EPISODE 1249

[INTRODUCTION]

[00:00:00] JM: Large portions of software development budgets are dedicated to testing code. A new component may take weeks to thoroughly test and even then mistakes happen. If you consider software defects as security issues, then the concern goes well beyond an application temporarily crashing. Even minor bugs can cost companies a lot of time to locate the bug, resolve it, retested in lower environments and then deploying it back into production. The company Speedscale provides an intelligent Kubernetes-friendly testing toolkit that runs at build time. Their virtual SRE bot works inside automated release pipelines to forecast and test real world conditions that the new code will encounter. This process requires no manual scripting, because Speedscale uses existing traffic to generate tests and mocks. The feedback is immediate after every build, and covers regression, performance, fuzzing and chaos tests automatically. In this episode, we talk with Ken Ahrens, and Matt LeRay. Ken is a founder and the CEO at Speedscale. Previously, Ken worked at New Relic as a senior director, solutions architects. Matt is a founder and CTO at Speedscale. He previously was the VP of product at Observe. We discuss testing in distributed environments, how Speedscale intelligently tests and mocks during builds, Kubernetes, and their future goals was Speedscale.

[INTERVIEW]

[00:01:24] JM: Guys, welcome to the show.

[00:01:25] KMT: Thanks a lot, Jeffrey.

[00:01:26] MT: Great to be here.

[00:01:28] JM: So you work on Speedscale. And we'll get into what Speedscale is. But I think of it as fitting into a lineage of products that are kind of like load testing, or just automated testing systems that you can use on your application to give an idea of how it's going to perform in the real world in a way that perhaps things like unit testing wouldn't accomplish. So I guess I'd like

to start off by maybe putting what you're building in historical context, how it compares to systems like load testing?

[00:02:12] KMT: Yeah, sure, I'll take this one. I think that this is a very common piece of feedback that we get. This looks like a load testing tool. I personally worked in the testing space previously for a company called ITKO. And we did some functional testing and load testing, and there are certainly some overlap, because when you're doing traffic replay, you're trying to put load on the system and understand its performance. So from that standpoint, there's absolutely some similarity, there's hundreds of tools in this space. It's been dominated in the last several years with open source. But of course, we also believe that there're reasons why those types of tools and systems don't work. And that's part of why we've created Speedscale.

[00:03:02] JM: So you do think the analogy holds some truth?

[00:03:05] KMT: Well, yeah. Of course, it holds some truth. There's a bunch of overlaps with load and performance testing. We're really focused, I think, on gaps and things that you can't get if you want to sit down and try to load up your system. And increasingly, in these kind of large deployment environments, you can't take a simple novel tool and say, “Hey, load me up. Hit me up with the exact same call 100 times and now I know the scalability of my system.” That's not going to work.

[00:03:35] JM: So let's talk a little bit about things in a modern context. So obviously, everything's on the cloud. We've got systems like Kubernetes. Or you could use GKE or EKS. The programming primitives, the release primitives are in large part different than they were five years ago. I think it's a little bit of a generational change from EC2 instances or obviously on-prem deployments. And obviously, whenever you have a generational turnover in infrastructure you have a generational change in the tools. Like the load testing tools of last year's infrastructure don't make as much sense as this year's infrastructure. Are there some paradigmatic changes in the world of infrastructure that are tailwinds for Speedscale?

[00:04:34] MT: Yeah, so one of the biggest ones, which you've already touched on is containerization, and specifically Kubernetes. But we shave a little bit at the term load testing, because there's a class of tool that are different than what we do. But it does solve the same

problem in a lot of ways. It says, when we send this application to production, is it going to slow down or have other kinds of problems? So yeah, I'd say that one of the things that we were super excited about in starting Speedscale was the shift to Kubernetes. I think there was a recent report from Andreessen that said that 80% of enterprises. Is that right, Ken? 80% of enterprises are utilizing Kubernetes in some way. And one of the things that really helps with what we do at Speedscale is that it's really two things. One is it increases the iteration speed. So there's more deployments going out constantly and smaller units of work are fitting into each one of them. That's actually super helpful for our approach.

And then the second one is that there's more communication going on between services, because what happens when you go to Kubernetes, is you shrink all the units of work and so that you can scale horizontally. So you end up with microservices or whatever. However you approach that, right? You end up with container, containerization. So what happens is, a lot of times the problem – The problem, the monolith world, was a lot of times inside of the monolith, right? So you valued this sort of deep visibility into getting like stack traces and looking all the way down. And what we've seen with Kubernetes and containerization is now a lot of the problems that arise in production systems are actually in the spaces between components, the communication between components, which is why we're pretty excited about applying this concept of traffic replay between the containers to understand their load performance in production. So that's not really possible without containerization. So we're definitely riding that wave.

