Krish (products.snowpal.com) (00:00.898)
Hey folks, Welcome to Snowpal Polyglot Software Development Podcast. Our guest today is Ajay Chankramath. Ajay is the head of platform engineering at ThoughtWorks, a premier software consultancy, championing modern software development. With 30 plus years of technology development and leadership experience, Ajay specializes in platform engineering, software infrastructure, cloud optimization, and observability. Ajay has been a global technology leader across multiple industry verticals.

So it's very nice to have you here. Thanks for taking the time.

Ajay Chankramath (00:35.101)
Thanks Krish, glad to be here.

Krish (products.snowpal.com) (00:37.682)
And you know the topic for today is measuring developer experience and improving it through platform engineering That's what we're going to talk about in the next 45 minutes to an hour and for folks who've you know watched the previous podcast We've had one conversation with plan about where we've discussed platform engineering with James So if you haven't seen they definitely check that out. This is like this is the second conversation This is an area that's kind of new to me So I have a lot to learn in the next 45 minutes to an hour from Ajay so without further ado

I want to get started, but before we get started Ajay, I know I've given you the brief introduction about yourself, but if you could just speak to it in your own words, just help the audience know a little bit about you.

Ajay Chankramath (01:18.869)
Absolutely, right. I mean, so I've been in the industry for way too long, then, I mean, much longer than I would like to admit, you know, several decades. I've done pretty much everything that you can imagine. I started off as a hardware designer, then I did software design, I did embedded design, then I did distributed computing, I built virtual supercomputers. And, you know, as I was doing a lot more software development over the years, one thing that came to my mind was what we call as developer experience today or what is more easily known as developer productivity.

And as being part of engineering teams, one thing that we all always think about is how can we be more productive? So that's what really brought me into this whole space of DevOps and platform engineering and developer experience and all that kind of space. But having done that, in the past I've been the senior vice president of lots of very large companies. And I've done a lot of architectural work.

design level, but now I put all my time and effort into platform engineering space. So that's what I've been doing. And also a couple of words about ThoughtWorks if you haven't heard of what ThoughtWorks is. I'm sure everybody listening to this have probably heard of concepts like in agile and if you look at agile manifesto we got a couple of signatories from ThoughtWorks in there you know I'm sure everybody knows what continuous integration is, that continuous delivery is. All those things sort of came out of ThoughtWorks.

even if you really look at some of the seminal work within the space of microservices, those things came out of ThoughtWorks and even data mission. So pretty much if you go back the past 30 years and look at all the kind of advances that we have had in the industry, software industry, ThoughtWorks have had some kind of a role to play within that. So I've been doing this at ThoughtWorks for about five years and there's some really exciting times.

Krish (products.snowpal.com) (03:08.308)
Awesome.

Krish (products.snowpal.com) (03:17.494)
Awesome, you know what I'll do is, you know, we'll include links to both your profile and link to ThoughtWorks, any other links you give me when we publish the podcast. Thank you for that introduction Ajay. So let's jump right in. For folks who are watching this listening to it, it's a technical audience for the most part. But let's say if you're talking to a developer

versus a platform engineer. I want to start the conversation there. If you want to kind of introduce the concept of platform engineering to two different stakeholders, one is imagine Krish as a developer who is just building software, doesn't matter what it is, I'm building something on a daily basis. And then you're talking to another person who's actually a platform engineer. What are the types of conversations you'd potentially have with each of these two stakeholders?

Ajay Chankramath (04:01.553)
Yeah, fantastic question. So the first thing I would do there is sort of take a step back. Why does it all matter? So what's the real genesis of all the things that we are talking about here? So if you really go back and look at some of the things that has been happening in the industry, we have heard terms like happy developers are productive developers. Productive developers deliver high quality applications faster. These are things that we all know. Then I remember Satya Nadella, the CEO of Microsoft, mentioning a while back about the single thing

most important thing to invest is the developer experience because happy developers make it more productive. So as any organization doing any kind of digital software development and delivery, which is pretty much any organization these days, you have to think about how do you actually make sure that your developers are more productive. And that's the whole genesis of the platform engineering or the DevOps and things like that. If you go back 15 years back when you know Patrick DuBois actually coined the word DevOps

You know one of the ideas that he had was You shouldn't have like a bunch of developers sitting in one corner and bunch of operations people sitting in another corner and throwing things over the wall to each other. He wanted that communication to get better. So One of the disservice that we did to that whole in a whole term was that the moment we actually saw the term DevOps We all latched onto it. We said oh, let's go and create a whole bunch of DevOps engineers Which to me is a complete anti pattern because when you create

set of engineers who are just dedicated to building and operating software, who are not writing software, then you're still creating that level of abstraction that creates that unnecessary dependencies between each other. So the way to think about it is that, you know, platform engineering, think of platform engineering as that abstracted out set of tools, principles, and techniques that will make your developer's life easier. So how does it all fit together?

It's a very simple way of looking at it, right? Let's start with DevOps. So we know that DevOps, we just talked about that DevOps being a cultural paradigm as opposed to being a team or a set of people. So in order for you to improve that experience of developers and operations folks and anybody in the whole developer ecosystem working well, you need to have those set of abstracted out capabilities. And that's what platform engineering is. So think of it like platform engineering sort of enables DevOps.

Ajay Chankramath (06:32.127)
Then you also have this idea of SRE. Even before the term DevOps came into the picture in 2009, in the early 2000s, Google came out with the idea of SRE. SRE is primarily applying

software principles to any kind of operational activities. So platform engineering supports SRE because in order for an SRE work to be done, SRE is not operations. We still deal with a lot of organizations struggling with what exactly is SRE,

title. So it's literally applying those software principles to operations activities. So platform engineering supports SRE because SREs require a lot of capabilities and just like a dev app would need it and platform engineering provides those capabilities. Now

I started this conversation by saying those developer experience and why the developer experience matter. And in order for that developer experience to be improved, you need to provide developers with those self-serve capabilities so that they could actually go and do the things themselves instead of being dependent on somebody else. Again, DevEx requires platform engineering. So think of platform engineering as something in the middle of all these activities of DevEx and SRE and DevOps and something that brings together all these things as set of tools, techniques, and processes.

Krish (products.snowpal.com) (07:48.242)
Okay, that's a beautiful introduction. So let's say I'm a developer, I'm building software. What, how does my experience change? And here are some of the assumptions that I'm gonna make based on my understanding so far. And please feel free to correct me where I'm wrong. In the case of platform engineering, is the platform engineering team going to provide a set of tools? You mentioned self-serve, an important word there. My question there really is, when you're talking DevOps, that those are pretty generic tools from my understanding and...

use of DevOps, meaning when you go from one organization to the other, one client to the other, as long as you know the fundamentals, you are sort of, you situate yourself quite comfortably. Whereas when we say platform engineering, and you're talking self-serve and tools being furnished to developers, does it mean that it's very specific to each organization? Meaning if I go from one client, one company to another, is my experience as a developer, when it comes to integration and specifically with platform engineering team.

