EPISODE 451

[SPONSOR MESSAGE]

[0:0:13.6] JM: Spring Framework gives developers an environment for building Cloud Native projects. On December 4th through 7th SpringOne Platform is coming to San Francisco. SpringOne Platform is a conference web developers congregate to explore the latest technologies in the Spring Ecosystem and beyond. Speakers at SpringOne Platform include Eric Brewer, who created the CAP Theorem.

Vaughn Vernon who writes extensively about domain driven design and many thought leaders in the Spring Ecosystem. SpringOne Platform is the premier conference for those who build deploy and run Cloud Native software. Software Engineering Daily listeners can sign up with the discount code SEDaily100 and receive a hundred dollars off of a SpringOne Platform Conference Pass, while also supporting Software Engineering Daily.

I will also be at SpringOne reporting on developments in the Cloud Native Ecosystem. I would love to see you there and have a discussion with you. Join me December 4th through 7th at the SpringOne Platform Conference and use discount code SEDaily100 for a hundred dollars off of your conference pass. That's SEDaily100 all one word for the promo code.

Thanks to Pivotal for organizing SpringOne Platform and for sponsoring Software Engineering Daily.

[INTERVIEW]

[0:01:43.0] JM: Ranga Rajagopalan is the co-founder and CTO of Avi Networks. Ranga welcome to Software Engineering Daily.

[0:01:50.5] RR: I'm happy to be here Jeff.

[0:01:51.7] JM: Today, we're going to talk about load balancing and some modern approaches to load balancing. Let's start with the idea of load. It's 2017, what are the different types of load that exhibit themselves on a computer system?

[0:02:09.2] RR: Load is nothing but a demand for any system. For example it's the amount of demand that you post on the system. More specifically for a computer system there are different types of load, the number of users who are accessing the computer system is a form of load, the request that they're posting to the system is a form of load, the amount of two-port or bandwidth that they are consuming has a result of those request or transactions, is a form of load. So, these are different types of load and the demand that they post on the system.

[0:02:56.2] JM: When a computer system is under lots of load that system needs to distribute that load so that no single component gets overwhelmed with the traffic, with the load that results from a traffic, and this is called load balancing. What are the different parts of an application that need to be scaled or need to have load distributed across them, to do this load balancing operation?

[0:03:29.2] RR: Sure, a closely tied concept to load is capacity. So, capacity is the ability to meet the demand or meet the load that’s exposed on the system. And, there are different bottlenecks in the system when there is more and more, demand on the system. Let's take some examples, when you increase the number of users who are accessing the system, let us say it's really a large application and there are literally millions of users accessing that application. So, now you have a million users logged on each user has been authenticated and each user has a session associated with that specific user.

So, that is one form of load. Now, let's say each user actually starts interacting with the application. An application can be your Facebook app on your phone, for example. As you start interacting with the application let us just say you are scrolling down your Facebook app you will constantly see that it's updating notifications, it's updating pictures, it's updating videos, each and every one of them is a request that goes back to the application and then a response comes back from the application. And, so as the number of users increase the number of requests and responses keep increasing. So depending on the load, different parts of the system can become the bottleneck.

So, we need to increase capacity to meet this load and there isn't a single server on a single storage that is capable today of literally scaling – increasing its capacity dynamically. So, a common technique is we increase the capacity in the system by using multiple servers and when we have multiple servers or multiple storage units then we need something that will distribute the load across those multiple servers and that is the load balance and that's what load balance is supposed to do.

[0:05:54.8] JM: This load balancing can take place within a single data center or we can scale up across multiple data centers, or maybe we're an enterprise company that's been around for a while we got a lot of on PRAM servers and we also have a Cloud. We've got a hybrid model where part of our infrastructure is on PRAM and part of it is on a data center on AWS or Google or Azure. When we would want to scale in to these different areas of our infrastructure like how would we want, would we want to scale in to the data center, or we do we want to scale in to our on PRAM servers, does it matter, what are our different options and how we should choose them?

[0:06:46.6] RR: Great question. Usually, for scaling a single application you almost always want to scale that application across multiple servers and storage instances within a data center. That's how almost all applications start, that's how they are deployed and through that you front end multiple instances of servers or storage with what is commonly called as a load balance. Now, almost all applications today especially enterprise class applications or even a lot of widely used consumer applications, are also deployed across multiple data centers there are several reasons for it.

