EPISODE 1253

[INTRODUCTION]

[00:00:00] JM: In software engineering, telemetry is the data that is collected about your applications. Unlike logging, which is used in the development of apps to pinpoint errors and code flows, telemetry data includes all operational data including logs, metrics, events, traces, usage, and other analytical data. Companies usually visualize this information to troubleshoot problems and understand problems and opportunities and how their applications are used.

The company New Relic is a modern observability platform built to optimize and observe your software stack in one place. New Relic includes a telemetry data platform that acts as a single source of truth for telemetry data. Built on top of that are tools for full stack observability to visualize and troubleshoot your data in milliseconds. In this episode, we talk with Zain Asgar and Ishan Mukherjee. They're the cofounders of Pixie Labs, which was acquired by New Relic. Pixie Labs developed technology to help observability in Kubernetes environments. We had a great conversation about Kubernetes observability and what the acquisition means for New Relic as well as Pixie.

A few announcements before we get started. One, if you like Clubhouse, subscribe to the Club for Software Daily on Clubhouse. It's just Software Daily. And we'll be doing some interesting clubhouse sessions within the next few weeks. And two, if you are looking for a job, we are hiring a variety of roles. We're looking for a social media manager. We're looking for a graphic designer, and we're looking for writers. If you are interested in contributing content to Software Engineering Daily, or even if you're a podcaster, and you're curious about how to get involved, we are looking for people with interesting backgrounds who can contribute to Software Engineering Daily. Again, mostly we're looking for social media help and design help. But if you're a writer or a podcaster, we'd also love to hear from you. You can send me an email with your resume, jeff@softwareengineeringdaily.com. That's jeff@softwareengineeringdaily.com.

[INTERVIEW]

[00:02:09] JM: Guys, welcome to the show.

[00:02:10] ZA: Hi, Jeff. Thanks for having us here.

[00:02:11] IM: Jeff, hi.

[00:02:13] JM: So you both have worked on Kubernetes monitoring. You worked on Pixie Labs before it was acquired by New Relic. And I'd like to start off by just talking about monitoring Kubernetes and get an understanding of why it's different than other environments. Why did new tooling need to get built to monitor Kubernetes specifically?

[00:02:38] ZA: Cool. I'm going to take a quick pass on that question. So I mean, part of the challenge with monitoring Kubernetes environments, or actually really any distributed system, is that they're usually a large number of services written in lots of different languages. And in addition to that, it's usually built – The applications that run on top of it are usually built by many, many different teams or different practices. So what you typically run into is when you're trying to debug a problem that spans to many different services, it's usually difficult to get like a consistent viewpoint on them, right? Like the monitoring system is running, but it's not giving you the baseline level of information, because not everyone has added instrumentation. Not everyone's instrumenting things in exact same way.

And the good news around this is that there're lots of new open standards like open telemetry and open monitoring agents are coming out that make this process easier. Part of what we wanted to do with Pixie was actually make it easy to get base level insights around the entire system. And we started working with this kernel technology called EBPF, which actually allows us to go and automatically instrument all the code in the background without actually having to go and change any code. And this means that once you get Pixie deployed, you can actually get access to all the information regardless of what language it was written in, or if the developers had added instrumentation ahead of time.

[00:04:05] JM: We've done a show about eBPF. Maybe you could talk about it in a little more detail and explain what it enables.

[00:04:11] ZA: Cool. Yeah, so eBPF is a technology. It stands for like the enhanced Berkeley Packet Filter. It was originally developed as a way to write IP filtering rules in Linux, right? And the goal over here was to make a programmable packet filter. But in the process of writing eBPF, the authors of the software actually created a small sandbox virtual machine, where you can basically run a restricted form of C code. And the Linux kernel can guarantee that that C code can safely execute.

Over time, the sandbox virtual machine got extended to be able to access other things within the Linux kernel. And part of what we can do is take a look at what system calls are being made, what's being sent over the system calls, and we can even go and use eBPF to go patch applique to be able to do things like data clogging. And, fundamentally, this technology that started off as a packet filter now has well outgrown its name. And it's used for all sorts of monitoring and other tasks that require running in a sandbox environment.

[00:05:16] IM: Right. On that point, one thing that for us, eBPF was a means to an end, right? When we were looking at just the monitoring space, and our Northstar was how can we kind of make invisible a lot of the manual instrumentation that we were doing when trying to collect traces specifically, but also like metrics and logs. And then when we came across eBPF, it was pretty early. To Zain's point, it was been used in the CNI side for security use cases. Our kind of idea was, "Hey, this seems like a really interesting approach for us to potentially collect a lot of the inter-service communications without doing instrumentation," right?" So eBPF, in its own, is like a truly, truly transformative piece. And for us, it was trying to get to that kind of developer experience point, which is like, "Hey, we can use this to essentially get data in seconds."