Is that going to be very different or can I carry over some learnings from one place to the other?

Ajay Chankramath (08:51.793)
Again, another great question. Absolutely, you can carry over, and you should be able to carry over if you are practicing platform engineering the right way. So let me again give you a little bit more of a context around this. So historically, prior to platform engineering coming to the fore and being adopted as the right way to build a lot of these abstracted out capabilities to create more self-serve abilities for the developers, what was happening was that people were doing automation.

years and I've done automation you know all that 35 years and I'm sure you know people used to do that prior to those times too in order for us to integrate our software tools and you know build and release various products but one thing that was missing within that whole space was the idea of product management so we always looked at our business products the customer facing products as something that requiring a product management or a product owner and the

And that was one of the fundamental problems. So I'll tell you what the problem was. The problem was that these DevOps engineers sitting in a corner trying to build those right kind of tools and making all the things because they know a lot about systems. They know a lot about software, the systems, the networking and all that. So they build solutions that eventually gets

used by everybody else. But the problem is that if you don't have a product manager or a product owner, you end up building things

wants to be built, not what developer wants to build, therein lies a fundamental problem. So that is the fundamental difference between the old way of automating everything and in the platform engineering side of things. So I'm going to actually list out some of the key components of you asked about, like what's common between platform engineering between organizations. I'll list out a few things. But before I do that, I say the single most important thing is the platform product management mindset.

Krish (products.snowpal.com) (10:23.447)
Right.

Krish (products.snowpal.com) (10:40.715)
Right.

Ajay Chankramath (10:51.547)
It's a very simple concept, right? You build, and you don't think that you're gonna build and they will come and use it. No, that will never happen. You build what somebody wants to do. If you've got 10 developer teams that you're supporting as part of your platform engineering activity, make sure that you talk to them through some kind of a product management function, make sure that you understand what the requirements are, make sure that you prioritize that, and make sure that you assign value to it, and then build it the way they want it.

It's no magic, right? This is how exactly any product should be built.

Krish (products.snowpal.com) (11:25.578)
So Ajay, I have a couple of questions. So before you list out the items you're gonna list out, you mentioned product owners. So am I correct in interpreting that as, typically you have a product team, product owners, product managers, and the whole gamut to understand product requirements, they engage customers from the outside, external activities and research and whatnot, and then bring requirements on into the organization for teams to actually build software, because you actually monitor.

making money and monetizing through that software that you're selling. But what you're seeing here, if I understand correctly, is you would also have sort of a product management team that's geared that plays the middle man or the middle person role between the platform engineers and the developers, meaning, hey, you know what? Much like I'm gonna understand what an external customer wants, I'm gonna spend time and money and effort building it. I'm gonna give you just as much respect, even though you're not really, quote unquote, an external customer, but you're still a consumer of some services.

So my first question is, am I getting that right? That is going to be somebody who's actually interested in understanding the developer experience internally within the company, and somebody is going to come and ask me, Krish, as a developer, what might you like and how should you go about doing things? Is that what you're talking about here, Ajay?

Ajay Chankramath (12:39.141)
Absolutely, absolutely on the point. I think we call those roles technical product managers because the fundamental difference between them and the other product managers that you might have is that every traditional product manager would have understanding of the domain and the product that they're building. They're domain experts. In this case, the domain is the technology. So we call them the technical product managers. But yeah, other than that,

everything that you said is absolutely right. And again, go back to all those quotes I was using in the beginning. Everybody says that, yeah,

developer experience is what's really going to help our bottom line and top line. If that is the case, then that is probably where you should be putting a lot of your focus in, and that's what most of the companies are doing these days.

Krish (products.snowpal.com) (13:22.014)
Okay, so let me play the devil's advocate here. It seems too good to be true, but I know that it's not. It is real. The reason I'm saying that is, it's like a, it's a cost center of some sort, right? I mean, when you're actually trying to find requirements from the outside to build your monetizing directly, but by doing this, you're improving.

a bunch of things that we will talk about but regardless at the start at the offset am I right in stating that it is actually a cost center you're not actually monetizing through this activity

Ajay Chankramath (13:53.917)
That is a long way of doing it, yes, but that's what most people do. So one of the ways in which, I mean, that's why I first introduced the idea of this product management, technical product management. So if we were to just drill down a little bit into that, one of the fundamental areas that you should look at is like, you know, we have this idea of platform value model and we have published and we have talked about that at various conferences and all that.

that value modeling is a key to understanding how that initial investment becomes justifiable. Otherwise, I've been there, I'm sure you've been there, and most of our listeners have been there. We know there is an excellent idea that we wanna go do, but nobody's ready to fund it because they see that as a call center. So how do you change that conversation? So the fundamental way of changing the conversation is that, so I talked earlier about like prioritizing your capabilities and things like that, that whatever you wanna build as part of your platform.

Go one more step, right? Really start assigning or simulating the value of how much value is that going to generate? How do you do that? Again, it can get a little bit complicated, but let me just try and simplify that as much as possible. A simplest way to look at it, a rudimentary way of looking at it is saying that, if I build this particular capability, Capability 1,

that capability one can save an hour of a developer's time. And now you start extrapolating that to all 10 developers in my team and to all 10 teams. Now pretty soon you got like one hour saved for 100 developers that saves 100 hours. Now it becomes a question of your business owner, your product, actual domain product owner to decide, do I actually get to increase my velocity or do I actually get to reduce my velocity?

Ultimately, it's a zero sum game for any organization to invest in all these things. But as a product owner, we know that every product owner is worried about the velocity. How can I get more done from my developers? And when we say, yeah, if you were to actually invest in the platform engineering, you're going to do that, people will say, yeah, theoretically, I agree with that, but I can't do anything because I don't have any money to invest in it. But if you actually flip that script, say that here's the amount of money that you're going to save this quarter

Ajay Chankramath (16:05.783)
increasing the productivity of those developers, which means that you are able to get more out of your sprint. So every sprint you're going to deliver more. I think that's the conversation that you need to have, which would make it a lot more easier from an investment point of view. And

And at some point, your platform engineering teams are only going as good as what your business owners want it to be. And if you think that you from a technology side of things is going to come in and change everything, that isn't going to happen. That's why all these things have to work together.

Krish (products.snowpal.com) (16:38.302)
Okay, makes a lot of sense. So you explain the value proposition to the relevant teams to say, you know what? This team can do a lot more if we actually spend a little bit more time. I wanna draw a parallel. You know, when we do these refactors, you can never explain the value proposition easily, easy enough. I mean, you couldn't take an entire sprint or two trying to refactor something, but you know, you kind of mix and match and find the sweet middle ground, if you will.

to make those incremental changes and reducing your tech debt at the same time delivering something of value. So in that context, I completely understand what you just said, but in that context, how do you quantify that? And actually, let me take back. Before I ask a question, sound of all of this, does it mean that this platform engineering, the one big difference between DevOps and platform engineering possibly, and correct me if I'm wrong, Ajay, is...

that this is much for larger organizations? I mean, platform engineering only makes sense and comes into play for larger companies and not so much for like, I mean, in other words, if you're a 20 person startup with like eight engineers, are we gonna talk platform engineering? Nope, that's not on the table.

