EPISODE 733

[INTRODUCTION]

[00:00:00] JM: Software companies today rely on group chat applications. The world of startups and small businesses is dominated by Slack, but for some large enterprises, regulatory constraints prevent them from using Slack. Slack is a web application that is hosted in the cloud and regulated industries such as banking often need to run their applications on their own on-prem infrastructure. Mattermost is an open source alternative to Slack that can be self-hosted, and this means that all of the networking complexities and scalability challenges that are controlled in the cloud by Slack need to be handled by open source code rather than by close source code and managed services that are running in the cloud.

Because it is open source, Mattermost has some interesting challenges, but there're also plenty of advantages. Mattermost can also be redesigned and customized. For example, Uber designed their own custom version of Mattermost called uChat.

Corey Hulen is a cofounder and the CTO of Mattermost. Corey joins the show to discuss the motivation for building Mattermost and the engineering challenges of building an open source chat system. For more episodes about building chat systems, we've done several shows about Slack covering the engineering, the security and chat system within Slack, the chat system architecture, and if you want to find all these episodes, you can always download the Software Engineering Daily apps for iOS or android. We have links to those at softwaredaily.com as well.

I want to mention that we're also looking for roles at Software Engineering Daily. We're looking for several different roles; journalists, and an entrepreneur in residence. We're looking for somebody who wants to research a specific topic and build a business around that, and we're also looking for a few interns who want to write iOS or android code. So you can find those job listings softwareengineeringdaily.com/jobs and would love to hear from you.

[SPONSOR MESSAGE]

[00:02:13] JM: Kubernetes can be difficult. Container networking, storage, disaster recovery, these are issues that you would rather not have to figure out alone. Mesosphere's Kubernetes-as-a-service provides single click Kubernetes deployment with simple management, security features and high availability to make your Kubernetes deployments easy. You can find out more about Mesosphere's Kubernetes-as-a-service by going to softwareengineeringdaily.com/mesosphere.

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

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

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

[INTERVIEW]

[00:04:32] JM: Corey Hulen, you are a cofounder and the CTO at Mattermost. Welcome to Software Engineering Daily.

[00:04:38] CH: Hey, thanks for having me.

[00:04:39] JM: Mattermost is a chat service. It's much like Slack, except it's open source and it can be self-hosted. Why do people want to host their own chat system?

[00:04:51] CH: There is a lot of reasons, but probably the biggest one is, is just security for high-trust organizations. We have a lot of customers and clients and users who really care about their privacy and care about their data. For them, that's a really big aspect of why they would choose Mattermost.

[00:05:09] JM: The famous story about Slack is that it was originally a chat system inside of a game company, and in the game company wasn't really working as a business. So they pivoted to doing a chat system. Mattermost apparently has a similar story, having been started within a game company. Can you tell that story?

[00:05:29] CH: Yeah, sure. I can give a high-level. It is a very similar story in the sense that we're a game company doing Facebook games. A lot of the original technology came from the chat system there or whatever. We've since sort of rewritten it in Go, but for us it just the frustration of needing a messaging platform and the frustration of moving messaging platforms and not being able to take your own data with you or you own your own data. That's also a really big core premise of what Mattermost is built on. It's not only open source, but in our mind is open data. You own the data. You control the data. It's yours. It's your intellectual property. You can do what you want with it.

For us, a lot of it grew out of our frustration of being in different chat systems and not being able to sort of own or migrate your data to where you want it, and we have, you could say, similar technology in terms of how to do messaging or chat. So that's sort of where we started with, what we started with.

[00:06:23] JM: So you get involved in Mattermost in 2014. I think this was at some point after the company had gone back and forth between the gaming and the messaging ideas.What made you join the company?

[00:06:35] CH: Yeah. So I came on it basically at the start of Mattermost with the intention of doing something like Mattermost. For me, that was a really big – A really big thing. Ian and I used to work together. We worked together in previous lives of different companies. One of the things we talked about is – Back then – I mean, maybe today, it's more obvious. But back then, for us, one of the things we talked about was just chat and how it's really a platform. If you wanted to make that platform ubiquitous, in our mind you needed sort of an open standard. The best way to do that is to be open-source.

So for us, that was a large part of it. It’s just building this messaging system as a platform. In fact, the original binary was named platform, which is kind of funny. But for us that was really the start of the vision where we wanted this messaging platform, and then we wanted to build verticals on top of that. So you could think of business intelligence or artificial intelligence as a vertical sitting on top of that platform. That's actually my background. I have 20 plus patents or pending patents in both those spaces.

But then you can even think of like true verticals, like sales, or human resources, or real estate sitting on top of this platform. So for us that was one of the sort of key visions in the beginning, was just this sort of open platform that you could then build on top of.

[00:07:52] JM: Interesting. This is in some contrast to Slack’s vision for the platform, which is built off of bots and API integrations, like GitHub. But your vision sounds more like what you've seen with Uber. Uber adopted Mattermost, but they adapted the application to suit their specific needs. Since you have the open model, you can really have an opportunity to hack on the chat system and build whatever you want.

[00:08:24] CH: Exactly. We call it layered extendibility, and I could kind of go through all those different layers, but there’s probably a few very interesting ones. So if we start from the sort of bottom-up maybe is the best way to think about it. For us, we’re an open data platform. You own your data, right? On top of that, we’re open-source. Now those two things buy you a lot, but there's still a lot of freedom and a lot of complexity with trying to do things at that level.

Then we started layering on futures on top. So we have a very rich plug-in framework where you can write plug-ins in our system that can do things you cannot do in other chat system, because you control it. A great example that we like to use here is we actually have an intercept hook, when you can intercept the message before it goes into the database or after it comes out of the database at this level. So that creates really interesting scenarios where you can intercept something that may look like a key, or a password, or a social security ID and say, “Hey, do you really want this message to go into this system of record?”

