**EPISODE 1234**

[INTRODUCTION]

**[00:00:00] JM:** A major change in the software industry is the expectation of automation. The infrastructure for deploying code, hosting it and monitoring it is now being fully viewed as automatable. Equinix Metal takes the bare metal servers that you would see in data centers and fits them with supreme automation and repeatability. This movement to modern metal as a service has brought with it specialty hardware configurations and options for customizability for cloud native applications. Equinix has one of the largest collections of interconnected data centers in the world. Their existing network capabilities bring an interconnectivity to its bare metal product that enables global scale. Their server plans are high-performing, secure, agile and have flexible pricing and they're also customizable to fit unique use cases designed for a DevOps experience that's pretty automated.

Nicole Hubbard is a principal engineer in the Equinix Metal delivery team and she was involved in building a world-class control panel for Kubernetes. She's been closely involved in the success of Equinix Metal and joins the show to tell us about it.

[INTERVIEW]

**[00:01:07] JM:** Nicole, welcome to the show.

**[00:01:08] NH:** Thanks for having me.

**[00:01:10] JM:** It's 2021, let's say I' want to spin up a container or a cluster of containers like on a Kubernetes cluster to run my application. What goes on under the hood on the cloud provider when I' m provisioning that infrastructure?

**[00:01:27] NH:** Usually there's a lot of things that have to happen including provisioning your base operating system. So laying down either Ubuntu or a container-optimized operating system such as Flatcar Linux or Google's Container OS that they provide. From there, once that's installed, you have to also bootstrap the Kubernetes cluster itself, which includes standing

up all of the container or all of the Kubernetes components, so the scheduler, the API servers, the controllers, and then you have to add all the nodes into the cluster. And once you've completed all of that, you're then able to actually talk to the API and schedule those containers.

**[00:02:17] JM:** Across that set of things that needs to be accomplished to provision that infrastructure, what's the long leg of latency or what are the long legs of latency? What are the places where you can encounter problematic latency?

**[00:02:34] NH:** It depends a lot on where you're deploying. If you're deploying on bare metal servers, you're going to generally see the longest leg being installing the operating system. If you're running on top of a cloud provider where they're just spinning up VMs based on images, generally you get your VMs pretty quick. And so it just kind of depends on your environment, but usually it's either OS installation or getting all of the networking pieces set up, so getting IPs and getting those routing and everything along those lines.

**[00:03:09] JM:** York at Equinix, and how would you differentiate Equinix from the other cloud providers that are out there like the AWS's and the Microsoft's of the world?

**[00:03:23] NH:** So one of the big things that we offer is all of our gear is physical servers. So when you're buying hardware from us, you're buying a physical server that we're provisioning for you. Those can be anything from small servers with 32 gigs of RAM and a couple of cores all the way up to some of our more massive servers that are over 256 gigs of RAM with I don't even know how many cores, but an insane number it seems. The advantage you get with this is you don't ever have to worry about some of the noisy neighbor problems. You don't ever have a node where for some reason one of your instances in your cluster is operating at half of the capacity of your other servers that are the exact same because you're the only tenant on that node.

**[00:04:14] JM:** And how does that difference between other cloud providers yield like a difference in the customer base? Like how does the customer base differ due to that difference in service?

**[00:04:27] NH:** Yeah, because of the infrastructure that you're deploying with us, generally a lot of the smaller customers that need just some of the really small cloud servers you can get aren't generally the first customers that we end up with. We end up with some of the larger customers, but this end customers that have more advanced requirements than you're able to get out of some of the cloud providers. For example, we offer BGP with all of our servers so you can actually announce out IP addresses from your nodes. You can announce your own IPs if you have them and you're able to do a lot of AnyCast routing and things like that that you can't do as easily with cloud providers.

**[00:05:11] JM:** So Equinix Metal is the product that you specifically work on, and it's my understanding that that's basically provisioning infrastructure closer to the metal. So when you're provisioning infrastructure, you're not dealing with the VM layer. Is that correct? Is that the main differentiation?

