EPISODE 714

[INTRODUCTION]

[00:00:00] JM: A policy defines which users and applications can access and modify resources in a computer system. In a file system, a user might have permission to read or write to a file in a cloud. In a cloud infrastructure deployment, a user might have the rights to deploy a new server. One microservice may or may not have the necessary permissions to talk to another microservice. All of these are use cases where a policy defines the behavior within this computer system.

Policies in a company can be managed in a range of ways; configuration files, dashboards and centralized permissions databases. A policy engine is a system for managing and automating the policy creation and deployment within an organization. Microservices need to verify each request that comes in to ensure that the request has the correct permissions. To check those permissions, a microservice can contact the policy engine. That policy engine has all the information from the whole organization about who is allowed to do what.

However, talking to the policy engine over the network can be a slow process. Open Policy Agent is a deployable agent that can run as a sidecar next to a service and check policies by looking inside of a cache.

Torin Sandall is a core committer to the Open Policy Agent project and he joins the show to talk about Policy Management, the Open Policy Agent and the Kubernetes ecosystem. Surprisingly, he also talks about WebAssembly.

Before we get on with the show, I want to mention that we are looking for sponsors. If you're interested in reaching more than 50,000 developers on Software Engineering Daily, send me an mail, jeff@softwareengineeringdothe.com or go to softwareengineeringdaily.com/sponsor to find out more information.

[SPONSOR MESSAGE]

[00:01:58] JM: Data holds an incredible amount of value, but extracting value from data is difficult, especially for non-technical non-analyst users. As software builders, you have a unique opportunity to unlock the value of data to users through your product or service. Jaspersoft offers embeddable reports, dashboards and data visualizations that developers love.

Give users intuitive access to data in the ideal place for them to take action within your application. To check out Jaspersoft, go to softwareengineeringdaily.com/jaspersoft and find out how easy it is to embed reporting and analytics into your application.

Jaspersoft is great for admin dashboards or for helping your customers make data-driven decisions within your product, because it is not just your company that wants analytics. It's also your customers that want analytics.

Jaspersoft is made by TIBCO, the software company with two decades of experience in analytics and event processing. In a recent episode of Software Engineering Daily, we discussed the past, present and future of TIBCO as well as the development of Jaspersoft. In the meantime, check out Jaspersoft for yourself at softwareengineeringdaily.com/jaspersoft.

Thanks to Jaspersoft from being a sponsor of Software Engineering Daily.

[INTERVIEW]

[00:03:31] JM: Torin Sandall, you are a software engineer at Styra. Welcome to Software Engineering Daily .

[00:03:36] TS: Hi, Jeff. Great to chat with you.

[00:03:38] JM: Today we're talking about opportunity daily budget of the city were talking about policy engines and policy management. What is a policy?

[00:03:44] TS: It's a great question, something that comes up a lot. I think it's useful to have a high level definition, and when we talk about policy, we typically talk about it just as like a set of rules, right? They use to govern the system. But policy typically means different things to

different people. If you talk to a network engineer, it means a bunch of ACLs or something like that or traffic shaping configuration on a network device, right? If you talk to someone from like infosec, it might mean a bunch of IAM rules in their platform. It means different things to different people, but the way that we approach policy with the Open Policy Agent and at Styra is we think about it as just a set of rules and govern how the system should behave.

[00:04:26] JM: Why do we need policies in our software systems?

[00:04:30] TS: That's a good question. Yeah, if you take a step back any you look at the way that modern systems are sort of built and operated, there's a tremendous amount of complexity. These modern systems and modern organizations are running all these different kinds of technology. They've got these very complex stacks. There're always different languages. There are in all these different execution environments, different protocols, different formats. There's all these different stakeholders that care about how the system is behaving.

At the end of the day, it's important you think about policy kind of holistically across all of these things, because otherwise if you don't get it right and otherwise the cost is really high. You face things like security breaches. You can get fined if you violate certain regulations. You can have downtime and so on. At the end of the day, the reason why we care about policy is because it helps us keep things running. It's about business continuity. It's about security. It's about privacy. It's both very important.

[00:05:34] JM: We can use policies to restrict access of humans or of computers, of automated systems. Give a few examples of how policies are – Just give us some examples that show off the broad spectrum of policies that are imposed upon a computer system.

[00:05:55] TS: Like I said, policy means different things to different people. So if you talk to someone that's working at the infrastructure level, they're worried about maybe network policy. They care about what IP's can talk to what other IP's. They care about what ports are exposed. You have an unencrypted HTTP server exposed on the public Internet. Those are examples of policies that apply at the network layer, at the infrastructure layer.

When you move up the stack a little bit and you start talking about platforms, like container management platforms and you care about what images are running on the platform. Where are those Docker images coming from that your developers are deploying through Kubernetes? That's an important question.

You go up to the app layer and you start talking about, “Well, every application basically needs some kind of authorization system in it. You need to be able to control who can do what in that application.” So then you’re talking about things like API authorization, and data filtering, and data protection, and data masking. So there's lots of interesting applications for policy in the application level and in the data level as well.

[00:06:56] JM: Yeah, I saw the example in a couple different slide decks of Netflix wanting to impose policy, or actually this was just a broad example, but I think Netflix used in one of their presentations. The example of if you have some kind of salary system, you want to be able to see the salaries of people who report to you and people who report below them. So you can see the salary tree of reports below you, but not above you. If you're my manager, I shouldn't be able to see your salary. This is yet another form of policy management at an even higher. Well, it's an example of the high-level of application level, I guess, data filtering or data privacy that you gave.

