EPISODE 1316

[INTRODUCTION]

[00:00:00] ANNOUNCER: Serverless computing is a cloud computing solution that lets developers deploy applications to containers without managing the servers themselves. Servers and resources are provisioned automatically, pay only for what you use, and experience little or no errors or downtime. Google Cloud Run is a managed compute platform that enables you to run containers that are invocable via requests or events. Cloud Run is serverless, transparent and easy to use. Common use cases include web services like REST API's, data processing, automation, and modernization. In this episode, we talk with Steren Giannini, Senior Product Manager at Google.

[INTERVIEW]

[00:00:41] JM: Recorder. Welcome to a new kind of recorder. **[inaudible 00:00:44]**. So I think we're live right now, by the way. Right. You're recording this, right? This is great. Okay, so this is like a perfect example of why I love Google. We could just start with just me being in love with Google. Like I am an Android guy. So I'm an Android guy through and through. I used Chromebook fairly seriously for a little while. But, I honestly, my biggest issue is that my digital audio workstation doesn't run on a Chromebook.

[00:01:14] SG: Yeah, I'm also doing some video editing on the side. And I would not do professional video editing or audio editing on a Chromebook even if there are some websites that are starting to use Wasm to provide some very, very powerful video and audio editing. However, for all of the rest, my day to day work, including development, Chromebooks are fine. So that's, by the way, you can either use the Linux, Chromebooks on the full Linux box, or I often code in the cloud with Google Cloud Shell or even GitHub Codespaces. These are great.

[00:01:51] JM: Okay, Wasm. Wasm is a great example of a technology that if I was in a technology management position at Google, I would have no idea how much to invest into Wasm, because it's so complex, right? And there's so much opportunity there, right?

[00:02:06] SG: So Wasm is very interesting tech, because it started from the world of the web frontend wanting to build web apps using not JavaScript, but using languages that could transpile into a binary, then it would be executed on the client. But we start to see Wasm being used as well now. You can you can extend Envoy, a proxy using WASM. I think some CDN companies are starting to offer like programmable edge using Wasm. So it's very interesting. This tech is very interesting. It's still very early. Like when I tried it, the ecosystem is very, very early, but it's something to watch.

[00:02:49] JM: Have you seen the company Gloo? G-L-O-O?

[00:02:53] SG: No, I don't think so. So there's a service mesh company that's doing Wasm modules. They've been talking about this for years now. Wasm modules connected to a service mesh. I don't understand it, to be honest. It sounds forward looking. But I don't fully understand it.

[00:03:11] SG: So it's probably because Envoy is a very popular piece of software used in service mesh. And you can extend Envoy notably using Wasm. So they basically picked the Wasm runtime and stand out as the extension mechanism, which is very smart, because there is a big ecosystem being created around Wasm so that you can transpile – You can compile to was I'm from your favorite language. I think things like Go, Rust and assembly scripts are pretty easy to transpile into Wasm. Other languages are still experimental.

[00:03:47] JM: And with regard to the service mesh stuff, so Istio looks like it's trying to be the Linux of the service mesh. So it almost seems like when service mesh started to become a thing, Google was the first company to articulate that service mesh is such a big platform. It's basically a platform that is of the scale of Kubernetes. And, therefore, Google was really trying to make a bet that was almost like a Kubernetes-sized bet into the Istio service mesh narrative. What would you say that's accurate?

[00:04:23] SG: I was not involved enough to be able to comment. I think Google, the same way Google was using containers for a while before Docker, Google was also relying on service meshes internally. And then when container became a thing and Kubernetes became popular, the next logical step was, “Okay, let's now bring service mesh to the world the same way we've

been using them internally." Istio is a great piece of software. I know there are other service meshes, but I'm not the best person to comment on them.

[00:04:55] JM: Got it. So I think probably the domain where you are best equipped to comment is the Cloud Run domain. Would you say that's accurate?

[00:05:02] SG: Exactly. I'm Steren, I'm the Product Manager Lead for Google Cloud Run, something that we had the chance to talk about the two of us two years ago when it launched.

[00:05:14] JM: When I think about Cloud Run as a product identity, I think of it as the best answer to the cold start problem. Essentially, you want to spin up infrastructure, we're going to write size that infrastructure for you. We're going to let you spin it up. And we're going to take away all the headaches associated with getting that thing started. Is that right?

[00:05:38] SG: Yeah, compared to Functions as a Service, it's true that Cloud Run addresses better the cold start problem. So to recap, Cloud Run auto scales container images based on certain factors, like incoming number of requests, as well as CPU utilization. So in some sense, you have nothing to worry about, like no cluster, no infrastructure management. So this is why we qualify it as serverless.

