EPISODE 847

[INTRODUCTION]

[00:00:00] JM: Containers are made to fail gracefully. When your container shuts down due to a hardware or software failure, your distributed application should be able to tolerate that failure. One Simple Way to be able to tolerate such a failure is to make all of your application logic stateless. If your application does not maintain state, then shutting it down in the middle of a computation is not a problem. You can just restart the application, restart your computation, and get the same result, but applications need to maintain state. We need to use databases and in-memory systems to manage long-lived user sessions and other interactions. A database is not just an on-disk abstraction. The database requires an application server to be accepting network traffic. We can run those database applications within containers.

There is a fundamental tension between stateful applications in the idea that containers are meant to tolerate failure gracefully, and Saad Ali joins the show to discuss that tension. He was previously on the show and he also spoke at KubeCon EU. He gave a keynote there, which I spoke to him about, and it was great as always to talk to Saad. He really understands the difficulties in communicating about Kubernetes storage, which we understand as well. I hope you enjoy this episode.

A few quick updates; I am attending a few conferences in the near future. Datadog Dash, July 16th and 17th in New York; and the Open Core Summit, September 19th and 20th in San Francisco. I hope to see you there. If you're going, just ping me perhaps.

We're hiring two interns for software engineering and business development. If you're interested in either of these positions, you can send an email with your resume to jeff@softwareengineeringdaily.com with internship in the subject line.

FindCollabs is the company I'm building. We've launched some new features. If you have a cool project you're working on, I would love to see your project. I checked out every project that gets posted to FindCollabs, and I've been interviewing people from some of these projects on the FindCollabs Podcast.

We have a new Software Daily app for iOS. You can become a paid subscriber for our ad-free episodes by going to softwareengineeringdaily.com/subscribe. I am happy to report that the interfaces improved a lot. The reliabilities improved a lot, and Android is coming soon. All these details are in the show notes. Let's get on to today's show.

[SPONSOR MESSAGE]

[00:02:53] JM: The OSCon has been a great place to go to learn about open source software development for the last 20 years. Today, open source is at the core of software development, and OSCon is still a great place to go to learn about all of it, from machine learning, to cloud technology, to distributed computing. All of these stuff is being shaped by open source tools, and OSCon is the open source conference.

You can go to oscon.com/sedaily. You can enter discount code SE25 and get 25% off your bronze, silver or gold pass, and you can learn about software engineering from industry specialists, such as Holden Karau, Rupa Cachere, Julien Simon from AWS. Holden Karau is actually on the show recently. OSCon is great. I've been a number of times. I think two times, and as at all O'Reilly events, there's great food, which I always look forward to. There's great people. There is a great culture. There is I would say a culture that doesn't come without the amount of work that O'Reilly has put in over the years towards calibrating their conferences in terms of technical material, community material, and the people who work at O'Reilly are super passionate about what they do and it really shows in the execution of the conferences.

So, O'Reilly OSCon is place that I can't recommend enough. Passes start at $746 when you register with code SE25 before June 7th. ASCon July 15th through 18th, 2019 in Portland. Portland, Oregon is a fantastic, beautiful place in the summer. So if you're looking for a great little excuse to take a vacation at the expense of your company, perhaps, you can talk to your manager about attending O'Reilly OSCon and you can also talk about the extreme discount that you'll get from using code SE25 to get 25% off your ticket after going to oscon.com/sedaily.

Thank you as always to O'Reilly for being such an awesome sponsor of Software Engineering Daily and for putting on great conferences that I will continue to attend throughout the years. Thanks again, O'Reilly.

[INTERVIEW]

[00:05:35] JM: Saad Ali, welcome back to Software Engineering Daily.

[00:05:37] SA: Thank you very much for having me.

[00:05:40] JM: Why do people have the perception that state management on Kubernetes is hard?

[00:05:45] SA: So, the fact is that storage is a very complicated problem. It encompasses a lot of things. Storage can mean everything from block and file, all the way up the stack to stateful applications, like databases and message queues and so much more. So, when you start talking about storage, you're talking about the problems in each one of these spaces. When you conflate them together as often happens, because they're kind of dependent on each other, it becomes a very challenging problem.

[00:06:17] JM: Historically, how have people managed stateful client server applications?

[00:06:25] SA: I think, historically, a lot of it has been manual, whether that's a human operator setting up storage systems and then connecting them to the clients that are going to use them manually or have some sort of kind of hack together scripts that are going to set up the storage system and then inject configuration into your application to make it aware of that storage. So, it's been pretty manual in the past.

[00:06:52] JM: Why is a stateful application more difficult to manage than a stateless application?

[00:06:57] SA: Stateless applications can run anywhere. So if you have a cluster, your stateless applications have far fewer constraints in terms of scheduling. Stateful applications just beginning with scheduling are more difficult, because your workload either has to follow the storage, and then your storage has to follow the workload. By that, what I mean is if you have, for example, a volume that is provisioned, the scheduler needs to understand that this volume may not be equally available to every single node within the cluster.

So when it makes a scheduling decision for that pod, it needs to be aware of that and make that decision intelligently. Then after the pod has been scheduled, the system needs to be intelligent enough to say, “Hey, I know this volume can be attached to any of these nodes. I have selected this node. Please attach this particular volume to this particular node.”

