**EPISODE 741**

[INTRODUCTION]

**[00:00:00] JM:** In a cloud infrastructure environment, failures happen regularly. The servers can fail, the network can fail, and software bugs can crash your software unexpectedly. The amount of failures that can occur in cloud infrastructure is one reason why storage is often separated from application logic. A developer can launch multiple instances of their application with each instance providing a stateless environment for serving API requests.

When the application needs to save state, it can make a call out to a managed cloud infrastructure product. Managed cloud databases provide a reliable place to manage application state. Managed object storage systems, like Amazon S3, provide a reliable place to store files. The pattern of remote cloud services does not work so well for on-prem and hybrid cloud environments. In these environments, companies are managing their own data centers and their own storage devices.

As companies with on-prem infrastructure adopt Kubernetes, there's a need for ways to manage on-prem storage through Kubernetes. This might sound like a complicated subject, but we have Saad Ali from Google to explain the subject in much more detail. Saad is a senior engineer at Google where he works on Kubernetes. He's also a part of the Kubernetes storage SIG, S-I-G, Special Interest Group. Saad joins the show to talk about how Kubernetes interacts with storage and how to manage stateful workloads on Kubernetes. We discussed the basics of Kubernetes storage, including persistent volumes and we get into more complicated subjects like the container storage interface and the future of Kubernetes, where the project is going and Saad's involvement in it.

Before we get on with the show, I want to mention we're looking for sponsors. If you are interested in sponsoring the show, you can go to softwareengineeringdaily.com/sponsor. We have 50,000 subscribers and many of them are software engineers, perhaps most of them, and we would love to get some sponsorship for Q1.

We are also doing a listener survey. If you go to softwareengineeringdaily.com/survey, you can find out more about that survey and you can take the survey if you're a listener. We would love to get your feedback and know what we're doing wrong, what we're doing right. With that, let's get on with the episode.

[SPONSOR MESSAGE]

**[00:02:45] JM:** OpenShift is a Kubernetes platform from Red Hat. OpenShift takes the Kubernetes container orchestration system and adds features that let you build software more quickly. OpenShift includes service discovery, CI/CD built-in monitoring and health management, and scalability. With OpenShift, you can avoid being locked into any of the particular large cloud providers. You can move your workloads easily between public and private cloud infrastructure as well as your own on-prim hardware.

OpenShift from Red Hat gives you Kubernetes without the complication. Security, log management, container networking, configuration management, you can focus on your application instead of complex Kubernetes issues.

OpenShift is open source technology built to enable everyone to launch their big ideas. Whether you're an engineer at a large enterprise, or a developer getting your startup off the ground, you can check out OpenShift from Red Hat by going to softwareengineeringdaily.com/redhat. That's softwareengineeringdaily.com/redhat.

I remember the earliest shows I did about Kubernetes and trying to understand its potential and what it was for, and I remember people saying that this is a platform for building platforms. So Kubernetes was not meant to be used from raw Kubernetes to have a platform as a service. It was meant as a lower level infrastructure piece to build platforms as a service on top of, which is why OpenShift came into manifestation.

So you could check it out by going to softwareengineeringdaily.com/redhat and find out about OpenShift.

[INTERVIEW]

**[00:04:53] JM:** Saad Ali, you are a senior engineer at Google and you're also on the SIG Storage for Kubernetes. Welcome to Software Engineering Daily.

**[00:04:59] SA:** Thank you very much. Glad to be here.

**[00:05:01] JM:** So when I talk to people in the Kubernetes community about the issues in Kubernetes and Kubernetes that are still not completely resolved, one of the things that seem to frequently come up is stateful versus stateless workloads and the idea that it is difficult to run a stateful workload on Kubernetes. Could you disambiguate what is a stateful workloads?

**[00:05:26] SA:** Sure. So let's start with what a stateless workload is. So if you deploy a traditional containerized container on to Kubernetes running on Docker, you couldn't have your application do whatever you want inside that container. It can write to what appears to be the file system. The problem is that as soon as that container is terminated, anything that was written to the file system is gone. So any time that container is terminated, it's moved, and if you're running in Kubernetes, your containers can move around if there's resource constraints, there is not enough resources on a given node. They can move around.

So as your container moves around, any state that it wrote to disk is gone. So if you're trying to run any type of application that needs to remember what it did, so you put something in your shopping cart and you had that request handled by an application running in a container. If it happens that application got moved to a different machine, you come back and your cart is gone. That's pretty bad, or even worse, the information about your profile, for example. So we need to find a way to be able to persist that state independent of the lifecycle of a single container. That's kind of the domain that we operate in.

**[00:06:41] JM:** When I spin up a container, it's given some section of resources where I can write to. Why would a container move?

**[00:06:54] SA:** Good question. So there can be a number of reasons. For example, let's imagine that you are running – One of the beauties of Kubernetes is you don't have to dedicate a single machine to a single application. You can kind of treat your machines as a large pool of

compute, compute memory storage and applications are going to be dynamically assigned wherever there are resources. These applications can have priorities. So you can have a number of high-priority jobs that come in that basically take up all the resources on that machine and your application might get evicted. So if it gets evicted, and depending on how you're running it, it will get rescheduled to a different node. But there is a number reasons similar things can happen. Your workload is not guaranteed to remain running on the same node.