First reason is just user proximity. The network latency delay which is just going by the speed of light is at least a hundred milliseconds coast to coast. So, that is just a huge delay users are going to see in response times if they have to reach out all the way across from east coast to west coast, for example. So, you almost always want to serve users closer to where they are, and that means multiple geographically distributed data centers, or regions if you are using a public clock. The other reason why you want to deploy applications across multiple data centers is high ability on disaster recovery.

So, if one data center completely on one region completely goes down, then you still want the application to be up you want the users to be served so you almost want a backup data center that will serve applications and that will serve applications and that will serve users. And the third is just for maintenance, if you want to bring down one data center on one region for maintenance then you have the ability now to reroute users to other data center. Solve them from that data center while your primary data center's old applications are going through maintenance.

[0:08:48.8] JM: We will come back to the question of where we should be doing our scaling. Let's talk about what we're scaling? There is the trends in application development that are changing because of Cloud, because of infrastructure as a service, because of changes in programming languages. What are the important trends in application development that are changing how we want to do load balancing?

[0:09:18.1] RR: Sure, let's take a step back. So in the beginning all applications were client server. So, that means the application was completely monolithic there was just one blob of the application, if you will, and clients instruct the application. In the 2000's even until now this sort of, became slightly broken up into a few tiers – where you almost always have a front end tier, that is usually called the web tear or the presentation layer. Then the actual application logic which is called the application tier and the storage of the persistence layer, which is the database. So this is usually known as the three tier application in the industry.

And, this pattern started appearing in the only 2000's and this is the most common pattern I would claim all 90% off applications in history are deployed in this pattern. But, this is also changing. The presentation has to last the application tiers are now being broken down further into microservices where each individual service performs a very specific function and there are few reasons for it. Primarily reason for this is being just agility. Imagine you make a simple change in your application of the that three tier model and imagine running through say ten thousand test cases to roll the change out to make sure that you haven't broken something, right.

That's just take a long time to make a simple change, whereas if you have broken down the business logic into say 15 or 20 microservices, and now when you make a change you only have to run say 500 test cases to roll that out then you can lock off. So, deploying these microservices almost all these means more agility and being able to update and roll off more changes faster. So, this is the fundamental way in which application development and deployment is changing. Another trend that is also making this possibility today is containers which makes application development packaging and the deployment, a lot better than what it was before.

[0:11:55.3] JM: We are breaking our applications up into microservices, we are putting those microservices in to containers. How does that change how we want to do load balancing?

[0:12:05.9] RR: So, in the traditional treaty architecture you still need a load balance in front of every tier, you would have a load balance in front of your presentation tier that would spread the load across multiple instances of those, you would also have a load balance in front of your business logic or application tier and if necessary you would have a load balance in front of your storage or database tier. But now, the business logic split apart into 15 or 20 different micro services and each one scales independently, each one is a service which has its own high daily requirements independently. So, what you really need is a load balancer in front of every one of those microservice.

So, where you needed a single load balancer in front of your application tier you need 15 or 20 load balancers in front of your microservice or setup microservices. And, this not just increases the number of load balancers you need, it also changes how your load balancers are deployed because it is no longer feasible or possible to have monolithic load balancers, where all the traffic is routed through that single choke point so it does not able to load balance. But now you have to have 15 or 20 different load balancers that are deployed closer, where the microservices are running in a distributed way.

[SPONSOR MESSAGE]

[0:13:51.7] JM: You're programming a new service for your users or you are hacking on a side project, whatever you are building you need to send email and for sending email, developers use Centigrade. Centigrade is the API for email. Trusted by developers, send transaction emails to the Centrigrade API. Build marketing campaigns with a beautiful interface for crafting the perfect email. Centigrade is trusted by Uber, Airbnb and Spotify. But anyone can start for free and send 40,000 emails in their first month.

After the first month you can send 100 emails per day for free. Just go to centigrade.com/sedaily to get started. Your email is important, make sure it gets delivered properly with Centigrade a leading email platform. Get started with 40,000 emails your first month at centigrade.com/sedaily, That's centigrade.com/sedaily

[INTERVIEW CONTINUED]

[0:15:06.4] JM: This load balancing that takes place, it's easy to imagine we want to scale an application and we're going to scale it up. We are going to do something to scale it up. But we actually need to get data about how our overall application architecture, how that application's health is performing. So we actually need to have some, kind of, monitoring in place to be able to respond to changes in the environment because change in the environment are detected by changes in the monitoring.