So, starting with scheduling. Scheduling is more complicated for stateful applications. After that, or even before all of scheduling, you have to think about provisioning. How does the storage actually get provisioned? Then how does it get made available? Those are the two big problems. Then before all of that, you have to think about the storage system itself as well.

With stateful applications, or stateless applications, usually you can just deploy the application in isolation. With a stateful application, you have to be in many cases keenly aware of the underlying storage system and the implications for your application.

So, for example, if we talk about high-availability. High-availability for stateless applications is fairly trivial. You just scale out the number of replicas that you have for a given pod, and then you're going to have multiple instances of this thing running, and you have a load balancer that can send traffic to whatever pod can handle it.

For stateful workloads, this becomes much more difficult, because let's say you're using of block storage that is available only on a single node. How do you do high-availability on that? You can't depend on Kubernetes to do high-availability. A least not for – Well, reality is this is another talk that I just gave. You can't depend on Kubernetes for high-availability. Kubernetes gives you automatic recovery. Any type of high-availability you need needs to happen at the application layer.

[00:09:21] JM: Distinguish those two terms; high-availability versus automatic recovery. Those sound like the same thing.

[00:09:27] SA: Yes, they do. So, the difference is automatic recovery is about eventual consistency. That guarantees that Kubernetes has are that eventually the right thing what you asked for your intended state. The system is going to continuously drive towards that, and eventually it will achieve that state. But Kubernetes makes no guarantees about how long it's going to take to achieve that state.

For high-availability, what you want is for your application to have minimal amount of downtime. So if you are dependent on Kubernetes to do high-availability. Let's say you have a pod that is deployed on a node and that node goes down. Kubernetes is going to take five minutes for the node controller to realize – Well, to evict nodes. So the way that it works is every node reports a heartbeat back to the API server. That happens every 10 seconds. The node controller is watching the API server and it will notice, “Oh, I haven't had a heartbeat from this particular node for a while.” If it doesn't get a heartbeat for 40 seconds, it’s going to mark that node as unavailable and stop scheduling you pods to it.

Then after five minutes of no heartbeats, it's going to evict or drain that node and remove all the pods that were scheduled there. So now if you're node went down, you have five minutes to having your pod actually removed from that node. Then the next step is – In this case we were talking about stateful. We were talking about a volume which was single attached, and we have another controller, the volume attach/detach controller. This controller is responsible for ensuring the right volumes are attached to the correct nodes.

It also tries to be as graceful as possible when it does it detach, because we want to avoid data corruption. So it will wait for the node to say that it has safely unmounted a volume before it issues a detach. Now, the problem is that the node is down. So what Kubernetes volume attach/detach controller does is it will wait a maximum amount of time and assume, “Okay, this node is dead. Let me go ahead and unilaterally detach this volume.” That timeout is another six minutes.

Now, that means you have about 11 minutes for the pod to get detected as unavailable. Get deleted and then get detached, and then you have additional time to actually have that volume attached and mounted to the new node and then have the container started on the new node.

So you're looking at upwards of over 10 minutes for a pod to recover from a node failure if it is using a single attached volume.

So, this is something that a lot of people don't realize. It guarantees that Kubernetes offers are not high-availability. They are eventual consistency. If your application requires high-availability, this is an application layer problem for now. It's something that you need to handle at the application layer. So you should be running some sort of active-active. May be using multi-writer. You need to architect your HA solution.

[00:12:46] JM: What is active-active?

[00:12:47] SA: Active-active, meeting instead of running a single pod, you should be running two pods which are both actively running and have a load balancer that can pass traffic between them if one of them goes down. But the high-level point here is that Kubernetes doesn't handle –

[00:13:01] JM: Sorry. In that active-active, you would have both containers connected to the same persistent storage medium?

[00:13:08] SA: For a stateful, that seems to be the best way of doing it. So you would have a multi-writer, let's say, NFS volume. You can have them attached to both pods. They would both be up and running at an active, but only one of them would be serving traffic. If one of them becomes unavailable for any reason, it moves to the other one.

Another option would be to have your volumes. If you're using single attached, to have the volume attached, but not being written to or not mounted. Then have it mounted in the application layer when you do the switchover. So that way, you could still have a single writer and have multiple pods that are backed by a single volume. But all of this is currently not handled by Kubernetes and it's something that applications need to be aware of and make sure that they handle it at the application layer.

[00:14:00] JM: And are people actually doing that? Are people like writing these kind of hacky –

[00:14:04] SA: No. I think a lot of people are not doing this.

[00:14:07] JM: So their pods fail and they wait 11 minute for the storage to reattach?

[00:14:10] SA: That's right. This is something where we'll see bugs come through where people say, "Hey, can you help me optimize that timeout? It waits five minutes for node to get drained. Can I optimize that and change that to two minutes?"

Our answer there is that that's the wrong thing that you're optimizing for, or take a step back. What is it that you're trying to achieve? If you're trying to achieve HA, this isn't the way to go about it. Now, it's not completely out of the question for this to be part of the Kubernetes API in the future, but it is not something that's currently being worked on and it's not something that's currently part of Kubernetes.

