EPISODE 818

[INTRODUCTION]

[00:00:00] JM: Google's options for running serverless workloads started with App Engine. App Engine is a way to deploy an application in a fully managed environment. Since the early days of App Engine, managed infrastructure has matured and become more granular. We now have serverless databases, queuing systems, machine learning tools and functions as a service. Developers can create fully managed, event-driven, highly scalable systems with less code and fewer operations.

Different cloud providers are taking different approaches to offering these serverless runtimes. Google's approach involves the open source Knative project and a hosted platform for running Knative workloads called Cloudrun. Steren Giannini is a product manager at Google working on serverless tools. He joins the show to discuss Google's serverless projects and the implementation details in building them.

A few updates to Software Engineering Daily land. Podsheets is our open source set of tools for managing podcasts and podcast businesses. A new version of Software Daily, our app and ad-free subscription service is available at softwaredaily.com. We're looking for help with Android engineering and other roles. You can find those at our Software Daily FindCollabs. We particularly like your feedback on the apps, the mobile apps, which are available in the iOS and Android app stores, and we definitely could use some help ironing out all the bugs there.

The FindCollabs hackathon has ended. FindCollabs is the company I'm working on. The winners of the hackathon will probably be announced by the time this episode airs, and we'll be announcing our next hackathon in a few weeks. So please stay tuned. The updates for today are in the show notes for today's episode, and let's get on with today's show.

[SPONSOR MESSAGE]

[00:02:05] The 2019 Velocity program in San Jose (June 10-13) will cover everything from Kubernetes and site reliability engineering to observability and performance to give you a

comprehensive understanding of applications and services—and stay on top of the rapidly changing cloud landscape. get 20% off of most passes to Velocity when you use code "SE20" during registration at velocityconf.com/sedaily

.

[INTERVIEW CONTINUED]

[00:04:48] JM: Steren Giannini, you are a PM at Google Cloud. Welcome to Software Engineering Daily.

[00:04:53] SG: Hi, Jeff.

[00:04:54] JM: So you work on App Engine, and App Engine I think was before its time in many regards. It was serverless before serverless was a word. I do wonder what would have happened if App Engine would have had the word serverless associated with it. Probably it would have grown even faster.

Tell me about the early architecture of Google App Engine, because I think was – What? Like 8 years ago, or 6 years ago? Something like that?

[00:05:19] SG: You know what? 11 years ago.

[00:05:20] JM: 11 years ago.

[00:05:21] SG: Yeah. We celebrated 10 years not that long ago, and now it's 11. So you're right. The term serverless was not coined at this time, but if we take the definition of serverless as fully managed auto scale pay for usage, that applies to App Engine. You're right.

So, yes, 10 years ago, Google had a serverless offering. Actually, Google Cloud did not exist at that time. At that time, App Engine was the platform. Actually to your question about the architecture, that's why within the App Engine product, the original one, were baked in many, many APIs in order to help people build apps. APIs to store blob, data stores, task queues. All of these were features of App Engine at the time. Of course, the original architecture was

around supporting very narrow set of runtimes, as we call them. I think the first one was fully Python, then Java and PHP came, but how we achieved that was by actually modifying those runtimes.

So we actually had Guido, the Python author, actually working on App Engine to make sure that runtime is safe to execute customer code on it, because this code is, at the end, running on the Google infrastructure, on the Google orchestrator next to the other workloads that Google is running. So that's why at a very early stage, App Engine was using custom runtimes, custom modified language runtimes, which is why we haven't released a lot of different runtimes I the early days of App Engine, because that was a lot of effort to bring even just a new language to App Engine.

[00:07:13] JM: So when you think about that evolutionary spectrum from the early days of App Engine to the richness and the variety of the different cloud APIs that are now there, tell me about the main milestones, the main timeline milestones you think about in the evolution of the products that you're building on Google Cloud from those early days to the modern day where you have the spectrum. It's almost like a monolith to a microservice, that kind of migration.

[00:07:40] SG: Yeah, yeah. Actually, we went from, as I said, one product, which was App Engine, to a whole platform. At the time, App Engine was the platform. So what I mentioned about a built-in data store. Well, of course you had to store your data, except that now App Engine has moved towards focusing on compute, and many of the GCP product have actually graduated out of App Engine.

Google Cloud data store is just the graduation of App Engine data store. Google Cloud Storage is actually App Engine blob store in its own product. One of my favorite one that we recently released as a standalone product is Cloud Tasks. So it's actually the same infrastructure as App Engine Task Queues. You send a task into a queue and the queue dispatches the task to a different service. This is now offered as a standalone service where you can not only target HTTP App Engine targets, but actually also any HTTP targets.

