EPISODE 1007

[INTRODUCTION]

[00:00:00] JM: A cloud provider gives developers access to virtualized server infrastructure. When a developer runs this infrastructure via an API call, a virtual server is instantiated on physical machines. That virtual server needs to be made addressable through allocation of an IP address to make it reachable from the open Internet. When the virtual server starts to receive too much traffic, that traffic needs to be load balanced with another virtual server. The backend networking code that runs a cloud provider needs to be fast, secure and memory efficient. Languages that fit that description includes C++, Rust and Go. DigitalOcean is a cloud provider and their low-level networking code is mostly written in Go.

Sneha Inguva is an engineer with DigitalOcean who has written and spoken about writing networking applications using Go. She joins the show to talk about her work at DigitalOcean including the implementation of a DHCP server, a network server that assigns IP addresses and other parameters to devices that sit on that network.

[SPONSOR MESSAGE]

[00:01:09] JM: DigitalOcean makes infrastructure simple. I continue to use DigitalOcean because of the low friction and attention to user experience. DigitalOcean has kept the experience simple and I can spin up a server in less than a minute and get high quality performance for a low price. For an application that needs to scale, DigitalOcean has CPU optimized droplets, memory optimized droplets, managed databases, managed Kubernetes and many more products. DigitalOcean has the flexibility to choose the right instance for the right workload and he could mix-and-match different configurations of CPU and RAM.

If you get stuck, DigitalOcean has thousands of high-quality tutorials, responsive Q&A forums and a customer team who treats customers respectfully. DigitalOcean lets developers focus on what they are building. Visit do.co/sedaily and receive $100 in credit over 60 days. That $100 can be put towards hosting or infrastructure and that includes managed databases, a managed Kubernetes service and more.

If you want to get started with Kubernetes, DigitalOcean is a great place to go. You can use your $100 to start building your distributed system and you can get that $100 in credit for free at do.co/sedaily.

Thank you to DigitalOcean for being a sponsor of Software Engineering Daily.

[INTERVIEW]

[00:02:46] JM: Sneha Inguva, welcome to Software Engineering Daily.

[00:02:48] SI: Thank you. Thank you for having me here. I've been a huge fan for a while, so I'm super excited and humbled to be on the show.

[00:02:56] JM: All right. Well, happy to have you on. You work at DigitalOcean, which is a cloud provider. Give me a few examples of engineering problems that you've worked on.

[00:03:04] SI: DigitalOcean, we are a cloud hosting provider. We have a variety of products in different areas, for example, with storage, with networking, as well as compute, which is probably I guess what most people are familiar with who use DigitalOcean. We have droplets or virtual machines that they can use.

But the interesting thing I think as a cloud hosting provider is that it's a little different from other companies in which – And that we have both physical hardware issues. We also have software issues, and then we also have a web application. We've had interesting problems kind of all over the place.

When I joined the company, I wasn't actually a network engineer. I was working on one of the internal delivery teams is what we called it. On that team, the biggest problem we were addressing was the difficulty in deploying and updating applications. So namely working with Kubernetes. That was definitely an interesting problem, because I think we addressed both the challenge in building an abstraction layer on top of Kubernetes that increased the just ease of deploying, because before that, people use Chef, and Chef was a little complicated in general.

Then on top of that, also getting buy-in from different teams to kind of use this new internal tool that we had. That's kind of one of the problems we've had that we've addressed.

[00:04:20] JM: As you mentioned, DigitalOcean is built around these abstractions called droplets. Can you say much about what a droplet is? Is it a VM? Is it a container? What am I actually interfacing with when I spin up a DigitalOcean instance?

[00:04:34] SI: Of course. It is a virtual machine. I think droplet is just our marketing speak for everything oceanic themes in our company, but it is essentially a virtual machine that is, I guess, technically co-located on servers with other virtual machines, and you can spin one of these up really in any location around the world. I think we have about 13 data centers. That's super fun.

I also heard you mentioned containers. Right now we don't quite have the containers as a service, but we have Kubernetes as a service. Technically speaking, you could kind of manage your containers as well, although droplet itself is just a virtual machine.

[00:05:13] JM: Got it. Now, when you join a company, it's always tricky to find the bounds of what you should learn and what you should ignore. It's hard to know just how deep to go, and I know that when one of these virtual machines is spun up, there's a ton of stuff that is going on under the hood. What was your process for figuring out what to learn the lifecycle of a user spinning up a VM?

[00:05:41] SI: That's a really good point. In fact, I think we still do this when someone – For our networking at least, we have a really good onboarding process. When I joined the company, not a networking, we also still had a pretty good onboarding process, but it was more generic and there is in fact, I guess, an onboarding session called how the cloud works where an engineer who's been at the company for a while actually goes through the entire process and kind of goes through all the microservices that, I guess, receive a request and send a response down to the schedulers that actually are scheduling the droplet placement on a particular hypervisor down to everything.

The thing is I think most people probably have a general idea of the different services that are being touched, but then when it comes down to the nitty-gritty of how exactly is networking set

up? How is our SDN configured? All of that. I don't think unless you're on that specific team, you are aware. I'd say it's kind of a T-shaped process in a way. So you have a general like breadth of knowledge of how, I guess, the "cloud works", but when it comes to the nitty-gritty details, you probably have a very good idea of just your specific area. I think it's impossible to have a very deep knowledge of absolutely every single service when you're at a company this large with this many microservices and with these many domains of expertise.

