EPISODE 771

[INTRODUCTION]

[00:00:00] JM: The nature of software projects is changing. Projects are using a wider variety of cloud providers and SaaS tools. Projects are broken up into more Git repositories, and the code in those repositories is being deployed into smaller microservices. With the increased number of tools and repositories and deployment targets, it can become difficult to manage software policy. Policy defines how different parts of an application can behave. Which parts of your application can access a given Amazon S3 bucket? Which parts of your application can communicate with the authentication microservice? Which developers are allowed to push a new build to production?

Shimon Tolts is the CTO and cofounder of Datree, a platform for policy enforcement and code compliance. Shimon joins the show to talk about continuous delivery, configuration management and policy enforcement. He also explains the motivation for his company, Datree, which performs analysis across a user's GitHub repository to map the committers, the code components and repositories.

Datree is quite an interesting platform and represents a lot of trends in the world of software engineering that we've covered in previous episodes, and I hope you enjoy this episode.

[SPONSOR MESSAGE]

[00:01:27] JM: Every developer uses open source dependencies in every software project. Whether you know it or not, your tools are automatically pulling in thousands of license obligations. You're also pulling in dependencies and potential vulnerabilities. This happens whenever you use a new component or simply run a build.

FOSSA is the tool for managing your open source dependencies at scale. FOSSA automatically scans your dependencies for license violations and vulnerabilities to prevent issues at build time and automate compliance obligations at release. Over 8,000 open source projects and

engineering teams at companies like Twitter, Docker and HashiCorp rely on FOSSA daily to manage their open source licenses and dependencies.

Get a free scan by going to go.fossa.com/sedaily. That's F-O-S-S-A, FOSSA. You can also check at the show we did with Kevin Wang, the founder of FOSSA, and hear how FOSSA can reduce your open source risks. If your company is good at developing software, then it should also be good at managing open source. Check out go.fossa.com/sedaily and learn how FOSSA can improve your codebase.

[INTERVIEW]

[00:02:57] JM: Shimon Tolts, you are the CTO and cofounder of Datree. Welcome to Software Engineering Daily.

[00:03:03] ST: Thank you very much. It's very fun to be here.

[00:03:06] JM: The subject that I want to begin with is continuous delivery. We've done a bunch of shows about continuous delivery. The practices continue to evolve. The tools continue to evolve. To give people some contexts about where we are today in 2019, why did the world move towards continuous delivery?

[00:03:31] ST: So I think the world moved to continuous delivery because people want to have reproductible automated processes. So as I look at it, what happened is that we used to have our source code, and then we have everything that happens up until we deploy our application on to production, and that used to be a black hole. I can imagine a company and it happened in many companies where they go like, "Hey, guys! We have a production issue. What happened?" and they go in, talk to the developers and the developers like, "Look! We haven't committed any code in the last two days. We haven't changed anything." They're like, "Okay. So what happened?"

So they understand that there is a black hole between the code of the application and being in production, and at this point I think that everyone started actually automating all the processes in order to, one, have them reproductible in order to know what are you doing, how should you

deploy this. The second one, of course, is speed and agility and like the time it takes you to deploy into production should be the fastest possible.

[00:04:39] JM: How are team structures changing in this evolving continuous delivery world?

[00:04:47] ST: Things are changing drastically. So if we look back even 8 years ago, we used to have like the development team, the production team, the security team and everyone was like a step in the way, the release management team. Today, the developers are responsible for everything end-to-end and they write the code, they test it, they deploy. They wake up 5 in the morning if there's a problem, and they're actually also responsible for the cost of how much it costs on the cloud.

The structures I think today is that all of the departments that used to be in the way between the developers and the production are now enablers. So you see like SRE and devops teams building tools to enable different development teams work better, but they're no longer the bottlenecks that everything has to go through them.

[00:05:47] JM: We've done shows about devops and some people say that devops is a movement. It's a cultural ideology. Some people say it's a set of best practices. Some people there are actual roles that you assign to devops, like this person is doing devops, like I'm a devops engineer. Do you have a perspective on devops and whether it's a role or if it's a description of practices and what the relationship between devops and continuous delivery is?

[00:06:19] ST: So I think that at the end of the day, we need to ship software and we need it to be reliable and as fast as possible. If you look at the devops, let's say call it transition, revolution, I think that it's giving more responsibility to each person and it's working in an agile continuous way. So it's no longer, "I wrote my code. Now it's QA's problem," and QA shipped into production, into like the team that puts it into production. Then when there's a problem, they go back and say, "Guys, the software has a problem." So the QA says the developers who made the problem, but the developers say, "No. You released it into production as a QA. So it's your problem. It's your fault," and they start fighting with each other whose fault it is, when actually instead of actually going and fixing it and having ownership.

