EPISODE 498

[INTRODUCTION]

[0:00:00.3] JM: A common problem in a distributed system; how do you take a snapshot of the global state of that system? Snapshot is difficult because you need to tell every node in the system to simultaneously record its state. There are several reasons you might want to take a snapshot. You might want to take a picture of the global state for the purposes of debugging or you might want to take a comprehensive snapshot of your system, including the database and port your system from one cloud to another, or you might just need to take a snapshot for disaster recovery.

When a Kubernetes application is deployed, its initial configuration is described in config files. After a deployment, the state of the application might change. Some nodes dies, some services get scaled up. At any given time, the current state of a Kubernetes cluster is described by etcd, a distributed key-value store.

Niraj Tolia is the CEO of Kasten, a company that provides data management, backups and disaster recovery for Kubernetes applications. Niraj joins the show to describe how Kubernetes deployments manage state and what the modern business environment is around Kubernetes. How do you build an enterprise business on this rapidly shifting landscape? It's a fascinating discussion of both the technical and the business conversations around Kubernetes. I hope you enjoy it.

[SPONSOR MESSAGE]

[0:01:33.4] 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]

[0:03:00.4] JM: Niraj Tolia is the CEO of Kasten. Niraj, welcome to Software Engineering Daily.

[0:03:04.8] NT: It is great to be here. Thank you so much for having me.

[0:03:07.3] JM: Absolutely. We're just coming off being at Cube-Con together and having some interesting conversations about Kubernetes, and one of the things about running a distributed application, which all of the applications in Kubernetes are, is that we need to be aware of how we are managing state, because we're always assuming that any of our application nodes can crash at any time, and so if my application's state is sitting on a container and the container crashes and I have not persisted that data somehow, then that state is lost forever.

How do modern distributed applications such as those running on Kubernetes, how do they manage state?

[0:03:48.2] NT: It's a great question. There two different ways people think about state management. For us it's just provisioning of something that is stable across reboots, and when we go look at the Kubernetes world, there are two different ways again of doing that. The first one that people are the most familiar with is something they call persistent volumes, and there is now within Kubernetes for the last few releases, there's been something else called dynamic provisioning of said persistent volumes, and so in that way you can get shared storage that is

attached to a container that will be persistent even if you are container or your pod is rescheduled on another node, gets restarted, gets upgraded. Your data always stays there.

In upcoming releases, we'll see also support for local storage that is useful for certain category of applications that really want your fast local performance coming out of local fast discs as an example.

[0:04:42.4] JM: So you're saying that what we have today is you can basically mount a file system to a persistent disk-based file system to your Kubernetes cluster, I guess specifically to a container and you can also mount that same file system to another container and you can have one container writing to that file system and have another container reading from that file system. Is that correct?

[0:05:07.6] NT: That this correct only in a few limited cases. That only works if you might be using a discreet file system. The majority of customers we talked at Kasten tend to be using block storage, which only allows single instance use. That is only a single container can usually read and write from the underlying block device.

[0:05:26.0] JM: Let's go through these different types of storage just to level set before we get into storage/Kubernetes. Block storage, blob storage, databases, memory, maybe you could just describe the different flavors of storage that we're using in our applications these days.

[0:05:44.1] NT: Sure. I'm going to answer in two different ways. We're going to talk about storage and then we're going to talk about state, because those two things are slightly different even in most worlds, but in particular for the Kubernetes world. When we talk about storage, the way I look at it is we will see three forms of persistent storage. We see a lot of block in these environments of different kinds of flavors. We see object, mostly S3 compliant objects stores. You have 800-pound gorilla in the room, that is AWS S3, that has a lot of usage, but then you also have open-source projects like [inaudible 0:06:18.1] that fill the gap, that themselves a containerized.

Then we have file storage. Generally, as you correctly mentioned, even if the amount of block device today within a pod or a container running within Kubernetes, you see that as a file

system instead of a raw block device as of today, but that might change in the future. But on the disparate file system side of things, this is something like an NFS server that is centrally located that multiple containers can mount a sub-part within those. That's one way of doing multiple reader-writer sharing, but that comes with different performance implications again.

There is obviously in-memory stuff that people use for caching etc., but that tends to be more ephemeral than some of the other persistent stuff. But it's also important to think about state, and that's where some of the other divisions come in. People generally think of state as relational data databases, as you mentioned, are no NoSQL systems, but in this world the other state we really do need to care about carefully is also configuration of the system and secrets. So if you're doing things like a time machine approach where you're going to back in time, you want those to be tied together, and those, again, things one wants to protect and remember as you have continued operation of a cluster in production.

[0:07:35.8] JM: When you refer to the data that is configuration data, what kinds of things are stored in configuration data?

