EPISODE 1444

[INTRODUCTION]

[00:00:01] JM: GitOps is a deployment and infrastructure management strategy based around continuous delivery and Kubernetes. With Git at the center of deployment workflows, policy management can be used to define permissions and rules around who can deploy and what constitutes a safe deployment. There's a synergy between GitOps tooling and policy management tooling.

Alexis Richardson is the co-founder and CEO of Weaveworks. Mohammed Ahmed founded Magalix, a security and compliance company that was acquired by Weaveworks. They join the show today to talk about GitOps and policy management.

[INTERVIEW]

[00:00:35] JM: Guys, welcome to the show.

[00:00:37] AR: Thank you, Jeff.

[00:00:38] MA: Thank you, Jeff. Pleasure to be here.

[00:00:40] JM: I'd like to start by talking about modern infrastructure security, particularly as it pertains to a world that's driven by infrastructure as code, configuration management, and Kubernetes. Mo, can you describe your experience building security tools around modern infrastructure?

[00:01:02] MA: Yeah, absolutely. Now, I'm coming from the DevOps world. So I learned about security by being a DevOps engineer and a founder of a startup in that space. And we noticed two things are taking place in that space. Number one, everything is shifting left, including how security is thought off, implemented and enforced. And that means that security as code is becoming a big thing. It's been happening for a while, but it's gaining more and more momentum.

And then the second trend that we see, or the second major change, is that everyone now is more and more involved in securing infrastructure in apps. It's not any more done only by security engineers, especially that everyone can impact the infrastructure and the application, whether they're developers, operators, or, of course, the security engineers. And so, it's important for all those roles to work together seamlessly. And that's what we see in the security tools and how they're now being developed and designed and rolled out to different teams.

[00:02:11] JM: Are there particular security snafus that you see affecting people in exposing vulnerabilities?

[00:02:20] MA: Well, since everything is now becoming codes, according to IDC survey that was done in 2020, 67% of cloud breaches or operational issues are caused by misconfigurations. And this is now becoming a big thing. And that's why you are seeing a lot of companies popping up into the space of policy as scope or in the supply chain analysis in general, and securing your supply chain. And that's what we're trying to do here jointly with Weaveworks and Magalix. We're trying to deliver basically a trusted deployment pipeline, as we call the trusted delivery. And that's now becoming a big thing, either for small teams or victims.

[00:03:06] JM: Tell me about the best means of remediation to security problems. Like, obviously, there are a lot of point scenarios that cause security vulnerabilities. But what are the macro practices or tools that you can put in place to alleviate security issues?

[00:03:28] MA: Absolutely. Now, there's no silver bullet. But as a general good practice, tackle everything as early as possible. Referring back to the shifting left everything, do not wait until bad things happen in your infrastructure or bad conflicts get leaked into your production environments. Tackle them as early as possible.

And this means that you need to build continuous security and continuous testing of your security posture as early as possible within your pipeline. And that what actually makes the GitOps as a great vehicle actually to implement that.

[00:04:06] JM: Well, I guess, it's a good time to talk about the interaction between you two. So, Mo, you're part of an acquisition of Magalix. And you are focused on policy management and the security that policy management can imply. Can you talk about why you started working on that and how that overlaps with what Weaveworks is doing with GitOps?

[00:04:37] MA: Absolutely. Now, as any startup, or many startups actually, they start in one place, and they end up in a different place. So when we first started as a company, we started as a container as a service. So we wanted to offer our customers a platform where they can run their containers without really worrying about the underlying infrastructure and the capacity of that infrastructure. We saw currently this is coming really fast from behind. So we pivoted a bit early on. And we focused on optimizing Kubernetes from a performance and cost perspectives.

When we actually started talking to our customers and to different users, we realized that most of the users are still early in their journey. They did not really reach day two ops on top of the cloud native stack. So we started actually to go back to the whiteboard, think again about what can be done so that we can serve the community better than just giving them recommendations around performance and cost optimization.

And we started with just giving them general advice about practices or best practices that they should apply, and how they could probably configure their containers, and their pods, and their network, etc. And then over time, we saw users more and more interested in the security space or security kind of recommendations that we offer.

