Google Prodcast Season Four Episode 10 | The One with Ben Good and Our Kubernetes Friends

[JAVI BELTRAN, "TELEBOT"]

STEVE MCGHEE: Hi, everyone. Welcome to season four of The Prodcast, Google's podcast about site reliability engineering and production software. I'm your host, Steve McGhee. This season, our theme is Friends and Trends. It's all about what's coming up in the SRE space, from new technology to modernizing processes. And of course, the most important part is the friends we made along the way. So happy listening, and remember, hope is not a strategy.

—

STEVE MCGHEE: Hello.

KASLIN FIELDS: Hello. I'm very excited for this very special episode, Steve. [LAUGHS]

STEVE MCGHEE: This is a very special episode. We're kind of double dipping here. This is great. So we're going to do one amount of work for two amounts of output, which is, I think, fantastic.

[LAUGHS]

So I'm Steve, and I'm pretty sure you're Kaslin. And--

KASLIN FIELDS: Yeah, maybe we should introduce ourselves since folks may not know us. Yeah.

STEVE MCGHEE: That's a good point. That's a good point. We have two sets of audiences, half of each don't know what's going on. So this is perfect. Why don't you go first, Kaslin? Who are you?

KASLIN FIELDS: All right. Hello, everyone who listens to the prodcast and maybe doesn't listen to the Kubernetes Podcast regularly or if you're just tuning in, wherever you may be listening from. I'm one of the co-hosts of the Kubernetes Podcast from Google. And I do all sorts of things Kubernetes, cloud native, containery related.

And I'm excited about our episode today to talk about platform engineering, which we'll get to in a minute. But first, Steve.

STEVE MCGHEE: Yeah, who am I? I'm Steve McGhee. I was an SRE for a long time, and I'm in DevRel now, which means I get to talk on podcasts and stuff about reliability and SRE-ing, but also other things that are related. So I was on a team with Ben, actually, our guest on a DevOps-focused team, and we kind of went into the platform space.

And I host The Prodcast, which is a podcast you should definitely subscribe to, if you're a regular Kubernetes Podcast listener. You should probably subscribe to both all the time for the rest of your life. That seems like a good idea.

KASLIN FIELDS: Definitely, yes.

STEVE MCGHEE: But there's a pretty good amount of overlap, I'd say, in the kind of reliability space and the platform space and the Kubernetes space. So as you'll hear today, I think we're going to talk about one of my favorite Kelsey Hightower lines is Kubernetes being a platform for platforms. And here we have someone talking about platforms. It's going to be great.

So dearest guest, Mr. Good, would you please introduce yourself? Who the heck are you?

BEN GOOD: Yeah, happy to. My name is Ben Good. I don't host a podcast.

STEVE MCGHEE: Oh.

BEN GOOD: I feel really left out right now.

[LAUGHTER]

I might have to change that in the future. So we'll see how this goes. But I'm a Cloud Solutions Architect for Google, which is a fancy way of saying that I take the different Google products and not Google products and smash them together into solutions that hopefully solve customer problems and make things easier for everyone trying to do things in the cloud. I've been doing that for, oh, quite a few years at this point. I lose track and have to do math. But--

STEVE MCGHEE: No need, no need.

BEN GOOD: Lots of fun stuff. And I've been doing platform engineering of late or, actually, for quite some time, actually, for now at this point.

STEVE MCGHEE: You've been doing it-- you've actually been doing it the whole time, Ben. It's amazing.

BEN GOOD: Yeah, that's--

STEVE MCGHEE: Who knew?

BEN GOOD: Prior to Google, I did operations for startups and then a few different startups in the Denver, Boulder area where I'm located. And I was doing platform engineering back then. And we just didn't call it platform engineering. When DevOps started to be a thing, it was DevOps. And now, it's called platform engineering but with a slightly different take.

So I've been doing it for a while. I just didn't necessarily call it that or realize that's what was happening.

KASLIN FIELDS: I'm really excited to dive into this. Forget about the podcast stuff. This is what we're here for. So let's talk about platform engineering and Kubernetes. That famous Kelsey Hightower quote that you were mentioning, Steve, Kubernetes is a platform for building platforms. Is that true, Ben?

