EPISODE 1435

[INTRODUCTION]

[00:00:00] JM: Managing Kubernetes nodes leads to operational complexity, security issues and nodes that are perhaps more expensive to run than necessary. Deferring the node management to an underlying platform abstracts away these problems and can improve operations.

Madhuri Yechuri runs Elotl, a nodeless Kubernetes platform. She joins the show to talk about the architecture and purpose of her company.

[INTERVIEW]

[00:00:22] JM: Madhuri, welcome to the show.

[00:00:24] MY: Thanks so much for having me, Jeff. Excited to chat with you.

[00:00:27] JM: Yeah. So you are working on Elotl, which is a system for running Kubernetes. And I like to just start by asking, what are the biggest pain points of managing a Kubernetes cluster?

[00:00:43] MY: Yeah, that's a really good question. The biggest pain points of managing a Kubernetes cluster from what we have seen in the field is that there are three parts. The first part is compute capacity management is a huge pain point. So when you hand curate your node pools and you have to figure out what shape of compute you would want, for example, on-demand spot, or Fargate on AWS, and have to figure out your cluster auto-scaling knobs for each of your node pools and constantly monitor and adjust your cluster autoscaling knobs. So that is the first pain point.

The second pain point is the operational complexity of managing your hundreds of Kubernetes control planes spread across multiple cloud providers is the second pain point. And the third pain point is multi-tenant security isolation is increasingly becoming bigger and bigger pain point as Kubernetes becomes the de facto model for deploying your apps to public cloud.

[00:01:47] JM: So why isn't this stuff addressed by simply running an EKS cluster in amazon?

[00:01:57] MY: Yeah, that's a good question. So if you are a DevOps engineer that is consuming compute on AWS, you provision your EKS cluster. You have to figure out if you want to consume the on-demand shape of compute, or spot, or Fargate. So that is the manual decision-making that is involved. If today you curate your EKS cluster and you figure out that, "Hey, there are 160 on-demand shapes. And I want to pick c4.extralarge as my node shape for my node pulls. That is the optimal decision that you make today.

Three months later, if AWS launches a newer, better shape, unless you are actively keeping tabs on what are the new shapes that are being brought to market by AWS, can you consume Fargate? Is there a better shape than c4.extralarge, you're missing out on picking the best computer option for your app. So you might make the best choice today. But that might not be the best choice three months later. Does that answer your question?

[00:03:00] JM: It does. So I guess the selection of the ideal underlying compute instances depends on kind of where your app goes over time.

[00:03:11] MY: That's correct, yeah. And the average utilization of compute in EKS or any other cloud provider is around 62. Average wastage is around 60% to 80%. So you're also constantly trying to optimize not just the instant shape, but also trying to figure out if you're actually using the compute that you've provisioned to 200% capacity utilization.

[00:03:36] JM: So as my app is changing over time, what I want to, for example, change to having spot capacity to lower cost? Or what are the other changes to that underlying infrastructure that could make a difference?

[00:04:00] MY: Yeah, that's a really good question. So there are two things associated with the optimal capacity. The first component is right-sizing your application. So if your app is you ask for one vCPU one gig of RAM for your app, and if your app is only using one vCPU half a gig of

RAM, then right-sizing the resource requirements you're asking for your app is the first level of right-sizing. And this you can achieve by turning on VPA in your cluster. And this is irrespective of whether you're using nodeless or not.

The second level of right-sizing is right-sizing the compute under the covers based on the resource requirements that are being shipped down from the control plane down to the compute. So whether the user has specified the resource requirements, or VPA has determined the resource requirements, adjusting the right optimal compute for your underlying parts is what nodeless does. So it could come in the shape of spot. It could come in the shape of Fargate or an on-demand instance. And you could also automatically figure out if you want to pack pods into a single compute node, or if you want to do one part per compute node isolation as well. So that's another dimension of optimization that is available for you as well.

[00:05:26] JM: So I guess we can start talking about Elotl, which is what you're working on. Can you describe how somebody would start to use Elotl and what benefits there would be from it?

[00:05:38] MY: For sure, yeah. So Elotl makes nodeless Kubernetes. The idea of nodeless Kubernetes is that you are now thinking of apps as cattle rather than pets. But you're still thinking of compute for apps as pets and not cattle. So what nodeless does is that it migrates your compute, the way you look at compute, from being looked at as pets to being looked at as commodity cattle. So what that means is that you can apply nodeless at two levels. You can apply it at a single control plane level. And you can apply it at a federation of control planes level.