So what we've seen over those 10 years is many things that were originally built into App Engine became their own GCP products. On the other side, of course, in 10 years, many things

have changed under the hood. Our customers have always benefited from day zero until today from a managed security and update of their applications. Of course, auto scaling was here from the beginning, but many thing have changed.

I think I want to call out at least two of them. The first one is the App Engine scheduler. So the algorithm and system that decides how many instances will your service get at a given point in time. I worked on the release of a brand new scheduler. I think it was two years ago, where we completely changed across the whole fleet the scheduling algorithm without any latency impact or billing impact to our customers.

So you can imagine, you have really a lot of customers on App Engine, including very big ones, and we seamlessly changed a core piece of the architecture while making save money, because at the end I remember 8% reduction in cost on average for our customers just by changing the way we scheduled instances. When we do those changes, we really measure all across the board. There's no latency impact, because at the end, when you use a serverless product, the infrastructure is not managed by you. So there's some kind of a trust relationship between you, the customer, and the platform provider who has a part of responsibility into the latency of your application into the availability of it and into its costs. So that was one of the architecture change I worked on and it took a very long time for the team to achieve, and now we are fully no this new scheduler.

But another big one I think even more important is the sandboxing technology. So as I mentioned in your previous question, originally, each language that App Engine was supporting was actually modified version of that language. For Python, we had to modify Python. For Go, the same.

What we realized is that every new version of the language, we had to redo those changes, and sometimes when the language changes a lot, you have to basically have to start from scratch and redo your sandboxing by plugging holes into the language to make sure that the code is not able to escape outside of this runtime.

So what we ended up doing is taking a different approach, taking an approach where we actually sandbox not at the runtime level, but at the OS level. So that's why Google has released

gVisor, which is the sandboxing technology that we use for App Engine, for Cloud Functions and cloud run.

[00:11:26] JM: What does that mean? What's the difference between sandboxing at the OS level versus the runtime level?

[00:11:29] SG: So now, in every second generation App Engine runtimes, what we use is just off-the-shelf Node.js, off-the-shelf Python, off-the-shelf Java, except that we run it into a secured container runtime sandbox. Now, the isolation is not by patching the language, but it's actually by making the language believe it is on a regular machine. It's actually not. GVisor will actually implement many, many syscalls and make sure that they are fully secured and fully isolating, the workload, from the rest of the environment.

Because, yes, I want to highlight that. Containers are not an isolation boundary. On a given host, you can – Something like Jira will the piece that will help you really sandbox those containers so that they cannot escape to the host machine.

[00:12:27] JM: Now, certainly security is one area of sandboxing, but there's also the noisy neighbor. If I co-schedule two containers on to the same OS and both of them happen to be super hungry containers, I'm going to start to get performance degradation from at least one of them. To what extent has the noisy neighbor problem been an engineering issue that you've tackled in the sandboxing work?

[00:12:52] SG: I don't think I'm the best person to answer that. I have not worked directly on this.

[00:12:59] JM: Am I wrong about that? It is an engineering problem.

[00:13:02] SG: I know, the App Engine has one way to schedule workloads. The other cloud providers have other ways. I do know that, yeah, we are allocating a certain amount of CPU time [inaudible 00:13:13].

[00:13:13] JM: To me, it seems like a super edge case and it's like pretty easy to solve by just shifting the workload in most cases, or you spin up another, "Okay. We've detected a noisy neighbor. Spin up a copy of this instance and start directing traffic to that instance, then delete the noisy neighbor." It's just an edge case.

[00:13:27] SG: It's also something that at scale balances itself. Of course, we say serverless, but I can tell you, there are servers, a lot of servers. When you have such a large fleet in a very large pool of host machines, if you want, then it's easier to have diversity in those workloads and to balance this problem within this pool.

[00:13:50] JM: I wonder if static analysis tools have gotten good enough to be able to predict the performance of piece of code. We're probably not there yet, right?

[00:14:00] SG: No. What we are offering on App Engine is the ability for our customers to control a little bit the auto scaling algorithm by telling us, "Okay, you should target that CPU utilization." For about 60%, that means that we will detect that your instance starts to use more CPU, then we will prepare another one so that the next request that comes in have an available instance if needed to be processed. So this is more to help the latency of the end application.

[00:14:30] JM: We've gotten pretty into the weeds here. Let's take a step back and just talk about that word serverless. So I talk to you before the show that you work with or at least are friends with Ville Aikas, who's on the show previously to talk about Knative. Knative is interesting and in some ways different than the other serverless perspectives that we've seen around the industry. I think Knative sees less of a distinction between the function as a service modality and the container as a service modality. Container as a service meaning, "Oh, this is a long-lived container. We want it to sit around for a while to service. We need to accept requests and we need it to be up basically indefinitely." Whereas the function as a service modality is like, "Yeah, you just hit this thing. It's like a Google Cloud Function. It spins up kind of on-demand," and that's fine. Then it spins down when it's done. Why is there a gradient between those two extremes?

