EPISODE 977

[INTRODUCTION]

[00:00:00] JM: Amazon's container offerings include ECS, which is the Elastic Container Service; EKS, the Elastic Kubernetes Service; and Fargate, a standalone container instance system. Through these different offerings, Amazon provides a variety of ways that a user can manage Kubernetes clusters and standalone container instances. The choice of which containerization system choose from depends on the needs of the user and the tradeoffs that the user wants to make on control and portability.

Amazon's container products have been designed in the context of a shifting competitive landscape. Kubernetes presents a potential long-term threat to Amazon's status as the most popular cloud provider. Properly responding to this threat has required Amazon to extend itself into the world of open source more fervently contributing to Kubernetes and having more conversations with customers to find out what those customers want around Kubernetes. Customers want the products to have a high quality user experience just like AWS, but also have the open characteristics of Kubernetes.

Abby Fuller is a principal technologist with Amazon who works on containers in Linux, and she joins the show to describe Amazon's perspective on containers and Kubernetes.

[SPONSOR MESSAGE]

[00:02:15] JM: If you are a SaaS or software vendor looking to modernize your application distribution to gain more enterprise adoption, checkout replicated.com. Replicated provides tools to deliver your Kubernetes-based application to enterprise customers as a modern on-prem private instance. That means your customers will be able to install and update your application just about anywhere.

Bare metal servers in a cloud VPC, GovCloud and their own Kubernetes cluster, vSphere. This is a secure way the your customers can use your application without ever having to send data outside of their control. Instead of your customer sending their data to you, you send your application to your customer.

Now, this might sound difficult and maybe you're not used to it because you're a SaaS vendor. You're a software vendor, but Replicated promises that recent advancements from tools like Kubernetes make it far easier than before, and the Replicated tools can help vendors operationalize and scale this process.

The Replicated tools are already trusted by noteworthy customers like HashiCorp, CircleCI, Sneak and many others. As a result, over 45 of the Fortune 100 already have an application deployed via Replicated in their infrastructure. That's a strong sign of adaption.

Go to replicated.com for a 30-day trial of the full Replicated platform. You can also listen to an interview with Grant Miller, the CEO of Replicated, that we did a while ago.

Thank you to Replicated for being a sponsor of Software Engineering Daily, and you can check it out for yourself at replicated.com and get a free 30-day trial.

[INTERVIEW]

[00:04:24] JM: Abby Fuller, welcome to Software Engineering Daily.

[00:04:26] AF: Thanks for having me.

[00:04:27] JM: When a software engineer is evaluating the different managed Kubernetes systems that are on the market, there's a spectrum for how much control the developer has versus the specific cloud provider managed part of the Kubernetes experience. So you have the basic Kubernetes experience and then you have the management layer on top of it. What are the components of that managed experience?

[00:04:53] AF: Yeah. I think what you're getting at is that I think Amazon in general talks about the shared responsibility model, and the difference I think between all those different management layers are how much of that is on us and how much of that is on you. I think it ranges from you can run your own Kubernetes and EC2 and none of that is managed by us. It's all done by you to EKS on EC2, which we manage the masters to something that we announced at KubCon a few weeks ago called managed node groups. That lets you automate the provisioning in lifecycle of your worker nodes, so your EC2 instances, and then we'll handle things like node updates and terminations to drain them from your application. Then it ends I think with something, EKS on Fargate.

We announced that at Reinvent a few – Last week. Feels like many weeks. But we announced that at Reinvent last week, and that's basically running your Kubernetes pods on Fargate capacity. So you handle everything only at the pods spec level. I think how you choose between that set depends on how much you want to manage. So if you're a Kubernetes operations expert and you want to manage everything down to the master nodes, you can do that. If you're just interested in using Kubernetes APIs and tooling but you don't want to have to worry about the workers nodes and the masters, you can run your Kubernetes pods on Fargate capacity.

[00:06:20] JM: You just gave a good description of how a software engineer or an architect should evaluate the different managed Kubernetes systems, but there's also the decision that a cloud provider has to make. So AWS for example has to make decisions in their design of a managed Kubernetes. Tell me about how AWS has assessed the different tradeoffs that could be made, the design decisions in making a managed Kubernetes.

