EPISODE 1480

[INTRODUCTION]

[00:00:00] ANNOUNCER: While Kubernetes has many benefits, there is often a need for teams to deploy a monitoring and observability stack to troubleshoot issues that happen within the cluster and the applications themselves. ContainQ is an out-of-the-box solution that allows engineers to monitor the health of their cluster and troubleshoot issues faster. ContainIQ is unique in its approach, and that it was built with eBPF in mind, and is able to provide an APM-like experience without being an APM.

Matt Lenhard is the Co-Founder and CTO of ContainIQ and joins the show to discuss the future of K8s monitoring and observability, as well as the unique technological approach he is taking with eBPF.

[INTERVIEW]

[00:00:50] JM: Matt, welcome to the show.

[00:00:51] ML: Hey, Jeff. Thanks for having me. Pleasure to be on.

[00:00:54] JM: There is a wide variety of monitoring platforms. And for some reason, Kubernetes monitoring specifically is a domain specific monitoring challenge that is worth devoting an entire company to. Why is Kubernetes monitoring different than other kinds of monitoring?

[00:01:19] ML: The reason we chose to focus on Kubernetes is really because it gives us a lot of the metadata we need for correlations. And that's because core to our product is eBPF. And we use that to parse out things like traces and metrics. And then we can use that data alongside the rich metadata Kubernetes provides to give you like a deeper insight into what's happening.

[00:01:47] JM: Okay. Is there a specific activity trace or data management system that you're using to gather that monitoring data? Like, are using Prometheus? Are you using – I know you

mentioned eBPF. Maybe you could talk a little bit more about the actual engineering breakthroughs that you're using to monitor Kubernetes.

[00:02:12] ML: Yeah, happy to walk through our architecture a little bit. I think that's the interesting thing. Most of the breakthroughs come from like the awesome work that people are doing on like the Linux Kernel. PCC Tools is another great platform as well for developing eBPF-based programs.

But I guess like the way we're utilizing is we basically look at every open socket on a given node and we parse out all of the socket buffer basically. And from that, say, for example like an HTTP request, we can see how long your applications are taking to respond to HTTP requests. And then we can associate that with the Kubernetes metadata. That socket's going to have two IPs, right? We can grab that IP. See which pod corresponds to that IP. And then we can give you information like, "Hey, on this given pod, it's taking this long to respond to TCP requests." Or parse out the actual like HTTP body and say, like, "This pod saw an elevated spike in 500 errors."

The core of the eBPF is really like tracing those networking calls, like specifically anything going over like the TCP sockets. And then we use the Kubernetes API, specifically like their Go client, which is like an awesome tool to work with, to grab a lot of information about like state and all of your pods. Really, like all of the kind of correlation metadata about your cluster.

It's really broken down into like kind of the eBPF side of things, which is like a daemon set that runs on every node. Gets like the socket information. Or, sorry, network calls. And then a Kubernetes deployment that grabs a bunch of like metadata from the Kubernetes API. Like everything from like – Like I said like, pod information, to what pods belong to which deployment, which pods are active things, things like that.

[00:04:13] JM: We've done some shows about eBPF. But can you just give an overview for what that is?

[00:04:19] ML: Yeah. I think that the easiest way to think about it is – And kind of do like two really cool things with it. And that's you can hook into Linux system calls. Every time like a

system call happens, you can grab like the arguments, the return arguments, which allows you to do like really cool things with like the kind of like underlying functionality of what's going on in the kernel.

Another really cool thing you can do with it is like attaching to different sockets. Specifically, we attach to the traffic controller and we parse out everything going through the different sockets. And then you can add like uprobes to different like binaries and basically like grab the same type of like arguments as you would with like a kprobe system call. Really, it's almost like JavaScript event listeners in a sense. Every time this function is called, do X, right?

[00:05:11] JM: Gotcha. What would some of the function triggers be in this case if you're building a monitoring platform?

[00:05:18] ML: Yeah. I think a great example of this is Falco, which I think everyone should check out. It's like a security tool built like on top of eBPF. That's basically looking at the system calls that are happening on your machine. The processes like that are calling them. And then like should they be happening?

