EPISODE 1105

[INTRODUCTION]

[00:00:00] JM: Managing microservices becomes a challenge as the number of services within the organization grows. With that many services, there's come an interdependency problem, downstream and upstream services that may be impacted by an update to your service. And one solution to this problem is a dashboard and newsfeed system that lets you see into the health and the changes across your services. With this kind of system, you can avoid accidentally shipping code that will impact other service owners. It can also help with testing, giving you and end-to-end picture for how a test can impact other services.

Anish Dhar and Ganesh Datta are cofounders of Cortex, a system for managing your services. Anish and Ganesh join the show to talk about their work building Cortex and the value it provides to the companies which use it.

In a previous show, we covered a company called Effx, which does something similar, and you can find more about that topic by searching for Effx in the show catalog. You can find all of our episodes about microservices and other subjects by going to softwaredaily.com, or subscribing to Software Daily, the podcast, or you can go to softwaredaily.com and look for all those episodes and become a paid subscriber if you'd like. Paid subscribers get access to ad-free episodes. So if that's something you're interested in, go to softwaredaily.com.

[SPONSOR MESSAGE]

[00:01:31] JM: Join us on August 26, 2020 for GitLab Virtual Commit! An immersive 24-hour day of practical DevOps strategies shared by developers, ops pros, engineers, managers and leaders. Attendees will hear from U.S. Air Force and Army, GNOME Foundation, State Farm, Northwestern Mutual, Google, and more and more about problems solved, cultures changed, and release times halved. Come and be part of a community of people just as passionate as you are about DevOps. Register today! softwarengineeringdaily.com/GitlabCommit

Thank you to GitLab for being a sponsor.

[INTERVIEW]

[00:03:00] JM: Guys, welcome to the show.

[00:03:02] AD: Thanks for having us.

[00:03:03] GD: Excited to be here.

[00:03:05] JM: We're talking about Cortex today, and management of "microservices", and the first thing I'd like to discuss with you is that word that is so commonly discussed, and specifically deployments of microservices within companies. I'm sure you guys have talked to lots and lots of customers, lots and lots of potential customers. I'd like to know some misconceptions that you hear or see around deployments of "microservices". What are the things that you think get talked about less than they should be talked about?

[00:03:49] GD: Yeah. I mean, I think one of the things that keeps popping is like this misconception that microservices, or just generally from a service-oriented architecture, people tend to say like, "Oh! We don't have microservices. We don't do that here." It seems to be a pretty common thing where there's some kind of stigma against it today. I think just because a lot of people have talked about the issues with microservices, and so people are like, "Oh! We don't have a microservice architecture here." And so then we're like, "Okay. Is it more of a service-oriented architecture?"

I think people generally feel more comfortable with that description of their architectures, because I think people tend to have this conception where like, "Oh! Microservices mean there's this crazy proliferation of services, like almost like functions, like Lambda functions." Stuff that just does a single thing. There're probably duplicates of that across your architecture. I think that's one of the things that we've heard.

Another misconception I think is just generally like people confuse like, “Oh! If you’re using Kubernetes, like does that automatically mean it’s a microservice architecture or something?” The tooling and the architecture are two separate things. I think the tooling helps you deploy your microservices, helps you manage and operate the stuff that you’re building. But that isn’t necessarily – It’s not like a one-to-one mapping with your actual architecture or the design of your services. I think that’s one thing that we tend to hear a lot as we do research.

[00:05:02] AD: And it’s interesting, because I think even within like a big monolith, you have individual pieces that have specific owners and kind of have like their own logical component. So I think what’s interesting is that we found that you still have very similar service complexity problems that are out from having a large monolith and companies have to still figure out ways to define ownership and kind of separate to business between the teams who own them.

[00:05:30] JM: Right. But I guess that business logic management is more feasible than it would be with the monolithic situation. And when you look at the companies that have been doing this for a pretty long time, maybe the Netflixes of the world, the Airbnbs of the world, what is the gap in tooling or the gap in proficiency when you compare the elite companies – Or I shouldn’t use that word, but the companies who have more experience doing this, and the companies who are less experienced or who are trying to modernize their infrastructure.

[00:06:10] GD: Yeah. I mean, I think there’s a whole bunch of stuff throughout the developer workflow. I think starting with just like how easy is it to spin up a service. Are there templates, standardized templates within the engineering organization that do all the right things? Like it hooks up to your log things, it has the right framework, the right versions of the framework. It has a right kind of documentation, or the readmes are standardized across services.

So just basically from like the starting of the tooling. I think that’s one thing that more mature organizations tend to do.

[00:06:37] AD: Yeah. Coming from Uber, I think – Especially, Uber built a lot of internal tools to help with managing services. I think what you end up seeing is that a lot of established organizations, like Uber, and Airbnb, and Atlassian, they end up building internal versions of Cortex to kind of help with service complexity. I think the question that they keep trying to answer is how do you define what service quality to use and actually scale that across your entire engineering organization.

And they've built these really robust tools that become one of the most popular internal tools within the company. Like Atlassian has this great tool called Microscope, and it basically is like a service registry, but also deeply integrated into their infrastructure. So it helps with things like service health, and deployment, and it's really like the first place new engineers go to when they're trying to information about a service or really just onboarding to the company.

And I think what we've seen with some newer organizations that are starting to see these problems, you really end up having to use rudimentary tools, like Excel sheets, Confluence tables, kind of hacking together different things to kind of define what service quality means. And then it becomes challenging to keep that information up-to-date and actually remember to update it if things change about a service.

[00:07:49] GD: I think the reason for that really stems from like an organization standpoint. The reason that organizations move to microservices or a service-oriented architecture is because the monolith tends to slowdown like velocity from a team level, not necessarily from a technical standpoint. It's like how do individual teams deploy on their own cadence, build their own features, choose their own technology?

As a result, like the service – Different teams don't have visibility into other teams. What are they building? It's almost like looking to avoid and saying like, "Hey, as a team. You own this domain." Whatever you're building is just there. We assume it works. We kind of know what it is.

I think these kind of tooling helps or like organizations understand like, "Okay. These are the things that we're actually building. Here's what they do. Here're the people that talk to them." So I think they've built all these tooling around their service-oriented architectures not just from a technical standpoint, but also from an organization standpoint. I think that's definitely a huge, huge component of it.

