EPISODE 931

[INTRODUCTION]

[00:00:00] JM: Modern applications are distributed systems. These applications require an installation mechanism that can run and update the software across multiple nodes. When a SaaS company starts to work with large enterprise customers, that company needs to figure out a way to deliver their software product to the enterprise. This requires the SaaS company to deploy the product to whatever infrastructure the enterprise is running.

Some enterprises use on-prem infrastructure. Other enterprises use AWS. Some use a variety of cloud providers and on-premise servers. A SaaS company with limited resources must be able to have a standard deployment model that satisfies all of these different use cases.

Ev Kontsevoy is the CEO of Gravitational, a company that builds software for application delivery. Ev's company maintains Gravity, an open source tool for application imaging and delivery. Ev was also the founder of Mailgun, a popular API for sending email. Mailgun was acquired by Rackspace, and in his time running Mailgun both in and out of Rackspace, Ev became deeply aware of the problems faced by developers and operators who manage server infrastructure.

Ev joins the show to discuss his experience building companies and the state of modern infrastructure. Full disclosure; Ev's company, Gravitational, is a sponsor of Software Engineering Daily.

[SPONSOR MESSAGE]

[00:01:39] JM: This podcast is brought to you by PagerDuty. You've probably heard of PagerDuty. Teams trust PagerDuty to help them deliver high-quality digital experiences to their customers. With PagerDuty, teams spend less time reacting to incidents and more time building software. Over 12,000 businesses rely on PagerDuty to identify issues and opportunities in real-time and bring together the right people to fix problems faster and prevent those problems from happening again.

[INTERVIEW]

[00:03:06] JM: Ev Kontsevoy, welcome to Software Engineering Daily.

[00:03:08] EK: Well, thanks for having me. I'm glad to be here.

[00:03:11] JM: You were the founder of Mailgun, and I think that's a tool that many, many developers know. What was it like building an API service back in 2010?

[00:03:24] EK: Before Twilio.

[00:03:26] JM: Before Twilio.

[00:03:26] EK: Yeah, Twilio is very important. Before Twilio, if you tried to explain that I have an API that I'm selling, a lot of people would think you're crazy. They will say, "No. No. No. First, you have to build a product, and then you have an API." Also, I wasn't the only founder. There were three other people involved. Yeah, that's true.

Interesting, if you're asking me how Mailgun got started, there's an interesting story about it. It was year 2008, like in the middle of financial collapse, like the whole world was burning. I mean, the financial part of it anyway. That's actually when Jeff Bezos, he was in this like go-to events,

and he went to startup school at Stanford here in the Bay Area and he was trying to convince people to use AWS. It was a brand-new thing.

I think S3 was probably launched earlier, but it was around the time when EC2 happened and then Amazon Web Services. He was on stage giving his famous presentation on computing his utility in front of various skeptical developers. One of them asked him a question, because there was a Q&A section, "How come it's not possible to send or receive an email out of AWS?"

It confused Besoz. He was kind of looking around, like there was a VP engineering maybe standing behind him. It's on YouTube now. There's this video. That's the very same second that Mailgun was born, because I already built – It was kind of a side project of mine, like a using thing using Python at the time that you could programmatically instantiate email domains and you could create inboxes, connect to them using IMAP. It was like full virtual, kind of API-driven system. You could build Gmail UI on top if you wanted. Yeah, that's how Mailgun got started. Got into Y Combinator in 2011 and then sold to Rackspace two years later. The rest is history.

[00:05:11] JM: What was hard about building an API business in 2010, 2011?

[00:05:17] EK: You see, if you love something, I don't think it's hard. Building things is actually incredibly enjoyable, especially if you don't have any customers, you don't have any business, you just like build and you play with it and then you feel good, and it was a lot of fun to be honest. It was also kind of sexy. Infrastructure is always sexy to certain type of a developer.

What sucked was actually getting into production. So a lot of people don't realize that Mailgun and our competitors, like SendGrid, these guys were doing really well as well. There is kind of two side to this business. On one hand, you do have customers. There are APIs you're providing. I don't know, control panel, building and whatever. All of that is standard stuff. But then there is this terrible part. It's all the crime that happens in the internet.

Email is extremely dirty business. There is phishing, there are viruses, people use stolen credit cards to open accounts, then they use those accounts to steal more credit cards. So dealing with bad actors was incredibly stressful, and it's completely invisible. You're actually spending a lot of your engineering time and money on fighting crime, and it's hard, because you basically

have to build a spam engine that works on outbound email. You're checking email for spam before you send it out, because you don't want your customers to be spamming everyone. It's kind of like black art that technology, and no one even appreciates it, because people don't see it.

I would see sometimes someone online said, "Hey, Mailgun hasn't changed in two years. What are these guys been doing?" They're like, "Oh my God!" I saved you from like massive spam attacks. That's what I've been doing. Meanwhile, the price of email delivery was just like dropping and dropping and dropping, because Amazon launched simple email service and there were lots of copycat competitors. So that was tough, basically dealing with this kind of dark side of email.

Also scaling was incredibly challenging too. Email is very heavy in terms of traffic and computational part that you need to have in order to deal with comparing to like a typical web app. Mailgun will have like 10X traffic in terms of bandwidth compared to usual web applications. Yeah, we've had to learn how to scale very early.

[00:07:29] JM: You were not originally an API company, right? You didn't yourself as a "API" company. We didn't even have the words to describe that back then. You were a mail client builder backend system or something, like mail infrastructure.

[00:07:46] EK: So API maybe is not the word how we used to call it. People started asking to give them a REST API to send email, and we understand why if, for example, using JavaScript. It's very natural to create like a JSON payload and convert it into email and send it out. But email has its own open standards. There is SMTP for sending, right? SMTP is actually not that terrible, and there are MIND format for messages itself, which if you look on how looks on the wire, if you read [inaudible 00:08:16], it looks incredibly similar to HTTP. They're also like same headers. This MIND format is pretty universal. So you don't need an API to send an email or to receive an email. For receiving you could use IMAP and POP.

So the first version of Mailgun only used open standards, and I've always been a huge believer in open standards in general. So I felt, "Hey, you probably don't want to have a vendor lock-in to your email engine." Every programming language has a library for sending email. Just use that.

But people kept asking for REST APIs. Eventually, yeah, we had to add that. Then there were more and more and more. Yeah, it evolved. But originally, it was very simple email system that supported all email protocols, including SMTP for sending, POP3 and IMAP for receiving.