What are the methods for gathering application health data? And then what should, when are the indications that health data is saying, "Okay, we're going to need something to scale, we need to do something to balance our load."

[0:15:59.5] RR: Right, great question. So, in the traditional way when load balance was first deployed we have established a need for load balancing. That is we need to scale applications and only way to scale them is by deploying multiple instances of that function. But what we have not discussed is how do you determine how many instances you need? That is, how do you determine how much capacity you need. So, this has been very static to reach them. Which essential means the operator would be monitoring the metrics that correspond to load, which often can be the number of users, the number of connections from those users, number of request going through the system, the bandwidth the throughput of the system, the number of transactions the users are making.

So, these are the common metrics that the administrator would be monitoring and because it's a human they would be monitoring macro trends. They would be monitoring for example, what it is this week or what is it this month? And so on. And, then they would be monitoring it in a more coarse way and increase capacity for applications if they see an upward trend or decreased capacity this year, a downward trend

But because this happens by humans it's quite infrequent and this also means there's a lot of ways to capacity because you have to account for changes that have happen for example within the week, within the month or sometimes within the day. So you have the capacity that would meet the peak capacity for example, every month. Or the peak capacity every quarter and so on.

So, that meant a lot of wastage that meant a lot of standard capacity. Now, so the metrics you still monitor in a modern infrastructure are still the same. Except the difference is, it is going to be just in time, the difference is going to be – the difference between for example, a car that you have parked in your garage which is there whether you need it or not. Versus Uber that you rent out, is exactly when you need it and when you don't need it. You don't own that piece of infrastructure.

[0:18:35.7] JM: As we're athering this monitoring data and understanding how the health of our application trends we might notice that, "Hey, you know, every afternoon we get a huge traffic spike at 4 o'clock." And with that in mind we don't necessarily need to have our load balancing scaling in response to that demand spike. We could learn to scale in advance of that spike. Or we're going to have fewer customers at a certain time maybe we can scale down our capacity. What are some good strategies for this predictable scale-ability, for this predictable load balancing?

[0:19:25.7] RR: Exactly. So, step one is a moving from a static coarsely monitored environment where the human monitors this and then decides by minor action. To increase or decrease capacity to more automated deployment model with the load balancer itself monitors load versus available capacity and then take some actions to dynamically increase or decrease capacity of the application. The next step in the evolution is learning the pattern of load on the application using machine learning techniques.

Almost all applications follow a pattern that is based on either time of the day, or day of the week, or sometimes even specific times of the month and so on and so forth. Once you've learned the pattern of an application then the algorithm can adopt itself to always keep capacity ahead of the load. So, for example if it's a business application that starts peaking at 8 a.m. the load starts going down at say 5 p.m. then you can't bring more capacity online say at 7:30 a.m.

So that is sufficient capacity available by 8 a.m. and then you can bring down the excess capacity let say by 6 p.m. So these are some techniques that you can do based on learned pattern of load on applications and this makes it very efficient in terms of how you actually spend your cost for that application.

[00:21:22] JM: You worked at Cisco for many years. I think a decade, something like a decade. And you saw the infrastructure management changes that came with the Cloud. You saw the before and after and we're going to get in to a discussion of how enterprises do load balancing. But first I want to understand how it could – because there's a lot of enterprises that were around both before and after the Cloud.

And you've probably seen as much as anybody else how those enterprises have changed their infrastructure overtime. What are some of the infrastructure challenges back then that, you know, when you were starting your career at Cisco, or in the early years of your career. What are the infrastructure challenges back then that newer programmers take for granted, that we don't have to do at all?

[00:22:18] RR: Sure now, programmers or developers almost always want something from the infrastructure. So traditionally it's servers, it can be storage, it can be access to services like load balancing or security like Firewall and so on and so forth. Especially, I would say in the mid 2000s when you needed more compute capacity for example, you would know how to open an I.T. ticket. And pretty much all servers at that time you'd still have to buy hardware so that meant typically weeks of lead time when I.T. actually ordered it, racked and stacked it.

And then you would have those source. So that meant delays, so that meant you had to forecast and you have to be accurate in your forecast. Your budget had to be approved and you had to buy and deploy the servers. It's not the situation. One situation changed that somewhat.