Let me start with explaining how nodeless works at a single control plane level. Say you provision EKS cluster. With nodeless, nodeless is simply an app that is dispatched to your EKS cluster. And you have no node pools to configure. No compute shape or size decisions to make when you start with. So you just have a single control plane, EKS control plane, that is running inside your AWS account. And by the way, this applies to any cloud provider on-prem or public cloud.

So the moment an app comes in, you specify what is the policy for your app. Say, you want low-cost compute for your application. Let's say your app needs one vCPU one gig of RAM. And the moment you start this application, nodeless will provision just-in-time right-sized compute for the application.

Let's say for one vCPU one gig of RAM, you have it being available by AWS as an on-demand instance at a dollar an hour, a spot at cents an hour, and Fargate launch type at cents an hour. If those are the pricing information at that point, nodeless will pick Fargate as the launch type for your pod. And the compute is provisioned just-in-time. And the part is shipped to the Fargate compute node. Once the part terminates the underlying Fargate compute, launch type is also terminated.

I'm going to pause here to see if the single control plane nodeless makes sense before moving on to multiple control planes.

[00:08:05] JM: Well, I guess I'll just interrupt you with a question. So what is the typical relationship between a control plane and an app? Do I need one app per control plane?

[00:08:19] MY: No, no. Not at all. Your control plane is your regular control plane, your EKS control plane. And you're provisioning as many apps as the control plane limits you to provision through the control plane. It's just that the underlying compute, it's just-in-time provision and comes up and disappears according to the app life cycle.

[00:08:39] JM: Okay, cool. And, so, how many apps are typically run across a control plane? Why wouldn't I just use a single Kubernetes control plane for just all the apps across my infrastructure?

[00:08:54] MY: That's a really good question. There are limitations both from Kubernetes' point of view and the cloud provider's point of view. Let me talk about the Kubernetes limitations itself. You might have the max number of pods, max number of services limits that you might hit. And secondly, there are also region limitations from the cloud provider. For example, in US West1, you might only have certain kinds of apps available in – Certain kinds of compute available. And US East1, you might have different kinds of compute available.

And the third reason could be that you might want to service your enterprise application through the closest cluster to where it is located. So that could be another consideration as well. Another consideration is that we have seen in the field that a lot of departments internal to enterprises, they want to differentiate and delineate the clusters belonging to different teams. So they want to be able to build easily and things like that. So you might want to limit each team to a separate control plane.

So these are some of the reasons why we have seen the plethora of control planes popping up. And none of these, other than the Kubernetes limitations itself, or the other reasons are more organizational reasons and non-technical reasons. Does that answer your question?

[00:10:24] JM: It does. Yes. So, I guess we can, at this point, go into the multiple control plane use case for why somebody would want nodeless Kubernetes.

[00:10:37] MY: For sure, yeah. Whereas enterprise adoption of Kubernetes increases, and this kind of ties into your previous question as to why there are multiple control planes existing. This is the way the adoption of Kubernetes has happened in large enterprises. So when we go into the field, the customer already has hundreds of control planes that they have provisioned for their organizational ease of management and billing reasons.

So what's happening right now is that the shipping of apps to each control plane is manual. For example, let's say you're running machine learning workloads and you have five control planes in US East1 and three control planes in US West1. Because there are certain GPU shapes that are available in both the regions. So what's happening now is that somebody has to manually ship each pod to the right control plane based on what is the availability of capacity on each of the control planes in each of the regions.

So the shipping of applications to each control plane in different regions is a manual task that that is happening right now. And again, this leads to the issue of the decision that is made at the dispatch time not being the right decision when the pod is actually running. It might be an outdated decision. For example, when the operator is shipping workload A to the control plane in US East1, they might have made the decision based on the capacity that's being used on US

East1 at that point in time at 9 a.m. today is lower than US West1. But once the pod actually starts running, maybe that decision is no longer the right decision.

So the operational complexity of hand managing each of these control planes, again, as as pets instead of cattle, is becoming an increasingly prevalent problem as enterprise adoption of Kubernetes increases. So what nodeless does is it is commoditizing compute across the clusters across various regions and on various cloud providers, whether it is all on a single cloud provider or multiple cloud providers. It's giving you a single API server to talk to. And based on the user-defined policy, the app could be shipped to the right control plane.