BEN GOOD: I think Kubernetes—

Is Kubernetes all you need to build platforms? Granted, Kelsey didn't say it was the only thing you need, but-- [LAUGHS]

BEN GOOD: It is definitely a tool in the tool kit, and it's a big one when it comes down to it. Kubernetes provides a lot of different constructs and capabilities that make it a whole lot easier to build platforms. So in my opinion, it is one of the tools in the toolbox to build a platform and make it successful. But it's not the one and the only thing.

Way back when, when I said I was doing platform engineering before it was called platform engineering, there was a company I worked for, built would have been called a portal in today's language. But it was a way that other engineers in the team could go and spin up VMs that were in the platform or databases that were in the platform.

When we containerized and went to Kubernetes, a lot of that stuff became so much easier and went away because the container abstracted away the VM-ness of things. So Kubernetes is super helpful, but it's not the only thing that you need to make a platform, in my two cents.

KASLIN FIELDS: So I'd love to know more about what else goes into it these days. I know, last year, two years ago, when was it that we were all excited about Backstage as a tool for-- when I think about platform engineering, I'm imagining teams at various companies who are taking the underlying infrastructure and making it something that developers and other technologists throughout the organization can consume.

So what does that look like for you today? So Kubernetes is the base of managing the underlying compute infrastructure. But there's so many other types of infrastructure that you have to deal with. And then you have to deal with also how are people interacting with all of that. So what kinds of tools are you using to make that happen? And does that describe what you do? [LAUGHS]

BEN GOOD: Yeah. No, that's perfectly it. That's a very good description of what it is. So the way I think about it is when you're doing platform engineering, you're taking the underlying technology, like you say, Kubernetes for compute, maybe managed databases for your database infrastructure, maybe self-managed databases. All the things that you need to make your applications run and run at scale, those are the things that you're building interfaces to and you're applying automation to.

And I really think that platform engineering is the process that you go through to glue all that stuff together. So from a technology standpoint, you're seeing lots of this typical-- a lot of Bash and a lot of scripting and a lot of Terraform and a lot of YAML. And it's all stuck together with some sort of automation tool in the background to run it and orchestrate it. And then that interface is fungible.

I've recently been working on a project where the interface is a document in Firestore. It's not a fancy UI. It's... write a document in the proper document format in the Firestore. And then automation kicks off, and magic happens. And then that is the interface to it. You can get more advanced or user-friendly with a tool like Backstage. But it doesn't have to be that. It's just got to be some sort of well-defined interface.

And that's really what it comes down to. And that needs to be easy for your users or your engineers to adopt and make use of.

STEVE MCGHEE: Yeah, there's a phrase-- I know it goes around with platform engineering, which is the idea of golden paths, which is if you're going to do this thing, if you're going to go down this road, don't just wander through the wilderness because there's a lot of branches you can get caught on. There's a lot of dark corners where bad things happen. You can fall into a well or something. I don't know where this is going.

But the idea is, if instead someone has already taken that path for you and has laid some bricks behind them, saying, this is the way, just follow this thing, then you're more likely to succeed. And often, from what I've seen, is this tends to be like-- it's kind of what you were saying with whether it's a document or a portal or something like that. It's really just, at the end of the day, a form of abstraction.

So you want the developers who are like, look, I just want to get to the end of this road so I can write my code. Don't make me learn about the entire forest of infrastructure. Just get me to the point where my app is running, and it's scalable and all this stuff.

And what you're saying is, yeah, provide a series of paths, please, in some form of abstraction. If that's a portal, great. If that's a document, great. Just something to make it a little bit easier on everyone. Does that line up with--

BEN GOOD: Yeah, and that documentation is a good counterexample to what everyone thinks of-- was like, it's got to be super easy and WYSIWYG and all those kinds of things. No. The golden path can be as simple as a document that lays out the three, four, five steps that you have to do to accomplish a task. And that is an example of a golden path.

You can make it a whole bunch more than just documentation if you want to, but it doesn't have to. It just has to meet the engineer where they're at and the tasks that they're trying to accomplish.

KASLIN FIELDS: And that's one place where platform engineering really gets tricky, I think, is that you've got so many different ways that you could find your way through this forest. And so one of the key things that platform engineers do within an organization is understand how different parts of the organization need to use that infrastructure.