On top of that we also have sort of what you think of as Slack compatible features, like webhooks and slash command and stuff like that. So you get this really rich, deep what we call layered extendibility or developer toolkit that sits on top of it, and depending on your organization's sort of level of comfort, they can start at the top and do very simple, quick things with webhooks. They can do more integrations with our rich plug-in framework, really deep integrations that you can't do in any other platform, or, for us, at the end of the day, we are open-source. So if you want to bring your own resources and kind of really do something kind of awesome and crazy, go for it.

[00:09:58] JM: It sounds in some ways like WordPress in the sense that WordPress is open-source. It's been hacked on. It's been adapted to different applications. I think WordPress gets changed or personalized for different companies that want to host it and maybe they alter the WordPress core code base slightly, but then they also have this plug-in marketplace. So there're different points of integration that you might want on WordPress kind of analogous to what you're doing at Mattermost.

[00:10:31] CH: Yeah, exactly. I think very analogous in lots of respects. We want to create a really rich ecosystem of these plug-ins and hooks and various things that sit on top of Mattermost, that sit in a marketplace kind of analogy where you can do all these really rich integrations and you can use it in such a way that may not originally been designed for. We see this happening all the time.

There's obviously our sweet spot of dev ops, chat ops, infosec, developers, but then we also have this other sort of interesting cases that come up from time to time. Like we have some customers who use us – So warehouse employees in a big warehouse can message each other on their mobile devices.

We have some use cases that end up being very interesting, and for them they custom theme that app. In fact, you don't even necessarily know it's Mattermost that's running. It's a very IT centrally managed to lock down app in that instance. So because of the flexibility of being open-source and having these different sort of integration points or layered extendibility points, like you can do some really rich interesting things, and we see customers doing that all the time, which is amazing.

[00:11:37] JM: Give me an overview of the engineering stack that Mattermost had when it was initially built.

[00:11:43] CH: Way back, as a game's team, a lot of it was originally built in Python and it was just JQeuery, JavaScript sitting on top of a Python backend. We started really thinking about wanting to do this as a separate product. We kind of looked at a lot of different technologies out there and we made a bet on Go. This is – What? I don't know. Three years? Maybe four years ago now?

Today it's a pretty safe bet. I would say back to then, not so much. One of the things we even struggled with back then was is there enough third-party packages that we want to leverage or rely on to help make our product successful? Today, it's sort of I think a great choice. Even for us back then, it was a great choice in the sense that it's such a – Go is such a powerful language, yet simple language. Especially as an open source community, allowing an open source community to come in, experience writing something and being very productive in Go and having a strongly typed language.

I love Python. I think it's a great hacker's language, but if anyone's tried to debug a million lines of Python code, you felt the pain before. I think being open-source community, one of those things is just having a strongly typed language, a very simple language. So for us, on the backend, that was really powerful.

The other one was the frontend. JavaScript is one of those, I don't know if you can call it necessary evils. It's the language of the browser. Everybody has to do it. But I think when you look at all the different frameworks out there, we chose React. We've had a lot of success with that and really loved it. You know, we were with React in the sort of pre-Flux days where you kind of did everything ourselves. Then we actually migrated to Flux style stores and then – Or maybe not recently. In the last year, we migrated everything over to Redux, and that's been a great experience.

Kind of the same thing there. I think as a technology, especially around technology running open source product wanting to, not only myself, work in sort of cool, new and emerging technologies, it makes it really easy as a community when your community wants to contribute, because you work in Go, or React and they want to get experience or see how really large projects are built.

For better or worse, ours is a really large project and I think there's – I think anyone say, it's like any codebase, there are some things we really love about it. There're some things that in hindsight we're like, "Yeah, probably could have done that different or better," but it's there. So it's really cool to see that.

[00:13:57] JM: You started with a stack that was Python and JavaScript. Python on the backend. JavaScript on the frontend. Then you rewrote it as Go in the backend and JavaScript – Well, React on the frontend. More specifically, what were the issues with the original stack that weren't going to work out, that made you want to re-architect it?

[00:14:17] CH: I mean, there are several, but I think probably one of the biggest ones was just simplicity of having a single compound binary. We knew people were going run this on their – We knew from the beginning we wanted to be on-prem, in the sense that we want people to run this on their own infrastructure and their own data centers or whatever it is. There're just a lot of little things like that that make it very easy for better or worse.

A single binary, it's just an amazing experience from a customer standpoint. You don't have to worry about what version of Python or what version of Node you have or anything like that. It's just a binary. You have a Linux box. You throw it on there and it works. For us, that was a large part.

In fact, a piece I didn't talk about was our database technology. We chose MySQL and PostgreS as our standard, and that was a very conscious decision. I have a lot of Cassandra or big data experience from previous lives, but we made that as a conscious choice, because same kind of thing. When we're talking about the customers and users we want to go to, SQL is pretty ubiquitous. People know how to scale it. They know how to set it up. They know how to pull data from it. Especially when you start talking about an open data platform, like the easiest way to get data out of those types of tools. There's a lot of experience out there.

For us, those were all kind of the factors that went into it. That and just ease-of-use. Knowing that we're going to have – Wanting to building an open-source community around this and knowing that we want –We'd prefer I guess is the best way to say it, a strongly type language. A language that's very simple and easy to grasp, but yet still very powerful. I think those are all things that sort of went into the decision.

[00:15:44] JM: If I'm an enterprise and I decide I want to have Mattermost within my enterprise, what's the process of deploying it?

[00:15:53] CH: Yeah, that's a great question. There's several different ways today. For us it's just kind of you go through the documentation and you install it on bare metal. That's actually not too hard. Our goal is to get to that classic WordPress five-minute install. I think we hear back ours is about 20 minutes. So we're super happy with that. That's one way.

But we also offer a lot of tools, because we live in a cloud world or whatever. So we offer a lot of tools, like Terraform scripts to help you set up in AWS or Azure. But one of the other big pushes we've been making lately is this is sort of more of a cloud native framework around Kubernetes. Making sure our application runs there really well. Making sure we can support that for large enterprises. So that's another thing we see as well.