[0:07:44.4] NT: That's a good question. So for example where to find other services, things that might be particular to the service of the application running, such as configuration parameters. All of those things belong in configuration data; usernames, etc. On the secret side of things, you have things like passwords and credentials to use an external service. So all of that fits under the bucket of secrets.

[0:08:10.2] JM: If I understand Kubernetes correctly, this configuration data is stored in etcd? Is that right?

[0:08:16.7] NT: That is correct.

[0:08:17.4] JM: Explain what etcd is.

[0:08:19.0] NT: Think of etcd as a key-value store, to keep it at the high level, where it can be used for multiple things. In the core Kubernetes use case, it is used to store configuration of the system and not just configuration, but also desired configuration, that Kubernetes works very

hard, which is called a declarative approach to reconcile the state of the world to what the desired state of the world should be. And all of that information as to what application the workloads are running where what configuration looks like, all of that that tends to be stored in etcd today within Kubernetes.

Obviously there's also possible for applications to use at etcd too, but that generally tends to be limited, because the way one looks at etcd is more often in-memory store as you had mentioned earlier. So therefore it scalability is limited compared to other system that can use this to take Cassandra, MySQL, PostgreS, etc.

[0:09:17.2] JM: This is one of the key selling points of people who might consider using a managed Kubernetes provider, is when you go with the Kubernetes as a service, like the ones from Google, Amazon, Microsoft CoreOS, etc., etc., etc., there's a lot of these, they give you high-availability etcd, and that's actually something that's pretty hard to achieve if you're rolling your own Kubernetes.

[0:09:46.7] NT: Yes and no. To our customers, if ever a managed services is available for running Kubernetes, I think that tends to be the strongly preferred option. However, the state of tools in the ecosystem whether you talk about things like COPs or other installers that are coming out in the open- source ecosystem, make it much easier to deploy HA clusters today.

This is, again, one of those things where the tooling has evolved such that it's easier to set up etcd in a multi-node configuration, such that if a single node goes down, it does not impact the availability of a cluster.

[0:10:23.3] JM: Can you talk more about — Just talk more about etcd, because first of all, I didn't know that. I didn't know that it was getting easier to make your etcd highly available. I mean, is it still hard to make it, for example, multi-cloud or to make your etcd more durable, because you said etcd, it's typically an in-memory store. Does it have a disk-based back?

[0:10:44.9] NT: It does back the disk. Let's touch upon the two parts of the question. First of all, the general configuration of the recommended configuration of running etcd in a highly available environment. Generally, people like it to stretch across different availability zones as AWS calls

it within the same region. That is data centers that have independent power networking sources. So they have the independent failure domains, but they're relatively close to each other, such as on the other side of the network or a mile away from each other. So latencies are low.

The general multi-node etcd deployments tends to be that. It's not built today to stretch across clusters that might be, say for example on different coasts of the United States or in different countries that are much further away from each other in Europe. That would be the recommended configuration.

Now Etcd obviously does back up on disc. So if you have, say for example, a full system failure, your cluster will come back up and etcd will read data of discs. There is that. However, the other thing that we talk to customers about is etcd by itself, by being a place where people store data, they want to be persistent. People care about disaster recovery of that, even for applications that might be completely stateless, because in the edge case and the IoT case, we run to customers. An example would be a retailer, where they're running Kubernetes and stores themselves, and in that scenario if the entire cluster goes down, and it's actually a disaster, someone accidentally wipe discs as an example, they want to bring the cluster back up as quickly as possible, and in which case they're trying to figure out what does it mean to protect etcd. What can they port over to a new cluster? What doesn't go across? There's some application logic there, but people do care about how to extract data that's in etcd too.

[0:12:36.9] JM: How severe of a problem is it if your Kubernetes cluster goes down and some of your etcd data is not saved and you have to reboot off of a slightly stale etcd configuration?

[0:12:54.5] NT: Two schools of thought of this one, and I'll tell you which school I belong to. So we run into — Again, talking from customer experience here. Also as a vendor, we provide solutions for customers, but we talk to everyone as to what they deploy, how they deploy. So there is an approach that says, "Let me go protect etcd, and I will go bring it back up and then I might need to manually fix things up, because things might have shifted since my last snapshot of etcd state," as an example.

However, a newer school of thought, which they I'm a bigger fan of, tends to be I've generally set up a CICD pipeline to not just deploy my application, but also my clusters, or I have a lot of

tooling around that. So what is simpler for me is that I just bring up an entirely new cluster and then the rest of my pipeline is also set up such that I can redeploy all my applications within that. Again, this works when your applications are mostly stateless or semi-stateless, but, for example, we've been talking to a customer that has multiple large Kubernetes spread across multiple AWS regions, and if for some reason, whether it be instability, something incorrect happened, a cluster goes down, this solution is to just load the entire cluster wave, bring a new one up, deploy everything. If you're talking in the range of 50 to 100 nodes, they manage to do this in under 40 minutes. Now, they can failover to another cluster while this is going on. So they are highly available still, but that's another example of what happens when a cluster gets destroyed.

