EPISODE 499

[INTRODUCTION]

[0:00:00.6] JM: Modern cloud applications store most of their data on hosted storage solutions. We use hosted block storage to back our databases, hosted object storage for objects such as videos and hosted file storage for file systems. Using the cloud provider for the storage systems can simplify scalability, durability and availability. It can be less painful than taking care of storage yourself.

One downside, the storage systems offered by the cloud providers are not open source. The APIs might vary from provider to provider. Wiring your application to a particular storage service on a particular cloud could tightly couple you to that cloud.

Rook is a project for managing storage built on Kubernetes. If you use a Rook cluster for your storage, you can port that storage model to any cloud and have a consistent API for object block and file storage.

In this episode, Bassam Tabara describes the state of cloud storage and why he started the Rook project.

[SPONSOR MESSAGE]

[0:01:17.9] JM: Azure Container Service simplifies the deployment, management and operations of Kubernetes. Eliminate the complicated planning and deployment of fully orchestrated containerized applications with Kubernetes. You can quickly provision clusters to be up and running in no time while simplifying your monitoring and cluster management through auto upgrades and a built-in operations console. Avoid being locked into any one vendor or resource. You can continue to work with the tools that you already know, such as Helm and move applications to any Kubernetes deployment.

Integrate with your choice of container registry, including Azure container registry. Also, quickly and efficiently scale to maximize your resource utilization without having to take your

applications off-line. Isolate your application from infrastructure failures and transparently scale the underlying infrastructure to meet growing demands, all while increasing the security, reliability and availability of critical business workloads with Azure.

Check out the Azure Container Service at aka.ms/acs. That's aka.ms/acs, and the link is in the show notes. Thank you to Azure Container Service for being a sponsor of Software Engineering Daily.

[INTERVIEW]

[0:02:44.0] JM: Bassam Tabara is an engineer who works on the Rook project. Bassam, welcome to Software Engineering Daily.

[0:02:49.5] BT: Hey, glad to be here.

[0:02:51.3] JM: So we are going to talk about storage on Kubernetes, and before we get into that, let's talk about more broadly how people store data in their cloud applications. We've got these different mediums, like object storage, block storage, file storage, memory, and I think there's plenty of people listening who don't have a great idea of how these different storage formats contrast and how they're used in different applications.

So what are the different ways that people store data in a cloud application? How does that map to different applications within the applications that people are building?

[0:03:35.0] BT: Yeah. I mean, essentially if you're running your cloud application, you're almost certainly storing state. Whether you know it, you're doing it directly or you're doing it indirectly. Otherwise, I think your application is — Stateless applications are very, very few or are almost nonexistent. The three kinds of storage I think you highlighted are most popular when people think about storage, although the term is expanding. So traditionally, block storage and file storage is what people thought of as storage. If you're running modern-day applications, you're probably not dealing with block and file directly.

Object storage is the new one. It started with S3 and a relatively new form of storage, but it is popular and some would argue that object storage is going to be — Most of the data in the world going forward is going to be somehow linked back to object storage or store it on objects storage systems.

On top of those, you could build other storage constructs like database record. If you're running MySQL or PostgreS or anything, you're likely using block storage underneath to implement both the storage backing up the database itself. So even if you're not, say, touching blocks towards directly, you're likely touching it indirectly if you're using a database, or if you are, say, using a key value store that's riding on top of a file system that's using block storage, or there are lots of forms of storage and they tend to be building blocks for other applications or other platform services that your app is consuming.

[0:05:14.6] JM: What the durability guarantees of these different formats when we talk about object storage, versus block storage, versus file storage, versus storage in a MySQL database? How should we evaluate durability?

[0:05:32.1] BT: So durability is essentially what are the failure conditions where your data does not survive. So it's different from availability in the sense that availability is about going down temporarily. Durability is about permanent data loss.

So if you look at — When you can achieve high durability with all three forms of storage that we talked about, you can certainly do 11 9's is what S3 talks about our or you can get that with all three forms of storage, the low-level storage. What you typically see is that it's all about redundancy. So if you want to store data with a high durability, you typically have either multiple copies of the data across failure domains or you have — Essentially erasure coding or other schemes where it's still a form of redundancy, but less expensive in terms of storage.

But the key trick there is to spread the data across fault domains, failure domains. So S3 has a good example, stores every object. When you write an object S3, it's stored in three data centers, so availability zones in Amazon. Every cloud vendor does a similar thing. If you store data in, say, Amazon EBS, it's rumored that it's within one data center. So it's in single availability zone, but it's written to two servers. So you could survive a server failure. If you

were to have two server failures in EBS, you lose a volume, and that's obviously happened for — It's happened to me. I'm sure it's happened to others.

So the more copies of the data you store, the more durability. The more copies of data you store across availability zones, the higher the durability, and you can achieve that really independent of the form of storage that you're using.

