EPISODE 618

[INTRODUCTION]

[0:00:00.3] JM: Shopify runs more than 500,000 small business websites. When Shopify was figuring out how to scale its infrastructure, the engineering teams did not have a standard workflow for how to deploy and manage services. Some teams used AWS. Some teams used Heroku. Some teams used other infrastructure providers. To manage all of those stores effectively, Shopify has built its own platform as a service on top of Kubernetes called Cloud-Buddy. Cloud-Buddy was inspired by Heroku and it allows engineers at Shopify to deploy services in an opinionated way that is perfect for Shopify.

Niko Kurtti is a production engineer at Shopify and he joins the show to describe Shopify's infrastructure, how they run so many stores, how they distribute those stores across their infrastructure and the motivation for building their own internal platform on top of Kubernetes.

Before we get started today, I want to announce that we're looking for a videographer. We're also looking for writers and several other roles. You can find those jobs at softwareengieneringdaily.com/jobs. If you're interested in getting involved in a lower commitment way, you can check out our open source community at github.com/softwareengineeringdaily. We've got mobile apps. We've got a website platform and we'd love to have you get involved. So you can check out those apps on the iOS App Store or the android App Store and contribute to them at github.com/softwareengineeringdaiy.

[SPONSOR MESSAGE]

[0:01:47.9] JM: This episode of Software Engineering Daily is sponsored by Datadog. With automated monitoring, distributed tracing and logging, Datadog provides deep end-to-end visibility into the health and performance of modern applications. Build rich dashboards, set alerts to identify anomalies and collaborate with your team to troubleshoot and fix issues fast.

Try it yourself by starting a free 14-day trial today. Listeners of this podcast will also receive a free Datadog t-shirt at softwareengineeringdaily.com/datadog. That softwareengineering daily.com/datadog.

[INTERVIEW]

[0:02:34.5] JM: Niko Kurtti, you're a production engineer at Shopify. Welcome to Software Engineering Daily.

[0:02:38.7] NK: Thanks.

[0:02:40.4] JM: So Shopify is an ecommerce website provider. We sell our Software Engineering Daily apparel on Shopify and it gives us a storefront. We don't have to worry about payments. We don't have to worry about inventory and things like those. It's quite a helpful application. Describe how people use Shopify.

[0:03:03.6] NK: So the users of Shopify are actually quite the different kind of group of people. So when we talk about the users of Shopify, I'm usually talking about merchants. So people that are selling online or selling on stores, like bigger model stores. In addition to offering like an ecommerce solution where you can set up your online store, we also offer hardware point of sale devices.

So, for example, this ice cream just next to my apartment and they sell t-shirts and I think they also sell ice cream online, but in additional to that, they have all point of sales devices in their bigger model store. So we try to offer the whole solution normally when you want to sell we kind of want to support you in that journey.

[0:03:47.5] JM: All of these ecommerce are in some sense sandboxed. It's not like a ecommerce aggregator, like an eBay or a walmart.com or an amazon.com. It's every store has its own segment of traffic, like a website builder type of application. Are each of these store websites running in their own application instance or are they getting served – Are multiple application instances, multiple storefronts being served from the same application instance?

[0:04:18.8] NK: Yeah. So we started and we are still doing shared compute. So we are sharing the databases. We are sharing the compute between clients. We can go a little bit deeper later on on our sharding and porting things, but in general you can think about it as a shared system.

[0:04:36.4] JM: Got it. Okay. Yeah, we'll go a little bit further. Shopify has been around for 14 years, I think. What was the infrastructure like when Shopify got started

[0:04:45.9] NK: Yeah. Shopify has been around for the longest of times, and when Shopify got started, and I don't have all the correct details, but it was basically a really scrappy startup. It started actually as an offshoot product of our CEO, Tobi. Tobi was really excited about selling snowboards back then and he was realizing that, "Hey, there's no good solution for selling snowboards online," and he was interested in programming and he was able to program. He just figured out why don't I do it myself? This was slightly after 2000, so you can imagine like that was probably the first time in recent history to get started in ecommerce because the .com bubble has just bursted. Really exciting time to get started in ecommerce.

[0:05:34.5] JM: The infrastructure was monolithic for a long time and monoliths scale, but eventually they become hard to manage and you have to break them up in some sense or scale your management solution to the monolith. How has the monolithic architecture for Shopify evolved overtime?

[0:05:58.7] NK: Yeah. So we definitely started as a monolith. Everything was done in a single application. In the past four years that I've been at Shopify, even in 2014 when I joined, we were still heavily focused on a single application. Back then we had split off our payment systems and we had split off a bunch of our data systems. In an essence, what actually goes and you being able to sell stuff online, that was all done by a single application still.

Even in 2018, that's still the case for the most parts. We still have a big monolith and we are piecing it off into other chunks, but essence, Shopify core, which is our monolith, is still a really big application and it does most of the things that goes into you being able to sell stuff.

