EPISODE 761

[INTRODUCTION]

[00:00:00] JM: HashiCorp was founded 7 years ago with the goal of building infrastructure tools for automating cloud workflows, such as provisioning, secret management and service discovery. HashiCorp's thesis was that operating cloud infrastructure was too hard and there was a need for new tools to serve application developers.

HashiCorp founders, Mitchell Hashimoto and Armon Dagdar began releasing open source tools to fulfill their vision of better automation. Terraform, Vagrant, Console and other tools created by HashiCorp gained popularity, and HashiCorp began iterating on their business model. Today, HashiCorp makes money by offering enterprise features and support to enterprises such as Pinterest, Adobe and Cruise Automation.

Over the last 7 years, enterprise software infrastructure has changed rapidly. First, enterprises moved from scripted-based infrastructure automation to container orchestration frameworks. Then the container orchestration world consolidated around Kubernetes. Today, large enterprises are rapidly adapting Kubernetes with a mix of public cloud and on-prem vendors. At the same time, these enterprises are also becoming more willing to consume proprietary tools from the public cloud providers.

Hashicorp has benefited from all of this change. Their different tools fit into a variety of workflows and are not closely coupled with any particular cloud provider or platform solution. Armon and Mitchell joins today's show to discuss the business model and the product philosophy of HashiCorp. We also touched on service mesh, zero trust networking and their lessons from the container orchestration wars.

It was a real pleasure to have the founders of HashiCorp on the show, and I hope you enjoy the episode as well.

[SPONSOR MESSAGE]

[00:02:06] JM: Today's episode of Software Engineering Daily is sponsored by Datadog, a monitoring platform for cloud scale infrastructure and applications. Datadog provides dashboarding, alerting, application performance monitoring and log management in one tightly integrated platform so that you can get end-to-end visibility quickly, and it integrates seamlessly with AWS so you can start monitoring EC2, RDS, ECS and all your other AWS services in minutes.

Visualize key metrics, set alerts to identify anomalies and collaborate with your team to troubleshoot and fix issues fast. Try it yourself by starting a free 14-day trial today. Listeners of this podcast will also receive a free Datadog t-shirt. Go to softwareengineeringdaily.com/datadog to get that fuzzy, comfortable t-shirt. That's softwareengineeringdaily.com/datadog.

[INTERVIEW]

[00:03:13] JM: Armon Dagdar, you are the CTO and cofounder of HashiCorp, and you're going to be joined eventually by your cofounder, but we'll start with just you. I want to start by discussing the strategy of HashiCorp, because there's been so much change in the world of software in the last three years. You have massively growing public cloud adaption. You have Kubernetes leading to enterprises changing their infrastructure really significantly. As a result of all of these change, there are some infrastructure companies that have grown much, much larger. There are some companies who have gone completely out of business. So they've just lost steam in their business.

You are one of the companies that grew stronger during this changer rather than suffer from all the change. Why is that?

[00:03:56] AD: Yeah, I mean I think that's a good question. I think over the last three years, I'd say that has been an interesting period for us as that's been kind of the period in which we kind of tried to graduate out of being a pure open source vendor. Because I think if you go back three years, effectively we had no real commercial basis. I joke we were just an extremely well-funded research project.

From that time we tried to figure out, “Okay, if HashiCorp is going to be viable in the long run, what does HashiCorp like the business actually look like?” and I don’t think we knew what the right answer was right away. So it was kind of a series of experiments. I think initially it was really around, “Hey, can we build a business purely on the back of support and just they’ll support around some of the open source stuff?”

I think that was useful at the very beginning, sort of just bootstrapping some relationships with customers and kind of the early revenue of the company was kind of built on that. But the challenge we found was there’s a moral hazard, which is like really good software doesn’t need a lot of support. So what we found is if we’re committed to writing really good software, some of the customers come back and be like, “Well, why do I need to buy support? It seems to work pretty well.”

I think from there we tried a few other strategies, one of them being sort of a hosted platform, sort of an integrated, the full Hashi stack end-to-end. We host a thing kind of like a Heroku, but built around HashiCorp tools. So we kind of gave that a try. What’s some reasonable success some recognizable startups that you would know were customers? But I think the challenge we felt there was we couldn’t sell to larger companies as a result of the model. So there was sort of this price ceiling that we kind of get above, and the challenge was how many of the customers we really need to have at this low price ceiling to make the company sustainable and the math didn’t quite work.

I think where we ultimately landed was just – Has been the most successful has been really saying, “Okay. Well, what are the specific and unique needs of the very biggest companies? The kind of global 2000, Fortune 2000, and what do they need above and beyond the open source? Can we package that as part of a commercial enterprise offering? Which of course has support with it and all of that and use that as how we build the business.” I think we found that that was supper successful, because if you really take any of our products and I’d say, “What does it take to use Terraform in the context of big enterprise or Vault in the context of a big enterprise?” There’s all these requirements that are either it’s because you’re a highly-regulated financial or you have a bunch of compliance obligation as an insurance company, or so on and so forth that you need a different set of features? I think if you package that up as part of the enterprise offering, that’s kind of best of all worlds.

We love the open source, we love that we can sort of figure out does it solve our problem? But there're some of these checkboxes we have to hit to be sort of a viable commercial – Whatever, insurance company. So great, we're willing to pay for software that solves that for us. I think that's really I think the sweet spot we found and I think what's helped HashiCorp to be successful.

[00:06:52] JM: My sense is that in the midst of all that change, the principles that were laid out in the Tao of HashiCorp were pretty useful, because you had something to lean on as there was so much change and so much opportunity going on. What's the most underrated principle in the Tal of HashiCorp?

[00:07:10] AD: Most underrated I think by far is the last, last but certainly not least, is reflection.