[00:07:11] JM: Totally. Now the reason I wanted to have you on the show is because I saw some talks that you gave. One specific talk about networking, and the term networking can mean a lot of different things. But I know that networking at a cloud provider and you being a systems engineer working at a cloud provider, you probably have some insights on the engineering that goes into the actual nitty-gritty of something spinning up within DigitalOcean. What does networking mean at a cloud provider? What does that term networking mean?

[00:07:49] SI: Networking at a cloud provider I think has two layers. There's of course the physical infrastructure that is set up. Of course, I think every cloud provider has physical switches, physical edge routers, physical gateway. That is definitely one layer. But then another thing that you have to consider is especially in a cloud provider where you are dynamically creating and deleting virtual machines, is that you are constantly adding different paths for networking packets to traverse and removing them as well. That's where software defined networking comes in and that's a completely different layer that you have to consider especially at a cloud provider.

In fact, at DigitalOcean, we actually have a team that deals with a lot of the physical details when it comes to physical switches in our data centers, but we also have, I guess, SDN team which has a lot of sub-teams that deal with a lot of the microservices that are interfacing and communicating with OVS, Open vSwitch, which is our virtual switch of choice that are actually making a lot of our networking products possible such as VPC, or firewalls, or even DHCP, a lot of these different things.

[00:09:00] JM: Tell me about some of the lower level networking concepts that you needed to know to build some of the projects that you've built within DigitalOcean.

[00:09:10] SI: Of course. I'll just take you through, I guess, when I first joined the networking team, we were coming out with a product called bring your own image. Previously, when people typically spin up a new virtual machine or droplet, they can select a predefined image, whether it's Ubuntu or – I don't think we have Microsoft, but a different version of Ubuntu or one of many different options. However, with BYOY, we started giving the option of bringing their own image.

The only issue with that is when we control the image ourselves, we can kind of control the cloud configuration, meaning allocating IP addresses and setting up a lot of configuration. But when they're bringing their own image, we need a way to dynamically allocate IP addresses for those droplets using that image, and that's where the DHCP protocol came in. That was something that I had heard of, but I wasn't super familiar with.

In general, I guess whenever you're building and you networking product that's using a new protocol, my first step typically is to read the RFC. I pulled up the DHCP RFC and then the DHCP V6 RFC, which is a little different, and started to learn about the protocol. I guess most people at home are probably familiar with it when they log in to their computer and they fire up the Internet. Their ISP router actually allocates an IP address for their home computer. So that's essentially using the DHCP protocol.

We were implementing our own, I guess, hypervisor level daemon to do that for different droplets in our data center. That was something that I started to learn about. Then the other thing when you're a cloud hosting provider is you start to learn about perhaps the ways in which you might have abusive actors and kind of look into security. So that was very interesting and then you start to do a lot of load testing and try to figure out how to mitigate any possible issues. That was also something else I started to look into when it came to the DHCP server.

[00:11:08] JM: That phrase you mentioned, the RFC, reading the RFC. I've read some core answers and Wikipedia recommendations about if you want to learn networking concepts, you should read the RFC, which stands for the request for comments. Why is that the best path to learning about networking protocols?

[00:11:32] SI: I mean, that is fundamentally where the networking protocols were designed, and some of these protocols were designed like decades ago. I think that, of course, you could read

Wikipedia articles, encyclopedia articles, YouTube videos. All of those are helpful, but I think that going to kind of the original source of where this communication protocol is defined. Of course, to be honest, the first time you read through any networking RFC, it won't 100% make sense. Obviously, going through it, marking up everything you don't understand, which then – Then of course every RFC is somehow linked to like 20 other RFC's. So then jumping to another RFC to kind of understand maybe another protocol that is used within a particular protocol kind of helps you, I guess, build sort of like a mental map or like a mental knowledge tree of what that protocol actually does and what it is.

[00:12:28] JM: Now, much of the networking code that you've written is in Go and the talk that I saw was about writing network services in Go. Explain what makes Go an appealing language for writing networking code.

[00:12:44] SI: I mean, there's a couple of issues – Or not issues, but a couple of reasons. First of all, Go is something that we've been using at DigitalOcean for a very long time, but then I think when it comes to networking in particular, I think Go is great because it has very easy to use concurrency primitives, and especially when you're writing like a DHCP server, or like an ARP proxy, or any sort of networking service, you do have a lot of – You're dealing with a lot of concurrency and you might be dealing with like syncing access to memory. The fact that you have these very easy to use concurrency primitives is very helpful.

The other thing is I think Go has like a fairly mature ecosystem when it comes to both networking and a lot of other like third-party packages for things such as HTTP servers or packages to kind of help like un-marshall different type of like layer 4 to layer 7 packets such as DHCP or ARP or what have you. I think that that also helps. Then quite often, especially because at DigitalOcean we're building a lot of networking services that run directly on the hypervisors themselves, a lot of them have like command line tools to interact with them, and Go is I think amazing when it comes to third-party tools for building command line interfaces.