And from that point, we started to add more and more features in that space. And we also saw policy as code becoming more and more top of mind for small teams and big teams. And from that point, actually we picked up that area. And we tried different tactics, including building our SaaS product to make it easy to adopt and help teams to hit the ground running with just a single command line that they can run at their clusters.

[00:06:28] AR: That's right. And we kind of picked up on this because we were seeing a lot of acceleration of cloud native application adoption among our open source users, which was turning into enterprise customers like Fidelity, who we have a use case of on our website, for example, who wanted to turn the GitOps pipeline, the continuous delivery pipeline, and the

GitOps management tools into a developer application platform that could run apps on Microsoft, and Amazon clouds, and on-premise as well. But these big companies wanted to have control. They needed to have security, provenance, guardrails, verifiable supply chains, and best practices and integration with their existing governance and security tools all baked into what we would do. And we saw this again, and again, and again. And started to think in terms of working with security and policy companies. And as we did more work with Magalix, we realized that, especially with their focus on policy as code, there was a really strong alignment, because GitOps is, of course, platform as code, infrastructure as code, application management as code altogether. And that means that, really, policy as code becomes a natural extension to it so that you could achieve within the pipeline all of the guardrails, compliance and assurances that you need even before deployment in some cases, Jeff.

[00:08:01] JM: So if I'm trying to secure my software delivery, and I want to use Magalix, which is now part of Weaveworks, can you just give a description for what, in more detail, it does? Like how does it actually ensure security? And how does it enforce policies? Does it use Open Policy Agent? Or do you have your own policy system for managing security? Talk a little bit more about that?

[00:08:33] MA: Yeah, Magalix is built on top of OPA, or Open Policy Agent. But we do three main things on top of it to make sure that our users get the value immediately and to the maximum kind of – And achieve the maximum security that they could have there. Number one is with the policies library that we already have and built. So we built hundreds of policies on top of it that all our users can use immediately. The second is providing an integration throughout the whole software development lifecycle. So whether you want to check your code at commit time, we're going to do this for you. We intercept any change. And we scan all your code and make sure that it doesn't get merged unless it is 100% compliant with your policies.

And our agents, which sets at the cluster, continuously monitoring the cluster and acting as a gatekeeper or a gateway for any changes that you deploy to the cluster. And if those changes are violating any of the policies that you would like to enforce there, it will also prevent those violations from being deployed to the cluster.

And the third piece is around the insights and the analytics. We know that there are going to be a lot of things that they need to handle for any team. And it's not easy really to prioritize these and have the right visibility. So we have our own dashboards that will show users what they need to prioritize. What's their general posture? What kind of policies are most violated? What kind of changes are being pushed? They need to look at and make sure that they're compliant, and so on. So those are the three pillars that we provide out of the box right now.

[00:10:17] AR: And if I may add, if you imagine applying this to a GitOps environment, then we'll be able to do several things that no one else can do, which is, for example, correlated between the deployment and the policies and the runtime impact. So that in the Magalix analysis tool, we can see the root cause, which deployments are in breach of policy at any stage in the pipeline. And that's really useful for explaining to businesses what's going on and keeping them compliant and correct at scale, especially in the presence of multiple clouds, multiple teams, multiple clusters, multiple use cases as well.

And the other thing that we can do, which I think is really cool, is if somebody is already using an open source GitOps tool, and in our case, we're promoting primarily flux and flagger here, which you could also use as with GitOps, then we can simply attach the policy system to that. So somebody can immediately go from free to policy investigation. It's a bit like adding something like Snyk, for example, where you can sort of incrementally try out different ways of introducing security into your DevOps pipeline very effectively. I think that's a very promising areas to go into.

[00:11:33] JM: Who is responsible for managing all the infrastructure at the software delivery point, the security infrastructure and, I guess, the GitOps workflow?

