EPISODE 1505

[INTRODUCTION]

[00:00:00] ANNOUNCER: This episode is hosted by Lee Atchison. Lee Atchison is a software architect, author and thought leader on cloud computing and application modernization. His most recent book, *Architecting for Scale*, is an essential resource for technical teams looking to maintain high availability and manage risk and their cloud environments. Lee is the host of his podcast, Modern Digital Business, an engaging and informative podcast produced for people looking to build and grow their digital business. With the help of modern applications and processes developed for today's fast-moving business environment. Subscribe at mdb.fm, and follow Lee at leeatchison.com.

[INTERVIEW]

[00:00:46] LA: Maintaining availability in a modern digital application is critical to keeping your application operating and available, and to keep meeting your customers' growing demands. There are many observability platforms out there. And certainly, Prometheus is a popular open source solution for cloud native companies. Yet operating an observability platform costs money, and all of the platforms are highly data intensive. Managing costs and data retention policies is critical to keeping your application operating healthy and operating available.

Chronosphere is the leading observability platform for scaling cloud native applications, focusing on managing costs by managing data. Martin Mao is the Co-Founder and CEO of Cronus fear, and he is my guest today. Martin, welcome to Software Engineering Daily.

[00:01:37] MM: Thanks for having me, Lee. And looking forward to our conversation today.

[00:01:40] LA: Yeah, so am I. So am I. I've come from the observability space, too. I've spent several years at New Relic. And so, I think I'm looking forward to this conversation –

[00:01:49] MM: I do. And I think that will make this conversation even better.

[00:01:53] LA: Let's start specifically talking about cloud native, since that's one of the things you focus rather heavily on. And cloud native is certainly the buzz right now. And scaling modern applications is important. So, I'm wondering if you can tell me, what makes scaling a cloud native application, fundamentally harder in your mind than a legacy application?

[00:02:16] MM: Yeah. Hopefully, the result of being able to scale a cloud native application is actually easier than a legacy application. Otherwise, I think going through the adoption, you may not be getting the outcome, well, one of the outcomes that you're aiming for. But I think one of the things that makes it harder is that cloud native environments and architectures are just so much more complex. There're so many smaller and more ephemeral services that have dependencies on each other. The infrastructure that you're running on is a lot more dynamic and ephemeral as well. So, you can imagine the overall architecture is a lot more complex and a lot harder to reason about than legacy applications.

And then on top of that, as an application developer, the onus to operate this is really on you, right? So, it's not only is it a tougher environment, but the responsibility for operating in that environment is now on the developer themselves. So, I think both of those reasons is perhaps why it's so much harder and more complex to create and scale these cloud native applications. And I think that's what makes observability or gaining visibility insights into these environments is even more important after you adopt cloud native.

[00:03:27] LA: Yeah, one of the things I like to say when we talk about microservices versus monoliths, which is not exactly the same thing as cloud native, but it plays into it very, very closely. Monoliths, the complexity is in the code. And in microservices, that complexity is in the interactions.

[00:03:45] MM: Mm-hmm. 100%.

[00:03:46] LA: And that makes monitoring the code maybe less important in cloud native applications, but monitoring the interactions between the services and between the services and cloud services, etc., that's critical, absolutely critical for keeping the system up and running.

[00:04:02] MM: You're 100% correct. Right. You can imagine what used to be two function calls, and you're not too worried about what's happening between those two function calls, is now a call over the network. And now, that introduces a whole bunch of other dependencies and complexities there, for sure. And also, it's outside of your code base, so you lose the visibility into what the piece of code on the other side is doing as well.

[00:04:25] LA: Yep, it impacts performance, but also impacts availability. We talked about this trilogy between scale, diversity of capabilities. I guess maybe nimbleness is the right word there. Scale, nimbleness and availability. So, what demands does keeping that trilogy working the way you need it to? What demands does that actually place on observability in general?

[00:04:52] MM: Yeah, it actually places a lot of demand. And you hit the trifecta there, right? People want more scalable, more dynamic and more highly available applications. And generally, like a lot of things in life don't come for free. So, being able to get these things does come at a cost. And one of the costs I would say is observability. It makes it much tougher from an observability point of view.

