EPISODE 353

[INTRODUCTION]

[0:00:00.3] JM: Kubernetes makes it easier for engineering teams to manage their distributed systems architecture, but it's still not simple to deploy and operate a Kubernetes cluster. Sometimes there're outages for reasons that you may not understand and you need to dive down into the code, and that's complex. Google Container Engine is a managed control plane for Kubernetes. Just as developers can use Google App Engine or other platform as a service tools, like Heroku, to deploy simple monolithic apps against a platform as a service. We can use Google Container Engine to deploy microservices against a platform as a service.

David Aronchick and Chen Goldberg worked on Google Container Engine, and they join the show to explain why platform as a service container engines are useful. Google is not the only cloud provider with a container engine. Amazon ECS and Azure Container Engine also allow you to run containers in a managed environment.

Software Engineering Daily is looking for sponsors for Q3. If your company has a product or service or if you're hiring, Software Engineering Daily reaches 23,000 developers listening daily. You could send me an email at jeff@softwareengineeringdaily.com and I’d be happy to share more information on sponsorship with you. Thanks for listening.

[SPONSOR MESSAGE]

[0:01:35.3] JM: Do you want the flexibility of a non-relational, key-value store, together with the query capabilities of SQL? Take a look at c-treeACE by FairCom. C-treeACE is a non-relational key-value store that offers ACID transactions complemented by a full SQL engine. C-treeACE offers simultaneous access to the data through non-relational and relational APIs.

Company’s use c-treeACE to process ACID transactions through non-relational APIs for extreme performance while using the SQL APIs to connect third party apps or query the data for reports or business intelligence. C-treeACE is platform and hardware-agnostic and it’s capable of being embedded, deployed on premises, or in the cloud.

FairCom has been around for decades powering applications that most people use daily. Whether you are listening to NRP, shipping a package through UPS, paying for gas at the pump, or swiping your VISA card in Europe, FairCom is powering through your day.
Software Engineering Daily listeners can download an evaluation version of c-treeACE for free by going to softwareengineeringdaily.com/faircom.

Thanks to FairCom c-treeACE for being a new sponsor of Software Engineering Daily, and you can go to softwareengineeringdaily.com/faircom to check it out and support the show.

[INTERVIEW]

[0:03:15.7] JM: David Aronchick¸ product manager at Google, and Chen Goldberg who is director of engineering at Google and Google Container Engine and Kubernetes. You guys are both joining us. We're going to talk about Kubernetes, we're going to talk about Google Container Engine, and open-source, and the future.

Let's start with the recent past and the present and just catch some people up on Kubernetes and containers. Explain why containers and Kubernetes are useful, why there is so much traction in this community.

[0:03:51.0] DA: First, thanks so much for having us on. I should start without, I suppose. I think what we've seen in the container and Kubernetes community is that people are really desperate to make their developers faster. For the longest time, you would have developers that would have to understand every layer of the stack all away from external networks, to the way their applications rolled out, to the underlying infrastructure, the configurations of VM's or Oss, and down to the hardware. Sadly, that led to a lot of issues around dependencies, around having people have to move a lot slower than they would normally run into, that they would have conflicts as they would roll out these applications.

The idea behind containers which certainly predates Google, but Google has been a very big investor in over the past 15 years, is that we wanted to separate those concerns. We wanted to let developers focus on just the things that were necessary for their particular application to describe the run state of their application in exactly the same way they would describe any other configuration as part of the build process and to roll it out without having to be concerned about the other dependencies and permissions and things like that that might be on the operating system. Containers provides all of those things.

Now, that said, just going slightly beyond that, running a single container is very easy. When you start to build distributed systems and spread them out across hundreds, thousands, tens of thousands of machines, that’s when things become very hard. Is the container that I rolled out here, is it going to conflict on a port level? Is it going to conflict on a disk level? Is the application across the fleet running in the configuration that I said? I wanted 431 copies of this container running across my fleet. Are all 431 out there? If I'm going to spin up a second application and have it communicate with those 431, how do I do that?

That's where you need an orchestration system to step in and help to manage a lot of those things. That's really why Kubernetes has risen to such great use and had seen so many people out there using it is because while running a single container is easy, all the things necessary to bring your application to production come in the box of your orchestrator and that’s what we provide with Kubernetes.

