EPISODE 1053

[INTRODUCTION]

[00:00:00] JM: NGINX is a web server that can be used to manage the APIs across an organization. Managing these APIs involves deciding on the routing and load balancing across the servers which host those APIs. If the traffic of a website suddenly spikes, the website needs to spin up new replica servers and update the API gateway to route traffic to those new replicas. Some servers should not be accessible to outside traffic and policy management is used to configure the security policies of different APIs.

As a company grows, the number of APIs also grows, increasing the complexity of managing routing logic and policies. Kevin Jones is a product manager with NGINX and he joins the show to discuss how API management has changed with the growth of cloud and mobile and how NGINX has evolved over that period of time.

Full disclosure; NGINX is a sponsor of Software Engineering Daily.

[SPONSOR MESSAGE]

[00:01:03] JM: As a company grows, the software infrastructure becomes a large complex distributed system. Without standardized applications or security policies, it can become difficult to oversee all the vulnerabilities that might exist across all of your physical machines, virtual machines, containers and cloud services. ExtraHop is a cloud-native security company that detects threats across your hybrid infrastructure. ExtraHop has vulnerability detection running up and down your networking stock from L2 to L7 and it helps you spot, investigate and respond to anomalous behavior using more than 100 machine learning models.

At extrahop.com/cloud, you can learn about how ExtraHop delivers cloud-native network detection and response. ExtraHop will help you find misconfigurations and blind spots in your infrastructure and stay in compliance. Understand your identity and access management payloads to look for credential harvesting and brute force attacks and automate the security

© 2020 Software Engineering Daily

settings of your cloud provider integrations. Visit extrahop.com/cloud to find out how ExtraHop can help you secure your enterprise.

Thank you to ExtraHop for being a sponsor of Software Engineering Daily, if you want to check out ExtraHop and support the show, go to extrahop.com/cloud.

[INTERVIEW]

[00:02:39] JM: Kevin Jones, welcome to the show.

[00:02:40] KJ: Hey, how is it going?

[00:02:41] JM: Going well. You work at NGINX, and I'd like to start off by getting a brief history of NGINX and the problems that it has historically solved.

[00:02:53] KJ: Yeah, sure. Originally, NGINX was actually created as a solution the C10k problem. I'm not sure if you're familiar with it, but it was essentially an issue where one particular server couldn't handle more than 10,000 concurrent connections. The idea was Igor Sysoev who was the original creator of NGINX actually created NGINX as a reserve proxy. So, just to sit in front of Apache and handle those connections.

Because NGINX was originally created as an event-driven asynchronous non-blocking architecture, this allowed all the connections to be offloaded on a per CPU or per processor based method. Essentially, NGINX has workers that can be used to handle those connections and offload those connections. If you have let's say an 8 CPU servers, you can have 8 workers that are all individually able to handle the concurrency in those connections.

Again, it was initially as a problem to offload that connection handling. Also, it kind of blew up into an application server to be able to serve not only static content, but also to be able to proxy to dynamic content like PHP or other protocols like UHT or SCGI.

[00:04:13] JM: NGINX first came out 16 years ago. How has web infrastructure changed since then?

[00:04:21] KJ: Yeah, I think primarily a lot of the web is still obviously HTTP. There has been adaptions of new protocols and new methods of communicating over HTTP such as HTTP 2, which essentially is a new protocol. There's been actually a mass adaption of SSL as well, so TLS or SSL connections, trying to essentially secure the internet as much as possible.

Then there also has been additional new protocols such as GRPC or other complex protocols that are maybe specific to particular use cases such as like MQTT. What we're seeing is a rise in obviously services, the amounts of services and also an increase in various protocols that are being used to communicate over the internet.

[00:05:12] JM: On the usage side, there are more users engaging with applications since 16 years ago. Some of these engagements are driven by mobile. How does the increase in mobile usage drive changes to backend architecture.

[00:05:32] KJ: Yeah, absolutely. I mean, with the drive of increased devices, iPads and iPhones and Android and the increase in applications, and then that combined with the adaption of newer type of architectures for creating applications such as microservices or service-oriented architectures. What essentially as a resulting is more devices, more connections, and more requests being processed throughout the internet, right?

If a user, the more apps that they install on their phone and the more real-time communication is being done with that device, is in fact causing the increase in connections and requests on the internet, right? That's why we've seen a drastic amount of APIs and other requests being made on the internet today.