We kind of offer several different flavors of how to install the software, but I'd say those are sort of the main ones. One is just read the documentation, install it on bare metal. Two is use our Terraform scripts. Three is run it in Kubernetes.

[00:16:42] JM: With Kubernetes, how many nodes are needed for a typical deployment and how has that multi-node situation evolved since the days before Kubernetes?

[00:16:54] CH: Yeah, that's a great question. I mean, I think you could get down to probably the simplest installation process where you reconsider a node or machine or you're not running multiple instances on the same machine or whatever. For us it's that kind of classic [inaudible 00:17:08] architecture. We have a proxy. We typically recommend nginx. In Kuberenetes, you don't necessarily have to have that there, but it does give you some other interesting things.

For us, there's the Mattermost app server, which is just a single binary. Then there's the database server that can be a single node as well. Now when you scale that out, you can scale that out horizontally. So our app layer is written in such a way that it's a masterless technology. Meaning, as new Mattermost nodes are spun up, they join the cluster, they start gossiping to each other and they can pass traffic back and forth. So it's written in such a way where it is almost stateless. Being a communication app where you have to hold the web socket connection open to that cluster. You could think of that as our only little bit of state. But the apps are pretty smart and everything is written – The client-side apps are pretty smart an everything is written in such a way when that state disconnects out, reconnect it to another node.

Then on the database side, you can kind or – Our classic one is a single database, especially if you're in a smallish sort of use base, but the sky is the limit there. We have some customers who have multi-master active-active configurations or who have master slave configurations and they're running 8 to 12 slave nodes. You could think of each of those 8 to 10 slaves, you could think of each of those nodes, each of those running in a different node in Kubernetes and even in potentially different regions depending on how you set up.

[00:18:33] JM: If I'm an enterprise and I set up Mattermost across a Kubernetes cluster, is there anything operationally difficult with managing that cluster as it grows over time, or is this something that – If I'm the enterprise, do I pay Mattermost to help me manage and scale this cluster? What's the scalability story?

[00:18:59] CH: Yeah. The scalability story, I mean a lot of this since it's on – Sort of in your own network, like you sort of have to figure out what you want to do. So we have a lot of customers who have small instances who care about things like uptime or not being done a whole lot. We have other customers that want to hit four, even five nines worth of uptime.

On those scenarios what we see is those teams or customers either work closely with us or they have their own internal teams where they're actively monitoring the system and scaling their resources accordingly.

Mattermost, the app server can elastically scale obviously up and down. Scaling the database up and down is a little more difficult for those who've done it, especially scaling things like read replica and stuff. So depending on where we're running at, you could even leverage certain technologies like Amazon Aurora as your backend in that essence.

For us, elastically scaling the app up and down is pretty easy. There's a lot of active caching built into the app. But what we find to be the bottleneck is scaling the database. Elastically scanning the base, that's a little bit more challenging subject. What we find most of our enterprises care about is just high availability of that database, making sure that they can either failover in a blue-green style to a different region or an entirely different cluster, or have an active-active database where they can failover in that essence as well.

[00:20:21] JM: What does that mean? Failover with the blue-green – What did you say there?

[00:20:25] CH: Yeah. A blue-green is like you have – Some of our large customers don't – I mean, they care about scalability, but they care about things like high availability and uptime more. So a classic scenario for us when we do that is what we call like a blue-green cluster. So you have a cluster that is active and Mattermost is all running there. Everything is running hunky-dory and you have an entire sort of duplicate instances of that where you have the entire Mattermost cluster and a database that's being replicated on the backend to a cluster that's just in standby mode. It's just [inaudible 00:20:55].

With a simple script or a simple chat ops command, you can basically redirect your entire traffic, whatever, 60,000+ concurrent users, from one of those sort of clusters or regions into another cluster region. So we have a lot of customers that want that kind of high-availability, and they want to do it between regions. I want to be able to failover from the U.S to Europe, or vice versa. So for us that's a really interesting scenario. That's some of our really large customers want to get into. Where it's fun, where you can almost – Machine resources at that point aren't necessarily an issue in terms of how they want to scale or how much excess capacity, because they'd rather have uptime guarantees. It's fun to see really crazy scenarios like that where you're failing over tens of thousands of concurrent users between regions.

[00:21:45] JM: How much do these deployments vary from customer to customer?

[00:21:51] CH: They can vary a lot. I mean, we have sort of – It's one of those things that's really kind of cool. You have that small, "Hey, if you have 500 people, great. Just run a single binary, run a single database. Don't worry about it. It's as easy as WordPress." That's kind of our attitude.

But then we also have, going from sort of 500 issues or customers are going to 200,000 users, 60,000 concurrent actions. So that's a lot of configuration and horsepower putting into it and a lot of different use cases, where it in the first use case it might be a small team with an organization. The second use case, it's a very big either government or really large corporation who's wanting to roll it out across their entire enterprise. The level of responses in the app, the uptime in the app, and those two situations have an extremely different expectation. At the team level of 50, 500 users, yeah, if Mattersmost is not up, it's no big deal. At the enterprise level where it's going across 50, 100, 200,000 users, uptime is a pretty big deal.

So some customers like that, where you've picked the most low usage point in the server in the most off-hour and they still have 15%, 20% of their traffic running in any 24-hour period. So you can't just, "Hey, we're going to be down for 30 minutes to do maintenance," or whatever at that point. So you end up having to come up with creative solutions within those own customer's environments, because lots of times this is running on-prem within their infrastructure. So it makes for a lot of fun and a lot of challenge.

[SPONSOR MESSAGE]

[00:23:36] JM: Managed cloud services save developers time and effort. Why would you build your own logging platform, or CMS, or authentication service yourself when a managed tool or API can solve the problem for you? But how do you find the right services to integrate? How do you learn to stich them together? How do you manage credentials within your teams or your products?