[00:06:24] JM: So if I understood correctly, you're saying that understanding the communication path and the communication performance in two containers talking to one another is something that you're focused on?

[00:06:39] MT: Yeah, that's one of the things we're focused on. Yeah. And again, we'll probably get into talking about Speedscale in specific. What it does more as we go along. But one of the first things that we struggle as engineers with is understanding how our processes communicate with other processes. So that's the first thing that Speedscale tackles. We're not a monitoring tool for production. We're not like any of the major monitoring vendors or any of that. But what we do do is we show detailed down to the payload information of the API calls going between two containers. And what that allows us to do, like, first off, is just illuminate what that container

is doing, both who it's calling to, as well as what calls are coming into it. And so that's kind of the basis for everything else that we do, and Kubernetes makes that a lot easier to do.

[00:07:35] JM: Okay, so I guess we can just dive into Speedscale. Tell me what problem you're solving in a little bit more detail, and what the insertion point is for a typical application.

[00:07:50] KA: Yeah. So the problem that we're working on is despite huge advantages in modern technologies, and cloud, and the pace that everyone wants to release things, we still commonly have a lot of incidents and make their way to production. It's too hard to understand how this code is going to behave when you have lots and lots of services, lots of different teams, and everyone is trying to throw their changes into the pot. So what we do at Speedscale is we let you separate the pieces out. If you're a team that's only working on the ordering system and you don't own the customer backend and you don't own the third party API's, we actually let you separate yourselves from those and understand your own performance profile. We produce what you would think of as SLO kind of metrics. Very simple metrics, latency, throughput, error rate, kind of the infrastructure. How much infrastructure is needed to run it? And we give you that data in pre-prod before you release it to production, where if you've made a mistake, it's hard to fix.

[00:08:56] JM: So give me an example of an issue that Speedscale would help to capture and prevent.

[00:09:07] KA: So let's say we're updating that order system. Maybe we're unlucky. It's written in Java. And we just go, “Hey, we want to update one of these Java dependencies.” We go into our Maven setting and say, “Grab the next version,” but that has some synchronized code in it that we're calling that's going to cause a performance degradation. We haven't made any other change.

Typically, this goes through, “Hey, we do a peer review. This is a real simple change. Let's approve it.” Build the container, all your unit tests and things have passed. And when you push it to production, you'll find out a little bit later that suddenly the system has started running more slowly. And people have to try to figure out, “What's changed? No code change,” bla-bla-bla. With Speedscale, we will literally create that event that you get in production, tons of love,

whatever is your throughput goal, we will push that in pre-production. You didn't have to write a test case. You didn't have to Sit down and try to think of all the ways that API is called. We know all the API is called, because we built this out of a traffic capture out of the real traffic that was used in the system. And we prevent that degradation of service, or incident, or outage from really happening and affecting customers.

[00:10:20] MT: Just kind of extending on that a little bit. Because we're doing that that traffic replay, , Jeff, to answer your question kind of specifically, we can ramp up and download by using clever tricks. Like if we only saw 100 users in the production system, we can make that 1000, right? And so we can start to uncover problems like concurrency, or race conditions, or hard work, inappropriately sized clusters or nodes, we can go and find resource leads, etc., etc. And we can do that without the pain of hitting like file new test, right? Without the developer actually having to think about what test cases do I want? What do I want my null testing to be? What do I want all these other – I don't have to design these tests. And so that's the kind of issues we find just by replay with automation.

[00:11:11] JM: So tell me a little bit more about what happens on the Speedscale side like when a test is being executed. So what happens like in the pre-release process on your side when you're on your infrastructure? Like what's being stood up and what's actually happening on your side?

[00:11:32] KA: Yes, so this part is actually pretty clever. We have in a Kubernetes environment, we have a Kubernetes operator. And we register to listen for deployments. So you're deploying the next version of order API, and you put an annotation on there that has a Speedscale instruction. It says, "Run this scenario." It's a named scenario. And like Matt mentioned, where maybe you want to do the 10 times the amount of traffic that was recorded. So that's like a configuration. You add those two pieces of data in your annotation. Our operator notices that you're doing that, and we prepare an environment. And this is actually run in the customer environment. So it's on, they choose the infrastructure to run it in. But we'll actually pull down the traffic and say, "Oh, you're making a third-party call." Like have a customer who's who makes calls to the Google Gmail API into Office 365. We'll pull down the correct traffic and we will actually mock out those on the fly. And we call that the responder when. When the responder is ready, then the application starts your deployment that you initially deployed. And

then once the deployment is ready, which we know from the liveliness probe, then we start the replay into the pod, that's called the generator. So we kind of have a sandwich with on one side we're playing transactions in. You have your code in the middle. And on the other side, we're taking care of all your dependencies. And when the thing is complete, we clean it all up. So default state is everything's off.