The thing we've done in the recent years is that we've started to draw this ecosystem. We have team stores, app stores, all kinds of other services our merchants can leverage and all those applications are separate from Shopify core. It's kind of a hybrid model where the core system so to speak is still a big monolith, but all the other stuff around it are being built at separate applications. Also we are working hard on making Shopify core a little bit smaller.

[0:07:14.0] JM: This is a common practice when a company starts off with a monolithic architecture, it starts to build up services around the monolith and gradually eat away at that monolithic architecture so that maybe the common services that are used all the time, or the common methods within the monolith that are used all the time are broken up into their standalone server instances. So in the early days of breaking up the monolith and splitting off, splintering off services, what were the choices that were made? What were the services that were broken off first?

[0:07:55.8] NK: A good example is our payment systems. Because we handle credit card information, all that information has to be PCI certified and [inaudible 0:08:05.6] where you want to separate your payment systems on everything else. That was probably one of the first things that got split off from Shopify core. Today and even in 2014 actually it was running in a separate environment in ABS as a PCI thing. Then there's other stuff, like you usually want to piece off your system based on the functions.

For example, for us, there's quite clear functions that people do on Shopify. As an end user, you might be buying something, but at the same time, our end users are the merchants, and merchants spend most of the time in our admin where they can put their products online, change their prices, see how the store is doing. That seems like a natural fit for splitting off, right?

When I joined in 2014, I think we actually had just then split off the admin into Java Script version of it. We actually developed a custom Java Script framework back then which the admin used. We did quickly realize that that's not going to be a good solution, so we hit all kinds of troubles with that and we actually went back and pulled the admin back to the Shopify core, but it's also true today that those kind of major pieces are the natural fit for splitting off to their own applications.

[0:09:22.8] JM: Overtime, you've broken up more and more of this monolith into services and the practice of doing that requires some rethinking around deployment, and databases, and testing and how these different services communicate with one another. What are the standardizations that have been built up within the company around individual services being stood up?

[0:09:51.4] NK: So one of the big things we've done in the last two years or what we actually wanted to do back in 2014 but didn't quite succeed in that, that we saw that we are going to building a lot more applications rather than being focused on a handful of applications. We running on bare metal hardware back then. We were using GIF and stuff like that and every time we stood up a new application, it was assisted by our operations team. So you needed a new application, the operations team would go and set up your service in ABS and provision them with GIF and all the cookbooks [inaudible 0:10:25.2] custom for that application. So it was really artisanal and manual work that was done back then, and obviously this doesn't scale well.

So looking into, let's say, 2016, we realized that we were running in multiple environments. So we had bunch of applications in our data centers. We had bunch of applications in ABS and we had growing chunk of applications on Heroku. Heroku was a natural fit for our developers because it was so fast and so easy to get started.

So in 2016 we started thinking about, "Hey, we actually need to do something about this. We need to have a single platform that the developers can leverage and we can kind of share good patterns between users." Meaning that we've done a lot of work tuning Ruby, for example, for Shopify core but we also want all our other applications to benefit from those best practices and that turning that went into Shopify core.

[0:11:21.6] JM: So the insight that you came upon was we're running all these different services in lots of different places. We've got custom infrastructure that's being set up. We've got a lot of services in Heroku. We've got a variety of ways that this stuff has being spun up. Maybe you're using Shift scripts like you said. Obviously, there was a desire to consolidate some of those patterns and move towards a more unified way of deployment, because if you have all these individual deployment patterns, then it's problematic if engineers, for example, move from one area of the company to another and all of a sudden they're wondering why is the deployment process completely different. Why is the runtime application, the platform layer for my service completely different? It would be much easier if I was moving across the company and everything was unified. That's one of the popular aspects of the container orchestrators, is with the container orchestrator or with a platform built on a container orchestrator, it gives you a common layer assuming everybody else in the company adapts this container orchestrator.

So there was this churn, this container orchestration war over the last two or three years where different people were running different container orchestrators. What was Shopify doing around that time when there was a lot of churn and there was Docker and then there was Mesos and Amazon's container service and this variety of ways that you could deploy containers and people were saying, "Okay. We want to deploy on Docker, but we're not exactly sure what our deployment medium should be." What was it like to be inside Shopify around that time?

[0:13:10.5] NK: Yeah. It was actually – So I joined back in 2014 when Docker wasn't even 1.0 released back then. So I joined Shopify and one of my interviews back then was about using Docker and how to deploy the core containers. I remember being really excited about this because I had read about containers on Reddit and Hacker News, but I've never worked with them. It was really exciting time to be working with that problem domain because it was so fresh and so exciting.

We tried to run on Mesos and we actually run a little bit of chunk of our traffic on Mesos back in 2014. The problem back then was that all the tooling was so young and at the same time we were trying to – we had a really small team. We had only four people working on the container platform and we were trying to do multiple things at the same time.