Manifold makes your life easier by providing a single workflow to organize your services, connect your integrations and share them with your team. You can discover the best services for your projects in the manifold marketplace or bring your own and manage them all in one dashboard. With services covering authentication, messaging, monitoring, CMS and more, Manifold will keep you on the cutting-edge so you can focus on building your project rather than focusing on problems that have already been solved. I'm a fan of Manifold because it pushes the developer to a higher level of abstraction, which I think can be really productive for allowing you to build and leverage your creativity faster.

Once you have the services that you need, you can delivery your configuration to any environment, you can deploy on any cloud, and Manifold is completely free to use. If you head over to manifold.co/sedaily, you will get a coupon code for $10, which you can use to try out any service on the Manifold marketplace.

Thanks to Manifold for being a sponsor of Software Engineering Daily, and check out manifold.co/sedaily. Get your $10 credit, shop around, look for cool services that you can use in your next product, or project. There is a lot of stuff there, and $10 can take you a long way to trying a lot of different services. Go to manifold.co/sedaily and shop around for tools to be creative.

Thanks again to Manifold.

[INTERVIEW CONTINUED]

[00:25:50] JM: You mentioned some challenges around the database scalability. Can you tell me more about the kinds of bottlenecks that you hit when the organization scales up or somebody's just sending tons and tons of messages or maybe you have a chat ops situation where like all the logs accidentally get funneled through Mattermost and then it just DDoSs the database? What kinds of issues do you have there?

[00:26:16] CH: Yeah, exactly. We have all those kinds of issues. Probably one of the funniest ones is we have some customers doing more than a millions a day inbound. If you think about it as a chat system, that's a million messages in that could be anywhere from 10 to 20 million broadcasts out. We have some customers where 50% of their traffic is actually from bots, right? Having a bot DDOS the Mattermost cluster is a very – I don't want to say common occurrence, but it has been known to happen. So we put a lot of stuff in there around things like a rate limiting and whatnot, but those are all the things that kind of occur and we've done a lot of sort of work in making sure those things happen less frequency. So those are all fun scenarios.

In terms of scaling the database, for us, relying on SQL technology, read replication lag is a big thing in our world. We've written a lot of smarts in the application to actually handle replication lag between master and slave. So we can actually tolerate a pretty high replication lag. We also do a lot of active caching in the server itself. So we can tolerate a lot of slow data, data that's slow to propagate.

But I'd say that's probably where a lot of our – Most of our issues come in. It's that classic trade-off of cost versus how highly available and how many different regions do you want to be. When you start talking about regions in the other side of the world, you do have to have a pretty low latency network to make that work and stuff like that. It's fun.

The cloud provider is a little easier, because we have very standard methodologies or tools to understand what their replication lag is and whatever, but when you're talking about an on-prem customer who has, let's say, their own physical data center in Europe, but they're using also AWS in the U.S., then you end up with really interesting scenarios of needing to span across those sort of hybrid environments.

[00:28:03] JM: We did a show, actually one aired yesterday, about Slack, scaling their messaging infrastructure. They have these situations where you've got a company with a 100,000 users in the same room. Do you have these kinds of issues with that volume of users and there any kinds of engineering problems you've seen that's unique to that volume of users?

[00:28:27] CH: Yeah. No. Definitely. We have that same challenge. We have people who have – We have current customers today who have war rooms, and those war rooms are generating more than – Can generate during peak load as much as 100 messages a second. A human can't even read them at that point. It's kind of fun or interesting to go see them, because they even have – Lots of times they have bots putting messages and pulling messages out.

So one of the things we did early on, and this is actually originally, when you think of this, a gift to the open source community from Uber, was a load testing framework, an open source load testing framework. Our sits out there. It's open source. It was originally sort of written by Uber. It was given to the community. We now maintain it as a community. So we regularly test Mattermost up to 60,000 concurrent users. So that's 60,000 web socket connections held up into the cluster.

Based on our knowledge of how customers use the application, that's probably somewhere in the neighborhood of 150 to 250,000 total users. So we have a load test instances where have 60,000 users all watching Town Square and watching messages flow by Town Square. We have customers, real-world customers, with kind of extreme scalabilities in that sense. They want these war rooms. They want all of their people out there, the entire enterprise in some of these things. So for us that's where you put a lot, a lot of our effort. Like I said, we publish our numbers. We publish our open source load testing tools. So if you're a particular customer and you have really interesting needs, like our recommendation is always like, "Hey, our load test tool is open source. Just go set up your environment and configure it in how you think you would like it to scale and what kind of horsepower you'd like to kind of put behind it and then just run the load testing tool against it."

Our load testing tool is updated very frequently based on real-world data that we capture from our customers that our customers share with us in terms of what are the hottest routes, what are the ones that we need to go optimize? We put a ton of work, and you can even say that was a really strong collaboration with Uber helping us do that work, and it was an amazing experience to work with them and to sort of leverage some of their resources in some respect to help us scale the system.

[00:30:37] JM: Yeah, let's talk a little bit more about that. So Uber wrote a post, a blog post last year about how they use Mattermost, or a fork of Mattermost. First of all, why did they start using Mattermost and then why did they fork it and make their own version?

[00:30:52] CH: Yeah, that's a question you probably have to ask Uber. I'm not sure how much I can talk about that. I mean, to high-level, there's two things right. They wanted a very high-performance system. Two, they're very much a company that wants to sort of control their own destiny and control what features and what – They do a lot of really cool stuff above and beyond Mattermost. Like using Mattermost as a platform to build on top of.

I probably can't go into the specifics of that, but they do some really cool stuff. I think for them, that both high-scalability coupled with high-flexibility in terms of a platform is sort of why they us. That would be my guess.

[00:31:34] JM: Yeah. Well, maybe you can connect me to – Or I can just send an email to the – If somebody's out there listening, if somebody from Uber, I would like to do a show on that. That'd be really interesting.