[SPONSOR MESSAGE]

[00:14:57] JM: Deploying to the cloud simple. You shouldn't feel locked-in and your cloud provider should offer you customer support 24/7, because you might be up in the middle of the night trying to figure out why your application is having errors, and your cloud provider's support team should be there to help you.

Linode is a simple, efficient cloud provider with excellent customer support. Linode has been offering hosting for 16 years, and the roots of the company are in its name. Linode gives you Linux nodes at an affordable price with security, high-availability and customer service. At linode.com/sedaily, you can get started with 2 gigabytes of RAM and 50 gigabytes of SSD only $10. There are also plans for cheaper and for more money.

Linode makes it easy to deploy and scale your application with high-uptime and simplicity. Features like backups and node balancers give you additional tooling when you need it. Go to linode.com/sedaily to support Software Engineering Daily and get your application deployed to Linode. That's L-I-N-O-D-E.com/sedaily.

Thank you, Linode, for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[00:16:27] JM: We’ve gone into the weeds quite quickly, although it's okay, because we did that last episode where I think we covered the basics of Kubernetes storage. Luckily, by the way, most of this nobody needs to know. It’s just a mere curiosity for most of the people. But it was interesting how popular that episode was. People really wanted to have this question of stateful Kubernetes demystified for them. I think it's because people would hear these terms like persistent volume claim, and persistent volume, and their eyes would glaze over and they would be very confused.

[00:17:09] SA: That's completely understandable. I mean, there's so much lingo and terminology that's unique to Kubernetes and understanding this whole volume subsystem. It's a matter of education at this point where we need to get the word out. So I think at every KubeCon, we've had a storage 101 talk for folks who are curious about this stuff. They should read up on it and understand what the benefits are that Kubernetes is now offering in terms of storage. I think the big key take away that I had for my keynote was, yes, storage is a very complicated space, but Kubernetes is introducing abstractions that allow you to handle this in smaller pieces that are much more manageable.

The way that I broke it down is storage has four problems that you should be aware of. One is figuring out what storage to use, and Kubernetes can't help you with that. This is something that you need to understand what options are available to you in terms of block and file and all the different data services, databases, message queues.

The second step is understanding what the requirements of your application are, because you can ask for the sun and the moon, but you don't necessarily need the sun and the moon. You should be aware of the kinds of minimum requirements, your application actually has and use that to filter down the options that are available to you. Then you have to weigh the tradeoffs. Storage is often a tradeoff between performance, durability, availability and cost. So you have to figure out you want to optimize for and what makes sense in your use case. Finally, you have to make a decision about what you're going to use, and Kubernetes is not going to make that call for you. You have to architect your own application.

Once you've decided what storage stack you're going to build on top of. The next question is how are you going to deploy that storage stack. The answer there is you have two options. One is to go with a completely managed storage stack, which means somebody else is deploying and managing it for you. If you're running in a cloud environment, Google Cloud for example, you could have CloudSQL as a managed data service or Google Cloud persistent discs as a managed block solution.

In those cases, how the storage is deployed and managed is not your problem. Somebody else is getting paid to worry about that and they're going to take care of that. But if you select a storage and want to manage it yourself, then you need to figure out how you're going to deploy it. This could be a data service, like a database, or it could be a lower level file or block. The same set of problems apply. How do you deploy the storage?

Often times, people think that the storage system itself must be deployed on top of Kubernetes in order to use it from Kubernetes, and that's not the case. The reality is that you can deploy your storage outside of the Kubernetes cluster and have your Kubernetes cluster consume that storage. The legitimate use cases there are, for example, if you have a database running on – A legacy database running on some VM somewhere. It is a service that's exposed over your network. That database doesn't have to be migrated to your Kubernetes cluster in order for your Kubernetes cluster to access it.

If we move lower down the storage stack and talk about block and file, you could have an external storage appliance that you paid millions of dollars to run, and now you have a Kubernetes cluster. You don't have to deploy a software-defined storage system on your cluster in order to use block and file. You can use that existing storage that you already have. Of course, all of these things have different trade-offs, but it is absolutely possible to have your storage outside of Kubernetes.

Now, if you do decide to deploy your storage on top of Kubernetes, what you need to realize is that at that point, deploying your storage is just like deploying any other stateful application. It's going to be consuming some resources in terms of storage from the cluster and it's going to be exposing some APIs at the top.

So let's talk about data services first. Data services are things like databases, message queues, etc. You could deploy it on top of Kubernetes and your data service could depend on another data service. For example, you could have a message queue that depends on a key value store, which is another data service, or you could have a data service –

[00:21:55] JM: Do you describe this in Kubernetes? You'd describe a –

[00:21:57] SA: As far as Kubernetes is concerned –

[00:21:59] JM: Oh you're saying more abstractly.

[00:22:01] SA: Yeah. I'm just saying abstractly, these things are – As far as Kubernetes is concerned, they're just different stateful application deployed on top of Kubernetes. They may depend on each other, but as far as Kubernetes is concerned, they're just applications running on top.

But eventually somewhere in this line, you need to write to disk. In order for your bits to be persisted, they need to get to disk somewhere. The fact is that containers are ephemeral. Anything written to the file system inside of a container is going to be lost as soon as that container is terminated. So this whole song and dance that we're talking about, all it's about is eventually we want to get bits to a disk.

