EPISODE 493

[INTRODUCTION]

[0:00:00.4] JM: Since Kubernetes came out, engineers have been deploying clusters to Amazon. In the early years of Kubernetes, deploying to AWS meant that you had to manage the availability of the cluster yourself. You needed to configure etcd and your master nodes in a way that avoided having a single point of failure. Deploying Kubernetes on AWS became simpler with an open source tool called kops, and that's short for Kubernetes operations.

Kops automates the provisioning and high availability deployment of Kubernetes. In late 2017, AWS released a managed Kubernetes service called EKS. They officially started supporting a Kubernetes service. EKS allows developers to run Kubernetes without having to manage the availability and scaling of a cluster. The announcement of EKS was exciting because it means that all of the major cloud providers are officially supporting Kubernetes, and we'll talk more about the implications of those in future episodes. But today we're talking about how EKS runs on AWS.

Arun Gupta is a principal open source technologist at AWS and he joins the show to explain what is involved in deploying and managing a Kubernetes cluster. We're going to talk about it agonistic of what it's like to deploy on AWS or a cloud provider and then we'll talk about what it's like to actually just to operate a Kubernetes cluster on a managed provider.

If you want to operate your own Kubernetes cluster, you have to figure out how to do logging and monitoring and high availability, and if you use a managed provider, it might be easier to do some of those things. If you're convinced that you want to use Kubernetes but you aren't sure yet how you want to deploy it, this is going to be a useful episode for you. We're going to compare and contrast managing your own Kubernetes cluster versus using a managed provider.

We also discussed how Amazon built EKS and some of the architectural decisions that they made. AWS has had a managed container service called ECS since 2014, and the development of ECS was instructive for the AWS engineers who were building EKS. Amazon wanted to make EKS available to integrate with both open source tools and the Amazon managed services. So it's kind of an interesting architectural problem and a unique architectural problem where Amazon just had to make it plug and play for both managed services, which are proprietary and open source tools. It's a fascinating discussion and I really enjoyed having Arun Gupta back on Software Engineering Daily.

[SPONSOR MESSAGE]

[0:02:53.3] JM: The octopus, a sea creature known for its intelligence and flexibility. Octopus Deploy, a friendly deployment automation tool for deploying applications like .NET apps, Java apps and more. Ask any developer and they'll tell you that it's never fun pushing code at 5 p.m. on a Friday and then crossing your fingers hoping for the best. We've all been there. We've all done that, and that's where Octopus Deploy comes into the picture.

Octopus Deploy is a friendly deployment automation tool taking over where your build or CI server ends. Use Octopus to promote releases on-prem or to the cloud. Octopus integrates with your existing build pipeline, TFS and VSTS, Bamboo, Team City and Jenkins. It integrates with AWS, Azure and on-prem environments. You can reliably and repeatedly deploy your .NET and Java apps and more. If you can package it, Octopus can deploy it.

It's quick and easy to install and you can just go to octopus.com to trial Octopus free for 45 days. That's octopus.com, O-C-T-O-P-U-S.com.

[INTERVIEW]

[0:04:24.9] JM: Arun Gupta is a principal open source technologist at Amazon Web Services. Arun, welcome back to Software Engineering Daily.

[0:04:32.0] AG: Thank you. Glad to be here again.

[0:04:33.8] JM: We're talking about Kubernetes today and running it on Amazon Web Services and I think we should first talk about the experience of the early adopters, because Kubernetes has been out for a while. People have been running it on AWS for a while. What was the experience of those early adopters before there was official support and service, a managed service that people could run?

[0:04:58.5] AG: Yeah, absolutely. If you look at the CNCF surveys, the most recent one came out a couple of weeks back and there it talks about how AWS is the primary platform by a significant majority on where customers are deploying their Kubernetes cluster. As a matter of fact, the number is 57% of Kubernetes deployments are running AWS. That's pretty cool. That means there's a lot of Kubernetes deployment that is happening on AWS.

Yes, we announced a managed service recently, but a lot of these actually really goes to the early trailblazers who have been working to add support for AWS and Kubernetes. Well, my personal huge props would really go to Justin in Santa Barbara who had been leading the AWS sig for the longest time. I mean if you think about the Kubernetes community structure, there are lots of special interest groups and Justin Santa Barbara has been running their sig for a very long time.

Chris Love was a coauthor for kops along with Justin Santa Barbara. Kops is a tool which we'll talk later, allows you to run Kubernetes cluster on AWS very easily. These guys would meet on a regular basis. They would talk about what is the functionality that needs to be done in Kubernetes from AWS perspective, "Oh, such and such new AWS feature has come in. How do we integrate that in Kubernetes?"

There was recently support added for network load balancer. That is very recent, but think about the early days where ELB, EBS, how would they be supported into stateful sets? Think about how would you do IM integration in Kubernetes. There are projects like Cube2 IM. It's really huge props to everybody out in the community. Then last but not the least, the Kubernetes community by itself, because there are a lot of people in the Kubernetes community that have been building the core Kubernetes infrastructure to make sure it runs on AWS. It's huge props to everybody out there who have been checking along and then contributing and then building support for Kubernetes on AWS.