[0:07:19.8] JM: Another axis we can discuss storage across is that of latency or availability, and when people talk about disk versus in-memory, in-memory is often used to refer to this is going to be a low access speed. It's going to be high-speed, and on disk, it's going to be slower, but is more durable. But these terms are often productive, because you start something in Redis, for example, and I think Redis has features where you can store it on disk or in-memory depending on what is the backing storage system of the Redis abstraction that you're accessing. I think there are other systems. I'm not sure. Cassandra maybe has like a caching layer that is in-memory and then it also has other parts of it on disk, although I'm not actually sure if Cassandra does that. I'm sure other databases do that where they just have in-memory caching layers.

In any case, there is this gradient of availability of access speed. So when people refer to something being in-memory or on disk, does that actually mean anything or how do you consider this gradient of availability versus being written to persistent disk?

[0:08:39.9] BT: Certainly, latency is complicated thing. There are many factors that can impact latency. In general, you should think of it as if I was to write a bit of data and I want it to be durable, so I want it to survivor a reboot of a machine or survive even a failure of a machine. How long the time measured between when I initiate the request to write and when it's completed, that's latency, and if you trace that from some application or some database server, whomever, initiating a request to write, all the way through until it's actually made durable and then there were completion coming back to the application, there could be many different systems involved.

If you're writing to a local storage medium, so think of a traditional hard disk. You're typically going through file system to get to a block device in the kernel. Now you're in the kernel level. Depending on a storage medium, if it's a traditional hard disk and there is no [inaudible

0:09:40.7] or mirroring or anything going on, you're going to the storage controller on the hard disk and that is going to write it to a platter or write it to memory first, and if it has an internal battery that's keeping it alive, and then that's written to disk and then that whole thing comes back and completes to the application. That could take for a local hard disk typically less than a millisecond and it could take tens of milliseconds depending on how busy the system.

If you go over network to do the same, say you're using EBS and Amazon or you're using a network file system, or now you have the network in between your app and the actual storage medium, and that could vary because it depends on the network. It depends on how busy the network is. If you have redundancy, you're writing mirrors or keeping copies of the date, the example I used earlier of every object written to S3 is close to three data centers. Your latency is measured in terms of how — You initiate a put request in Amazon and you have to talk to three other servers or two other servers in two other data centers before the put request completes. There your latency is much higher, potentially 400 milliseconds is an average in Amazon.

You mentioned memory. Well, memory is typically attached to the processor. So your application is initiating the request to something that's in-memory. That completes in nanoseconds. It's super fast. The storage medium itself, it's not durable. So it writes it to an internal chip [inaudible 0:11:22.0] of some sort, and it's not durable, but it's complete super quickly.

One way to say it is that it's really all about trade-offs. You're trading off latency with persistence or durability, which sometimes your trading availability with durability. There is a very wide spectrum of choices here. Some apparent, some not as apparent as others, but it's a very wide spectrum and you could probably fall anywhere you want on that spectrum, especially with new devices like the NVMe drives or 3D XPoint that Intel released recently. It challenges our traditional understanding of memory, versus disk, versus tape, say in the 80s or 70s. We now have a very wide spectrum of storage devices.

[0:12:13.0] JM: And we didn't even introduced the excess of cost, because that's another one that you can explore; availability, latency, durability, cost, and that's why you see so many different storage systems today.

[0:12:27.4] BT: That is exactly right. Then there is consistency, which is yet another dimension. Durability and consistency are not necessarily the same thing, especially when you have multiple users. Yes, you're right. It's a very complicated space.

[0:12:40.3] JM: And then there's of course are you going to operate it yourself or somebody else is going to operate it for you. Lots of dimensions. There are open-source file systems. For example, Gluster is an open-source file system. There's open-source object storage systems like Ceph. It's obvious why people are going to use these in some context. If you've got your own on-premise hardware and you want a file system for it or an object storage system for it. If you're on your own hardware, then you can't use AWS services, so you would want to deploy this. But developers also sometimes use these open-source systems even in public clouds. Why would you use Gluster instead of something like Amazon elastic block storage and why would you Ceph over something like S3 even on a public cloud?

[0:13:33.9] BT: It's a very interesting questions. In general, I'd say there is a dimension of shall I use something that's open-source or shall I use something that's close source? I'd actually put S3 and EBS and cloud storage systems as close source. They're proprietary. They're specific to a cloud vendor. So there is that dimension. If I'm looking to build an application or has data-intensive workloads or is managing storage, the choice of store system is pretty important. Typically, what you'll see is if people are outside of cloud environments, so they're running on-premise or they have scenarios where they need to be hybrid, hybrid scenarios or across different [inaudible 0:14:18.7] that they own, the choices, they tend to be either traditional storage systems. Like think of old traditional IT hardware, dedicated storage appliances that you can buy, and they're almost all proprietary, or its open-source, and Cepth and Gluster are probably the two leading storage systems there. Obviously there are things like ZFS and others, but in the software defined storage category, I'd say Ceph and Gluster are popular choices.

In public clouds, I think that's less of the case. If you were running in Amazon, you are likely using S3. You're likely using EBS. However, more recently, I'd argue that people are starting to look things like Gluster and Ceph and Rook in this category as well being where storage in public clouds is starting to look dated when you look at running it for very dynamic applications, like the ones running on top of Kubernetes.