You could do like – You could look at open TCP connections. You could see which programs are being executed. You can really hook into anything that's happening. In a sense, the sky really is the limit. I would also tell people to check out like BCC tools. There's like a huge list of tools that they have that can kind of like get you started and show you like the possibilities of what you can do with eBPF.

[00:06:02] JM: If you start to use eBPF to develop a monitoring platform, is a lot of your work just defining those custom triggers and the events that are going to occur as you're listening across your infrastructure?

[00:06:20] ML: Exactly. The first step is to figure out the exact system calls you want to like hook into, right? At its most basic, maybe you want to hook the right call and parse out all the buffers going through there. Yeah, you need to identify like what information you want and like which calls you need to hook in order to grab that information.

The next step that can be difficult, it was a lot of work for us, is getting it to work across a bunch of different kernel versions. Because support can differ basically depending on what kernel version they're using.

There's something relatively new called core compiled once, run everywhere, which tries to solve a lot of these issues. But kernel support isn't great for that yet. So yeah, the step one is you know figuring out which system calls you need to hook into. Step two is figuring out how do I get it to work on like the widest array of kernels.

[00:07:14] JM: Why is that important? I mean, isn't there a pretty standard set of kernels that it would run across?

[00:07:21] ML: No. Because the structs change between kernels. System calls can change as well. There's definitely work that's involved getting it to work across like the different kernel versions. Like I said, there's a lot of work going in right now to kind of fix that problem with BPF type format and the core compile once, run everywhere.

I mean, just personally we've had to do a lot of work to get it to work across different kernel versions. The verifier will also complain a lot of times on earlier kernel versions. Because every BPF program that's run is put through a verifier to check that like loops are bounded and the number of instructions. Something might run on 4-3. But on 4-2, the verifier is yelling at you. Those two things, like the verifier changing and the changes in system calls and structs basically leads you to a scenario where you can run into issues across different kernel versions.

[00:08:20] JM: Can you tell me about how you fit into a more comprehensive monitoring stack? I imagine, there's a lot of your customers that use Datadog already. Are they willing to adopt a second monitoring platform?

[00:08:39] ML: Yeah, that's a great question. I would say, typically, we've had customers migrate off of Datadog. But typically, we grab people before they start using something like Datadog. And like our, I guess, core value prop for a lot of them is, "Hey, you've got this microservice architecture. Currently, you've got some APM set up on – Or you're going to set up an APM on every single microservice."

So maybe at a smaller company, they've got 10 to 15 microservices. They're going to have to instrument every single one individually with like whatever package manager they're using for that language. Well, with like ContainIQ, we're like, "Hey, here's this helm chart. Install once. We'll automatically collect all of the requests and traces like across your entire cluster." I guess kind of the selling point is like, if you have a lot of microservices, you can install once and set up to and having to install on every single one.

[00:09:35] JM: So, do you have to compete with Datadog? Or do you feel like there is kind of room for a differentiation that can be expanded upon?

[00:09:46] ML: Yeah, I think it's kind of building on top of like the move to microservices and the move to Kubernetes. If you can provide an out-of-the-box solution that instruments every microservice automatically, you're just going to save people a lot of time from having to set up like something like distributed tracing and Jaeger. And just by being kind of like Kubernetes native, you can just provide insights and have kind of a better UI and like an all-in-one view. That's a little bit better than like somebody that has to focus on every tool imaginable.

[00:10:17] JM: If we go a little bit deeper into the deployment, if I'm instrumenting my Kubernetes cluster with ContainIQ, what's actually happening? Are you installing agents across the infrastructure?

[00:10:32] ML: Exactly. There are three main agents. There's the eBPF daemon set. That sits on every node. That's a little bit more involved because like we have to install like the Linux headers on every node. We need like some elevated permissions as well. Then there's the – Like I mentioned earlier, the deployment, or there's a single replica deployment that's collecting all this like state information, as well as Kubernetes events and some like high-level metrics.

