EPISODE 1119

[INTRODUCTION]

[00:00:00] JM: Every software company works off of several different development environments. At the very least, there is staging, testing, and production. Every push to staging can be spun up as an application to be explored, tinkered with, and tested. These ad hoc spin ups are known as release apps. A release app is an environment for engineers to play with and potentially throw away or promote to production. Release apps have been made easier due to technology such as infrastructure as code, continuous integration, and Kubernetes.

Tommy McClung is the Co-founder of Release app, a company which makes it easy to spin up release environments for your software. Tommy joins the show to discuss release workflows and his work building Release app.

We're looking for writers for Software Engineering Daily. If you are interested in writing or contributing research material for Software Engineering Daily, send me an email, jeff@softwareengineeringdaily.com. I'm also looking for investments in infrastructure or developer tooling companies. You can send me an email also if you are somebody building a company, jeff@softwareengineeringdaily.com. I'd love to hear from you.

Are you spending too much time debugging in production? If you are, the days of endless logging and debugging sessions should be long gone. You can give your engineers 48% of their time back and fetch data instantly from production systems with Rookout. To check it out, visit rookout.com/sedaily. That's rookout.com/sedaily.

[INTERVIEW]

[00:01:37] JM: Tommy, welcome to the show.

[00:01:39] TM: Thank you.

[00:01:40] JM: You work on release app.io or a company called Release, and it's a system for developing release environments or developing staging environments. Can you explain why it's necessary to have staging environments?

[00:01:58] TM: Sure, yes. I've been building technology for around 20 years back to the early days of when blade servers. That's when I started was back in the blade server days. If you're old enough to remember that, you'd rack up a whole bunch of servers in a little tiny rack in a data center, and your job was to make those servers as useful to developers as humanly possible, and the idea always has become like how do you create these environments for developers prior to releasing to production that replicate what production actually use. For the last 20 years, there's been various flavors of this starting from blade servers to VMs to now with Docker and Kubernetes. Getting developers access to environments that look like production helps them produce higher quality releases, better code, fix bugs, get their software in a state that when they release it to production, it's going to perform and be as bug-free as humanly possible.

I think the other thing that's happening today is as applications are getting more and more complex, building environments is becoming more and more challenging. Production is the ultimate output of all of this, right? You have a prod environment. Your users go and they experience your side or your app. But the preproduction environments, getting those to replicate your external production site is incredibly challenging, and this is something that we felt just recently. I was prior to starting Release. I was the Chief Technology Officer at a company by the name of TrueCar, and we had about 250 engineers, and replicating environments for developers was incredibly challenging.

That's where we really felt the pain was having staging environments on demand really for the developers so that they all didn't have to share a single environment, and that's generally what you see even to this day is that most companies kind of rely on a single staging environment where all their code from their mainline kind of makes its way to before it goes to production. When you only have one of those and as your engineering team grows, it becomes a place for a bottleneck. So we believe that you should have an environment that's as production replica as humanly possible created every time a developer does a pull request, and it should just happen. It doesn't need to be something that you have a big DevOps team going off and creating. It should just be kind of happening behind the scenes. So if a developer needs to test something that looks like production, it just happens behind the scenes.

[00:04:34] JM: Now, this is something that Kubernetes has made a lot easier presumably. Kubernetes is my platform for very easily spinning up new containerized environments. Why would I need an entire product? Why would I need an entire platform, a SaaS solution for this kind of thing when Kubernetes does it so well or I could use ECS or AKS, one of these containerized spin up environments.

[00:05:03] TM: Sure. It’s kind of the difference between rolling it yourself and buying something off the shelf, right? Kubernetes, we’re actually using Kubernetes behind the scenes to orchestrate this, so obviously that's a big part of the reason we can do what we’re doing, and I would argue that a lot of products in the world today don't need Kubernetes for their production environments. It's a lot of configuration. You likely need to have a person or a team that's configuring and managing it in order to get the benefits of Kubernetes and in order to build this kind of on-demand environment creation. That’s a whole bunch of tooling internally that you would have to build on top of Kubernetes, which as most people know isn't the simplest thing to operate on. That's why there tends to be a team of people kind of running the Kubernetes clusters.

We tend to find success with customers who haven't yet jumped into a full-fledged Kubernetes environment, have Dockerized. They’re running Docker locally for developments. They may push out to ECS when they go into production but their environments preproduction are still kind of in the state of like we still have a single staging environment. We haven't jumped into Kubernetes yet. The investment in the Kubernetes is a big one. If you've ever spent time setting up a Kubernetes cluster, even for the simplest apps, it’s a configuration nightmare inside [inaudible 00:06:28]. The debugging and the output isn’t – When something isn't working right, isn't great. So you’re going to have to put a pretty big time investment into making it work. Our thesis here is if we can make that pretty turnkey, you would get the benefits of Kubernetes in these preproduction, kind of on-demand ephemeral environments without having to become organizationally like a Kubernetes expert.