So we have these fancy data services at the top, databases, message queues and all these other things that make it easier to be able to figure out how to persist those bits and makes it structured, makes it easy to access. They have different optimizations for the bits that you're trying to read or write. But, eventually, something needs to write those bits down to the disk. That's where block and file comes into play.

So I like this differentiation of breaking storage into data services and block and file. Block and file are the – Where the rubber meets the road, where the bits land on the disk, and that is where Kubernetes really shines. Kubernetes has the community storage API as well as the container storage interface, which enable application portability. They enable workloads to be deployed independent of the blocker file storage that they're going to use. They enable automatic provisioning. They enable storage to automatically be moved around with the workload. All of these Kubernetes storage magic happens at that layer.

So when you are going back to the deployment problem. You're figuring out how to deploy your storage. You have a storage stack. So we talked about the data services at the top. If your data service depends on another data service, that's fine. It's just another Kubernetes application that you deploy. You deploy two applications. One depends on other. But, eventually, one of them is going to depend on blocker file to write to disk. That means that you also need to have some sort of blocker file storage running or available to your cluster.

Like I said before, that block and file could be an external storage appliance or it could be something that you deploy on top of Kubernetes. So you could have a software defined storage system, which aggregates all the disks that are available to your nodes and then builds out a distributed file system on top of that, like Ceph or Gluster. Those are great open source options. Then there is a ton of proprietary options in this space.

But, yeah, if you want to deploy your block and file on top of your cluster, that's also possible. But this is where it becomes interesting, and the point I want to drive home here is a software defined storage system deployed on top of Kubernetes is the same as any other stateful workload that you're trying to deploy on top of Kubernetes.

So your database that you’re deploying on Kubernetes is the same as you deploying Rook, which is a Ceph operator on top of Kubernetes. It just so happens that the software defined storage systems consume local disks and they expose a block and file storage, whereas the data services may be depend on another data service or maybe they depend on block and file.

[00:25:29] JM: By that way, in that couple cases you just illustrated. I may configure Ceph to manage my file and block storage using Kubernetes, perhaps through Rook. I may deploy also a database on a Kubernetes cluster that is using the storage offered by that managed Ceph cluster.

[00:25:58] SA: Exactly. Exactly. So this is why I want to differentiate between these problems. So the first one we talked about was selecting your stack. You talked about – Okay. You selected a database and you selected Ceph, and then you selected local disks. That the stack that you selected. That's great. The next challenge is how do you deploy these things. For your database, it's a stateful workload. For your Ceph, you're using Rook to deploy it on top of Kubernetes. Again, it's the stateful workload. For your block and file, you selected local storage off of the machines, and the Kubernetes storage API offers a persistent local volume API that allows you to consume local volumes off of the machine.

So, for these stateful applications, the fact is that it can get fairly complicated to deploy and manage these applications, particularly management. So if you think about a database, for example, scaling the database in and out could require custom operations for things like replicating the database when you scale out, or when you do an upgrade, it might require some sort of operation, custom operation. Kubernetes doesn't handle those automatically for you.

So the recommendation is if your application, your Kubernetes application is sufficiently complicated, which a lot of stateful applications are, then use an operator. Operator is just a controller and a CRD that manages the lifecycle of another Kubernetes application. So you could have an operator for Cassandra, for example, that will manage the deployment and management – Deployment and kind of scale out and scale in an upgrade of Cassandra. It simplifies that. You could always do it yourself, but an operator makes it easy.

[00:27:47] JM: By the way, not to not interject, but does the operator solve that problem that you illustrated with my container becoming unattached from the storage and Kubernetes taking 11 minutes to notice.

[00:28:03] SA: It could. You could build HA logic into an operator. It would require you to have some sort of monitoring and probing of the pods that are serving your application and then have some sort of rectification logic. But I haven't seen that in the wild yet.

[00:28:25] JM: They got to get on that.

[00:28:27] SA: Yeah, I think it tends to be –

[00:28:29] JM: I know you're listening to this, Cassandra folks.

[00:28:32] SA: Well, I mean, I think it make more sense to – Actually, I'm not sure. I'd I don't know what the best way to build a HA Cassandra cluster is. That's out of my depth. So I'm not going to touch that.

[00:28:43] JM: Not HA Cassandra cluster, but it's just a mix of the conversation. What is the right solution for that problem that you alluded to earlier, this 11-minute eventual consistency where I have to write application logic to solve a problem that's happening in Kubernetes storage?

[00:29:04] SA: Yeah. I mean, the easiest way to handle that is to use multi-writer, and we talked about –

[00:29:08] JM: Multi-writer. Oh, we talked about the active-active.

[00:29:10] SA: We talked storage is always a tradeoff between performance –

[00:29:15] JM: The onus would fall on who to write the active-active? Would that go in the operator?

[00:29:20] SA: That would go in the operator. Yes. Actually, that's a good point.

[00:29:23] JM: So you need the operator to solve some kind of – I mean, they already is solving the high-availability thing.

[00:29:29] SA: Yes, but I'm not sure if any – I don't know how most operators are built. So I don't know they're doing it this way or if they're building it directly into the application or not.

[00:29:39] JM: Yeah, fair enough.