[00:07:14] JM: I thought it was pragmatism?

[00:07:16] AD: Is pragmatism the last one? Pragmatism is the last one in the Tao of HashiCorp. We actually have two documents we publish. We have our Tao, which is our sort of design ethos. As we design software and think about sort of how to build things, the Tao of HashiCorp is what guides as and it has things like focusing on immutability, pragmatism, a focus on workflows over technologies.

But we also publish our principles of HashiCorp, which are more about sort of how the company should feel and run, and those are focused on kindness, and reflection, and pragmatism is actually one of those. So pragmatism, actually the only thing that is both in the Tao and the principles of HashiCorp. So we try to be deeply pragmatic. But reflection I think for us is the important principle, because I think sometimes that process of saying like, "Hey, we're going to try this thing like support, and then we're going to reflect on it and say, "Okay. Well, we've tried it. But, hey, here are the limitations of it." It's hard to get customers to renew, because if the software is good enough, they're not using the support? We tried the platform and reflected on it and said, "You know what? The price ceiling is going to make it hard to build a large, sustainable business." So I think that's been the most useful bit, is the ability to try something and then ask our self, "Is this thing working? Should we double down on it or should we try something else?"

[00:08:33] JM: Right. Google gets criticized for sun-setting products, and Amazon gets complimented for supporting the same products forever and never decommissioning products. But as software developers, should we be happy when software engineers are just allocated towards projects that are gaining traction or projects that are worthwhile? Why is there this sort of sentiment against sun-setting products that are not working?

[00:09:02] AD: Yeah, it's a good question. I think it comes back to the question of like, "How deeply is the product being used?" I think some of the challenge with Google, and I felt this personally with – I was a big fan of their RSS reader tool, and it's one of these things where you're like, "Okay, it's like I know I use it. I can name 10 people at the top of my head that I know use it, and Google is deciding to sunset it."

I think in an environment like that, people get upset, right? Because like here's a product people use and love and are getting value out of and you're taking it away from them. I think it feels irrational to those of us who are using it to be like, "Why are you getting rid of those things? Clearly, a lot of people use it."

At the same time I look at a product that we build for a while called Otto. It was a tool that we announced at HashiConf, a lot of fan fairs. One of our sort of big temple investments. But ultimately about 12, 18 months after we announced it, we made the very public decision to sunset the product, and we had a blog post where we announced it. We updated its homepage to sort of say like, "It's being end of life and this is why."

I think with that product, we had to really look at it and say, "You know what? There's a problem we tried to set up to solve." And we set up at that time with Otto to really look and say, "Hey, infrastructure is so complicated. There are so many moving pieces even in our own portfolio," that if you say, "I want to just deploy a simple Rails apps on AWS and I'm going to use the Hashi stack." We'd say, "Great. You have to learn Packer, and Terraform, and Vault, and Console to do this sort of successfully, and that's a huge learning curve.

We said, “Could we build you a tool? Otto?” They would sort of help you automatically “try and do some of these stuff,” where you just describe your high-level intent. Say, “It’s a Rails app on AWS,” and we’ll translate it to the lower level, Terraform, Packer evolve for you and so on.

I think what we found, that was the goal. A year and a half after release, so this being well over two years of development, it felt like we’d missed the mark, that the abstraction wasn’t right, that it was too leaky, that the workflow wasn’t what we wanted. So I think it was through that process of reflecting and saying, “Okay. Is this how we nailed the essential product experience, and now what we’re doing is adding features and polish and things like that, or do we really feel like we missed the mark?” I think to us it felt like we missed the mark.

So I think at that time we had to make that decision of do you keep investing and double down on it even if there’s a few people who used it? Certainly, it was a Google reader level. Or do you say, “You know what? Time if precious. Let’s make a whole new attempt at solving this in a different way rather than trying to iteratively fix something that we feel like just totally missed it?” I think in that context we made the decision, “Let’s kill Otto,” and we did it publicly.

So I think if you do that all the time, there’s sort of a trust issue of should we trust any of your tools? But I think if you’re selective about it – I think, for example, if today we decided we’re going to end of life Terraform. I think there’d be outrage. But I think if you look at a tool that was earlier in its life that was still sort of figuring out what I wanted to be when I grew up, I think it’s okay to make that decision. I think it’s hard. I think it depends on where the tool is, where the product is. If enough users are getting value out of it, you’re kind of stuck with it forever.

[00:12:11] JM: In the earlier days, HashiCorp was – I don’t know the exact timeline, but was churning out products very quickly, new products. At a certain point it seemed like you said, “Okay, we’ve got enough products. Let’s double down on some of them.” I don’t think it was for any lack of new ideas for different products. I think I heard mentioned that there is now like a research group inside the company that looks out maybe 12 to 18 months and says, “Okay. Now that we have some product market fit on at least four different products internally, we can have most of the company focus on those core products, iteratively improving them and we can have kind of the research team think more deeply about what really big bets do we want to make.”

Can you tell me about the strategic change there and how you went from just saying, “Okay, we’re just going to keep releasing new stuff,” and you had a more deliberate split between doubling down versus new products.

[00:13:08] AD: Yeah. I think when we first started the company at that time me and Mitchell had kind of a vision for I think what the problems was. I think our sort of feeling was the whole operational space, all the existing tools felt broken to us when we’re talking about things like Zoo Keeper and Nagios and Capistrano and kind of all that stuff that was there felt like we’re taking stuff out of kind of the legacy data center and sort of dragging, kicking and screaming and trying to use it in the cloud.