[00:09:10] JM: When you were acquired by Rackspace and you moved over to their infrastructure, what did you learn about a large scale migration like that?

[00:09:22] EK: How do you know we had to do large scale migration? You can guess.

[00:09:26] JM: I would assume so.

[00:09:28] EK: Okay. Interesting. When Mailgun started –

[00:09:30] JM: I guess you could have technically just said, “Hey, Rackspace. How do we scale this on Amazon?”

[00:09:35] EK: You have to realize too that Amazon back in the day couldn’t do email. Remember, that was the question that developer was asking.

[00:09:43] JM: Oh, the answer was no?

[00:09:44] EK: Yeah, because they were blocking all traffic. So Mailgun was running on software layer, which at the time was one of the most modern bare metal platforms you could run on. Then the reason we picked SoftLayer is because I looked at all these like awesome startups that were growing, like exploding in popularity. I just like look up there [inaudible 00:10:03] they were serving their endpoints from. I think Dropbox was one of them, and that’s how I saw that, “Hmm, a lot of really smart people use SoftLayer.”

So we used SoftLayer for Mailgun and we were – So we picked their Dallas data center for latency reasons so you get kind of same response time to both coasts. When Rackspace acquired us, of course, yeah, we had to migrate to Rackspace. Now we’re kind of slowly shifting towards like Gravitational, my second company, because I started seeing these pains [inaudible 00:10:31] with that massive migration, because Rackspace said, “Hey, you need to take this

whole thing and migrate it from SoftLayer to Rackspace," and it was painful even though we were relatively small company.

I also remember around the same time, I went to – I think it was PyCon and I met some Instagram people and they also – Migrating for them was apparently painful too. I think before Facebook acquisition, they were running somewhere else. So they also were kind of saying that, "Yeah, recreating environment from scratch. It was just extremely painful if you want to change providers."

Yeah. So migration, it's something I'm fascinated, and this whole area of dev ops, it's very disappointing for me that we ended up in this state. Because when I was starting as a developer, it used to be simple. You build your app and it would be like an executable, originally for Windows, and then for Linux. You would give it to people and they would just run it. There was no need for ops, because it was before clouds. Desktop software is kind of fun this way. If you do a good job, it will just run everywhere.

[00:11:36] JM: Yeah, installation wizard.

[00:11:37] EK: Yeah, like millions of users if you're doing really well, and there's no need for ops. You just like focus on building features and whatever. But this cloud stuff and the requirement to have like 80% of your team focus on what we call dev ops now, which is basically just glorified system administration. That's just extremely annoying, and that's really what I saw at Rackspace over and over again. We would have these customers that come to us and then we would try to sell them more of a cloud, better cloud, cloud features and they would say, "You know what? We're sick of infrastructure. Virtual cloud bare metal doesn't really matter."

Infrastructure is not really the biggest pain point. The pain point is just keeping applications alive. That's what they've kind of kept asking for us to figure out how to do, especially if they wanted to have exact same codebase. Exact same app running at the same time, like on Rackspace, on AWS, or maybe on Azure, that is extremely hard to do, because sometimes you would have to have like three different ops teams with knowledge that is specific to each platform you want to run your application on. That's really kind of what eventually forced us to

like this idea, like what if we could figure out a way to run cloud apps with the same simplicity, like drag and drop simplicity that desktop applications are running?

[00:13:00] JM: I think Rackspace was using OpenStack back then, right? Wasn't that somewhere around the high-point of OpenStack, or was OpenStack later?

[00:13:06] EK: It was one of the reasons it was sold to Rackspace, yeah. Because again, as I said, I started my career when Microsoft was dominating. I didn't want AWS to dominate, and Rackspace had this very compelling message to use open source, which was OpenStack at the time to basically allow everyone to have their own cloud and their own servers.

[00:13:25] JM: The pitch for OpenStack sounded a lot like the pitch for Kubernetes, right?

[00:13:28] EK: There are lots of similarities between two projects, and I am of an opinion that Kubernetes actually has potential to completely replace virtualization in general. Unless you need multitenant security, which is probably last. But even then, you could have containers that run in lightweight VMs. Yeah, running Kubernetes on top of bare metal is probably how I would build like a data center from scratch.

Yeah, there are plenty of similarities. Two big differences I think between Kubernetes and OpenStack are like the technology. Kubernetes benefited from being born within a single company. So you could see how things fit together. It has much more cohesive design. Whereas OpenStack was a little bit of kind of piece by piece design by community, because the storage component was built separately from compute, and then there was this network control plane that was designed by someone else. So there was a little bit of that.

The second difference is how these two projects are governed. I think Kubernetes has a much tighter organization that kind of manages – I'm talking about CNCF and how the Kubernetes management happens, like the project control, because OpenStack was just, again, suffered from massive fragmentation. There are all these different OpenStack distributions that were hardly compatible. So it was really hard for you to have like an OpenStack deployment target, because different OpenStacks would be different from each other.

[00:14:55] JM: The Kubernetes governance structure is actually kind of beautiful, because when Kubernetes came out, part of what made it so exciting is like this is the distributed systems equivalent to what Linux was to Windows.

[00:15:07] EK: Absolutely.

[00:15:08] JM: Not only that. Rather than being maintained by a group of scattered individuals, like Linux was, which worked out like quite well. But I think that could have been kind of a once in a lifetime thing. But Kubernetes is backed by Google, which to some people is like kind of scary and now it's kind of centralized, like centralized corporate style Linux. But it's more like centralized corporate style Linux plus the decentralization factor – It's very interesting to watch. It's a war of the powers but in a way that's so good for developers.

[00:15:49] EK: Very much so. I'm not super knowledgeable about kind of innerworkings on Kubernetes government, for example. But looking mostly from the outside, I think any project, just like with software engineering in general can always benefit from like the one central source of like ideas. You have to have like a vision, and there needs to be a person, or maybe a small team that basically maintain that vision and evolve this vision. That's basically what Torvalds is doing for Linux.

Then once you have this vision part laid out, and you can allow individual contributors to compete for best possible implementation of parts, of pieces of that vision, and that's the thing with Kubernetes. I think it has an extremely clean and obvious vision where it's going.

Yeah, then you have all these competition for like individual features like these special interest groups. The best implementation will eventually win. So Kubernetes now has this evolving standards for how do you define an application, because kind of funny thing about Kubernetes, originally it came out, there was no way to describe an application. It was a cluster and then a bunch of low-level components. Then Helm came around and all these other things.