So to me devops is giving ownership to a person end-to-end in terms of how to build the software and ship it. I don't really believe in the title devops engineer. For example in my company, we only have software developers. Some of them do more frontend. Some of them do more backend. Some of them focus more on like infrastructure as code, but at the end everyone's a software developer and I expect everyone to know how the software reaches production.

This is actually what I ask people in interviews, "How did the code that you wrote at the end of the day served customers in production?" and many people go like, "Gee! I have no idea." So I think that devops is making sure that you have responsibility end-to-end on to writing and shipping the software.

[00:08:09] JM: One subject that we've covered recently that relates to what you're doing at Datree is that cloud and Kubernetes have made it much easier for developers to spin up infrastructure. In some cases it's led to infrastructure sprawl, where you have lots and lots of little pieces of infrastructure, and as the infrastructure expands, it begins to outstrip the number of developers and their visibility into that infrastructure. Have you seen this increase in infrastructure sprawl?

[00:08:48] ST: Yeah. So infrastructure sprawl can be different things, but I can see where people might spin up lots of resources in the cloud without having accountability and understanding what's going on. But I think that as you sale and as you progress, at some point you see like, "Okay, our bill is very, very high and we need to take care of this," and then companies switch to infrastructure as code, and they understand what's going on, and when that is done, it's easier to take control.

But I think that giving the ability to – On the one hand you might end up with an infrastructure sprawl, but on the other hand you have developers who are going trying different services, trying different architectures actually building stuff, and I think that it's a great enabler.

[00:09:41] JM: In this world that is changing, the continuous delivery process has become more and more pervasive, and the continuous delivery process allows a lot of bugs to be caught throughout the continuous delivery pipeline, whether it's caught in the integration test process or

some other stage in the continuous delivery pipeline. But bugs still do make it to production sometimes. How does that happen? How are bugs still making it to production?

[00:10:12] ST: I think that it's okay. If we try to reach a point where bugs never reach production, I think it's a very problematic place that will cripple us down. I think that we should constantly deploy into production and make as many changes as possible, but make them small changes. Then if you have a problem, its blast radius will be small. You can go and write a one line of code and then write 10 million lines of test code in order to test this line.

But I think that you will always have bugs in production, and that's okay, as long as you have good processes, you can revert, you can check, and I'm a strong believer that actual testing is done in production, because building a theater and staging and trying to theater your application, it's nice. You should have tests, you should have unit test, integration test, whatever test that you have, but it will never be the same as the real thing. I really believe in canary release and like really seeing maybe a future to 1% of your traffic, see how it handles and then deciding whether you want to revert that or go 100%.

[00:11:29] JM: With the emphasis on continuous delivery and many other trends, there're increasing usage of GitHub for doing so many different things in the software development process. GitHub has surpassed the expectations of almost everybody, and probably including the people who actually make GitHub. But GitHub can't be all things to all people, and sometimes it feels like the product is used for so many different things that there's not really room for the product to expand further. What are the places where GitHub is insufficient?

[00:12:08] ST: Insufficient? So I think that GitHub is doing a great job. I think that they really enabled a git to be the leader. That's it. Git has won. Git is the de facto standard. All companies are using that or switching to it or in the process. I think that GitHub is doing a great job in terms of allowing everyone to use it, allowing the community to leverage it.

More in terms of the product, I think that – And I've met with Nat, the CEO of GitHub when he was travelling around, and we talked about it. I think that their main focus is taking you and building a specific project and building the best like repository and then maybe using GitHub

actions and deploying into production. In my point of view, they're looking at it repo at a repo, like making the best experience repo by repo.

Where I think that there are still many challenges, which actually we at Datree help solve is that how do you look at all of your repositories as a composition? Let's say you have 300, or even 3,000. We have customers with thousands of repositories, and then you ask yourselves, "Okay, wait. What do I have there? How do I manage that? How do I understand who is doing what and where and how can I drive these practices and policies on top of this composition and not a repo by repo basis?"

[00:13:40] JM: You mentioned earlier that there's a benefit of what has led us to infrastructure sprawl, and that is the developers have much more autonomy in spinning up infrastructure and they feel more free. I completely agree with that as a developer myself. Now, this maps to certain issues in the world of Git, where you could potentially have sprawl in the number of Git repositories that people are managing. Do large companies have issues managing a large number of repos?

[00:14:16] ST: Yes.

[00:14:18] JM: Well, can you tell me more about those issues and how they manifest?

[00:14:21] ST: So it starts with whenever can create a repo and use it. So it starts with no one is actually responsible for the composition of all the repos. So each team has their own repos, and you start a repo for anything, right? Any POC you do, any simple line of code that you have, you just start a repo.

