EPISODE 1252

[INTRODUCTION]

[00:00:00] JM: Cloud computing provides tools, storage servers and software products through the internet. Securing these resources is a constant process for companies deploying new code to their cloud environments. It's easy to overlook security flaws because company applications are very complex, and many people work together to develop them. The company Bridgecrew is a cloud security platform helping prevent mistakes from happening. Bridgecrew integrates into developer workloads to automatically find infrastructure errors and cloud accounts, workloads and infrastructure as code. Their platform also monitors code reviews and build pipelines to prevent errors from being deployed into production. If an error is found, then Bridgecrew software reverts that code back into its last known correct state. In today's episode, we talk with Barak Schoster, CTO and cofounder at Bridgecrew. Barak previously worked as a senior software architect at RSA Security, and as a software architect at Fortscale before that. We discuss cloud security, infrastructure as code, and architecture.

[INTERVIEW]

[00:00:59] JM: Barak, welcome to the show.

[00:01:01] BS: Hey, Jeffrey, thank you for having me.

[00:01:04] JM: You are one of the founders of Bridgecrew, and Bridgecrew is based around infrastructure security. And that is a wide breadth of subjects that could be explored. So I'd like to first just talk about what area of infrastructure security are you focused on.

[00:01:26] BS: So I see Bridgecrew not only as an infrastructure security company, but also as a productivity company. And I'll explain and through that LDAP to the area that we focus on. Our goal is to help SREs, DevOps engineers, cloud engineers, platform engineers, there are many names for the same set of activities of people who just want to access and modify resources in the cloud. And the way that Bridgecrew helps with those infrastructure cloud resources is we hook into infrastructure manifests, like Terraform, CloudFormation, Kubernetes, serverless, ARM, etc. And we help to take those configuration files and make sure that they have best practices configured as part of those configuration of cloud resources. So anything around infrastructure a code and cloud misconfigs can be handled using Bridgecrew group and our open source tools on a DevOps workstation or pipeline.

[00:02:29] JM: Is that basically like static analysis for configuration files?

[00:02:35] BS: It is correct. So we run static analysis on infrastructure as code. But we also correlate some of that with the running environment. So it's both static analysis, and also runtime analysis of the cloud configuration. So while doing the static analysis on the PaaS themselves, the Bridgecrew platform have the capability to query using API's, the different cloud provider and make sure if there is any drift between the code configuration and the cloud configuration and if there are also some resources that are only managed manually in the cloud and not through infrastructure as code, we have the ability to understand if there is a drift or a misconfig directly there.

[00:03:22] JM: What are some of the typical sources of misconfiguration or the prototypical problems that would be found?

[00:03:32] BS: So the sources for misconfigurations. So like most of the engineers, the thing that I personally, as an engineer, like to do most is copy pasting from Stack Overflow or similar. And when you look for an example of how to create an S3 bucket, or an EC2 instance, a security group, it doesn't mean that necessarily the example that you're copying from have best practices in mind. You might have a public S3 or an over-privileged IM role or any kind of other misconfiguration. So one source of it is just copy pasting from the wrong source, lack of knowledge on security best practices, and also there is such a vast majority of people that are not aware of each and every configuration state that it's a knowledge gap. And the common ones that we see when we query – So about a year and a half ago, we did a research querying all of the open source Terraform code out there and also cloud formation code and other manifests that are available on the public GitHub.

And one of the things that we saw that is commonly misconfigured is that you don't have auditing, like logs enabled on most of the services, and you don't have backup and recovery configured as an encryption. Those are the three main ones that we've seen, and we've scanned several 1000s of repositories and we saw those as repetitive issue. Does not make sense?

[00:05:03] JM: It does. And if unpatched or unfixed, do these problems like typically get exploited or are they just kind of bad practices? Like is it common to see infrastructure get exploited for these kinds of mistakes?