[0:14:32.0] JM: Let me explain something that I don't quite understand. So if I've got an application, it's running on Kubernetes, I'm using all these different types of storage. So I use object storage, like S3. If my users upload a video or upload some other kind of media and I use MongoDB if I'm storing data about a user. I'm storing their username and password and their preferences and stuff kind of there. For etcd, my understanding is that this is like the configuration of the cluster. So it's how are load balancers interfacing with each other? How is my — Maybe what are the IP addresses of certain services?

What I don't understand is why — Like in under what conditions that data needs to be durable? It'll be durable in the sense that maybe I have like a version controlled set of YAML files that describe how I'm deploying my cluster, but why is it important to have the state of the cluster at any given time, which is an etcd, which can change a lot of the time. Why is it important to have that information be durable, because just redeploy another Kubernetes cluster based on those YAML configuration files?

[0:15:50.3] NT: So I actually agree with you, right? So I belong to that school of thought to where —There are two kinds of data when you think about etcd in particular, but you brought up a lot of more interesting topic that is application data that we'll dig into in a little bit too. But from the etcd point of view, there are some data that is temporary. Current IP addresses assigned to pods as an example.

There's some other information one wants to remember, such as service definitions, as you mentioned, load balance examples. Now, why you might want to remember that? I believe we want to remember that within the context of the application verses from the context of the cluster. So that's the difference in terms of what we're talking about, that is instead of trying to do, let's say, full cluster disaster recovery, it's more meaningful to capture the state of an application as it existed when we talk about all the four kinds of states we touched upon earlier. The reason to do this is for multiple reasons, because it's not just about the operator of these platforms and how we make their lives easier, but it's also about the developer, giving them a better developer experience, giving them better developer agility.

From the operator point of view, they might want to go restore the application state to something that might exist an hour ago, and any shift in some of the service definitions, they might want to roll those back to. So that's an example of that.

From the development point of view, there might figure out, "Hey, I want to get a snapshot of my application data included as what it looked like a month ago or a week ago, because I need to go do some debugging on this thing, or I want to clone this entire application stack into another cluster, into another namespace, because, again, I need to run my newly revised code against that older data."

Those examples show up and for that to make it really easy for them, so they don't need to worry about what is an ingress of this look like, what do the service definitions for this look like. Capturing that state simply makes that workflow much simpler for them.

[SPONSOR MESSAGE]

[0:18:05.9] JM: The octopus, a sea creature known for its intelligence and flexibility. Octopus Deploy, a friendly deployment automation tool for deploying applications like .NET apps, Java apps and more. Ask any developer and they'll tell you that it's never fun pushing code at 5 p.m. on a Friday and then crossing your fingers hoping for the best. We've all been there. We've all done that, and that's where Octopus Deploy comes into the picture.

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

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

[INTERVIEW CONTINUED]

[0:19:37.7] JM: That certainly gets us towards talking about what your company Kasten does. So I guess the purpose of doing a snapshot of this etcd data and basically the cluster-wide data, it's mostly to audit how your Kubernetes cluster can maybe get into funky states, or like if you're looking in your database, you're looking in your Mongo database, like this is — How did the user data get to this type of situation? You look at your different application instances, your "micro-services," you're looking through the code and you're like, "I can't understand how this would have occurred," and maybe you're even looking through your distributed traces and you're saying, "It just does not make sense how this would've occurred," and when you look at the state of your cluster, even the present state of your cluster, you look at your current etcd and you would say, "Oh, it still does not make any sense how this situation would happen." Under those kinds of conditions, you might want to be able to look back in time and say, "How did the state of the cluster look two hours ago when this data was written to my database, because that's the only way that I'm going to debug this problem."

[0:20:48.4] NT: Yes. As you mentioned, the other tools that will probably get you there faster, such as if you have the right logging tracing set up. You will have application audit logs. Kubernetes has been doing a bunch of work in that space too.

The way I look at it is this is one of those who done it kind of puzzles, where the multiple different moving pieces, and that is probably a subject for much longer conversation as to how you debug some of these newer micro-service applications in production. You're correct that

there's data that flows in for multiple different components that tie into a larger picture to help you debug as to what might've happened and why.

[0:21:27.3] JM: I see. So your company does — If I understand the technology correctly, take snapshots of the application data across an entire Kubernetes clusters. Is that right?

[0:21:39.6] NT: That is correct, but I think we should step back a second to look at the high level goal, right? I think, generally in the community today, there is — And we saw some of these at Cube-Con, where when we were on the show floor, we spoke to people and generally folks fell into two different kind of bucket. Some of the people have already done stateful in production and they're moving forward, but there's a lot of uncertainty or confusion around is Kubernetes ready for running stateful production workloads.