Then organizations are faced with hundreds and thousands of repos that they don't know who they belong to. They don't know whether are they connected to any code that's running in production or can they archive them. They don't know if they're active, not active. Do they have any vulnerable code in them? Maybe they have secret keys in them and they're just at a place where it's overwhelming. Then they find there's a problem.

I'll give you a real-world example. A company has a production outage, they post-mortem it. They say that the database went down, then they research and they found that by using – I don't know, a specific version of the driver and NPM module, version number four is incompatible with their setup of their database. So now they go like, "Okay. So we've identified this problem. Now which one is my repositories has microservices using this module? Which ones are using this version of the module? Then how am I going to make sure that no one's actually going to use this module? Am I going to send an email to all of my developers?" "Hey, guys. Please stop using version number four." Going to write the wiki page?"

Then if you look at the other side, me as a developer, am I supposed to remember those emails? Don't use this version or that version, or use and don't use this technology. It's crazy, because now I have the autonomy as a developer, but I also have the responsibility, but how am I to know what to use? So it becomes very tricky for organizations.

[00:16:24] JM: There's a term that I have done a show about called GitOps, and it's related to the release process. As I understand, GitOps ties your push to GitHub more closely to the release of code. Can you describe what GitOps is and what it represents?

[00:16:51] ST: So GitOps is a fairly new term. I think that Alexis from Weaveworks coined it, and we see ourselves as a GitOps company. If you look at the devops days, so like we moved from like all the developers have the autonomy now and the responsibility, and then they started using different tools. So they had like 30 different dashboards like for Jenkins and AWS and all of the different services that they were using. Then what they did is they actually started automating everything and everything became codified. I call it YAMLified.

Now that you have everything within your source control, let's say GitHub, you're doing your operations though pull requests. So it's GitOps operations through Git, and everything is a pull request. It's interesting, like some of our customers, we have a customer, they actually use Git and pull requests for legal. So they actually have different contracts and they open pull requests in order to do changes and propose changes and have reviewers, and they're actually using GitOps for legal. It's very interesting.

If you think about it, you can order a pizza using pull requests, right? So in my mind, GitOps world is where your repository is a representation of your production an you have your source code, you have your infrastructure as code, you have your CI/CD configuration, and everything is codified and everything is in the repository.

So now you no longer have any black holes between what you have in your code and what is running in production. In a GitOps world, you should be able to redeploy your application 100% just by the code that is within the repository.

[00:18:54] JM: What are some of the unsolved problems in typical GitOps workflows?

[00:19:00] ST: So I think that first of all you have much more code than you ever had before. Today we have more code, because we used to have people and processes doing stuff that are now codified. I call it YAMLified. Everything is a YAML file now. So now you have gatekeepers and people that would help you before you deploy something into production. Maybe you have a security review. But now it's automatic. Maybe you had someone that would go and deploy an EC2 server, but you no longer have that, because you use Terraform or CloudFormation and it will automatically spin up your resources.

If you used to have a person that would look and tell you, "Hey, you're spinning up an S3 bucket. It should be encrypted." Now you no longer have this person, because he can't be a bottleneck and manually spin up all the buckets, but now you need some kind of automatic mechanisms that will be a sort of a gatekeeper to you that will help you and do that in an automated way.

So when you open a pull request or submit code and you have a Terraform file and you create a new SQS bucket, it won't let you merge your pull request and it will tell you, "Hey! Listen, Shimon. You forgot to enable encryption on your SQS bucket, and we're a SOC 2 compliant company. So we need to have everything encrypted."

I think that this part is missing. All of the safety nets that you had with people and processes and policies that were set in place, now there is actually no way of knowing whether the policies are actually being applied or not and knowing what is the actual current status, and you have no

way of doing policy enforcement in order to help your developers. In this new world, you're kind of riding the freeway 100 miles an hour without any guidance or direction. Everything is on you.

[SPONSOR MESSAGE]

[00:21:15] 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]

[00:23:22] JM: One thing you're describing here, this YAMLification, is a push towards more declarative code, and the declarative code sits in the configuration files and it describes the state of the infrastructure that you desire. This is in contrast to much of the code that we've

been writing for a long time, or much of the code that most people have been writing, which is imperative code. It's not declarative code. It's imperative. It's, "I want to do this, and then I want to do this, and then I want to do this. For each, do this."

Declarative code is very different. Is there something about the transition from imperative code to declarative code that has taken people by surprise and people aren't exactly sure how to deal with it?

