EPISODE 1454

[INTRODUCTION]

[00:00:00] JM: Loft is a platform for Kubernetes self-service and multitenancy. Loft allows you to control Kubernetes clusters with added multitenancy and self-service capabilities to get more value out of Kubernetes beyond simply cluster management. It allows for cost optimization, more efficient provisioning and other features. Lukas Gentele joins the show to talk about Kubernetes multitenancy and the engineering behind Loft.

If you're interested in sponsoring Software Engineering Daily, reach out to us at sponsor@softwareengineeringdaily.com. We'd love to hear from you and bring your message to our audience. We reach over 250,000 developers per month.

[INTERVIEW]

[00:00:39] JM: Lucas, welcome to the show.

[00:00:41] LG: Thanks so much for having me, Jeff. It's a pleasure being on here.

[00:00:43] JM: Yeah, it's great to have you. Now, we're talking today about your work with Loft, and I've done many shows on managed Kubernetes systems. And it's always interesting to see new ones come around. Tell me about the gap in the market for managed Kubernetes that was available, despite the fact that there are so many options already.

[00:01:12] LG: Yeah, of course. I mean, there's a whole bunch of providers out there that help you set up and manage Kubernetes clusters, obviously, OpenShift, Red Hat, and then Rancher, and a whole bunch of others come to mind in that space. So yeah, very valid question. Why do I need another Kubernetes management tool? I guess. I think the way I look at it is we start with a kind of end in terms of functionality, and we don't cover a lot of things that they actually cover. So it's not like Rancher or Loft, it's typically Rancher plus Loft. The way that I think about it is these tools to help you spin up clusters, set up monitoring, logging day two operations in different cloud platforms, we don't do any of that, right? With Loft, you cannot even spin up Kubernetes cluster in AWS, that's not what we do.

What we help you do is when you have these clusters set up, and you're already managing them with any of these tools, or you're just using the plain EKS or GKE by the cloud provider directly, what we help you to do is make these clusters accessible in a self-service fashion, and shareable across your entire organization. So let's say you have 500 or even 5,000 engineers, and then need access to Kubernetes, we help you get them self-service access to these shared development clusters.

[00:02:41] JM: Got you. So what is required to build that kind of permissioning and sharing system for clusters?

[00:02:52] LG: Yeah, a whole bunch of things. I think, we're touching like this Kubernetes multitenancy topic, how do you host multiple different tenants and tenant workloads in the same Kubernetes cluster? That's a really tough challenge. Because inherently, Kubernetes is a system that – I mean, of course, Kubernetes has our back and things like that. But it's typically designed as a single tenant system. It's really hard to share Kubernetes cluster, and we automate everything around that. So we're really putting these guardrails in place for folks. But at the same time, we also help you do it in a very pleasant way for engineers.

I think one of the biggest advantages that you have with Loft is that your users have a great experience when they're using the system. So they can just even create namespaces in a self-service fashion in Kubernetes. They can even create virtual Kubernetes clusters. So you don't want to spin up 5,000 Kubernetes clusters for 5,000 engineers. But for each individual engineer, that would be kind of ideal. Actually, what would be even more ideal is if each engineer had the permission to create 10 Kubernetes clusters, but who wants to manage that and who can afford that, it's really tough.

What we let them do is spin up virtual Kubernetes clusters or simple namespaces. And inside these namespace, they can do whatever they want. And inside these virtual Kubernetes clusters, they feel like they own the entire Kubernetes cluster and have full access, they feel like an admin. It's kind of like a feeling like being in a VM. You're admin inside, but in the outside system, you're still are restricted to this to this virtual layer that you created.

[00:04:40] JM: Why isn't this kind of functionality already provided by the larger infrastructure providers like the OpenShifts or the Ranchers?