[00:15:33] SG: I think there is one very important takeaway that people should remember when listening to this podcast is serverless doesn't apply only to functions. Let me explain why. I

think Ville explained it a little bit in the previous podcast. When you deploy a function to Google Cloud Functions, for example, your function is a piece of source code. The first thing that we will do is build that code to something that we can execute and auto scale.

Today, the industry standout for that is the container. So the first thing that happens when you deploy to Knative, your source code, is that Knative build is used to transform this source into something that can be auto scaled and managed by Knative serving. There's another piece of Knative, which is Knative eventing, that this piece will take care of making sure that this Knative serving service is triggered when this other thing happened in your cluster or even outside of your cluster.

So eventing is about binding things together, triggers, and serving is about actually serving your workloads, your containers, and building is about coming from source. This source could be just a simple function. Not even something that can execute by itself, but that is wrapped into what we call – In Google we call that the function framework. So that's what's happening when you deploy to Cloud Functions.

Let's take Node.js as an example. You give us a one file that exports one function. That doesn't execute by itself. Even in your local machine, you cannot run that. So what we do first thing, we inject what we call the Google Cloud Function frameworks for Node.js, which by the way we open sourced today. So now the piece that we use to wrap your function around is open source, and we take that. Now we have a Node.js application that actually you can start with npm-start, it would start on your local machine, but that's not enough to deploy to serverless, or serverless infrastructure, or to Knative. You need to build that into a container, and many tools are helping you do that. Google Cloud Function does that completely seamlessly for you without you realizing it. You can do it by hand with Docker, with Google Cloud Build.

Personally, I'm a very big fan of Buildpacks. Something that entered beta recently, and you have the tac command. You run tac on your Node.js source code, and boom! You have a container. This industry standout of containers allows us to all agree that this is the piece to distribute and deploy software, and then it's about the infrastructure to run this container. In that case, under the hood, Google Cloud Functions, once we've build your function to a container, we are

executing it on the same serverless infrastructure as what we use in this new App Engine runtime.

To continue, it'll end up being sandboxed by gVisor and executed in an auto scaled manner on our infrastructure, and it's the same for Cloudrun, the new product we announced today, which instead of taking source code, this time it allows you to give us any container.

So tie that back to Knative, Knative serving is serving containers within your Kubernetes cluster. So what we've done with this latest product that I mentioned, Cloudrun, is that now we are exposing the same API, the same Knative serving API as a fully managed solution running on our infrastructure that you don't have to manage, you don't have to worry about the cluster, or you don't have to worry about provisioning infrastructure, because we do that for you. As a consequence of that, we can deliver a pricing model that is really, truly, pay per usage, which is the serverless pricing model. When you use Cloud Functions, you pay for your number of request and the execution time of this request. Well, it's the same for Cloudrun, except that this time we do the same with your container

[SPONSOR MESSAGE]

[00:19:53] 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, signup for a free trial today and get a free t-shirt. Visit softwareengineeringdaily.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 CONTINUED]

[00:21:45] JM: So let's say I build an app on Google Cloud. Actually, I am doing this right now in the sense of Firebase. I'm using Firebase today. It'd be awesome to see Firebase eventually become this level of open source. But let's just assume we're talking about Google Cloud, and let's say I start building this app today and I do it entirely through Knative, but using the hosted options on Google.

Then let's say in a year a half I decide I want to also be multi-cloud. I want to move this entire serverless infrastructure deeply integrated with serverless APIs. I want to replicate it on a different cloud provider. How easy it for me to do that?

[00:22:28] SG: So starting by the compute part of your application, what we ensured with this new product, Cloudrun, and then its sibling, Cloudrun on GKE and with Knative is that those three products share the exact same API. So sharing an API means that any tool that work with one will work with the other.

Let me dive a little bit more into Cloudrun on GKE. Cloudrun is something that we announced today, but it has two versions kind of. The first version is really fully managed, don't worry about anything. This is probably the easiest one to get started. It has a user experience, which is simple developer experience console CLI. Using that same developer experience, you can, if you want to, deploy inside your own Kubernetes cluster this time with Cloudrun on GKE.

So from a developer perspective, it's literally the same, except that this time, because it's within the cluster or your organization, that you are in charge of managing this Kubernetes cluster and, of course, the influence. Within the cluster, your workloads are auto scaled, and we are using the Knative serving auto scaler for that.

But from a developer perspective, the container that you deploy is strictly respecting the same runtime contract, and the user experience you have is strictly the same, including at the API level, which means that we have partners, like GitLab, who have integrated with this Knative serving API. This means that those partners are now able to deploy to Cloudrun too because of this shared API.