[00:24:13] ST: I think that the difference between the two, and I think that's why everyone, like why de facto we're using declarative code for infrastructure, for example, is because so called regular code is meant to execute and do things, and that's the most important thing, to do things to make them. In infrastructure as code, it's not only making sure you spin up those resources. I think that many of the things is also understanding what's going on.

So when you look at the Terraform file or a CloudFormation file, you get the sense of what's going to happen. What is it going to spin? There's like no endless resolving of resolving of resolving of different if statements and loops and stuff in order to understand what's going to happen. It's fairly simple. Spin up this EC2 instance and use this security group. Then you go and you look, "Oh! This security group is open to this IP and so on."

So I think that it's much simpler and more declarative in the way of understanding what's going to happen. It's harder to do complicated things, but I think that simple is better than complicated, and I think that if you do crazy shenanigans to spin up your infrastructure and crazy loops and ifs and crazy conditions, it would be very hard for you to understand what's going on with your infrastructure later on.

[00:25:46] JM: One part of configuration management that has grown and grown and grown recently is the configuration around Kubernetes, and the thing I hear the most in the Kubernetes configuration frustration is YAML. I've got this really long YAML file that describes a lot of stuff, and it's in YAML and I don't like the indentation. Why do people complain about YAML? What's the big problem with YAML?

[00:26:21] ST: I don't know. I think it's fine. There's YAML, there's TOML, there's many different things, and people will always complain. But I think that it's good. If you think about it, you cannot spin up an instance, like a container in Kubernetes without actually writing declarative code. Think of the paradigm shift between having to actually have a server and install it and then moving to, going to, for example, EC2 and spinning up an instance. Now, if you're using serverless, or for that matter, ECS container or Kubernetes, you cannot manually spin up a resource, you just cannot. You have to declare it and run it, and I think that this is the main indicator of the major shift that's happening. I think that it's good, because it forces us to work in a repetitive way. You no longer will be faced with logging into an EC2 and going like, "Oh my God! Why do I have 200 servers? Who spinned them up? What is going on?" and trying to understand like what is going on.

Now, everything is documented. You have a file and you know who wrote it. Hopefully you revision it in Git so you know what changes were made, who are the persons that did that. I think that that's a very good transition. Then whether it's a YAML, TOML, JSON, to me it's all the same. I don't really mind. I think that the transition itself is the most interesting part.

[00:27:52] JM: One thing that we're managing with our configuration is policy. What is policy?

[00:27:59] ST: So policy – That's a great question. It can be very different and it can be different for different organizations or persons within the organization. So you could have a policy that would say for every resource that you spin up, you should have at least two of them. That's also like SOC 2 basically, like not have one container, because it might die. Have at least two or three.

Another policy might be don't commit secrets into code, because then it will have like – You might be exposed, then everyone knows what constantly happens with like AWS secret keys and so on. Another policy might be use a specific module and a specific version. Then another policy could be you need to have at least three reviewers in order to merge your code.

I think that in general, a policy is an abstraction layer to say, "Here are the things that you should follow and do, and if you do that, we think that we will help you and you will do the right things and prevent you from – I don't know, possibly doing the wrong thing." Now you need to

think of which policies you follow? Which ones you don't? How do you follow them? How do you implement them?

[00:29:17] JM: What are some ways in which policy gets mismanaged?

[00:29:21] ST: I think that the number one thing that policy gets mismanaged is that people write different policies and then no one's accountable for them and there's no way of actually knowing whether someone's following them or not. Just writing out policies without having the visibility of whether are they being enforced or not, whether they're being followed or not. I think it's the most important thing, because just defining a policy doesn't give you anything.

[00:29:52] JM: You cofounded Datree. What is Datree?

[00:29:56] ST: So Datree is a GitOps policy enforcement solution. It allows customers to define different policies for development best practices or possibly any custom policy that they would like. Then for every change that happens on top of their Git vendor, for example, GitHub, it will automatically run against the policy engine. Then it will check whether the developer made code changes that are compliant with the policy of the organization and they can decide whether they want to block or enforce this pull request and make sure that it cannot be merged into the master branch, for example.

[00:30:39] JM: What problems does Datree solve for developers?

[00:30:44] ST: So Datree solves the – I would say synchronization and preventing mishaps in production. So at the end, like if you ask me what's the number one thing that we're proud of, what's our number one goal? So the goal is to help people prevent downtime in production, and all of our customers, it goes down to we meet with them and we go like, "Listen, we had 1, 2, 3, 4, 5 production outages. Let's codify those things. Let's make them a policy and let's make sure it doesn't happen twice again. By the way, what other built-in policies do you have that will protect me from making risky changes, mishaps in production, security issues?"