[00:06:49] AF: Sure. I mean, assuming that I can figure out how to unpack that question. I think on our side, there are parts of Kubernetes that the community feels really strongly about, and I think that's the open source side and that's the community and pooling side. I think something that's been kind of non-negotiable as we've designed these pieces is that we never want to lose that. So we don't want to lose kind of the upstream part of Kubernetes.

I think a lot of our discussions have been around how can we make things like upstream Kubernetes work and the exact same way that it does if you manage it yourself, but with us managing a different percentage of the pieces for you.

[00:07:27] JM: As you mentioned, at Reinvent this year there was the announcement of AWS Fargate interfacing with Amazon EKS, and this was a pretty useful announcement because Fargate has always seemed like a pretty interesting and useful projects since it launched because it's the idea of running a single container in a managed fashion, which for many kinds of applications is really desirable, because like a hobbyist project doesn't need a full Kubernetes cluster. In fact, it's going to be really wasteful to have a full Kubernetes cluster if all you need is a single node.

But, of course, if an application becomes successful, you want to be able to scale up that single node to a complex distributed system and you want to have all the granularity there. The ability for Fargate to interface with EKS is pretty useful. What I'm wondering is what was the engineering work that was required to make that a reality? Because I'm sure that was part of the vision from day one, but there must have been some significant engineering work that was required to make it a reality.

[00:08:33] AF: Yeah. It's definitely the kind of project that we spend a ton of time thinking about and thinking through before we kind of brought it up there. It would challenge something that you said a couple of minutes ago on Fargate being for a single container.

Fargate I think is ultimately a capacity provider for EC2. So by using Fargate, it's the same EC2 instances, but we're managing them and all that means is that you define things at the pod spec for EKS or the task definition for ECS level does not limit you to running anything less than a full

distributed system. Anything that you can run in kind of upstream Kubernetes or on ECS and EC2 mode works just fine in ECS or EKS on Fargate.

Moving into kind of the architecture choices, not a lot I think that we want to share super publicly on that. I think someone, one of the software engineers that worked really closely on the Fargate and on EKS side. A couple of them did some really interesting, I thought, Twitter threads on kind of how we had thought about some of the process. One was an under the hood talk on AWS Fargate from [inaudible 00:09:46], and they talked about basically what happens when you call run-task. What happened under the hood on Fargate?

[inaudible 00:09:56] also did an interesting thread on how Fargate works, and that one of her kind of design tenants from the beginning is that you needed to be able to use your existing tooling. Any of your kind of higher level abstractions, like deployments or replica sets, they works as fine on top of EKS on Fargate.

Then from what the team has said, kind of the next piece of that is what else do we need to do to Fargate to make it work for things like EKS, and they talked through some of this at Reinvent, but ECS and EKS on Fargate work in similar ways. They get dedicated hardware virtualized environments to run on. So that means a dedicated ENI, which is attached to the customer VPC, dedicated ephemeral storage base, scratch base, and then Fargate is what handles adding up all the compute and network and storage.

The next part of it I think is the roles part of it. So how do you things like capabilities and permissions? EKS and Fargate uses a profile. The profile includes a name, the subnets that you launch that profile in, and a pod execution role, which is similar to the task execution role in ECS. That includes permissions that you basically need to run the pod. For example, pulling image from ECR or apply labels.

Those are a couple of the things that we thought about when we were building this. The thread was a good read. The Fargate under the hood talk is on YouTube already. So if you're looking for a deep dive from some of the engineers that worked on this, I'd recommend giving that a look.

[00:11:30] JM: All right. Well, we'll have to add those as some related links for this episode, the Twitter threads. Shifting back to the point of view of the engineer, the software engineer, the DevOps person, whoever is managing the container infrastructure for an enterprise or for a startup, and let's assume they're managing that container infrastructure on AWS. What are the common operational challenges that they're dealing with on a day-to-day basis?

[00:12:00] AF: Yeah. I think that kind of the first operational challenge that people tend to run into is that a lot of the work that goes in when you're setting up a cluster for the first time or you're just trying to run a single task or a single pod or you're trying to follow the workshops or the documentation is that they kind of end, and that you've set it up, you've pushed your image, you've pushed your task definition in your pod spec. You have your test application running on the cluster, and then it feels like all the kind of day two operation stuff is sometimes missing. I feel that's the biggest challenge, is that with distributed systems and containers comes lots of extra moving pieces. I think kind of the first hurdle is that how do I monitor and observe and scale all of these moving pieces in a way that actually lets me benefit from how these containers are working?