So when we started the company we really said, “What if we had a blank slate? What if we could rethink the tools we used operationally to do our provisioning and our monitoring and our deployment and kind of the whole nine yards? What would that look like?” and where would you draw the dividing lines differently this time? I think in some sense, that old universe of tooling grew up very organically where the dividing lines is like, “Well, look. Where does the last tool leave off? Okay. Well, that’s where the next tool will pick it up.” Versus being able to really think about it from a clean slate and sort of build your own little universe.

So I think that was sort of how we’d approach it. We said, “Okay. If we had to decompose and say, “What were the pieces you need? We felt you needed a tool that would let you write and test code. Well, that was Vagrant. You needed a tool that would let you package your application up. That was Packer. You needed some way to actually provision infrastructure and deploy your packaged app. Well, that’s Terraform. We need some way to do secret management, Vault. You need some way to glue these things together so the apps can talk to one another and route, and it turns up that’s Console.”

The final piece as I get into more containerization, sophisticated platforms, how do I do sort of a higher-level, I’m deploying an app, not booting a VM? That was Nomad. To be able to think about application level lifecycle and not machine level lifecycle.

So from the get go, I think we had this vision of this portfolio of products needs to exist, and none of it exists when we started. So that first few years was very much just focused on building out that whole portfolio that we sort of only existed on our mind's eyes. So we knew it all had to come together.

Once we'd at least finish that portion of it, I think there was a slowdown of, "Okay, let's catch our breath, but actually double down on the products that exists and not just keep adding new ones." But to your point, I think there's sort of no lack of ideas. I think even today we keep getting shutdown, me and Mitchell, from products saying, "No. You can't launch any new products. We have enough as it is. Stop launching things."

I think the research team was, to a degree, an outlet for that, and the goal for them was very much not sort of pie in the sky academic research, but it's very much industrial. So the goal is that the things they work on either ends up as features or enhancements to the product, or as products themselves. I think in one case we saw that sort of the – It was a paper that we actually end up publishing about Console, called Lifeguard, where we really looked at the kind of gossip properties, because we used sort of a gossip algorithm under the hood. It starts to get a little misshapen as the cluster gets too big. You start seeing erratic behavior as you get into 5,000, 10,000, 20,000 nodes. So we were having customers at that scale who were kind of reporting these weird issues.

So the whole core of that focus was how do you make these gossip algorithms much more sort of full proof at these really high-operating scales? That was the basis of that paper and that enhancement and landed in Console. So now we have users with 35,000, 40,000 nodes in a data center and it's running smoothly. So that was a really good example of working well. Again, right now we're working on this product that we just talked about at HashiConf called Vault Advisor, which is really like how do we analyze the way a system is configured? In this case evolved, but it could be generalized to other things and look at it from a security perspective and say, "Hey, if you tweak your configurations in this way, you can reduce your surface area attack of a compromise or a bad user by 80% or 90%."

So it's looking at this sort of like how do we give you sort of expert guidance how to reconfigure a very complex system? Ultimately, our goal that that becomes sort of a feature of all of our tools, is that they're more intelligent about, "Hey, if you reconfigure a console this way, you can improve your security posture."

[00:17:06] JM: Yeah. So Mitchell just joined. Just so you know, the setup is like he kind of talk into – This is the main one, and then that's kind of the backup. So given what you just said, I want to talk about the service mesh category as an example of HashiCorp product development. So Console, which is a product that you guys have been working on for a long time was around and was doing a lot of the things that are now categorized as service mesh today. What's your vision for how service mesh gets deployed at a large enterprise? Will there be multiple instances of a service mesh? Will it be one big unified service mesh?

I think at this point it's been validated that people actually want this. Now, it's a question of how it makes its way across a given enterprise?

[00:17:56] AD: I think the view – In some sense I think it goes back to – And we see this all the time with even tools like Vault and Console today, is like how do those get deployed in an enterprise? I think it comes back to what is their corporate structure. What I mean by that is you see some companies that they're very much – There's a strong centralization, strong standardization and there's a platform team that's very, I guess, controlling about what the environment looks like. So they can impose centrally and say there's going to be a single service fabric. We're going to deploy Console. That's the standard. The whole enterprise has to use Console and there's one giant logical deployment of it. But that's not a lot of companies.

A lot of the mega-scale enterprises we work with, and some of it is just because they're so large, but a lot of it is because it's through acquisition. You're a bank that you bought a mortgage company and then you bought an insurance company, you bought a wealth management company. So in some sense you're actually 8 different companies that have been merged into one and they have 8 different IT departments with 8 different technology stacks, in some cases 8 different data centers.

So in those environments, it's very difficult to get to say, "We're going to deploy one mega console." So instead what you tend to see is sort of a federation. Each one of those business units might run their own clusters, their own environment, then they're going to federate them all together. So I think it really ties back to what is the corporate structure look like. If it's highly-centralized and standardized, it might be a single super deployment. If it's a highly-federated business through acquisition or just that's how they're structured, it's likely to be many independent federated deployments.

[00:19:28] MH: Yeah, I think something that's interesting that's happening right now is all – I mean, not all, but AWS for example just announced App Mesh, which is a totally proprietary their own version of things, like even at the protocol layer, like totally different from Istio and Console and things like that. It will be interesting to see sort of will something like Console that works across clusters, across on-prem to cloud sort of win in that space, or will you want to use App Mesh within AWS and then want to use Console for on-prem, and we're really enabling both right now.

So obviously will Console work across clouds with itself? I mean, that's the easy problem. But also at Reinvent, AWS talked about how we partnered with them to work on App Mesh to enable it to work outside of AWs, because AWS's point of view is unless you're on AWS, of course App Mesh won't work. So can they find a partners, and we've always been multi-cloud. That's been our ammo. So we're a very natural partner to work with to enable that, and that's what we're doing right now.