The larger organizations who have like a Kubernetes team, they may be rolling something like this on their own. Over time, we feel like organizations are going to realize like, "Hey, we're spending a lot of time and energy in this. If there's something off the shelf we can use, we'll jump into this when the product is mature enough to support our use case." The other thing I would say is that we're not just dealing with Kubernetes. I think the other preproduction issue that you run into is Kubernetes is kind of like where your applications live but you have data. You have native cloud services that you have to deal with, whether that's no Kinesis or Lambdas, RDS or any databases you might be using. Kubernetes deals with kind of the application container tier, but orchestrating ephemeral environments that include all of those things is a pretty big challenge, which again requires people to go tool that internally. So we look at the entire kind of ecosystem and are working towards building ephemeral environments that don't just include containers but also include native services as well.

[00:07:51] JM: Why is that difficult? Why is it difficult to integrate these native services into ad hoc pull request environments?

[00:08:00] TM: A lot of it is namespacing, and so whenever you have a portal – Let's say you have a branch feature A that you're working on. What you really want is an entire isolated ecosystem that just contains feature A. So that branch, you get a pull request inside of Git or Bitbucket or GitLab. You want that then to flow all the way through, including kind of the URLs, the DNS entries, the namespace inside of Kubernetes and even the native services. So ensuring that you've created an isolated self-contained ecosystem with all of the various different pieces put together is a Kubernetes issue for sure where you've got to get the namespacing correct and you have to get the ingress rules correct. You may have to have an Nginx proxy included that's directing traffic where it needs to go based on the names of all the containers that come up that are unique based on that feature branch. But then you also have to connect that into all the various different AWS services that you might be using so that your environment can connect to all the services behind the scenes.

For instance, RDS, you might want to have a copy or a snapshot of production or a C data set that comes up dedicated to just that environment, and that needs to be accessible by that environment through environment variables to the right URLs. So there's just a time of orchestration that has to go on to connect all the dots. If all you're trying to do is bring up a

Kubernetes cluster in the namespace, yeah, sure. You can probably go do it. But when you want to make it look like a replica of production, there's a lot of moving parts behind the scenes that you have to – Essentially, if you're doing this yourself, you'll have to script up and tool on your own.

[00:09:40] JM: The space that you're working in, it's funny because I'm sure you know this was a feature of Heroku for many years before you started a company around it, and I think that it was probably easier for – So if you deploy your application in Heroku, you can have these staging that you can integrate with GitHub and at these staging environments for every new pull request in GitHub. My sense is that that was easier for Heroku to do because Heroku had strictures around how you're going to be deploying your application. If you do it in the Heroku fashion, therefore it's easier for them to put constraints around release environments that could happen. Do you have any perspective for why Heroku was able to do this for so long before a few companies came up around doing what you're doing?

[00:10:29] TM: Yeah. I think you hit the nail on the head there. Within the Heroku ecosystem, they've put pretty well-defined parameters around what can run in Heroku, right? Prescriptive environments become a lot easier there because they have all of the control over everything that's going on behind the scenes. We've actually had some customers who were on Heroku, so I think this is a good differentiating point of like where do you kind of grow out of Heroku and what do you do, right? If you're on Heroku and you start getting more complex applications where you might have multiple micro services, static front ends, your application is starting to grow outside of what kind of is simple in Heroku. But you tend to really like the features, and these review apps are one of those things that I think a lot of users took for granted within Heroku that's like, "Hey, these things just kind of can happen."

But as soon as they start to grow out of that, whether they want to start containerizing, they start wanting to potentially move into Kubernetes and start becoming a more kind of complicated application where they might be using native AWS services that are difficult to kind of replicate within the Heroku environment. This is where we see that the opportunity exists because Heroku doesn't fit all needs. But the features within Heroku and review apps tend to be one that users kind of liked quite a bit, so we actually looked at that as an opportunity to kind of go tackle the next generation of more turnkey kind of platforms called a platform as a service if you want to, where you can have complex multiservice, microservice applications that get some of those same features that you might get within Heroku, and review apps is one of those.

We felt this need in a larger organization for ourselves where our application like TrueCar could not fit within a Heroku environment. It just had too many moving parts. We had Hadoop clusters. We had all sorts of stuff going on there. When we tried to kind of look for products off the shelf that would solve this, there really wasn't anything that was full stack. You had Heroku that would deal with kind of small applications, medium-size applications. Then you had frontend review apps, kind of the Netlifys and Vercel [inaudible 00:12:43] or whatever you want to call it now where they were tackling the front ends. But if I wanted an ephemeral environment that went from frontend all the way down to native services in AWS, the only solution at the time was to go build this yourself. So we see that as the opportunity which is to kind of take that full stack from end to end, which you can't really fit within the Heroku world.

[00:13:06] JM: Indeed. Just to take the business perspective for a moment, there are all these different ways that people are dealing with their staging environments. There are these different Git workflows where like I push to Git and then it spins up a release environment automatically on Kubernetes. Then I promote it, I like it, and there's that. We talked to a company, FeaturePeek, recently that kind of does what you're doing, but they also spin up in this environment that makes it very easy to share. I just want to know about this from a couple perspectives. One, is it too busy? Are there too many different people to actually have a differentiated enough solution to stand out to people? Two, can you actually cover all the bases that people need out of release environments?

