EPISODE 765

[INTRODUCTION]

[00:00:00] JM: Infrastructure software is having a renaissance. Cloud providers offer a wide range of deployment tools, including virtual machines, managed Kubernetes clusters, standalone container instances and serverless functions. Kubernetes has standardized the container orchestration layer and created a thriving community. The Kubernetes community gives the cloud providers a neutral ground to collaborate on projects that benefit everyone.

The two forces of cloud providers and Kubernetes have led to massive improvements in software quality and development practices over the last few years. But one downside of the current ecosystem is that many more developers learn how to operate a Kubernetes cluster than perhaps is necessary. Serverless tools are at a higher level than Kubernetes and can improve developer productivity, but a risk of using a serverless tool is the potential for a lock-in and a lack of portability.

Knative is an open source serverless platform from Google built on top of Kubernetes. Ville Aikas is a senior staff engineer at Google who has worked at the company for 11 years. With his experience, Ville brings a rare perspective to the subject of Kubernetes and serverless and the infrastructure lessons of Google.

Ville joins the show to discuss Knative and the motivation for building it and the future of serverless infrastructure. It was a great privilege to talk to Ville because he's worked at Google for a longtime and brings so much experience to the conversation. I hope you enjoy this episode as well.

[SPONSOR MESSAGE]

[00:01:54] JM: Today's sponsor is Datadog, a cloud monitoring platform for dynamic infrastructure, distributed tracing and logging. The latest AWS X-Ray integration from Datadog allows you to visualize requests as they travel across your serverless architecture, and you can use Datadog's lambda layer to collect business critical metrics, like customer logins or purchases. To see how Datadog can help you get a handle on your applications, your infrastructure and your serverless functions, sign up for a free trial today and get a free t-shirt. Visit softwareengineerdaily.com/datadog to get started and get that free t-shirt. That's softwareengineeringdaily.com/datadog.

Thanks to Datadog for being a continued sponsor.

[INTERVIEW]

[00:02:51] JM: Ville, you are a senior staff engineer at Google. Welcome to Software Engineering Daily.

[00:02:56] VA: Thank you so much for having me.

[00:02:58] JM: Today I'm excited to talk to you about serverless workloads and the various ways that serverless can be engineered. Let's start off by defining that term. What is a serverless workload?

[00:03:12] VA: Sure. That's a great question, because there's a lot of buzz around serverless and there's also just this much confusion on what it means, because it can mean various things for different people.

For Knative, we have really started defining it as a developer-phasing concept, which means that you as a developer do not have to worry about the underlying infrastructure. You can just focus on writing your software and solving business problems than worrying about dealing with the servers, what the servers look like, how do you upgrade them and things like that.

[00:03:46] JM: Why is it useful to omit the operational burden of operating a server?

[00:03:56] VA: That's another great question. So while some people really do enjoy playing with the underlying infrastructure. For the most part, developers have told us fairly loud and clear that there's a lot of burden with it that comes with it. So while it's fun and easy to go and get started with the bringing up a VM or if you want to go super hardcore and plug it in the rack and wire it together, setting up one is fine and easy. It's tricky to go ahead and keep it up and running and make sure that it's always up-to-date.

What happens if you need to go ahead and all of a sudden your application that you write becomes super successful and you need to get hundred machines and now you need all kinds of tools and automation to go ahead and keep those up and running and make sure that they run smoothly, and when things happen, somebody goes in and fixes things. So there's a very clear division of labor and rules and responsibilities. So by drawing the line by saying that for the developers, "You know what? If we can give you a nice illusion that you have one big computer." That has been received quite well.

[00:05:00] JM: There has been a market shift towards usage of Kubernetes and towards usage of containers. These are still server abstractions, and there's already fatigue of people wanting to move their workloads to Kubernetes. So somebody listening to this might be thinking like, "Oh, gosh! I'm already implementing my Kubernetes strategy at the company that I work at. I'm in the middle of doing this, and now you're telling me that I should just not use servers and maybe I should just be using these serverless systems?" So how would you comfort that kind of person?

[00:05:38] VA: No. Yeah, that's fantastic. So one of the things that we are doing with the Knative – So I actually was involved with the Kubernetes very early on myself. So one of the things that Kubernetes in my opinion did quite well is it provided what I think of as the assembly language for the cloud native computing. So it gave you really good, clean, very low-level primitives for being able to go ahead and get started with the cloud native computing.

So when you go and put all those things together, things work very nicely, and we have seen many, many, many fantastic solutions running on Kubernetes and best practices and everything else. With the Knative, we are saying, "Great. Let's go ahead and build something on top of those primitives that makes it more approachable for the developer."

So I sometimes talk — that's the standard Lib, or Lib C on top of the Knative. So that's where we want to go ahead and codify and abstract the way the best practices that various people in the community have learned about how to run Kubernetes and make those higher level constructs that are more easier to deal with for the application developer, because those abstractions are quite idiomatic, if you will, to what are their main concerns that they care about when they want to get their software up and running on the Kubernetes.