**[00:05:31] NH:** Yep, that's correct.

**[00:05:33] JM:** So when you don't have the VM layer to orchestrate with, is there anything about that that makes it more complicated or makes it harder to deal with?

**[00:05:47] NH:** There are some things, yes. A couple that come to mind, one of them, if you think about some of your cloud providers, they provide the ability to take backups of your OS disk so you can snapshot them. When you're dealing with physical hard drives, that's a lot harder to do. So some of those capabilities aren't there. You can't just easily clone one of your servers into another VM like you can, but you're still able to provision and then install everything to it, but you can't do things like packer and provide a custom image that you want laid down on the machine at provisioning.

**[00:06:29] JM:** You've worked at other places that have had a lot of virtualization in place. Can you tell me any other differences between working at an infrastructure stack that has virtualization in the mix versus one that just uses bare metal?

**[00:06:45] NH:** I would say one of one of the biggest differences is not having the overhead that you get with virtualization, but that comes at the cost like I was mentioning. You lose some of

the advantages for being able to clone a machine, but you gain being able to run more workloads closer to the metal. You gain the ability if you need GPUs without having to worry about any of the virtualization layers on those. And I think those are some of the first ones that come to mind.

Another major one that I've had issues with personally dealing with virtualization in my experience is networking can be a challenge once you add the virtualization layer into it and dealing with physical hardware relieves a lot of those problems that you have at the networking layer.

**[00:07:41] JM:** Gotcha. So as principal engineer, give me an overview of some of the hard technical problems that you've worked on.

**[00:07:49] NH:** Yeah. So one of the things that my team is responsible for is actually providing Kubernetes clusters and all of our facilities that we use to run all of our stack on inside of that facility. So all the stuff that handles provisioning and all of those tools. My team runs the hardware that that runs on. So one of the hardest things we had to solve when we started out was how do we provision our hardware when the entire provisioning stack that provisions hardware for us in a facility runs on our cluster? And so we had to go and build our own provisioning stack that could work globally across all of our facilities and provide that same functionality and it's one of those times where when we finished with it I' m like, "I remember why customers like having this done for them."

**[00:08:44] JM:** So Equinix Metal was part of an acquisition. It was the acquisition of Packet. Have you worked on the refactoring or the merger of the two companies?

**[00:08:55] NH:** I' have a little, yes.

**[00:08:57] JM:** And what's that been like?

**[00:08:58] NH:** It's been good. For the most part, the Packet organization is what is now the Equinix Metal organization. There's still a lot of autonomy. Everything still feels very much the same. I joined uh shortly after the acquisition, but before the merger it really happened. And so

I've seen the merger happen and gone through that and it's actually been one of the smoother ones I've ever seen, and that's having largely been on the big side of the company buying the smaller company. I've seen it be rough. So I can only imagine how much worse it was on the smaller side. So this time being on the smaller side it's actually been the smoothest of any I've seen.

**[00:09:42] JM:** And your work today, does it typically focus on like the frontend? Like leaning towards the front end like the provisioning side of things? Or like the backend, the actual management on the infrastructure provider side of like all of the different clusters that are running across the infrastructure?

**[00:10:01] NH:** It largely deals with the backend side of things. So we're building our teams, working and building out a lot of automation and a lot of software that helps build that automation so that the things we have to do today manually we have software that does it for us. It's very much an infrastructure as code type of thing except for building controllers that actually do that for us, and that's what I' focus on mostly.

**[00:10:33] JM:** When you say building a controller, what do you mean?

**[00:10:37] NH:** So in our case it's specifically around basically kind of a control loop that runs and tries to reconcile state. So we're working to move towards being able to set a desired state and then having software that runs that will get us to that state.

**[00:10:53] JM:** And give an example of how the infrastructure would fall out of that state and how it would be ameliorated.