And I love that you gave the example of using a Firebase document as the interface for the workflow of doing whatever this thing is because I think Backstage offered a vision of a unified interface of here's how you request compute from different sources and things like that. But it can be much simpler than that, depending on the use case. Just give them a Firebase document that they can put the thing in, and it'll go. I love it.

BEN GOOD: Yeah, exactly. And in the case of the Firebase document, you could write that through a little CLI. You could do that through Backstage. There's no reason you can't make that call via Backstage. There's millions of different ways that you could get that document into Firestore. It doesn't really matter how it happens so long as it's easy enough for the people that are trying to use that system.

KASLIN FIELDS: Flexible but structured platform engineering.

BEN GOOD: Well said.

STEVE MCGHEE: It sounds like a good tagline. We should put that somewhere.

BEN GOOD: We should.

STEVE MCGHEE: So I have to wear my SRE hat now. Once you've gone through the phases in your platform, whatever methodology you're using, where you've abstracted away a lot of these inconvenient truths about infrastructure, you've said, don't worry about it. Just follow this path. You're good. You're basically getting the user, the developer through the provisioning stage to the initial deployment stage.

But it turns out, there's more stages. We're not done yet. And this is where I come in. This is where the SREs come in, where it's like, OK, cool. We've deployed the thing. It mostly works. What's next? What do you have to do next in terms of the life cycle? Sometimes we call this day two operations or day two whatever or just observability.

What are all the things that you would now want to make sure that your developers have access to maybe through a portal, maybe through another method, beyond just that startup process? What happens in the operate phase of this life cycle?

KASLIN FIELDS: Yeah. So I think if you lump all that together, you need to provide visibility into what's happening with the workloads that are running on the platform. And that takes lots of different shapes and form factors.

So if I think about the "traditional," in air quotes, operations metrics, like what's my latencies? What are my failure rates? Did my deployment succeed? Those kinds of things, that's a level of visibility that you can expose back up to those engineering teams using this platform engineering concept. Make it easy for those developers and engineers to see what's happening.

But I think that there's more to it, to your point, than that. There's visibility into, is the application working in the way that the users expect it to? So are the features and functionality happening there? That's something the platform can provide up. Cost controls can come up through there, visibility into different security and compliance of regimens that they have to adhere to, those kinds of things that need to come up into the platform.

And those are, to your point, all day two things. Great, you got the first couple releases out. You got us some users. Now, you have to keep it up and going. And the platform can provide those ways and those golden paths to make that easy to see.

STEVE MCGHEE: Yeah. Sometimes they're referred to as nonfunctional requirements. This has nothing to do with widget selling, but you still have to do it. Sorry. The worry that teams have when they're not building the stuff through a platform is like, now that I've deployed my widget-selling device, now I have to do day two, but it's really more like day 222.

It's going to take so long to-- which regulation are we subject to? And how do you generate these artifacts for which lawyers to look at? What are you talking about? So what are some patterns that you've seen customers of ours adopt to-- how do you do this in such a way that it's like part of the platform and not just like a thing you have to do? It's like a tax on your time. How do you do this properly?

BEN GOOD: Yeah, I think that the platform can provide those things for you. So if we go back to using Kubernetes as an example, there's things that you can do in your Kubernetes deployment around making sure that the right policies are in place, the right RBAC, all those kinds of things are in there such that when you get a namespace, that those policies just come along with it.

And then that's an example of where you're getting those things via the platform for free. You can do those same things with metrics and observability and logging. You can plumb those up through such that the application teams can go in and see their logs without having to do any sort of extra work or magic to make that happen.

STEVE MCGHEE: Or log in to another tool or something like that. And like, oh, I need the credentials for the logging app now. Great.

BEN GOOD: Yeah. So it's not like those aren't super-- those are examples. Every implementation about how you go about it is a little bit different, depending on the tool, stack, and all those kinds of things. But that's what the platform can do for application teams and those other engineers is just do that for them. The term is shifting down. You might have heard that before.

Those are examples of where you can shift those responsibilities down into the platform, and you just get that stuff for free because your application is running on that platform.

KASLIN FIELDS: Observability is so important. I often hear platform engineers that I talk with when we ask for feedback on features that we're developing and things like that, one of the things that I often hear is this kind of consideration of, well, that feature is great, but when I bubble that up to my users, how am I going to make sure that they know the pieces of that that are going to keep them from spending a ton of money or running up a crazy amount of compute or something like that?