The first big challenge to me always seems like how do I get the right amount of information about kind of the health and how my application is doing and how can I use that to inform choices like scaling? I think that's kind of the big first one that people think about with the kind of one and a half thing that people are thinking about, being things like identity and access controls.

How can I make sure that following the principle of this privilege that all of these pods or tasks or containers or services have just the access that they need to do their job effectively but without anything else, because I think that's been one of the benefits of breaking down all these systems is that I don't have to just have this application can do everything even if it doesn't need that. I have lots of different moving pieces that only have to access kind of the bare minimum, and I think implementing that in production can be really challenging.

[SPONSOR MESSAGE]

[00:13:55] JM: Apache Cassandra is an open source distributed database that was first created to meet the scalability and availability needs of Facebook, Amazon and Google. In previous episodes of Software Engineering Daily we have covered Cassandra's architecture and its benefits, and we're happy to have Datastax, the largest contributed to the Cassandra project since day one as a sponsor of Software Engineering Daily.

Datastax provides Datastax enterprise, a powerful distribution of Cassandra created by the team that has contributed the most to Cassandra. Datastax enterprise enables teams to develop faster, scale further, achieve operational simplicity, ensure enterprise security and run mixed workloads that work with the latest graph, search and analytics technology all running across hybrid and multi-cloud infrastructure.

More than 400 companies, including Cisco, Capital One, and eBay run Datastax to modernize their database infrastructure, improve scalability and security, and deliver on projects such as customer analytics, IoT and e-commerce.

To learn more about Apache Cassandra and Datastax's enterprise, go to datastax.com/sedaily. That's Datastax with an X, D-A-T-A-S-T-A-X, @datastax.com/sedaily.

Thank you to Datastax for being a sponsor of Software Engineering Daily. It's a great honor to have Datastax as a sponsor, and you can go to datastax.com/sedaily to learn more.

[INTERVIEW CONTINUED]

[00:15:35] JM: I've read a blog post that you wrote about efficiently managing container infrastructure, and you included some suggestions and ideas for how to configure, essentially, garbage collection for container infrastructure. This was news to me, somebody who doesn't actually work with container infrastructure. Can you explain how the idea of garbage collection applies to managing containers?

[00:16:03] AF: Yeah. So that was an oldie but a goodie blog post.

[00:16:07] JM: Yeah, exactly.

[00:16:09] AF: Digging up from the post-Reinvent fog, if I'm remembering that blog post correctly, there is kind of –

[00:16:15] JM: By the way, I think that time gets dilated or compressed during conference season, but time definitely – The time space continuum definitely changes during conference season.

[00:16:26] AF: I feel like just like the rules of like physics and time don't apply in Las Vegas, and I think that that is the problem for me, is that it feels like every hour there, I lived a thousand years. Not to bring my eye cream on to this software engineering podcast, but like I feel like from the second I stepped off the plane to like the second I got back to Seattle, I'm like, "Oh my God! It's like I've been in the dessert." I was like outside in the sand for weeks, but it's not. Just being dramatic, but terrible.

[00:16:59] JM: No. No. That makes two of us. It was my first Reinvent, and it was a baptism by fire.

[00:17:03] AF: Yeah, and there are some good parts, right? I love the density of customers there. There's no better way to reach so many folks and hear what they're looking for and what they want us to build. There's no more efficient way to do it than this. From that perspective, that's great. I love seeing their reactions to announcements. I love hearing from customers what they want us to work on next, but Vegas itself is challenging.

We're turning to the blog post in garbage collection, which is one of my favorite topics. Kind of two parts to that, right? The first part I think of those posts was how to make more efficient container images themselves. I think a lot of folks when they first get started with containers or even some of us when they've been using them for quite a while, they treat them just like a VM or just like setting up an EC2 image, right? They put the whole operating system in there and it's not really an environment that's for their application. It's more of like a general compute environment.

I think one of the advantages that a lot of us can get out of using containers is that those images can be much more minimal, and that means a smaller attack surface. It means a deploy and build and download faster. A lot of that comes from being mindful about what we include inside the container image itself. For example, if I'm writing a Python application, maybe I want to use a container image that has all the Python dependencies, but doesn't have a lot of the kind of unnecessary tool chain things. So that's part of it.