**[00:07:44] JM:** I could just attach a database to my application and write my state to the database, or I could use something like Redis and have a dedicated Redis cluster that I'm writing my state to. Is there anything wrong with that approach?

**[00:08:01] SA:** There's nothing wrong with that approach, but then the question is where does your Redis database or your external database store its bits? Are you to run those in containers? If you're going to run those in containers, you're going to hit the same wall again. Somewhere this thing needs to persist beyond the containers.

**[00:08:18] JM:** So the root idea here is that Kubernetes itself manages infrastructure as if the infrastructure is ephemeral rather than infrastructure that's going to be around for a long time. We've been managing infrastructure in the cloud for a while and I thought cloud infrastructure was always supposed to be treated as ephemeral, whether you're talking about a node or a VM that's running – I guess those can be the same thing. But these things are always supposed to be ephemeral. So how is this anything new?

**[00:08:53] SA:** So the interesting thing is that when you're running an cloud, you have a lot of managed services offered to you. So a lot of stateful – Or stateful services are offered to you, like Cloud SQL on GCP, for example, where you can write your state through an HTTP transaction and the storing of that state is now the problem of the cloud. They figure out where it's going to get stored. What disks it goes to? How it gets replicated?

If you want to run your own database, if you want to run some sort of application that needs to read or write to a block device or to a file system and you want that information to be persisted, it turtles all the way at the bottom, but someone's got to store something to disk, and if you're responsible for that, you're going to need a layer to be able to do that outside of the container.

**[00:09:44] JM:** Okay. So could you define the term container attached storage?

**[00:09:49] SA:** Yes. Think of it as there are lots and lots of different types of storage. Storage is a very broad term. So let's start with what container attached storage is not talking about. There are databases. There are messaging queues. There are all sorts of high-level storage systems, and those don't really fall into container attached storage. When we talk about container attached storage, we focus on two areas; file and block storage. The reason we focus on those things are that the data path for both of those has been standardized in this part of standard operating system.

So in the case of file, it's POSIX. In the case of block, it's SCSI. With Kubernetes, as long as the data path is standardized, we can control the control path. So for both block and file, we're able to take block and file volumes and make them available inside of a container for a workload to use regardless of where that workload is basically scheduled. As it moves around from machine to machine, node to node, we will automatically move that storage with it, and if that storage is capable of being persisted beyond the lifecycle of a single container, you are able to store your state beyond the lifecycle of a single container. So that is what I call container attached storage.

**[00:11:13] JM:** There are people who are listening who are thinking, "Wait, I store my stuff in a database. What is file storage? What is block storage?" Can you explain the relationship between those storage types and a database?

**[00:11:26] SA:** Absolutely. So if you think about the way that a database works, it is just another application that's running somewhere on your machine and it's going to be consuming CPU memory while it's running and it's going to be writing its states somewhere. Traditionally, if you're running on bare metal or a virtual machine, you're going to probably write it to that local disk, or you're going to have some remote disk that's available to you in the form of like a SAN or a network attached storage system, an NFS system. If you're running in cloud, you have these cloud virtual volumes. But somewhere, that state needs to be persisted.

Traditionally what happened was that the folks who wrote these databases and ran these databases would kind of make the determination of where they're going to store the bits and

they were responsible for really setting up the entire environment for this system. With Kubernetes, what we say is if you need to figure out how to write to a disk to store your state persistently, you can do so through the Kubernetes ecosystem and we take care of that automatic provisioning, automatic attachment, automatic mounting of the underlying discs, where the bits are actually going to be stored into your container.

File and block are two ways to access the disk. Raw block is you're accessing the disk directly. Most applications don't do this. They will instead consume a disk with a file system on it. When a disk has a file system on it, then you can consider it as a file volume and it will basically be a mounted directory that shows up inside your container and anything you write to that directory gets passed back to that volume to be persisted. I think that's the big difference between block and file.

**[00:13:15] JM:** So what is the relationship between a database or there are also objects stores, like there's S3-like interfaces for different systems that you want to run yourself, objects storage systems. How do these databases and data storage systems or something like Redis, how did these utilize block or file storage?

**[00:13:39] SA:** Sure. Databases vary from implementation to implementation. Some choose to write directly to block for performance reasons. They choose not to have a file system layer between them, and some will choose to support both block and file and some will choose only file. It depends on the implementation of the database. Then objects storage is a new cloud phenomenon, which is a very interesting pattern for persisting state.

The interesting thing about object is that the data path has not yet been standardized. If you go to one cloud provider, you have one interface for how you read and write data. If you go to another cloud provider, you have a different interface, and now you have on-prem objects stores that are popping up with their own interfaces.

So the data path hasn't been standardized, and what that means is that your application needs to be aware of the type of object store that it's consuming and it's kind of an application layer thing where they will modify the application to point to a specific object store and read and write from it.

The beauty of block and file is that since the data path has been standardized, your application can be portable with Kubernetes. If you consume block and file, you do so through a Kubernetes primitive called a persistent volume claim. A persistent volume claim is a way to be able to request storage in a generic way that is independent of the actual implementation of the storage. You say, “I want a terabyte of storage and I want it to be read-write.” That's pretty much it. Those are your requirements. Then the system takes care of figuring out how to make that available to you.