I need to make sure that it's all observable and visible to them through the platform that we are building. So--

BEN GOOD: Yeah.

STEVE MCGHEE: Yeah, I feel like another thing that we talk about that is kind of what you're getting at, Kaslin, is like when you're developing a platform-- basically, a platform doesn't just come out of the box in perfect form, as Ben has described, has all these things. You have to develop it.

And I think this is a basically per customer problem, like per entity like org, because every company is a little bit different and has different requirements and has different-- there are different levels of capability and just staffing even and regulation. And they're all slightly different snowflakes. And I mean that in not in the pejorative sense, but just like they're all very different from each other.

So one thing that I've heard-- and maybe, Ben, you can back me up on this-- is that whenever a team builds a platform, they tend to name it. They give it a name. And it's sort of a pet. And then they take care of it like it is-- this platform is a pet. And you love it, and you feed it. And you extend it to exactly what you need as a team because it's always going to be a little bit different. Does that line up with your experience?

BEN GOOD: Oh, yeah, yeah. Platforms are very much bespoke things to the engineering organization that they serve because, to your point, every engineering team operates at a different skill level, a different layer in the stack. You might have different teams that are responsible for different types of infrastructure that need to come together to work on the platform together.

The one that I was lightly referencing to earlier, it was called Divine Spork. That was the name that GitHub auto-generated for me. I'm like, yeah, that's kind of fun. We're spoon-feeding, but it's kind of pokey. So it's a spork. And you get things by magically, so it's kind of divine. So that's what we called our platform at that company was Divine Spork. So yeah.

They're very much bespoke things. They have names. They have teams that rally around them. But yes, you do care. You care and feed for them and love them, and hopefully, they love you back, and all that kind of stuff. But yeah, every platform is different. And I think it's a bit of an antipattern to go and look for the platform in a box that you can just click button Install and like, woo! I have a platform. That doesn't typically happen.

STEVE MCGHEE: Yeah, as we alluded to before, like Kubernetes being a platform for platforms. I've actually heard of Backstage being a not a portal but a system for building portals. So it's the same idea of like, it has all these knobs. Don't use them all. Don't just click Install. You got to think about what it is you're trying to get out of this.

And I think we've said in the past, you want to-- the abstract idea of a platform is just a grouping of capabilities. And so what are the capabilities that your team wants to adopt and then deliver through this platform? Whether it's through the Kubernetes side or through the Backstage thing or through the observability suite that you have through another company or through the-- whatever.

There's a load of different things that you could be, a lot of stuff you build yourself too, right? But at the end of the day, it's the set of capabilities and the fact that they're working together in concert, hopefully, that makes it beneficial to your company.

BEN GOOD: Yeah, very, very much so. The folks that install and are successful with Backstage, they have teams or a group of people that are modifying and building plugins for it and creating the different templates. It isn't just something that you get to install, but you build on it, that exposes those critical user journeys through the portal in a way that works. And that takes development effort and maintenance and care and love as well.

KASLIN FIELDS: And this customization for different organizations and for different types of users within organizations, there's a term that we used in our notes in preparing for this interview, which was deployment archetypes. And I feel like all of this lends itself to that term, which I hadn't really seen before, but I kind of really like it. So is that what you all think of as deployment archetypes is making these workflows customized for the environment that they exist within and the users that they're serving?

BEN GOOD: I think that what you want to do is you have a pattern for ways that you deploy things. And those patterns are what you end up exposing out through the platform. So those would be the archetypes. And you can largely group them like we've done with-- Steve is well aware of-- different reliability characteristics built into them.

So I have this thing of this shape and size. I need to deploy it on this type of infrastructure with this load balancer configuration, this database configuration. And that supports this shape of user across these different regions of the world or this form factor. And you provide n number of those things in the platform. And those become the archetypes, if you will.

And those can be genericized. And we've published some documentation around those. But that's the type of thing that you want to expose out through the platform. And then making it easy to pick the right archetype or change between archetypes is a task that the platform helps you with and gets you down that journey.