[0:06:24.3] CG: If I may add, definitely we see that the adoption of containers and container orchestration is about the agility and engineering productivity, but what we see now, again, is something that Google is using containers for is about resource efficiency which allows you to schedule in a smart way different type of workloads using the same resources, and this is what Google has been doing so successfully running services like Gmail, and Maps, and even Google cloud platforms is running on containers. I expect that in the future we’ll see more and more incentive and motivation of people to adopt this technology because of that.

[0:07:05.5] JM: Okay. So we've motivated why Kubernetes is useful. Of course, it also creates some complexities when you adopt Kubernetes, and I want to explore some of these complexities because that will motivate a discussion of why people might want to adopt a container engine that will help them manage the Kubernetes with a platform. What are some of the difficulties in setting up my own Kubernetes cluster?

[0:07:36.1] CG: I think that actually setting up your cluster and kicking off the tire is something that is not how this one might think. I know that that was the perception of Kubernetes probably a year ago, but Google ended and the rest of the community invested a lot in making that easier. Today, there are tools to allow you to spinoff a cluster very easily. You can do it with a Google Container Engine and other tools like KOPS and minikube if you want to do it on your own work station.

The challenge is what happens next. For example, how do you keep your cluster up-to-date? How do you update your services? You want to do it in a —

[0:08:13.6] JM: It’s not as much about the setting up at first part. It's about the operations once you have it set up.

[0:08:20.0] CG: That's one aspect of it. The other one I think is also how you connect your Kubernetes cluster into the underlying resources. Kubernetes, we need to remember, runs anywhere, so you can run it on Bare Metal or on open-stack or in any of the cloud providers, and that also can be challenging.

[0:08:38.7] JM: Okay. Let's dive in a little deeper into some of those operational difficulties of running your own Kubernetes cluster. For example, if I'm running a Kubernetes cluster of a bunch of different applications or a bunch of different sub-operations of an application, I might have different pieces of software that need to be updated across my cluster or Kubernetes itself. I might need to update Kubernetes itself. How does a software update work on a Kubernetes cluster, Chen?

[0:09:09.9] CG: I think it depends on two things where your Kubernetes is running. If you are running Kubernetes in a cloud or maybe on another virtual environment and it's easy for you to spin-off another node maybe. That would be one way to do it so you can — Talking about a Kubernetes version itself, you can upgrade the master. We do have a backward compatibility of the notes to the master so you can have a difference of two versions between them so you can do an upgrade of the control plan.

Later on, for example, you can drain the node, kill it, spin-off a new node and create pods and Kubernetes itself would just make sure that you have enough replicas as you would expect. That's one thing. If you're doing it, of course, on Bare Metal, that becomes more challenging because sometimes it's not an easy job to do it and you need to make sure that you won't have any downtime for your application.

[0:10:07.3] DA: One thing to think about as you think architecturally for about Kubernetes, again, the idea of separation of layers, separations of concerns. There's basically three key layers that you're always going to be thinking about here regardless of where you're running to Chen's point. You have the OS — Actually, we'll call it four. You have the hardware layer. If it's a cloud, then you don't have to worry about that at all, but that's something you may be concerned about is if a disk goes bad, memory goes bad, something like that.

Then, on top of that, you're going to have an OS, that's going to need updating, maintenance, paying attention to. On top of that, you'll have the actual Kubernetes components. Kubernetes works in a master and minion mode, so there is a Kubernetes master that has a set of things, an API server, etcd, and a variety of other things on it. Each of the nodes also has a set of components on it, not an enormous amount, but it has something called cubelet which actually listens to the API server. It also has a version of Docker that's running there and some various other scripts and things that you'll need to keep up-to-date. On top of that, you'll have your applications.

The beauty of containers, the beauty of Kubernetes is that if you're developer, you really only need to think about the top layer. That said, no matter how big or how sophisticated your organization, you're still going to need to think about those other layers. Someone in your organization is. We at Google — I'm not going to use real numbers, of course, but for every, call it one person, that is working on the kernel that we use here at Google and the OS image, you'll have, call it 10 people, working on our internal version of Kubernetes, called org, and then maybe you'll have 1,000 people who're actually at that application development layer. It's how you think about maintaining all those layers overtime .

That said, to Chen's point, there are great tools out there, a lot of things out there to help make this much easier. Something that we take pride on with Google Container Engine is we take care of all the layers of the stack for you, and so you get to your application. We take care of automatically updating your OS for you. We take care of automatically repairing your OS for you. If you opt in, you can do it yourself if you'd like, but that's certainly an option for you.