So we were trying to move from bare metal [inaudible 0:13:58.6] that run GIF and basically deploying with Capistrano. We tried to into the Mesos world and run CoreOS and everything being stateless. We just tried to eat too much basically.

We ended pivoting back to running Ubuntu still, still provisioning with GIF, still deploying with the same tooling, but the thing we did was adapt containers. So I think we were one of the biggest users of Docker back in those days and we really reaped – I want to say we reaped majority of the benefits of containers back then in terms of being able to deploy a lot faster and our runtimes were looking a lot more identical to each other.

The problem we had was that all these work was – We were supposed to introduce like a platform back then, even in 2014. The problem was that we were aiming for Shopify core, the monolithic, and the platform we ended up with, this custom platform was that it was really highly tuned for Shopify core and it just didn't fit any of the other applications. It needed a lot of work to be able to bring to the other applications, and that's really the solution or the problem we are trying to solve with our current platform is that we need a platform that not only scales from like Shopify core, but it also scales to, "Hey, I just wrote 10 lines of Java Script. I'm going to run it somewhere. What do I do? Can I get it done in five minutes? Can I get on open internet with SSL, with DNS, with all that good stuff in five minutes?" Kind of like the same thing you can do on Heroku, for example.

[SPONSOR MESSAGE]

[0:15:42.9] JM: Support for Software Engineering Daily comes from Wunder Capital, an easy way to invest in large scale solar energy projects across the United States. With Wunder Capital, you can earn up to 7.5% annually while helping to financing renewable energy projects. Get started with Wunder Capital at wundercapital.com/sedaily. That's W-U-N-D-E-R-C-A-P-I-T-A-L.com/sedaily.

A few years ago, we did an episode of Software Engineering Daily where I interviewed somebody from Wunder Capital about how the software infrastructure works there and as well as why people would invest in solar-based projects. So if you want to learn more about Wunder Capital, you can certainly listen to that episode.

Since 2015, individuals have invested tens of millions of dollars using Wunder Capital's solar investment platform, and alongside individual investors, Wunder Capital also works with financial institutions. There is a prominent Wall Street hedge fund that has invested over $100 million with Wunder Capital. So if you're interested in investing in businesses going solar, go to wundercapital.com/sedaily. That's W-U-N-D-E-R, wunder with a U, wundercapital.com/sedaily.

[INTERVIEW CONTINUED]

[0:17:19.9] JM: Taking a step back from the infrastructure conversation for a moment, as you've said, you were trying to bite off more than you could chew as an organization. You were trying to do too much. We weren't exactly sure should we do X? Should we do Y? Well, let's try X and Y at the same time. This is a problem for engineers. It's a problem for engineering organizations. I see it time and time again. It's a problem with my own personal life. I try to do too much and it seems to be a product of being an engineer and being excited about things and seeing all these different technologies and all these different directions that you could go in and kind of trying to hedge your bets and going in multiple directions at once, and it always seems to go wrong in retrospect. The more you try to do, unfortunately, the less well you do it.

Maybe it's obvious to people who have been in the business for a while, but I think people make that mistake time and time again. So more abstractly, what did you learn as an engineer going through that process where maybe Shopify as a company was trying to bite off more than it could chew in terms of an infrastructure change internally?

[0:18:31.1] NK: Yeah. So the thing that really resonated to me about that whole thing was that you should never try to do a big change at once. So we wanted to do CoreOS. We wanted to do Mesos. We should have done either CoreOS first or we should have done Mesos first and then figure out the rest, right? Or what we ended up doing with just the containers is that we should have first introduced container platform into our ecosystem, get people familiar with it and then think about the scheduling problem of it.

Running CoreOS, for example, in hindsight being 20-20, that had really little to do with the overall problem we are trying to solve. CoreOS was more like, "Hey, this is really cool and this is what we want to do. Why not do it at the same time while we are breaking everything else kind of thing?"

In a sense, it's more about being able to shift something, get benefit from something rather than trying to do a whole big project over the span of multiple months, especially with companies like ours. You aim for a moving target, right? You start with something. You have a clear goal in mind. You want to aim for this, this and that, but six months down the road, those three things might be totally different. So you are aiming for a moving target all the time.

[0:19:46.0] JM: When did you eventually decide to move to Kubernetes as an organization?

[0:19:51.3] NK: So we were looking into multiple orchestration platforms in early 2016/late 2015. I personally looked into Terraform's Nomad, which was really young back then. We had hack days project. So we have hack days that last two days every quarter, and we had a hack days project on Kubernetes where we wanted to see if we can run Shopify core on Kubernetes with two days worth of work basically.