[00:11:46] AR: So we see three or four distinct teams, or individuals, or sort of persona in the product management language within the enterprise. And also, we deal with non-enterprise customers, too. But for the purpose of this, I think we'll focus on the enterprise. So number one is the application owners themselves who are delivering products that business customers will use for the company that's using all of these. And they just want Kubernetes to work. They probably think it's a little bit complicated. They want possibility across clouds. And for them, the application deployment workflow is simply push changes, verify they're correct? Tell me if

something has gone wrong. And then at scale, do things like rollouts. So they don't necessarily want to see all the bells and whistles.

When you get to the next layer, you have a group, which is sometimes called operations, or platform operations, or the platform team. And they may decide which are the standard components that all of the application teams are going to use. So they might choose Prometheus monitoring, Helm for templates, Fluidd for logging. And then those things might be wired up to other existing enterprise systems like AppDynamics, or Splunk, which collectively all form parts of the platform.

And so the job of the platform team is to not only wire up those enterprise systems, but also to set the policy and security of how they are used. So that when they're used by the app teams, it is done in a correct manner. And then the third group that we sometimes see is security operations, who we don't necessarily talk to, but they come in and verify that everything is done according to enterprise-wide security policies.

Typically, what we're seeing now more is the platform app teams and even individual developers are empowered to meet security requirements with, again, I mentioned tools like Synk as an example of that. If they do that, then the SecOps team is usually pretty happy. But sometimes they will come and say, “Tell me how you do this.” And you need to kind of go through a process of validation for your solution. But generally speaking, for us, it's about the application teams shipping product and the platform teams providing internal support for that.

[00:13:59] JM: As far as building a policy engine, can we talk about some of the engineering required for that? Like if I'm – Maybe we could start with just an example of a policy. And you could describe from the top-down how that gets put in to Magalix or into Weaveworks, and then how it gets pushed down and applied. And then I guess what effect that has on software delivery?

[00:14:33] MA: I can take a stand on that. Now, let me talk about three possible scenarios. Scenario number one, that you do not have any policy engine. The second one, you are trying to use the open source. The third one is trying to use Weaveworks trusted delivery, or formerly Magalix policy as code.

So the example would be, let's say you have a container. You would like to run this container in your cluster, making sure that it has the right access rights. It does not have root access to your VM. That's the most common scenario that we see. Without policy as code, without the OPA, for example, you would have to go through your own code reviews. It's a manual process. You need to check that the right flags, the right values are there in the pod manifest file. And that process is really long.

So the developer writes it, or an operator writes it, another one reviews it, and then you want to make sure that it's being deployed properly and no one is really touching the cluster or changing that configuration after that being deployed to the cluster. If you're using an open source tool, that can be somehow automated. And you need to do the following. Number one, you definitely need to install the engine. You need to understand how Weaveworks works. You want to make sure also you have the right agent and you plug it into your pipeline, and then write the policy for it. That will automatically check for those values, and make sure that those values are properly enforced throughout your software development lifecycle.

With Magalix, or with Weaveworks trusted delivery pipeline, actually all what you need to do is just run a single command line. You have the agent installed. It's relatively easy also to integrate it with your GitHub or GitLab repo. And that's it. You basically get your change checked every time against all the policies that we have. This specific policy is one of them. And you get an immediate feedback just within less than a minute.

And on top of that, actually, you get a possible remediation or an auto remediation for that specific issue. And all what you need to do is just to accept the automatically generated pull request and get merged into your original one. And then you're ready to go. And it’s secured. The agent also monitors sys at runtime. And you're also having the peace of mind that no one will be able to make the change out of band or outside of that pipeline. So this is just to walk you through an end-to-end for one of the very common scenario, and the source of a lot of evil actually and problems that you may have within your cluster. Does that make sense?

[00:17:23] JM: It does. And if a policy gets violated, what happens?

[00:17:31] AR: You can dial now – Or you can control what happens and you can have a vibe for this. So the most extreme is you're just prevented, prevented from going through. Now, some developers or operators might be interested to just letting this go through into a sandbox environment or a dev environment and see from that point what they need to do. But you can just make it as simple as a binary thing, the go or no go. Of course, on top of that, you can get all the alerts and the notifications to basically go back and see what is going on. So yeah, I see most of the time that people say, "Look, I'm not going to allow this to go through anyways."