So now you are able to break up your [inaudible + 00:23:21] servers in to smaller pieces, that meant you could give out which machines to developers.

Quicker but still it would mean days of reading instead of weeks because you would still have to open a ticket and someone have to get to it and clear the VM make sure it had the right connections and everything and then make that VM available. That changed then in to some service smaller, which is what we see today with that Cloud. Where there is no one in between the developer and the infrastructure. The developer goes to a consul, obviously it's an API, to get access to compute.

It can be storage every [inaudible + 00:24:08] would be network resources but it is connectivity it's a sub net, or it's not balancing, or it's security encoding Firewall [inaudible + 00:24:16]. So that fundamentally changed how soon or how fast developers have access to infrastructure, which made everything much faster of course.

[00:24:29] JM: As we said earlier, these large enterprises that in the past had to do the procurement process of racking and stacking their servers that they ordered. Today they have access to the Cloud but they haven't moved their entire infrastructure in the Cloud. There aren't many large enterprises who are just getting started with the Cloud. In this kind of hybrid deployment where they have on PRAM and Cloud resources. What are some of the infrastructure management challenges that these enterprises encounter?

[00:25:09] RR: So enterprises face a few challenges with this Cloud model. So first challenge is the ability to migrate their applications from data center to the Cloud. This is a non-trivial problem, it is non-trivial because most applications are still deployed as [waching + 00:25:28] machines with their own packages and differences across pocket versions security acquired, and so on and so forth.

So they often have rules that have hardcore [inaudible + 00:25:41] IP addresses and then so migrating them to the Cloud often is a remarkable task. The second one is of course all enterprises are just definitely worried about security and just to understand the right security models and security postures, compliance and audit for deploying applications across Hybrid Cloud is taking a lot more time.

And thirdly, newer applications that are being build, even as we speak here today on the best fit actually for the Cloud. And those are often moving a lot faster than traditional applications that are being deployed to the Cloud. So, what we have now is we have a situation where there will be multiple data centers and Public Cloud Regions that have to be managed within the same enterprise for the foreseeable future. And, we have a pattern where a lot of newer applications and newer departments are being deployed in the Public Cloud.

A lot of the applications are slowly migrating to the Public Cloud. Things like development and testing are moving to the Public Cloud but you also have a substantial portion of the more traditional applications and applications and production, that are still running with the data center and I suspect that will continue the case for many years if not a decade or more.

[00:27:17] JM: Your company Avi Networks builds elastic load balancing technology that suits the needs of the type of enterprise that we're talking about. Describe what your products does?

[00:27:34] RR: We have been named, next generation load balancer, what I mean by that is as the Cloud architectures of the world from more hardware-centric and individually managed architecture, to more software-centric elastic and centrally managed architectures. As clustering has evolved from more individually managed clusters, to centrally managed and centrally available clusters. Similar to that, we have built a centrally managed load balancing service fabric that works across the cluster today.

It's all software and you really go to a single instance or our web page to access the service and everything that happens by that control or by master process, including providing to service, dealing the type availability and deploying this whether in the data center or across Public Cloud. So all of these are centrally managed and it's also fully elastic because it's software and we're able to scale the capacity available, depending on the load that's put on the system.

[00:28:57] JM: And, how does that change my work flow as somebody who is leaving an engineering company, that I have this new load balancing technology where I have a centralized control plane and I can set up load balancing rules and what not. What exactly changes about my work flow?

[00:29:22] RR: Yup, few things change in the fundamental way so instead of an army of operators who have giant spreadsheets, where they manage a fleet of hundreds of load balancers. You really have one place to manage all the other load balancers and that is a huge advantage in terms of just operations and not making mistakes and so on and so forth.

The second one, is elasticity and scale. So because it's a software it's completely elastic so that it means it only uses as much capacity as you really need. And with that I make the scales the available capacity as you need it. The third is it completely unlocks different application architectures so previously with the monolithic load balancer, was a good fit for more traditional tricky architectures where traffic naturally flows through a choke point. A choke point in the system where they can perform load balancing.

But now with applications disintegrating in to microservices. You also need to disintegrate the load balancer in to more distributed services, or service proxy, or service mesh, as we call it. And this is only possible if you fundamentally rethink the architecture into a more Cloud like architecture. So those are few ways in which the Avi load balancing platform changes how you use, deploy, and consume load balancing.