[00:13:55] TM: Good questions. I think the fact that there are multiple companies kind of going after this space means that there's definitely a need in the market. I think that's the first thing that I think. Everybody's kind of tackling it from a different angle. You have kind of more frontend focused review app environments, which the FeaturePeeks and kind of Netlify and Vercel. They're kind of going after it from that perspective. Then you have companies like us that think that like, "Hey, if you're really going to solve this, you do have to tackle what you said, which is how do you cover the bases of all the things people are going to need?" Obviously, you can't do that from the beginning, right? You can't say, "Hey, we're going to solve the world's problems and every single kind of service that everybody needs right from the beginning." But there are common problems that you tend to come up over kind of repeatedly, and one of those is data, right? Staging environments, if you just have the applications up and running without representative data of what's actually happening in production, you kind of have a staging environment that is almost good enough to replicate what's in production but not quite there.

So we actually think that data is a really, really big problem in the space that all of these are kind of like not attacking directly. If I want to have an environment that looks like production, you can test in production, and there's companies out there like feature flagging companies and [inaudible 00:15:17] that are kind of coming at it from that perspective. But if you are an organization that believes that, “Hey, we got to get our stuff ready before users see it.” If you want to have a staging environment that replicates prod, you’ve got to have production like data. So I think that data is kind of a key piece of this that when I look at the space, sure, there are lots of players in it, but I think the company that’s going to unlock this is really going to solve the data side of it because I think the tooling around like GitHub and GitHub Actions and being able to create containerized environments within Kubernetes, it’s worked. It’s definitely worked. It’s not rocket science and it can be done, but really solving how an environment looks like production so that developers trust it to kind of be a representative of an environment that they can trust so that when I go to prod, it looks like prod.

I think from a business perspective, that's the opportunity that we see is going end to end, frontends all the way down to data sets where we even have customers that are running Hadoop clusters that were helping them create environments out of. I think the challenge is like picking which services matter, right? If you go look at AWS and you say, “Hey, we need to cover everything that they have or GCP and everything that they have,” you’ll chase your tail for a long time. But there are common problems in the space that keep coming up over and over again, and data is a really big one of those.

[00:16:46] JM: Yeah, data. I mean, the thing about data is if you want to replicate a pull request in a data environment, you may need to replicate the entire data lake or you could decide to read from the data Lake but there's problems there. You might actually – If end up writing back to the data lake, it could be a problem, so maybe you want to give this pull request environment only read access and not write access. What are the challenges that come from developing release environments around large data applications?

[00:17:19] TM: I think you hit a couple of them, and we actually recommend not trying to share data environments because of exactly the reasons that you spoke about. As soon as you start saying like, “Hey, I need to only give read access to an environment,” you're not truly giving it the functionality that looks like production. We actually tend to leverage snapshots quite a bit, and that's one of the nice things that AWS provides with most of their data sets is kind like these nightly snapshots that are automatically created for your RDS instances, and we support RDS today, and we’re going to expand this in the future. But those snapshots actually become really good either starting points for obfuscated C data. Or depending on your organization, like just direct snapshots that are exactly prod replicas.

We have a feature called instant data sets which we create those kind of ahead of time. So every night, when a snapshot is created, we will spin up a set of kind of ready and waiting RDS instances so that when your environment comes up, it can just check it out and instantaneously it has a production or seed, obfuscated production replica, data so that you can instantaneously have access to really, really large data sets.

Now, of course, there’s a cost trade-off that goes with this, so you're kind of trading off speed versus cost. If you want ready access for 100 developers, you’re going to have to have a queue of ready and waiting databases. But if you really, really want to move fast and you want your developers to have access to production like data, we have a solution that creates these kind of ready and waiting instant data sets that you leverage some of the things that the AWS already does with snapshots built-in. So we think that there are a lot of things in and around the space of data, extending that outside of just kind of databases, into data lakes like you spoke about where creating those kind of replica environments when you can is the best way to go about it. If you can’t, then you have to have rules around, “Hey, are these read-only? Are these shared data sets? Can they be write-only?”

We’ve seen customers struggle with this, where you have a single staging environment that's connected to some prod replica data, and a lot of front ends use that to kind of point back to data sets, and they stomp on each other's toes all the time. All of these things are challenges that we think need to be solved in order to kind of solve this staging environment as a service, a solution kind of ubiquitously.

[SPONSOR MESSAGE]

[00:19:55] JM: A lot has changed since the early days of the data warehouse. There's been so much change, and data teams today effectively manage many thousands of data tools and processes, and want to avoid being captured by old technology. Since 2006, TimeXtender has been helping companies build modern data estates with their data management platform. TimeXtender is software that can be deployed on premises or in the cloud to capture and organize the business logic required to build and operate a modern data estate. TimeXtender makes it possible to easily add new data sources, combine sources from different systems, optimize transformations and loads, and serve clean and timely data for analytics, automated intelligence or machine learning. It's now on version 20. TimeXtender works seamlessly with all Microsoft data tools including Azure Synapse, Azure Data Factory, Power BI, and of course Microsoft SQL Server. Through its close relationship with Microsoft, TimeXtender builds new capabilities into its platform as they are introduced by Microsoft. You can take advantage of them on release day.

