EPISODE 1173

[INTRODUCTION]

[00:00:00] JM: Infrastructure at Spotify runs at high speeds. Developers work autonomously building and deploying services all the time. Backstage is an open source platform built at Spotify that allows developers to build portals for making sense of their infrastructure. Backstage developer portals are powered by a central service catalog with centralized services and streamlined development.

Stefan Alund and Frederick Weston joined the show to explain how Backstage works and their role in developing it.

If you want to support Software Engineering Daily in a greater capacity, go to softwaredaily.com and become a paid subscriber. You can get all of our episodes without advertisements.

[INTERVIEW]

[00:00:43] JM: Stefan, welcome to the show.

[00:00:45] SA: Thank you very much. Nice being here.

[00:00:48] JM: It's nice to have you. So at a large company like a Spotify, there is a problem of service discovery. And one of the ways that service discovery is solved is through a service catalog. Can you explain what a service catalog is?

[00:01:05] SA: Yeah, sure. So service catalog is at least the definition that we have internally at Spotify is where you basically keep track of all the different software that is getting developed in your company. We actually look beyond kind of service catalog and track all of our software like libraries, data pipelines, machine learning models, website, mobile features, all of those in a central repository. What we have done on top of that central repositories is kind of built what we call a developer portal on top of that, consistent data source of like our software. And a developer portal is kind of you can look at it as like a single pane of glass essentially towards your whole infrastructure and software development platform.

[00:01:57] JM: And why is a developer portal useful?

[00:02:00] SA: So there's a couple of reasons. First of all like we believe in small autonomous teams that can build and own their software end-to-end, and ideally we kind of want those engineers to be able to as quickly as possible sort of build their product end-to-end and spend as little time as possible kind of thinking about the infrastructure and how to do things and how to find information and all those things. So backstage kind of helps those teams with a standardized way of creating software. And so Backstage is kind of the starting point for all software creation processes.

And the software that you create is also then put into this central repository, which makes it possible for the other teams like that are together building a bigger product such as Spotify. Those other teams can find what software has already been built, what APIs are available, what technical documentation there is, etc. So it kind of allows teams to work independently but still like reuse same tools and same technologies.

[00:03:16] JM: Can you give a specific example of how backstage has been used?

[00:03:21] SA: Yeah, sure. So I think the best way of kind of explaining how an engineer uses Backstage at Spotify is whenever you join as a new engineer, for example, what we have is something called golden paths. So if you start as a backend engineer or a data engineer or machine learning engineer, what you do is you go through kind of a tutorial on how to build stuff at Spotify. So in that tutorial you kind of go through a number of different steps, and most of those steps are like increasingly different things you do in backstage. So as an example when I start the development process, instead of going and creating a repository and starting writing code and trying to figure out what tools to use etc., we have this concept of template, software templates, and those software templates kind of creates a standardized approach to how we build software at Spotify.

So rather than trying to figure out what frameworks to use, I go into Backstage and then I'd select the golden template as we call it or like the golden path template. And after filling in some information, what I get is kind of a Hello World application running in production. And everything is set up for me, like the CI is setup automatically. It starts to build automatically. I get documentation. We use a docs like code approach at Spotify. So you get like default documentation together with your software. You get like you know Kubernetes configuration files. All those things are kind of created for you.

And increasingly the engineer can spend – They can quickly get the standardized approach to building software. The Spotify way if you will, and they can quickly start to just build the feature or build the business logic that they wanted to build rather than trying to figure out how do I use all these different infrastructure tools and services that are growing as we are growing the company.

[00:05:30] JM: What was your motivation for creating Backstage?

[00:05:34] SA: So the initial motivation was it actually started as a service catalog. Spotify adopted a microservices architecture very early I would say. It's almost like 10 years ago now. And we also adopted like this model of operation ownership in teams inside the teams, which is common DevOps practices to these days. But what we realized then or what they realized, I was actually not there at the point that time, but what they realized was that – So you have all of these microservices and they depend on each other of course. When one service goes down it has ripple effects across like the system, right? Common problem.