[00:06:26] JM: NGINX was also started before the cloud. Did the usage of NGINX changed significantly after it's started to be deployed on to cloud infrastructure?

[00:06:38] KJ: I think it's been permanently the same. I think one of the nice things about NGINX is NGINX was always designed to work on Linux, and it's the primary operating system of choice. Given that someone using NGINX for the duration of their time on-premise, whether it'd be a Linux server sitting in a closet or a data center, being able to move that to the cloud

really doesn't affect the way that NGINX is deployed, because you obviously can deploy Linux on any cloud environment, right? Whether it's a public or private cloud. That's one of the sweet things about NGINX, is it can be really anywhere on any infrastructure.

[00:07:17] JM: There's this variety of applications that NGINX is used for, so caching, load balancing, API gateway. Are you using a different kind of NGINX server that's completely dedicated to solving just one of these problems depending on what problem you're trying to solve, or is NGINX just the same whether you're trying to solve caching or load balancing, for example?

[00:07:47] KJ: Yeah, the configuration of NGINX actually can be independent or it can be combined. Because NGINX is a reverse proxy, a lot of users deploy NGINX in a situation where it's sitting in the DMZ protecting the infrastructure behind it. In that sense, it can serve multiple functions, right? It might serve the functionality to do authentication and authorization and then it also might serve to protect against web application threats, so like used as a web application firewall or as something to protect against like DDoS.

Yeah, you can consolidate all the features and functionality into one particular layer of NGINX, or if you decide to in certain scenarios, it makes sense to split those up, right? Example would be perhaps you wanted to deploy a security layer at the DMX that's doing the authentication, authorization, DDoS protection, all the high-level stuff from a networking perspective, but then you need a caching layer behind that that might be distributed. That would be a situation where you'd want to have those separate so that you could utilize different architectures to distribute the cache.

[00:08:58] JM: What kind of configuration is required to set up NGINX for these different applications? Could you maybe just go into a little more detail about like if I'm setting up a load balancer versus setting up an API gateway, how is my configuration going to differ on NGINX?

[00:09:19] KJ: Yeah, absolutely. NGINX is purely driven by what we call directives. So directives are essentially configurations. There're about a thousand different directives that are used with NGINX and each directive has an additional functionality or a different area of the configuration that can be used. The way NGINX is configured is if you enable a directive or set a

directive. It might open up other functionality within that, right? In other words, like let's say you want to do load balancing with NGINX, you would have to configure an upstream directive or an upstream context, and that would allow you to do load balancing, which would open up the ability for you to do things like session persistence, or code key persistence, or whatever type of load balancing algorithm you're using.

NGINX is just as powerful as the configuration that you put into it, and it is very interesting because the configuration files, they look similar to JSON type of configurations, and this allows you to nest your configurations down into the different various layers so that you can have a different behavior based on the different configuration that you have. In another words, you might have a separate configuration for a certain virtual server than you would for another virtual server because they're in different nested locations. So that gives a lot of power for you with NGINX to do what we call multitenant configurations where you have multiple applications running on a single instance of NGINX.

[00:10:52] JM: I'd like to get into a conversation of the application of NGINX to API management, and I'd like to start by just talking about APIs generally speaking. As a company grows, it needs to stand up more APIs. Why is that?

[00:11:12] KJ: Well, I think it goes back again to what I was talking about earlier where you have a certain type of adaption of the way you're building applications in your company or in the architecture. If you're doing something somewhat what we call monolithic, you're usually deploying an application as a whole. That application is being deployed on the single instance or a single Server.

Previously, older, more monolithic type architectures didn't have a large amount of servers behind them. As you switch to a service-oriented architecture or a microservices-oriented architecture or a microservice architecture, you start to break up those monolithic applications or services into smaller services.

What happens is as you break apart those monolithic apps into smaller applications, those applications need to be able to talk to each other and they need to be able to do things depending on the requirements for the business logic. They need to do things either more often

or they need to do things in a different way. In other words, maybe they need to be secure between those services because they're distributed across clouds or different availability zone. So that might require that you add additional complexity likes TLS. It also might require that you add additional authentication and authorization of your application. In another words, what applications can talk to other applications? When they do talk to them, what access do they have? In other words, what can they do? Can they write? Can they only read? What access do they have from what we call auth-Z versus auth-N? This is where you get more complex communication and the need for an API gateway or an API management and API gateway plane to be configured.

[00:13:03] JM: What kinds of configuration goes into defining that APA Gateway, that API plane?