If you'd like to see TimeXtender in action, you can sign up for a demo and a free trial at timextender.com/sedaily. Software Engineering Daily customers will get unlimited access to TimeXtender's solution specialists, as well as a free two-day proof of concept. In two days, you can have a running prototype of your new modern data estate so that your organization can see the real benefits of automating data warehouse operations. Go to timextender.com/sedaily. That's T-I-M-E-X-T-E-N-D-E-R/sedaily.

[INTERVIEW CONTINUED]

[00:21:40] JM: Okay, let's talk a little bit about what Release actually is. It is an actual app. Can you explain what the Release app does for you?

[00:21:52] TM: Sure. I mean, simply put, it creates an environment on every single pull request, right? In your source control system, whenever you do a PR, we will behind the scenes automatically create a pull request environment with a URL that tracks the branch that you are working on. Let's say you're working on feature A. You do a pull request. An entire environment with all of the services, a snapshot of production date if you want it will up a containerized application. Let's say you have 5, 10, 15, 20 Docker containers. We will spin those up inside of a Kubernetes namespace. As a customer, you don't have to know anything about Kubernetes. This is kind of behind the scenes how we're doing it. But it will create all of the internetworking connections. It will make sure that there are public facing URLs for the services that need to be exposed to the Internet. We will provide environment variables that define kind of all of the custom networking host parameters that you need.

So when you have a PR within GitHub, you'll have a link that gets sent to you right in the comments inside of GitHub that says, "Hey, your review environment is created. Click on the link." Then you have standalone version of your entire ecosystem at your ready. The way that we're seeing customers use this is a couple of things. One is they're using it to kind of debug complicated problems that only arise when you're connected to production data. The other thing that we're seeing it used for is kind of the product manager, the designer, the team that's working on a collective kind of project using these environments as their review environment. It's somewhat like you talked about in Heroku but you have full access into that ecosystem, and we provide this to our professional plan and above users. We actually use K9s, which is an incredibly great tool. It's a plug for K9s where we automatically allow the developers kind of to be able to get at their running environments, console logs. Being able to shell directly into those containers, and it's a standalone environment that just tracks that entire branch. You're completely isolated from every other change and you get to make sure that your change works. As soon as you merge that into your mainline branch, that environment just disappears. It automatically gets destroyed, saves resources. These are ephemeral in that way where they spin up and spin down.

[00:24:22 JM: Gotcha. You mentioned namespaces as being something that's particularly difficult to engineer around. Tell me about building the infrastructure to namespace a sample environment properly.

[00:24:37] TM: Yes. A lot of it is making sure that your Kubernetes manifests are done properly. But it starts with kind of architecting all the way from what are the external services that you're going to expose all the way down through how does that make its way into the Kubernetes cluster itself and into the AWS services that you might be leveraging. We tend to kind of tag the branches or, sorry, the environments based on the branch that you're building. It might be, like I said, feature A. That then becomes kind of the anchor point for all the rest of the services in the name spaces, and environment variables tend to be the very most kind of effective way to kind of communicate amongst all of these different components. We'll create all of the environment variables that are automatically generated from all the ingress rules that get created so that as you're talking into it externally or container point ports and node ports within the cluster, all of those host names have to be consistent inside of that namespace so that all the services can communicate within the Kubernetes cluster and then outward into the AWS ecosystem as well.

I think the other part of that that was particularly tricky was you how does that flow all the way from kind of pull request, getting it back into where the developers are, the GitHub kind of side of things so that when an environment is created, it's very clear that, hey, this is a branch for this feature and that this developer is isolated within the namespace. So those namespaces can't talk to each other. If I have feature A, feature B, and feature C, they might all be sitting inside of a Kubernetes cluster. All of those things have to be isolated from each other. We tend to just lean on what Kubernetes did there from namespacing and security rules inside of Kubernetes. But that architecture, kind of keeping the namespacing consistent across all the services are things that we had to tackle in order to pull this off.

[00:26:36] JM: Is there anything that you've had to do to make it fast and responsive? Because I think making it buttery smooth is something that's pretty important, just given that we are competing with these workflows that people have stitched together or ad hoc staging environments, and it's always five clicks and three pull requests and a shell script around or something like that. Anything you've had to do to make it faster?

[00:27:04] TM: Yeah. I think one of things are build. Technically, Docker has some really great features built into their build system with Docker builds being able to cash layers and making sure that that's done very well. Frontends are a good example of where this always bites people. Your node modules, if you end up building that over and over and over again, it's just really, really slow to do that, so being really smart about how you use the Docker cashing. Then also inside of the Kubernetes subsystem is a whole Docker build kind of engine sitting in there. Our builds run within the Kubernetes cluster that the customer is running on which makes it so that cashing layer works on behalf of the user incredibly well. We want to have like the fastest builds humanly possible so that when these environments are created, the builds themselves aren't the bottleneck which tend to be the case in a lot of these systems. You got to wait 30 minutes for your build to finish. Then your environment can get spun up. By that time, maybe you've lost interest.