**[00:11:02] NH:** Yeah. So one example is with all of the clusters we run, two of the main things we have to manage in them is both the operating system version that we're running as well as the Kubernetes version running on top of those. So we have software we're building right now that allows us to specify what version we want those two components to be at as well as other components within the cluster. And then that software will go and check the operating system running on that local node. And if it's not what we want cue up and update, get that update ready to go, and then get it applied. And then we're also working on the automation to upgrade

Kubernetes so that we can do the same type of thing there where when a new version is released we can just update it for that cluster and we have software that will automatically go and update those.

**[00:12:00] JM:** Now tell me more about what goes into creating one of those controllers.

**[00:12:05] NH:** A lot of testing and a lot of failures. So especially when you're dealing with things like upgrading Kubernetes, there are a lot of cases where going from one version to another you're going to have things that get deprecated. And so if we just – I believe it's either the upcoming 1.21 or upcoming 1.22 release, the support for like the beta ingress object inside of Kubernetes is going to go away because it's been moved to stable. There's still some charts that use that for when you deploy things into the cluster. And so we've been going through and getting those upgraded. But when we do the upgrade, if everything's not already moved, that can cause an outage because those will just go away, or the upgrade will fail because it's like you have these objects that you can't have anymore. And so having to write checks into our automation that will know to go look for those things and kind of do an audit when you're migrating to certain versions. We've got some upstream tooling that helps with that, but just incorporating all of that in.

And then with like operating system upgrades, when we do decide to roll those out, we want to roll them out slowly to start to ensure there're no failures and that we don't start seeing anything weird. And so we have software that will slowly roll that out to a couple nodes first. And then over time it will roll them out to a larger pool of nodes a lot quicker once we've validated it.

**[00:13:48] JM:** What kinds of bugs have you found in that vast testing process?

**[00:13:54] NH:** Largely they've been around missing things that we should test for that we didn't think about because like when you sit down and brainstorm like, “What could go wrong?” You're only going to come up with maybe half the things that could actually go wrong. We had an operating system release that came out on us shortly after we started using this new operating system we run on now. And in that release systemd was updated, which through the release notes seemed pretty minor, is a fairly minor version bump. I think it was only like two versions

up, and nothing in the release notes like stuck out as like, “This is going to break everything for us.”

And then we rolled it out and the nodes that it went on, like the entire networking stack just quit working. And so we're like, “Oh! That's cool. Somehow our automation didn't catch it,” and it still kept rolling because everything was still working except for that one node. So Kubernetes was just not scheduling to it. So we had to re-tweak some of our ways we monitor that.

**[00:15:10] JM:** How does your testing infrastructure run relative to updates to your infrastructure? Like do you have a continuous delivery process for the backend tooling, for the backend infrastructure?

**[00:15:25] NH:** That is something we are still kind of working on right now. We have different environments. So we have a staging environment. We have production environments. Generally, updates happen in staging first and then go into production after. Another interesting scenario that we have is, as a company, we're constantly bringing online new facilities. And so when a new facility is coming up, we're standing up a whole new cluster. And so this gives us a clean environment that we can start with new versions of things. So we'll upgrade things into a new facility, ensure all of the software for that facility comes online is working, customers aren't having any issues. And then we can go back and we can start upgrading it in the existing facilities. So it's not the continuous delivery that we're hoping to get to eventually, but it's the process we found that allows us to test things with minimal risk and be able to then retrofit and deploy out in other places.

**[00:16:31] JM:** So you mentioned you're constantly bringing new facilities online, and people who are not familiar with how Equinix works and I guess the concept of colos, might not be familiar with what that means, can you explain what a colo is and how that affects your work?

**[00:16:49] NH:** Yeah. So Equinix as a whole has over 200 data centers, we call them IBX's, worldwide. We're constantly bringing online new facilities, and inside of there we sell space to customers for them to bring their own gear in and run their own servers. Inside of Equinix Metal, we are kind of a customer of that in that we have our own hardware that we end up bringing in deploying, setting up and then selling to customers for them to use. And so we are in a subset of

those facilities, but as Equinix brings online new facilities and Equinix Metal goes into those facilities, we're having to provision and build everything for those.