We support dynamic provisioning, which is we’ll provision these volumes automatically for you if your cluster supports that, or we also support pre-provision volumes where a cluster admin would go ahead and create persistent volume objects ahead of time to make available to you to consume. But you as the application developer don't have to worry about that storage administration layer. You just worry about what the requirements for your application are and you define that, and the definition in the form of a pod definition or a higher level workload, like stateful set, or replica set, combined with the persistent volume claims, that configuration is portable across cluster environments. So you don't have to modify your application if you're running on top of Kubernetes as long as Kubernetes is running in whatever environment you're on.

The way that I think about it is all these managed services ultimately need to read and write bits down to a disk somewhere, and this is the layer at the bottom. But it does beg the question of – Ultimately, as an application developer, I don't really care about the lower levels. I just want to store state and do it in it as easy a way as possible, and I think this is kind of the next challenge in terms of making things easier for folks.

If I want to run a stateful application on Kubernetes, how do I do it? Today, what you have to do is go to one of these – The folks that are running these databases need to provide custom installations for Kubernetes and manage their PVCs, PVs.

**[00:16:46] JM:** PVCs and PVs, meaning persistent volume claims and persistent volumes.

**[00:16:50] SA:** Right. Actually, they don't have to manage their persistent volume. They have to manage persistent volume claims. They have to figure out what the makeup of that application is. Is it a replica set? Is it a stateful set? This is custom for every application. So if you're not deeply familiar with how Kubernetes works or deeply familiar with how that particular stateful application works, it's a challenge to get it running. So when you hear folks saying that it's difficult to run stateful workloads on Kubernetes. It absolutely is.

Kubernetes provides a very, very powerful volume plug-in interface at the lower-level, which makes it easy to be able to consume persistent storage wherever your workload is. But the challenge is at the higher level, how do you actually run a stateful application with these primitives that exists? So the pattern that's beginning to emerge now is operators. If you have an application, that application will provide a set of custom resource definitions, which are Kubernetes API objects as well as a controller that runs inside Kubernetes to manage the lifecycle of that stateful application.

So it provides a custom Kubernetes-like interface for that application to be able to provision a new application in a very easy way, and that all the difficult logic of figuring out what the primitives are that they need to deploy for Kubernetes to run this application are part of that custom controller and CRD and they're abstracted away from the folks that are trying to deploy these things.

So you're going to see this operator pattern begin to emerge and evolve as we move forward. The SIG aps is pursuing a first-class, or a CRD called application, which is going to get married to operator soon and all of that is kind of being figured out right now. The application CRD just moved to beta and there's a lot of discussion going on in terms of what is it going to look like and all these. So a lot of exciting things going on there.

[SPONSOR MESSAGE]

**[00:18:53] JM:** This episode of Software Engineering Daily is sponsored by Datadog. Datadog integrates seamlessly with container technologies like Docker and Kubernetes so you can monitor your entire container cluster in real-time. See across all of your servers, containers,

apps and services in one place with powerful visualizations, sophisticated alerting, distributed tracing and APM.

Now Datadog has application performance monitoring for Java. Start monitoring your microservices today with a free trial, and as a bonus, Datadog will send you a free t-shirt. You can get both of those things by going to softwareengineeringdaily.com/datadog. That's softwareengineeringdaily.com/data.

Thank you, Datadog.

[INTERVIEW CONTINUED]

**[00:19:48] JM:** Let's unpack some of that terminology. There is a subsystem in Kubernetes called persistent volumes. Persistent volumes abstract the details of how storage is provided and how it's consumed. So you have a persistent volume that gets consumed by a container through the persistent volume claim. So a container can say, "I want a persistent volume claim for some type of storage," and the persistent volume API is going to say, "Okay. We've got that kind of storage." Then persistent volume is the one that's talking to the actual storage layer. Can you talk through a little bit more in detail about how users use persistent volume claims and persistent volumes?

**[00:20:34] SA:** Absolutely. Ultimately, we try to break up users into two distinct groups for Kubernetes. One is application developers and the other is cluster administrators. The reason we do this is application developers are a set of folks that we do not want to expose to the underlying implementation details for a cluster, but cluster administrators are people who want to be aware of what's going on in their cluster. They want to control it. They're the ones who are setting everything up.

So there's actually a third object – Well, there's three objects in play here. There's the persistent volume claim, the persistent volume and the storage class. The persistent volume claim is squarely an application developer-facing object, and application developers should only interact with the PVC. They shouldn't worry about storage classes or TVs. So they request storage in a generic way and magically it should be fulfilled.

In order for that magic to happen, somebody, the cluster administrator, needs to make sure that they make some sort of persistent storage available. When we first started Kubernetes, there was only the persistent volume object storage classes didn't exist, and the way that it worked was a cluster administrator would go ahead and provision a bunch of disks and say, “These are the set of disks that are available for my application developers to use,” and for each disk they created – They manually created a persistent volume object to represent that disk.

In a cloud environment, you can imagine this is something like a GC persistent disk Amazon EBS volume. On-prem it could be actually like something backed by EMC or NetApp or something like that, a file store. But regardless, the cluster administrator had to do a lot of manual work to set up these disks, and the challenge here was, one, it's a lot of manual work. Two, they had to predict what the usage patterns were for their application developers, and that was very painful.

So we introduced a new concept called a storage class and a provisioner. A provisioner is just a controller for being able to automatically provision new volumes on demand. The storage class basically says, “When somebody comes along with a request to persistent volume claim, a request for storage, please use this controller, this provisioner, to provision a new volume and pass this provisioner these set of parameters. These are opaque parameters that only the controller understands.”