We also give you a secure OS that is locked down in the way that we would run here at Google further reducing your overall service area, and its extreme efficient. It is literally optimized for containers. It is called the container optimized OS, and it's specific to our cloud. Going 1° slightly beyond that, again, something that we do at Google which we highly recommend go and read the SRE book. It's free online. It talks about how we do things at Google.

You talked about updating, and something we do at Google is we'll do these exponential rollouts, we'll canary rollouts of our own applications and our underlying infrastructure. That's something on Google container engine that we make very very easy.

As Chen said earlier, something you can do on premises is you can create a second node, you can join it, and then make sure things are working properly and then spread your workloads across it. On Google Container Engine, we have a very sophisticated tool called node pools which allows you not just to create a single set of node of a new version, but actually an entire pool of those nodes. Then spinning up and joining new nodes and bringing down your old notes is as easy as single command plus one here or minus one over there. Very very sophisticated and it's really based on the operational experience we have at Google. We want to enable people running a Google Container Engine to that.

Again, one of the things we're very proud about is that anything we do here, we're not forking, we're not doing anything special. We're using Vanilla Kubernetes, and you can do it yourself. That said, we want to take care of that for you. We want to make it click button easy.

[0:14:01.6] JM: The idea is that with Google Container Engine, this is a way to experience Kubernetes where you can go down into the depths of it and configure what you want to, but if you don't want to do that then you can treat it as more of a black box and have it take care of a lot of the operational burden for you. It's sort of like a platform as a service, like a Heroku, or Google app engine, where a lot of the frustrations are taken away from you.

[0:14:34.6] DA: It’s kind of interesting. In a lot of ways, that's how we love people to think about it. Really, Google Container Engine really fits right down the middle of the two ends of the spectrum. You talked about Paz before, you have Google app engine, Heroku, many other excellent providers out there, and they don't let you even see the existence of a VM, right? They just take your code and run it. You don't know that you're running a container. You don't know that you're running a piece of code, or you have no idea what the VM is what, et cetera.

Then on the way other end of this spectrum you have totally raw IS, infrastructure as a service, where you have an OS and a package manager and it's up to you to fill in the details, missing things. Google Container Engine really splits the difference. To your point, we don't allow infinite configuration. We have some strong opinions about the best way to run Kubernetes and we’re going to do that for you. We’re not going to make it hard and we’re not going to fork in order to accomplish that. You could still run Kubernetes on raw VMs, but we're going to do it in a certain way. If you do it, if you can adapt your workload to the way that we suggest, we feel like you’re going to have the best experience.

I want to be honest here. The number of things you can't configure about your Kubernetes cluster running on GKE is quite quite small. We’re talking about things like kernel modules and things like that which most people, the vast majority, don't need.

That said, if you run in that way on Google Container Engine, we can do what an enormous amount of things on your behalf, ensure stability, ensure reliability, uptime performance. To Chen’s point earlier, we see people running on Google Container Engine two, three, four times more efficient than the exact same workloads running on GCE or other VMs that we’ve communicated with. It's like you’re a quadruple the size of your data center for free.

[0:16:28.7] CG: If I may add to that. I think everything David said is the key reasons why people are using Google Container Engine so they will not need to worry about those aspects. We also see more sophisticated users and they appreciate that Google Container Engine is powered by Kubernetes, which is technology they know and understand because they probably use it somewhere else.

Many of them, they don't control everything, but they do want to understand what's going on, so they'll be able to debug and understand how their system is behaving. They talk about observability, and this is something that is very easy to do with Kubernetes. For example, linking the logs to Google platform log system and just by knowing that technology of open-source, it allows people to do that. They can choose the do-it-yourself or just go into the customer and see for themselves what’s going on.

[0:17:23.5] DA: That is an absolutely terrific point. One of the key visions of Kubernetes is really freeing people from lock-in entirely, and that’s one of the reasons why we haven't made any changes whatsoever that would be incompatible. If you're a large customer and you have a large data center or you have an existing contract with the cloud provider, it is trivial to add additional Kubernetes through Google Container Engine or a second cloud provider, get yourself almost instantaneous disaster recovery, portability, migration. Because the version of Kubernetes that runs at Google Container and then the same that runs at the one everywhere else, you can be assured that you know how it works and that your applications will work properly.