And then with colo specifically, usually what you're paying for is power and space inside of the data center on the raised flooring, which includes the cooling and everything else you need inside of a data center. I' haven't dealt a ton with all the details of how we sell that. So I' don't know if networking and connection is included with that or if you have to get your own transit and things like that, but those are other things you also have to think about with colo.

**[00:18:15] JM:** So tell me more about the onboarding process for a new colo. So a new colo gets stood. How do you get the infrastructure deployed and provisioned and tested and checked and so?

**[00:18:26] NH:** Yeah. So when Equinix metal goes into a new facility, tons of hardware is ordered. It all gets shipped to the facility. It all gets installed by the techs that work at that facility. One of the first things that has to happen once it's installed is networking comes in and they configure all of our backbone routing and connect it to our network. From there, one of the next – Usually the absolute next thing that happens is our servers that are what we call the control servers get handed over to us and we provision all of our software onto those as well as work with the teams that are responsible for all the software that actually manages the data center. From there, once we have our entire controllers stack up, we'll hand it off to another team, which goes through, and they take all the other servers that have been provisioned and they start doing burn-in testing. They then enroll those servers. And then once the facility goes live, we're able to start selling those servers to customers.

**[00:19:37] JM:** What happens during a scale-out or a scale-up of customer infrastructure? Like when there's some app that's getting a lot of demand and the user scales it up, what happens on the Equinix side?

**[00:19:54] NH:** Yeah. So largely customers will make the request either through our API through automation that they have or through our user portal through the UI that we provide. From there, the request will go. It will look at the available hardware that's available and select the best nodes for that customer and then we'd provision those nodes. Depending on a couple factors,

we do some optimization to improve provisioning time. And so sometimes that can result in us handing servers over extremely quickly because we already have servers somewhat provisioned already with a base OS laid down to them and then we configured the OS and the networking and everything. And sometimes we have to run through the full stack of rebooting servers, booting them over the network to install the operating system to drive. And then once the OS starts, configuring the OS and then handing them over to customers. But from there, once that's complete, it gets handed over to the customer and they're able to do anything they want with them.

**[00:21:03] JM:** So in contrast to a cloud provider like AWS that has like hundreds of different managed services, customers are deploying their own infrastructure onto Equinix with probably more of a self-management attitude in mind. How does the contour of running their own infrastructure compare to a major cloud provider that has that managed infrastructure? Are there challenges that a customer on Equinix might run into when trying to deploy their own Kafka cluster or trying to deploy their own Cassandra cluster that they might not encounter if they were running on managed services?

**[00:21:52] NH:** Yes. If you've spent enough time working with cloud providers, there are certain things that clouds provide that you just take for granted today. The big one for me is load balancers. If I need to make something accessible that's running on four servers and I' m working in a cloud environment, I can just go in and I can configure a load balancer, point it to my four servers, set up a health check, load balancer will remove and add nodes. If I scale it up, generally if I've put that in some kind of scaling group, then the load balancer can point to the scaling group and automatically get the new nodes. When you're doing this on bare metal, that's not as simple. You have to either deploy your own load balancers or you have to leverage BGP and announce the IP of your load balancing VIP if you will from numerous nodes so that all of them receive traffic, and then on the nodes do the health check to pull that announcement out of the network so that traffic's no longer routed to you and add it back when things are healthy, which inside of Kubernetes, specifically you can leverage tools like MetalLB to do this for you to get a lot of that cloud filling that you get from the cloud providers where you stand up a load balanced service and it just gets an IP and works. MetalLB can automate that and provide that same fill for you.

When dealing with things like databases, you have to start thinking about all the different things. Like if you're using a managed data store, say, from Google using Cloud SQL for a Postgres database. What are all the things they do for you? They're doing backups. They're doing replication and all of those types of things you now have to take on the responsibility for, which in a lot of cases gives you more control and more insight into what's happening and is usually a good thing as you get larger anyway, but as you're starting out can be a lot of additional work.

