EPISODE 1295

[INTRODUCTION]

[00:00:01] JM: The company Dynatrace provides intelligent observability, continuous automation and causation based AI to help CloudOps, DevOps and SRE teams move faster, innovate more and transform business outcomes. Dynatrace offers application performance monitoring, or APM. They also offer infrastructure monitoring, cloud automation, application security, and much more. While being an industry leader in simplifying complex cloud applications, Dynatrace is continuously innovating and bringing a wide array of solutions to the market. So who is Dynatrace for and what can it really do for software development and DevOps teams?

In this episode, we're going to get to know Dynatrace a little bit better with their VP, and Chief Technical Strategist, and Head of Innovation Lab, Alois Reitbauer. Alois drives product innovation at Dynatrace and knows more about their products than most. We discussed the use cases and benefits of Dynatrace as well as its involvement with open source projects. Full disclosure, this episode is sponsored by Dynatrace. But don't worry. We're going to get into immense technical detail. It's going to be a fascinating episode despite the fact that it is sponsored. We go into a lot of history of Dynatrace. Dynatrace is a famous observability company. They came to market at a time when there was some revolution in observability. And it's actually a great story. I hope you enjoy it.

Our first book is coming soon. *Move Fast* is a book about how Facebook builds software. It comes out July 6, and it's something we're pretty proud of. We've spent about two and a half years on this book. And it's been a great exploration of how one of the most successful companies in the world builds software. In the process of writing *Move Fast*, I was reinforced with regard to the idea that I want to build a software company. And I have a new idea that I'm starting to build. The difference between this company and the previous software companies that I've started is I need to let go of some of the responsibilities of Software Engineering Daily. We're going to be starting to transition to having more voices on Software Engineering Daily. And in the long run, I think this will be much better for the business, because we'll have a deeper, more diverse voice about what the world of software entails.

If you are interested in becoming a host, please email me, jeff@softwareengineeringdaily.com. This is a paid opportunity. And it's also a great opportunity for learning, and access, and growing your personal brand. Speaking of personal brand, we are starting a YouTube channel as well. We'll start to air choice interviews that we've done in-person at a studio. And these are high-quality videos that we're going to be uploading to YouTube. And you can subscribe to those videos at YouTube and find the Software Daily YouTube channel.

Thank you for listening. Thank you for reading. I hope you check out Move Fast. And very soon, thanks for watching Software Daily.

[INTERVIEW]

[00:03:20] JM: Alois, welcome to the show.

[00:03:22] AR: Hey, thanks for having me.

[00:03:24] JM: I'd like to get into a discussion of Dynatrace as it is today, and how the company has evolved from its initial product suite. Can you just start by telling me how Dynatrace got started and what the initial product set did?

[00:03:42] AR: That's a very good question. I mean, Dynatrace has been around for a very long time. So I've been with Dynatrace almost 14 years. So I'm not even like one of the first employees of the company. So initially, we started building – Well, as the name suggests, tracing tools, that were used or that were capable to be used in a production environment. So back then, there weren't facing tools available. So these were the J2EE times back then. But most of them were pre-production only. So data collection and overhead was a big issue. So the big initial focus of Dynatrace was can we provide a tracing solution that you can just use on your local developer machine, and you can actually use it in production. And that you can use in production in a way that it has low-overhead, it’s easy to install. So back then it was used usually the case that he had to install RAW files on the machines where you wanted to trace the overhead. And sometimes it was only 20% overhead – And use up the tools.

And how Dyntrace emerged, it was really the first tool that gave developers this insight into production environments. Obviously then, a lot of time passed by and we focused more and more on ops use cases, adding monitoring capabilities to Dynatrace. So targeting more the ops use cases while still having the tracing use cases mind, then bringing everything together. And as we were doing this, the environments we were working in we're constantly growing. So I remember, in the early days of Dynatrace, 100 servers. So back then, J2E servers. There was a big environment. And then later on, suddenly, they were like thousand, or 10,000 servers. And today, when we talk about containers, we suddenly start talking about millions of entities. So we saw this constant growth of entities in the environment, and also the environments changed. And that led then to more complete reimplementation of the Dynatrace platform, because we saw that there is just so much data available. It’s really hard to manually analyze it. So asking the right question and knowing what data to look at was really hard.

And back then, we were training a lot of people how to use a monitoring tool and helping people how to understand monitoring, how to explorative analyze analysis, how to use monitoring tools to find problems, optimize your architecture. And at some point we said, “Why don't we automate this? We're teaching people actually always the same things. It's a process that can be taught to pretty much every technology savvy person that's out there. So why can't we teach software to do that automatically for us.”

And that’s part of this reimplementation. It's where we then started to implement machine learning and narrow AI algorithms into Dynatrace to automatically find the root cause, to automatically find changes in the environment, correlate them to certain events in the environment, like a new deployment, a change of user behavior, comparing different releases to each, are looking everything that is connected by building a real-time model of your application. So pretty much everything the human operator would do or an architect would do, but trying to automate it as much as possible of this analysis process.