For all of these reasons, I think Go is great for networking. I know that a lot of people, when it comes to super high-performance stuff, perhaps like rely on C, or I know there're a lot of people who rely on Rust. But from what I've heard, the ecosystem isn't as developed with Rest. Then with C, there's of course the issue of a lot of people are concerned about accessing memory

and memory safety. To be fair, it's also maybe not the most readable language. It's not as syntactically simple as Go. I think for all of those reasons, Go has been great for us at DigitalOcean.

I think the other thing I also mentioned is that we already used it and we have been using it for a long time. We have a very rich Go mono repo with a lot of shared libraries. When it comes to instrumenting our networking services and then getting custom metrics or just general metrics out of them, a lot of that happens automatically by virtue of the service being in the mono repo itself.

[00:15:03] JM: Since you mentioned that the term mono repo, maybe we could take a deviation from talking about Go lang and talk about that word, mono repo, what that actually means in terms of your work atDigitalOcean.

[00:15:16] SI: Yeah. We have a fairly large mono repo, meaning just a giant repository and most of the Go applications that are written at DigitalOcean are basically packages within the gigantic repository known as the mono repo. The mono repo also has a name, which I think is kind of funny. It's called Cthulhu. It even has a logo actually at this point of like a giant sea monster.

The mono repo, I think there're been a lot of debate about the mono repo at DigitalOcean and the idea of mono repos in general. I feel like this is one of those topics that could always delve into kind of a holy war in a way, but I think at DigitalOcean, I think it predates me a few years when the mono repo was originally started. But I think it honestly did help us get to a point where I would say, compared to a lot of companies, we're very on top of instrumenting our services and using observability. All the observability primitives such a centralized logging, using Prometheus metrics and then also using distributed tracing.

If you've looked at some of the distributed tracing libraries or kind of have to set up your own logger, quite often I think a lot of engineers are just – I mean, I wouldn't say lazy, but perhaps in a rush to deploy really quickly and might skip that step. But I think by having these shared libraries in the mono repo that can very easily be included in a project without having to import or go get a package or anything. I think it just really lowers the bar and makes it quite easy to

have best practices for a code. This even applies to, for example, GRPC. We use GRPC, but we have a wrapper around GRPC that also automatically instruments all our services. So they have metrics, logging and tracing.

[SPONSOR MESSAGE]

[00:17:12] JM: Being on-call is hard, but having the right tools for the job can make it easier. When you wake up in the middle of the night to troubleshoot the database, you should be able to have the database monitoring information right in front of you. When you're out to dinner and your phone buzzes because your entire application is down, you should be able to easily find out who pushed code most recently so that you can contact them and find out how to troubleshoot the issue.

VictorOps is a collaborative incident response tool. VictorOps brings your monitoring data and your collaboration tools into one place so that you can fix issues more quickly and reduce the pain of on-call. Go to victorops.com/sedaily and get a free t-shirt when you try out VictorOps. It's not just any t-shirt. It's an on-call shirt. When you're on-call, your tool should make the experience as good as possible, and these tools include a comfortable t-shirt. If you visit victorops.com/sedaily and try out VictorOps, you can get that comfortable t-shirt.

VictorOps integrates with all of your services; Slack, Splunk, CloudWatch, DataDog, New Relic, and overtime, VictorOps improves and delivers more value to you through machine learning. If you want to hear about VictorOps works, you can listen to our episode with Chris Riley. VictorOps is a collaborative incident response tool, and you could learn more about it as well as get a free t-shirt when you check it out at victorops.com/sedaily.

Thanks for listening and thanks to VictorOps for being a sponsor.

[INTERVIEW CONTINUED]

[00:19:02] JM: If we talk about a mono repo, are we saying that if I want to spin up a new service at DigitalOcean, there is this mono repo that I'm going to fork and it gives me a boilerplate system of different libraries that I can call when I'm building my service and it gives

me a great out-of-the-box experience. Is that what is or is it like we're all committing to the same mono repo that gets deployed as just like copies of the same big single service?

[00:19:38] SI: Oh, no. I would say the first, as what you described. It's a mono repo, but we don't have a monolithic application. We have a mono repo with a ton of subfolders and sub-packages and all the different microservices are included within the mono repo, but they're deployed totally separately. But then the mono repo itself just contains all the services, and then it contains shared libraries that are used by all of the services. Then we also have CI and CD set up that periodically tests everything in the mono repo.

But I will say that I think using it and especially when I was onboarding as a new engineer definitely help me learn proper Go practices and then kind of have an idea of how to even build like a very simple REST application or how to build a very simple server, a GRPC server and use protobufs, for example.

[00:20:35] JM: Because that repo comes out of the box with a lot of bells and whistles and like an easy onboarding experience or like an easy tutorial experience for just building a network service?

[00:20:45] SI: Yet, exactly. I think it comes out-of-the-box with I would say pretty thorough documentation on how to use a different shared libraries and then how to use GRPC, for example, or for networking, we have a lot of shared networking libraries that have been written by engineers on the networking team that also have like extensive documentation. So, yeah. Basically, for all of those reasons, it definitely improves the onboarding experience.