[SPONSOR MESSAGE]

[0:18:12.3] JM: Ready to build your own stunning website? Go to wix.com and start for free! With Wix, you can choose from hundreds of beautiful, designer-made templates. Simply drag and drop to customize anything and everything. Add your text, images, videos and more. Wix makes it easy to get your stunning website looking exactly the way you want. Plus, your site is mobile optimized, so you’ll look amazing on any device. Whatever you need a website for, Wix has you covered.

So, showcase your talents. Start that dev blog, detailing your latest projects. Grow your network with Wix apps made to work seamlessly with your site. Or, simply explore and share new ideas. You decide. Over one-hundred-million people choose Wix to create their website – what are you waiting for? Make yours happen today. It’s easy and free. And when you’re ready to upgrade, use the promo code SEDaily for a special SE Daily listener discount. Terms and conditions apply. For more details, go to www.wWix.com/wix-lp/SEdaily. Create your stunning website today with wix.com, that’s wix.com.

[INTERVIEW CONTINUED]

[0:19:39.2] JM: My sense of Google Container Engine is that you're trying to make a platform that's similar to how an average application engineer at Google experiences the Kubernetes or the Borg experience. For example, if somebody at Google who works on Gmail or who works on maybe YouTube, they don't necessarily want to think about the underlying mechanics of Kubernetes.

I worked Amazon, and Amazon's infrastructure is obviously not the same as Google's, but similarly to the application engineer experience at Google, what I imagine at least is that if I’m deploying a random service that just does something like calculates the — Let’s say, like the shipping fees on an item. That's not something where I really want to know how this is going to be deployed against Amazon's numerous servers. I just want to know that it's going to be deployed, it's going to stay up. The platform itself is going to figure out how many deployment units it needs to scale to. I don't want to think about that. Is that the same experience that the average Google engineer wants to have when they're actually deploying a piece of application code?

I also imagine there are some engineers that would want to configure some more of that granular stuff. Maybe you could give me some perspective on what that deployment experience is like for the average application engineer at Google.

[0:21:16.6] DA: If I might, I have to say if we provide the average user the deployment experience at Google, I will have considered us a complete failure. The amount of understanding that a Google engineer has to know about our overall infrastructure of things is much much higher than we would expect the average person to know. The old saying; the cobbler's son is the last one to get new shoes. I think that’s very much the case here. You can kind of see it happening in real time because so many new products at Google are launching not on Borg, but on Google Container Engine. For example, our brand-new cloud ML service, the one that integrates with TensorFlow and GPUs and all these various things, that runs Google Container Engine, not Borg.

Now, under_the hood, under, under, under, under the hood, it runs on Borg of course, because everything at Google runs on Borg. But you're exactly right. The experience that we're looking to give to the average developer out there is really not thinking about those things.

To Chen's excellent point, that we should be transparent. It should be a white box. You should know the version of the OS you're running on and be able to look down and see what the kernel mods are all these various things, but you shouldn't have to think about that. It's an option for you, and the vast majority of people shouldn't need to think about it, but that is exactly what we're trying to do. You, as an application developer, you know how to build an image, you know how to test that image, you introspect, debug that image.

You how to describe across your application, across your orchestration fleet, that this image needs to be replicated 14 times, and it needs to have port 80 open, and here's where you check to make sure I'm still healthy, and then off you go. You figure out where to run it. If it dies, I want you to restart it. If someone is looking for you, I want you to handle service discovery. I want you to aggregate the logs. I want you to aggregate the monitoring. I want you to handle updates so that when it's time to scale one down and scale went up on a new version, I want you to support that.

I want you to support using advanced underlying infrastructure without me having describe it. For example, on Google Container Engine, you can use preemptive VM's. If you know your application can be taken down at any given time, you should go to town with it, and you can do that and save you up to 70% of your money, I think, is the latest figure. But you're just describing your application and you're letting the system figure out all the other production ready components that you need to have it run.

[0:23:51.7] CG: I think that was a great example, what David mentioned of how internal services within Google decide to use Google Container Engine, and we have many teams within Google that are dog-fooding Google Container Engine and provide feedback for us, which is great.