Interesting back then, I remember some of the first demos we did of this new platform, our customers and early users told us back then, “But if you already know what the problem is, like half this algorithm is built in that does automatic root cause analysis. It finds the technical root cause of a problem.” So very high-level. We look at all of the dependencies in your application. And there is a semantic model that Dynatrace understands. For example, that this loading of a service is related to the resources it's consuming. And if services have dependencies, we would also understand that like if one service goes down, and another one is dependent on it, that it means slows down that there is a causal dependency in there. Obviously, the bigger the environment, the less obviously these things are to see.

So we did this. And then our customer said, “Well, if you already know what the problem is, why don't you fix it?” And back then, when we started, why don't you fix it was a really hard question to ask. You knew what the problem was. But this was before the very early days of infrastructure as code. So fixing an environment was like really hard, because they were so diverse. And a lot of it was still manual deployments. So it was hard to achieve this. But then was the rise of Kubernetes infrastructure as code and these automation platforms, it became much easier. And this is also what we can see today. So we're now moving from that very initial problem where it was really about collecting the data, collecting it efficiently towards automatic analysis, and now automating processes on top of it, what we refer to as data-driven automation. So the idea is if you find, for example, that the problem is that you have **[inaudible 00:08:28]** is exhausted for insert deployment in Kubernetes, we will automatically trigger a random book that would scale up this deployment, for example. Or if there are other issues that you make changes to the environments automatically. So I think the short story of almost 15 years of evolution from data collection to automation that we have in Dynatrace.

[00:08:54] JM: Let's talk about the evolution of infrastructure more generally over those 16 years that the company has been in existence. So you mentioned that the initial application deployment medium was like a J2E application, right? So over time, I think the deployment models have really expanded in terms of variety a lot of its containerization-based, a lot of its cloud-based. The abstractions tend to be easier to work with and therefore easier to instrument. And to my mind, you can also do a lot more in terms of what you are building internally, because of the rise of cloud infrastructure, because of the rise of infrastructure as code and better deployment mediums. So I'd love to know how the company has changed in terms of its software delivery and go-to-market and kind of the product suite in parallel with that overall improvement, the improved evolution of where software has gone since those are early days.

[00:10:02] AR: In the very early days, we started to think as many software companies back then started. It was a very classic enterprise deployment. So you’ve downloaded the software, we installed it. You ran a separate database in combination with the software. And then you had to add your software agents. So we always had an agent-based technology that did automatic instrumentation of the application. But you kind of had to add them to your server scripts. Usually, and it was deliberately saying J2E, because this was really the early Java Enterprise days. This was really before JE that later on became. And this was the first thing that people were struggling with, like where are the scripts to add this? And we talked about this massive WebSphere scripts. And you had to, back then, for example, for a Java application, you had to add an environment variable to the scripts that were significantly long. And if you got it wrong, your JVM might simply not start up anymore. So your web3 servers might not start up. And it was really when the operators had to dig deeply into those files. We later on optimized it by automatically finding the places for a lot of these technologies.

But obviously, this was always kind of cumbersome to manage. And that's also why we then – The first thing we then did, also when we moved to a new Dynatrace platform, we did something very obvious that other companies at that time like New Relic and others were doing as well, because we moved to a SaaS model, because you have kind of like an inception problem if you deploy monitoring platform, because you have your application that's monitored by the monitoring platform. And then the monitoring platform, which requires to have higher availability of the application, you will then again need a monitoring platform for it. And you're like kind of going into this inception model of, "Yeah, who's monitoring the monitoring platforms?"

And especially for customers, we started to say, "Well, we don't want to take care of that monitoring platform." So we switched to a, back then, very innovative model where we said all you can buy is SaaS, which is mostly just consumed it via an API. You still have to install the agent in your environment, which is the only way to get the data directly out of the application and to instrument it. But the rest is done by us. And we also developed a managed version. It's more or less you get still on-prem deployments, because even today, we have customers who want us to deploy in an on-prem or in a dedicated environment that they want to run on. Think, for example, governments or highly-restricted industries like healthcare. So they want to have a more or less shadowed lockdown environment that they're run in.

But what we did, we added our own remote management capabilities on top. So we are managing these clusters for the customers so that they don't have to hire additional people to run even these managed environments, which then brought us from an automation perspective in a situation where we had to manage the upgrades, the management, and the operations of thousands of clusters, where most of them were not running on our own infrastructure. We got the machines, but most of those machines did not belong to us. So they were the customers' environments. And again, these were still early cloud days. We could run it perfectly fine on a cloud instance. We were obviously running on cloud instances as well. But still, it was mostly operating system cloud instances. This was still before really containerization kicked in and environments like Kubernetes.

