EPISODE 1509

[INTRODUCTION]

[00:00:00] ANNOUNCER: This episode is hosted by Mike Bifulco. Mike works as a developer advocate at Stripe and is also the cohost of the API's You Won't Hate podcast. Follow Mike at mikebifulco.com.

[INTERVIEW]

[00:00:14] MB: As software engineering teams start to build products that become more and more mature, it becomes necessary to be able to debug complex issues with tooling that enables understanding of the full scenario. This can come from application scenarios where API's are running multiple versions, where users are using your app from multiple devices, device types, OS versions, browser versions, things like that. You may also run into issues with network connectivity or functionality that is intermittent based on sort of use cases and user interactions. And for this sort of scenario, we've come to start to use a class of tools for debugging called observability tools.

Observability tools help developers to understand the full scenario of an application by digging into the data that is being used and recorded. And it can help to pipe data across various solutions that are used within enterprises to read and discover what's going on from analytics. One such tool, one product used for this is Rookout. And I'm here today chatting with Liran Haimovitch. We'll talk a little bit about the product and his history, as well as the use case for Rookout and sort of where the team is headed.

[INTERVIEW]

[00:01:19] MA: Liran, how are you today?

[00:01:21] LH: Awesome, Mike. It's great being here.

[00:01:23] MB: Yeah, really nice to chat with you. I appreciate you taking the time. Why don't we start here? Why don't you tell me a little bit about yourself and your career before you came to Rookout, and then we can talk about what you're doing your workout and the journey from there?

[00:01:33] LH: Sure. So, I'm going to be 35-years-old next month. I'm married. One kid. One dog. I love them both. I have the dog way longer, and doesn't take away as much of my sleeping hours as the kid. But I guess I have to love him as well.

Anyway, I've been a software engineer for over 15 years now. Spent the first decade or so doing cybersecurity and a lot of research and development for endpoint security, network security, kernel mode, user mode, tons of stuff all over the place. And then about five or six years ago, I've decided I want to move away from that. I want to focus more on developer tooling, and especially seeing how can I bring a fresh perspective coming from the cybersecurity mindset and background to the dev tools? How can we use something refreshing and interesting developer tools? And that kind of led me down the path to developer first observability?

[00:02:32] MB: Yeah, that's great. That's actually probably a good place to introduce the idea of observability formally. How would you explain observability to a dev who's not familiar with the concept?

[00:02:41] LH: If you look at traditional observability, observability tried to answer a problem that we had with traditional monitoring, if you look at monitoring, it tends to be a very black box form of monitoring. I mean, if you go back far away in time, then monitoring was server level. Is the server up or down? Does the operating system have enough free space on disk? Those kind of things. And then you're just installing an exchange server on that machine. And God bless it, whatever.

We moved on from that. In the next level, we're starting to monitor the application itself, whether it's an off-the-shelf application as exchange server. But even more so, as we've moved into SaaS, we care more and more about monitoring our own applications. Not just is the server up. But also, is the application up? How many requests is it serving? How many users is it serving? What's the latency? What's the error rate for each endpoint? And so on and so forth.

Now, that's very – It helps you get a gauge of is it – Rather, is it green? Is everything working well? Or is something terribly broken? But it has two main limitations. One, it doesn't help you with the finer points. I mean, the system might be up. Everything might be good. And you might be serving every customer out there with a 200-page saying the system is down. But it's 200. So, monetary doesn't catch it, because it should be 200 okay. That's the one problem.

And obviously, there are many examples that go way beyond that, into something that's broken within the system. You have a bug. You have a problem. Monitoring doesn't sync. But even more importantly, let's say you understood something goes down. Something is bad. What do you do about it? How well you have information into the system? That's a place where metrics traditionally have been less than ideal. And logs have been fairly useful.

But things can get messy with logs. Do you have enough logs? Or do I have enough context with each log? Our logs structured or unstructured? I mean, we could talk about it for ages. But at the end of the day, logs end up not solving the problem or not? More often, they do. And big part of the problem is that reading through logs, reading through metrics requires tons and tons and tons of domain knowledge to connect the dots.

I get those metrics. I deduct those logs. If I'm an expert in the system, if I know a repeat line of code, if I've been working on it since day one, I'm probably going to be able to connect the dots. If I'm new to the company, if I haven't been there, if it's somebody else's code, if it's a lot of legacy, if 100 engineers have worked on the system, and I don't have the knowledge, connecting the dots is going to be so much harder. And that's where traditional observability or distributed tracing came into play.

Distributed tracing tried to solve the problem with a hammer, so to speak. So, let’s try to get to more data. Let's make every log super fancy. Let's bring every log with the login information itself, with text. Let's bring a ton of context, including where is the request from, some metrics that’s being able to thread logs between each other so that we can connect all the logs that came from a specific request, even across microservices, and so on and so forth.