[0:06:55.6] JM: When Kubernetes first came out, there was a choice between many different container orchestration frameworks and people didn't really know how this space was going to sort out. There was Kubernetes, Swarm, Mesos, there still is, and then the response from Amazon was to release their own container orchestration system, which was ECS, and which made complete sense at the time, because if you're Amazon, you have no idea which of these open source systems to bet on. Well, maybe it just makes sense to have an abstraction that is fully integrated with the AWS environment. Maybe if you have some context, take me back to that time. I know you weren't at AWS at that time, but what was that environment like for Amazon? Why did they choose to go with the managed service that was not — Like they didn't do, as far as I know, like managed Docker Swarm or managed Mesos. They went with their own thing. Why did they do that and why did that fit well for the people who wanted to run containerized orchestrated workloads?

[0:08:03.4] AG: Yeah, Amazon is a very customer-driven company. Everything that we do, 90% to 95% of our roadmap is driven by customers in that sense. So when Docker came to popularity about 3, 4 years ago and our customers started asking us that, "Okay. Give us a solution," and given this AWS, they wanted to be fully managed something, that, "Give me something managed where I can focus on my core competency, which is application building, and you do the undifferentiated, heavy lifting of managing the service cluster and all those things."

So if you go back when ECS or the EC2 container service was launched essentially, elastic container service now was launched, this service was launched as a fully managed service. Instead of picking — I mean this was too early in the game. There were no "winners" so to say, "Oh! This still seems to be the default platform that is everybody is flocking to."

It was too risky for Amazon perspective to pick a container framework and then we're going to bet on this all the way to the end. The important part is Amazon customers also like the super seamless integration with other rest of the Amazon infrastructure. So that's sort of the Genesis for — Think about as an elastic container service. Now given today, the situation might have been different. I don't know, but the way I look at it today, today Amazon or AWS particularly is about offering choices to customers. Now customers want a service that is fully integrated, fully managed, multitenant service, but is fully IM integrated, cloud vouch, cloud trail integration, code pipeline integration. If you weren't all of that, then you come to ECS or elastic container service, or if you want something that is more managed by customers or partially managed by customers where you need to run your own control plane, where you are responsible for your own version upgrade or at least partially, and where the IM model may not be fully comparable with Amazon taste. So there are pros and cons. You can still go with Kubernetes, but there are things that you get with ECS that you don't get with Kubernetes today.

[0:10:10.2] JM: And also vice versa, right? If I wanted to take advantage of the Helm ecosystem of just the one click package install, which Helm is this package manager for Kubernetes, I wouldn't exactly be able to do that on ECS, right? So on ECS I would get the fully managed tight integration, but I wouldn't get the Kubernetes community experience.

[0:10:35.3] AG: Correct, because the essential part to understand is Kubernetes has an API, that means the technology or the concepts or the resources that you create over there are like pods, services, replica set, and those other APIs that you artifact that you create as part of the configuration file. When you come to the ECS land, there you're talking about task services, task instances, and that's the language that you talk about.

End of the day, there is still a common layer between them which is a Docker container. If you are trying to bring a monolithic application to the container world at Amazon, you still have to containerize it, but once you have containerized it, you can choose either the ECS route or the Kubernetes route, and the benefits of each of them are different. As you said rightly, Kubernetes does give you a Helm capability, but then in Kubernetes you can package up your tasks and use a CLI to deploy directly to ECS as well.

[0:11:30.8] JM: If we're comparing the experience of somebody who is thinking about going with ECS or going with EKS, it's not like EKS has a superset of functionality of ECS. There is still some stuff that you would get. These are disjoint — Well, not disjoint sets, but they are intersecting sets of functionality. What kinds of companies, what kinds of enterprises who are still going with ECS despite the EKS offering, or is everybody going to EKS at this point?

[0:12:03.6] AG: At this point, the important part to understand is EKS is only limited preview. That means we are handpicking the customers and onboarding them on the EKS platform. It's not available for general preview. At this time, if you're looking for a managed container offering, then ECS is the only solution from Amazon today generally available.

Now, when EKS does become generally available, then there will be two solutions, and at that point of time, the customers will start looking at it, “Okay, what is it that works in their environment? What is it that makes sense for them? Do they want a hybrid environment where they were able to run it locally on their on-premise environment and in AWS cloud as well?” Then Kubernetes may be the right approach for them, or if they wanted something which is fully managed, then they will go the ECS route.

Now in general sense, if a customers is all in. The way I mean by all in is they’re using all of AWS tech. The integration with ECS is far more intrusive and far more detailed and far more battle tested to show you exactly how this thing works. For example, at this reinvent, McDonalds actually gave a talk and they were talking how they have built a system using ECS where they’re able to handle up to 20,000 requests per second using ECS. That’s sort of the scale that we’re talking about. When EKS goes GA, that scale, I have no doubt it will be bigger and better only, but that scale is still a TBT, at least unknown factor at this point of time.