And then more from a technical standpoint, I think just stuff around like deployments. So if you have a whole bunch of service that depend on each other, how do you test that locally? You have mocked versions in your codebase that you can test. You spin up an entire set of services locally. Do you have a dev environment where you can point your local service and that data gets blown away every now and then? Do you do like blue-green deployments? Canary deployments? How do you make sure that one service doesn't take down 20 different services? So I think there are all these tooling around that that large organizations, more mature organizations have been able to build out.

[00:09:13] JM: So you guys have created Cortex, which is I don't know how you describe it. What's your boilerplate pitch for it?

[00:09:22] AD: Yeah. Cortex is a tool for engineering teams to track and improve their service quality. We automatically catalogue all internal services with their dependencies and then link it to accurate ownership information. That's usually linked to your identity provider like Okta. And then using that data, we can surface interesting insights to your team, like team level SLA conformance or scorecards that let you really define what service quality means for your team.

[00:09:47] JM: And I think the first question around that is the fact that this is something that is mostly going to be useful if everyone in the organization has adapted it. So you have these large organizations with heterogeneous models of managing their different services, and this is a system where it'd be useful to know, for example, if I've deployed a breaking API change and your downstream, your service is consuming my service. It'd be useful. You would like to be notified of that breaking change that I've deployed. So how do you easily get people throughout the organization to uniformly adapt Cortex?

[00:10:35] AD: Yeah. I think one of the interesting things we've found is that different stakeholders in an organization have different priorities for what service quality means, right?

Your SRE team might have a set of guidelines that they adhere all services that all services need to meet to get SRE support. Your security team might have security policies that your services need to meet. So what we found with Cortex is that we have this product called score cards, and score cards at a high level, they let you apply a set of rules to a group of services.

So what we found successful with this, like partnering with SRE teams who have very specific guidelines on what constitutes service quality and then they institute it for the services that they work with. And basically from there, we let other teams kind of define what service quality means. And the goal is that service quality should be this moving target that continues to improve the services and let different stakeholders kind of add to it. That's kind of how we get buy-in from different like pieces of an organization within Cortex.

[00:11:33] GD: I think what's pretty powerful is like – So before this, I was at LendUp as a software engineer, and I joined right when we split out the first service for the monolith. And I was there as we built out a whole bunch of services, like a real service-oriented architecture. And one of the things that we realized, that we kept making the same mistakes over and over again in every service. The very first service that we pulled out was almost a clone of the monolith in terms of like infrastructure tooling, monitoring, all that kind of stuff.

But as we started building out more services, and like we had a DevOps team building out like our pass internally, we started doing a whole bunch of stuff like repeatedly, things like, "Okay. Are the logs being – Are they going into Sumo Logic? Do we have 500s alerts? Do we have 400s alerts? Do we have latency alerts? Do we have an on-call rotation?" All these like basic things that a service needs to have for a team to reasonably operate that service.

So what we realized is that like a single team can build out this kind of like checklist, for example, within scorecards. And so they can apply that to just their services, or like a security team can say like, "Hey, this is a tool we're going to use for our security audit. You need to answer these questions," and we're going to use Cortex to track that across all our services. So you don't necessarily have to onboard everything into breaking API changes or the service catalog. It's like a tool for a single team to answer questions about other services. I think that's what makes it pretty powerful, is the ability to really understand services.

I think the service catalog portion of it is definitely more useful. The more people that onboard it, the more people that are using Cortex across organization is definitely more useful. But I think scorecards are where we can say like, "Hey, as a single team, like you can start getting value out of it."

[00:13:04] JM: Okay. So two things; service catalog and scorecards. Service catalog, pretty easy to understand. This is I can scroll through all the different services that exist within a company so I can know does a billing service already exist. Or perhaps I know that I want to find the billing service. I want to know who's associated with it. I want to know recent deployments associated with it. That's pretty easy to understand. Tell me more what the scorecard is.

[00:13:33] AD: Yeah. Scorecards are basically let you define SLIs for your internal services. And so for example, one common use case that we are able to solve with scorecards is one of our companies, they used to have like a DevOps meeting every week. One engineer would go and compile a bunch of different heuristics for the services on their team, like how many Jira incidents were there this week? Is there a GitHub repo still attached? Has it been deployed in the last few weeks? Checking SLA's for each of these services. That's a pretty time-consuming task and is prone to a lot of errors, because someone as a contingency, go and update it.

And so basically scorecards let you define these metrics, and we directly linked via read-only API keys to likings like New Relic, or PagerDuty unity, or even like looking into your Grafana dashboard for certain metrics. And so that way, basically, as soon as you open a scorecard, you're able to see a stack rank list of services and basically how they adhere to those quality metrics like. Each scorecard is given a set of points, like maybe 100, and each rule is given a weight within that point system.

Basically, for an engineer now, you don't have to go in and manually update this XL sheet. There are services you're just continuously tracked against these quality metrics that are automatically updated. And as these services are added to your team, they're automatically added to a scorecard as long as it's tagged with an appropriate team structure.

The second way that we build these report cards, so like as a CTO, for example, I might care about like how many customer-facing incidents to we have. Do we have any compliance issues that were filed this week? Is our meantime resolved holding up? Like as a CTO, I might build this report card. But as an SRE, I might have a different scorecard. I might say like, "Hey. Are there enough run books for every single service?" And so because this is like pulled in real-time, we can say we can catch services that are nonconforming before something bad happens.

So like the reason in the first place that you have these tech ops meetings where people sit around and they look at these metrics and they like analyze those services, because at the end of the day, what you care about is are my services are reliable? Are they performing the way they're supposed to? And because if they're not, that's business impact.

And so I think scorecards, by building these report cards, you can say like, "Okay. These services are at risk. We can go and fix the things that are making them at risk." So hopefully we can prevent things from going bad, instead of having to cast something after an incident. I thin k that's where scorecards really comes into the picture.

[00:15:49] JM: Can you tell me more about what the process of deploying cortex looks like? What is the adaption pattern for an organization who is adopting Cortex?

[00:16:03] AD: Yeah. So we have a couple different based on Word, and this has evolved as a spoke into more companies. Initially, we basically looked into your APM tool. So if you're using New Relic or Datadog, would use a read-only API key to basically catalog all services and dependencies in one sort of onboarding flow. And then we have heuristics that basically map your services to the on-call rotations, to latency graphs, to the documentation, and we try to automatically do that using like naming heuristics and things like that. But by the end of the flow, basically you have mapped your services to the data that represents them. And how Cortex works is that each service is represented in a YAML file, which contains all the metadata for that service. And then you would upload that to Cortex as part of your CICD process.