[00:13:12] KJ: Mm-hmm. Yeah. Essentially, there's two terms that are commonly thrown around today. We sometimes hear the term API management and we also hear the term APA gateway. Sometimes they're used synonymously. They're typically the same referenced name. But in fact, they're actually two different things. An API manager is essentially the control plane. It's something that's actually pushing the configurations to the API Gateway. The APA gateway is where the actual proxying, something like NGINX+ or NGINX open source would be in API gateway, and something like a control plane, so like something – We have NGINX controller, which is our controller plane, or we have something like maybe a customer wants to use Ansible, or Chef, or Puppet. That might be their control plane. The idea is that you're pushing those configurations to an NGINX instance.

Some instances, you might not have the control plane. Everything might be done manually again with some kind of tooling. But the idea is that you have one API management plane that's handling the policy and the configuration for how those applications should be managed and accessed, but then the actual routing and the load-balancing and authentication and authorization is usually handled at the API gateway layer, if that makes sense.

[00:14:34] JM: It does. What's the difference between an API gateway and a load balancer?

[00:14:42] KJ: An API gateway is a function of a proxy essentially. You would consider NGINX as a reverse proxy, and a lot of times what that just means is that it's opening connections on one side and also opening connections on the other side.

A gateway, API gateway, is just another term to define a reverse proxy with a specific set of functions. An API gateway is really a reverse proxy or it can also do load-balancing. That's why you may see that some people deploy an API gateway not doing load-balancing. They'll just reverse proxy to maybe an additional load-balancing layer that they already had set up. But the idea is the function of the API gateway is to handle authentication, authorization, denial of service protection, maybe web application security or web application firewall, and it's really just protecting the APIs. But in some cases, again, the API gateway might also do additional functionality like load-balancing or some other complementary reverse proxy configuration.

[SPONSOR MESSAGE]

[00:15:58] JM: This episode of Software Engineering Daily is sponsored by Datadog. Datadog integrates seamlessly with container technologies like Docker and Kubernetes so you can monitor your entire container cluster in real time. See across all of your servers, containers, apps and services in one place with powerful visualizations, sophisticated alerting, distributed tracing and APM. Now, Datadog has application performance monitoring for Java.

Start monitoring your microservices today with a free trial. As a bonus, Datadog will send you a free T-shirt. You can get both of things by going to softwareengineeringdaily.com/datadog. That's softwareengineeringdaily.com/datadog.

Thank you, Datadog.

[INTERVIEW CONTINUED]

[00:16:49] JM: Once I have an API gateway or I guess the load-balancing layer and my API gateway is set up, what is involved in standing up a new API?

[00:17:02] KJ: Yeah. Again, if you're sitting an API gateway in front of your APIs, there's usually a reason why you're doing that, right? You're trying to protect either areas of the API. In other words, let's say you have a certain who API that gets created on the backend from a developer and it gets stood up on a set of application servers. In order to enable that to be able to be accessed through the API gateway, you have to specify, well, what domain name is it going to be available on. What URI are users going to be able to access it? In other words, what routes are going to be available to the user or the client?

Once the client does come through what kind of access, again, are they going to have? Are they going to be required to pass an API key or a JSON web token or what kind of authentication rules are going to be in place? This is all handled through the API gateway through configuration. Again, you're usually defining the actual hostname that the application will be available on, the URL or the URI we would call it, which is the /whatever. This is the route of which a client can actually come through to the application. Then you're also setting all the additional things that are there to protect the application. So authentication, authorization, denial of service. Maybe you want to also check for the things like what methods can be used, like HTTP methods. Maybe you want to look for certain URL arguments that are being passed. This is all done through configuration. That's where an API manager plane really comes into play to help build your configuration, because the API management can handle that configuration for you based on whatever configuration that you want to allow.

[00:18:51] JM: Could you explain what policy management means as it pertains to APIs?

[00:18:59] KJ: Yeah. Typically, in an environment where you're building APIs and you want to publicly expose them, you want to set certain policies that are enforced. A policy would be let's say that you have a business logic API that applies to the general public, right? Someone wants to come in and they want to get access from that API, right? You would probably give them some kind of method to authenticate against that. You might give them an API token or you might give them – I'm sorry. API key or a JSON web token and you would set the configuration, which is the policy essentially, on when that user comes through with that token, what can they access and when can they access it and how often can they access it?