[00:04:52] LG: There's essentially just two main approaches to create virtual clusters out there at the moment. One is what the multitenancy, our working group and Kubernetes is working on. They're working on a standard for this. But it's not ready for I think the end user at this point. I know some firms and many of the main contributors are using it internally, but it's very tailored to their needs. What we're trying to do with the VCluster, our virtual cluster distribution is a general solution that can be easily adopted by pretty much any company. And we're pretty much the only project out there that makes this work, and VCluster is a certified Kubernetes distribution. So all the conformance tests, et cetera, that CNCF has out there, we're passing them, so you won't be able to tell the difference between a virtual cluster and a real Kubernetes cluster.

I assume, though, that providers like Rancher, et cetera, will in the future integrate either what the multitenancy group is doing, or what we're doing with the VCluster. Obviously, we're hoping that more providers will go for the VCluster route, and the approach that we're taking. But I'm pretty sure, manage providers will look into virtual clusters as well, eventually, but I don't think any of them is there yet.

[00:06:11] JM: I'd like to get into the engineering of Loft. So as you've mentioned, it's useful for defining these virtual clusters for smaller teams or individuals within a company to spin up their own Kubernetes resources. Can you describe what's going on under the hood in more detail?

[00:06:38] LG: Yeah, so essentially, the way it works is we typically talk to a platform team, a central team within the organization that offers Loft kind of like as a product within the company, and makes that offering. They take our product and our platform, and we kind of describe it as a platform for platform builders or platform for platform engineers. They're building on top of Loft, they're adding things, we're very customizable for their particular needs within the organization, and we have to be because we addressing anything from like a post series B funded startup, all the way up to Fortune 50 financial institutions and large manufacturers. We need to be very, very flexible.

So what happens is, we integrate with a whole bunch of different other solutions. For example, regarding authentication, we're integrating with all the standards out there, LDAP, Open ID Connect, you can hook up pretty much anything, whether you're on Active Directory, or you're using Octa. It’s pretty straightforward to hook up single sign onto Loft. And then what you really do, as a platform team, you define members of this Active Directory Group, for example, should get access to those five clusters, and they have the restriction of, they can only create five namespaces, and three virtual

clusters. And altogether, each one of these members in this Active Directory Group can use 20 gigabytes of memory, and then Loft is enforcing these things.

Another interesting thing that Loft actually add to this is cost optimization. Because a big part of this is when you're nailing the developer experience, when you're nailing self-service, when it's very smooth for hundreds or even thousands of engineers to start using this internal platform that you're building for your engineering teams, they will spin up a lot of things, and they will deploy a lot of workloads. And the big question comes up is how do we control cost and how do we limit cost? Of course, you can limit engineers in terms of CPU and memory. But one of the biggest issues that companies have is figure out, can we delete this? Is this idle? Is anyone still using this? It's really tough to make those decisions.

So we have a feature in the product as well that's highly customizable, but what it does effectively, it monitors traffic to these virtual clusters, or to these namespaces that the engineers are creating. And when they're not being used, we put them to sleep automatically. That means, engineer stops working at 8 PM at night, and 30 minutes later, or so whatever you configure, that virtual cluster or namespace goes to sleep, doesn't cost any resources, no CPU and memory cost at all. But the entire state, everything that the engineer has created inside of there, still preserved. And when they're starting to work again at 8 AM in the morning, the next day, things spin up again, and it means throughout the entire night, lots of hours through the entire weekend is typically 70% to 80% of time that people are actually not coding. All of the time, things are not costing any money to the organization, and also allows your engineers to have like 10 virtual clusters in parallel, preserve the state, and then they don't have to reset them and throw them away. Because only the one that they're actually working with right now is going to run, the other ones are going to automatically go to sleep. It’s a very, very interesting kind of feature that we are on top of, the self-service provisioning part.

[00:10:30] JM: So is that to say that these are only useful for test clusters? Because if these are clusters that you want to make fall asleep when the developer is now working on, I mean, that wouldn't be really useful for production clusters, right?