With Kubernetes now, we had an even greater or easier way to even install our software, because suddenly we could use containers. So we have more and more of a homogeneous environment that we can install to. So suddenly, there's this opportunity you can install on-prem or a software that you run dedicated in a customer's environment very, very similarly to the software that you run for your own SaaS environment. There are still differences though. It's not that is really write once run anywhere that you want it to get, but it's getting very close. And also the deployment became much easier. With operators in Kubernetes, it's much easier to deploy monitoring software, not just monitoring software, but any kind of software, because you can have operators that modify your deployments. You see it for our use cases as well, maybe a service mesh, or tools like OPA and others. And it's also easier to understand what the scripts look like. It's not that there're no flakes anymore.

So the nice thing about what we achieved, obviously in Kubernetes, but also with things like CloudFormation, and similar approaches also exists in Azure with ARM and so forth, we have a standardized declarative way how we can understand what is actually deployed and how we can interact with those environments. We can also use this as metadata. The early scripts where you were you were like remote SSH-ing into the machine and then just executing some shell scripts. You had to more or less discover or reverse engineer what the engineer writing the script wanted the environment to look like based on what is actually running in the environment. And now it's this really declarative infrastructure as code or deployment as code approach. We know what the desired status, which is what like one of the great advantages of Kubernetes. We know what the actual status. And obviously, we can see a configuration with. And based on this information, we can take much more informed decision, whether the environment is doing fine or not, and how it would have later on to change the environment to get better.

[00:15:30] JM: Can we pause on that on the operators and the automatic remediation side of things? So I've done a few shows about operators, and the extent to which I understand Kubernetes operators is their way of defining how a certain application should run. So when you think about Kafka, for example, Kafka can run on top of Kubernetes and use Kubernetes to manage its replication strategy, and manage its nodes, and its processing and stuff. In through the operator, you can manage the common failures that might occur and describe to your application what the remediation strategies are. So I'd love it if you could tell me if I'm correct about like describing what an operator actually is. And assuming I'm correct, explain to me how the rise of operators, or operator-like technology, how does that benefit Dynatrace?

[00:16:24] AR: Yeah. So obviously your description very well matches what an operator is. So you have your desired state. And then you have the reconcile loop, as what it's called, that checks the actual state of the system. And the operator more or less abstracts everything away that you don't have to need to know, because it creates like Kubernetes language. You're using what's called a custom resource definition. So think of it as the main specific definition of what you want to deploy.

So you can write like a definition, you want to run a database-backed blog, or you want to run a blog, which means you will most likely have a cache in front of it, you have a web server, you have a database, and you might have other things in there. But you don't really care about what those things are, because you just say, “I'd like to have one blog.” And the rest is really abstracted away from you. Also when it's created, but it’s also abstracted away from you. How do you think it can be upgraded? You think, “I don't want to run version one. I want to run version 1.2.” And the system is supposed to figure out how to do this.

And there's like multiple capabilities that operators do. So some of them just install. Some of them happy to upgrade. Some of them look at this reconciliation loop and say, “Well, the way you're running it right now is not really ideal, or some components are missing. I have to change the environment.” And the next level would then really be not reconciling how many instances of my database do I want to have running? Or how many like web frontends we want to have running? And how should they be distributed. But you’re starting to trigger on a different set of metrics, like I want my response time to always be 200 milliseconds for my map lock users, or wherever it's supposed to be. So that's what it described, and you want to have a system behind that that figures out how to actually achieve this for you.

And depending how complex this environment is that you're managing, the more complex these decisions might get. Because what does it mean that your response should only be 200 milliseconds? Are you creating more cache instances? Do you need more database replicas? Or do you need to scale up other things? So you need to have more knowledge about how the application works. And this is still an area where some operators do a reasonably good job, but a lot of innovation is happening. And which then perfectly ties into this idea where we go with monitoring. We're not just like monitoring or even observability. But we want to get into this kind of like actionability part where we can change the environment because we know how it's supposed to look like. Then look at what we use as runbooks as code. Think of it like an ifft, but for operators.

The idea is, as a developer, I know if you change the configuration of some of the components of my service, it will behave differently. This can be feature flags. This can really be scale up. Depending on what the service does. Just, for example, the recent Fastly outage. You might define, “Well, if I can't reach my CDN, I might just switch to origin delivery, but I'm trying to cache it massively, or I'm trying to reduce all images that I don't really need.” So the developer, instead of writing a wiki page of a runbook, they're writing this code. And what the monitoring system and the automation layer can then do, it looks at what the actual problem is. Checks it against, “Okay, what can I do to mitigate the situation?” And then starts to either execute these actions automatically or propose them to the user. So it takes like this process that used to be manual and starts to automate it as good as it can. Obviously, this is not like a healing situation for everything. We're not making systems to run fully autonomously, but it can take a lot of these manual tasks out of the equation. It only escalates to a human operator when it's really necessary.