And then there's another daemon set that's like our logging agent that's collecting all of your logs as well. You can kind of get like in one view, like, "Hey, here's a request. Hey, here are the logs at the time of the request, or the in-context logs. And here was the metrics for this container or pod at the time of the request." And this all is rolled out as like either – You can install Helm or like a collection of YAML files.

[00:11:30] JM: I understand. And once it's deployed, does the user define all the alerting and like outliers that would be triggered? Or are there out-of-the-box alerts?

[00:11:47] ML: Yeah. We're actually in the process of rolling this out now. Basically, rolling out like pre-configured alerts based on your historical data as well as like common Kubernetes events. So, like things like out of memory errors, or node CPU limit reached, or node memory limit reached.

Funny enough, yeah, it's something we're like actively working on now, hopefully out in the sprint. Because it's definitely difficult for people to kind of figure out like good baselines to set like for their alerts.

[00:12:20] JM: Do you have your own storage backend for tracking all the data that aggregates to monitoring issues and logging data?

[00:12:29] ML: Yeah. We use Clickhouse for like OLAP database, which is it's been amazing to work with. Like just the performance of Clickhouse has been super impressive. And some of like the metric queries you can do with it just blows a lot of the other tooling like out of the water. It's great at aggregation queries. But yeah, so all of the metric crunching, looking at like, "Hey, what's the p95 latency for this endpoint over the past X minutes?" right? That's all done through Clickhouse.

[00:13:04] JM: Why did you pick Clickhouse?

[00:13:06] ML: Funny enough, I started reading about it on like a bunch of Hacker News articles. That's kind of like what got me interested. And then I've read through a bunch of like benchmarks against some of the other time series databases out there. And I ran a few of my own. And it just seemed to blow everything else out of the water. The compression is amazing. The query latency was great. It was a breeze to work with. Yeah, kind of all of those things rolled up. It's what led me down that path.

[00:13:37] JM: Do you have a sense for how it compares to the other – I guess, I guess the biggest Clickhouse competitor is like kind of Snowflake, right?

[00:13:51] ML: Yeah you see a time scale gets compared to them a lot as well. That's at least what all the articles seem to compare against Clickhouse. If you're Googling like the Clickhouse comparisons, it's generally against time scale. Yeah, any of the other like time series DBs, they get compared to a lot. Even something like a Prometheus and Thanos. And there's other time series based databases as well. But, generally, just from like reading blog articles, it seems to always be against time scale.

And just from like the internal testing I did and some of the benchmarks I've seen other people publish, it seemed to do a much better job than most of the other tools on the market.

I guess like the nice thing about timescale is that it's a Postgres extension. So you're working with something you might already be using in a lot of cases. Clickhouse, definitely – I mean, there's going to be some added overhead of managing another database. It's not easy setting up like replication and all that stuff. But I think the performance, it's definitely worth it if you're crunching a ton of data.

[00:14:59] JM: How do you decide when to perform operations such as like aggregations on – Or maybe you could just tell me some of the operations you're placing over the Clickhouse infrastructure. Obviously, you're logging a lot of events and monitoring data into the Clickhouse database. But what are some of the operations you're performing to do aggregations and rollups and create some meaningful value out of the high volume of data?

[00:15:28] ML: Yeah. You can do some like cool things with like the newer materialized views. But for the most part, we're leveraging your stereotypical date trunk with like their quartile and like p95 functions that are already built in to give you most of the information you're looking for.

A lot of the like aggregation type queries that we do, Clickhouse kind of just has the functionality built in. Whether it's grabbing the average for a metric over a given time period. Or it's day trunking all the metrics by minute and then getting like the p95 value for that metric by that minute. Most of the functionality you're going to need is built in, really.

[00:16:18] JM: If you take the monitoring that I would have from just a Prometheus installation and compare it to what ContainIQ gives you, can you talk a little bit more about that comparison?

[00:16:33] ML: Yeah, I would say the biggest difference is that we're capturing all of like the network requests inside of your cluster. You install ContainIQ and you can say, "Hey, what's the p95 latency for our auth endpoint in our Node.js application?" We can provide that with like a one line install. And we can do that across every single application in your cluster.