[0:13:32.5] JM: Yeah. I mean as far as scale, like I did a show with Segment a while ago and actually I think that was a few years ago. That was really before any of the cloud providers had a managed Kubernetes, and Segment was using ECS and they had integrations with all these other Amazon services and they loved it because it’s just — They have a high margin business. They don’t want to think about infrastructure if they don’t have to. But it will be interesting to see like how their decisions change in the future. Maybe they’ll just stay with ECS or they could also spin up an EKS cluster and have ECS and EKS, I guess. If there are some Helm package they want to use specifically on Kubernetes, they can have that as a separate cluster.

When you talk to customers — Back in the day. I guess when I say back in the day, I mean like six, maybe 10 years ago. Maybe this is still going on in some places. There are still enterprises that question, “Should we be on the cloud or should we have partial cloud? How should we do cloud?” I wonder how this is going to shake out with Kubernetes? Are people going to have Kubernetes clusters that are — For companies that are old and they’re on-prem or maybe they’re half on-prem, half on cloud, do you think that they’ll have like local Kubernetes — Or on-prem Kubernetes deployments and cloud Kubernetes deployments? How do you think that these companies that have hybrid cloud deployments will manage their Kubernetes? Do you think they'll have on-prem and cloud deployments?

[0:15:02.0] AG: Another way to phrase this question is; if you were to self-manage your Kubernetes service, what does it involve? Because if you are running EKS, for example, then of course, Amazon is doing the heavy lifting for you, but now think about this, you are an application developer. Your expertise is an app developer. You know how your spring boot application works. You know how your tomcat behaves. You know what logs you want to collect. You know how your application scales, but are you an infrastructure developer? Do you know how to manage etcd? When you're setting up a Kubernetes cluster, should the etcd be collocated with master or not collocated with master? How many nodes of etcd? If you want to set up your local Kubernetes cluster, what kind of networking should you use? Should you use a default cube net? Should you use [inaudible 0:15:53.4]? Should you use flannel? If you're setting up a Kubernetes cluster on AWS, should I use a VPS? Should I use security group? I want to use a cluster auto-scaler? Is my Kubernetes master intelligent enough to say, "Oh, by the way, I'm receiving too many request and I need to scale up? And how should I scale up? What should be their criteria?"

To top it off, Kubernetes itself releases every three to four months. So if you are doing the self-management of a Kubernetes cluster, that means are you ready to upgrade your cluster and how often should you do that? What should the strategy be? Should I do the workers only? Should I do the masters only? Should I do both? And what is my downtime?

The point being, if you do self-management of Kubernetes, you do far more time managing Kubernetes as supposed to using Kubernetes. So my feeling is we'll see developers that, "Yeah, Kubernetes is a plumbing. I don't need to worry about it. I just want to continue focusing on my applications," and that's what our customers have told us. Everybody that we have talked to about is their number one line is run Kubernetes for me.

I don't see a reason why even their test and dev clusters would be running in AWS itself as supposed to doing anything locally. There would be reason, there might be cases where they would like to run it locally for if I'm on a laptop and I want to run mini cube and things, but anything meaningful I would be running on the cloud.

[SPONSOR MESSAGE]

[0:17:24.8] JM: This episode of Software Engineering Daily is sponsored by Datadog. We know that monitoring could be a challenge. With so many services, apps an containers to track, it's harder than ever to understand application performance and to troubleshoot issues. Built by engineers for engineers, Datadog is a platform that's specifically designed to provide full stack observability for modern applications.

Datadog helps dev and ops teams easily see across all their servers, containers, apps and services to monitor performance and make data-driven decisions. To get a free t-shirt and start your free trial of Datadog today, visit softwareengineeringdaily.com/datadog to get started.

Datadog integrates seamlessly to gather metrics and events for more than 200 technologies including AWS, Chef, Docker and Redis. With built-in dashboards, algorithmic alerts and end-to-end request tracing, Datadog helps teams monitor every layer of their stack in one place. But don't take our word for it, start a free trial today and Datadog will send you a free t-shirt. Visit softwareengineeringdaily.com/datadog to get started.

[INTERVIEW CONTINUED]

[0:18:52.0] JM: And let's just go through some of the things that people who do have to — For people who are choosing between running in a managed environment and running their own Kubernetes cluster, whether it's on their own machines or on raw EC2 instances or other cloud computing instances, the things that you have to keep in mind if you are deploying your own Kubernetes cluster, if you are keeping watch over your Kubernetes cluster, one of those is high availability. Explain what somebody would have to do to deploy their own high availability cluster. I guess it's worth even defining what high availability even means in this context.

[0:19:35.7] AG: Sure. Now, if I were to kind of open the Kubernetes architecture, there are already four key components that I would say that exist in there. There is an API server. That API server is what is basically listening for their requests. You say cube kernel, apply resource configuration file, and the API server listens to that request. It takes that request and it translates it to the Kubernetes backend and it stores it into etcd.