[00:10:47] LG: Yeah, absolutely. In production, you’re going to have different like auto scalers, and things that you really configure for each individual micro service, and you want to make sure they're highly available, but then they're scaled elastically. That's what you want to do in production. In pre-production, so when you're thinking of all these development, and CI/CD workloads and things like that, a lot of that needs a much – you don't want to set up all of this auto scaling and things like that, right

from the start. That's not typically how each developer starts out with a new project. You want something that does it on autopilot, and does it more rougher. But that's completely fine in pre-production. That's kind of our main use case at this point. We see that some customers are exploring virtual clusters in production as well, because they do have security and isolation benefits there and help you with, for example, hosting customer workloads in multitenant clusters.

But our core product is currently targeted towards preproduction workloads. The best use case I can give for this is people may be very familiar with review applications. You create a pull request, your CI/CD tests run through, and you have an instance of your product being deployed to some demo URL, so that a QA tester, or even a potential end user or someone can hands on test the application. If you have that review app spun up, and you have that pull request created, how long does it take to actually someone takes a look at this? Maybe a week? Maybe three weeks? That review instance is going to be up and running for all that time, right? Sleep mode is going to put it to sleep automatically, and then wakes it up as soon as someone hits that link in your GitHub or GitLab. And then fires up that namespace or virtual cluster again, so that you can actually access the application, a second later. That's very, very powerful that automated mode. But yeah, I completely agree like in production, you want to have like different auto scaling systems. You're not going to use sleep mode on production.

[00:12:57] JM: So the open source project that this functionality is based on is VCluster, correct?

[00:13:05] LG: VCluster, yeah, is the core for spinning up virtual clusters. It’s a certified Kubernetes distro. So it's like a distribution just like, I guess mini cube is a distribution, EKS, GKE, everybody creates their flavors. Rancher has their Rancher Kubernetes Engine, RKE, and we have VCluster, that's the only certified distribution for virtual clusters. So clusters that run inside of other cluster, that’s why we’re the only distribution that was able to do this and that's completely open source.

[00:13:37] JM: Hasn't there been other efforts at having virtualized Kubernetes clusters inside of another Kubernetes cluster?

[00:13:49] LG: Yeah, I think the biggest difference from what we're doing with VCluster and what other people have tried in the beginning, exploring this like kind of nested cluster approach is related to spin up these clusters with very minimal privileges. So what happens is, you're essentially just creating a single pod, single deployment and a service that you can connect to. So you have API server of

Kubernetes, and a minimal control plane inside a container, and then you have our VCluster sinker in another container and they talk to each other. And then you talk to that virtualized API server.

The big difference to other solutions is how to actually – when I create containers with my virtual cluster, where they run, that is the biggest difference. There have been previous approaches where when you start a container in your virtual Kubernetes cluster, that container runs inside that API control container, and then you have a whole bunch of like issues regarding performance, having to run Docker inside or making the Docker host on the node available to that particular container and that's all really tough. What we try to do is create virtual clusters with minimal privileges and minimal changes to your underlying cluster. And that's why VCluster is the only solution that works pretty much out of the box in all kinds of Kubernetes platforms, right?

So what are you having a localhost mini cube cluster, or you have a full-blown, highly-secure, air-gapped cluster running in AWS. I'm very confident VCluster is going to work either way. It works in pretty much any Kubernetes cluster. Of course, there's some minimal requirements to it. We don't require you to make any changes to the underlying nodes, to the underlying cluster configuration. We don't even require a central control plane that the admin needs to deploy. If you today, already have access to an isolated namespace inside Kubernetes, you're not cluster admin, but you're able to spin up your application, then you're typically able to spin up a virtual cluster as well. That's the beauty of virtual clusters and our VCluster implementation of it.

[00:16:10] JM: What was your motivation for building Loft?

[00:16:15] LG: Yeah, that's very interesting. So we actually started out with a project called DeafSpace. That's an open source developer tool for Kubernetes. You can find DeafSpace to this age. It's available, the source code is available on GitHub. It's Apache 2 licensed. It's a client only tool that you download, and that essentially replaces your Docker Compose. It's like Docker Compose, but for Kubernetes. It allows you to run a single command to stand up a microservices application that consists of 10 microservices, pulling them in from different Git repositories. And then establishing that hot reloading and fast debugging development workflow, with these microservices integration, testing them debugging them.