Kind of the metrics we provide are like a nice to have on top of that. So you can see, "Hey, what's the – We're seeing a spike in p95 latency for this end point. Was there also a corresponding spike in CPU or memory?" But the core value we're providing is we're capturing every single network requests. We're rolling out like even MySQL and support for Postgres queries as well so that you can basically time every request in your cluster, whether it's internal, external, pod to pod, whatever it may be.

[00:17:30] JM: Have you had any interesting challenges around monitoring really, really big Kubernetes installations? Like, really, really wide platforms?

[00:17:41] ML: Yeah, we have I guess more than issues with like network throughput. People who are just had – I mean, had a ton of requests going in and out of the cluster at any given moment. And I actually found like something related to like the wake-up time of like in the core of like lib eBPF that we had to kind of like configure a little bit. And actually, like right as we were getting ready to patch, somebody else opened a patch for it.

Yeah, the hardest thing is we're capturing all of these – If you're capturing every single HTTP request or like SQL request and like large clusters that are seeing a ton of activity, it's very difficult in the beginning for our tool to be able to keep up with those requests. And so, we had to mess around with, like I said, the wake-up events with like the size of the buffer when we're pushing all of this out to user space. There's a lot of kind of optimizations that went into that.

And fred, who's like one of our engineers, and he does an awesome job with a lot of the BPF stuff. Like he's even got like a calculation now like how many cycles does it take the process, like per byte of TCP traffic.

[00:19:00] JM: Does the work that you're doing feel hard enough to be defensible as a company? Do you feel like what you've built with eBPF is so difficult? Or maybe you can enumerate some more of the difficulties that it's not going to just be built into whatever name your AWS monitoring service?

[00:19:28] ML: That's a great question. I think what we built is definitely hard. There's a ton of smart engineers out there. And I think most people – With enough effort and bodies thrown at something, really, anything is kind of possible. I don't want to toot our own horn and say that nobody could replicate it. But, I mean, it did take a lot of work. Really, again, I want to give a shout out to Fred. He's like our kernel engineer. And he's made a ton of progress with me on this as well.

I definitely think we're ahead of most of the other tools out there other than maybe something like Pixie. But we kind of have to take advantage of like the – Kind of the, I guess, like lead we're building and what we're building here so that we can maintain it.

[00:20:14] JM: It feels like UX is often a differentiator in these kind of tools, or developer experience. Have there been opportunities where you've made the developer experience really shine?

[00:20:27] ML: Yeah, I think it's core to what we're building, because I believe that if you focus on like a singular problem, you can do a much better job solving it. And so, by focusing on just Kubernetes clusters, we can do a better job solving monitoring for that.

With a tool that does everything, it's just going to be more confusing to use, harder to understand, more cluttered, and can't provide like as many out of the box like sane defaults, because they're just handling too many use cases. And so, we're singularly focused on just this one use case, Kubernetes. And we try to do everything that you're going to need for your Kubernetes cluster, whether it's metrics logs, traces, latency, events. Try to really give you

everything you'll need like out of the box. And I think like the core reason we can do that is because we focus in on this one specific use case.

[00:21:26] JM: I mean, there's entire companies devoted to each of those categories; metrics, logs and traces. We can focus on distributed tracing, for example. I know how difficult it is to build distributed tracing infrastructure. How did you build that almost as just like a feature?

[00:21:44] ML: Full disclosure, I'm sure the tools that focus only on that one specific thing probably have like deeper functionality in that given, I guess, like set of features. They're going to be deeper in like the kind of one specific thing. But like what we're really focused on in a high-level is like here's one helm chart. You install it. And we get you most of what you need without any of like the configuration. It's not going to be like maybe as deep feature-wise. But we try to get you like the core value without like any of the upfront instrumentation work.

[00:22:25] JM: There's two questions I want to ask. One is around features. And one is around architecture. How do you think the monitoring features that you'll need to build will change? And how will your internal architecture change?