However, big players in the service space are also the Functions as a Service category. And here, in addition to this auto scaling, and fast rapid scaling and scale down to zero capabilities, they also force you to write your code in the shape of a function, which makes a lot of sense when you want to do one thing reacting to events. However, another characteristic that come from that restriction is the fact that the function instance will always process only one event at the same time. Therefore, if you go from zero event to like a sudden, being sent at the same time, that means that the infrastructure has to spin up a sudden function instances to absorb those events.

And on Cloud Run, it's a little bit different, because you can process multiple events or multiple requests at the same time on a given container instance. And as you know, many software, many programming languages do very well with what we call concurrency, Node.js, Go, Java, they all have been designed to handle multiple requests at the same time on the same process

or container. So that is one reason why cloud run addresses better the cold start problem than Function as a Service.

Now, since we talked two years ago, we've added to Cloud Run the ability to just say, “Hey, give me some warm instances pretty wild.” So basically, customers can say, “I want two instances always warm. I will pay for them a little bit less than when they are processing requests. But I want them to be here so that I don't have this zero to one cold start.”

And lastly, we've been constantly improving our infrastructure. So when we launched two years ago, if you had N instances and suddenly we decide that you need N plus one, we were starting this instance, this N plus one instance, but we were also sending traffic to it right away. So that means that if it took like 30 seconds to start and we see customers having container images taking 30 seconds to start, notably around Java, spring boots, those kind of things, then you might have had a request that was waiting for 30 seconds to be processed. So what we changed is that now we wait for your container to be listening for events or listening for requests before we send anything to it.

[00:08:44] JM: I've done a number of interviews over the years with the Heroku team. And do you know anything about how Heroku does like instance pooling and stuff?

[00:08:55] SG: So I've used Heroku in the past 10 years ago. I was really a big Heroku user. I built a startup on it. However, I've never dig deep into how they do auto scaling. I think there are many strategies for auto scaling, even the one that are used in some other clouds or even in open source projects like Knative, they are different from the one we use in Cloud Run. So I think everybody is using a specific source for auto scaling. I do not know which metric Heroku is using to scale out or in the number of instances.

[00:09:32] JM: So Heroku is this big optimization problem, because they have to acquire enough infrastructure to be responsive with the various peaks and troughs in demand. And their whole margin structure depends on how well they can do that. It's kind of crazy. Like they just have to acquire lots of instances, slice them up into pools of dynos. I think, literally, their businesses like acquire EC2 instances, slice them up and warm them properly, and then build a lot of technology around that scheduling problem. And then like doing that internal scheduling

exercise has been just a continuous problem at Heroku. It's like Heroku is one of those things. It's like Dropbox, right? Where people look at it and they say like, “Hey, why isn't the product like advanced over the years?” And the thing is like, actually, the product has advanced. It’s advanced. So like they build things like Heroku Kafka, for example. It has advanced on that tier. But they basically have to advance pretty slowly because the problem that they're solving is so big and immense and like categorically hard that they really have to be kind of careful about it, right? Because they're the first layer to cloud provider. Like they're the first successful layer two cloud provider, and they don't really know how to manage it properly, because there's no like well-trodden way of doing this.

[00:10:59] SG: Yeah, that's very interesting, because what you described here is basically what we have to do every time we open up Cloud Run in a new Google Cloud region. So as of today, Cloud Run is available everywhere. Google Cloud has a data center. When we met two years ago, Cloud Run was available in maybe one region or four, I forgot. But today, it's like more than 25. And anytime we open up a new region, well, we do have to pre-reserve a number of capacity, like a certain capacity, because the promise we give to Cloud Run users is you will deploy to Cloud Run and then you will scale out potentially very fast if you need to.

So we, at Google, need to pre-reserve some machines on which we install basically the cloud run infrastructure, and we reserve for Cloud Run. So yeah, I can tell you that anytime we open up a new region and when we have zero customer, the margins are not so good, right? But the whole thing is about, over time, customers adopt this region. And it is sized appropriately so that it can absorb a peak and it can be automatically resized if needed. But, really, like the promise that we give to customers of faster scaling, it's not fast scaling to infinity. There's nothing like that. Actually, in Cloud Run, if you want to scale to more than a thousand instances, you have to click a button to increase your quota, because we want you to talk to us in that case, because we want to walk alongside the customer to make sure that we have enough capacity in location where they want to scale out to meet their demand. That's why if you want to go more than a thousand, you click a button and you talk to us.

And that's not a problem, because the infrastructure we are using is very close to the one we've been developing for App Engine, which was released maybe alongside Heroku a long time ago. And so Google has a lot of expertise into this rapid automatically scaled kind of products.