Now, etcd is a core state store for Kubernetes. There are other in-memory caches, etc., but etcd is the system of record. Anything you talk to an API server, it gets stored directly into etcd. That's that part of it. Then comes scheduler. Scheduler looks for the location. Using cube kernel, you said create a pod, that metadata and the state, everything is stored in etcd. Scheduler says, "Okay. There are parts that are aren't assigned to a node, because we have worker nodes that basically need to be running. So I need to assign a pod to a worker."

It also examines a state of the cluster, picks a node that has free space and actually schedules a pod to it, or basically binds a pod to it. That's the scheduler part of it. Then comes the controller. Now, controller looks at, "Okay. You ask me to do run a replica set." Replica set says, "I need to run three replicas of it." So the controller will batch the replica set resource and the set of pods based upon the selector and then it will make sure that the expected and the desired state really match with each other. That's sort of the four main structures essentially that are part of the Kubernetes architecture.

Now, if you think about the master node of Kubernetes that has the API server, controller manager and scheduler. Now, etcd by itself would be sitting with the master or away from the master. Typically, in most common language, when people talk about high availability of a cluster is they talk about how many instances of API server am I running? Am I running one API server? That means there's a single point of failure? Or am I running three API servers? Then I could round robin between those so that if in case one API server goes down, I can still be able to talk to the API server and create my resources.

Now, API server is pretty stateless, but controller manager and scheduler are a bit stateful. Even though they're talking to etcd for all the state information but they have some in-memory caches, etc., in-memory transactions that are running around. There is a process by which the controller manager and the scheduler can pick a leader if there are three masters running. They pick a leader. If the leader goes down, then another leader can be self-elected.

Then the last part is where you have an etcd server running, let's say etcd is running outside of master. You want to make sure your etcd is constantly backed up. The state is restored, because that's your system of record.

Essentially when you're talking about high availability of a server, you're talking about high availability of a master. There's an in-built rack consensus protocol by which controllers and schedulers are elected as a leader. There is a high availability of etcd server. How often is this backed up? That's sort of the aspect that people talk about in terms of high availability of a Kubernetes server itself.

[0:22:47.5] JM: Much of these has to be — Is it that as an operator, if you're operating your own cluster, you would have to make sure that all of these set up and they're communicating with each other properly and it's not like there is like a one click get this deployed on your own self-managed system. This is something that you — Basically, if you want to get the easy way out, you have to go with one of these managed service providers, which is why there are so many managed service providers.

[0:23:15.3] AG: The good thing is the interaction between the API server, the controller manager, the scheduler, the etcd, is all baked into Kubernetes itself, but whether your etcd is collocated with master or not collocated with master, that is an operator decision when you are deploying your cluster, and there are so many ways to spin up your cluster. If I'm spinning up my etcd cluster, what is the instance type should I choose? How big the database can grow? What if my database grows beyond the capacity of the instance type? How do I scale it? Those all factors is what you need to be aware of. The basic plumbing is automatically provided by Kubernetes for you.

[0:23:54.1] JM: That decision about whether to collocate your etcd with your master or not, what's the debate there? Is that because if you collocated them, you would have some correlated failures?

[0:24:06.4] AG: I have not particularly seen any data on this as such whether etcd should be collocated with master or not with master. One of the tools by which you create Kubernetes cluster on AWS is kops, is by far the most popular tool today. By default, when kops create a

Kubernetes cluster on AWS, it has the etcd node collocated on master. Now, there is also a standing poll request, which means that poll request is for — A standing issue, not a poll request yet, but a standing issue which says it should have an ability to spin up an etcd cluster outside of master node. One model is already supported. The other model, there is an issue for it, so hopefully somebody will send a poll request and it will get merged.

Then the important part to understand is when we are provisioning EKS as part of the managed service, there we are setting up etcd on a separate EC2 instance altogether. Essentially, if you ask for an EC2 EKS manage control plane, which we'll talk about later is that we give you three masters and three etcd instances in the backend. The reason we did it that way is because we think etcd is far too critical to be put on the master node. We always build highly available architecture at Amazon and this reduces a single point of failure for us.

[0:25:21.5] JM: Is that because you want to run etcd on a different instance type than you want to run these less durable master nodes?

[0:25:29.6] AG: Exactly, and that's the richness of AWS instance types essentially. If you look at the master, it's not really this contensive. Etcd is a database in that sense and you need something, a different type of instance type to run etcd essentially. The API server is just receiving inbound request and calling to etcd server to store the database. The CPU, the memory, the desk requirements are a bit different for the master and etcd.

Also, as I said earlier, in terms of the availability of it, if the master goes down, we don't want the etcd to go down as well. If the master goes down, etcd is still running on a separate node and it can continue to operate.

[0:26:08.2] JM: Yeah. What are the other core tenants of EKS? You just described some very interesting design decisions that I think give insight into distributed systems. I'd love to know some more of those. Some of the philosophical tenants of how the service is built and how those lead to specific design and implementation decisions.