Essentially, think of distributed tracing is replacing logs with mega logs. Now, the problem with that is that the biggest issue today with logs is signal-to-noise ratio. I mean, most of us who are running large systems are probably overpaying our log aggregator providers. Many companies are paying Slack millions of dollar, or Elastic, or whatever you want to choose, millions of dollars every year, sometimes tens of billions of dollars every year, just to store all those logs nobody cares about. Because you just can't separate important stuff from not important stuff.

And in that regard, distributed tracing didn't help us all that much. Because obviously, those mega logs are super useful, and are definitely going to get you the answer if you have them. But because mega logs or distributed tracing are 100 times more expensive to process and store than the traditional logs, which you're already trimming out because you don't have the capacity to store them. And then all the sudden you discover that distributed tracing gets sampled very aggressively or just gets deducted. And none of the sudden, it would have been great in you had distributed tracing. But you just won't, because you're throwing 99% of that to the trash.

And so, that's where we came up with the concept of developer first observability. Now, if you think about it, developers honestly don't care most of the time if the system is up or the system is down. That's somebody else's job. And besides, it's a solved problem. We know how to do that. What we don't know how to do is to understand the system. Why is the system operating down? Why is there a bug? How do I develop a new feature that's going to make performance better? How do I make sure this new feature doesn't make performance worse? And so on and so forth.

The thing is, software engineers get a different task every day. One day, it's fix a bug. Another day, it's developed a new feature. Another day, it can be adopt a feature, deprecate a feature. Or even an onboarding task, such as learning a new piece of code. And trying to find one piece of data, one body of data that's going to answer all your questions for all of your tasks, that's impractical. It's much better to being able to be more agile, to be more dynamic, to be able to adapt the data you're collecting in real time based on the tasks you're currently working on. And that's the concept of developer first observability, of dynamic observability. We want software engineers and other stakeholders to be able to solve problems with the data they want, rather than with the data they have.

[00:08:27] MB: Yeah, that's great. So really well put. And a really interesting story of sort of the evolution of observability. It's, I think, useful to winnow it down to the maybe one-dimensional version of observability really did start with us pinging a server every few minutes to see if it was up and running. And then you slowly add more and more information to it. And eventually, if you've never seen the logs that kind of gets sent to traditional logging services, like it can be thousands of lines per second of information sent to these services. And leafing through them manually is a super human task. Like, there's really no way to do that. And so, for a while, it turned into developers getting really good with grep, or paring things down with whatever tools they had at hand to look for what might be issues. And obviously, that becomes a really massive task. And it's really interesting to look at this as something that then you see through the prism of the developers using this. And so, your goal, it sounds like, is to provide developers with the information that they're interested in for whatever task it is they're doing. So, maybe debugging something or plugging into a new functionality. Is that right?

[00:09:27] LH: Yeah, exactly. Think about it. The thing is, you're probably not really looking at the console or at the output of the logs. What you're probably looking at right now if you have a bug is your source code, because the bug is somewhere in there. What you don't want to do is start graphing around for strings that may or may not be useful.

Now, if you think about it even more, maybe the code you're debugging doesn't have a log. Maybe routed didn't bother putting in a log line. Or maybe that log line stemmed too much so it was removed or moved to a low verbosity. And maybe there is a log line in that function, but it's too high up and you want something within specifically or a specific loop. Maybe you already have the log line in the right line, but it's missing a couple of variables. And you really need those variables to get the data. And all of a sudden, you're trying to do a lot of guesswork. So if I'm going to find get this slide, and this slide existed, and that variable was here, and that was the value, can I hypothesize what was the value in another line and didn't make it to the other line or not – And you're doing a lot of guesswork, and your grepping for the logs, desperately grasping at straws, trying to figure out, "How can I come to the conclusion I want?" With the regular data, somebody else, which might be your past self, decided to put into the logs. And they've tried to – Maybe they tried to predict the problem I'm having? Maybe they didn't? Maybe I will get lucky. More often than not, I'm not going to get lucky. And I'm going to be spending hours, as I mentioned earlier, connecting the dots between the logs already have.

How much better would it be if you could just say, “This is the line.” I want to know what's going on it? I want to know, am I reaching this line? Or am I not reaching this line? I want to know who is calling me? Which IP am I being called from? Which function am I being called from? What are the arguments passed within the function? What are the variables specifically on that line? No guesswork. No trying to figure it out. Just tell a system, a machine, what you need. Another machine to get it for you, instead of going through endless amounts of logs and metrics trying to guess the data you need.