And specifically, it's probably four things here that it actually adds on the demand side for observability. The first of which is, for an observability system, because you have so many more pieces in this, that they're so much smaller, generally, the amount of data being produced and observability data being produced is much higher than before. So, even if you have – You can imagine the same infrastructure footprint overall handling the same amount of request for the business. Just by having that architecture be cloud native, as opposed to not, generally produces a lot more observability data. And hence, you need an observability solution that's also a lot more scalable that can handle that increased volume of data.

The second demand it places is, because these environments are generally more complex, you generally need to be able to slice and dice that data in many more ways that perhaps you didn't need to in the past before. So now, for you rather than just running on that single monolith, you may want to know exactly which host or which node you're running on, or which AZ you're running on, which version it is and things like that. And all of these extra dimensions generally leads to a problem called higher cardinality. And hence, not only do you have a lot more data, but you also need to query and pivot the data in a lot more ways that you didn't need to before.

And in that adds a demand of requiring an observability system with much higher performance requirements than before.

And both of those things, that the more data, and the higher complexity, and the higher cardinality generally leads to higher costs. And you mentioned this earlier. So, normally, more data equals more cost. The sort of need on demand on observability is perhaps you need a more cost efficient solution than ever before.

And then the very last thing is, because it's a very different type of architecture now, it's a lot more distributed than it was before, we are working with micro services as opposed to monoliths, then you need a specialized feature set that's built for this type of architecture. And this is where things like distributed tracing really comes in and becomes a core requirement of an observability system. Whereas perhaps, in the legacy environment, APM would have been a justifying solution there.

[00:07:22] LA: Yeah, I was just going to say that traditional APM is really less and less important, it seems like, as we're moving to more microservice-based architectures. And so, talk a little bit about the difference. For listeners who are maybe new to this space a little bit more, what's the difference between distributed tracing for observability and a more traditional APM approach, application performance monitoring approach, to observability?

[00:07:51] MM: Yeah. I think the high-level concept is very similar. And really, it's about capturing the flow of a request or the flow of a workflow perhaps through your system. So that higher-level concept of understanding how a particular request interacts with the various parts of the system remains the same. And that's what APM sort of gave us before. And that's what distributed tracing gives us today.

Now, one of the main differences there is that, as you talked about earlier, when we used to have these monoliths, it was more about how that request went through one giant piece of code. And you were very interested primarily in how that piece of code or have interacted with the operating system in the underlying hardware, and you're looking for performance optimizations there.

However, you can imagine these days with a microservices-oriented architecture, generally you don't have those giant portions of code anymore. Each of your microservices do less and less. And it's a lot more about sort of the interaction of that request and the dependencies between all of your microservices there. And you can also imagine that, generally in your microservices, you're running on more of a container-based infrastructure, where perhaps you don't even have access to the underlying operating system or the underlying hardware, or all of those layers have been abstracted away from you. So, it's trying to achieve the same thing, which is trying to gain visibility and insights into how the request flows through.

However, because the architecture is so much more fundamentally different now, there just needs to be a different sort of tool set that can actually track these requests across the network through all of your different microservices and give you a view of that instead.

[00:09:30] LA: You used a phrase earlier, and that was a higher cardinality. That's really gotten me thinking and intrigued a lot by the impact of that, and really the impact on observability to that. Because you're right, when you're running a monolithic application and you start your code path, you know what type of server you're on. You know how much memory it has. You know which instance of the application, which version of the application within. Because it doesn't change throughout the call for the most part.

When we start talking about a microservice-based application, we're calling service after service after service, each of those separate services is on a separate host potentially in a different data center. But we won’t even go there. But it's running a different version, perhaps a different version of the same service than other calls are running. And there's that extra dimension of data really is critical for solving some of these sorts of problems. But it is an extra level of dimension of data that never really existed before and really wasn't important for normal planning.

How do you deal with that quantity of data? And you've talked a little bit about pivoting to focus on those dimensions. But what are some of the tools? What are some of the techniques that you can use to actually take advantage of that additional data and use it to diagnose problems?