[00:06:11] JM: What were some of the early engineering problems that you had to solve in building Pixie and trying to build useful monitoring tools?

[00:06:22] ZA: So part of the goal of Pixie, I think, from the very early days is that we wanted to provide a very seamless developer experience, right? We wanted to get to a point where Pixie gets installed, and you get access to a lot of data out of the box without having to go and modify your source code, without having to do a lot of configuration. So a lot of our early challenges where we're actually just trying to figure out how to make this work. And as Ishan just mentioned a little bit earlier, eBPF was kind of a means to an end for us, right? We wanted a way to do easy instrumentation. There's been other technologies in the past that people have used to do instrumentation like binary modification and things like Dtrace. But none of those things were particularly easy for us to deploy at scale. eBPF made this a lot easier. So that was one of the challenges.

The other challenge we had was we were able to collect so much data. And actually at relatively low overhead, we have to engineer an entire system to be able to handle this and not introduce a ton of overhead into the system. And this ultimately led to the way Pixie is designed today, where we basically run everything, what we call the edge, where everything actually runs on a node that it's collected. We try to figure out what's actually interesting to send upstream so that we don't saturate the network with all the data that we're collecting.

And on top of that, we needed to make this data easy to understand. So we actually started looking at building machine learning stuff to make it easier to find interesting pieces of data that we can actually send upstream to the user.

[00:07:56] IM: Right, just to kind of build on that. As you were getting deployed initially, Jeffrey, like to Zain’s point, like the architecture is in some way truly distributed, right? So you can start installing it in a five node Kubernetes cluster. Then what we started to see is that five nodes became 15 nodes and now we run in kind of 1000 plus node clusters. And running that entirely in the customer’s kind of environment where with the split architecture, the data plane lives inside of that environment and our control plane is in the cloud. That was something that we were pretty early on kind of thinking about, so just the first kind of year, year and a half, two years, which is it was probably unglamorous, but just figuring out that install, and then post-install scaling piece was like huge.

[00:08:44] ZA: I guess one more thing to add to that. Conventionally, the Wisdom is to move everything off of customer environments and into a SaaS base cloud platform, right? And, to some degree, our system is similar where, as Ishan mentioned, we have a control plane that's running in the cloud. And then we have a data plane that runs in the customer environment. But one of the nice things is, since we were on Kubernetes, we were able to use the fact that Kubernetes is relatively standardized to make it easier for us to actually build a pretty substantial deployment on the customer side without having to go through the pains of all the installation and management pains that you typically have when people just have a whole bunch of Linux machines.

[00:09:23] JM: You mentioned using machine learning to try to find some of the more insightful outputs that you could generate through all this monitoring data. Tell me a little bit more about the machine learning side of things.

[00:09:37] ZA: Col. Yeah. Actually, we'll be having a blog post on this in Google's TensorFlow blog, probably next week, but there're actually a few different things we're looking at. And we're continuously adding new things. There're relatively simple things that we do, like figuring out like URL clustering and SQL query clustering. So we can actually go and group a whole bunch of related queries together and allow you to facet them by like a different user type and also latency. It’s hard to understand if like certain keys are a lot slower. So those are like the relatively simple things.

We also do things like do schema detection, right? So if you have like an API call that's being made, will basically try to go figure out what the schemas of the API calls are. And one of the things we're using this for is to enable better compression of the data, but also allow users to detect schema changes. And then there're much more complex things that we're starting to look at, which is actually trying to connect different pieces of information. So like different traces together based on fields and the messages. Ultimately, we're trying to connect, “Here are set of messages. Here's how they're related. Here's how we see them impacting the latency.” And allow you to figure out and take a look at like example traces so we don't have to collect every single HTTP request with as full detail and send it up to the user.

[00:10:59] JM: So as a Kubernetes operator, what do I want out of a monitoring platform? Like do I want a system that's just going to alert me when something is running out of memory, or when I have an outage? Or are there more innocent kinds of alerts and detections that I'm looking for? Tell me a little bit more about the kinds of features that I want from a monitoring platform?

[00:11:30] ZA: Sure. So I think in general, in a monitoring system, you definitely want it to send you alerts and find problems. And one of the interesting things is that at Pixie we actually don't send alerts, nor do we actually store data for long periods of time. We actually built a lot more like a live debugging system. We really see ourselves as a way to enable developers to actually build and operate applications in both production and also any kind of debug environment that they have.