Ajay Chankramath (17:47.326)
There is always a tipping point beyond which it'll make a lot more sense. But the way to think about platform engineering is not how big your organization is, but about how many products are you building. So if you talked about a 20% startup, but if the 20% startup is actually building five different applications.

and they are on application five and they still have to maintain the other four applications prior to that, then that's an excellent candidate for platform engineering. But if you are a 20% startup working all on one application, is it really worthwhile for you to go in and do all of those platform engineering, prioritization and all the things that I'm talking about? Absolutely not. But it is not just the size, it's about that. So think of it like, this is where I want to tie that SRU part back into it too.

A lot of the times, what we recommend is that organizations should have your developers own your product through the life cycle, unless and until it has reached a point where it's so mature and it's bringing in so much revenue that you can invest in an SRE on it. Your SRE resources are going to be very valuable resources to invest in it. So if you are going through that life cycle, you can, as an organization, decide which part of your life cycle that you are actually going to be bringing in those advances.

capabilities to abstract things out so that you can build everything the same way. To my example about five different applications in a 20% company, think of it like if all five applications require some kind of a pipeline to deliver the products, would you want to have each of them go build different pipelines or have use one pipeline to use it? Or you want to have, you know, your CEO of your 20% company wants to see all the applications, seeing the state of all

in one place, in one way, would you rather have an observability platform and dashboard that tells you all these things done, all the applications are instrumented the same way and

providing the signals the same way? You probably would want to do that. So it's never too early to do it, but at the same time, if you are just starting something off, definitely not.

Krish (products.snowpal.com) (19:59.538)
Okay, lovely. So let's say I'm a company of that size. Let's pretend that we have more than one application because a lot of us do and we at Snowpal also have more than one application. How, let's pretend, let's play some roles here, right? Let's say this is the first conversation we are having and you Ajay are the platform engineer and I am one of the developers building one of the applications. What would be some of your initial questions to me? Perhaps I said you're a platform engineer, maybe you're a product manager or a technical product manager.

What would be some of the questions that you would ask me initially to understand what my developer needs are so you could actually build something that provides that rich developer experience that we're talking about.

Ajay Chankramath (20:36.273)
Yeah, great question. And there's this very simple answer to that. I'm sure our listeners are familiar with value stream mapping. So essentially, any kind of path to production. So any developer will have a path to production from the point of defining your product to actually getting that deployed. Lots of things happen in between. And we call that using various terms. We call that using value stream mapping, path to production mapping. Some people just call it SDLC. Whatever those things are, first thing to do there

be to just as a product owner or a platform engineer talking to a developer, first thing to do is to have the developer articulate how's that app working out.

How are you actually writing your code? How are you building your code? How are you testing your code? How are you deploying your code to your lower environments? And how are you actually delivering? As if you're talking about some kind of continuous delivery. That would sort of uncover your areas of friction. So this is where, so I had earlier referred to like in a different, and a listing of different planes within the platform engineering, right? So let me just go back to that a little bit, which would make a lot more sense here. As you actually go through the

VSM, value stream mapping, not every pipeline or every SDLC would have the same kinds of friction. For example, your organization, that particular developer, might be using certain tools or systems that are very advanced with respect to one or more of those functions within that.

whereas they might be struggling with something else. So if you were to do that whole mapping, you find that your overall time for delivering your product.

Ajay Chankramath (22:16.273)
assuming there is no continuous delivery, some kind of continuous deployment is there, the overall time to deliver that product, including the testing and all that, is a day. And then you really start breaking that down. See, like, within the day, so okay, day, 24 hours, so start breaking down that 24 hours to see where are you actually spending most time? Where is your biggest friction? How can you actually improve that? That becomes the first requirement or the first capability for a platform that you would end up building. So...

To map this into those planes I was talking about, the reason I talk about it that way is because it's easier to scope the platform engineering. Otherwise, it still remains some sort of an abstract concept. So there are five key planes within that. One is what I call as a developer plane. So this is where you really talk about things to your version control if you're using Cloud, your IAC activities, your development tools, and maybe even things like pay roads. Another one would be something like a compliance

governance plane. And these things are applicable just as applicable for startups as well as for large organizations these days. In the old days, we would think that we would just build some application and then think about your compliance and governance later on. These days, that might be too late for you. So obviously, things like your pipelines, your lightweight governance, your FinOps compliance, and your compliance at various points of change, lots of things can actually fit into that plane. The third one is probably most popular these days. Everybody talks about it. It's the delivery.

Krish (products.snowpal.com) (23:26.507)
Right.

Ajay Chankramath (23:42.315)
time plane, right? You're deciding should I be using containers, should I am I using serverless, am I using Kubernetes, how am I doing my workflow orchestration, all that kind of things. The fourth one would be more if you're networking and connectivity activities. So with, you know, most of these are SDN and things like that these days today, I know if you do in VPC and your external connectivity and third-party connectivity and all that, and which is more true for smaller organizations, but we've seen this across organizations of all sizes.

The kind of dependencies that products have on, you know, weird third party tools running on some kind of a VM somewhere on somebody's desk is almost always a reality. So really have to think about it from that point of view. What are those networking and connectivity activities they should be doing? And the fifth plane is what I call as the security plane. So this is where you're really talking about your IAM, your secrets and encryption management and those kinds of activities. But all these five planes have one, two things in common. One, we already spoke about,

everything should have that product management mindset. The other one is observability because the biggest mistake people make with observability within any kind of software development phase is that we think about observability as an application centric activity.

which is sort of entirely wrong, right? Because the way we should be thinking about observability is that it spans all the things that we spoke about already, which means that it has got activities around your developer experience, your portfolio performance, your infrastructure, and obviously applications, and then you actually think about it from your platform itself. You might also have, you might be using cloud services, and what's your observability with that? You might have incidents coming in from the field, so what's your observability with respect to that? What's your service health as a whole?

So ultimately what we are really talking about is the overall your business operations observability. So pulling the observability as a key element that spans all of these five planes is an excellent way to understand what's the scope of a platform engineering would be.

Krish (products.snowpal.com) (25:41.758)
Lovely. You know, you broke it down beautifully there. Let me ask a couple of questions. So now you've had this conversation, as you know, a technical product manager with the developer, but before we even get to that point, let's say we are a company, let's take us as a company, we have a web application built using React and other platform, other tools. We have a number of APIs built on a number of different languages. Some are microservices, some are still monolith, for instance.

