EPISODE 1079

[INTRODUCTION]

[00:00:00] JM: Kubernetes has become a highly usable platform for deploying and managing distributed systems. The user experience for Kubernetes is great, but it's still not as simple as a full-on serverless implementation. At least that has been a long held assumption. Why would you manage your own infrastructure even if it is Kubernetes? Why wouldn't you use autoscaling Lambda functions and other infrastructure as a service products?

Well, Matt Ward is a listener of the show and he's an engineer at Mux, which is the company that makes video streaming APIs, and Matt sent me an email that said that Mux has been having success with self-managed Kubernetes infrastructure, which they deliberately opted for over a serverless deployment. I wanted to know more about what shaped this decision to opt for self-managed infrastructure. I wanted to know about the costs and benefits that Mux has accrued as a result. Matt joins the show today to talk through his work at Mux and the architectural impact of opting for Kubernetes instead of a fully managed serverless infrastructure.

If you have an idea for a show, you can write about it on softwaredaily.com. We are always looking for new show ideas and you can also post about your company. You can post about the projects that you're working on. You can post jobs that are related to those companies or projects, and we're going to look for the best ideas and we want to cover things that are interesting as well as share the job postings that people post that are particularly useful to the audience. You can check all that out at softwaredaily.com. Thanks for listening.

[SPONSOR MESSAGE]

[00:01:53] JM: Today's episode is sponsored by Datadog, a modern, full-stack monitoring platform for cloud infrastructure, applications, logs and metrics all in one place. From their recent report on serverless adaption and trends, Datadog found half of their customer base using EC2 have now adapted AWS Lambda. They've examined real-world serverless usage by thousands of companies running millions of distinct serverless functions and found half of

Lambda invocations run for less than 800 milliseconds. You can easily monitor all your serverless functions in one place and generate serverless metrics straight from Datadog. Check it out yourself by signing up for a free 14-day trial and get a free t-shirt at softwareengineeringdaily.com/datadogtshirt. That's softwarewareengineeringdaily.com/datadogtshirt for your free T-shirt and 14-day trial from Datadog. Go to softwareengineeringdaily.com/datadogtshirt.

Thank you Datadog.

[INTERVIEW CONTINUED]

[00:03:06] JM: Matt, thanks for coming on the show.

[00:03:08] MW: Thanks for having me, Jeff.

[00:03:09] JM: You work at Mux, and Mux is a company that makes video APIs for developers. Explain what Mux does.

[00:03:16] MW: Yeah. Mux was founded because we believed that video is complex and really hard for a lot of people to get started with. There's kind of a really high barrier to entry. You have to understand what video codex and containers are. Understand what players can support playing back those different codex and containers, and we believed that basically your choices were your hosted options like YouTube, where I used to work, or kind of you had to roll it on your own. There was no simple way to get started with online video without really digging into a lot of technical details. We wanted to create a high-level API that made it easier for developers to get started with and build applications around video.

[00:04:12] JM: Describe how developers interface with the Mux products.

[00:04:17] MW: Yeah. At a very high level, developers have a couple options for getting videos into our systems. You can give us a URL and we'll go pull any recorded video or even audio file you have on your servers or on S3 bucket or Google Cloud Storage, wherever it is as long as

it's accessible via HTTP. We can pull that down, process it and ensure that it's streamable to any player that is capable of HLS playback.

[00:04:52] JM: Mux is a pretty complex and useful service and there's a lot of services that we'll explore and we'll explore your deployment options and how you came across the decision to emphasize the use of Kubernetes instead of serverless stuff, which the conversation is going to focus on. Before we get there, I want to talk a little bit more about your background and some of the issues at Mux. Tell me about some of the canonical engineering problems that you encounter in Mux.

[00:05:22] MW: I think the first kind of challenge that we encounter that kind of all developers encounter is API design, especially in kind of developer-focused applications. One of the most important things for us to do well that we think differentiates our products from other products is how easy the API is to work with. Then because we're also a Go GRPC microservices platform internally, we have the same API design all the way down through our stack.