So to answer your question, what happens if I want to move out of GCP? Well, we designed all of these from the ground up to answer that specific question. The thing we've heard in the past ten years on App Engine was it's great, but I feel a bit locked in, especially with those proprietary APIs that I mentioned early on in the product.

So now our answer is, "You know what? We are actually providing you the open source projects to run the exact same thing on your own or anywhere else Kubernetes runs. To do that, just install Knative in this cluster and you will have the exact same API able to run at the exact same containers than what you have on the fully managed version of Cloudrun. Really, we ensure that the container contract is strictly the same. We are working on conformance test to ensure the APIs are strictly the same. This is something that is core to our strategy.

So last year at Cloud Next 2018, we started to pave the way towards this vision. When we announced Knative, that was the first step. Okay, first step, open source project. Then I don't know if you noticed, but we also announced containers on serverless, and we announced the GKE serverless add-on. All of these was actually the very first step towards today, which is Cloudrun on GKE and Knative allowing that portability.

So you should really see Knative as a reference API that Cloudrun, Cloudrun on GKE and/or many other partners are implementing, as well as a reference implementation that Knative, the open source project itself, is providing on GitHub.

[00:25:56] JM: One thing I've heard – I've never built an application that scaled enough to have this problem, but I've heard that the lock-in doesn't necessarily come from open source or close source. Well, it could come from the close source APIs obviously, but it can also come from the IAAM layer, the identity and access management layer.

So if I build my app entirely on Google, I've got certain access policies and role management and stuff that I'm doing that is closely associated with the Google Cloud identity and access management system. My entire devops workflow, my entire permission system is now deeply integrated with this proprietary IAAM layer. I've got 200 people. If I want to migrate my entire serverless infrastructure, I now have to remap the IAAM roles to the entire – Whatever cloud provider I'm migrating to IAAM's system. I'm never going to want to do that.

[00:26:58] SG: So this is true, and that's why when I started my answer, my first answer, I started by, "If you want to migrate your serverless compute, we designed it to be easy." Then I just said, "The landscape is way bigger than compute. There is all of the security and identity. There's all of your data. There's all of potentially software as a service that your application uses that is not delivered as any open source equivalent.

So, yes, indeed. If you want to migrate your entire architecture, you have much more to worry about than only the compute. But at least the compute piece, we have built it so that it's easily portable. We have Cloudrun alpha testers, or actually using both Cloudrun and Cloudrun on GKE, and they start this Cloudrun and they realize, "Oh! Maybe I need access to a GPU to do some graphical computation. They do image analysis."

Well, they just – "Okay, fine. I just moved to Cloudrun on GKE with the same tooling." Actually, these customers are also looking to guarantee to their own customers that they will not be locked in. So what does that mean? They do IoT on water systems, in cities. So when a city signs a contract with them, it's [inaudible 00:28:14] by the way, where they send a contract with them, they want to make sure that even if they change contractor, the whole infrastructure will keep running. After all, it's water systems that people are relying on for living.

So they want to make sure that by contract, they are able to move to a different cloud provider, to on-prem if needed. So the Knative story resonates a lot with that customer, because they have the guarantee that those workloads are portable. The tooling they are going to build, they are going to easily be able to target different backends, if you want. Yeah, there will be a bit of identity transition to do if you want to move to another provider.

That being said, Kubernetes is becoming a platform on its own, and Kubernetes RBACs are something that are shared across managed Kubernetes offerings. Even the identity layer is starting to kind of become standardized via Kubernetes.

[00:29:19] JM: Sure! SPIFFE and SPIRE I think are kind of related to that, or no. That's more like workload authentication, I guess. I think the whole lock-in discussion is like – I don't know why people focus on it so much, because the cloud providers are competing down to a

commodity level cost, and maybe there's going to be some cost differences in the margins, but kind of a whole reason we're all in the software industry is because the things we build are pretty high-margin. Why do you think we're all so obsessed about this whole multi-cloud thing?

[00:29:54] SG: I think for us it's really, "Okay, you can do it on your own. You can do it elsewhere," but we believe we have the best offering. When I say that, it's more, "Look. Yeah, you can totally install Knative on your own Kubernetes cluster that you run on bare metal or on VMs," but also at the same time, you can take a look, and we offer a fully managed version of the exact same API where you literally have to provision nothing in advance and can just deploy in one command. Maybe you will pick that, because that's way easier to use a developer so that you can focus not on building your own infrastructure, but on building values for your business.

So that's Cloudrun. Yeah, you can do it on your own, and we give you the tool and the code to do that, but we also give you this hosted offering, where, literally in two clicks, you deploy a new container auto scaled in your all serverless environment and you don't have to pay for the infrastructure either. You only pay when you use it.