If you took that as generic examples, and if I came to a platform engineering team, my first question would be, let's pretend that we are not doing anything dramatically different in terms

of approaches. We're solving different problems, hopefully unique problems. But if their fundamental concept is we're building microservices, we're building APIs, we're building web applications, what are some of the things that can potentially be unique? In other words, what

Another company that's also building APIs, maybe let's say our APIs are built predominantly using Golang, and another company is doing that as well. Would the platform engineering team have to build solutions slightly differently for these two clients, or is it more like a 80-20 rule where 80% of what a lot of the developer needs are going to be reasonably similar, except that there's going to be some nuances where you need to customize their experience.

Ajay Chankramath (27:01.749)
Absolutely that 80-20 rule, right? I mean, so this is where that economy subscale would come in. So most of what we would typically do, so this is why that again, breaking down into those planes sort of makes sense. So for example, the delivery plane would be, would have like X number of patterns, right? So AWS in the last count, they had 17 different ways of deploying a container, right? Maybe more now. So if you know those things, then those are the patterns. And if you cover those patterns

platform, meaning like if you enable your developers to use those patterns to deploy the containers the way they want to do it, that's it, right? You know, there's no other way you can do that. So there is that limited scope that you can actually scope out by having these planes. So that's on the delivery plane side of things. On the developer plane, it's far more easier, I would say, on some side of things. For example, if you're doing IAC. So most people are using these

Ajay Chankramath (28:01.483)
if you're in cloud formation or any of the other tools that you're using, depending on your cloud provider, obviously there are some base, the foundational things that you can provide there. And that becomes far more easier for developers to now be self-sufficient to provision their resources without shooting themselves in the foot. So that's where you have your governance and compliance built in, come in. But there might be some component of things that we spoke about that 80, 20, or that 20 person would be more along the lines of your paved roads.

Because you obviously I mean Netflix and a popularized the concept of paved roads and have these kinds of more repeatable patterns We call us reference architecture or starter kits and things like that those things are Where you would actually create take those common components out of these five planes

and some of the observable instrumentation, all that, then create something that is very specific to your organization. So if you are creating those fundamental set of platform capabilities, then you have the ability to actually take that and create some paved roads initially, which makes your developer's life so much more easier right off the bat. Now they can just use that reference implementation and then say, OK, I'm going to make some tweaks and go and implement my functional logic in that. So that's how a lot of it together.

Krish (products.snowpal.com) (29:19.23)
Okay, so let's say the other two options here. Let's say if I'm a company that has a number of applications is large enough and I need to work with a dedicated platform engineering team, I have a few questions there, but let's say I put that in one bucket. And there's another company, a startup, that has platform engineering needs, except that they don't have the money to afford a dedicated team that can have this technical product manager and work with them that closely. For that,

If I took, first of all, is it correct to take them as two different sort of buckets, and if that is not totally wrong, for a startup, are there platform engineering tools that you can sort of like a manage service, manage platform engineering service that you could leverage where you're like, it's self-serve not just from a developer standpoint, it's also self-serve from an onboarding standpoint, meaning I'm talking to nobody, I'm talking to no companies, I'm signing up for a platform engineering service, and it walks me through.

in porting over however it is that I'm doing, whatever it is that I'm doing using DevOps today I can sort of seamlessly migrate to a platform engineering world. Is that a possibility?

Ajay Chankramath (30:23.532)
Uh.

That is becoming more and more of a possibility. And so I'm sure you have all heard of this term called internal developer platforms. So these IDPs, and sometimes the word IDPs are used strongly. Some people use that for talking about internal developer portals, like backstage and all that. But that's not what I'm talking about. What I'm talking about is internal developer platforms. So typically, the way to think about internal developer platforms is that anybody doing any kind of development should have some kind of a pipeline.

your use case, a small organization using something, they're building something on Azure, they're using Azure DevOps in ASDO. So within that, you know there are certain steps that sort of pre-packaged for you. It's sort of like available right out of the bat. So you have this availability that you can just go in and do all your typical SDLC activities within Azure DevOps and get those things in. At a high level, yeah, that is sort of like an engineering platform. But what it really doesn't do

what you're doing within your business. So remember I talked about some of the paved roads and activities like that, that really increases the developer productivity, reduces the cognitive load of the developers. That really doesn't happen until you bring in some kind of, let's say, an orchestrator. So this orchestrator would be, you know, what I'm really talking about here is not like a container orchestration, more like a platform orchestration. So if you're doing the platform orchestration, the kind of things that you should be thinking about is like, is there a way in which developers

Krish (products.snowpal.com) (31:46.178)
Right?

Ajay Chankramath (31:53.559)
and define what they need using some kind of a simple language and have a tool go in and essentially orchestrate all these other components around it, all the other these components. This might be your CI, it might be your testing, your automated testing, it might be a CD, it might be any of those components that you typically use in your SDLC. So the industry is moving towards a way in which we take things like prepackaged

Ajay Chankramath (32:25.459)
engineering platform ecosystem like an Azure DevOps and applying things like orchestrators in it using some standard languages that can make that process easier. So I know this is not a direct answer to your question. So what people want to do most of the time is that, can I just go out and buy something and get it over with?

Krish (products.snowpal.com) (32:46.978)
Right.

Ajay Chankramath (32:47.453)
that we, I don't think we are in that level of maturity yet. This is where Gen.AI is helping a lot. So one of the things that we are doing some research in, and there's some companies who are coming out with some of this product.

Krish (products.snowpal.com) (32:59.986)
You know Ajay, you mentioned the buzzword I was going to mention in the next 10 minutes, but sorry. If you don't mind, I want you to hold your thought there because I want to actually get to it because no conversation can, even a dinner cannot end today without a conversation about NVIDIA or JNAI or something of that nature. But I want to talk about, okay, so I think I'm getting the picture here. So platform engineering, it's adding value. Now as a developer, what are some of the...

straight up differences I'm going to notice. Let's pretend that I'm, I've been building software and just to set the context here, you know, as a developer, I'm selfish. I want to build my feature. I want to fix the bug. I want to implement whatever it is that I'm doing. And I honestly don't want to do anything else outside of that. I want to throw it over the wall and say, Hey, it's your headache now. You want to push it to the lower environment, the higher environment, manage it using Grafana or Prometheus, all of those, those observable observability things and everything that needs to be done.

Ideally, if the developers don't have to worry about it, we can build more features, fix more bugs and build more services. But that's not always possible. So things have evolved over the years. You mentioned DevOps, you mentioned SRE, and now you're talking platform engineering. What would be my, on a day, on a, let's say I come on a Monday morning, I show up at work, and now things have dramatically changed. It's platform engineering is the word of the day.

What would my day look like? I mean, how different is it gonna be and what are those things that I will be directly impacted by?

Ajay Chankramath (34:26.869)
Great question. So here's a way to think about it. When we were throwing things over the wall, there was a perception that I didn't have to worry about things that I didn't want to worry about. But what happened was that people started measuring these things. I know most of our readers are familiar with the Accelerate book that talked about, that sort of introduced those four key metrics.