So I'll give an example. Let's say you're running into a performance problem and you've got an alert from a system. You're trying to figure out where the performance problem is. You can very quickly go into Pixie, look at the pod, and then we have a feature called continuous profiling where we're just always showing you where is CPU time being spent in your application. So once you go to that, you navigate to the pod level view, you take a look at this profile, and you can immediately see which functions in your code are slow. So part of what we're actually trying to do is move away from monitoring as a very passive concept to being able to use this data to actually help with the operation and debugging of your application. And to add to that, one other point is that we have this concept of a pixel script, which basically is a Python script that's based on Pandas that allows you to work with the data in Pixie. And we have a whole repository of scripts that allow you to like debug different use cases.

So if you're trying to figure out why you're running at a certain type of Redis problem, you can run one of these scripts, and then actually get some pre-canned things. And as you learn more and more about your system, you can actually go and codify that knowledge. So just to step back, ultimately, we think that it's not just about monitoring and getting alerting, but we want to basically allow us seamless debug and development experience,

[00:13:24] IM: Right. To kind of build on that, the types of engineers who end up kind of using Pixie and hopefully loving Pixie, Jeffrey, is beyond just kind of platform for engineers or maybe ops engineers. The core idea is like how do we expose as much data as possible in an API where, hopefully, all or most engineers can get access to this data in a programmatic way. So we see a lot of actual application engineers, whether they are on API teams or other kind of backend teams using our data, writing scripts, to run kind of use cases like, “Hey, here's my auto scaling logic,” or some sort of heuristic-based kind of flags. And it's going to be on just kind of ops and traditional infra.

So Pixie is kind of like a data system with an API, which essentially allows a ton of different engineering personas to get access to this information. And they're writing kind of distributed topic like utilities. So it is kind of a new paradigm. And the last point is we do unify kind of metrics, traces, spans or logs and events in a unified kind of data structure. And being able to kind of access that, we've seen kind of engineers who are not married too much to like metrics, traces, logs, kind of pillars of observability. Like once they get all of the data in like one clean format, the types of things that people are doing are super fun and interesting. Like somebody wrote a script to flag internal services, calling external services with kind of email information. So as Zain was saying, Pixie is kind of this kind of access, programmatic access to this fire hose data, where you can kind of essentially run these kind of decision jobs or pipeline jobs. And then for longer term, retention and alerting and traditional monitoring, you can pipe that to your Elastic or Grafana stack or any kind of SaaS monitoring stack.

[00:15:21] JM: Tell me a little bit more about how you see the typical stack of monitoring tools developing for a platform operator. So what would they use in tandem with Pixie Labs and what would be, I guess, the data pipeline? What would will be the different workflows involved in such a monitoring stack?

[00:15:44] IM: Right. Traditionally, it obviously depends on the type of engineering team and also their scale. Let's just say for kind of medium to kind of large kind of organization, let's just say you have five to 10 plus engineering and a core platform team who are responsible for your kind of dev kind of pre-production and production platforms. And then you have a set of engineering teams. In that what we usually see is from an instrumentations side, kind of platform engineers advocating for and standardizing on open source standards, whether that's Prometheus for metrics, Jaeger, Zipkin and similar projects for traces, and then using FluentD, Fluent Bit for like logs. So we see most of the collection and related activities happening in the customer's environment being increasingly, if not all, open source.

Once you're collecting that data, then there's a decision point, do you build your own data storage and compute stack? So we see a lot for logs, for example, like kind of a team setting up an Elk-based systems. And on top of that, writing using Grafana for alerting, right? So I think a lot of infra teams call it the packstack, like Prometheus alert manager, Grafana, and with Elk for kind of unstructured data. So that's one.

The other is obviously for kind of teams who want to look for managed solutions. They use tools like Datadog, and New Relic, and Splunk and what have you. So a stack, most engineering teams like on the instrumentation side are standardizing an open source. Most platform infra teams, because they have that band, they have some sort of open source stack. And then there's a decision, a lot of engineering teams kind of have these kind of SaaS observability platforms where increasingly all of this telemetry is going into like one place.

[00:17:41] JM: As far as the Kubernetes stack itself, you've had to integrate with the various container platforms like EKS and GKE, and I imagine Fargate, and the other container instance systems. What are you seeing in terms of container platform usage? Like how are people deploying their containers these days? And what is the contour of their infrastructure deployments look like?