[00:21:08] JM: Can you talk more about those primitives that are in the mono repo? When you're like looking through this mono repo, you're exploring, you're getting started. You know you're going to have to write some DHCP server, and I think you said you had a load balancer. Later on we can get to that stuff. But you're starting to think about writing your own network application. You're looking through this mono repo. What are some of the systems and the services or method calls, libraries, things that you're seeing in this mono repo?

[00:21:38] SI: Yeah. For the DHCP server, for example, I know that it has a couple of aspects. First of all, I know that I'm going to be reading packets off the wire on marshaling them to some sort of data structure, processing them, and then responding to them. There is the Go net package of course, but then we kind of – Within the company, within the mono repo, and within the networking team, there's a lot of internal networking libraries. I guess do a little more in terms of processing the packets and perhaps validating them. That's one thing I would look into.

For the packets that I want to read, are there any internal packages that are available that help with just structuring them and reading them and then like marshaling them back and sending them on the wire? That's one thing. The next thing for any sort of, I guess, DHCP server. I would start looking at, "Okay. I have this DHCP server. I probably want to have some sort of metric. I want to see how many requests are coming in. How many responses are going out? Perhaps have a count of how many errors are happening." So then I would think, "Okay, I probably need a metrics package and let me look at the internal metrics, the shared internal metrics library, called gauche metrics, and see how I can use that to take account of requests and responses and then maybe look at errors."

Then the next step of course is that within my DHCP server, I definitely want to log any errors and I want to make sure all of those logs get to centralized logging and then have all the appropriate additional, I guess, keys and values so I can easily process the logs. So the next step would be also looking at the internally-shared blogging libraries. Then I guess one of the things I didn't even mention is – So we use GRPC. So for our DHCP server, of course, know it's a server that's listening on an interface. It's in a multicast group, but then perhaps we want our server to be able to receive messages from another service for whatever reason just because of how it fits in our ecosystem.

In our case, our DHCP server is in fact getting RPC requests from another service. We would then want to create a GRPC server. Of course, I could do this with just the normal Google GRPC packages that are out there, but I know that within our mono repo, we definitely have an entire section on using GRPC, on using protobufs, on generating Go code and best practices for that. So then I would go to the GRPC proper libraries, look at the instructions, figure out how to do that, figure out how to use SSL and then configure that as well.

[00:24:13] JM: Let's come back to Go and then we'll get back to the networking stack, but I want to get back to Go before we get into this. Tell me about some of the concurrency primitives provided by Go and why those are useful for writing networking code.

[00:24:26] SI: I think I had mentioned earlier, one of the most basic concurrency primitives are Go routines. Then sometimes I think I wonder if that perhaps the ease of using Go routines also makes it possible to abuse them. But in general, I would say that it's extremely easy to start like a Go routine. Simply put the word Go in front of a function. I think that makes it pretty easy.

On top of that within Go there's a package called Sync, and then Sync has Mutexes that one can use to kind of lock access to shared memory such as a particular function or a map. In fact, there's something called Sync map. This is also super useful.

The other thing, the thing that I've definitely used a lot I would say is error groups and weight groups for a lot of Go services actually, whether it's the DHCP server, or whether it's just trying to make my own port scanner. These are tools that allow you to simultaneously kind of start a variety of Go routines and unblock on the main thread and perhaps return early from these Go routines if there's a single error in the case of an error group or just block on the main thread and wait until all of your subroutines have finished. These make it very easy, I think, to kind of spinoff subtasks when you're processing a packet, for example, or perhaps like scan a bunch of different ports if you're building like port scanner. I think that that makes it super easy.

The other thing that we've done a lot just to make sure that we don't have like a Go routine spike is to kind of use a semaphore in the form – Like basically having a channel to kind of control the number of worker Go routines that we have. We did this a lot for DHCP just in case the DHCP server happened to receive a lot of concurrent messages. Of course, we have protection outside of DHCP. We're using OVS afterall. But for the service itself, just to protect the service and make sure it can't fall over, we use the concept of channels, which I guess are basically a communication conduit perhaps between different variables or different parts of the program, different Go routines in the program just to ensure that we had a limited number of Go routines.

[00:26:35] JM: That's great. Getting into an application, particularly the application you built. So there is this acronym, DHCP, dynamic host configuration protocol. This is a protocol that assigns an IP address to some other configuration so that devices can communicate with each other with other IP networks. Describe what the purpose of DHCP is.

[00:27:02] SI: Yes, of course. In general, DHCP is used I guess by a router on a subnetwork to assign IP addresses to our particular host such that the host can communicate with other hosts on that subnetwork. When it came to our case, of course there's a few ways to assign IP addresses to a particular virtual machine on a hypervisor. Just meaning a server with a bunch of other virtual machines. We could of course just assign them statically when they're spun up, and that worked for a long time.

But in our particular case, we needed to kind of dynamically assign them as soon as they were being spun out because we could no longer control all aspects of the virtual machine itself. So a virtual machine would come up. It would send a DHCP request to a particular multicast group, which happened to be the all routers group that are DHCP servers in on the hypervisor. That hypervisor would then, I guess, check in an internal – I think we're using DHCP is a little bit different than – That's an interesting thing in general at DigitalOcean. I think the way we do some networking is it's a little bit different than perhaps how it'd be done traditionally, but it's also different just because of our general layout and infrastructure and how we're OVS.