STEVE MCGHEE: So a good way to think about this for the pure Kubernetes' viewpoint-- a good way to think about this is like when you're trying to make a system more reliable and more robust, often what you do is you're like, we have one of these things. How about we have two? Just have a second one. And that's great. It's a good idea.

That's generally the method that we use to make things more robust is that we have two of them instead of one or n of them, right? But one thing that you could do naively is you're like, we have a cluster of stuff. Let's make a second cluster. Done. Stamp it out. Good to go. Ready, set, go. Except you put it in the same zone. You have two clusters that are in the same zone. And then when there's a problem in that zone, guess what?

You actually only had one. You did not have two. Too bad. Bummer. You get a bad day. So there's this idea of failure domains. And this will be familiar to a lot of SREs out in the world. But you have there the matryoshka dolls of like, you have your app, and then you have your namespace, and then you have your cluster, and then you have your zone. And then you have your region, and then you have the universe or something.

And so failure can happen at any point along these things. And so there's this paper that came out-- I don't know-- years ago at this point called deployment archetypes, which is how to apply this understanding of failure domains to when you're doing deployments. And this is where zonal deployments versus regional deployments versus global deployments versus multi region versus multizonal versus blah, blah, blah.

There's these very abstract ideas of basically where to put your app, in which clusters, at which time. And it's too much. It's hard to understand all of it. And so what we've seen people do in the past-- and Ben has done a really good job of this with some customers that I've seen of-- is that there will just be a dropdown in the portal or in that doc or something. It's like, do you want this to be a big one or a little one?

And under the covers, it's picking one of these. And it's making a bunch of choices for you so you don't make the wrong choice. So it's like instead of saying, just give me two clusters in the same zone, it's like, no, no, no, no. Would you like a big one or a small one? And you say a big, and it's like, OK, I'm going to give you two clusters in two different regions and not tell you about it.

But then also the deployment is going to be able to take advantage of that. And the observability is going be able to take advantage of that. And you don't even know the whole time. It just magically happens. That's the vague idea that I get. But Ben has actually done it, so I'm curious if that holds true when you hit reality.

BEN GOOD: Oh, very much so.

STEVE MCGHEE: Cool.

BEN GOOD: Yeah, it definitely holds true. And the thing that you're doing with that little dropdown is you're abstracting away a lot of the super nitty-gritty detail of what cluster, what subnet, what this, what that that you might have to go and know how to do otherwise. I think, a lot of times when folks are trying to build a platform, you, like, throw a bunch of Terraform or throw a bunch of YAML at folks and say, OK, go do the thing with the YAML or the Terraform or whatever your favorite language is.

KASLIN FIELDS: A lot of the things that we've talked about in our conversation today are kind of evergreen topics-- reliability, making sure that the systems keep working, making sure that you understand how they're working, building golden paths. But I think a lot of that has shifted recently due to industry trends. I wanted to know if you've seen any of that in your day-to-day work, Ben, if the types of golden paths that you're building these days are different and how and all of that kind of stuff? [LAUGHS]

BEN GOOD: I think the underlying technology changes a little bit, but really, it's still the same thing, I think that we've been doing, just with slightly different twists or name or a change of focus on how do we do it at scale, if you will. I think the rate of change in the industry as a whole is-- it's always ever increasing, right? So I think the things that we have been doing, we just have to do more of those things and maybe with different technology, not doing something different. So--

KASLIN FIELDS: Yeah. In the Kubernetes space, what I've seen a lot of is hardware accelerators. It's a lot of the same kinds of stuff. But now we're enabling folks to use different types of hardware than we've used in the past and for applications that are very particular about how they're using that hardware. So Kubernetes has traditionally allowed folks to-- a lot of the management of that underlying infrastructure is handled for you through Kubernetes.

And then you add another layer on top of it, the platform, that users actually interact with to further abstract that. And now, folks want a different type of abstraction for those types of workloads that really need careful control over those underlying hardware resources. So that's where I've seen Kubernetes doing a lot of work recently to adapt to new changes. [LAUGHS]

BEN GOOD: Yeah, no. That's fair. Back when Kubernetes was young and when I started using it, it was CPU and memory. Those were the things you had to be concerned with. And now, there's networking. And to your point, there's GPUs. And how do you go about getting access to those things?

But yeah, you're beginning to add more to it. But really, you're still orchestrating the access to the underlying infrastructure and providing more control than you were way back forever when. But it's the same process or the same thing you're accomplishing, just a little bit different.