Yeah, as long as it's compatible with just the overall vision for the project, then I think eventually like best implementations win. OpenStack, they kind of didn't have that. There was just a lot of kind of random just going on. The fact, for example, that Kubernetes has this kind of compliance

test that you could – To become a certified Kubernetes distribution, you have to make sure all your APIs are working properly. That's just, again, manifestation that they do have a clear idea where the project is going.

[00:17:34] JM: Do you think the service mesh stuff was kind of the Istio launching with like a lot of hype and the developer community kind of striking back against that. Do you think that was at all a referendum on the governance structure?

[00:17:50] EK: I really can't say. All these open source communities, sometimes they just –

[00:17:54] JM: They get angry.

[00:17:56] EK: There's just like drama and interesting dynamic going on.

[00:17:59] JM: Which benefits those media companies. That's why I asked about it.

[00:18:03] EK: Yeah, exactly. So it's also kind of interesting what's happening inside the cluster is maybe less interesting for me against this broader vision of what Kubernetes is. You compared it to Linux a little bit earlier, and I love this comparison, because back in the day we'd look at a server and it's a thing in itself. It's really your universe. You would deploy a software into a server. The first job I had for an internet company, like our production environment was just one really expensive [inaudible 00:18:30] machine. There was a second one that's like backup. But the idea was we just copy files into a server. That's your deployment. It was like in RSS search engine.

But these days, software is so big and it has so many dependencies and microservices that need so much data. It doesn't fit into a server. It's a stupid idea. So you need the actual cluster of machines. So now this cluster is basically a new computer. That's really what you're deploying into. So the old computer was just an individual machine, and now it's basically a data center. If you have new definition of a computer, you need new type of an operating system, and this is what Kubernetes is. It's an operating system for a data center.

So we basically went from mainframes to a bunch of small boxes, and then we're going back to like mainframe era, because if you zoom out, that's what data center is. You could think about it as one giant mainframe that needs an operating system.

[00:19:20] JM: Do we have a clear picture of how we should be doing these deployments? For example, I think about if I'm deploying Kafka. Do I deploy that Kafka to the same Kubernetes cluster that I've got some random microservice API running on, or do I spin up an entirely new Kubernetes cluster devoted only to Kafka?

[00:19:44] EK: I would say the answer to this question depends when you're asking this question. I think to properly compare, like being compatible with Kubernetes vision, it needs to be deployed into the same cluster. Because, again, if it's an operating system, an operation system should be capable to run all of your applications just absolutely fine. So you might have – I don't know, like a PostgreS database in a machine and the chat app running together should be totally fine. Why wouldn't that apply to an environment?

However, when things are just born and they develop overtime, Kubernetes originally lacked a lot of capabilities and features to make it possible. Again, just Linux back in the day, like the kernel wasn't – There was like very limited support for threads, for example. Then it evolved, and now Linux is the most capable multiprocessing operating system that we have, I think.

Similarly with Kubernetes, just as it's evolving, it's gaining more and more capabilities to run mixed workloads, because that's really what you're pointing at. Same thing on application side, when Kafka was built, Kubernetes wasn't around. If someone is creating like brand-new Kafka or something to replace Kafka, well now you understand that it probably should be deployed into Kubernetes. Maybe you should be building it in a different way.

But because everything is changing and we're talking about this like we're in the middle of this transition, then I guess it probably depends on the use case. If you're comfortable running Kafka on the same Kubernetes cluster, it doesn't affect performance of all the things. Yeah, sure. Go and do it. I don't have a strong opinion. I think it always depends. That's always the answer that you probably want to hear.

[SPONSOR MESSAGE]

[00:21:29] JM: As a programmer, you thinking object. With MongoDB, so does your database. MongoDB is the most popular document-based database built for modern application developers and the cloud era. Millions of developers use MongoDB to power the world's most innovative products and services; from cryptocurrency, to online gaming, IoT and more.

Try MongoDB 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]

[00:22:24] JM: Your experience at Rackspace, the disappointment in how ops turned into such a large part of what developers are doing. How did that shape what you're working on right now?

[00:22:39] EK: Well, first of all, it had nothing to do with Rackspace. We all have dev ops teams. This is just general state of cloud computing if you will. So you have 20% of your engineering team, again, working on the application. Then you have the rest of your team basically keeping that application alive, which feels kind of like a waste of time and effort.

Why couldn't we – The model, the operational model of cloud software I would like basically ends on GitHub or GitLab. I want to push my code into my git repository and just go home, just go sleep. I want the robot to build my software, run all my tests and then create a deployable artifact and push the – Stick that artifact into a data center somewhere and it will just run. I wouldn't want to use monitoring, alerting. I wouldn't want to be scaling anything. I just want this whole happen automatically for me. Kubernetes is a step forward towards that future.

Because if you think about what is like your ops team is doing, they're constantly compensating for the fact that the infrastructure, your cloud computer is just really dumb. It just cannot really do some things automatically.

I remember back at Mailgun, we would always struggle to predict what percentage of our compute needs to be allocated to API servers versus background machines. It would be hard to figure out what kind of capacity we need to run MongoDB reliably. You'll only figure out these things as you go. It's like, "Oh! My latency is increasing, or I'm getting connections dropped. We should get more hardware," like for this particular piece of the application.

You're basically doing manual scheduling, and that is theoretically what Kubernetes solves for you. So if you have your app and your app has like a certain performance envelope and starts running and it starts increasing, and the envelope keeps being stretched because you get more traffic. Well, Kubernetes should just allocate resources appropriately, right? If something is failing, it will restart it for you. That's the whole idea.

That is why it's such a critical first step towards this future in which we shouldn't be doing dev ops. I'm kind of waiting for that to happen. That's why we started Gravitational too, is because a lot of people they see Kubernetes has one or two things. So some believe that Kubernetes increases developer productivity. I would say it's debatable, because my productivity is insanely awesome if I stick my code to GitHub and I go home. I don't even need to know that Kubernetes is there. That's really how developers I think should be thinking about their own productivity.

The second component is actually legit, is that if you look at your AWS spending, Kubernetes can shrink that dramatically, because it can automatically manage hardware utilization for you. In other words, using Kubernetes in a good way could basically just dial in. I want my infrastructure to be always busy, always utilize, I don't know, like 60%, 80%. Kubernetes will keep it so.