For example, a common thing that you hear when people are running, say, applications on Kubernetes in Amazon, is that EBS is too slow or it get stuck or we do all these work with trying to optimize failover for pods in Kubernetes, yet the EBS volume takes 10 times, sometimes 100 times more time to failover than, say, a pod does. Or if you want to run across multiple availability zones, then you can't failover EBS volumes across availability zones. You have to actually take a snapshot, send it over to S3 and then restore back in the other available [inaudible 0:16:09.6], which could take hours, if not days depending on the size of the volume.

You're seeing people looking at, “I'm running a public cloud, can I get a storage system that runs just like, say, my dynamic application would?” It fails over quickly. It has all these cognitive properties, it’s dynamically managed, it’s running across availability zones. It's has high performance. It has high durability. So you're seeing people looking at essentially running storage systems like they are running applications on these new cloud native platforms.

[SPONSOR MESSAGE]

[0:16:54.9] JM: DigitalOcean Spaces gives you simple object storage with a beautiful user interface. You need an easy way to host objects like images and videos. Your users need to upload objects like PDFs and music files. DigitalOcean built spaces, because every application uses objects storage. Spaces simplifies object storage with automatic scalability, reliability and low cost, but the user interface takes it over the top.

I’ve built a lot of web applications and I always use some kind of object storage. The other object storage dashboards that I’ve used are confusing, they’re painful, and they feel like they were built 10 years ago. DigitalOcean Spaces is modern object storage with a modern UI that you will love to use. It’s like the UI for Dropbox, but with the pricing of a raw object storage. I almost want to use it like a consumer product.

To try DigitalOcean Spaces, go to do.co/sedaily and get two months of spaces plus a $10 credit to use on any other DigitalOcean products. You get this credit, even if you have been with DigitalOcean for a while. You could spend it on spaces or you could spend it on anything else in DigitalOcean. It’s a nice added bonus just for trying out spaces.

The pricing is simple; $5 per month, which includes 250 gigabytes of storage and 1 terabyte of outbound bandwidth. There are no cost per request and additional storage is priced at the lowest rate available. Just a cent per gigabyte transferred and 2 cents per gigabyte stored. There won't be any surprises on your bill.

DigitalOcean simplifies the Cloud. They look for every opportunity to remove friction from a developer's experience. I'm already using DigitalOcean Spaces to host music and video files for a product that I'm building, and I love it. I think you will too. Check it out at do.co/sedaily and get that free $10 credit in addition to two months of spaces for free. That's do.co/sedaily.

[INTERVIEW CONTINUED]

[0:19:14.8] JM: Let's get into this discussion of managing data in Kubernetes. I wanted to give people some background for different storage systems and you were very generous in how you just provided the history and the context of how people store data in cloud applications. When Kubernetes got started, when it started getting popular, I guess, a year and a half, two years ago, three years ago, how were people doing data management? How did people store their data on Kubernetes in the early days?

[0:19:50.0] BT: The notion of volume plug-ins existed pretty early on as if I remember correctly. The notion that a container could use storage, and that shows up as a volume, has existed for a while, and Docker has their own volume plug-ins and now Kubernetes has volume plug-ins and now we're moving to world where there'll be a common volume plug-in called CSI. So that notion existed.

However, implicit in that is that you're using storage that is external to the climate of environment, and that is the status quo. That saved the majority of people, majority of deployments of Kubernetes. If you are using any form of persistent storage, you're typically interfacing with some external system. Most cloud deployments are using things like EBS or Google persistent disk or whatever your cloud vendor has in terms of block storage, and they're interfacing with it throughout volume plug-in. Going forward will be CSI, right now the existing volume plug-ins that ship as part of Kubernetes.

That means that the most common deployments of Kubernetes don't actually run storage inside the cloud native environment. They're external to it, and that's a very common deployments. If you are wanting people to manage your storage for you, which is not necessarily a bad place to be, that's the most common way that people do it today.

[0:21:17.7] JM: Let's talk about the different things that people are storing. Kubernetes deployment, the whole excitement about Kubernetes is you are using this as the underlying medium for your entire application infrastructure, and there are a lot of different types of data that are created in an application. With Kubernetes, you've got etcd which manages your configuration data. It's kind of, if I understand it correctly, it's like the state, basically the state of your application at any given time in terms of routes and DNS stuff and the number of different instances you have. Etcd is really the source of truth when it comes to the estate of your application, but then of course you've got your user data. That's probably in a database somewhere. Maybe it's in a Mongo database. You've got logging data. Your logs are stored somewhere. You've got monitoring data. You've got, I don't know, stack traces and stuff.

There're all these different kinds of data. So what are the different stories — If we're talking about — And we will get into Rook eventually and how people are using it. But if we're talking about these kinds of applications where most of the time, like if you're on a cloud provider, if you've got a Kubernetes deployment on a cloud provider and you've got all these different types of storage and you're mostly using, I guess, external storage systems, what are the different storage mediums that you're using for these different data sources; etcd and log data and so on?