[00:05:59] JM: Can you tell me more about the storage and compute requirements?

[00:06:02] MW: Yeah. Kind of the high-level simple API design stuff is just the one aspect. The reason that our kind of product becomes so challenging to engineer if you were to say roll it on your own, is there's a lot of computation that needs to be done in order to transcode or repackage a video. That management layer as well of what transcodes are running, what requests are coming in for videos that are either already transcoded or actually need to be transcoded adds a bunch of complexity, and we actually don't store your video fully transcoded. If no one watches it, our videos aren't actually transcoded and sitting on disk. We wait for someone to actually start watching it before we started transcoding it. So that gives us a number of benefits on the storage side and we kind of handle that with complexity in our applications at request time.

[00:07:03] JM: The bitrate ladder calculation I think is relevant to point out here. Just as an example of something that's going to take some load on your server, so like somebody uploads a video, like I've got Jeff's cat video. I'm going to upload to Jeff's awesome blog, and Jeff's cat

video needs to be transcoded into multiple different bitrates, because lower-end devices or mobile networks, maybe they need to consume lower bit rate video.

This is just an example of like if somebody uploads a video, there's a bunch of stuff to do with it on the compute side. You have these different compute services that you're going to need. Then you're going to need to store the different videos. Give us some perspective on the infrastructure problems. How did developers work with each other at Mux? Is there a platform engineering team? How are services deployed and what's the set up for different teams that want to spin up services?

[00:07:56] MW: Yeah. I think we've been iterating on our team structure a good bit over the past couple years, and at this point, we've more or less split up into a group focused on our video product, a group focused on our data product and another group kind of focused on the shared tooling between the two and kind of we're acting more as consultants in the software reliability engineering realm as well. Then there is our product team where we have a head of product and a product manager for each of our products and we have one designer that jumps between the two as he can.

[00:08:38] JM: Okay. We've covered kind of what engineering looks like at Mux. Now let's talk about how you've architected this system. You connected with me over email and we were talking about serverless versus Kubernetes, and this is a decision that a lot of companies have to make, like do you want to roll your own infrastructure in the sense of Kubernetes these days, which is much easier than rolling your own infrastructure in the past, or do you want to defer everything to serverless and use managed services, managed databases, managed machine learning systems, AWS Lambda functions? Tell me about the pros and cons of building on a serverless platform.

[00:09:18] MW: Absolutely. I think one of the biggest pros for going serverless is you have to worry a good bit less about capacity planning and ensuring that you have enough compute resources to handle your incoming requests. A bunch of platforms such as AWS Lambda give you some knobs here, but most of the time they're kind of handling kind of the autoscaling under the hood for you and you don't have to really worry about what size instance do I need. You kind of just almost don't think about capacity.

As you're kind of growing, you don't have to, again, worry like, "Oh! Were the decisions and commits that I made last month necessarily the same decisions that I want to make again this month or this year?"

Auto scaling is kind of one of those things that is actually working really well in Kubernetes now as well. You can kind of tie in to Kubernetes APIs and run the horizontal pod auto scaler and your cluster auto scaler and get a lot of those similar features out of your Kubernetes cluster that previously your serverless APIs would just kind of handle for you.

There's a good bit more configuration that's required if you're going to be running on Kubernetes and you have to obviously learn a good bit more I think when you are running Kubernetes clusters and your application on top of Kubernetes. But generally, I think there's kind of a good bit of convenience provided to you if you're sticking with serverless.

In contrast though, you're tied to kind of the specific pricing models of whatever serverless platform you are working on. So you have a little bit less freedom when it comes to buying compute and storage because of kind of the platform you're tied to. You're not able to kind of evaluate as many options when you're trying to figure out where and how do I run my application.

[00:11:27] JM: What kinds of scalability problems can you encounter with serverless products, or do those exist? Do they just scale to infinity?