So we are looking – We are now looking at control planes as the unit of deployment. And your application coming in with a policy, that is giving the input to the scheduler to the multi-cluster scheduler to figure out which control plane should this application run on. Does that make sense?

[00:13:30] JM: It does. So, again, why would there be a benefit to doing this sort of load balancing across multiple control planes?

[00:13:41] MY: The benefits are along couple of accesses. The first one is that it eliminates the manual need and the manual task on the operator's plate to manually ship the parts across various control planes. And secondly, it is allowing you to consume the most cost optimal or any other policy optimal compute.

So, for example, if your app wants to – If you want to run your application, and you do not care which region your app is running on, but you want it to be low-cost compute. So based on the pricing of the compute available in various regions, the application would be shipped to the right region. So optimizing for cost is the second reason.

And the third reason is disaster recovery and high availability, where if you want to spray your app across multiple regions, so that if one region goes down, your service is not interrupted. For example, in the recent past, when US East went down for AWS. If the app is spread across control planes that are in multiple regions, the end user service would be uninterrupted. So these are some of the reasons, the benefits of shipping apps across multiple control planes.

[00:15:00] JM: Gotcha. So help me understand a little bit more, if Elotl is managing my underlying infrastructure, what is it actually doing? How is it hooking into underlying resources?

[00:15:18] MY: That's a really good point. So we were very conscious from the beginning that we do not want to be in the critical path for the user infrastructure. And we want to make use of the discounts that the enterprises have negotiated for. So if we want to provision a compute node for your application, we want to be able to see the price that you negotiated with your cloud provider and not have the app run inside our account. So your application, you, as in the enterprise user, your app is running inside your cloud accounts. And the app is nodeless is running inside your cloud accounts as well. It can also run in a SaaS mode. But if you do not want to run it in SaaS mode, it runs inside your Kubernetes clusters as well.

[00:16:12] JM: Got it. And so do you connect to existing Kubernetes clusters and then become the mediator of the underlying infrastructure? Or do you expect people to spin up entirely new clusters just for Elotl?

[00:16:30] MY: That's a really good question. So we connect to existing Kubernetes clusters. So we do not require users to spin up any additional infra. Basically, nodeless is an Kubernetes API server. So all of your dispatching functions point talk to this server, to nodeless server. And nodeless server is the server that's connecting to your control plane API servers that you have in place already.

[00:16:58] JM: Got it. So does this make sense for just like a typical app that's running on Kubernetes? Just like a CRUD app? Or is it more for bursty workloads?

[00:17:14] MY: That's a really good question. So nodeless working at a single control plane level, the just-in-time provisioning, the benefits are more evident and apparent for bursty workloads because you can visualize it very clearly, that if your CI/CD app or your machine learning app is not running at all, then you do not want to consume any infrastructure.

So the early adopters of nodeless have been CI/CD applications. A lot of Buildkite users, for example, find a lot of value in scheduling their CI/CD workloads through nodeless. And also, there are a lot of machine learning folks that are noticing the benefits of nodeless right away. For example, folks from the Ray platform have seen quite a few benefits from the cost point of view and also operational simplicity point of view.

Having said that, we did a survey of the average utilization of compute for long-running production workloads that do not fall in CI/CD and ML buckets. And we have seen that the average utilization of long-running enterprise workloads running on Kubernetes across all verticals and across all sizes of companies to fall between 20% to 30%. So the utilization of compute is still pretty low compared for long-running enterprise applications as well, even though they're not bursty.

So nodeless is also a good fit for the regular production workloads. But it's very intuitive to think that it's a really good fit for CI and ML. So the early adopters of nodeless have been CI and ML workloads that are bursty in nature as you pointed out.

[00:19:01] JM: So, if it's mostly just CI and ML use cases, would I want to allocate just like a specific cluster for doing that bursty workloads, for doing those bursty workloads, and leave the rest of my infrastructure just on normal Kubernetes clusters?

[00:19:22] MY: Yeah, that's a good question. So what we have seen is that folks boost the CI cluster, one or two CI control planes, to nodeless. And then they see that the operational simplicity of nodeless is applicable to the regular clusters as well. So they start increasing their adoption from CI to regular clusters as well. So CI is a good first fit. But it is not the only fit. CI and ML are not the only fits for nodeless.