[0:22:48.1] BT: Let's take the case where you're running in a public environment. Let's say it's AWS. Most deployments — I talked to a lot of customers that are running Kubernetes in AWS. Almost, I'd say — Yeah, the majority of them are running — Their computes and maybe network intensive workloads on Kubernetes, but the data intensive workloads are completely outside of it. So for example if they're using a database, they're very likely using Aurora or DynamoDB or any of the database services through RDS that are in Amazon.

If they are doing any big data or big data analysis, they're typically using EMR or Hadoop or Spark or Kafka that's provided by Amazon as a managed service, or if they are — All of those

services that Amazon has built are themselves using things like EBS and S3 underneath the covers, and of course they're also running on EC2 underneath the covers, but they service themselves to a typical customer as a managed service. That is the status quo. People are running all these amazing data services, platform data services not on Kubernetes. They are running as part of the cloud environment that they're in and they are specific to the cloud environment that they're in.

That's the most common deployment. Most people are not running databases in production on top of Kubernetes. Most people are not running queuing systems and big data frameworks on Kubernetes right now.

[0:24:26.2] JM: Right. What you're getting at is whether we're talking about my log data or my monitoring data or my user data, there is a managed service that can take care of putting this data inside of it and the managed service is built to be a domain-specific system for that type of data. We explored all these different axes that you can have for a data storage solution and there are certainly databases that are well-built for logging or time series data, and many of them are managed services, but I think what we're going to get at is that — Like you said, today, many people use a Kubernetes cluster to manage their application logic, and this is kind of low intensity workload stuff. Like if you're got services that are calculating things, but they're not doing it to manage their storage. They're not using Kubernetes itself as a data warehouse. They're writing to some cloud storage provider, like S3 or Aurora or Google Cloud Spanner or a hosted Mongo provider, and instead of this pattern, they could use a Kubernetes cluster for storage. You could run MySQL, or Redis, or Mongo, or Ceph, or Gluster, or whatever you want on your Kubernetes cluster. Why would you do that? Especially if you're on a cloud provider, why would you do that? Rather than just using the AWS or the Google managed services. I mean you're on the cloud provider already. Many Kubernetes deployments are on these cloud providers. Why not just plug into these managed services? Why would you want to figure out how to run your own database or time series database or file system on a cloud provider under your own Kubernetes cluster rather than using these managed services?

[0:26:20.5] BT: I think it goes back to why are they interested in Kubernetes and cloud native in the first place. The same reason that got these people to move to Kubernetes on top of, say, Amazon or on top of Google I think is the same reason that they would want to move more and

more on to that cloud native environment. Essentially, it's the cloud native promise of being able to be multi-cloud, run across cloud environments. Whether it's being able to run the same application, you write it once and being able to run it almost exactly as is across dev staging and production environments, as well as multiple cloud environments, or on-premise, or hybrid environments. That is the dream of being cognitive, and I think that if you are not able to run your data-intensive workloads, we're not going to get there.

The other part is that, really, it's still early days. I mean where Kubernetes is essentially three years old as I understand it. Typically, storage systems are the last to fall or the last to follow, because of the complexities that we just discussed. I don't know if you remember. I'm old enough to remember the last platform shift when we went from physical to virtual servers, and storage was the hardest problem even back then. Getting hypervisor and building all these amazing infrastructure, that happened fairly quickly. The problem quickly transitions to how do we run storage system in virtual environments.

If you go to a VM world even these days, it looks like a storage conference, and it's because storage is still one of the hardest problems to solve. Take the last platform shift that we went through 15 years ago almost. It is early days. I think the jury is still out. I think people would probably — If there were good storage platforms to run the on Kubernetes , I think then people would use them for the same reasons that they are using Kubernetes in the first place.

[0:28:21.7] JM: Okay. We've set the stage appropriately. If I am running my applications on one Kubernetes cluster and I've decided that I want to manage my own storage. I want to have a storage system that is native to Kubernetes, there're a couple patterns I could take. I could add more Kubernetes pods to the cluster that I'm already running and I could have those pods manage the storage of this pre-existing cluster, or I could spin up an entirely new cluster that is dedicated to storage. This would be like having a storage cluster, almost like where you reach out and you go to S3 or you go to Aurora, these remote databases and access it, you would have a Kubernetes cluster that is specifically for storage. How would you contrast these two approaches of just adding storage onto your application Kubernetes cluster versus spinning up a new cluster entirely devoted to storage?

[0:29:26.6] BT: Honestly, I think it comes back to trade-offs. There are both very reasonable approaches to the problem. In fact if you look at how cloud vendors themselves build their infrastructure, I'd say Amazon is, internally, they separate storage from computes, like if you were to walk logically into an Amazon data center, there'll be racks that are doing storage and there'll be racks that are doing compute. Google, I think is the opposite. They run more converged. Storage and computer on the same pizza boxes that are in their data centers. Both perfectly valid approaches.