[00:11:34] MB: Yeah, that's amazing. That sounds very utopian from the perspective of a developer who is having a problem right now. It also sounds like it avoids the problem like going and telling some user to recreate the issue now that I've added the right logs or added the variable that was missing from my logline. What does that look like from the developer’s perspective? So, what does it like to integrate with a product? And once we have this in place, how does the developer use it to debug an issue?

[00:11:59] LH: As I mentioned earlier on, we've come from a cybersecurity mindset. And one of the things that are very common in cybersecurity are agents. Everybody writes agents. And they can go very deep into your application or into your servers and do a lot of fancy things.

In our case, we built agents focused on observability and developer first observability. And we built out six of those agents, one for the JVM, one for the .NET, one for Node.js. one for Python, one for Ruby, and one for Go. And each of those agents can be installed in as little as 10 minutes. And that's about it. Once you install our agent on your application, then you can start seeing what's happening there in real time. You can go into our web platform, select one or more servers you're interested in debugging. And part of the magic of Rookout is give you a debugger-like experience, a local debugger-like experience where you attach to a process or spawn a process and literally set breakpoints anywhere you want. But you can get that experience in the cloud, experience in Kubernetes, in serverless. And you can literally debug – We have customers debugging 5000 servers at the same time, because that's a use case. They have requests coming in into a load balancer. It can reach any one of those 5000 servers. So, just set a breakpoint and get data from any one of them where the relevant code is invoked. And that's it.

We instantly tell you what code is running on which server, so you don't have to do any guesswork. You can even narrow it down to specific commit revision. So you know exactly what's running. And let me tell you, quite often, we hear from a customer, once they're connected, “Oh, the bug, the problem in the code, it's running the wrong version, or it's not running the version I thought was.” And then just add a non-breaking breakpoint. And as soon as the code invoked, you’re going to get a full snapshot, stack trace, variable values, tracing information, everything you want. And you can instantly know. I know what's going on that line of code. No guesswork. No trying to get through logs. No simulations. No reproduction. Just see what's happening in the environment you care about. It can be your own machine. It can be the cloud. It can be staging, integration, even production.

[00:14:15] MB: Yeah. Wow! That's almost the dream of having like a DVR-like experience for seeing some user session of like, “Get me to this line. Let me know what was happening here.” And you don't need to infer at that point. You have all the information that was available. You can probably see what was wrong. Or at the very least, find the right person to help the whatever subject matter expert for that particular part of the application.

From there, we have all the data, we have all the information. I was sort of carrying around on workout site before looking at some of the functionality that the product has. Can you tell me a little bit about the integration side of things? So, you plug Rookout into your application stack. It looks like it talks to quite a few other services. I'm curious about how things like source control and collaboration tools and all that work. What are some of the useful features are there that teams find helpful and maybe some of the use cases that you find most interesting that you've seen teams using?

[00:15:06] LH: So, first and foremost, we integrate into your source control management. So, we can instantly tell you which code is running on which server and saving you a lot of headache and warning about that. That’s kind of the obvious first step. On top of that, we see many customers are – You're not always debugging alone. In fact, if you're debugging a complex issue, or a high priority issue, or just the help hand, you're often debugging with a friend. And sometimes you're doing that sitting next to each other, or Zoom, or whatever. But sometimes you're alone. And you get stuck. You set a breakpoint. You’re seeing a snapshot.

And all of a sudden, you're saying, “Hey, this doesn't make any sense. I want to ask my colleague about it. I want to ask an engineer from another team. I want to ask my product manager. I want to ask somebody about why is this value five and not seven. It should have been five, or whatever.”

And so, you can instantly share, whether it's slack, Microsoft Teams, or other collaboration tools, JIRA tickets, and just say, “Hey, I've got a snapshot. It doesn't make sense. Or shouldn't be there. Or somebody else has a bug.” Take a look at the snapshot. Now, think about, instead of trying to describe, “I’m seeing these in the logs.” Or, I'm thinking the value is whatever.” You can instantly share entire snapshot that shows the full state of the application in that line, and everybody can dig in even deeper into that. And seeing that you're making the data much more objective, much more detailed, way richer. And the third and I think the most interesting integration is how we connect with other monitoring and observability tools. Whether you're a fan of logs, metrics, or see what the tracing. One of the biggest hurdles is that you constantly have to change them. You need more logs. You need less logs. You need more metrics. You need less metrics. You need more, less than. And you need to change a piece of code that has been dormant for ages and has been causing you any trouble. All of a sudden, causing many support issues, and you want additional logs. Or a piece of the code, all of a sudden, you're investigating.