In addition to that, we were looking at Mesos again, Docker Swarm, and out of these solutions we just felt like Kubernetes was probably the best solution for us, basically. There was multiple reasons that that. The major point about that was that we were wanting something hosted. So we didn't want to spent a lot of time figuring out how to run the container platform. We were also excited about moving majority of our workload into the cloud. Google offering a hosted solution was a big thing back then. Also, Kubernetes was getting a lot of traction even in 2016. So we knew that this is probably going to be, if not necessarily the winner of the container wars, this is probably going to be one of the major contenders at least.

[0:20:58.7] JM: Your strategy for moving to Kubernetes eventually became to move Kubernetes, to use a hosted Kubernetes provider, but to also build a platform as a service on top of that offering. Can you talk about that strategy?

[0:21:13.9] NK: Yeah. The overall goal always was that we want to build a platform for the company that everybody can use. No matter if you're building the hack days applications, those 5 to 10 lines of JavaScript or if you want to run Shopify core. The platform should be able to do both, which is a really ambitious goal, right? The technology choices in this case being Kubernetes came after that. So we needed to find something to solve that problem. That's basically it.

Kubernetes was just a technology that we saw that we can extend, we can be able to pull on rather than something. We never really wanted something that you can buy off-the-shelf, but because we already knew that our use case was so complex that there's not going to be a perfect product for us. So something like, let's say, Racher or OpenShift or any other stuff like that, look really good and all that, but we knew that no matter which one of these we choose, we still need to build on top of it and, again, Kubernetes being hosted by Google and Google offering our support on it, had a major role in the fact that we ended up with Kubernetes.

[0:22:17.7] JM: Okay. So the managed Kubernetes providers, they offer a bunch of services on top of Kubernetes that make it easy to run your own Kubernetes. For people who are less familiar with Kubernetes, the whole thrust of Kubernetes is that it is not a platform as a service. It's something that's lower level than that. It's a way to schedule infrastructure and orchestrate infrastructure such that people could build higher level user interfaces on top of that and management interfaces for it, and then people have built things like Rancher or OpenShift on top of Kubernetes. But I thought that Google's hosted Kubernetes or Amazon's hosted Kubernetes or Azure Kubernetes service, all these different Kubernetes things, I thought that those were PaaS layers. Why would you need to build your additional Shopify PaaS layer on top of these hosted offerings?

[0:23:16.6] NK: Right. So all the major cloud providers, and we have to remember that back in 2016 the only cloud provider that offered Kubernetes was Google. But even later and thinking about Kubernetes today, I don't consider it to be a platform as a service. I consider it to be just the container orchestration platform. Again, thinking about the use case of, "Hey, I'm a developer. I'm doing my application development work. How do I get my workloads to be running on the cloud?"

Kubernetes plays a major role in the compute part of it. It plays a major role in maybe setting up load balances, all that kind of stuff, but you still need a lot of other stuff around that. So in our case, for example, we have a custom tool called service STP, and service STP keeps track of all our applications, what versions they are running, what kind of runtimes they do have, and it also works as the interface for the developers to provision new runtimes for their applications. So they can just click in service STP, service STP will come up with Kubernetes templates that will BR into their repo and then the deployment pipeline will then put the workloads on Kubernetes.

Especially with how we've extended Kubernetes with our Kubernetes controllers, that's really the part where the platform as a service came to life rather than running Kubernetes, which really have to extend to fit your use case. Again, you have to think about how something like Docker Swarm or Kubernetes is different from something like Heroku. We are trying to hit the Heroku level. We are not trying to be Heroku or even like replace Heroku in any way. We are trying to be more like an opinionated Heroku for our internal purposes.

Again, sharing those best practices, making it super easy for developers to get going, but also it gives a lot of power to our products and engineering team where when we have a single place to run our workloads, we can think about scaling issues in a single place and we can also think about security stuff. So when we control the images, when we control the runtime, we can be sure that all of these stuff that we run, being run in the same way, and if there's security incident, we know how to batch them all in a uniform way.

[0:25:26.7] JM: Scalability and security and micro-services challenges, these are things that every scaling organization has to deal with. What is so specific about Shopify's infrastructure that you would want to have opinionated aspects of a PaaS?

[0:25:46.5] NK: Yeah. Again, it just comes down to the fact that we are mostly a Ruby house. So most of our applications are Rails, and we have done a lot of work around – So we have multiple people working on Rails core at our company. We also have people working on Ruby core, and these are the kind of things that you have to – We have a custom Ruby version, for example. We need to have custom images and this is something you cannot really do on, let's say, Heroku, for example.

Same thing with security, right? So you have to have some sort of interface you can use for our own tooling. For example, in service STP, we periodically poll all the repos and look at their gem files and stuff like if there's vulnerable gems or anything like that basically. It's all about customization around how opinionated we want to be about how we run our applications and how the applications are being run.