But instead of doing that with Docker, like Docker compose those locally, we do it with a Kubernetes cluster that may run locally or remote. That's kind of how we started out by building that project. We

originally built that project in my previous company, when we were still doing custom programming services and consulting work in the Kubernetes space. We built a tool essentially for ourselves, and it got pretty popular, and we thought about, “Interesting.” If people are starting to use remote clusters, let's say they start using local clusters, they're going to run in a whole bunch of issues, because not everybody is a Kubernetes expert, right?

But if you're using remote clusters, it gets very expensive really quickly if you're handing out individual clusters to everybody, and you have central admins managing all these hundreds of clusters. And then we are thinking, “Okay, what's the right approach to share clusters and make these operations around self-servicing?” A lot of users that start using these new cloud native development tools like DeafSpace is, and there's a whole bunch of others, there's like Scaffold from Google. There’s another great one. And there's like hundreds of other tools popping up that you can use to really do Kubernetes native development, moving from the direct Java execution or from the Docker Compose workflow towards a more Kubernetes, more cloud native workflow. If you do that, you do have that challenge around managing that access and self-servicing all these users, and that's essentially when we started working on this company, and we wanted to really solve this problem.

[00:18:38] JM: So another project that you're working on within Loft is jsPolicy, which is a policy engine. Is a policy engine necessary for making the Loft product that you've been describing so far?

[00:18:55] LG: Yeah, so policy is definitely an important part and admission control in general. Obviously, if you're looking at the policy space, workspace, open policy agent as the most established solution out there, the most advanced project out there, that we’re the earliest one in the space and it took quite a few years for the number two to appear, which is Kedano, and with jsPolicy, we created a third one now. It's definitely important to have policies in place. A lot of our customers that use Loft are using a policy agent, because the guardrails that we're putting in place are sufficient for a lot of things, but there are more complicated things that you want to ensure on top of that, that you can only do with admission control, and you need a framework to do that with.

So we've gotten a lot of questions around integrating open policy agent into Loft, but we've also – asd when we were speaking to customers, and we talked to over 100 companies that have started using our tooling at this point, we saw that there are some issues around writing these policies, maintaining these policies. And then quite frankly, just operating of policies, because it's such a large system, and if you're just wanting some few additional guardrails in place, in your pre-prod environments, you may

want something more lightweight and easier to maintain, easier to set up, easy to get started with. That's essentially why we started working on jsPolicy.

The idea was, essentially, “Hey, why are people using Ricoh, which is the language that old policy agent is using to define policies, when they could use a more popular and understandable language, like JavaScript?” I think one thing that we kind of leaned to as an example was actually NGINX’s kind of history. NGINX had this extension module where you could write Lua code to extend custom functionality, custom routing, and things like that in NGINX. I was actually using that several years before, and it was a little bit of pain, because like Lua is obviously not the most common language and feels a little antiquated and inflexible this these days.

So they actually integrated JavaScript at some point and made it possible instead of using Lua to use JavaScript. There's like V8, which is Google's JavaScript execution engine, which runs in your Google Chrome and lots of other browsers and pretty much in a lot of environments that execute JavaScript. So it's very easy to run that. It's obviously battle tested, and highly optimized because they use in the browser, and when NGINX use JavaScript, and I was playing around with that, I'm like, “Oh, my gosh, the experience is so much better.” So, we were thinking, “Hey, can’t we do the same in a policy space? Can’t we create a project that is more lightweight?” JavaScript has an easy getting started path, but also makes it easier to maintain, and share policies, because there's already existing tooling and ecosystem in place, right?

You have like NPM, as a package manager. You have npm JS as a central repository, a lot of companies already have their internal NPM registry is where they can share packages with. So it makes it much, much easier to share policies when you're thinking about testing policies. There's so many testing, rock solid unit and integration testing tools in the JavaScript and extending to the TypeScript ecosystem as well. And then you get static typing as well. That's actually perfect to look something like this with JavaScript.