[00:18:14] IM: Yeah, I can take that. So I think kind of fun kind of anecdote is, as we were talking about eBPF, and Zain and I were thinking of Pixie, we got really infatuated with this idea of giving a developer experience where you get access to all data that you want without doing any work, because all of us are somewhat lazy. And at that point, we had to make a bunch of decision points, and Kubernetes was one. So we chose kind of Kubernetes because that kind of gave us this platform substrate where you could deliver this experience. At that time, it was still relatively early days, I would say like year three, year four of the Kubernetes adoption journey. Kind of fast-forward now, like there's no question that it is kind of generally agreed as the kind of the consensus standard way to deploy kind of containerized applications. So whether that's on our open source stuff, or kind of generally, I think it's at a point where 50% to 60% plus kind of customers that we work with have some sort of Kubernetes in production, which is like quite amazing about like how that's going to come together.

And in terms of self-managed versus using kind of cloud manage distributions, initially when we started Pixie, we used to see a lot of self-managed clusters actually because of the architectures and walkthrough. We were kind of agnostic to what kind of Kubernetes it was because Pixie just kind of Kubernetes native. So whether that's a self-managed cluster, or EKS, or AKS, it didn't really matter.

So we saw a lot of early developer kind of engagement from folks running self-managed clusters in their own data centers, or maybe in their kind of EC2 clusters, right? So we used to see a lot of that, but right now, like all the kind of Kubernetes versions, EKS, AKS, are really, really, really great products to be very honest. And we see more of that getting adopted across customer bases. As an example, initially for the Pixie project, a cohort of developers that really resonated with what we were doing were folks working on content kind of streaming teams. So these are kind of large streaming applications that we know and love, and we use a lot.

So they had kind of containerized workloads already, which kind of range from there serving applications to recommendation systems to even their ad serving models. All of them, like fast-forward two and a half years, have gotten a Kubernetes-based kind of production systems. And those production systems now are primarily kind of cloud manage Kubernetes environment. So that's kind of what we're seeing. But to kind of complete that, we definitely don't view Kubernetes as the only platform. It is an increasing part of like a portfolio of substrates. So most customers, the middle to large sizes have their kind of – Let's say AWS, for example, you have your EC2 clusters, you have EKS clusters. You have a little bit of kind of Lambda, some Fargate as well. So it is definitely a portfolio. Like it's much more heterogeneous than one would like to imagine. So yeah, that's kind of where we see it.

[00:21:32] JM: And how have you seen the evolution of functions as a service? I realized this is not exactly closely connected to what you're doing. But I imagine you work with a lot of customers who have deployments where they're doing some work with their Kubernetes clusters and some with their Functions as a Service environments. So I'd love to know how that fits into the typical deployments that you see.

[00:21:57] IM: I can talk a little bit about the community, and then I'm going to hand it over to Zain, because it's kind of interesting how Pixie is useful. And then a candid story, like when we're thinking about designing Pixie, we were talking to like a ton of kind of developers and platform engineers in the community. We did come across a bunch of startups who were purely functions, which is like quite stunning and quite exciting. I think that continues to be the case to this day. But from our standpoint – And there might be a bias obviously like with the community that we engage with. We don't see kind of Functions as a Service being a major kind of wheelhouse type of workload as yet. It may, and we hope it does happen. But it's definitely not something that we come across a lot. Zain?

[00:22:46] ZA: Yeah, I think where we have mostly seen functions as a services, people using it for – I don't really want to say like side tasks, but things that are like mostly stateless services, like do this translation, resize this image, or something that's like very self-contained, and not really a core, necessarily a core part of the stateful business logic that people have. And as Ishan mentioned, we probably have a very bias set on this based on the users we interact with.

In terms of how we plan to support it, the current state when you're running in – When you have a function as a service, the infrastructure is completely abstracted from the user. It also means that our ability to run on that infrastructure is also completely abstracted. But we are actively working on trying to provide the same level of visibility for customers that are running things on like Firecracker. And it's mostly an engineering challenge of trying to build out similar infrastructures in that space.

[00:23:40] IM: I actually thought of a one pretty interesting conversation Zain and I had. This is a leading kind of media company who have a legacy printing business. And to Zain's point, like they actually use functions to do these very self-contained tasks around like, "Okay, the CMS has outputted like the content with the framing, and now you need to tell the printing side of the house like print." That was a function. So I just kind of thought of that.

[00:24:08] JM: Gotcha. So as the product evolved over the course of the few years that you – Whether you worked on it before it got acquired. What were you hearing from people? What was the product iteration process like as you sort of collided with the market? How did your customer conversations shape how the product evolved?