[0:26:42.0] JM: It reminds me a little bit of Netflix. Netflix always gets to the point where if they need to build something custom and they have these internal platform engineering teams that are always doing things where if you're looking at the platform engineering team you could say, "You know, in 5 or 10 years, this will probably be offered as a service or will be more commodified somehow, but Netflix needs it today. So Netflix built it for themselves." It seems similar for your story. I'm sure it's true for certain other companies as well.

So I think the name of the platform as a service is Cloud-Buddy. So you built your own internal platform as a service called Cloud-Buddy. Is there an internal platform team that builds on this thing and manages it?

[0:27:24.4] NK: Yeah. We have a dedicated cloud platform team, which I'm a part of. We have about I want to say a dozen people may be right now. The size of the team has been changing over the years. I think we started with maybe two or four people or something like that. The Cloud-Buddy you referenced to, this is also about our customization and powering up the Kubernetes stuff.

Cloud-Buddy is a framework similar to, let's say, CoreOS's operators, which is just a fancy way of saying we really like writing Kubernetes controllers. The Cloud-Buddy's framework is just the opinionated framework we've used internally to writing those controllers. What those controllers do is actually really exciting. So we looked at the operations one has to do and setting up an application. So let's say you have your Kubernetes cluster and it has some deployments running on top of it.

The application is running there. All is really great. You might even have an ingress set up, but you still need stuff like, "Hey, what creates my DNS records for my service? What kind of stuff - which is SSL certificates? How do I get the database? How do I get, let's say, Memcached, Redis, whatever? All that kind of stuff is what we've built into the Cloud-Buddy.

So before we were running Kubernetes and before we were able to write these controllers, we were leveraging stuff like custom repo for our DNS records where you would have to do a [inaudible 0:28:47.2] into the repo and a script would call on DNS provider API. This is all done automatically right now. The user doesn't have to care how the DNS records gets created and our [inaudible 0:28:58.7], which is part of this Cloud-Buddy framework. [inaudible 0:29:00.7] it just looks at the ingres hosts and automatically creates those DNS records for us.

Another cool thing about this is that we can be, again, opinionated about how we want to create those DNS records. For example, if you look at some of our application services and you end up looking at the DNS records, you can see that we have multiple pointers where, let's say, docs.shopifycloud.com points to a different name where it's something like .do3.shopifycloud.com or whatever. These are the kind of stuff we can do to leverage that. We can move these applications between clusters without actually changing the origin DNS records.

We can internally be swapping around the mid-player DNS record and point that service to different places, again, without the developer even knowing about this stuff and even have to care about it.

[0:29:53.4] JM: Let's talk a little bit more about the deployment structure. Earlier we were talking about the fact that what Shopify is is a bunch of different stores, different storefronts, because you have different entrepreneurs on Shopify, different merchants that are selling things like t-shirts, or they're selling posters, or any kind of ecommerce or, like you said, the physical commerce aspect of it too. There is some notion to shared infrastructure where these different sites, where you think of like a site builder, like a WordPress or a Squarespace kind of thing. Different sites are being served off of the same infrastructure, but I know that there is some partitioning by store. So can you describe the sharding model from the top-down?

[0:30:45.5] NK: Yeah. Back in 2014, we started to see scaling issues with our data stores. Specifically, we were using MySQL back then we are still heavily using MySQL, and you can scale databases only so far. I remember when I joined Shopify, we were running on these custom Dell servers, then we had bought even like some random – So I'm not a hardware expert, but it was something like a fancy SSD drive that was in a PCA lane or like ridiculously fast I/O just because we couldn't even scale the database servers anymore if we done it.

We needed something to kind of release that pressure in that bottleneck and we ended up sharing our databases first. So you might think that it's natural that you have X amount of store so you can just shard by the shop ID. So you would have shops 1 to 10 on one database and shops 11 to 20 on another database over and so on and so on, and this is actually where we started, is that we started sharding these shops to different databases.

Later we've been taking this model even further where we not only shard the databases, but we've been sharding all the other services, like Redis and Memcached and even the compute to isolate, not only isolate performance, but also isolate failure mode. If there's an issue with certain database, for example, that will only impact certain percentage of our customers.

[0:32:14.4] JM: Okay. How has the migration from the previous infrastructure to the Kubernetes-based, your own platform infrastructure, gone for that sharded Shopify infrastructure?

[0:32:29.2] NK: The Kubernetes stuff didn't have a big impact on this. We basically modeled running Shopify core in the cloud pretty much similarly that we run it DC. One of the big things that happened at the same time that we were moving into Kubernetes the DC setup was that we used to have shared Redis and Memcached instances, and right now we are also porting those.

So that's one of the big things that happened, but otherwise it's a pretty similar model to what we have in the DC. So we have multiple clusters, Kubernetes clusters running, like the web workers and job workers and stuff like that, and then we have separate clusters running Memchached and Redis stuff and in those clusters we have specific instances of, let's say, Redis, so that Redis is responsible for certain pods, and certain pods have certain shops in them.