[00:13:11] JM: Naive question, how is this any different than just unit testing? Like how is this different than just standing up a service in staging and testing your battery of unit tests against it?

[00:13:31] KA: It's pretty different from unit testing. We write unit tests. We love unit tests. It’s fantastic. Unit tests are focused on your algorithm, right?

[00:13:40] JM: Sorry. I guess I should say integration tests would be the thing that it would be more similar to.

[00:13:44] KA: Yeah. Yeah. Yeah. Okay. So this is a type of an integration test. So in the customers we're talking to, they typically have to build the entire end-to-end environment. And so you have to build the whole thing. And then the integration test is done at the GUI tier. And increasingly, what you're seeing, and especially when you move to Kubernetes, is a lots and lots of your code is actually in the backend. And it's buried in these various services. And they're all owned by different teams. So I'm a team whose five steps away from the GUI tier. And every once in a while something that happens on the GUI causes a call into my API. It’s is very hard to understand. If I'm making a change, is my code change going to have a performance impact? Are we going to load up the entire cluster so that 1% of the calls weaves their way over to my API? Because if I have a performance problem, it's hard to detect it that way.

So instead, what we're doing is we're disintegrating the problem down into small chunks that each service gets its full treatment. Instead of building the entire thing and trying to test it as one giant unit, that’s not how they're developing the software. They're developing it as the separate components. Each component gets its own full test harness.

[00:15:00] JM: Does the Speedscale eliminate the need for other forms of integration testing.

[00:15:09] MT: So, Speedscale eliminates a certain percentage of integration testing, right? So one of the problems of integration tests, as Ken said, is that it is rare, or it's very difficult to have a correct copy of production data, right? And then every time you run through your test battery, you are going to have to reset the databases and get them back to their initial state. You're going to have to run up your Gmail bill, if you're – Let's say, you're sending emails, right? You're actually going to have to set some sort of tests with the Gmail API, or Office 365 API, etc.

So for many types of integration tests, the Speedscale, sort of playback of responses we've already seen, and that sandwiches Ken talked about, is actually good enough. Now, it's disingenuous to say that it replaces all integration tests, because there're still some things that you can't do, other than getting the entire system together. If you've made significant API contract changes, or whole re-architectures, or big, big, big shifts, obviously, you still need to do integration tests.

What we're really trying to do in some ways is make it so you don't have to run that integration test very often. And you can think of this sort of system or container to container test happening almost as part of the CI system. You could theoretically do it on every PR if you want it to, or every merge request or whatever. Yes. So Speedscales can sort of run in the background and give you 60%, 70% of an integration test. But there's still room for an integration test in the world. Ken, I don't know if you'd add anything to that.

[00:16:41] KA: Yeah, I would add to it that there's a big use case around the responders. This is a key enabling technology that enables all different types of testing, actually, including things like chaos testing where you say, "What if I make a call to this third party, but the third party has a problem?" What if I'm making a call to a payment API, and it fails? How does my system respond?" So we can actually force those failure modes and error conditions to happen as well. So in a way, we're also expanding the test capabilities and the types of things that you're going to run into. Another reason why I think it's a little bit of an alternative to that integration testing, is the real world is messy, the real world of production has gross things happening, it has a robot that's crawling you. We have a customer whose Promethium is always hitting the wrong endpoint with the wrong configuration and it causes an error. That is a normal condition that exists in production, and you should replicate it. And a developer would never sit down and

make a weird test like that. But when we record the traffic, we will play it back that way. And if that error happens to cause a problem with your code, you'd be happy that you found it in pre-prod and not after the release.

[00:18:01] JM: So you mentioned chaos testing. How does this kind of integration testing compared to chaos testing?

[00:18:11] MT: So actually, chaos testing at the application layer is part of what we do. So this is not the same thing is like the early Netflix stuff, where we just shut down a server. What we do instead is we do chaos at the transaction level. So for example, let's say that I'm on call, I'm testing a service, and that service is making a call to a backend database. What we have the ability to do at Speedscale is randomly force one transaction out of 10 to slow down, or all transactions to slow down, or give a bad response once in a blue moon like a road transaction. All the things that happen in production, but are very, very hard to simulate. So that's actually a chaos feature that we have.

So in one sense, it's different than infrastructure chaos. And that you're not just playing whack a mole with servers and things like that. But it's something that at least I'm not aware of. You'd have to write it yourself. But it's sort of built into our product where you can go and say, “What's going to happen if I start throwing four fours where I used to get to hundreds? How resilient is my system at the application transaction level?”

[00:19:20] JM: So you have a system where you can store traffic, and then replay traffic so that basically stakeholders can see what happens for actual sessions, for actual traffic replay. Can you explain like what is the worth of traffic replay? Like, why is that core to what you do and who does it provide value for?