On the garbage collection side, something that's kind of gone hand-to-hand with containers is a lot of us are deploying much more often, and say I make a container image and it's like 1.2 gigs. For a server, not that big of a deal because I probably deploy lots of other things. If I have a monolithic application that's 1.2 gigs, it's not that big of a deal, right? I deploy it maybe once a week, but maybe more like once a month or once a quarter. It's not that big of a deal.

With containers, folks are deploying often many times a day, many time an hour. If that leaves a little trail of 1.2 gig images behind it, that's not a super efficient use of space. Garbage collection in that context I think is about cleaning up after ourselves with containers. So making those images smaller, but maybe not keeping around a lot of unused, untagged container images.

I think there's a pretty fine line to walk on that one between I want to keep this kind of recent image, because a lot of the layers are the same and I want to use that as a cache. Caches are reinvention, but also between I don't really want to run out of disk space. Speaking as someone who's definitely been woken up in the middle of the night for running out of disk space, and then in true distributed systems fashion, then running out of disk space everywhere else also. The garbage collection in that article was about cleaning up after yourself. Not keeping around tons of blogs that you didn't need on the host itself. Not having all these unused container images hanging around.

[00:20:14] JM: How should someone who is using a container system like ECS, how should they configure their garbage collection, or what are the tradeoffs? Why wouldn't I just always turn this kind of garbage collection system on?

[00:20:29] AF: Yeah. So the good news is, is that because that article was an oldie but a goodie, a lot of the orchestrators have improved how they handle garbage collection since then.

With things like ECS, I can't remember the exact flag off the top of my head, and I've learned my lesson about furiously typing the fact check on podcasts.

You can configure how many images the scheduler will keep. You can garbage collect a little bit more conservatively or a little bit more aggressively. Kubernetes has a similar flag. Basically, how long and how many of these old images should I keep. The tradeoff I think is similar, right? So you always want to have in my opinion, which doesn't work for everyone. You always want to have some level of garbage collection. We don't want to be hoarders. But two kind of mitigating factors here. One, I don't want to garbage collect too much, because I might want some of those layers, right?

I don't want to maybe garbage collect all the way up until my second most recent image, because I'm going to take a little bit of a performance hit when that image deploys the next time, because it won't have any cache to build from. The second part I think is that beyond kind of I want to have some of this, a lot of us have moved towards more kind of throw away infrastructure as we've moved to a distributed systems, right? So if I'm setting everything up with the containers, then I'm not doing anything on the host. For something like Fargate, maybe it doesn't actually matter because I'm not managing any hosts to run out of disk space on. I'm managing all of those just at the container level and I'm not worried about the EC2 hosts. It's our thing to do if they ran out of disk space.

I think that that is – It's something to think about if you're running ECS or EKS or if you're running Kubernetes and EC2 or another sort of schedule on something like Fargate, you don't have to really think about it much at all.

[00:22:25] JM: Great summary. There are – Something I try to do with these conferences. I went to KubCon and then I went to Reinvent, and I don't go to many sessions, but I use the conferences and opportunity to walk around the expo hall, walk around the crowds, talk to people at lunch. Just kind of get a feel for what's in the water, and I think that's one thing that conferences are useful for.

It's almost like pull. You're taking a pull or a pulls check across the world of infrastructure. One trend that keeps coming up is kind of the question of who should be deploying Kubernetes. How

many Kubernetes things should they be deploying? Should it be fully managed? Should we have it be totally raw Kubernetes so that we have no “lock-in”?

There are all these questions around like how should a “enterprise” adapt Kubernetes, but I think the most – Honestly, for me, the most acute question is should everybody have a Kubernetes plan, or are there companies who, if they’re on legacy infrastructure, should they just kind of punt the question of replatforming to Kubernetes or containers and just stick to VMs, because business logic is too important and they don’t have the time or they don’t have the resources to do a replatforming effort to containers. Can you give me your perspective on this kind of pulls check and tell me if what I just said resonates with you?

[00:23:58] AF: Yeah. I think the part that resonates to me is that I think folks should always have a modernization plan. I don’t think that they should have like a tool specific plan. I think it’s about what kind of benefits do I need? What direction do I need to take my architecture in and less about I want to use this specific tool.

So an example for me might be that I think a lot of folks are looking at things like ECS or EKS on Fargate, because it removes one of those hurdles and it leaves you with just one, right? Just that I have a legacy application. I’d like to move it to the cloud. I’d like to run this in a more modern way, but it means that they don’t have to think about setting up all those EC2 instances or auto-scaling policies or anything like that, that they can just manage it at the Fargate container or pod spec level.