So everything that you have done on the Kubernetes is still applicable. It's still available to you and you can go ahead and just start reducing some of the objects, if you will, in the Kubernetes that you don't necessarily have to care about. All the expertise that you have built on the Kubernetes still applies. So no worries, this is just a more goodness on top of Kubernetes.

[00:07:24] JM: We're going to unpack the abstraction layer that Knative is sitting at, because it's subtle and people might not understand it if we don't explain it in the right order. So I want to start by talking about functions as a service. So these functions as a service systems exist in cloud providers. You have Google Cloud Functions, AWS Lambda. Explain what happens under the covers on the actual servers when I deploy a serverless function.

[00:07:58] VA: Sure. So let's go ahead and take a – Let's start it from the deployment side of things, where I have some software sitting on my laptop, for example, or on my Pixelbook and I want to go ahead and get it up and running into the cloud. Happily I go in my Emacs and I write up my function and at that point I want to go ahead and say, "Please turn this raw source code into an executable code." So that's sort of kind of the first step.

At that level, that is a clear action that you as a user have to do, so there is something in the Knative called Knative build. So this is the responsibility of the build stack, if you will, is to go and take source code, whether it comes from your laptop, GitHub, Google Cloud Storage of wherever your source code happens to be and it takes those source codes and it runs the compilation and the packaging for you, whether it's in the cloud, or whether it's using a cloud service, or whether it's in your own cluster. The first step is getting into an executable container, so Docker image. Now that defines an immutable execution unit for your function. So that's the first step.

[00:09:15] JM: Okay. So there's also this event-driven model, event-driven programming model that is often tied to these serverless functions. Can you describe the connection between the event-driven model and the serverless functions as a service?

[00:09:33] VA: Sure. There again, there's a little bit of – I'm just going to explain just one little thing about that, is your absolutely right. So now that we have our function code sitting there is quite – It's fun, but how do you go ahead and actually execute it? That's where typically the event-driven comes into play. In the Knative, what we have done is we have decoupled or allowed for a late binding of being able to say when something happens, say, a new object is loaded into storage or something happens in your database, you want to be able to go ahead and decouple the definition of what happens from when it happens.

So when we build our container or the function, nothing happens by itself. There's an explicit act of binding, if you will, that function to an event that occurs somewhere else, and that is how you go ahead and say, "When something happens over in my event source or the event generator, that I would like that function over there to be invoked." Does that make sense?

[00:10:41] JM: It does.

[00:10:43] VA: Great.

[00:10:43] JM: There has been a gradient of different standalone container deployment environment. So Google came out with Google App Engine a long time ago, and that could be – People could say that's the first serverless thing. There's also Heroku. I think Heroku did something similar by allowing these dynos to deploy your application that automatically scale up and down. Dynos as far as I know, were containers basically under the hood. Then you have more recently these more programmable container instances, the long-lived container instances, like Azure Container Instances or AWS Fargate. These are longer lived container instances.

Then on the other end of the spectrum, you have the serverless functions, where when you deploy the serverless function, it doesn't immediately spin it up. It only spins it up when you actually want to demand it. So you have this gradient of how long the code is going to be in a running container where you're getting charged for it. So the scale up and the scale down, there's a gradient there.

Can you talk about the different, I guess, lifetimes or lifecycles of ways that people can deploy code on to standalone containers?

[00:12:04] VA: Sure. Yeah, we definitely have seen that there's certainly different ways of running your containers and there's different kinds of workloads. So for example, in the Kubernetes, you might do batch jobs, and it still runs as a container. So it can very long-lived.

But the case in point for you that you brought up is that is there something very specific about the fact that it is a function or it is just an application? I'm just going to drill a little bit into that, because one of the things that we have been thinking quite a bit with the Knative is that what really makes – Like from a developer standpoint, why is a function so different that when I write a function code, I have to go ahead and choose my toolset and my tool chain early on in the early process when all I want to do is write some software and eventually I might want to tie it into eventing structure. Is that really that different from saying, "I want to tie it into a user invoking a webpage." That's the one thing we wanted to go ahead and really think about like why is it really different. TLBR, we think it's not.

[00:13:12] JM: Right.

[00:13:13] VA: The second point that you brought up is like the lifecycle there, which is like, "Okay. Well, is there a good reason why – Or something that makes it so different for the fact that a code runs for 10 seconds versus 10 minutes. Typically, the reason for running long-running things is that the might have some state associated with them. Typically, the serverless workloads have been stateless. Now, this is where again you kind of squint your eyes a little bit and you say, "Well, according to best practices with the microservices and anything else, you really should be building stateless microservices wherever you can."

So for many of the traditional microservice architectures, there really is not that much of a difference between a function and a microservice. Something calls you and you do something. Then if you rely on the existing services that your cloud or your IT department might do, then you can get very far from the software developer's perspective defining your application and your functions very similarly and having the same workflows and the same abstractions available to you, because if you think about an application, you brought up App Engine. So if I were to go ahead and write my web app, I kind of want the same kind of characteristics out of that, which is pay as you go. So if nobody is using my service, let's how that never happens. But if nobody is using my service, I shouldn't have to pay for that.