This is where an API gateway really becomes powerful, because you can specify whether clients or a user with the token or a key can have read-only access or write access. You can also specify what methods they can use to access that API. You can also specify how often they can access the API. So maybe they're a basic user and you only want to allow them to access the API, say, 100 times a minute. That's where you can define the policy of that user. What pool is that user in and what kind of access does he have based on his policy? This is where policy is a critical part of the API gateway, because it really allows you to have fine-grained control of the clients or the users that are trying to access that API.

[00:20:33] JM: As the application grows, there's going to be a ton of policies that I'm going to be managing. I mean, I may have policies across all of my different APIs. What's required to manage all those policies across the different APIs at scale?

[00:20:52] KJ: Yeah, that's a good question. Some organizations have built-in ways to do that. It usually involves that the API gateway has some kind of communication with an IDP. So some kind of identity provider. Typically, NGINX, you can set up the configuration to have and store specific keys or you can have it store specific API keys or JSON web tokens or what we call JSON web keys, and you can essentially allow that configuration based on that configuration. If you're using open source, you can do that with just a flat configuration file that's owned by root. So it's rather secure. If you're using our commercial version, like NGINX+, we have a key value database that can used to be able to store those configurations. But the most effective way is for you to talk to an IDP that already has that hierarchy or that authentication and authorization spec already in place, right?

So what API Gateway can do is it can talk that IDP to get access to the key file that is specific for those particular clients and then it can store that key file locally so that it doesn't have to necessarily go to the IDP. It can look up whether that particular request has come through with a proper, basically, signature, and it can validate whether that client is accessible to certain configurations.

Yeah, it can get highly complex, and I guess the most effective way is, again, to use an IDP to manage what access can be had. Then for the actual authentication – I'm sorry. Sorry. For the authorization, this is where an API management plane really comes in the play, because you

can build those configurations and store them in the control plane and push them to the data plane using API management. Does that make sense?

[00:22:47] JM: Yeah. It definitely makes sense. Now I'd like to know how an API gateway fits into a typical software architecture. A typical large software company has several large services. They might have some microservices, and I want to know how the API gateway communicates with the other components of the architecture.

[00:23:14] KJ: Yeah. It always depends on the infrastructure. Every company, I would say, is rather different. We have some customers that their only interest for an API gateway is to protect their edge services, so to protect their applications that are sitting at the edge. This would be consumers where consumers are using APIs to get information and primarily the business is all driven based on consumer API usage. An example would be some company like maybe Expedia, right? Expedia, a large chunk of their revenue is actually driven from consumer-based API access. Basically, 90% of their APIs are publicly exposed.

But there're other companies that need, because their organization is growing so much and they're so distributed, that they want to build APIs internally, right? This is where a more of a clustered API gateway approach makes the most sense. The idea is that you still have an API gateway for edge services, but you might have separate individual API gateways that can be deployed on a role-based access type of way where basically each team can have its own API gateway that controls the policy and the access to those APIs. This way, you can get a little bit more distributed and kind of break up the need to have all that configuration in one place. This will allow you to build an API gateway plane that is somewhat distributed and, again, individually owned. If you have 10 different development teams or 10 DevOps teams that are managing their various applications or business logic applications, they can all have their individual API gateway. They can have its configurations.

[00:24:54] JM: Does the API gateway keep all of the API routes in-memory or does it use a database? How is the API gateway storing all the information about the different routes?

[00:25:08] KJ: Well, in NGINX's case, everything is stored in configuration files, but essentially that configuration file is loaded into memory. Just to kind of rewind a little bit, NGINX, the way it

primarily works is it's a flat file configuration, but there is – If you're using NGINX+, there is a certain level of configuration that you can do in-memory using an API. It all depends on the configuration and the user. If they're using open source, they're most likely using, again, just a flat config file, and they're using the binary to load that configuration into memory. You would do something like an NGINX reload, and that would load in the new configuration.

You essentially make a change to the config file, load it in memory and then you instantly get that new configuration. The nice thing about NGINX is it does that in-memory. I'm sorry. Not in-memory, but it does that gracefully. If you are pushing changes in real-time, NGINX will gracefully shut down the old connections in the old configuration so that it doesn't affect any clients that are currently connected and processing the request.

[00:26:14] JM: If I want to update the route of an API in my API gateway, what is the process for doing that?

[00:26:22] KJ: Yes. If you weren't using a control plane and you were just using an NGINX instance on the server, you'd essentially load up the config file or push the config file with the new change. Make the change, save it, and then you would just issue an NGINX-s reload, which is just saying, "Hey, NGINX needs to reload that configuration," and it will load that into a configuration.