So the first use case of backstage was actually kind of a Yellow Pages for like a read-only catalog of all the different services that existed in Spotify and who owned them. And then kind of on top of that read only catalog, some clever engineers started to add like integrations with various tools. So you could, for example, click a button to kind of page of duty integration so that you could wake up the team. Whose service was like on fire if they hadn't noticed? You could find their Slack channels easily, etc., etc. And it kind of continued to grow from there.

And what happened was that it kind of started to grow outside the microservices or the services domain as well. So when we spun up our data infrastructure organization, we started to build out all this infrastructure for managing data pipelines and finding datasets and to reuse, etc., and we kind of use the same principles of kind of extending that common read-only catalog. And sort of when I joined was like roughly four years ago, it had become evident that there was a lot of potential in this system like that had kind of accidentally grown into a platform. And at the same time we had like a lot of proliferation in our tooling at Spotify. So we kind of doubled down and build the backstage as we rebuilt it from the ground up to kind of be a proper platform if you will. And kind of my team was responsible for like as a central platform owner would ensure that it was easy for like other teams inside infrastructure or platform teams that wanted to add to Backstage.

So a really important sort of realization that we had when we built that platform was that we wanted not only to create kind of a consistent developer experience for our end users, the engineers at Spotify, but we also focused very much on kind of the contributor experience of Backstage. What I mean by that is that we wanted the service infrastructure teams to continue building things into Backstage. We wanted the data infrastructure organization to do it. We wanted the machine learning teams to sort of build their tools so that we could get that kind of single pane of glass that I talked about in the beginning.

And in order to get there, we focused on making it super simple to build one of these integrations, and we call them a plug-in essentially. So you could you know create a plug-in very, very easily and you could get that plug-in into the central backstage service very, very easily. And then kind of after getting approval from the central team, my team then, we kind of made sure that every team that kind of built their plugin could do that independently from us so that they were kind of set loose on the platform and could build their products on backstage in a very, very easy way.

So you asked about like what use cases. I think now it's even hard to describe all the different use cases. They're now over 140, I think it's 42 different plugins that have been built by over 60 different teams at Spotify. So Backstage is kind of this massive platform that does everything or like it's an integral part of kind of all the different software development processes that we have at Spotify.

[00:10:06] JM: So if I'm a service owner or an owner of like five or six services, explain how I'm using Backstage and how I'm integrating it into my project.

[00:10:14] SA: So first of all, the first thing you see kind of when you enter Backstage it's a list of the services that you own, and from there you can either obviously create new services in a streamlined way as I said before. But you don't create – Even though we'll create a lot of microservices, you don't do that on a daily basis at least. So what the engineer does in Backstage is kind of get an overview of all the different services that they have, and so the state of them. So there's a plug-in for looking at your CI, recent builds and CI. There's a plug-in showing tests and test flakiness for example that you have run. There's a plug-in that shows like the cost, for example, of your services in a service or operator-centric view. So you can see not only the you know the total cost of Spotify's cloud spent, because that number is not relevant for a single engineer. But an engineer can see from the lens of their service ownership, they can see, "Okay, so how much does my stuff cost?" and are there anomalies, for example, in that.

What we've also found super useful is kind of driving larger upgrades if you will or like technical migrations going from technology A to technology B, and we can use backstage to kind of leverage that surface and put information in front of engineers that, "Hey, your service is not using the latest technology. Click this button to kind of automatically create a pull request to move your system over to the preferred approach," or just give them the information they need. So yeah, that's kind of – They use Backstage to manage essentially the end-to-end delivery of their software. That's the easiest way to talk about it.

[00:12:12] JM: Tell me a little bit about the engineering of Backstage. So like how does it run? What is the runtime model for it?

[00:12:22] SA: So first of all, Backstage as a platform is pretty un-opinionated. What I mean by that is that what you can build a plug-in, what the plugin can be is kind of anything. So essentially a plug-in is a piece of JavaScript. Almost like a standalone web application that you run in a single monolithic single page application. And those plugins are built with standard UI components that we provide as a central team and also like UX guidelines, etc. So a typical engineer, a team that builds a plug-in, they build a UI frontend, a plugin, and then they own the backend part of that plugin themselves pretty much.

