces of this. And I think it's allowed for a lot of velocity and a lot of ownership in this space.

I think what the Jamstack has encapsulated, and this is part of the reason we think FeaturePeek works really well, and we can get into that. But I think if anything, there's so much more velocity because of people adapting Jamstack, Jamstack applications. I mean, we use Gatsby at FeaturePeek. Our marketing site is Gatsby application. And we think that, if anything, that's made it easier to collaborate. I think it's really opened up. I mean, I know Gatsby has got that great review feature specifically for this, is people are very much building tools to easily collaborate. And I think Jamstack applications has really opened up that potential a lot more.

**[00:09:52] JM:** In a typical review, when I've deployed my software and it's in review, what kinds of issues are going to be discovered by the various stakeholders who are taking their gander at my release?

**[00:10:08] ES:** Oh, sure. Lots of different issues ranging from this doesn't work in my browser. This has some – Maybe we're looking at this against production data and this component breaks. We have some overflow issues. We've had some of our users even just tell us, "Oh man! I had a local font installed," and I looked at the FeaturePeek environment and we didn't have the right font. It's really trying to solve the works on my machine problem. So being able to actually have it up there, mimicking what it would be running out there in the wild. You can catch a lot of different issues. It could be anything small from this shadow doesn't work in my browser all the way to this component is totally broken. This doesn't even work for me. Lots of issues like that.

**[00:10:51] JM:** And the design issues that occur, are these pretty innocent, or do the minor design issues that might crop up, do they significantly diminish product quality? Because I think one thing I see in FeaturePeek is there's a way to highlight and uncover and zoom-in on very particular minor design issues. And I just like to get your perspective on how really minor design issues might affect the quality of the product.

**[00:11:22] ES:** Well, I think it's more about the handoff – The handoff process right now is the designers work very hard on handing something over that they feel confident about, and the

engineers are going to give it their best shot. They're going to try to match the spec maybe not all states were designed for. There's always going to be some gap in communication there. Those could be either functionality issues or they could be I had to take some liberties with a hover state, or something doesn't necessarily flow on the page the way the design intended.

It could be something as simple as the shadow or this corner doesn't look right, and it can be nitpicky stuff. But what we're trying to really do is move all of that feedback earlier. So even if it is nitpicky, it can be screened appropriately or it can say, "Oh, that's not as important. That's not a blocker." Or while I'm already working on this, I can tune-up those bugs. The problem that we're trying to really solve is getting all that influx of issues, be it small issues or major functionality issues the very last minute is really going to – That will block the release. That will impact the quality of the release. Having all that feedback earlier, you at least get to decide and get to treat those issues the right way.

**[00:12:36] JM:** The minor issues often take the form of things that are like pixel width or the distance between different elements. These things are pretty important in design. They're kind of hard to report back to the developer. Maybe you have some particular – One thing I think here is the developer doesn't necessarily speak the design language. A developer doesn't necessarily know, "Hey, this needs to be a more gradient." I don't even speak the design language.

There's a role for screenshots and screen recordings and just kind of taking a visual perspective on what is going wrong in a design. In a typical like release process, I think about the Apple screenshot taking tool or the QuickTime screen recording tool as being pretty pivotal if you don't have some better way of gathering feedback. In terms of gathering visual feedback, what's the state-of-the-art in reporting that back to the developer that's managing the release?

**[00:13:43] ES:** Right. What we would expect, and this is something that we have, I've had in my projects in various different roles, is a lot of people add a bug template. A lot of people say, "Okay. Here's how we on our team like to have good bugs. We need browser metadata include screenshots, include steps to reproduce," and that's different from team to team depending on how they screen bugs or take in tasks and issues. We've built that in specifically, because we know that if it's right in front of you and it's very easy for you to take a screenshot, record a

video, you'll do it. I think the problem is – And yes, it's great. The Apple shortcuts to take screenshots are great. The problem is, in the moment, you don't always do it. I mean, how many bugs have you seen, and I've seen hundreds, where you just go and say, "Oh, this doesn't work." "Oh, this is broken." It's like, "Okay. I need contexts." Engineers really need that kind of context.

Just in the same way that engineers don't speak the design language, design product, others non-technical stakeholders don't necessarily speak the engineering language. So what can we do with tooling to really kind of bridge that gap and make them not have to speak the language and still file really great issues and communicate great feedback?

