EPISODE 822

[INTRODUCTION]

[00:00:00] JM: Google's strategy for cloud computing is centered around providing open source runtime systems and a high-quality developer experience. Google is highly differentiated in its approach to the cloud. To understand Google strategy, it is useful to first set the context for the current competitive environment of cloud providers. I'd like to talk through that, and if you want a fast-forward straight to the interview with Eric Brewer, please skip about 30 minutes into today's episode.

Amazon Web services popularized cloud computing in 2006. AWS was first to market, which is given the company a large competitive advantage over the rest of the industry. It took Google several years to realize how big the public cloud market was and how good the economics of running a cloud provider could be.

Microsoft also realized the opportunity several years after AWS was started. The multi-year lead that AWS had on getting to market has given the company tremendous leverage, because AWS is the most widely trusted accepted default in the market, AWS has able to deepen the relationship more and more over time with its customers. Despite the fact that the proprietary APIs of AWS create a level of lock-in that bears some resemblance to the lock-in of Microsoft Windows or Oracle's relational database systems.

The brief history of software gives us some examples of what is supposed to happen next. Historically speaking, whenever a large company has operated a proprietary developer platform, the open source software ecosystem reflexively comes out with an alternative open source solution that is better than the proprietary system. We saw this with the Linux project, which channeled the developer resentment of the proprietary Windows operating system software into the development of the best server operating system to date, which is Linux.

The difference between Microsoft Windows in the 90s and AWS today is that, for the most part, developers do not resent AWS. AWS keeps its prices low and embodies the spirit of innovation despite the fact that AWS is partly built around a repeated process of taking open source projects, packaging them into cloud services and integrating them closely with other AWS tools, making it harder to move away from AWS.

In its pioneering offerings of managed services, AWS made it easier to set up tools that had previously been impossible for less experienced developers to operate. This included distributed systems tools that now became approachable. Companies like Netflix and Lyft and Heroku were built on the simplified operational model of AWS. Amazon Web Services also innovates outside of this business model of repackaging open source projects.

When AWS pioneered the function as a service model, they created an easy way to scale stateless computation. They presented the developer world with an entirely new computer model, event-driven programming with services communicating to each other via functional glue code running in cheap, transient containers.

There is strong criticism of the event-driven AWS Lambda-bound serverless compute model, and the critics of Amazon to have a valid point. AWS Lambda is a proprietary API. To build your app around an event-driven architecture glued together with Lambda functions is to lock yourself tightly to Amazon infrastructure, but the critics of this event-driven Lambda bound model did not seem to be the ones who are actually “locked-in”. Critics of Amazon's proprietary strategy tend to be those with a strong incentive to be critical.

Another common criticism of AWS comes from commercial open source companies, such as Redis Labs and Elastic, which are vendors of the Redis open source in-memory data system and the open source Elasticsearch search and retrieval system, respectively. These vendors argue that AWS has violated the spirit of open source by repackaging open source software as profitable software and failing to contribute back to the open source ecosystem with commensurate open source contributions themselves.

AWS has repackaged both Redis and Elasticsearch as AWS ElasticCashe and Amazon Elasticsearch Service, respectively. These vendors frame their relationship to AWS as zero-sum, and they're turning to licensing changes as their strategy of choice for creating a new defensible moat against AWS.

In various internet forums, the indignant commercial open source software vendors and AWS have both made their cases to the developer community. If you're developer who just wants to build your software, these arguments are an unsettling distraction. In addition to worrying about fault tolerance and durability guarantees and cost management, you now have to worry about licensing? As you observe the circumstances with open source vendors accusing cloud providers of improper behavior, you might begin to wonder, "What actually are the rules of open source? By what set of commandments are we judging who is good and who is bad, and who is to judge what is fair or unfair activity by Amazon, which was the first company to prove to all of us just how influential cloud computing could be."

This is the competitive landscape of the cloud, circa April 2019, when I attended Google Cloud Next, Google's annual cloud developer conference. Throughout the conference, the optimism and playful spirit of Google was present everywhere. There were colorful shapes and cartoonish diagrams that looked like building blocks made for children. In the Expo Hall, there were food pillars from which free candy and salted snacks could be grabbed all day long. On one side of the Expo Hall, a demonstration from IKEA showed how the company was partnering with Google to run its furniture business more like a software company. Stretching across the Expo Hall in a carpeted distance of multiple football fields, the various vendors in the room showed off their latest wares in a bazaar of multi-cloud, hybrid cloud and Kubernetes compliant sales pitches.

Over the last decade, large enterprises have opened up their wallets more and more realizing just how valuable cloud infrastructure can be for their businesses, and the more these enterprises spend, the more efficient their workers become and the faster their businesses can improve. For modern enterprise, the cloud is the closest thing to magic that we have ever invented.

Knowledge workers are becoming unshackled from the painful, slow pace of early technological corporatism. The cloud can empower all of us, and rather than replacing our jobs with machines, the cloud will allow us to better express our humanity within our work. The twin forces of consumer mobile computing on the client side and cloud computing on the server side are compounding faster than we can measure.

Five years ago, a common analogy was to compare the smartphone to a “remote control for your life”, as investors like to call it. Today, the smartphone feels more like a magic wand than a remote control. We can summon tremendous forces from the magical cloud by merely speaking the right command into our wand-like smartphone. Someday, even the world of Harry Potter may look antiquated relative to our reality. Why carry around a wand when you can use your airpods to cast your spells via hands-free voice command. The cloud’s magical effects on corporate enterprise operations are exciting to watch. I go to a lot of conferences, and I prefer to walk around the Expo Hall rather than to go to any sessions. At the Expo Hall, you see the distance between the vendor hype and the realities of the enterprise.