I think part of what it comes down to is do you have the same team managing storage in computes? Do you want to provision hardware differently for those, and it comes down to a set of tradeoffs. I would think that going forward — I mean storage systems have requirements around — They run better when there is more networking available to them. Obviously, if you are running a store system, you probably want to provision different kinds of disks, or maybe in some cases if you're archiving less expensive disks. When the differences in hardware start to become more apparent, then sometimes you'd want to have a class of nodes or class of machines that are maybe even dedicated for storage. Even in that scenario though, I'd say you could call it a single cluster. I mean, Kubernetes has amazing facilities for labeling machines and selecting machines that are of a certain kind or have a certain class of hardware. You could run one cluster and you can partition in many different ways, and that's something that people do today. I think from a management standpoint, it makes sense to have less clusters than more, but I don't have a strong opinion which one is an obvious better solution. I think they're both equally good.

[0:31:20.2] JM: Rook is a project for managing storage on Kubernetes. Give an overview for the goals of the Rook project and maybe give a little bit of history about it.

[0:31:31.6] BT: Yeah, sure. Essentially when we started Rook, the idea was essentially kind of what we're talking about, which is why are people running storage systems, which are themselves distributed software. We have all these great cloud native environments and orchestration engines and containers brought this amazing, essentially new construct that we could use for running distributed systems. Why are we running storage systems outside of the cloud native environment?

When we started Rook, we thought, “Let's actually take one of the popular open-source storage systems,” Ceph in that case, “and let's figure out how to run it really well on top of Kubernetes.” The idea being, if I was to do that, then I can light up a Kubernetes cluster wherever I want and I can run my own EBS or I can run my own S3 or I can run my own EFS right there on top of the same Kubernetes cluster and have persistent storage of the three popular kinds; file block and object available to me wherever I am running. That was the motivation for Rook.

It turns out it's a really hard problem. It involves, one, making sure that the storage system itself could be packaged in the containers and orchestrated in this way, and it also turns out you have to build custom orchestration systems for storage. The pattern that we took with Rook was we built a custom controller, some people call it an operator, that is all about orchestrating storage systems and extend in Kubernetes to do so.

[0:33:02.7] JM: Rook, as you said, provides file block and objects storage, and we discussed these a little bit earlier. Describe the requirements for a storage system that can support all three of these storage types.

[0:33:19.2] BT: Rook orchestrates Ceph, and Ceph is one of these unique storage systems that can actually offer all three popular presentations of low-level storage in the same cluster. A typical thing is you see a lot of people build — They’d one medium and they build the rest on top of it. So you'll see people build object on top of file, file on top of block, block on top of object. All those combinations are possible.

Ceph essentially at its heart is in object store, but they built — The exposed block and they expose file systems on top of object storage. Gluster does it differently. Gluster of file system that exposes block and exposes object storage on top of file system. There are pros and cons to each of these, but it's really how they’re architected the backend, and so it is historical too.

[SPONSOR MESSAGE]

[0:34:23.4] JM: If you enjoy Software Engineering Daily, consider becoming a paid subscriber. Subscribers get access to premium episodes as well as ad free content. The premium episodes will be released roughly once a month and they’ll be similar to our recent episode, The Gravity

of Kubernetes. If you like that format, subscribe to hear more in the future. We've also taken all 650+ of our episodes. We've copied them and removed the ads, so paid subscribers will not hear advertisements if they listen to the podcast using the Software Engineering Daily app.

To support the show through your subscription, go to softwaredaily.com and click subscribe. Softwaredaily.com is our new platform that the community has been building in the open. We'd love for you to check it out even if you're not subscribing, but if you are subscribing, you can also listen to premium content and the ad free episodes on the website if you're a paid subscriber. So you can use softwaredaily.com for that.

Whether you pay to subscribe or not, just by listening to the show, you are supporting us. You could also tweet about us or write about us on Facebook or something. That's also additional ways to support us. Again, the minimal amount of supporting us by just listening is just fine with us.

Thank you.

[INTERVIEW CONTINUED]

[0:35:58.6] JM: Correct me if I'm wrong. I think of an object storage system as there is a unique URL for every object in this system, and if you've got the URL of that object, you can access it using that URL. If you wanted to build a file system on top of that, you would have to build a directory structure that organizes those different URLs. Similarly, if you wanted to build a — I could be getting out of my comfort zone here, but if you wanted to build a database on top of an object storage system, you would have to write a way to have an index of these unique URLs of objects storage systems. Am I getting this right?

[0:36:46.4] BT: Yeah. I mean, object storage tends to be — Like you said, you have a URL to an object. However, most object storage system implement S3 interface, and you could actually think of that as a file system. You're not far off if you think of every slash as being a directory. In fact you could list all the objects in a given prefix with S3. You could say, "Give me all the objects that are in this that have the same common prefix," and that starts to look like a directory listing..

The difference in my opinion is actually not how you think of structuring the data, but more about the consistency guarantees around the data. Object storage is fairly loose consistency. Multiple people can access the data without putting all these strict locks on them. File systems tend to be very strongly consistent. Think of the days where you're sharing a file system across a network and two people are working on the same file and they have to lock each other out of it to make sure that you don't lose data.