**[00:14:53] JM:** All right. Well, we've kind of laid the foundation for what you're building. Explain how FeaturePeek works.

**[00:15:00] ES:** Sure. FeaturePeek spins up a dedicated deployment preview for every pull request. As your code is in code review, getting reviewed by other engineers, there'll be a comment right in the pull request that will link you to a running site with your code. On top of that, we have what we call our drawer, this little drawer overlay. And so on top of that, right laid on top your page, is a little button you can easily take screenshots. You can record a video. You can connect your issue tracker. So we integrate with GitHub issues, Clubhouse, Trello. We're rolling out new ones every couple weeks. You can file tickets right from the page. You can see who on your team has actually viewed these changes. Actually, if you want, you can leave a comment and even block a PR just on having a member of your team actually look at your changes.

What we're really trying to do is add that visual part of code review and make it interactive and get all stakeholders on the same page before these issues hit your dev environment or your staging environment. Really, being able to review features in an encapsulated environment and isolation. Find those issues before they're merged in and really try to bridge the gap between engineering, design and product and really fill in the missing piece of the review cycle.

[SPONSOR MESSAGE]

**[00:16:22] JM:** Scaling a SQL cluster has historically been a difficult task. CockroachDB makes scaling your relational database much easier. CockroachDB is a distributed SQL database that makes it simple to build resilient, scalable applications quickly. CockroachDB is Postgres compatible, giving the same familiar SQL interface that database developers have used for years.

But unlike older databases, scaling with CockroachDB is handled within the database itself so you don't need to manage shards from your client application. Because the data is distributed, you won't lose data if a machine or data center goes down. CockroachDB is resilient and adaptable to any environment. You can host it on-prem, you can run it in a hybrid cloud and you can even deploy it across multiple clouds.

Some of the world's largest banks and massive online retailers and popular gaming platforms and developers from companies of all sizes trust CockroachDB with their most critical data. Sign up for a free 30-day trial and get a free T-shirt at cockroachlabs.com/sedaily.

Thanks to Cockroach Labs for being a sponsor, and nice work with CockroachDB.

[INTERVIEW CONTINUED]

**[00:17:45] JM:** So there're already a lot of tools in the development lifecycle. You've got Trello. You've you've got Slack. You've got GitHub. You've got all these different things that you're interacting with. Make the case for inserting another tool into that development process.

**[00:18:00] ES:** Sure. We were talking earlier about developer handoff. Right now, after designers handover to developers and developers are in their sprints, right now, there's no place for the rest of the stakeholders to actually give feedback, do their own testing, do their own verification of work that they're highly invested until it hits their staging environment. Usually, by the time that happens, those engineering teams are already on to the next sprint. They've already moved on.

So there's no place, there's not an actionable place in that workflow to give feedback when it matters, and engineers have that, right? Engineers have code review. Engineer specifically

have a time carved out in their development cycle to get feedback from the rest of their team. And that doesn't really exist for other stakeholders. So we're latching on a point in the cycle where there's already a pause, where we're already saying, "Hey, I want to take in feedback. I want this feedback now, because this is top of mind. I'm already working on it. What is the best way to get all the actionable feedback on that work?"

To do that, you need people to run it. Even for engineers, I mean, I'll code review something and then I'll end up having to stop my local development. I'll have to check out someone's branch. I'll end up starting a dev server on my local development to try it. I want to be able to visually test someone's work as in the same way that I want to read the code. So we're using time that's already set out to get feedback to bring more people, more stakeholders in and give really actionable feedback and keep the team on the same page.

**[00:19:34] JM:** Could you give a little bit more of a description for the workflow for how a release is turned into something to look at with FeaturePeek? I've got my updated CI file, or I've got a new building and it's ready to be released and observed in a pull request environment. How does that work?