I think all of that relates to the decisions that the team founding Kubernetes decided this needs to be open-sourced rather than closed-sourced, because what the team had in mind, of course, they wanted to run anywhere, but also building the community around it is really helping us to get different voices into the requirements. There is this positive cycle by, as the Kubernetes adoption increases, we get more and more users using — They'll, for example, over 200 meet-up groups worldwide for Kubernetes, which is amazing because we get a lot of different opinions and a lot of different use cases into our product Kubernetes subordinate, and that brings more developers that want to build tools for Kubernetes. Of course, vendors that are part of it.

In our mind, it was not for the Google engineer, but for every engineer including Google engineers. That's one of the thing that is important for us when building both Kubernetes and Google Container Engine.

[0:25:09.9] JM: If I understand correctly, when you decided to build Google Container Engine, Kubernetes was non-necessarily in a state where it was easy to build a platform as a service on top of it. What kinds of changes had to be made to Kubernetes that were eventually rolled back into the open-source project in order to open Kubernetes up to the platform as a service to be built on top of it?

[0:25:38.5] DA: The reality is there were no changes. What happen was the initial version of Kubernetes that was built was created by some of the folks who knew Borg very very well and they basically went about creating almost a clean room implementation of Borg. If we were going to look at the APIs that Borg provides and re-implement them from the ground up in Go, what would we do? They began building it in that way.

Almost from day one, we had external folks contributing to the core of the platform, Red Hat, CoreOS, and so on, and it was because they all saw the need as well. How do we get containers to be production ready? Wow! Google's been doing this for 15 years. They probably have a lot of expertise in doing this, and we should know join forces and work together. I don't want to understate the amount of distributed architecture and engineering that that they knew about as well.

By and large, this has not been any — We have not required any changes. There are some things where it predated Kubernetes. For example, Red Hat OpenShift predated Kubernetes. As a result, overtime, we've continued to migrate in new features from the external projects into the core Kubernetes so that they can focus on providing a great pods and things like that, and there are other platforms of service that are similar to that. From our perspective, no. We didn't actually need any changes.

[0:27:05.9] JM: Okay. Another random question about operational Kubernetes stuff, and maybe you can tie in how this relates to container engine. If I'm managing my Kubernetes cluster, I've got a collection of pods and each pod is a collection of containers, and so I've got to decide how to arrange the pods and which containers I'm putting together in each pod, because each pod is going to be a collection containers. How do I choose which containers to put into which pods.

[0:27:43.4] DA: Architecturally speaking, the best practice is kind of similar to the way that you think about microservices today. That is, you generally should start with a single service. Again, this is in the academically pure sense. You start with the core of your application that runs in a single thread. It could be a web service. It could be a caching layer. It could be anything of that kind.

Then, if you have dependent services that you use for your overall system, that should share a lifecycle with that core application. Then you should probably put them in the same pod. That might be, for example, a log shipping agent from Splunk or Datadog or [inaudible 0:28:30.1]. It could be a security scanning system from something like Twistlock. It could be something like SSL termination. Maybe you have you know and NGNX server that you always want to terminate your SSL so nothing goes over encrypted over the wire. That would also exist in the same pod.

That said, I do want to be clear. My guess, so I don't have data on this, roughly 90+ percent of all pods are one container and one container only, and you should keep them separate so that the system can move things around. There are many times, and this is very much how we do things a Google as well where you do have additional processes and additional services that you want to share the lifecycle of your application and those are the ones that ones that you should run in the same pod.

[0:29:18.4] JM: Chen, you and I before the show, we're talking about this newer project; Istio, which is a service mesh, and I think this ties in well with this part of the conversation which is that's you often — Sometimes you're going to want maybe kind of a sidecar, which is another container that's going to run alongside the main container that you have an in a pod and it's good to be doing something like providing some supporting services around blogging or service proxying or other things. Can you give some examples of when you would want a sidecar container sitting alongside your main container in a pod and doing certain things, and maybe talk about Istio, this new service mesh that was announced by Google and IBM yesterday.

[0:30:09.8] CG: Yes, and Lyft.

[0:30:11.7] JM: And Lyft, right.

[0:30:12.3] CG: I think the value here, we talked about the separation of concerns. Definitely, when talking about developer and operations, if you want to separate that and you want to make so that developers would like to just focus on the application but then and there are different policies, for example, that need to be enforced between different systems, between development and production. This is an area where a cycle approach which does not impact, your application container is valuable. Definitely, if you have many teams that develop different types of applications but you want to get the unified view, one control point for how everything is working together and you don't want to rely on a developer to do a specific development within their application to get it, this is another case.