Maybe I'll explain, the way Datree works is seamless. So you just install a GitHub application, and that's it. Then we scan the entire repositories of all of your organization and we extract only

the metadata so that all of the source code packages, like NPM, Python, Ruby, C#, whatever you're using, all of the infrastructure as code, CI/CD configuration, basically anything that has a code footprint in it, we scan it. Then we build a visibility layer, which we call a catalog. Then you can actually ask yourself simple questions, like, "Who is using what and where? Which version, which package, where do I have Travis CI in my organization? Where do I have a Docker file? Where don't I have a Git ignore file? Where don't I have a code owner's file that will define who is responsible for this organization?"

Then once you have the visibility layer, the catalog, we allow you to define smart policies on top of that. So this will help you actually make sure that all of those policies that you want are automatically enforced. So then you don't need to configure anything. We will automatically, for every repository that you have, for every pull request is being open, we work with GitHub checks and we automatically trigger upon any change, and it will run according to our policy engine and we will very fast give indications to the developer whether he or she are working within the best practices and policy or not, and we can actually block a merge of a code if you so desire.

Then they will click to find out more, like what's the problem? Then they will see, "Oh! You're committing secret keys. You should fix that, or you don't have a version for a package, an NPM package that you put here." So basically every time you build this code, it's like going to the casino, because it will generate a different version every time. So you should pinpoint a version. We help the developers understand like, "What is the problem? How do I fix it?" Then they fix it, commit the changes, back to the pull request. Then the engine gives them a thumbs up and they can merge it into production.

[00:34:03] JM: Let's go in to some of those details of it. You said you extract metadata from GitHub. So when I'm onboarding with Datree, because I want better policy management and I want better config management, what metadata is there in GitHub? Can you talk in more detail about what that metadata is and what you're getting from it?

[00:34:26] ST: Yes. First of all, there are two options of deployment. One is full SaaS and the other one is like a hybrid mod, where the analyzer, the thing that actually scans your code runs within your environment, and we only send the metadata. Now, what is a metadata? So for

every Git repository, we read all of the history. We rewind all of the commits and we actually understand which person did which commit, which changes to which file? We know to understand and read different formats. For example, all of the major programming languages, so we know to extract from a package.json which modules are you using and in which versions.

We can see that, for example, we can scan all of the different files and identify whether you have a Docker file. Basically anything that has a code footprint inside your Git repository, we will send that metadata only like which package, which version, which file name and so on. We will never send your code, and which person actually did interactions with it.

Then when we send all these data from your hundreds of repositories, it will then allow us to create this tree. You will have the tree of all of your different modules and where are you using them, in which repositories, in which person is using what and where? It will have an eye bird view. So you can ask yourself, "Where am I using a MongoDB driver?" For example, Mongoose. Then you will see the information cross all of the repositories.

[00:36:12] JM: So this bird's eye view of the different tools that are used in different areas of my Git repository, why is this useful to me?

[00:36:23] ST: So this is useful to you. If we go back to the example that I gave. So let's say you know that a specific version of a module is not good for you or you're in a transition now. Let's say you want to have 100% of CI/CD within your organization. You say, "Guys, we're using CircleCI, or Travis, or Codefresh," or whatever you're using. We want that. Let's go. We're going to go 100% CI/CD. Now you're asking yourself, "So, wait. What is the coverage that we have? I don't know." So you need to go repo by repo checking whether you have a CircleCI file or not, or a Jenkins file, or whether are you using a specific package? Let's say there is a vulnerability in a package and you need to make sure that you don't use this version of a package or the package itself. So how are you to know which – Basically, it's a bill of materials. It's a catalog of all of the things that build your software. Then you can actually go and see what you have and where.

[SPONSOR MESSAGE]

[00:37:33] JM: HPE OneView is a foundation for building a software-defined data center. HPE OneView integrates compute, storage and networking resources across your data center and leverages a unified API to enable IT to manage infrastructure as code. Deploy infrastructure faster. Simplify lifecycle maintenance for your servers. Give IT the ability to deliver infrastructure to developers as a service, like the public cloud.

Go to softwareengineeringdaily.com/HPE to learn about how HPE OneView can improve your infrastructure operations. HPE OneView has easy integrations with Terraform, Kubernetes, Docker and more than 30 other infrastructure management tools. HPE OneView was recently named as CRN's Enterprise Software Product of the Year. To learn more about how HPE OneView can help you simplify your hybrid operations, go to softwareengineering daily.com/HPE to learn more and support Software Engineering Daily.

Thanks to HPE for being a sponsor of Software Engineering Daily. We appreciate the support.