I think for me the important part is about having a modernization strategy, and I think where that pulls comes from is that folks hear about all these interesting challenges and developments with things like distributed systems, or Kubernetes, or ECS, or service mesh, and they want to get those benefits, and they’re not quite sure what the right way to do it is. There’s no one right way to run your application. It’s about what works for you.

If it were me, I guess, and I had all of these tools available to me. So Fargate wasn’t a thing when I worked at my last non-Amazon job. I think I would have started with Fargate, right? I would choose how I wanted to interact with my applications. So either ECS APIs or with EKS

Kubernetes APIs, and I would start running my infrastructure on Fargate so that I only had to think about the container piece and not about the EC2 piece.

Then as I gained more scales and more confidence and more kind of specific customize needs, I would move to EC2 mode when I needed it. So I would mix and match. For applications where I needed that extra customization, so say that I want to run a daemon process on a host, maybe I'd move to ECS or maybe I'd move to EKS. But for the applications where I didn't need that kind of fined-grained knobs and dials on the infrastructure, I think that I would just start with Fargate.

[SPONSOR MESSAGE]

[00:26:22] JM: When I'm building a new product, G2i is the company that I call on to help me find a developer who can build the first version of my product. G2i is a hiring platform run by engineers that matches you with React, React Native, GraphQL and mobile engineers who you can trust. Whether you are a new company building your first product, like me, or an established company that wants additional engineering help, G2i has the talent that you need to accomplish your goals.

Go to softwareengineeringdaily.com/g2i to learn more about what G2i has to offer. We've also done several shows with the people who run G2i, Gabe Greenberg, and the rest of his team. These are engineers who know about the React ecosystem, about the mobile ecosystem, about GraphQL, React Native. They know their stuff and they run a great organization.

In my personal experience, G2i has linked me up with experienced engineers that can fit my budget, and the G2i staff are friendly and easy to work with. They know how product development works. They can help you find the perfect engineer for your stack, and you can go to softwareengineeringdaily.com/g2i to learn more about G2i.

Thank you to G2i for being a great supporter of Software Engineering Daily both as listeners and also as people who have contributed code that have helped me out in my projects. So if you want to get some additional help for your engineering projects, go to softwareengineeringdaily.com/g2i.

[INTERVIEW CONTINUED]

[00:28:11] JM: Continuing that discussion of replatforming or modernization, the decision of whether to replatform on to Kubernetes or to whether to modernize or how to modernize, it often falls on the platform engineering team, and at these last few conferences, I don't know if this is some kind of sample bias or just variance or whatever, but I got the sense that there are more and more companies that are creating a platform engineering department or a platform engineering team. It's almost like an outgrowth of the SRE trend or something like that, and it's kind of a new role.

I think in the sense that platform engineering in the modern sense seems to mean like a selection of technologies. It's like you've got this huge buffet of cloud providers and then you have individual companies that you can buy some particular monitoring solution or logging solution that the company particularly emphasizes. It's like platform engineering. Some mix of technology selection and architecture and determination for how a replatform is going to be determined.

Do you have a sense for how big a company needs to be to start to spin up a platform engineering team?

[00:29:30] AF: Ooh! Interesting one. Maybe this is a controversial answer, but I think it's a similar job that we always have a different name for. Sometimes it's SRE. Sometimes it's DevOps. Sometimes it's – I don't know, a platform team, or an ops team, or an infrastructure team. I think they're all kind of solving similar challenges and the term is changing as the people change and as kind of the technology that we're working with changes.

I think a lot of it is kind of the same question folks have already always been solving, which is how do I make smart architecture infrastructure and tool choices. How do I make that scalable to enable my developers to move to production as quickly and as safely as possible and keeping our availability as high as possible?

I don't think that you have to have a team of a certain size to be thinking about those things, right? I don't think that's a function of team size. I think at a really small startup, that might just be a single person. It might be two people. It might six people. But I don't think that's a function of team size, and I would actually maybe spin that in a different direction, which is that I think everyone should be thinking about that at any size of company. Maybe when you get to a certain size, you have folks that just think about that. But Amazon very much has a culture of you build it, you run it.