I think specifically with Istio, this is even broader than that because Istio was developed in mind that not everything runs in Kubernetes. Meaning, that we would like to have Istio, for example, in cloud foundering and in other — Apply it to other services, because reality is that most of the systems today are not composed just by microservices running on Kubernetes. Although sometimes, in our world, we think this is the case. This is not the case. So a solution like Istio, recognize that and in the future, it will allow you to connect different microservices from different systems into one service mesh.

[0:31:41.1] JM: Okay. I did a show with Matt Klein from Lyft. He worked on Envoy. I think he started Envoy, and what I talked to him about was Envoy is this sidecar proxy where every time you have a request that is sent between two different services, it speaks to Envoy. You have your message that you're going to send to some other service, but before it gets sent to that other service, it gets passed through Envoy and so that you have this service proxy. By service proxy, we mean it's a translation layer that adds information and can be sort of this layer that is ever present in your service messaging, your inter-service communication so that you have this degree of consistency and you also have an overview of things that are going on across all your services because Envoy can collate information about all those different things that are going on. What is the difference between Envoy, the idea of a service proxy, and Istio; the idea of a service mesh.

[0:32:59.1] CG: So first of all, exactly as you said, the proxy allows you to have that consistently layer. Within Istio, we have two components, we have the data plane and the control plane, which is the service mesh itself.

While you have those cycle within the pod, you also need the manager and the mixer and the [inaudible 0:33:18.7]. For example, how do you distribute certificates and how do you collect all the logging and monitoring data from the different proxies to provide the consistent and centralized management clear.

[0:33:31.1] JM: Now, this is one of many different projects that's has sprung up around the Kubernetes community. This falls into the purview of cloud native — And this is another thing we're talking before the show. I was at this cloud native computing event recently and — Well, I've been to a couple of these recently, and they put up this —There's always this slide. It's like, "Here's all the projects in the cloud native space," and there's just a ton of projects, different things around logging and monitoring and service proxying, service mesh.

Are these things that I want to be able to configure — When we talk about Google Container Engine, this to me seems like — Maybe perhaps an opinionated way of running Kubernetes where maybe the Google container engine will use some of the projects in the cloud native computing space, but it's going to kind of have an opinionated view on which ones should be pulled into the container engine service rather than the experience of running your own Kubernetes where you're going to have to pick and choose between which projects in this massive cloud native landscape to use.

[0:34:51.6] CG: That’s a great point, and we think Google Container Engine, we still want to give you that choice. For those users that want us to make that decision for them, we definitely have an opinion, and everything will be preconfigured. If you have different tools, for example, for monitoring, or logging, or tracing that you want to look into the system, or you have a specific CICD tool chain, this is something that is definitely doable on Google Container Engine. Very relevant when you are running your Kubernetes in more than one environment, you want us to have the same — You want to have a consistent experience with these tools as well.

[0:35:28.3] JM: Can you talk at all about what happens under the hood, like on Google's cluster when, or on Google's — I guess, on Google's data centers when I spin up a Kubernetes cluster on Google container engine, just some of the plumbing that occurs underneath.

[0:35:44.0] CG: Yeah, actually I can provide a super cheap plug. For my Cube-Con talk, I actually went into this for over an hour about all the various components that we launch and manage for you. There’s really not an enormous amount of secret sauce. It's just that we watch it all happen for you. When you spin up a Kubernetes cluster, we create a master for you. We create the etcd. We store the etcd on a persistent disc. We create all the VMs for you in a managed instance group.

We create advanced routes to make sure that all the nodes can see the master and vice versa. We also then create additional overlay networks to allow all the pods to communicate with each other. We wire all the Kubernetes cluster into the overall global load balancer. We wire up your logging if you've requested it. To log into big, we log in. We wire up your monitoring to peg the stack driver and on and on and on.

There's nothing secret here. This is no different than you would do when setting up Google or Kubernetes on your own, it's just that we take care of it all for you and make sure it lands properly.

[SPONSOR MESSAGE]

[INTERVIEW CONTINUED]

[0:38:30.2] JM: I did a show with Netflix a while ago about how they schedule these really complex data engineering workloads. I think of Netflix as one of these companies that has a really distinct architecture and they have distinct requirements for all of the engineering needs take place on their platform.