[00:10:53] MM: Yeah, there's probably two things here. So, the first thing we found, and we haven't gotten into this yet, but the history of the company here at Chronosphere is that myself, and my cofounder, and a lot of the core team used to be the observability team at Uber. And we solved this problem for them many years ago.

And what we found with the high-cardinality use cases is, as you rightly pointed out, you sort of need these dimensions. And without these, you sort of lose a lot of that visibility. So, the first thing that we did was just to build backend technologies and infrastructure that could scale and handle these in a fairly cost-efficient manner.

So, you can imagine there's no perhaps a magic trick there. You just sort of have handle it. Stall that data and scale up to handle it. And then sort of hopefully have the backhand be cost-efficient enough, thus, that when you do do that, it doesn't cost you an additional arm or a leg to achieve that.

And what we found the last few years that that pattern continues. So, what we achieved at Uber was – I think when we went through this with our technology, we sort of made the backend technology about 7X more efficient for the same amount of data, which was great. But what we found over time was that this trend doesn't slow down. You keep producing more and more data over time.

And now here at Chronosphere, we're really focused on smarter ways of dealing with that. And what we mean by that is really having a look at the problem from a different perspective. Not just what dimensions do I need to put on? And what dimensions could I possibly need? But starting to look at the problem from, “Okay. Well, what is the outcome of the data that I'm storing? What are the views of the data that I need to make?” And particularly subsets of data. How long do I need that data for? And at what level of detail do I need that data for? And almost working your way backwards from the value of the data to then producing and storing that data with that level of dimensionality, and perhaps retention periods and things like that.

So, now at Chronosphere, it's more of having a look at the outcome and the value of the data that you're storing, and then manipulating that data thus that the storage and the cost of it matches up with the value and the outcome that you're trying to achieve with that dataset.

[00:13:03] LA: Makes sense. Makes sense. And I think we're going to get into this a little bit more, too, as we go on. But we talked about the data collection is one thing. But then observability is how do you utilize that data to understand how your application is working and solve problems that are occurring with your application.

Now, I know Chronosphere talked about three phases of observability. I think the three word, know the problem, triage the problem, and understand the problem. Do you want to talk a little bit more about what you mean there and how Chronosphere helps with those three stages?

[00:13:37] MM: 100%. This is a similar vein to what I was just saying where we took a little bit of an outcome-based approach to observability. I think you look at the industry right now, and a lot of people are defining observability as logs, metrics and traces. I mean, yes, you do need the three different data types. However, perhaps that is – You can imagine just by having those three and producing those three data types doesn't guarantee you observability or even great observability there, right?

We sort of looked at it from an outcomes-based approach. And from that we sort of looked at, "Well, what are you trying to get your observability system to really do? What value does it add?" And for us, it comes down to these three things. Can you know when something is wrong? Can you triage how bad that issue is? And then can you understand the underlying root cause and go and fix it? And it just comes down to that. And I think that's been the same when even before observability to the – Or when it was all about APM solutions, as you know, you're trying to achieve the same outcomes there.

Of course, the environment and the architecture is very different. So, you need a different solution there. But the sort of ultimate outcome is really the same. And I'd say that it's not just these three phases. But the ultimate goal here is to get to a remediation. Is to fix whatever the current issue is and get the system into a stable state again. And while the three phases of knowing, triaging and understanding are ordered in that particular order, what's interesting is that you could almost get to remediation from any of these phases. So, you don't have to go through each of the three steps, which I think is pretty critical these days.

If I was to give some concrete examples of what that looks like and perhaps how Chronosphere helps, we can maybe start with the no phase of like can you ever detect when there are issues here? So, to give a concrete example, perhaps you're doing a deployment of your microservice. And all of a sudden, as you do the deployment, you detect that your microservice and your endpoint is returning more errors. So, all of a sudden you can imagine that if you can detect that that's an issue, you don't even need to triage and root cause it. You may not even need to understand, “Well, what about my code change is causing this?” The first thing you probably want to do is get the system back into a healthy state. And you can imagine rollback that deploy.

[00:15:50] LA: Rollback.

[00:15:51] MM: To achieve a remediation there, right? In that particular example, with the Chronosphere platform, of course, we're storing all of your infrastructure and your application in your business metrics. And we have an alerting platform on top. Most observability systems out there do that. And you can use these systems to be notified and check issues.