[00:18:15] JM: When you converse with organizations on the subject of before and after using policy management, how does software delivery change when policy management is more thoroughly in place?

[00:18:33] AR: I think, for me, the key benefit is automation. So this is something which matters when you do scale for enterprise-wide operations. So, obviously, if a single individual is making one change a week to one system, which is broken up into three or four pieces, it's relatively easy for them to do everything right. It's when you have a team of let's say, I don't know, 50, 100, 500 developers working across 1 or 2000 systems, maybe microservices across many clusters and apps that things become much more difficult to do manually. And I think what's beautiful about this approach is that we can automate the policy checks as well. So that when you say to the boss, "We can do twice as much throughput. We can deliver twice as many changes per day if you want, and validate them with customers if you want." And the boss says, "Well, can you do that while also maintaining control?" And the answer is yes. So you'd have a sort of natural slowdown that's provided by kind of manual security checks or manual policy chains. For me, it's really powerful.

[00:19:42] MA: Exactly. And as I mentioned at the beginning, it's now more and more of a shared responsibility. I mean, I'm talking about securing the infrastructure and the apps. And you want to provide a common way and a common platform for everyone to do this, and do this more frequently and safely. So as we talk about the theory, you want to move fast, but you do not want to break things.

[00:20:06] AR: Move fast. Don't break things. That's the plan.

[00:20:08] JM: Right.

[00:20:09] MA: Exactly.

[00:20:10] JM: Can you take me a little bit more inside the engineering workflows that compose modern GitOps? And describe how that differs from typical continuous delivery systems.

[00:20:30] AR: So there's lots of different variations of this. And obviously, as you add things like code generation templates, fleets, and stacks, it can get more complex. So I'm going to just describe with just a very simple case. So, frequently, we see people using a tool like Jenkins, or GitHub Actions, or GitLab Runners to manage the flow or straight the flow of developer, build tests around a particular change to a system. And then when it comes to deploy the change, that is done by having jobs that are managed by the CI tool, or GitHub, or GitLab as actions or runners respectively, which then are responsible for initiating the change into the system usually by wrapping, in the case of Kubernetes, something like kubctl. So that's what we call a push-based change.

With GitOps, what we do is we push all of our changes into the repos. So we update the repos with new configuration files and new images, and maybe signed images. So there might be changes to the signature, so that when there's an update, the version of the signature will reflect that. And then what happens is that instead of pushing the change into the system normally, an agent, which is living inside the runtime, will become aware that something is available to load into the system and will pull it in, which we call a pull-based update.

And this is something where the security properties can be inherited from the core running system. And there's no need for any human or automated external system to touch production, it’s because it's pulling changes in rather than having them pushed in from outside. Or lots of variation to this. People who like push can make push look more like pull and make pull more like push. But the main issue that's changing is, for the last mile, an agent is responsible for making sure not only that the change is loaded into the system. But also, then it is applied correctly, and there is no drift from the correct state. So this is a way to guarantee that your cluster stack and app are all in the right state. And I can tell you, if you go to any enterprise and

you ask them, “Is your Kubernetes stats in the right state?” Most of them will look at you funny. And they'll go, “I don't understand. How can I tell that? I don’t even know.”

And so if you say to them, “Well, wouldn't it be great if you knew that you always have your stats in the right state?” People will go, “Hell, yeah. I want to have them in right state for all of my fleet.” And that's what GitOps gives you, because it's doing this continuous verification based on pulling in changes and applying them correctly, and then converting them through the agency of the runtime orchestrators into the right state and then picking up drifted apps and forcing it back on track, which you just can't do with an external orchestrator because it doesn't understand the state of the system.

So people say, “Well, will I have to change how I do things?” And the answer is not much. Because instead of pushing changes directly into the cluster, you push them into the repo, and then the cluster will do the rest for you. So actually, you do less, not more. It's a lot easier.

[00:23:38] JM: I'd like to talk about how GitOps affects different areas of a workflow. So if we think about the management across different services and the interactions between different services, the connections between different services, obviously you want independence. You want independence of operation between those different services –

[00:24:13] AR: Independent lifecycle, loose coupling, things like that are good. Yes. Is that what you're saying, Jeff?