If I'm at a company like Netflix that has very specific requirements, how much configuration and customization and custom scheduler stuff do I need to be writing? If I have all these big data engineering workloads, how much can particular configuration do I need to do on Kubernetes?

[0:39:15.3] DA: Again, I know this is a pretty unsatisfactory answer, but it basically is. It depends. I would say the vast majority of customers do nothing. They use the system exactly as it is. They use vanilla OSs under the hood and they use the default scheduler to take care of it. Actually, if you can do with, that is recommended, because that will allow the system to pack things as tightly as it can do and evict things in an appropriate way and so on and so forth.

That said, Kubernetes does have the ability to do a variety of advanced scheduling techniques and other tools. For example, we just shipped in Kubernetes 1.6 affinity and anti-affinity which means you can instruct the scheduler to say if this pod is there, I want you to not schedule, or please preferentially try and schedule these two pods on the same node because I want them to share data or a common disc, and things like that. We offer things like what we call pod disruption budget which allows the scheduler — You can inform the schedule and say, "Hey, when this application is out there, I want to make sure at a minimum — I don't know. 10 pods are always available no matter what.

Then when you're doing underlying cluster operations, the system will be aware of that and say, "Okay, I was going to reboot this node, but I can't, because there are only 11 pods out there, and if I take this node down, it'll all be down to nine, or something like that, and it all tell everyone to pause. There are just a variety of things on top of that. All the tools are there for you, but I would say by and large, we do recommend that folks just use the vanilla roll out and let the system kind of take care of it for you.

[0:41:04.9] JM: I did a show yesterday about these different ways of doing distributed deep learning, and I know you guys are probably not experts on distributed deep learning, and I'm certainly not either. The take away I got from that was that when you get to these really big machine learning jobs, and these jobs, there's going to be more and more these as time goes on. Thanks in large part of TensorFLow and other machine learning frameworks that people are building. These jobs, they run a lot more efficiently if you can distribute them. Either you split up this giant model that you're building and you can allocate different portions of data to — Or you can allocate the same data to different portions of the model, or you can break up the data and apply different portions of the data to copies of your giant model.

In any case, it makes for these really complex distributed systems problems and these get translated from a model that you perhaps have represented in TensorFlow to infrastructure that you might have represented in Kubernetes. Are there any interesting conversations going on between the TensorFlow teams and the Kubernetes teams around how do we present the right APIs between these two open-source projects that make things really easy but also really efficient for the people who are writing big, complex machine learning jobs.

[0:42:35.0] DA: One thing that Kubernetes is really proud of is that we actually feel like we understand the orchestration space really really well and we want to give people who run on top of Kubernetes a great experience, but we don't want to dictate to them what or how they should run. On top of that, we know explicitly that we're not going to be smart about it.

To be clear, Kubernetes is already an enormous use around deep learning and other ML-type workloads. I already mentioned CloudML, of course. Open AI is using Kubernetes on, I believe, a 2,000 node cluster to do their learning and overall processes. There's no question that things are working properly and we're excited to continue to advance that in the near term.

That said, this is a perfect example of really separations of concerns. TensorFlow is certainly adopting, and I believe already has adopted a number of different concepts around awareness, about being distributed. I don't know if you know much about GPUs, but the reality is that as they exist today, the actually don't split shard very well. By and large, when most people do it, they have a single container take over the GPU or take over a pipeline until it's complete and then kind of release it for something else to pick it up. That kind of awareness really has to happen at the ML framework level.

That said, certainly, we talked to folks across the spectrum not just at TensorFlow, but everywhere, to help understand what their needs are and make sure that Kubernetes is providing those frameworks with the signals necessary to help them make whatever decisions are appropriate for their own infrastructure and platform.

[0:44:21.6] CG: Yes. This is true not just for ML or deep learning. In the past year, you probably also know of Helm, which is another project here on Kubernetes, and idea there is that we invest a lot of helping users of Kubernetes to know how to run different type of workloads and come with a recommendation.

Helm is a package manager for Kubernetes, and we have what we call charts which are certified ways to deploy different type of workload. There is a very vibrant community around those charts, and a lot of knowledge sharing between people in the community. We do the charts, but also we are adding capabilities into Kubernetes, like StatefulSets that was a move to beta in 1.5 that allows you to run Stateful workloads easily on Kubernetes.