From my vantage point, that distance between hype and reality actually gets shorter every year. Enterprises truly are adopting devops, and continuous delivery, and machine learning, and data platforms, and service mesh, from insurance companies, to banks, to farming companies, to furniture outlets. The digital transformation is actually happening, and this is great for consumers. It's great for employees at the large companies that are being digitally transformed as well.

What is it like to work at a large 100-year-old enterprise that is going through digital transformation? Well, it's certainly much more appealing than it was a decade ago. The servile, bureaucratic, rent-seeking hierarchies of the 80s, 90s and early 2000s are slowly being refurbished into enterprises where bottoms up innovation, individualism, and artistic energy, are assets rather than liabilities.

We are moving out of the age of boring cubicle industrialism towards a vision in which work is truly creative and fulfilling for every member of the corporation, and Google evangelizes this ideal stronger than any other company in the world. With the tremendous cash flows from its advertising business, Google reinvests heavily into its employee base. Many employees at Google have worked there for a decade and showed no intention of leaving. Having found a job, but more importantly a culture and an engineering environment that is unrivaled.

Speaking subjectively, my sense is that Google's internal engineering tools are the best in the world. For many years, Google has been the strongest magnet for talent in the realm of internet infrastructure and machine learning. In terms of sophistication, the rest of the software industry has been playing catch-up to Google since the days of the MapReduce paper.

Speaking of the MapReduce paper, even back in 2004, Google was willing to share its findings with the outside world. Google was tactical about which innovations it would open up about, but it does seem that Google has truly tred to embody some of the publishing philosophies of academia. Google's early investments in an academic-like environment have borne considerable fruit. The curiosity and cross-institutional collaboration enabled by Google's willingness to speak openly about its research have made Google a refuge for academics, including today's guest, Eric Brewer.

Beyond its considerable contributions as a publisher of computer science theory, Google has also become the model software engineering practitioner. Google has scaled its internal tools to make its thousands of developers highly productive. Given its success with internal developers, it should come as no surprise that Google has strong opinions about the best way to build services for external developers in the cloud.

[SPONSOR MESSAGE]

[00:11:52] JM: A thank you to our sponsor, Datadog, a cloud monitoring platform bringing full visibility to dynamic infrastructure and applications. Create beautiful dashboards, set powerful machine learning- based alerts and collaborate with your team to resolve performance issues. You can start a free trial today and get a free t-shirt from Datadog by going to softwareengineeringdaily.com/datadog.

Datadog integrates seamlessly with more than 200 technologies, including Google Cloud Platform, AWS, Docker, PagerDuty and Slack. With fast installation and setup, plus APIs and open source libraries for custom instrumentation, Datadog makes it easy for teams to monitor every layer of their stack in one place.

But don't take our word for it, you can start a free trial today and Datadog will send you a free t-shirt. Visit softwareengineeringadily.com/datadog to get started. Thank you to data dog.

[INTERVIEW CONTINUED]

[00:13:00] JM: I have often wondered what it is actually like to work within Google as an engineer. I've worked inside of Amazon and I've seen their internal tooling. At Amazon, the level of tooling gave engineers tremendous leverage. But from talking to more experienced engineers, my sense is that nobody holds a candle to the internal tools of Google.

Drishti's CTO, Khris Chaudhury, was a recent guest on this podcast. He spent nearly a decade at Google working on computer vision projects, and today he's building his own computer vision company. When I asked him if we yet had Google infrastructure for everyone, which is a saying, he sighed deeply and wistfully before answering with an unequivocal, "No. We do not yet have Google infrastructure for everyone," and this hints at what Google wants to do with its cloud. Google does not think of Google Cloud as commodity cloud computing services. The vision for Google Cloud is to be the premier cloud, the deluxe set of services and hardware that provides Google infrastructure for everyone. This is a moonshot, and in order to accomplish it, Google will need to forgo certain short-term opportunities for cache grabs, but it is undoubtedly a more fiscally-wise strategy for Google to optimize growth of the 2029 Q3 cloud market rather than that of 2019 Q3.

From a conglomerate standpoint, Google already has a cache cow. Google has one today's war, and it only makes sense to focus on tomorrow's. In Google's approach to the cloud, we see a cultural distinction between Amazon and Google. The cultures of Amazon and Google are partly deliberately different, but partly driven by the nature of their business's respective revenue streams.

Amazon built its e-commerce business in a low-margin, highly competitive environment. Amazon won its customers over through a slow process of building trust. In its delivery of physical goods, Amazon formed close relationships with customers. Amazon would repeatedly listen to these customers and use that feedback to improve their products. This flywheel of iterative improvement is the core engine that keeps Amazon improving incrementally. Amazon has also made far-flung ambitious investments, but the size and scope of Amazon's moonshots were constrained for many years by its lack of any singularly large cache cow.

Fire Phone notwithstanding, Amazon's moonshots have often been thrifty, asymmetric bets, requiring minimal upfront investment, but presenting huge potential payoff. Google, on the other hand, has been in a much more luxurious financial position for most of its life. Google has a high-margin advertising business that accounts for 84% of its revenues, subsidizing everything else in Google, and Alphabet also. Because it has such a big cache cow in a totally different area of the business, Google can afford to take an extremely long term approach to its vision for the cloud.