[00:05:21] BS: So, the lingo is around those misconfiguration varies. Some might call it exploitations, some [inaudible 00:05:28], and some are just human errors around misconfigurations. And we do see people that are using those bad actors that are using those misconfigurations to gain access and data to other company's assets. There is a famous one around the Capital One breach around two years ago with an over-privileged IM roll off an EC2 instance and over-privileged access to an S3 bucket that created a leakage of credit cards data. But there are also a lot of those that are unreported because you can find a lot of public assets out there in the wild.

Actually, one of the interesting stats that we've seen while scanning all of those open source infrastructure as code repositories is almost 50% of any open source repositories that you'll pick and choose to use will be misconfigured by default. And we are seeing that all of those repositories at the time, at the beginning of 2020, had more than 20 million downloads. So if you have a single repository that has several 1000s of downloads, those are several 1000s of misconfigured resources that are provisioned in one of the cloud providers. So they are commonly seen, but in some cases, we do see a trend of people starting to use open source tools and commercial tools to start and fix those misconfigurations.

[00:07:06] JM: Okay. So when does a scan like this run? Does it run like during a CI/CD process? Or do you have some other like periodic scan that people typically run? How does it typically fit into workflows?

[00:07:24] BS: Alright. So development lifecycle of cloud infrastructure has several stages. The earlier you identify an issue, the easier it is to fix, you have the context. And you are the person who’ve actually created the infrastructure configuration. And later in the process, you will see a misconfiguration, it's more expensive, it's already out there. So let's go through the stages.

The first stage to find a misconfig is while you're writing the actual code. So when you write a new Terraform block, for example, or a CloudFormation block, within your ID, let's say it's VS Code, you have the ability to use VS Code extensions like Chekov, which is our VS Code extension, and identify while you code and lint for a misconfiguration. That's the most productive way to do it. You're an engineer, as an individual you have issued like some kind of a security adviser on your workstation within your ID telling you, “Hey, you wrote that S3 bucket. You forgot to add encryption or versioning or logging to it. And this is how you do it.” And this VS Code extension will actually add the missing pieces of code or will guide you how to do that.

The next stage is let's say, you've created a lot of code and you want to commit that code. You can have a pre-commit hook that will prevent you from committing to a shared repository like a GitHub repository code that is misconfigured. But like everybody have issues in code and in some cases they want to force push or skip a pre-commit or they have like time restraints that an engineer might say, “Hey, I don't care about linting. I just want to push it to the right branch in my version control system. So the next thing that would kick in would be, bless you, CI/CD running process. Of you have a tool like Chekov or another static analysis tool scanning your code itself for any misconfig in the scope of an entire branch or an entire directory as another CI step. And you can have on top of that a set of comments that can be integrated into your pull request if you're using GitHub or merge request on GitLab, where you can actually start to collaborate.

So on CI/CD stage, as part of a pull request, Bridgecrew you can have an automated comment, and what we sees people in different teams starting to collaborate over issues that are coming from an automated bot. So let's say that you and me are writing the same piece of software and we're both maintaining this software infrastructure. Whenever a bot will tell you, “Hey, you really forgot to add logging.” You can start a conversation with your peers and say, “Hey, we are actually logging everything inside a central bucket or a central account.” And you should configure it by the bot guidance, but in that specific way. So the advantage of having that in a pull request stage is collaboration and visibility to teams.

And the next stage would be on continuous deployment. Right before applying a change to a running environment, you can scan the plan of a Terraform, which essentially is telling you "Hey, those are the changes that are going to be happening on your cloud environment." And unlike pure static analysis that relies only on files, the plan stage already have the context of what's going to be changed. What are the specific values and variables that are going to be provisioned under different attributes of a resource? And having another scan on the plan phase will help you understand, "Hey, is it really safe to provision this resource to production with its final configuration?"

And the last stage would be runtime. So even though you have full infrastructure as code workflow from IDE to pre-commit, CI/CD stuff, pull request comments. And right before the deployment, as an SRE, you might get a PagerDuty saying, "Hey, we need to add two more EC2 instances or to scale our database." And you will log in in the middle of the night because you got a PagerDuty, and you'll do a bunch of configuration changes. And you might have missed something or you provision a new instance through the console and you forgot to handle some of its configuration.