[00:20:13] JM: Great. Now, I'd like to relate this to a conversation about DevOps and site reliability engineering. And the whole DevOps SRE movement is still very young. And it's evolving very quickly. And part of the reason it evolved so quickly is because the state of the art in tooling has evolved really quickly. Like just over the last six years that I've been doing this show, we've seen a change in continuous delivery software, a change in container delivery, container management. And just basically the entire tool chain changes every 18 months pretty much. At least the state of the art that's available, and then people adopt that at different rates. But I'd love to know from your perspective, as the tooling has evolved so aggressively, how has the change, how has the nature of the SRE or that DevOps job changed?

[00:21:08] AR: I think early on, when we talked about DevOps, it was a different conversation. It was just automating what is done manually. So very often, it was, "Okay, don't SSH into a machine anymore." Plus, really looking at the culture that the people had in their companies, where it was like really, very often, you had even like the dev team and the ops team in different buildings. Very often, they were in different, not just organizational units, but different companies. There was very opposite bank X. And this bank X had a separate company. It was running operations. Or they outsource it to third parties.

So the first part of that movement was like really achieving more of this, like looking at the processes. And that's what they keep telling people. I mean, there's always this ongoing conversation. DevOps is not a tool question. DevOps is a process question. It is supported by tools to support your process. But early on, it was like – I think companies started struggle to see that they couldn't really ship software anymore. And the idea of shipping more frequently than maybe every half a year like really scared them. And they really have massively had to rely on people to do this.

And they started obviously with CI/CD. I think a lot at the beginning was, "Well, we want to automatically be able to deploy an application," that for a lot of companies was a massive change. Because, very often, there was somebody who was still deploying the application. So again, going back to the J2E applications at that time, there was somebody going into the WebSphere console and uploading a new offer and then rolling it out across the cluster. So the first step was really about automating manual processes just to increase speed mostly on the delivery side. So that's where that whole – Suddenly CI/CD became the big mantra for all companies to move in that direction. We then saw that people said, "Well, it can shift faster. But how do you ensure that they still keep up quality?" So that's where testing still come in. And it was we're not for access more application, but for very big applications, automated testing, and taking decisions on top of the test results was still a big issue, especially for those applications in large enterprises that have massive dependencies and applications that might have been around since the 1970s. And still, the goal was just making it faster. Being able to increase your release cycle, because they wanted to faster react to customer demand, which is felt that they didn't want just to have like two releases, three releases a year. And when they failed, it was always a massive problem.

And they also didn't feel great for developers. It's not fun to work in environments where you see the results of your work maybe half a year later. Then you realize that nobody actually wanted for it yet. And that was it back then other companies were challenging them to be fair. So for me, that first wave was very much about accelerating. Now, when you start to accelerate, I think that's the second phase that we're in. If you look at tools that came in, and when we started to talk about progressive delivery suddenly in terms of GitHub. We said, "Yeah, we do this automatically, but we also automatically check whether it actually works. And then we take countermeasures." So taking more than a day two operations concepts into account, as well as automating also more of those cases. Because the problem with speed is it's great that you're faster, but your frequency of change changes. And your processes that you had to manage these environments no longer obviously works that well either. If you're deploying three or four times a year in a multi-tiered replication out of the five tiers, it's totally different if you then compare deploying two or three times a day 300, 400 microservices, and obviously there're environments that are much bigger. So you also have to automate this process. How do you deal with this increased velocity?

And I think that's also with the SRE practices. And those are what so until the data Dynatrace. You need this conversation between the site reliabilities and their developers how to best build applications in a way that they can be easily run and easy to manage, and not meaning that they become classic ops teams again and running them for the development teams. But it is a different skill set, just like it's a different skill set to do UL, something completely different to do UX design than it is to build a web applications. While many web developers might have user experience know how or skills, they're not UX designers. And the same is kind of like from a depth perspective. Developers might understand what it means to run apps in production. But the people who do this every day who have seen massive outages, who have been through all of these situations, can give you a different type of input on how to work in this environment.

And from me right now, it's way more in this day two operations focus that we talked about automating more, trying to experiment more, feeling more comfortable in situations where companies are like really moving fast. And fast is, by the way, for me different from everybody out there and what's convenient for them, because sometimes when we're working with customers, they ask us, "So how many releases should we have per day?" So you're asking the wrong question. This is not like a contest. Like I'm doing 15 releases, you're doing 25 releases, and you win. I'm always telling, "Can you release when you want to release? And do you feel comfortable releases? And can you make sure that your customers are not impacted when you're getting it wrong?" That's the question that you usually ask yourself. Not is it 10? Is it 15?