Then on the other end if all of a sudden I get a lot of requests coming to me because everybody loves it, then please, Mr. Cloud Provider, or Mrs. Cloud Provider, spin it up for me as much as I need to, because there really is no difference. Service these requests is really what should be the core competency of the platform of the serverless that gives you regardless of whether it is a function or an application.

[00:15:09] JM: Let's get into Knative. What is Knative.

[00:15:13] VA: Great question. So Knative is really the building blocks and the abstractions for bringing in serverless workloads into the Kubernetes. That is in a nutshell. So one of the things that, again, with the Kubernetes, with the early APIs, there was a very principle set of abstractions that were really focused on, again, using the term assembly language for the cloud native computing. So Knative brings very principled resource model on top of Kubernetes to go ahead and target the application developer, not the operator, which typically has been kind of a split hat. So if you typically run on Kubernetes as an application developer, you're kind of wearing two hats. You're wearing a little bit of the operator hat and a little bit of the application developer hat. So we want to go and have a really clean resource model that allows you as an application developer to come in and give that operator hat somebody else and not worry about it.

[SPONSOR MESSAGE]

[00:16:22] JM: HPE OneView is a foundation for building a software-defined data center. HPE OneView integrates compute, storage and networking resources across your data center and leverages a unified API to enable IT to manage infrastructure as code. Deploy infrastructure faster. Simplify lifecycle maintenance for your servers. Give IT the ability to deliver infrastructure to developers as a service, like the public cloud.

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

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

[INTERVIEW CONTINUED]

[00:17:46] JM: So in order to build an understanding for people for why this is useful, I want to present two different companies. So let's say one company is a database as a service company. Like let's say I've got this new NoSQL database that I have created and now I'm offering it as a service, and the way that I'm offering it is like I've built Kubernetes and I have built provisioning systems for my database. Then another company is, let's say, an insurance company, an insurance company where I've also provisioned Kubernetes and I'm running my application containers on top of Kubernetes.

If I'm hearing you correctly, I think that the database company might want to think about Kubernetes as well as Knative, but the insurance company, they should only be writing application logic. They shouldn't have to think about configuring a Kubernetes cluster. So would you say that's an accurate way of thinking about where somebody should be in terms of their Kubernetes operation necessity?

[00:18:59] VA: Yeah, I think the way at the high-level that is definitely a great way to go ahead and draw the split line there. I'll come back to the database company just a little bit. But, yeah, as far as the insurance company's developers, they should never have to think about what is a Kubernetes service and what is a deployment and things like that. They should be able to go ahead and focus if somebody, for example, is writing some software that is doing risk analysis or something like that. Then that should be their core competency and they should be able to focus on writing software that deals with that, not have to understand the intricacies of where it runs.

As far as the database company, the one thing that they might want to do is go ahead and expose certain things about that database so that it's easier for the people or the developers that use that service, say, on a Kubernetes cluster to go ahead and extend that application or more naturally to go ahead and react when something happens in there.

So back in the day we had store procedures in the database. So if you would think about like when a new row is created something, you might want to go ahead and do something interesting with that. Sorry, I'm now popping up the stack a little bit more when we talk about the late binding. That's a great way where you can go ahead and say the database company could go ahead and start, for example, firing off events when, say, new rows are inserted without having any idea who is consuming them, or if anybody is consuming them, because if nobody is consuming it, it's more like a tree in the forest. Nobody cares. If nobody is listening to them, nobody should care, but it should be available to you so later on the developer can say, "I wish I would know when a new user is inserted in the database," and that event is fired up, that's a great example of the late binding, where then they can go ahead and just write their function and then later on say, "I'm actually interested in those events." Does that make sense?

[00:21:06] JM: Yeah. So we're seeing a few reasons here to think about serverless in some different context. One is the stored procedure-like example, and I've seen this with a few infrastructure companies. So there's Auth0. Auth0 is an authentication as a service platform and they have something called Auth0 functions where anytime somebody authenticates, you can have a function execute as a call in response to that authentication API request, and that's really nice, because maybe the authentication only happens a couple of times a day and you don't want to have this dedicated server that is watching for authentication events. You just want to have it execute only when the authentication happens.

MongoDB also has a product called MongDB Stitch, and if you have a database hosted with MongoDB, maybe you want to execute in code in response to a change in your database, and that is exactly the example you gave, where you have a stored procedure like example. So you want to – And you also want to have that code execute close to the database probably, so it's maybe faster. Those are a couple of examples. Also, you've just got this idea more generally where at the insurance company you don't want to have the an actuarial data scientist have to think about, "How many containers are running? Did I configure the load balancer setting correctly?" and it's just like, "No! Stay in your code and you should be able to write your code and have it scale up appropriately and you shouldn't have to think about Kubernetes."