What we allowed for Rookout to do is you can generate the three logs, the three pillars of observability; logs, metrics, and traces, on the fly from any line of code and pipeline to your favorite tool of choice. So, if you're using Elasticsearch and Kibana to aggregate logs and analyze them, you can instantly add new log line from anywhere you want and add that to your Elasticsearch. And then you're going to see the new logs side by side with your old logs.

So, as you mentioned around grepping, that could now grep the new logs with the old easily. The same goes for metrics. Let's say you're seeing an endpoint is malfunctioning, and you want to break that metric down into two code paths. So, you can easily add a metric for each of the new code paths and see how they’re behaving differently, and which one of them is being called more often, and so on and so forth. And this way, you could gain even more from your existing monitoring tools without having to constantly redeploy the application to adapt the data you're collecting.

[00:18:20] MB: Yeah. Those redeploys can be very expensive functionally and organizationally, too. Oftentimes, adding a log and deploying it to, like you said, 5000 servers. Can take, I don't know, maybe at best 10, 15 minutes. But in many cases, hours and hours or require like a next day deploy or overnight deploy, or something, which loses you valuable time when you're trying to debug an issue, especially if it's mission critical or you're potentially losing business over it. The ability to change that stuff dynamically is really interesting and really helpful. You're injecting in the right part of the process to be able to sort of supercharge that experience there.

[00:18:52] LH: You know, today, we hear about many companies that deploy in 10 minutes new version. And working with a lot of customers across the industry, I have to tell you, it's a bit of a myth we tell ourselves. And I know so many people that feel bad, they're not in that lucky group where things get deployed in 10 minutes. Trust me, very few companies, very few teams are.

And there are two reasons for that. One is that many of us are working on complex systems, whether the testing takes longer, whether deployment requires downtime or causes other disruptions, whether we have more complex deployment strategies. Deployments more often than not even for – In fact, even if you're looking at the Kubernetes deployment, state of the art, changing pod by pod is probably going to take you more than 10 minutes. If you're even looking at it with a transpilation or container build, those things can easily take 10 minutes around just building without even going into the deployment phase.

But I think even more importantly, when somebody tells you they can deploy in 10 minutes, that's usually best-case happy path scenario. That's when you have an engineer next to you or the manager who can approve the deployment right next to you, clicking it instantly. That's where the CI/CD is green. When you don't have downtime to any component of the system, you don't have any feature freezes, you don't have anything at all in the world to worry about.

And the truth is that even if that's the happy path, chances are that if you look at the average, if you look at the media, and if you look at what's happening in the real world, quite often, those are not real numbers. I would argue there are plenty of use cases that are useful, even if you can deploy in 10 minutes. And we actually have a handful of customers that use Rookout,

deploy in 10 minutes and see a huge value from it. But nobody should feel obligated to say I can deploy in 10 minutes, because that's rarely, rarely the case.

[00:20:47] MB: Yeah, that's a hard and arbitrary thing to achieve for a lot of enterprise-grade solutions. And often, maybe you're targeting the wrong problem if you're looking at your deploy time versus your actual uptime, or user satisfaction and things like that.

[00:20:59] LH: Yeah. Besides, you can deploy quite often. Even if your deployment takes you an hour, and you're deploying five times a day, then you are pretty much in a pre-90, if not pre-99 percentile of excellence, especially as an enterprise.

[00:21:13] MB: Yeah. Yeah, certainly. Pivoting a little bit. I'm curious, can you tell me a little bit about the tools that you use to build Rookout? What is it built on? What does the stack look like? Things like that?

[00:21:22] LH: Sure. As I mentioned, we build agents for the six runtimes. And essentially, each of those agents is built from the ground up to provide the best performance stability, as well as ease of deployment for all of those runtimes. We use a combination of mostly dominant language for each runtime. For Python, it’s always t's going to be Python. For JVM, Java. .NET, C#, and so on. But we also sometimes include some SQL, whether it's for Go, or Python, there's things that you are too low level, too complex, and you want to go down to C or C++ to achieve some of the fancier stuff. We do a combination of that. And all of our packages are available in the public repositories.

Fronted application is a single page reactor. We've kind of shifted across the state management systems over time. That's a long story all by itself. For those of you who are into frontend engineering, state management is quite the rage for the past decade or so.

Our backend itself is running on Kubernetes. And we use a mixture of Go and Node.js for our servers, with most of our code, especially business logic residing in Go, and some of our more Internet-facing stuff. Some our integrations are of focusing on Node.js because of the great ecosystem and community libraries, where you can just integrate with pretty much anything really, really easily.

[00:22:50] MB: Sure. Wow! So, not only do you have domain experts or code being written for domains like .NET and Python. You have some C developers. You have people writing in Go. You have JavaScript. You have quite a myriad of languages there. So, can you tell me a little bit about the team behind Rookout? How many people are you? Do you have developers that are sort of focused on specific languages? What does that look like?