In this case, moving from certain servers, running certain read.ices, for example, there's certain clusters running certain read.ices the model didn't change that much. So it's been those things have luckily been quite separate.

[0:33:31.8] JM: How many shops are there across the Shopify infrastructure at this point?

[0:33:37.0] NK: I think the public numbers right now are that they have more than 600,000.

[0:33:40.8] JM: Okay. And how many shops can you run in a Kubernetes pod, like in a single Kubernetes pod? How do you divide that up?

[0:33:49.4] NK: Yeah. Sorry. So I talk about pods and I have to be specific about what ports I'm talking about. So we did unfortunate thing back in 2015 where we took the sharding thingy a little bit further and we started to pod. Pod means that it's a single instance of Shopify core that can run independently. So you would have a database. You would have Redis, Memcached. That's what we call the Shopify port, which is totally separate from a Kubernetes pod.

[0:34:17.9] JM: Oh, I see. So you have like a notion of what kind of infrastructure needs to be co-deployed with a single shop instance, and that notion probably does not map to one specific Kubernetes pod. The vocabulary is just the same.

[0:34:35.8] NK: Yeah, exactly. It's a really unfortunate collation in our vocabulary, and it's really confusing for the people joining Shopify after 2015 and even internally when we talk about this stuff. It's a really unfortunate name collation.

But going deeply into the pod stuff. A pod serves X-amount of stores. So depending on the store type – So we have customers that might sell, let's say, 100 USD worth of stuff in a year. Those are like people that are just getting started in commerce or maybe they are doing test of some sort or whatever. They are something that don't generate much traffic for us. Then there're stores like, I'm going to say Kylie Cosmetics is a good example. Kylie Cosmetics runs a bunch of flash sales, and what a flash sale is that they have a huge campaign of new products or some sort of sale at a specific time, and usually those flash sales running out of stock inside, let's say, 5 to 15 minutes. There's a huge demand for those products in those times.

I can't share the numbers of Kylie Cosmetics, but for example, a generic number I can throw you at is that we've done multiple millions of sales for a single shop within a couple of minutes. So you can imagine that the infrastructure and scaling needs for that kind of shop with those kind of flash sales is totally different from, let's say, a user that only sells a hundred box worth of stuff in a year. In TLDR, a pod might have few high volume shops or it might have thousands of shops based on the traffic that they generate for us.

[SPONSOR MESSAGE]

[0:36:20.4] JM: At Software Engineering Daily, we're always analyzing data to determine what our listeners care about, and we actually have a lot of data even though we're just a podcast. So it always reminds me that organizations with much more engineering going on have an order of magnitude more data than a podcast like Software Engineering Daily, and that's why the job of data scientist is such a good job to get.

Flatiron School is training the next generation of data scientists and helping them land jobs. Flatiron School is an outcomes-focused coding boot camp that offers transformative education in person and online. Flatiron School's data science program is a 15-week curriculum that mixes software engineering, statistical understanding and the ability to apply both skills in real-life scenarios. All of the career-changing courses include money back guarantees. If you don't get a job in six months, Flatiron School will refund your tuition and you can visit their website for details.

As a Software Engineering Daily listener, you can start learning for free at flatironschool.com/sedaily. You can get $500 of your first month of Flatiron School's online data science boot camp and you can get started with transforming your career towards data science. Go to flatironschool.com/sedaily and get $500 off your first month of their online data science course.

Thanks to Flatiron School for being a new sponsor of Software Engineering Daily.

[INTERVIEW CONTINED]

[0:38:12.5] JM: In the process of getting the infrastructure moved from the prior hosting model to Kubernetes, I think you built this custom infrastructure piece called the Shop Mover. What was the Shop Mover system?

[0:38:27.9] NK: Yeah. So we needed to move all our data from the data centers into the cloud and we were spinning up new database servers in the cloud and we needed some way of gradually moving these shops into the cloud based on how much we've kind of gained confidence in our platform and how we were doing in there. So basically we were doing a shop by shop moving process. Well, underneath it's paralyzed, so we can do multiple shops at the same time, but you can think about it as gradually moving shops.

What Shop Mover simply does, it copies the tables belonging to the specific shop from the origin database to the new database kind of similar to how do you do database migration wide. You have the same data in two places, two tables in this case, and then you have a synchronization of the same data is in multiple places. You generate the lock. You flip the new database or the cloud database in this case to be the master, and then you unlock it and allow writes again and you point all the writes into the new database.

This is kind of what we do and there's lots of fancy things going on Shop Mover. We have to do pin-lock verification and we also move any jobs that are in rescue. All that kind of stuff goes into the Shop Mover, but the big part really is just moving data from the origin database into the new database instance.

[0:39:49.3] JM: At this point we've talked a fair bit about the infrastructure. I'd like to talk about operations. So as operations have evolved in parallel to your changes in infrastructure, what has been your experience at Shopify has changed? Have you done any changes around monitoring, or logging, or incident response in parallel to this infrastructure revamp?

