# EPISODE 1405

**[00:00:00] KP:** Happy New Year, Software Engineering Daily listeners. I'm really looking forward to bringing you a great set of interviews with software professionals in the coming year. And one of the other things we're working on here at Software Engineering Daily HQ is trying to broaden the number of voices and diversity of voices you hear on the show. We're currently looking for part time hosts who are interested in conducting some of these interviews. If you think that's something you'd like to explore, please visit softwareengineeringdaily.com/applying-to-host. And that's T-O for to. Softwareengineeringdaily.com/applying-to-host. There's a short set of instructions there that tell you exactly what to do to get going and to get details on what the commitment is.

Separately from that, now that 2022 is here, we're starting to roll out our plans for the new year. There's a lot of great stuff on the agenda. And if you're a software company, or any company that needs to get your message out to a software engineering audience, you're a listener, you know the type of person that's here. If we're a good match for your company or the company you work at, please tell someone in your marketing department that they should consider Software Engineering Daily. If you're interested in sponsoring the show, send an email to sponsors@softwareengineeringdaily.com.

Now on to the show. Infrastructure as code is a concept that has delighted software engineers, DevOps and engineering management across the board. It's neither fun nor efficient to configure the infrastructure and environment software teams require. Operating software at scale on a cloud, on-prem or hybrid model is a problem of modernity that many enterprises find surprisingly challenging.

Rob Hirschfeld is CEO and cofounder of RackN, whose software helps IT teams maintain distributed multi-vendor operations with consistent operational control. In this interview, we discuss the approaches being adopted by modern software enterprise teams for infrastructure management.

[INTERVIEW]

**[00:02:07] KP:** Rob, welcome to Software Engineering Daily.

**[00:02:10] RH:** Kyle, it's a pleasure to be on the show.

**[00:02:13] KP:** To kick things off, can you tell us a little bit about how you got your start in software?

**[00:02:18] RH:** I would be happy to. I came at it from an engineering path. Although I was always a software person. And I sort of been trying to build software for a long time. But because I went through this engineering, specifically industrial engineering, I'm very process-focused in my outlook because of my background. I've never done a day of real industrial engineering. But when it comes to supply chain conversations, and process, and delivery pipelines, I actually have a lot of background in that that served me very well over my career.

**[00:02:52] KP:** And this is going to be a difficult question, but is there any way you can encapsulate some of that wisdom in a fortune cookie? What are some of the lessons that someone with a computer science degree is maybe going to have to learn on the job?

**[00:03:04] RH:** You know, I'm a big fan of Goldrat and Theory of Constraints, which we talked about with Lean software. Although I think that people don't necessarily always understand that. And the fortune cookie to me is you can't run systems at 100%. And slack is not a bad thing in a system. As a matter of fact, it's a critical thing in the system.

And one of the things I've seen with developers and developer productivity, the sort of drumbeat of make my developers more and more and more and more productive, is that when you do that, you have a tendency to look at your time and your efficiency and what you do as the thing you have to optimize. And it's not. You have to optimize the performance of the whole system. And sometimes that means that you actually should go slower and do work to support other groups, or add in protections or more quality, right? Do those things. You running at 100%. If everybody has that mentality, the whole system is actually much more fragile, and much more likely to break.

**[00:04:10] KP:** Good note. Yeah. So you're a cofounder at RackN. What does RackN do?

**[00:04:16] RH:** RackN is an infrastructure as code platform. And we're a software. So we're a little bit different in that we help companies run and operate infrastructure. So everything from bare metal data centers up to cloud infrastructure. And we do it in a way that lets them do it themselves. So customer autonomy is actually our driving mantra. So rather than taking operational control away from people, we're actually looking for ways to make it easier for them to maintain and run their infrastructure.

**[00:04:45] KP:** So what does that look like from an engineer's perspective? If I want a SQL Server database and two or three other things, how can I speak them into existence?

**[00:04:57] RH:** That's where we talk about something of an infrastructure pipeline, right? Getting down to an application at the end of that pipeline is the final steps. So what we really focus on is not as much how do I build SQL server. There are good ways to do that. It's how do I prepare my environment, so that when I go to do the SQL server work, it just works reliably, repeatedly, on any infrastructure I want to choose.