The differentiator here for Chronosphere is that our sort of ingest latency is extremely low. So, within, I would say, a few 100 milliseconds, the data, the hat detection of the fact that errors have been returned are detected extremely quickly. And then our alerting intervals are at a one-second interval as well. So, you can imagine within a fraction more than a second, you can detect this. And then you can also integrate it with your perhaps CI/CD system and automatically roll the deployment back, right? These are the things that we could do for the first phase. And you may not even need a triage and record the issue at all.

Sometimes you may have to triage and get there. And sometimes just by knowing that something is wrong is not enough to remediate. So, perhaps a second example is, all of a sudden, things are fine in production. You haven't rolled anything out. And you notice that the error rates for your endpoint for that micro service are increasing, and you're getting more and more errors there. And, of course, you can get detected when this happens. But because you're not actively introducing a change to the system, maybe there's no rollback to do. So, that's when you would have to start triaging and knowing like, “What is the impact? And what is really wrong here?” Because you can imagine, if you get paged at two o'clock in the morning, perhaps

you want to know, “Is this something I have to deal with right now? Or can I go back to sleep and deal with it in the morning here?”

And I’ll say with Chronosphere and with most of the other solutions out there, again, you have the metric data, so you can sort of see what's going on and see your error rates through the dashboards that are built on top of that. I'd say that this is where that concept of high-cardinality perhaps would come in.

So, if you have increasing error rates, perhaps having a look at the view of, “Well, are the error rates happening across all of my deployments? Is it just one particular AZ? Is it just one particular cluster that this is happening to?” Gives you the insight of like, A, where is that particular issue? And also, how bad it is.

So, you can imagine for our particular example, perhaps you're seeing increasing error rates. But as you drill into the data and as you start to pivot it, you can see that they are all coming from one AZ, for example. And it's really that AZ from your cloud provider that is causing the issue. And you can imagine here, again, you may not want to do the complete root cause of what is causing it. You could perhaps, at this point, remediate the issue via let's say routing around that particular AZ that is impacted and just having your request go to the other AZs that your stuff is deployed in.

So, you can imagine an observability tool can definitely help with the triage phase, for sure. And again, a lot of tools out there can store the metric data and have the dashboards on top. I'd say what's unique about Chronosphere in this particular instance is that the backend performance is a lot more performant than a lot of the other systems out there. So therefore, as you do these queries over high-cardinality data, they will be a lot more – The results will return a lot quicker than –

[00:18:56] LA: You’re focused on those extra attributes. And so, you're able to perform those sorts of –

[00:19:02] MM: Exactly. Exactly. So, there are particular use cases with customers we’ve seen where they get like an almost 8X increase in query performance. That as they're trying to look at

the different views, it comes back a lot quicker. Or perhaps another customer we've been working with, they couldn't actually load data if they look back more than an hour. So, all of a sudden, with increased performance, they're able to see a lot more data than they have, and they get a lot more insights that way as well.

And the last one here is perhaps root cause analysis. So, sometimes you do have to do root cause analysis while the issue is happening in production. Ideally, not because I feel like for a lot of developers out there, you don't really want to do root cause analysis while the system is down. That's probably the worst time. And ideally, you can remediate ahead of that and then take your time to do the root cause analysis. But perhaps sometimes you just have to do it.

And I'll tell you that this is where you really want a system to handle more than just metrics and show alerts and dashboards. You really probably want to bring in things like a distributed tracing to really get to the root cause of what is your particular issue.

So, if we look at one example here, maybe you're an ecommerce company, and your checkout flow is broken. And you know that your customers are not checking out successfully. However, when you look at all of your backend service SLOs, everything is green. And we see this from time to time, where it's not obvious what is the root cause of this. But clearly, there's an impact to your customers. Clearly, there's an impact to your business.

And imagine this case, it's great to have a tool like Chronosphere to both alert you that, yes, your checkout flow is broken. But then also sort of be able to then pull out all of the particular requests end to end for that checkout flow and then do the analysis for you to really try to show you why that issue is happening, because it may not be obvious in your individual service SLOs there, and sort of present the root cause to you. So, that's another area where I'd say chronosphere, in particular, can help with that particular phase there.