[00:11:36] MW: I have heard mixed things on this. Again, at Mux, we don't actually run anything serverless. I've heard some things around Lambda and there are some certain levels that you hit. I think it's like 3,000 concurrent Lambda invocations where after that you need to start purchasing additional compute from Amazon more explicitly. But I'm not super familiar with the details of once you get up there with scaling a Lambda application, because we don't do that at Mux.

[00:12:07] JM: If you did go all-in on serverless at Mux, what would you be using?

[00:12:13] MW: I think if you went all-in on serverless, I would probably be looking toward something like your Lambda APIs. There's also Knative, which is starting to become an accepted platform for running serverless style applications on Kubernetes. I'd probably be looking at one of those two.

For our compute, we are running transcoding software. So we'd be eyeing how long can those processes run for? How much RAM can they actually use because they have to pull a lot of video in-memory in order to actually do the transcoding processes and how many parallel cores can we take advantage of in order to actually run those transcoding processes? I am not sure that most serverless providers will give us enough resources, and storage is kind of the last thing there. You have to figure out how you're interacting with a disk in order to – Where you're writing out that data.

[00:13:14] JM: Just a little bit more on serverless products. Are there monitoring issues with serverless products?

[00:13:21] MW: Yes. I have only looked at Amazon's Lambda most closely here, and the most interesting challenge I see around monitoring with Lambda is you hit a certain point where you can say, "I will get metrics every 30 seconds or one minute." I forgot exactly what the minimum there is, but different platform providers will give you different intervals at which they will scrape your application for metrics or your application can push out metrics and their monitoring systems aggregate those metrics. there's kind of a limitation to the options provided there that kind of pushed us more towards the Prometheus and kind of having a metrics endpoint on our applications and being able to really control the frequency at which we scrape different applications. That gives us a better understanding in closer to real-time power applications are performing. If we don't feel that we're getting enough real-time feedback, we have the ability to kind of tune down the scrape intervals on our metrics endpoints so that we can see more quickly whether the actions that we've taken are helping to resolve an ongoing incident, or issue, or if maybe you were making things worse by accident.

[00:14:49] JM: Okay. So rather than talking about fiction, let's talk about reality. You are running Kubernetes. Give me an overview of the actual infrastructure deployment that you have at Mux.

[00:15:01] MW: Yeah. We run across two clouds today. We run on Amazon and Google's cloud. For legacy reasons, most of our data infrastructure is currently on Amazon and most of our video infrastructure today is on Google and we are currently working to kind of expand the number of clouds we're running our video infrastructure on.

Our frontend API servers actually today all run in Amazon and they are kind of the frontend to our two kind of backend services. For the data product, that's mostly databases in AWS that we run and host inside of our Kubernetes clusters. Then we talk over HTP over to our clusters in Google cloud, which is where we have a much more complex infrastructure around our video product. There're a number of workers. We're doing transcoding. There's kind of this scheduler that orchestrates all of that and kind of access the broker between incoming requests for video chunks or segments, and kind of these workers which will actively transcode them.

There's obviously the traditional kind of database applications there handling our internal management of what assets do we have on file and what customers do we have. There's a billing, an accounting system there as well so that we can know how many minutes of video we have actually received from a given customer and how many minutes of video we've delivered, and that kind of ends up being a separate system running on top of incoming CDN log data.

On the delivery side, we don't just deliver right from our Google Cloud regions that we run in. We actually partner with a couple CDN's, and those CDN's help us get kind of a better global delivery footprints so we can get those videos closer to the end viewers.

[00:17:14] JM: All right, great. Are you using managed Kubernetes service or are you deploying your own Kubernetes entirely on something like raw EC2 instances?

[00:17:24] MW: Yeah. We're running our own Kubernetes right now built on top of Kops. We've actually forked Kops for a couple reasons. The first one was that when we were using Kops in GCE, we wanted to use some local SSD's and a couple other features that didn't exist in Kops. We had to make some modifications there. But generally we have rolled our own Kubernetes in order to unify the developer experience across our two clouds. When our developers go and look for monitoring, they know the exact host names to go to to pull up Grafana or Prometheus and rerun kind of the Elasticsearch stack for logging so they know how to get to Kibana and use

that and whether or not their application is running in Amazon or Google. No matter what, they have the same kind a set of tools to debug and actually run their applications.