So these different serverless experiences, how does Knative cater to them? I mean, I see kind of a difference there, like on the one hand with the actuarial data scientist, maybe you've got a very compute-intensive system that needs to stand up containers that are going to live for a longtime. Then with Auth0 or serverless on database thing, it's more like short-lived things. So tell me more about the architecture of Knative.

[00:23:14] VA: Sure. So we talked a little bit about the build side of the things. There's a couple of different ways that we have broken up the Knative. So it might be useful, because we only covered the piece on how different source into a running containers. So maybe I'll spend a little bit of time talking about the serving side of things, because I think that one allows us to go ahead and tackle what the developer experience is for the user or did you actually want to go and go into how is it implemented in the Kubernetes side of things. I'm happy to talk about either one.

[00:23:43] JM: Let's start with implementation, and then you can talk about usage.

[00:23:47] VA: Great. Okay. Yeah. So within the Kubernetes there is a way to go ahead and extend the Kubernetes API. So Kubernetes has a very specific way that the resources are handled and there's a standard way of describing them. So each Kubernetes object has a type, if you will. So I am a pod, or I am a load balancer, and then it has a spec, where the spec is where you tell the Kubernetes how you would like the world to look. So that's your specification.

Then the last piece of the Kubernetes model is a status, where the object then relays its status back to you as a user so you can see, for example that, "Okay, the pod is up and running," or "No, the pod is not running, because the image can't be pulled," for example. So that is kind of the API model.

The way these are implemented is that there is the object spec and then there's always a controller. So the controller or the reconciler is responsible for looking at those objects and then making the world match that or die trying. So in the case of the Knative, we have extended that model so our objects look exactly like native, no K, just native Kubernetes objects and we have our own controllers than then watch those objects.

So this model is built on top of CRDs, custom resource definitions, which is a standard Kubernetes way of extending the Kubernetes API. The last piece for CRDs is something called a webhook that allows the platform to go and ensure that only well-valid and well-formed objects are entered into the database. So the reconciler or the controller can only go ahead and deal with valid objects so that nothing in there comes in. So that was a mouthful. So let me just summarize really briefly.

So on the Kubernetes side for Knative, we have a few controllers that watch for these objects that I talked about. So since I have mentioned build, there is a build CRD that when a user creates, it goes into the webhook, webhook validates it to make sure there's no typos and things are sane. Then it gets thrown into the API server of the Kubernetes, which is backed by an etcd. But then our controller for the build is watching for types of build objects. It looks at it and says, "Ah! Okay, here's a new build. I haven't done anything." So it looks at that, the spec for that and says let's say you give it your source code from the earlier example from a GitHub. It would say, "Ah! Okay. I need to go pull that code from GitHub," pulls to GitHub over and builds it and it goes, "Nobody ever makes mistakes. It's going to work right the first time," and the container is produced and then the controller will update the status and saying like, "Okay, I get that thing you did and here's your Docker image. Please go fetch it from some registry." Okay?

So this one basically describes how the objects turn into actual resources. Now, the kinds of resources we care about is not only builds, but we also have to go ahead and worry routing things to the correct location. So for example in the case of the insurance company taken and user login requests and everything else, we want to make sure that those requests actually show up, whether it's a function or whether it's your application. They go in the right places.

For those ones, we use Istio today as the default ingress gateway, so that when a request comes in, we program the Istio. Again, you as a Knative user will never see Istio so that you don't have to go in and do Envoy configuration and things like that. You can just say when a user shows up on this URL, please invoke that function over there, or my application over there.

[00:27:52] JM: Okay. Let's say I'm building a CDN, like I'm a company that builds a CDN and I want to develop a serverless platform for executing functions that are related to content on that CDN. So if I wanted to use Knative to build my own serverless platform, what do I need to do?

[00:28:15] VA: Sure. So just to ask a little clarifying question there, are you thinking about exposing those functions as something that the user, so somebody who puts the content into your CDN, that they can do certain things when things happen there, or is it more for you as the operator of the platform to go ahead and be able to go ahead and extend the system?

[00:28:38] JM: It's for the developers. So let's say this is a developer-focused CDN and developers want to be able to resize their image, for example, and it's like a very flexible –

[00:28:48] VA: Perfect. Yup.

[00:28:50] JM: Yeah.

[00:28:51] VA: Great. So a couple of cases having dabbled actually in that area a little bit myself. So there're a couple of things that you might want to go ahead and expose to them. For example, if you are pushing in a lot of things into the CDN, say, you have a whole website that you're putting in. You probably don't want to go ahead and push the images in there and then trickle them in.

For example, let's say I have 10,000 images. I'm overloading the term images, but I have 10,000 – Let's call them images. So I have 10,000 images, I push them in there. If one of them fails, would you still like to be able to go ahead and say flip the content over to it? In those cases you might be able to go ahead and say something like when a user initiates, say, an upload for the authoritative content for the CDN, they might be able to set up things like, "Please tell me when certain things happen."