What this allows is for dynamic provisioning on demand of storage when the user requests it. So now if you have the ability to provision these volumes dynamically instead of a cluster administrator and creating these PV objects ahead of time, they just create one or more storage class objects that point to a controller that knows how to provision volumes automatically, and that's it. They're done. Now their application developers are going to come along and they're going to create a PVC object. In that PVC object, they can directly reference a specific storage class if they want to, or if their cluster administrator marked one of the storage classes as default, they don’t actually need to specify the storage class at all.

So when they create that PVC object, Kubernetes will say, “Okay, there's a storage class here. I'm going to use that. I’m going to call out to this provision or to provision a new volume,” and

the provisioner will provision the new volume and automatically create a PV object to represent that new piece of storage and bind it to the persistent volume claim and the user is off and running.

So now as an application developer, if you have an application that's consuming a PVC, your application configuration file in the form of a pod or a higher-level workload combined with a PVC definition is portable. It worked on this cluster, but as long as if I move those to a different cluster, as long as that cluster administrator provides me a storage class, I'm going to have persistent storage automatically provisioned for this application when it starts.

So there's a lot of really cool magic going on at that lower level. It's a very powerful interface, and this idea of dynamically provisioning volumes on demand as application developers needs them is kind of unique to Kubernetes. Kubernetes incubated this, and it's kind of taken off. So at that lower level, there is a lot of really cool things going on. But at the higher level, how do you actually tie all of these things together into a stateful application. There're still a lot of challenges.

**[00:25:03] JM:** If I understand this API correctly, I just want to repeat it, because this is taking me a long time to grasp a little bit and I still don't get it very well. So I'm sure there're people out there that this will be helpful for. The interfaces that if I'm the cluster "operator", then I am standing up file storage and/or block storage with a persistent volume interface in front of it on a Kubernetes cluster. The application developers can get to say, "I know I need a SQL database." Through Kubernetes, they're going to make a persistent volume claim for file storage for that SQL database and they're going to describe – Under the covers, their part of the Kubernetes application is going to say, "Okay. I need a persistent volume claim with – Or let's make a persistent volume claim with this kind of configuration. I need this much file storage." Then the Kubernetes master is a cycling through the cluster and looking for persistent volume claims that have not been satisfied yet, and the Kubernetes master is going to assign a persistent volume that matches the persistent volume claim.

**[00:26:20] SA:** Or dynamically provision it.

**[00:26:22] JM:** Or dynamically provision it. Dynamically provision. Okay –

**[00:26:24] SA:** Using the storage class.

**[00:26:26] SA:** On the fly, there hasn't been enough persistent volumes that have been created yet. Let's us schedule some more.

**[00:26:32] SA:** Automatically create a new volume. Automatically create a PV and use that instead of having those pre-provision. That's exactly right. We can dig into a little bit of that lower layer if you're interested. So how does a cluster administrator expose storage? So this is very interesting. When we first started with Kubernetes, we had a set of volume plug-ins that we supported and they were built into the core of Kubernetes. Those included the obvious cloud volume. So on GC, GC persistent disk. On Amazon, EBS volumes. But also NFS, iSCSI, fiber channel and a number of other volume plugins were all baked into the core of Kubernetes. What these volume plugins would do is they define that for this specific type of storage, here is how you provision that volume. Here is how you attach that volume to a given nod if that is a concept that makes sense for that volume plugins, and here's how you actually mount that particular volume into the container, because ultimately those processes are different for the type of underlying storage that you're using.

If you're using, for example, say Portworx versus using a GC persistent disk on cloud, that is going to have a different interface for how you interact with it at the lower levels. So we created these volume plug-ins as an interface that can be implemented, and then Kubernetes uses those to do the operations that it needs, which are provisioning, attaching, mounting.

So that interface as we grew with Kubernetes kind of matured, but the number of volume plug-ins continue to expand, and it became very challenging for us as Kubernetes maintainers, because all of this is essentially third-party code that is living in the core of Kubernetes and it's very difficult to test. Often times for a lot of volume plugins, it actually went untested. It also meant that since these plug-ins were part of the core Kubernetes binaries, any bugs in this third-party code would cause core Kubernetes binaries to crash. It also meant that any – The security privileges you provide these security binaries are automatically given to these volume plug-ins.

So we wanted to get away from that model, and we found that vendors also wanted to get away from that model. If you're a storage vendor, you don't want to be tied to the Kubernetes release process, and if there's a bug you got to wait and follow the massive Kubernetes release process, and maybe they don't want to open source their code in some cases.

So, together with a vendor community – And Kubernetes at that time wasn't super mature. So what we did was we partnered with other cluster orchestrators, including Mesos, Cloud Foundry and Docker. We had Docker Swarm at the time. We all got together and decided we're going to create a standard called the container storage interface. The purpose of this was to have one standard for how a storage system, a storage vendor could plug their storage into a cluster orchestrator. A cluster orchestrator being Kubernetes, Mesos, Docekr Swarm, Cloud Foundry, and that work started about two years ago and we're very happy to announce that CSI went 1.0 this last month, and Kubernetes 1.13 pushed CSI support to GA. What this allows now is that vendors can basically develop new extensions for new storage systems independently of the Kubernetes release. The way that they develop them is basically just like any other Kubernetes. It's containerized. There is an interface that's defined. They implement that interface and they deploy on Kubernetes using the Kubernetes primitives that exists. You just do a Kubectl apply, and now support for a new storage system exists within Kubernetes and Kubernetes is able to allow application developers to begin to use new kinds of storage without having to modify the core Kubernetes code.