So this is not less running at a single control plane level, right? For nodeless that is running in a federated mode, each control plane could be running in nodeless mode, or it could be running regular cluster autoscaler or some other way of manual capacity management. So the multi-cluster mode can run with or without each cluster being run in the nodeless fashion.

[00:20:19] JM: So, how does this compare to just using like Lambda functions and just off-shooting all of my bursty workloads to Lambda functions?

[00:20:32] MY: Yeah, that's a good question. So Lambda functions come with their own limitations. As in, you would need to write your app in an event-driven function as a service format. And the app needs to finish within the time limitations that Lambda function compute comes with.

A better fit for it would be something like Fargate, which is able to run more heavyweight applications. So if your app can fit nicely for its resource form factor in a Fargate launch type, and it can fit within Fargate launch type, the difference between hooking it up yourselves, hooking up Fargate to EKS yourself, versus using nodeless is that it's nodeless would be more cost effective, because Fargate is more expensive than the same form factor of on-demand or spot. So the equivalent of Lambda would be Fargate. That would probably be a better fit for your regular enterprise workloads, including CI workloads.

[00:21:34] JM: So, I guess the core benefits that I’m understanding for going to a nodeless platform is, I guess, cost savings and not having to deal with manual scalability. Are there any other core benefits?

[00:21:57] MY: That's correct. Yeah. Expanding on the manual scalability, it reduces the operational complexity of having to manage the compute for your Kubernetes clusters yourself. There is also a third benefit, which is a stronger multi-tenant security. Because nodeless makes the choice between bin packing of packing apps into a single compute, versus bin selection, where each app gets its own bespoke compute. So you have stronger multi-tenant security isolation, because app A belonging to you and app B belonging to me would never be collocated on a single compute node. With container level isolation boundary, they would be getting a separate bespoke compute node that will give us a virtual machine level isolation, which is a stronger isolation than the container-to-container isolation.

[00:22:58] JM: So that isolation is mainly like a security benefit.

[00:23:02] MY: Yes, yeah. That's correct.

[00:23:05] JM: I see. So, can you get a little bit deeper into the engineering internals of Elotl? What you've had to build to enable this?

[00:23:17] MY: Yeah, for sure. Yeah. So our vision was to commoditize compute in the multi-cluster, multi-region, multi-cloud. That is our big vision. And in order to get there, we wanted to make sure that each control plane itself can work in the nodeless fashion. So what we had to build for was we had to solve for commoditizing compute at each control plane level. And we also want to make sure that if a customer is not interested in commoditizing compute at a single control plane level using Elotl's nodeless, they can use other options, like they their own cluster autoscaler, or products like Carpenter that was brought to market by AWS. So irrespective of which option you use to optimize your control plane at a single control plane level, the federation of the control planes works irrespective of your decision for a single control plane.

So the order in which we prioritized implementation was we solved for a single control plane to begin with. And we also wanted to make sure that we would be agnostic of the choice of your control plane. So it doesn't matter whether you're using EKS, or GKE, or you're deploying your own control plane from the top of the trunk of Kubernetes project. Irrespective of the control plane you are using, we wanted to be able to deploy nodeless and supercharge the existing control planes to a nodeless fashion.

So once you have boosted your existing control plane to nodeless, then the federation level kicks in. And for the federation, we again wanted to make sure that you can federate across on-prem public cloud any permutation combination of the cloud vendors. So we wanted to be vendor-agnostic and make sure that we piggyback on the networking decisions that the user has in place and not come with the bespoke network communication asks from our own end. So these are the priorities of implementation from engineering point of view. And this is, again, based on what we've seen in the field and the early feedback from early adopters. Does that answer your question?

[00:25:38] JM: It does. Can you talk a little bit more about the architectural decisions you've made to accommodate, I guess, the average use case? Like, were there any subjective decisions you made and how you architected that?

[00:25:56] MY: Yeah, that's a really good question. So when we were first solving for boosting a single control plane to nodeless, we went with a virtual kubelet-based implementation, because, ideally, that is the best suited vehicle for implementing nodeless for a single control plane.