So for most of the plugins there is like a microservice that backs that plugin. So in that cost insights example that I gave, there's a team the team that owns the cost insights plugin in Backstage. They operate both the data pipelines that kind of crunches and summarizes the building information as well as an API, microservice API that exposes that as an API to their Backstage front. So like the single web application is built on React and using Material-UI and sort of a theme, a light theme on top of it. But on the backend side, everyone can kind of bring their own microservice, and we don't really have any strong opinions about you know those architectures.

[00:14:00] JM: So say a little bit more about what happens when an instance of Backstage gets spun up.

[00:14:07] SA: So Backstage, there's actually just one instance of Backstage. Then inside that instance there's multiple sort of plugins or different features spread out. And obviously now with making Backstage open source, we're investing quite a lot of time making sure that other companies can take and create their versions of Backstage, because kind of the beauty of Backstage is that every company has different infrastructure tooling and uses a different stack essentially.

Our kind of vision for Backstage open source is that there's this shared like common foundation, which is the service catalog or the software catalog, these templates and technical documentation. But on top of that, you should be free to kind of choose and pick and mix from the kind of plugins and the integrations that make sense for you. So let's say that you're you know using Jenkins for your builds. There's a plugin that you can use to add to your version of Backstage. And if you're using you know Grafana, there's a Grafana plugin. And if you're using Rollbar instead of Grafana for monitoring, there's a plugin for that as well. So that's kind of how we envision now other companies using Backstage. Not identical as Spotify, but kind of using the same core foundation, but then kind of building their own integrations and quite frankly kind of hooking up and building also proprietary like private plugins for the internal tooling that they have, because everyone is not using obviously like the off-the-shelf open source. There's a lot of you know homegrown infrastructure in every company, I'm sure. So Backstage allows you to kind of mix open source plugins and open source integrations with your home built, homegrown integrations as well.

[00:16:08] JM: Go into a little more detail about how a backstage plugin works.

[00:16:13] SA: So the first thing you do if you want to create a backstage plugin is that you clone the repo and then you run a command line. There's a command line interface that basically just like creates, scaffold a plug-in for you. And in that plug-in it kind of adds it to the backstage routing, the global routing. It gives you a dedicated URL on this existing single page web application. And then it kind of just gives you a default layout. This is how we envision titles and buttons and grids of data should be laid out. But then we kind of leave it up to individual plug-in developers like in their imagination and their use cases to kind of fill in that plugin with information.

And our job as kind of the platform providers or owners is to make it as simple as possible for a team to own and build that plug-in. And we usually like joke internally that like at typical engineer that builds a plugin is like a backend, hardcore backend engineer from an infrastructure team that doesn't really like writing JavaScript. So we try to like make it as easy as possible. It's not like a drag and drop easy, but still there's a lot of like existing UI components that you can kind of pick and mix from so that you don't have to spend a lot of time ideally to kind of this build UI or build JavaScript UI. Most of that should be available in the platform and you should be able to just like hook in your data essentially and sort of make those customizations.

And I think we've done a fairly good job. One good example of that is that we have something at Spotify, which is called hack week. So every year we have like a full company one week where everyone can hack on whatever they want. And we as the backstage team, we typically do a little bit of like a developer evangelism before and try to like have introductory sessions to how you build a plugin. So far, almost like every year since we have had this platform, there's at least like a handful of new plugins are getting built that week. And I think the cool thing about that is that it's often not just like the platform engineers that builds a plug-in. Actually what we see is actually our customers, the engineers who are using Backstage on a daily basis. Tow they either want to scratch their own niche or they have an idea for a plugin that would make sense for them and their team. And most of the time they can actually ship the fully working plugin within one week and kind of demonstrate that at the end of the week.

[00:19:10] JM: So just for people who might have lost track of what we're talking about, can you describe in more detail how do the services across the Spotify infrastructure or whoever is using Backstage, how do they get indexed into Backstage so that ownership of these different services is understood?