[SPONSOR MESSAGE]

[INTERVIEW CONTINUED]

[00:32:16] JM: Let's talk about how this is implemented. You have these agents that are sitting in every service because we want to scale the different microservices in our application based on certain rules that we want to set or certain schedules that we want to set. So you've got agents that are sitting alongside every microservice and the agents are communicating with a central control plane. Describe that architecture in a little more detail.

[00:32:46] RR: Yeah, that's exactly how it works Jeff. So that is a central control of management plain and that is the API end point. What I mean by that is that it's where you create the service and dynamically the central control plane creates agents or proxies, as many as you did for every microservice running in the environment. And it's the controller's responsibility to manage the life cycle of these agents or proxies.

So it creates them, it destroys them, it monitors them, it upgrades them, it scales them out as necessary. And, when you create a service in the Avi system corresponding to a microservice, the controler is responsible for example for allocating an IP address for that service. It's responsible for mapping that IP address to a DNS, or a domain name entry. And it's also responsible for pushing the policy out to the agent which actually performs the load balancing function. So the actual work of performing the load balancing, the data plane is the agent or the proxy. The controller primarily is the brain service system.

It's responsible for creating this and managing this agent and pushing down the policies or the service contribution information to these agents.

[00:34:13] JM: We've done shows about a service mesh. There is an opensource project around the service mesh like STO and Linkerd, where every service gets a sidecar container and the sidecar container monitors and aggregates the data in the service instance and that can help with load balancing. You know, just kind of an opensource model that sounds somewhat similar to this but it's probably a little less managed and I mean there's probably some other differences. How similar is that approach? It's kind of a patter maybe the sidecar pattern or whatever you want to call it. How similar was that with what you do with Avi?

[00:34:56] RR: Exactly, service mesh is a term that really just means that the set of services you need like load balancing, are deployed in a distributed way or in a form of a mesh or a fabric in your system. Now there are 2 or 3 patterns on how you can actually deploy and use a service mesh. The first architectural pattern or deployment is as you described, which is every instance of a microservice has an associated sidecar, or a proxy, or an agent with it. So this is the most granular approach of how you deploy these proxies in a service mesh.

One step below in terms of granularity or about in terms of courseness is if you have a proxy per microservice. So if you think about a traditional treated application, which typically has the presentation, application and database here. On your break up the presentation – the application TRN to say 50 or 20 different microservices and each microservice let's say has 10 instances running in it.

With the sidecar model, let's say you have 20 microservices with 10 instances, that's 200 instructors. So you would have 200 sidecars associated with every instance of the microservice. With per microservice model you would have 20 instances of proxy with each proxy serving a microservice.

And all of these ultimately run on nodes whether those nodes are actually VMs or [inaudible + 00:36:49] or they are actually physical bare metal hosts. There's also the third model where the proxies deployed as MV node. So this is a port node model. Say, if these 20 microservices are deployed across five nodes then you'll have 5 proxies. So these are sort of three different deployment models of a service proxy.

And like everything else that are trade offs and advantages and disadvantages in terms of scale, performance, security, cost and so on and so forth.

[00:37:31] JM: The model that you have or you have these agents deployed on every service and you have these centralized control plane, it seems like you could do more than Load balancing. You can build other services that are supporting features that you would want in a sidecar. Are there some other things that you building other than load balancing that you can get out of having this sidecar agent?

[00:38:00] RR: Yeah, so we actually support two models, it's a support model where that is a proxy part of service on agent per microservice. We also support a model where there is a proxy on every agent on every node.

[00:38:13] JM: I see.

[00:38:13] RR: Yeah, and the set of functions that we provide are load balancing and what is called as service discovery, where every microservice is associated with the name typically via DNS. We also support a set of security functions. That includes things like for example very simple black list white list on who is allowed to talk with service to all the way to unless having what is called a web application Firewall, where we have threat prevention against attacks on HTTP request and responses and so on and so forth.

We also do other things like authentication where the authentication function is off loaded from the application in to the proxy. Then other things associated with CICD where you want to deploy an application using a blue-green deployment pattern. You want to auto scale applications, which we spoke about briefly earlier. So, we provide a comprehensive set of services that you need for an application deployment. We today do that in all deployment form factors, whether that’s bare metal, whether it's a machine, it's a container and whether it's on a data center or on a public cloud as well.