[00:30:52] JM: Have you talked to any companies that have done a really significant multi-cloud expansion?

[00:31:02] SG: I haven't, and I'm not the best person to talk about that for – I mean, I know we released many things today around multi-cloud, around hybrid, and I would recommend you to interview somebody who has worked more on that. For us, it's really about we give one piece of this whole story, but we don't give – Me, in the serverless team, we don't give the full story. We give Knative, which is part of this story of you run serverless on-prem, how you run serverless on your Kubernetes cluster. But the whole story, I will recommend you to interview somebody else. I think the field is very interesting.

[00:31:34] JM: But let's say you're running a giant insurance company. You're never going to move everything to multi-cloud. You're going to have some disaster recovery story that's multi-cloud. You're going to checkpoint your Kafka logs to like the other clouds, bucket store system perhaps. I mean, why would any of these companies want to spend that much time on going multi-cloud with their compute infrastructure? Is it just portability or like –

[00:32:08] SG: It's also reliability too, right? When every system that you design, if it's at least N+1, redundancy is going to help in case of one of the piece has an outage, for example. However, it's probably very hard to shift an entire stream of incoming request and tell you to one side or the other.

[00:32:31] JM: So maybe it's an ideal. It's an ideal we're aspiring to as an industry to get to this point.

[00:32:37] SG: Yes. What I do know is that we at Google Cloud are actually giving a lot of tool to achieve it. It's not easy. We are making it easier and easier, and let's take a closer look at the announcements of today in that area. But it's not easy.

What we've seen more, for example, one Cloudrun anecdote. People were running batch scripts on their machines, because, yeah, batch scripts runs maybe once a day with a serverless architecture that allows you to run any language, because it runs any container. You can run batch scripts on a serverless way, and that's how people start to realize, "Oh! I can get rid of that small machine that I have on-prem and move that to the cloud. Thanks to container. Yeah, if I want to move to another cloud, I know this other cloud can also run containers. That's the industry standard."

Of course, containerizing, it's definitely a no-brainer for hybrid clouds, for moving to the cloud. Even containerizing a batch script is the answer. Yeah, you use cron as a service that we offer a newer cloud, which is Cloud Scheuler. Then you can easily say, "Run this service once a day," and it will run your batch scripts once a day. Whenever I like to do some, say, data backups or anything, any batch scripts can actually run in a container. That's quite interesting.

[00:34:00] JM: Interesting. So when you open source something like this, there're a lot of questions around how do you want to do licensing. What should your relationship be with open source kind of foundations, like the CNCF? Was there any interesting decisions around how you structured the open sourcing and relationship to foundations and relationship to other corporations?

[00:34:27] SG: From the beginning, we've build Knative with many partners. Very early on, as soon as we decided that we would walk in the open for Knative – I mean for this project, it was not named Knative at the time. But we decided to start sending some emails and contacting others to see if they would like to join us in that journey.

So that's where we were when we announced Knative last year, where from the day of the announcement, many of our partners were already contributors to Knative, even some of them had already managed Knative offerings, and we did not, but they had.

So from the beginning, we've been building that with our partners. I mean, the API working group of Knative. In that group, there are many people from outside of Google, and the discussions are really respectful and listening to many opinions and many feedback from all respective user-base. We don't come all from the same area. I think Google has, as we discussed, a lot of expertise, thanks to App Engine, and later Cloud Functions, and Kubernetes or course. But some partners are bringing a different perspective on things, and that's how we all reach consensus and build Knative. This is happening today, and all of these is happening in the open. The recordings are on the internet and everything.

Now, to your question. Do we need a foundation? I think this is more – You should not start with that. Start by building a successful open source project. Start by building a community. Start by building partnerships with very strong partners. Only then you think about, "Okay. So now how do we make that sustainable in the long-term?" I think it's always too risky to think more in terms of CNCF instead of actually what problem are we solving here. Maybe it just doesn't solve any problem and we don't even need to ask us this question. So we really want you to solve your problem with Knative first.

[SPONSOR MESSAGE]

[00:36:30] JM: DigitalOcean is a reliable, easy to use cloud provider. I've used DigitalOcean for years whenever I want to get an application off the ground quickly, and I've always loved the focus on user experience, the great documentation and the simple user interface. More and more people are finding out about DigitalOcean and realizing that DigitalOcean is perfect for their application workloads.

This year, DigitalOcean is making that even easier with new node types. A $15 flexible droplet that can mix and match different configurations of CPU and RAM to get the perfect amount of resources for your application. There are also CPU optimized droplets, perfect for highly active frontend servers or CICD workloads, and running on the cloud can get expensive, which is why DigitalOcean makes it easy to choose the right size instance. The prices on standard instances have gone down too. You can check out all their new deals by going to do.co/sedaily, and as a bonus to our listeners, you will get $100 in credit to use over 60 days. That's a lot of money to experiment with. You can make a hundred dollars go pretty far on DigitalOcean. You can use the credit for hosting, or infrastructure, and that includes load balancers, object storage. DigitalOcean Spaces is a great new product that provides object storage, of course, computation.