[00:24:32] ZA: Yeah. I can take a quick pass, and Ishan can add a lot more detail. So we were extremely iterative in our product development, which is actually pretty surprising for most infrastructure components, especially ones that's deployed on someone else's machine, someone else's like clusters. But from the very, very early days, Pixie had an auto update feature, and ability to iterate quickly. So based on like feedback, we'd find out like, “Oh, what kinds of applications are people running? What things are they interested in?” We had very early on developed the ability to script things in Pixie. So the two ways we had, quickly iterating. The easiest one was we could create entire new views for people by creating a new script. And this was a way that we could show both like a completely new capability in the product, but also kind of show how you can use scripts to build all these workflows. And then the second thing was when we fundamentally needed to add a new way to capture data, like capture Redis data, be able to go put that in, and then usually ship an update to somebody and all the daemon sets will eventually update automatically, and they will get access to the capability.

[00:25:43] IM: Right, and that's a really like great point there. Just going to share a little bit of context on that and also like build on that, there were a couple of decision points for us. Because initially, just like any kind of monitoring, any SaaS product, we had these really fancy complicated UIs that we mocked up, right? Which would be the intuition and the normal part to go there, right? Like build an end-to-end SaaS application with really involved kind of user experiences. But we had this one decision point where, as Zain was sharing about the script, where “Hey, should we go and build the end-to-end magical kind of UI experience? Or should we build a platform out in a way where it can run in production at scale?” And we ended up deciding to do the latter, because the core idea was we were getting this kind of traction from streaming engineering teams, and their scale was just ginormous. We felt like if we can get deployed in production and actually deliver real – Like, I guess, “value”, which means our data is getting used to like do triage, troubleshooting, writing, powering the load balancer, that is just a better validation for the technology than kind of going, building out the full UI. So that helped us make that decision that Zain talked about, which is like, “Okay, let's just focus on this API and the scripting interface and also build out the install, upgrade, scaling pieces so that we can run in production at essentially consumer scale.”

At that point, we had these difficult decisions where we decided to work more with large scaling teams and less with kind of smaller earlier stage kind of startups, because in an early stage teams, we just did come across – The scale wasn't the issue. The need was really like simple intuitive experience. And yeah, we decided to prioritize a lower than the scripting interface and the scaling pieces.

[00:27:42] ZA: Yeah. I guess just to add to that, to build on that a little bit. Because we're getting access to all those data and the early traction that Ishan mentioned, we kind of went down the path of essentially targeting power users with Pixie. And that was a decision that we made early on. And our goal eventually was to try to build more and more higher level interfaces over time. But our method of doing that from the early days was actually just powering everything using scripts, because ultimately, we wanted our users to be able to build all these things themselves.

And what this lead to is like in every major org we talked to, there'll be a couple of people who would kind of like take on the thing of like, "Oh, let me learn the system and be able to write new workflows around it, and shared with the rest of the team."

[00:28:33] JM: How do you see the Kubernetes environment evolving in the next five years?

[00:28:39] ZA: Yeah. So I can try to take a quick pass of that. One of the things that I think is going to happen, as it does in most systems as they mature is that to some degree we'll see Kubernetes more and more just disappear from the developer workflow. And I actually think that's a good thing. And part of what's going to happen over here is as Kubernetes itself matures, developers will start writing applications using other frameworks that'll manage all the deployments and setup and configuration. And they'll abstract themselves away from the low-level infrastructure. To some degree, you're actually going to kind of see this entire like re-evolution of the system where people will go and effectively be like Heroku or AppEngine or whatever, but running on top of Kubernetes.

[00:29:33] JM: Anything to add, Ishan?

[00:29:35] IM: Yeah. No, I can't agree more. It might be a little weird for product builders like Zain and myself talking about Kubernetes getting abstracted away. But we fundamentally agree. So like our goal has always been as developers like how – Fundamental rule has been like, “Oh, we’re all lazy. How do we save developer time?” And, ultimately, we want to build a system where developers get access to the data that they need and not worry too much about, I guess, the platform primitives as they say. So, like we do see this already. Like do most dev teams and developers care about Kubernetes that much? Not really. It's already abstracted out. Because if it's not an API, it is like a platform infra team, right? They ship – They’re managed in a container and like the world takes it away from there. So we definitely see that happening.

So in a lot of our kind of discussions, we spend more of our time focusing on “Hey, do you need a live debugger? How do we make our API better?” and less about kind of the deployment substrate. On our side, we tried to give a pulse and try to make sure Pixie is deployable in as many kind of environments as possible. Over time, there might be a next thing and hopefully Pixie is deployable anywhere. That's our goal. But yeah, I agree with Zain.

[00:31:00] JM: So if Kubernetes gets abstracted away, what are the backend developers doing?

[00:31:06] ZA: Yeah. So I don't know if backend developers necessarily means that they have to worry about every layer of infrastructure. So I can get some context for my time at Google, right? It's no secret that Google runs this thing called Borg, which effectively is the predecessor for Kubernetes. And when I used to build code at Google, actually, in the beginning, didn't care a lot about Borg. And then eventually, when you start deploying things in production, I was like, “How do I cargo cult enough scripts from other projects so that I can just get something around Borg?” right?