[00:22:40] ML: That's a great question. I can talk you through like our product roadmap a little bit. But kind of the next step of things we're building is like profiling. Like using eBPF. And then like eventually being able to hook into the different run times to get information on maybe the framework you're using and pulling out interesting information from that. If we – Like a high-level, could hook into like the function entry or exit in Python. We could see every function call. And how long that specific function took to execute?

And so, I think there's an added overhead with that. But there's some really cool things you could do with that where you could tell people latency for each function call in a given pod over time, as well as like, in some cases, even like the arguments of that function call. I think that's kind of like the pipe dream, is we could do that. We could associate it with like all of the other metadata we're collecting.

In the near future, we're really focused on adding like profiling and like flame graphs, again, using eBPF. So that you can – You install something into your cluster. And we're automatically –

Based on the config options, going to be profiling whatever applications you want us to do. And so, again, getting you like that kind of core value without any of the instrumentation work of like you don't have to install the profiler in your application, download that NPM package. We do it automatically.

[00:24:19] JM: Can you tell me more about your own deployment? What your infrastructure looks like? And are you using Kubernetes yourself? Or are you managing to use like a lot of serverless functions? Or just tell me what a typical deployment looks like.

[00:24:36] ML: Yeah. On our end, we're using Kafka for like our kind of ingestion and ETL pipeline. All the information we're getting from our users gets dumped into like various Kafka topics. And then we have kind of like two main consumers, one for alerting, one for kind of like dumping that data into the various data stores and some light ETL operations. And so, those are in Go.

And then our backend is like a Node.js backend. Pretty standard and express. React on the frontend for like all of the visualization. Postgres for like transactional type queries. Think like, I don't know, user management and state updates. Like, which pods are active? And how the conditions changed? And then Clickhouse for metric aggregations, log storage and retrieval. Yeah, any like analytic type query.

[00:25:41] JM: Can you talk me through the data flow between a metric logged on the user side, stored in Clickhouse and then sent to a dashboard? Give me the end-to-end data flow.

[00:25:58] ML: Yeah. I guess like the first step is our agent like collects one of the metrics or traces. We will associate the Kubernetes metadata with that. In the trace example, we'll say, "Okay, this is the IP of the connection. What pad belongs to that IP?" And then we can attach like the pod, the service and deployment that belongs to it. That gets sent to our systems, or our producer, which based on where the data is coming from, dumps it into like the corresponding topic.

Then like I said, we have the two consumers that pop the data off that topics. One of them is Go-based. One is a JavaScript-based. The Go-based one is used for like alerting. And so, it will

basically pop the metric data off and say like, "Hey, does this match any of the alerts they've set up?" Whether that's like a log alert. It's like, "Hey, has this log line occurred greater than X times over the past Y minutes?" It's a metric. Like, it's your stereotypical like threshold or percentage change-based alerts. Or if it's a trace, it's calculating like the latency.

On the other end, we've got the other consumer that dumps it into the storage systems. Based on the topic it's in, we know what to do with the data. If it's like coming off of trace topic, we can you dump it into like the trace table, essentially, in Clickhouse. If it's like a state update, which would be like, "Hey, this pod was deleted. Or this pod had its CPU limit changed. Or like its condition went from this to this." That will get sent to like Postgres where we'll update the like state of that pod currently.

A lot of the flow comes from basically which topic it's coming off with. And then the consumer has logic built in to say, "Based on that topic, what should I do with it?" And, yeah, then like the node express backend was querying all of that data. I guess, one thing I missed earlier too is we have like a go-based microservice that sits in front of Clickhouse for like all of the Clickhouse queries. And it communicates with GRPC to our node app.

From like a user's perspective, say, for example, you search, "Show me the p95 latency by minute for our /auth /whatever endpoint." That will send the request from React to express node. And then that will open up like a GRPC connection with our Clickhouse operator, which communicates kind of like the query of what you're looking for. And then the Clickhouse operator will ping Clickhouse and say like, "Hey, here's the data they requested." It returns it back to node, which returns it back to the frontend.

[00:29:08] JM: What are the areas of the architecture that have the most load placed on them that that to scale the most regularly?