So if you're a cluster administrator, previously you were limited to the set of storage plug-ins that shipped with Kubernetes. You are of course responsible for setting up that storage system, but connecting that storage system to Kubernetes kind of worked out of the box. Now there's an additional step for storage administrators, which is you set up your storage system, set up your Kubernetes cluster, but you also deploy a CSI driver on your cluster to be able to allow Kubernetes to interact with that storage system.

**[00:31:01] JM:** We did a show on the container storage interface, and one thing I took away from that is these different storage providers, they're in a spot right now where – Well, I guess prior to the CSI, one of the problems that the CSI solved is if they want to write an API between their storage system and Cloud Foundry, for example, they have to write that and then they have to go and write a separate one for Kubernetes. These things are nontrivial to create, to

stand up a storage system and connect it in a reliable way orchestrators. So you have this end-by-end potential matrix and the CSI sort of solves that by standardizing that API layer.

To go back to that API layer, like let's just talk in terms of Kubernetes. The persistent volume claim versis persistent volume API. So as we talked about in the beginning, these are ephemeral clusters and you can stand up a container. You've got your persistent volume claim. It connects to a persistent volume. You've got your PostgreS database. It's writing to file storage that is supplied by the persistent volume that's meeting your persistent volume claim. What happens if your container gets blown away and your database application is no longer running? The database file has been written somewhere on a persistent volume. What happens then?

**[00:32:30] SA:** So it depends on how you set up your workload in Kubernetes. Traditionally, you'll set it up as a stateful set or a replica set, which means that when that pod that's running has an issue, for example, the container inside of it crashes for some reason, or that pod gets terminated because there's not enough resources on that machine. Kubernetes says, "Okay, there's another controller." For example, like a replica set controller or a Daemonset controller or a workload controller, stateful set controller, for example, will say, "I expect there to be maybe three pods running on this cluster for this particular stateful set, but I see that there's only two. One of them has crashed. I am going to create a new pod to ensure that there is always three running, because that's what the user requested," and it will create the new pod object.

When the new pod object is created, it is referencing the PVC that already exists in the system, and the PVC exists independent of the original pod, the original container, and the PVC is the pointer to this state that has been persisted outside of the cluster somewhere. So now when the pod get started, the Kubernetes controllers work to make sure that that storage that that PVC is pointing to gets the attached and mounted into that new container inside that new pod. When that new pod starts to run, all of that state is available inside that pod.

**[00:33:57] JM:** You touched on the operator pattern earlier. Can you explain the operator pattern in more detail?

**[00:34:03] SA:** Sure. It's definitely not my expertise, but I can touch on it. My understanding is that if you have a complicated app, it doesn't necessarily need to be stateful, but stateful apps

tend to be more complicated. Doing operations, like basically day two operations, things like, “I want to upgrade my application.” “I want to scale out my database.” “I want to scale in my database.” Those require a lot of careful orchestration that is above the Kubernetes layer.

This orchestration is unique to the application. So what these operators are, they’re just like every other component in Kubernetes. There is a piece of API, which is the CRD, which allows a user to be able to interact with this operator, and the operator itself is just a controller, which is a pod with an application that's monitoring these API objects and doing something.

So the idea is you deploy an operator and the operator starts to run. You create a CRD that defines, “I want my application to look like this. The operator sees that CRD and says, “Okay, the customer asked, or the user asked me to start a Redis instance, and these are all the configuration specifications for this instance. I am going to translate that into what is required for running that instance on Kubernetes. So it will automatically provision the workload objects and the PVC objects that are required to get that instance running.”

Once that instance is running, users can then create more CRDs to do more operations against that operator. So for example, if they want to upgrade to a new version of Redis, they can modify configuration file to say, “Instead of Redis 1.0 or whatever, I want to move to 1.2.” The operator sees that change and it orchestrates the update of the Redis instance by carefully deleting and updating the Kubernetes components that make up that instance. So in that way, it's basically a layer above Kubernetes to manage complicated applications.

**[00:36:09] JM:** You work at Google on GKE. You're also on the SIG storage. So that's the special interest group that is focused on open source efforts and conversations around storage. So you have a lot of conversations with both practitioners at Google and practitioners elsewhere around how they are working with storage, the frictions they’re encountering, the challenges at the cutting edge. Give me some picture of the best practices for managing stateful Kubernetes workloads and – Just whether you're an operator or an application developer.

**[00:36:50] SA:** Sure. A lot of this honestly is still being flushed out. It's early, but the general patterns that are beginning to emerge are, if you have a very simple stateful application, you can use PVCs directly, is a very powerful interface that'll work and you need to understand that

you're going to be responsible for figuring out how you're going to upgrade that application if downtime is an issue. How you're going to scale out the underlying storage if you need to. Things like that.

For larger stateful applications that are more complicated, that is not going to fly. You cannot expect end-users to figure out what the stateful sets and all the different Kubernetes primitives are that they're supposed to deploy on Kubernetes. They're going to have to rely on the application developer to provide some sort of higher abstraction that they can interact with, and that would be operators.