If using something like Ansible, Cheff, or Puppet, that eases that, because you don't do that manually, right? You can push that out using some kind of configuration management tool and then issue that command remotely. If you're using something like NGINX controller, which is the commercial, essentially, control plane for NGINX, that's all handled using an HTTP-based agent. An agent is running locally to NGINX. It's called the controller agent. The controller is pushing that over HTTPS using mutual TLS. A user might go into the GUI and make a change and then click a button to deploy that change and it would push it over that HTTPS connection, or maybe they are a little bit more of an advance shop and they want to use an API to do. That they would access a controller's API to make that change.

Essentially, if they had 10 API gateways sitting behind NGINX controller, they can just issue one API call to the NGINX controller and it will push that configuration to all 10 of those instances.

It's a real easy way to automate and orchestrate at scale. Whereas something like if using something like Chef, or Puppet, or Ansible, you're most likely going to need to make that change on all the instances individually. This is where something like a control plane really helps with easing the management of the configuration.

[00:28:08] JM: What role does an API gateway play in logging?

[00:28:12] KJ: Yeah. I mean, obviously logging is important, right? It's something that you need to know what is going on at that particular gateway layer, or the proxy layer in most cases. The idea is that you're trying to capture whatever information is important to you. One of the nice things about NGINX is anytime a request is made through NGINX, all the available information is available in the form of a variable, and there's a pretty large number available is available and you can essentially log those individually.

If you choose that you want to log, let's say, the remote IP address, you can do so, or if you want to log all the headers, HTTP headers, you can do so. There's a lot of control in what you want to log and what access is being made through the proxy. You can log that information and make it available. At that point, yeah, you can just forward that log to something like Splunk or some other log management tool.

[00:29:13] JM: What about distributed tracing?

[00:29:15] KJ: Oh yeah! Distributed tracing is also very – I would say it's a newer method to be able to basically trace the request that's been passed through or something like a proxy or API gateway. Yeah, NGINX is very useful for that, because we do have an open tracing module that can be used. What it allows you to do is really capture somewhat of a GUID or an identifies and then pass that down into the application so that you can figure out what kind of tracing is being – Or I should say what kind of latency or request you're being passed down through that request. It gives you a lot more insight.

I would say logging is important. Tracing is even more important especially if you're doing microservices and you're having lots of connections and lots of distributed connections. Especially if you're using something like the service mesh, I would say that open tracing is one

of the most important things that you can design into your infrastructure when you're doing something with those services.

[00:30:18] JM: Service discovery is how services find each other. Does an API gateway have a role in service discovery?

[00:30:26] KJ: Yeah, absolutely, especially if you're using NGINX or your API gateway. In this case, for load-balancing, because the API gateway really needs to know what services are available behind it and where to route those requests obviously. The old school way is you would make sure that in your configuration you have all of your servers and IP's and ports that relate to those servers and you would have to hardcode those in the configuration, but obviously with containers and more immutable infrastructure.

Today, the services are constantly scaling up and down and being somewhat destroyed and created. The idea is that if the proxy layer or the API Gateway is not able to communicate with the services, you're going to have clients that just are basically going to timeout or have issues accessing those services. Yeah, it's very important that the API gateway has some level of service discovery integration. A lot of times, that can be done through the control plane. But if you want to do it in real-time, some of the best methods actually resolve around DNS In other words, having the API gateway, being able to look at a DNS entry point to be able to figure out where that route needs to go and what services are available behind that DNS entry point. That's one of things that NGINX is strong about, is it can actually talk to a service discovery tool over DNS and route that based on the DNS resolve.

[00:32:00] JM: One architectural pattern is a two-tier gateway. Explain what to two-tier gateway is.

[00:32:07] KJ: Yeah. I briefly talked about it earlier, but the idea is with the two-tier gateway, the main components are separated in the sense that the edge gateway, which is the primary gateway that comes in trough the DMZ, is usually doing a high-level security related functions. It might be protecting against access control. It might have an access control list. It might have DDoS protection. It might have external authentication and authorization. In another words, anyone trying to come through that initial edge gateway needs to be able to authenticate and

authorize on what they can and can't access. Then it might have other functionality, but primarily that's the main component, is around security.