You have a pretty unique product. This open source tool that is deployed on-prem, but it does something that is similar to this proprietary tool that tons and tons and tons of startups use, Slack. I'd like to know how that translates to how you arrange the engineering team and the product development team and how you think competitively how to structure your teams to really double down on your core competency and establish yourself in a rock-solid position. Because, obviously, like if you can get there. I mean, it sounds like you really are in the lead at this point, but it's just such a great business and I assume you're really trying to get things, the structure of the company built so that this is a sustained advantage.

[00:32:34] CH: Yeah. For us, I think we love open source and love the community. So we want always some very large subset of our community to just be able to use Mattermost as it is today for free. But what we find is there are those customers who come from very high-trust or very high-privacy sensitive environments. Just even a lot of European governments or Europeans in general come from that space, which is a very different attitude than over here in the U.S.

For us, that's tend to how where we structure our time and effort from a company standpoint. Things around like security and compliance, high-availability, scalability, the things that resonate with those super large customers. So for us, it's that sort of facetted reproach of making sure we're still very honest and open with our community.

I always like to say, the only way to do that is just be honest and open. We actually spec everything in public, design everything in public. When we're talking about, let's say, an enterprise feature. We openly discuss it with customers and community like, "Hey, this is what we're working on." We probably way over share in terms of that content. About the only thing we don't do that point is little tiny bit of source code that we have in the enterprise version. But for the most part, our goal is to make sure a large part of our audience can just use us and build on that platform.

For those large customers that really care about things like security, or privacy, or high-availability, running in their own data centers. Lots of times we have a lot of customers like that, running on air gaps network or networks that are completely isolated from the internet or whatever. So that's where we tend to focus – From a business side, that's where we tend to focus our efforts.

[00:34:15] JM: What's been the hardest part of building Mattermost so far?

[00:34:19] CH: That's a great question. I don't know. I don't know if it's the hardest part, but like community is this amazing experience, being able to leverage community, I think for us is directing and guiding that community in sort of an honest and open way. It's always something we're very conscious about. I think when you get a community, or a community member who's really interested in contributing or doing something, being able to shepherd that process and direct them in such a way that benefits other community members.

I think that's one we always sort of think a lot about and put a lot of time and energy about. We're successful because of our amazing community, Mattermost, the open-source project, and I think giving back to that community is something that as a company you have to really sort of put time and thought and effort into it and making sure you're doing it in an honest and open way. For us that's something that always kind of linked to the top of our mind.

[00:35:15] JM: I don't know how much you have focused on this since you're a CTO. So you always see the whole infrastructure, but have you seen any unique challenges to managing a large electron app? I guess I should say, for people who don't know, electron is a way of basically running a Chrome browser, but it's like an application that runs on your desktop. So a lot of desktop apps these days, Slack, for example, are electron apps and they're built entirely in JavaScript frameworks, just like a web app, but it feels like a desktop application. So have you seen any challenges to managing a large electron app?

[00:35:50] CH: That's a great question. I don't know if there's any challenges to electrons. Specifically, maybe more challenge is to us. I don't think they're even challenges, but maybe uniqueness to us. A large part of electron app is actually developed by the community. So it's fun or interesting having people that we call core committers. They are community members who were not Mattermost employees who sometimes have as much access or as much rights to our GitHub repos or our private channels as I do. It's a fun and unique experience. I think you only see in this world of open source. For us, leveraging our community to sort of do that.

The interesting thing, the person who actually leads up our electron app actually sits in Japan. That's in a time zone that is extremely different from the majority of the company. So we actually dog foot Mattermost ourselves. We have a community server. It's where all of our community sits. It's where all of our employees sit.

I mean, interact is one big asynchronous written form communication project, and it's amazing. It's amazing to see that work and to see the challenges of that, especially when you're talking about being in all time zones around the world. But it's something I think we as a community have – I'm very proud of. We've done really well, and it's by evidence of the fact of things like this. The person who, for us, leads the electron app effort is a core committer who sits in Japan. We interact with him on probably a daily basis, but in just very extremely different times. So that's really fun to see.

Electron, specifically, I think it's been a great technology for us. It's really allowed us to get a desktop app out there and to make customers really happy. I think the hardest thing for us, I think is more in general, being on-prem software, making sure from a security standpoint and stuff like that. Those things are very important to us. [inaudible 00:37:35] things like electron or other packages it uses is always top of our mind, top of our radar. But I don't think that's any more so than probably any other sort of project that uses it.

[00:37:46] JM: So the open-source community, that sounds cool, because you see a lot of these companies where the open core model or the open source business model, the people who are actually writing the code are almost all at the company, like the company that makes the open-source project. But it sound like you've got a community of people who are not even – Who don't actually work for Mattermost that are contributing to the code.

[00:38:11] CH: Yeah, we do. We typically tend to hire those people, but we have a really large set of core committers who not only commit to the Mattermost project itself, who commit to a lot of integrations for us, which is amazing, or a lot of these other libraries. I can go through and name different repos.

Our Docker repo is actually maintained 100% by a community number, our electron repo or desktop app. There're just large things like that that are sort of maintained and developed by the community. A large number of our integrations are developed and maintained by the community. What the company tends to focus on is kind of two key areas. Those things that ethically we think we're going to charge from, we don't want community work on, because we just don't feel like we should do that.

The other thing is sort of all that [inaudible 00:39:02] that community doesn't want to work on, like the hard work – I don't want to call them the hard problems, but the esoteric performance issues, or the really hard things that take a lot of resources to do. Running a 60,000 concurrent load test, even though we have terraform scripts that can probably do it in 20 minutes, you're still talking about spinning up 10 to 15 AWS boxes. It's not a cheap thing to go do or to run for any length of time. Those are the things that we tend to focus on. Where sort of our ability to organize and bring resources can help out as a company, and then the community does a lot of help for us in other areas, and it's amazing to see. It's amazing to see those core commit, we call them core committers, in our channels interacting with us. They even vote on things like MVPs.