Runtime configuration scanning will give you the continuous assurance that the production environment is in a good state. The issue with the runtime environment is the good thing it has all of the context. You know what's there. The bad thing is, in some cases, it's not that easy to attribute a change to a specific user, if you're using assume roles in a cloud environment. And in some cases, it might be a little too late. So there are advantages to each and every stage from individual productivity on the IDE to full visibility on runtime and collaboration in between. And my personal opinion is that you should have something out there looking after you on each and every stage.

[00:13:01] JM: Got it? Let's take a moment here to talk a little bit about the engineering behind what you do. So can you describe how you build a static analysis tool for infrastructures code?

[00:13:17] BS: So behind the scenes, when we built a static analysis tool for infrastructure as code, we wanted to do several things. One is to have an open source community around it, because one of the key things of a good static analysis tools is to have a good set of rules and best practices. And one of the issues that we've started talking about at the beginning of this talk was that there is actually a lack of knowledge of what are the best practices. So we created an open source tool called Chekov and we made sure that as an open source tool, the easiest thing to do would be to contribute new content, to contribute a new set of rules. Like how to encrypt S3 was a contributed rule and how to make sure your SQL servers are secured.

And we saw people contribute content around all the three major cloud providers in different infrastructure as code frameworks. And the next phase we wanted to make sure that it is easy, and we started to launch Chekov 2.0, was to give context and awareness in rules to the different relationship of cloud resources. By relationship, I mean that when you look on a cloud resource, you understand that it needs further context. It does not leave unattached to anything in the world. Usually, a compute resource would be attached to a network interface. And the network interface would be attached to access controller sources. And there is also IM resources to grant privileges to that compute resource to do modifications or CRUD operations over different databases.

So this complex relationship between access control lists, compute resources, data resources are actually managed in a DAG, a directed acyclic graph, on the different infrastructure as code frameworks. So, the infrastructure as code framework piece made sure that we can define resources in a form of relationship. And we wanted to take this craft theory of provisioning cloud resources and their dependencies and to apply it to the concept of policy as code. And in those policies, we wanted to be able to express rules like make sure a VM is not public. And it's a complex rule, because there are various ways to make a public instance. You need the security group in AWS to be public. You need the subnet to be public. And you would need to have a listening endpoint within your machine. So having all of those different attributes expressed under a single policy is a complex graph data model. And this is another thing that we've enabled on this static analysis tool, which is also important to do on any static analysis tool for infrastructure as code.

[00:16:29] JM: And in building Bridgecrew and building the static analysis, and also the tooling around it, can you just tell me some of the infrastructure decisions that you made, like cloud providers, whether use Kubernetes? Just your general build versus buy approaches?

[00:16:51] BS: Sure. When we started to build the Bridgecrew SaaS offering, we knew that we should handle a large scale developer as a product-lead growth product that is meant for developers. We wanted to support any scale of scanning. So we decided to stick to Lambda. AWS Lambda is our core compute infrastructure for short leaving tasks, and to ECS Fargate, for long leaving compute tasks. And the reason that we chose those two core compute components were developers, which are our audience works in peaks. Meaning they do not work on weekends. So we don't need to pay for resources to be up and running over weekends. And they have relatively predictable working hours, I hope, excluding some cases. So we knew that our event boss or the different events that we're getting as a platform have hours that we get a lot of data, and some hours that we don't get a lot of data, and we didn't want to have a VM that is up and running always and listening.

So early on, we decided to focus mainly on Lambda, and we have hundreds of Lambda running in production. And it actually helped us to scale without a lot of issues for any of the short living tasks. And we chose ECS Fargate versus Kubernetes because it was very simple to configure, very simple to start with, and we did not need the same scaling and configuration experience that Kubernetes has. So we wanted to really dumb it down for maintenance of our infrastructure. And this is where actually Fargate excels. If you don't need something massively complicated like Kubernetes, Fargate would usually do the stuff that you need for you.