So you can cancel the rest of the instances that you're not using, which automatically means that you shouldn't stop worrying about what type of instances to use. Just get the fastest, the most awesomeness servers you could get and let Kubernetes slice and dice that compute into partitions that are best suited to your application. Stop tinkering around trying to figure out which

instance type to use. Just use the most capable hardware. That is I think much more interesting way to look at Kubernetes value.

But then there is another, a third reason to use Kubernetes, and that's what my new company is about, Gravitational. If you make applications just run on Kubernetes and ignore the underlying cloud, so then you have a potential to make your applications run on a new cloud you want as long as there is Kubernetes there.

The problem though is what people are doing today with Kubernetes clusters, and they turn them into these special snowflakes. So you could have one Kubernetes cluster that's configured to do authentication one way. Another Kubernetes clusters doing something differently. So we're kind of going back into this let's do dev ops by hand, and there are all these providers and Kubernetes distros that would sell you like a control plane, or we call these different parameters.

But we thought would be sexy, what if you could have a tool, like this magical tool that you can point at your Kubernetes cluster and it will just snapshot it and save it into a file? We'll call it cluster image. Then we used to have VM images so you could capture state of a server. Then we had like container images so you could capture state of a Linux process or a microservice if you will. Continuing that very same idea, what if we could capture state of a cluster? Then you could use that image to have exact replica running somewhere else.

That's really what our project is doing that we're working on. It's called Gravity, and it's open source. You have to start somewhere. Right now it's only supporting Vanilla upstream Kubernetes, but the idea here, if you could package your application this way.

Going back to this original vision, like I'm a developer, I push my code into GitHub or GitLab. I want GitLab better actually lately.

[00:27:48] JM: Really?

[00:27:48] EK: Yeah, because it's open source. Because it also allows you to run anywhere you want.

[00:27:52] JM: Let's save that conversation a little bit. I want to hear more about that.

[00:27:56] EK: I just really also like how the company runs. Anyway, switching back to our original – Yeah, imagine you stick your code into –

[00:28:02] JM: Well, let me ask you a question. I've gone to like four or five KubeCons, maybe six. I'm definitely going to San Diego. Are you going there?

[00:28:09] EK: Yes.

[00:28:10] JM: Okay. Cool. I love walking on the Expo Hall. It's such a window into the opportunities that people are seeing in this space, which is a wide open market with lots of differing opinions, which is really interesting. You walk around and you see all these different providers, like you said. They're selling you different flavors of Kubernetes, different management planes, different ways to install Kafka and install this and install that, run, whatever.

You take a step back and you think, "Is it possible this might be all madness? Is it possible that we are in a certain specific time when it looks appetizing to do all these crazy work, to do our year-long Kubernetes migration. Is it possible that that is all madness?" We're going to look back in a year and say, "Well, I wish we just waited a little bit longer and waited for a better way to deploy our applications."

[00:29:10] EK: Oh, I think this is how our industry works. You might be disappointed or you might embrace it, but usually when something new comes out, there's this explosion of randomness and craziness, and eventually this technology becomes boring and the hype moves on and there's this kind of residual that's left behind. So that's innovation. That's really what – So the best ideas that come out of this kind of explosion of new things, those that survive and stick around, those are useful bits that we will take with us, and that's how I think the progress is just moving forward.

It's kind of fascinating to watch that in the end, we will be in a better place. So, yeah, there will be less need for ops. I have no doubt in my mind. But, yes, we have to endure this noise. You

have to understand too why this is happening. Imagine you have a company that does – I don't know, something like scanning for vulnerabilities. You just look at someone's code and you find if there're any vulnerabilities or someone is importing a library that's been hacked or something.

So then Kubernetes comes out. Now you need to rebrand everything you're doing with Kubernetes in it. Why? Because then you could be at KubeCon. You'll have something relevant. Even though the core value you're delivering has nothing to do with Kubernetes and it's perfectly applicable to Kubernetes workloads anyway, you have to rebrand it as Kubernetes, because then it's new and sexy, which means that if a company has a budget to migrate to Kubernetes, they will start looking for everything for Kubernetes. They would just type in a browser like, "I want automatic code scans for vulnerabilities." They will always add for Kubernetes, because that's the kind of budget they have.

That's really how this industry works. Yeah, it's unfortunate, but I'm kind of used to it by now. It's almost fun. You could clearly see that, like a lot of these things. They're clearly useful. But interestingly, they're clearly useful outside of Kubernetes too.

[00:31:05] JM: Putting yourself in the shoes of a bank, what would your "Kubernetes strategy" be today?

[00:31:12] EK: First of all, congratulations. You just promoted me to like a really crappy bank CTO. First of all, I have no idea about how banks are running their compute. What if they're using mainframes? Some of them actually do. Someone buys mainframes. Those are probably banks. But being in charge of technology in general, I'm not really sure where this bank thing is coming from. Banks are no different from others. They probably need to – Everyone needs to be as secure as a bank.

[00:31:41] JM: I like the bank example, because banks have the entire archeological dig of past technologies somewhere in their stack.

[00:31:48] EK: Oh, that's so true. Yeah.

[00:31:50] JM: It's like – So whenever a new technology comes out, they're always thinking about how to leverage. I mean, because they have the money to leverage it. But they have all the past things. I don't know. I don't know how to answer the question either.

[00:32:04] EK: I suspect that most organizations like this, they probably have like – I like your analogy. So they have every archeological layer of technology preserved.

[00:32:12] JM: The whole strata.

[00:32:13] EK: In an organization forever. I was talking to – Actually, I do know someone who works at a bank, and he told me that some of their older systems, he said, "We lost the ability to reboot machines." What do you mean? If you press the button, it will reboot. It's a hardware button. He laughed and he said, "No. What I mean by that is we're not sure if we reboot it, if it's going to come back." That's like the whole software survive reboot. I'm not sure if he was completely serious, but he wasn't smiling.

But then there's kind of next step from there, is like we lost the ability to upgrade Linux kernels. So we have some software that just has dependencies on like Kernel, like 2.X series and we can't patch vulnerabilities anymore. So that's the kind of stuff that banks have to deal with. Kubernetes is not going to help them with that.

But if you're just – I don't know, like running engineering or software development for any sizeable organization, I think it is wise to generally have a strategy to being able to run your applications on like any cloud provider anywhere. Any form of compute you can get your hands on, you should be able to just do it without incurring massive operational overhead.