But other than APM tools, we also support – We have a Kubernetes integration, and we even have like an Excel sheet integration for companies that had been tracking this already in Excel sheets.

[00:16:58] GD: Yeah. I think the way it works is a single team can onboard. So like [inaudible 00:17:01] services, like we're talking about previously. As an individual developer within a team, I can go and onboard my services and start getting value out of it. And I think the YAML file approach is like more of a GitOps thing, where ideally the onboarding flow walks you through this thing and like you get a whole bunch of YAML files at the end for all your services. But over time, as you adapt Cortex as like a single source of truth, you would have this YAML file was as part of your service template. So every time you spin up a new service, you would have this YAML file there by default. And so that way you know every time a new service has spun up, it's already in Cortex. So that kind of where it ends up over time as an organization adapts Cortex more heavily.

[00:17:40] JM: There've being a lot of service registry tools in the past as far as I know. What's been your experience assessing the wreckage of previous service registries? Or is it okay to have duplicative functionality in a new service registry?

[00:17:59] GD: Kind of story – This kind of gets into the story like how we started Cortex in the first place. When I was working at LendUp, we had a whole bunch of services that we spun up over the course of a couple weeks, and I was looking for, "Hey, is there some way to like organize documentation for these services?" I [inaudible 00:18:14] on Swagger for like API documentation. I'm like, "Okay. Can I can I repurpose this for service documentation?" It didn't really seem to be a good way to do that. Swagger Hub was a way of compiling Swagger files for a whole bunch of services in a single dashboard, but nothing really about the service itself. It was more focused on the APIs.

And then separately, you had like machine-readable service registry or service catalogs. And so those told you like, "Hey, these are all the services," but didn't tell you anything more than that. You didn't actually tag information about them. And then eventually I ended up trying like Hugo

Micro [inaudible 00:18:45] to static markdown files within the repo. They got published to a single place, but just setting all that up was a huge pain.

And so then eventually I started looking for like is very human readable service registry? And so we actually didn't really find anything at the time when we started it. I think there a couple people like working on that now. But at the time, there was nothing really. And do we're like, "Hey, is there some way that we can make a human readable service catalog where the focus is around like, as a person, as an engineer, I want to answer the questions about like who should I talk to if the service goes down? Where is the documentation for this?" Like a machine-readable service catalog doesn't really give you. So that's kind of where Cortex really came from.

[00:19:21] AD: Yeah, and to kind of touch up on your point about maybe why catalogues haven't been so successful in the past. I think what we found is that the service catalog alone isn't enough to really ingrain yourself in like the cultural process of a company. I think it's what you do with the data that you put into the service catalog and how useful you make that for engineers and for managers. So that's kind of their approach we've taken with something like scorecards. And then also some of the integrations that we have specifically for engineers.

As an engineer, when I push code, I'm thinking, "Is my code going to break any number of other services? That's like the cognitive overhead. You have to keep track of as an engineer." So we have like a GitHub integration that will comment on your PR and tell you like, "Hey, this API change is going to break these three other services, and here are the owners." And so I think it's more than just a service catalog. It's also about how can we present that information to engineers when like they face service complexity the most in their day-to-day jobs.

[SPONSOR MESSAGE]

[00:20:26] JM: You probably do not enjoy searching for a job. Engineers don't like sacrificing their time to do phone screens, and we don't like doing whiteboard problems and working on tedious take home projects. Everyone knows the software hiring process is not perfect. But what's the alternative? Triplebyte is the alternative.

Triplebyte is a platform for finding a great software job faster. Triplebyte works with 400+ tech companies, including Dropbox, Adobe, Coursera and Cruise Automation. Triplebyte improves the hiring process by saving you time and fast-tracking you to final interviews. At triplebyte.com/sedaily, you can start your process by taking a quiz, and after the quiz you get interviewed by Triplebyte if you pass that quiz. If you pass that interview, you make it straight to multiple onsite interviews. If you take a job, you get an additional $1,000 signing bonus from Triplebyte because you use the link triplebyte.com/sedaily.

That $1,000 is nice, but you might be making much more since those multiple onsite interviews would put you in a great position to potentially get multiple offers, and then you could figure out what your salary actually should be. Triplebyte does not look at candidate's backgrounds, like resumes and where they've worked and where they went to school. Triplebyte only cares about whether someone can code. So I'm a huge fan of that aspect of their model. This means that they work with lots of people from nontraditional and unusual backgrounds.

To get started, just go to triplebyte.com/sedaily and take a quiz to get started. There's very little risk and you might find yourself in a great position getting multiple onsite interviews from just one quiz and a Triplebyte interview. Go to triplebyte.com/sedaily to try it out.

Thank you to Triplebyte.

[INTERVIEW CONTINUED]

[00:22:42] JM: What are the biggest engineering problems that you've had to overcome when developing Cortex.

[00:22:49] GD: Yeah. I think one of the things is just like third-party integrations. It's always a challenge. And obviously have like rate limits and things like that. So how do we integrate with third-parties in such a way that we can make the right number of API calls? We're not getting rate limited when we evaluate your scorecards and things like that. I think that was definitely one challenge.

But overall, I think from an engineering standpoint, it's been pretty straightforward. I think most of the challenges have really been this understanding like customer use. How they use the product? For the most part, it's like all driven from this YAML file. So like the integrations is there with like GitHub. Breaking API changes was an interesting challenge. Like how do we detect like breaking API changes? Things that are like backwards incompatible. So if you change something that's knowable to not knowable in a request body, that's a breaking change. How do we detect that sort of stuff? The challenges are mostly in like figuring out how does that integrate with the rest of the flow? To be honest, not that many like tough interesting challenges in building the product itself.

[00:23:43] JM: Yeah. More of a kind of product design set of challenges.

[00:23:47] GD: Yeah. Like understanding, like from our perspective, what's ironic is that Cortex is a monolith. We built it as a monolith. I think that's the right way a startup should do just from the early days. And eventually once you realize like, "Hey, here are the things that are logically separate." You start pulling those out into services.