[00:19:49] KA: So it's good that you mentioned this because we actually see two places where value is coming from with our platform. We kind of sprinkled in both in, but just to sort of summarize. There's once added value, which is, what is the data flowing through my application? When a call comes in, what's in it? What's in the HTTP headers? Is there a body? What's in the body? Why does this XML have all this extra junk in it? That's XML for you. Why am I getting someone hitting my API with the old definition? I had a customer who said – We still

have a customer hitting us with a six year old version of the API request, and we still support it. Guess what? None of the new developers test with the six year old version of the code.

So one part of what we deliver and offer, Matt mentioned this earlier, is visibility into what's going on. We tell you all the inbound transactions. We show you all the outbound transactions. And we have a little analysis mode, which will tell you here're protocols, technologies that you're using. It's a mix. We had a customer who – We've actually had multiple who said, “That's not connected to that.” We didn't make it up. It's what was observed. Or they said, “I didn't realize for the same backend API, I have to call three different host names.” Well, that's your backend API vendor. That's how it works. So there's one whole aspect, which is just getting more visibility. It's not aggregated. It's not summarized. It's not being filtered in some way. It's very pure, and it gives you a lot of data.

And then the second part is a use case of what we do with this data is we'd let you replay it. So instead of sitting down and writing tests, replay the traffic from 15 minutes ago in and production against the next version of the code. It lets you answer the question, “What if I deployed this right now? What would it look like? How would this exact one, you can look at your monitoring tool and what the performance profile is, and you can compare it to what you play back?” And that comes out? It looks like a report. It looks a lot like a monitoring system? What your response time and throughput and stuff? So we see two different value areas, if that makes sense. One around visibility and seeing what's happening, and this other one around, kind of preventing production incidents by replaying what they're about to experience.

[00:22:09] MT: So can I add on top of that a little bit, is one of our customers, just to make it concrete, they call a backend API that is – They call a back end API that's very, very expensive. It's actually the Gmail and Office 365 API, as I mentioned before, and they call it with thousands and thousands and thousands of calls. And the issue that they get into is that they're having a performance problem. They know about the performance problem, right? They know they have something wrong, but it's incredibly hard to fix. And it's hard to fix, because they can't replicate the production environment. And they can't cost effectively test against those backend API's.

So as Ken mentioned, with the responder service, what they do is they basically make a copy of those API's based on the traffic that we've recorded, and move it to the developer’s desktop,

right? So every developer, 50 developers, each have their very own copy of production, Gmail and Office 365 that they can bounce against. And so it allows them to very rapidly test like the fixes to their code. So they go in and they say, “Okay, I want to speed things up. Let me try this. Okay, now run it against those backend services,” and it doesn't incur any cost because the responder is like a container that they can run themselves, right? It's all real data. It responds just like the production system does.

And so they go through this rapid iteration loop where they can make – I think they claim to 500% performance increase very, very quickly, because they're not relying on the systems that are flaky and falling apart or systems, they're going to get metered on or throttled, or etc. So it kind of moves the production environment on to the developers desktop.

[00:23:46] JM: I guess I lost the lead a little bit there. So if I'm – Let's say I'm a service owner in an organization, and I've got other dependencies that I'm hitting with my service. And those downstream dependencies might be like an external API, like the Gmail API. Instead of having to hit the actual Gmail API, you're giving me a mocked container that replicates what the Gmail API would be doing?

[00:24:23] MT: You got it. And more specifically, it's the exact calls. It's aware of the exact transactions and payloads that the system in production sent to Gmail. So it's not like a Rube Goldberg machine, where you send a request, and we have some code that runs, a mock that we made up, all these other things. It's actually going to play back what we recorded. So if you say – If your production system says, “I sent an email to my user number one, and then I got a response back that said, “Yes, we got that.” And then another email came in. All of that is going to be replicated in that container that the developer has access to.

[00:25:02] KA: I think that this is the key. Hitting the Gmail API one time, there's no problem, right? The issue is the data that's in it. So specifically, what we did is we actually went and recorded like three different email addresses the testing team had set up. They each had thousands of emails, calendar items all over their calendars, and tasks, all these kind of features you use in Gmail. Then we flipped the switch and said, “You can send any email address you want and we'll respond with one of those three.” And this enabled them to do a 10,000 email address, which would have taken them weeks to generate all that test, fake test data. There's

also no tear down, no reset, or anything, it's just running in this little Docker container for them. So it's not just that I can call the API and get a response, but that it has the correct response, it has a realistic response. When we did Office 365, you create all these folders, the test account had like 300 folders in it, and each of them had emails in them. So we didn't guess it or try to supply data. It was real data from the real life system.