STEVE MCGHEE: I've heard that also customers will have different motivations for platforms than they used to, like in the post zero interest rate, ZIRP phenomenon. There's been a lot of consideration around cost management and cost observability and things like that. And frankly, part of the problem with a naive level way of doing these abstraction through platforms is to just hide everything. And then you get a bill, and you're like, oh, whoops, we did something.

And so the ability to expose usage patterns and like, oh, we accidentally-- all the databases early and often, right? So you can actually see in somewhat real time your consumption of the database or whatever the expensive thing of the month is and be able to control that and make a change. So if you do a deployment that accidentally writes 10 billion times, you should be able to, as a developer, notice that and rectify that, all on your own.

So self-service is one of these things that is really, really ubiquitous in the marketing of platform engineering. But often we think of that just in terms of like, well, I want to make my service for the first time like that Greenfield step. But self-service also applies in the operational context as well. So does this come up as a motivation when customers are talking to you about platform engineering? Or am I making this all up?

BEN GOOD: Yeah. There's some customers that approach it from a cost control perspective. Not many approach it from a self-service standpoint. I get a lot of customers that look at it from a security compliance governance perspective. And those are reasons to go about it. But that can't be the angle that you take, in my opinion.

So we can't take the angle of, well, we're going to do platform engineering to reduce costs, and we're going to begin to make our engineering teams do something to reduce costs. We want to improve compliance and governance and those things. But forcing engineering teams onto the platform doesn't typically yield the desired outcome, if you will. There's, oftentimes, reasons why that happens.

Or you want to listen to those engineering teams and begin to design your platform in a way that supports those engineering teams but also accomplishes the security and compliance and governance and cost controls. And those things come along when you begin to actually do platform engineering. So the motivation can be there, but it can't necessarily be the reason that you go to engineering teams and say, we're going to do platform engineering, and you're going to use the platform because of this. That usually doesn't yield the best platform.

KASLIN FIELDS: That opens up a whole new can of worms that we could dive into for a whole nother episode, at least, diving into security considerations, regulatory compliance issues, and working all of those into how we build platforms for specific users. But I think we've covered some really awesome stuff today with making sure that you're designing platforms to serve the users that they need to serve and that a lot of the underlying concepts are the same as they've ever been.

It's about making sure that your infrastructure is usable by the teams within the organization. And I'm excited for all of those platform engineers out there listening to do that

STEVE MCGHEE: Another thing that we didn't get to, which I'm amazed, Ben, how did we not mention DORA? I'm even wearing the shirt under this jacket.

KASLIN FIELDS: That's true.

STEVE MCGHEE: So I know, we blew it. But being able to track your team's success through metrics and being able to show that these capabilities are actually making an improvement to the lives of the developers and to the business, go to dora.dev to see what that even-- what all those words mean. But delivering it through a platform is totally a good move. I believe you would agree with me on that.

Side note, Ben and I work on the platform engineering chapter of the DORA survey every year, so I know he agrees with me.

BEN GOOD: It's a lot of fun. I definitely agree. A fun thing to think about from a DORA perspective in your platform engineering endeavors is using DORA metrics to understand your feature velocity and your application development practices but also your platform engineering practices and your platform velocity, which I think is a fun thing to think about.

STEVE MCGHEE: Yeah, platforms are software, too. Well--

BEN GOOD: That's right.

STEVE MCGHEE: --thank you, Ben. we'd love to have you back either on each of our episodes or maybe another double episode. Who knows? We'll, it'll be great. You never know what's going to happen in the future as we just learned.

BEN GOOD: This is a lot of fun. Thank you. I really appreciate it. Thank you much.

—

[JAVI BELTRAN, "TELEBOT"]

JORDAN GREENBERG: You've been listening to Prodcast, Google's podcast on site reliability engineering. Visit us on the web at SRE dot Google, where you can find papers, workshops, videos, and more about SRE.

This season's host is Steve McGhee, with contributions from Jordan Greenberg and Florian Rathgeber. The podcast is produced by Paul Guglielmino, Sunny Hsiao, and Salim Virji. The Prodcast theme is Telebot, by Javi Beltran. Special thanks to MP English and Jenn Petoff.