[00:23:10] LH: So, our engineering team is just shy of 20, and is divided into three groups. One group is focusing on the user experience both on the frontend, but as well as on the other components that directly supported that, or the business logic directly supports the frontend, or full stack engineers. The second group is our infrat group that deals with the infrastructure level. That can stop as well as DevOps, all of our internal developer experience, the resilient security of our platform, as well as more complex groundwork around the database management and similar stuff.

And we have the agents' group. The agents' group are essentially very talented folks that you can write multiple languages every day. Because sometimes you're working on .NET. Sometimes you're working on JVA. We do try to keep the code base very, very similar. So, it's easy to transition from each – At runtime to the other runtime. We try to keep the code base similar. We try to keep the concept similar. And yet, there are a lot of nitty-gritty details of how we do things. And how does each platform work? But it definitely shifts as you're moving from one to the other.

[00:24:23] MB: Yeah. I'd imagine that specific language runtimes have their own quirks and features that you need to work around, such that writing a .NET agent probably looks a little bit different from a Python one from a Node.js. Pretty incredible to have a team with such flexibility and expertise, especially within only around 20 people. That's pretty amazing. It's really interesting to hear about. I'm also kind of interested in hearing a little bit about, you mentioned your history is in sort of privacy and security, and that world as well. It sounds like there's probably quite a bit that you've had to take into consideration here to make sure that Rookout is usable by companies that need to comply with like GDPR, for example. There's probably a lot of boxes that need to be checked to make sure that you're not I'm collecting PII at the wrong times

or leaking PII, or that you can delete things as needed. Is that something that your team has had to spend a lot of time sort of considering?

[00:25:09] LH: Yeah. I think having my background in cybersecurity has brought a few advantages and a few unique angles into how using Rookout. Some of that on the more technical perspective of how to write those agents and how to make them the best possible agents. We see ourselves as a company that's focused on data collection, rather than on data aggregation.

So, if you look at companies such as Elastic or Splunk, they're doing the best data platform out there. They want to allow you to collect all your data and fuse it and do fancy stuff with it. Our focus as a company is being able to extract you the data you need, and then you can use it however you see fit. We do provide basic analytics tool. But as I mentioned, we also integrate with those advanced analytics tools, so you can get the best of both worlds.

Now, obviously, when you're dealing with extracting data, then there's a lot of security considerations that come to play in both security and compliance, by the way. And there are quite a few considerations. I think to my experience, and few of the other teammates, we've kind of taken on security early on in the process. We've kept a very secure software development lifecycle from the ground-up, whether it's around using tools such as software composition analysis, static application, security testing, or dynamic application security testing, a lot of those buzzwords. As well as good practices around security or based access controls, and so on and so forth.

And on top of that, we've added a lot of security-oriented features around single sign-on integration, audit logs, and so on and so forth. But even more, so I get to learn a lot about compliance that I didn't know much about early on. And I kind of had to learn quite a bit, both around how we achieve compliance internally, whether it's our SOC 2 compliance, ISO 27001, HIPAA, and so on and so forth. But also, about understanding the legal implications, working with large enterprises, and all of those elements. And when you're working with large enough enterprises, I can say that not just about having some significations on the wall. It's also about being able to communicate your security posture very clearly, and being able to offer the

security guarantees they want, which are often on top or different than whatever is in the compliance documents themselves.

Now, a couple of years ago, we figured out that no matter what we do, no matter how much security, how much compliance we're going to bake in, at some point, it's going to be hard for some organization to trust us, whether it's because their trusted stuff is hard to earn, especially as a very young and small company, or because moving data is very complex from them. And not necessarily from a legal perspective, it can be data governance, it can be legal.

In fact, if you look today at organizations, security-wise, moving data around is quite often the biggest security risk to them and the biggest compliance risk to them. So, we've came up with a hybrid architecture. In the hybrid architecture, we keep all the customer data within their environment while still providing a SaaS service. Think of it as splitting our SaaS service into two. The major boiling pot with all the business logic, with all the complexity, with all the stuff you have to worry about remains on our end, while a very small slim part that does all the data processing ends up in the customer networks. And so, the customers' data never leaves the network.

This means that especially when you think about privacy, most of the things go away. The data doesn't go to us. The data never leaves the customer. If there are any reporting obligations, such as the reporting obligation under GDPR, the customer doesn't have to report the data is moving, because it doesn't, which can save a lot of headache for them. And so, we're offering this enterprise approach. We found that it's much easier to convince customers, these enterprises, to work with us because they're securing the knowledge. They keep all the data within their environment. And we never access it. It might also be a good time to mention that we never access customer source code. We figured early on, we don't need it.