Get your free $100 credit at do.co/sedaily, and thanks to DigitalOcean for being a sponsor. The cofounder of DigitalOcean, Moisey Uretsky, was one of the first people I interviewed, and his interview was really inspirational for me. So I've always thought of DigitalOcean as a pretty inspirational company. So thank you, DigitalOcean.

[INTERVIEW CONTINUED]

[00:38:38] JM: One engineering area that we have explored on all of these different episodes about serverless is the "cold start problem", the problem of, "I've got some code I want to run in something that's serverless, and I need a machine to be provisioned and I need the code to be loaded on to a container on that machine."

In the early days of the "serverless world", the cold start problem was brutally slow for many applications and it was too slow. I understand it's gotten a lot better across the industry for a variety of reasons. Can you tell me in your mind the history of the cold start problem when it comes to serverless functions and what innovations have happened so far and what room there is for improvement?

[00:39:26] SG: So we have to define what it is first. So cold start is when there is a request or an event that happens that needs to reach your service or function, but this service or function

has been scaled to zero. So the first thing that needs to happen is an instance needs to be scaled up so that it can process this request.

As a consequent of scaling of this instance, your function, or service, or code, actually does things when it starts. It's not ready to process request right away. It needs to load all of your modules, or maybe do some initialization before actually being able to process that request.

So we've seen cold start problems often on function as a service kind of products, because they have what we call a concurrency of one. Let me dig a little bit more into that. So Lambda or Cloud Functions, each, let's say, instance of your function, functions are automatically scaled to many, many instances if needed. But each instance of that function is going to be able to process only one request at a time. So that means that, yes, of course. The first request will hit a cold start. But if a second one happens before the first one has finished, then this other one will also hit a cold start, because a second instance needs to be scheduled now.

On Cloudrun, the concurrency number is not set to one, but it's set to 80 by default, and you can change it as much as you want. What that means is that this very first request, yes, might hit a cold start, and I can come back to that. But the second one, if your code is designed to handle multiple requests at the same time, which honestly a lot of frameworks are helping you to do that and a lot of languages are actually built to do that. Then this same instance is going to be able to process many requests at the same time, and as a consequence –

[00:41:23] JM: Wait. I'm sorry. But the first iteration of this world, you said with that concurrency of one, like I spin up a serverless function. If that thing is running when another request for the same function is called, it's going to hit another cold start. It's going to have to spin up another container. Why don't those requests get co-scheduled on to the same function, the same container function?

[00:41:50] SG: So the thing is the platform providers that offer function as a service have, by choice, decided to add that restriction for the simplicity maybe of the billing model or the simplicity of –

[00:42:04] JM: And that's public or just by practice, you're saying? Is that public that they say they do that?

[00:42:09] SG: The bidding model we offer for Cloud Functions could only be offered with a concurrency of one, because you are paying for execution time. So, of course, if we are going to send two requests on the same instance, you will be basically paying for only one request.

So Cloudrun goes a little bit deeper into its pricing model, where it's actually making you pay for the exact CPU and memory and number of request that you use. So with that billing model, you can now decide, "Okay. Now I decide to send one request at a time," and now you get the functions as a service kind of cold start problem, and also billing model, or you can decide, "You know what? I will actually optimize my code, because maybe my code does an API call to another API. Until the API returns, my container does nothing, but I pay? Maybe that's some CPU I should use to process a different request?"

So this is actually better software practices to have code that can be multi-threaded, or like Node.js have an event loop that is able to process multiple requests at the same time. I put aside to be one piece of the problem, which was the very first request that comes in. This one, even if you have a concurrency of 80 or 1, if there is no instance scheduled, this one will still hit a cold start.

Here, there's also a bit of responsibility on the developer side. When you deploy to a serverless environment, it's also your responsibility to pick the libraries and tools in the language that will not take 30 seconds to start. So, for example, minimizing the amount of modules that you load at startup time, or if you initialize some global – I don't know, some assets, don't load them asynchronously. Those kinds of things will put them in your container.

[00:43:55] JM: Okay. I kind of like what you're saying. This is like an encouragement for people to slim down their services.

[00:44:00] SG: We've seen a lot of improvements just by – In Node, just by reducing the amount of files that I'll write in-memory or those kinds of things. Yeah, that's roughly one advice for the very first cold start.

Then one thing that people should know is that Google, when we schedule your container instances or function instances, we don't directly scale them to zero as soon as the request is over. Because, well, maybe we don't need to get that capacity back. As I said, we have a very large capacity where all of the Cloud Functions are running. So until we actually need it, we will maybe keep your instance here for some time.