[00:33:27] JM: You mean like if you're –

[00:33:28] EK: Just application portabilities. Yeah, we have this app, we can run it everywhere. Kind of like GitLab. You can run GitLab on Amazon. You can put it on Raspberry Pi probably. There is something about it. Just being able to do these things, and that's something Kubernetes I think can definitely help. So if you make a Kubernetes application, yeah, put it on a Kubernetes cluster.

[00:33:47] JM: GitLab is kind of interesting to the same extent that Kafka is interesting, because you have all these different things to deploy. You need memory. You need disk and you need it to scale. GitLab is kind of like it's the perfect example of a very hard to run application. It has all the different components that you – It's kind of the perfect stress test for an application delivery mechanism. So like there are people who are working on various kinds of ways to deploy complex applications on top of Kubernetes, and GitLab, it tends to be the example that they use.

So if I want to just deploy GitLab on raw Kubernetes, I mean, does the operator pattern do it for me? Does a Helm chart do it for me? In term of the open source mechanisms, what's the best model organism for how to deploy GitLab?

[00:34:40] EK: Actually, it's for GitLab specifically. Maybe I'm wrong, but I think I saw that it comes with a Helm chart. So they kind of pick the answer for you. I might be wrong, but I just for whatever reason clearly remember that is possible.

Look. I actually never really – I guess I understand value of things like Helm, but just like basic Kubernetes object [inaudible 00:35:00] low-level deployments, like things like pods and individual containers. It never felt to me like particularly hard problem. Yeah, these are not hard tools to use. But I guess people just need higher level abstractions. Yeah, we support them at Gravitational as well. So if you have a Helm chart, we can make an image of your Kubernetes cluster with everything that your Helm chart contains that will go into that image so you could deploy it somewhere else.

[00:35:31] JM: So if I wanted to deploy GitLab using your delivery mechanism, why would that be more useful than using Helm?

[00:35:41] EK: Oh, because it's a completely different operational model. If you're using Helm, you have to have a cluster set up and configured so it runs and then you can stick your application into it. Your application might be too big for this cluster. For example, imagine you have some code that requires GPUs to be present or something, or you have like certain

requirements for disk latency or something and then your deployment will fail. Sure, you'll stick the code in. It will not run.

The way Gravity works is that you feed it to Helm chart. It will go fetch all of containers and it will look into each container's image and it will break it into layers. It will de-duplicate layers and it will start packing all these layers into this image, and it won't stop there, and it will start packing Kubernetes itself into that image.

So the image will become like a fully independent. I like calling it like deployable artifact. It's the thing that doesn't have any dependencies. You could put it into a USB stick. You walk into a data center full of raw servers. Stick it into like a closest like Dell machine, and the full cluster will be recreated, including Kubernetes, and it will run in this kind of read only mode as an appliance. So it doesn't need management.

That's really the idea behind Gravity, is that you package Kubernetes along with the application and everything it needs into this just one file. So you don't have to worry about container registries. You don't have to worry about setting up Kubernetes. You don't have to worry about setting up Kubernetes security and compliance. You don't have to worry about synchronizing SSH access and Kubernetes access.

So this image contains everything. That's really the simplicity. Again, we just fundamentally don't like the idea that software needs to be managed. Gravity allows you to have this Kubernetes appliances that are self-running. So now we're one stop closer to this vision where development's job ends with git push. Everything else is just irrelevant. You just get this image at the end of your CICD pipelines. The robot effectively will produce the image and you could stick it into Amazon and it will run there. You don't need to have a preexisting Kubernetes cluster and you don't need to manage this cluster.

Obviously, this model doesn't really work for 100% use cases. The most popular use case today, that's really how we make most of our money is that companies would use this tool to create these images that contain their SaaS offering. Then other companies will just download these images and have full replicas running inside of their own data center or inside of their own cloud accounts. An example of something you could do only with Gravity today.

One of our customers get this request from their customers like, “Hey, we want your giant SaaS thing deployed into our AWS account. But we’re not going to give you access to it.” How do you do that? That is really how you do it. You basically just make this image. Give this image to your customer and they’ll just run it and it will get up and running. They will not even realize that Kubernetes is involved. That’s really the magic. Just like your desktop software. You might not know what kind of like C++ library a desktop software is using. It’s irrelevant. So that’s really kind of what we do with Kubernetes. We’ll package it into this image and make it completely invisible to the cluster user.

[00:39:00] JM: So if I’m using a compression format, the computer on which I am uncompressing it needs to know how to uncompress that file format. Similarly, with your application delivery system, whatever computer, whatever Amazon cluster you’re deploying it to, that cluster needs to know how to unpackage this thing.

[00:39:27] EK: Oh, it’s a TAR file. So we don’t really have any proprietary data formats or APIs. So it’s a simple TAR file. If you unpack it, inside you will find a directory, for example, with all the binaries that Kubernetes itself will be packed there and it will find all of Docker images. They’re already compressed, by the way. So we don’t really need to compress and decompress anything.

[00:39:48] JM: I was just drawing an analogy. I didn’t mean –

[00:39:49] EK: No. It’s a good analogy. I understand really what you’re asking. Because it’s TAR file, so you move it to this machine and you unpack it and then you see like install command. You execute the install command and it will give you an instruction. It will say, “Hey, here’s a command. Go copy+paste this command on other machines.” As you do, these machines will form a cluster. So it will just automatically create Kubernetes for you. It will put the application inside and it will do a bunch of other things. It will do things like hardware validation to make sure that the hardware you’re using or the cloud instances you’re using, whatever, that they actually have the capacity to run this application.

Again, we like this analogy to desktop software. If you ever bought like a game, like back in the day, like software in the box and you look on the side, there would be like system requirements and it would say like, “Oh! This app requires like 8 gigs of RAM and a CD-ROM or whatever.” Gravity allows you to define similar requirements for your cluster. It will say that it needs to have XFS file system and a Linux kernel not older than 4.2 or whatever, and the disk throughput needs to be no less than this, and network latency needs to be that. So you could actually set all these parameters.

Before getting your application up and running, Gravity will check if your environment is compatible. But once it’s running, it will run hopefully forever. Our operational model is like the extreme use cases that you could take this cluster image and kind of stick into satellite, shoot it into space. It will just run there without any supervision. So that’s the level of simplicity that we’re aiming with this project.

Again, we’re not there yet, and yes you do need to do a little bit of ops, but it’s nothing compared to what it would take to just manually run several Kubernetes clusters using some kind of control plane or something.