[SPONSOR MESSAGE]

[00:18:38] JM: When I'm building a new product, G2i is the company that I call on to help me find a developer who can build the first version of my product. G2i is a hiring platform run by engineers that matches you with React, React Native, GraphQL and mobile engineers who you can trust. Whether you are a new company building your first product, like me, or an established company that wants additional engineering help, G2i has the talent that you need to accomplish your goals.

Go to softwareengineeringdaily.com/g2i to learn more about what G2i has to offer. We've also done several shows with the people who run G2i, Gabe Greenberg, and the rest of his team. These are engineers who know about the React ecosystem, about the mobile ecosystem, about GraphQL, React Native. They know their stuff and they run a great organization.

In my personal experience, G2i has linked me up with experienced engineers that can fit my budget, and the G2i staff are friendly and easy to work with. They know how product development works. They can help you find the perfect engineer for your stack, and you can go to softwareengineeringdaily.com/g2i to learn more about G2i.

Thank you to G2i for being a great supporter of Software Engineering Daily both as listeners and also as people who have contributed code that have helped me out in my projects. So if you want to get some additional help for your engineering projects, go to softwareengineeringdaily.com/g2i.

[INTERVIEW CONTINUED]

[00:20:26] JM: Okay. How many Kubernetes clusters do you have?

[00:20:31] MW: Yeah. We run each cluster in a given region. Our clusters are high-availability and that they span multiple availability zones. That ensures that if a single availability zone has

issues, that whole cluster should keep operating without any problems. We kind of run bigger Kubernetes clusters. In Google, we run in two regions. That's two clusters there, and then we have kind of a staging and tests cluster. Then in Amazon, we've got kind of a similar set up, but we're running in one region there and kind of, again, we don't have a staging and test set up. I think that totals about five and then we've got the random tinkering going on all over the place. There're probably more than 5, but 5 that are actively maintained.

[00:21:26] JM: Now, if you're rolling your own everything, then that means you got to run your own Redis and databases and Elasticsearch and whatnot, and these things all require servers. Kubernetes is your server infrastructure. That means you are managing servers that are underlying these databases and data services. I believe that the sophisticated way to do this is with a Kubernetes operator in many cases. Can you explain what a Kubernetes operator is?

[00:21:55] MW: Yeah. Kubernetes operators operate on top of what are called custom resource definitions. What you're able to do is define higher-level constructs in the Kubernetes API that allow you to express things, like I want a Redis cluster with three Redis Sentinel nodes. So you don't have to worry exactly about the details of how those Redis master and slave setups are handled. The operator ends up constructing the actual stateful sets or deployments or whatever the operator decides is the appropriate Kubernetes objects to create given your inputs in the custom resource definition. That makes it so that as developers, we don't have to think about every environment variable, every volume, or every kind of networking interface between each of these individual instances of the applications we're running. Instead, we can just say, "I want a Redis," and it is able to help orchestrate the creation of that application.

This kind of makes it a lot easier for developers to get going. It almost brings it up to the APIs that you might see from something like dedicated service providers where you kind of would place a request and say, "I want a Postgres database," and that Postgres database you might specify, "I need a two-core database with two gigs of RAM or four gigs of RAM." Whatever you think is appropriate for your application.

A lot of these operators have that level of control that makes it feel very much like what is your serverless style RDS or other hosted database product that you might use. But more and more people are creating operators that mimic a lot of that behavior down even to backups. You can

define another custom resource definition, let's say, for your Postgres database that says take backups every two hours or every 12 hours, whatever you think is appropriate for your application, and the operator will say create a Kubernetes cron job for you.

A lot of the community is really coming together to standardize on implementations for a lot of these applications that normally you would kind of reach out and say, “Oh! I don't want to manage this database, or I don't want to manage figuring out how to organize a Redis cluster. I don't want to read the documentation.” Operators are starting to get to the point where they provide a higher level construct that lets us work more quickly and easily.