These are metrics like as a developer if I'm actually deploying something, what's my deployment frequency? What's my lead time? What's my you know mean time to solve issue? No, no the issues so when you really start looking at those 4k metrics one of the things that researchers found was that Your developers are not being very productive by having that additional set of people in the mix What do you really want to grow those capabilities enabled for you? So as a developer what changes?

Fundamentally, what changes is that more advances that we make on the platform engineering side, more of those things become invisible for you. It becomes your NFRs. So your non-functional requirements are built for you. I'll give you a very simple example.

Your compliance team, and as a developer, you're writing your software and you're getting it deployed. Your compliance team is going to come, used to come back to you at the end of the whole thing, the two days before the releasing, they stop the release because you're not compliant on these things. What if you had an opportunity using the platform capabilities?

to build your compliance at every point of change within your SDLC, then you are never really going to go beyond a certain, because you're gating. I mean, these are concepts that we have played. At least I played around for 35 years. I'm sure nothing has changed on that front. But we use different terms and have a more informed way of looking at it.

Krish (products.snowpal.com) (36:08.694)
Right.

Ajay Chankramath (36:17.297)
you make sure that your compliance gating is done better, sooner, without you knowing about it. How would that happen without you knowing about it? Because as a developer, I actually write my code and I actually go to my IDE and click a couple of buttons and it does everything and it goes to deploy. But in between that, the kind of feedback I get, the feedback loop is faster feedback. I get, say, hey, it's a compliance error here that is happening because of this. Now, I don't need to know as a developer. I no longer need to know all the things that I need to do

for me because this is available as a platform capability that is common across every software developer. So that's the improvements that developers would typically see. The other thing is that we talked about internal developer portals earlier. So I'm sure most people are familiar with the idea of backstage and how that helps and all that. I was working for a company about 20 years back. I remember building my own portal to do the kind of things that we are talking about today.

Krish (products.snowpal.com) (36:53.582)
Amazing

Ajay Chankramath (37:17.111)
The idea there is very simple, right?

you really see that what are the kind of things that a developer need and can I actually make sure that you don't really have to go through a ticketing system to get things done. If I need a VM in the old days, I have to go open a ticket and wait around for something. If I need something, I have to go talk to somebody. How about if I just take that completely out of the picture and just have an internal developer portal to take my request, translate that, use an internal developer platform or whatever automation behind the scenes

that on a real-time basis. That's the other thing that changes. For example, if I need something for the cloud, do I have to write Terraform to go do that? So that was the interim period, the early days, prior to cloud, we are all using some kind of configuration management things to do that. Now we can use Terraform, but how about if you don't even have to do that, if you abstract that out? So that's what you should really expect as a developer, to see making sure that your development life is far more easier.

Krish (products.snowpal.com) (38:17.726)
You know Ajay, you read my mind that was literally going to be my next question. I was going to say, if I'm actually dealing with Terraform and infrastructure as code today because I did one podcast not too long ago, just exploring Terraform for instance. I think it's a couple of months back. So in the context of platform engineering, would I still deal with Terraform directly or would there be a layer of abstraction between me and Terraform, something provided by the platform engineering team?

Ajay Chankramath (38:46.897)

You could go either way. So it's a choice that's made by the developers through the product management function, right? You can, as an organization, decide to abstract that out because any kind of, I mean, the kind of things that we specify in our, you know, the requests that we make to provision the resource on the cloud, we know what those things are. Once you know what those things are, you can automate it. I have, my team have actually gone into organizations

platform that uses Terraform under the hoods and also provides Terraform and the kind of cloud formation of whatever that you need if the developer wants that level of flexibility. But could you actually take out of that 80-20 rule? Absolutely. You can actually make that far more easier by making it like a sort of a click-offs way of doing it.

Krish (products.snowpal.com) (39:38.59)
And now I'm thinking it's more a 60, 40, then 80, 20, the more you talk about it because you mentioned a layer of abstraction with Terraform, and we talk cloud formation as well. We've internally done some comparisons to see what might work better for us as an example. It tells me that given the number of three major cloud providers as we speak today, at least the highest level, Azure, Google, and then AWS. And even within each of the providers, there's more than one option. Even with AWS, sometimes you go.

find out what you want. There's like 16 different things available because of the legacy of when something started and what supported and what's not, which means it's not straightforward saying, hey, I want a persistent, I don't know, a NoSQL database, it's simple to use. If I go to AWS, I have like seven different options, as an example. So does it mean there is a lot more customization than perhaps even what I thought at the top of this call? I said 80-20 because I thought a lot of this could be pretty commoditized if there's such a word, and there's 20% that's very specific.

But the more you speak about this Ajay, it sounds to me like things could be quite different between two organizations because of two fundamental reasons. One is sure, the technologies that they're using, the providers that they actually work with is one. And two, the skill sets of the team. If you're working with a development team that's super comfortable with IAC and Terraform, and maybe the technical product manager is gonna figure out gaps elsewhere,

and try to fill those gaps and not worry about creating this level of abstraction that sort of gets terraform out of the picture. I don't know, I said too many words there. What I'm trying, I don't know if it makes sense or I can repeat it. What I'm trying to ask here is, is this going to be customized to each organization to allow to that level because you have to know not only the technologies that the organization is using, but also the skill sets of the team because you don't want to reinvent the wheel and solve problems that they don't actually need to be solved because they're pretty good at solving those problems already.

Ajay Chankramath (41:36.121)
You get the nail in the head, right? I mean, so this is the primary reason why this cannot be a product, right? Because the skill sets of the teams are different. The requirements of the teams are different. Even though within the space, everybody wants to do some kind of infrastructure provisioning. But how do you wanna do it? What's the level of applications that you have? What are your runtime requirements? Remember when I was talking about runtimes, I was talking about the patterns that you would have.

Maybe we are dealing with an organization and one of the big banks I work with has got a mandate in there that's one of the biggest banks in the US has got a mandate saying that they should be serverless first. So when you have that mandate, most of their investments, most of their platform investments are going to be tailored towards that serverless activities as

opposed to something like a Kubernetes kind of activities. So eventually that's the difference. But do I see that converging at some point?

Absolutely. And that converges, we can get into how that converges might happen, but until that happens, there is still going to be organizations that are going to be extremely similar, but they're going to be very, very different too. So conceptually, these are all the same concepts. These are all these five planes that I talked about. Everything fits into that five planes plus observability and product management. Nothing else is there. But...

What happens within that space, within each of those planes, that differs. And that's the part that makes it difficult to make it sort of replicate things from one place to the other.