[00:29:41] SA: Now, the interesting thing talking about operators and how they make stateful applications easy –

[00:29:48] JM: Go ahead and explain the operator pattern. We've explained it once or twice on previous episodes, but I think it's worth mentioning.

[00:29:53] SA: Yeah. I mean, Kubernetes CRDS, custom resource definitions and a controller, just like any other controller that Kubernetes has, Kubernetes ships with a bunch of controllers that manage things like the node controller, which monitors and figures out whether nodes are available and drains nodes. It's anything that monitors Kubernetes API objects and then updates them and reacts to them. So the controllers is the bit that implements the logic to monitor and update.

So, for an operator, it's a kind of specialized controller that is responsible for managing the lifecycle of another application. So you deployed it on Kubernetes using Kubernetes basic primitives, and it is going to run an application that will monitor customer resources that are specific to your application. So for a Cassandra operator, for example, there would be a Kubernetes CRD for Cassandra and it would have YAML that is custom for that controller. That takes things like what are the number of replicas that you expect for this Cassandra cluster?

So instead of the user having to declare all the pods are stateful sets and PVCs and everything else that they need to do to get a cluster up and running a Cassandra cluster. All they have to do is create a YAML file that says, "This is how many replicas I want," and then the operator is able to go off and do the provisioning and create the Kubernetes storage primitives to make that happen.

Then it's not just about deployment. It's also about management. So day two, day three, you decide three nodes is not – Or three instance is not enough for your cluster and you want to scale out. Instead of having to fiddle with Kubernetes storage primitives, or Kubernetes native primitives. You can actually just update the CRD for your operator, and then the operator has the logic to be able to observe that and then do whatever it needs to do at the Kubernetes layer to make that happen.

So, it's adding a layer between Kubernetes and the deploying and managing the application and just simplifying that process so these operators are custom written for each type of application. Some operators are better written than others. Yeah.

[00:32:12] JM: What are the biggest outstanding problems in the Kubernetes storage ecosystem?

[00:32:22] SA: That is a good question. I think one of the biggest challenges is coming up with an on-prem. I think on the cloud side, a lot of these stuff is taken care of for you and managed for you. I've been looking a lot more into the on-prem options recently for storage, and it can get very, very confusing, because there are so many different options. How do you select which one? How do you evaluate the difference between Ceph versus Gluster versus one of the proprietary software defined storage systems?

[00:32:58] JM: How popular are those Ceph and Gluster installations?

[00:33:02] SA: They are fairly popular from what I hear. It depends on –

[00:33:05] JM: What kinds of companies deploy those things?

[00:33:07] SA: Yeah. I think Red Hat is big on Gluster, and Ceph is – I'm not sure who –

[00:33:13] JM: So if I'm running a big company like an oil refinery or something and Red Hat is my technology provider. Maybe Red Hat deploys Gluster to my data center, but they help me manage it.

[00:33:28] SA: Exactly. That's the interesting thing with software defined storage, is there are open source options, but they tend to be pretty difficult to manage and run. So, typically, you want to have somebody else run them for you and there's a ton of proprietary options. Even for the open-source ones, for example, in the case of Gluster. You'll have somebody else who can manage it for you.

Of course, software defined storage is one option, then you have external storage appliances that you could use. So you could have a classic or Dell or NetApp storage appliance that's sitting somewhere and use that for your cluster. So I think that's a huge challenge of, “I have all of these options. Which one should I be using?” I think a lot of people start with just what they have.

[00:34:17] JM: But how is that a Kubernetes problem?

[00:34:19] SA: That's a good point. It is not. It is absolutely not. I think one of the things I said my keynote was, a lot of the confusion in terms of people that think Kubernetes storage is hard –

[00:34:31] JM: They just think storage is hard.

[00:34:33] SA: Exactly. Exactly. Storage is complicated. Kubernetes adds a lot of obstruction layers that simplify it. But the fact is that storage is a big, complicated problem and you need to do your homework. You need to understand the tradeoffs.

[00:34:47] JM: As I've kind of covered this space, it has been like kind of an archaeological dig of these different storage appliances and things. I entered the software industry when the cloud had started. So I had to know what these things were. As we become – I mean, I think you're about – You've probably been in the software industry about the same period of time I have. Isn't it just like an archaeological dig? Like looking at some of these installations and some of these tools that have been around for a while and you're like, “This doesn't make any sense to me.” It's only by looking into the history that you kind of understand, “Oh, here's what this thing exists, and why we should still be using it.”

[00:35:32] SA: Yeah. The reality of it is decisions make sense in the timeframe that they were made. When you look back at them without context, it sometimes seems like why the heck did they do that? That's often true of code. The reality is that especially for enterprises and businesses, once they deploy something and it's working for them, they don't really want to change. It's not their expertise. It's not the thing that they want to invest money in. They want that thing to be as minimally invasive as possible.

For tech companies, companies where tech is kind of foremost, they love hacking and creating new things and experimenting with the newest, coolest thing. For companies whose core deliverable is not tech, they want to focus on their product. They don't want to focus on how to make that product available. So they want to do as little work as possible.