For Google, the goal is not to maximize the profit margins of the cloud over the next two years. Google can afford to think of cloud profitability in the increment of decades. In the business of cloud computing, Google has turned the weakness of being a late mover into a wide set of strengths. The AWS Console presents its users with the sprawling array of possibilities. Google Cloud has a lower surface area than AWS. Google is more opinionated about the right way to do things, and today, it's easier for Google to build in an opinionated fashion, because there are fewer legacy customers and edge cases to support.

AWS supports the majority of the market by far. So it is in a position where it must keep those customers happy in order to hold on to its moat. So, what are Google's strong opinions about the way that a cloud should operate? Google's espoused division is that of the “open cloud”, a cloud environment where organizations could easily move workloads from one cloud provider to another. If we take the purest, most aspirational interpretation of “open cloud” the full stack would be open source. Identity and access management systems would be portable as well, and cloud providers would work together to reduce the switching costs between each other even in cases of data gravity.

As virtuous as this idea of the open cloud sounds, it is also strategically convenient for Google. Since it lags behind Amazon and Microsoft in terms of cloud adoption, a gradual shift towards a widely standardized open cloud would theoretically make it easier for Google to recover market share as the cloud market matures.

Whatever Google's true motives are, the “open cloud” strategy has been tremendously bountiful to the developer community. Let's start with Kubernetes. By open sourcing Kubernetes Eddie's and pouring resources into it, Google brought an end to the painful, wasteful container orchestration wars. In its donation of Kubernetes to the Cloud Native Computing Foundation, which Google is also a heavy financial donor to, Google created an ostensibly open, positive-sum environment for the rivaling cloud providers to congregate productively.

In the area of machine learning, Google open sourced TensorFlow and invested heavily into tutorials, documentation, YouTube videos and other resources of content marketing. Those are practical resources as well. I am using content marketing a little bit cynically, I suppose. Google also built JavaScript libraries, and auditability, and visualization tools, and really important things that have helped the TensorFlow ecosystem. Google truly has marshaled an entire ecosystem around TensorFlow.

Some of Google's commercial open source efforts have had less favorable results. The Istio service mesh project seems to have been promoted with the same playbook that Kubernetes and TensorFlow followed, but with a less usable tool. In Istio, we see Google's expertise in marketing perhaps taken too far. Why is Istio a problematic case study? Because despite the fact that there are multiple open source service mesh products on the market with significant production usage, Istio has managed and makes so much noise about itself that is convincing the market that the battle for service mesh superiority is a foregone conclusion.

In spite of widespread reports, at least from what I've read on Twitter, that Istio is currently difficult to operate and not ready for production workloads. This is in contrast to things like Linkerd, Kong, Console from HashiCorp. Perhaps NGINX, if you consider that a service mesh. From the blog posts, to the KubeCon programming, to the Expo Hall hype, the cloud native developer community has been hammered with the same messaging about service mesh, “Istio is the best. Don't bother with the rest.”

Perhaps Google is doing us all a favor with Istio. Maybe it really will be the best service mesh. It's just not quite there yet. Maybe Google is just ironing out the kinks and the marketing roadmap happened to proceed at a faster pace than the engineering roadmap, or maybe I’ve have been talking too much to the other service mesh companies and I'm totally biased. Honestly, I don't really know how these meshes trade off against each other, though I have certainly asked previous guests for some comparisons, and you can listen to those episodes too. Go into those in more detail.

It's so early in a huge space, this huge space that's been bucketed within the category of "service mesh". Maybe Google is actually acting with best intentions and trying to just get out ahead of another container orchestration war. Service mesh does, in a certain light, look like yet another container orchestration war waiting to happen.

Speaking as a developer who prefers to work on application abstractions far above the world of security policy, and load-balancing, and canarying, I have no interesting dealing with any of these things. I would personally be fine with Google running its open source marketing playbook with Istio, winning everyone over and saving our energy for more productive deliberations. Honestly, I am fine with Google picking winners where it deems such actions appropriate. But this does get at the tension of the "open cloud". What kinds of openness are we really talking about here? If I cannot see the strategic roadmap of Google Cloud, if I can't hear the backroom conversations, is it still open? When the open source repositories remain in Google's control, is that open source or is it marketing?

When Google tinkers with an operating system like Fuchsia in the open, but we are left to speculate as to why they are working on Fuchsia, or what purpose it serves, is that open source or is it marketing? If Google is such an open cloud, why not open source a detailed 10-year strategic roadmap for the cloud and pass the buck to Amazon to do the same?

In any case, I am not passing judgment on whether Google has done something morally wrong by pushing Istio so hard. I just think it was strategically unwise, and frankly a little bit hilarious. With Istio, Google made it a little too obvious just how much narrative control it has over the public developer market, and perhaps this narrative control should come as no surprise. Of all the major clouds, Google is the most well-versed in open source. From its contributions to Linux, to its maintenance of a complicated android ecosystem, Google knows how to play its cards in the game of open source diplomacy.

In battle, a classic strategy for competing with a rival that has an advantage is to force that rival on to territory that you are more familiar with. There are many historical cases where a small army was able to defeat a large army, because the small army was able to maneuver the larger army on to territory that the larger army is less favorable within.

So, through its open cloud strategy, this is exactly what Google is doing. Google is open sourcing the best way that it knows how to build and run infrastructure software. This is happening slowly but steadily.

To some extent, the other major providers, including Amazon, will have no choice but to follow Google on to a battleground that was built by Google. As developers, this is pretty sweet. We will get to reap all the rewards of this competition. Our infrastructure will become more standardized, more fault-tolerant, cheaper, better design and easier to use. Who knows, maybe someday we will actually be able to easily move workloads from one cloud to another.