[00:19:33] SA: So what we have is a model where essentially every repository or every – Together with your code that makes up the software that you want to track. You store a small YAML file, and that YAML file is inspired by the Kubernetes custom resource definition I think it's called. Basically show – It describes, like a manifest essentially of like what is this software? What type is it? Is it a website? Is it a sort of backend service? Is it a mobile feature? Etc. And then it includes information like descriptions and tags that makes it possible to browse and search for that and find it across the system.

But I think the most important part of that is that it includes the owner, and we are pretty stringent about like ownership of software at Spotify. So any one of these components as we call them, which are a piece of software essentially, needs to be owned by a single team. And a piece of software cannot be put in production unless it's owned by a team. So that's also kind of how you – So if you want to change the metadata about a specific service or even change the owner of a service, what you do is that you open a pull request and you change that manifest file, that YAML file. And after some time, Backstage kind of goes around and crawls all of those manifest files and index them and puts them in the central service catalog. But the engineers don't go and click buttons in Backstage to change metadata. They basically – The sort of truth for what the information is in the catalog is coming from these YAML files that engineers maintain themselves.

And then there's in Backstage the backend system that kind of scrapes and picks up all that information, we also add you know a bunch of like runtime information from other subsystems. So kind of injecting additional metadata from various subsystems. One example could be that we have something called test certification at Spotify where basically we have like a list of requirements that you want to fulfill to reach a certain level of certification. So if you want to show kind of to the backstage ecosystem that your service, you're following best practices when it comes to testing and the way you deploy it, etc., we have a background service that kind of checks, goes around and looks at all the services and kind of evaluates them based on these different checks.

And like that information that we collect sort of deeper down in the infrastructure is then injected back into the service catalog. So the UI can be kind of enriched with additional information. So you can get – Like engineers and their teams can get like badges and sort of gamify, certain parts like quality and accessibility, compliance and security and all those other things can be gamified. It's maybe a bad word for it, but like you at least visualized sort of the state of your software.

[00:23:01] JM: There's so much surface area that's covered by Backstage. So it's a single pane of glass for all your infrastructure. You've got metrics and monitoring data. You've got service health. You've got all kinds of dependencies and documentation and stuff in there. Is there a problem with scope creep and the ability to have all this information be displayed in a meaningful fashion?

[00:23:28] SA: Yeah. I think we've had to kind of rethink the layout in a certain number of times as the ecosystem grows or plugins and consolidating certain plugins that are doing similar things but in different plugins into a single plugin and stuff like that. But I think what we've seen is that it's really valuable to kind of make it possible to create or to integrate all of these different things that engineers needs to know about. They need to care about security. They need to care about compliance. They need to care about cost. They need to care about test quality and making sure their services is not – Their latest deploy is not breaking production and those kind of things.

And what we've really focused on is kind of making all of those different additional things that an engineer needs to care about part of the developer experience. Part of sort of your Backstage views. The alternative is that you know you have many different islands of infrastructure or islands of tools. And what we found, the bigger problem that we had at Spotify was that as we build more and more of these tools, as we needed them, essentially we started creating more and more of these islands. And therefore we kind of decided to it's better to try to bring all those islands into one big system. And then once we have all those things into one big system, we can start to create like consistency between how the tools you work and basically create the map of the tooling ecosystem or an information architecture that cannot make sense that is not spread out, etc.

Actually one of the biggest blockers that we found when we went around and talked to a lot of engineers at the company was that one of the biggest problems that they had was that they didn't even know where to start looking for information. This was like before Backstage. So different teams put their information in markdown files in their GitHub repositories. Some put them in like Google Docs and some put them in wikis and Confluence and etc. There was like a huge mix. but there was no kind of central information architecture or even a central starting point before Backstage.

So even though when you have, as we do, like 140 different plugins and growing, sure, you have a different kind of problem of like how do we make sure people find things inside Backstage. But that problem, at least what we have found is way like smaller than the other problem of having to teach and force your engineers to kind of jump between these different various tools that are like completely on – They're not built together. There's no like a red thread that goes around them.

[00:26:42] JM: What are the other difficult engineering problems you've encountered in developing Backstage?