For example, when all the content has been pushed, say, to all the locations on the edges, that might be something that you might want to care about. The other one might be when you can say things like, lie your resize example is you might say, "Okay. Well, I don't actually want you to go ahead and have to worry about things like giving me all the videos or images in the thumbnail formats and I don't want you to have to worry about videos coming in and you putting in subtitles."

So one thing you might want to be able to say is say, "You upload the original image, or the video, or whatever you're doing," and then once it has completed, you might be able to go and set up events like, "Oh! If it is a video of certain size, please fire off these 10 functions where they might be doing things like transcodings." Then the user would just provide the transcoding function that would only be invoked once the image has been successfully uploaded and like checksummed and make sure everything is good to go. So then the user again can focus only on, "Hey, I know I'm getting good, known good video and I can just focus on running the transcoder, or the subtitle or the image thumbnailer on it."

Yeah, and the nice thing is that where the Knative comes in in this case is that there could be a cluster, for example, that you could be operating on behalf of the user if they don't want to go and run their own cluster so that, again, they would just be focusing on the Knative API and they would just say akin to, "When new videos shows up, here's my GitHub repo that I want you to invoke at the very highest level," or if they want to go and run their own containers, they could just say, "Okay, please execute this function for me," so they wouldn't have to know anything more, except operate at the source code level.

[00:31:31] JM: As the serverless function operator, if we're building the CDN, you and I, what do we gain from using Knative? What does that do for us as supposed to – I don't know, doing this serverless platform some other way?

[00:31:45] VA: Ah! That's a great question. There're a few different answers to it. So one of them is that it gives you a little bit – Actually, a lot more freedom, because Knative is an open platform, so that you can run this anywhere. So we might be able to, for example, host that on different cloud providers for resiliency. Well, there're very many reasons. You might also want to go and host it in-house. Also, you might have a customer who says, "Oh, my goodness. My data is –" This might be a bad example, because it's going into CDN, but let's play with it, "I need to go ahead and run this in my own data center for something."

So by having a portable workload definition, it means that the customer doesn't have to learn new – You and me as a CDN operators do not need to expose the customers the different cloud providers intricacies, because we can give them the Knative API that is portable across the different clouds.

[00:32:40] JM: I've done interviews with a number of different companies or open source projects that are working on open source serverless systems that deploy on top of Kubernetes. There's Fission, there's Kubeless, there's Fn. There's a bunch of these things. They differ across some different characteristics. The one that stands out is their approach to the cold start problem. I'm sure there are other dimensions. Tell me how you see Knative relative to the other open source serverless systems.

[00:33:15] VA: That is a great question. So one of the hopes and dreams for the Knative is the fact that all of those projects are solving many of the very similar and I'm going to call them boring, not in a bad way, but in the sense that these are the problems that are not something that we should all be solving differently, okay? This is the core competency of the serverless platform. So if the community solves those problems together, if you think about the plumbing, then we can go ahead and build way better serverless platforms and add value at the top instead of go ahead and everybody solving – You mentioned cold start, for example. So if we can bring together a community and go ahead and solve, for example, the cold start problem in a way that allows us to grow not only the serverless, but also the Kubernetes community, then everybody benefits from this.

There's the feedback that we have gotten from various people, for example, the TriggerMesh folks who have been very, very active in the Knative community as well as like IBM, [inaudible 00:34:25] folks and very many experts in this very area. We are focusing on solving those problems together as a community and it's going to go ahead and allow us a community to go ahead and then build higher-level cooler things on top rather than everybody kind of reinventing the wheel at, for example, the cold start level.

[SPONSOR MESSAGE]

[00:34:53] JM: Triplebyte fast-tracks your path to a great new career. Take the Triplebyte quiz and interview and then skip straight to final interview opportunities with over 450 top tech companies, such as Dropbox, Asana and Reddit. After you're in the Triplebyte system, you stay there, saving you tons of time and energy.

We ran an experiment earlier this year and Software Engineering Daily listeners who have taken the test are three times more likely to be in their top bracket of quiz scores. So take the quiz yourself anytime even just for fun at triplebyte.com/sedaily. It's free for engineers, and as you make it through the process, Triplebyte will even cover the cost of your flights and hotels for final interviews at the hiring companies. That's pretty sweet.

Triplebyte helps engineers identify high-growth opportunities, get a foot in the door and negotiate multiple offers. I recommend checking out triplebyte.com/sedaily, because going through the hiring process is really painful and really time-consuming. So Triplebyte saves you a lot of time. I'm a big fan of what they're doing over there and they're also doing a lot of research. You can check out the Triplebyte blog. You can check out some of the episodes we've done with Triplebyte founders. It's just a fascinating company and I think they're doing something that's really useful to engineers. So check out Triplebyte. That's T-R-I-P-L-E-B-Y-T-E.com/sedaily. Triplebyte. Byte as in 8 bits.

Thanks to Triplebyte, and check it out.