[00:41:43] JM: So if I’m a SaaS company, like a Kafka provider or log management provider and I want to make my SaaS offering portable. You have an open source project, that Gravity is open source.

[00:41:58] EK: Yeah, open source. 100%.

[00:41:59] JM: I can just take your open source project and use that to package my software and give that to the target customer.

[00:42:07] EK: Correct.

[00:42:08] JM: Where’s the business model for you in that?

[00:42:10] EK: There are certain enterprise capabilities that our customers require. So obviously they want a vendor support. That’s obviously number one thing. But in terms of

product features and capabilities, if you try to sell a software like this to any serious organization, they will have some requirements for you. They would say, “We want FedRAMP compliance, or we want detailed audit logs, or we want your application to integrate into our corporate SSO.” So these are the kind of things that we charge money for. So if you want this cluster replicas to just seamlessly integrate with other things that exist on like a target cloud where you’re deploying, so those are some of the enterprise features and capabilities that we offer.

[SPONSOR MESSAGE]

[00:43:04] JM: LogDNA allows you to collect logs from your entire Kubernetes cluster in a minute with two kubectl commands. Whether you're running 100 or 100,000 containers, you can effortlessly aggregate, and parse, and search, and monitor your logs across all nodes and pods in a centralized log management tool.

Each log is tagged with the pod name, and a container name, and a container ID, and a namespace, and a node. LogDNA is logging that helps with your Kubernetes clusters. There are dozens of other integrations with major language libraries, and AWS, and Heroku, and Fluentd and more.

Logging on Kubernetes can be difficult, but LogDNA simplifies the logging process of Kubernetes clusters. Give it a try today with a 14-day trial. There's no commitment. There's no credit card required. You can go to softwareengineeringdaily.com/logdna to give it a shot and get a free t-shirt. That’s softwareengineeringdaily.com/logdna.

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

[INTERVIEW CONTINUED]

[00:44:24] JM: As a business, I mean, you have to offer whatever those features are, compliance features, to the SaaS company that purchases it for you. If I’m a log management company, you got to charge them a high price, and there are only so many SaaS companies, and it seems like a tough business to be in, because there’s only many SaaS companies and

you're going to have to get all of them to pay a high price and they might have the engineering savvy to build compliance mechanisms themselves.

[00:44:56] EK: Forget about business for a second. I think the most important thing is you want to build something useful that kind of changes the landscape of the industry, and that's really the sexy part of what we're doing.

I just recently completed reading this book called *Innovators* by – I forgot his last name.

[00:45:13] JM: Isaacson.

[00:45:14] EK: Yes, correct. It was interesting to kind of follow how our industry was changing and it's just always like some fascinating individuals with crazy ideas and some companies. Not all of them were commercially successful. There are so many failures. Like Lisp machines and almost anything Lips, there was no big money error. Building something that changes, like makes a lasting change, that to me is the most important thing.

What I want for Gravity to accomplish – By the way, it's not just Gravity. You can get into a much better place, in this amazing future by using solutions that are kind of agreeing with our vision for the future, which is you're developing your application using open source components and using open standards. You're not relying on any kind of proprietary APIs. You try not to use many kind of cloud provider-specific APIs. If you're capable of doing that, then you have this ultimate freedom to your application anywhere, especially if it does not acquire administration.

I think most developers would agree with me that it's a fantastic world to live in. It's going to take us a while to get there, many years. Many startups will have to help. HashiCorp, basically, on a same mission. They have this multi-cloud message. What is terrform? It's a way for you to have exact same codebase that gets your infrastructure identical everywhere. What is vault? It's an open source version to proprietary secret management platforms that all these clouds are adding. No. Don't use proprietary stuff. Use open source stuff from companies like HashiCorp or from us.

That to me, what's important about Gravity. If we succeed, that is going to be enormous market opportunity too, because we have to make sure that cloud providers, AWS, Azure, GCP, all of these guys. They need to be just dumb providers of commodity compute. They are absolutely uncomfortable with this message and they will always fight us. But that is what developers want. If you've been following this industry for a while, and I'm sure you have, you see that developers are getting more and more power in decision making within organizations.

So that is really what we're betting on. So counting SaaS companies maybe like a little bit counterintuitive, but just counting people who code or just counting like lines of code written every day. That is just enormous, enormous part of the world's economy now and enabling this like multi-cloud future where cloud providers are irrelevant. That to me, that is a good reason to end up in Innovators book written 50 years from now. So that's really where Gravity is going, just to make sure that the future is going to look like this, like I just described, and not as like the 90s when Microsoft was dominating, never again.

I had some of my friends who stopped programming and went and gotten MBA degrees or something.

[00:48:13] JM: Seriously?

[00:48:14] EK: I don't want that to happen again. Yeah.

[00:48:16] JM: Because they were so disgusted by the ecosystem?

[00:48:19] EK: Because it was extremely grim to be a developer in like late 90s, because you knew, if you build something out value, then Microsoft will just make it built – They will just build it into Windows like they did with Netscape Navigator. Then you're also at the mercy of Microsoft APIs. They will just announce, "This way of building apps is obsolete. Everyone is migrating to this." You would basically have to go and redo your application.

[00:48:46] JM: The funny thing about that is like whether or not there would have been regulation. I think Microsoft would have lost its position like inevitably. I think like between Linux and just –

[00:48:59] EK: It's actually exactly what happened. Regulations didn't do a thing.

[00:49:02] JM: Well, I mean they got the CEO to basically resign, because he was so exhausted, right?

[00:49:07] EK: Then look at their market cap. What happened after the CEO resigned? They quadruped in size or something. They continued to do really well without Gates. What happened to Microsoft was internet, was Linux. Windows just became irrelevant. So I would argue that regulations didn't really harm Microsoft that badly. The only reason that Apple is getting away with basically doing the exact same thing in the iPhone platform is because Android is around. No one can claim that Apple is monopoly, because what's the market percentage they have? It's probably like single digit or something?

[00:49:41] JM: Yeah, something small.

[00:49:42] EK: Yeah. But, yeah, they all want to do the exact same thing. Just lock you in and charge you a fee on everything. So what do you think happens if AWS becomes the only cloud you can run it? I guarantee you that pricing for everything on AWS will start increasing. Simple email service will be like five times the price.

[00:49:59] JM: I don't know, man. What's interesting is how Bezos has used the AWS cash flows to subsidize the marketplace. Somebody told me that the market –

[00:50:07] EK: That's a good point.