In our case, we actually have something pushing data to our DHCP server which happens to store a lot of IP address, MAC address, kind of mapping internally. So then when the actual droplet comes up and is online, it would talk. It would send a DHCP request over to the DHCP server, which would process the request. It would do some sort of validation. Then it would acknowledge the request and then send over a particular IP address that could be used in the DHCP. I probably need to like check this protocol again. Another reason to read the RFC repeatedly. It would then confirm that it is in fact using the IP address for a particular lease time.

[00:28:59] JM: Got it. When you talk about building a DHCP server, I mean this to me sounds like something that's been implemented a million times. Why isn't this something you can just take off the shelf? Why do you have to roll your own?

[00:29:14] SI: That's a really good question. I mean, there's definitely probably large-scale DHCP servers that are out there that have been made available by their company. But I think when you're building your own software-defined network and you have a very particular – Like a way of doing things. For example, in our case, we have an instance of OVS on every single hypervisor, and that kind has OVS flows or shuttling packets a particular way on every hypervisor. We knew we wanted to use the DHCP protocol to assign IP addresses, but then we also needed to do that within our existing infrastructure and I guess the least disruptive way possible, and I guess one way to do that would be by building a very simple DHCP server that implements just the bits of the protocol you need to kind of make the IP allocation happened, but just on a hypervisor.

Like in your home network, for example, your DHCP server is serving a lot of different computers and a lot of different machines, but in our case, we have an instance of this hypervisor level daemon DHCP server that's on every single hypervisor that just has like a much smaller, I guess – I wouldn't call it bandwidth. Just like a much smaller area that it covers.

Of course, I think there's probably third-party Go packages that we could use for that, but I think at the time we felt that kind of rolling our own would be the way to ensure that we had like the least amount of disruption to our existing architecture.

[00:30:44] JM: Just to make it clearer what DHCP is, can you go a little bit deeper on that comparison. So what DHCP would mean for my router sitting at home versus what it means for DigitalOcean?

[00:30:59] SI: So your router sitting at home would probably have I think maybe even a subnet of IP addresses that could allocate. Your router sitting at home, like when a computer comes up on the network, the computer would send over – I think probably – Or the router might send over a DHCP discover message. Your computer might send over a DHCP request message and then it would be allocated an IP address that it would essentially use for a particular period of time. Then the router itself is responsible for both determining the IP address and then allocating it to that particular host.

However, in our case, we have a totally different way of allocating IP addresses. Our DHCP server is just limited to actually using the protocol to assign the IP address. It's not actually allocating that IP address itself. We have a different kind of algorithm and a different set of services that handle the IP address management. So we just needed something that already knows what IP address it's going to give to the virtual machine, but it's not necessarily doing the process of determining that IP address. It's kind of a little, I guess, simpler. It just sends that IP address information over to the machine.

[00:32:16] JM: And let's get into talking about the engineering of this particular application. What is the code the you're writing actually do?

[00:32:26] SI: It's an interesting question you asked, because I think when I also joined the networking team, having come from a totally different world. I was like, "Okay. I know what a DHCP server is. I know what a lot of these services are, but how do you even write this?" I think in my mind I thought it would be a lot more complicated than it actually was, because ultimately all it's doing is your DHCP server almost – It's just like an HTTP server. It's like listening on a particular – In our case, the DHCP server is listening on a particular – I think we're using packet sockets. So it's listening on a particular interface on our machine just waiting for packets to come in. When a packet comes in, we are essentially reading the different like – Reading bytes of the packet, then breaking that down, reading the different parts of the packet. Based on the parts of the packet, for example the DHCP header, versus the payload, we can determine the type of request it is.

So based on the type of request, we know what it's asking for. Is it DHCP confirmation message, or is it a request asking for an IP address, or what exactly does it want? Then once we actually read that address, we might do some sort of validation to just confirm that the request is legitimate. Then all we do is we simply send a response back from our server through this raw interface, which is then sent out back to the virtual machine.

It's essentially just doing some packet processing validation. There's some logic in there. I mean, I was going to make a joke. Just a giant if else statement. It isn't. It's a little more complicated than that. Just having some logic in there, crafting a response and then maybe like grabbing the IP address if it's a request message and then sending back.

[00:34:10] JM: Was there some kind of moment where you felt, “Oh! This is actually just like writing an HTTP server,” which most of the people in the audience have probably stood up. The analogy is very close between an HTTP server in a DHCP server. They just need to do different things.

[00:34:29] SI: Yeah, for sure. I think probably after I read the entire RFC in detail and then realized that it's essentially – HTTP is also kind of request response based, and DHCP is also a layer 7 protocol that’s also a request response-based. It's very similar. We have a client. We have a DHCP server and they’re sending messages back and forth.

The main difference is, in the case of an HTTP server and especially in using the Go net package, we’re using TCP sockets under the hood. But then in the case of our DHCP server, because we wanted to kind of get the entire ethernet frame, we’ve dropped down a few layers and we’re actually using raw packet sockets. So how we’re reading the data is a little bit different. The sort of socket on the system we’re connecting to is different. But really we're just getting in some data, getting in some bytes, parsing it, reading it and sending a response back. It was definitely an aha moment I think after I read the RFC.