One way to think of object storage is it's actually a much looser file system, if that's one way to think about it, and it's more web friendly. You can access it through a URL. Honestly, they look very similar in terms of usage. Most people to use S3 use it as a replacement for file systems and it feels somewhat natural because of how data is stored there. Looks like files and directories.

[0:38:16.5] JM: Okay. So if I am using Rook, which uses Ceph under the surface to be an object storage system and I want to build my applications storage on top of Rook. If I'm building a system that manages comments on a forum or in a social network, it's not a huge deal if there is a read and write race. For example, if somebody makes a comment and then I read from a different node of that system and I see a stale series of comments. That's not a big deal. But if I was building a banking system on top of that same infrastructure, if you build like a banking system, you need it to be highly consistent, because if you have two different transactions that withdraw all of the money in a bank account and those two transactions are served by different nodes of the distributed storage system, then you're going to get two separate withdrawals of the entire bank account and you end up with a negative bank account balance, which is not okay. So when we're talking about this storage system that is at its base a loosely consistent object storage system, so are people building strongly consistent object storage systems on top of it?

[0:39:38.0] BT: So consistency is yet another dimension. In fact, the basis of Ceph is in object Store, but it's actually a consistent object store. So it's very strongly consistent. In fact even different implementations of object storage in public clouds can vary in consistency. Azure, for example, Azure is equivalent to object store is very consistent. Reads and writes. They're not delayed. S3 is eventually consistent.

Your point is right on if you were to do something like build a banking application of some sort, you're likely doing that on top of some sort of records, databases, and those are typically being built on top of things like file systems or block store systems, and if you want your database to be consistent, then your block store underneath it has to be consistent regardless of whether that itself is being built on top of object storage or not. The case with Rook and Ceph in this case is Rook orchestrates Ceph. Ceph is very strongly consistent. It's safe to use in the scenarios we just talked about.

[0:40:46.2] JM: So let's take a higher level data's access application, so elastic search, for example. Could I use elastic search with a backing storage system of Rook? Would that even make sense?

[0:41:00.4] BT: Yeah. Elastic search essentially relies on putting its data on local file systems, and you could use — An example deployment of this could be deploy — Say you want to run all of these on Kubernetes. You could, in your Kubernetes cluster, deploy Rook. Rook will provide Kubernetes with block storage, reliable persistent block storage. You can then run your Elasticsearchpods that consume block storage from the Rook cluster or the Rook pods, and you now have a completely contained Elasticsearch deployments; consistent, reliable, can scale all on top of Kubernetes without having to even reach out of the actual cluster, the Kubernetes cluster. It's all running in it.

[0:41:50.7] JM: So what are some other like database or data services that I might want to run on Rook? We can to think of so many. There's like HDFS, the Hadoop distributed file system. There is MySQL, Mongo. Can I just deploy any of these on my own Kubernetes cluster with a backing data system of Rook?

[0:42:15.2] BT: Yes. Essentially any of the ones you just mentioned and more could run directly on top of Kubernetes. If you are running Rook, then you can choose to — Some of those use — Like a good example is the one you just mentioned. If you take Elasticsearch, Elasticsearch is going to use block storage. Rook can provide that on Kubernetes. Sometimes you also want to say do data protection around Elasticsearch. You might want to take snapshots of it and save

them off to an object store. You can use object storage provided by Rook in this cluster or on a remote cluster do complete your DR story.

[0:42:58.8] JM: DR, disaster recovery.

[0:43:00.7] BT: Yes, sorry. Your disaster recovery story. The key idea here is that you can do that all with software that's running in containers, in pods on top of your Kubernetes cluster that are under your control.

[0:43:14.5] JM: Just to be clear, Rook can be your backing store for Elasticsearch or HDFS or MySQL or Redis or whatever, and Rook is backed by Ceph, which is this open-source object storage system. Just to complete the picture, what is backing Ceph? Is it just like discs?

[0:43:34.8] BT: It's turtles to the bottom.

[0:43:36.0] JM: Turtles to the bottom. No. What is backing Ceph?

[0:43:38.8] BT: Local storage. This is a technique that everybody does. We talked about earlier about — We can ask the same question. What's backing S3? What's backing S3 is a local disk on three independent servers spread across three independent availability zones. It's the same story for Ceph. You are storing data on storage nodes that you should put across failure domains, whether it's across different machines or across different discs, and those data is being replicated across them to make it reliable, to make a durable.

When you write to Ceph. It's writing based on configuration that you set. It's writing that same bit of data two or three other storage nodes, and you're relying on the property that not all three storage nodes are going to fail at the same time. That's the fundamental property that most storage systems rely on. If all three fail at the same time, then you are going to lose data. Statistically, if you do your job at making sure that these things are across availability zones or across failure domains, statistically, that doesn't happen and that's why you get numbers like 11 9's for durability.

[0:44:55.9] JM: Yeah. Much of this is taken care of by Ceph. So what exactly does Rook do? As I understand, Rook basically is providing this interface into an object storage system, but it provides the file and the block and the object storage. Is that what Rook does? It's a file interface or a block interface into Ceph?