Although jsPolicy in itself, written in Go Lang, obviously, like everything in the Kubernetes space seems to be, but the actual policies that you're writing, are written in JavaScript, and that's kind of what jsPolicy does. And jsPolicy is integrated in Loft, so it's for customers pretty much a no brainer to get started with adding policies. But of course, we still play very nicely with open policy agent, and if people are already having hundreds of policies in Opa, there is no need for them to rewrite everything in jsPolicy. It’s just going to work out of the box in Loft as well.

[00:23:29] JM: Very cool. So can you talk more about what the setup involves for a user who's deploying the self-service namespaces, the ability to spin up these virtual namespaces across like a large enterprise?

[00:23:48] LG: Yeah, so the setup complexity really varies depending on how complicated you need it, how big your enterprise is, and how distributed for example, your teams are. In the very easiest case, let's say you're not even an enterprise, or let's say you're a startup, or you are an enterprise that just wants to run a POC with one of their R&D teams. The easiest setup is to just deploy Loft to a single cluster, and then make that very same cluster available to engineers. So you're running Loft alongside the workloads that your engineers are going to spin up. So you have a namespace that one is Loft, and then you have all these other namespaces that users are creating. And then you have virtual clusters potentially inside of these user created namespaces.

We do have a really nice CLI to deploy Loft, that you literally download. It’s like a single binary, you put it in your path and then you run Loft start, and all you need is a Kubernetes cluster. You have a cube context. So if you can run you know cube CTL get namespaces or cube CTL get pods, any of these commands, then you’re good to go. You run off start, it's going to investigate your cluster. So it's going to see, “Hey, is this a remote cluster? Is this a local cluster? Is it running in GKE? Does support load balancers?” It does a couple of these like preflight checks, and then it configures the values for hand deployment.

Of course, you can also deploy Loft directly via Helm, but it's usually easier to go through that Loft start command and beginning, because it helps you determine the right values to set things up, then deploys the application, and then starts port forwarding. So you can immediately access the UI, login via the Loft CLI and get started. And then the next thing for you would be hooking up the domain, and then hooking up your authentication system, and then give other users access to the system, and let them explore it. And then more advanced scenarios, you typically don't want to have – you want to treat Loft more like a production service, Loft itself. So you want to separate it from these workloads. You typically don't want to have like, if you're a large enterprise, and you have like thousands of engineers that need access to Kubernetes and to Loft, then you typically run Loft as a production service, and you want it to always be available, and you want to have monitoring and logging for it in place, and these workloads that engineers are spinning up, these femoral virtual clusters for CI/CD, these Deaf namespaces and engineers are debugging with, those things should run in separate clusters.

And then you deploy off this production service, and you're connecting 5, or 10, or 20 different other Kubernetes clusters, which are then supposed to be available for your engineers. In the most advanced setups, we also support geographically distributed teams with a feature called Direct Cluster Endpoint. That feature is part of our enterprise offering. And what it essentially solves is latency issues for distributed teams. So, imagine you have an engineering team in the US and you have another engineering team in India, right? And you spin up a Kubernetes cluster to host your teams in the US and you spin up another Kubernetes cluster in India, for your teams in India, right? But Loft, itself, also needs to run somewhere, right?

So let's say Loft also runs in the US in your production cluster there, what would happen for your engineers in India, because Loft is kind of like an API gateway in a way. So when the engineers in India want to talk to their cluster in India, the request first goes to Loft, which is in the US and then goes back to the Kubernetes cluster, it says in India. Obviously, that's a round trip around the world and not, you know, very efficient, a lot of latency. So what Direct Cluster Endpoint allows you to do is when you enable this feature, we synchronize certain things into all of these connected clusters that you are managing with Loft, and that you make accessible for your engineers, and it connects your engineer directly with the cluster without having that round trip.