[00:07:46] TS: Exactly. Yeah. Yeah. The challenge that a lot of organizations face is that if you solve for all of these different policy problems independently and you don't try to unify them at all, then it makes it really, really hard to kind of control the system or even know what's the actual policy, overall policy sort of status that it’s place right now. So if you want to just know like an answer like, “Is it possible for somebody on the public internet to access like sensitive data in the system?” That might require evaluating network policies, application level policies. It might require knowing who has like SSH access to a machine, because if you get on the machine, then you can probably access the database or something like that that’s running on it.
So it's really important when you start thinking about policy in large organizations to think about it holistically and to trying to solve for that.

[00:08:37] JM: Now, this sounds like a gigantic set of, well, policies. If you're talking about from the top level of who can see whose salary to the bottom level of what microservice can provision

additional instances or additional units of memory, there's a giant range of different policies that could be imposed across our infrastructure at a given company, and I think you're advocating for a single system to manage all of these different policy. Am I understanding you correctly?

[00:09:13] TS: Yeah. I mean, I think that at the end of the day, you do want like centralized management. You want to have one place that you can go to that you can ask those kinds of questions of, but I don't think that you necessarily need to enforce that policy centrally, and there are going to be places where you want to express the policy differently. So I think that the important thing to think about or one of the important requirements to keep in mind, and this is something that we talked about with Netflix, which was that these modern systems are kind of characterized by heterogeneity. You've got no always different languages and protocols and identity types like you mentioned, like you might have batch jobs running. You might have scripts. You might have services. You might have users all accessing the system. You've got all these different identity types that you need to try to unify over.

The challenge is that it seems like you're trying to boil the ocean. So the starting place I think for solving this in a better way is to think about decoupling at least the policy decision from the policy enforcement. So that's like a big part of what we try to advocate for with OPA. Whether you're using the Open Policy Agent or you're using any policy management or anything like that, the idea of thinking about decoupling your policy decisions or policy enforcement in your systems is super, super important, because that's how you solve for that heterogeneity. You don't hardcode the policy in every place that needs to evaluate it and you kind of think about separating it out.

[00:10:31] JM: Policies are nothing new, at some regard. I mean, our systems are new. So the systems that we're dealing with, the cloud infrastructure and the sprawl of the size of our systems have certainly grown. The size of our organizations have grown, but the idea of managing policy, this goes back to read/write privileges on a UNIX machine. How have policies historically been managed in software infrastructure?

[00:11:00] TS: That's a good question. I mean, I think that there have been plenty of policy systems in the past. The networking space is full of them. The protocols like RADIUS and so on that help you manage AAA in network environments. There are plenty of policy systems at the

host level in UNIX environments, things like PAM help you manage SSH and pseudo. There are other examples, though more kind of general-purpose policy systems. So there's a framework called XACML, which stands for eXtensible Access Control Markup Language. So as the name indicates, it was like an XML kind of based policy system. People that work in the policy space definitely are familiar with XACLM. They've heard about it, but it never really saw wide adoption, and that's for probably different reasons.

There are existing solutions in the space, but I think that today what we're seeing with, like I said, all these kind of compile time heterogeneity, all these different languages and so on being used to build systems, and also like sort of like runtime dynamism. Like you've got container workloads that are spinning up and spinning down and you've got all these different identity types worried about workload portability and all these concerns. It's becoming more of a problem that people need to think about solving, because the alternative of like relying on kind of like traditional perimeter base network security doesn't really work in these newer environments. You just don't have the same kind of guarantees.

[00:12:23] JM: We just did a show with the SPIFFE and SPIRE Project, which is all about zero trust networking, and it sounds like this project, the policy and open policy agent, that's what's an OPA, open policy is, right? It sounds like some of the ideas, there's some overlap in ideas in the changing nature of our systems.

[00:12:46] TS: Yeah. Yeah, exactly. So I think that we're starting to see a lot of projects like SPIFFE and the service mesh special projects and then the Open Policy Agent kind of start to address these things by providing kind of like libraries or reasonable building blocks that people can kind of take and put together and solve the problem holistically. When you start to talk about zero trust, zero trust, it kind of just means you have two parties that want to communicate and you don't want to rely on some kind of like implicit security in between those parties to secure the system. So you want to basically push the security out to the edges to those two parties.

So the first thing you need to do there to achieve that is you need to establish some kind of identity. You have to have some kind of authentication system in place, and that's what systems like SPIFFE provide. But once you've sort of solved authentication or identity and authentication, which to a large extent today are kind of like standardized to a large extent, you

have to sort of start thinking about authorization and kind of like how you're going to enforce rules that are very specific to your organization. That's where the [inaudible 00:13:45] comes in. Once you've dealt with authentication, then you can start thinking about authorization and policy on top of that.

[00:13:50] JM: Now, before we get into modern solutions and the idea of a policy engine, there are some anti-patterns with policy control that some organizations are probably engaged in, or maybe anti-pattern is too strong of a word, but there is this spectrum from manual policy control to automated policy control. Can you help us understand the gradient between manual policy control and automated policy control and explain what's problematic about somebody doing manual policy control; manually editing a config file, or changing a little dashboard thing that doesn't propagate in a consistent way? Help us understand those things.