To the extent that I am a software engineering journalist or a podcaster, I don't really know what I am, I do feel inclined to scrutinize all of the cloud providers. But to the extent that I'm an engineer and a business person, I feel only admiration and love for the cloud providers. Cloud computing has brought the cost of starting an internet business down to zero. Cloud computing has opened up my eyes to a world of creative possibilities that knows no boundaries, and for that I will always be a fan of all of the rivaling cloud companies, because they all have played a role in creating the current software landscape.

Today's guest, Eric Brewer, is a Google fellow and VP of infrastructure. He's well known for his work on the CAP theorem, a distributed systems concept that formalized the tradeoffs between consistency, availability and partition tolerance in a distributed system.

For more information about Eric and the CAP theorem, I recommend checking out a past episode. There might be two episodes about the CAP theorem on Software Engineering Radio, where Eric Brewer was aghast. That is truly one of my favorite podcast episodes of all time. I'm a huge fan of Eric. At Google, Eric is as much a strategist and a product creator as he is a theoretician. He has worked on database systems such as Spanner, machine learning system such as TensorFlow, and container orchestration system such as Kubernetes and GKE. Eric joins the show to talk about Google's philosophy as a cloud provider and how his understanding of distributed systems has evolved since joining the company.

A few quick updates in Software Engineering Daily land before we get started. FindCollabs is a company I started recently, and we now have a podcast. The FindCollabs Podcast is out, and that the links to these are in the show notes. Also, FindCollabs is hiring a React developer. If you're an excellent developer and you know how to program React and you're really good at it, please send me an email. Details are also for that in the show notes and everything else I'm going to mention.

The FindColalbs hackathon has ended. That detail is in the show notes. We're booking sponsorships for Software Engineering Daily for Q3. So if you're looking to reach developers, feel free to go to softwareengineeringdaily.com/sponsor and learn more about our opportunities.

Podsheets is our open source set of tools for managing podcast and podcast businesses. We're looking for people who want to contribute to that open source project. A new version of Software Daily is out. That's our app and ad-free subscription service. We're still working on some updates to the mobile apps, but the open source project is coming along.

With that, let's get on with today's show.

[SPONSOR MESSAGE]

[00:27:50] JM: The 2019 Velocity program in San Jose (June 10-13) will cover everything from Kubernetes and site reliability engineering to observability and performance to give you a SED 818 Transcript © 2019 Software Engineering Daily 2 comprehensive understanding of applications and services—and stay on top of the rapidly changing cloud landscape. get 20% off of most passes to Velocity when you use code "SE20" during registration at velocityconf.com/sedaily

[INTERVIEW]

[00:30:33] JM: Eric Brewer, welcome to welcome to Software Engineering Daily.

[00:30:35] EB: Thank you so much.

[00:30:37] JM: I was initially going to ask you about the CAP theorem, of course, but I was in some discussions earlier this week with people who work on products, and they told me that you might think that Eric Brewer just sits around Google writing on chalkboards and describing theorems all day long, but actually he works on product. He's very focused on product.

One of these people was somebody who's working on open source container platforms. Another of these people was working on distributed streaming systems. So these are diverse distributed systems problems. How have the applications of distributed systems that you've seen at Google – How have those changed your CAP beliefs?

[00:31:21] EB: I would say the most obvious one, which I written some about, is how the Spanner build a global database that's very highly available, and there's a complicated answer to that, but the short version is it still follows the CAP theorem. It's a CP system, meaning consistent, but not available under partitions. But Google has done a lot of work, none of which had anything to do with on how to make the network super reliable through multiple PaaS, through changes in how to do configuration management. That took five years, basically. But now that is really around five and half, six nines, and so the overall availability of Spanner is actually higher than many systems that would say that they're highly available on CAP spectrum.

[00:32:04] JM: I've heard you speak about the importance of invariants in improving distributed systems. What's the relationship between invariants and distributed systems?

[00:32:14] EB: I kind of feel like you can't build complicated systems unless you have some number of simple invariants. It's one of the things you can hang your hat on when things go badly. So, for example, Spanner hangs its hat on the fact that if we do get a partition, we really can prove to ourselves that whatever is on one side of the partition is consistent. They may not have a quorum. It may have a quorum, but you can really say, "We know what happens in that case. It's pretty simple. We know that orders are applied, updates are applied in a particular order that matches real-time." That's a great invariant. When you know that invariant is true, it rules out a whole bunch of weird behavior. That's really the purpose of true time for Spanner.

There're lots of these things. I really feel like even the way Kubernetes works, I would say a big change that people don't understand is its model of configuration is you declaratively specify what it is you want. The underlying system has to make that true. We call that reconciliation over time, and that's one kind of generalized version of an invariant. It's a policy, the YAML file that you are giving it for, say, creating a collection of pods, that is, it's not exactly an invariant, but meeting the same goal of being very clear and what the intended state is.

Classically [inaudible 00:33:30] systems, we don't actually know when we have an error. Is the system in the intended state or not? Because there's no separation of the intended state from the actual state. It turns out having an explicit intended state greatly simplifies a bunch of things, because now you could say, "Oh! It has four pods right now, but we know it's supposed to have five." The four is unintended lost, rather than someone intentionally deleted a node using some your CRUD API or something like that. So a lot of it comes down to intent and declared specification for the new version that they were building for Istio and Kubernetes. This is a part of custom research definitions.

[00:34:08] JM: Do you use formal verification in your work at all?