Krish (products.snowpal.com) (43:13.378)
Totally, right? And also one more thing I can think of is tell me if this makes sense. If I did not have a platform engineering team, sometimes you end up going with a certain provider, not because you want to vendor lock-in, it's simply because it's easy enough. Like for instance, at Snowpal, we use AWS quite heavily. We have some dependencies on Azure, very little and some on Google Cloud, but not a whole lot, right? It's almost like 95 and then two and three or something if I had to put a number to it. And the only reason is not because we don't believe

the other ones could be better in certain cases. It's only because you're used to the console, the CLI, the interface and yada, yada. And you're like, you know what? Unless there is a significant value add, I don't want to learn one other thing that may not bring value here. But whereas if I integrated with a platform engineering team, I feel like it brings an advantage here because you as a platform engineering team know what is the best way to solve this problem. Maybe Azure is a better way to go as opposed to AWS. Hypothetically, I know there are a lot of comparable services most of them offer.

but there are still those differences. Does it make sense that as a company, you get a benefit from a lot of these different providers because now the developer is not worried about learning every different provider and their services and you provide this nice, this two part question, but the first part is that there is this nice layer of abstraction that the platform engineering team brings to the table. Is that like a reasonable pro to call out for here?

Ajay Chankramath (44:38.189)
It absolutely is. And I'll take that one more step further. Right. Most of the companies these days that we work with, and this is going to be a reality with any large organization, it's going to be multi-cloud. So you don't really have an option of saying that I'm Google Cloud versus Azure versus AWS. No, that is not an organization for a business. I mean, for a business to decide, why would you have to literally migrate all my AWS workloads from my last M&A that I did to Google Cloud? There's no particular reason.

Krish (products.snowpal.com) (45:06.541)
Great.

Ajay Chankramath (45:08.103)
services are a lot comparable. If it's not comparable today, it will be comparable tomorrow. So why would I do that? So you're really operating in a multi cloud world. If you're operating in a multi cloud world, expecting your developers to be familiar with all these services, the nuances of these services is completely ridiculous, right? So that's where that platform extraction would make a lot of sense because then you need to define things one way, let the platform take care of some of the other additional complexities and additional

for you. So make sure that it's validated. So as a developer your productivity increases. Again, continue to focus on your core functionality, your core domain, what you need to do in whatever business that you're trying to make money for. That's where your focus ends up being.

Krish (products.snowpal.com) (45:53.77)
Perfect, that is a positive. Now I wanna play the devil's advocate again. Does it mean I have to learn something else that's not standardized? In other words, it's one thing to learn Terraform because I could use that learning no matter where I go. But if a platform engineering team tells me, Krish, here is our DSL, this is our process, learn something entirely new, I mean, it's not necessarily a negative. I mean, if you're a large organization, people are there for long periods of time, it's all right.

But let's say if it's reasonably fluid and you're moving from one place to other, like a managed services provider, then does it mean that team, the development team has to relearn every single time because every platform engineering product is gonna be slightly, ever so slightly different from the other?

Ajay Chankramath (46:36.189)
This is where the whole concept of enablement through developer portals come in. So developer portal simplifies that whole thing. So remember the days in which we had to go in and look at all these kinds of complex libraries when you're doing, I don't know if you have done C++ development, I mean you do those kinds of things the old days, or C development have done those things. You have to go figure out all those libraries and figure out all those templates and interfaces and all that kind of things.

Imagine a situation where all those things are simplified for you through some kind of a portal that enables that for you. And that's where we are really going with, you know, sort of marrying the whole portals idea with the platform engineering idea.

Krish (products.snowpal.com) (47:16.558)
So the sound of portal comes across as extremely fabulous to me actually, but just a little bit of skepticism for no good reason. Are those portals gonna be available to that extent or is it gonna be like, hey, you can do like 20% of what you need through the portals, but the other 80%, you're gonna have to do the plumbing yourself.

Ajay Chankramath (47:34.94)
Um.

You know, good question again, but it is a combination of those two. The reason I say that is that portals cannot build everything for you. Right. So if you take something like backstage, it is like Jenkins of the old days, right? Jenkins is great, but Jenkins by itself is no good. You need to have your million plugins to make it work. Same concept with something like backstage. You know, you can add any plugin you want and who does that plugin edition? That is your platform engineering team adding those plugin

So the way it should literally work is that if your platform engineering team is building this cool capability that's actually going to make sure that your developer's life is easier, instead of having to teach that developer the whole thing or every developer having to learn the whole thing, again which increases the cognitive load which is completely losing the plot, make sure that those plugins are available in your portals so that you have extended your portal. Your portal didn't come with those capabilities but you have extended the capability of the portal.

having the best of both worlds that way.

Krish (products.snowpal.com) (48:37.614)
Okay, you know, we have like another 12 minutes. I probably need more of your time. Hopefully I'll convince you to come back again. But in the 12 minutes, I want to get into the topic that I interrupted you. Sorry, very intentionally. So, a generative AI. Now, let's say you had platform engineering. We talked DevOps, SRE, and then platform engineering. Now let's say somebody is pretty comfortable with the concept of platform engineering, understands the value, has integration. Everything is hunky dory. Now, 2024.

Nvidia and all the buzz around a generative AI even seeing that you know We've seen stock prices like if somebody just said it you see a dramatic difference How different I mean what is the role what are we looking at right now? How different is it going to be in the next three to six months? from a developer experience or is it going to be completely abstracted because The platform engineering team is going to benefit from all of this generative AI stuff from building the service standpoint

So the developer's experience stays intact. They just don't know how the magic happened. The magic somehow happened. I've kind of mentioned two extremes. I just want you to explain to us where we're gonna be.

Ajay Chankramath (49:43.261)
Yeah, so before I get into that very specific topic, what I want to talk about is at a very high level, right? What are we really trying to solve here? Ultimately, we are trying to make us all more predictive, using Gen.AI, meaning the whole idea is that if organizations could use Gen.AI to produce software, they can fire all the developers and make more money. That's that ultimate, that not star goal. But...

Today, if you really look at improving that operational productivity improvements, there are a few components to it. The two main components to me is like that data overload and that proactive issue detection. I mean, those two are huge components of it. So maybe...

Those two things can lead to, again, from an operational point, it can lead to more root cause analysis, whether you're coding or whether you're deploying or whether you're having your customers using it, right? Your RCA can be done. Once you know your RCA, then there is some level of automation that can be done, right? I mean, so to make sure that there is self-healing and those kinds of things happening. Then once you do that, then it's an optimization thing. You know, if you're doing something that takes like 10 hours to fix something and I've worked in a financial technology industry,

processing is always a huge issue, right? You're throwing all kinds of automation and remediation. What if you can start optimizing that? We've done lots of work around this. So if you take those five areas and really trying to see how can you actually apply GenAI to improve this, improve the overall experience.

even before GenAI became a big thing in the past couple of years, we have been using predictive AI for a while, right? This is generative AI, but we've been using predictive AI for a while. Predictive AI has been more along the lines of self-healing. So if my disk space is actually dropping down to 80 or dropping or going up to 80 percent, I might better go in and remediate now, otherwise my disk might actually fill up and my process will fail, right?

Krish (products.snowpal.com) (51:36.216)
Right.