Then I guess the other thing that I love to do is look at existing packages and then kind of just like really dive deep into them to try to find out like within the net package or within the raw package, what are the sys calls being made? Then I understand like, under the hood, “Oh, this is how it works.” It’s like not that complicated at the end of the day.

[SPONSOR MESSAGE]

[00:35:57] JM: As a programmer, you think an object. With MongoDB, so does your database. MongoDB is the most popular document-based database built for modern application developers and the cloud area. Millions of developers use MongoDB to power the world's most innovative products and services, from crypto currency, to online gaming, IoT and more. Try Mongo DB today with Atlas, the global cloud database service that runs on AWS, Azure and Google Cloud. Configure, deploy and connect to your database in just a few minutes. Check it out at mongodb.com/atlas. That's mongodb.com/atlas.

Thank you to MongoDB for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[00:36:53] JM: Coming to another question about Go, I think a lot of people listening, probably, they've written an HTTP server in like Node.JS or Java or something. My sense is that, again, Go was particularly useful for this lower level networking that you're doing.

Just revisiting the question of Go, why was Go in its concurrency primitives so useful for writing this DHCP server?

[00:37:23] SI: I would say, in the case Go, I think just – I mean, I haven't used Node.JS in a while. But for Go, I would say that it was very useful simply because within the standard library itself, there are a lot of networking packages that – Well, two things. There're a lot of networking packages that make it very easy even if you don't have package that allows you, for example, to drop all the way down the layer two using the existing packages in the standard library. You can craft your own package to do that. So that's one thing. Then the second thing is the concurrency primitives make it very easy to handle like simultaneously a lot of traffic, but then also prevent your service from kind of being DDoS'd or knocked over if there's a lot of primitives to kind of control the number of simultaneous requests that are being processed, for example, or the amount of access to shared memory.

But then kind going back talking about the Go package, so I guess a specific example I can give for DHCP, for example, and especially when it comes to Go is that if you're using the net package in Go, you can kind of drop down to layer 4. So you can use TCP sockets or UDP sockets and then kind of process your packets. However, in the case of our DHCP server, I was working on this with my colleague at the time, he wanted to drop down all the way to layer 2 in order to kind of have access to the entire ethernet frames.

With the TCP packet, you wouldn't have access to additional headers such as the source MAC address, but he wanted to have access to that. So he looked in Go. There really wasn't anything that existed, but there was the sys call package, there was the OS package, there is a UNIX

package, there were a lot of ways to get all the way down to layer two and make sys calls. He ultimately kind of rolled his own package. That was real easy to do using existing packages that were already within the standard library. That made it very easy to jump all the way down to layer two and read from I guess the raw packet sockets.

But then when we actually do get those ethernet frames and then we want to kind of play around with them or like read the data within them, because Go I think is like a fairly mature ecosystem when it comes to networking, there were a lot of third-party packages. There's actually a package called DHCP that helps you parse like bytes of data and do DHCP packets with all the like necessary, I guess, different fields in the DHCP packet and that made it really easy to – We could easily combine a bunch of packages and then build this DHCP server fairly quickly.

[00:39:56] JM: Got it. Makes a lot of sense. Another application that you built was a load balancer, right?

[00:40:02] SI: Yeah. This is something that I did kind of outside of DigitalOcean, but it was a lot of fun because load balancers is something everyone is familiar with. Well, okay. A lot of people, maybe not everyone on the planet. A lot of people are familiar with. It's something that you probably might wonder how it works, and then I think it's something that also gets you kind of intimately familiar with the different layers of networking.

For the load balancer, I think I had – I'd been on the networking team at DigitalOcean for a while. But talking to some of my friends outside of DigitalOcean, I mentioned that I kind of just wanted to start building different networking primitives to really gain like a deep understanding of how these things worked. Obviously, they wouldn't be production-ready, because to get a production-ready load balancer is complicated. You have a lot of algorithms. You really care about performance. Throwing that all aside, I just wanted understand how it worked.

Then I also realized that there's different ways to create load balancers and you could handle everything at like layer 7 and make an HTTP load balancer or you could drop down all the way to layer 4 and then make like a TCP proxy, which you wouldn't have access perhaps to all the additional headers, but then you could have any other protocol going over the load balancer.

I ended up working with them and we kind of programmed every weekend and we made both of these load balancers then we decided to kind of compare what we could do with each one. Then the final step I guess was determining how we would fix this if we wanted it to be production-ready. I think the best example of that for the layer 4 TCP proxy, for example, is like my extremely naïve load balancer implementation. I was kind of waiting for an incoming connection and then would open a new connection to one of a few backends every time. But in reality, that's pretty inefficient. Then you're also doing like a TCP handshake for every connection you're opening to the backend. Real-life load balancers, or I guess the Linux networking stack has IPVS, which is another player four load balancer, don't necessarily do that. They might just munge the incoming packet and then send it straight to the backend and connections are formed directly with the backend rather than as an intermediary within the load balancer.

[00:42:14] JM: You're writing of a load balancer was to prepare you for writing the DHCP server.

[00:42:22] SI: Not quiet. I think it was actually after the DHCP server. I think the writing of the load –
[00:42:26] JM: Oh. Okay.