[SPONSOR MESSAGE]

[00:20:35] JM: MongoDb is the most popular non-relational database and it is very easy to use. Whether you work at a startup or a fortune 100 company, changes are that some team or someone within your company is using MongoDB for work and personal projects. Now, with MongoDB Stitch, you can build, secure and extend your MongoDB applications easily and reliably. MongoDB Stitch is a serverless platform for MongoDB. It allows you to build rich interactions with your database.

Use Stitch triggers to react to database changes and authentication events in real-time. Automatically sync data between documents held in MongoDB mobile, or in the database backend. Use Stitch functions to run functions in the cloud.

To try it out yourself today, experiment with $10 in free credit towards MongoDBs cloud products. MongoDB Atlast, the hosted MongoDB database service, and Stitch. You can get these $10 in free credits by going to mongodb.com/sedaily. Try out the MongoDB platform by going to mongodb.com/sedaily, get $10 in free credit. It's the perfect amount to get going with that side project that you've meaning to build. You can try out serverless with MongoDB. Serverless is an emergent pattern. It's something that you want to get acquainted with if you're a developer, and getting that $10 in free credit to have a serverless platform right next to your Mongo database is really a great place to start. So go to mongodb.com/sedaily. Claim that credit, and thanks to MongoDB for being a sponsor of Software Engineering Daily. I love MongoDB. I use it in most of my projects. It's just the database that I want to use.

Thanks to MongoDB.

[INTERVIEW CONTINUED]

[00:22:43] JM: In some ways, the App Mesh is reminiscent of what Amazon did during the container orchestration wars with ECS. Is the service mesh area – Do you think it's going to play out similarly to the container orchestration wars where it's like a winner take all thing, or do you think it's more of a multi-winner environment?

[00:23:04] MH: I think it's very different to me than Kubernetes and container platforms, because you could kind of sneak Kubernetes in application by application, but the benefit of service mesh is really the sort of networking security across applications and across services. So it requires a certain level of coordination that I don't think platforms require.

So as Armon said, like depending on your company, centralization versus not is different. But I think networks in particular are something that have held on to more centralization than other things. You saw a lot more VMWare, versus open stack, versus bare metal, versus cloud heterogeneity, but really the networking tier in the back was always consistent. It's just less heterogeneous. Although, again, if you acquire a company, there's some differences there.

So I think it is different and that there's not a lot of value in sort of sneaking it in for your one application, because yeah you'll get your microservice deployment. Maybe you get better blue/

green or security or whatever you're looking for, but it doesn't help the security problem, which I think is actually problem number one that you're trying to look at.

[00:24:16] JM: What is the connection between a service mesh and zero trust networking?

[00:24:22] AD: They've got a little conflated, right? I think when we talk about zero trust networking, it's almost a philosophy more than a technology, and I think the core of zero trust is really saying like, "By virtue of being on the same network of me, nothing is granted." There's no implicit authorization that a service is allowed to talk to a database because it's on the same network."

I think that's what we're really saying is like, being on the network provides zero trust. I think that's the core of the philosophy. So technology how you implement that, I think there's sort of several different approaches. One approach is kind of software-defined networking, where you do kind of like overlaying networks and sort of virtualize the network to an extent, totally different approach, which is I think more in the camp of console and service meshes is sort of a TLS-based approach where we say, "Okay, instead of trusting our network at all, what we're going to do is distribute certificates." Every service gets a certificate that encodes this service as a webserver or this service as a database, this service as a cache, and these are kind of like standard TLS sorts that we would use of going to a secure HTTP site or something. But we're going to distribute that into apps internal to our data center instead of just internet-phasing. Then any communication service to service, our webserver to our cache, our webserver to our database, etc., is going to lean on that certificate. So the webserver has to present it to the database and prove it is in fact a webserver. Database is going to present it back to the webserver to prove it's a database.

So what they're going to do is establish a mutual TLS connection even though they're inside of a private network. So historically it would have been like, "That's overkill. It's a private network." Like, "We're trusting that virtue of the network being private, we don't need to encrypt data in our own data center. We need to authorize service-to-service, because we're behind this trusted four walls." But I think if we go to that point where we say, "There's zero trust. The four walls don't mean anything," then you have to do that. You're encrypting data within your data center. You're authorizing within your data center.

I think that sounds great, but then you say, “How do I operationalize that?” Because now I have – Call it two or three major problems. One major problems is I know you get certificates everywhere. The webserver needs it. The cache needs it. The database needs it. These cert expires so you have to refresh and rotate them all the time.

So now I have this like massive pain of certificate management and distribution. The second problem is these apps already exist. So now I have a problem of, “Okay, my webserver, my database. Great! Then I’ll use TLS. They’re not aware of TLS.” Just because I’ve provided them a certificate, I don’t want to retool a thousand applications to make them actually use it. I think that’s where Console and service meshes have sort of said, “We can step in and solve both of these probleems. If we solve the certificate distribution problem, great, that’s one big problem we’ll take off your hand. We’ll just generate the certs, distribute that managed rotation for you. But two, because we’re deploying proxies at the edge, we’re deploying something like Envoy, the proxy can intercept the traffic that is not using TLS from, let’s say, the webserver or the database, and transparently impose it.”

So instead of not having to retool a thousand applications, it’s like, “No. No. No. Just deploy the service mesh,” that thing sits alongside your app that’s unmodified and it’s transparently imposing the sort of TLS security behind the scene.

So I think that’s where we get a bit of that overlap between the discussion of zero trust and the service mesh. The service mesh is very much a technology, where the zero trust is an approach. But if you look at, for example, AWS’s App Mesh product, they don’t use any certificates. There’s no security component to it at all. So it’s a service mesh, but without the sort of zero trust kind of capabilities that you might talk about it. So I think it’s important to kind of make that distinction between some of the – With App Mesh, they focus on sort of the observability routing sort of level 7 traffic management pieces without focusing on any of the security pieces. I think they are different. In the case of Console, we sort of look at both of them and say, “Hey, two birds one stone,” but it’s not necessarily.