And from based on the early feedback we got from this implementation, what we learned was that virtual kubelet, even though it's technically the best fit for implementing nodeless for a single control plane, a lot of folks didn't really understand how virtual kubelet work. A lot of folks, as in lot of enterprise DevOps teams. And also, virtual kubelet is not Kubernetes compliant, which means that only a subset of your apps could be run in a nodeless fashion through virtual kubelet.

So these two reasons, the first one being lack of understanding of virtual kubelet, which made people suspicious of, "Hey, how are we going to debug this product if it is running inside our environment and we don't know much about virtual kubelet?" combined with the limited compliance of virtual kubelet made us realize that, even though it is the best fit for implementation, it's not the right fit for bringing the goodness of nodeless to wider audience.

So we moved away from virtual kubelet-based implementation to a regular Kubernetes-based implementation. And that has jump-started adoption of nodeless quite a bit because it got rid of the two pain points that user faced from adopting nodeless in their enterprise environments.

[00:27:45] JM: Can you tell me more about like why you see the market as big enough to support this product? Do you think the number of ML workloads and CI workloads are significant enough to build enough of an audience to have a gigantic business?

[00:28:08] MY: That's a really good question. So the federation of control planes, the ease of management of that, has nothing to do with ML and CI workloads. So the adopters of nodeless for federated clusters have been beyond ML and CI. For single control plane, nodeless CI and ML are the early adopters. And like I said earlier, what we have seen is that the utilization of non-CI and ML workloads – Of compute utilization of non-CI and ML workloads is less than 20% in enterprise Kubernetes clusters. So that is what we are expanding into.

So the market that we are targeting is enterprise adopters of Kubernetes that have to manage large number of Kubernetes clusters. We want to simplify the operational complexity and reduce the spend and enable them to only spend as much as needed while improving multi-tenant security, which is applicable to all enterprise workloads that are running on Kubernetes and not just CI and ML.

[00:29:16] JM: Are you seeing uses of Elotl in hybrid cloud environments as well?

[00:29:26] MY: Yeah, of course. Yeah. So some of the early adopters of multi-cluster nodeless are hybrid cloud folks. For example, you might have some cost in your compute and on-prem data center. And you might want to burst to public cloud for your burst needs. For example, if you have burst needs during Black Friday or something like that that you want to burst to public cloud and bring your compute footprint back to on-prem once the burst need passes, that is a really good fit for nodeless as well. And we are seeing adoption for those use cases as well for multi-cluster federation.

[00:30:05] JM: What are the biggest engineering challenges that you're working on right now?

[00:30:10] MY: So the engineering challenges, biggest engineering challenges fall in two buckets. The first one is engineering challenges for solving for nodeless on a single compute plane. And the second one is the multi-cluster federation. For the single control plane, what we are solving for, the biggest challenges we are solving for, is actually making the best decision for pricing, for compute based on pricing based on historical pricing data and future predictions, and figuring out the right shape for compute for the pod. That is the biggest engineering challenge.

And for the multi-cluster federation, the biggest challenge is a policy-based placement. As in if the user says that, "Hey, I want my app to be spread across multiple clusters." Then how do we make the decision for accounting for the ingress/egress cost of network communication if your apps are spread across multiple control planes? So taking those costs into consideration as well while making the placement decision, that is the biggest challenge for the federation component.

[00:31:23] JM: Can you talk a little bit more about federation? What federation means in this context?

[00:31:30] MY: Yeah, federation means that you have multiple control planes that you want to treat as a single control plane. And you want to place your application across multiple control planes based on the user-defined policies.

[00:31:47] JM: Okay. Right. I just wanted to clarify there. So if I'm, say, spinning up a resource consumptive machine learning job and it's going to be queued into Elotl for scaling up, can you walk me through what's going on under the covers and how that machine learning job is scaling up?

[00:32:12] MY: Yeah, for sure. Yeah. Would you want this scenario explained in a federation mode or for a single cluster mode?

[00:32:20] JM: Whatever is more realistic.

[00:32:21] MY: Yeah, for sure. In a single cluster mode, the scheduler is taking in the application, and Elotl's lotus component reacts to the application's resource requirement and spins up the right-sized compute at the right price point. So let's say it needs compute shape A, and compute shape A is available on the cloud provider via on-demand sort or Fargate shapes. Nodeless will pick the best fit from cost point of view for the application, and it'll spin up the node just-in-time. And it will earmark the node as, "Hey, do not run any other apps except my new application, so that no other app is scheduled by Kubernetes scheduler on this compute node." And the scheduler will schedule your pod onto this just-in-time node that has been earmarked for this pod. And once the port terminates, the underlying compute is automatically terminated.