[00:26:49] SA: One of the interesting problems that we had is that as Backstage kind of grew and grew, we started – Because it started to become really like mission critical for Spotify. Obviously like when we had like you know bigger outages, for example, and actually one time Backstage was also down in a bigger outage that we had. That created like a lot of problems for our engineers, because they are used to kind of using backstage as this single pane of glass and this single starting point for both troubleshooting, failing services, etc.

So when they didn't have access to Backstage because it was also down, that created like a bigger problem essentially. At that point, we kind of had Backstage running in a single region and we learned the hard way that that's not enough for a company of our size. So we had to kind of re-architect the way we deploy Backstage and we moved it to Kubernetes and made sure that it's running in all the different regions that Spotify uses. So that regardless of if a big part of the cloud is going down, we can still kind of manage and divert traffic to other more healthy regions, etc. So that was a pretty major effort that we had to do. And as part of that process we kind of reclassified the Backstage service as what we call a tier one. So tier one services. Other tier one services inside Spotify are the ones where kind of if this service is down, then music stops. Like these are kind of the hard services that basically breaks Spotify if they are down. And we had to reclassify Backstage as one of these tier one services, which meant that we have to be really careful about making sure that we have great uptime and additional testing procedures in place to make sure that we don't break production and those kind of things.

But on top of that I think the biggest challenge otherwise is not I think engineering. I think the bigger challenge is more or less like how do you get a company with very many small teams that are autonomous and kind of have an idea of their roadmap to buy into this common shared vision of building everything into one experience and making sure that whatever those teams, these different teams are building plugins, coordinating so that they build experiences that overlap. Or sorry, that kind of make sense for the end user but still maintains the possibility for a single team to kind of own a vertical slice of backstage if you will. Because if they cannot own a vertical slice of Backstage and kind of iterate on that themselves, the risk is that they go outside and kind of build another island of infrastructure. And that goes against the whole strategy of kind of creating one single pane of glass.

[00:30:05] JM: Tell me more about the templates for services that people can use with Backstage.

[00:30:11] SA: Sure. So this is a pretty important thing. We talked a lot about like how to balance autonomy and enable teams to work autonomously. And sometimes you think about autonomy, you think about it in a way that like every engineer should be able to be free to make any choice they want to. While we kind of allow anything, we try to kind of create standardization at Spotify. And I think this is a sort of a common thread among at least a lot of companies that I’ve talked to like since we released Backstage open sources that you want to have autonomous teams making their own decisions, but you also don't want to have anarchy and you also don't want to support like every tool in the book. And you want to try to create some consistency in the way you build software so that ownership of software, it changes hands all the time and you don't want it to be like a dreadful experience kind of inheriting a backend service from another team. Ideally, you want to standardize on kind of best practices and the way we do things in our company. And the software templates kind of solves that or at least a starting point of that problem.

So in those templates, the templates are essentially code, and you as a company can decide like when we create, let's say, microservices or when we create websites, this is the tooling that we use and this is how a kind of fresh newly created software component should look like according to our best practices. So the way we look at it, like we have in the backstage open source project a couple of like example templates. And those are just examples, because every company has their own stack that they're using. Some companies may be building Spring Boot and Java microservices and now they're using Go and now they're using a different CI system, etc. Backstage is flexible in that. It allows companies to set up their templates and kind of their golden paths as I talked about in the beginning.

And I think one of the keys to sort of making this work and making it feel like a strike a good balance between standardization and autonomy is that you really want to like invite the users in to kind of have opinions about those templates. So as I said, they're just code. And any engineer at Spotify, if they strongly disagree with kind of the software, the framework that we're using or the infrastructure that we're using, they can just open a pull request and suggest a change to that template. And if they can argue their case, good enough, and we kind of agree as a company. We merge that pull request. And then from there on, like everyone uses the updated version of the template. So that's kind of how we introduce standardization without it feeling like you're kind of tying your hands behind engineers' backs.

And on top of that, we actually also allow like teams to add their own kind of standard templates. They're not going to be like the golden versions. They're not going to be like know the recommended or fully supported templates that we and kind of the infrastructure provide. But it's fully possible for a team to kind of – If they have a technology that is not following the conventions or they need to experiment with a new technology or they have a different business need for a different mix of technologies, we don't prevent them from using Backstage. They can actually add their own template there. And we just make sure to show that this is not like the unofficially supported thing.