[00:27:14] JM: I don't know if you know, I'm publishing a book called *Move Fast*, on July 6th. I spent about two and a half years writing it. And the whole premise of the book is around – I mean, it's a study of how Facebook build software. But the whole premise of the book is around why and how you should move fast as an organization. And as you said, it does not have to do with these like vanity metrics in terms of how many times you can release software per day. It's more about does the company feel agile? Does the company feel like the product development cadence is fast enough?

[00:27:51] AR: Yeah, I have a very interesting anecdote here. When we started also to move into this very fast release cadence in the early days, when we bought our next generation product, we, for a short period of time, switched away from this like very frequent deployments that we were running. These were just some internal things we wanted to work on. And we said, "Let's not release it. Let's wait a bit longer. And it sounds like something very interesting happened." So the same developer who used to work with and we're constantly releasing. So yeah, he was showing me some changes he had. Let's push it out. Suddenly he went silence. What's the problem? Yeah, it feels weird like that. Why does this feel weird? We have been doing this for so long. Yeah, but we haven't done it the last couple of weeks. And suddenly, it starts to feel weird. I think this whole process of releasing things, things that work perfectly, taking them back. Like having all this in place is also almost like working out. I mean, you can't stop working out, but you will see the effects at some point. So it's not just being fast for the sake of being fast, but making things that are usually exceptional. Something that you continuously do and the team feels highly comfortable doing. Like back to the very early days, these were something highly uncomfortable for people to do, because they were doing it two or three times a year. And maybe it's not just – I mean, I like the idea of move fast. For Cloud Foundry, somebody once was wearing a T shirt for one of my presentations that move fast and don't break things. So against the Facebook mantra. It's also good to not break stuff. So I think it's do what you need to do well frequently. Practice what keeps your business running at what is an important practice in your environment. And I think that that's another aspect of like this increased speed and velocity that you want to see well beyond like all of these activities. If you're not doing it frequently, you won't feel comfortable doing it.

[00:29:49] JM: Okay. So we're kind of talking about changes in the management science of software, where the expectations these days are to be moving faster than we were moving 16 years ago when Dynatrace was started. Just the expectations by employees within the company are that the product releases are going to be more fast and the best employees are going to want to work at that kind of place. Now, in the midst of that change, you continue to release products, you continue to have new advances in what your customers are going to be getting in terms of software that allows them to move faster. And one of those things is cloud automation. So in the midst of thinking about the elements of DevOps, where you have these well understood abstractions of continuous delivery pipelines, Kubernetes, infrastructure as code and just monitoring. The kind of monitoring that is at the core of Dynatrace's product DNA. Explain what the cloud automate – Like the vision for cloud automation is. What part of the DevOps journey are you trying to build within with your cloud automation?

[00:31:08] AR: The idea of car automation really goes back to this, as I mentioned before, these early days when people were telling us, "If you know what the problem is, why don't you fix it? Or if we know that the environment is not behaving in a way, why don't you change it? So traditionally, monitoring tools, which is looking at data and giving you dashboards and maybe some webhooks, but they were not really actively part of this control loop. Or they hadn't fully implemented. Or it was always some tool that you put on top to have this end-to-end control loop.

One example is if you're testing in a pre-production environment or even run some load tests in Dev, or in a shadow deployment, whichever stage you're at and what your preference is, you kind of run those tests, but then you need to feed this back into yet another system. And the system might then decide, "Is this release doing fine? Is it not doing fine?" And then change the environment. The same thing when things break in your environment and you figure out what potentially breaks. And these were always custom-built environments. Like these were these automation platforms that people were building on top of your monitoring data and other data sources. So security, obviously plays a role here as well, and in business data as well. So like conversion rates and like doing AB test, not so much differently. You look at business data, but not performance data or other metrics. And then decide to change the behavior of this environment.

And what we realized is that most of – These like all one-off platforms. Like everybody that wants to get into this environment, in this situation, they're starting to build their own custom built platform. Like, let's build a platform where we integrate 15 tools together and eventually orchestrate them and make them work together. And what we saw when we're doing the first implementations together with our customers is there must be a better way to do this. I mean, basically, we're doing all the same things, but we're just doing them differently over and over and over again. And also people were kind of misusing tools for purposes that they were never build for. It's like the story of using Jenkins for pretty every type of automation just because you can. And there's nothing wrong with Jenkins. It was just built for a very specific purpose as a CI tool. It was built to build software. But then we had continuous delivery, continuous deployments coming in. Then we switch to progressive delivery, where we take decisions based on information we get off the environment. And every move into runbook with automation, suddenly everything is done with the same tool and like kind of cobbled together. So we decided let's build and not just replaced individual tools, but let's build an orchestration platform on top that properly orchestrates those tools based on the data that tells us how the system is behaving, and then automatically taking the right actions. That's more or less the idea behind cloud automation. It's just ensuring the end-to-end flow of software and keeping it up and running, but building it in a way that's reusable and reproducible.