So this is the workflow from what the user sees. So from what the user sees, a just-in-time node came up and joined the Kubernetes cluster, and that node has been destined and bespoke node for your application. And the app is dispatched to that node. And once the ML job terminates, the underlying node is automatically disappeared. It's no longer visible in your Kubernetes cluster. So this is what happens for a single cluster in a single cluster mode.

[00:33:57] JM: And the selection of underlying instance type, that's handled on your end?

[00:34:07] MY: Yes. Yeah, that's correct. Yeah.

[00:34:08] JM: Okay. How does – Just to reiterate, can you outline what are the frustrations or the annoying parts that you're taking away from requiring the user to deal with in that process? Like, what are the manual scalability challenges that you're removing the necessity of dealing with?

[00:34:31] MY: Yeah, for sure. Yeah. So the operational complexity pain points that this gets rid of is you might have thought that when you created the cluster's node pools, you might have thought that, "Hey, for running my ML job A, the compute shape X is the best fit for my ML job." That compute shape X would have been provisioned and attached to your Kubernetes control plane. GPU nodes are expensive. So this compute shape X is always on. And you cannot scale down the node pool number count to zero. So you have wasted spend associated with it. And also, there is the operational complexity of the decision that you made for compute shape X three months ago. It might not be the best decision for your application when your app starts up. So they might be a compute shape Y. That is a better fit for your app when you're starting the app. But relying on the decision made three months ago when you actually stitched together the cluster.

So this is the operational complexity point of view. So we've addressed cost savings, operational complexity. And the third pain point is the multi-tenant security. So if you're building a platform for running your machine learning job A and you have multi-tenant requirements. So your customer one and customer two both have job A that needs to run. You wouldn't want both these jobs to be collocated on this compute node X. So with nodeless, you get bespoke nodes that are right-sized for customer one and customer two. So they are not sharing a container to contain a boundary. Does that answer your question?

[00:36:23] JM: Yeah, absolutely. So you've built this functionality for all the major cloud providers. And I'd like to get a sense for how those cloud providers compare in terms of the auto scalability that you build on top of them.

[00:36:41] MY: Yeah, that's a good question. So what we have seen is the biggest adoption of Kubernetes we have seen in the field has been on AWS, followed by GCP. And some of the healthcare apps are running on Azure due to HIPAA restrictions. So we would probably rank the adoption numbers we are seeing with AWS being number one, GCP being number two, and Azure being number three.

With respect to auto scaling feature set that is coming out of the box, GKE has the best autoscaling capabilities out of the box. GKE's autopilot is pretty good. Followed by EKS. Followed by Azure's Kubernetes Engine. So we rank GKE, EKS and AKS in that order.

[00:37:32] JM: And how do they actually differ in terms of what those – Like, what the infrastructure looks like when they're autoscaling?

[00:37:41] MY: Yeah. So EKS is your regular cluster autoscaler. So you have to figure out your compute shape. So you have to figure out if you want the autoscaler to autoscale node pools of on-demand instance type or spot instance type. For Fargate, you have to do the plumbing yourself. So the cluster autoscaler doesn't include Fargate at this moment.

GKE's autopilot will give you the right-sized VM, which is priced differently from the regular node pool VMs. So from the pricing point of view, it might be the case that bin packing onto a regular worker node on GKE might give you a lower price point than getting the autopilot price compute shape. So that is the differentiation there.

And for AKS, you have the regular cluster auto scaler where you have to figure out what is the compute shape for each of the node pools. And you have the knobs available for you for tuning the auto scaling for each of the node pools. But this, again, leads to the decision being outdated, because the knobs that you have tuned three months ago might not be the best fit right now.

[00:38:51] JM: I'd like to zoom out and get your perspective on the broader Kubernetes ecosystem and where the opportunities lie for you to expand into. Like, what kinds of other functionality you might be building?

[00:39:04] MY: As I mentioned earlier, the need for federation is increasingly become more and more important and more and more urgent. So right now, the federation component is simply an application that you can run inside your Kubernetes accounts, inside your cloud provider accounts. But doing it in a SaaS fashion is something that we see as it being an increasingly useful product space for us to enter into. So SaaS-ifying the federation is something that we see that it could be valuable as Kubernetes becomes the de facto standard for deploying your platform across various cloud providers.