I would actually say that the responsibilities that fall to an ops team or a platform team are really things that we should all be thinking about, right? How do I deploy quickly and safely and keep my availability up is something that everyone in every kind of area of software engineering, we should all be thinking about that? That if we build the service, we should run it and we should be responsible for keeping it up. Maybe we get some help around the tooling and the infrastructure and the platform itself and the tools that I have available from a platform or an ops team? But there they're really more like cultural changes that we should all be thinking about from the very beginning regardless of size.

[00:31:34] JM: Who should adapt a service mesh today?

[00:31:37] AF: I don't know. Does anyone have an answer to that?

[00:31:40] JM: You.

[00:31:41] AF: Me? I think service mesh kind of as a concept is something that we're still very much filling out as an industry, and I think it's coming out of folks running into similar kinds of challenges, right? Which do my services discover and talk to each other? Is there an easier way of monitoring all these sidecars? I think people are – Talking about service mesh is coming from a set of similar problems, which is basically monitoring and networking is tough and I don't want to do all that for distributed systems.

[00:32:13] JM: And security policy management.

[00:32:16] AF: And security policy management. That's a big one. I think it's coming out of folks talking about the same kind of challenges and running into some of the same roadblocks. I think it's really, really early for us to be talking about things, like standardizing service meshes. I don't think we know enough to know what the right standard is.

As for who should be adapting it, I think when you're running into challenges like that, like applying security policies across your infrastructure or how do monitor from a single place rather from every application individually, then I think maybe it's time to start looking at them. But I am very comfortable saying that I don't quite know. I don't know how to get this one right yet. I think the only way to get it right is by listening to the folks running into these challenges that are building distributed applications that are solving challenges like that and working backwards from there.

We have AWS App Mesh, which is a service mesh, but it is the first time on our side that we've run a product in the open like this. So App Mesh has a preview channel. Everything hits the preview channel before it hits general availability so that customers and developers can play with things and give feedback even earlier in the product development process, because we don't quite know what customers want. I think the right way, the way for us to build the right things is by working really closely with folks as they're solving these challenges to make sure that we're solving all the right issues.

[00:33:40] JM: So embedded in what you just said, there was a hint of skepticism over the sidecar model, and I know that one alternative to that is the library-based model. I think maybe one way to contrast those two is that the library required a developer to embed the service mesh or service proxy library in their application code. Then the sidecar model just requires the platform operator, which may be like a centralized platform team to just click that they want a sidecar container alongside every other service. So the tradeoff there is it can be the platform operator's duty, but then you have the additional overhead of all these sidecar containers. Is there another approach we could potentially have? Are either of these approaches worthwhile?

[00:34:38] AF: I think anything that you can manage is worthwhile, and I think my skepticism is not the sidecar themselves, because I think that we're solving a problem that's challenging? I don't necessarily want to manage all the sidecars. So if there's a platform team or a service that

can do the sidecars for me automatically, like an App Mesh, or service mesh, great. If it works, it works. But I do try to be mindful of giving people more and more kind of bits to manage in the interest of solving a problem that originally came out of having a lot of bits to manage.

So the library model is a way of doing that. I think people like X-Ray use SDK and a lot of monitoring tools have used that approach, which you instrument. Then agent picks up what you've instruments and sends it somewhere else. They are all valid, but I'm not sure that I have a one size fits all right answer here.

[00:35:34] JM: How does KubCon differ from AWS Reinvent?

[00:35:38] AF: Well, Vegas, as previously discussed, I think is a divergence there. The big difference to me, and I went to KubCon Barcelona this year and Reinvent, but the North American KubCon was two weeks before and I could not like spiritually make that work.

The big difference for me is that reinvent is about AWS, and I think that that's actually kind of telling to the ecosystem in general. So KubCon is rightly so about Kubernetes and it advances with kind of a single community and the set of tooling and products and companies that have sprung up around it. Reinvent definitely has folks talking about Kubernetes and EKS and ECS and Fargate, but it's also about AWS in general.

I think that folks when they go to build on AWS, it's because of AWS, and a lot of folks are using Kubernetes tools because of Kubernetes. I think kind of the range of what you cover at Reinvent is pretty vast and that it involves more I think than just the container orchestrator. You have folks from EC2, they're from networking, from database and storage to different kinds of instance types, to machine learning, to serverless, to identity and security. I think there's an incredibly wide range of Reinvent, and I think what I get out of that – I go to KubCon when I want to learn about advances in Kubernetes specifically. How are people deploying and managing their Kubernetes clusters?