[00:26:12] JM: So is it basically the goal is ensuring that my guy communication between my the service that I own and the downstream service that I'm calling, is it ensuring that the compatibility doesn't break?

[00:26:34] KA: So if you go back to Matt's use case, there're a couple things. One is you definitely don't want your compatibility to break. But the second one is you want to iterate fast, and you want to try this, try that, run experiments, do whatever. And if you think of the sort of pain of merging to master, getting a bill, getting it deployed into a staging environment, running an integration test, and all this stuff, and when that takes hours and days, you're not going to run a lot of experiments. When I can run it locally on my own machine, or in my own cluster, I'm going to run a lot of experiments.

And this is why they got this huge cycle time improvement. And actually, that customer in particular was doing a major refactor of their code. They were taking a big monolith, that they were breaking it into microservices. It was moving from EC2 instances into Kubernetes clusters. And this whole thing around the dependencies was a non-issue. They didn't have to worry about how they called all these other systems. And it enabled them to rapidly speed up their own application development. So it's not all a testing thing, if it makes sense. Like there's this developer efficiency that I get that I don't have to go and talk to everybody and set up all these things, or engage another testing team or something like that. I can be really productive with knowing I'm in my box and I can do everything with my code. And all the stuff outside of my responsibility is already taken care of me by Speedscale.

[00:28:04] JM: Let's talk a little bit about the engineering behind how that actually works. So you've got the traffic that I've generated from my service. That traffic is stored. I just like to know what you're doing to spin up this mock downstream service. Like what's actually happening? How is that engineered?

[00:28:32] MT: So as you said, the first step for Speedscale is we have to get what we call a traffic snapshot, okay? So that's the recording of the inbound transactions and the outbound calls. So usually, that's based on timeframe, but it can be filtered by other things, other criteria. So now we have this snapshot. Like it's like two scissors snipping a piece of ribbon, right? So we take that snapshot, and we put it through an analyzer process. Because one of the things that you figured out pretty quickly if you've ever tried to solve this problem on your own, if you've ever tried to create sophisticated mocks or sophisticated load generation, like automation, is that when things are figured out, is that there's a lot of parameterization and tokenization that needs to take place in the data in order to make it seem realistic. So sort of the easy example would be dates, right? If you've ever written an application that has JWTs, or jots, right? JWT's are used for authentication, and they are signed within a particular time frame. So let's say that that time frame is 10 minutes, right? So if you start playing all that traffic back, it's not going to work, right? And there're 100 different variations to this problem, dates, and authentication, and usernames, and all kinds of different things that have to be automated and changed.

So when we take that traffic snapshot, we put it through this analyzer process, and we parameterize everything and we prepare it to be played back. Alright? So from there, it’s like we take a pair of scissors and cut it down in the middle. And on one side, on the left side, is all the data that we're going to send to the service. So like, as you said in the beginning of this conversation, that's what we make the load test out of, okay? So that goes into we kept things everything nice and simple as we basically put it into a file, and the file gets loaded onto our generator container. The generators, like 100 generators that have been written throughout history. That's not the special sauce. But the generator is sort of a tokenized aware load generator. And it's ready to play the file back.

And it's pretty straightforward. It's JSON, it's readable and editable by the developer. Obviously, this is a tool for engineers. So everything's kind of changeable as you go along. So that's the left half, right? Is we cut it down the middle, the snapshot. So that's the left half, we do the load generation. Now the right half, when we cut the traffic half, is all the traffic that we saw coming out of the system. So that is the calls going out to the Gmail API, the Office API, or integration buses, or whatever it is, the databases, etc. And that gets downloaded in a file. Goes through the same sort of parameterization process, and then it gets loaded onto our responder

container. So those are the two main components, a responder and generator component that are loaded with this snapshot of data.

Now, once you're ready to play it back, there're kind of two modes right now that we see. Mode number one is you have access to a Kubernetes cluster and you want to go and run a new piece of code. So you go, and as Ken said earlier, you take your YAML. Let's say it's a YAML file, or a Helm chart, or whatever it may be. And you put an annotation that says, "When I deploy this, I want Speedscale to take over." And so what we do is when we see that annotation in the YAML, we basically grab control of that deployment, and we halt it for a second, right? Just a second. But we halt it and we say, "Okay, they want Speedscale to make this – We need to make this service think that it is running in a production environment."