Ajay Chankramath (51:40.735)

Those kinds of predictive AI has been there. Now with generative AI, what we are really trying to see is that could we actually use prompting to create, now start creating some of the DevOps LLMs, which we don't have anything today, that has gone, these models would have a lot more knowledge about your context within what you're doing. But again, apply that 80-20 rule, right? Those 80% of the common stuff plus your 20% of the stuff that you might add to it. So you could literally take some of those

common LLMs and train it yourself and tune it yourself to improve each of those layers that we are really talking about. Right? I think that's the way it is going. There are some products that's already out there that is actually applying some of the prompt engineering for doing that translation to do some of the regular tasks. So most of you have, most of the listeners, I'm sure have used things like chat ops, right? If you're using Slack and if you have done some chat ops activities, you know, you know, you can even create some,

webhooks that actually look for some keywords and do certain things, right? And I've implemented some of those kinds of things many times but a simple thing might be like, yeah I need to know how something works. So if I actually go in and put something in my Slack channel that has got this webhook, it basically looks for that keyword and says it, okay this person is asking for this thing so I'm going to provide you that document as a simplest level. What if you take that to the next level with the prompt engineering? What

Krish (products.snowpal.com) (53:05.707)
Right.

Ajay Chankramath (53:10.375)
defined my scenario saying that here's the kind of you know remember that terraform example that we are using earlier what if I were able to go in and say that

I want to write a couple of sentences. I'm actually building this application for this particular activity, and I need some resources provisioned. Because as a developer, I don't even know what resources I need. I'm just saying that here's my application requirements. What if that prompts can be translated to everything that we spoke about so far, your IAC, your platform capabilities that translates and creates and provisions the resources for you and gets it based on that prompt?

Krish (products.snowpal.com) (53:35.307)
Right.

Ajay Chankramath (53:50.135)
That's the level of maturity that we are here today. What we don't have is that those, you know, customized LLMs that can actually do some of the things that we are talking about, we are far from it today as an industry. So when people talk about having some kind of major, um,

you know, impacts in software development. And copilots have actually, copilots, essentially multiple copilots have actually, I think, you know, created the sense of illusion that, you know, everything is going to be now coded, you know, by something else.

And if you think about it a little bit, you know that is not in the near future. It's probably several years out, if it ever happens. But some of these activities within platform engineering is far more closer, right? If you can actually start making sure that some of the LLMs are set up properly, you build on your chat ops activity, build on your AI ops work that you had done in the past. Try and invest a lot more in the prompt engineering side of things, and then try and make sure

you can actually start implementing those platforms you're incapable of is a lot more easier, making sure that your developers can, the cognitive load of the developers continues to drop. But then you don't. Sorry.

Krish (products.snowpal.com) (55:03.638)
So actually does, sorry, does that mean, you remember you mentioned the portals, right? Self-serve portals and dashboards, et cetera. Now, given what you just mentioned the last few minutes, does it mean that entire experience could potentially change meaning rather than me integrating or leveraging platform engineering through the self-serve portal, where I have to learn the pages, the user interface, the actual features and whatnot?

my integration is going to change as a developer. Now I'm going to deal with it at a Slack and a chat level where I'm going to ask these conversational questions and sort of a conversational UI, if you will. So am I, I mean, are we talking about getting rid of these self-serve portals in lieu of actually being able to ask these questions and type those sentences? Or do you see a world where both of them are going to coexist?

Ajay Chankramath (55:53.833)
So for the foreseeable future, both of them are going to coexist. But if you really look at.

companies who are investing in portals, you know, there are lots of different vendors who are doing lots of very interesting work within the space. Pretty much everybody is looking at the impact of Gen.AI. Everybody's looking at how would they actually at least start with the prompting side of things and make sure that interaction becomes a lot more easier and less error prone. Because the biggest challenge with any kind of interface or any kind of software is your, the amount of cognition that you need to understand what the system requires

and then translate that from what you need, right? And if you can actually bring it closer to that developer, which is your developer's language, if the developer is speaking in English, make sure that the developer can actually communicate in English, if the developer is speaking in Mandarin, make sure that the developer can actually converse in Mandarin to get what he or she wants. I think that's the level at which we are sort of starting to see things. Right now, for the foreseeable future, these things are going to coexist because you'll never really be able to take one off the other.

People would always want to simplify things, anything that you want to do. I mean, these days, every second person go look at AskChatGPT. So like that, you want to have your chat GPTs for your platform engineering or any kind of developer assistance. But eventually when it matures, I see a product where internal developer portals, internal developer platforms, and the prompt engineering all sort of come together to become that new platform engineering stack as we see it.

Krish (products.snowpal.com) (57:27.498)
I'm going to share a quote here. You know, internet attributes quotes to the wrong people, but this one is typically attributed to Mahatma Gandhi, but hopefully he's the one who actually said this. And hopefully it's not the, I'm going to tie something here, at least I'll give it a shot. You know, the quote is, you know, first they ignore you, then they laugh at you, then they fight you, and then you win, right? Whoever, I think it was Gandhi, but it's a great quote. And the reason I mentioned that here, Jay, is I want to share an anecdote.

You know when I'm building software today at snow pal The first one of the things the questions we seem to ask ourselves all the time is if we do if we are solving a problem Like we would have done it six months ago three months ago We are not comfortable. It makes us uncomfortable because we are like here I mean not doing it right because you know that things

have changed quite a bit There is other they know in the space of JNI. There's so much happening

So should we not be doing it differently? So it's kind of a weird situation because I don't think this would be the question we would ask ourselves on a daily basis. It's almost, when I'm writing code, it's very uncomfortable personally. If I approach the problem the exact same way that I did six months ago, I was like, oh Krish, am I doing the right thing here? And the example I wanna give is the other day I was using Copilot and I was trying to add a comment, right? And this is a code comment for a method, Go lang methods are like 10 lines.

and we don't comment the obvious. It's not one of those companies where you're stating it get restaurants is like, oh, this is returning a restaurant, right? So it was a comment that was more business related, not so much because if you can't, if the code doesn't tell you, then it's bad code, right? So it's not the code commenting. So I put the two slashes and I was trying to formulate the sentence to explain the business, not the code. I cannot explain, I know it was remarkable. After like two seconds, it wrote something and I'm like, this is...

I felt very uncomfortable and it was very eerie because it's almost like somebody read my mind. It was not there in the code. And I'm like, how in the world is this thing possible? And I've shared that in a couple of other conversations and I share that here with you particularly because if I do convince you to come back again, I want to hit upon this topic of Gen.AI and what it means in this space specifically in platform engineering. And I'm trying to have this conversation about everything we do because if you're doing product management and creating tickets, like you mentioned that too.

Krish (products.snowpal.com) (59:44.47)
We don't want to create tickets the way we did that like six months ago. You want to go to Slack and say, you know what, create a ticket, assign it to Ajay and have him work on it, etc. And then it should all be done. Things have changed that much. So I want to kind of pause here because we've had a lovely conversation. And I'm actually running out of time with you. I just want to if I could ask one question to end this conversation for somebody watching this or listening to it because we have published on multiple platforms, people watch and listen.