So we're transparent or we try to be. We have our community server, and the reality is, with the exception of things like security and some few customers, we pretty do everything in the public. We have our community Mattermost server and everything's kind of done there in public. Even the enterprise features. I guess the only things we don't discuss publicly would be security issues or specific customer issues or specific customers.

[00:40:15] JM: So security-wise, is that to say that there are potential security issues that you might know about the you don't want to tell the community or you patch these privately? What do you mean by that? Why can't you discuss security in the open?

[00:40:29] CH: Exactly. It's like specific issues that we either want to fix or that we want to fix or specific issues in libraries where we want to make sure we're up-to-date in those libraries. We'll invite sort of core committers or core community into some of those channels, the ones who have the expertise to help out. But we don't want the larger sort of user base to – Just like any sort of security-related incident to understand it and be a threat.

The other interesting thing is we got that kind of support too. When we're going through a lot of our spelling challenges early on in the early days, our community really rallied around us and helped us solve some of those issues. We would provide the logs and the resources to run the load test and they would come in and like, "Oh! The problem is over here. Did you guys look at this?" For them, it's really fun. You get these fun really hard challenges to work on that you probably wouldn't normally do in your off- time because of the resources required to do something. That's a cool way to see community involved as well.

[SPONSOR MESSAGE]

[00:41:36] JM: Logi Analytics is an embedded business intelligence tool. It allows you to make dashboards and reports embedded in your application. Create, deploy and constantly improve your analytic applications that engage users and drive revenue.

You focus on building at the best applications for your users while Logi gets you there faster and keeps you competitive. Logi Analytics is used by over 1,800 teams, including Verizon, Cisco, GoDaddy and J.P. Morgan Chase. Check it out by going to logianalytics.com/datascience. That's logianalytics.com/datascience.

Logi can be used to maintain your brand while keeping a consistent familiar and branded user interface so that your users don’t feel like they’re out of place. It’s an embedded analytics tool. You can extend your application with advanced APIs. You can create custom experiences for all your users and you can deliver a platform that’s tailored to meet specific customer needs and you could do all that with Logi Analytics.

Logianalytics.com/datascience to find out more, and thank you to Logi Analytics.

[INTERVIEW CONTINUED]

[00:43:03] JM: Open-source is something that continues to amaze me. It's almost unbelievable that you still just see so much of the best software in the world be written by people who don't have really like a skin in the game to contribute to something, but they contribute to it nonetheless. Yeah, I don’t know. It's almost unbelievable.

[00:43:26] CH: I mean, I think open-source eats the world. I think it's such an amazing experience to be part of it. If people are not part of the community, I want to encourage them. Mattermost is open source. Of course, I want to encourage you to go there. There's all of these other great open-source communities that it’s just amazing to be a part of and see the power of those communities and what they can do. For us, it’s amazing.

[00:43:47] JM: The motivation for an open source contributor, is it often times that community? Like they're looking for people to interact with or maybe they work on piece of – They like a piece of software and they’re like, “Well, I’ll just get involved contributing to it,” because people who like this software are probably somewhat similar to me, and in that allows them to find the community and then they can contribute to it to make the community stronger. Is that the feedback loop?

[00:44:11] CH: That's part of it. I think there’s kind of three – I talk about it in kind of three areas with three sources. The first third is just people who want to learn, whatever it is, “Oh, I really want to learn Go.” “Great! Go look at our project.” “I really want to learn React.” “Great. Mattermost is here.” So we get a lot of that where people like, “I really came to you because I want to see a really large complex Go project,” or React project. We get a lot of that learning.

Really high-caliber people but who may not be high-caliber in a specific technology. We get a lot of people – Like a third of our people are people like that who are just – In their spare time, they want to learn and contribute in kind to give back to a community or be part of a community.

The second third is probably your typical, like what you think of – Like what you described. Just your open-source kind of people. They're really there for that community. They'd going in – We tend to see those people who go across multiple projects or who come in at different times and lave. They come in, they're really interested in a very specific aspect of your product or application, or open-source community. They want to work on that piece, because it's really interesting to them and they want to contribute back and then they kind of go away, or sometimes they stay as well. So that's this second third.

The last third is this interesting sort of community, where they're more partners. Half the time we don't even know until we start reaching out to them. I'd say another third for us at least is you'd almost call them partners. They're customers or users of Mattermost who are sponsored. You could think of them as workers of our core community who are sponsored by corporations in some respects. Because we'll have some people in our community who've been amazing, they've been here for – Working doing amazing work for three or four months. One of my goals is I have this thing called community buddy where I try to meet a new community member every week and I'll reach out to those people and I'll start talking them. They'd be like, “Corey, we're a customer. We've been using you for two years. I talked to over here.” I'm like, “Oh! Oh!” You get those customers who are just sort of sponsoring development on their features, and that's the powerful, because we see customers really taking advantage of that.

A great example is like you have – We these all the time. We have two customers who you could never ever get in the same room for whatever reason. They're competitive. They just hate each other. Who knows what? In that sense, open source kind of asks likes like Switzerland, like this neutral ground, where you have one customer whose contributing stuff on one side and they're doing it in the sense of just good karma kind of going back into system, because they know there's another customer on the other side doing the same thing.

I think that's one of the cooler aspects of open source, is just this ability to bridge the gap and get all of these different sort of people to work together and you end up with this sort of amazing product in the end, open source project in the end. That's really cool to see. We generally see about a third, a third, a third. Ebbs and flows really fluctuates depending on the time of the month or who you're talking to or what's going.

A great example, like Hacktoberfest is always fun for us in October, because we probably get 90+ percent of our contributions just come from like random one-time community members, which is totally awesome to see. Part of our goal is to try to get those community members the stick around obviously. It's really fun to see. So it really depends on the month or what's going on. But it's fun to manage sometimes.

[00:47:30] JM: So those kinds of security challenges that you mentioned, the things where you may have a security issue that is not completely solved yet and so you don't want to disclose this to the open source community. Actually, it makes me think about like there's actually a lot of coverage over Mattermost that you would have to attend to. It's like a big security surface. Can you tell me about some of the security challenges? Maybe some of the ones that you've resolved at least, security challenges in building Mattermost?