**[00:19:57] ES:** Sure. For FeaturePeek, we have a one line, a step in your CI file, and we're listening to web hooks via GitHub. So when you put up a pull request, we listen to the web hooks. We know you've created a pull request and we wait for CI to run so we get pinged via your CI. Once that happens, we'll pull down your build either if it's a static site, which I know is not always a great term for what it is, but a static bundle or a Docker image. We support both. We'll pull down either your Docker image or your static bundle. We'll set that up and launch that. When it's up and ready and healthy, we will put a comment in your pull request. It takes a few seconds to do. Put a comment in your pull request. We can also ping your team in Slack if you have that setup. Some people have a Slack channel for their pull request. You can also say, "Hey, here's a future environment right here." You click it, it opens up this environment where you can just see it running. We'll have your pull request comments and notes right on the sidebar so you can see all the activity going and you could just leave comments right from there. You can create screenshots and videos right from there.

It's really as seamless as possible, and we specifically built it to be infrastructure-agnostic. You don't need to – We don't care if you're on AWS, if you're on Google Cloud. We've intentionally designed it to not be dependent on any of your own infrastructure, because we know teams change their infrastructure, teams change their deployment patterns. This is something that you set up once and forget about it. And then it's just always running. You'll get pinged when you put up a pull request and your team can just jump in and start checking out your changes.

**[00:21:30] JM:** How does the release cycle change with the addition of this pull request environment?

**[00:21:38] ES:** I would say the release cycle changes because environments actually get to be used for their intended purpose. If you have – Let's say you have dev staging and production model. In my experience, the development environment is where everything gets emerged directly into. After pull request, everything lands on dev.

Dev should be where you are fighting through integration issues. It should be kind of an integration hell. Integration warzone if you want to say. It should be where all these different changes have landed and now you're not actually troubleshooting the individual features or changes that you made. You're really troubleshooting issues that come up because the interoperability between them. So that's where you would have crashes because maybe someone removed a dependency or this is broken. It's really a place to resolve integration issues.

Staging then becomes purely for either automated QA or release QA or kind of like an external QA. It's specifically the just about to go to prod. We're just doing smoke tests there. We've already validated these features. We've worked out integration issues. It actually allows the staging environment to be almost prod, which is what it's intended to be. It allows you to move into a much more nimble release process where stable is supposed to be production and dev is actually used to work out developer issues between issues, and then you can review all of your features as they're being developed in deployment previous.

I'd say it moves some of the process up a step. But ultimately, it allows a much more functional process across each stage.

**[00:23:10] JM:** All right. Well, the idea of working with these poor request environments probably makes sense at this point. Let's talk a little bit about the engineering stack. What are you doing under the hood to facilitate the creation of these pull request environments?

**[00:23:29] ES:** Sure. On our backend, our main backend is anything that interacts in the user space is a Ruby on rails application. And then anything that deals with the environment specifically in terms of dispatching management, we're leveraging Kubernetes. And we have some custom code we've written to easily bring up environments and ingresses really quickly just to bring up these environments. Everything's name spaced. Everything is set up differently by – Is encapsulated by team. But we're running Kubernetes in the backend and then we manage that through our system. If you're a static asset, let's say you're not a Docker application, we will on-the-fly Dockerize your static bundle and deploy it.

**[00:24:09] JM:** That's pretty useful. I guess it's just worth taking it a moment in a time here. How do you think this would've worked pre- Kubernetes?

**[00:24:20] ES:** It's interesting, because there were some tools out there that kind of tried to do this pre- Kubernetes, and they were using like Docker Compose. I think pre- Kubernetes, post-Docker, or pre both.

**[00:24:34] JM:** Let's say pre both.

**[00:24:36] ES:** I think pre both, and it's funny, because when I was at Apple, we did some kind of hot swapping of environment sometimes in our internal tools. I hate to say it, but I think pre-Docker, we probably just would've been swizzling Symlinks on some drive somewhere if we had. I think we probably would have had a really long Enginx config that we wrote too dynamically, and it probably wouldn't be very stable and we would probably be kind of doing a lot of directory hopping and kind of organizing flat files on a drive to encapsulate it that way and we'd have to probably use a lot more monitoring and tooling to manage those, and especially from a security's perspective, it'd be a lot more challenging.

I think part of the reason we're doing this now is because DevOps technology has gotten really, really good and really manageable to be able to affect other parts of teams and in their workflows. I think the first kind of phase of Docker adoption in Kubernetes was look how much effort we can save in the DevOps perspective, or look at some of the things we gain from a release perspective. Now, I think that tooling has gotten so good that we can really start affecting teams that aren't just engineering by leveraging some of this technology.