[00:24:50] JM: What's required to deploy one of these systems with an operator?

[00:24:54] MW: Yeah. The only thing that tends to be required is Kubernetes and the controller. Operators join into your Kubernetes cluster via a custom controller. If you are – We’ll use the Redis example that we’re talking about earlier. If you're deploying Redis and you want to use a certain operator, you deploy the controller for the Redis operator, and that controller installs a custom resource definition, which is basically an API spec that extends the Kubernetes API, and that controller then reacts to the creation of these custom resource objects in the Kubernetes API and it's able to then go and create the appropriate other objects.

In the case of this Redis example, it would know, “Okay, I see you want a Redis. You've asked for a three node Redis cluster. I’ll do that as one master and two slaves and it will bring up the master first and then the kind of two slaves and set up replication appropriately and ensure that that cluster stays healthy.” The cluster continually reflects the spec that you have provided in the Kubernetes API. If all of a sudden you change your mind and say, “Oh! I actually want a five-node cluster.” The operator then that’s running sees this object has changed and it's able to do what's called a reconciliation, and it goes and says, “Oh, well. Right now we’re running three Redis instances in the cluster. You've asked for five. Let me go and add an additional two and configure those correctly and ensure that they stay healthy in the cluster as well.”

[00:26:45] JM: Okay. The deployment of an operator-based system, like if you wanted to deploy a Redis, it sounds like operating a Redis cluster with operators on Kubernetes is not considerably more difficult than deploying ElastiCache on AWS, something like that.

[00:27:08] MW: Yeah, I think that that's totally true and that's kind of what we're really seeing with this whole Kub native approach that people are talking more and more about, which is less and less do we have to worry about the details of how to actually spin up applications if the kind of application creators are providing these operator controllers that are able to help set up these applications. I think more and more what we're going to see is these applications won't just set up. They'll continually actively ensure the healthy operation of those applications as well.

They might have things, in the case of like an Elasticsearch, if you're able to codify in your custom resource definition that this is a logging cluster and I have logs written every day, you might be able to freeze older indices automatically in order to optimize the performance of that Elasticsearch cluster. A lot of these kind of more complex operations that traditionally we as application human operators have been taking on will move into these more complex pieces of software and they will almost emulate your entire software reliability engineering team and a lot of the kind of reactive things that they do.

[00:28:37] JM: Tell me a little bit more about what cloud provider systems you are actually using if you're not using managed services like ElastiCache.

[00:28:47] MW: Yeah. I think there's a certain set of products that cloud providers offer that you really need to use. Otherwise, you have kind of more downside than upside. A perfect example of this would be Google Cloud Storage is something we used to actually send our completed trance codes off of the in-cluster disk out to kind of longer-term storage. We don't exactly want to manage our own storage cluster because the cost of doing that in Google compute engine and keeping hot disks online there is so much higher than actually writing out to Google Cloud Storage. So we choose to use Google Cloud Storage as kind of our storage backend for a lot of our application because of that kind of cost difference.

In addition, on the Amazon side, we actually still do use some RDS. We have had some problems running our Postgres in a really, really reliable fashion early on. Some of our older databases are still running in RDS, because a year and a half ago, a lot of the operators were not as mature as they are now.

[00:30:08] JM: What would be easier to do across your infrastructure if you were using serverless infrastructure? Is there anything, any stress or pain points that serverless infrastructure would relieve if you were using more of it?

[00:30:20] MW: Yeah. I think one thing that comes to mind here is, early on, when we were starting to write our applications, we had to kind of write our application platform. I don't think that being Kub native means that you don't need an application platform. For us, we chose to use GRPC. I am an ex-Googler, so I like my [inaudible 00:30:47]. That system was pretty familiar to me and I was able to quickly slap together some software that got our team building pretty quickly. I think that if you don't have someone that's really willing to pay down or own that application platform, you definitely get a lot of benefits by starting with something like serverless. If you have an application that doesn't have very demanding latency and compute requirements or if you don't expect to have those in the future, I think that also is a point where it makes a lot of sense to consider writing your application serverless especially if you're a small team.