[00:28:18] MH: Yeah. I think that's really because when we talk to our customers, the security aspect was the problem that they were facing. So both philosophically and practically, that was what we wanted to – That's why –

[00:28:30] JM: Security rather than like blue/green deployments and service discover or something.

[00:28:35] MH: And telemetry and stuff. I mean, they want that. That has to be part of the roadmap. That has to be on the roadmap. But the challenge that they're facing on their day zero is the secure connectivity, because they're moving from an environment where the way they would solve this on-prem are basically like top of rack switches and firewalls on the racks or even close. They're just F5 hardware or Cisco hardware, something, they would solve it physically. That way through various layers of firewalling, hardware firewalling, when they move to the cloud, that doesn't really exist. The equivalent there is a crazy number of security surrounding tables or something. So that's how I would manifest, is these companies you would look and they would take their architecture diagrams from on-prem and just reproduce them using cloud concepts and it ended up being horribly expensive, not necessarily financially, just like a lot of complexity to automate the creation of accounts and routing tables. If you wanted to deploy software, you had to make sure it's in the right location and they had the right firewall rules to access things. So it's very slow to deploy.

It's also just very abrasive against the trends of things like container management platforms and schedules. It's like if the whole idea of a scheduler is you want to consolidated your compute to most efficiently sort of utilize those resources. But if you need segment physically with the network, you necessarily have so much compute that's being underutilized and it only exists for the network security.

That was the both financially expensive, complexity expensive problem that they were sort of facing that was slowing down the deploys. So that's what we sort of tackled first, which is this gives you in a way to deploy all your applications literally right next to each other and maintain that same level of security.

[00:30:18] JM: What other lessons did you guys takeaway from the container orchestration wars?

[00:30:22] MH: Well, I mean I think one thing we learned was that there's really two types of people looking at container orchestration tooling. I think we've done very well with one, and I think we've done subpar with the other type of person. One type of person that's really looking for getting as close to a platform experience is possible, getting as close to a I write code and it handles the routing and the load balancing and the scaling and all sorts of concerns for me.

Then there's a second type of person which is really looking for way to best utilize compute, and these are CI systems, heavy batch systems, a lot of financial markets for crunching numbers. Those sorts of people that are just looking – They don't need a load balancer. They don't want that sort of things. They want to give you a Docker container, or a VM, or just binary and they just want it to run and they want it to run efficiently and with the right permissions and things like that.

I think our solution with Nomad has done very well on the batch processing data side, the scheduler side. I mean, it's an excellent scheduler. People independently come to that conclusion and it's been widely deployed that way, but it's also our philosophy, our product philosophy has always been pretty focused. So Nomad is very much a scheduler. It doesn't include a load balancer. It doesn't include a lot of these bells and whistles that other container platforms did.

I think that that hurt us really early on, but it hurt us but also we philosophically followed what we believed. We're solving a problem that – We solved the problem we're trying to solve, which was having a really good scheduler that does high-performance scheduling and stuff like that, and we do integrate closely with Console and Vault and we're working on integrating more things now. But, yeah, that's one thing I learned.

[00:32:05] AD: Yeah. I mean, I think that's a big one for sure. I think is just maybe a misunderstanding I think for us of like what the market was looking for to Mitchell's point of a platform versus a scheduler. I think in retrospect we're like, "Would it have made more sense to have a Hashi stack, for example, that had a more tight integration of the different components that had more of that platform feel?" Probably, but sort of hard to say in retrospect.

I think the other thing that we sort of didn't anticipate is in the early days, it's really hard to separate sort of hype from reality, and I think one of the things that we probably did badly that I think we've sort of acknowledged was I don't think we took the Kubernetes adaption seriously enough. I think in our mind we're like, "Oh! Is this a technology that's actually going to escape velocity and people are going to use it, or is this just hype and it's not worth the product investment to integrate with at more deeply?"

I think by the time we finally realized like, "Oh, no! The adaption is real." We can start to separate some of the hype from the reality. How do we play nice from an ecosystem perspective and integrate Terraform with it? Integrate Vault? Integrate Console?" etc. I think we're probably 6 to 9 months, honestly, behind where we should have been. I think that was just sort of a product management decision of, "You only have so many hours in the day to pick what technology you integrate with and you have to choose like, is technology A, B, C or D the one you pick to integrate with, because you don't know which of these is going to be the next big Kubernetes and which of these is" – In some sense we spend a lot of time integrating with Rocket, from CoreOS. That was necessarily a time well-spent now.

So it's just really hard to tell especially in an environment that's just evolving so rapidly, and I think that was a really important sort of a lesson, is I think you have to like – You can't just make up your mind about a thing and then not revisit that decision on a monthly, quarterly basis. If only because these markets are evolving so rapidly that it's like, "Yeah, maybe we were right when we first decided it was more a hype than adaption." But the time we reevaluate it, we were sort of pretty late. I think that was a good lesson, was like check your assumptions frequently and often.

[00:34:10] MH: I think that's a really good product management story too, because we – Our customers for a long time bias towards people that were not using Kubernetes. So from a product management perspective, when we were selling Vault or Terraform or what have you, we didn't see a lot of Kubernetes in our customers. So we're like – We see a lot of people talking about. We're not seeing it. So we don't really need to integrate with it. It's not a big deal.

In the back of our mind, I mean I think something was telling us like, “Maybe that’s just the type of people we’re talking to,” and when we finally made the decision, we sort of just made it to integrate with Kubernetes much more closely. I don’t think there’s any real impetus besides they just felt like the right thing to do, even though we’re late. There’s no customer asking for it necessarily or things like that. We just did it. As soon as we did it, suddenly all our customers wanted it.