For application developers who manage these stateful, complicated stateful applications, they should look at the operator patterns. The operator pattern is starting to become standardized. There is machinery that is being built to allow kind of automatic generation of at least the basic code that's required for it, the basic API objects. There're a lot of work that's going on in that ecosystem. I'd say get involved with that. Take a look at what's going on. Figure out what's the latest and greatest and basically double down on operators.

**[00:38:13] JM:** What else goes on in the SIG around storage? This special interest group.

**[00:38:19] SA:** It has been a lot of work over the last year or so to get CSI support into Kubernetes. Independent of the SIG, there is also the CSI community, and in the CSI community we've been standardizing the interface. In Kubernetes, we've been picking up that interface and exposing it into Kubernetes. We're also working on features, feature additions to the storage system.

So when Kubernetes first came out, it only supported file. It didn't support raw block. Raw block has been moved to beta as of the 1.13 release. We also work on features basically at that layer to make it easier for Kubernetes users to be able to consume storage systems that are not necessarily accessible to every node in the cluster.

So you can imagine if you're running in cloud, you might have a cloud volume that is only accessible by a single zone, or if you're running in an on-premise environment, particular volume may only be available on a given rack. So we've been working to create an interface

that allows a storage system to advertise that, “Hey, this volume is not available to all nodes. Here are the subset of nodes that it will be available to,” and do so in a generic way so that we don't have to hardcode the concepts of rack, zone, etc., etc., into Kubernetes, but allow the Kubernetes scheduler to be aware that this volume has limitations and constraints and use those constraints to smartly schedule where that volume is going to be placed.

In addition, it could take a step further instead of basically having the workload follow where the volume lands, we now have the ability to have the scheduler influence where a volume is provisioned. Because community supports dynamic provisioning, we now have the ability to have a volume not be provisioned until the workload that's using it has been scheduled, and the scheduler at that time can help make the determination of where the volume should be provisioned and pass that information along to the storage system.

So we’ve been working on a number of features to kind of extend and make that lower level storage interface more and more powerful. Some of the areas that we’re going to be looking at in the next year or so are more storage management type features, things like how do you take a snapshot of a volume? The ability I think you alluded to that a Kubernetes cluster should kind of be ephemeral. You should be able to take down a cluster and be able to re-create it and everything should just exist. But what about the storage that backs it? If I lose a particular volume, how do I recover that?

Traditionally, there have been a set of operations that allow you to snapshot and restore a volume or back it up to an off-site and restore from that off-site, and we’re trying to figure out what those operations are going to look like in that Kubernetes interface. So last quarter, we released an alpha release of the snapshot, volume snapshotting and restoring. But there's going to be a lot of work in figuring out how that's going to mature and then how is it going to tie into that higher application level concepts of operators that we've been talking about.

Ultimately, there's a lot of cool work that we do at this storage layer, but application developers are not likely going to be using this directly. They’ll be using it through operators or CRDs and how do we make sure that we are exposing an interface that makes sense end-to-end?

 

So one of the things that I'd like to do over the next year' is work more closely with my partners in SIG aps to try and make sure that we come up with interfaces on both ends that meet in the middle and serve users in the way that that's best for them.

**[00:42:07] JM:** Let's talk about the example of snapshot. Why is Snapshot an important problem for Kubernetes to have built into it? I guess explain what snapshot is.

**[00:42:18] SA:** This can get very controversial depending on who you talk to. Within the SIG, the general idea is that it's a point in time image of the current state of a volume, and that point in time image can then be used at a later point even if you continued to write other things to that volume. It can be used to restore a volume to that previous state.

The current alpha implementation allows you to create a snapshot of a given volume and be able to restore that snapshot to a new volume. The reason that it's important to be able to do this is, for example, if you're a database administrator and you want to make a risky change to your database, it would be nice if you had some way to be able to undo the damage of something bad were to happen.

So traditionally what they do is they'll reach out to the storage system, create a snapshot of the volume, do the risky operation. If things don't work the way that they want to, they have the ability to restore to the previous state and continue from there. Now, they can still do this with Kubernetes, but the problem is that in order to do so, they have to go around Kubernetes and understand what the storage system is and how to poke that storage system for which volume to actually trigger that snapshot.

So in the beginning, we were questioning whether snapshots were something that makes sense in the Kubernetes API. What we landed on was, yes, because it is an operation that is not solely the domain of the storage administrator. Application administrators are interested in being able to issue this operation. We want to make the application administrators only have to interact with Kubernetes. To enable portability across different environments, we don't want them to have to go around and build systems to directly tickle some specific implementation of this cluster.

So for that reason, we decided that snapshot is important and we want to surface it in the Kubernetes API. But it is a challenging interface to try to introduce in Kubernetes in a thoughtful way for a number of reasons. One being the fact that Kubernetes is a declarative API where it is eventually consistent, but a snapshot is very much kind of an imperative, like take a snapshot now operation. So marrying that with the Kubernetes API has been a challenge. But we're going to continue to revise this and make it as useful for application developers as possible as we go forward.