Other than that, we stick to API Gateway and CloudFront because they give you most of the stuff that you need around CDN, integration to a web application firewall, and managing the different access keys and scaling. And we also chose to use AWS Cognito to manage all of the different users and tenants of our customers. And the nice thing about sticking to serverless and those managed services in AWS is it's really – One, it's really easy to scale and maintain. You don't have a lot of tribal knowledge. As its engineering leader, it's really easy to onboard new team members, because everything is already documented on a much better scale that you would ever do for your own infrastructure.

So my general approach to this is we should buy tools, assuming that they're good for our cause, because it just makes it more easy to have a common language around it within your engineering team. And third, it's really easy to pass things like audits, because if you've used it right and you've configured it correctly, you can easily pass through audits, because a lot of the infrastructure, at least in the shared model, some of the infrastructure is maintained by the cloud provider.

[00:20:22] JM: Got it? And has there been a downside to building so much of your infrastructure around Lambda?

[00:20:31] BS: There were some hiccups. The things that we wanted to learn is, what happens if you get a single event that will produce a very long living Lambda? You'll get a Lambda timeout. In some cases, we call it internally Lambda hotspots, because it's just like having hotspots issues on database partitions, where a specific Lambda got an event that do a query to a database that was just too long, or gave back too much data for a single Lambda to process. And in those cases, we needed to make our architecture slightly more complex. We had to add SQS queues and partitioning logic that will actually take one message that produces a lot of data and a lot of compute requirements into a queue that will act as a message bus for the different Lambdas.

So in an essence, when you need to scale a Lambda further from is 15 minutes limit to multiple Lambdas and you cannot add too many threads to a Lambda, you are limited in the number of cores that you have there. You need to do some partitioning logic within your architecture, which makes it slightly more complex. But if we were like on VMs, we would have encountered the same issue at some point. But the upside of VM vs. Lambda is that in some cases you can just add more CPU and you can add more RAM, and then the problem is done. And the Lambda, you are more limited to what you can do. But the upside of it, it makes you do really scalable architecture and you're really focused on solving the scaling issue around the subset of microservices that are really the bottleneck. And one of the other things that we decided to add very early on and purchase is an APM that really helps us to understand where the bottleneck is and to give us hints on how we can fix those bottlenecks. So we are being a little proactive about where we have bottlenecks in our Lambda pipeline.

One of the other cool things that we did around it was we took that APM and we've decided to configure it not on the entire stack, but we gave tags and attributes of ownership to each subset of microservices within our production infrastructure. And whenever we have a defined threshold that is being triggered, a specific set of teams will get a PagerDuty on top of it. So the entire proactive approach of defining a threshold, seeing where Lambda is about to hit that limit or that bottleneck, and then alerting to the owning team really helps us to scale and to make sure that our product works well in production. And when I'm saying scale, it's not about actually adding more compute, but predicting an issue and scale your engineering org to have the right tasks in their Jira board, for example, around what's the current bottleneck? Who's owning it, and who should fix it, and when it should be fixed? So it's an early decision that we were really happy about.

[00:24:01] JM: Let's get back to the top down perspective of what you're building. So let's say I'm a company and I want to implement a security around my infrastructure as code. Maybe I've got a ton of infrastructure as code as it stands, and I want to just be more secure about it. What are the steps to implement a security practice around that infrastructure as code?

[00:24:31] BS: So the way that I am looking at it, you should approach to that issue as if you would like to distribute it across multiple teams because usually – Well, there are two options. Either you have a centralized DevOps or SRE organization, or you have squads where SREs are scattered across those squads, which we'll get at some point when you scale your org.

So the way that I would recommend is start with identifying what are the critical assets in your cloud environment that you want to have repetitive and predictable deployment of. Migrate those resources to infrastructure as code if you have a running production environment. And after doing that, when you start writing the code, make sure that you have the basic open source tools that will identify and lint those misconfigs. So you can use Bridgescrew’s tool, which is Chekov. It's under Apache 2 license, but you can also use tfsec, or tflint and other alternatives to make sure that your code is secure.