And so what that looks like, we have a software platform called Digital Rebar. And that software platform lets you assemble and connect together different infrastructure automation components in a sequence, what we would call a workflow. And then those workflow components are all infrastructure as code module. So you pick and choose the components and modules that you want. They can use other tools like Terraform, or Ansible, or Bash scripts, or PowerShell, and then string them together into this end-to-end flow. And that actually starts. We have customers where that starts even before a server is delivered on their data center floor. It can start when they get a manifest and import the data for the manifest to tell the server what it should be when it's plugged in and turned on. And then for day two, they can then come back and say, “Oh, I need to rebuild and reset this infrastructure and take that all the way through the process.”

**[00:06:20] KP:** So is Digital Rebar a no-code solution?

**[00:06:24] RH:** It isn't. And we really haven't tried to build it as no-code. And there's a couple of good reasons for that. One is because of the operational aspect of automation, that when operators do work, a lot of times they need to have the transparency and controls to fit that automation into their environment. So doing it as sort of drag and drop modules that you piece together can undermine your effectiveness as an operational platform. That doesn't mean we don't have a lot of pre-built components that then fit together. And we have a lot of what I would call opinionated infrastructure pipelines that you start them and they just run all the way through the system.

The value of having people like when I think of no-code, I think of it as like I have a graph and I drop segments like I used to do in way old back Visual Basic programming. It’s like, “Click this to click that. And yay!” For us, and in our experience, that's not as good an operational pattern for automation.

**[00:07:20] KP:** Gotcha. Makes sense. When you look across adoption, is this something that maybe a software engineer feels empowered to do themselves? Or is it something maybe an IT or DevOps professional is building?

**[00:07:33] RH:** Generally, we're talking to DevOps professionals. But today – And I would actually say SRE. From an SRE’s perspective, coding and writing that automation is a big part of what they do and what they're trying to do. The thing that I would say is, if you're looking from a development perspective – And this is an interesting question, because there's definitely different rules of thought on some of the lower level infrastructures as code platforms, like Pulumi, where it's very developer-focused, write code Terraform, where it's very config file, right? YAML files and JSON files oriented. We're actually very API focused.

And so from our perspective, what the developer wants to be able to do is have an API, a declarative way to say start this work. Let it run. And there's a service that actually runs that work for you. And then you can monitor and check and watch things that are happening.

So as a developer mindset, it very much becomes starting a domino chain, if you want to think about, but the API's that enable you to say, “I need this thing to go,” and become something else at the end of that chain.

And there's an interesting thing. This is why there are infrastructure pipelines. A lot of times, what you want is, I want to start an infrastructure pipeline with these settings. Maybe run something on one cloud. And then you want to be able to, “I want to use that same pipeline.” And this is where the infrastructure as code mentality is so important. And where RackN is really about this reuse and portability concept. I want to take that pipeline and let it run on a different cloud or a different operating system. And as a developer, I want to be able to say, “This is a thing I can use. I can just start the pipeline and let it run, and then copy it and reuse it and get the benefit of that modularity in reuse.

**[00:09:19] KP:** And what about destinations? I mean, we talked about deploying to the cloud. There's lots of other places people want to ship code to these days. What's the reach that Digital Rebar can help me achieve?

**[00:09:29] RH:** So we actually are one of the only products anywhere that has the depth. Really, only has the depth and bare metal and all the way down at the physical layer on the infrastructure. So we have in our past automated everything from Raspberry Pi's to regular server class, enterprise class infrastructure that has clean, very well thought out API's, and everything in between. That adds a lot of useful complexity, if I will, to the operational environment. You can't take a server and say, “I want to ignore how the networks work, or I want to ignore how storage works. You have to be able to adapt to that. But the ability to do that actually allows you to build a pipeline that abstracts it away. And then you can say, “Oh, yeah, I'll rely on Digital Rebar to do the OS provisioning, and set up the rain BIOS, and patch the systems and firmware and get all that stuff going. So I don't have to worry about that.” Even down to building ESXi clusters or installing a virtualization platform.

**[00:10:28] KP:** I know this is a bit of a non-sequitur, but we're on the topic of what you're building for people. How has this recent log for JADE bug impacted you and some of your customers?

**[00:10:38] RH:** Us, personally, everything we do is going. And so we have very minimal exposure from a Java perspective. Every one of our customers is scrambling to one extent or another, if nothing else to prove that they're not affected. But what we do is absolutely critical for customers who are mitigating Log4j. And this isn't true for just Lof4j. It’s true for BIOS. It's true for routine patches. What we've enabled customers to do is have this end-to-end automated process.