So what we do is, inside of Kubernetes, we're kind of Kubernetes – We've gotten to be kind of – I don't think anybody's an expert on Kubernetes. But we've certainly learned a lot about it over the last year and a half. But we go into that service and that deployment, we're kind of holding it in spaces, and we make changes to the network and the configuration of it so that we can transparently make it think it's running in production. So that means we wire it to our generator on one side. So it thinks that real clients are sending things to it. Then we wire it to our responder, so that things – When it calls out to the Gmail API, for instance, that it's instead going to our responder, but it's not aware of that. And so there's a bunch of network foo that we do there, and also modifications and other things. But it's all transparent, right? It's all automated. Okay, so once we've got those things spun up, our generic responder, we've rewired everything, then we let the deployment go and it acts like a normal Kubernetes deployment. And so when it goes out, then it runs through a test and we can run whatever scenarios we're looking to run against it. So that's mode number one for deployment.

So mode number two, which we've also talked about, is more of a piecemeal approach, where a developer might use something like Docker Compose to test it on their desktop. And this is – So what that looks like for us, as we said, some of our customers do this. This is more like a developer efficiency thing. So I spend a lot of time in Go. And I'm always trying new experiments and new iterations usually to fix things that are broken, and I need like a mocked backend service that I can run against and make it feel like it's real. So in that mode, we go and we take

that responder, which has all of the data that we cut out of the snapshot, and then I can run it inside of my Docker environment, and I can use my service – Can replay it directly against it.

[00:33:42] JM: It makes me curious about what some of the key engineering problems that you've had to solve and actually making that a reality.

[00:33:52] KA: What Matt talked about, the tokenisation problem, that's a major class of challenge. And like he said, you can't just take traffic and replace it. So that is one whole class of it looks kind of like an ETL process where we're going through and figuring out, “Oh, this has to be transformed. This has to be transformed,” and so on. That's one of the big challenges. Another is actually how we're doing the capture itself. Most of the interesting services that people use are secure and have security. So we have to deal with that security, typically at the network layer. So things like dealing with the TLS certificates, getting things configured as transparent way to collect the data.

That's a pretty big set of challenges that we worked on. And I think Matt’s also kind of oversimplifying. There’s a big scale of this. We're taking advantage of kind of cloud data warehouse type of solutions around how data is stored, and run through our analyzer engine so that we can kick out these files. The file at the end looks real simple, but the process to get it is actually quite involved. So not to mention, we want to be keeping a Kubernetes microservice environment ourselves. So we're real familiar with the challenges customers have. And so we do it with one arm tied behind our back while we deploy Kubernetes and keep all that running as well.

[00:35:13] MT: I'll just double click a little bit more on those, because this is an engineering podcast. So we'll go into a few of those. So as Ken said, there're kind of three, three major category of engineering challenges that we face. The first one is the recording system, which is we have to do in some ways what a monitoring system does, except we actually have to grab the payloads. So like most of your monitoring systems, they can aggregate things. They can roll things up into like a metric that reports once a minute, or whatever it may be. We have to go and actually look at the payload. So there's, a lot of complexity around things like caching, or filtering, or deduplication and a lot of other things, right? Because you can't just do like straight dededuplication. You actually have to understand that certain parts will change and certain parts

don't. Then you have to move hundreds of gigabytes of data around, and you have to do it fast without impacting the production system. A lot of our engineering effort goes into that problem actually.

Then, as Ken said, there's a there's a storage issue, which, fortunately, a lot of what's going on in the cloud right now, we're kind of piggybacking on a lot of the cloud data warehouses are doing. In fact, one of the big things we get from engineers is they say – Especially when they started Speedscale, they say, "This seems impossible. It seems impossible to record this amount of data." And then we'll say, "Well, actually, look at all the advances that have been made for other thing like – Whatever, right? analytics tools and other things," and then it's made a lot of this stuff really possible where it wasn't before.

And then the other category of problem, honestly, it's just that Kubernetes is a fantastic system, right? It's a fantastic platform. There's a reason so many people are going to it. But it also is something that changes a great deal. So as you can imagine, we're integrated into the Kubernetes – We're integrated pretty tightly into Kubernetes. Much like Istio would be, like some of the service meshes. So getting along with service meshes and writing the operator and other things that make it work, that's just a massive – Like it's like the Batman decoder ring. Just we're constantly finding new combinations, and making sure everything's good. Now, the good thing is, is that there's an 80-20 rule to that where we hit a lot of the issues and then they don't really repeat, right? We find those. But there's always some work to be done there. So that's why we're hiring. But anyway, there's no shortage of engineering challenges.

[00:37:28] JM: Okay, now, I'm pretty curious about the data side of things. So you mentioned you're using cloud data warehouses, and that's to store the traffic data. Is that right?

[00:37:39] MT: So there's several different kinds of data that we – Like every other system, right? But the raw data, the traffic data? Yeah, that's what we're using. Yeah, that's one piece of it.

[00:37:51] JM: And are you using the cloud data warehouses as like transactional data? Because it doesn't sound like you're using them like analytical databases. You're not

aggregating them or anything. You're actually using the traffic data that's stored in the data warehouse.