[00:48:00] CH: Yeah. No. That's a great question. I think to answer that, it's kind of to take a step back and think about what open source does. Having open-source, and this is my sort of philosophy variably. Having open source at its core is one of those things that challenges you to write products in a secure way. Because whether it's conscious or subconscious, as a developer who's releasing his code publicly like, you're going to spend a little bit of extra time looking that over and you're going to spend a little bit of extra time making it more secure, because you know there are people and there are tons of other eyeballs on the other end looking at that source code. I think that's one aspect of it that's really interesting being open source where you get people just wanting to do their best work, doing a better job.

We obviously have a very security-focused nature here. We have a security. We actually have a security meta team that pulls in people randomly to look at specific issues, but then you can kind of extrapolate that one step further before we kind of get to – I'm trying to think of some specific security context. But the interesting one to think about is you have these different customers who have their own infosec team that need to sign off on different forms of software. We see this time and time again where we have a really large savvy, sophisticated customer has their infusec team look through our [inaudible 00:49:16], look through our plug-ins and they're super happy at what they see

You get this amazing, super talented internal security team at a really tech savvy company who looks through your source code. We get multiple of those. As we get new customers or attract people. So we'll get two or three people doing this – Two or three high-caliber security teams doing the exact same audit. It's really fun being sort of the mediator passing the information in some respects between them.

I think in those two respect, that's where open source I think really wins in terms of security. I know a lot of people say, "But it exposes your tax service." I don't know. To me, that's really security through obscurity. I don't know if it really does. I think the benefit of having engineers and people with a security focused mindset looking over that code is way better than not, especially when you talk about these different engineering teams looking over your code and finding issues. That's amazing.

Specific security ones. I'm trying to think of ones that are – I can't think of any off the top of my head. I can go back through and we can look through our security reports and show them. I mean, luckily the vast majority of ours are things that are reported by very sort of high-caliber in-depth discovery. I think a lot of that community is very much wanting open-source to be successful win. So we get a lot of great contributions like that. Really, obscure things that you know in someone else's code base would've never been checked. We see that kind of time and time again. I shouldn't say time and time again, but we see that. It's always fun even as a team that works on this to go look at those things sometimes because you're like, "Ah! That's impossible. How could that – that's impossible to be exploited," right?

You actually go in and look at it and you're like, "Oh! Oh! Yeah, that's not such a super savvy way." My imagination immediately goes to that like, "Wow! This was found because we were open source. Imagine if you weren't. This would be an exploit that would live and never get corrected." Because it's that classic thing, and I know I've done it. In past lives, there's been a lot of code that I have checked in to a repository somewhere that I know is never going to see the life of day. It works and I'm done with it, and the quality of that code is in such stark contrast to the quality of code we usually end up seeing.

[00:51:42] JM: You get these enterprises, and they deploy Mattermost on-prem and they've got it going. Then let's say there's and update Mattermost to themselves?

[00:52:02] CH: Yeah. It kind of depends on what we're talking about, but there are certain things where you can push out and help. But for the most part, they would need to update themselves. We have a long-term supported version. We also support three versions back for security. Whenever an exploit is detected, we go patch that and we even give a lot of our customers and even big users, they don't have to customers, guidance or knowledge ahead of time, like, "Hey, this patch is coming. We really think you should apply it because we think it applies to you," stuff like that.

A large part of it is making sure those teams are in a place to be able to do – To roll out the update and do it in a secure way. Some of the customer obviously, because they're on an air gap network or they're in a very locked-down network. There a lot less of a security threat for them. We'll explain the situation to them and they'll be like, "Oh, that's really interesting. Yeah, we'll take it on the next rolling update." But for us, it's not a high priority issue because of the way our environment is set up, or because it effects this piece of the system that we don't use. We have that obviously. We have a lot of cool features too like TLS mutual auth so you can lock down your clients, things that really give you a lot of extra or added security on top.

[00:53:05] JM: I'd like to go into a little bit of engineering detail in terms of just the simple path that a message takes. So if I'm in front of Mattermost and I sent a message to a channel that maybe has 300 people in it, what happens? What's the process of like me sending the message? The message getting accepted by the server? Getting written to the database? Getting pushed out to the rest of the people? Can you walk me through the life of a message?

[00:53:34] CH: Yeah. Sure. We kind of leverage two different technology. I'll probably talk about this in the more interesting or complex scenario when we're talking about a clustered Mattermost server in a high-availability sever and what's going on there.

So we have sort of two kinds of connections where we receive data. One is through our web socket connection. That's our sort of real-time messaging pipe. Another is just sort of your standard HTTPS connection. So if it's something that's happening in real-time or being delivered in real time, then it almost always goes over the web socket. If it's something that's, then it’s goes over the HTTPS connection.

A great example of that is when you first start typing a message, you’ll get the user is typing. What that is, is that's actually a web socket message being sent up the pipe. It goes up the pipe. It hits a, let's say, like in this case an nginx proxy server. It may get geo-load balanced or load balanced in some way. So you go to a particular Mattermost app server. That app server then rebroadcasts that message using its [inaudible 00:54:32] protocol to the other nodes in the network. Then all nodes in the network basically re-broadcasts that message back down to the users who are in that channel. So you see Corey is typing.

This is one of those where I talk about inbound versus outbound, like sort of broadcast mentality. A great example of this is let’s say I’m in Town Square doing this at a company with 20,00, 30,000 people in this 10,000 people live on the server right now. That means when I push a user's typing message up to the server, it has to turn around in real-time, gossip that to the entire network and then push it or broadcast it all back down to those 10,000 connected users that, “Hey, Corey is typing.”