It’s one of those things where like the groups that are using Kubernetes came out of the woodwork and was like, “Oh! We haven’t been looking at Console because we were afraid of the lack of Kubernetes integration. Now that’s there, we want to talk.” So now I have weekly customer calls that just want to talk about Kubernetes. Yeah, it’s one of those things where it’s like, “You can’t take your customer feedback too directly, because it’s the absence of the customer that’s telling you something not necessarily.”

[00:35:26] AD: Same thing with the community. There are people asking in the community for Kube stuff, but the exact same thing happened in the community. The Kube users naturally selected themselves away from our software in certain cases. So we weren’t hearing issues that loudly either.

[00:35:39] JM: That’s pretty amazing. So one of the first questions I asked him was about like kind of the – Because the Kubernetes thing I felt was this pivotal moment where some companies were not able to survive or have really not done well since some decisions that they made during the container orchestration wars and it was this pivotal moment where if you made the right strategic decisions, then you’re in where you guys are now, which is really good position. You made the wrong strategic decisions, you’re in a really tough spot.

So talking about strategy, HashiCorp brought on an outside CEO. This strategy worked extremely well for Google. Are you modeling the company leadership model after Google at all? I’m not trying to flatter you at all. I’m just trying to say like it is kind of reminiscent of that, because you guys are pretty on the same page in terms of like how much input and how much respect you have for each other’s opinions.

[00:36:32] MH: So I was laughing or I think we were both laughing, because the way that came about was we floated the idea, and this was now three and a half years ago or something. It's quite a long time ago. Dave has been here for two and a half years or something. We've floated the idea of it, but really we thought it was a long shot. So like we're going to look for a more focused, we're looking for a VP of marketing, VP of sales, some executive that wasn't quite a CEO, because we didn't we could find somebody.

But Armon actually used the analogy that was like, "If we could find our version of Eric Schmidt, would we hire that person?" and I think we both said yeah. It's funny, because that was the analogy we actually used, was I think that our skills and our expertise are best served in the company not doing most of what a CEOs role is.

[00:37:15] AD: At least right now.

[00:37:15] MH: Yeah. Yeah. So if we could find someone that we felt was culturally aligned, aligned with the vision, then we would entertain the idea. Mutually, for Dave to join, for us to want him to join, it took over a year of us talking to him. It was not like love at first sight, we found the person and we're like, "Oh, yeah. This is going to be the person," and we felt really good about Dave when we first met him, but it took a lot of meetings, a lot of dinners, a lot of whiteboarding to get to the point where both of us were comfortable doing it, and it has been great.

[00:37:48] JM: Was there some particular weakness you guys were wanting to hedge against when you were looking for an executive CEO? Because if you look at like the Google history, it seemed like they knew that they were a little too like, "No managers, and let's do everything." They knew that about themselves. So they kind of looked for somebody who could correct those sort of lack of adult supervision things.

[00:38:12] AD: Yeah. I think it goes back to what we're talking about I think earlier, which was over the last three years, sort of the evolution of the commercial strategy. Because I think the forcing function really for us really became – In the early days, like I said, we were a well-funded research product and really didn't know what the commercial was, and I think by the time we'd sort of made that decision to say, "You know what? We actually want to focus on the enterprise as being the sort of target customer-based for us." Then all of a sudden me and Mitchell had to sort of look back and say like, "Okay. Well, are we the right people to build an enterprise sales team and an enterprise marketing function, like an enterprise go to market sort of story?" I think it was sort of painfully obvious, the answer was no.

So I think that was really where the weakness we were trying to balance out was I think me and Mitchell continued to be very, very involved on the product and engineering side in terms of evolution of product strategy, architecture of the products, things like that, but we needed someone to round us out on sort of the go to market. How do we position the company? How do we sort of build the right sort of executive team that has the enterprise expertise? I think that's where we felt like what we needed was not necessarily a boss, but a partner. I think that's really when we found Dave, what we were looking for was, "Hey, we want someone who will be sort of our partner on thinking about sort of the enterprise go to market side while we focus on sort of the product engineering side, and I think that's continued to be a pretty healthy sort of partnership between the three of us.

[SPONSOR MESSAGE]

[00:39:47] 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]

[00:41:54] JM: How have you developed that sales and marketing perspective, because my sense is I've never built out the sales marketing team. My sense is that this time is really different than any previous time in software, because you got these giant cloud providers and you have to have this kind of like friendemy relationship with them, and then on the marketing side of things, things are also changing, because the developers is having more buying power than ever. So you're not necessarily marketing to a CIO in a suit and tie. You're marketing to a developer. How have things changed and how are things the same?

[00:42:31] MH: I'm going to break that down a little bit, because I think there's a few different things. From a – Let's talk about sales, from a sales perspective, I think a lot of it given our strategy choice is the same, and a lot of it is different. The same is that it's enterprise sales. It's still you got to talk to the person. You have to be present. It's a long sales cycle. It's the same type of like budgeting and things like that. So that's a lot of the same.

I mean I think a lot of the different is the understanding that the practitioner is the person making the decision to adapt the technology, not necessarily the person with the budget. So the way we've done that is I think still today, but for a very long time at least, we didn't approach any companies to even try to sell to them unless we knew 100% that they were using our open source. We didn't want to explain what the tool did from a practitioner standpoint to them. We wanted them to – We need to align with an internal champion that already use the tool and likes it and understands why they believe their company needs it, and we work together internally and us externally to get a deal done to sell into that company, and I think that's different. I don't think I would hazard a guess that Oracle doesn't do that. So I think that was very different.