So if they have an on-premise data center that they're required to run because of regulatory reasons and they have storage appliances that are running there. They want to be able to use that. The reality is that they absolutely can. Kubernetes, the underlying storage doesn't have to run as part of Kubernetes, but the messaging around that is very confusing

If you ask around, people will say, "Oh! Well, you got to run the software defined storage system." Why? It depends on your workload. What is the requirements for your workload? Are they being met with a storage that you have? If so, great, like keep running your storage where it is. If you are not able to afford an external storage appliance and you want to use the disks that are available on your compute cluster as an aggregate and have some sort of replicated storage, then a software defined storage system makes sense. But you have to understand that either of those options is okay. As far as Kubernetes is concerned, it's up cluster operators, storage operators, application developers to make these calls.

Kubernetes introduces the obstruction layers that allow all three of these audiences to kind of focus on different parts of the problem. But, ultimately, somebody has to make those decisions, and Kubernetes isn't the one that's going to make those decisions for you.

[00:37:45] JM: Okay. So just to revisit this question, why are you thinking through these issues? Are these things that are cropping up in Kubernetes discussion groups online or something? I guess I'm not fully aware of what exactly you do at Google, what your responsibilities are. So, tell me what you do at Google today and why it is in your purview to be thinking through how to connect Kubernetes to legacy on-prem storage solutions.

[00:38:14] SA: So my role at Google is I'm tech lead of the GKE storage lifecycle team. Our purview is making sure that –

[00:38:23] JM: So that includes GKE on-prem?

[00:38:25] SA: Exactly. GKE and GKE on-prem. Wherever GKE is running, we want to make sure we're able to access the storage that customers require.

[00:38:34] JM: Right. Okay. Well, that explains the question. So I guess you have enterprises that you are working with, and they say, "Look. We've got this legacy storage devices. We want to connect to them directly. Help us do that."

[00:38:47] SA: Yup. Kubernetes offers the primitives to do that. You deploy a CSI driver for that legacy appliances and use the same storage primitives that you would if the storage was backed by STS for local disk. The abstraction layer exists in Kubernetes to be able to do that. But I think the education isn't there to say that, "Hey, you don't have to worry about that."

[00:39:12] JM: CSI driver, that is a driver that you deploy a with the legacy storage device?

[00:39:23] SA: That's a good question. So I talked earlier about the four problems for storage. First was select storage. Second was you deploy your storage. So we talked extensively about how you deploy it, and we talked about how it could be external or internal. The third problem is integration. So this is where CSI comes in. How do you integrate with your storage?

Let's ignore the data services. Let's talk about block and file. So for block and file, the way that you integrate would be a CSI driver deployed on your system, and that CSI driver will connect to whatever storage you have, whether that storage is deployed within your Kubernetes cluster as a software defined storage system or external.

So, for example, if you have a legacy external appliance, you'll deploy a CSI driver for that, and that will teach Kubernetes how to provision volumes, attach, mount and all the things against that appliance. Similarly, if you're deploying a software defined storage system, Gluster, Rook with Ceph, or any other type of software defined storage system, you would deploy a CSI driver that tells Kubernetes, "Here is how you actually provision, mount, attach and all the basic operations for that software defined storage system.

So this is why I keep hammering at you need to decouple how you deploy storage from how you integrate and consume that storage. You can have the actual storage system inside the cluster or outside the cluster regardless, at least at the block and file layer, it's very simple to be able to integrate with either of them as long as there is a CSI driver available for them.

Then once you have the integration completed, the last step is how do you actually use that storage from inside of a container? For block and file storage, the answer there is the Kubernetes storage API. You deploy a pod that uses a persistent volume claim and their storage classes and PVs get created and everything that we talked about in the last episode. All of that magic basically abstracts you away from that underlying storage. So you are building against APIs that are consistent. If you happen to move around to different clusters, the underlying storage can be changed, but the way that you deploy your application and the bits for your application don't need to be modified, whereas if you are deploying your application without Kubernetes, you would have to be aware of these problems. You would have to figure out how you make the storage available. How you provision the storage? How you make sure that the volumes are attached to the correct node? How you make sure that they're mounted into the correct place to make them available to your workload. Then if something happens, how do you move all of that around?

So Kubernetes handles all of that for you automatically. If you take these four steps, the selection of storage, the deployment of storage, the integration and then consumption, you can actually see that there are three different kind of roles that would be responsible for each one of these steps. For deploying storage, that would be your storage administrator. For integrating, that would be your Kubernetes cluster administrator. Finally, for consuming your storage, it would be your application developer. If you don't have Kubernetes, then since the problems in those layers are conflated, it becomes much more fuzzy and much more difficult to manage.

[SPONSOR MESSAGE]

[00:43:04] JM: You probably do not enjoy searching for a job. Engineers don’t like sacrificing their time to do phone screens, and we don’t like doing whiteboard problems and working on tedious take home projects. Everyone knows the software hiring process is not perfect. But what’s the alternative? Triplebyte is the alternative.

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

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

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

Thank you to Triplebyte.

[INTERVIEW CONTINUED]

[00:45:24] JM: What are the other issues that you see encountered by these companies that are deploying Kubernetes to their on-premise installations?

[00:45:41] SA: In terms of storage?

[00:45:42] JM: In terms of storage or more generally.

[00:45:44] SA: I think one of the biggest challenges is what the stack is that you're going to be running on top of. What hardware you're going to be using? What storage? Any hypervisor?