So that's why if you send one request and then one minute later you send another request, this other request, even if it's technically a new one one minute after the other one or maybe 10 minutes after the other one, this new request might not hit a cold start, because we actually – The platform provider or auto scaling algorithm decided to keep your instance here until it was really sure that it did not need it anymore.

So it's not every request, every new request that comes in, which will hit a cold start. It's really one where, "Okay, you function has been scaled down, because it has been inactive for a very long time." In that case, you as a developer, can may be do a few things on the startup of your code to reduce the time that it takes.

[00:45:24] JM: Now, I could talk to you about scheduling for the entire length of a podcast episode, and there's lots of improvements. Scheduling is this bottomless problem. There's also the hardware level or the VM level. There have been other people who are focused on serverless who have recently focused on the VM level. Making some open source VM innovations that are related to having a platform, a hypervisor level platform as far – I haven't looked at that project in detail. A hypervisor level platform that is optimized for running these functions as a service. Why is that? Why is there space for innovation at the lower level? I guess the thing that's sitting – Like the hypervisor level, for example. Why would the hypervisor level be something that we would want to modify based on, "Okay, this is a server that's dedicated to running functions as a service or containers as a service."

[00:46:20] SG: It's all a question of startup time. How fast can you bring up a container instance ready to serve?

[00:46:25] JM: So we have basically the same cold start issue at a lower level.

[00:46:31] SG: At a lower level.

[00:46:31] JM: Interesting.

[00:46:32] SG: Google with gVisor is taking a different approach, where the VM is here but the isolation is performed in-user space. I encourage people to read at the gvisor.dev.

[00:46:42] JM: I'd love to do a show on gVisor, by the way.

[00:46:44] SG: Yeah, and I can put you in contact with the people working on it. It's fascinating. It's not only used by Google serverless infrastructure. It's used in many different places at Google. The interesting part is that it passes the security review of Google, which is why we are able to run customer code on our infrastructure in a secure way.

[00:47:07] JM: Right. Actually, I think somebody else said this. You don't really use VMs. You just have containers running on raw infrastructure, right?

[00:47:16] SG: Maybe Brian Grant would have been something better to answer this question.

[00:47:18] JM: No, I think that's literally who said it. He's like, "Yeah, we don't –"

[00:47:21] SG: Yeah, exactly.

[00:47:22] JM: We don't use VMs. I was like, "What?"

[00:47:23] SG: I know you interviewed him. I mean, probably, he's way more knowledgeable than I am.

[00:47:26] JM: Okay. I'll ask him about that.

[00:47:28] SG: Maybe the podcast can maybe even answer this question, the podcast you did with Brian.

[00:47:31] JM: Yeah. I realized this might be – Okay. Yeah. Totally. That's fine. Also, we learned about scheduling from this experience.

[00:47:38] SG: I mean, to me, on the Kubernetes cluster with Knative, you have more control. For example, if you want to remove the cold start, well, you can set the min instance to one and, sure, you don't have any cold start anymore, but just know that you always have an instance running your service. That's actually a flag that we gave on App Engine for users who want to constantly pay for one instance. You can do so.

So what I realized is it's a complex problem, many different approaches. From a product perspective, the hard part is how much control do you give to the users? Because, I mean, if I want to optimize my scheduling of my, let's say, App Engine, I have, okay, a number of concurrent request. I have memory, or the instance size by itself can make my instance able to process more or less request. There is the CPU utilization targets. There is the throughput utilization target. There is the minimum instances, max instances. So many, many parameter.

[00:48:39] JM: Cost, priority.

[00:48:40] SG: Exactly. So at the end, for our customers, it's exactly what you said. It's a balance between optimizing for costs or optimizing for performances, because you can always schedule large amount of instances and have amazing latency, amazing performances, but then your cost will be higher.

So on App Engine, we give a few more knobs to tweak to help you choose between this or that. Actually, this depends maybe on the workload you are running. At the end of the day, if you do some background job, then maybe you don't care about the latency so much, as long as the job is done. On the opposite, if you expose some frontend API [inaudible 00:49:20] probably care that it is quite snappy and able to answer a load that varies overtime.

[00:49:28] JM: As we move towards a world in which more people are running their applications on "serverless infrastructure", does it change the requirements we need from storage systems?

[00:49:42] SG: Oh! So, for sure, you should not store state in your container.

[00:49:46] JM: Yes. Okay. All right.

[00:49:47] SG: We want to state the obvious. But the depending on the type of data you want to store, you will find many different storage options, some of them that we qualify as serverless. If I take, as an example, cloud file store, you put nothing it. You pay nothing. You put as much as you want. It's scales with you. You pay for usage.