[0:45:21.9] BT: Yeah. Think about it as the actual storage engine is Ceph. It could also be Gluster or Rook. We're actually looking at orchestrating Gluster as well, but the actual storage, the software that's responsible for the durability and the performance of accessing data is Ceph in this case.

Rook is a storage orchestrator. It is taking Ceph and essentially putting it in containers, and it has all the logic, the cluster management logic for running Ceph reliably on Kubernetes. It's doing the bootstrapping, the configuration, the scaling of Ceph, the rebalancing of the pods across availability zones as they become available. It's doing all the work that, say, an admin, a dedicated admin storage or a team of people that are responsible for running the storage system would have done with scripts or would have done with them manually sitting in front of a CLI or SSH to do all of that.

Rook makes that all work seamlessly on top of Kubernetes such that you get to the point where it's running my own EBS or my own S3. Looks like a one line command to get it all up and running. To do that, we had to write an orchestrator for the backend storage system, and that orchestrator is what's the unique part about Rook.

[0:46:57.1] JM: This is the operator pattern. Rook is managing the Ceph cluster using the operator pattern. Explain what the operator pattern is.

[0:47:06.0] BT: I think of Kubernetes as two things. One, it is a common set of abstractions that people use on top of infrastructure. Then two, it's a set of controllers that essentially make those abstractions happen run well. So for example, if you were to ask Kubernetes to run a piece of software and you want three copies of the software to be running at any time. The way you do that in Kubernetes is you create a replica set. This is the abstraction. Then Kubernetes internally has a controller for replica sets that will look at the current environment and ask the question,

“How many of this software is running right now?” If the answer is not three, it'll go make the changes necessary to keep it at three. That’s the controller.

Rook is essentially using that same pattern to extend Kubernetes to storage systems. We've added a concept of a storage cluster and a storage pool and an object store and a file system. Those are all new abstractions that we've extended Kubernetes to support. Just like say replica set is a built-in abstraction. Storage pool is a new one. Through Kubernetes extension points, we are able to add these new abstractions.

We also followed a similar path that Kubernetes team internally uses and created a set of controllers that understand these abstractions and are able to implement them in a cluster. When we see somebody create a storage pool, we will go turn around and do what's required to make the storage pool run in Kubernetes. That piece of logic is what people would say is an operator or a custom controller, and do so in a way without sacrificing the durability guarantees or the availability guarantees that storage systems must provide. It can get fairly complex when you want to, say, make changes, yet still serve all these data to applications. That's the pattern that Rook followed.

[0:49:14.3] JM: Okay. Walk us through a read and a write and the delete and some of these primitives that you would need to build to allow for Kubernetes to have the correct abstractions for managing storage.

[0:49:31.1] BT: Yes. The obstructions that we created or defined in Kubernetes as extension points are really about the deployment and management and scaling off the cluster. When an application wants to do a read or a write of a bit of data, it does not directly by going to things like an S3 interface. That path, the data path if you will, is almost unchanged and that’s a really good thing, because these things are super, super difficult to get right.

[0:49:58.2] JM: As in like remove all of the instances in my code where I write to S3 and replace them with, “I am writing to Rook.”

[0:50:05.2] BT: Yes, that would be a nonstarter, but you'll be surprised how many people still attempt to do that these days. Yes, your you're exactly right. Rook is more on the control path,

on the control plane. It will ensure that Ceph is running well or Gluster is running well or whatever is running well in your Kubernetes cluster such that when your application calls S3, it's there and available and scaling. Does that make sense?

[0:50:29.6] JM: Yes, it does. It makes complete sense. So what if I want to do like in-memory storage, because there are some in-memory distributed storage systems? Does that what I also use Rook for that or is Rook only good for interfacing with like the Ceph object storage system, the highly durable storage?

[0:50:50.1] BT: Yeah. No, I think that would be separate. Say if you are running Redis or a MemCachier or something like that, you're likely not using Rook for those scenarios. They themselves could actually [inaudible 0:51:01.6] Redis. I know it will actually backup to disk. There could be interface points there, but you're likely not deploying Redis with Rook.

Rook is really mostly about software defined storage orchestration. It's not a mouthful, but things like Gluster, things like Ceph or even other storage systems that expose file, block an object. That's the scope of Rook at this point.

[0:51:28.0] JM: We met at Cube-Con several weeks ago and it was a great conference. There's a whole lot of energy there, and I'm sure you talked to a number of people or you saw some presentations where people were using Rook in interesting ways. Can you describe the path that people take to using Rook? Are they migrating off of other systems to Rook? Are they just deploying fresh systems using Rook? Just give a description for the usability and the community around it.

[0:51:59.3] BT: Yeah. I mean we were just floored by the reception we got at the last Cube-Con in Austin. Essentially, people are wanting to do more and more of their stateful workloads on Kubernetes and they find Rook organically. I think if you just Google cloud native and storage, Rook shows up really high on the list just organically. There is a really healthy community building around Rook right now, and some level, it's great to see because it's validation that this is a problem of people once solved, and it's really great to see a community build around this.