And I would also recommend to have things like AWS secrets or Git secrets, which you scan for plaintext secrets within your code and will prevent you – Prevent from you as an engineer to provision secrets into a version control system or a production environment. And then integrate those same tools to a CI pipeline and then scan a running infrastructure, either using commercial tools or cloud native tools like the ones that AWS, GCP and Azure provides. And make sure that you have a governance methodology where you can distribute each and every issue to the right owner. Because at large scale, you will have a lot of engineers doing a lot of commits to your cloud infrastructure. And if you deploy every day or every few hours, it is really easy to get lost in the monitoring and observability of those cloud resources. And if you have the right governance around it, you can have an automated and distributed escalation process around cloud misconfigs where each owner gets and owns over time the state of that cloud infrastructure.

[00:27:07] JM: Tell me a little bit more about the upkeep of the tooling around infrastructure as code security. Like how often am I going to need to be checking my security tools? How often am I going to need to be updating my infrastructure as code files? Basically, like, how much time am I going to have to spend dealing with this?

[00:27:30] BS: So think of it like unit tests. If you want to have good coverage on a new piece of software, you're also asking yourself, “How many tests do I need to write? What is a good coverage for my organization? And how would it help me to scale?” If you have a set of unit tests that are common to multiple teams, those can actually help not only you as an individual scale, but other teams scale too. So if you want to integrate an infrastructure security tool, you can look at it as a way to go forward and have less issues in the future on your cloud infrastructure, because it will guide you to do the best practices around not only security, but also reliability, like backups and recovery and observability, like having audits and logs provisioned. So it's really out there for you. And actually, you don't have to do a lot. To use Chekhov, it takes only two minutes to deploy. It's a simple pip install or brew install or Docker pull command. And you can find it on your infrastructure as code directory and get results. And it already comes with 1000 different best practices.

So when you are thinking about how much time to allocate that, start with doing those two minutes actions of downloading and pointing to the right directory. And again, it's free under Apache 2 license, and you'll get 1000 tests out of the box. And now if you want to fine tune or create your own custom checks within Checkov, you can allocate some time to read, but I think that for most of the people, the out of the box community contributed 1000 policies are really the best place to start. And as you scale and create more complex cloud infrastructure, you can allocate more time to that, but it's probably a good way to start.

Actually, one of the things that were super interesting for me actually as an engineer is that when I started the write Chekov and I published it to the open source, it had only 40 rules. And on those 40 rules, I had coverage on the off I think 10 or 20 services of one of the cloud providers. And when the community started to contribute policies back, we called it either policies or checks, I've actually had people from Airbnb, and Netflix, and Pinterest, and Salesforce and other companies contributing to my open source tool from their knowledge.

So for the first time, when I wrote a new set of infrastructure as code resources I finally had, for the first time, new tests failing, new policies failing that I didn't know that existed. Best practices that I didn't even think about were contributed by the community help me as an engineer to write a better code. So it's not just about having those open source rules out there for the first time. But it's also having that open source motion of community that is maintaining the rules and adding more and more. So if you lose track of once and keep updating it, every once in a while using brew update or pip install update, you'll get a continuous notion of more and more tests being added for you. And I do encourage you to contribute some of your custom policies back to the community so everybody can enjoy the same experience. Does it make sense?

[00:31:16] JM: Yeah, totally. At this point, you've worked with a lot of larger companies that have adopted your tooling. What have you noticed about how the really large companies like a Robinhood uses the security tools that you've built?

[00:31:34] BS: So on large enterprises, the nice thing about it is that you have to collaborate to succeed on building a complex architecture. And what we see that is really working for teams is that when they're using the collaboration features and compliance features, it helps to standardize the lingo between team of how a good environment should look like. So as a SaaS offering, the things that we cared about is, one, have a very fast onboarding and guidance for new users when they collaborate. And also have a good integration to different version control systems and CI/CD pipelines that actually gives added value and added context that people can actually talk about as part of their development lifecycle. And what I'm meaning to talk about, it's not only on a podcast, or a Zoom like we're doing right now, or Slack, it's also talking about it on the pull request itself.