**[00:25:43] JM:** Do you use any infrastructure tools or platform as service products that have surprised you and how useful they are?

**[00:25:54] ES:** I've used Google Cloud for Kubernetes for a while now. This is not my first project on that. So we're pretty set on that. We've actually moved a lot of stuff into a lot of tooling into Google's kind of ecosystem specifically. We don't use that many direct platform as a service stuff internally. I'm trying to think what I've been really impressed by in terms of tooling, tools that we've just use internally. I mean, we use Auth0, which we're really, really happy with. We use Skylight for performance monitoring that I've been very, very impressed with. But no, we do a lot of our specific like environment management. We've written a bunch of that. There are some really great products for Kubernetes out there. We're not necessary leveraging them right now.

**[00:26:39] JM:** And have you always been a Google cloud person or were you on AWS? Did you use AWS before that in the past?

**[00:26:44] ES:** The last startup I was at with my cofounder Jason, we were on AWS for a bit and then we wanted to move to Kubernetes, and Google's Kubernetes support was a little further along. I think, actually, we asked to be in the AWS beta, and they never got back to us, and we got impatient. And so we switched.

**[00:27:05] JM:** What's the hardest engineering problem that you've encountered while building FeaturePeek?

**[00:27:10] ES:** There are a few. I think when we started, there were a lot of different ways to kind of manage the environments and kind of leverage Kubernetes, and that's gotten – We've

really kind of  honed in on the right level of touch to kind of interact with this. Don't fight the system, but write your own kind of management stuff. I'd say that was one.

I think we definitely have spent a lot of time really trying to add some features that – Trying to leverage some of the technology to create features that we don't think other people necessarily would or could. For example, if you're using OAuth, you can stay logged in between feature environments on FeaturePeek. That was something that a customer asked, like, "Well every time I open up a FeaturePeek environment, I have to login again with OAuth," and we worked in to actually being able to persist that across environments, things like that.

I'd say also just being able to create our overlay and adding features to that has been one where we made some decisions, had to tweak a few things and then kind of build up from there. But at this point, it's really about, from a deployment standpoint, we feel pretty good about kind of getting there. But I think really figure out the right level of touch in how we interact with Kubernetes was probably the hardest thing to get right.

**[00:28:22] JM:** That overlay the you've mentioned, I think this is a key to describe in more detail. Can you explain what you mean by the overlay?

**[00:28:29] ES:** Sure. When you're looking at your page in a FeaturePeek environment, there's actually little button floating in the bottom left corner, just kind of hovering over the page. If you hover over it, you'll get a little button for a screenshot or a video. If you click it fully, it'll actually slide out.Like a drawer will slide out from the side and you'll see a list of a timeline, basically, of what's happening with your environment. So let's say Bill reviewed this and looked at the page, and he looked at the page three times in Chrome and then made this comment. You see a timeline just like you would in GitHub of comments, but you'll see it with activity.

So you can see, "Oh, someone viewed my page in all these different browsers, and someone left a comment and said this is a really bad issue, and they blocked the PR based on it." You can really get a full timeline of who's looking at your work. Who's reviewing it? And from there, in-line screenshots, you can add comments, you can file new issues to your bug trackers. You can add comments directly to the pull request. It's really this interactive pain overlaid on top of your page, literally hovering on top your page. That adds all these really powerful features.

That's where we really think the differentiator is between other deployment previews or other feature environment tooling is that we really think that a feature environment is useful, but if it's just a page that you're not going to file a good bug on top of or just get – A link isn't enough. You need to be able to do something really powerful on the page.

**[00:29:53] JM:** In building that overlay on the page, can you tell me how that's implemented? You just got this – Is it just you put the entire page in an iframe and then put the tray outside of that iframe?

**[00:30:06] ES:** Yeah. It's kind of a mix of an iframe. We serve all that content to the iframe, and then because we're running it on the same domain and every team gets their own unique domain. So that because we're on the same domain, we can send messages in and out of the frame – Sorry. Down into the frame. So we're communicating directly with the frame. The frame is communicating downwards. And then that overlay sits in the middle.

**[00:30:32] JM:** And what has been the product development process? When you are interacting with users, how do you understand better what to add to the product?