Somebody's starting their career. They are excited about platform engineering, about everything that's happening. Where should they start? I mean, if it's programming and it's coding, there's one other answer, but if I'm interested in platform engineering, either I'm new, I'm just getting out of college, or I've done this a while, been coding, but I wanna get in the space, what would be some pointers that you could give us?

Ajay Chankramath (01:00:36.069)
Yeah, this has been one of the big challenges that we've been seeing. So I, together with a couple of my colleagues, took the liberty of writing a platform engineering book. And our book will be coming out sometime later this year. This is published by Manning. It's called Effective Platform Engineering. So right now, this is gonna be one of the huge challenges to go find out where the right kind of information is. So again, for selfish reasons, I'm going to recommend ThoughtWorks.com.

and just type platform engineering and you would get a very nuanced view of what platform engineering is and how it sort of applies to it.

Krish (products.snowpal.com) (01:01:15.186)
It's in a jade anything but selfish because given the value the kind of value that someone like you brings to the table just in terms of me learning in the last 60 minutes. I think I would certainly would include the link there but please go please keep going.

Ajay Chankramath (01:01:29.169)
Yeah, absolutely. So I think, you know, so the big problem is that if you just go do a Google search on what platform engineering is, you're gonna get lots of opinions and perspectives, not what we believe, or I personally strongly believe based on my years of experience, and my co-authors of this book who collectively we got close to 80 years of experience writing this book. So what we believe as platform engineering is what you might hear somewhere. So that is

a huge problem within the industry, which is the reason why we decided to write this book. So, again, best place to start right now is that come to thoughtworks.com, just type in platform engineering and you can read some of our perspectives there. I would also, for those of you who are more technically oriented, most of you might know chief scientist, you might have heard of him at least, Martin Fowler. So, go to MartinFowler.com, you can try and type in platform engineering and platforms.

amazing articles there and of course Martin is one of the leaders in this industry within software engineering for many years now. So those are two reasons.

Krish (products.snowpal.com) (01:02:39.114)
Super, and then the last question I had was, actually two questions. One is, what are some, you know, it's one thing to learn something, but the other thing is, how do I know if that would be something I would enjoy? The simplest answer is do it, right? If you don't enjoy, move on. But short of that answer, are there some traits that people could have as engineers that makes them a better fit for platform engineering than being a developer?

Ajay Chankramath (01:03:06.365)
Yeah, I think absolutely. As you know, I for again, for this is my biased view of this, but I'm going to throw out my bias anyway. There is no reason why somebody would not love if you like software, there is no reason why you would not love platform engineering because platform engineering is software, you know, end to end software. It's just that you take your software and make it more of a technical implementation as opposed to going and writing your software for your business applications of your new

Ajay Chankramath (01:03:36.379)
technology that you're doing. This is a pure technology software work. This is all about building software the right way. There is no cutting corners and saying that all the principles that we talk about and how to build right software and how to write the right software. 100% of those things are applicable here. If anything, this is a little bit more complex because of the fact that you have to really think about your infrastructure, you have to know about your networking, you have to know about your whole system as a whole.

this is where a lot of people should look into but in a it's probably not for everyone because a lot of people don't like some of the aspects of infrastructure and things like that they find that to be you know a nod to their to their taste with respect to what they want to do but the other thing is that you can apply any kind of languages you know we use a lot of Golan we use a lot of you know Java we use a lot of lot of different languages within that so there is no real reason to feel like platform engineering is something

less sexy when compared to when you look at the overall software engineering space.

Krish (products.snowpal.com) (01:04:41.994)
Got it, so you answered all the easy questions, the last question is the most difficult question. What is your most favourite food and why?

Ajay Chankramath (01:04:50.389)
Good question. Yeah, I know this is really tough because I'm a foodie. I mean, those of you know me knows that I like all kinds of food. So my favorite food, I would say, is Thai food. And specifically within Thai food, the one I like most is the tofu putt fat. I mean, this is something that I've had the luxury of and the privilege of having for many, many years. I absolutely love it. Tofu putt fat.

Krish (products.snowpal.com) (01:05:15.07)
No sorry Ajay, what is the name of the dish?

Krish (products.snowpal.com) (01:05:19.786)
Wow, okay, I've... Okay, I will look that up.

Ajay Chankramath (01:05:22.606)
I'll send a link over to you. I even have a rating of the best restaurants that serve that food across the... It's vegetarian, yes. It's tofu, yeah.

Krish (products.snowpal.com) (01:05:30.145)
Is it vegetarian or is it?

Krish (products.snowpal.com) (01:05:35.186)
Okay lovely, we'll certainly check that out. How do you spell the last, you said tofu?

Ajay Chankramath (01:05:42.161)
Yeah, so let's just do a Google search on tofu.

Krish (products.snowpal.com) (01:05:50.61)
I ask this specifically because we are a big fan of anything tofu related. So I definitely wanna check that out.

Ajay Chankramath (01:05:56.986)
Yeah, Tofu P-A-D-P-H-E-T.

Krish (products.snowpal.com) (01:06:02.77)
Okay, cool. Awesome. That's all I had and anything else that you want to add in terms of closing comments to this podcast. First of all, thank you so much. This is a wonderful conversation. There's plenty of learning personally for me and I have no doubts folks watching this will not only learn will have more questions as well. But if you want to sort of wrap this up with anything that you want to say.

Ajay Chankramath (01:06:25.585)
Oh yeah, so thank you, thanks for the opportunity. You know, I always love to talk about platform engineering at any time in the middle of the night, somebody calls me up and says, let's talk about platform engineering, I'm happy to talk about it. But yeah, I think the ask I have is that, this is not a fad, this is not a passing fad that you should really be thinking about. This is there to say, this is the evolution of software, develop software, development and delivery, as we have seen over the past, at least over the past 40, 45 years.

So there is no escaping it. This is not replacing anything. I mean, some of the ridiculous arguments you hear is that platform engineering is the new, that it's replacing DevOps. No, it is not replacing anything. DevOps doesn't go with it. That's a cultural paradigm. It is not, it's not a construct like platform engineering, which sort of brings together that tools, techniques, and process together. So pay attention to that, understand what it is, and whether you like it or not,

platforms, you are already using engineering platforms in your day-to-day life as a software person. So that's where I would end it.

Krish (products.snowpal.com) (01:07:33.686)
Perfect. So I think with that, we'll end this podcast. Again, folks, this is Ajay, who's the head of platform engineering at ThoughtWorks. This conversation is scratching the surface about platform engineering, as far as I'm concerned, or at least a second time scratching the surface as part of our podcast. The topic, the title was measuring developer experience and improving it through platform engineering. There's a lot of learning. We're gonna publish this, include the links, so you can check out ThoughtWorks, and I'll include Ajay's LinkedIn profile as well.

So Ajay, thank you so much for taking the time. I'm just gonna end it here.

Ajay Chankramath (01:08:09.685)
Thank you.