Then internally, there might be a second tier that is more of what we call a router gateway or a router mesh. The idea is that you have a separate concern there where you're more worried about services internally that need to talk to each other. You might not have as much complicated configuration from a security perspective in there. Services might be able to talk a little bit more freely and you can specify whether particular clients internally can access other APIs at a different area.

That way, if let's say a microservice is sitting behind that internal router gateway or router mesh layer, they don't have to go all the way back out through to the Internet and come into the DMZ again. They can just handle all their connections internally. This allows security and net ops or other teams to manage the infrastructure at the edge and it allows DevOps or operations or maybe SREs to manage the configuration inside the network. It kind of helps with role-based access and it also helps with what I call multiple cooks in the kitchen where everyone is trying to make changes to a single layer. It can get a little bit more complicated to manage, and this gives each team the ability to really manage their configuration separately.

Then you could even go a step further and you can even have distributed more than one router gateway or router mesh internally. You could have the specific router mesh or a router gateway for each individual team. This even gets more distributed where each team can be responsible for their own internal gateway.

[SPONSOR MESSAGE]

[00:34:43] JM: You probably do not enjoy searching for a job. Engineers don't like sacrificing their time to do phone screens, and we don't like doing whiteboard problems and working on tedious take home projects. Everyone knows the software hiring process is not perfect. But what's the alternative? Triplebyte is the alternative.

Triplebyte is a platform for finding a great software job faster. Triplebyte works with 400+ tech companies, including Dropbox, Adobe, Coursera and Cruise Automation. Triplebyte improves

the hiring process by saving you time and fast-tracking you to final interviews. At triplebyte.com/sedaily, you can start your process by taking a quiz, and after the quiz you get interviewed by Triplebyte if you pass that quiz. If you pass that interview, you make it straight to multiple onsite interviews. If you take a job, you get an additional $1,000 signing bonus from Triplebyte because you use the link triplebyte.com/sedaily.

That $1,000 is nice, but you might be making much more since those multiple onsite interviews would put you in a great position to potentially get multiple offers, and then you could figure out what your salary actually should be. Triplebyte does not look at candidate's backgrounds, like resumes and where they've worked and where they went to school. Triplebyte only cares about whether someone can code. So I'm a huge fan of that aspect of their model. This means that they work with lots of people from nontraditional and unusual backgrounds.

To get started, just go to triplebyte.com/sedaily and take a quiz to get started. There's very little risk and you might find yourself in a great position getting multiple onsite interviews from just one quiz and a Triplebyte interview. Go to triplebyte.com/sedaily to try it out.

Thank you to Triplebyte.

[INTERVIEW CONTINUED]

[00:37:00] JM: If two gateways can have value, are there situations where even more gateways have value? What is the spectrum of tradeoffs between adding more gateways versus keeping it more consolidated?

[00:37:15] KJ: Yeah, I think the more you add, the more complicated the infrastructure gets and the more of a need for a solid robust control plane is important. When you first initially set up, if you were to set up a single edge gateway, everything can be in one single configuration. When you start adding two, then you realize, "Okay. Well, now I need to manage two different configurations." When you start splitting that into even more teams, let's say you have 10 teams and then all of a sudden you're managing 12 different or 11 different configurations. If you distribute that even more, it gets even more and more complicated on keeping track of who can talk to what and what kind of access control and authentication can be used at each layer. You

also have to keep track from a security perspective as well, because you need to make sure that all those instances are keeping up-to-date with patching of NGINX or the underlying open SSL that's being used. It gets a little bit harder to monitor as well and maintain those configurations.

This is where as you start to even get more and more proxy configurations, you start building the need for something like a service mesh, and this is where your control plane really pays a lot of value because it's going to help you with configuration. It's going to help you with monitoring. It's going to help you with management and give you that visibility into the entire, basically, infrastructure.

[00:38:40] JM: How does the service mesh pattern compare to patterns involving API gateways?

[00:38:49] KJ: Yeah. I mean, really, the idea of a service mesh is that you are trying to make almost micro gateways or somewhat hat we call sidecar gateways where you're essentially deploying a proxy layer all the way up into where the service is running. Let's say you have an application running in a Docker container. You might have a proxy that sits right next to the service inside that container and the service inside the container is not publicly exposed. It's just listening on its own internal network. The proxy layer that's sitting next to it as a sidecar gateway or how NGINX would be used as a service mesh is handling all the connections that come in.