**[00:30:45] ES:** We've been really fortunate to have very vocal users, and a lot of them will just shoot us a message in Crisp and say – A lot of it is, "Hey, can I do this?" Or "This kind of bugs me," and they've been really – For example, this persistent login across environments. That was one that it was kind of just a complaint, like this bugs me. That every time I have to log in, that's kind of a barrier to me being able to test something efficiently and quickly. We're like, "Yeah, how the hell are we going to solve that? We went back and thought about it and workshop that for a while.

I think a lot of this stuff that we're working on now is really trying to put better feedback tools in front of people really trying to connect more with people's existing services. So we've rolled out Trello integration and Clubhouse integration. We're going to roll out Asana and JIRA, and really just meet people with the tools that they currently have and try to integrate with those. But a lot of it has been how are people using this. What do people like about this and really just trying to ask them and go back and brainstorm it.

I mean, originally, at least for the first couple months, especially before we had users – I mean, Jason and I were kind of just building this for ourselves. We were at a startup before this and we're looking for something that really did this. We were beyond frustrated that design and product would wait until – I think we shipped on Thursdays. And Wednesday nights, I knew not to make any plans. Not because the release wasn't done, but when I thought the releases was done, but I knew that we're just at 4:30 we are going to get an influx of lots a little nitpicky bugs and people finally looking at staging. I knew we are going to just have to call the team back and either fix everything or maybe ship a .1 the next couple days. It was just frustrating.

So we looked for a solution for this. For the first couple months, Jason and I were really just building what we wanted for ourselves. Then fortunately, when people started using it, they had a lot of really great feedback and a lot of great ideas on what to build next.

[SPONSOR MESSAGE]

**[00:32:53] JM:** JFrog Container Registry is a comprehensive registry that supports Docker containers and Helm chart repositories for your Kubernetes deployments. It supports not only local storage for your artifacts, but also proxying remote registries and repositories and virtual repositories to simplify configuration.

Use any number of Docker registries over the same backend providing quality gates and promotion between those different environments. Use JFrog Container Registry to search the custom metadata of the repositories. You can find out more about JFrog Container Registry by visiting softwareengineeringdaily.com/jfrog. That's softwareengineering.com/jfrog.

[INTERVIEW CONTINUED]

**[00:33:49] JM:** I think it's a great and useful product. I wonder what the next thing you can add to it. I mean, you got kind of the basics of what the user would want out of a pull request environment and out of a feedback-giving tool. What else could you build on top of it?

**[00:34:07] ES:** I think there's a lot. I think not only kind of integrating with other tools, but I want to be able to get closer to something like a Figma experience. I'd like to experiment more with

how people really like collaborating. I mean, I think especially now, we're learning so much because we knew that this would be valuable for teams that were remote or distributed, but we also know that it's very valuable for teams that sit in the same office, right?

But what we're really interested now is how are teams working together as they're remote? How is that process changing? How are power people collaborating with stakeholders as they're now moving to a remote situation? I think there's a lot there that we're still really interested in learning from, and I think there's still a long way to go in terms of kind of how can you do a more over the shoulder? Is this what you meant kind of review? I think we're getting close to that in terms of, "Here, use this and give feedback." But there's still a long way to go in terms of like really trying to make up for that distance gap that we're really interested in.

**[00:35:04] JM:** Any particular pain points you can identify?

**[00:35:07] ES:** I think we work really well right now specifically around asynchronous, right? If you're a developer working with a remote team, if you're in a different time zone, you can just get a bird's eye view of everything your team is working on and kind of just jump into each of those different environments and try them and see what's going on. I think that's really valuable.

I do think there's also some value in more synchronous communication. I think there's more value in getting closer to being able to integrate Figma designs and the running page more. I think there's a lot of different ways that we are interested in kind of expanding too. But right now, I think it's more about us just kind of learning from our users. I think we're still very – I mean, we're not very new, but I think we're still – I still think every time we have users really active on FeaturePeek and really passionate about it, they end up telling us something that we didn't know before. And so it's still very exciting, and I'd love to let them lead.

**[00:36:02] JM:** Are there any other changes to web development that you think are coming soon or have recently occurred that might affect your work on FeaturePeek?