At the very high level, what we try to do as a company, both via open-source efforts as well as via commercial product, is to tell people and make it easy for them to build, deploy and manage stateful applications running in these environments. We want to take a lot of the challenges away, and this is one of those cases where perceptions have a half-life of people thinking that things weren't stable, and a lot of the world has changed as far as cloud native environments go. I think the timing is right. So, in general, those are some of the pain point that we're trying to address today. In particular, snapshot is just a set of what we do for our customers, because our goal is to be much broader.

Let's go back to the questions you raised about things like etcd, about things like Mango, and the other sub-point you raised in passing that is applications today tend to be polyglot, where they use multiple data services underneath them, but there'd be a combination of Mongo or Redis and Cassandra or a NoSQL and more traditional SQL system in the same application. We see a lot of that happening today. So how do you coordinate these kind of applications? How do you approach data management from and application layers what will look like?

There is, again, using the application as a layer of encapsulation. How do you make it easy both from the operator side of view that care about at the enterprise scale, a lot of different applications, a lot of different businesses running on the platform, but also about the developer,

making sure they have a great experience, that for them it is as easy to build and test stateful applications as it has been for stateless once today.

[0:23:46.4] JM: Stateful application. So for example, when I ask the serverless people — When I'm doing an interview about serverless application I say, "Okay. So how do serverless applications manage state?" and typically they say, "Well, you write your state to Redis," which is an in-memory storage system that can be made to be persistent as well, or you write to blob storage, or you write to your database, and it sounds like the same rules of a 12-factor app. You don't keep state in-memory on the container itself. You don't keep state in-memory on your serverless container. Similarly, you wouldn't keep state in- memory on a Kubernetes container I think is the widely accepted wisdom. What's changing? Help me — When you say, "We want to be able to build stateful applications on Kubernetes," where are we talking about a state, because we can put state in a database, we can put state in S3, we could put state in Redis, we could put state in Mongo. That's a stateful application. So what kind of stateful application are you wanting to enable?

[0:24:45.6] NT: Okay. From the broad perspective, from Kasten's perspective, we aren't clearly concerned about where state resides. Whether it'd be a managed service within the container or even some other place like a blobster. We will go handle all of that. I think the underlying point independent of what we do is that there is this 12-factor app manifested that came out a number of years ago. Heroku was one of the first production deployments. The way I look at, as people using that philosophy at scale. Really, they all have state. They've just put it into some other location generally outside the cluster, which tends to make management harder.

In the Kubernetes world, you're again correct, where things started off in a very stateless manner, but I do believe that we are now at a place, especially with the 1-9 release coming out, that it is possible in production and with confidence to the stateful stuff within your containers itself, and that actually has a lot of advantages that sticking state outside of the system doesn't bring along with it. Weather it'd be for test their workflows, whether it'd be for foster developer agility, combined snapshotting. Those are the things that it leverages, as an example.

Let's say I am using an application, but my state is in some multitenant database that's outside of my cluster. The issue with that is if I ever want to snapshot my entire application stack, I don't

have a good way of doing that. But if I bring it all within Kubernetes itself and use the same tooling and the power that Kubernetes API gives you, there is a lot of value to be had there, because you can now snapshot not just data, but the entire application stack, and that's a very powerful construct when you think of it both from the developer point of view, but also the operator point of view that cares about keeping the lights on and making sure things are in sync. Sometimes it's for regulation purposes. Sometimes it's for other internal business objectives, but bringing state into Kubernetes in particular, both the system is ready to do that today and it gives you a lot of advantages in terms of not having to manage two systems, being able to manage it coherently that I believe is the right thing to do.

[0:27:00.2] JM: Okay. So let's pick these two concepts apart, like first, why we would want a stateful application that is entirely within the container, with on Kubernetes, and then we'll talk about snapshotting. Why snapshotting is useful.

I think about my experience on my laptop, my Apple laptop, and the experience is entirely self-contained. I'm not thinking about what are the stateful operations. Like we're just talking about — Let's say I'm on an airplane and I'm not connected to the cloud and I'm just like writing a text document and I'm like editing a text document, and then I go and write a post-it to myself or I go and create a piece of music, and I do all of that while I'm on the plane, and then I shut my laptop, and I'm not really worried about losing any of these data, unless the plane goes down or if I drop my computer perhaps. Why aren't containers like that, or are containers getting like that?

[0:27:57.4] NT: I think containers are getting like that. So this is when we touched upon persistent volumes much earlier in this conversation. Containers are getting like that, such that once you store stuff to a persistent volume in the Kubernetes world, you know your data is going to be there. You come back up at some other location, it's still there. You come back three hours later or a week later, it's still there. So that has changed.

[0:28:22.0] JM: Okay. Right. So really what you're talking about is the idea of this persistent volume that we can attach to our containers. That's basically — Like I have the hard disk that is connected to my computer. Even if my computer goes down, I have to restart or it runs out of

battery or whatever. My information is saved on disk. That's basically the same as these mounted file systems that we can attach to our Kubernetes containers.