We've spent a lot of time really in and around the Docker build and Kubernetes build subsystem, making sure those builds are incredibly fast, and we're leveraging things like ECR so that we've got a really fast access inside of EKS subsystem where we're getting our images incredibly fast. They pull down very quickly because I think at the end of the day, like you said, this just needs to be seamless, right? If a developer does a pull request, that should take at most minutes for that environment to be created so that instantaneously when that pull request is created, that environment kind of spins up on the other side. It tends to be how you deal with cashing along this kind of chain of things that happen. So you do a build, you cash that build. Within the Docker subsystem, you're leveraging what's happening there. Within the Kubernetes subsystem, you're leveraging all of the Docker build kind of functionality that lives underneath the covers there. Then cashing those images within EK, sorry, the ECR repository so that when those images get pulled again, it's incredibly fast. That streamlined layer of functionality is as fast as humanly possible.

[00:29:17] JM: Among all the engineering teams you've worked with, the pull request environments that you've had to work with, people on spinning up, is there anything else you've seen that's surprised you? You mentioned the data stack, the Hadoop type of environments that you've had to help people with. Anything that's thrown a wrench into your app?

[00:29:38] TM: Actually, every customer is unique. I think that's the thing that has been most surprising to me. It's incredible the differences in the way people build software. The way that Docker gets used in real life, it's not as pretty as the tutorials. A lot of times, there's a lot of messy infrastructure, a lot of messy uses of technology that when you're trying to build a generic platform that anybody can use, you have to deal with that stuff, right? I think ultimately the most surprising thing has been how do these environments kind of tend to like need to be created in and around what the customer already had.

We had an initial assumption that Docker Compose was going to be a really good starting point for users, right? Docker Compose, if you're going to look at those files, they're kind of a blueprint to what they want in their environment but they're usually built not with a running environment in mind. They're usually built with kind of a local kind of development ecosystem in mind. Then when you try to translate that into Kubernetes, there's a ton of stuff that just does not work. You have networking differences. You have kind of command differences between what Docker commands and Kubernetes commands deal with.

Translating from the world of kind of Docker and Docker Compose into Kubernetes and then coupling that with all the various different ways that people are leveraging all of those has been an incredibly challenging problem. I think a lot of times what you find is that any company can kind of sit down and put a DevOps team together, kind of home grow a solution to this that fits just for them because they know all the nuances of their Docker ecosystem, their applications, their networking. When you try to build a platform that's generic that will kind of consume anybody's application and turning it into a running environment, you have to deal with all of these permutations.

For us, I think that's been the most complex and challenging piece of this is really surprised at how much difference there are in everyone's Docker subsystem and ecosystem that you think Docker is kind this one-size-fits-all thing. It's really not. People use this in amazingly different ways across organizations.

[00:31:57] JM: What about the integrations with Slack, the integrations with GitHub? Have you seen anything in the integration side of things that has surprised you or been particularly difficult to implement?

[00:32:08] TM: Yeah, a couple of things. The GitHub integration is an interesting one, because we want to be right within the pull request that the user is doing within their normal workflow. The status updates in GitHub give us fits, and what I mean by that is we want to post a status message into GitHub when a build starts, a deployment starts, and an environment has been created, and all of those various different kind of status updates like have to be done incredibly timely and correctly because depending on a user's GitHub settings, they might have it so that if a status check fails, the PR can't pass and you can't promote to production. If our build for whatever reason the status doesn't come back correctly or it comes back as an error when it actually finished, all of those kind of integrations into status connections, into GitHub PRs are finicky.

I don’t know if that’s a GitHub issue or a release issue, but it's been a pretty tricky kind of integration to get solved, and I think it's important though because of this needs to be pretty seamless with what the user kind of – The end user being a developer, it needs to be seamless. So those status updates are incredibly important, and we've just seen that getting them correct and right has been an incredibly difficult problem technically because we’re kind of having to go down into Kubernetes. Use kubectl to get status information on the Kubernetes cluster, pull that out, parse it, get it into the format that it needs to look like in order to surface it back up to kind of the GitHub workflow. Making sure that that happens with the correct data at the right time and round tripping has been very, very interesting and difficult to deal with.

On the Slack side, that one was pretty straightforward and easy. We just use that to kind of post updates when PR environments are ready. There's not too much magic on that one, but the GitHub and the source control integrations obviously being where the developers are are kind of the trickiest places to deal with. If you’re pulling this on your own, like you got to figure out all this stuff. That’s part of the other answer to your original question, which is why shouldn't people just build this on their own. If you’re really going to do this and make it seamless for your developer, you’ve got to tackle all of the problems that I've been talking about end to end, and so it’s a really big project to pull off.

[00:34:23] JM: Your infrastructure itself is something I wonder about. You’ve got this app that I can use to spin up staging environments, which is great. But I imagine you've got some stuff going on the backend, maybe some lambda functions that are orchestrating things, or maybe you’ve just got a long running container service that's a virtual machine that's running your program to orchestrate the spinning up of these different apps. Tell me about what your backend actually has.