**[00:36:14] ES:** I think the baseline of development in terms of – And I don't want to say the bare minimum, and I don't mean this to say that some of the like Jamstack frameworks or anything like that are simplifying the process. But I do know that the one thing I've noticed is that there

are a lot of – It's definitely shifting roles a little bit to have frontend engineers work very, very closely with design or even moonlight as design or vice versa, where the expectation of a designer is to use WebFlow and just do it all there and you don't have to care about it.

While I think that's very attractive to say, and it probably sounds good in an article about how development is changing, I can't see that really happening. I think people have complex enough applications. I think people are going to need to maintain their complex web applications. Not to mention legacy applications to be able to reduce everything into a template, to reduce everything into a WYSIWYG. I think people have tried before and I don't think it's necessarily going to happen. But I do think that certain tasks gets simpler, right? Gatsby has made certain things incredibly easy to do. I don't think that would necessarily change our product other than it's about us being able to support more diverse types of products, including simple applications to more complex applications.

**[00:37:37] JM:** On the front of Gatsby versus other frontend frameworks, like the NextJS, do you have any perspective there? How you differentiate them and how the prototypical Jamstack unfolds in the future?

**[00:37:54] ES:** When we talk about NextJS, I think that is way bigger than I would say – I consider NextJS and Gatsby pretty different just in terms of the level of complexity that people are building on top of Next is massive. I mean, we have a Next component to our app. I mean, people are doing some very serious heavy lifting there. If anything, I think that – And I think I heard you or heard someone say this in one of the episodes I was listening to. I'm trying to remember which one. But I think if anything, frontend engineers are just doing way more engineer because of some of the power of some of the tooling now, especially around Next.

I think that it's not that I necessarily differentiate. I think the whole idea of those are let's add some really powerful frameworks to build out the base case. If the base case get simpler, that's great. I think that the one thing that changes is if you have widespread adoption of certain patterns or certain frameworks, tooling will be made specifically for that. I think that's maybe a danger of just if everyone is using one thing and that's widely adopted, again, I don't – While I think Jamstack is very, very useful and it allows people to be very powerful by creating websites very quickly, web applications very quickly, you're always going to have more complicated use

cases. Again, the simple cases get simpler, the common cases get templatized, and that's great. But at the end of the day, there's still going to be heavy lifting that needs more dedicated resources.

**[00:39:31] JM:** What have you found in the Kubernetes ecosystem that's particularly lacking, or have you seen any pain points in the current Kubernetes ecosystem as of mid-late 2020?

**[00:39:45] ES:** I think, first of all, just in the few years that I've been using it, things have gotten considerably better. And I do think some of the tools around it – I mean, you have entire companies now just built on helping people get into Kubernetes. SO I think because of that, it's kind of like what we were talking about before. The base cases are going to get a lot easier, and the more complicated stuff maybe has some fuzz around the edges.

I think, for us specifically, ingress design and routing and more specialized routing is really lacking. I think people really assume very basic ingress path routing. And I think sometimes there's not as much support elsewhere for more complicated ingresses. But in general, the one thing I'm still seeing – The thing I've never seen anyone do, and there's actually an application now that's the closest I've seen. I'm on their mailing list, and I use them not fully not. I haven't completely switched over this being my kind of window into our clusters, but it's called Infra. And it's done the best I've seen about trying to really interact with your cluster and monitor your cluster, individual workloads and deployments. I'm pretty impressed by that just in terms of usability standpoint. But I actually think some of the introspection tools aren't that great. I mean, most people I know, and I do to, just end up using kubectl, and you make some batch aliases and you're on your way. But I think applications specifically around using monitoring and interacting with your cluster and your deployments, your workloads, that I've never been too impressed by. This one is close. It's the closest one so far.

**[00:41:24] JM:** And what's the division of labor among your team? How big is your team?

**[00:41:29] ES:** So we're four right now. The team is four. Fortunately, Jason and I are both technical cofounders, and then we brought on a backend infrastructure engineer in October. And then we have a former Segment enterprise sales person who's helping us out a little bit as well on the sales front. Yeah, four, three engineering, one sales.

**[00:41:50] JM:** What is the sales model for a product like this? I imagine it could be pretty good, because if you have a really big organization, everybody has to have access to it.