Again, I think it comes down to there are lots and lots of options. Then for lots of enterprises, there have been legacy decisions that were made that they want to continue to work with, because they were very expensive. So that adds in another –

[00:46:12] JM: Layers. What layers the stack?

[00:46:14] SA: All the way down to the hardware. So, you're talking about what kind of servers you're running? What kind of storage you selected. If you're running a hypervisor, I think, for example like VMware 6.5 supports a wider range of Intel CPUs. 6.7 drop certain CPUs, because it uses features from a newer version. So this whole stack, there are so many options and tradeoffs. I think that's one of the biggest challenges for people who manage on-prem clusters.

Once they have something locked down, then it's more a challenge of how do you actually deploy Kubernetes, or how do you actually deploy your workloads is the bigger higher-level question. If you're not on Kubernetes, a lot of people see the benefit of Kubernetes of having kind of a consistent API that they can deploy against. So a lot of people are moving towards Kubernetes, but the challenge is that Kubernetes can be fairly difficult to deploy.

If you've ever seen Kelsey Hightower's Kubernetes the Hard Way, there are a lot of steps especially like getting the networking right and all of these things.

[00:47:23] JM: I have not. I should probably watch that.

[00:47:24] SA: It is very difficult. So doing it yourself is hard. So what a lot of enterprises want is can somebody else do this for me? This is not my core competency. There are companies that are much better suited at this. So at Google, we started GKE before anybody else had a managed service. We've kind of nailed the ability for you to spin up and spin down clusters very quickly. Within an order of minutes, you can just click and you've got a new cluster.

So now taking that magic and that management and all that automation and making it available on-prem is our new goal with Anthos.

[00:48:09] JM: First of all, thank you for mentioning Software Engineering Daily in your keynote.

[00:48:14] SA: Absolutely.

[00:48:15] JM: That was great. Second of all, I want to apologize, because I was late to your keynote. So I missed that part. I don't know what you said. Why do you think people are so interested in this question of state management on Kubernetes when the application takes care of it for you? Who cares?

[00:48:39] SA: I think the reality is that there are a lot of stateful applications. It's very, very difficult to run anything useful without having to persist state. So if you're talking about user profiles. You're talking about records for customers, like any kind of useful application that you want to build that is going to be distributed. Likely, there's going to be some state that you're going to want to store somewhere.

Yes, if you use a managed storage system, then things are very easy for you, and that is definitely an option on the cloud. But the reality is that on-prem, you don't have a lot of great managed options. Even on the cloud, there may be data services that you want, which don't have a managed option. There might be a new, cool, hot the message queue that you want to use, and there is going to be no managed version of that available to you. So you're going to have to figure out how to deploy that, and that data service is going to need to persist a bit somewhere. So now you're very much interested in figuring out how to deploy stateful workloads on Kubernetes.

[00:49:49] JM: What would be required [inaudible 00:49:50]. Let's say I make Jeff Queue. It's a queuing system. It's a lightweight pub/sub queuing system for podcasts based workloads. I want to deploy it to Kubernetes. What is required for me to do that?

[00:50:05] SA: So, first, you need to decide what state you need to store and how you're going to be accessing it. So understand the requirements for your application. Then you need to figure out what of all the many storage options that are available to you will be sufficient to satisfy that. Then you're going to have to figure out, "Am I going to use a managed version of that or an unmanaged version of that?" If you're looking at an unmanaged version of that, then you have to repeat the same problem again, because this unmanaged thing is another stateful that you're figuring out how to deploy.

You need to understand the requirements for that application. You need to do the same evaluation again and figure out where it is going to persist those bits. Is it another managed application? Is it an an unmanaged application? If it's an unmanaged, eventually, it will make its way down to the block and file layer.

So something is going need to write bits to disk. Then we're going to be at that layer and you need to decide what block and file am I going to use. Is it going to be local disks directly, which is an option? But there is a tradeoff. They're not replicated. So that's an issue. Are you going to use some sort of replicated software defined storage system or some high-availability external storage system? Those are all options that you're going to have to make. So you're going to have to make that call all the way down the stack.

Now, a lot of these options may already be made for you depending on what environment you're running in. If you're running in an organization, somebody may have said, "These are the three options that are available to you. Pick one." That often makes it easier. With Kubernetes, that kind of delegation of duties is more possible, because you could have a cluster administrator, a storage administrator, select a bunch of storage options and make them available to a cluster. Then as an application developer, instead of having to weigh all the possible storage options in the world, you look at what is available from your cluster administrators and figure out if that satisfies your application's needs or not. If not, go talk to the folks that are responsible for setting up your storage or setting up your cluster.

[00:52:14] JM: Have you talked to Bassam from Upbound?

[00:52:16] SA: Yes. Not recently, but yes, we definitely talk.

[00:52:20] JM: So they just did this demo, and I did an interview with him earlier today. So it's kind of in my head. But they did this demo where they made cross-cloud GitLab deployments easier. So GitLab is a big complicated application. It's got Redis. It's got MySQL. I think it's got – I don't know. Other services involved. You've got to spin up a number of different clusters.

Maybe it's got a Kafka queue in there somewhere. I mean, you're managing git. You're imaging git and you're managing a heavyweight application system around it. It's like 30 things to deploy. But their way of doing it is spinning it up on AWS, or Google, or Azure. They do it on this thing called Crossplane, which they built.