So what came out of this is an open source project Keptn, which is more or less the open source implementation of all of this this automation. We've also started to standardize events or to create standardized event definitions. So today, if you want to use a testing tool, you want to use a deployment tool. You want to use many other tools, you're always coding against the proprietary API. And they kind of all do the same thing, just slightly different. And we said, "Okay, let's start to standardize on what those building blocks," although they're pretty much the same everywhere. And that's just translated to the tools that they work with, the tool providers to provide a more uniform view on this.

Interestingly, we started this a bit more than two years ago. And now there is this initiative going on in the continuous delivery foundation with SIG events, where they now made it the primary charter of a whole now six with special interest group or with a working group to exactly standardize this. So it's easier for me to exchange tools and define my processes independently of tools and have this choreography layer on top. And this is where we what then took for cloud automation, where we bundled everything together the automation piece, the data collection and analysis piece, which more or less enables you to build systems that scale to very large environments, very high-deployment frequencies, very large teams working on them without having to scale it with the operational load on the other hand, because you could automate most of those tasks in an easy way and also in a way that you can plug and play components that you need for automation.

[00:36:05] JM: So what you're trying to do here is pretty ambitious. And I want to break down how it works to some extent. So I think it's key to understand the open source project, Keptn, and particularly what you said about events, so an open format for events. Can you describe in more detail why the open source project is important and why it's important to standardize events? How that event standardization works to facilitate this kind of automatic remediation platform?

[00:36:43] AR: The reason why we built it open source was two reasons. First one was we wanted to really work with the community because some of these like very evolving practices that we're looking at it. I mean, continuous delivery, we're more like industry agreement how it should work? Still, if you look at blue-green deployment, and depending on which tool it is done, it is still done slightly different. But once you start to combine day one and day two operations, so the delivery and then the remediation piece, that there's very little out there in the market that does actually both. And although remediation is still in its early phases, so we decided let's work on something that we can work on together with the community and take this automation and make it available. And we built it on a number of assumptions. So we look at how it's done mostly today, which is scripts and workflows. Again, most of the automation we have to ship applications, to your earlier question, how DevOps tools changed. And we have, for operations, are built on the assumptions that we're releasing infrequently and only a very small number of components.

And we’re arriving these days even to more extremes, where with feature flex different people see different versions of the application. And most of these tools are not built for like this complexity. So the idea was build an automation platform that looks exactly the same way as your software stack looks, because that makes the deployment artifacts the same. So every service has its own deployment strategy. It has its own remediation strategy. So to make smaller building blocks that you can combine at runtime, because what you will see was – Specifically talking about the remediation piece – I worked blogger a while back, I call like micro operations, like operations for microservices. You can't write like this one big runbook for a microservice environment. You can like one component, but it might impact on two or three other components of your environment that you then need to change. An example would be say, we have an application. And just the newest release creates more database queries. So your database is kind of slowing down because it's particularly just too heavy.

So one step might be to scale up the database nodes, but the developer might have also decided, “Well, then cached information that we don't need to query from the database in real-time.” So you might go for that caching approach. As you go for that caching approach and then realize, “Well, there's now way more load on our Redis caches.” So we now need to scale up all Redis caches. Or we start to scale at the front. And so the more building blocks you have that are referencing each other, the more complex these operations workflows get. So you can't put them in a script. It's more the smaller bits and pieces that you then need to orchestrate and bring together at runtime. So the goal was to build your automation logic the same way you built your application. And why do I need to standardize on events?

So what events do, they must take steps in the software development lifecycle. This might be trigger a deployment to my development, my production environment. I'd like it to be a blue-green deployment. And here is the artifact I want to get deployed. And only if all my SLO's are met, I want to keep the deployment running. That's the advice I give to the platform. Pretty much to that same idea where we had working more declaratively how this happens. So I can define all of those processes independently from the tools that you use. It can define a software delivery process that's the same for the entire company. Like these steps always have to happen, but different departments still choose different tools that they want to use, either because of preference or technical necessity, depending on which technology you use, you might still simply use different deployment tools, different testing tools, whatsoever.

And the idea is to decouple that what should be done, like the flow or the sequence of events, or what the process looks like, for how it should be done. Like which tools you're actually using? And the interface between those two are the events. So you're more or less that orchestration engine is emitting events. I want you to deploy Redis environment, I want you to run a functional test. I want you to run a performance test. I want you to – And to propagate this environment in a blue-green fashion to the next level, or to the next stage in the environment. And then people can plug in tools that they're using.

There are several advantages to this. The first one, when we talk to big banks, is governance and auditing. So you can define a process that you find with an entire company without having to look at all the individual scripts that are running. I remember talking to a customer that said that they have to, for compliance, validate every single script that's deploying something in production, or that's automating something, which is a nightmare to maintain. And quite frankly, you never really know, because these scripts get like, incredibly long, like thousands of line, lines of code. And there's also very little reusability. As soon as you start scoping like one script over to the other and then change in two or three files. We did even an internal study at Dynatrace where we looked at how much overlap we had in individual deployment scripts.