**[00:41:59] ES:** Yeah. I mean, the way I see us kind of approaching sales, yes, it works really well. It's the most powerful when your whole team is involved. I mean, what we've kind of scene is the first month, we'll have maybe an – Could an engineer all set it up? So we'll see an engineer come in, set it up, maybe invite one other person or just use it themselves. Then because once it's set up, it just runs. Everyone starts to kind of trickle in a few weeks in. They're like, "Oh! I've seen these links," and then you click on it and you join the team and start using it.

Usually, we see it kind of grow naturally as it's just already set up. For us, the most important thing for us is get set up and try it and then invite your team. Also, we've specifically designed our free tiers to really make it easy to jump in. There're two different free tiers. One is called FeaturePeek Indy, and it was basically our experiment cutting out all of the steps to get set up. Like what is the absolute fastest way that you can create a feature environment or a deployment preview? That's just a command line tool that as you're developing, let's say you're running a Gatsby site. Let's say you've built your static bundle locally. You just type peek, and it'll spit you back a link. You can share that link. And that's all free for public projects. Then our teams, free tier, is specifically for the open source community. Bbut any public project on GitHub, if it's public's, it's free to use. So it's really only paid for private projects.

**[00:43:23] JM:** That Indy Peek feature, so what does that do exactly? How does it work?

**[00:43:28] JM:** So we set up the Indy feature specifically so you don't – It doesn't listen to pull requests. The team's product, the main product listens to pull requests. It's a GitHub app that you install and you put our little call, our one-liner in your CI pipeline. The command line tool is for really just kind of Indy developers as you're developing. It's a command line tool. You install it with Homebrew, and you just type Peek as you're developing. You build it locally. Like you run yarn build or whatever, what build command is and then you just run Peek and it'll upload those assets to us, spit you back a link and you can copy that link and share it and invite people to your team that way. It's creating an environment based on a commit instead of a pull request.

We're really excited about that. I mean, we think that was – It was really an experiment for us to cut out some of these steps. But we really think – And we are talking about kind of the sales process being interesting for us in terms of spreading across the team, but we also think the most important thing is just having people use it. So we've specifically designed it to – We want people working on open source projects to try out. We want people developing their own personal projects to try it. People who are in engineering boot camps who are learning JavaScript, making their own web applications to just be able to try it for free. There're a lot of options to just jump in and try it.

**[00:44:49] JM:** That's great. Is there anything else you want to add about FeaturePeek and what you built?

**[00:44:54] ES:** I think we really want teams to collaborate better and review better. We think that the review process, especially for teams now more than ever, as teams are moving remote, as teams are distributed, as you have your designer in Austin and you have some engineers in San Francisco and maybe abroad, we really think that there's so many ways to improve that feedback and review process. Yeah, we're really excited about it. We're having a great time building it, and it's always fun to build something that you originally wanted for yourself. You don't get a lot of opportunities to do that. So it's been really fun.

**[00:45:28] JM:** Eric Silverman, thanks for coming on the show. It's been great talking.

**[00:45:30] ES:** Thanks for having me.

[END OF INTERVIEW]

**[00:45:40] JM:** Apache Cassandra is an open source distributed database that was first created to meet the scalability and availability needs of Facebook, Amazon and Google. In previous episodes of Software Engineering Daily we have covered Cassandra's architecture and its benefits, and we're happy to have DataStax, the largest contributed to the Cassandra project since day one as a sponsor of Software Engineering Daily.

DataStax provides DataStax Enterprise, a powerful distribution of Cassandra created by the team that has contributed the most to Cassandra. DataStax Enterprise enables teams to develop faster, scale further, achieve operational simplicity, ensure enterprise security and run mixed workloads that work with the latest graph, search and analytics technology all running across hybrid and multi-cloud infrastructure.

More than 400 companies including Cisco, Capital One, and eBay run DataStax to modernize their database infrastructure, improve scalability and security, and deliver on projects such as customer analytics, IoT and e-commerce. To learn more about Apache Cassandra and DataStax Enterprise, go to datastax.com/sedaily. That's DataStax with an X, D-A-T-A-S-T-A-X, @datastax.com/sedaily.

Thank you to DataStax for being a sponsor of Software Engineering Daily. It’s a great honor to have DataStax as a sponsor, and you can go to datastax.com/sedaily to learn more.

[END]