[00:50:08] JM: Not an Amazon insider, but things are still cheap on Amazon. Literally, every article of clothing I'm wearing is from Amazon. I interviewed the guy who runs Confluent, the Kafka company, and I made sure to wear only Amazon essentials clothing while talking to him, because they're having this kind of issue with the Kafka licensing stuff. Anyway, I didn't even tell that to him. That was just a personal irony.

I actually like the positioning that like where you're starting is with a thesis about the way that the future is going, and you have a very specific. I know you said getting away from the business, but like you have a very specific value add that you're trying to do with what Gravity and Teleport are doing right now. You have a very specific market.

With that core competency, there are a lot of adjacencies that you can expand into. One of the reasons why I think it's great to position yourself that way, and I think this kind of strategy in business is underrated, the idea of going into a adjacencies is they opportunistically emerge. I mean, one of the reasons it is underrated is because it gives you optionality.

The Kubernetes market, we have no idea what's going to happen. That's just the reality. We have no idea where this thing is going. It's going up into the right, but we don't really know the other axes on which it is going up or sideways or down or whatever, right? You pick something, just something, and then have a vision for what the future is and expand into adjacencies as they present themselves. So what would those adjacencies be?

[00:51:48] EK: For Gravity or –

[00:51:49] JM: For your business, Gravitational. Where do you want to take it? What do you want to do?

[00:51:54] EK: Well, there's going to be a lot of – Right now, I think Gravity is just like the only open source and somewhat open standard way to do downloadable SaaS. But you can look into the tenant management, for example. If you're running SaaS business, not any type of SaaS, but there are plenty of situations where you'll have customers who'd commit and say, "We don't want to share the underlying hardware with other customers. We like your SaaS. We want to hit like signup button, but can you make sure that my account is isolated from other tenants? Put me on a separate hardware." Sometimes people would say, "Put me on a separate VPC. We don't want to share network with other tenants on your platform."

So these are the things you could try to do inside of a Kubernetes cluster and it gets complex pretty quickly, or if you're using imaging tool like this. Someone clicks signup button. You instantly deploy a brand new instance of the app from that image. Takes like a few seconds.

Now you have completely isolated cluster that runs, again, as an appliance. We don't need to monitor it just for that tenant. That's a pretty magical capability.

Obviously, if they ask you, "Hey, can you have many points of presence for us? It's just for us." For different regions, you could expand geographically and you could just do it with ease without having to think, "Which cloud is available over there. Oh! [inaudible 00:53:18] region. Fine. Or there's like an equinox colo facility. Sure, we'll stick in there." You just absolutely don't care what kind of compute you have. It's a huge use case.

Actually, at Mailgun, we had this demand from our customers to have a European POP forever. Look, we have a pretty competent engineering team and we've kept telling ourselves it's just going to take us a long time to ever consider having two regions instead of one. It's a serious undertaking. It's true even today for most SaaS companies to expand from one region to two, let alone 20.

With a solution like Gravity, you could do it as long as you adapt this kind of image-based deployment model, because once you have 20, 30 or 200 instances of your cluster running somewhere, you want them to be absolutely identical, which means that, no, configuration tweaks on a per cluster level. They should not be allowed. So that is something that Gravity does as well. Each cluster is basically a read-only. It can only accept code updates. So that allows you to kind of run many, many, many, many clusters with extremely low operational overhead. I think it's incredibly valuable even today.

Yeah, Gravity is doing well. We just open sourced it I think less than a year ago, just a few months ago, and there is already a significant interest in the industry and from investors as well. So, yeah, it's a project that clearly has legs. I'm really proud of it.

[00:54:51] JM: What's been the toughest engineering problem that you've solved in building the open source project?

[00:54:59] EK: I guarantee you that the toughest engineering problems were solved by toughest engineers we have, and it's not me. But I've been always fascinated by – We have some engineers who not just experts at being like awesome coders, but they're also domain experts.

We have this one guy who's like absolutely amazing with security and compliance. He'd be looking at FedRAMP spec like on one side. On the other hand, will be like figuring out how to make all of these things true with like zero configuration for the end user. That's really like the true value creation. When you're converting a government regulation into a robot, that enforces all these rules for you. That to me is pretty tough.

So on one hand you're dealing with something boring. With your left hand, you're dealing with something boring. With the right hand, you're creating something beautiful, which is the product on the other hand. Maybe it's not the toughest thing, but it's definitely fascinating to me.

[00:55:58] JM: That'd be pretty tough for me. I wouldn't be able to concentrate long enough to complete that task.

[00:56:03] EK: Yeah. If you enjoy reading RFCs, if you enjoy understanding all these different standards. Yeah, then Gravitational is a fun place to work for sure. We're definitely not one of those kind of move fast and break things companies simply because none of our customers are like that.

[00:56:20] JM: What are the other problems in infrastructure management that when you talk to customers or you just talk to people in the space who maybe they're not your customers but they are shopping around for this kind of software. There're a lot of people listening to this podcast that are looking for ideas in this space to start companies within. What are the other problems that you see, just predominant problems?

[00:56:48] EK: They would probably agree with me if I said that most companies want to implement basic rules, like the way you run your SaaS, like mob obvious things. Developers must know a touch production data. You don't want random Googler reading your Gmail, right? When you approach a startup here in San Francisco and you ask if their developers can actually touch production data. You will be surprised how many companies, how big they can actually get before they start fixing it.

The fact random developers at so many companies can see production data and touch it is pretty crazy, and how they do it, yeah, if they have an SSH key to get into a machine that

production is running, they can totally do that. It's not very sexy area for some people. Because if you try to solve this problem, you're running into this risk of placing developers off, because developers don't like getting access denied or getting some authorization. I know some organizations, they have interesting rules with really funny names, like four-eye role. In order for you to get an interactive session into a remote server using SSH or kubectl exec, like any of these commands, someone else needs to be watching you, like four-eye policy. It's almost like extreme programming if you remember what that was.

Yeah, implementing these things, it's absolutely critical for companies as they scale to clean this up, because you could probably reliable if you don't do it overtime. It's just a complete madness what's happening out there right now.

We do solve like part of this problem. We have a lightweight product that we – It's not an open source project. It's called Gravitational Teleport. It's basically a really, really lightweight, easy to understand replacement for open SSH. You just put it on your machines instead of SSHD, and you could do some of these things. But there's like a long list of things companies want to do. You probably want to get an alert when there is an SSH event. It could be a log in or a code execution, remote code execution, or deployment from an engineer who's on vacation, right?