[SPONSOR MESSAGE]

**[00:44:59] JM:** HPE OneView is a foundation for building a software-defined data center. HPE OneView integrates compute, storage and networking resources across your data center and leverages a unified API to enable IT to manage infrastructure as code. Deploy infrastructure faster. Simplify lifecycle maintenance for your servers. Give IT the ability to deliver infrastructure to developers as a service, like the public cloud.

Go to softwareengineeringdaily.com/HPE to learn about how HPE OneView can improve your infrastructure operations. HPE OneView has easy integrations with Terraform, Kubernetes, Docker and more than 30 other infrastructure management tools. HPE OneView was recently named as CRN's Enterprise Software Product of the Year. To learn more about how HPE OneView can help you simplify your hybrid operations, go to softwareengineering daily.com/HPE to learn more and support Software Engineering Daily.

Thanks to HPE for being a sponsor of Software Engineering Daily. We appreciate the support.

[INTERVIEW CONTINUED]

**[00:46:23] JM:** We did a show about Kubernetes snapshotting, and it was with a vendor that does this as part of their service offering, and there are a lot of vendors that are filling in the gaps of the storage world in Kubernetes where there are all of these little frictions that exist in managing state and Kubernetes. There are vendors that provide solutions to some of these things. Why is it that vendors are able to build solutions to this but it's not easy to just snap your fingers and get into Kubernetes API?

**[00:46:59] SA:** That's a great question, and the answer to that is that vendors work with a very tight set of – Like they control the full stack, essentially. They control the storage system and they can choose an interface that will work nicely with their system. When we introduce one of those operations into Kubernetes, we need to ensure that it works not just with a single vendor, but that it works across vendors and we need to do so in a way that doesn't end up being the lowest common denominator interface. We don't want to stifle the innovation that's going on at the storage layer with these storage vendors. We want them to be able to expose the full power of their storage system, but at the same time do so in a way that is going to be portable for application developers.

Doing so requires a lot of care and a lot of thoughtful consideration on what those API primitives are going to look like. So you're going to see storage vendors actually be far ahead of where Kubernetes itself is, and what we do is we look at the storage vendors and we figure out what are the common set of operations that would be useful to expose to application developers and then thoughtfully work with the storage vendors as well as end-users and everyone in our community to try to come up with an interface that will work for everyone.

**[00:48:19] JM:** There are open source projects related to orchestrating storage – Well, I guess I should say specifically, Rook is an orchestration system for storage that is deeply integrated with Kubernetes. Explain what Rook does.

**[00:48:32] SA:** Sure. The way that I see it is Rook is an operator for Ceph. Ceph is a storage system that can be deployed on top of the existing disks and provides – It's a software-defined storage system. So it's one way to be able to pool the underlying disks that you have and expose storage systems to your cluster. Running Ceph manually has been very challenging for those who are familiar with Ceph. So what Rook did was come up with an operator for deploying Ceph on top of Kubernetes.

I try to decouple the consumption of storage from the making available of storage, and Rook is squarely in the domain of making a specific type of storage system available to a cluster to use. Where Kubernetes storage SIG is interested is kind of the line outside of that, which is some

storage system exists. It could be Ceph via Rook. It could be NFS. It could be something else. How do I actually consume that and make that available to application developers?

**[00:49:35] JM:** If I understand correctly, Ceph offers file object and block storage through its API, and if Rook is on top of Ceph, Rook is a layer into Ceph and Rook provides a Kubernetes friendly interface to working with Ceph, what else do you need? Why do you care about something that's agnostic of Rook? If Rook takes care of all three of the storage formats, what else could there be?

**[00:50:02] SA:** Rook is one option for storage. There are a number of options, and as a cluster administrator, you have to look at what the requirements for your application developers are and choose what's right for you. Their storage can vary in the way that it's exposed. Rook will pool the underlying volumes that are available on every single machine and expose that to end users. But perhaps in your environment, you have a separate multimillion dollar storage system that you have that you would like to use. So we don't want to dictate any of that. We want to work with whatever storage is available to you in whatever environment that you're in.

**[00:50:37] JM:** Right. So if I'm a bank, I've got decades of different storage type that are running in appliances on my data center. I can't just say I, “All right, Rook. Solve my storage problem. I need persistent volume interfaces for all those different storage systems that are on my cluster,” if I want to run Kubernetes over them.

**[00:50:59] SA:** Yup.

**[00:51:00] JM:** So there's more and more cloud providers these days. Now there's like cloud providers built on cloud providers, and many of them have some kind of Kubernetes offering. I mean, all the problems that we're discussing for a single Kubernetes cluster trying to manage storage and manage the application developers in a friendly way, these problems are just exponentially harder for a cloud provider that is trying to do this in a systematic fashion.

If you’re a cloud provider, what are the additional challenges for thinking through the storage layer of Kubernetes?

**[00:51:37] SA:** Sure. Honestly, I think to a certain extent it's easier to be a cloud provider than it is to run on-prem. When you're in a cloud environment, you get to dictate the exact type of storage that your customers are going to have available to them, and you can provide a set of volume plug-ins that will allow your Kubernetes instance to be able to communicate with those volume plug-ins. Prior to CSI, those plug-ins were built into the core of Kubernetes.

So the challenges were in figuring out the bugs in those storage systems and the bugs of the Kubernetes layer, but there aren't inherent major complexities there. If a particular cloud the customer chooses to use other storage on top of what is provided by the cloud provider, for the most part that is the responsibility of the customer and the storage vendor that they choose on top that provides an additional software defined storage layer to figure that out how are you going to deploy it. How is it going to interface with Kubernetes?