And so, as a security best practice, we just made up a company policy. We don't touch process, access, customer source code in any way. And those two guarantees about how you can secure data and how you can secure source code are imperatives in how we can offer security, compliance and peace of mind to enterprises. Lastly, I will probably mention that Rookout can also be seen as an opportunity for security. Today, you probably have some processes in place for debugging production. Whether people are SSH-ing into servers, or kubectl exactly into

pods, or taking database dumps and then re-inserting them into development. And all of those things carry huge risks.

Rookout offers you a portal, a secure portal, with all the security guarantees you need to provide developers access to production. Or you can control exactly what they see, how they see it, when they see it. You have all the clouds over everything. And so, we've seen many organizations use Rookout to increase developer access to production, which in fact, is in many ways the mission of the company.

[00:30:21] MB: Yeah. Wow! That is an incredibly thoughtful answer to what was seemed like an innocuous question. But there's a lot to take into account there. One of the things that I often discuss with founders of companies who are building products today, whether it's SaaS, or apps, or whatever the case is, is when it comes to things like security and privacy and keeping data safe and compliant, is to try and not become the expert yourself if you can avoid it. And to rely on services and people who are experts in the field. And it certainly sounds like your background is very helpful there. And then having a clear mission to not collect data from your customer servers, right? It never crosses the network. You're never actually looking at their source code. You're creating very important and useful boundaries that are also very functionally easy to explain. You don't have to dance around what's actually going on there. And that's pretty incredible. That sounds like quite the – Not only from a marketing perspective, a feat that is very useful to pass on. But an engineering feat that's also something that's pretty incredible to be able to say you've done too. That's really, really interesting. And that's definitely one of those things that sounds like a standout feature set there.

So, let's shift again. I want to hear a little bit about – Again, if you're talking to me as a developer who is looking at using observability and looking at integrating with Rookout, what's maybe the first feature you would tell me to look at, or the first thing you would tell me to do to actually make it useful for my team?

[00:31:41] LH: So, I will tell you, the first features is just see it works. I mean, early on in the day, people just wouldn't believe me when I would tell them that. This has been somewhat – It's been easier these days. But still, whenever I'm out at a conference and I'm showing people

demos, they're like, “Okay. So, where are you simulating my code?” Or, “This is just simulation, right? You put those breakpoints ahead of time.”

So, that's not. We're not simulating anything. We're not moving your code away. We're not moving the data away. We're not preparing anything. We're literally instrumenting your code on the fly when you need it, as you need. And the simplest thing to do is just see it for yourselves. You can literally just go through the process, install an agent, set a non-breaking breakpoint, invoke the code and see that you're going to get the data. You're going to get the exact same experience you would have in a local debugger, except it doesn't break. And you're doing that with an enterprise production-grade tool. And the benefits of that and the options that are endless.

But first and foremost, see it's working for yourselves. And on top of that, we can show you tons of cool features. You've got conditional breakpoints. You've got target integrations. You've got profiling baked in. Tons of advanced features, but everything is based on the very basic concept that you can decide in real time what data you want from a running server without writing more code, without redeploying it, without even having to restart it.

[00:33:11] MB: Yeah. Wow! I mean, use the product is a great pitch. Like, just try it. It works. Is really interesting. I mean, that's enough to perk my ears. I'm sure some folks listening to this are really interested in that, too.

Okay, so tell me about have you seen any unconventional uses for Rookout or things that you didn't expect that teams would end up doing that have proved to be useful?

[00:33:28] LH: Early on, we've heard the one of the interesting use cases for Rookout, especially in a microservices environment, is that, often, when you're debugging, it's not just your code, or it's not even just your microservice. Because you might be debugging somebody else's code. And we think about it. I mean, if I tell you that you have to add a log line to your code. I mean, that sucks. But you already have that checked out on your machine. You know how to rebuild it. You know how to redeploy it. Adding the logline set is probably going to take you a couple of minutes. Then you have to go to the CI/CD process, get approval. It might take

you 10 minutes. It might take you an hour. It might take you a week. But at least you know how to do that.

What happens if you have to add a log line to another microservice in your environment, or even to an open source library you're using, or closed source libraries in your company? All of a sudden – First of all, you have to find the correct source code for that application, whether it's an internal or external repository. You have to check it out. And that by itself might take you over an hour for some odd reports. But let's put that aside. Now, you have to get the correct revision that you're using.

You have to dive into the code. And if it's another micro service, it might be written in language you're not familiar with. You might not even be familiar with the code base. But let's assume you've got a tiny place. Now, you have to rebuild it. Do you know how to rebuild another person's code? Quite often, what I find in most organizations, standardizing microservices is not that easy. And even if you've got to a great degree of standardizing, they're using generations.