[INTERVIEW CONTINUED]

[00:36:43] JM: Were there design decisions that you needed to make with Knative that diverged from the other projects? Because you could have just picked up one of those other projects and latched on to it and said, "This is the one that Google is going to work on," but instead you said, "Okay, we've got Knative and this is the one that we're going to focus on."

[00:37:05] VA: So there're few different things that kind of led into it. We did spend a lot of time looking at it and looking at the platform, looking at the existing landscape. The one thing that – Well, there's a couple of reasons for it, but one of them is something that I mentioned a little bit before, which is we don't' necessarily see a difference between, say, applications and functions and there was a lot of – This is what I kind of mentioned, which is when you come into this system as a developer, you have to make a choice upfront, which is I know I'm going to be writing this as a function. I personally view kind of the developer space as more of not either or. I view it more as a continuum, where you might have a monolith application that is exposing, say, 10 end points. So it handles all of your application logic in one. Then on the other spectrum, you might have 10 independent functions. But in the middle, for example, you have things like cooperating functions, and they need to be all rolled out as one and so forth.

So we wanted to go ahead and really start thinking about the fundamental primitive objects that would allow us to go ahead and tackle both problems at the same time so that the application as well as the function writers could all benefit from this. So that if I start with the function and I decide that, "Oh! Well, it actually makes sense to go ahead and bundle up things into it and it starts growing into a bigger program, or vice versa, I have a monolith and I want to start pulling things apart from there, it shouldn't require me to rethink the way I approach the problem or requiring a new set of tools and everything else."

With these object models, we were hoping to go ahead and unify the application developer, because it is not just applications and functions and like Emacs an VI battles.

[00:38:58] JM: That's so beautifully said, and how closely does that relate to the cold start problem? Because it feels like if I've got short-lived container functions that just need to execute occasionally and there's not like a strict SLA on them, then I can just have them scale down to zero. Then when they actually need to be called, they just get spun up and they get executed. But on the other side of the spectrum, you have my entire monolithic application that I want running all the time. So I can't afford to have the cold start problem when a user request happens. How closely is the design of Knative related to that cold start problem?

[00:39:43] VA: So what we have done is we have been able to go ahead and – First of all, let me say about the application also. It also applies to things like scale up and scale down and everything else. So there's different kinds of ways of scaling up. So it is not only applicable to functions, but if we have an application that actually can scale as the user request comes in, there's not a cold start problem there, but there also is a scale up event or scale down event and whatnot.

So we wanted to go and make sure that those apply again both the functions and applications the same, because you probably want your application to scale up as well faster, right? Nobody ever said, “Make it slower.”

As far as the cold start problem, there’s been a lot of research in it. We at Google have done a bunch of work on this. Many of our friends have done things like that as well, and there’re been various approaches to it. So what we are doing is solving this as a community effort and making sure that the object model and the architecture reflects this cleanly so that you can choose to go ahead and take advantage of your particular platforms implementing the Kubernetes if you have different ways of speeding things up, for example. Does that make sense?

[00:40:57] JM: It does make sense.

[00:40:58] VA: Yeah. So what we wanted to do is make it into a component that can be used all by itself, just like we started with the autoscaler, because when we looked at the Kubernetes autoscaler, that’s some limitations that would not allow us to use it as is. So we have an autoscaler and we have a way to go ahead and extend that autoscaler or to plug it into it.

Joe Burnett gave a fantastic talk, if that’s your area of your curiosity over at the KubeCon, because that’s a full-blown talk in itself. It was at KubeCon in Seattle last December.

[00:41:33] JM: About Kubernetes autoscaling.

[00:41:35] VA: This is about Knative autoscaling. How to plug it? How to extend it and everything else, and you could do a full talk on that. But anyways, so that’s part of the reason why we’ve been carving out these – Rather than saying you take everything, you take Knative or nothing, is that it’s very composed into pieces that have clear set of roles and responsibilities and they are very principled objects so that you can go ahead and come in and either replace a piece. You can extend it. You can plug it and you can choose to only use certain pieces or the Knative.

So there are some people who come in and say like, “I really like your stuff, but I don’t want your build. Thank you very much.” There are reasons why I need to build my stuff the way I do or regulations and everything else, and it’s totally fine. It’s great. You can use the other parts of Knative, but it is not a traditional do it this way or else.

[00:42:31] JM: Let’s talk about those three components of Knative. You have build, eventing and serving. Explain how these three components work.

[00:42:40] VA: Sure. So the build component’s responsibility surprisingly is taking source code and turning it into runnable containers, as you might have guessed from the name. So this, again, is a very good set of roles that you as a developer might not care about. You just want to write your software, and you don’t necessarily want to be running things like Docker builds on your machine, or you don’t necessarily want to run it within your cloud. You might just want to say, “Turn this into source in the runnable image.”

So we talked a little bit about the build [inaudible 00:43:15] earlier on. So I think we might have covered that. But the serving stack is responsible for taking incoming requests and routing them to the appropriate handler. I’m just going to be vague on the handler for now, because I don’t want to overlook the term, whether it’s a function or an app, but let’s call it handler.
So the serving stack introduces a couple of different resources that you probably as a developer care about. One of them is what is called a configuration. So configuration is your code, okay? Let’s call it your code because it could be either image or a source in this level, but let’s call it your code. Then it also bakes into configuration of your code.