Why would someone who's physically in Hawaii, and no, they're not working form there. They're just on a real vacation. Why is that person running a deployment? Ask how many companies having something like in place that solves it elegantly, and it's not really known. You will see that it's not that common. People kind of keep kicking that can down the road until they get hacked and end up in the front page of Hacker News, or yeah, until they hire expensive security people who come in and fix. So just the fact that relying on open SSH and its simplest form for critical parts of our infrastructure, it's kind of laughable.

[00:59:35] JM: How is the experience building Gravitational compared to building Mailgun?

[00:59:39] EK: Okay. The positive thing about Mailgun is it's almost impossible for me to go into a party like wearing Mailgun shift. So many people will be shaking my hand and saying, "Hey, we use Mailgun. It's an amazing product." You feel like Paris Hilton on a party. It's really fun.

Gravitational is like obviously a little bit younger, although we are now getting more and more kind of brand recognition. So that's on the kind of positive side towards Mailgun. But the positive side towards Gravitational is we're not a SaaS company. We actually don't have any ops.

There was a weekend not a while ago when Google Cloud was down, and I had a breakfast scheduled with a friend of mind and I just ended up eating it alone because his entire team couldn't figure out how come their application was down. It happens every once in a while. I didn't care, because we don't have any servers. We make open source software. It sits on GitHub. People come in, download, and it's incredibly liberating. Our website is a statically-generated thing. It sits in S3 bucket. I can physically shut down every single instance we have in the cloud and the business will continue to work fine. That is unbelievable relief compared to my Mailgun experience.

I'll tell you a story real quick. It was like Saturday and I was walking my dog. It was year 2012 and someone texted me, "Hey, dude. Mailgun is on front page on Reddit." I'm like, "What?" I pulled my phone, www.reddit.com. Yup. The title says, "If you get an email from this terrible company, Mailgun, make sure not to open it or something." I was like, "What is going on?" We started looking into it. It turns out Mailgun had a customer. I'm not going to name that company, and they had one of those contact us forms.

When you see them online, you go to these website and there's like from, to subject body. One of the fields they added was put your email here if you want copy of your message to be sent to you. On the surface it seems like an innocent thing to do. Remember I was telling you about this crime that happened in the email space? What they do, they find these forms and they do a robot that basically start sending spam using that form by sticking real email addresses of people.

As it was going out, it was sent delivered by Mailgun, because it's the part of email signature. That's how we ended up in the front page of Reddit. So not having to do deal with these issues is incredibly great, and this is why I'm so happy that Gravitational is not a software as a service company.

[01:02:10] JM: Okay, last question. You could feel free to skip this one also. But you worked in venture capital in Austin for a little bit.

[01:02:17] EK: A little bit.

[01:02:18] JM: I'm from Austin.

[01:02:20] EK: I lived for 12 years in Austin. It's my favorite city in the world.

[01:02:22] JM: Oh, really? Oh, okay! I grew up there.

[01:02:25] EK: Congrats! You had a very good childhood. Yeah.

[01:02:28] JM: It was great. When is it going to be possible to do a startup in Austin?

[01:02:32] EK: I'm going to be honest with you. I left Austin for this very same reason. Just one reason that I could not raise any kind of capital when I lived there. Ironically, Mailgun is now a Texas company. So it's a fairly sizable company based on Austin or San Antonio. So even though I had to leave the state to start my baby, the baby ended up being a Texan. So I think it's pretty ironic.

So I think that startup system in Austin right now is like hundred times better than it was originally. It feels to me that things are different, but you'll never really know until you try to build your own. I was a venture partner at a local VC just briefly for one summer. I can't really share much, because I just quickly realized that's not for me the experience, because you have to have this insane context switching ability if you're an investor. You need to be able to quickly fall in love or hate a specific idea or a specific entrepreneur and do it several times a day. That to me is just emotionally exhausting. So I couldn't do it. Yeah, so other than that, I can't really predict when Austin is going to surpass Silicon Valley.

[01:03:45] JM: Yeah. Part of me thinks you could do it today because of – I mean, same reason you said why you love GitLab, why you love how GitLab built its company.

[01:03:53] EK: Totally. Also, I'm finding that the changes happening on both sides, that investors in the valley, they're way more open today to fund a company that is not based here.

[01:04:03] JM: Yeah, the thing is I don't think it's an investor problem. I've said this before, but it's just like there's something about Austin where it's like – Man! I love going to Austin to relax and hanging out with my friends. It's like that's the wrong atmosphere to be building a startup in, or maybe not. Maybe actually it's fine. Maybe that's fine to be able to startup at this point.

[01:04:21] EK: I think it's fantastic. I've worked at a startup in Austin for a while. It was great a great experience. Company was called Pluck. It was awesome.

[01:04:28] JM: Dude! It's actually funny you mentioned that. When I worked at Spiceworks, I was really happy. I did an internship with Spiceworks and I loved the people there. The software was great. The company was great. I don't know why I'm there, but – I don't know. There's something in the water here that I really like, and it's just people are hungry and it's just nonstop craziness. But maybe that's just – I don't know. Maybe it's a matter of time. Hard to speculate.

[01:04:53] EK: I hope it is.

[01:04:54] JM: Yeah. Ev, thanks for coming on the show. Great talking to you.

[01:04:57] EK: Likewise. Thanks for having me.

[END OF INTERVIEW]

[01:05:08] JM: Monday.com is a team management platform that brings all of your work, external tools and communications into one place making cross-team collaboration easy. You can try Monday.com and get a 14-day trial by going to Monday.com/sedaily. If you decide to become a customer, you will get 10% off by coupon code SEDAILY.

What I love most about Monday.com is how fast it is. Many project management tools are hard to use because they take so long to respond, and when you're engaging with project

management and communication software, you need it to be fast. You need it to be responsive and you need the UI to be intuitive.

Monday.com has a modern interface that's beautiful to look at. There are lots of ways to use Monday, but it doesn't feel overly opinionated. It's flexible. It can adapt to whatever application you need, dashboards, communication, Kanban boards, issue tracking.

If you're ready to change the way that you work online, give Monday.com a try by going to Monday.com/sedaily an get a free 14-day trial, and you will also get 10% off if you use the discount code SEDAILY.

Monday.com received a Webby award for productivity app of the year, and that's because many teams have used Monday.com to become productive. Companies like WeWork, and Philips and Wix.com. Try out Monday.com today by going to Monday.com/sedaily.

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

[END]