The challenge I think is for folks trying to run Kubernetes on-prem in their own environments and trying to figure out how do I run these storage systems? How should they interact with Kubernetes? Who is going to provide me my CSI driver? How do I get it running? If I'm running within a virtualized environment, I'm running VMware or something, how do I expose storage through that VM layer into the VM so that they're available to the containers inside those? At least for our specific cloud provider, GKE, Google Cloud Kubernetes Engine, where we recognize that problem when we're trying to provide a managed solution for on-prem as well where we'll say, "We're trying to solve a lot of the problems that exist in that space and give folks their managed solution as well."

**[00:53:24] JM:** All right. Well, we're here at KubeCon and there's a gigantic amount of people here, 7,500 people. You work on GKE. So you're at the epicenter of Kubernetes development as much as anybody. Give me some predictions or tell me what you're excited about in the Kubernetes community.

**[00:53:44] SA:** That's a good question. I am very, very excited to be here. Tim Hawkins, who is the person that recruited me to the Kubernetes team early on in 2014 likes to say that when I'm at KubeCon, I feel like I'm with a thousand of my best. Honestly, I feel that way too. It's very nice. It's a great wonderful community. In terms of what I'm most excited about, I think there is a couple of things. One is just continued adoption and growth of Kubernetes and realizing that

we're reaching a state where a lot of the innovation is not necessarily happening within Kubernetes. It's happening at a layer above Kubernetes and figuring out how we enable that and how we bridge the gaps so that this ecosystem continues to grow.

Then the second is basically an extension of that, which is what are the kinds of features that we're going to expose there that are going to make application developers' lives easier and make deploying and managing applications in clustered environments dead simple. The way that I like to think about it is that back in the day, before there were operating systems, folks who are developing applications had to be innately aware of the specific hardware that they were deploying that particular application on, and that was painful and anytime they had to move that application they have to re-architect their application, and operating systems came along, exposed to standard interface and you had this proliferation of applications for individual machines.

In the distributed systems world, we've kind of been operating in that archaic way for a very long time. Distributed system developers architect their application for the specific environment that they run in, and that's painful. It's slow. They should be investing their efforts in optimizing and making their applications better, not battling with the environments that they run in. I think there is a lot of growth or a lot of opportunity to continue to do that. Kubernetes in my head is like an operating system for distributed system environments, and it's a basis. There's going to be things above it. There's going to be things below it, but I'm really excited to see where it goes over the next year.

**[00:55:56] JM:** You already see this, the higher stuff starting to happen with the degree to which frontend developers are feeling empowered. You can now be a team of a frontend developer and a designer and build a startup, which is kind of amazing. When you're talking about that layer on top of Kuberntes that's a higher level layer, what are the problems there that are being solved? What are you seeing on top of that layer? Why is it exciting to you?

**[00:56:24] SA:** Oh, man. Some of the projects that stand out there, of course we talked about the operator pattern a lot. This is just for running specific applications. But there's a couple other projects. Istio and Knative that are very, very interesting. When you start running applications at scale, you run into a set of problems like, "How do I secure my communication between the

different microservices that make up my application?" "How do I do logging?" "How do I do monitoring?" "How do I do service discovery between these applications?"

The way that a lot of folks initially approach this problem is that they make it the business of the application to figure that out. So every application has bits of this code tucked in to try and do this, and it doesn't really scale. If you want to change the way that service discovery happens, you need to update every application within your ecosystem. If you're running at large scales, that could be hundreds, thousands of microservices that you need to update.

So Istio is an open source project that was formed that built on top of the other primitives that Kubernetes provides and tries to solve these challenges of trying to make it easier to glue together microservices so that application developers can be even more – One more step removed from the complexity of how the application runs, and they can focus more on the application code.

Then Knative is a play in the serverless space which is taking things a step further and saying, "Just give me your application that you want or your code that you want to run as a container, and this system will figure out how to run it for you. You don't have to worry about Kubernetes. You don't have to worry about how things interact with each. We'll make it even more simple for you."

So I am really excited to see where these projects are going to go in the future and see adaption grow there and see how they're solving problems for customers in terms of just, "Don't worry about the infrastructure. Focus on applications."

**[00:58:26] JM:** Saad Ali, thank you for coming on Software Engineering Daily.

**[00:58:28] SA:** Thank you very much for having me. It was a pleasure.

[END OF INTERVIEW]

**[00:58:33] JM:** GoCD is a continuous delivery tool created by ThoughtWorks. It's open source and free to use, and GoCD has all the features you need for continuous delivery. Model your

deployment pipelines without installing any plug-ins. 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 project.

With GoCD running on Kubernetes, you define your build workflow and let GoCD provision and scale your infrastructure on-the-fly. GoCD agents use Kubernetes to scale as needed. Check out gocd.org/sedaily and learn about how you can get started. GoCD was built with the learnings of the ThoughtWorks engineering team who have talked about building the product in previous episodes of Software Engineering Daily, and it's great to see the continued progress on GoCD with the new Kubernetes integrations. You can check it out for yourself at gocd.org/sedaily.

Thank you so much to ThoughtWorks for being a longtime sponsor of Software Engineering Daily. We are proud to have ThoughtWorks and GoCD as sponsors of the show.

[END]