So one goal is to separate what the project looks like, which in Keptn, we call the shipyard file. This just defines the sequences of how things should happen. And then we decide which tools we want to use, which will load at runtime and add in there. This allows us, for example, to react initially to a deployment file. Think of it that way. If I want to decide that after a deployment, I might want to run a security scan. I would have to touch each individual delivery pipeline to run that security scan. Well, in this case, I'm just describing to that deployment event with a specific tool, and it will run across all of my deployments that I'm running. I can also way more easily modify how an environment should behave. If I have like this process that I'm reusing, like in our case, the shipyard definition. An example is running an ecommerce shop. During the year, you’re totally fine to run experimentations whenever you want. So you can do AB test and whatever you want. But during Black Friday and Cyber Monday, you most likely shouldn't experiment with your application too much, because that's when you're making most of your revenue.

So the idea here is usually you would have to touch all of those pipelines, or you're sending out an email telling people, “Please don't deploy right now,” if you have it fully automated. In this case, it would just change that one configuration. And it will be deployed across the environment in seconds. And then you simply switch it back. And the last advantage is separation of concerns. If you look at a standard – It might be Jenkins or any other tool out there. You have the process in there, you have the tools in there, and you have details that the developer puts in there. It's basically three people working on one file, and you have no separation of those concerns. And people can more or less interfere with each other. By separating this out into individual artifacts, it's way easier to manage as well.

But I agree. It's a mouthful what we're trying to do there. And it is kind of like a new approach and how to deal with this. But we see more and more interest, like this event-driven declarative approach of day one and day two operations that we're working with.

[00:44:27] JM: Well, the event-driven model, the configuration-driven model, declarative model, is it seems to be the direction that the entire software industry is moving in, whether you're talking about backend or frontent. React, on the frontend, looks in many ways very similar to this this sort of declarative model, and it should be possible, if not today, then soon for more and more of our software to be self-healing and to be declarative, just easier to work with. I feel like it, today, we still are dealing with lots of problems in software that we shouldn't be dealing with that are the result of infrastructure drift or just avoidable bugs. And so, as you're developing this vision, which is to have more of your cloud infrastructure be automated, and fewer problems, you have gotten involved in the CNCF yourself, the Cloud Native Computing Foundation, which is at this point, I would say, as impactful on the direction of applied distributed systems as Linux is for like single node operating systems. So the CNCF is tremendously influential. I'd love to know what your engagements with the CNCF have been and what kind of picture you're getting for the future of distributed application development.

[00:45:50] AR: So our involvement in the CNCF started in a very different are, or an adjacent area. One that makes a lot of sense for Dynatrace, but it’s mostly around back then the

OpenTelemetry project. So this obviously made a lot of sense for a monitoring or observability company to be part of OpenTelemetry. And we worked with lots of people in there well before OpenTelemetry. But as we then moved more into the application delivery space, I was then part of the team that was proposing, “Well, we should have an app delivery working group.” Back then it was a SIG in the CNCS. So that’s the exact reason they renamed six into tax main reason. Was because Kubernetes has six, and it was just confusing. So it's a technical advisory group that specifically deals how to deliver and operate applications. So this was something we founded a while ago. I'm still co-chairing this initiative, where we look exactly into like problems like this. How can we build systems that are easier to deploy? How we can bring tools together? A lot of discussions right now are how can we, for example, built composable or cooperative delivery platforms where we can link tools together? I want to like take for my continuous delivery this tool. For validation, I want to take another tool. For monitoring, I want to take another tool. I want them to work together. I wanted to pick best in breed. And I want to exchange them as I go, which is like super hard. That's also like back to events, because all of them are running proprietary API's that I would have.

So we started to work in there. And therefore, it was kind of like natural when we were working on Keptn. At some point we decided, “Okay, if we really want people to use this and collaborate with them on it, we should do it in a way that the overall industry expects us to do this, and to make it fair.” And we also saw that, as we talk to customers, they wanted to have the security or certainty that, yeah, we will not be able to actively influence this, because we're building it in house. We want our things to make it in there. And we said, “Well, then let's donate it to a foundation.” It was open source overly before. Said, “Let's donate it to the foundation. It's no longer purely controlled by Dynatrace. Everybody can contribute. Everybody can actively influence the direction,” which then led to the sandbox submission there.