[INTERVIEW CONTINUED]

[00:38:56] JM: So intuitively, I completely understand. That sounds really useful to me. Can you tell me more about the use cases that people have found to be the most relevant when, okay, you get this tree that describes the different things, the different tools that you're using in different places, and it's a tree, because everything is being used from the top node of the tree. Then as you go down into deeper and deeper branches and leaves of the tree, fewer and fewer software packages are used. So you can just kind of see where different software is used and what places. But what particular use cases, how is this actionable for me?

[00:39:37] ST: That's a great question. So it is actionable together with the policy engine. So then we have different policies, such as you shouldn't have secret keys in your code. You should have a Git ignore file. Very different – You shouldn't commit your configuration files into code and so on and on and on and on.

So when you sign in for Datree for the first time, you immediately get the current status of your organization according to those policies. So for example, the moment you sign in, we build a catalog and then we immediately know whether you have secret keys in your code or not. So

now you get this list of all of the repositories where you have certificates, spams, keys and so on.

So now you go like, “Okay. So now I’m going to open a Jira ticket and have my team take care of that ASAP.” Now, you ask yourself, “How am I to prevent this from happening again?” So now you can set a policy and make sure that it doesn’t happen again. So that’s the first use case of like knowing what’s happening currently within your organization according to the policies that you set.

The second thing is that we have smart policies. For example, because we know all of the different repositories and different modules that you have, you can set up a policy. We have a policy that if you’re introducing a new package for your organization that has been never used in your organization, it will notify it into Slack and you would possibly want to start an internal process of like verifying this package, making sure that it’s green light for being used, that it doesn’t have like a GPL license. That it is compatible with your stack.

So this is one unique thing where it’s not just policies that are if this, then that. You have the notion of what you currently have within your catalog. So you can set a policy, for example, “For my internal authentication authorization NPM module that we built, all the microservices should use up to the three latest versions. If there’s a new version, they will be notified and they will know that they need to update it,” or if I’m building an internal package and I want to depreciate a specific API or SDK call, I’m asking myself, “Wait, which other microservices are using this package?” So by using our catalog view, you can actually get the answer.

[00:42:22] JM: And just to make this a little bit clearer to people, if you are somebody who has ever deployed a project to, let’s say, AWS, and you have a code in an open source repository and you include a secret in that open source GitHub repository, like you include the access key to an S3 bucket. That’s a vulnerability, because if anybody saw that GitHub repo, they could just take your key and start spanning your S3 bucket with all kinds of stuff and start just using your storage and charging you for it.

[00:43:07] ST: And that's the good scenario. The bad scenario is that is your root key and they lock you out of your own account and start extorting you, and this happened. You can Google that.

[00:43:17] JM: That happened on AWS? That kind of thing happens on AWS?

[00:43:20] ST: Yeah. They lock you out of your account and, yeah, they start blackmailing you. If you don't pay us, we're going to delete your account. True story.

[00:43:29] JM: In that scenario, not to take things off topic. In that scenario, can't you just go to AWS customer service and be like, "Hey, my account got taken over?"

[00:43:39] ST: Yeah, but it's a time sensitive issue, right?

[00:43:41] JM: That's true.

[00:43:42] ST: They can also start just publish your code open source. They can the list of your customers. They can make changes to things. It's a very nasty thing. It's very dangerous. There is a company that got wiped from the earth because it happened to them, and it actually – It closed the company, because their backups were in their same AWS account, only in a different region. Then when they got locked out and they didn't want to pay, they just deleted their entire account and their backups.

[00:44:14] JM: Was that like two or three years ago? I think I remember that company.

[00:44:16] ST: Yeah. It was Code something.

[00:44:19] JM: Yeah. Yeah, that was a scary thing. I remember reading a TechCrunch article and I was like, "Seriously? Wow!" Anyway, that's the stakes that we're talking about here. That's why even if we're just talking about the level of enforcing secret management, this is pretty important.

Now, I want to understand better why this is a hard problem, because if AWS is able to just throw on some feature where they scan repositories for keys, and I've gotten automated emails from AWS before that are like, "Hey, you've got an S3 thing exposed in your – An S3 key exposed in your GitHub repository. You need to clean that up." If Amazon is able to do this, why do we need a solution from you, from Datree, to scan all of my stuff for all these different kinds of secret management?

[00:45:16] ST: Yeah. So secret management is only one example, right? Again, secret management is not just your AWS keys. It's maybe your certificate, your SSL certificate. It may be anything that you have in order to access different resources. That's just one example that is very clear and simple to understand. You shouldn't have your secrets exposed, because then bad people can use that and make things happen. I think this is the number one. We have 100% of our customers using this policy, because everyone understands that having secrets in your code is very, very dangerous.