[00:34:52] TM: We build release with release. The entire backend is on an EKS cluster, so we actually run in a production environment within the release ecosystem, which you can do. We don't promote that much. But if you want to as a customer, you could run your production workloads through us all the way through kind of end-to-end CICD. But we run in Rails. Kind of our backend service is Rails, just because we've been doing Rails since like 2006 OS developers. We know it really, really well. That's what kind of hosts the app, the APIs, like all the services that the front end connects to.

Then on the backend, it's all running within containers that we have running inside of our own namespace for Release that were – Containers come up and down. It's just like any other Kubernetes ecosystem, as far as like our backend infrastructure is concerned. We do have to connect to the Kubernetes clusters directly, right? One of the things that we do is we'll create – When a customer comes on, we actually spin up an EKS cluster for them to run Release inside of, and we actually run it within their AWS account. When you sign up for an account and you upgrade to our professional plan, we're actually going to create an EKS cluster within your AWS account where all of your release environments live. The reason we do that is so that those environments can have access to other AWS native services that that application might need to leverage.

Again, the complexity of everyone's application kind of warrants having these not running within like a hosted environment like you would with Heroku, but what you're doing is you're pushing basically the compute engine out to EKS within their AWS account. We're remotely kind of controlling and dealing with laying down containers. When a PR environment spins up, we make sure the namespace and everything is correct. The URLs are correct. We update DNS entries, but those containers end up running within an EKS cluster within our customer's AWS account, which then really gives them the ability to go off and connect to all of their own internal AWS services, whether you've got lambdas, whatever it might be.

I think that one of the trickiest parts of what we're building is kind of the central hub managing lots and lots of remote EKS clusters. I think we've had to do a lot of things with cueing. We use sidekick to kind of fire off build jobs and deployment jobs and making sure we have consistency throughout the ecosystem. Audit trailing, making sure we know every single thing that's going on within all of these clusters. There's a lot of technology behind the scenes to kind of make sure we always know what happened and also so that we can roll back in the event that something did occur a customer didn't want, you can click a button and roll back to a previous version of what you might want to try to do there.

Technically, we’re kind of trying to leverage as much Kubernetes as we can but realizing that the customer isn’t going to only be living within a container environment. Applications now are spread across containers and native AWS GCP services, and it has to fit within that ecosystem.

[SPONSOR MESSAGE]

[00:38:11] JM: GitLab Commit has gone virtual. You can join the virtual conference on August 26th, 2020 for a completely free event filled with practical DevOps strategies shared by leaders and development, operations and security, and there’s more than 60 speakers from more than 20 industries. You’ll hear innovative stories from Microsoft, T-Mobile, VMware, State Farm, the US Air Force. Attendees of all experience levels will find opportunities to network and engage in a community where everyone can contribute. Register today at softwareengineeringdaily.com/gitlabcommit. That’s softwareengineeringdaily.com/gitlabcommit.

[INTERVIEW CONTINUED]

[00:38:54] JM: Do you see this becoming a platform as a service that is stretching beyond the just like ad hoc staging environments? Do you see this eventually trying to get into the production serving kind of area?

[00:39:14] TM: I think that's going to be kind of a case-by-case basis. We definitely see that it can be done because we’re doing it ourselves, right? We’re running Release with Release in production, and we actually have a couple of customers that are running production workloads as well. That being said, it isn't where we tend to find the pain point today. The pain point today is like, “Hey, how do I create kind of on-demand environments for my developers?” We think that that problem is huge, and it’s only going to continue to get harder as applications get more and more complex. What the pattern that we've seen is that when somebody comes onto Release for staging in ephemeral pole request environments, they see how simple it is to do that and they say, “Hey, why can't we run our production workloads this way as well?” For a couple customers, we've done that and we’re talking about whether or not that's kind of like a future direction for us. But I think that, yes, the answer is that we do kind of see a world where that can happen. But today, we’re just kind of laser focused on the staging problem.

[00:40:11] JM: Yeah. Because, I mean, one issue I can imagine with the actual production environment is I assume you have to scale up at some point. I mean, if your service gets busy, you'd want to scale it up. If you're just spinning up staging environments, these things just – You only need one of them and it gets torn down eventually. If you're doing a production, you would actually have to solve for the case of scalability.

[00:40:34] TM: There's a lot less problems to solve. Well, different problems. I wouldn't say less on the ephemeral kind of pull request environments. Yeah, you're right. Production monitoring and alerting up time guarantees, making sure that you have scalability, which EKS solves a lot of that for you but you still have to operate it and there's – We've learned a lot about Kubernetes in this kind of buildout of what we're doing. Just because Kubernetes does it, it doesn't mean it's easy, right? You have the ability to set the number of replicas. You can set how much do I want this to auto scale, and we'll even do this on a ephemeral kind of environments as well. A customer – When we set up any EKS cluster, we might say, "Hey, set that cluster to auto-scaling so that if they get more and more developers creating pull requests, it's automatically in a spin up the cluster size and then spin it down maybe during the middle of the night or when they aren't using it as much.