[0:28:49.4] NT: A little bit more powerful. To take your analogy a little further, it's equivalent of you have your applications, whether it'd be iMovie or notes application, etc., on your laptop. Think of that as a part of your compute and your compute configuration, and then you have the date that corresponds to those applications. So now what we're saying is it really doesn't matter whether it's a different piece of hardware to different physical server or VM. It really doesn't matter where things run. When you come back up, all your applications are going to be there as you remember them at the version. Everything self-contained and the data associated with that.

So when you talk about the advantages of bringing state into the system, I think the notion of being self-contained is very powerful, because from the developer point of view, they have the choice of picking and verifying against their version of the application stack they want to use. Whether this version of Mongo or this version of Cassandra, because it gives them extra features, versus what ops has said, "Hey, we only have this version of MySQL and this version of PostgreS available." They can use what's best for them. It's self-contained. They have no external dependencies, and that's a lot more powerful than sometimes saying, "I depend on an external service someone else might be managing."

[0:30:01.5] JM: I see. So from the operator's perspective, I'm looking out at all the containers that are running across my cluster, and in a world without these mounted volumes, the operator is saying, "Oh gosh! I have to manage a Mongo container cluster and I've also got to manage the application container cluster, because the application container cluster is — It's going to fall over all the time and it needs to write all its data to these external Mongo clusters, and so I've got to know operational how to deal with both of these things," but if instead the developer made a container that was backed by a file system and a durable volume, and then just deployed Mongo within that self-contained world, the dev ops person, the SRE, the operator is just looking and they just see an application container. They don't see an application container and then a Mongo container somewhere else.

[0:30:55.6] NT: Yeah. That's a pro and a con. That's, again, where we come in. Some of the challenges I'm going to now talk about don't always exist if your company is small, a small

startup like us or if it's a single product, a single application company, but as soon as it get to enterprise scale, the scale of problems change. We want to impart developers exactly in the way you mentioned; give them the flexibility and control over the applications they are building and however they want to build that application, but ops from that side, they still care about keeping the lights on, and it's not just about making sure my application stays up, but it's about all your traditionally boring requirements, such as you have business continuity, you have disaster recovery. Can I protect myself against ransomware? Can I support complicated test step workflows for performance, etc., without having to file IP tickets or without a person needing to get involved? How do I move data from production to test step?

All of those workloads still apply, and in some companies that we speak to, a lot of that has been pushed onto the shoulders of the developer, which again believe is a wrong thing to do. Sometimes is what a skills gap in the tooling gap. We haven't provided developers with tools to do a lot of these things, but it's also not a part of the core application focus, and I don't know if they should be responsible for all these business objectives style goals. That's, again, very common by saying, "Let's approach things from an application layer," which balances the needs of operators and developers. We call ourselves being ops-focused, but dev friendly.

From the ops point of view, we want to give them the global compliance, policy-based management that's highly dynamic. We want to give them visibility to what's happening in particular with state. With stateless stuff, it's very easy. As soon as data enters the mix, you want to figure out that, "Hey, is my state protected? Can I reuse it in different ways?" Whether it'd be for analysis later or whether it'd be for test dev or performance testing.

But from the developer's side of things, as a mentioned, we want to give them control over the stack. We don't want to slow them down. So those are the two requirements that we balance in our platform today, and what we do is enable use cases, such as cloud migration as an example. We enable use cases such as moving data in an automated within production test dev, backup and recovery, disaster recovery, so all of those things fall out of our platform.

[SPONSOR MESSAGE]

[0:33:29.2] JM: Your company needs to build a new app, but you don't have the spare engineering resources. There are some technical people in your company who have time to build apps, but they're not engineers. They don't know JavaScript or iOS or android, that's where OutSystems comes in. OutSystems is a platform for building low code apps. As an enterprise grows, it needs more and more apps to support different types of customers and internal employee use cases.

Do you need to build an app for inventory management? Does your bank need a simple mobile app for mobile banking transactions? Do you need an app for visualizing your customer data? OutSystems has everything that you need to build, release and update your apps without needing an expert engineer. If you are an engineer, you will be massively productive with OutSystems.

Find out how to get started with low code apps today at outsystems.com/sedaily. There are videos showing how to use the OutSystems development platform and testimonials from enterprises like FICO, Mercedes Benz and Safeway.

I love to see new people exposed to software engineering. That's exactly what OutSystems does. OutSystems enables you to quickly build web and mobile applications whether you are an engineer or not.

Check out how to build low code apps by going to outsystems.com/sedaily. Thank you to OutSystems for being a new sponsor of Software Engineering Daily, and you're building something that's really cool and very much needed in the world. Thank you, OutSystems.