Every year, they change the standardization, because technology shifts. And if you have to build a microservice from a couple of years ago, it's probably going to be different than a microservice from five years ago, or microservice from now. And all of a sudden, rebuilding and then redeploying and changing the dependency, all of a sudden, doesn't sound like 10 minutes of work and some waiting time. It can easily turn into two or three days of work.

Now, I've heard that use case quite a bit. And it's been around. And we've seen many customers use it. But earlier today, I saw what I was amazed by. And one of our customers is using a feature flag, an internal feature flagging framework. And they had about one of the client teams. They have a big application. The microservices team started debugging. They didn't understand why feature flagging was acting a lot. Everything rooked off. And they start debugging with Rookout. They've gone in. They've gotten in. And all of a sudden said, “Hey, we're getting the wrong data on that microservice.”
And then on the Zoom, one of our – A couple of people from our team had gotten into the process, and a handful of them debug into this very complex issue that has been bothering them for a week now. And now all of a sudden, they said, “Hey, it's the other team's default.” And the Zoom session, people from the other thing started coming in. 2, 3, 4. All of a sudden,

we had 15 people on the call. And they've started digging into the code. And they said, “Hey, show me what's happening here. Show me what's happening there.” And because you can do it dynamically, they could literally see the flow, see through the code. And they started blaming the cache and the API's. And all of a sudden, they discovered the design was unclear, that sub-flags behaved differently than flags.

And so, that if you turn the flag, the overall flag, it doesn't automatically turn the sub-flags off. And they turned the discussion into a design discussion. All of the sudden, real time, because they could see the code, they could see its behavior in real time in production. And they could understand, what are they doing? Is it good? Is it bad? How are the two teams interacting? Not by looking at static source code. This is my truth. This is your truth. Go fight over it. This is how the two pieces of code meet. This is what's happening in real time. Now we can discuss, is it working for us? Or isn't it working for us?

[00:37:27] MB: Yeah, that's an amazing cascade. You've up leveled the conversation from – I mean, first of all, you probably helped them fix a bug that was plaguing the engineering team for a week, which is an expensive issue, potentially. To now it's more of a collaborative architectural discussion, something that the engineering team needs to reckon with and come to some agreement, whether – It doesn't really matter who's right or wrong at that point. They all need to be on the same page. And I can't imagine how long they might have taken to solve in other scenarios. I've certainly worked on engineering teams where there have been long standing historical problems with like, “Yeah, every time we do a beta, we have these problems.” And this is just the way it is here. We just have problems with you know turning on feature flags, like you said, or dealing with API versioning, or interleaving languages and runtimes on different platforms. You're giving them the ability to zoom-out and see how everything is actually functioning in real time. And they can have much more meaningful discussions there. That's very, very cool. That's one of those stories that probably makes it easy for your team to demonstrate the value of the product too and sort of – At that point, you're not even selling your product. You're delivering value from the first moment there. That's really cool.

Let's talk a little bit about what you've learned while building Rookout. I'm wondering if you have any experience that you've gained from this or anything that you would pass on to founding teams, developers for building new products. Just nuggets of wisdom, or tools, or patterns,

tactics, techniques that you've used along the way? What would you pass on to someone who's building a company today?

[00:38:51] LH: When I started Rookout, I focused on a very personal pain of mine. There was a time ago, almost – It's been over 10 years now. I was managing a project. **[inaudible 00:39:02]** all the roles. It's been a big project. It was waterfall. And like most waterfall projects, it launched six months later than scheduled. Someone might say 18 months later than scheduled, depending on which schedule you're counting. But a different discussion again.

And the launch was terrible. We had tons and tons of bugs. In particular, we had a bug that would cause one of our database to corrupt and the endpoint agent to fail, and having to reinstall it. Now, at the time, we were pretty much far out from CI/CD. And the build process was manual, fairly complex. And it took about a couple of hours to build the new version. And if you wanted to release a new version, at best, it would take you two days full.

Now, I've added probably 10 to 15 new logs working on that issue. Think about how much time we've spent releasing new version. If we had to release 10 to 15 new versions just four logs, each of them took us a couple of days of work. That was a big part of my inspiration for building Rookout.

Now, along the way, what I found, that software engineers tend to focus very much on how their code is operating on their machine, which is in a way what we did early on when we developed that project, because he could easily test that tool on your machine, unit tests, end-to-end test, whatever. And we've focused heavily on that. How do you run your code on your machine? If it's working on my machine, it's great. If it doesn't work on my machine, then something is bad.

But that's not what's really important. We focus on that because it's easy. It's easy to see the log. It's easy to attach a debugger. You get a sense of things you have control. You have visibility. So, that's what you focus on. But actually, the further away code goes from your machine, the more important it is.