On the marketing side, I mean for a very long time our only marketing expense was our annual user conference. We didn't sponsor any conferences. All the conferences we attended we're speaking at, so they were free. We still today have never spent anything on ads, digital ads. We don't do any of that kind of marketing. So that's very different. It's open source practitioner base. So I think our marketing teams are very good at understanding that the journey is really why would you use it? Actually using it, reading the docs, engaging, contributing, like there's a practitioner path that doesn't involve dollar signs at all that gets them all the way to the point where they're successfully using it and they're very likely contributing in some form back to it and without paying anything, and you have to nurture that pretty much. So I think that's the difference there.

[00:44:34] JM: Some people try to size the market for enterprise software, and to me a modern sizing seems almost impossible. Like the total addressable market can't really be calculated, because it's impossible to know how much software a large bank is going to need in five years, or how many oil refineries are going to need Terraform. Do you have any useful metrics that allow you to think about how big this market is?

[00:45:01] MH: I'll let Armon think about the metrics. I mean I think qualitatively, something that's really interesting we mentioned at HashiConf is that everything is growing. So when we go into a customer, they're bringing us in usually because they're in the midst of a cloud transition or something like that. But despite that, if we now have customers that have been – Customers long enough that we could look back and spend three years and we look back and it's like, "Yeah, their cloud usage has gone way up, but their physical data centers are growing too. They're not really shutting them down. Their footprint is growing there too. The number of clouds is growing. The number of applications simultaneously is growing." I'm sure that's all correlated, but everything's sort of pushing for more, and at the same time more and more companies that traditionally weren't software companies are still in their own transformation of becoming software companies that started decades ago.

So this all points to the fact that we think the total addressable market is much larger than what was available for let's say virtualization, that phase, but it's also much larger than what it seems like it's going to be right now. For us in particular too, because we have our distinct product, we view it as even larger, because we could just sell Terraform to every company, and that's a really, really big successful company. But we have a totally different group that sells Vaults there, with totally different group that buys Vaults, and totally different group that buys Console. So we view the total addressable market really as all the cloud adaption maybe like times four, and that I think is interesting.

[00:46:26] JM: Does it make you want to go after the market more aggressively or do you prefer to take that power of the market and allow it to let you be more deliberate and just sort of go at your own pace and so that you can retain calm and make sure you don't fall in your face?

[00:46:45] AD: I think it'd be a nice luxury to go at our own pace certainly. I think there's a few sort of competing pressures I think in the market. I think on one side, there's a reality that cloud is growing at 80% annually if you look at just the numbers reported by Azure and AWS and Google Cloud.

The market is growing at that clip. So there is this poll that means naturally just from sort of our customer base and our user base to sort of satisfy their demand. So when the tide is rising at 80% year over year, you have to rising at that percent just to keep pace, otherwise you're not even keeping up with customer demand.

So in some sense, because the market is itself growing so rapidly, I think there isn't as much of that luxury to grow at your own pace. I think you have to be sort of at least trying to keep up with your own customers if nothing else. So I think that's been sort of this kind of forcing function for us. I think when we'd look sort of our growth, our hiring, it's really as organic as possible. It's just how do we keep up with just the inbound demand of our customer base, our user base, our community, filing issues, opening tickets, trying to buy software, engaging with us in some way, and how many people do we need to sort of just keep up? I think that's been the biggest sort of driver for us.

I think last year we went from 150 people to 400 basically today. I think looking at sort of this year, somewhat less dramatic, but probably in that same sort of 80 to 120% growth expected this year, and part of that is just how do you keep up with the baseline, but then how do we make sure we keep up with all of the inbound basically? So I guess going back, it's like, "I wish we had the luxury for our own sanity's sake to grow it slower, but I think it's not as much of an option in this market.

[00:48:37] MH: Yeah, and I think there's some people and companies that would say like, "You could. You could slower." In addition to what Armon said, I think the problem is the type of software specifically we sell is they're choosing their sort of company-wide or business unit-wide at least solution for provisioning. They're not really using Terraform and cloud formation. They're choosing Terraform because it represents the ability to standardize. The problem is if we're not in the room, if we can't get in the room, they will choose something else and they will choose a different standard tool and we're probably out for 5+ years. They chose their technology for a long time.

So it's really scary and stressful and difficult, because we would like to grow slower and things like that, but a lot of our fear is – And we've had this happen a couple of times, where you go to a customer and it's like, "If you talked about six months ago, maybe, but we are already chose our three-year plan. This is our three-year plan. Come back."

That hurts, because you don't even have a foot in the door. So a lot of our motivational growth is you can make your company more efficient. You could have individual sales people do more deals, but time – You can't change the amount of time you have. So the more people you have, the more time you have to work with to actually spend an hour meeting with somebody. You can't have one person take more meetings. There's just a limitation on time. So that's the balance we have to make, is how do we grow a successful, sustainable business while at the same time being as present as we can so that there's awareness in the market.

[00:50:05] JM: Let me ask you one more question before you go. What's the best and worst part about working on a somewhat equal level with a close friend?

[00:50:14] MH: I think the best part, it's kind of like a sibling in the sense that I think there's a really deep understanding of what we want to do, how we're feeling, when we need each other, when we don't need each other. I think that that's very strong. So that's really nice. That's just unspoken in a way.

I always say, I think for me the downside is there's – I think there's a lot more emotional investment that I need to have. So it's like on one side it's just a job. Yeah, it's your company, but it's just a job. Then on the other side it's like even if you are in the pit of despair and you're like, "It's just a job. I don't care." It's like, "But I care about him or I care about the other people I work with," and I think that's hard by having a close friend. It's less of a business relationship.

As much as I love working with Dave, I can't say I feel the same way about Dave, although he's awesome. It's very different.

[00:51:03] JM: Cool. Thanks, Mitchell.