[00:42:27] SI: Yeah. I guess this is something I want to start doing again. It was sort of like a personal exercise where I wrote down a list of, I guess, different networking primitives that I wanted to know how they worked and I figured the best way to do that would be by building one. So I think load balancer was one of them. Port scanner was one of them. Then I think at some point I just want to build a router just to see how that works. It was part of like an exercise to see how to do it, and then I figured it would obviously be useful for work just like figuring out how I would theoretically build these things in the event that I ever had to – I mean, to be honest, I think most people actually use – There's a lot of open source load balancers that are really good. Sometimes people use balancers. So maybe not load balancers, but other services.

[00:43:17] JM: Something I'm pretty sure you actually did – A different project you did for work was you worked on sharding a Prometheus server, right?

[00:43:24] SI: Yeah.

[00:43:26] JM: We've done a couple shows about this topic, but my understanding is basically Prometheus is a really widely used monitoring tool in the Kubernetes ecosystem, although it doesn't – You don't necessarily have to use with Kubernetes. It's like a cut distributed monitoring or a metrics server. If you collect a lot of metrics from Prometheus, you're going to have a large set of metrics. You're going to have a gigantic database of metrics. There have been a multitude of scalability projects, people trying to scale the Prometheus database. Am I describing that problem statement correctly?

[00:44:03] SI: Yeah. That's definitely correct. I think there're been a lot of different approaches. It's something that we kind of also faced at DigitalOcean. In our case, we had moved to using Prometheus a few years ago. Things were going well, but I think things are going so well in fact that a lot of teams spun out their own instance of Prometheus, which is great, but when it comes to managing all these instances of Prometheus, there is an observability team that was formed, and I think that the theory was that it would be best if the singular observability team were able to kind of handle everything to do with Prometheus, like just monitoring Prometheus itself. We're doing a lot of self-monitoring, monitoring different from Promethea. Making sure they could update the Promethea and then kind of just being there to help offer support and guidance for different teams that were using Prometheus.

To do this, they decided that they needed to kind of have like a horizontally – Or I guess functionally shorted Prometheus set up in a way to kind of dynamically configure each of these Promethea especially – Or I guess there's a lot of arguments about the plural of Prometheus, but I'm going to go with Promethea for the duration as this talk.

[00:45:08] JM: That's fine. I accept.

[00:45:09] SI: Excellent. There's a lot of talk about how to actually configure all these Promethea, because when you end up with like 200 instances, it's obviously not – It's not really scalable to manually handle anything. Especially when you have a lot of services that are

coming up or going down and then you also have Kubernetes clusters that you're like managing, it's best to kind essentially manage this in an automated way.

One of my previous colleagues kind of came up with an interesting solution where we have kind of a – I think there's been a lot of talks on this as well at PromCon where we have an internal repo where all teams will add whatever services or like – I don't know, databases or what have you, they want to have scraped. Then we actually have CI that runs. Kind of parses the changes that the team has made to this particular repository. Of course, managed by YAML files and then kind of pushes these – I guess it's like almost like a controller and then a bunch of different controllers on the different Promethea instances.

These changes are pushed to the central controller and then that sends whatever changes to all of the controllers that are running as daemons alongside the different instances of Promethea and updates their configuration dynamically so they happen to know what new services or what new instances they should be scraping.

I think it was actually a very clever solution that my colleague came out with and it definitely, I think, has just made it so much easier to kind of manage all of these different services that are being scraped, and especially as new services are being added or even new hypervisors or new droplets are being added, it makes it like flawless in a way.

[00:46:48] JM: It's remarkable the challenge that you encountered with the Prometheus scalability and the fact that you just have all these different teams that are scaling up their different Prometheus servers. I feel like I've heard that story a couple times at least once talking to people from Uber. Have you talked to other people at other companies that have had similar issues with scaling Prometheus or creating observability systems or protocols within the company for dealing with this particular piece of infrastructure?

[00:47:17] SI: Yeah. Yeah. I definitely have. I think a lot of people have first maybe tried to kind of vertically scale it or just put Prometheus on progressively larger virtual machines until of course they realized there is a point at which they absolutely had to shard it and then it kind of came down to what's the best way to sort of shard or like divide management of all these different services.

I think the interesting thing is I do think every – The trend is that everyone kind of proceeds to maybe the same way of functionally sharding. It's a term that someone from Prometheus used when I kind of described how we were sharding. He was like, "Yeah, that's really how Prometheus was created to be used. Not to be like one gigantic instance, but something that's functionally sharded and can be used with like different clusters of services."

But then just when it comes to figuring out how to manage those clusters or services and update that configuration, I guess that's where things get a little complex. Then, yeah, I would say some companies probably moved to like an on-prem solution or something where they create something themselves. Then I think a lot of other people moved to some sort of managed Prometheus service. I think there's a couple out there that have been created by people who were part of the Prometheus founding team. Yeah, I do think that a lot of people trend towards kind of the same problem and then either build something themselves or find something that already exists and pay for it.

[00:48:36] JM: What other observability challenges have you encountered within DigitalOcean?