[00:34:12] EB: I have looked at it. I do think it's a powerful tools. It's not one that I use personally, because I feel like it's a kind of specialized tool that you need a fair amount of experience to use well. Certainly, I've have taught classes on it and I feel like it's an important technique, and we are using plenty of it in Google in small ways. Certainly, things in the kind of compilers and languages area use lots of techniques like that to find bugs, prevent bugs through static analysis. We do a lot of that kind of stuff, but kind of model checking or other kinds of proof systems. I would say that still has a ways to go in terms of the techniques we use internally so far.

[00:34:48] JM: Now, you would think if anybody would need to use formal verification, it would be a cloud provider. So what are the substitutes for formal verification or proof systems? There must be some substitute if you need to build a bulletproof system.

[00:35:05] EB: I think the substitute really is to figure out how to do evolution with high-availability. That's really the name of the game, because if you can evolve a system quickly –

And this is really honestly the raison d'être for Kubernetes in my mind going back to its very beginning. It's good at evolving the system quickly.

So why this works is basically – So suppose you have a product that's evolving quickly, which is very likely in the cloud space, the problem with that for formal methods is you have to reprove that it's correct for every evolution of the product, which means every quarter. That's an unrealistic goal.

So the alternative approach is really make sure that when you make a change, you have a way to deploy it quickly in a small way, what we call a limited blast radius, meaning it can do limited damage and actually test it with live traffic and roll it back quickly if it doesn't work. So I would say our approach is much more about, "Let's do lots and lots of releases in a safe way with progressive rollout," meaning, meeting small nodes first, eventually to the whole fleet. A typical rollout might take four days for a big change at Google to spread out the risk and make sure we can always roll back to a safe place. That methodology allows us to have both high-availability with some detection of problems, but also have a high rate of evolution, which is what we're actually after.

[00:36:31] JM: In this world of cloud and open source software, we have these two trends; cloud and open source software, that are playing off of each other in a mutually beneficial compounding ways, and this has been incredible to watch as a developer, because it's led to so many tools that are highly useful to me and they seem to be released at a faster pace every single year.

It does create some confusion for the market. People don't know when to choose an open source tool. They don't know when to choose a proprietary tool. They don't know which cloud provider to go with, or which bundle of cloud providers to go with. So there's this fog of war right now. As a cloud provider in this fog of war, is it better to have a clear strategy, or is it better to move fluidly with the whims of the market?

[00:37:28] EB: Interesting question. I would say it's important to have a clear strategy that fits well with the long-term trends, because those aren't going to be changed by the whims of the market. I would say open source is a good example of the. That's certainly a place I've been deeply involved with Google. I was involved in the open sourcing of Kubernetes, which was a pretty controversial thing internally at the time.

The bet there is pretty simple. It’s that we really would like the whole industry to move to what I call the new cloud, by which I mean the cloud that's based on containers and APIs and services rather than on VMs and physical disks. That new cloud has much more leverage for developers. It's more productive, but that's a big change. To do that change, Google can't do it by itself. So a lot of the open sourcing strategy was really about we can lead this change, we can help direct it, but if we don't get a lot of friends on board, it’s not going to happen.

Making it open source means that it can work on other clouds. It can work on-prem. People can customize it for their systems, even in other countries, in other languages. There’s a whole bunch of things that happen that we would never get to. So I would say open source, and what I call the open cloud, is a core Google value for this space, and I feel like it fits naturally with our kind of culture and also with our belief about how this is going to work out in the long-term.

[00:38:51] JM: That determinism of having a vision for what the cloud should look like, that can come at odds with the dynamics of the open source community. In what ways have you seen that conflict manifest?

[00:39:10] EB: Well, without naming names, there’s quite a collection of open source projects whose direction I don't believe in. Sometimes I give them feedback. Sometimes not, and occasionally I’m wrong. But I do feel like it's good that they exist and some self-correct and some don't. But, really, what matters is that there're thousand experiments going on and some of them will be great, and those are ones that we can join with. For example, Docker was one that I noticed early on and I said, “This is good for us, for the vision that we want to move to cloud to.” So I met early on with Docker and said, “This is something we want to leverage. We’re happy work with you on how to do that. There are parts that we care about that you're not currently working on,” which ended up being orchestration in Kubernetes. This is before Kubernetes existed.”

It’s not like there weren’t other ways to do packaging of software, but they had a good – They got a few things right and they had some good momentum and I felt like by helping them out a little bit or endorsing them, that would move the industry in the right general direction. As long as there are some good open source companies and there are plenty, that's all we need as an industry.

[00:40:21] JM: When Google is in a situation like that when you're a fan or endorser of the project, it can be a very harmonious relationship. But when there's perhaps a product whose vision you don't agree with or you don't believe in, how do you calibrate your measure of diplomacy when you're dealing with that project, because it's an open source project and Google has been such an immense contributor value to the open source community and you want to contribute value in the right direction, but you don't want to be seen as throwing your weight around improperly.

[00:41:00] EB: Yes. We call that picking a winner. We don't want to pick a winner in general, rather the customer base do that. So I'll never typically publicly say something negative about an open source project I don't like, but I will sometimes give feedback to that team or at least explain what concerns me about it. I'm trying to think if ii can give you an example I can tell. It's not easy without gotten permission from those groups ahead of time. I think there are some that wouldn't mind, because they did make some changes based on my advice. But it can be things like mistakes we've made.