[00:34:22] JM: So what is Backstage itself written in?

[00:34:26] SA: Backstage is written in JavaScript or Typescript and it's uses React and it uses – For the UI layer, it uses a very popular and fantastic Material-UI open source project. And we kind of add our own – On top of sort of the existing UI components that exist there, we also provide our own or like higher level components if you will, like UX or like UI/UX patterns. So a more simplified and standardized table component and graphing components and those kind of things that most plugins need. That's on the frontend side.

And in the open source, the back end is written also in Typescript and runs on Node. And even though a plug-in can kind of use any backend they wish to sort of power their plugin, we do provide also like a simple wrapper to create a simple node-based backing service if you just want to solve course headers or if you want to just like store secrets or call out an external API and you want to proxy that in the backend. We do provide a little bit of like a simple layer to create that simplified backend. But typically an engineering team that builds a plugin also brings their own backend, which is where Backstage you're kind of un0opinionated about how you developed that.

[00:36:00] JM: So if I'm a developer and I've got a new pull request for my service, walk me through how that pull request is going to integrate with Backstage.

[00:36:10] SA: So first of all, all workflows doesn't intersect with backstage. We like to think about it in like what are kind of the main interactions or main surfaces that an engineer should care about. And first of all is you have your code editor obviously. Like you should spend hopefully the vast majority of your time like writing code and debugging code and shipping new features. And then of course when you're ready to open the pull request, we use GitHub internally, GitHub enterprise or GitLab or something similar. But for everything else, you have your code editor, you have your code revision software like GitHub or GitLab. And then like the third, that's Backstage. So for everything you need on top of kind of what you don't get from Backstage – Or sorry. What you don't get from the first two. We want all of that other things to be in Backstage. So if you want to look at, let's say, your Kubernetes deployments and how it rolls out into production after you've merged the pull request, you can do that in Backstage. Uh you can even look at like get a more detailed view of your CI setup and additional custom things that you're running. You can get that also as part of Backstage.

But let's say after you merge your pull request, Backstage is there to kind of let you follow your change into production. I think that's the kind of the best way of describing it. And sometimes for certain workflows, I mean for a typical microservice or a website, once you merge your pull request, and most teams use custom or continuous delivery so that you can kind of see the code going out into production immediately. But for certain disciplines, like let's say mobile engineering, which is also represented in Backstage, the feedback loop for your pull request after merging it is way longer, right? So for example you merge your pull request, it eventually ends up in the build that is sent to the app store or to the play store. And then we have built, or our mobile infrastructure team have built backstage plugins that lets you go in and look at kind of the part of the mobile app that you own, that you change, that your team is responsible for. And from that lens kind of look at other things related to that software. Could be like how many crashes have the features that we own in the mobile clients. How many crashes are we responsible for? And you can see those in Backstage. And similarly you can see performance metrics – Sorry. Not for your service. But for your mobile features and whatever flaky tests that you might have introduced, etc. Also, it's not always like one continuous flow going from merging all the way to production, but there's also sometimes like a longer feedback loop for those use cases.

[00:39:21] JM: What's on the roadmap for Backstage in the immediate future?

[00:39:25] SA: So we launched Backstage open source I think it's about six months ago now. And after kind of releasing this, the initial version of Backstage was very limited in what it could do. We had this like grand vision of creating a single pane of glass for all your infrastructure, but you actually had to build a lot of that yourself, because there wasn't a lot of like open source plugins existing from the beginning or actually very few. There wasn't a service catalog, etc. So people were kind of confused with, “Okay. So I like this vision and I have this problem of wanting to centralize all my tooling into one place, but this product doesn't really give me that out of the box.” And what we heard was that they really wanted primarily kind of the software catalog or the service catalog to be there in place, because that's kind of the foundation.