[00:29:16] ML: Yeah. The biggest concern for us is always the consumers, right? Because if we see a spike in consumer lag, that means that we're not ingesting the data fast enough. That's my biggest worry, right? If I get an alert about consumer lag, that's what's waking me up in the middle of the night. That's my biggest fear.

In terms of like the outside of that, like the Clickhouse operator, like the Go microservice I was talking about earlier, that's another big concern. Because all the inserts go through that as well. We need to be able to scale up insults if we're getting push – Or insert. Sorry. For being pushed out of data. Measuring like the latency there as well is like super important.

The actual like aggregation queries, we don't see a ton of latency issues there. It's really on like the kind of insertion and consumption of data. We have to be the most careful about.

[00:30:19] JM: Are there places where you feel the infrastructure is potentially subject to bugs? Like are there some particular areas of the product that you worry bugs might emerge in? I think there's like – Every product has its domains where bugs are prone to cropping up. Are there any places where you have some canonical bugs that keep coming up?

[00:30:49] ML: I mean, every software's got bugs, right? Part of what we're working with. But, yeah. I would say the hardest bugs to deal with are the eBPF-related functionality in other people's clusters, because it's not something that like we have direct insight into. And there's just some, there's changes between kernel versions. Last week, we spent a few days getting it to work on like 4.19. And it's hard to do when you don't have direct access. That's something that like we've definitely, I guess, fixed a bunch of bugs there before. But it's almost not even on new functionality. It's more so getting it to work across like every kernel version.

Most of our ingestion pipeline has been relatively tight, because we spent a lot of time on that, and a lot of code reviews. Maybe like your standard bugs in the UI and in node. But the biggest ones being bugs that happen on other people's infrastructure because they might have a very weird setup, right? They're using a CNI that no one else uses. And they're using this weird kernel version that's like super old with a different CRI than we're used to.

We're actually focusing now on building out a bunch of tools to test across like the myriad of different configurations you could have. We have a Kubernetes cluster that has multiple different – Each node is like a different kernel version. So that we can like install our daemon set across all of them and make sure it works everywhere.

[00:32:29] JM: Is there a feedback loop between monitoring data in ContainIQ and being able to trigger changes across your infrastructure? Like spinning up a new container in response to some alert that emerges from ContainIQ?

[00:32:48] ML: No. But it's something we've been asked for and something I've been thinking about a lot, specifically around like latency. We're collecting the latency for all of your microservices, whether it's like SQL query, or it's a HTTP request. Where we send that data is configurable. It would definitely be possible for someone to install us, export all of the latency information and use that with alongside like the cluster auto scaler or some auto scaler to either auto scale the number of nodes or pods based on that latency information we're collecting.

With the metric information, a lot of that's like already possible. But I think for people who are looking to auto scale based off of latency, it's definitely something we could do. You just have to hook us and do like another system.

[00:33:41] JM: As we begin to draw to a close, what's the biggest challenge you're encountering right now building ContainIQ?

[00:33:51] ML: I'll give you kind of a jokey response than a real one after. I think something that we're showing with now is keeping up with all the feature requests from like users. It's like, "Hey, I would love to have this as well." Just keeping up with user requests and keeping our current users happy. And that's something near and dear to me, because I'm a people pleaser. So, just making sure like everything's working and everyone's happy. I think that's always going to be a struggle for me and a struggle for us as a company is just pleasing everybody.

From a technical standpoint, I think one of the coolest problems we're solving right now is we are basically parsing out SSL connections. Say, you have encrypted traffic between pods. Something we're now able to do is – In one of two ways. Either adding a uprobe to open SSL to hook into their write functions when they are encoding the packets. Or if you're using like ephemeral keys for like the actual encryption, we can pin those keys and use that to like decrypt the traffic.

Yeah, in summary, the technical answer is parsing out socket buffer from SSL connections. And the personal answer is just keeping up with all the different user requests.

[00:35:13] JM: Cool. Well, thanks for coming on the show, and it's been real pleasure. Best of luck with ContainIQ.

[00:35:19] ML: Thanks so much. It was great meeting you and talking to you, and really enjoyed the conversation.

[END]