Here's a good example, one that isn't tied to any particular system. But I mentioned earlier that when we're doing configuration in the Kubernetes new way, it's really a declarative specification that is then interpreted and executed. Pretty much, the previous kinds of solutions that were all variations on scripting languages where you have some script that configures something and it's in a fairly general-purpose language, and Google has done this many times internally. We had configuration systems based on Python, and custom languages, and all kinds of things. The problem with those is that those are – General purpose languages, they aren't actually too powerful. It allows you to write configuration code that is – Especially when it's dynamically typed like Python is, that can actually fail at runtime in production in ways you can't predict ahead of time, and people abuse it because it can generate very complicated config systems because you can run a program in the middle of your config part.

So I'm saying, “That’s not really what you want to do. It's too risky.” So we’ve been moving away from that internally, but I have told groups like Terraform, or Chef, or Puppet, who have essentially in the inside essentially a scripting model that the reason I worry about that is exactly that it's easily subject to abuse.

Terraform is boot in the direction of trying to be more mutable and more infrastructure, but it's basically still batch underneath and you can abuse it if you don't use it correctly. So that one I think is in the middle, which is generally supportive for because it’s going in the right direction of much more of an immutable specification of infrastructure. But that's a complicated topic, and again, Google didn't get it right either the first several attempts. I think we’ve been working on config languages for at least 15 years, and most of them had been bad.

[00:43:25] JM: When you started Istio, or when you open source is Istio, what were the subjective design decisions that were being made in other open source projects that are focused on service mesh which you disagreed with?

[00:43:44] EB: I don't think it came about that way. It's more like we had a collection of tools that allows us to have services in large numbers. I don't know the current number, but last summer I checked, which is several years ago, Google had more than 10,000 services running all the time, which is a number that doesn't make sense to me. That’s something – At the time it was like three engineers per service on average, which is a ridiculous ratio. So it's probably 20 or 100,000 thousand services now.

But when you have a lot of services, lots of microservices or even big services, you need some way to have – You need a service framework that gives you some leverage on how to do monitoring, how to do authentication, how to do quota. So we had built internally a bunch of platforms that give us this framework that make it such that when you're running a new service, you really don't worry about anything except what are the actual handlers do for the API. You don't worry about the telemetry, or the security, or the axis control, or the quota. All that is handled by the service framework.

So Istio is really a vision of rewriting that set of tools in an open source way, because that, it has a kind of a broad mandate, almost too broad, at least for how it started. So I’d say it's broader than the other open source projects that are many that tackle particular things, like access control, or indemnity management. But it was trying to be a more holistic view of what do you need to manage a whole bunch of services.

In fact, we're trying to make it more modular so that it's easier to kind of, ala cartete, take the pieces of Istio and replace them with other pieces that could come from other groups or other open source ideas as a way to get a better long-term foundation for the whole space.

[00:45:30] JM: How ready for production is Istio?

[00:45:33] EB: That's a good question. I would say I don't really know. I certainly know though 1.1 version has fixed a lot of issues, and I believe people are using it in production. But I don't – It's hard for me to know exactly how well that's going. It's definitely much better than it was six months ago, and I feel like this release is something that people are giving serious effort to. There are definitely groups using the production now. So I feel like it is capable. What I don't know is how often it's going to get updated. Do we have major changes we need to make that will have none backward compatible changes. It's still a .5 version. Sorry. It's 1.1. [inaudible 00:46:14]. I'm thinking of something else as .5. It'd come to me later. But it's still early in that space.

So I think we're getting feedback from the community and making some changes based on that, and that will make it better for sure. Most of that will come this year. It's not like it's super far out. But I do feel like we're still in the let's explore this space as a community and help people understand why they need it and what it needs to be more broadly.

[00:46:42] JM: I was at KubeCon before Istio came out, and there is a presentation from a bank, Manzo Bank, and they were using Linkerd in production. So they were using a different service mesh tool. The whole presentation was about how there had been a bug in Linkerd, but Linkerd altogether was a phenomenal project for them and it was working quite well for them in production. After Istio was open sourced and there started to be a lot of presentations around Istio, I started to ask people who is using Istio in production. There was much less use of Istio in production relative to the number of talks and the amount of promotion around Istio that I was seeing.

At the same time, you would see the fading number of talks about Linkerd. There was a sense that a winner had been picked in the service mesh space. Do you think the amount of weight that Istio was launched with was perhaps too much?

[00:47:56] EB: Too much. It's hard to say. I feel like people were legitimately excited about it. So I think that enthusiasm was good in general. In fact, if I take my Google hat off for a minute, it’s more important that the industry have a functioning service mesh and service framework than [inaudible 00:48:14] Istio per se, although I do think Istio represents a lot of useful pieces that deserve to be widely adapted. But I would say we have to converge on what it is we want out of our service mesh. That’s why I think the modularity is going to be important in general direction, so that you can make choices about telemetry independently of how you do authentication and quota management. That is possible. It was kind of all put in together in the first versions, because that was the vision of think. I think that's fine.

One of the big benefits and success stories for Kubernetes has been vetted as relatively extensible and people can extend it at multiple points simultaneously without having to worry about each other. So that has allowed a proliferation of experimentation, and there’re 20 functions as a service groups building functions as a service on top of Kubernetes. There're lots of groups looking at node management and key management and all that is super healthy. That's the best path to great solutions overtime.

So I kind of feel like I'm seeing that dynamic in Istio and there are startups around it. There is enough entering groups, including customers and such, working on it for real that it's revolution is high and robust. By robust, meaning that for everything you want to do, there’s more than one thing being tried to do it, and that's good.