And so because we're so – We're building the thing in a monolith, we had to step out and say like, "Hey, in our old days and the days we were actually building, like working on services, what are the challenges we faced?" And so we went through all these different sessions where we said like, "Hey, what are all the tools a person uses during their daily engineering workflow? How do they use those tools?"

And so trying to map that and understand really like what are the root causes of service complexity? I think that's where the most the challenge has been for us as supposed to like the actual engineering work.

[00:24:33] JM: And had there been any emergent problems that you can solve for people? Like things that you didn't anticipate or things that even people didn't anticipate until you started giving them this tool that is sort of the service catalog plus scorecard? This oversight into all the different microservices. Any emergent problems that people realized, “Oh, you can solve with this tool.”

[00:24:59] GD: Yeah. I think one of our customers actually using it for contract testing. And so like [inaudible 00:25:03] effort internally where they wanted to make sure, because they have an external- facing API, they wanted to make sure that none of the changes they’re making internally or externally are breaking that contract.

And so one thing that I've heard a lot about recently is like contract testing. How do you make sure that your services are conforming to a predefined contract? And so because we have this breaking API change detection that can say, “Hey, we don't want to spend time spinning up a whole contract testing suite. We just have to pipe in the API documentation into Cortex and Cortex will tell us if something is breaking.” So I think that's a really powerful tool separately from that, like security audits.

We build scorecards like a service quality thing, but we heard like, “Hey, why don’t we use this first security audits? Can we make sure that vulnerability scans are being piped into Cortex? Can people answer questions about their services from a security standpoint?” So an SRE team, or security team, I can go through and say like, “Hey, these are the services that are at risk. They might have security vulnerabilities from our third-party security vulnerability scanners. The audit hasn't completed for these services.” I think there are some interesting use cases there for sure.

[00:26:02] AD: Yeah. And just to quickly add to that. I think what’s also interesting is that we originally built Cortex for microservices, but we found that many customers actually start representing non-network services in Cortex, things from like Kafka consumer, even simple libraries, any module really with an owner. And I think that really extends the use case as well, because I think you start to see a lot of the same problems emerge for really any like logical entity with the owner. It’s just easy to start tracking with Cortex, and it’s like a world of difference, like searching through data on our platform versus Excel sheets or Confluence tables where you just can't really search their APIs or owners or even like documentation. And so that's also been pretty interesting to see.

[00:26:44] GD: Yeah, I's a great point. Like even one thing we didn't expect going into was like monoliths. Within a monolith, you would think that because all the code is in one place, it's easier to keep track. But as a monolith matures, I think within the monolith, you end up with these logical modules. Like you might have a section of the codebase that just does payments. You might have a section of the codebase that does like identity. And naturally over time, the teams even get split up into those domains. So even within a monolith, you have the logic

And so we've seen people use Cortex to define the boundaries within a monolith so you can say like, "Hey, within this monolith for this payment's codebase, this is the team that owns it. Here are the people that are part of it. Here's the documentation for it."

Really, we've seen people use it for like just logical modules, things that do certain things. Like not necessarily just microservices or network services as Anish said. That's been a really cool use case that we didn't really expect going in.

[00:27:33] JM: I'd like to talk a little more about deployments. So let's say my company is entirely instrumented with cortex and some service, internal service, deploys a new version. What happens? What do I see in cortex and what's going on along the stack?

[00:27:55] GD: Yes. You can actually push an information about deploys. So you can say, "Hey, this is the latest deploy for this service." And the cool thing is like, for example, if you get alerted for a service, we can actually send you those deployment information, like, "Here are the list of services that just got deployed recently. Here are the latest deploys. Here are the commits," and stuff like that.

I think the other cool thing is you also see your scorecards update. So you might have a rule that says like, "Hey, because we're following 12-factor, we need to be doing frequent deploys. And we have a rule that says, "Hey, this service must have a deployment within the past three weeks." Immediately, you will see that scorecard update. You'll see like, "Hey, this service that was not deployed recently, it now has a deploy. So it's no longer at the bottom of the stack rank. It's is a little higher up now."

So like there's a whole bunch of thing that gets triggered as the function of a deploy. And so like that's one of the cool things that scorecard does, is yes, it's cool to know that, "Hey, we deployed a service." But what does that really mean? So because with a scorecard you can say like, "Hey, a deploy is actually an important thing from an organizational standpoint. We care that we're doing deploys recently. You can build a rule on that and have those rules update." And so that's one of the things that you can do with deploys."

[00:28:58] JM: And how is that information being fed back into Cortex like during a deployment? Is there some kind of observer pattern going on?

[00:29:10] GD: Yeah. There are a couple of different ways. Like one is like a web hook, where you would push the data into Cortex. So with an external-facing API, where you can push in custom data to build rules on for your scorecards. For example, vulnerability scans. You might want to push in data from your CI pipeline about vulnerabilities. So that's one example.

Deploys, we have a separate web hook for deploys, but you can even push in like separate data about that. Another thing that we do is like third-party integration. If you have information, like if you have a Heroku integration, for example, we can hook into Heroku, get the information from there. Or if you're using a different tool that has an API, we tend to prefer pulling data from the source of truth directly. Because our theory is that, "Hey, Cortex should be this layer that sits on top of your existing tools. We aggregate information and we give you new context that you didn't have before."

And so if you have Heroku handling deploys, then, really, that is your source of truth about deploy information. So give us access to Heroku with a read-only API. Okay. Let us pull that information from Haroku directly. Or if you have a different service that has that information, we'll pull it directly from there. But if you don't have a source of truth, then we'll try to pull it. You can push that information into Cortex.

That philosophy, really, it goes across Cortex everywhere. That's like on-call rotations. PagerDuty is a source of truth. Let us get it from there. So like where can we pull the information? Cortex is not the single point of failure. We're actually pulling it in from where the data should live in the first place. And so deploys is just another thing where we do that.

[00:30:37] JM: What about tests? How do tests interact with Cortex?

[00:30:45] GD: Yeah. I think, as I mentioned [inaudible 00:30:47]. Contract testing can be replaced with Cortex. You can push in a coverage information. So like within scorecards, you might want to report on like, "Hey, do all our services have 80% coverage or something like that? Because I think one of the things that we've seen is that as part of your CI process, some companies that care about coverage sometimes put like a hard stop. Like you have to hit 80% coverage until this thing could be merged in.