[0:45:10.1] JM: All right. As we begin to close off, let's talk a little bit about serverless. Serverless workloads are often these jobs that are stateless. If I'm a developer, I can write a piece of code that will execute against a random server somewhere. It's going to get spun up. It's going to do my job and then it's going to get spun. That's typically the contract between the developer and the serverless architecture.

Now, there's a couple of things. Perhaps people are building stuff where their statefullness or their longer running application stuff is going to be handled by Kubernetes and then their shorter-term stuff is going to be handled by serverless, but there's also people building serverless frameworks on top of Kubernetes where if you want to stand up your own serverless world, then you could have a Kubernetes cluster sitting there that you're managing. Then if users have requests for serverless workloads, you handle that under the covers with your own Kubernetes infrastructure.

What are you both seeing in the serverless related — The overlap between the serverless world and the Kubernetes world. Chen, I know you've got a run in the sec, so maybe you want to go first.

[0:46:31.1] CG: Yeah. In general, what we see with Kubernetes, that it's an easy way to run at distributed workloads, which also of course makes a good fit for serverless technologies. What we see in the community right now, there are several serverless framework built on top of Kubernetes where I assume what they have in mind, the reason why they chose Kubernetes, is that they want that framework to work anywhere.

I'm pretty sure that — By the way, I think Istio will also help significantly for this kind of work to connect to other systems. David, you want to add?

[0:47:05.7] DA: Yeah, sure. I think Chen is exactly right. Again, it kind of goes to that separation of concerns. Sadly, I think serverless has become this a bit of a catch all for a lot of things. Was Heroku serverless? Well, kind of. App Engine, same deal.

What we want to do is empower exactly as Chen said, people to run distributed workloads in whatever way they'd like, and there are a number different platforms on top of Kubernetes. Because Kubernetes is so good at running distributed workloads, the frameworks handle the listing and distribution of a particular function that is running as part of serverless, and then it's underlying run on top of Kubernetes inside a containers somewhere. That, of course, is running on top of a VM, or a CPU, or whatever it might be.

Again, the idea is always give your developers the tools that make the most sense to them, that are appropriate for the workflow. If you're doing a Fibonacci sequence, great. Serverless is a homerun for you. If you're doing a distributed multi-cluster database, probably not going to run in serverless. It's just giving people the tools all the way up and down the stack that are appropriate for them.

[0:48:18.4] JM: Cool.

[0:48:18.4] CG: I do expect that this will also bring some new requirements to the Kubernetes framework, and there is some discussion happening in the community around it.

[0:48:28.3] DA: Yeah, I'd like to take a moment to make a pitch for our Kubernetes community on GitHub. Everything that you see here has been done in the open-source. We have an open-source futures repo. I actually can specifically remember at least a couple of features right off the top of the hat around, for example, what we call scaled to zero, which means workloads that literally scale all the way down to not running at all, kind of in a hibernated state, and only on request do they started. Again, things like that are perfect for serverless. Those are running on top of Kubernetes. People who are actually using it are the ones who come in and help guide the actual shipping of the feature.

[0:49:10.3] JM: Okay, great. Chen and David, I want to thank you both for coming out Software Engineering Daily. It's been a pleasure talking to you.

[0:49:16.3] CG: Thank you for having us.

[0:49:17.8] DA: It was my pleasure. Thank you so much.

[END OF INTERVIEW]

[0:49:26.3] JM: You have a full time engineering job. You work on back-end systems of front-end web development, but the device that you interact with the most is your smartphone and you want to know how to program it. You could wade through online resources and create your own curriculum from the tutorials and the code snippets that you find online, but there is a more efficient option than teaching yourself.

If you want to learn mobile development from great instructors for free, check out CodePath. CodePath is an 8-week iOS and android development class for professional engineers who are looking to build a new skill. CodePath has free evening classes for dedicated experienced engineers and designers. I could personally vouch for the effectiveness of the CodePath program because I just hired someone full-time from CodePath to work on my company Adforprice. He was a talented engineer before he joined CodePath, but the free classes that CodePath offered him allowed him to develop a new skill, which was mobile development.

With that in mind, if you're looking for talented mobile developers for your company, CodePath is also something you should check out. Whether you're an engineer who's looking to retrain as a mobile developer or if you're looking to hire mobile engineers, go to codepath.com to learn more. You could also listen to my interview with Nathan Esquenazi of CodePath to learn more, and thanks to the team at CodePath for sponsoring Software Engineering Daily and for providing a platform that is useful to the software community.

[END]