[00:31:35] JM: Are there any other downsides to this self-managed infrastructure approach, the Kub native approach that you've been experiencing?

[00:31:44] MW: I think the biggest one that is an ongoing challenge for us is figuring out how many pods to repack on a machine. How do we pick the right machine sizes? A lot of these kind of decisions require an active kind of participation in your cluster management. Your actual pods, you can't say I want 16 cores if your maximum instance size is only 8 cores. There is still some of that kind of management that I'd say takes a nonzero amount of time.

[00:32:26] JM: What about people on your team? Are there people on the team who have trouble operating the Kub native approach, perhaps, who are more accustomed to working out of the AWS console?

[00:32:36] MW: I think the biggest challenge for our team has been the development environments and there is some nice things around the consistency that you get that I think Docker really brought to our community a few years ago. You have a really consistent build and bills and you know that your application is going to run same on your laptop versus in production.

With Kubernetes, we have a similar challenge, which is if you're writing YAML for, say, a deployment that require storage and you're trying to develop that on your laptop, how do you know that deployment is going to deploy in the same way on AWS or on Google. Where are Kubernetes platform has different storage constructs than you might have available to you on the Kubernetes instance that's running in your development environment on your laptop?

I think there's still a little bit of inconsistency there across our development environment to production that slowly the community is starting to have ways to work around that or make it a little easier. The other challenge is when you're actually building and running your applications, where does the compiling happen? Where does the running happen? A lot of times, you're running these Kubernetes local clusters in like a Docker and Docker style setup or maybe you're just using Docker as Kubernetes, and those environments might not necessarily be where the code that you're editing is. How are you actually building and deploying your application in your development environment to make that iteration process as quick as possible has been a good bit of a challenge for our team.

[00:34:32] JM: How do the costs of running your own infrastructure like this compare to if you were all-in on serverless?

[00:34:39] MW: I think that, for us, because we're dealing with video, and video necessitates a lot of CPU, RAM, and cold storage or like even hot storage as well. We tried to think about those as like raw resources. When we go do talk to our cloud providers, and right now we're actively renewing our contracts, we're trying to figure out, "Okay, if we place a three-year commit here or a three-year commit there, who's actually going to be able to offer us those CPUs at kind of the best price so that we can pass forward our infrastructure cost reductions to kind of our customers? We hope to kind of offer a continually cheaper product to our customers. But the only way we can do that is if we can work with our cloud providers or even potentially physical bare metal providers to get to the point where our cost per CPU or RAM is as cheap as possible. That's only really possible for us to even negotiate because we took this kind of b native approach. If we can get a Kubernetes cluster running on the hardware, we can get our application running there.

[SPONSOR MESSAGE]

[00:36:10] JM: Over the last few months, I've started hearing about Retool. Every business needs internal tools, but if we're being honest, I don't know of many engineers who really enjoy building internal tools. It can be hard to get engineering resources to build back-office applications and it's definitely hard to get engineers excited about maintaining those back-office applications. Companies like a Doordash, and Brex, and Amazon use Retool to build custom internal tools faster.

The idea is that internal tools mostly look the same. They're made out of tables, and dropdowns, and buttons, and text inputs. Retool gives you a drag-and-drop interface so engineers can build these internal UIs in hours, not days, and they can spend more time building features that customers will see. Retool connects to any database and API. For example, if you are pulling data from Postgres, you just write a SQL query. You drag a table on to the canvas.

If you want to try out Retool, you can go to retool.com/sedaily. That's R-E-T-O-O-L.com/sedaily, and you can even host Retool on-premise if you want to keep it ultra-secure. I've heard a lot of good things about Retool from engineers who I respect. So check it out at retool.com/sedaily.

[INTERVIEW CONTINUED]

[00:37:46] JM: What about incident response? If you're managing your own Kubernetes infrastructure, you're going to need to handle your own outages. Is that ever an issue?