And what happens is people end up writing all these really funny tests just to push the coverage above a certain limit, which I don't think is really valuable in the long run. So as part of your party testing process, You might generate a code coverage report and you can push that into Cortex and build a scorecard on top of it. Instead of having like a gate on your pull request, you can say like, "Hey, show me all these services that don't have good code coverage."

From there, it's more of like a cultural thing. Instead of having that gate, you can say like, "Hey, as a service owner, if my service is at the bottom of this list, and it's because my coverage is bad, I'm probably going to go in and fix that," because there's like a cultural element to that. So that's kind of where it fits into that testing pipeline.

[00:31:47] AD: Yeah. And what's really cool, we're releasing this feature pretty soon, but we're actually starting to track the historical data of all the metrics that you start to push in to Cortex. And so for a per service level, you can see how is my code coverage changed over time? And even alert, if like it goes under a certain rule boundary, you'll get like an email or a Slack message.

So we really want you to be able to understand the quality of things like test coverage over time and try to improve it. One of the companies we work with, like every day during stand, they'll pull up their test coverage for like their lowest performing services and just check has it been going up over time?

[00:32:23] GD: Another thing is like flaky tests. Like I know when we – At LendUp, we this monolith, right? And so flaky test was a huge problem. There was a lot of – It was like a fin tech company. So a lot of daytime related stuff. We're like moving through time. And so a ton of like flaky tests. So one of the things you might care about is like, "Hey, what percentage of things that we're merging in have flaky tests? Or like a check suite fails?"

And so you could like imagine building a rule on top of that as well. You can say, "Hey, if the percentage of failed commits is above a certain threshold, then flag that." That might be a good way of caching flaky tests. You can say like, "Hey, if 60% of our commits are failing, then that might be because they're flaky tests on master." So that's another way of reporting on top of that.

[00:33:05] JM: What about CICD workflows? Anything to add about what Cortex does in interactivity with CICD?

[00:33:14] GD: Yeah. I think like deploy, like you said, is one things. So you can push information on deploys. You might do like the breaking API changes. So we don't really hook into CI workflows necessarily, but you might integrate with Cortex for those things, like breaking APIs, change detection, vulnerability scans that you run every time like on every commit as a part of your CICD. So you want to push that data in. That kind of information would go into Cortex.

I think the cool thing again is like this is not something we're really thought about. Off the top of my head, I can't think of anything interesting that you might want to report on with a CICD process. But because like scorecards has this like – It's almost like a rule of language, like it's like a pluggable engine where you can plug in different integrations. And so if a customer comes to us and says like, “Hey, we want to have information about our CI process.” Then they can push that information in. But as of right now, apart from like breaking API changes and like the custom data you can push in, there’s not really that many integrations with CI.

[00:34:06] JM: Let’s talk a little bit about the engineering behind Cortex. Can you describe the stack of technologies?

[00:34:12] GD: Yeah, for sure. So we’re using Springboot and Cotlin on the backend, React Typescript on the frontend, and then a Postgres backend. We've deployed on GKE, and that's pretty much the stock end-to-end. We used Dokku and DigitalOCean for a while as we’re getting spun up, which is pretty easy just to deploy things. And we’re still using it for our demo deployments. So we have like a demo box that we call where we push like a demo version of a product that we give to potential customers, we send to people saying like, “Hey, go play around with this.”

So that right now, we actually just have more like this Dokku, which for people that are not familiar with, it’s just like an open source Heroku equivalent that you can deploy. It’s like Dockerized. You can just deploy it on like an empty VM. We have that on DigitalOcean, which we hope to eventually bring into GKE. We hope to like use Terraform and stuff. But nothing really mature on that front.

But the actual code, as I said, like Cotlin, Typescript, we’re definitely very strong on types. I think it come from the fact that me and another cofounder, we came from like fintech backgrounds. And so we have definitely understood the pain of like caching issues early, because any small bug can result in like compliance issues, regulatory issues. So we definitely like our types.

So Typescript has been actually a huge, huge improvement in our workflow, I think. Just being able to catch bugs ahead of time in Typescript. And we’ve definitely run into this. A lot of people say like, “Oh, you can – Static typing helps you cache this stuff.” And we’ve definitely seen it over and over again where in one data type it might be called ID. In another thing, it might be call service ID. And if you're not keeping track of that, you can easily have a bug that you don't cache until you're in production. And so typescript have definitely helped us cache those kind of things ahead of time, which has been extremely helpful.

[00:35:52] JM: What about the hosting? The cloud provider of choice?

[00:35:57] AD: Yes. We're using GCP and GKE just for – Like we don't want to be locked in to a single provider. Postgres in the backend. So we're using cloud SQL for that. But we really only have a single service hosted on GKE right now. We're using Firebase for static site hosting. I did look into cloud storage for that, like whether we could do like static buckets and just point like a domain to that and use it for static hosting. But even I think GCP recommends Firebase for static website hosting. So our React app is entirely just on firebase. And then we use like GitHub actions for CI process just because the integration was so simple with our GitHub workflows.

[00:36:34] JM: Has firebase gotten pricey for you?

[00:36:37] AD: So that, I think the static hosting itself is pretty cheap. We don't use any other APIs apart from static hosting. So that's been relatively cost cost-effective. I think it's deftly more expensive than it would have had just to spin up like a single container or something with like [inaudible 00:36:50] with static files. But the ease-of-use is so simple, like just point your DNS to their server and everything is good to go, and obviously because of like startup GCP credits. Like it's too much of a concern at this point. But overall, it's been pretty cost-effective actually.

[00:37:06] JM: So I'm a little curious about the competitive landscape. I've talked to Buoyant for several years as they've expanded from just Linkerd to thinking more about this service catalog, service management side of things. I talked to Effx, who I think is more of a direct competitor. And I guess the first question around competition is I think the difference in taste. Do you have any – I'm sure you've seen the competitive landscape. Are there any notable distinctions in your positioning of what you're trying to do with the product? Or do you think it's just kind of a feature for a feature dogfight?

[00:37:48] AD: Yeah. That’s a great question. So the first thing I will say is I think it's really great that there's a couple smaller competitors. I think what's interesting is we all started around the same time. It’s interesting how that works, but I think it's really good that there are multiple people working on this problem. Because if you think about the ecosystem for microservices, there have been so many tools built for the technical problems that come with services. But I think you're starting to see that organizations are starting to face some of the organizational challenges of using services. And so I think all of us are tackling that range of problems.