But I think that the main value comes when usually people who are in the devops role or SRE or operations, they come to us and they say, "Listen, we have different best practices and guidance that we want to propagate and implement within our organization, but it's nearly impossible." Even if the developers agree with them and they want to do them, they don't always remember to follow them, and it's not always feasible to do that. So by defining centralized policies, it's the first time where you can have a centralized management for all of your Git repositories in one place.

[00:46:38] JM: One benefit I can see from that is overtime you get maybe network effects. Tell me if I'm wrong about this, but you get network effects where people are introducing policies like don't allow – I've got secrets for this obscure piece of SaaS software, and I want to enforce the policy around that piece of obscure SaaS software. If somebody else creates a policy, then when I start using Datree, if Datree scans my metadata, Datree can see if I am using that obscure piece of SaaS software, and now it can say, "Okay. What are the other policies people have made on the Datree platform? Let's suggest these as policies that maybe you should have on your repo."

[00:47:25] ST: I totally agree. This is totally right, and this is something that we're working on. We're working on some of the open source play now, where anyone can share and like have their own policies and best practices, and we will be just the engine to do that. Actually, we're just doing all of the heavy lifting and understanding like mapping everything and cataloging it and working with all of the Git vendors APIs, which has no interests to any company. This is now any company's business. But it's everyone's business, like the cherry on the top to write the policy itself.

So by having customers share different policies and by having our engine saying, "Hey, you're starting to use Kubernetes. Would you be interested in this policy pack that has the Top 10 Best Practices for Kubernetes?" It's like, "Yeah! Of course, I want that." We're actually working with different vendors in order to build those best practices packages, like best practice pack, rule pack for serverless, for Kubernetes, for SOC 2 security. That's like one side that we're focusing on. The other side is making the platform as customizable as possible, because we have all kind – You open it to customers. They do different crazy things that you wouldn't think of, and it starts from very simple things, like I want to enforce the branch name. I want to enforce the commit message, or if this happens according to something else – Like they have their own internal policies and things.

Sometimes also companies have complicated processes internally that they just want to customize for their own needs that might be not relevant to everyone, but are very, very unique and important to them.

[00:49:14] JM: I think this is interesting, because this is a kind of business that is built on top of GitHub, and obviously there are other businesses that have been built on top of GitHub, but I think this is probably the first one that I've done a show on, where it's like this is kind of using GitHub as a platform to build an entire other platform on top of. Because I can imagine, there's probably a lot of potential that you could potentially get from doing this kind of code. I mean, there have been code, like static analysis businesses. What are the other market adjacencies that you could expand into since you have this tree of information about repositories?

[00:49:59] ST: So the way I see it is we just happen to be integrated on top of GitHub now, and we are also – We already have early version with Bitbucket and GitLab. The way I see it is that if

you take a time machine 10 years ago and you ask a software company, “Would you like all of your best practices and policies to be automatically enforced whenever developer makes a change?” Of course they would say yes. It’s not a new problem. Everyone wants that.

The problem was that everything was scattered. Nothing was codified. Many of the things were in people’s heads and were done manually by people’s hands. So it was just not feasible to do. Today, we scan code and we build the catalog and we build the enforcement engine. So it just happened to be that all of the code is nowadays in Git and it just happens to be that GitHub is one of the most powerful Git vendors. If you think about it, we could download code from AWS Lambda, which sits there in a zip file and do the same thing for all of your AWS Lambdas. For us, it doesn’t matter.

[00:51:14] JM: I’m not sure I understand that. Can you clarify that more, that example?

[00:51:19] ST: So today, what’s important for us is to understand what’s going on within your code. So it just happens to be that all of the code is today mainly in Git. But if you think about it, when we catalog all the code and everything that happens there, one example could be that we could go to AWS Lambda and all of the code there is in a zip file and we can download it from the Lambda and we could actually tap into CloudWatch and see CloudTrail and see how uploaded it and so on, and we could build the same catalog based on zip and code that is in AWS Lambda and IAM roles that different people use in order to upload it. It just happens to be that everyone’s in Git and –

[00:52:14] JM: I see. I see what you’re saying. So as soon as that code is going to execute, you want it to execute in a way that takes into account the policies that you’re including. Whether those policies are related to security, or they’re related to hitting certain endpoints, or they’re related to rate limiting. You’re talking about just a policy management engine platform.

[00:52:37] ST: Yeah. If you think of policy, number one, identifying and understanding what is the current status. Number two is applying enforcement and gatekeeping, what goes in, what goes out. Number three, by the way, if you look at the classical security case, is remediation, right? This is what – Those are the three main pillars.