But again, all of it, the no phase, the triage phase, the root cause phase, all of it is really about getting to remediation as quickly as possible. And if you have to go through all three, fine, so be it. But if you can do it just by detection, or if you can do it when you're triaging, you can get to remediation even quicker. And ideally, you only root cause while the issue was not actively happening there.

[00:21:13] LA: Yeah, it seems to me there's detection, triage and root cause understanding, but you want to do as little of that as possible prior to remediation, as much as possible to post-mortem.

[00:21:25] MM: Yeah. Because if you imagine, if you can roll back, now you have all the time in the world. Your downstream dependencies. Your customers are not impacted. You have the time to really take the time to understand why, when I wrote out, did this cause issues, right? As opposed to you can imagine trying to figure that out while your customers are complaining, while the other teams are complaining. Generally, under stress is probably the worst time to go and try to do this. But, yeah.

[00:21:46] LA: And one of the things that you haven't talked about, but I'd love to hear how Chronosphere helps with this, is early detection. So, before problems occur, you can notice trends, notice problems. Your service is starting to go a little flaky, but hasn't created problems yet. But you know it's going to based on the trends of the direction that the specific metrics are going. I'm wondering if you can talk about that dimension of this problem, and whether or not Chronosphere has anything unique that helps in those areas?

[00:22:17] MM: Yeah, 100%. And I think he hit it spot on there. To sort of be more preventative, you're really trying to look at more historical trends. And that's, obviously, the trend of the data in the most recent periods, or perhaps the last hour or last day or so. But also, interesting trends that happen week on week, right?

So, yes, you can imagine on a Friday afternoon, the trendline may be going up. But maybe that's expected, because every Friday afternoon, your business gets a lot busier than normal. So, I'd say, from that perspective, what Chronosphere is trying to do is to essentially give you particular – And honestly, these are like data science techniques of doing things like week on week standard deviation analysis to see how your expected trends are expected or not expected based on the historical data there. I wouldn't go as far as calling that AI or AI Ops. It’s really, for the most part, a lot of data science techniques and sort of data manipulation techniques that can sort of help you find these outliers and these trend lines more than anything else.

And I think the second part there is, as we talked about high cardinality earlier, and we're producing so much more data. It's not just doing the analysis. You now have to do the analysis on so much more data with so many more dimensions than you ever had to before. So that just requires, you can imagine, a backend that can, A, store all of that data. And B, be performant enough to perform all the analysis for you as well. So, those are probably the two areas where Chronosphere differentiates against other platforms out there in this particular space.

And again, it's all because it goes back to, as you're adopting cloud native architecture, you then all of a sudden produce much more data or produce a much more high cardinality data, and that sort of introduces that requirement again.

[00:24:02] LA: Yup. That makes a lot of sense. So, I'm going to ask a question that I hear a lot. And you started to answer this already. So, this isn't an attempt to necessarily repeat the answer. But I think it's really just to drive the point home. The question I get is, “Well, Prometheus does all of this. So, why can't I just use Prometheus for cloud native monitoring? Why do I need something more than that?” What do you tell people when they asked you that?

[00:24:31] MM: Well, the first thing I say is, A – And we, here at Chronosphere, do love Prometheus. And I think it is a solution that you can use. And it's a fairly self-contained solution. So, you can imagine with a single binary, you can get basic monitoring of your cloud native environments, which I think is fantastic. And beyond that, I'd say that it did serve well in sort of standardizing the instrumentation, and the protocols, and the query language across an industry for cloud native. For all of that, it is fantastic. And you can definitely get started on it.

I will say that because it's a single binary solution, a lot of companies out there start to vertically scale that up and you can imagine start to use larger and larger instances to run that on, and perhaps eventually run into a limitation with vertical scale. And unfortunately, it was not really designed for horizontal scale there. There is a high availability mode where you run two of these instances, but you can imagine, it was never built to be a distributed system. So, the data consistency has issues and things like that.