So your engineers in India, if they are working with the cluster in India, they would directly work with it and all the information about these users and their permissions would be kept in sync between the main Loft instance and that cluster in India. So the cluster in India can do the authentication, authorization, and all these guardrails that are in place. It can do these themselves without having to contact the central instance, which obviously reduces latency and also has a benefit regarding resilience of the entire system, because you really have like a very distributed setup for Loft. Even if one of your connected customers goes down, one of your central instance goes down, you'll still be able to use your existing cluster in India, for example. That's very, very powerful as well. That's the most advanced kind of set up for Loft.

But in the easiest case, it's really just downloading the binary, running Loft start, opening the UI, playing around with it. You can even test Loft in a mini cube cluster, on a local host cluster. Obviously, you can't make it accessible for your fellow engineers. But it's a good way to kind of get started with things. We do have a feature called user impersonation. So even if you spin it up on your local laptop, in a local host Kubernetes cluster to test around with, you could create a demo user to test the permissions

and the guardrails and then set some things up for that user, create the self-service experience, that these guardrails in place, and then impersonate the user and test that experience for that particular user. It's very powerful to kind of have this initial exploration with Loft.

[00:29:58] JM: What's your process for debugging Loft, given that there's a lot of edge cases that can arise when you're doing something as complicated as spinning up virtual namespaces?

[00:30:14] LG: Yeah, I think first and foremost, we're super accessible to our customers. So what we typically do with a lot of our customers is with all enterprise customers that allow that from their company policy, is we set up things like Select Connect, we are available to them to essentially debug issues with them. The cluster at this point is a very stable solution. So we don't get a lot of issues around virtual clusters not working. But if there are edge cases where there are maybe issues with scheduling workloads inside the virtual cluster, or you have things like – recently we had things like people want to use volume snapshots or pod disruption budgets, and VCluster was not quite supporting them yet, in a way that these users needed, they can just open issues on GitHub as well, even if they're just using the open source VCluster distribution.

We also have a public Slack channel. We're very, very accessible on that end. In terms of debugging, it can be tough sometimes when customers run into or users of the open source solution run into issues. Because Kubernetes in itself is so hard to debug a lot of times. When someone tells us, "Hey, we're trying to spin up this and the virtual cluster is not starting", I would probably say, like, 70% of the time, it has not even to do anything with VCluster itself, rather, it has to do with the underlying Kubernetes cluster. Your volume can't be bound. You don't have dynamic provisioning for persistent volume set up. It's really hard for users at this point to have that level of deep knowledge in Kubernetes to see, "Hey, why is this not working?" I think are some solutions in the space that kind of address that observability part and debugging your clusters and seeing what is not set up in a correct way. I think those solutions definitely deserve a lot of attention and credit in the future, because I see that as a – from our customer base, definitely, it's a challenging problem for a lot of folks.

Of course, we're pretty deep down in Kubernetes. So, if issues arise in our own development workflow, we're usually very quick at seeing what's going wrong here. But then, if you have a third party involved, we need to ask for a lot of logs and a lot of information about the cluster setup, before we can even make a guess about what could go wrong, just because Kubernetes has so many moving pieces. So yeah, debugging anything as Kubernetes is a challenge.

[00:32:47] JM: Is it important to instrument these clusters with observability agents and have good monitoring around them? Or are they mostly treated as indispensable and therefore not worth observing?

[00:33:04] LG: Yeah, I think there's like two different levels. I think the typical, like production level, monitoring, alerting, and things like that, that can get critical as well in pre-production. When they're really switching over to cloud native development workflows, and they start using Kubernetes, in that day to day operations when they're building applications, it's very important not to block them, right? Because obviously, you want your engineers to be as productive as possible. So these clusters need to be up and running. It's kind of like, imagine it like Cloud IDE, right? If you switched everything over to Cloud IDEs, and then your Cloud IDE is down for the day, nobody can program for a day, pretty much. I mean, sure, they can set things up locally, again. But it will probably cost them like two or three hours or like half the day is gone by setting everything up locally, again.