**[00:24:07] JM:** You've been in the Kubernetes ecosystem for a pretty long time now. Tell me about the newer tooling that you're using and how Kubernetes is changing.

**[00:24:16] NH:** Kubernetes seems to change every year a lot. I've been involved for – Oh, I' don't even remember now, five years, six years. Some of the things that I've seen recently that I' like the most are the support for end user experience and solving like some of the issues with running things inside of Kubernetes. Specifically, for example, it's been out for a while no, but stateful sets were released to support the ability to run things that needed specific requirements around state, databases, Kafka clusters, things along those lines. So you're able to say, "Give me three instances. They always have the exact same name.  You can get an IP that remains static for each one of those through the use of a service directed to each one," and that allows you to also run steps that say, "When this one comes online, ensure that the data is fully replicated before you take the next one offline to do the upgrade for it," and things along those lines.

Onother one is the tooling. Specifically one of the tools we use is Argo CD, and this provides a UI to manage all of your deployments into clusters. Argo has had some fairly major releases in their last couple releases that have dramatically improved both performance and usability for our users because they're managing hundreds of apps across multiple clusters and it allows them to filter down to just a single cluster or they can look at one app across all clusters and kind of pick how they want to drill down into that. So it's largely all the tooling that's being built to make it easier for the engineers running things in the cluster that have been the most useful improvements lately I' think for me.

**[00:26:30] JM:** Are there parts of Kubernetes that still seem too hard to operate to you?

**[00:26:36] NH:** Oh! I love this question yes. Part of me wants to say all of it given the choice, but really the stuff that gets extremely hard is when you're having to deal with state in the cluster, stateful sets makes this a lot easier, but you still have to make sure all of your tools can handle that. Specifically like if you're running a Postgres database inside of your cluster, don't just run one instance of it like you might if it was a physical server, because pods are way more likely to get rescheduled than a server is to get rebooted. And so you can end up having small windows of downtime as pods get changed or rescheduled or as nodes under them get taken out of the cluster to be rebooted or maintenance done on them and then added back. And so it's largely around having to think about the fact that at any time what I'm running inside of this container could go away. And if that happens, how do I ensure that my apps can handle that? And especially when they're state involved.

When you're running just applications that talk to a database that's running outside of the cluster, life gets a lot easier because you can just roll and restart those apps. You can run 10 copies of it or however many you need. Roll through them, update them one by one or two at a time, whatever you decide. But once you add state, things get a lot more complicated.

**[00:28:18] JM:** How was managing a cloud provider infrastructure different before Kubernetes?

**[00:28:26] NH:** So in my experiences working on the cloud, I think one of the big differences is before Kubernetes, a pattern a lot of companies took were deploying a single application per VM or per instance. You would stand the instance u, pit would pull the latest version down, or you baked images that already had the latest version. And so it would start up with that version, and that's kind of how you did your continuous deployment also was through new images or updating the version of it running on the servers. You also had to consider things like now if we go – Well, if we go all the way back before containers, you also had to consider things like the version of components running on the host operating system. Say you're deploying something written in Ruby and you need a certain version of Ruby. You have to ensure that the OS has that version, same thing with Python or Java or any other runtime.

With containers, you get away from that because all of that's bundled in. And then with Kubernetes, it provides the ability to easily run multiple things on the same servers, which wasn't the pattern that tented to happen before the move into containers and Kubernetes.

**[00:29:58] JM:** Tell me about the observability stack that you use for infrastructure on Equinix.

**[00:30:05] NH:** Yeah. So within all of our Kubernetes clusters, we leverage Prometheus. All of our software emits metrics that Prometheus is able to scrape. So we scrape those metrics. We export them into long-term storage and then we're able to query those with Grafana to be able to visualize them. We also have alerts through alert manager as part of the Prometheus stack that can send alerts for us through. We use PagerDuty. And then we also are doing some work with tracing. So starting to use some of the open telemetry tooling to actually collect traces and send those out so that we can watch requests flow through our entire stack. From the central API all the way into the sites is the ultimate goal there. And then we also have logs from all of our containers that get shipped to a centralized logging system where we're able to go and query logs to see anything we need to see.