[0:26:29.6] AG: Yeah, totally. I would say four core tenants of EKS or the elastic container service for Kubernetes. Well, the first one is EKS is a platform for enterprises to run production grid workloads. What that means is that EKS is a platform for enterprises to run — All of your production will get workloads. Some of the largest and the most innovative companies in the world are AWS customers that have given us feedback on what they need EKS to do for them. As a result, and we think we expect EKS aims to provide features and management capabilities to allow enterprises to run real workloads and in a very classical Amazon way at real scale, reliability, visibility, scalability and ease of management are our priorities. That's sort of the first tenant.

The second tenant, which is very critical, is to provide 100% native and upstream Kubernetes experience. What that means is any modifications or improvements that we make in our service must be completely transparent to the Kubernetes end user. This means if your existing Kubernetes experience and knowhow applies directly to EKS. If you are running your Kubernetes cluster on AWS on your own and you come to EKS, that experience should exactly match alike.

The third tenant is if EKS customers want to use additional AWS services, the integration should be completely seamlessly and eliminate any undifferentiated heavy lifting. We are focused on making contributions to projects that allows the customers to use AWS components they currently know and love with their applications in Kubernetes. At the same time, it's important that the freedom to choose off-the-shelf open source options as well.

For example, if they want to use cloud watch for all of their logging, sure, we'll provide integration for that. But if they want to use FluentD for logging and that works in standard Kubernetes world, that would be permissible as well.

Last but not the least tenant is the EKS team actively contributes to the Kubernetes project, and that is fundamentally important, because we're going to contribute to the Kubernetes upstream. As I said, it's going to be done in a very open source, transparent and a collaborative way. Those are sort of the main goal tenants of EKS.

[0:28:43.7] JM: Could you describe any ways in which those core tenants lead to difficult or interesting engineering decisions?

[0:28:52.0] AG: I mean EKS is a unique service in that sense. The way EKS is built essentially, we are offering a control plane, and the control plane we think, “Okay. The master is there. The etcd server and database is running,” and we’ll take care of it in a managed way. Everything else in that service, we have RDS, we have EMR, we have all sorts of open source offerings that are offered as a managed service, but those are more — We have taken the service and we are providing a managed service on top of that.

The way EKS is being built and because of other core tenants, we want to provide 100% native and upstream Kubernetes experience. One of the mind shift that’s happening at Amazon is where we have to make sure that anything that we are building is done in the open source. For example, now when we are looking at — The way networking works in Kubernetes is, for example, the CNI plugin or container networking interface plugin. Now, when we’re building EKS, we built a CNI plugin and released it in open source completely.

What that means is, yes, when you are coming to EKS and when you are creating a cluster, by default we will use CNI, but what that also means is, “Hey, by the way, the CNI plugin is out in the open. So now if you are building your cluster using kops, which is okay, but you can still use the same CNI plugin.

The ability to be consciously aware that, “Okay. We want to keep the core tenant that it needs to be 100% upstream experience. It needs to be a very native experience.” Anything that we build out has to be done out in the open source and in a collaborative way, because a lot of customers are very interested in this service, but they want the flexibility where they can run the cluster by themselves as supposed to using EKS as well.

[0:30:36.4] JM: I’ve done a few shows at this point about operating a modern Kubernetes cluster, and that is modern in the sense of late of 2017, early 2018. I’m sure some of these things will change in the next five months, maybe shorter. But let’s talk through some of these operational issues and how people do these on EKS or not. Just general patterns around Kubernetes and maybe you could contrast the approaches of how people might do this in the managed environment or the non-managed environment. We’ve got logging and monitoring, for example. What are the patterns that you’re seeing around logging and monitoring and what are the ways in which people are using open source stuff or the Amazon tools?

[0:31:22.5] AG: Yeah. If you're an Amazon customer and you are looking at a managed service, the way you look at logging is, "Okay. Give me something with cloud watch integration. If you need auditing. Okay, give cloud trail integration. You need something with authentication and authorization. Okay, give me IM integration. I need to build a pipeline. Okay, give me code pipeline integration."

That's typically how Amazon customers think, and that's one of the core tenants essentially is if you are an Amazon customer and if you are coming to EKS, we will do all of those integrations for you. If you're used to that, you will get all of that in a first class way in EKS itself. But what that also mean is, because we are keeping 100% upstream experience, what that also means is if you want to run EFK stack, which is elastic search, FluentD and Kibana, if you want to run that, you should totally be able to do that, because essentially the way EFK stack works is you deploy that as a [inaudible 0:32:15.3] set and you can use FluentD to collect your logs and then use Kibana dashboard. What that also means is if you need distributed tracing, sure, we will provide you an integration with AWS x-ray, but if you want to use Yager, we can totally do that. If you want to use Prometheus as a monitoring dashboard, sure, we can do that too, because that 100% upstream experience.