I like Reinvent because it gives me I think a very wide view of kind of the whole shebang, right? Not just the container orchestrator that I'm using, but how do I manage my networking? How do I manage my identity and access policies? What are people thinking about for CVEs and

vulnerabilities and how to do secure isolation? The range is much broader, and I think that's the big difference for me, is that one is about a single community and one is about a giant cloud provider.

[00:37:49] JM: What reflections do you have from Reinvent 2019 specifically?

[00:37:54] AF: Yeah. A couple of interesting ones for me. I really enjoyed the focus this year on more under the hood talks. You saw in Werner's keynote on Thursday that Claire Ligori, who was a PE, principal engineer from the container side did a really nice deep dive under the hood on Fargate and Firecracker. We also talked about [inaudible 00:38:16], another kind of parts of the AWS infrastructure, and I thought that was – I really enjoyed that this year, because I think what a lot of folks want to learn is how and why we build the things that we do. I think that was also reflected in the launch of the Amazon Builder's Library. So that is a collection of articles from principal engineers and senior principal engineers about how we think about things like load shedding, and isolation, and shuffle sharding, and queues, and exponential back off.

I do really like what I think is the begging of more push from us on talking about how and why we build the things that we do the way that we do and in sharing some of that kind of hard won internal engineering knowledge. So those are a couple of my big takeaways. The one that did not surprise me was the interest on EKS and Fargate. I think that's something that customers had been really interested in.

It was an interesting reinvent from the container side, because in November of last year, actually during KubCon last year, we opened sourced all of our roadmaps. So for ECS, EKS, Fargate, ECR, app mesh, we ran all of our product roadmaps on GitHub. I definitely felt like I had a – I was pretty clued in to what our customers were looking for before we got to Reinvent, and it's nice closing that loop even more so that the feedback during Reinvent feels really targeted, like, "I see you're working on this, but what I'd like you to do is tweak this a little bit and bring us this other thing." I found that really interesting and I think it makes it more productive for all of us.

[00:39:49] JM: Yeah. I would say one thing this year has really been the increased granularity of what open source means and what the different options are for ways that companies can be successful in open source or perhaps debates. Whether it's debates around licensing or what is

the definition of open source? How religious do you want to get about it? It's like whatever your opinion I think that the exploration of all these different ideas is useful long-term to the community as a whole.

[00:40:23] AF: Yeah. I think a lot of what stands out to me for open source in general, it's a community that you can always get in other ways, right? That sense of we're building this thing as a group that works for all of us. We're solving similar problems and challenges and we're building something that solves that, or I can learn for other people, or act from other people, or I hadn't thought about running it that way. That's a huge benefit of open source for me, is being able to learn and listen to that community. I think it's a lot of why we open source the roadmaps, is we wanted people to feel like they had transparency and visibility in a community around something that hasn't always had that.

[00:41:09] JM: Abby Fuller, thank you for coming on Software Engineering Daily. It's been fun talking to you.

[00:41:12] AF: Thanks for having me.

[END OF INTERVIEW]

[00:41:22] JM: Being on-call is hard, but having the right tools for the job can make it easier. When you wake up in the middle of the night to troubleshoot the database, you should be able to have the database monitoring information right in front of you. When you're out to dinner and your phone buzzes because your entire application is down, you should be able to easily find out who pushed code most recently so that you can contact them and find out how to troubleshoot the issue.

VictorOps is a collaborative incident response tool. VictorOps brings your monitoring data and your collaboration tools into one place so that you can fix issues more quickly and reduce the pain of on-call. Go to victorops.com/sedaily and get a free t-shirt when you try out VictorOps. It's not just any t-shirt. It's an on-call shirt. When you're on-call, your tool should make the experience as good as possible, and these tools include a comfortable t-shirt. If you visit victorops.com/sedaily and try out VictorOps, you can get that comfortable t-shirt.

VictorOps integrates with all of your services; Slack, Splunk, CloudWatch, DataDog, New Relic, and overtime, VictorOps improves and delivers more value to you through machine learning. If you want to hear about VictorOps works, you can listen to our episode with Chris Riley. VictorOps is a collaborative incident response tool, and you could learn more about it as well as get a free t-shirt when you check it out at victorops.com/sedaily.

Thanks for listening and thanks to VictorOps for being a sponsor.

[END]