Ever week I hear about a new use case where somebody is using Rook, and I'm always amazed by that. I think the one that you're probably talking about is the HBO folks were using Rook on top of their Prometheus servers in Amazon to solve some of the issues they're seeing with EBS. Then we found out that's actually powering Game of Thrones and we're like, "Oh! That's kind of cool," and they talked about it on stage in Austin, which was really nice to see.

Essentially, I'd say there is a category of people that are trying to run Kubernetes, say, on bare-metal or in their own environment or on their Jenkins server or staging servers and they need a storage story, and that Rook is by far the easiest thing to get deployed for those environments. It beats having to call your IT department to ask them for provision and NFS share. You'll be surprised, but that's a common thing we hear.

Then there are the class of people that are unhappy or could do better than, say, what they're getting in their public cloud in terms of storage and are starting to look at multi-cloud scenarios on storage. How do I run my entire stateful workloads on the Kubernetes environment? And Rook shows up there too.

[0:53:52.1] JM: I want to end our conversation with a bit of an exploration on the longer-term impact of Kubernetes. Since you're so deeply into this environment, to me it seems like Kubernetes is going to have a profound impact on the relationship between different cloud providers. It seems like there are going to be a host of new business models that are enabled by Kubernetes, like new types of companies that can be started because we have a unified, basically a distributed operating system that every cloud provider is running. So what are the long-term impacts that Kubernetes is going to have and what are the business models that people are going to build?

[0:54:37.1] BT: I'm one of those people. I got so excited about this space that I left my job and started a company now around this. So yes, I'll tell you I'm looking at it. You're exactly right. For the first time in the history of the cloud, we now are standing in a point where we have a common set of abstractions on top of all infrastructure, and that's a very profound place to be. You could provision infrastructure today using the Kubernetes API, and that is available in every cloud vendor and on bare-metal and really almost everywhere at this point.

Think about what that means. That means that if I wanted to deploy a set of containers that are running an application, I can do that in the same way regardless of the cloud vendor I choose, or whether I am going to do that on bare-metal or I'm going to do that in a hybrid environment. It's exactly the same operation that I would do on all of them. That's kind of interesting.

Like you said, we now have kind of the operating system of the cloud. One, kudos to the community for getting there. Two, now what? The now what is interesting, because if you do have that, well I think it's time to start looking at the ecosystem above that layer. What are the things should we build on top of that that we like to have the same properties? We'd like essentially to run databases, or other platform systems, or other things that we want to run uniformly across this new layer that we just worked so hard at getting to.

[0:56:16.5] JM: I won't keep you much longer, but one thing, I interviewed Brendan Burns recently and one thing he said that has just stuck out in my mind is the idea that you could have a proprietary binary that you could sell to people, and you could sell like a $99 — I was just thinking about Zendesk. You could sell a $99 version of Zendesk that runs on Kubernetes and I'm the Kubernetes on Zendesk provider, and your transaction with me begins and ends with purchasing $99 worth of software. You don't have to pay a subscription ad infinitum, because you're just paying whatever cloud provider you deploy your Kubernetes Zendesk thing to. You just pay the cost of the infrastructure as a service, which is going to be massively cheaper than paying for monthly Zendesk subscription. Is that too far out there? Am I being crazy talking about that, or does that sound conceivable to you?

[0:57:17.0] BT: It sounds conceivable for applications that could get to the level of automation and self-management, that you don't actually need an SRE or an ops team of some sort that is managing it. So Zendesk could get there. Zendesk could say, "Look. We've built all the automation, all the smarts for managing Zendesk itself. Looking at its logs, making sure it can scale, making sure it can handle all the users that want to use it, and we're going to ship it all in a $99 piece of software, a binary that runs on top Kubernetes, and when it does everything is happy from that point onwards."

I think in reality, however, is that you will see that there will be a class of services, especially when you start looking at databases of storage systems or others that have a lot more

complexity to them. I don't know if we'll get to the level of automation where it's 100% it manages itself, and then even if you did, I think you'd still — Large enterprises would still want an SLA of some sort. Still want somebody to pick up the phone or somebody to say, "I am responsible for this."

I guess my answer is I think we are in a much better place right now than we were even before this, and 80% of automation is actually great. I'd take that over zero. I don't know if we'll get to 100% of automation, and you still need a vendor to take the burden off of your hands of running these complex services.

[0:58:46.1] JM: All right. There's a lot to discuss, that we could go down a long rabbit hole. Anyway, Bassam, thank you for coming on the show. It's been great talking to you. Very detailed and bariatric conversation.

[0:58:56.8] BT: Yeah, thank you so much. That was fun.

[0:58:58.5] JM: Okay. Thanks, Bassam.

[END OF INTERVIEW]

[0:59:02.8] JM: GoCD is an open source continuous delivery server built by ThoughtWorks. GoCD provides continuous delivery out of the box with its built-in pipelines, advanced traceability and value stream visualization. With GoCD you can easily model, orchestrate and visualize complex workflows from end-to-end. GoCD supports modern infrastructure with elastic, on-demand agents and cloud deployments. The plugin ecosystem ensures that GoCD will work well within your own unique environment.

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

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

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

[END]