[00:14:39] TS: Yeah. Yeah. Yeah. Yeah. For a lot of people, policy is basically, it's a wiki page or it's a spreadsheet or it's a document. It's a PDF. It's a checklist and a PDF or something like that. The problem with that approach is that you get absolutely no guarantee that your policies are being enforced. So any time that you want to know if the system is secure, if you've got a good security posture, you have to go and talk to some person on another team on another floor or something like that and hopefully they know the answer. Hopefully they know the answer. Hopefully they've set the configuration file just right and it hasn't changed and the system hasn't changed in a way that it validates that.

Those sort of manual approaches the rely on tribal knowledge and kind of just documents, those are very brittle, because the system evolves and they'd fall out of date and they provide absolutely no guarantee that things are being forced. So that's sort of on the manual side. As organizations grow and as their systems become more and more dynamic, that quickly falls over. So what a lot of the time happens is that they kind of transition to an automated approach and when they transition to that automated approach, a lot of the time they're under constraints. They need to ship other features. They need to get things out the door. So they can't necessarily think about the problem holistically.

So a lot of the time you find policies being kind of just hardcoded in the software, like policy checks that you think, "Okay, today, this is what the policy is today. This is what the rules are

today. So I'm just going to write a few lines of code in my service and that will be in enough." Well, that gives you a stronger guarantee. The problem is, is that a lot of the time especially like in large enterprises, a lot of the time the policies are very kind of like organic. They evolve over time and they change. That means they're difficult to predict and that means that the thing that you've hardcoded today is no longer going to be valid tomorrow.

Where some people or some organizations kind of fall into a bit of a trap is that they end up kind of codifying their policies and software in such a way that it's difficult to manage them over time. That's what I would say there. I mean, you can obviously build your own kind of – You can kind of build your software to be as extensible as you like, but it takes a lot of effort to do that and it's hard to get it right, and a lot of the time, the extension mechanisms that you build in, like a lot of time if you try to build domain-specific language or something like that, it might meet half of your – 50% of use cases, but then you couldn't predict the ones that you have tomorrow. So then it falls over there. That's what we see in the space.

[00:17:19] JM: A policy engine is a tool that we can use to get us closer to more automated policy management. What is the policy engine?

[00:17:29] TS: The way that we look at it is that policy engine, it's a service that basically helps you offload policy decisions from your own service. So whenever your service needs to make a decision, like should this API request be allowed or should this user be allowed to log in to this machine, or should this container be allowed to be deployed, right? Those are all examples of policy decisions and those are things that a lot of the time are very organization-specific. They've very dependent on context. They're dependent on information from the rest of the system. You don't want to go hardcoding those things into your software. Instead, what we say is take those decisions and offload them to a dedicated engine that has kind of programmability as a first class citizen that it empowers the administrator to kind of change and tune and analyze it as needed.

Policy engines are just these services that allow you offload policy decisions, like API authorization is a great example, but they allow you to offload those authorization decisions to a dedicated component. Then you can focus on building out that component, hardening it, making it more performant, making it more scalable, making it more usable, building higher-level tooling

and so on. It allows you to – You kind of want to think about decoupling again. You want to split the problem out so you don't have to try to solve it N times. You want to be able to solve it once and then investment in that one solution.

[SPONSOR MESSAGE]

[00:19:00] JM: OpenShift is a Kubernetes platform from Red Hat. OpenShift takes the Kubernetes container orchestration system and adds features that let you build software more quickly. OpenShift includes service discovery, CI/CD built-in monitoring and health management, and scalability. With OpenShift, you can avoid being locked into any of the particular large cloud providers. You can move your workloads easily between public and private cloud infrastructure as well as your own on-prim hardware.

OpenShift from Red Hat gives you Kubernetes without the complication. Security, log management, container networking, configuration management, you can focus on your application instead of complex Kubernetes issues.

OpenShift is open source technology built to enable everyone to launch their big ideas. Whether you're an engineer at a large enterprise, or a developer getting your startup off the ground, you can check out OpenShift from Red Hat by going to softwareengineeringdaily.com/redhat. That's softwareengineeringdaily.com/redhat.

I remember the earliest shows I did about Kubernetes and trying to understand its potential and what it was for, and I remember people saying that this is a platform for building platforms. So Kubernetes was not meant to be used from raw Kubernetes to have a platform as a service. It was meant as a lower level infrastructure piece to build platforms as a service on top of, which is why OpenShift came into manifestation.

So you could check it out by going to softwareengineeringdaily.com/redhat and find out about OpenShift.

[INTERVIEW CONTINUED]

[00:21:08] JM: I've got some detailed software architecture in my company. I've got hundreds of employees. What is the process of adopting this policy engine? I've already got policies in place all throughout my company. How am I going to adopt something for this purpose?

[00:21:27] TS: I think a lot of the time it's difficult to just – You can't solve everything all at once. So you want to identify like a use case or a policy engine is a good fit. So if you're looking at building out, say, like a platform within your organization where you think that this platform is going to be used for a number of different purposes. There's going to be a number of different teams relying on this platform. They're going to have different requirements around things like resource management and security, then you start to think, "Well, okay I need to build out this thing and it's going to be extensible. I don't want to design myself into a corner. So I'm going to, basically, identify the places and the system where policy decisions have to be made and I'm going to kind of make sure that I have the right extension points in place so that I can kind of change and evolve the system over time to meet the different needs that customers are going to have.