So, there are particular reasons why it exposes an API for other solutions, like M3, which was our open source time series database and scalable metrics backend for Prometheus to be introduced to work in conjunction with it. And really, that or perhaps Cortex or Thanos, are other technologies you can use to go help solve some of those horizontal scale challenges that Prometheus was not really designed for. I think it was designed to help folks get started in an easy fashion. You can imagine a much more complicated distributed solution is going to be much harder and more complex to run. So, it does that job extremely well.

The other thing about it is it is a contained solution for monitoring, which means it does ingest and store metrics, which I think is great for detecting when issues go wrong, and perhaps the triaging as well. However, for a whole observability suite, you start to need to piece it together with things like a dashboarding solution. It doesn't really come with dashboards out of the box. As you start looking for requirements for root cause analysis, you probably need something else that can handle distributed traces and store those and whatnot.

So, I'd say it's an okay thing to get started. And we are big fans of it. However, as you start to scale and have a larger and more complex environment, I'd say that that's when you probably need something a little bit more than Prometheus.

[00:26:57] LA: Yeah, and that makes a lot of sense. And one other thing that I tell clients when they asked me that question is think about the cognitive load required to understand what Prometheus is telling you. The quantity of data that you get and that you have to deal with is incredible. And so, trying to get useful tidbits of information out of that can be a challenge, unless you have something on top of Prometheus or alongside Prometheus that helps you with that. Yeah, something like Chronosphere, certainly.

So, what's the general answer to this question? Besides Prometheus, there's obviously lots of other observability platforms out there. I spent seven years in New Relic. There’s certainly Datadog. There's certainly lots of other observability platforms out there. What truly makes Chronosphere unique? We talked about the cardinality effect for cloud native applications, and that is something that is truly unique. Both what else? What is it that – Why would someone choose Chronosphere over one of the other platforms?

[00:27:59] MM: Yeah, I mean – And I'd say, I'll precursor this answer with perhaps Chronosphere is not the ideal solution for every use case out there as well, right. So, I think it really comes down to what the challenges are that a company has out there. And depending on the challenges, I think Chronosphere definitely is an ideal solution for some and perhaps less than ideal for others.

I'd say, at its core, and we touched upon this a little bit, the requirements that we went through in terms of what you need of an observability solution as you adopt cloud native in terms of scale, the high cardinality support, the cost efficiency, and then the end user features, those are really the tenants that Chronosphere was built around.

So, you can imagine if you're not adopting cloud native architecture, if you're not adopting container-based infrastructure, if you're not adopting microservices, perhaps an existing APM solution would be just fine. You can imagine, everything here at Chronosphere has been optimized for that particular type of environment.

And it starts off with the backend that we originally built at Uber for the scale and the high cardinality of the data and the performance requirements that you need to even query the data. So, you can imagine, even at its core, yes, producing and storing and making metrics, let's say, available, is not very different between a lot of the solutions out there. However, as you sort of adopt cloud native and you have so much more data than you ever had before, then you can imagine that that is when these properties really sort of come to differentiate themselves a little bit more.

On the cost efficiency side, that is also an area I would say where a lot of companies are struggling with right now. You can imagine with an existing solution, in a pre-cloud native world, because we're already, I would say, perhaps high, but I would say perhaps reasonable for a lot of companies out there. And then what ends up happening is that, as they do this transition over, what we often find companies find is that they're not necessarily paying the cloud providers anymore for a cloud native environment. You can imagine, really – You can imagine, if you're running on top of Kubernetes now and you weren't running on them before, you're really paying the cloud providers for the underlying VMs, and the memory, and the CPU, and the storage that you get. So, perhaps all the resources haven't changed underneath the covers. And

hence, your bill to the cloud provider and your overall infrastructure bill has not changed. However, what ends up happening is your bill for observability goes through the roof relatively.

So, all of a sudden, it's not only that these challenges are there. But the cost of solving these challenges become extremely high. And this is where Chronosphere not only provides better unit economics and makes it sort of cheaper for the same amount of data that you're producing. But this is where we really start to differentiate in terms of how to help a company understand the data that's been produced and then help manipulate and customize the storage of that data. Plus, that it matches the use case a lot more, so that you're really getting the value out of the data that you're storing in the particular way that you're storing it as well.