But you tend to have other issues with pods needing to move nodes, like all sorts of things that you have to deal with, even just in the ephemeral side of it that, yeah, EKS can do the production side of it but a production-ready kind of Kubernetes cluster is very different than a kind of staging environment ephemeral. We get a lot more forgiveness, right? If the staging environment or one of your ephemeral environments doesn't work for some reason, that would be a lot less problematic than, "Hey, my production cluster is not working, and what did you guys do to it?" We're kind of like really focused on the staging side. We see the world where production could be very interesting and we're kind of dogfooding that for ourselves. We're running, like I said, Release inside of Release right now, so we're kind of figuring out all of those things. If it makes sense, we may end up releasing some stuff over there as well.

[00:42:19] JM: What has been the hardest engineering problem you've had to solve in building Release?

[00:42:24] TM: I think it goes back to kind of the vast number of ways that people build out their technology and I think that will continue to be the challenge in this space for the foreseeable future. When you start looking at, first of all, like how people use Docker, right? That one itself like, again, was super shocking to me. But even just making sure that we support all of the various different ways that Docker and Docker Compose can be used, that problem is kind of like chasing your tail. You'll come across – Basically every new customer we bring on, we learn something about Docker and Docker Compose that we did not know before. Build ARGs, config maps, how those have to flow into Kubernetes, how you can do volumes, how the commands that you can use within Docker make their way into Kubernetes. All of those things are super challenging.

People have said, "Well, why don't you –" There's this open-source project that's actually really good called Kompose that starts with a K which takes Docker Compose and kind of brings it into Kubernetes manifest, and people always ask us like, "Why don't you guys use that, and the answer to that is because we're trying to build this kind of very special purpose, end-to-end pull requests all the way to kind of Kubernetes with the namespacing and the routing with DNS entries and all of those things done. It just didn't quite fit, but there are a lot of things that you can learn by looking at that stuff because if you go look at the Kompose with a K codebase, you'll see like the various number of permutations that had to be solved for to get that project to go. We have to solve for that, plus we have to translate it all into kind of this ephemeral environment universe within Kubernetes.

That technically has been a pretty big challenge for us, which I think – Then you start extending that beyond just like Docker containers into, "Hey! Now everybody –" How do they do RDS? How do they do lambdas? How do they do all of these various different services? I think that is the number one most complex problem that we're going to continue to face forever within this company.

[00:44:36] JM: Tell me more about how you see the next few months going for you. What are you trying to build out right now?

[00:44:44] TM: Sure. I think one of the things that is it kind of goes to like applications aren't one-size-fits-all anymore. You tended to start this with the idea that Docker and Docker containers and Kubernetes were going to be kind of the core which they are, and I think we're pretty much got that solved kind of. As long as you're on Docker and we can spin up a Kubernetes cluster for you, we can make your environments run. But that's not all of what your environment is. Static frontends, kind of Jamstack frontends, making sure that seamlessly you've got CDN namespacing. We talk about namespacing a lot today. But when we bring up an environment within Kubernetes, it has a namespace but an application will likely also have a JavaScript single page up frontend now too.

Originally, we did that by saying, "Hey, just fire it up in a Docker container and serve it out of the container." But our customer said, "No, no, no. That's not how I run in production." I actually put this on a CDN. So being able to kind of end to end all the way from even a JavaScript app that doesn't run within a container at all, it runs in a CDN, kind of getting that to seamlessly work within the environment, so we can really do this kind of, I call it Jamstack to full stack environment idea, is really what we're focused on at the moment. A customer that comes in and says, "Hey, I've got this technology ecosystem," inevitably it's got JavaScript frontends included in it, and we have to seamlessly work with that, all the way through the Kubernetes subsystem, and so we're kind of heads down working on solving that problem right now.

[00:46:23] JM: What are the kinds of applications that you cannot currently support with Release?

[00:46:28] TM: Anything that's not on Docker currently. We kind of have said, "Hey, we're going to focus on people who have containerized." We really need to have – The customer is either in the process of Dockerizing and containerizing their applications where they've already done it so that when we build the environments, we're kind of building a containerized environment within Kubernetes. I think that's the kind of core. We tend to focus on AWS the most because customers that are on a professional plan can run this within their Kubernetes or, sorry, their Amazon ecosystem within EKS. That being said, we do have some customers who are on other cloud platforms that are using us, but we're actually hosting it for them almost kind of like Heroku. For them, it's kind of a Heroku review apps, even though they're on GCP. But those apps tend to run within our hosted environment instead of on their cluster.

I tend to say our ideal customer is on AWS. That being said, we do have some that aren't, but they're kind of doing a hosted version of it. Dockerizing, containerizing, being on AWS are ideal. Shortly, we'll have the static frontend build system kind of out there too, so you'll have kind of full stack, all the way Jamstack, all the way to like containerized environments all the way in the end.

[00:47:46] JM: Before this, you mentioned you spent a total of 10 years on a previous company with an acquisition. I wonder if after those 11 years if you had any other ideas for companies aside from Release.