[00:38:06] MT: Yeah. So without getting too much into the secret sauce, I guess, in some ways or what's going on cloud side, because it's constantly evolving. But what we try to do is create the fastest pipeline from the network to disk, at least initially. And that's a lot of our engineering effort is getting the fastest pipeline straight to disk. Then when have, as Ken mentioned earlier, we have kind of like an ETL pipeline that goes and transforms the data over certain stages. And it's like a large funnel, right? So we have custom processes written in Go Lang that will go and narrow the data down and parameterize it until we can work on it in a traditional database format. So what you said, the question I think you asked is like can you apply the analytics database? Yes. But after we go through a few stages of the funnel, we'll use that to make a performance report or a chaos report to show you, “Hey, here're all the chaotic things that we did,” stuff like that. But that's sort of the end result. And that's where we really take advantage of the massive horsepower that a lot of these cloud systems provide.

[00:39:08] JM: Wait. So say more about – You're trying to find the fastest path to disk for this data. What do you mean by that? Why is that important?

[00:39:17] MT: So most of the environments, or not most, but some of the environments we run into have very high transaction throughput. So if you install Speedscale in our recorder, right? Yeah, if you call our proxy service, or whatever it may be, it installs like Istio, right? It's just like you put an operator on. It’s pretty transparent. It takes care of things for you. Just add an annotation. Most of those systems have high-data volume to start with, right? So if it's an order system, as Ken mentioned, that's one of – People love putting us on the order system for some reason. If you’d put us on an order system, if it's a successful business, there are going to be gigs and gigs of data that are flowing through both sides of the application. So duplicating all of that could be extremely heavy, right? Duplicating payloads.

So a lot of what we – Kind of what we spend our time working on, is figuring out clever ways to filter out what we record and also so that it still is realistic and can be reproduced, but filtering out data that we have, and then getting everything to disk as quickly as possible and then coming up with data management strategies. So like you can't keep stuff around forever. What

parts of the data do you keep on the customer's cluster? Right? Because if we're recording to disk on the cluster, what parts do you keep on the cluster, or send to our central repository? Things like that. I can't really sum it up in a 10-minute conversation, but there's a lot of work that we do around figuring out those formulas, if you will.

[00:40:47] JM: Interesting. So you're basically developing an entire, like, tiered storage system?

[00:40:56] MT: Yes, which is great for our customers, but much to my pain. Yes, we're selling like a tiered storage system. Yeah, it's like a funnel. We think of it like a funnel, at least. Some things can be discarded, like a ring buffer, and then we got to kind of move up the stack from there, and get progressively more detailed in what we actually keep around. Otherwise, you don't want to go bankrupt. I hear that's bad for startups to go bankrupt. So we try not to do that.

[00:41:22] JM: Just help me understand, like what are the – So if you think about the tiered storage system, and like some of it is fast to access and expensive, some of it is slow to access and cheap. What do you need along those tiers? Like, what kinds of data for your customers do you need to keep along those tears?

[00:41:38] MT: So what we store at the – And actually we're still developing this. So we are only a year and change old. So it's not like we have a sweet, a perfect answer for all of these things. But when we think about what we keep in the cluster, that is a full fidelity copy is the way to think about, right? We're going to dump it. Try to keep it as full fidelity as possible. And then what we're looking for in our [inaudible 00:42:01] service, which is like an agent. It's not actually an agent, but it's just a pod that runs in the Kubernetes service. What it's looking for is patterns, right? Like filtering, like things that we can filter on, and recognize the pattern instead of having to send everything up, right?

So from there, we send it over a transport mechanism, which is in Flux, right? But with our early customers, it was AWS Firehose, because that's just what's easiest to implement at the time. But we set things up to – Send out to AWS Firehose, and then we go and say, "Okay, what parts are we going to keep into – Put into S3 buckets, let's say, or whatever, right? Then from there we go and we reconstruct the data, right? So we go and we piece it back together. And then

that's part of our analyzer process. So I'm probably not doing a very good job of explaining. I don't know if Ken wants to jump in. But –

[00:42:51] KA: Yeah, I think one thing I would add is, as part of the ingest, we have – Think of it like images, right? So there's a rich amount of content. That's like the original traffic that was recorded, but we also gathered all this metadata about it. So where are we got it from? What was the URL? What was the pod that it came from? And so on. So we actually split, we keep everything. So we keep that original part. But that metadata we load into a faster type of database so that customers can visualize, "What's my throughput of what's going through right now?" What are the URLs that are being accessed right now? What are the host names that people are hitting? And they can then make informed decisions and say, "I want to drill in." And then when they go to that drill in, then that's when we're tapping that slightly slower storage, if that makes sense. We have this kind of fast, sort of time series database-esque type of data that we want to load quick for the user. But then we can dip in deep. This is actually part of our sort of clever engineering that people may not notice on how we're able to dip in and get that huge amount that's attached to that little time series metric.