The configuration represents what you as a developer care about, which is, “Please run my code with this configuration.” What happens underneath is that gets turned into a revision, where revision is immutable. So this one is we basically take a snapshot of your code, your binary as well as the configuration according to best practices, you never ever, ever modify running code. You make changes to it, so you have a history and you can run back to it and so forth. So revision is immutable snapshot of your stuff in time, okay?

Then the other piece you care about is route my stuff into it. So if I go ahead and say, “Okay, I’m running Knative and I want to go ahead and take my example.com/login, I want to wire it into the revision that just created. That's really all I care about." At the highest level, we have something called a Knative service, or Kservice, which bundles those things together. It's more of a convenience function so that you as a user come in to the day zero experience and just put a pointer to GitHub and saying, "Run this for me, and don't screw it up." So we handle that. But those are the four objects that we care about.

The nice thing about this model is that it allows you as a developer come in and start with the day zero, get up and running very quickly and get your code up and running, and you never see any of the underlying concepts for things like traffic splitting or anything else like that. But as you grow, many of the systems or some of the systems out in the world have a very appealing day zero and maybe even day one characteristics, which means that you can get started. Then once you start go ahead and actually running the system and maintaining it and upgrading and things like, then you have to go ahead and jump out of that world and do something totally different.

With the Knative, the serving stack, is responsible for growing with you. So the reason why we have broken the object model into routes and revisions is because there's a big difference when you are, say, a single developer and you might just push your code. Whereas if you're running a big company where you might have an SRE organization that is responsible for being able to go ahead and do things like experiments. You might want to go and do things like, say, "Oh, I have this new version of code that the developer pushed out." Developer might not – You might not want to allow developers to flip all of your traffic at Friday night at 10:00 just because it seems like a great idea and you might have roles and responsibilities set up such that developers can push code all they want and they can do their tests against it, but none of the user-facing traffic is affected by this. So these are all the concerns of the serving stack as far as the incoming traffic.

On the backend, the serving stack also handles things like zero to one scaling, autoscaling, making sure that things are up and running, health-checked and things like that. So there's kind of true pieces in that, and I think I kind of covered already the autoscaling and the cold start things, but those are all handled by the serving stack. Does that give you a good enough picture for the serving stack?

[00:47:21] JM: Absolutely, and it reminds me some of the conversation I had recently with David Aronchick from the Kubeflow project, because these problems that you're describing where you have, for example, a machine learning model that is operated one way when you first deploy it, and then that operation will change overtime or it needs to scale up or you need to do traffic splitting to A-B test different machine learning models. You don't have to have to go through this complex re-architecting of how you're deploying that machine learning model. It'd be much better if the platform itself was taking care of that so that you could just think about the machine learning attributes.

[00:48:06] VA: Exactly. Exactly. Yeah, and that is precisely the reason why we spent a lot of time working on the object model and making sure we get it so that it's idiomatic to the user and it allows you to have clear set of roles and responsibilities. So as a developer, I can just write my software. I have all the tools necessary for me to go ahead and run my code, but I don't have to worry about exactly the things you said. How does it happen? I don't care. Just do this thing for me.

[00:48:36] JM: You've been at Google for 11 years. When you look at the design at Knative, are there some lessons that Google has learned that come to mind that you're bringing in to Knative?

[00:48:49] VA: Yeah, there's a lot. So a lot of – Again, for example, with the App Engine. So App Engine was truly the original serverless platform in my opinion, and I also personally spent a bunch of time in the cloud storage. One of the things that became very clear early on in that land for me was when people were asking about things like, "Hey, we need listing to be faster," and I was like, "Well, why do you need it to be faster? They were like, "Well, so I can see if there's a new object there." So I scratched my head and I was like, "Well, that seems – I think we can do something a little bit better. How about if I tell you if there's a new object there?" They were like, "Oh! You could do that?" I said, "Well, I can write that."

So it was like this whole event-driven thing was the ability, which is instead of sitting there and pulling things, just like I tell you something. So that was the object notification we did for cloud storage back in the day and it was very, very popular.

But anyways, so there's a lot of operational aspects that we have done over at Google running container workloads forever. So a lot of those learnings we first exposed in Kubernetes, right? Then the second point that we have done there is really said, "Okay, great." Again, just going back to my assembly language anecdote if you will, and it is not a diss. It's a good thing. Really, is being able to go ahead and then take those things and codify them into best practices, and this is not only at Google by the way. I just want to point out, because we have a very vibrant community and a lot of experience from many, many folks in the industry that have been running big, big workloads and clusters and serverless.