A lot of the time, the place to start is just to ensure that you have those extension points, that you have like a WebHook or something like that where you can configure it to call it to another system and get a get a decision back. That's sort of like the simplest way to approach the problem. Identifying those places where policy decisions need to be made and then ensuring that the thing that has to ultimately enforce the decision can get it from something other than its own internal code.

[00:22:45] JM: Right. Maybe in a gigantic organization, like a Netflix kind of organization, if you decide you're going to implement the policy engine, maybe somebody is in charge of standing up that policy engine and then they can tell everybody else in the company, "Hey, I know you're standing up new services all the time, you're standing up new databases all the time, those databases have some access control logic and some roles associated with them. You're going to need to build policies around your new services. From now on, if you want, you can use this policy engine that I've got." Then gradually people in the company can start using the policy engine to have a programmatic way of managing policy.

[00:23:26] TS: Exactly, yeah. I think that in a lot of organizations that go down that road, the application team, the people building the applications, like the product engineers, they're happy, because they don't have to worry about rolling another authorization solution again. They kind of rely on sort of battle tested internal solution for the for the problem.

Obviously, it's difficult though to scale one of these things out in a large organization. By scale, I don't mean like system scalability. I mean, you have people that have to author policies. They have to write the rules, but that's not their day job. They don't spend eight hours a day writing policy. So the challenge is how do you expose the policy system then in a way that's easy to consume. I think that usability is a big challenge and it's actually probably the most important thing to think about when you're talking about adopting a policy and adopting – Or trying to rule out a policy system in an organization.

[00:24:27] JM: There's a variety of ways that our services, or our databases, or our workstations, our laptops, can interface with the policy engine in order to get permission or to check permissions. What's the deployment model for a policy engine? Do we want to just embed a library in our code? Do we want to have a sidecar that's running alongside all of our services? Do we want to have a standalone service that's a monolithic way? Give me an understanding for how we want to interface with this policy engine.

[00:25:07] TS: Yeah, that's a good question. I think that in the past, some of the policy systems in the past have been kind of very centralized and you want that for managing policy. You don't want to have to go and update policies in 15 different places whenever the requirements change. You want a single place to manage it.

But when it actually comes to enforcing policy, you don't want to have to couple every single microservice in your organization to a single internal centralized service. There are a bunch of reasons why you don't want to do that. Two important ones are performance and availability. What Netflix does and what a lot of companies do is they think about enforcing policy in a decentralized way, in a distributed way.

Boils down to, is typically likely was then to having a host level daemon or agent that runs on every single machine let's say, or runs as a sidecar next every service. Then the service can talk

to it basically over a local host, or over a UNIX domain socket or something like that. So the result of doing that is that you pay less cost in terms of latency when you need to make a decision, and you don't impact availability obviously as much, because you're not reaching out over the network every time you need to make a policy decision. I think in larger organizations that have moved to like a microservice-based architecture, that's particularly important, because you might four or five different services involved in the request chain to service a higher level application request.

If at every hop in the chain you need to call out over the network and wait for a decision to come back, you're going to pay the price and terms of latency and availability. So I think the model that we're seeing people adopt more and more is a distributed model where the policy enforcement and the rendering of decisions happens out at the edge, but then you have this like centralized service that provides a portal effectively for people to author policy and see what policies are deployed and see how they are performing and see those kinds of things, to audit them and such.

[00:27:07] JM: I think what you're describing is, and this might relate to the open policy agent architecture, is you've got your service that has some kind of policies associated with it, and that service, maybe fi we're talking about Kubernetes, it's running in a Kubernetes container or a Docker container and there's a sidecar node that has the policies that are associated with this service. Maybe it's embedded in a small database, a small little cache that every time a request is made to this service, the service can make a quick check in that sidecar to see is it okay if this access happens. Does this person – Does this API even have – Or does this service that's contacting me, does it even have the permissions, the correct role to be accessing this service? Because if not, then no, I'm not going to accept this request. But if yes, then yeah, sure everything's fine, but you don't want to have to make an additional network call to a far network call, an expensive network call, a highly latent network call to some other service. It's convenient to have this sidecar that sitting there in another container just right next to your container and very quickly get the policies. But if you need a larger comprehensive set of policies, you can always reach out to the centralized database that contains all the policies. Probably your sidecar policy cache is only going to contain the policies that are relevant to your service at the last time that you warmed the cache.

[00:28:43] TS: Yeah, and that's the right way to think about it. It's much better if in order to render a policy decision, if that decision can be made locally. It's just much simpler in terms of operations, and it's also simpler when you talking about testing. If you want to test the policy offline, it's nice if the policy is kind of self-contained. If all the information required to make a policy division can be encoded in the policy and encoded in the state that follows the policy around, then it's much easier to test it and kind of analyze it offline.

Obviously, there are cases where that's not feasible. You can't, for example, replicate know the entire LDAP database to every node in the system. So there are cases where having the ability to execute a call out from the policy perhaps in a kind of a dynamic manner makes sense, but that's sort of like the – That's not the norm. The goal is to kind of keep all the evaluation local and then that's just kind of like a special case, is when you need to make a call.