[0:40:17.7] NK: Yeah, absolutely. When I started at Shopify back four years ago, we were quite traditional in a way where we had a specific operations theme. I think we had maybe 12 people, then we have a couple of teams that were more software-oriented, that were running process like creating the container platform, for example. But we quite quickly figured out that this is not the model that can scale well.

A traditional operations model where you have people doing manual operations and being kind of responsible for everything is just not something that scales well with a high growth company. So I think back in 2015 already, we started this revamp into a production engineering organization where we have all these sub-teams that are responsible for their own expertise area. So we have teams like data stores that are responsible for our databases, then we have my own team, which is cloud platform, which is responsible for the cloud platform. We have teams that only deal with Elasticsearch or Kafka or ZooKeeper and all these kind of stuff. We kind of split into smaller sub-teams that have their own really specific high level ownership over a domain of operations.

[0:41:29.3] JM: What's your procedure for incidence response?

[0:41:32.7] NK: Yeah. A couple of years back we introduced – We used to have this generic on call rotation where somebody from operations was always on call, and they not only deal with the incident itself, like fixing and figuring out what's wrong, but they also dealt with the communication part of it. So setting up status pages, making sure our support organization knew what was going on, communicating to the leadership if there was something really major going on. This is really taxing on a single person, like communication is really a key part of incident management and it's also really hard to do if you are trying to fix a problem at the same time.

We split into this role of having an IMOC rotation. IMOC means incident manager on call, and incident management on call is responsible only for the communication part of it. So you are responsible for figuring out what teams needs to be informed, how do we want to communicate that externally? Do we need to pull some expertise into the incident and also making sure that we have some of incident, like a service disk corruption incident created after the fact? Also running root cause analyses after the incident is over, stuff like that.

Yeah, incident manger is more of a – It's not really a leadership role, but it's more of a coordinated role during an incident, where the incident manager is making sure that we have the correct people working on the issues and everybody in the organization is aware of what's happening and what's going on with the system.

[0:43:02.9] JM: At Shopify you have different service tiers and the tiers, tiers one through four, and you can rate services with a tier based on how crucial that tier of service is. You can make judgments around buckets of services based off of what tier that service is. So can you explain what the principle of service tiering is?

[0:43:32.4] NK: Yeah. So this also came out of the need of having some sort of system of specifying how important something is. When your company grows, most likely you will have multiple fires happening at the same time. So if there's an incident happening with, let's say, tier one and their three, for example, obviously you would prioritize working on tier one. It's also good indication of if we talk and these applications and somebody is not familiar with the domain of the application or if we talk with names, somebody might even not know what the application does. So it's easy to say that this is tier two service or this is tier four service, which keeps the other person a good idea of how critical this service is for the ability for us being able to sell stuff basically.

So back in 2015, I think we also introduced this tiering system, and the original idea was that we also need to have certain classifications of requirements for this applications. For example, if you're building a really critical application that has some major part in our overall product, you also have to figure out on-call rotations, how you monitor your service, how you deal with incident response. All that kind of stuff goes into the classification of your application, and it's also a good indication of how much help you might need from a production engineering organization.

Usually, the higher tier, the higher volume also. So if you have scaling issues or if you need something special to be done, we have some applications that special like, let's say, PCI requirements or stuff like that. It also has something to do with how much help you're going to get with that and how critical your service becomes.

[0:45:13.8] JM: Can you give an example of a tier one and a tier four service?

[0:45:17.1] NK: Yeah. Tier one service would be obviously our Shopify core monolith, which is basically Shopify core is down. Due to the porting fact, Shopify core is fully down, but let's say Shopify core would be fully down, you wouldn't be able to even see the products. You wouldn't be able to buy anything. So basically you wouldn't be making money anymore, and more importantly, our merchants wouldn't be able to make money anymore. This is a really important thing to remember that people's livelihoods are in question here. If we are down, it's a bad thing for us, but obviously it's even worse for our merchants who can't even sell anything.

Their four service on the other hand would be something like a hack days project. So we are actually in the middle of hack days right now and one of the applications I was just looking at is called Purify.

Purify, it seems to be a Ruby on Rails applications that hooks into our peer tabs at the office, and it should show how much peer tabs and what choices do we have on the peer tabs. That would obviously be a tier four service. If it's down for a week, nobody really cares, especially our merchants don't care. But something in tier one or tier two being down for a week, basically we would be in a lot of trouble.

[0:46:27.1] JM: So we got connected because you're going to be giving this talk at CubeCon New York. In the near future you're going to be talking about Shopify's infrastructure and you gave me a bit of a preview for the stuff you're going to talk about at CubeCon, and one of the things that I was curious about that I know you're going to explore in that talk is operational practices that an organization needs to avoid if it wants to scale. There are plenty of difficulties operationally, organizationally that we could explore as we're talking about scalability. What specifically are you going to explore in that talk? What are the lessons through your four years at Shopify that you've encountered that inhibit an organization from scaling?