So as a developer, I never really cared to understand deeply how any of that stuff work, because it just did. But I can tell you, like towards the later years when I was at Google, we even had other systems that would be like, “Okay, you have this built out. Specify what the container is,” and it will just automatically manage, effectively manage the cargo culting for you so you don't have to do it. And that's what I think is going to happen, right? Where people will stop having to write like 1000s of lines like YAML files for most cases. I'm not saying that they're not going to have to do it for some special cases, like, “Oh, we're running some fancy database and we need to really worry about how to manage the storage and communication and everything.” But for most applications, it will be a lot simpler.

[00:32:25] IM: Right? Just to kind of add to that, like something that's interesting and curious, sparks our curiosity, is like how much of that abstraction and experience is done by a customer's kind of platform infra developer experience team versus the cloud's kind of Kubernetes distribution, right? So like GKE, as an example. They're just shipping so much like an interesting stuff where like with auto scaling, and now they're changing the pricing as well. Where it kind of points to a future where it speaks to kind of Zain's vision, which is developers will increasingly like not really care and they might not spend the cycles to kind of get wrapped themselves.

An interesting anecdote here is like we were talking to somebody, and I was asking them like, "Hey, are we talking about a prod or a pre-prod environment? Or is it a test environment?" And they said, "I don't know." They just worked with their CI/CD system. The usual norm is that the way Google does engineering is pretty futuristic. But we actually see that happening, like that kind of developer experience happening a lot more in kind of early stage and kind of traditional kind of enterprise teams, because CI/CD and getting that to be seamless and building like a really nice developer experience is important across all companies. And it's kind of getting to that point.

[00:33:48] ZA: Right. Just to build on Ishan's point, one of the things that we do see is that most companies that use Kubernetes, they have another team that's basically dedicated to setting up all the infrastructure and making all the application deployments happen. So to some degree, the developer abstraction right now is happening through humans and preset processes. And as you know, Ishan was just alluding to, I think a lot of this stuff will actually move to the core platform over time.

[00:34:17] JM: How does the architecture for Pixie compared to traditional monitoring systems where you have an agent that you install on each of the nodes, and the agent collects all the data and sends all the data to a centralized server and you can query the server and the server does all kinds of things like look through the data for things that constitute alerts. Just tell me a little bit about how your architectural decisions have Pixie contrast with traditional monitoring systems.

[00:34:50] ZA: Yeah. So, roughly, I think there's kind of like maybe three different types of architecture paradigms that we should consider, right? So one of them is the one you just mentioned where there is an agent per node and you're basically shipping data off to a centralized system for processing and analysis. And a lot of like infrastructure monitoring tools use this. There's another model, which is actually much more common in APM use cases where you're actually trying to get deep code level insights, which is where you actually have a library or something that you actually add to your binaries. So for every application that's running on your node, you actually have effectively an agent that's running, relaying data to some centralized service.

And most systems today use some combination of the two, right? One of them to collect the infrastructure data, and the other one to collect the application level data. And the third paradigm is what Pixie is currently using. And you can totally mix and match these in different ways. But the way Pixie works is we actually have effectively one agent, which we call the Pixie Edge Module that runs on every node. And this collects both the infrastructure data and the application level data. And in addition to just collecting the data, we do most of the data processing and storage locally. And part of the advantage of doing that is, with Pixie, you get visibility to every HTTP request that has happened. And you can go back and be like, "Okay, let me know what the HTTP request was that caused this problem." If you have that level of details, you're basically running into issues just egressing the data out of every node, right? If you have like a gig per second of never traffic, you're probably not going to be able to egress that out to any centralized system.

So what we very quickly have to do is take a look at the network requests and determine, "Is this a reasonable request for us store? Is this a sample of something that might be happening that's weird?" So we can store the time and the metrics for every request. But we can't feasibly store the data for everything and ship it over somewhere else. So what we typically do is that we store a bunch of data on the actual add module. We then ship off some data that we think is interesting to another node that's running within their cluster. And then, ultimately, the user can access the data from the hottest data and the most interesting data from our semi-central service. And if they really need to, can go access the raw data from every single edge module. So what this overall leads to is just a lot more efficient of an architecture where we introduce relatively low overheads.

[00:37:28] JM: Tell me about your process of debugging Pixie as you were building it. Like it to me it seems like a very difficult challenge to iron-out bugs and problems in a core infrastructure tool like this. Just tell me about your workflows for doing that.