[00:29:42] JM: So what's the process of propagating policy through a system? If we have the sidecar model – So I've got a service, for example, that people within the organization can ping this service and say, "What are the salaries of my direct reports?" or they can say, "Here is a person. I want to know their salary," and the policy agent would have to permission them to be able to check if they have the rights to knowing this other person in the organization's salary, this Netflix example that we gave earlier. What's the process by which that check, that policy check, is made? What's the process of deploying changes to the policy? What if somebody gets promoted? You've got to update the policies throughout the system. Give me an overview for how this works and how policy propagates through the system.

[00:30:35] TS: Yes. So it's important the keep in mind that there's policy, kind of the rules, the logic if you will that needs to be enforced. Then there's sort of like the data, the context that's required for those policy decisions. So things like who reports to whom or the groups in the organization, we typically think of that as context, as data, that needs to get loaded with the rules, with the logic.

So when it comes to making that available to the policy engine or to any policy agent, there're kind of different models that are available. So it's standard stuff, like do you need pull-based – Rather, can you start with kind of like a pull-based approach, which is usually simpler to deploy, a more reliable, or do you need like a push-based approach, which can be more performant,

can allow you to do kind of like lower latency updates? Policy into context but comes with kind of additional cost in terms of complexity? There're different tradeoffs there when you start talking about how policy distribution should happen.

Usually the answer is that like a hybrid approach is what you want to go with. You want for 80% of your use cases a sort of a simple pole-based approach works well enough, but then there are special cases where you need to kind of distribute policy much quicker or you need to distribute context much quicker, or when the context changes, it needs to be propagated very quickly. So then the pull-based approach doesn't work as well and you need to think about it a little bit differently.

There's different kind of like architectural decisions that you need to make when you're looking at that, and that's where you start getting into this idea of having a control plane for policy. So having something that allows you to both control the policies that are in place that are distributed in the system and being enforced, as well as collect telemetry basically on those policies. When was the last time that the policy engine got its policy? What version of the policy is it running with? How is it performing? As well as just like having an audit log of the decisions, for example, that have been rendered. So those are kind of like a separate set of concerns that you get into when you're managing this kind of thing at scale.

[00:32:42] JM: We've been talking about policy engines in the abstract. You work on Open Policy Agent. What is Open Policy Agent?

[00:32:49] TS: The Open Policy Agent is it's an implementation of these ideas. It's an open-source general-purpose policy engine where we're hosted by the CNCF is a sandbox-level project since earlier this year. So we restarted the project a couple of years ago, like two years ago, a little over two years ago. The goal really was to create a new policy engine that could be used to enforce a variety of different kinds of policies across the stack and it was sort of built with what at the time wasn't called cloud native, but really was kind of like a cloud native set of principles. We wanted it to work well in a container-based environment. We wanted it to work well in large scale kind of distributed systems environments, and we wanted it to have first-class support for very complex kind of deeply nested data model.

When you look at modern APIs and sources of context today, it's all document – Or it's like JSON data effectively. So JSON is characterized by these kind of like deeply nested objects are maps and arrays and so on. So what we wanted was a policy engine that was capable of understanding that kind of like rich contextual data that's often represented as JSON. We built out the kind of core of the policy engine against that and then since then we've been kind of taking it and applying it to different systems and seeing and solving actual policy enforcement problems.

[00:34:15] JM: Let's give a little of the case study. How does Netflix use Open Policy Agent?

[00:34:20] TS: Yes, that's a great example. They were one of the earlier adopters of the project, and for them, basically, they have a security platform team that's responsible for engineering security sort of infrastructure for the organization. So they're concerned about securing access to basically the internal resources in Netflix. So HR systems, payment systems, batch jobs, sensitive data and the like, and basically the internal resources that make up the Netflix backend. So they have a number of different use cases involving things like microservice API authorization, access control over pub/sub systems like Kafka, access control over who can just SSH into a box and so on.

They have a number of different use cases where they need to enforce access control or authorization. I think what they found was that the architecture they were building, the Open Policy Agent lent itself very well to that. So they have this kind of distributed enforcement model where they have effectively an agent or a process running on every machine, and that process embeds the open policy agent as a library. Then whenever services on that machine need to make a decision, whether or not to allow an API request, they call out to that agent and they get a decision back.

That's sort of it at a high-level. They obviously have this kind of like more centralized management system as well in place that's not part of OPA, that's their own internal proprietary system. But that allows them to manage the policies that are deployed. It allows the audit policies that are deployed. It allows them expose a portal to their end-users so that they – Rather, to users inside the company that have to define policy and so on.

For them, OPA really filled this gap of providing a kind of a building block, a reusable building block, a library if you will to express the policies with and to evaluate them. So there's not a lot of value in most companies, like reinventing the way that you express the policy or implementing the evaluation engine that takes those policies and kind of answer queries against them. So they recognize that and they leverage the Open Policy Agent there.

[00:36:28] JM: Right. You're concerned with the agent, which is the sidecar cache that's answering questions for your microservice. You're not reinventing the centralized policy engine from which this agent is taking the cache data, right?

[00:36:45] TS: Exactly. Yeah. So the Open Policy Agent is really focused on providing people with kind of like a reusable language that they can use to express policy in and then kind of like the infrastructure that you need to get that deployed in your system, which is at the core. It's the kind of like evaluation engine, the thing that you feed with policies and then you can ask questions against. You can ask, "Should this request be allowed?" and then it will return an answer of, in that case, allow or deny or true or false, but you can ask other kinds of questions as well. But that's just the simple example. That's what the Open Policy Agent really focuses on, is just teams, companies, organizations with that reusable building block that there's not a lot of value in kind of reinventing all over the place.