The nice thing about that is you can really have a fine-grained configuration all the way up into the container. In other words, let's say you're trying to do SSL everywhere even all the way into the container. You can have NGINX manage the SSL and TLS configuration. Again, it gets more complicated. I'm not sure if that answers your question, but service mesh is probably not for all organizations. There's some kind of strict requirements that you have to look at to whether you should adapt a service mesh.

[00:40:02] JM: How has Kubernetes changed how API gateways are used?

[00:40:07] KJ: Well, Kubernetes is definitely made it easier for organizations to adapt a microservices type architecture. If we look at Kubernetes as a whole, its main objective is to

help with the orchestration and I would say the overall configuration of your services. The service discovery, it has built-in service discovery. It has basic load-balancing. It has layer four load-balancing configuration built-in. But what Kubernetes does allow and where NGINX and any other API gateway really plays is that you can bring your own load-balancing configuration to Kubernetes, and that's in the form what they call an ingress controller.

The nice thing about that is if you use something like NGINX as an API gateway, essentially you're deploying NGINX as a ingress controller configured as an API gateway, and it's able to take advantage of all of the features and functionality that Kubernetes offers. Kubernetes has its own built-in services discovery. We can talk directly to that service discovery to do service discovery and basically be more dynamic.

Because we can also interact with the Kubernetes API, we can do things in a very graceful manner. In other words, we can orchestrate the configurations in the load-balancing and the routes and all that stuff. It really makes for actually a really good experience especially for a person that doesn't necessarily know NGINX configurations super well. If they know Kubernetes and they understand the Kubernetes YAML configuration, they can build an NGINX gateway as an ingress controller rather easily just by creating a YAML configuration with all the detailed information that they want to build NGINX as.

What the ingress controller will do is it'll handle all of the internal creation of the NGINX configuration and it will handle all the service discovery and it will handle the help checking and everything that Kubernetes provides, it will work together with the NGINX ingress controller to do that functionality, if that makes sense.

[00:42:25] JM: Yeah, it does. What kind of performance tuning might be required at the API gateway level?

[00:42:33] KJ: Yeah, I guess it depends. If you're looking at deploying NGINX on bare-metal or virtual machines, there is a good amount of tuning and configuration that you can do, because you have to think about things like file descriptors. You have to think about things like available connections. It's not only from NGINX perspective, but also from a Linux perspective.

There's a certain level of tuning, yeah, that you need to do if you're deploying NGINX on a bare-metal or on virtual machines. If you're looking at deploying NGINX on containers, that gets a little bit less of a requirement, because the idea around containers is you're just deploying larger amounts of virtual, basically, operating systems. NGINX is being more distributed. So there's less of a need to tune in a container environment, I would say, than something like virtual machines or bare-metal. Yeah, if you are deploying on virtual machines or bare-metal, it's definitely recommended to look at things like file descriptors, connections, SSL. What kind of optimizations you can do around SSL? There's a lot you can do to help tune the infrastructure and basically maximize the efficiency of your API Gateway layer.

[00:43:49] JM: At a really large software company, you're going to get to a place where you have lots and lots of APIs. You're going to have lots and lots of routes to manage. What kinds of problems do these enterprises get around API gateway management as the volume of things to manage behind an API becomes large?

[00:44:13] KJ: Yeah. I would say the number one thing is security becomes a very crucial component, being able to audit the access that is available for those APIs. That's the one thing that companies are always worried about, is data and publicly exposing that data and protecting the data behind those APIs. Second would be, again, dealing with the service discovery. I would say a good number of customers that we work with don't necessarily have a good grasp on what instances are actually running in their environments, and I think where something like an API gateway or, essentially, it's something like NGINX can help, is to be able to deal with more dynamic nature environments and be able to keep track of that information.

Again, I think service discovery is definitely a crucial part and a difficult thing for some companies to deal with. I think it's just goes back to that usually most organizations have a large number of development teams and each development team has a different way that they're building applications. A lot of times, there is no standard on how those services are kept track of. Again, having some kind of standard way to do service discovery in your organization is key.

I think lastly, I would say being able to be agile is getting even more and more difficult. The idea is that companies are trying to build applications faster so that they can come to market with features and functionality quicker, which in turn gives a better customer service experience. If

you want to do something like that, you need to be able to deploy quickly. I think building a robust CICD pipeline for your applications, basically a continuous integration and continuous delivery of your applications, is very critical to you being agile in those environments. I think that's definitely a struggle in some ways for customers.