So switching to a Cloud IDE, it's really important that Cloud IDE is always available and your repositories always spin up, and the same thing counts for what we're doing with Loft, right? These clusters, and the workloads and the virtual clusters and namespaces you're spinning up, are inherently ephemeral. But they become mission critical in a way. The underlying infrastructure hosting these ephemeral workloads, there should be monitoring, alerting, and observability in place for your IT teams that actually need to ensure they have like a service contract in the end with your developers, because your developers switched over to Kubernetes, switched on their local Docker-based workflow to remote EKS cluster and expect these EKS clusters to be available.

One of the advantages is though, if you, let's say, since these applications should be spun up in a replicable fashion like really repeatable, so should be able to stand up your stack with a single command in a minute or two, right? If worst case, we've seen that recently read an AWS data zone goes down, and then that cluster is not available anymore, it should be very easy in Loft to just let the user access a second Kubernetes cluster that is running in a different AWS availability zone, or in a different region, so that your engineers can just spin up their workloads there again. I think that is the benefit of hosting ephemeral workloads versus being responsible for hosting actual production workloads. But yeah, similar thoughts around resilience, that you have in production as well. What happens if this zone goes down? We need to shift workloads over to this app or so, potentially.

[00:36:01] JM: There are these ephemeral environments for CI/CD and testing. But are there any production level use cases for ephemeral environments? Maybe like just rapid scalability of some kind of machine learning job or something like that?

[00:36:21] LG: Yeah, that's actually a very, very interesting question. So if you're thinking about virtual clusters in itself, what you can effectively do – so you can do two things in virtual clusters and production. And we've seen both explored by customers at this point. One is, you have these large, multitenant clusters, with lots of different workloads, and they're getting really complicated to manage, and they're really, like the API server is under fire. You're having a lot of custom resources in Kubernetes. There are a lot of requests to the API server all the time, the controller manager is under high pressure. SED may even become a bottleneck, right? Everything is super critical.

What you can do with virtual clusters is you can create boundaries, and you can create that additional layer on top to kind of make the underlying cluster less under pressure, because 90% of the requests that happen inside the virtual cluster, remain inside the virtual cluster, is kind of like a VM on a physical machine, right? Certain things are shared, but other things are totally encapsulated. What happens with the VCluster is it has its own API server and its own data store. It could be SED, it be Escalated, or MySQL, but it's separate from the underlying clusters at SED and underlying cluster at API server.

So if you can reduce, if you can take away 90% of the requests on the underlying cluster, the underlying cluster can be much dumber. And it means it can scale much further out, than it could without that kind of like – we're effectively sharding that cluster in these mini virtual clusters running on top of it, and they can get bigger as well over time. But they're going to be much, much smaller than the real underlying thing. I think that is one interesting part. And then the second part is what we already see people planning to publicly launch and put in production this year. We have two customers already working on this, is hosting customers, and let's say you are you have a managed service, you're hosting an instance of your product for each one of your customers. You have two options of Kubernetes. Either you create separate Kubernetes clusters for each customer, which is really expensive, hard to manage, right? But obviously great from a security standpoint. Your customers entirely isolate it into separate Kubernetes clusters. You can change the version of Kubernetes independently, do upgrades independently. If something fails, it only fails for that particular cluster on that particular customer. But again, super expensive, really hard to manage.

If you have a multitenant cluster instead, then you typically use namespaces in Kubernetes, much, much cheaper, much, much more flexible, it's easy to onboard new customers, much, much quicker. But you have the challenge that when you're upgrading this cluster thing that's like super critical, you have 80 customers running in this cluster, or a thousand customers. And then when you're upgrading things, you're upgrading it for everybody. Really challenging. And it's really hard to isolate these customers from each other. If you're putting the VCluster as a layer in between and Loft helps you obviously to do that, then it's much safer to operate these different customers and the application you spin up for them, because they're running in a separate virtual cluster. Each one of these virtual cluster has its own API server.