[00:37:27] JM: What are the most difficult engineering problems in working on Open Policy Agent?

[00:37:31] TS: The Open Policy Agent is used for a number of different use cases. Like I mentioned a couple, like API authorization is one like in a microservice environment. But it's also used for doing things like enforcing invariants over infrastructure. So putting rules in place that prevent people from deploying bad container images in Kubernetes communities or exposing services in a test environment on the public internet let's say. It's used for a number of different use cases.

So there are interesting challenges that come up there. You need to design a language such that it's not too tightly – It's not tightly coupled to any of those particular use cases, but then still remains kind of like usable and approachable for people that maybe aren't spending their entire

day thinking about policy and thinking about writing policies. That's one challenge, is like how you craft a language that is expressive enough for all these different use cases, but also easy to use and easy to ramp up on.

On the more kind of like technical side, a lot of the use cases where OPA is applied have a fairly tight performance constraints associated with them. It's not at the point where we're suggesting that people run OPA in the data plane of their network, and we wouldn't do that in any case. But certainly in the microservice API authorization use case or in a pub/sub use case, the engine has to be capable of rendering decisions and certainly on the order of like about a millisecond, but it's much nicer if you can kind of treat it as something that doesn't cost anything. So then you want to have the policy decision rendered as quickly as possible in the order of like a few microseconds or a hundred microseconds or something like that. So we pay a lot of attention to performance and latency, scalability of how the policies and data are represented in the engine. So there's a bunch of optimizations that we've put in place to kind of address that, and we've got a blog post about that online if people are interested, but that's another challenge.

[SPONSOR MESSAGE]

[00:39:31] JM: DigitalOcean is a reliable, easy to use cloud provider. I've used DigitalOcean for years whenever I want to get an application off the ground quickly, and I've always loved the focus on user experience, the great documentation and the simple user interface. More and more people are finding out about DigitalOcean and realizing that DigitalOcean is perfect for their application workloads.

This year, DigitalOcean is making that even easier with new node types. A $15 flexible droplet that can mix and match different configurations of CPU and RAM to get the perfect amount of resources for your application. There are also CPU optimized droplets, perfect for highly active frontend servers or CICD workloads, and running on the cloud can get expensive, which is why DigitalOcean makes it easy to choose the right size instance. The prices on standard instances have gone down too. You can check out all their new deals by going to do.co/sedaily, and as a bonus to our listeners, you will get $100 in credit to use over 60 days. That's a lot of money to experiment with. You can make a hundred dollars go pretty far on DigitalOcean. You can use the credit for hosting, or infrastructure, and that includes load balancers, object storage.

DigitalOcean Spaces is a great new product that provides object storage, of course, computation.

Get your free $100 credit at do.co/sedaily, and thanks to DigitalOcean for being a sponsor. The cofounder of DigitalOcean, Moisey Uretsky, was one of the first people I interviewed, and his interview was really inspirational for me. So I've always thought of DigitalOcean as a pretty inspirational company. So thank you, DigitalOcean.

[INTERVIEW CONTINUED]

[00:41:39] JM: I've got my salary checker service, and as we said, every time a request comes in, there's going to be a check in the policy agent to make sure that this request gets approved, and that consists of accessing that sidecar, looking in the cache and the language through which these policies are expressed is a domain specific language for creating and evaluating policies. You're saying that there is performance demands, because obviously if you're going to add a check to policies for every single request that comes in, you're going to really have to – If you have a latency-specific service, that's a high bar to be able to not degrade the latency specifically. If you're going to be deploying this to every single service potentially an organization, then you've got a really high bar. You've got the high constraints.

Now, on the bright side, it seems like you don't really have to solve problems like – I mean, tell me if I'm wrong, but like things like network partitions or node failures or the cash getting too big. It seems like these things are not really in your purview as much. The problem is more like you need to be very responsive to request. You need to be able to evaluate request, API requests really quickly, and that requires building a super performant domain-specific language and I guess parsing engine and whatnot.

[00:43:12] TS: Yeah. Yeah. We spent a lot of time working on things that help. For example, we often can kind of pre-evaluate large portions of the policy. When it comes to actually enforcing the policy, maybe it's already done a bunch of the work to pre-compute portions of the policy that are known ahead of time. That's one example of like an optimization that OPA does.

It also does things like build indices for the rules that you give it. So that means that basically when you ask OPA for a policy decision and you provide a bunch of input data, instead of having to kind of scan through all the rules linearly, let's say, OPA can kind of just look at the index and then jump to the specific rules that need to be evaluated for that input. So that'll provide like an asymptotic speed up in certain cases.

We kind of invest putting the right data structures and algorithms in place to ensure that OPA is scalable to a great extent as possible. Then there are obviously a bunch of kind of like micro optimizations that make and we try to make to do things like avoiding copying data around unnecessarily during evaluation and so on. That's kind of one area.

At the same time, if you start talking about enforcing policies that depend on large amounts of contexts that you can't fit into memory, that you can't fit into RAM on a single node, then you have to kind of go in a different direction. You can't necessarily like replicate all the context that you need, say, out of like a SQL database into OPA and then ask it, "Should this request be allowed?"