I think where we differ specifically is I haven’t used like Dive or Effx personally, but I think on Effx’s front, I think they’re a little bit more focused on the incident response space. Like there are so many different pieces in a service when you get alerted, being able to see like a timeline of your events. I think that’s definitely a valid value proposition. I think where we differ is that we’re really focused on service quality. And so that kind of goes back to scorecards, being able to define what service quality means for your team. And that really just goes back to our time as engineers where especially at Uber, I think one of the most frustrating things for me was that the quality between services is very inconsistent. And I think that ultimately led to the biggest challenges from an engineer perspective, but also from a management perspective. Just getting like a report of your services and how they’re conforming to those standards. And so that's kind of an area that we’re really diving deep into and focusing on with like scorecards and kind of experimenting what we can do there.

[00:39:18] GD: Yeah. I think, obviously the common point across all is like the service catalog. I think, as engineers, I bet like the founders of those other companies also felt the same thing, like, “Hey, I don’t even know what services we have.” That’s just like the common problem I think we all had, and I think as we like go deeper into the problem, as Anish said, we found like different aspects to that. And so we all started with the service catalog, but it’s like, “Now, okay – Now that we have the information, what can we do with it?” I think Effx is like, “Okay. We can improve your incident response. We can show you this timeline. We’re going to focus on service quality.”

So it's really like how do we take information about your services and make it more useful to these organizations. I think that perspective on service quality is something that makes us a little different. I think, separately, just from like a technical standpoint, I think we're really focus on like the GitOps style approach, where like everything is defined with this YAML file. It's not really like we do have an editor in the UI, but it's mostly focused on like, "Hey, check in this YAML file to your git repo." That's the real source of truth, and like upload that to Cortex. That approach I think is a little bit different from the other people in the space. I think that's just an approach that we took.

We heard a lot from some the people that have built this out, like mature organizations, like Atlassian and Zendesk and stuff like that, where they have taken this YAML approach. I think engineers tend to find like some comfort in that saying, "Okay. This information is owned by us. It's version control and it's checked in to git. We're the real source of truth." And Cortex is just like this aggregation layer on top of that. I think that's one like technical difference that we have compared to some of the other players.

[00:40:45] AD: Yeah. Also, I think a lot of the other products are a lot more focused on like this service complexity phase as a manager, like the audit report I think is something that a couple other people do as well. But I think where we also differ is we found that service complexity also lives with the engineer, right? It's kind of going back to, as an engineer, you have to think about is my code going to break any other number of services?" Or when I get alerted for a service, do I have all the context for that service that I'm working with?

So in that regard, we're built very specific integrations that individual engineers can take advantage of with GitHub or with PagerDuty or Opsgenie. So you kind of have these two suites of products where you have products for service complexity for engineers, but then also for like managers and above who need like audit reports and kind of service quality reports for their team.

[00:41:32] GD: Yeah. I think now that you mentioned that, like one interesting thing, and then the difference almost that we are focused on like before, something going wrong. Can we help you understand things that are at risk before something goes wrong? I think some of the other players are like, “If something goes wrong, here's all the information that you can help you fix it faster.”

And so I think those are two very valid problems and two problems that as an engineer is we've all felt time and time again. I think we’re almost like taking this fork in the road where it’s like can we help you understand issues with your services before they go wrong. Whereas, like, as supposed to let's help you fix those services while something is wrong.

[SPONSOR MESSAGE]

[00:42:15] JM: Scaling a SQL cluster has historically been a difficult task. CockroachDB makes scaling your relational database much easier. CockroachDB is a distributed SQL database that makes it simple to build resilient, scalable applications quickly. CockroachDB is Postgres compatible, giving the same familiar SQL interface that database developers have used for years.

But unlike older databases, scaling with CockroachDB is handled within the database itself so you don’t need to manage shards from your client application. Because the data is distributed, you won’t lose data if a machine or data center goes down. CockroachDB is resilient and adaptable to any environment. You can host it on-prem, you can run it in a hybrid cloud and you can even deploy it across multiple clouds.

Some of the world’s largest banks and massive online retailers and popular gaming platforms and developers from companies of all sizes trust CockroachDB with their most critical data. Sign up for a free 30-day trial and get a free T-shirt at cockroachlabs.com/sedaily.

Thanks to Cockroach Labs for being a sponsor, and nice work with CockroachDB.

[INTERVIEW CONTINUED]

[00:43:38] JM: What has been the competitive discussion with your customers? Have there been any customers who have multiple of these things deployed internally? Is it that mature or a market? It still feels like a pretty nascent market where probably if you're talking to a company and you're saying, “Hey, try out Cortex, please.” They're saying, “Okay. this thing looks cool. We don't have anything like this internally. Let's go for it.” Has it gotten to the point where you have multiple of these things deployed and they can't talk to one another and you have the real kind of battle for different service management systems?

[00:44:19] AD: Yeah. No. That’s a great question. No. To be honest, we haven't seen that really. I think it's a very early market. The biggest competition is just the internal tools that companies build themselves. And I think it’s just been by force so far, because there really hasn't been a great replacement for that. And it's not really the core competency for a lot of the companies that have been forced to build it internally. So I think it's still a really early market. I think everyone is still trying to figure out what is the right application of service complexity that will resonate with everyone who has services. I think, yeah, we really haven't seen a situation where we spoke into a company and they’re already deeply integrating with our competitor or something like that. It’s usually just really fighting like Excel sheets and internal tools.

[00:45:03] GD: I think the reason for that is just because it is a totally new market. I think Anish mentioned earlier, but like there’s been this whole suite of tools for the technical challenge of service-oriented architectures. You have like the Datadogs and New Relics, the Splunks, API gateways. All the stuff to really help you deal with the technical challenges of microservices and service-oriented architectures. But I think this whole crop of like us and our competitors came around this time where it’s like, “Hey, most of the challenges, like the technical challenges are pretty much solved. Yeah, there are things we can do to make it better and deployments can be easier and all of that.”