And one of the things that we encourage, and our customers love, is this idea of repaving their data centers. It's a ransomware component to and regulatory for some of our large bank customers. But we allow you to say, “You know what? I need my system to reset itself.” And you can push a button and tear down infrastructure, rebuild it and reset it back into the states that you want it to be or into a new state. And when you look at like Log4j, as an example of across the industry, if you don't have a process by which you can quickly roll out infrastructure configuration and patches and have it predictably used, then you're in real trouble. And fundamentally, that's what infrastructure pipelines are doing.

If I could, it's very similar to what we're talking about with CI/CD. Like, we're very excited about CI/CD platforms that can take code and rebuild it and get new binaries. What we haven't seen yet is what Digital Rebar does for infrastructure pipelines, where you can do similar type of, “Oh, I need to roll my cluster and my patch my systems and make that part of a routine.” “Oh, I have a new patch to roll out. There's an automated process that just takes it through that whole lifecycle.”

**[00:12:27] KP:** So what, historically, have companies done in lieu of having a solution?

**[00:12:32] RH:** Oh, they've run around with their hair on fire, frankly, or hope they don't get caught in the breach. I mean, a lot of times, we just ignore these problems, right? There's been a series, a drumbeat, of firmware patches even at chip level execution problems that should get patched. And a lot of times we just don't worry about it. We see people move very slowly through picking up new versions of software, even when they know their security vulnerability is at stake.

Log4j is unique, one in its breadth, and also its severity. And people are realizing that systems that they didn't think they would have to patch are all of a sudden vulnerable. I hope that it's spawns a conversation about software lifecycle. But really, how you're doing these ongoing maintenance and process operations?

**[00:13:23] KP:** Yeah, I've always found the sort of lack of concern in some organizations to be a bit shocking. Like if you're a startup and you have limited budgets, I'm not saying that excuses you from having good security policies. But at least you're looking at some bottom line. If you're a large enterprise company, you can add one headcount. And that might be all it takes to just do some basic pen testing sometimes. Why isn't this happening?

**[00:13:44] RH:** Yeah. I don't think it's a pen testing problem. It is a rollout. It's a pipelining problem. And there's a couple of reasons that I see that are endemic in the industry. One of them is that we are siloed organizationally. And so you get into cases where it's very, very hard to get teams to collaborate together, because they each have their own responsibility for that. And because of that, and sometimes there's expertise reasons for it. Sometimes it's politics. Sometimes it’s budget, right? Let's accept styling status quo. But we have turned around to build tools, because it's easier to sell to a silo than it is to sell across silos. And I know this firsthand, because RackN is really a cross-silo app tool, right? Our goal of building pipelines means you want people to collaborate.

And so when we’ve built things, we haven't worried about, “How do I hand off my Terraform work to my Ansible task?” We built them as two different tools for two different teams and two different users. And nobody sort of got upset that you couldn't integrate those tools together easily. We did. RackN said, “Wait a second. What we do has to connect all the dots together.” And that's what makes the seamless internet work. But as industry, it's so much easier to sell – To make it all about the sales motion, not about the tech. But sometimes that's what it takes to get a tool ubiquitously adopted.

**[00:15:16] KP:** Well, another element from the news recently is we've had a pretty serious Amazon outage. It's got a lot of people thinking that maybe they need to be multi-cloud. Is that a common path you're seeing? And can RackN help me do that?

**[00:15:27] RH:** Oh, dear. Multi-cloud is real, although maybe not because of the Amazon outage. And we don't think of it as multi-cloud. We think of it as multi-infrastructure, which would sort of include hybrid. But the goal that we see customers wanting is they don't want to care about which infrastructure they're using. So multi-cloud is about being able to abstract out the cloud to an extent so that they have portability. It doesn't mean that I'm running an app in three locations, right?

So the Amazon outage, I don't think anybody's waking up and saying, “Oh, I really wish my Amazon application had parts of it running in another cloud.” Because the reality is you're actually making the systems weaker because of the complexity of those systems. What they are waking up and seeing is, “Okay, did I – Not realize how connected I was into these different systems? Or am I more locked into Amazon than I might want to be? Or maybe I should, instead of assuming that Amazon is the magic bullet to solve all of my operational woes, which it never was, and we need to get over that cargo cult, but they're coming back and saying, “Wait a second, maybe I have all this infrastructure and all this IT experience that I'm already using. Maybe I should make that better. Maybe I should slow down my role a little bit from that perspective.” And those are healthy conversations for it.