I think the key part is anything that Amazon customers are aware of in terms of these are the tools that I'm used to, we will do all of that integrations with them really well. If you're looking for something that you have been used to from the Kubernetes side of it, no matter where you come, from the Kubernetes side or the Amazon side, we want to make sure we take care of you well.

[0:32:59.2] JM: Does AWS have a service mesh yet?

[0:33:02.7] AG: It depends what you mean by service mesh. It means a lot of things to a lot of people. If we were to look at sort of the standard service mesh offerings, which is like Envoy or Istio or LinkerD and things like that, we don't offer that as a managed service today. There are a lot of examples by which you can install any of those service meshes essentially on top of Kubernetes, and because of the core EKS tenant essentially that we want to provide an upstream experience. You can have an EKS cluster and then you can deploy your Istio or Envoy as a sidecar over there and then bring your service mesh, use service mesh for whatever you want to do it for essentially on top of EKS itself.

[SPONSOR MESSAGE]

[0:33:55.1] JM: If you are building a product for software engineers or you are hiring software engineers, Software Engineering Daily is accepting sponsorships for 2018. Send me an email, jeff@softwareengineeringdaily.com if you're interested. With 23,000 people listening Monday through Friday and the content being fairly selective for a technical listener, Software Engineering Daily is a great way to reach top engineers.

I know that the listeners of Software Engineering Daily are great engineers because I talked to them all the time. I hear from CTOs, CEOs, directors of engineering who listen to the show regularly. I also hear about many newer hungry software engineers who are looking to level up quickly and prove themselves, and to find out more about sponsoring the show, you can send me an email or tell your marketing director to send me an email, jeff@softwareengineering.com.

If you're listening to the show, thank you so much for supporting it through your audienceship. That is quite enough, but if you're interested in taking your support of the show to the next level, then look at sponsoring the show through your company. So send me an email at jeff@softwareengineeringdaily.com.

[INTERVIEW CONTINUED]

[0:35:23.4] JM: Let me ask you. The service mesh category, do you think this is like the container orchestration thing all over again where now we’ve got multiple different service meshes, and we haven’t come to a consensus yet, people are going to be afraid to build on them or do you think that — Is that now what’s going to happen?

[0:35:42.7] AG: I don’t think that’s where we are, because service mesh resolves part of the problem. When you’re bringing your application, you need to containerize it. So that step is required no matter what. You need an API but which you should be able to deploy those applications. You need some sort of orchestration framework.

Now, let me give you a specific example. Let's say we are doing a blue-green deployment with Kubernetes. One of the ways people go blue-green deployment is you will create a deployment, which is blue. You want a new deployment, so you'll create another deployment. You'll slowly scale the green deployment and scale up the green deployment and scale down the blue deployment, and eventually the older one becomes zero replicas and the newer one is higher replicas. That works for a lot of people.

But the issue with that is let's say during your canary testing, you want to only direct 5% of the traffic during A/B testing. You only deploy 5% of the traffic through the B set of users. You don't have that capability in Kubernetes itself. Now, if you're using service meshes by using sophisticated mechanism, it provides you those mechanisms. Service mesh is not a solution by itself, but it burns very well on top of this orchestration framework essentially.

[0:36:59.6] JM: My question is more like the service mesh provides these abstractions around gathering data from each of your different instances through the service proxy and it gives you a control plane for doing communications with your services. For example, you said this canary example where you could go to your control plane, whether it's an Istio control plane or a conduit control plane or a LinkerD control plane and you could say, "Hey, send 5% of the traffic to this set of instances and send 95% of traffic through this other set of instances."

You could imagine higher level abstractions being built on top of those control planes, and I guess I was just wondering if there's like race to be the platform for the control plane, but maybe that's just me like thinking about this as a competition, when in fact it's not a competition.

[0:38:00.6] AG: I think the key part is think about three years ago when we were looking at the container orchestration frameworks. There were so many of them. I'm not saying a particular framework has won the war, and different frameworks meet everybody's different needs. And I think that's where we are today in service meshes.

I see that there are going to be different service meshes that will have their own certain set of features. Yes, there will be a common set of features, but each service mesh will have a specific feature that will cater to a particular audience. It's too early in my opinion to call out that, "Oh, service meshes are an abstraction and they're going to make the container orchestration redundant."

It's too early that shake up needs to happen. More service meshes to come out in my opinion and they need to really fight each other and figure out what is going to be the right pattern. Is service meshes the right pattern? I don't know. Maybe there is something else at a higher abstraction that needs to come out.

[0:38:54.5] JM: When we talk about a company that is using Kubernetes, do they typically have their entire application on one Kubernetes cluster or are they deploying multiple clusters?

[0:39:06.7] AG: Right. We have seen different examples, different ways developers are building these clusters or operators are building these clusters, and there are multiple topologies. There is no one size fit all. We have seen customers who are building large clusters. In that large clusters they're using name spaces to allow you virtual clusters, because with name spaces you can start qualifying that how many resources, how many parts, how many services, how much memory, how much CPU is available to you. You can start kind of sharing a large cluster between different teams essentially.