So even if they would get full access to the API server by exploiting some bug in your application, that would still not reach the underlying cluster. If you're upgrading a virtual cluster, and something fails, it only affects that one particular virtual cluster and that one customer, rather than everybody. So you're kind of getting the best of both worlds. You have the benefit of this isolation and better security, and better encapsulation and separation of things. But at the same time, you're effectively running in the same Kubernetes cluster, in the same underlying infrastructure, and that can save a lot of cost and create like this dynamic environment that you may be looking for dynamically scaling up and down your customers and really be very resource efficient. I think those are super interesting use cases in production.

Another interesting use case, you've seen a bit as well, and we're using it for our own product for this as well, is sales demos, right? So let's say a sales team just needs to spin up an instance of your application and demo it to the customer, maybe you want to give them access as a trial account or something like that. That's usually a pain and a burden on the engineering team to set it up for sales. With Loft, it's pretty straightforward. We have these virtual cluster and namespace templates. So you define a template, you pre equip it with data, you put your application in there, and then your sales team can effectively – you can put a nice UI for them on top of it as well. But you can also give them direct access to Loft and they do two clicks, the application spins up, they hit a URL, and they're ready to give that demo. That's a very interesting use case as well. I would call the production in a way, I guess it's a little bit production, but it's not as critical as your customer workloads. But I'm sure there's like hundreds of other cases that we haven't even explored. But those are the ones that we've kind of seen in our customer base so far. But again, we're just getting started. We’re on this for a little bit over a year, and there's so much more to do. We're just getting started in the space.

[00:42:12] JM: What's your vision for the future of the company?

[00:42:17] LG: I mean, obviously, it's exciting to build a quickly growing startup that is really put to use, product that is put to use by customers. I think we really want to be the leader for Kubernetes multitenancy, so we want to be one of the leading solutions in when it's about sharing clusters and sharing workloads, and making sure they're securely isolated. I think we're in a good path to become a leader in that space. And obviously, I see VCluster, as becoming the standard for spinning up virtual clusters in any Kubernetes environment.

Again, we talked about this at the beginning of the conversation, there are potentially so many other vendors that could integrate VCluster as an open source project into their products, and I would really be super excited to see that happening. I think being the leader for these two topics, multitenancy and virtual clusters, I think that's what we're out to do, really increasing that level of developer experience and self-service. A requirement for that is that multitenancy, and this level of virtualization is solved in a way and we want to be the essential building block for this.

[00:43:33] JM: Well, is there anything else you'd like to discuss that we haven't explored about Loft?

[00:43:40] LG: I think we talked about a whole bunch of different topics. I think we covered a lot of ground today. Maybe one thing that's interesting, and kind of upcoming, is we're actually launching a plugin system for VCluster to make it more extensible, and more flexible and customizable. So if you're having use cases, and if you're trying the VCluster today, and you're saying, "Hey, what we said earlier, volume snapshots are not covered in the future." Again, we're trying to be a vendor that supports VCluster to become a standard solution, but we don't necessarily need to be the one that implements every use case of the plugin system.

If we don't support anything today, obviously, you can raise a PR and see if we want to incorporate it in the core. But if we can, or it's like something that maybe too far on the edge or too specific to your customer resource definitions, et cetera, then you'll be able to use that plugin system to create a plugin for VCluster. That's going to be the next level of extensibility for this virtualization layer. I think creating that standardized interface for extending for the virtual cluster should do and how it should behave and how it should launch workloads, that's something I'm super excited about. I can't wait to launch this, and yeah, that's going to come up in just a week or two.

[00:45:06] JM: Cool. Well, congratulations on all the development and I look forward to seeing the continued success of Loft.

[00:45:14] LG: Definitely. Thank you so much for having me on the show, Jeff. I hope to meet you in person one day, maybe at the Cube Con. Not sure if you're going to be there. I know things are kind of in the air right now with COVID Obviously being constantly changing the situation. But thank you so much for having me. It was a great pleasure chatting with you today.

[00:45:29] JM: Yeah, thank you so much.

[END]