[00:39:46] JM: So, can you tell me more about like what else you see in the Kubernetes ecosystem? And are there any other opportunities you see?

[00:39:56] MY: Yeah. So due to selection bias, we have been focused on solving problems in the compute space. But the compute is actually the earliest sneak preview that you would get into a problem that is going to appear in the networking and storage space as well. So since the need for federation is increasing in compute, and we are providing the first solution in market to address this need for federation for compute, what we do predict is that, moving forward, we'd see the need for solving for federation for networking and solving for federation for storage as the natural follow-ups to the need for federation for compute.

[00:40:38] JM: So it's kind of just expansion onto the same functionality that you've always been working on.

[00:40:45] MY: Yes. Yeah, yeah, yeah. But we are focusing on compute only. And that's our vision moving forward as well. So we would love to partner with the folks that are solving for network federation and solving for storage federation moving forward.

[00:41:00] JM: Yeah. So, I mean, that raises an interesting question. So as an application autoscales, like, if we're talking about that that big machine learning job, I can imagine that as all those instances spin up, you need to issue scalable networking across those different nodes. What are those network scalability challenges?

[00:41:22] MY: That's a really good question. Within the context of a single control plane, each node that we provision comes up as your standard kubelet worker node. So the networking decision that the user made for solving for networking for the control plane, that is what we use by default. So if you want to use AWS CNI, the kubelet is configured with AWS CNI. If you want to use Cilium or something else, that is what the kubelet worker is configured with.

So we piggyback on the choice for networking that the user has made already for a single control plane. The bigger problem set comes into light when you are looking at it from a federation point of view. If you have one cluster in US East1 on AWS and a second cluster sitting on GKE on US West1, how would you set up the networking plumbing? And that is a very nascent area at this point. And we are very excited to see what the network players bring into market for solving for multi-region federation.

[00:42:31] JM: So when you talk to your existing customers, are there any pieces of feedback that you've gotten that have led to some – What are the engineering issues that they've come across in the platform that have led to new developments within Elotl?

[00:42:54] MY: For the federation product, some of the requirements that we have seen is things like, "Hey, if I schedule my application A," and you have decided when it's scheduled for it to be run on cluster X in US West1. Let's say the app dies. The scheduler, is it going to keep the app sticky to this cluster in US West1? Or is the scheduling algorithm going to kick in? And it could be scheduled to US East1 to a different cluster. So these kind of requirements, with respect to the when is the scheduling component invoked? And who is the owner of scheduling for this application? Is it the master control plane that Elotl is providing? Or once the first scheduling is done, do the individual control planes, that are the worker control planes, do they own that workload for its entire life cycle moving forward? So some of these kind of requirements are very interesting for us from engineering point of view to solve for.

[00:44:03] JM: Well, as we begin to wrap up, do you have any other comments on where the company is headed and what you intend to build in the near future?

[00:44:14] MY: Yeah, for sure. Yeah. So, we started predicting the need for commoditized compute for Kubernetes five years ago. And last year, we saw the need actually be realized in the market for the early adopters of Kubernetes for a single control plane. And we predicted last year the need for federation. And we are starting to see the early adopters of Kubernetes have the need for federated compute this year.

So we want to make sure that not only the early adopters, but the regular companies, regular enterprises, who are getting on Kubernetes as the de facto platform are able to focus on their core business and not have to spend thousands of DevOps hours or time in having to hand curate compute for Kubernetes. So moving forward, we want to make the multi-cluster, multi-cloud Kubernetes, as a single Kubernetes platform that will ship your apps to the right control plane on the right cloud provider using the right compute. And you don't have to worry about making these fine-grained decisions for managing your compute on each cloud provider as pets. So we want to enable customers to treat compute on various cloud providers as cattle. So that is our big vision. And we want to be able to provide it in both in an air-gapped deployment model, as well as a SaaS model.

[00:45:41] JM: Madhuri, that sounds like a good place to close off. Thank you so much for coming on the show. It's been a real pleasure talking to you.

[00:45:46] MY: Same here, Jeff. Thanks so much for giving me the opportunity and asking so many interesting questions.

[END]