[0:47:18.7] NK: Again, when I joined Shopify we were mostly focused on a single monolithic application. We had a handful of other applications, but majority of went into Shopify core. Overtime we've moved into this model of where we have hundreds and hundreds of publications. I don't think there's an application coming up every day, but every week for sure. So something that would scale for a handful of applications is like somebody could manually create DNS records. Somebody could manually create databases for you. That scales fairly well if you're creating applications like once a month or not even that often. But if it happens all the time and it has to be reliable, you need to figure out automation.

I think that's the main takeaway from my presentation at CubeCon is going to be how certain things just doesn't scale anymore when you go into high volume a month. So manually operations in a traditional operation environment, creating issues for people to deal with, just stuff that doesn't scale well with high volume applications.

[0:48:22.1] JM: That sounds very much like the SRE mentality.

[0:48:25.4] NK: Yeah, absolutely. We use the term production engineer ourselves and we've definitely have read the SRE book and we've really liked ideas from it, but we are not copying Google's model one to one. So our products and engineering role is pretty close to the SRE role, but we definitely aren't trying to monkey it off.

[0:48:43.1] JM: Shopify moved to Kubernetes. It built a platform as a service, Cloud-Buddy, and this platform as a service lets people write custom Kubernetes controllers. What is being worked on in that PaaS today? What are doing moving forward?

[0:49:01.4] NK: Lots of work have gone into polishing a bunch of the work we've done in the past two years. So, again, about the moving target problem. Cloud platform was doing majority of work for everybody possible thing you can imagine. So not only were we spinning up the Kubernetes clusters and figuring out good storage of that, we were also dealing with stuff like how does user get in database for example?

We were writing controllers, Kubernetes controllers to provision those automatically. We created controllers to create your DNS records, your SSL records. Today we are trying to handoff multiple of these things into the shop teams that we have. For example, cloud SQL being MySQL underneath belong to our data store teams. SSL, DNS would belong to our traffic team and so on.

Another big thing for us is the custom tooling that we have, for example, we have a tool called cloud portal, and a cloud portal is basically a web UI that our developers can use to look into what's going on into your production infrastructure. So we don't give Kubernetes access to our users and we don't expect developers to know anything about Kubernetes, but cloud portal is the way of going in and figuring out what's running, why isn't something running? They can also use the tool, for example, get in terminal console into the products and stuff like that.

Those abstractions and those automations, we are still trying to nail down. And big part of that is actually our deployment storage. We started with this crappy solution of putting the Kubernetes manifest into people's repos. During a deploy, we just deploy those templates, which are grainular Kubernetes templates. These has become a little bit of a bottleneck where the Kubernetes templates aren't that human readable for people without Kubernetes context, and sometimes we need to change something across all the applications. So far, we've been using this mass BR tool where we create yaml chains and then we mass BR it to all the repos and we try to make sure that people actually merge those BRs. But obviously this is not something that scales really well. We are really thinking hard about how do we want to actually specify the runtime for people's applications and create the correct level of abstraction for it.

Again, we don't want to build another Heroku. We don't want to make it too generic. We want to be opinionated about the needs of Shopify and we definitely don't want to height the internals from our developers, because developers are obviously smart people and they might want to do something really special that fits their use case well. So we tried to make it so that it's kind of a paved road, but if you want to dig deeper, if you can dig deeper, and that's really the problem we are having right now, is figuring out a good balance of hiding details, but at the same time leveraging the possibility of users being able to go into detail and go into the nasty bits of how internals work.

[0:52:00.3] JM: Niko Kurtti, thank you for coming on Software Engineering Daily. It's been great talking to you.

[0:52:03.4] NK: Thank you so much.

[END OF INTERVIEW]

[0:52:08.7] JM: GoCD is a continuous delivery tool created by ThoughtWorks. It's open source and free to use, and GoCD has all the features you need for continuous delivery. Model your deployment pipelines without installing any plug-ins. Use the value stream map to visualize your end-to-end workflow, and if you use Kubernetes, GoCD is a natural fit to add continuous delivery to your project.

With GoCD running on Kubernetes, you define your build workflow and let GoCD provision and scale your infrastructure on-the-fly. GoCD agents use Kubernetes to scale as needed. Check out gocd.org/sedaily and learn about how you can get started. GoCD was built with the learnings of the ThoughtWorks engineering team who have talked about building the product in previous episodes of Software Engineering Daily, and it's great to see the continued progress on GoCD with the new Kubernetes integrations. You can check it out for yourself at gocd.org/sedaily.

Thank you so much to ThoughtWorks for being a longtime sponsor of Software Engineering Daily. We are proud to have ThoughtWorks and GoCD as sponsors of the show.

[END]