In those circumstances, in those scenarios, we have a different approach where basically instead of moving all the data that's required to make a policy decision into OPA and then evaluating the policy inside of OPA, we take the policy and we translate into, say, like a SQL query or something like that that can be pushed down to where the data lives. Instead of moving the data to where the policy is, we move the policy to where the data is and have it enforce it at the data level. They're all kinds of different tricks you can play based on the use cases effectively.

[00:45:10] JM: You hinted at a use case with service mesh earlier, or maybe you were saying that that's kind of another thing that's kind of in the same space as SPIFFE and SPIRE. Service mesh is another thing that gets often deployed as a sidecar that sits alongside this service in and helps with things like retry logic and circuit breaking and feature flagging. Service mesh is this kind of the wildcard of different purposes that a service can use through. Is Open Policy Agency something you would want to deploy in that same sidecar or would you deploy it as a separate sidecar?

[00:45:51] TS: I mean, people have different opinions about how that works. Obviously, you don't want to have a separate sidecar for every kind of function or capability in the system, because suddenly you're going to have your application process and then a dozen sidecars running next to. I think that in the service mesh case, having a sidecar model is very useful. So we see people running OPA as a sidecar, but when you start looking at projects like Envoy, for example, that's a very popular service mesh, or service proxy, we're starting to see a lot of different kinds of programmability features going into Envoy and other service proxies.

You can imagine having your policies effectively compiled down to a format understood by the service proxy and enforced kind of like in-line inside of the service proxy without having to require like a callout to another sidecar. That's a much more kind of like – It's sort of harder decision to make and it depends really on your organization and who manages what. Sometimes there's one team that has to be worried about policy and security and then there's another team that's worried about like, say, just service conductivity. In that model, having two separate systems might be advantageous, but it's really sort of an organization-specific decision.

[00:47:04] JM: I've been covering this space a lot, and I'm very interested in the business, the go-to-market strategy of different companies and the places where there is some competitors. One area of competition that I find intriguing is the Istio and LinkerD service meshes and I don't think it's a winner take all world at all. I think it seems like there're room for both of those projects to be extraordinarily successful. What are the design differences between those two service mesh systems?

[00:47:40] TS: I wouldn't call myself like a service mesh expert. I think that they have similar designs to a large extent. They rely on a service proxy to handle the data plane and then they rely on a centralized control plane to manage the routing rules and all of that as well as things like identity, and then also collect telemetry and help you kind of like understand and observe the system.

From that respect they're very similar. I think that Istio is solving a lot of problems and it's trying to go inside of just Kubernetes, like you could use it outside of Kubernetes. I think that LinkerD is more focused now on Kubernetes specifically, and there are advantages to both. Being able

to focus on one platform means that you can kind of provide a tailored sort of user experience for that platform, which I think a lot of people would appreciate. At the same time, not everything is going to run Kubernetes or will in the near future at least to run on Kubernetes. So having the ability to connect multiple different systems together with something like Istio is compelling.

From a policy point of view, I think that service meshes are great. They help you offload, like you said, different functions from services, like rate limiting and feature flagging and authorization and authorization between those services. But at the end of the day, there are going to be application level policies that need to be enforced in the services that are running on top of the mesh. Those application level policies, you can't really solve those with like a simple protocol level or request header level policy system. As soon as you want to reach into the message payload in the request, or you want to do anything based on like the data that a service is returning, then the service meshes don't really address that kind of a problem. So that's one angle.

The other angle that's a popular use for OPA, as well as other policy engines, is just enforcing invariance over infrastructure. Again, part of the challenge here is that you don't want to end up having like N policy systems in place, because then you lose the ability to control and understand what's going on. So I think there's definitely room for something that's like outside of the service mesh that helps you manage policy. Then maybe it leverages features provided by the service mesh to do the actual enforcement. But from a management point of view, you don't necessarily want to rely entirely on like a service mesh policy system.

[00:50:03] JM: Speaking of business, you are working on Styro, which is the company you're building related to OPA. What is Styro? What's your plan for the business?

[00:50:13] TS: Yeah. We're one of the main kind of backers of the open policy agent today. We kind of are the main contributors today. We also provide helping large enterprises move workloads to the cloud, and a big part of that is helping them solve authorization across a number of different systems as well as helping them move internal systems up to the cloud. The big part of that is solving authorization there.

I can't say too much about what we're doing at Styro just because we're still in stealth. Generally, we see that large organizations, large enterprises have need to unify and control authorization across multiple disparate systems, like microservices, like SSH, like public cloud, Kubernetes and so on. So we see an opportunity to help them do that using open-source technology like the Open Policy Agent and son on.

[00:51:03] JM: What the exciting areas in this space, in the cloud and native space, right now that are interesting to you or exciting to you?

[00:51:12] TS: For me, right now, the thing I'm most excited about is WebAssembly. I'm not sure how familiar you are –

[00:51:17] JM: Oh, yeah! We did four or five shows about it a while ago.

[00:51:21] TS: Oh, great!

[00:51:22] JM: I'm surprised to hear you say that, but really interested to hear why.