So we've seen those scalable data storage solutions that are actually very well-suited for serverless workloads. Of course, we allow you to – I mean, on App Engine and Cloud Functions, to use a SQL instance if that's what you want, but just know that, well, if your function is scaled up very fast and maybe you will exhaust your number of connections to your database. So these are problems that if you use a serverless data storage solution, you don't have anymore. But this is maybe structured data, but unstructured data, we all know about Google Cloud Storage or blob storage solutions. Then if you want to do some – As always, depending on the use case, you want to do some analytics, maybe not a little bit less real-time, then use some data analytic solutions. BigQuery is amazing. I use it very often as a product manager to process data, and this is a different use case and something that does it very well, and also scales on-demand and on usage.

[00:51:06] JM: So I love BigQuery. I love Firebase. I don't really – Personally, I don't care that much if something is open source or not. I'm more of a product kind of person. For me, it's all a means to an end. That said, everything about the multi-cloud and the open source, I get why some people need it, and there definitely are people that do need it, and I'm happy that this migration towards like open source and cloud, or this kind of a merger between those two, or both the areas of open source and cloud are expanding, but perhaps there's more and more overlap between them in the Venn diagram.

What's cool about working at Google, from what I understand, is that to some extent, you're going to see the future of what's going to be available in cloud providers. Because if you work at Google, you have all these infrastructure that's really taken care of for you. If you want to deploy

a service, if you want to embed a machine learning model – Okay, the machine learning model is already trained for you. It's just like an import statement. You're like, "Really? Recommendations are that easy? It's just like I import and then it's like two lines of code and I've got a recommendation system out of the box?" I know that's what it's like to be at Google. We don't have that in the cloud yet. I mean, maybe we do to some extent. It's probably more lines of code if I want to have my own recommendation system using cloud tools today than what you have within Google.

What I'm excited about in these cloud conference, I'm just like looking and I'm like, "I want the new tools. I want the highest level, craziest API that I can get leverage from." What's in the future? What are we going to see in five years? From your time looking around at Google, what am I going to be able to do as a developer with public cloud tools that I cannot do today?

[00:52:51] SG: So I think today, first of all, we already introduced something quite new. When I joined the serverless team and I realized, "Wait, we have the infrastructure to run arbitrary containers on all serverless infrastructure in a serverless way, literally scaling up and down on demand." We should give that to developers.

There's not necessarily the requirement to ask them to write in a given language, because if we are able to execute containers, let's just give them containers. That's already what you've seen today has moved one step forward. Literally, we went from a set of five languages, seven languages, to any language you want, any software you want, put in a container, which by the way is an industry standout and now is run on the same infrastructure as what your Cloud Functions were running on. So I think I'm very excited about this product, because we have introduced something new here, more than just functions. It's containers on a serverless infrastructure.

For developers, I think – As you said, bringing more of the Google goodness to the internet world is – Internet is actually our direction. You've seen it with Tensorflow, for example. Google, from the beginning, decided to instead of building a proprietary internal machine learning framework, decided from the beginning to open source it. This is very different from before. If you look at Hadoop and MapReduce, before Google was publishing the paper and leaving the world to take care of the rest.

Here at Google is now getting the implementation of things. I mean, you mentioned – Can I just add an impulse statement and have a prediction model imported? Well, that's what we see today on NPM and Tensorflow.js. So right now, I did it myself. By importing one node module –

[00:54:41] JM: So it is that easy.

[00:54:43] SG: Yeah. With one node module, I imported – What was that? Like face detection algorithm build with Tensorflow that was running in my browser. So we are reaching that level of, "Yeah, sure. I want to run a Tensorflow model. I just important it as a backend, because why not?" This is something that in the past we would not have believed possible, but today, thanks to open source – Actually, thanks to something that become standards, we are able to share those pieces together.

On the Docker ecosystem, people are sharing base images. On the package ecosystem, people are sharing modules, and this is – To me, it's not only about open source. It's about agreeing on a way to do things. If you agree on doing containers and – Okay, let's all support containers.

[00:55:30] JM: Let's all do it.

[00:55:32] SG: It's all better.

]00:55:32] JM: We've got better things to worry about.

[00:55:34] SG: Exactly.

[00:55:35]JM: Steren, thanks for coming on the show. It's been a real pleasure.

[00:55:37] SG: Thanks, Jeff. Thanks for having me.

[END OF INTERVIEW]

[00:55:43] JM: GoCD is a continuous delivery tool created by ThoughtWorks. It's open source and free to use, and GoCD has all the features you need for continuous delivery. Model your deployment pipelines without installing any plug-ins. Use the value stream map to visualize your end-to-end workflow, and if you use Kubernetes, GoCD is a natural fit to add continuous delivery to your project.

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

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

[END]