I do think the other component of the Amazon outage that people should be awake about is how complex the Amazon operating environment is. So we've been told for a long time, Amazon is smarter than us, they're better than us, right? They wear capes when their operators show up. They're just that much better. And we should give up trying to be like Amazon and let them be like Amazon.

One of the things that this outage exposes is Amazon's operating environment is much, much more complex and harder than any other company's operating environment is because they're operating at a different scale with a whole bunch of services they've chosen to connect together. And so I think, and I hope that when companies look at Amazon's operational capabilities, they start thinking, “Wait a minute, I don't have to operate at the level Amazon does to be an excellent operator.” And our hope is that this is sort of RackN’s mission, that we are showing up with a whole bunch of out of the box proven processes and things that are wired together in best practices from global financial institutions and security aspects from hyperscale hosting providers and media companies. And when we've built something, you can just lean on it as working. But you're not leaning on us having to operate it. You have a degree of isolation and separation.

**[00:18:17] KP:** Are there any compliance advantages to adopting your solution?

**[00:18:21] RH:** There's a lot of compliance. One of the things – So when we run – We don't run our software. We help our customers run it themselves. And to us, it's important to us, because I actually don't want my customers secrets, and credentials, and tokens, and log messages running on my servers. That's always struck me as a problem. And the fact that SaaS companies take on that burden is always made me scratch my head when I sign up for any SaaS and give them credentials, or secrets, or special information.

So there is a big degree where doing the operational work and having that sensitive information and managing it yourself does reduce your compliance risk considerably. And there's a balance, right? You're responsible for it. But you're responsible for it even if you sign a contract with a SaaS that says they absolutely promise to not give it away. If they give it away, you're still your data.

**[00:19:21] KP:** What's a typical onboarding like for a company that maybe doesn't have a solution today and decide they want to adopt? Who gets involved? And how long does it take?

**[00:19:30] RH:** So we work really, really hard on keeping things simple for that out of box experience. So a lot of our prospects come on board, they get the system running in an hour. By the end of the day, they're doing pretty sophisticated operational tasks in a lab, and that is by design. We work really hard for that out of box pre-wired components to work so that people can be productive.

The challenge with operations is that that's really great if you follow everybody else's pattern exactly. And the hint is nobody ever follows everybody else's pattern exactly. And so from that sort of baseline, you do need to start learning enough to start to integrate your systems, your operational environment into the way Digital Rebar does it or sort of vice versa. And we spend a lot of time creating easy ways to extend. And this is another one of our core missions. I mean, so let me back up a second. And if I could, I'll actually do with for the story from our original day. So when we were originally before RackN, and we were a team at Dell, charged with shipping clouds from the Dell factories. So the idea was you buy a cloud, drop ship it, you get a rack? Yeah, everything's great. And that didn't work didn't work, because every operational environment was different. And so by the time people started turning all that stuff on, the differences in their operational environments meant that we actually rebuilt every single rack to conform with the operational environment.

And at first, we were really frustrated and angry about that. And eventually, we realized that was the reality of the system. And so what we had to do at RackN is fine this balance between a whole bunch of stuff that works out of the box, and that we can continue to maintain and patch and give you our automation, right? Standardized automation libraries. And give standard extension points and override components in places to inject configuration so that your unique operating environment can be reflected back in the automation also. And that's the infrastructure as code magic, of being able to say, “Alright, I'm using standard modules and a standard process. And also added in the unique things that make my environment a little bit different, my IP ranges, my naming conventions, the systems I have to interface with, or data I have to pull in to make everything work.” And the extent that we do that without then breaking it for you being able to patch an upgrade your automation process, that's taken years of refinement.

**[00:22:11] KP:** Well, having the API is, I think, a fairly unique feature of your service. Do you have some typical use cases for how people do that? Is it part of another automation system on their end? Or a developer who's pulling that? Why an API, I guess?

**[00:22:26] RH:** Oh, boy, my cofounder has always had this vision that the best use of our system is when it's fading into the background, right? That you can just count on it working, and feeding, and interacting with the systems around it. And that's a big part of how we look at operational environments, right? Nobody wants a person to have to put their thumb on a button to make operational logic work. It's really an anti-pattern.