And it creates a network effect, where a single engineer that have started to use with an infrastructure security tool is having a conversation, having a comment from our bot within their pull request, and then having a conversation with other team members that see the bot comments. And more and more people are starting to use this bot, and it spreads in the organization like wildfire, where everybody are taking advices from a virtual security advisor that is actually behind the scene set of policies being contributed by the community and being maintained by the community. So this is one way that larger organization uses that tool.

And the other way that those teams are using the tool is correlating the code changes that they're doing into the cloud provider changes that they're seeing both under a single pane of glass, if you will, and attaching those to their real security and compliance needs. So Robinhood that you mentioned, for example, is a company that holds sensitive financial data. So they might have requirements around PCI or SOC 2 or otter compliance best practices that they would like to follow. So the way that the platform would guide an engineer in a FinTech company would be, “Hey, if you're really interested about securing financial data, those are the different policies that you should actually follow.” So it also help you to focus on your actual needs so you don't need to have 1000 different rules, but you can have the subset that are really relevant for you.

[00:34:29] JM: And is the usage and just the class of problems that they're trying to tackle at a really, really large company, do the infrastructure workflows, infrastructure as code workflows, necessitate kind of a different usage just because the scale is bigger?

[00:34:48] BS: I think that it's just scattered across more ways of deployments and more pipelines. In some cases, on larger organization, you see a single path of deployment, like a unified pipeline to manage and control the cloud infrastructure. But in other organization that chose to work on a distributed model like squads, you see that each team is responsible end-to-end on their own pipeline, or in their own application, from frontend, to API, databases, and infrastructure provisioning, from Docker, Kubernetes, Terraform. And it usually happens on large organization, and the hard thing about it is how do you make sure that everybody have the same set of policies defined for all the different teams? And if you create a set of custom policies that should work for your organization, how do you share them?

So one of the nice concepts on some of the policy as code engines is the ability to version control policies and to have a set shared repositories around them. So on larger organizations, I think that is different is that it's harder to control. But you can create a paved road if you will of a subset of policies that everybody should pick and choose in a service catalog notion to apply on their own team and subset of microservices. And when you manage this kind of library, and you internally advocate other teams to use it, you see that a lot of the issues that you had before in production are now being prevented very early on, which is great experience, because each team is feeling more productive, and they have less overhead of unplanned security tickets as part of their backlog.

[00:36:56] JM: Alright. Well, since I last did a show with Bridgecrew, one of your cofounders, you guys have gotten acquired. So Bridgecrew joined Palo Alto Networks. And I guess I'd like to get a picture for the synergies of that acquisition and where you're planning to take the product post-acquisition.

[00:37:20] BS: Yeah, so I'm super happy about this event. And the synergy – For Bridgecrew, it is really an opportunity. Palo Alto is probably one of the largest security companies out there. And they're making a lot of investment into cloud infrastructure security. And Palo Alto have created an amazing platform called Prisma that knows how to monitor cloud infrastructure past the API's, and the workloads like containers and Kubernetes clusters. And with Bridgecrew, it's now the code itself. So you have full visibility from code, to workload, to cloud. And you can monitor your security posture or workload on each and every stage of your lifecycle. And the way that we see Bridgecrew is we see us continuing to help more DevOps engineers and on more stages of the pipeline, and more technologies that are part of the pipeline. So we recently introduced Docker file analysis, which is an infrastructure that we haven't supported before. And we actually have great plans about open sourcing more and more capabilities for DevOps engineers. So as a team, a large team of Prisma Cloud, which is part of Palo Alto, we have different units that are responsible for different stages and personas, and Bridgecrew as part of the acquisition is going to be very focused on the DevOps or SRE role and trying to create more productivity tools for those engineers.

[00:39:07] JM: What predictions do you have for how cloud infrastructure security is going to change and how that's going to change the direction of your product?

[00:39:16] BS: So the nice thing about cloud infrastructure is that for every developer out there, the cloud providers are really trying to simplify and add more capabilities to each and every resource out there in the cloud. And when they do that, they expose new features. And when they expose new features, they expose new API's and new capabilities, and those API's can be configured correctly or not. For us, those are more opportunities to give more value around scanning the infrastructure as code that is provisioning those resources.