[00:48:01] TM: Well, I was in the automotive space, so I could tell you about 100 different automotive startup ideas that I had. But I kind of wanted to be back to the core. My background, I was in computer engineering, firmware engineering early on, systems engineering at the earliest days of my career. Throughout my entire career, I've always kind of gravitated toward the technical side of the problems that we were solving. Even though we applied it to various different spaces like parental controls, it was one of the companies that we started, a company by the name of IMSafer. CarWoo was another company that we started in the automotive space. We always took technology into nontraditional spaces. We built really amazing product and technology teams and technology stacks.

Then when we went through the acquisition into TrueCar, it was clear that like – I ended up becoming the CTO there and I just kept gravitating towards the technology side of things, and so me and my cofounders were like, "We have to do something on the technical side. It just kind of fits our DNA." We've run public technology company teams all the way down to startups, so solving the problems there. I've always kind of been in our sweet spot. That being said, there was probably 100 different automotive ideas that I can go do, but I kind of wanted to get out of the automotive space. I spent 10 years there. It was time to move on.

In the technology world, developer efficiency is something that whether it was ephemeral environments, PR environment, staging environments, or anything else, I still believe that developer efficiency is kind of the most important thing that you've got to do with inside your organization, and it's super expensive. We had to spin up a team of like 10 DevOps people to build the platform at TrueCar and I always – As the CTO of that company, it felt like, "Man, that

10 people, if we could focus their energy and effort on our customer problems and not our infrastructure problems, that would be the best scenario." Every idea that we have had is been around like developer efficiency, whether that's – There's tools like GitPrime that helped organizations kind of understand developer efficiency. This on-demand staging environment is another one. There's tons of stuff within the ecosystem around APM. We think some of this stuff the Datadog does is great, but there are opportunities in that that could be solved. Generally speaking, it's all been around developer efficiencies of ideas that we've been working through.

The other one that we talk about all the time is making data scientists more productive because I think data scientists, they live in a land of notebooks, and then getting their code running in production is still a really difficult problem. I know there's a ton of companies working on that, but we saw that firsthand and we tried a bunch of products at TrueCar to try to solve that problem, and that's a difficult one that I think is a big opportunity for somebody to go solve.

[00:50:52] JM: Indeed. The notebook side of things – This is kind of a strange question, but the process of going from data scientist on a notebook to production machine learning task, why is that significantly harder to solve than getting a staging environment and promoting it to production?

[00:51:13] TM: I don't know if it's harder. It's different. It's a different problem set to go solve, so data scientists, generally mathematicians doing their modeling in a notebook. What we tended to see was once those models were created, the coefficients were built. There was a data engineering team kind of sitting on the other side of that that needed to translate that into a running production model. That handoff was always tricky, and I think that ultimately is where companies like DataRobot – I know that AWS has a tool. I can't remember exactly what it is that tries to help with this too. But the idea is that if you can get the notebooks to kind of become production runnable models, then you can kind of skip that step of having the data science team need to translate the model into running code.

I think that problem is not necessarily – I mean, technically, it can be done with people. But automating that end to end I think is a – It's tricky because the notebook itself isn't written with kind of production grade thinking behind it. It's a bunch of data. You're testing your models. You're wanting to see if you can fit the problem to the line. When you get something that works and somebody's got to go implement whatever model that is within whatever technology ecosystem the company has chosen to run. So you have a problem of like, "Hey, there's a number of different technology ecosystems that those things we need to translate into." Then making that a super seamless thing from notebook to running production code is a pretty tricky problem to solve.

[00:52:46] JM: All right, Tommy . Well, is there anything else that we haven't covered that you'd like to explore?

[00:52:53] TM: No. I mean, I think that developer efficiency and I think ultimately we're a pretty mission-driven company. We're trying to help develop or ship things faster, not because that's the end. The end is getting great ideas out into the world, right? We think the world needs as many great ideas. These can make their way out there. When technology becomes a limiter on how fast you can get your ideas out, I think that's the area that we want to solve. We think staging is a big one. We saw it ourselves where we had 200 engineers kind of stomping on each other's toes, sharing a single staging environment. That's a big bottleneck, but we're going to end up solving developer efficiency and ultimately because we want to see the best ideas make their way out into the world, and I think that's the kind of focus of what we're up to. Staging environments is a good entry point because everybody struggles with it.

[00:53:44] JM: Okay, Tommy. Thanks for coming on the show. Great talking.

[00:53:47] TM: All right. Thank you so much.

[END OF INTERVIEW]

[00:53:57] JM: Operations teams can find themselves choosing between two options, either take full control over the infrastructure yourself or give all developers permission to access production. The first approach increases the operations team's workload, which often results in overwhelming situations and ops becoming a bottleneck. And the second approach allows for rapid deployment of changes to production, but it causes a serious risk to infrastructure uptime.

With Octopus Deploy, operations teams have a third option. The Octopus platform can be authorized to run approved steps and play an intermediary role. Octopus delivers self-service without sacrificing control over production, and it also provides a comprehensive audit log of the changes that are being made. Developers can enable self-service with automation. By automating the processes that are forming a bottleneck, developers can free themselves from the waiting game.

You can check out the most frequently requested run book templates by going to octopus.com/self-service. You can find those run book templates at octopus.com/self-service. And I want to thank octopus for being a sponsor of Software Engineering Daily.

[END]