[00:37:47] ZA: Yeah. So I think there are two things, and probably more than two things that made debugging challenging with Pixie, right? The first one is we're actually running on someone else's environment, which means getting access to data can be difficult. And I think the second one is that we have some stuff running in eBPF, which is not the easiest system to debug. So one of the things we thought about from the very early days of Pixie was just like how do we actually add enough capability in the tool to be able to access our debug information and even generate more debug information. So one of the things is, for example, we have a mode in Pixie where even if everything is broken, we will still continue to do similar function so that we can pull the debug information out without having to go dig through all the logs in Kubernetes. This allows a user of ours to very quickly send us the information by just running our CLI command.

So we spent a lot of time trying to make sure that there's at least enough stuff working that we can go access information. In addition, we have a whole bunch of debug hugs so that we can actually go – When you're running our scripts, you can actually go and add a debug hug that will dump out a bunch more information when it's running. Like how did this thing get executed? What was the query plan? Where did we spend time in various steps, steps with a query plan? And we capture all this information so that we can easily debug the system.

I guess another thing to add to that is one of the good things about architecting these types of systems especially on Kubernetes is that we're basically built to be resilient, right? And that was one of the core premises, is that we should be able to function even if things are crashing or going out. And this is like a good methodology whenever you're building a distributed system to make sure that you can function even if pieces are missing and just report that there're partial results available. So we're pretty careful about doing that.

[00:39:40] IM: One thing that comes to mind and actually Zain and I haven't talked about even is that we had this kind of internal kind of development notch start that we want to be able to use Pixie to debug Pixie, but it took us a while to get there, right? Like it was not from day zero. Like when we wanted to dog food while we were building on the platform. And I would say we started late 2018, 2019. Maybe like late 2019, early 20, we got to that point. Right, Zain? Like it took us a while. And we were never happy with it. Like, “Hey, we should be able to dog food it.” But we're also building the bricks at the same time, like building out the platform.

[00:40:15] ZA: Yeah. One of the challenges of dog fooding your own monitoring and debugging system is when your own system doesn't work and it makes it very challenging to debug your own system. This actually came in two different forums, right? Like the first forum was we weren't necessarily supporting all the protocols. Like, for example, Pixie used HTTP 2 and GRPC. That was not the easiest protocol to build support for. So we didn't actually have support for it in the earliest version of Pixie. So we weren't able to look at our own traffic. Then there were a set of challenges where if Pixie was down, then we won't be able to capture any data and we wouldn't be able to debug. But ultimately, over time, we added enough protocol support and there's enough resiliency in the system now that we're usually able to get debug information out. And to Ishan’s point, it took us a fair amount of time to get there. It was a goal of ours because we want to build a dog food and use our own system.

[00:41:07] JM: To what extent is Pixie open source? Or what open source components do you use?

[00:41:14] ZA: Yeah, so two fronts over there. So what part of Pixie is open source? Basically the entire thing is open source. We have a hosted version that's also completely free to use. But you can download Pixie, run your own version of what we call Pixie Cloud, and then deploy Pixie point to your own personal Pixie Cloud. So everything is open source and there's no tie into any commercial service to be able to use Pixie. We do offer a hosted version. That makes it a lot easier for people to get started. But even that is completely free to use.

In terms of leveraging open source, adding as of most modern software, we have probably hundreds of dependencies on open source, on other open source libraries and frameworks. Some of the core ones are things like we're heavy users of BCC, which is the BPF Compiler Collection, I think is what it's called, which basically allows you to write code to interface with BPF. Also, our entire data system is built on top of Apache Arrow, which is something I worked on a very, very long time ago when I was at Trifecta. So those are I think some of our two big core dependencies. And then of course, we depend on Kubernetes to run.

[00:42:31] IM: We also use V3 and Vega.

[00:42:34] ZA: Yeah. I guess to Ishan’s point over there is that our scripting languages is based on Pandas, even though we don't actually use pandas directly in our code base. We do actually do all our visualization using this library called Vega, which is effectively a way to declaratively specify visualizations. So not only in Pixie can you select one of the default visualizations, which we’ll use Vega. But you can actually code your own visualizations and add it to your script and leverage Vega to do that, which, again, is an extension. Well, I guess it's built on top of V3.

[00:43:14] JM: You mentioned the use of Arrow. I remember doing a show about Arrow. Done a few shows discussing it. My recollection is that mostly used for like interoperability, like data interoperability, like being able to share data structures between like Java programs and Python programs. Basically, it lets you – It's like a data sharing format or a serialization format, something like that. But can you tell me a little bit more about how that's integral to your system?