Crossplain includes – Or I guess in building it, or maybe this was built already. But they took the idea of a claim or maybe – Correct me if I'm wrong about this. The idea of a claim, which I was like, "Oh! Persistent volume claim." I say, my application, my GitLab, needs a persistent volume, because like you said, I've got to write the bits to a disk eventually. He said, "Yeah." Then like let's say my GitLab needs Redis. So you make a Redis claim." I said, "Oh! So you mean like a persistent volume claim for Redis?" He said, "No. It's actually different type of claim specifically for Redis." Are you familiar with that idea or does that sound – Is that a new thing?

[00:53:58] SA: This is kind of a new idea. It's I haven't looked at Crossplane. So if I understand it correctly based on what you're describing, it sounds like a way to be able to provision instances of an application. So this pattern of PVC storage class and PV is very generally usable. The benefits of it are that you can have a controller that knows how to provision a thing. So at the PVC storage class and PV layer, it's a volume. You could have two different objects that allow you to consume and provision separately so that you can swap out that controller or be able to provision multiple instances of the same thing.

So if we take this pattern and we move it up to the application layer, if you look at the way that applications are deployed today, what ends up happening is you create the equivalent of the PV object, not a claim. You create an application CRD or an operator CRD saying this is what I want and that is what gets provisioned and you manipulate that object to update that instance that was deployed. If you want multiple instances of that application, you kind of don't really have the concept of like a storage class or any kind of abstraction to say, "Hey, this is the template that I want you to use to just provision new instances of this application."

I think if I understand based on what you said, it sounds like he's taking that pattern and applying it at that application layer so you can say, "Hey, I want a new instance of Redis and it'll provision a new instance," rather than you having to say, "Hey, I want a new instance, and it should look like this, this, this, this." You create, what I would imagine, a storage class or like an application class once and then you specify all the templates you want for any new instances of Redis. Then instead of having to specify that every time you need to spin off a new instance, you would just create a single claim for that application.

[00:55:59] JM: And just specify what cloud it goes.

[00:56:01] SA: Right, because our controller would be able to swap out any kind of implementation details for you. The thing that's going to provision you Redis instance. Yeah, that makes sense. Then you would be able to swap out.

I think one of the challenges at the Kubernetes storage layer is, yeah, the block and file layer is completely swappable and extractable. But at the data services level, it's not really portable, because your application has to be aware of the data service its consuming. So if you're using the Jeff Queue. Then your application needs to have an SDK for the Jeff Queue that knows how to discover it. How to authenticate with it? How to read and write to it? If you in the future decide to move to the Saad Queue, then you're going to have to rewrite your application, pull the Saad Queue API into your – SDK into your application and then re-architect everything.

So that portability that's possible at the block and file layer isn't really possible at that application layer. Though you could potentially imagine that if a controller knows how to provision a stateful application using either the Saad Queue or the Jeff Queue, then that becomes something that you specify in the equivalent of an application class and then your claim becomes kind of – Well, it will be an application claim that just will provision based on whatever cluster you happen to be on. So if you're on my clusters, it's going to use the Saad Queue. If you're on your cluster, it will use the Jeff Queue. It seems like a neat idea.

[00:57:34] JM: And he actually took it one step further. He said that there was – There were some application that they – Or some other use case they were talking to where they were somehow using this claim interface as a way to translate PostgreS workloads to CockroachDB. Maybe I'm remembering it wrong. We should wrap up obviously. But I find it interesting that you could potentially use these Kubernetes storage interfaces as translation layers from one database to another. If the CockroachDB creates the right interfaces to store in PostgreS workloads over the wire in a more consistent fashions that’s maybe easier to run, that's kind of cool. That's like cross-database.

[00:58:20] SA: Yeah. No. That seems like a really cool idea. I think it’s something that I've been talking to my colleagues in SIG Apps, and it's something that they are also interested in. So, I think they should definitely talk to each other to make sure we all land in the same place, because it seems like the goals that we want are very, very, very much aligned.

[00:58:38] JM: Totally. Cool. Well, Saad Ali, thanks for coming back on the show and congrats on your keynote.

[00:58:43] SA: Thank you very much.

[END OF INTERVIEW]

[00:58:47] JM: GoCD a continuous delivery tool created by ThoughtWorks. It's open source. It's free to use, and GoCD has all the features that you need for continuous delivery. You can model your deployment pipelines without installing any plugins. You can use the value stream map to visualize your end-to-end workflow, and if you use Kubernetes, GoCD is a natural fit to add continuous delivery to your cloud native project. With GoCD on Kubernetes, youdefine your build workflow. You let GoCD provision and scale your infrastructure on-the-fly, and GoCD agents use Kubernetes to scale as needed. Check out gocd.org/sedaily and learn how you can get started.

GoCD was built with the learnings of the ThoughtWorks engineering team, and they have talked in such detail about building the product in previous episodes of Software Engineering Daily. ThoughtWorks was very early to the continuous delivery trend and they know about continues delivery as much as almost anybody in the industry.

It's great to always see continued progress on GoCD with new features like Kubernetes integrations so you know that you're investing in a continuous delivery tool that is built for the long-term. You can check it out yourself at gocc.org/sedaily.

[END]