And so we have a big media company that just hooked us into their CI/CD infrastructure, right? They have a big Spinnaker, very big Spinnaker infrastructure. And we got involved with them because they wanted to repatriate some of their rendering workloads. And it also had to be done from a Spinnaker job. So when work happens, it goes through Spinnaker. Spinnaker calls us. It does the work, and it calls it back. With a game company, exactly the same thing. And they don't log in to the system very often. They can watch everything happen. Transparency is really, really important to us and to our operators.

But fundamentally, the process starts, it runs, it ends, and we're just feeding the system back and forth on either side of that. And that's a really important thing. Same thing with a financial institution, where they bring in racks of servers all around the globe. And they turn those servers on. They've already pre-populated the inventory. Servers recognize their entry. And then they go through a complete process to build a vCenter cluster with no additional interactions.

And critically, this is really important, is if something doesn't match what it's supposed to do, the system stops, raises a red flag and says, “Hey, you need to fix this before I do any additional work.” And that in itself saves incredible amounts of time and money.

**[00:24:16] KP:** Well, I'm curious about the core deployment, this being able to run anywhere. It’s got to be a challenge under the hood. Especially on like, I guess, the cloud, if they honor their posted API's, that's maybe easier. But you have to code it perhaps three or four times. But bare metal, I mean, there's any operating system configuration. How do you test infrastructure like this?

**[00:24:38] RH:** Oh, boy. Some of it's about scars, and scar tissue, and resilience from that perspective. We didn't reinvent like BIOS flashing tools or out of band management interactions and things like that. We use other vendors tools. So that's part of part of what we've been able to do here is have a very flexible way to help handle, “Oh, wait a second, this RAID controller is different than that RAID controller.” Or if you looked at the way we do cloud interfacing, every single cloud is completely different. It's not just that their API's are different. Their operational patterns are different. And you have to build things in special ways for each cloud to make it consistent and repeatable.

And so a lot of what we've done here is we've decomposed systems and collected data incrementally. So the way we handle state is really important to this whole model. And it's worth expanding a little bit. Our expectation for state is that we are not a single source of truth, or a single source of truth, especially since everybody needs to be one as an anti-pattern. Instead, when we build up state, we assume that it's evolving and being added to and incrementally updated. And so we can get state updates from outside of our system. Ee can build state over time during a workflow and handle the fact that we didn't know things when we started a process to when it was finished.

And then all of the infrastructure as code components that go into that actually have typed variables. We have the option for ad hoc, but we encourage, as an infrastructure as code platform, typed variables. And you can actually come in and say, “This task requires me to have this information. And it's got to be in this schema,” so that you know that you’re going into that place. But it's a combination of pulling all those pieces together to give you the resilience to handle the variation. The long answer. I can dig deeper if you want. There's a lot of moving parts.

**[00:26:39] KP:** Absolutely. I mean, the state management is often a challenge for a lot of systems. Can you explore, like, what my interactions as a developer are going to be? Do you make any consistency guarantees, or maybe what am I typically going to store in my state?

**[00:26:52] RH:** I'd be happy to. So what we've done – And state management for us, this is actually one of the things that Digital Rebar had to do. Digital Rebar is a self-contained go Lang binary, right? We, by design, have no external dependencies when we install the system, which means we don't use a standard database. We actually built a database into Digital Rebar to handle the fact that some of the components, a lot of the components, in an environment are immutable and are uploaded as content packs and read only to an individual Digital Rebar server. And some of the pieces are very state specific. And like when you build up a machine state, you build up that machine state.

What we did was instead of trying to create objects – There's a hybrid approach. There's a lot of things in the models, machine model, or cluster model, or resource broker model, that are defined, right? Things that we know we need, an address, or a MAC address, or what workflow you're doing, or the name of the system. And there's also an incredibly flexible parameter system where you can add information to any system and define it. You can let it be ad hoc, but you can also define exactly what it is, and then build that up. And that parameter system, combined with like, here's known things about my object, and here's known things about the parameter system, allows us to do incredibly dynamic capabilities. So then we can say, “Alright, I'm in Amazon's cloud. There's information I need to collect for Amazon cloud. So it's unique to Amazon.” And there's some that's completely standard across all the clouds. And so you can define and set these parameters in a uniform way that is discoverable and understood across every infrastructure that we have. And you can also say, “You know what? And there's some unique stuff to that I only care about if I'm an Amazon.”