And since then, we just started in this ecosystem, which also made a lot of sense for us, because that's where all of the other tools that we are interacting with kind of like we're living as well. And it allowed a lot of integrations. And we have seen a lot of great things happen. Like one of our collaboration was a lot in chaos engineering. So chaos engineering tools, so like litmus actively approaching us and work together how we can more easily integrate chaos engineering. Back to the to the idea of having like these declarative platforms, how can I bring chaos engineering into my delivery process without anybody having to touch any pipelines, but still ensuring that these applications would work fine in production? Or even taking it a step further, what we did was we said, “We have these runbook automation scripts that usually nobody ever tests.” So what people usually do with those runbook automation scripts? They say, “Well, let's see whether my scale of script actually scales up.” But people really test whether it solves the problem with the application. Why you want to actually scale up. And even if they do that we want. It's not like a continuous testing of these environments. So we use chaos engineering to most have a test-driven operations approach where we automatically validate where it is remediations, the self-healing actions actually work. And we're not trying it in production when things break. Also, customers, or people were super fascinated by building self-healing applications. I keep telling the people, “Self-healing applications are like the emergency slides on an airplane. I'm super glad that they are on the airplane. I'm also super glad that I'm normally never seeing them.” And that's the way I have to think about remediation in productions. It's not your standard behavior that you deploy something that necessarily breaks.

And maybe to round this up on the CNCF, it was just great engaging there, because we wanted to build this ecosystem, we're all tools work easier to gather focusing on cloud native tools. And that's how we got engaged. Like with the chaos engineering tools, obviously, with Kubernetes, with R, Go, now working with Falco on the security side. That's how these infrastructure components are built. And I think it’s also the right way to build them, because you build this into the core of your enterprise delivery platform, and you want to have the security. This is built in an independent way. And with like-minded people, you can involve yourself into this as well. And that's definitely what the CNCF and the governance model for CNCF are providing there.

[00:50:48] JM: Cool. Well, as we wind down, I think we've given a pretty good explanation for what Dynatrace is up to as a company, and placing that company in the context of the broader software industry. Just to close off, I'd love to get your vision for how the future unfolds over the next five years or so in the DevOps world, and how that's going to affect the product direction of Dynatrace.

[00:51:17] AR: I think the industry is moving towards more of a platform idea. And I'd like to see a lot of the tooling become more and more of a commodity. Like you would assume that is the end goal, I think for all of us should be provide platforms that make it easier to ship innovation fast, which is like very high-level. So the idea is we should, as developers, feel safe to commit code without having to understand the entire process end-to-end. And at the same time, have the confidence that there are safety measures in place, that in case we have deployed something that does not work or that does not behave properly, the system can – Or the platform that you have from a liberty perspective can deal with it and try to remediate as much as possible.

Awhile back, I used to like an analogy of like self-driving IT, or self-driving applications, which kind of I liked it because it also has like different degrees of how self-driving actually work. From taking over one function, to taking over multiple functions, to human intervention only being the exception there. Because the big problem that did I see is, and I'm quoting one of my favorite books, like *Team Topologies*. And what key component of Team Topologies beyond the platform concept that we discussed right now is also cognitive load. We always wanted developers to do more. First, we wanted them to write code, which was great. Then we wanted them to do testing as well, which obviously made a lot of sense that developers write their own tests. Then we started to have them define infrastructure. Now we want them to take over operational responsibilities. Now we want them to manage platform. So the scope just gets bigger and wider. I think that there is value in it, that you build it, you run it, which is more as you build it, you take responsibility for it and you understand the processes end-to-end. But at the same time, as an industry, we're complaining that we don't get enough people to build all the software innovation that we want to get. And these two don't fit together. We can't expect people who we want to spend more time on building business differentiating functionality to also take on more roles that usually were done by other people in the organization. So that's why my vision is I think we want to bring things closer together. We will build more interdisciplinary tools that can work better together. But a lot of this like repetitive work just to get things out there, we need to more commoditize this. We shouldn't be talking about it. It shouldn't be a key concern.

Like if I want to have a blue-green deployment with three stages, and it automatically validates in five criteria that I'm specifying, I want the thing just to work exactly what I said and not have go to my engineering productivity team, to my DevOps, to my SRE team, whatever the company calls it, to first build this. And then I'm getting it later on. Or then somebody has to manage it as well. And what a lot of these like delivery and operations platform to really appear in behind the scenes, and we stop thinking about them. Maybe like people now start talking about like

Kubernetes becoming boring. It might be like software delivery tells like really – And operations, like really becoming super boring.

And to be fair, there are, or there have been initiatives in this area for quite some time when we think about serverless and other concepts that are emerging that take more and more of this into the platform and exposing less and less of this complexity. So that's the vision that I have. And where I see Dynatrace fitting, and we always need this to be data-driven. We need to automatically drive decisions and drive automation. And automation works better the better your data is. Doing automation on faulty or incomplete data does not lead to good automation. And our goal is really to provide the best quality data to do the best analysis of this data so that you can build high-quality automation on top. And obviously allowing this to do seamlessly in an easy and quickly that this does not become a massive integration project by itself.

[00:55:34] JM: Wonderful. Well, Alois, thanks for coming back on the show. It's been a pleasure talking to you.

[00:55:39] AR: Yeah. Thank you for having me.

[END]