Staging is more important in your machine. Production is way more important in both. Because today, most engineers don't have tools to see what's happening in production, they kind of shrug

it off. Because if they can't see it, if they can't act on it, then it can't be their responsibility. And I think that's something huge, right?

Today, when we speak with customers, they care about the cultural shift even more than they care about the tool itself, because you want your engineers to focus on production. You want to have your engineers focus on the business impact of their code. And that's the truth. And in fact, this became Rookout's mission. We want engineers to focus on production. We want engineers to focus on whether code actually matters.

And I think for most founders that are working on their personal problems or building companies for any other incentive, find that bigger truth, find the bigger visions. Because at the end of the day, you're selling. If you're building a company or building a product, you're selling something to someone, and selling a dream, selling a cultural shift, selling an impact to the organization is more important than just selling a tool, whether it's a screwdriver, or a software development tool.

[00:42:01] MB: Those are extremely wise words. I feel like that can apply across industries, across whatever you're building. It's more than just the little problem you're fixing. It's about this shift that it provides. The value head above just the product itself. That's fantastic advice. I really like that.

What's next for Rookout? Are you tackling big problems? Are you looking to expand into new functional areas? What's on the horizon for you?

[00:42:25] LH: So, Rookout started as a live debugger. We allow you to do all this magic I've mentioned around debugging your code **[inaudible 00:42:30]**. Late last week, released what we like to call Live Logger. Live Logger allows you to essentially play around with the repository level of your applications. Turning it up and down dynamically. But even more so, contextually.

So, let's say you have a user reporting a problem. You can turn on logging for that specific user. You can even turn it automatically. Once a ticket has been opened, once support has been activated, let's increase that log verbosity. Because as I mentioned earlier on, the biggest

problem with logs is signal-to-noise ratios. And most companies I know have 99% of the logs turned off at any given moment, because they can't afford to turn it on.

Live Logger allows you to turn on logging on demand for the specific part you care about, specific server, specific user, specific account, specific source file, based on what you need right now. So, you get the data – Again, you get the data you need, when you need it for a fraction of the cost of collecting everything all the time.

And we're also looking at expanding beyond that. We're looking at offering tools. So far, we've been focusing mostly on backend technologies. We're looking into moving into frontend and mobile. We are looking into offering dynamic developer-focused observability around cloud infrastructure as well. And also, so far, we've focused our experience on providing the best debugger in the web. And we felt that that's the best approach that will apply to most people. But we've been hearing more and more people that are interested in being able to use Rookout it in their IDE. So we obliged. And we're rapidly prototyping, rapidly developing our new IDE-based debugger for Rookout that will allow you to debug from the convenience of your IDE. And we're actually eager to see, will customers like it more or less than working with our web version?

[00:44:24] MB: Wow! Yeah, certainly lots of really challenging problems that you're tackling. I think the IDE thing sounds really interesting. I think a lot of developers tend to be kind of set in their ways of like, "Hey, I'm used to working this way." And probably bringing the tool to them is going to be something that is agreeable and useful for a lot of developers. But I'm kind of curious to hear what the feedback you get on that is as well. So, at the moment, is your team hiring?

[00:44:47] LH: We're definitely hiring across all positions. We're hiring for engineering work. We're hiring for sales engineering and solution engineering goals. We're hiring for developers. So, hit us up if you have any thoughts on the matter.

[00:44:59] MB: Got it. So that's a great lead into the next point here. Where should they go if they want to get started with Rookouit? And where should they go if they want to look for jobs at Rookout?

[00:45:06] LH: If you're interested in Rookout, go to rookout.com. If you feel you have the flexibility, the will to do it, you can always sign up directly online, install one of our agents and get started. It's free forever for a single user. If you are working for a larger enterprise, have some security concerns, or other questions, feel free to reach out on the website with chat or our contact us. We'll rapidly call you back and give you a no bullshit answer to whatever is plaguing you. I promise. We also have a careers page. You can check it out if you are looking to maybe join us. And you can always reach out to me. I am on Twitter, Liran_Last. I'm on LinkedIn, Liran Haimovitch. So, feel free to reach out and let me know if you have any questions or thoughts.

[00:45:56] MB: That's fantastic. Liran, it's been so nice chatting with you. It's really interesting to hear about Rookout. We'll make sure that all of the applicable links are in the notes in the description for this podcast. It was wonderful chatting with you, Liron. Have a great day. And thanks so much for your time.

[00:46:09] LH: Thanks, Mike. It's been a pleasure. Thanks, everyone, for listening in.

[00:46:12] MB: Right on. And for Software Engineering Daily, my name is Mike Bifulco. Have a great day.

[END]