[00:49:42] JM: I mean, as a developer, I actually want winners chosen for me as quickly as possible, frankly. I mean, I was reporting during the container orchestration wars and it was a total disaster. There were so many investments in core infrastructure that ended up having to be torn out and rewritten and it was kind of a disaster. I think I wasn't as closely focused on this at the time, but my sense is that the same has been kind of true in the machine learning framework space, and then Google came out with TensorFlow and TensorFlow was not as performant to some other machine learning systems at the time, but it doesn't matter, because most people don't actually need super high quality performant machine learning systems. What they need is really good documentation, really good onboarding, really good tutorials for that.

How do you choose what domains to focus your open source efforts on?

[00:50:43] EB: I almost think that Google are of a national bottom-up nature in the sense that if there are champions the believe in a particular project direction. It isn't going to go anywhere so you recoup Nettie's I wasn't one of the first to broaden code by any means, but I feel like I was the first executive to understand why we needed to bet on it and why you need to be open-source and so I can for my job is to encourage great ideas to bubble up from engineers and others and and help guide those and then really when we start to see some momentum. Make sure we understand how this fits in the long-term and certainly tensor flow is something that you know Jeff, Dean and I discussed the time of really honestly is a it's related all things to big table connection there is the Jeff worked on big table. Release the paper but not the source code. The consequence of that was that people be built.

You know, Cassondra and H basin various things that were big table implementations and open-source based on the paper but they didn't exactly match big table APIs. For example, because we didn't actually tell people what the guys were they made up their own, which is fine and good for them. The problem with that is, many years later. Google class is over. We should have a a big table service but we actually have to. We can't use the table APR for that because no one uses the big table APIs that use the eight space PPI so that the big table equivalent on Google cloud today is really the a space API backed by the actual big table so the learning from that was had.

We've actually done more open sourcing of the original big table. We would actually do not have a whole separate version development open-source that actually conflicted and large or small ways with what we were doing so tensor flow is awaited to say let's actually open-source. The key components of early and make sure the open-source community has loose the chance to build around those things and that's even more important for tensor flow because we want. We were new we were going to build specialized hardware to execute so that combination of an open-source bed around things that we understand. Plus customize hardware goes to fit really well together, but if the open-source had a different plan for machine learning a different set of APIs are her room would help

[SPONSOR MESSAGE]

[00:53:13] JM: As a software engineer, chances are you've crossed paths with MongoDB at some point. Whether you're building an app for millions of users or just figuring out a side business. As the most popular non-relational database, MongoDB is intuitive and incredibly easy for development teams to use.

Now with MongoDB Atlas, you can take advantage of MongoDB's flexible document data model as a fully automated cloud service. MongoDB Atlas handles all of the costly database operations and administration tasks that you'd rather not spend time on, like security, and high availability, and data recovery, and monitoring, and elastic scaling.

Try MongoDB Atlas today for free by going to mongodb.com/se to learn more. Go to mongodb.com/se and you can learn more about MongoDB Atlas as well as support Software Engineering Daily by checking out the new MongoDB Atlas serverless solution for MongoDB. That's mongodb.com/se.

Thank you to MongoDB for being a sponsor.

[INTERVIEW CONTINUED]

[00:54:36] JM: There is a Software Engineering Radio interview you did about four years ago with Robert Blumen, and Robert is a mentor of mine. That interview is actually one of my favorite podcasts of all time. I've listened to it several times and it's very durable contest. I was actually surprised at how durable it was.

[00:54:54] EB: I'll have to re-listen to it.

[00:54:55] JM: It's about distributed systems. So durable is not u pun that is intended. In that interview you said that in 10 years we will see a lull in distributed systems. But presumably, that was four years ago. So in six years, if you take that 10-year statement, we'll be coping with things like machine learning interpretability. That's kind of a distributed systems problem. You've got the self-driving thing. That seems like a pretty tough distributed systems problem. Do you still think that we're going to see a lull in six years?

[00:55:27] EB: What I really meant in that statement, assuming I can pull it out correctly, was –

[00:55:33] JM: What do you mean you don't remember exactly what you said four years ago in a podcast interview?

[00:55:40] EB: But I've said some things like that. So, assuming it was like that. What I was saying was that a consequence. The main consequence actually of the CAP theorem has people felt encouraged to explore the full space of all the options between availability and consistency, and that's super healthy, because in the year 2000, we were really worried about kind of just two versions. The internet, which is highly available; and databases, which were consistent, and the two camps I think [inaudible 00:56:15] didn't even know how they were related, and were very biased in their beliefs that their approach was the correct approach.

So starting around 2005, which is surprisingly later than the CAP theorem, which is '98, '97 or so, groups like HP and Amazon and others started actually building systems thinking about it this way. That led to many, many things. This is where DynamoDB and Cassandra and React and many things came – I kind of feel like what I'm saying is that exploration of all these options is a transient thing, and we'll find good solutions that have the right combinations in this 10-year period, now six years left.

The full space will be well covered. There'll good systems that have a range of tradeoffs, but there won't be like new system started every year. There won't be like people trying to understand new aspect or new ways to do it in the same way. So it's kind of a little Cambrian explosion of database management systems and that will lead to good systems. Once we have a good set of systems for all the use cases, that exploration will roughly slow down.

[00:57:34] JM: What innovations will we see in distributed systems coming out of the self-driving car set of innovations we'll see?

[00:57:42] EB: I'm skeptical we'll see that many, because the car can't really depend much on any information outside its immediate domain other than may be maps, which are essentially read-only for the purposes of a car. So they need lot of sensor data. You need a lot of computing. The question is what do you need that's not in the car to do a good job? I don't think there's that much. You might like to know traffic information for routing, but that's not really about the self-driving thing. In fact, human drivers would like to have the same information and kind of do.