And all of that, because it's done with this parameterized system, you can make – We actually use patch a lot in strictly HTTP speak, right? People are used to get put, post, delete. There's also – And I strongly encourage, as a whole hour discussion, people to use patch in their API's where you can make individual field changes to objects. And our whole system encourages that behavior. And so when I'm making an update to an object, you can make up updates down to the very edge of that object and just that one thing, and test it to make sure you haven't had a race where somebody else changed the same thing.

But that thinking about state where we're not changing a machine object at a time, but we actually have a decomposed object where I can make incremental changes to sub-components of its state for automation is transformative. We can do things and interact with other tools and manage state across a lifecycle in ways that are incredibly powerful.

**[00:29:49] KP:** And can I trigger something off of a state change?

**[00:29:53] RH:** There's a couple of different ways. Yes. And everything we do is invented. And one of the things that we enable people to do is take an object that you want to watch and subscribe to the event stream for that object for that reason. So you can totally do that you can also build triggers that come back and watch for objects to change in specific ways and then take additional action. Or when they change for them, like we have a plugin for Filebeat, which then can feed like an Elasticsearch. And that can then say, "Oh, I'm going to monitor this class of equipment for this change in this field and then push that into another database." So yeah, everything we do is invented. So it's incredibly powerful what you can do with that.

**[00:30:43] KP:** I'm wondering if there's a story for traceability. I'm imagining a scenario in which something bad has happened. The root cause is certainly a few weeks ago, and I've got a digital archaeology mission I have to undertake. How can your tooling help me?

**[00:30:57] RH:** In a couple of ways. And this is where in automation, when I look at logging, and what type of data I'm collecting, there's layers of action here, right? So we have sort of Digital Rebar logs, where we know who took what on which machine with which permissions and sort of the classical role based auth. And we also have a job logging system. And this is real time. So it's super fun to watch. Like, if I'm running a Terraform plan, Digital Rebar is running a Terraform plan on my behalf, I can actually watch live as that Terraform plan is run, even though it's running on the server. And I don't have to worry about logging out or being disconnected. It's going to keep happening. The server managing that process. But I can see live logs.

In that job logging system, that information is incredibly helpful for diagnostics. So I can go back in time and say, “Okay, this job I had, ran. It decomposed into all these components,” and I can actually see exactly what happened through the course of that work stream and figure out everything that had to get done, everything that was supposed to happen, what updates were made, and things like that. It can generate a lot of data.

We had a customer cross over 10,000 machines managed by a single Digital Rebar endpoint. And for them, their system had generated so many jobs, like billions of jobs, that we actually had to reorganize how we store jobs to accommodate that type of workload. Because they didn't want to throw away too much of that data to solve. They didn't want all of them. But they definitely needed to be able to say, “Yeah, I’m a bank guy. I care about seeing what happened with the system for a little while.”

And we also had a customer who pushed it all to Elastic and then would start to do like time-based analytics. Why is this job taking longer? Why does this task take on average, 30 seconds, but every once in a while take three minutes or 10 minutes? That's incredibly powerful. When you can go back and say, “Okay, I have anomalies in my operational environment,” and you have the data to make that determination and you can come back and actually resolve what's causing it. Because a lot of times anomalies like that are operational failures that are lurking in the background for you.

**[00:33:16] KP:** If you start a startup that, I don't know, sells t-shirts, or coffee mugs, or something, I don't want to say you can be laxed about security and whatnot. But you can kind of adapt tools and not look too much under the hood. The antithesis must be true at a bank who has a bunch of worries that the average company doesn't have and has to ask a lot more questions than the average person. What's it like to make yourself available are exposed in the right way for banking technologists to look at the product and decide to adopt it?

**[00:33:46] RH:** Transparency, transparency and transparency from that perspective. We have a design philosophy. And this is super important for tech that's doing automation and provisioning and ops. Any magic is bad. So black box, it happened. I can't open up why and look, is fundamentally a problem. And this is one of the reasons why it's nice not to be a SaaS, right? When customers are running our software, they can see everything that's going on. They can see the logs, right? I don't have to worry about seeing some bank information or a social security number, because it's not us, it's them. They already have jurisdiction.