[INTERVIEW CONTINUED]

[0:35:22.4] JM: In order to do any of that stuff — You know what? Let's just take a top-down approach. So plenty of people want the feeling that they can lift and shift their Kubernetes cluster from one cloud to another. This is one of the big levels of excitement around Kubernetes, is that what Kubernetes, the platform is, is it's something that allows you to lift and shift your entire architecture from one cloud provider to another and this puts downward pricing pressure on the cloud providers. It puts pressure to not create lock-in to the cloud providers, and this is a

beautiful thing for developers, but I've not actually heard of many people doing a lift and shift yet, and I think that's probably because people are kind of terrified of doing that and it's not like a one click sort of thing.

So explain what is required. Like we've done enough shows about people migrating to Kubernetes, like maybe you're on virtualized instances and you're migrating them to Kubernetes. We've covered that before, so let's say you're already on Kubernetes on some cloud provider and you want to lift and shift it. What do you do or what is needed? What are the requirements for that lifting and shifting?

[0:36:33.8] NT: It's a great point that you raised. This is something our customers have expressed a lot of love interest over the last few months. We do see a lot of cloud migration requirements. The majority of people wanting to use them to-date and to be across different regions in the same cloud, such as from US East to US West as an example, but we are seeing an increase ask, and this is another reason why people both pick Kubernetes, that is, as you mentioned, for increased portability, but also why they want to reduce dependence on cloud provider-specific obstructions.

So if you're using AWS's DynamoBA, you're pretty much locked into the platform as an example. Bringing things into Kubernetes itself gives you a lot more of that portability. Now, what you need to do to really move things across clouds is a couple of different things. Some of which Kubernetes has already made it easier for you, that is to capture the state of your application and infrastructure independent manner. We already have a lot of that via lie container definitions and application definition.

There are some minor things that need to be swapped across, but a little bit of preplanning can take care of that, such as definitions of things like storage classes. The more stateless stuff again, much easier. As soon as state enters the mix, it becomes a little bit more complicated. And so we do it in a couple of different ways. We have a lot of features built in for both data mobility and manipulation. So why an open-source project that we have called Kanister. You can check it out at kanister.io, and that's Kanister with the K, and to that we allow you to extract data from applications that are not volume level snapshots, but extract data from an application point of view.

So, for example, let's pick Mongo. We don't take — We have a three replica Mongo cluster. We will extract data at the Mongo API level and put it to a platform-independent artifact stored in an object store S3, and this makes it much easier to port over to another cloud provider compared to using volume snapshots, which are not easily migratable.

There are stuff that we do for that that very easily allows you to move across different cloud providers, and sometimes there still is 10 gap between data as to how load balancers are ingress work across different cloud providers, but those are easier to solve. A lot of the complexities in the application itself, which is what we capture for our users. Does it make sense?

[0:38:58.5] JM: You're a describing a Mongo cluster. If I've got a three replica Mongo cluster on AWS or some other cloud provider. Just for distributed systems people, you keep three replicas because you want to have both replication, because if one of those nodes gets vaporized and you only had one database node, then you would lose all of your customer data, and you don't want to have two, just because if you had two, then if there was a disagreement between the two instances, then you wouldn't know how to resolve that disagreement. So you have three, so that you can resolve differences and have these replicated states. You've got a three node Mongo cluster on a cloud provider and you want to lift and shift it to another cloud provider, and you're talking about the difficulty of doing that lift and shift.

I guess what I didn't quite understand what you're doing to snapshot that data. Like if I was going to naïvely snapshot that data, I would like export it to a CSV file or whatever and then just like email it or put it in Dropbox or something and then like go to the other cloud provider and upload that CSV. What's the problem with that? Why would I do something different?

[0:40:11.0] NT:A number of different things. So A, you never want to do anything manual. This has to be automated, because we talked about a single instance. We have customers which have hundreds of instances of Mongo running around, and how did they do this at scale is a challenge for them. So we build in a lot of that automation. We do a lot of policy-based stuff. So you can simply say, "Every Mongo application in my environment, I want to do this." But no

matter what application it belongs to, no matter what namespace it belongs, go the protector that, but it can be even deeper than that, “Do this at the application layer.”

So one of the demos we show is with GitLab as an example, that is you deploy GitLab, it uses PostgreS, Redis, a few other things in it. Protect just the entire GitLab application, all the stateless stuff, all the stateful stuff, so that you can clone it, migrate it to another cluster, recover from it. It really doesn't matter.

So what we do is we hide a lot of the complexity, make it easier for people to scale to do at — It doesn't matter how many instances you have of things running around, which might be separate, and we give you the ability to define policies on these things. All of these things happen on a regular cadence. You’re not worrying about, “Do I need to do something today, yesterday? Did it happen? Did it succeed? Will a restore actually work?” All of that comes in from the platform itself, versus doing it manually.