And what that really does is help you sort of control that cost as you continue to grow over time. And that's one thing we found at Uber was that, again, we sort of – It was great that we're going to instantaneous cost saving. However, the rate at which this data was growing was pretty astronomical. So, if we couldn't control that growth over time, that was going to cause problems down the line there a little bit.

And then I'd say the last differentiator is perhaps on the sort of end user interfaces side of things, where, on top of this sort of performance that you get, we're really starting to build more customized user experiences to help you do the root cause analysis. So, you can imagine a lot of investment is being made into sort of automatically performing the analysis of root cause for you, and perhaps even customizing that experience for each individual end user. Because for an observability platform, almost every perhaps infrastructure-focused person or developer needs to use it day to day. But what they want to get and how they interact with the system is all fairly different across the user base. So, we're starting to sort of customize and optimize that experience for each of the different types of end users that come into this system as well. So, at a high-level, those are probably the ways in which Chronosphere are fairly distinct in the market.

[00:32:14] LA: Obviously, the piece that's common with everything that you mentioned was the focus on cloud-based applications. So, you're providing better solutions that are more customized for cloud native applications, but also provide them in a more cost-efficient mechanism and better usability. So, cloud native application is the key.

[00:32:35] MM: Yeah. And I'd say that that's how – I would say, you can imagine the observability market is packed with options not just for different types of environments and architectures that we're talking about. They're also packed with a lot of options on particular use cases, right? There are particular tools out there for end user monitoring and things like that. So, I'd say, for customers out there and for companies out there, really looking at what specific problems they're trying to solve or what type of architectures. And that would help them scope down the options base, at least, to fewer options, and then make a more informed decision there.

Because I must admit, as you look at the landscape, there're so many options out there, and every tool claims to do everything well. And generally, as we both know, that's not the case.

[00:33:17] LA: That's not true.

[00:33:18] MM: That this a tool, a platform that do everything well.

[00:33:20] LA: Right. In fact, that's actually one of the things that I'm impressed from what I've seen from Chronosphere so far, is the fact that you know what you're good at, and you focus on that. And I think that's great. And the thing you happen to focus on is incredibly important to the industry today.

[00:33:35] MM: Well, I think it's where – Like, what we see is this is definitely not, I would say, what the majority of the industry is at today. But we feel like this is where the industry is going in the future. So, we're trying to meet people with where they want to get to eventually. And it is a journey and will take a while.

And you can imagine for a lot of companies out there, there are going to have a hybrid mix of some cloud native workloads and some non-cloud native workloads there. So, helping support them through the transition in this journey is going to be critical for us.

[00:34:04] LA: This is almost a straight man lead into my final question, which is, certainly, as cloud native is in its infancy, as you've mentioned. And it's going to mature. It's going to grow. Cloud native is going to become the main way that modern applications are built in the

foreseeable future. And as cloud native matures, how do you see Chronosphere expanding to meet what new demands may be coming out of the cloud native market? And as more mature cloud native ecosystems start to form, what are you doing to Chronosphere to keep at the front of cloud native world?

[00:34:45] MM: Yeah, that's a great question. As you mentioned, and I said this earlier, too, I think we're just at the beginning of this migration. And I think at the beginning of such a migration, a lot of companies out there are sort of looking for more tactical solutions today, right?

So, you can imagine, as you roll out a new architecture, you just need basic infrastructure visibility and basic application visibility, for sure. You can't really operate without these things. You've always had tools there. And you need to continue to have tools there. So, you can imagine, we're trying to help solve some of those particular problems for sure and sort of help companies regain that visibility they've always had before. Just, they perhaps lost it, or it's much harder to get. Now, they adopt this new type of architecture.

I think what ends up happening as companies mature is that they start seeing observability less of a tactical must-have an more of a strategic tool that can give them competitive advantage over their competitors out there. And I think what ends up happening is that more and more, you are trying to get insight into the business in real time, because our technology stacks are really running and operating the products and services that we offer out there. As you start to leverage these tools for more sort of insight into the business, that will sort of, you can imagine, create a ton of new requirements both for the users, but to the business as well.