Now, we don't remediate per se, but the next thing that we can go on is like we call it maybe automation or actions. So we scan everything. We see that you have those – For example, we talk a lot about the secret keys example. So we see the keys. So number one, we put a policy that you don't commit anymore keys, and now we're automatically open pull requests to remove those keys for you and to suggest you to use new ones. It's like an existing paradigm only on the new era.

[00:53:33] JM: Speaking of new era and policy management, we did a show fairly recently about Spiffe and Spire, and we also did a show about open policy agent, and these are the open source tools for doing policy management within Kubernetes. I think this is a kind of different type of problem, because I think a lot of what they're doing there is – There are policies that you can write around the secret management and stuff, and I think you could do it with Spiffe and Spire and open policy agent if you wanted to. But those are – I feel like those are more related to like access management. Is this service allowed to access this other service? Is this user allowed to access the services in this thing? As supposed to like secret management. But I'm just wondering in the Venn diagram of responsibilities, where does Datree overlap with Spifee and Spire and open policy agent?

[00:54:36] ST: So I think that we're complementary to one another. We're a swift lift company. So we try to be as close as possible to the developer. Other companies also like Fugue, other like policy engines of shift and like Shift Inspect or Hashicorp, where they put on policies on the runtime itself and they try to enforce and do things on the runtime, on the, let's say, cloud assets themselves.

But if you think about it, if everything is shifting into code and everything is going to be managed through Git, then anything that you do will start and end from Git. You're not going to twist and do anything within your AWS account. Many of our customers, their AWS console is locked or maybe view only mode, and you cannot create new resources.

As I see it, we focus more on the code level and we also provide policies that are more also like cultural or more towards a programming best practices and not only the runtime, but it also talks about runtime. I'll give you an example. One company that we spoke with, they're using Chef, and one of the developers made a change to the Chef recipe and then it opened a pull request

and they missed it. The pull request that they removed one line and this line that he removed was the security agent that was installed on all of the machines.

Now, what Ansible and Shift and Puppet and so on, they make sure that your infrastructure is up-to-date with your recipe. So what actually happened is that it went and removed the security agent, uninstalled it from production. Then the security team is like horrified, like, “Oh my God! What are we going to do now?” because even though the developers don’t have access to production, everything goes from code. If we don’t have any policy in place, where if you’re attaching this security agent, it should be approved by security team. Then they have no control. That they have nothing to do. What can they do?

[00:56:54] JM: Well, Shimon, it’s been really interesting talking about all these different subjects, particularly policy management and the idea of a bird’s eye view of GitHub repositories. Is there anything you want to close off with? What’s in the future? What else do you want to build and what would you like the audience to know?

[00:57:13] ST: So I think that one thing that every listener should ask himself is, “Do I know how my code reaches production?” I think that it’s a very important lesson and very important like test to do to yourself whether how are you really involved and are you really transitioning into this new world where the developer is responsible end-to-end and knows how everything works?

Of course, for us, we just see more and more demand for what we do, because people actually fact those problems and they have no idea how to manage all of their Git repositories. Of course, you might build unmaintainable scripts and you might try to tackle it by yourself. But for us, we actually want to enable customers to focus on their business and just to use our engine.

So going forward, we’re going to introduce support for more Git vendors and more infrastructure as code languages, and we’d love to hear feedback from people and they can just sign in into like Datree.io and sign in with their GitHub and they will immediately get like a report of what is their status. Are their GitHub repositories, let’s say, healthy or not in a way? I’m always happy to hear feedback, and if they have any questions, they can reach out. My email is shimon@datree.io. I’m here in San Francisco and I’m happy to chat.

[00:58:47] JM: Shimon, thank you for coming on the show. It's been really fun talking.

[00:58:50] ST: Thank you very much for having me. It was very interesting.

[END OF INTERVIEW]

[00:58:57] JM: GoCD is a continuous delivery tool created by ThoughtWorks. It's open source and free to use, and GoCD has all the features you need for continuous delivery. Model your deployment pipelines without installing any plug-ins. Use the value stream map to visualize your end-to-end workflow, and if you use Kubernetes, GoCD is a natural fit to add continuous delivery to your project.

With GoCD running on Kubernetes, you define your build workflow and let GoCD provision and scale your infrastructure on-the-fly. GoCD agents use Kubernetes to scale as needed. Check out gocd.org/sedaily and learn about how you can get started. GoCD was built with the learnings of the ThoughtWorks engineering team who have talked about building the product in previous episodes of Software Engineering Daily, and it's great to see the continued progress on GoCD with the new Kubernetes integrations. You can check it out for yourself at gocd.org/sedaily.

Thank you so much to ThoughtWorks for being a longtime sponsor of Software Engineering Daily. We are proud to have ThoughtWorks and GoCD as sponsors of the show.

[END]