[00:46:11] JM: What about routing between multiple clouds? Does an API gateway typically sit in front of multiple clouds or do you have like an API gateway in each cloud if you're a multi-cloud and you're a gigantic enterprise?

[00:46:28] KJ: Yeah. I think if you're multi-cloud, you're more likely going to have somewhat of a distributed API gateway for each cloud. You might have something like – You might use a cloud load balancer initially at the edge, right? You might use something like an NLB or maybe a Google Cloud Load Balancer just to accept the initial connection. Then immediately sitting behind, that you probably have some level of protection from a high-level perspective, so DDoS, maybe a web application firewall, something at the edge, right? Usually, either at that layer, you also have additional functionality for NGINX or something like that as an API gateway, or you'll proxy into another layer that's handling API gateway functionality.

In this way, you can distribute. If an application that's in AWS needs to talk to an application that for some reason is only available in GCP, they can go out to the Internet, come back in and be protected. Essentially, have a secure connection from each cloud to that application.

[00:47:29] JM: The CDN infrastructure is becoming more richly featured. Does that affect API gateway infrastructure at all if I'm moving more and more functionality to the CDN to the edge layer?

[00:47:45] KJ: Well, there is a lot of power there to be able to offload requests and traffic to the CDN as much as possible especially if you're dealing with requests that are very static, right? There's a huge benefit to trying to bring that data as close to the client as possible, right? Because not only you're saving on the client's requirement around latency, because they're able to hit that data locally, but you're also saving on network infrastructure costs. The additional latency or the additional bandwidth that needs to come into the cloud provider or to your data center. You're also dealing – You're saving on compute, right? So you're not having to do a TLS

connection. You're not having to run CPU cycles on that machine. It frees it up to do other things. Yes, absolutely. Having a CDN and a robust CDN at that layer is very effective. I think that's why companies like Netflix who has a very highly distributed CDN, they try to bring that data as close to the client as possible, because there's a lot of value there.

[00:48:53] JM: What are the outstanding problems in API management today? What do you think needs to be solved or what improvements that could be made to the API management layer?

[00:49:05] KJ: Yeah, it's a good question. I think right now with a lot of the API gateway tooling that's out there, is it somewhat monolithic in nature? As you start to add a lot of features and functionality into an API gateway/API management plane, it starts to become very, I would say, bloated and hard to deploy and distribute especially if you want to distribute it quickly or inside of a he container or something like that. I think the best thing we can do is try to adapt the same methodology that our applications are trying to adapt, so microservices, and being able to have distributed functions of the API gateway API management plane. In other words, having the ability to keep the functionality at a very basic or a minimum, but solving all the problems that the business needs. That is one of the things that we tried to do with NGINX controller and NGINX+, is we wanted to build a solution that basically separates the control plane and the data plane, and we've done that, and now we're just trying to solve and make it easier for developers to be able to publish their applications. I think there's a lot of value in keeping that same methodology of making the applications and the functions of the API gateway management small, but also putting in a lot of functionality that makes it easier for the developers to publish those APIs.

[00:50:30] JM: All right. Well, wrapping up, it's been great getting your perspective on API management and NGINX, but I'd just like to know since we're in this crazy time, how has your work changed as a result of the virus both at the level of what you're doing for the company and just how your day-to-day life has changed?

[00:50:50] KJ: Yeah, it's a good question. For me, not much has changed other than my travel, obviously. Travel restrictions. I do a lot of training events for our resellers and I also do a lot of training events for our sales staff and I also do a lot of customer visits, and obviously a lot of that

has halted. My travels definitely gotten a lot lighter, but I think we're just using this time right now to kind of focus on the product that we're building to basically reevaluates the functions and start building roadmaps that are going to help solve problems for the customers. So we're pushing all of our energy into being able to build features and functionality right now. Also, we're also looking at improving our training program. I think there's a lot of that stuff that we can kind of reel back internally and start kind of helping provide some solutions and just some future plans that will help the users of NGINX as a whole.

[00:51:47] JM: Kevin Jones, thanks for coming on the show.

[00:51:50] KJ: Yeah, no problem. It's great to meet you, and thanks for having me.

[END OF INTERVIEW]

[00:52:01] JM: We're happy to have NGINX as a sponsor of Software Engineering Daily. NGINX is not only used for routing and load-balancing, but also API management, API gateway, service mash and other applications. To hear more about what Kevin Jones discussed around API management, visit nginx.com/sedaily for webinars and other resources that describe how NGINX works and how it can be used to solve problems in your infrastructure.

[END]