[00:37:55] MW: I think anyone that tells you that instances aren't an issues is lying. I think no matter how big or small you are, there's always going to be incidents. How you handle them and how you learn from them is really what separates really good teams from kind of your average teams. I think people that have really strong postmortem processes, really strong discussions around what failed and why and continually question the software decisions or even hardware decisions that you've made will lead to a stronger and better product and hopefully a more reliable one as well.

For us, we've seen failures at basically every level. Sometimes it's a little more frustrating than others because it's a little out of our control, but for like disk failures or instance failures, we hope that our applications can kind of migrate those workloads and we have replication in place to ensure that like a single machine failure, for instance, doesn't take down our whole application.

We have seen some kind of problems in our Kubernetes platform, but I'd say they're extremely rare compared to the number of developer kind of introduced issues. I think the old software reliability joke of like the most reliable software is the software that no one's working on holds true here.

[00:39:16] JM: Have there been any lessons learned from running your own Kub native infrastructure that you would advise or try to pass on to anybody that's listening that is thinking of deploying and running their own Kubernetes?

[00:39:31] MW: I think the number one thing I would pass on is there're, I guess, kind of some people that have called a myth of running multiple clouds are being able to kind of move your application wherever you like. I don't think that that's a myth at all. I think Kubernetes is really making it possible for us to treat our cloud providers or our bare metal providers more like compute storage kind of raw materials, if you will, in the software engineering sense. That's enabling us to really drive our own cost down.

If you know that you're going to have kind of a lower margin business, it makes more sense to invest in these platforms that can really enable you to have those negotiations and have those conversations where you're trying to figure out what is the cheapest way I can run this software while delivering a highly reliable product?

[00:40:32] JM: Are there any other guidelines for like what companies should or should not be managing their own Kubernetes? If there's a higher margin company, for example. If I am running some super high-margin business, a trading company, for example, or some SaaS service, like Figma maybe, should I be entirely serverless? Because if my margins are high, then I should just be operating and running as fast as possible.

[00:40:58] MW: Yeah, I think there's definitely a tradeoff kind of early on as well like I mentioned with the kind of application platform and like how big your engineering team is as well and how experienced your engineering team is with writing lower-level server code, handling shut down signals correctly, is something that we have to worry about. If you're writing an application that's serverless, you kind of just have to implement the one function.

If your team doesn't have a lot of experience with those lower-level constructs of software operations, I would probably steer away from early on investing in kind of Kubernetes native applications without kind of having that framework around you. I think the application framework is one of the most important things that a team kind of decides on early on, and that's almost as important as kind of how large or how small your margins are. It's kind of just what your team is comfortable with and knowing your own knowledge and where you want to invest, and then you can figure out, "Okay, is serverless the place where I'm going to get the most return on my investment or is growing our own application server the right thing to do right now? Because foresee challenges in that scaling of compute that we don't think we can get out of a serverless platform, which I think is closer to the situation we were in.

[00:42:25] JM: Have you had many other conversations with companies who have made the decision to go entirely with a self-rolled Kubernetes infrastructure?

[00:42:33] MW: I have talked to a couple, and I actually just started and advising role with a company that is the opposite. They are currently all serverless. I think a lot of companies that are using Kubernetes are working on kind of hosted platforms because you kind of don't have to worry a lot about that. Okay, what's the API server doing? Do I need to scale out kind of that component as well? You can at least reduce the amount of Kubernetes that you need to worry about if you're using hosted Kubernetes like in EKS or a GKE. That kind of reduces your worries a little bit, and I think most people that I've talked to are leaning down that road rather than kind of rolling completely their own Kubernetes.

[00:43:19] JM: Tell me about the other engineering problems that you're working on at Mux these days and how using Kubernetes native infrastructure is involved in the deployment and the engineering efforts.