[00:39:43] JM: I'm glad you caught my confusion there because I was a little bit confused as to whether you deployed a per service agent, or a per node agent. Just so people know the difference because I think this is important. Let's say I have my purchase item service that processes purchases for any user and if I'm running a huge eCommerce site then I'm gonna want to spin up maybe, 50 instances so I can process – so I can have 50 nodes that are processing all of the purchases that are coming through my system. I could have a single agent that manages and monitors all 50 of my purchase item nodes, or I can have 50 agents, I can have one on each service – purchase item service node to more closely monitor them. How should I choose between those two models?

[00:40:38] RR: Right, that's a great question. So, the typical deployment pattern is that, you have a cluster of nodes, and that cluster depends on what kind of workload you want on that cluster. They vary anywhere from let's say, 10 to 20 nodes to a few hundred nodes, say 500 at a thousand nodes. And each node typically has a computer memory associated with – again anywhere say, 8 to 16 or even 3 to course, and anywhere from say, 32 gigs of RAM all the way to 128 gigs of RAM and so on and so forth. And then you usually want to manage this cluster just like we manage load balancers centrally. So you would create micro-services on some central orchestration system.

Say, you create a purchasing micro-service and when you create that you'd say I need 20 instances or 50 instances of this purchasing micro-service. And so, the orchestration system now would spin up 20 or 50 containers and schedule those containers across these. Let's say you have a hundred node cluster across these hundred nodes, and you'll also say each instance of this needs, let's say, one core and 4 gigs of RAM. So now, the orchestration system is also doing resource management. It says, "Okay, on node 1, I'm running one instance of the container so if I have 8 cores, I will be able to consume 1 core, I have 7 cores left."

So when you create an O-Ring Microservice, now it also schedules that across the cluster. So, this orchestration system performs resource management, scheduling and management of all the microservices and the computer resources and memory resources on this cluster. So, typically, you have two deployment patterns, or three deployment patterns like this [inaudible + 00:42:34]. So, one deployment pattern is every node, let's say you have a hundred nodes, every node is running one copy of this proxy or an agent. And in this deployment model you'll have a hundred proxies running in the system and every proxy will be responsible for servicing all the containers running on that node.

Second model is your deploy proxy for every microservice running in this cluster. So, with that, if you had 20 microservices or 50 microservices, you will have 50 agents of proxies running in. Third model is the sidecar model, at every instance will have its own copy of proxy. Now, the trade-offs are ready and are different. So, in the first model, when every node is running a proxy, you would usually have the fewest number of proxies running there. So that means it is more efficient in terms of, how it utilizes the resources and also, in terms of how they scale the number of proxies that are running in the system.

What you lose here is, of course, the sharing because every node is running one copy that means if you have 10 containers on that, they are sharing the resources available to that proxy. In the second model, when every microservice has its own proxy, the trade-off there is that, every microservice has access to its own proxy and it is also – the scale will be under control because, usually, the number of microservices will be a much smaller number, than the number of instances. But the trade-off there is that all the instances of the microservices all share the performance of that proxy.

Now, the third model, is where every instance has its own proxy. Now, this is the scaling problem because if you have really large clusters with, let's say, thousands or tens of thousands of containers, suddenly you have tens of thousands of proxies. But on the other hand, because you have a proxy per node, there is little or no sharing here and so, every instance has access to resources of that instance completely. So, these are fundamental, sort of, architectural trade-offs that you need to keep in mind.

[00:45:04] JM: This is great explanation of proxying and some of these different decisions we could make. I want to shift the conversation to talking about building Avi Networks. This seems like a very challenging product to build and one of the things that stands out to me as a challenge is, the different platforms that you could potentially be deploying to. So, you've got Private Clouds, Public Clouds, bare metal, VMs, containers, Kubernetes, Mesos. What are the strategies for maintaining compatibility with such a wide range of potential deployment targets?

[00:45:42] RR: Yeah, that's a great question. So, early on when we actually architectured and built the Avi product, we sort of, had in mind and we knew that this has to be architect in such a way that it will run across different deployment methods, it will run across different types of infrastructure and it will have to serve different types of applications. So, one of the key aspects of design when you have to do that, is abstraction. So, let me give you an example, so when the Avi software runs in VMware Cloud for example, then it needs to talk to this sphere to create a VM, it needs to talk to this sphere to attach a network to a VM.