[00:24:17] JM: yeah. And I'd like to get a sense for how GitOps workflows relate to having better service partitioning and service organization.

[00:24:32] AR: Interesting question. So what I would say is that you're asking a question, which falls into the bucket of advanced GitOps, if you’d like. Or some people have been calling it GitOps 2.0, which is once you have the basics, which is what essentially what I described of self-healing, reconciliation based on agents that reconcile the desired state and the running state always and automatically, and then figure out how to scale that. What about all of these other things?

And one of the other things is how do I deal with multiple partitions? Multiple clusters? Or how do I interact with external services? What if I have to do something to a database to change it into an alternate state while I'm doing an update to Kubernetes. And all of that stuff can be achieved by attaching external tools to the core GitOps agents.

And one of the things that we love about policy at Magalix and OPA and also other policy tools like Kyverno, and people are starting to use **[inaudible 00:25:34]** as well. So all of these things can be part of a more richer workflow. So that I can actually say, “Make this change to one Kubernetes. Make this change that's dependent on it. Make another change and another change. And then check that this is working. And report back to the user and only when that's all correct. Tell the user, “Is everything done?”

So I can actually – As part of the policy engine, I can do a post-deployment check or a pre-deployment check on what could be a quite a complex set of actions. And that's one thing that I think people don't appreciate about these policy engines, is they're really powerful integration tools as well. So if you've been in computing since the days of integration brokers, CORBA, ESBs, back before 2010, you'll know that in the enterprise, there's a lot of scope for pulling things together using tools that connects to external systems. And the policy engines typically provide you with that, because they have to – In order to check that certain things are true in respect to the external enterprise systems as well. So that's generally how it works. Tell me, Jeff, I've explained it. Because I’m think about it now, it probably was a bit of a mouthful. But I think that, conceptually, it's fairly simple.

[00:26:47] MA: Yes. And I would like to add just a little bit of that from a policy scope perspective and how the compliance and security can be done there. Now, it can be utilized for either a centralized security management or a configuration management and distributed also model. And in some organizations, they may have their teams or a subset of their teams focusing on writing the right policies and educating everyone about them and enforcing them throughout the whole system.

But at the same time, you do not want to take away developers and operators’ autonomy. You still want them to deliver the value of their platforms and their apps really quickly multiple times, and by doing multiple updates in a single day. And that actually – Hopefully, the way that we're

presenting it will help teams to adopt that model instead of have the maximum possible control without really jeopardizing the autonomy and agility of the teams.

[00:27:54] JM: So when you're talking to developers who are transitioning to GitOps and they presumably need to make some specific changes to their workflows and the way that they manage their code, can you talk about the specific changes that they need to make as teams to allow for a better GitOps workflow?

[00:28:21] AR: So, usually what happens is that they install a GitOps tool like Flux to their existing Kubernetes cluster. That's an important step. And then they have to wire up flux so that it can point at the image repo where the container images will be available for loading into the cluster, and also the config files. And then they need to understand that when Flux detects changes in either of those two repos, it will essentially pull them into the cluster.

That's kind of it, which means that the only thing we need to stop doing is whatever jobs they had pushed directly into the cluster from their CI or manually and replace those with things that simply update the repos. So it's a relatively small number of changes.

[00:29:06] JM: We've covered GitOps in pretty considerable detail in previous episodes. Maybe we can return to the subject of policy management and policy deployment. So if I have an existing infrastructure and I'm getting started with policy management, is there a way to simply apply a set of standard policies? Are there some just kind of boilerplate policies that can be applied to my infrastructure?

[00:29:40] AR: There are. And I'd like to say a few words, then hand it over to my colleague, Mo. So this is one of the advantages of having a solution that bundles policy with something that has an extra purpose in the case of GitOps deployment and management, which means that we can do things like have standard policy policies that everybody should follow when they're doing a deployment to Kubernetes.