**[00:31:17] JM:** Do you use any kind of like scalable metrics database to scale your Prometheus storage?

**[00:31:25] NH:** Currently we leverage the Grafana enterprise offering, which if I remember correctly uses Cortex on the backend to actually do that work. So we're currently using a hosted service for that because we had that in place from before. As a team, we have been researching all of our options with Prometheus and the work that's done in Cortex and Thanos to figure out which one of those we want to try to move forward with so that we can build our own solution that can scale and handle all of our metrics.

**[00:32:08] JM:** Are there any areas of the observability stack that you feel are underserved right now? In particular, when you're running a cloud provider?

**[00:32:17] NH:** I think one of the bigger black boxes that we run into is networking. There's not as much insight into what's happening at the network layer specifically once we start getting above like outside of our servers. So we have a lot of insight into what's happening with the network in our clusters. But once it leaves the switch we're connected to, it starts to get a little more muddy in terms of what we can easily visualize and get through all the data. So I would say more things around networking would be my number one wishlist item for observability.

**[00:33:03] JM:** Can you tell me about some other – I mean, you mentioned the Grafana choice. Can you tell me about some other build versus buy or just SaaS selection processes that might reveal some interesting decisions in the stack?

**[00:33:20] NH:** Yeah. It's one of those we tend to always take the look of, “Is this one of our core competencies? Is this something we need to host ourself or not?” For example, with all the tracing stuff we're doing, we're evaluating Honeycomb as an option for tracing as well as we could leverage something like Jaeger and run it ourself, but we're evaluating Honeycomb currently. We leverage tools like Datadog to get some metrics as well largely due to it was in place previously, but we still leverage it now even in our new stack because we had a lot of tooling built around it. So even though we have a lot of that same details, rather than trying to go and build to get the details we don't have in our current stack, we just let Datadog continue to handle that for us.

Another example is logging. Instead of trying to maintain our own logging cluster and running, for example, Elasticsearch, we actually outsource that to a provider so that they can manage all of that for us because it's not something – We need to be able to access it. We need it to be reliable. The vendor we're using is able to do that for us. And so we don't have a need to run that ourself if we don't have to.

**[00:34:53] JM:** Do you have any good debugging stories or like war stories that you can tell from your time at Equinix thus far?

**[00:35:02] NH:** I’ think one of the more interesting ones was like I talked about earlier. We had an operating system upgrade come out that upgraded systemd and it took offline all of our container networking inside of Kubernetes. And it was one of those where when it happened it was interesting because we had been testing a new stack or a new facility and we stood up our controller, which brought all the Kubernetes cluster online. Everything was up, everything was good. We were doing a lot of testing of our provisioning because this is when we built a new provisioning stack. And so we still had a few days to get this cluster turned over. And so we decided to reprovision it a couple times.

And in one of those, we upgraded the operating system. We reprovisioned, everything came up. Cluster was online. We thought everything was healthy and then we went to reprovision it again just to validate the provisioning stack and a few changes we had made that were completely unrelated to anything to do with networking, and comes online, and we're getting close to the time to actually hand this cluster over so that the other teams can start deploying all of their software into it. So we hand it over. And the next thing we realized, like none of the containers running in it have working networking and nothing can be accessed that's running in the cluster. All of the nodes are fine. All of the software that runs through like host networking on those nodes is working, so the API server for Kubernetes, the scheduler, all of that's fine. And it's like what happened all of a sudden? And it took us digging through GitHub releases and all these other things before we found it and turns out like a very minor change in one of the sys control network settings completely broke networking for us and it changed in the systemd version. And so some of the operating systems were overwriting this change in their released version, and so we didn't see it on all of our existing nodes or all of our older nodes that we're running like Ubuntu, for example, and we're like, “We've got the same stuff over here. It works over here, but now all of a sudden over here it's broken.” So like just kind of going down that rabbit hole, it took us almost half a day for two or three of us going down the rabbit hole of figuring out what broke before we realized, “Oh! This sys control has changed.” And so we just had to set it and then update our provisioning to set that for us by default so that it wouldn't completely break things. We reported the issue to our OS provider and they were able to go through and they actually updated it in a future OS release so that it does automatically set that same as some of the other operating systems. So that one was interesting.