[00:43:46] ZA: Yeah. So the primary person behind Arrow is Wes McKinney right now. And actually, many years ago, when I was at Trifecta working on Trifecta's data system, we were actually collaborating on working on Arrow. And Arrow is kind of based on this concept of essentially creating like a standardized in-memory format, right? There are a whole bunch of different memory – A whole bunch of formats for keeping data on disk, like things like Parquet, or Orc, or whatever. But the goal of Arrow was to create an in-memory format that would be efficient for compute, right? And it's a row-batched columnar format, which it does very efficient things for modern CPU architectures to use. And in addition, to enable some of the interoperability that you mentioned, right? Because it is a standardized memory format, you can to another program, "Hey, this data is located in memory over here. Go work on it. Produce some results. Write it over here, and then I can access it in the same format."

Arrow ultimately solves like two problems, right? So one of them is just like a specification for how data is in-memory. And then the other one is the standardization of that format so that you can share it across different languages. So we use both aspects of Arrow, right? So when we build up Pixie's is data system, we wanted to build on open standards. So we use Arrow to build an efficient data system. And the second part of it is we use Arrow to exchange data with other things. For example, in Pixie, we use TensorFlow. And we use Arrow to exchange data with TensorFlow so that we don't have to have TensorFlow worry about reading some like specialized internal format.

[00:45:23] IM: Right. Actually, on a broader point, on Arrow, like we see a lot of customers who are familiar with Wes and Pandas and Arrow actually become Pixie customers. And on the other side, there projects around Arrow like Rapids and BlazingSQL, which are actually much wider spread out. Then Zain and I thought it were like large kind of banks and enterprises and stuff are increasingly looking at Arrow. And then Rapids is essentially a way to like deploy that on GPUs specifically to run kind of analytics pipelines.

[00:46:00] JM: Well, as we begin to wind down, you talk a little bit about the acquisition. So you guys got acquired by New Relic? What are the dynamics of an acquisition like that and what kind of synergies do you see between the two companies?

[00:46:16] ZA: Yeah, I can give a quick pass on that. And Ishan, please add details to this. So we were getting ready for Pixie' launch actually, and we started having some very early conversations. In particular, we got a chance to meet with Lou and we kind of have this like shared vision to make all these like monitoring and debugging tools easy and accessible to all developers, right? So that we were kind of driven by the shared goal, and effectively that led to us deciding to join forces with New Relic on trying to effectively create Pixie as a platform that will be open and available to all developers to use.

[00:46:58] IM: Right. And as we open sourced all of Pixie this year, like we definitely get asked a lot from the community about like, "Why open source now?" And like, "What was the journey?" To be really honest, when Zain and I started Pixie, we went super opinionated on open source versus SaaS. We were just trying to build, we thought, this magical platform, which helps get you the data that you need without doing much work. And kind of we followed the norm of – we're on the path to build kind of an end-to-end kind of a product or a SaaS product similar to a New Relic or Datadog, but do it with the community. So we had our Pixie community and we had this basically a free forever product. And we were on that journey. And around that timeframe, as Zain mentioned, like we aren't very familiar with it. But like as we spent time with like Lou and Bill Staples, the Chief Product Officer and President, and the team, we got to know like their view of essentially standardizing on open telemetry and open sourcing all software which collects and processes data in the customer's environment. So that shared vision plus this kind of idea of making Pixie ubiquitous kind of got us pretty excited. And then from a technical standpoint, we felt like "Okay, like this allows us an opportunity to open source all of Pixie. Kind of keep building out the project to a point where any developer can use it, similar to Prometheus or Kubernetes itself." So that was a big kind of exciting point.

And now that we've joined forces, they've been super awesome. So like the entire kind of Pixie engineering team and this fast growing kind of completely works on the Pixie open source project. And we start the process of donating it, contributing into the CNCF. So hopefully, like the Pixie community and the project kind of catches on and just keeps growing. So it's pretty good. And to your point about how it ties into like New Relic's side. So what's happening with the broader observability space is that the industry is kind of standardizing on open standards, whether it's open telemetry, or the ingest spec, and then all the agents. So New Relic was already – They open sourced all of their agents, kind of the Ruby, Java, all that stuff over the last kind of 8, 10 years of R&D. So they were on that journey. So it can align really well. So our goal is to kind of expand how broadly kind of Pixie is used. And if a customer wants to build their own Elk stack, they can do that. If they want to use kind of SaaS stacks, they could do that as well. So there's kind of a good kind of engineering alignment.

[00:49:39] JM: Awesome. Well, anything else you guys want to add or should we wrap up?

[00:49:43] ZA: Oh, that was good. Thanks a lot for having us here.

[00:49:45] IM: Yeah, thanks, Jeffrey.

[00:49:47] JM: Awesome, guys. Well, thank you so much for coming on the show.

[END]