[00:48:42] SI: That's a really good question. I think probably the biggest issue that everyone always faces is maybe like absurd cardinality in their metrics, where in Prometheus, when you add a particular label, that actually creates an entirely new time series. For example, adding a UUID as a label is kind of like a terrible idea in general, because you're adding kind of a new time series for every label and you don't really want super – That's something that is better to show up in logs versus in metrics, but unfortunately especially when you're using a lot of third-party exporters, they don't have the best practices or the best behavior for metrics labeling. So you end up with situations where you have cardinality explosions. For example, I think there're some CICD metrics exporters that have been created by third parties that just kind of add a label. They add a lot of labels for different CI runs, but that creates a massive amount of labels and a massive amount of time series and that could in fact just cause the time series to take up a huge amount of space on-disk and then the Prometheus server falls over.

I think teaching people – Or learning how to properly use metrics and then also kind of navigating the whole area of using third-party metrics exporters for existing things like CICD or

databases and then making sure that those don't actually kill your Prometheus server is definitely one of the challenges I would say that that's been faced.

[00:50:09] JM: While we're on the subject of – Well, because we're talking about Prometheus, I guess that's tangentially related to Kubernetes. But I would be curious about how Kubernetes has affected DigitalOcean. I mean, you mentioned from the product perspective, the idea of spinning up a Kubernetes cluster on DigitalOcean is obviously useful. I'm also just curious if it's useful internally, if Kubernetes is used by internal teams at DigitalOcean or how the technology has affected the cloud provider overall.

[00:50:37] SI: Yeah. Before we even had Kubernetes as a service, we used – Or we offered it as a service. We used it for a very long time. I think, honestly, it was amazing introducing Kubernetes as a service. Prior to that – This was a few years ago, and actually this is the team that I was first hired to work with. Most people would have like the Chef guy on their team. The guy who is really good with Chef and really good with Ruby and writing Chef tests and everything.

Quite often – And I'm not even joking about this. It would take longer to configure a Chef cookbook or a recipe or make changes to Chef tests than it would take to actually update a service itself. It was quite complicated and not everyone knew how to do it well. So we went from kind of that era to having an abstraction over Kubernetes, which is called DOCC, and then having a CLI tool to interact with this abstraction. It was incredibly simple, and just having the ability to deploy applications within seconds.

I think the best evidence of that is we actually had a hackathon maybe like a few months or a year after our abstraction layer on top of Kubernetes that we used internally was rolled out to different regions, and it was crazy. I think there were like hundreds, if not thousands of applications in the hackathon that were just really easily deployed to some of our like stage in Kubernetes servers. I feel like that's like the best evidence that it just made it very easy to deploy and update applications.

Then the other thing is it also made it very easy to add TLS and security to your applications, because we had a really interesting way where we used different Kubernetes primitives and sidecars to kind of almost automatically do that.

[00:52:21] JM: Cool. Well, Sneha, it's been really great talking to you. I wonder if you have any closing reflections on working at a cloud provider. I haven't interviewed too many people that work at cloud provider. You also work remotely, which I actually didn't know about that before this call, but I always thought of DigitalOcean as a – I mean, because I've been to the office. So I didn't actually envision that it was a remote organization. I also think of cloud providers are these people that have to work all centralized together because there's like servers nearby. But I realized that the people who actually build and maintain the servers are at a very different place than the people writing the code that gets deployed to those servers. But maybe you could just give me some reflections on working at a cloud provider and the kinds of problems you're working on today.

[00:53:05] SI: The funny thing is I think a lot of people have that view as well, but the company is actually 60% remote. I think especially when you're a cloud provider that have co-located data centers, there's often like another – Of course, we have people at data centers like handling everything when it comes to like physical server stacks, but there's also like teams to help just with emergencies as well. So there's definitely people like physically there, but I would say like a lot of the work, believe it or not, is kind of done not from within the data center itself.

I personally – And like I think just in general about working at a cloud provider, I think it's amazing largely because there's so many different areas that you can dive into. Especially, I think at DigitalOcean, like on the cloud side, once I moved from kind of like an internal team that serviced to other developers to being closer to kind of like our cloud primitives, I realized that there's a lot to learn like on a near constant basis.

Right now, I'm in networking, which I find fascinating and I've like worked on multiple projects in networking. Previously, I worked on DHCP. Now I've moved to a team that's kind of like focused on some of the scaling issues at DigitalOcean and kind of trying to change our networking architecture from layer 2 to layer 3 and then counted looking at how that'll improve performance and then kind of looking at how to fix like existing issues of maybe IP addresses, for example.

I think that the problems that you often face at a cloud, there's like not that many people who deal with that level of scale or like that many companies. I think you constantly find super interesting problems or scalability issues and you can like really drop down pretty low and do a lot of systems level work. I think for that reason it's been a lot of fun working here, and I've even thought like if I didn't do networking, what would I do? There's like an entire world of storage that I haven't even explored that's also incredibly important when we have a lot of storage products such as object or block. I think for that reason, it's very interesting and like you constantly see new problems.

[00:55:12] JM: Awesome. Well, thanks for coming on the show. It's been really great talking to you, Sneha.

[00:55:15] SI: Thank you, and thank you for having me on.

[END OF INTERVIEW]

[00:55:26] 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.

[END]