It needs to talk to this sphere to detach a network from a VM and the set of APIs that is used for this sphere is specific to this sphere. Similarly, when it runs a DMWS it needs to do similar functions but used in WSS own APIs and its own clients, and so on and so forth. So, an abstraction is very important. So, the rest of the system should not change regardless of where the software is running. So, we abstracted that out with a piece what we call it earlier as a cloud connector. So, what it does is it provides a set of functions or APIs, it's a microservice.

It provides a set of functions and APIs internally to the rest of the system that they access to create a VM, to attach a network, to scale out and do other things. But then, it uses the right set of APIs depending on what the enrollment is and where it's running, to actually effect that change in that environment. So, the abstractions like this have served us really well in making sure that the fundamental software works the same way. Regardless of what environment it's running, what form factor, it's running on VM, container, bare metal, or what application it's actually serving.

[00:47:59] JM: Tell me about another big engineering challenge that you've had building Avi Networks?

[00:48:06] RR: The biggest challenge, like, almost all other, I think, a lot of companies is really attracting good talent. The competition for talent has been quite fierce, so we have some advantages because we're a startup, and naturally engineers want to work in smaller companies because they get to learn a lot, they get to build things out from scratch. Also, you can move faster and you can really advance your career faster as the company grows, and so on and so forth. So, those are some natural advantages, but we're also competing for talent with a lot of other extremely well-funded startups. So, that is one challenge that we will constantly face and will continue to face.

The second one right now that we're facing is just scaling the company. So, we're going through a huge spurt of growth right now, we are hiring and expanding the company in all functions, engineering, sales, marketing, support, QA, everything. So, again, scaling the company, bringing all the new employees up to speed and in sync with the rest of the company and making them wholly productive, again is one of the challenges that all high-growth companies go through and we are also going through that right now.

[00:49:25] JM: That's funny, because none of those sound immediately like a technical engineering problem, it's all like human and cultural issues.

[00:49:33] RR: Absolutely. I think, fundamentally, I've always found that it's people and employees who solve problems, so the best people we can find and we can hire, and they will figure out ways to solve any technical problems. So, finding good people and hiring them and retaining them is almost always the number one challenge I've seen in successful companies.

[00:49:59] JM: Tell me about the sales and the integration process when you're working with a large enterprise, because I think there's probably some people in the audience who are building software that they are trying to sell into large enterprises, they're trying to connect with a large enterprise, they're trying to figure out the best way to negotiate what is the integration process and the pricing process. You have any tips for that sales and deployment process?

[00:50:26] RR: The two ways I've seen this successfully sold in the large enterprises either the technology finds its way to developers, where developers find this new piece of technology that they like, they use. So, that implies that it is readily available, it's easy to use and developers know of them and it works, of course. So, that's one way I've seen technologies being successful large enterprises. The other way is, the more traditional route to an enterprise sales organization. In that case, almost all of these, when it's a new product or a new technology, you have to find a champion who believes in it within the large enterprise.

Then you need to work with the champion, be extremely responsive, listen to him or her and make whatever changes are necessary and then enable that champion to be successful within the larger organization. And these are the two fundamental patterns that I've seen when young tech companies like ours are successful at large enterprises.

[00:51:36] JM: What's the big vision for the company?

[00:51:39] RR: So, we sit at this critical juncture between applications and the actual users of these applications and we provide this set of functions that are extremely strategic. Load balancing is one of them, security is the other one, visibility and analytics is the third one. So, we are setting almost at this nexus of functions that are pre-served by at least three different markets, which is the traditional load balancer, EDC Market, the Security Market and then the performance management or the application or network performance management market.

And so, we have a fundamentally modern architecture that makes providing these functions very efficient and elastic and also easy to consume. So, in the long-term, our vision is to provide all these functions using our architecture and make them work in a seamless way across all the enterprise customers environment. So, that is really our long-term vision.

[00:52:53] JM: Ranga It's been great having you on Software Engineering Daily, I really enjoyed the conversation.

[00:52:58] RR: Absolutely, thanks Jeff, thanks for the opportunity of being here and talk to you later.

[END OF INTERVIEW]

[00:53:05] JM: Thanks to Symphono for sponsoring Software Engineering Daily. Symphono is a custom engineering shop where senior engineers tackle big tech challenges, while learning from each other. Check it out at symphono.com/sedaily.

Thanks to Symphono for being a sponsor of Software Engineering Daily for almost a year now. Your continued support allows us to deliver content to the listeners on a regular basis.

[END]