But for them, they actually need to be able to watch and see what's happened and go through that log and decompose the activity. It's really helpful for us when we go to troubleshoot, because we can pull those you can pull those logs or have them send it and we can let them figure out what's going on. But that's been a key a key part of what we build and how we build it.

**[00:34:49] KP:** With any large distributed system, there's an adage we have to keep in mind that assume any component can fail at any time and that everything has to kind of work out in some way. Can you describe what the failure experience is and how maybe Digital Rebar can help me get back on track?

**[00:35:07] RH:** I'd be happy to. And for Digital Rebar, one of things we built into our database was a Raft consensus algorithm. So if you to have a HA environments at the provisioning level, and the more often you provision, the more immutable, faster turns that you get in your infrastructure, the more important that becomes. So just at the Digital Rebar perspective, we've we built in those HA capabilities.

But there's a different element here that I think from our customers, they're not as worried about Digital Rebar staying up. They're worried about their data centers staying up. And one of the things that we do, and this will sound counterintuitive, is that when we find that there's a problem, the automation stops. And you see this even like with the Amazon outage. One of the things that causes outages to become much worse is when automation that retries something when it shouldn't be retrying.

So our design philosophy – And there are ways that you can tell the system, “Yeah, I know this is going to fail. I do want you to retry one or two times.” But by and large, when something breaks in an automation process, what we have people do is root cause it. Figure out why it's broken. And then we'll work with them to fix the operational components that are making that break. So that the next time –We're constantly getting through root cause. And that actually builds much, much more resilient systems.

Now, there's also a component of because of the infrastructure code capabilities, customers do a dev test, sit test, production rollout. And one of the ways that you also create very resilient systems is you don't make experimental changes in production. What we've enabled them to do, because everything can be locked in immutably and then replicated in a production environment, you can actually go through and do a dev test prod cycle and know when you promoted into prod that you've actually tested the systems really reliably and you've put them through their paces and it doesn't keep everything from happening. But it makes a big difference.

It also helps, because it's immutable, that you're not questioning, "Hey, did Rob slipstream in a change to a script somewhere that I wasn't aware of?" You can go back and say, "Okay, this is exactly what was deployed. This was how it changed. This is what the inputs were. And that's one of the places where infrastructures code to us, as a platform, infrastructures code platform, you're really treating this much more like a development experience, then you would like we've done in the past, where you reach into a system and say, "Oh, yeah, I'm just going to SSH in and nudge this service back to life." We really work hard to make it easier for our customers to fix problems the right way than sort of backdoor and fix stuff.

**[00:38:03] KP:** Well, Rob, before I let you go, I want you to tell listeners about your own podcasting work. Where can they find you on the airwaves?

**[00:38:10] RH:** I would be delighted to. And our podcast is really a community effort more than Rob's voice at this point. Since the pandemic sort of settled in and became a fact of life, we started doing a hallway track meeting called Cloud 2030. And people can find it at the2030.cloud is the website for it. And we have a DevOps conversation over lunch and then strategy conversation over breakfast. So Tuesdays for DevOps and strategy on Thursday mornings, where we have topics that we – We have an agenda for topics and we will sit down and roundtable real deep how are we dealing with software supply chains, and what is edge?

We do a lot of edge technology. What does DevOps look like? What does open source look like? And we have these amazing conversations where we have some long tenured people who've been coming for a long time. And we always have new people showing up who want to add their own voice. And that becomes a podcast for us. So The 2030 podcast takes those weekly sessions and then streams them so people can catch up on these discussions.

They are unlike anything else that I've seen in the industry, because it literally is this hallway track where we sort of decoupled people from vendor relationships and allow them to think through a 10-year horizon in what's going on. And it's open. So people can show up and say, “I have opinions I have questions,” and be part of that conversation. It's been remarkable. I'm in awe at the community and the conversations. And they make excellent listening. I learned stuff going back, going back through, and when I’m in-person too.

**[00:39:55] KP:** Absolutely. And where can listeners learn more about RackN?

**[00:39:58] RH:** RackN is simple, rackn.com is your place to drop in and see more about infrastructure pipelines. Download a trial for Digital Rebar. Try it on your own infrastructure and show yourself how infrastructures code could really be transformed into a development process.

**[00:40:17] KP:** Absolutely. Rob, thanks for coming on Software Engineering Daily.

**[00:40:21] RH:** Kyle, I appreciate the questions. This has been fantastic.

[END]