We've also cases where they're building separate clusters. They're completely want to isolate an application that this application is running on this cluster. No matter what, if this cluster goes down, nobody else gets impacted. So I'm just going to build out five or a 10 or a 15 node cluster here, run my application here. If I want to build a different application, I want to spin up a new Kubernetes cluster and that's where I'm going to deploy my application tool.

The important part in that though is to automate the cluster creation. That means you have identified, you have done your cluster planning. You have done your cluster sizing. You know what is your instance type is going to look like. You know how big your container is going to be. You allocate that much memory, that much CPU, and then that planning is done automatically. You know what networking you're going to use. You know what topology you're going to use. You know how many masters are going to be. You know all those details. You have scripted them well, so that in case you need to spin up a cluster, you don't really have to wait for days to spin up a new cluster, but it's really a script. You just run it and in a few minutes the cluster is up and running. We have seen both topologies and I see how both could work for the customers essentially.

[0:40:51.3] JM: Let's take the example of storage. I did a show recently with the Rook project, and with Rook it's this idea of having storage on Kubernetes. Basically managed storage system, and he was talking about two models of doing storage. If you wanted to have a Kubernetes cluster that was managing your own storage — If you want to have a database that you want to manage yourself through Kubernetes, you could, for example, deploy an Amazon EKS cluster and you could have a Ceph Filesystem running on that cluster or you could have a database running on that Kubernetes cluster that is backed by something provided by Rook.

He presented two models where in one model you could have a Kubernetes cluster that is managing all your operational stuff and then you've got another Kubernetes cluster that manages all your storage stuff and you can attach EBS volumes to that, for example, Amazon EBS volumes, and then you present another model where you have a cluster and you would just attach all the storage directly to the cluster even while that cluster is doing operations. Have you thought about this at all? What would be a motivation for having your storage on the same cluster as your operations versus splitting them up?

[0:42:08.7] AG: Yeah, I've looked at the Rook project, and I think one of the things about Rook is I think it only works with Ceph today. That's one of the things, and I know there are discussions around how it could be possibly added more storage providers. That's one part of it.

The second part of it is I think Rook is also into discussion of being a CNCF project. Hopefully a better integration with Kubernetes will come along. Now, two things I want to say. First is, typically when you are thinking about storage on Kubernetes, whether it's block storage or a manage database or something as simple as S3 or a file system, you think in terms of stateful sets. You don't really think in terms of — You think in terms of stateful states, and the way it is being provisioned is using persistent volume and persistent volume claim.

Now, that's where the role between operator and dev kind of sits in there. The operator says, "Okay. Here is the consistent volume that I've created for you." How that consistent volume is provisioned and [inaudible 0:43:06.3] that is sitting on the — Collocated on the host, or is sitting on a network attached storage. It's completely transparent to you. Here is your resource identifier which you can use in your application and assume that this kind of a storage or what about the quality of the storage is being given to you, we can honor that.

I think that abstraction in Kubernetes is really cool, and that's what I've seen a lot of customers are using. Now, if you're building a cloud native storage, in certain cases you may want — Like if you're deploying a database, for example, you may want an EBS attach volume right there with your host itself and you want to make sure that database is able to write to the host, because if it is run on a network and depending upon where that EBS volume is provisioned as opposed to your host, there could be a bit of a latency between writing from a database, writing from the container to the database.

I think things like those is what comes into my consideration whether I want an attached storage, whether I want a network storage. The most important part is, because you have the concept of a stateful set, it really simplifies that abstraction from the developer perspective.

[0:44:11.3] JM: When a team has deployed Kubernetes, whether they've just migrated to Kubernetes or they are spinning up an entirely new project with Kubernetes, what are the operational challenges that you see people dealing with on a day-to-day basis?

[0:44:27.5] AG: Yeah. I think the key part, we talked about some of them earlier already. When you are creating a cluster — I think if you are spinning up your own cluster, it pretty much talks from the day one itself, from the beginning itself, "Okay. I want to create a cluster. How many masters should I have? What [inaudible 0:44:46.1] of availability would work for me? I want one master, or two master, or three master. It's always recommended to have odd number of masters to avoid the split-brain problem."

Okay. I will go with — I need high availability, so I'll put three masters. Okay. Now, what is my instance size going to look like? Is that instance size, if I'm attaching ENI to it, that instance size is limited by the number of IP addresses that can be assigned to it. Okay. What networking provider should I use? Should I use Wave? Should I use flannel? Should I use a default cube net? Should I use an AWS CNI provider? What gives me what? Is etcd collocated or is etcd sitting separately? If etcd is sitting separately, how often should I back up my etcd cluster? How often should I take a snapshot? What is a good plan? Then you start Googling for those instructions.

Now, that is a day one problem, and Amazon is a day one company. We're always like excited about our customers as day one. But with Kubernetes, I think what happens is a day two scenario, which is more entrusting. All right. Now, your cluster is up and running. Now, a new version of Kubernetes comes in and you want to really use the feature that is out in the new version. How you're going to operationalize it? Should I spin up a new cluster? Deploy my applications over there and flip a switch at the DNS level? Or should I do in-place upgrade? I'm looking for experience on people who have done in-place upgrade and I only hear war stories or horror stories. Is that a good thing for me?