[00:51:26] TS: As you know, WebAssembly is this kind of standard instruction set that's being kind of pushed by large software companies like Google and Mozilla and others. So what it provides is kind of like an open platform to run code on. There's a lot of interest in taking arbitrary code that you've written maybe for a backend or in a language other than JavaScript and running it in the browser. That's the main kind of like thing that you hear about WebAssembly today is taking codes, say, written in Go and running it in the browser. That's really cool. But within the WebAssembly community and in general, there's a lot of interest in using WebAssembly outside of the browser using it in the backend, using it in the datacenter. So there's a lot of interesting applications for it.

Essentially, it provides a safe, efficient, portable runtime for a code. So whenever you have this need for programmability or extensibility in a system, that's a really nice – Those are nice properties, and WebAssembly is this kind of nice open standard that has those properties.

We're starting to see a lot of different players kind of start to announce support for WebAssembly, like, recently, Cloudflare, for example, and now it's the ability to run WebAssembly on their workers, in their different – They've got 150 something points of presence around the globe. So you can imagine taking applications and compiling them down to WebAssembly and then running them at the edge. I think people are also talking about using WebAssembly for serverless applications. There's a lot of interest, I think, in the block chain community around WebAssembly. So there's a lot of different kind of folks that have all become kind of interested WebAssembly and interested in using it to provide greater kind of programmability and flexibility in backend or whatever you want to call it, backend kind of software.

Exciting part for me about this is that because you can kind of run code or logic on WebAssembly, it provides kind of like a platform for policy enforcement. Today, with OPA, the Open Policy Agent, it's written Go. So that provides kind of like a barrier for people that want to embed it in a library and non-Go languages. You can run it as a daemon, obviously. It's an agent. You can run it as a daemon, but that comes with its own set of challenges. So WebAssembly is kind of like an exciting or it's an interesting target for compiling policies down into and using for enforcement.

The idea is that you can basically write your policies in OPA's language and then we give you a compiler that takes those policies and compiles them down into a native WebAssembly. We released an initial version of that a few weeks ago, and the plan now over the next few months is to kind of extend the fragment of the language that is supported by the compiler so that you kind of have complete coverage [inaudible 00:54:02], the policy language in this WebAssembly compiler.

[00:54:05] JM: To boil your enthusiasm about WebAssembly down to a few distinct points, so one of them is the interoperability, or the fact that you could use Rust or C++ in the browser, which is fantastic. Another side of it I think is WebAssembly is an abstraction that restricts the resources that a program has access to, right? Is it a memory-constrained environment and it also has security constraints as well?

[00:54:41] TS: Yeah. Yeah, exactly. You can verify statically that the memory accesses that are being made by the program are safe. So you can kind of barring certain kinds of attacks and you can kind of trust that that thing is going to run safely. You can also put limits on how much memory can be consumed, which is important when you're talking about something like a policy query. It's kind of easy to kind of constrain it in the host environment, and it's fairly easy to embed. We're starting to see language, or rather WebAssembly runtimes pop up in different languages like Go and other languages. You could embed V8 or you could use a native implementation of it. Yeah, it basically enables deep programmability in all of these different backend systems.

[00:55:25] JM: Very interesting. Well, Torin, it was great talking. I'm excited about OPA and very interested in your WebAssembly. What else do you think is going to change due to WebAssembly? When I did these shows, I was like, "This technology is blowing my mind. I don't understand how is this going to change things." It's very hard to predict. It's also hard for me to predict how long it's going to take to rollout. I mean, are w headed for a world which just everybody using Chromebooks?

[00:55:51] TS: Maybe. That would be interesting.

[00:55:52] JM: But I guess it actually doesn't matter, because even if you're using a Mac OS, you could just have all the applications run in WebAssembly and they would feel like non-browser.

[00:56:01] TS: Yeah. I mean, a lot of the goals are similar to Java, or the JVM. It was like you could write code once and they would run anywhere.

[00:56:07] JM: Of kind of Docker, Docker in some ways too.

[00:56:08] TS: One of the differences here – Yeah, like Docker. Some of the difference here is that the WebAssembly spec is very well specified. It's not very complicated right now. Hopefully it stays that way. But I think that one of the – One of the exciting things right now within WebAssembly is that you see WebAssembly support popping up in all of these different places kind of at the same time. You're seeing like CDN's are supporting it.

[00:56:34] JM: Crypto.

[00:56:35] TS: You're seeing crypto people getting into it. You're seeing all these languages adding backends, like Rust and Go. They've kind of really just jumped ahead and added a support for WebAssembly, like C compilers, like LLVM and so on. They have like really good support for WebAssembly. I can't think of another kind of – I have a hard time coming up with another example of technology that's being like widely adopted as quickly as WebAssembly is in the kind of backend infrastructure space. So I think it's really surprising.

[00:57:07] JM: What's funny about it is kind of an apolitical technology, because nobody could really productize it. It's more like everybody productizes it all at once.

[00:57:15] TS: Yup. Yeah, everybody wins, I guess. Maybe that's the secret to this stuff.

[00:57:21] JM: Interesting. Okay, Torin. Well, thanks for coming on the show. It's been really great talking.

[00:57:24] TS: My pleasure. Yeah, great talking. Take care.

[END OF INTERVIEW]

To find out more about continuous delivery and GoCD, check out gocd.org/sedaily. GoCD has been a sponsor of software engineering daily for more than two years and we’re proud to have their support. If you're getting started with continuous delivery yourself, check out gocd.org/sedaily.

[END]