Like, for example, if you deploy a container and check that you have an accident, we have secrets available for people to see. And there's just 100 things like that. And if you can provide people those out of the box, then they can avoid 80% of the problems they run into if they are

starting from scratch. So that's really, really valuable to a lot of people. And you can also adapt that set of policies over time as you learn more about best practice. You can include more and more best practices in your policy. So the policy tool becomes a channel for communicating best practice to the end user without the end user necessarily having to be an expert. I think that's really, really great.

But Mo, why don't you say a few more examples and a few things that you've seen in the few years that you've done Kubernetes policy, which I'm sure a lot more sophisticated than my brother **[inaudible 00:30:50].**

[00:30:50] MA: : Right. Absolutely. Now, just to simplify it even further, anything that can be expressed as code or, declaratively, you can write a policy for. As simple as that. Now, the common examples that we see are basically scenarios where developers or operators can shoot themselves in the foot. And like the example that I mentioned at the beginning, running containers as root on your VMs, especially if you're using open source libraries. You can easily have a few cryptocurrency mining libraries sneaking into your code and exploiting your VMs. This is one example.

Controlling the network traffic might having a few network policies is another category of policies that you would like to have here. Another one that can also impact the stability of your system in general, which is controlling the resources that each of your containers or pause use out of your VMs. A lot of time to, actually, developers do not advocate CPU and memory with a maximum and a minimum for their containers. So as a result of that, either the VM would be overloaded. Or sometimes the containers would be really starving getting enough CPU and memory and eventually crash or just having something similar to the denial of service attacks.

And then there's the other side of it, which is policies that would cover general industry standards, something like PCIDSS. So if you have parts of your system processing some financial data, you want to make sure that you have the minimum there in terms of checks and controls to meet the PCIDSS. HIPAA or GDPRS is also another standard that you want to sometimes enforce in some of your systems.

So the policies can get pretty broad in their scope. As I mentioned at the beginning, if it's something that can be expressed declaratively or through code, you can write a policy for. The ones that I've mentioned are the common ones. We have in our system hundreds of policies covering all of these, and we keep adding those every day.

Now, one last point to add about policies, and what developers need to consider when they're writing policies, is the policy itself is a living entity, and it evolves with your system and with your requirements. The policy, sometimes people think of it as just an FL statement, which is, in a lot of sense, it is an FL statement. But there are going to be lots of exceptions that you need to consider. There are going to be lots of possible parameterization for your policies that you need to have there. You need to test them in different environments. You want to make sure that they're always giving you the same results no matter how your developers are expressing different configurations of the same thing, and so on. So that's also another thing that makes writing policy sometimes a complex task. And this is why we consider this is one of, I would say, the value propositions that we offer out of the box for everyone.

[00:33:52] JM: What, in more detail, does Magalix give you on top of Open Policy Agent? I mean, there's a lot of functionality you get from just open source Open Policy Agent tooling. Can you just give a little bit more detail on what you provide on top of that?

[00:34:14] MA: Yeah, absolutely. So that the first thing is the policy. So the Open Policy Agent does not give you that wide range of policies that you can get tested, verified for almost all the common environments. So that's the first thing. The second thing is the easy integration with your GitHub, with your CI/CD, with your runtime. All of that is being offered easily without doing really a lot of things in terms of connecting a lot of installable software components.

The last one is, and this is a piece that's not easy to get with the ops versus the reporting part, and the dashboarding. We provide two kinds of visuals for our users. The First is, as we call them, the tactical dashboards, to help developers and operators to act immediately on what is broken and what needs to be fixed. And the second one is more kind of a strategic, broader view of the system that will tell you what is your overall posture? What the hot areas you have in your cluster or in your infrastructure in general that needs attention over time. And then the last piece, if you think of all those three horizontally, that is how we deliver that? So we have a SaaS

version of OPA, if you will, that makes it really easy and quick for anyone to realize those values.

So single command line, less than five minutes, and you get all the possible violations in your cluster, as opposed to spending days just trying to understand how the engine works, install it. And a few more weeks to understand how Weaveworks works and write policies for that.

[00:36:03] AR: There's also a key UX **[inaudible 00:36:03]** benefit. Once you've helped the user to be successful with this generation of policy tools, you can bring them along as those policy tools evolve without changing the GUI. So if somebody comes up with a brilliant new way of doing something important for cloud native using Kyverno or another tool, we can just make that part of the solution that already exists, rather than the developer having to learn a whole new system. And that I think is really, really helpful.