I think those are the kind of questions that start coming to your mind when you're operating it. I heard federation of clusters is a good thing, but it is an alpha feature and it's not going to supported in EKS, because it's an alpha feature. Should I use federation of clusters? Or it worked in this case, but not in this case. I don't know what's going to happen in my case. I think there are a lot of uncertainties or a lot of decisions that you have to go if you are running your own Kubernetes clusters or the operational challenges that you need to be aware of, and I think that's the value that EKS provides. We let you focus on your application development as supposed to figuring out, "Okay. What is the instance type? How is it going to scale? How is it going to integrate with other technologies?"

[0:47:01.6] JM: What about people who are on a managed Kubernetes provider? Obviously being on a managed service will help, but there are still decisions that need to be made around being an operator. What are the operational decisions? When we're not talking about, "Is my cluster high availability, or how am I going to do updates?" What is the life of a Kubernetes operator?

[0:47:23.8] AG: As a Kubernetes operator, I run Kubernetes clusters in a regular basis, but essentially if I have a managed service, then the API server is sitting for me. Essentially, for EKS for example, we will give you an ability to say automatically upgrade, which will take care of the CVEs and the patches and the security patches for me automatically.

If that process is seamless, I'm good to go actually. From my side, all I need to worry about is, "Okay. I got an application up and running here." I have an application developer who's writing application in Java or in node or whatever language of their choice is. Once they have written the application, how do I help them get that application running on the Kubernetes cluster? Then you start looking at the developer tool chain eventually over there. All right, you wrote your application in Java using Maven, for example. Let me take you this fabricate IO Maven plugin which you can use essentially to convert your Java application on a JAR file so to say to a Kubernetes configuration file and also give you the ability to deploy directly to a Kubernetes cluster, because essentially it means do we really integrate it into the tool chain as supposed to, "Oh! Learn these three new scripts in order to take your application and deploy it to Kubernetes."

Once the heavy lifting of managing the service is done, then we start on focusing on really the application development tool chain. As a matter of fact, that's where people [inaudible 0:48:49.5] thinking. They should not think about, "Oh! I need to operate a cluster, or I need to a manage a cluster." Let EKS do the heavy lifting for you. Let this managed services do the heavy lifting for you and you focus on your application using Kubernetes as supposed to managing Kubernetes.

[0:49:06.3] JM: Okay. Just to close off, as you have alluded to, Amazon is kind of changing its philosophy around open source. So what is changing there? Since Amazon is starting to contribute more readily to open source, how is that philosophically impacting the AWS culture?

[0:49:25.6] AG: I think it's a very fundamental question here. As I said earlier in the very beginning of the call, AWS is a very customer-centric company. We want to be the most customer-centric company on the earth. That's sort of Amazon's logo anyway. Our customers have been asking us to run Kubernetes for them. Our customers have been asking us to contribute to Kubernetes out in the open to make sure it stays relevant and successful.

Once again, it is very driven by customer and it's a very top-down initiative and it's a very bottom-up initiative as well. There are a lot of developers in Amazon. We have been doing open source for a very long time. What a lot of people don't realize is there are about 800+ GitHub repos if you look at different Amazon repos. We have been using a lot of open source projects and we have been contributing to a lot of open source projects.

Now, Kubernetes just by the virtue of developer mindshare it is, it's probably going to be one of the most visible parts where we are going to be contributing, and I think that's what it comes down to. There are a lot of motivation inside Amazon to contribute not just to Kubernetes, but to other projects as well.

[0:50:33.5] JM: Okay. Arun, well it's been great talking to you once again on Software Engineering Daily and I look forward to talking again in the future.

[0:50:38.8] AG: All right, thanks a lot. I had a good time.

[0:50:40.4] JM: Okay. Thank you.

[END OF INTERVIEW]

[0:50:46.8] JM: GoCD is an open source continuous delivery server built by ThoughtWorks. GoCD provides continuous delivery out of the box with its built-in pipelines, advanced traceability and value stream visualization. With GoCD you can easily model, orchestrate and visualize complex workflows from end-to-end. GoCD supports modern infrastructure with elastic, on-demand agents and cloud deployments. The plugin ecosystem ensures that GoCD will work well within your own unique environment.

To learn more about GoCD, visit gocd.org/sedaily. That's gocd.org/sedaily. It's free to use and there's professional support and enterprise add-ons that are available from ThoughtWorks. You can find it at gocd.org/sedaily.

If you want to hear more about GoCD and the other projects that ThoughtWorks is working on, listen back to our old episodes with the ThoughtWorks team who have built the product. You can search for ThoughtWorks on Software Engineering Daily.

Thanks to ThoughtWorks for continuing to sponsor Software Engineering Daily and for building GoCD.

[END]