But for the most part, there’s a ton of tooling out there. But now it’s like, “Hey, we made this transition and as a whole, as an engineering community towards a service-oriented architecture.” And now because we have fixed it, technical challenges, all these are organizational challenges are being exposed. And so a lot of companies are still in the mindset like where they’re starting to realize this. They’re just starting to face said. And so I think as a result, it's more of convincing them like, “Hey, this is the right way to solve the problem,” as supposed like where the right tool as supposed to other tools. It's been more of like educational mostly, like saying, “Hey, the service catalog that does all these things is going to make your life easier. Just always been like more of the educational component than anything.

[00:46:12] JM: When you look at the tools that have been built internally at some these companies, so you mentioned Atlassian. I know Airbnb has built one of the – I think a lot of these companies build some kind of newsfeed for services internally. It seems like there's going to be a gap between what you can build for the masses and what kinds of quality you can get out of some tool that’s built internally with all the nuances of the custom deployment infrastructure and the custom this and the custom that of a mature company. Is there any kind of compromise that you have to make when you are designing a product that is sort of consumable by the masses as supposed to tightly coupled to mature infrastructure?

[00:47:03] GD: Yeah, definitely. And I think that’s one of the things that we keep hearing, is like, “Hey, how do we make sure this thing is going to stay up-to-date?” Because when a tool, like an internal tool is so tightly coupled and so tightly integrated with their existing tooling, it can get a lot of information automatically. It can – Like ACL stuff. If you already have tooling for ACLs, you can probably figure out like, “Hey, if somebody has access to the service for these APIs, then they might be owners of this service.” Or if you have an even more mature organization where you’re saying like, “Hey, my service needs to have access to this other service,” and that service has access to another service, you can start to figure out the dependency graph. There are all these things that you can figure out just because we have all these internal tooling. You might have access to infrastructure. If you like pass internally where Microscope Atlassian does, you can get all kinds of information from the pass that you can push into like their version of scorecards. Like, “Hey, this service does not have HTTPS. This service does not have enough replicas.”

And so that’s the kind of stuff that you can get very easily if you have internal tooling – Tool tightly coupled with those tools. I think what we need to figure out is like can we get this stuff automatically? Can we reduce the friction for ultimate organizations that have their own internal tools? Can we flash that information automatically? Can we reduce the friction of getting that into Cortex in some way? Because, really, that's the bottleneck. How do you convince somebody to say like, “Hey, all these information, like how do we pipe it into Cortex?”

I think that’s where philosophy of like integrating with third parties extremely heavily comes from. It’s like, “Hey, you have all these internal tools. We can fetch that stuff from you. We’re not as tightly coupled as these internal tools might be. Instead, we’re going to hook into your third-party tools and fetch fence all the stuff, as much stuff as we can.” And so like how we reduce that friction in a way t0hat feels almost like it might feel if you have an internal version of Cortex build out tightly couple with your infrastructure. I think that’s really the biggest bottleneck, is like the ease of use and the ease of getting that information into the tool.

[00:48:57] JM: As you get more organizations with their company entirely instrumented around Cortex, or Cortex is entirely woven into the company. What other kinds of things can you build on top of that?

[00:49:13] GD: I think in the near future, like scorecards – The things we can do with scorecards are extremely interesting. We can do things for an organizational standpoint. We say services are at risk. If a service has an optic group assigned with it, like, “Hey, these are the people that own the service,” and the service, like the team goes to zero because this is a reorg.” That's now an orphan service.

That something that’s an organizational risk, right? And so if we can alert you with saying like, “Hey, this service now is owned by the billing team, but the billing team has zero people in it.” We can fly that and say like hey, “You should go fix this. You might want to assign it to a different team.” I think there are a lot of things we can do from an organizational standpoint.

I think the near future is just really like integrations. Can we really focus on security? Can we help security teams do audits in a more powerful way? Can we integrate with Synk for vulnerability audits? Can we pull that kind information and make it really powerful for that front? I think scorecards are really where we’re going to be investing heavily in the near future and making an extremely powerful tool from a technical standpoint, from an organizational standpoint. There are a ton of stuff that we can do on that front. So I think that's pretty much what's coming up in the near future.

I think making stuff more powerful from – Breaking API change detections, like we already do GRPC an open API and GraphQL. Can we make that stuff even more powerful, different kinds of API detection. How useful is it and replacing contract testing? What are other things we can replace that people are doing manually today with almost all these information that we've compiled about it? Can we integrate with tools like Blameless? Can we give you information during incident response, during postmortems that may make it more useful? I think those are all the different directions that we can go. But the near future is definitely scorecards. I think there are so much we can do with it that's really exciting.

[00:50:54] JM: What about pricing? How have you determined how to price Cortex?

[00:50:59] AD: Yeah. So that's something that we continue to iterate on. Right now, it's a combination of how many services and how many team members are in your org. And we kind of scale it up from like – It's like free for a certain number of users and services and then kind of goes off from there. But generally, I think for us right now, it is really been over index on like feedback and trying to get Cortex to a position where if you remove Cortex from your organization, all your engineers would really hate that.

I think that's something that we'll continue to work on and just iterating on, because we wanted to be able – Like one of the most viable tools within your company, not just from a technical standpoint. Kind of going back to like the cultural significance of a tool like that where you just start relying on it for all sorts of information around anything to do with the service.

[00:51:45] GD: Yeah. People always say like pricing is a dark art, and I think we've definitely come to realize that over time. We started out with like per service, per engineer. We've gone through all these iterations. I think we're still learning like how people use Cortex. Is it mostly the dashboard? Is it that can have integrations? I think that's really going to drive the pricing model going forward. So it's really about learning how do people use a product overall today? It's been interesting set of learning for sure.

[00:52:13] JM: Are there any opportunities around documentation or architecture mapping that you see?

[00:52:22] GD: Yeah, definitely. We, right now, have like a dependency graph where you can see like, "Oh, these services are dependent on each other, even at the API level." So you can say, "Hey, this service depends on this particular API for another service." That's a pretty rudimentary way of visualizing your architecture.

I think one cool thing might be like architecture diagrams embedded within the YAML. So you have like all these markdown-based architecture diagrams. So if you can embed that in your YAML file, then in theory, you could do architecture diagrams across services. So like this piece of the architecture diagram touches another function in this other service. And so you might be able to build these really complex architecture diagrams where I only have to maintain the diagram from my single service.