So after releasing that, we were extremely happy to see kind of more and more companies adopting Backstage and people started to understand more of like this is how the product should work. And when they could put that service catalog into production, start to map out their software to it, it was also more obvious like what a plugin is and how I can contribute like my tool. So we're pretty happy to see kind of now that they're starting to become kind of an ecosystem around backstage and the plugins are being built. So companies that are using different stacks than we do at Spotify, they are kind of coming up and say, “Hey, we're using Jenkins.” So we built the Jenkins plugin and then we contributed back to the project so that everyone that walks up to the project can get that for free now.

So we really – Like our main focus right now with Backstage is kind of to grow that community. Backstage is kind of – For Backstage to become the product that we wanted to become, we need to be involving a lot more other companies. Spotify is obviously going to be front and center and heavily invested in this and continue sort of putting for the foreseeable future the vast majority of the resources to bear. But still, for Backstage to succeed, we really want to create like a really vibrant and inclusive community where all the companies are using Backstage can feel like they can contribute back to it as well.

So on the immediate roadmap, we have things like API, stabilizing the API so that if you build a plugin, you don't have to fear that we're changing the APIs under the hood. Then you have to sort of constantly rebuild it. We want to provide some stability there so that people can feel confident in sort of shipping out and using Backstage in production.

We're also working, or actually one of the teams, not my team, but are the teams that are kind of running our Kubernetes deployments and our clusters. They are building a new kind of experience for simplifying and streamlining the way you interact with Kubernetes. I think that's pretty exciting actually, because the promise of Kubernetes is that you as a service owner can get a lot of stuff for free. You just basically – You build your service. You put it into the cluster and then you walk away and you can sort of trust that Kubernetes does a lot of things for you like killing malfunctioning instances and spinning up new capacity as needed, etc.

But I think actually, like in my view at least, is that Kubernetes is still very complex for a like non-infrastructure engineer. So we are trying to build like a plugin in Backstage now that kind of simplifies and takes the perspective of the service owner primarily so that you go to Backstage, you have your services, and then you go into the Kubernetes view and you see kind of only the relevant information for your service. Not like all the other stuff that is happening in the cluster or even knowledge of what a cluster is shouldn't really be there. So that's something I’m pretty excited about, like creating a great developer experience and kind of simplifying ideally a little bit to what it means to be like a user of Kubernetes. So that's on the immediate roadmap.

[00:43:55] JM: Fascinating. Have you seen much uptake from other companies using Backstage?

[00:44:02] SA: Yeah. As I said, when we released Backstage six months ago, we were really happy to see like on day one people were working in, opening pull requests and starting to contribute to the project. And as more companies started to adopt the Backstage and started to put it into production, they realize that if we want Backstage to be successful in our company, we need to have a central team that kind of runs it and evangelizes it and helps other platform teams or infrastructure teams to get on board and build those plugins so that they get this single pane of glass.

And when those companies are kind of – When they made the decision to go with Backstage, and there's like at least 20ish companies that I know of today that have like taken that decision and started to create teams to manage Backstage. What we saw was that they also started to make some really significant contributions back to the project. And even though we got a lot of contributions initially, those were typically minor things, fixing UIs, and improving the documentation, and making it easier to install and stuff like that. All fantastic things, but what we're seeing now is kind of a shift and the contributions we're getting is like way more mature.

And one of the long-term kind of hopes we had with releasing Backstage into the open was that Spotify would also start to benefit from collaborating with the other companies. Getting innovation sort of that we didn't have ourselves. And I think it's pretty incredible that we've actually already seen that. A very good example is that there was a company who they had – One of the major use cases they had for Backstage was central discovery of APIs. So you don't have to know what API you're looking for. You can go to this explore experience essentially and just find all the APIs that exist inside a company. And they built like – Added and contributed a fully working system for that that scans standard open API definitions and makes them available. And then other companies started to add on top of that, adding like GraphQL support and AsyncAPI support and GRPC API support and stuff like that. So there's now like a fully working feature feature-rich like API explorer in Backstage. And that is not something that we had internally at Spotify. So we're pretty amazed by how quickly we can also now start to leverage and sort of get back from the community already.

[00:46:58] JM: Well, Stefan, it sounds like a great place to close off. Thank you for coming on the show. It's been a real pleasure talking to you.

[00:47:02] SA: Yeah, thanks for having me. Keep up the good work. It's a great show.

[END]