The other things revolve around things like efficiency and complexity. That is if you say, for example, a moving volumes snapshot across different cloud providers. Everyone does it differently. Some people involve copies. Some people don't. We take care of a lot of that. From the user, developer, operator perspective, they don’t see any difference in how they’re managing the applications across different cloud environments, even if they’re on-prem or in the public cloud. So that complexity is, again, taken away from them, allowing them to concentrate on what really matters. That is the business value they’re providing.

[0:42:02.9] JM: Okay. So snapshotting a —Let’s, I guess, keep it simple for now, a three replica Mongo cluster in order to lift and shift it. I took a distributed systems class in college. It was really hard, and one of the things that we learned was the snapshot algorithm. Snapshot is not simple. So can you explain what it's like to write a snapshot system in production?

[0:42:26.9] NT: Sure. As you talked about, it’s eventually consistent system. In particular, it's really hard. Snapshotting is not for the faint of the heart, because there is a law, when you want to gather consistent state of the world, it's difficult, because it’s data stored in multiple places. That could be data in-memory that still hasn't hit your persistent disc as an example. There’s data in disc, but it might be in a log and not applied to your database. Discs themselves have

caches. It's calling about the right algorithms, and then being able to say at a consistent point in time, “This is where I want it to be.” I guess more complicated when you have multiple discs even on the same node, but even further complicated when you have an application spread across nodes, that it's hard to coordinate between, so doing a distributed snapshot as an example.

What we do is we allow people to use a variety of different tools at multiple different layers. If I say, for example, you do not have application awareness. We will do things at the volume levels. Take a snapshot of that, and we have hooks if you want to go quiesce the application before you do that. So that enables you to get a consistent point of view of everything being stable on disc. But the other thing that we do, again, with the same kind of applications, is we use the application level API as a Mongo API or within PostgreS by using the logs. Do extract data from the application itself, and these can use incrementals again. So you are making a full copy of the database, but using application level APIs to extract data which you know will be consistent, versus doing it at the slightly dumber level, which is the volume or the storage API level. So we give a lot of flexibility to users as to how they want to perform the snapshot of the system and what later they want to do with that.

[0:44:14.4] JM: Okay. I don’t know if you can dig into a little more detail, but just to give people some distributed systems know how. Why is it hard to do a snapshot and how do you actually implement that? Can you talk about that at a little bit of a lower level? Because like I've — For example, I just remember reading like these proofs of like, “Here's a proof that a snapshot actually takes —” If you decide, “Okay. We’re at T-0, and we’re going to take a snapshot at T00, and we need to multicast across the entire system that we’re going to take a snapshot at T-0, and then actually getting that state that was consistent across the world at T-0. Can you just explain that in more detail?

[0:44:56.8] NT: Sure. I think the two different concepts that one talks about is the concept of transactions, and a lot of these systems whether they’d be traditional relational systems as well as NoSQL systems, and then there's a concept of snapshots there. So both of those definitely apply, and we shouldn't confuse ourselves with — There are, again, two kinds of snapshot. There is what MySQL says might be a snapshot version of the data where SQL server says this, versus what the volume says at a snapshots. Sometimes we need to be careful about the terms

we use, but it's what you talked about, that there is a notion of, at some time, T-0. this is a state of the world.

Now, what happens is that when a transaction enters the system at T-0, it is possible for one node to say, “Look, I have committed this,” and then send the results out to other nodes for application. The question is, “What is the real state of the world when you capture data at that state? Does it mean that once a single node has committed that transaction, is a snapshot at that point in time what we call stable, or does it have to be acknowledged by every nodes? Different systems of different properties and [inaudible 0:46:03.6] as to how [inaudible 0:46:05.1] some of the stuff. Some people use something a lot more heavy rates, such as what they call a two phase commit.

But all those things to really make a difference as to what, how strong you want to consistency, and that's what makes it complicated, and they come with different performance impacts too. The heavier weight the process is to ensure consistency across different places in your system before you go capture that data, the slower operations will be, including run time stuff. That's why when you go look at MySQL as an example, the difference, what they call serializability levels, and the same thing applies at lower levels too.

For example, in the Mongo case, for really large databases, what people implement them is actually do a three phase process to take snapshots, because they say, “I have a 4 terabyte database in MongoDB, which is replicated in different places. I know I’m going to bring this cluster back up at some point in time, whether disaster happens or for performance testing, etc.”

What they do is, first, they take a volume level snapshot of all the three nodes and they know for a fact it will be inconsistent, because transactions are continuously flowing through the system, and that in some sense is okay, because what they do then is that they bring this cluster up in a sandbox environment as an example, and then they run repair tools to figure out exactly what you talked about. If we look at different points in type, say time T-0, was the state of the world consistent then? If not, go fix it up, and sometimes that might mean going back behind in time a little to reach a point where you know you were consistent. Then quiescent the system after you fixed everything up, and then capturing another snapshot of that, because at this point in time you know there's no data flow into the system. A lot of the complexity happens, because there’s

a lot of data flow into to the system while you're trying to — That's modifying state, while you're trying to capture a consistent point of view.