So, from Chronosphere's perspective, we're sort of preparing for a lot of those use cases and sort of helping our customers not only solve the infrastructure and the application observability use cases, but really start to up-level observable data as a whole and show them what positive impact it can have on a business. And I think along with that, you start to introduce more and more use cases and more and more end users to the system. And generally, the additional use cases and users are perhaps less technical. They're less developers and engineers. So, what ends up happening in the product is that, while there's a lot of cool features and very powerful features, we're really focused on automating a lot of that for these end users who may not

understand – You can imagine, a business unit may not understand retention periods of data or resolution and things like that, but they sort of understand the value and the use cases they're getting out of it. So, perhaps sort of automating a lot of the pieces. Like, optimizing their retention for them without having them think about it is sort of one area that we are moving towards. As well as the concept that I mentioned earlier, where we're trying to sort of customize the experience and the views of the data depending on who it is coming into the system, is what we're trying to do it at a higher level. So, I hope you can imagine sort of expansion of the use cases well outside of just the infrastructure and the application performance use cases that we generally see today. And then sort of optimizing the product for that expanded set of use cases is what we're focused on moving into the future.

[00:37:37] LA: So, the person who is listening to this and is considering, or let's say maybe they've already made the decision that they're going to be investing in building cloud native applications, whether it's porting existing applications to become cloud native, or whether they're rewriting to become cloud native, or building new applications, it doesn't matter. But they're going to be focusing moving forward on building and supporting cloud native applications. And they're just getting into understanding what that means and the advantages of it, etc. What advice would you give to them as far as what should they be looking for in all of their cloud native understanding? But specifically, what about observability?

[00:38:25] MM: Yeah, I think with the transition like this, in general, even outside of observability, my advice may be to think about not just the technology change, but the sort of skill to change and organizational change that has to come along with such a migration as well. It's a very different way of operating. And again, it's not just that your architecture is different and your technology stack is different. Almost how you have to set up your organization, the mindset of the organization where the developers are the operators of everything and the sort of skill sets that each of these folks need are very different from perhaps the past. So, perhaps taking a more holistic view of what this migration would entail. And of course, there are benefits on the other side. So, I think it's worth it. But I think a lot of companies out there perhaps are not taking that sort of broader holistic view of all the changes that needs to occur in order for a successful cloud native migration.

Now, to help support that, I think observability is key because it's the thing that can even measure how successful such a migration is, whether you're getting the benefits that you're trying to get by doing such a migration. Plus, giving you insights into all of the applications and the infrastructure and the business on the other end. So, I think, on the observability side, sort of really understanding that perhaps, compared to the past, monitoring and observability plays such a more key role now than ever before.

And not just that. Sort of thinking about in the future, if you could have real time visibility into your business, not just into your technology stack, what can that really do for you? Especially in the macroeconomic climate, you can imagine of attempting to sort of provide the best service or the best product in a very competitive landscape right now. What role can observability play in that component? And then as you're doing these things, I think, hopefully, scoping down your search space for the options out there for companies and products that are sort of optimizing trying to achieve these outcomes for you. And then making a decision from there.

But I'd say, taking a step back from that, we're always constantly talking to companies out there, even if they're not actively looking for a tool today. We're trying to constantly talk to companies out there about this migration to cloud native, about what it means, about how we view observability, and perhaps how they could view observability just because I think it's a thing that it's going to take the whole industry many years to complete. And it's actually quite a complex move. So, we're more than happy to be helpful in the meantime even if a company is not actively looking for a tool today.

[00:41:04] LA: That's great. That's great. Obviously, I talked a lot about observability. I think observability is critical to building highly scalable, highly available applications. Whether you're using cloud native or not, observability is central. But definitely in cloud native applications, which is the way you need to be moving in the future. Observability is absolutely essential for understanding how your complex system that you're building in this cloud native environment interacts with its environment and knows what needs to happen and knows how to make everything.

So, it's great talking to you. I very much appreciate your spending time with me today. Thank you very much, Martin.

[00:41:47] MM: Of course.

[00:41:46] LA: My guest today has been Martin Mao, the Co-Founder and CEO of Chronosphere. And Martin, thank you very much for joining me on Software Engineering Daily.

[00:41:57] MM: Thanks, Lee. I really enjoyed the conversation. And hopefully, I'll talk to you again soon.

[END]