I think that really comes down to, again, like as a service owner, I know the most about my own service. And so I should declare that. Whereas today in an architecture diagram, you have this like giant [inaudible 00:53:12] I/O files that are embedded in a Confluence page with like 100 different services talking to each other. And if I want to go in and update this thing, I don't really know whether I should be touching a certain link between two services. Whereas with Cortex, you can say like, "Hey, my service depends on these other services. Here's the architecture diagram for my service. And because we have information about all your services, we can compile that into this like living, breathing architecture diagram for your services. So that's something we actually thought about that I think might be very, very powerful.

[00:53:39] AD: Yeah. One other thing that we're also working on to kind of make it more live is like if we detect there's a bad deploy or like you break a rule in the scorecard, that we'll like change the color of the two links between services to red or something like that. And so you kind of get this live view of your architecture and how it's behaving over time, which I think is a pretty powerful tool just from a visualization standpoint.

[00:54:01] JM: You mentioned earlier the benefit of having worked in fintech a little bit earlier. Are there any other previous experiences you have that have shaped how you are architecting and designing Cortex and the company?

[00:54:14] GD: Yeah. I think there're definitely a few things, like when I joined LendUp, it was a massive monolith. The monolith number really went anywhere during my time there. We pulled out a whole bunch of stuff. When I first started there, I was like, "Oh man! This monolith is so painful. It is slowing us down. It's a real challenge to develop in."

But over time, I kind of realized like why do organizations have these models in the first place? Because when you're moving fast, it's easy to build things within the context of your existing codebase. It's like the unfortunate truth, but it just makes your life much faster. As a consequence, your data models tend to become very tightly coupled. And so that's one thing that we're keeping a close eye on, is like making sure our data models are not tightly coupled as much as we can. We're not reaching in. We have good abstractions from a service levels. So we have all these abstractions within the codebase where if one part of the codebase wants to talk to GitHub, for example, for integration, that is abstracted away through like a service abstraction. That way, if we want to pull out a third-party integration into its own service one day, we can do so. And that service maintains its own like data models. Everything is pretty packaged in some sense. That's something we learned definitely over time.

Because we are in fintech, we had like all these – We had like payment, payment schedules. If you're paying off a loan, you have the schedule. You have like the actual balances. You had all these different things about a user, that as it became tightly coupled, like a bug in one place affected a whole bunch of other things.

And so, obviously, the risk is not as high for us today, but taking those learnings and building it in a way that we can set ourselves up for success when it becomes more and more complex, which I think it will be. As we build out these rules, every rule in the scorecard has a set of complexity with it. So building that out has been – Having those abstractions I think is going to make our lives much easier down the road as you want to pull things out into more self-contained components. So that's one of the things we learned.

Another name, as I mentioned, is like type safety, like numeric thing. As we're building these rules, having type safety has been extremely powerful. And I don't think it's as important for us as it might've been in our previous jobs. But I think because we're used to that, because we think in that mindset of like, "Hey, let's think about the types first. Let's think about the API shapes. Let's think about what it looks like, and then go build it." That workflow has been extremely useful for us, especially now because of COVID. We were all remote for quite a while. And so being able to sit down, like having that engrained in our workflow saying like, "Hey, what are the types look like? Can we define the shapes first and then go build the implementation?" helps us actually, as we were remote, do things in a more streamlined manner. Like, "Hey, I can define the types." Then we'll split up the work, and so we're not really effective from like not being in the same place. We don't really need those ad hoc conversations as much as we're building things.

So because we came from this fintech background and then we liked type safety, that has directly affected our ability to work better as a remote-first organization as we were during COVID. So it was an unexpected thing, but turned out to be pretty positive.

[00:57:12] JM: What would you guys be working on if not Cortex?

[00:57:15] GD: You want to this one, Anish? We were doing a couple of things in our past lives.

[00:57:18] AD: Yeah. Yeah. No. We've built a lot of – I mean, Cortex is on our first triad building a startup. I think it was the first time we individually felt the pain point. And so it was pretty like good project to work on. I think in the past, we've tried all sorts of things. We started a real estate company where we bought fractional shares of real estate and sold it to our friends and family. We've attempted a meme company that was very short-lived. But I think – Yeah, it just kind of goes –

[00:57:45] JM: You said a meme company?

[00:57:47] AD: Yes. Yeah. Yeah. We did attempt a meme company.

[00:57:50] GD: Yeah. I think for anyone out there who's listening to this, it basically around like discover of memes. I think we have all these proliferation of like Facebook groups today. Like the zoo memes, like the college memes and stuff, and it's so hard to find good memes on Facebook. The only way for me to find them is to be added to a group and I'm like, "Okay. This is a great group that I think is funny." I don't think it's a good way of discovering new memes, say, like Instagram. Explorer is kind of on the front, but it's kind of muddy. So we were trying to come up with this like newfangled idea where you could discover memes that you enjoy with different topics and stuff.

So it was short-lived I think mostly because we are not – Our consumer app shops were a little weak, and we came upon Cortex as we were working on the meme app. And it deftly hit closer to home. And so we pivoted to Cortex pretty early on in the life's lifecycle of the meme app. But I still think there's something out there for the app. I do really believe there's a meme app out there willing to be built.

[00:58:51] JM: All right, guys. Well, it's been a real pleasure talking to you about Cortex. Anything else you want to add?

[00:58:56] GD: No. It's been great talking to you. We really enjoyed the conversation.

[00:58:59] AD: Yeah. Thank you so much.

[END OF INTERVIEW]

[00:59:09] JM: Your code is going to have errors, even code written by an amazing developer. And when bad things happen, it's nice to know that Honeybadger has your back. Honeybadger combines error monitoring, uptime monitoring and cron monitoring into a single easy to use monitoring platform for less cost than you're probably paying right now. Honeybadger monitors and sends error alerts in real-time with all the context needed to see what's causing the error and where it's hiding in your code so that you can quickly fix it and get on with your day.

The included uptime monitoring and cron monitoring also lets you know when your external services are having issues or your background jobs are going AWOL or silently failing. Honeybadger.io is 100% bootstrapped. It's a monitoring solution that's bootstrapped, and this is important, because a self-funded business means that they're priced simply. They operate the business simply and they answer to you, not investors.

Software Engineering Daily listeners can get 30% off for 6 months of Honeybadger, and you can simply mention Software Engineering Daily when signing up. They'll apply the discount to your account and you don't need a credit card. Thanks for listening, and thanks to Honeybadger for being a sponsor.

[END]