[0:48:01.7] JM: Indeed. Yeah, okay. So TLDR, distributed systems are hard. Snapshot is hard. Go read the proofs and whitepapers and stuff if you're curious more about that. Now let's assume we have snapshot in a box for our distributed systems. So what do people want to do with snapshot in a box? I mean you obviously mentioned lift and shift. You can take a snapshot of your database, your giant database cluster and shift it to a different cloud provider, but I think you've also hinted at some instances where people just want to run snapshot on a regular basis across their entire infrastructure. So what are the different instances where people are going to want to use snapshot?

[0:48:39.8] NT: Let's talk about how a customer is using snapshots for non-lift and shift or non-backup recovery workloads. Those are definitely table stakes in these environments. I think some of the more interesting workflows one is enabling is that by taking the snapshot out of your backup system and saying, "How do we make them more useful to the end developer?"

In terms of how people are using it, we see a couple of very significant use cases and it's not just data snapshot. It's the entire application stack. But people — For example, we have a networking firm that is using or wants to use snapshots to bring the real-world data into a scale testing environment, and a lot of customer cases, synthetic data doesn't make sense, because your customers are never the same size, your data exhibits strange properties in the real-world, and so they had code that works well in synthetic staging environments that doesn't work well in production. They want to capture realistic data, run performance tests of new deployments to go to make against it..

The other thing that shows up often is both development and debugging. That is on an automated manner, bring up an entire set up from a developer to go use, whether they'd be replacing a section of code within that or debugging why something is failing in production. In that scenario, masking or data masking also shows up as a requirement, because it might be sensitive data. Everything from your typical Social Security numbers, but generally things like addresses, people's personally identifying information that you don't want to have in a not very well-locked out environment. So you mask data and then you move it into another cluster. That,

again, shows up very often for the customers we talk to. That is, how do we speed things up for the developer by allowing them fresh access to data on a very regular basis?

[0:50:29.5] JM: All right. I know we're running out of time. One of the things that I was so interested in at Cube-Con was walking around the Expo Hall and seeing all of these vendors who are building different businesses in the Kubernetes space, and this is just something I've learned when you go to these expo halls at different conferences, anybody who is been to a technology conference and has talked to the vendors. Oftentimes you'll get these vendors, different people have different perspectives on the way that the future is unfolding, and it's very interesting to see the sometimes disjoint perspectives on the way that the future is going to unfold, and then in a you stay in this space another year, a year and a half and you see the vendors who are right and the vendors who were wrong, and some of the ones who are wrong disappearance and some of the ones who are right grow into giant companies. I've seen that just doing this for basically two years. So that the timeframe can be very compressed and it's getting even more compressed, because it's easier and easier for people to adopt these technologies, which brings in ARR and makes these companies just grow tremendously fast.

Tell me a little bit about building a company in this space. What are the considerations that people need to make and what are the struggles that you're dealing with?

[0:51:49.6] NT: It's a great question. Company building in a new emerging ecosystem is always difficult. Let's put it this way; it is very easy to find hard problems to solve, but those are not the ones that always deliver value to customers.

I think the best piece of advice I can give people that are looking to do stuff in this space and I highly encourage folks to look into it, is that be very customer-focused. They will tell you what the real problems are, and sometimes it's surprising. They will tell you what they're comfortable with. They'll tell you what they're not comfortable with, and nothing beats that input. Really, what we have seen even in my previous lives in the industry is not listening to the advice of the customer is the biggest mistake one can make.

So then when we look at folks that have opinionated views on how to solve problems in this ecosystem. I think that is good, but as long as it is being driven from a customer perspective, for

us personally, the challenges we run into is to make sure we're making the right technical choices so that we are — It's an overused term, somewhat of a buzzword, but that we are truly cloud native. That is we fit into the world of the way the ecosystem is changing. There will be some vendors that say, "Look at this legacy product that works for VM's. I'm going to tweak a couple of things and make it applied containers too." That really doesn't scale in this environment, because we have this blurring of lines between infrastructure and applications as an example. We have this very dynamic quote, "compared to the world of old."

Explore how the world is fundamentally changed. Talking to customers I think sets people on the right path, and that generally is a much stronger predictor of success than anything else one might see from the external perspective.

[0:53:26.8] JM: Niraj Tolia, thank you for coming on Software Engineering Daily.

[0:53:29.4] NT: Thank you so much for having me. I really enjoyed this conversation, and hope everyone else does.

[0:53:34.0] JM: Likewise. It was great.

[END OF INTERVIEW]

[0:53:38.4] 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.

[END]