That’s kind of like the basic flow, and messages take a very similar flow. When a message gets posted, it actually goes up on an HTTP request. It goes sort of to that same flow. Before it actually gets saved into the database, we allow you a chance to hook the message. You can have a plug-in on the server side, a high-trust plugin on the server side that can hook the message. It could do several different things. It can rewrite the message. It can filter out the message or it can outright reject the message. That's kind of really interesting. We see lots of times, like I said, social security numbers or cuss words. We have certain customers who don’t want [inaudible 00:55:49] violations or whatever. So they filter out those messages before they even get in the database. They may even alert somebody, like, “Hey, this is happening.”

So then what happens is the message gets persisted into the database. For us it’s a SQL. That message will then – And the SQL database will then get sort of – It will get replicated to its slaves. Td then database, what happens is we give you another chance to hook the message. So here's an example where you just want to do some extra load in its own thread, but you don't want it to necessarily affect the performance of the system or you don't want out right reject the message. So this is a way for you to sort of peak at the messages going through to the system and kickoff some other backend flows.

Because maybe it’s not you want to outright reject the message, but you want kickoff a workflow that says, “Hey, I think Corey just pasted his password in here.” Do you want to go check on that?” or some private key or something like that and like, “We should go look into that.” So you get those kind of interesting use cases as well.

Once it’s kind of through those basic steps, then it kind of starts its broadcast outflow again, and that's the same thing that happens. The message then gets gossiped around to other machines in the cluster and then that message – Other machines in the cluster push that message down to other users via the web socket if they are active and part of that team and part of that channel. If they’re not part of that part or part of that channel, the message gets filter out for them and they never see it.

[00:57:11] JM: Well, that’s a ton of detail and I’m sure we could dive into a lot of different areas of that, but I know we’re kind of running up against time. So I want to ask you a little bit about the business. Can you give me a perspective on the size of the on-prem chat market and may described in more detail how you think about the competitive landscape. Why are there some enterprise that can't use Slack? Can they use Microsoft teams? How do you think about this market?

[00:57:38] CH: That’s a great question. I mean, we think about this a lot obviously. I think there's a lot of – There’s multifaceted answers to this, but I think the one for us that we keep coming back to is there is a larger segment of the audience and I people realize where they just want to own and controller their own data. Whether it’d be the European government or a European company who’s very centered around privacy. So that goes along a way.

Then you get into other industries that are regulated. Healthcare is a great example. Finance is another great example of that. They’re heavily regulated interested in industries. Not only they care about a security aspect of it. They care probably even more about a compliance. The classic one is finance. You talk finance. Insider, trader A cannot talk to trader B, because if they do, that's an insider trading violation and you need to know about that kind of stuff.

There's a lot of those kinds of scenarios where you get compliance type reporting and you want to be –They need to guarantee that they own that data and that they keep it for years and years and years and stuff like that. For us that's where we see obviously a lot of our business now. With that being said, we see a lot of people who just love Mattermost, the open-source nature of it, the ease of being able to run it on their own servers and who just goes spin up what we call a team edition server and just start running it for their own community or team, and we would that. It's one those amazing things that we just love to support, because, yeah, if you're running a small team and you want to keep your messages, you don't want to be deleted after a certain number, a certain number of time, then, yeah, great. Go set that up and run that.

For us, when talk about things like Slack and Microsoft teams, in respect to certain customers, those are threat, but there's whole other set of customers where it's not a threat. One more thing too is this huge push in industry. A lot of these customers just – It's just like what they're doing. They want to commoditize the cloud. There's this huge push to cloud native and commoditize in the cloud, and I think that's true with a lot of these vendors. They don't want this. I think the days of vendor lock-in and these bigger, savvier enterprise is getting locked into these vendors is really on top of their conscious right now.

If you could provide a service that's as scalable, as performant, it gives you all the features you need, and through things like Kubernetes or Terraform does a lot of the self-healing and management for you. Why not run it yourself?

[01:00:03] JM: Do you spend any time talking to the Slack team or is it pretty much kind of a very different companies and you're kind of fighting over the same space?

[01:00:14] CH: Yeah. I think it's very different companies, varying different philosophies. Like I said, there's a not a whole lot of intersect. I think there's a ton of intersect in our community, because just the open source nature of it. But when you talk about customers, for us, paying customers, there's not a whole ton of intersect. There is here and there but usually those customers are – They have certain features requests or price conscious or who knows what they are, and Mattermost is still a great fit for them but it's probably not the only fit, if that makes sense. I think [inaudible 01:00:43] great company, it’s a great product and I think it definitely has a need out there and I think we overlap with that to some extent, but I think when you look at the core of where we focus, it doesn't overlap that much.

[01:00:56] JM: Okay. Corey Hulen, thank you for coming on Software Engineering Daily. It’s been really fun talking to you.

[01:01:00] CH: Yeah, awesome. Hey, thanks for having me.

[END OF INTERVIEW]

[01:01:05] JM: This podcast is brought to you by wix.com. Build your website quickly with Wix. Wix code unites design features with advanced code capabilities, so you can build data-driven websites and professional web apps very quickly. You can store and manage unlimited data, you can create hundreds of dynamic pages, you can add repeating layouts, make custom forms, call external APIs and take full control of your sites functionality using Wix Code APIs and your own JavaScript. You don't need HTML or CSS.

With Wix codes, built-in database and IDE, you've got one click deployment that instantly updates all the content on your site and everything is SEO friendly. What about security and hosting and maintenance? Wix has you covered, so you can spend more time focusing on yourself and your clients.

If you're not a developer, it's not a problem. There's plenty that you can do without writing a lot of code, although of course if you are a developer, then you can do much more. You can explore all the resources on the Wix Code’s site to learn more about web development wherever you are in your developer career. You can discover video tutorials, articles, code snippets, API references and a lively forum where you can get advanced tips from Wix Code experts.

Check it out for yourself at wicks.com/sed. That's wix.com/sed. You can get 10% off your premium plan while developing a website quickly for the web. To get that 10% off the premium plan and support Software Engineering Daily, go to wix.com/sed and see what you can do with Wix Code today.

[END]