[00:51:04] MH: Yeah, thank you.

[00:51:05] JM: Okay. Can I get your answer to that question?

[00:51:08] AD: My answer is probably really similar to Mitchell's. I think I describe it as almost like – It feels like we're left and right brain. Sometimes it feels like we can – It's almost telepathy as we sort of talk about product decisions and company decisions, and I think that's – I think that only comes out of like that chemistry of having like been best friends for so long that it's like a body tick is just enough to sort of like you'll read into that. You don't need to have that conversation about it.

I think there's that efficiency of the communication, and I think the other strength has been a startup is not I would say an easy adventure. There's a lot of sort of for every up, there's many downs, and I think having someone else that you can sort of talk to and relate to and commiserate with. For whom it's not totally abstract. Even my partner, it's not the same when I go home and tell him, because he's not close enough to the company and to the problems to really understand it in a way that like Mitchell totally gets. I think that's a huge plus.

Downsides? I honestly don't – I don't even know what I would say.

[00:52:06] JM: We've got to let Mitchell leave the room. So you can say the downsides.

[00:52:10] AD: Probably every once in a while when we sort of disagree on things to start trying to — maybe it takes longer for us to actually come to a decision than if you were sort of instead of being two-headed, if there's a one-headed sort of leadership, you probably get to a decision faster. But at the same time, I think that has its own pluses, which is like you don't rush into decisions when there're sort of other opinions and viewpoints on it. So that's probably the biggest challenge I think.

[00:52:34] JM: So you've got this change that we kind of discussed earlier where the developer can now buy things from a dashboard, but then you've gotten got a CIO or a CFO that sort of puts a gate up and says, "Okay, we don't want you buying all of that," and to some extent that's inconvenient, obviously, for the developer. But also we've seen the rise of the cloud cost management companies, which is a gigantic business. There's like five major companies doing cloud cost management at least, and that's because you get some companies that are just spending tons and tons, because they gave carte blanche to the developers and like, "Oops! They're spending a lot of money." What do you think is going to happen there? Like is there going to be like a tightening or a loosening of the wallet?

[00:53:18] AD: I mean, I think your sort of observation – I think we're seeing the market try and say, "How do we solve this through a product lens?" because I think there is this this natural tension, and we just see it all the time, which is you talk to a CIO who's sort of like hand-wringing that they're like, "I gave my developers access to the console and the first thing they do is go spin up an 8X quadruple large with 6 TPUs attached to it because it's cool, and we're getting billed $20 a minute," and then they leave it running for four months because they forgot about it.

So I think you hear that super, super common. So I think there's this tension between how do we make the developers productive and give them sort of self-service access and let them run and go? With the other side which is like, "How do you maintain some sanity around cost management, around security, around some of the governance capabilities that you need to be a sane business?" You can't have developers who are spending unlimited amounts of money and opening up your whole network to the public internet.

So I think the right answer is probably something in the middle as with most things in life. I think that's been a big investment for us, is how do you give CIO that, "Hey, install policies that control what you can do, when you can do it, who can do it?" then make that part of a self-service workflow for developers. As long as they're playing within the sandbox, as long as you're not launching an 8X quadruple large, great. The system allows you to go through in sort of an automated way. It gives you that self-service experience.

But that moment you try and step out of the sandbox, how do we say no in an automated way? Because I think that key is the automated way, because otherwise what you end up seeing is organizations will say, "Okay. Well, great. I'm just going to create a checkpoint for you where you have to file a ticket. I'm going to manually review what you're doing. Now, there's no self-service. You have to wait four weeks for someone to review a thing."

So I think as long as you can impose those controls in an automated way without breaking the self-service, it's sort of a win-win. CIO gets what they want. They get their governance. Developer gets what they want, which is no one is in their way. They can self-service. But I think it's finding that balance of usability, governance, automation that makes that all work. I think that's been an area we spend a lot of time in.

[00:55:23] JM: Okay. Armon, we'll leave it there. Thank you for coming on Software Engineering Daily. It's been really fun talking to you.

[00:55:27] AD: Thanks so much for having me. It's been great.

[END OF INTERVIEW]

[00:55:32] JM: Kubernetes can be difficult. Container networking, storage, disaster recovery, these are issues that you would rather not have to figure out alone. Mesosphere's Kubernetes-as-a-service provides single click Kubernetes deployment with simple management, security features and high availability to make your Kubernetes deployments easy. You can find out more about Mesosphere's Kubernetes-as-a-service by going to softwareengineeringdaily.com/mesosphere.

Mesosphere's Kubernetes-as-a-service heals itself when it detects a problem with the state of the cluster. So you don't have to worry about your cluster going down, and they make it easy to install monitoring and logging and other tooling alongside your Kubernetes cluster. With one click install, there's additional tooling like Prometheus, Linkerd, Jenkins and any of the services in the service catalog. Mesosphere is built to make multi-cloud, hybrid-cloud and edge computing easier.

To find out how Mesosphere's Kubernetes-as-a-service can help you easily deploy Kubernetes, you can check out softwareengineeringdaily.com/mesosphere, and it would support Software Engineering Daily as well.

One reason I am a big fan of Mesosphere is that one of the founders, Ben Hindman, is one of the first people I interviewed about software engineering back when I was a host on Software Engineering Radio, and he was so good and so generous with his explanations of various distributed systems concepts, and this was back four or five years ago when some of the applied distributed systems material was a little more scant in the marketplace. It was harder to find information about distributed systems in production, and he was one of the people that was evangelizing it and talking about it and obviously building it in Apache Mesos. So I'm really happy to have Mesosphere as a sponsor, and if you want to check out Mesosphere and support Software Engineering Daily, go to softwareengineeringdaily.com/mesosphere.

[END]