[00:36:32] JM: As we draw to a close, I'd like to just talk a little bit about an acquisition and the process of unifying the software stacks of two different companies. Can you tell me about the engineering work that's taking place in unifying the two products?

[00:36:51] AR: I think there are two answers to that. Which one of us should give the other? So I can give you a high-level answer, which is that the beautiful thing about Magalix is that its additive. So we can have existing Flux users, existing GitOps users and offer them enterprise solutions with a Magalix technology as an add-on. So the integration cost to that is simply doing the authentication dance for you so that you don't have to do it yourself, and coming up with specific integrated policies that solve specific GitOps problems. But the general tool is available right out of the box.

Now, having said that, there are some other things that we and Mo's teams have done around policy as code to bring it closer to Flux, on-premise work and other things. So that's really cool. Mo, why don't you talk us through those? Because I think there's more real software engineering going on there.

[00:37:42] MA: Yeah, absolutely. So as I mentioned, we're born as a SaaS company. And our team add that twist to the OPA, which is being offered as a SaaS. And that by itself definitely

gives an opportunity to a lot of existing GitOps users or Flux users to add those guardrails on top of their existing GitOps. And we have that offered right here today. And we have 30 days free trial for anyone to try our policy as code. And then from that perspective, actually, we can evolve an exciting and a common platform where everyone can jump on to the trusted delivery wagon in a matter of minutes.

[00:38:24] JM: Last question. You guys have both seen the continued development of the cloud native ecosystem, the growth of things like service mesh, and obviously, GitOps, as we've discussed. I think probably multi-cloud is also becoming a little bit easier. Are there any areas of the cloud native stack that you're paying a lot of attention to right now? Or that you see as the future of new areas for growth?

[00:39:02] AR: Yes. Certainly, integration with service mesh, and everything from layer four up to layer seven in what in Europe we call routing. And Americans, sometimes they call routing. It's something that can work with GitOps with Flagger, for example. And what we're doing now is expanding that to fleet. So that you can have very large numbers of stat clusters, which are managed individually. And you can have rollouts across these fleets with complex strategies.

But the other areas that I think are really interesting are how do you deal with individual stacks so that you can manage dependencies? Because everybody's talking about how do I rollout to developer platform? And they're sort of wondering, "Well, what was wrong with Heroku?" And the answer is, "Well, Heroku is fantastic."

But what enterprises need if they need their particular choice of platform components wired up in a specific way for multiple use cases? So is there a way to encapsulate all of those choices as a set of dependencies, which you can rollout systematically on top of clusters and then maintain, keep them up to date, patch them, manage them like applications, and rollout applications on top? And the answer is yes, you can do all that with GitOps if you interact with things like Helm charts, customisations and other Kubernetes extensions in the right way. And this all sounds like a lot of words. But what it boils down to is people can have any stack they want, wherever they want. And it's always in the right state. And I think getting that right is going to be the big win for this year. So that's one area. And then the other one is just making the scale, as I said, to fleets is really interesting. Adding things outside the clusters. How do I do a

cluster change in Kubernetes and a non-Kubernetes service like a database change? That I think is the big next step as well. And the future, the flag on the hill is an entirely GitOps managed data center. And then things will be really, really cool.

[00:40:59] AR: If I add to that pretty quickly from a policy as code perspective, we actually experimented at a certain point adding policies for the service mesh frameworks. And we found that really promising. Because again, with more technologies, promising technologies like this one, there's a lot of great stuff that you can do. But also, there are many things that can go wrong and that can basically send your infrastructure south. And you want to make sure that you're doing the proper guardrails on top of it. And that's also a perfect piece that can be added to any of those technologies as they gain more momentum and steam in the community.

[00:41:39] JM: Awesome. Well, guys, thank you so much for coming on the show. And it's been a real pleasure. Congratulations on the acquisition.

[00:41:45] MA: Thank you, Jeff.

[00:41:46] AR: Thank you, Jeff. Pleasure to be here.

[END]