One of the harder ones is when we were standing up a new facility, we actually had – I was doing this stand up for this facility in particular and we use the same two models of hardware everywhere we deploy things. And in these, they're Dell servers, and we configure a hardware RAID one for our operating system drives. For whatever reason, on these servers, whenever we would go to install the operating system, the RAID drive just wasn't there. It would show as two separate hard drives. So in completely ignoring the RAID one which shows the individual drives, it would install – The automation installed it to the individual drive because it thought that was the RAID volume. And then on reboot, the RAID controller starts like the RAID’s out of sync. Everything's broken. Nothing can boot. And we're like, “What is going on?”

And in this case we had gotten to help speed up the time it took us to bring the site online. We actually shipped our nodes from another facility. They pulled nodes from another facility for us and shipped onto this one so that we wouldn't have to wait for Dell to get us new servers. But it turns out these nodes had an older – Some older firmware on them. And so the RAID controllers needed their firmware upgraded to work with the newer version of the Linux kernel we were deploying, or else it was reporting drives as individual drives instead of a RAID drive. And so that one was not a lot of fun to track down and took quite a while to figure out like, “What is going on?” It's like, “These are the same.” And like when you've dealt with just cloud servers, you kind of forget that firmware is a thing you ever have to think of. So just took a lot longer than it should have, because trying to get used to having to deal with some of the things you got to deal with for bare metal devices when you're deploying them yourself and managing them yourself like my team does.

**[00:40:59] JM:** Great anecdotes. Do you have a vision for how cloud infrastructure changes in the next five to ten years?

**[00:41:10] NH:** So bear with me as I answer this, because it's not as much of a hot take as it might seem from the start. I’d like to see Kubernetes go away in the sense that users don't interact with Kubernetes directly. Usually Kubernetes, people are deploying deployments or stateful sets into it and they're having to deal with things like ingress and all of these other high-level components. And you can already kind of see work happening with this. OpenShift that Red Hat does is kind of starting to address this a little. Some of the tooling that Microsoft has built through their open source teams, which the name of it is escaping me right this second, but it starts to abstract away the Kubernetes components and allow you to deal with your application’s components. So instead of having to think about like services and ingresses and stateful sets and deployments and all of these other Kubernetes primitives, you can just think about, “I have this application. It has this container. I need three of them running this much memory and I need it to be accessible through this web address,” and that can abstract that into all of those Kubernetes pieces for you. So kind of like I don't want to say building a platform as a service on top of Kubernetes, but providing the abstraction to make it easier for developers to work with to help with getting onboarded into it and to not have to fully grok all of the primitives that Kubernetes offers especially as Kubernetes keeps growing and there's even more. And depending on things you deploy, for example, we use Cilium our container networking inside of

our clusters. And Cilium provides the ability to create Cilium network policies. Kubernetes has network policies, and it's one of those, “Which one should I use? What's the difference?” And as a user, most the time they don't care. They're like, “We need these things to be able to access it,” and that should be enough for it to get abstracted into the correct things inside of Kubernetes. So I’ think that kind of abstraction on top of Kubernetes is one of the things I look forward to the most, and it seems there's a lot of things happening in that space already.

**[00:43:51] JM:** Very interesting vision, and I’ guess Equinix is pretty well-positioned to capitalize on that future.

**[00:43:59] NH:** Yeah.

**[00:44:01] JM:** Cool. Well, Nicole, thanks for coming on the show. It's been a real pleasure talking to you.

**[00:44:05] NH:** Yeah, you too. Thanks for having me.

[END]