[00:43:33] MW: Yeah. One of the problems we're working on right now is live streaming ingest. It's an interesting one, because this requires really long running servers that are handling really long-lived TCP connections. We found that actually our cloud load balancers aren't up for that task. If we are running a 12-hour live stream, that load balancer needs to hold on to that connection for 12 hours going to the backend that is currently transcoding that incoming live stream.

What happens when you're doing load balancing behind a load balancer is you have to respond with a health check that says, "I am healthy and able to handle more streams." Well, at a certain point, these why ingest servers can't handle more than, say, two or three live streams at a time. If they got another one, it would be overloaded and all of a sudden our transcoding would not keep up with real-time.

The health check, we have to fail, and we have to tell the load balancer stop sending me new traffic. What happens is the load balancer then might say, "Oh, that backend is unhealthy," and maybe it loses track of what TCP connections are actively being sent there. That might lead to connection drops and resets. That leads to your live stream dropping out, and no one likes it when their live stream drops out. So we've had to navigate a number of complexities there around reconnections initially, and now we're trying to figure out how we can actually have long-lived TCP connections with load-balancing that preserves the active connections and enables us to add more nodes into the pool.

[00:45:29] JM: You used to work at YouTube. How does Mux compare to YouTube?

[00:45:33] MW: I think the biggest difference between working at YouTube and working at Mux on video is the interactions that we have with our content delivery providers. At YouTube, we were lucky enough to have Google's infrastructure, and Google has their own CDN product. They have internal products for storage and compute that kind of clearly advertise their tradeoffs as best they could in a lot of places. But most importantly, they have the engineering teams right there that are willing to support you.

When you are starting a video company and you are buying CDN's, your kind of almost buying yourself and engineering team and buying yourself a support team. That's a very different

process and a very different interaction than we all work at the same company together. We all have the same end goals in mind. You kind of have a more transparent conversation, I would say, when it's inside your same company and you own everything versus I'm trying to decide whether or not I want to buy this thing from you or not.

In addition, when you're building features, a good example of this would be geo restriction or URL signing, is like what technologies do you support and how do you support the different features that you end up needing? I think it was really unique to be able to file a ticket and instantly get the person that wrote all of the code that you are reliant on.

When I have an issue with one of our CDNs, I have to file a ticket and it goes through a support tier and through your partner manager and whoever else gets involved with handling your ticket. Kind of the interaction is a little bit more masked, I would say, than when it's all kind of in-house. I think that's probably been the biggest difference.

I would say that's the same for kind of the compute and storage side with like our cloud providers and everything as well. I think you just kind of have a very different interaction when you own the whole entire system end-to-end.

[00:47:49] JM: Okay, Matt. Well, it's been really great talking to you, and appreciate you reaching out with your ideas around Kubernetes versus serverless. I admit, we probably have been – At least my opinion on the show has been overly leaning on the side of just go all-in on serverless whenever you can. It's nice to have a measured opinion of the downsides of the serverless or the upsides of managing your own Kubernetes however you see it.

[00:48:14] MW: Well, I hope I did it justice, but it will continue to be a war I think for a while.

[00:48:18] JM: Awesome. Thanks, Matt.

[00:48:20] MW: Thank you, Jeff.

[END OF INTERVIEW]

[00:48:30] JM: Software Engineering Daily has over 1,000 episodes with lots of interviews with engineers from Google, Facebook, Uber and lots of other engineering companies. We also have interviews with investors. We have interviews about the philosophy of technology and culture and strategy around starting a software business. You can find all of our episodes in the Software Engineering Daily app for iOS and Android. These apps have all of our episodes sorted and searchable and easy to find in categories with related links and commenting features. You can see our greatest hits, the most popular episodes that have stood the test of time.

If you don't want to hear advertisements, you can become a paid subscriber for $10 per month or $100 per year. Just go to softwareengineeringdaily.com/subscribe. We have put a ton of work into building the apps for Software Engineering Daily. We're creating the best listening experience for our users, and you can check it out today by downloading the Software Engineering Daily app from the iOS or Android app store, and I'd love to get any feedback you have on the apps or the show. You can always email me at jeff@softwareengineeringdaily.com.

[END]