So Google is bringing definitely things in the table. Well, we are not alone. I just want to point that out. So there's a lot of operational experience that we have there. We also have a lot of understanding on what does it mean to really run serverless at scale, whether it's app engine, whether it's cloud functions. We also understand what a model looks like where you rely heavily, extremely heavily on everything being microservices, so that you don't really have so much state flowing around everywhere, where if you can go and start doing these loosely coupled services to get there, it makes for really, really nice user experience.

Then, of course, there is Istio that allows us to go ahead and not worry about every application developer, having to worry about authentication and authorization, because it's tricky, right? So let's go ahead and not worry about that. So there's a lot of these best practices that we want to make sure that our developer friends or developers like myself can go ahead and do the thing they care about, which is write code, write great software that makes people happy.

[00:51:41] JM: Google Cloud Storage, you mentioned, that you had worked on that. That's the object storage in Google Cloud, right?

[00:51:47] VA: That's right, yeah. Yup.

[00:51:49] JM: Like the bucket storage?

[00:51:50] VA: Blob store, yeah.

[00:51:51] JM: So I presume that within Google, there was some sort of object storage that developers could use like internally before Google Cloud Storage made it an external experience, right?

[00:52:04] VA: That's correct. Yup.

[00:52:05] JM: So why did it take the external developers to say, "Hey, we would love to have some sort of event-driven model." Does it surprise you that that demand didn't come from internally before the Google cloud demands on it?

[00:52:21] VA: Yes and no. This is my gut feeling, right or wrong. It wouldn't be the first or the last time I'm wrong. But Google, there was a lot of batch processing that the blog store was used for initially. So very, very large, large items. So if you think videos and everything else. So it was lending itself more in the batch processing land rather than very event-driven. That's just my gut feeling.

[00:52:48] JM: So interesting. It sounds like there were probably internal benefits for starting to – Because since you've been there 11 years, you've seen the rise of Google Cloud. So you've probably seen the benefits of externalizing into a cloud provider. There's probably some learnings that have transmuted to the internal infrastructure too, right?

[00:53:10] VA: Oh, definitely. It has been mutually beneficial transition. Transition is the right word, but it has been – It is fantastic to go ahead and get more – Be able to go ahead and talk so much more freely with the customers. I mean, that is one thing that I truly enjoy, is talking to developers and customers and users and partners and being able to share our learnings and then be able to go ahead and learn from them as well so that it isn't just a – Shall we say internally-focused, where all the users are internal. It's a different mindset. I think that you touched on that very, very nicely, but it's a different mindset when you're producing software that is used by your own developers that all understand it and they are expected to work within a certain – I don't want to say mindset, but they have a certain set of tools and things that are available to them. Whereas when you go into the outside world they're like, "Well, hey. Why doesn't this work on this random thing off of the interwebs that I just found? It's really cool, but I want it to work," and it's like, "Yeah, why doesn't it work?"

[00:54:17] JM: So we've seen the adaption of Kubernetes by all the cloud providers and by many other companies who are not cloud providers, and I'm wondering how you think Knative adaption will effect different cloud providers in different organization. In the next two or three years, what are we going to see in terms of impact of Knative?

[00:54:40] VA: Ooh! Let me look at my crystal ball. So my hopes and dreams are that it really is going to go ahead, and we have seen early signs of this already, where it really is going to go ahead and be the new target way that the application developers think about getting software running into the cloud regardless of whose cloud it is. As long as it's got Knative, it's got Kubernetes, then it really is an easy way for the developers to come in.

So I expect that, because we have been able to pull all of our resources together and focusing on what I call the boring problems. Then the platform is going to go ahead and get better and better. But where I really see the excitement coming at the higher level is like what are the kinds of tools and applications that people are going to be building on it? So if we think about the Kubernetes when we first open sourced it, it's like, "What is this thing?" Then people are like putting things on top of it, and now again we are getting Knative, which simplifies things and makes it in our opinion, of course, much more delightful to use so that now then people can go ahead and come in and build really neat things on top of it and make things even better.

So I'm hoping that this makes the software writing approachable and really large scale distributed computing available to people without having to worry about being an expert in it. It's more like visual basic was awesome, because it basically allowed people that we're not computer scientists who write software, and I'm hoping that by Knative and serverless computing paradigms, we can go ahead and let people to start tinkering and writing software that they don't necessarily need to understand about all the things underneath there, but they can build cool stuff. That's what I want to see. Help people write cool stuff on top of Knative.

[00:56:32] JM: Ville, I'm excited for the same future that you're describing. Thanks for coming on Software Engineering Daily. It's been really fun talking.

[00:56:39] VA: Thank you so much for taking the time. I really appreciate it. It was fun.

[END OF INTERVIEW]

[00:56:46] JM: OpenShift is a Kubernetes platform from Red Hat. OpenShift takes the Kubernetes container orchestration system and adds features that let you build software more quickly. OpenShift includes service discovery, CI/CD built-in monitoring and health management, and scalability. With OpenShift, you can avoid being locked into any of the particular large cloud providers. You can move your workloads easily between public and private cloud infrastructure as well as your own on-prim hardware.

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

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

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

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

[END]