[00:44:00] JM: Okay, really cool. Can you tell me more about what are some of the other cloud services or infrastructure components that you're using in your architecture? Maybe non obvious ones.

[00:44:15] KA: I think we've mentioned many of the ones that we're using. We use Kubernetes for our own back end system. S3 is about the least expensive storage option, cloud based storage option that you can get. We're using different time series databases in different ways of visualizing the data. So we're actually running some experiments. We're using Elasticsearch, we're using TimescaleDB, we're using InfluxDB. These are all ways to graph kind of the metadata. We've had to build a lot of our own tools for certain things. We were kind of on the foot and back to the customer side of things. We were hoping we could leverage some of the CNCF projects like Envoy and Istio, and Istio has a web assembly module that came out a couple months ago. And initially, we weren't able to get those to work the way we wanted, because we wanted a – Kind of control freaks, we wanted a lot of control of how the data was captured and that kind of thing. But those are some we're exploring now. We might be able to switch our collection or supplement, our collection and use other types of proxies that people

are already using. And that list will become very long. So if you look, there're tons of things like ingress and API gateways. There's a whole class of those, again, in sort of CNCF, and we'll be able to tap those and say, “Okay, we just want to tap in and get a little stream of the traffic that's flowing through. You don't have to use our sidecar,” right? So I don't know if that's part of what you're looking for. But in the in the Kubernetes ecosystem, this is part of the thrash and change and constant of changing that you see is happening. I saw 121 just dropped last week, and we got to go through and figure out, “Do we have to change any of our manifests for it?” But there's actually a lot in that Kubernetes ecosystem that I'd like to tap into overtime.

[00:46:16] JM: Great. Well, we've covered a lot about Speedscale itself. I guess I'd like to close off by just zooming out and getting your perspectives on how infrastructure’s changing and what kinds of changes you anticipate in the next five years or so. What do you think is – Or do you think there's any dramatic effects that are going to change in the near term?

[00:46:44] KA: So I know I just went, but I love this question. Because when we were starting Speedscale, and three co-founders, we were all talking about different parts of the problem we could work on, I really wanted us to work in the Kubernetes ecosystem. And I could tell, it already was five or six years old at that point. But its growth has been incredibly fast because people want to have more control over how their cloud native services work. They want to say, “I want to run it in my own data center.” “I want to run it in Google's cloud.” “I want to run it in AWS.” And you're seeing serverless is starting to go in a similar way. Instead of building a server list that only runs in one in one cloud environment, potentially, if I can rehome it into one of these Kubernetes offerings, or CNCF offerings, then I can run it anywhere I want.

And this is going to unlock a kind of freedom for the people who are doing deployments where they say, “Instead of getting kind of stuck with a cloud vendor and a certain version of their database, or a certain version of their storage, or whatever. Storage is another big, huge area for Kubernetes. If you can containerize, if you can pay the manifest tax and get your thing re-homed into Kubernetes, it becomes incredibly portable. That was one of our first major projects we worked on was we mentioned this customer who’s rehoming their EC2-based apps into microservices, into Kubernetes. First thing they did was to run it in Google. And that gave them this level of flexibility. So I think that's a huge trend that I see is Kubernetes is just the enabling technology. Not just Kubernetes, but all the ecosystem of things around it that lets you have a

lot of ownership. You can move it, you can run mini cube on your desktop, keep your fire extinguisher close by in case your laptop catches on fire, but it can be done. And I think that's awesome. And I love it. So anyway, I'll let Matt add his thoughts as well.

[00:48:44] MT: I think my thoughts on this are an extension of yours, Ken, but they are – I think what's going on with the changes in data, like privacy, the way that people are looking at privacy, is radically changing the enterprise architecture space. So a few years ago, I don't think it was as big of a priority for your fortune 2000 companies to – They were kind of okay with having some of their data sprinkled around. Our observation with the people we've talked to is that that's shifting. There's a fear, a downright fear of losing control of that data and getting out in ways that you didn't expect. And I think as it kind of extends into what Ken said is that Kubernetes is really an enabler for customers to retain some of that control while still getting the benefits that SaaS gave you. SaaS sort of made everything run more efficiently. SaaS services, they live and die based upon the efficiency of the infrastructure in some ways. And Kubernetes is a way for a lot of these enterprises to still retain some of that, seat some control, but really, retain that data privacy that they must have now. Even in just the last couple years, a lot of those security questionnaires we get have just changed radically around that.

[00:50:00] JM: Cool. Well, it sounds like a great place to wrap up. Thanks for coming to show guys.

[00:50:04] MT: Thank you.

[00:50:05] KA: Thank you very much for having us.

[END]