So as the cloud providers grow, there is a bigger opportunity for Bridgecrew to help and solve a larger problem on the infrastructure as code security space. And if we'll take a vision of like five years from now, the cloud providers would probably either be even more developer friendly and be integrated to more steps in developer pipelines, in my opinion, or will create more tools like no-code for the less experienced engineers or other departments of enterprises help them create applications with scaling capabilities and security capabilities without having the need to maintain larger SRE organization. And the reason for those two predictions in my opinion is it's really hard to get the right talent in the market to have and it's really hard to recruit more SREs and DevOps engineers to develop that expertise.

So I think that the cloud providers are really thinking hard on how to simplify that experience, either by granting more developers the capabilities to create resources in the cloud, in the ways that they know. And we started to see that by seeing CDK, which is bringing infrastructure development into imperative languages that are common across different developers, like Python, or in TypeScript and JavaScript, etc. Or no-code capabilities that will grant other departments like marketing departments or business development or sales to create applications that will apply to their needs. And we've already seen some of those applications within Oracle Cloud actually, and also Salesforce, if we'll count that as a cloud provider.

[00:41:55] JM: All right, well, as we wind down, I'd love to just get reflections on building a company and having a successful outcome. What were the lessons that you've learned along the way? And what were maybe the hardest problems that you had to solve along the way?

[00:42:15] BS: I think that the things that we really learned is, I know it might sound obvious, but listen to your customers. I think that before we've developed the first line of code to go to production, we had more than 100 different conversations with DevOps practitioners, security practitioners, cloud engineers across different companies across the globe, East Coast, West Coast, Europe, Mediterranean, and even Korea, I think, and we got all of the feedback. And we've asked and interviewed a lot of them, asking them, “How do you provision cloud infrastructure? How do your architecture look like? How do you scale those?” Similar questions to the ones that you've asked me.

And they've described the flow of how they govern and maintain and review cloud infrastructure. And they've told us about manual process that they do as teammates of reviewing pull requests. And it gave us the idea to have those capabilities as a product with a bot, within a version control system. But it also gave us the idea of, “Hey,” we asked them, “How do you know what's good?” And it gave us another idea about open sourcing that, because a lot of people had knowledge sharing issue. So when we are listening to our customers, we are really realizing what their pain is, and he pretty helps us to identify what would be the matching solution. So that's one lesson.

The second lesson that worked well for us for a portion of time was we actually started with giving professional services, augmenting teams, really at the beginning of the company. We're not doing that today. And when we joined SRE teams, we saw how their day-to-day looks like and how their daily experiences looks like. And it gave us the ability to see what is the favorite tools of each team, and how they are used using those tools to communicate and solve problems. And it really helped us to narrow down to what are the integrations and languages we should focus on. So that's the second lesson.

And the third lesson since we had – A lot of a lifetime of the company was during COVID, is an internal one actually, is how to improve internal communication of our own product, engineering, marketing and sales team in a period of time when we cannot fly to visit each other and we cannot meet each other at the office. So we worked a lot about having communication and off hours where we are just drinking coffee together and enjoying how much we could be quarantine period. So we'll have a really good communication within the engineering team, and it really helped us to be, one, more caring for one another, and two, actually more productive, because the communication was really better. So those are the three lessons. And the fourth, actually, is how to maintain an open source tool and community. So we use the open source tool as another strong validation to what features we should develop. In a lot of the cases when we saw a community member opening an issue, we guided them how to contribute back, but we also tried to follow up with a meeting to understand what other needs an open source contributor has, and what other pains do they have and what other expectation they have from an open source tool. And it really helped us to create the next set of features of the open source tool. And also, it helped us to focus on where the value of a paid SaaS offering should be. So if you are trying to build a product-lead growth company around open source and to have an open core, you shouldn't just create an open source and leave it out there. You should talk with your open source users and see how you can help them more and what would make sense to be a free open source feature versus a paid feature. I think that's it for my lessons.

[00:46:37] JM: All right. Well, Barak, thank you so much for coming on the show. It's been a real pleasure talking to you.

[00:46:42] BS: Thank you, Jeffrey. Thank you for having me.

[END]