[00:58:15] JM: But presumably, there will be some length of time in which – So like the model of we're going to put a big server and a car and drive around and it's going to be as performant as possible, and hopefully we can squeeze everything that we need into that amount of compute that we can fit on the car that's driving around.

Presumably, that has some kind of limitation. There will be some advantage to network request that that car would be making that would be maybe have – Maybe they're not completely safety critical, but they have different varying degrees of functionality. Presumably, you don't think that's a problem at all? Maybe you need some hierarchical system of like what kinds of systems you need to have access to?

[00:59:00] EB: I think there's – I mean, I can name things that would be valuable to have at least some hierarchy, but you can't depend on network in general. It's nothing the car can do that really requires the network, because there're too many places that don't actually have a network. Who did depend on the network and you lost it unexpectedly, what is the risk of that and how do you test it? Unclear.

So I kind of feel like the information you can benefit that's remote is really stuff about, and I would say there's two cases there. There's nearby stuff, which is it might be really nice to coordinate with all the cars nearby. That's not much information, but it's some, and it would help. But you're still going to use your sensors for ground truth. But it'd be nice if you could agree on, "I'll go this speed," and actually coordinate our speed, coordinator our lane changes. Lane changes are one of things that are very hard for a self-driving car, at least when you have human drivers also, which is going to be the case for a long time.

So I do feel there're things you can use at least local network data for that are interesting. That's mostly coordinating with the cars around you. Connecting to kind of the proper cloud is I think mostly going to be more about telemetry, routing, traffic flow, things that are not actually that high bandwidth and not actually critical to driving. When you're in rural Montana without connectivity, that'll be fine. You don't have much traffic to worry about anyway.

[01:00:30] JM: How has your beliefs about distributed systems changed in the years since the Bitcoin white paper?

[01:00:39] EB: That's a good question. I would say a few things. I think there're lots of applications for a ledger that can't be modified. But I actually don't believe that currency is one of them, and that’s a more complicated discussion. But it has two things that strike me as odd about it. One is that it has the history of all transactions, which is an anti-privacy thing that I think is pretty fundamental. So people think of it as being private, but in fact it's actually recording all the transactions. If you think someone did something bad, you can actually see who they traded with and what the currency was eventually used for by other people. That chain is all in the – All that data is in the blocking. So that bothers me as a kind of foundational piece.

But the other one that bothers me is that it actually does require connectivity of the majority of players. So it doesn't actually play very well with the CAP theorem, right? It means, in particular, that if you partition your majority, you will actually not be able to advance the watching.

[01:01:39] JM: But that's okay until the partition reconnects. If you have some kind of vulnerability that occurs while you have the partition, if somebody believes there's a vulnerability that’s occurred, they could just fork the blockchain at that point.

[01:01:52] EB: If there's enough people on their side, I guess they could fork the majority and get it to stick, but I would think I could do a denial of service attack by continually partitioning the blockchain from key players and actually stalling progress. They are already relatively slow at updates even without that.

[01:02:13] JM: So, do you see of a way to build a decentralized trusted ledger without a currency?

[01:02:20] JM: I think I have not thought about it deeply, but I do feel like, yes. The real issue is, is it just based on something like proof of work or is anybody complaining – The normal way I would design to the systems is actually know who the players are. Even if I don't know their real identities, having of a fixed number of players makes the game much easier. This is really anyone can play, and that means you have a tax-based on people working together, collusion. You see some of that now, where small groups can actually potentially dominate the chain and have in short times. That's the fundamental risk of that approach.

So I think you have to go back and say, "What is it we want to get out of that system? Is it privacy? Is it independence from traditional banks? Is it just agreement and how many players do we need to agree? Do we need to know who the players are?" I think that those are all interesting questions, and I don't feel like I have good answers to them.

Again, I'm kind of glad the space is being explored even though I'm not working on it myself, and I do think, again, the idea of a reliable ledger has value, for sure. It's an interesting construct and useful. I'm not sure it's perfect for currency, but I'm glad people are trying it.

[01:03:38] JM: What about smart contracts?

[01:03:39] EB: A little more interesting. I think those are complicated and a little unclear how to enforce or even what they mean, right? A lot of contracts has to do with what do the words mean? If you don't agree in what the words mean, the contract isn't going to go well. Then you need how do you do arbitration and what are your legal remedies? What country has jurisdiction? So I think in narrow spaces where the terms are very well understood, a smart contract can be a very useful direction.

[01:04:08] JM: Eric Brewer, it's been an honor speaking with you. Thank you.

[01:04:10] EB: Thank you. I appreciate the time.

[END OF INTERVIEW]

[01:04:16] JM: GoCD is a continuous delivery tool created by ThoughtWorks. It's open source and free to use, and GoCD has all the features you need for continuous delivery. Model your deployment pipelines without installing any plug-ins. Use the value stream map to visualize your end-to-end workflow, and if you use Kubernetes, GoCD is a natural fit to add continuous delivery to your project.

With GoCD running on Kubernetes, you define your build workflow and let GoCD provision and scale your infrastructure on-the-fly. GoCD agents use Kubernetes to scale as needed. Check out gocd.org/sedaily and learn about how you can get started. GoCD was built with the learnings of the ThoughtWorks engineering team who have talked about building the product in previous episodes of Software Engineering Daily, and it's great to see the continued progress on GoCD with the new Kubernetes integrations. You can check it out for yourself at gocd.org/sedaily.

Thank you so much to ThoughtWorks for being a longtime sponsor of Software Engineering Daily. We are proud to have ThoughtWorks and GoCD as sponsors of the show.

[END]