[00:13:10] JM: With Heroku, you have this economies of scale thing, where the constraints of the dyno, the idea of the dyno is their unifying abstraction that they can play with. And then they have this big instance pool that they're allocating across the entirety of Heroku. With Cloud Run, is Cloud Run more like – Well, Cloud Run is the Google Cloud implementation of – What's the open source thing? Open source –

[00:13:38] SG: Knative is the project –

[00:13:40] JM: Knative. Right. Right.

[00:13:41] SG: Yeah. So let me maybe recap for the audience the relationship between Cloud Run and K native. So two years ago, when we launched Cloud Run, we already had launched Knative, which is an open source implementation of basically a container auto scaling system on Kubernetes that makes it basically build your own Heroku on top of Kubernetes. If you use Knative in your Kubernetes cluster, you basically have a Heroku, except you don't have the build step. With the build step, you can easily do it using other open source pieces.

So Cloud Run does not use the Kubernetes cluster or the Knative open source project. But Cloud Run offers portability with the Knative API spec, because the Cloud Run API actually implements the Knative API specification. So that means that if you use Cloud Run, the resources that you create, you can copy paste their YAMLs into a Kubernetes cluster with Knative installed and you will have the same thing running. So this is a relationship between Cloud Run and Knative.

And yeah, to come back to Heroku and dynos, I think, overall, you should think about Cloud Run as a system that auto scales container images. And when they are auto scaled, they are auto scaled as container instances. So a number of auto scaled container instances will be similar to a number of dynos on Heroku.

[00:15:16] JM: And so if I spin up my own Knative system, does that mean that I'm assuming the responsibility of doing some of the scheduling? Or like how much configurability is there to

that like pooling system? Or is there a pooling system in Knative? Like the instance pooling kind of thing? Or how does that look?

[00:15:36] SG: So the idea is that Knative offers the same in cluster auto scaling as you would get if you were to use Cloud Run on Google Cloud. So the idea is that Knative will auto scale container instances quite fast depending on the number of requests they receive, or the number of the CPU utilization. I think you can even customize the exact metric you want to use to auto scale in Knative. And all of those container images that are auto scaled, they implement what we call a service. So a service has a unique endpoint and a service is auto scaled by the system. So the number of container images, or the number of container instances that a certain service will have depends on the number of requests this service receives, as well as how busy the CPU is for the existing container instances of that service.

So, as I said, the implementation in Cloud Ron is different from the one in Knative, but the idea is the same. A service can scale down to zero. A service, when you create it, you give it a container image. And then the system will auto scale those container instances to handle the incoming load. On Knative, of course, you are limited by the capacity of your cluster. On Cloud Run, it's a little bit different, because it's multi-tenant fully managed system. So the only limit that exists is the limit of the capacity of Google Cloud Run in the given region you are in. And in that case, it's usually way bigger.

[00:17:17] JM: I'm really curious about how you do the open source versus closed implementation. Do you look at Knative is a reference implementation for Cloud Run? Or just like a vague inspiration? Are you actually using code from Knative in Cloud Run?

[00:17:35] SG: So we made sure to respect the same API specification. So Knative is both an API reference as well as a reference implementation. And Cloud Run really implements the Knative API specification, but does not use the Knative implementation. This means that we had to collaborate with the open source community on this API standard. In the very early days of Knative, the Cloud Run team was very involved into defining that API specification. And the implementation itself, I think we might be sharing maybe a little bit of tests, and maybe some ways to generate API clients. But the actual implementation for the auto scaling system is not shared. It's two different implementations of the same API specification, which I think is very

healthy, but they are not like 100% matching. Like some things are supported in Cloud Run and some things are supported in Knative, but the majority of the contract stays very, very much the same.

[00:18:50] JM: Fascinating. So in productizing this, what I think is interesting, you have an interesting product development process because you're offering a kind of abstraction that doesn't really exist. I mean, there are parallels. So the closest thing is like what Amazon ECS or Azure – What is it? AKS? Is that what they have? The Azure – Or ACI. Sorry. ACI. ACI. Azure Container thing, whatever.

[00:19:24] SG: So we have to mention that Amazon has its own infrastructure for basically containers without VMs. And they gave it the brand Fargate. But Fargate is basically a backend for the container service, EKS, or, I mean, they have two container services. So EKS or ECS. And as you as you said on Azure, the player is actually ACI, which is also containers without VMs. However, interestingly, that was the answer I would have given you to the question a month ago. Very recently, Amazon released something else, which is AWS App Runner, which is very much similar to what Cloud Run is. So it brings – In addition to the containers without VMs, it brings the developer abstractions that Cloud Run has been bringing for the last two years.

If you have used Cloud Run, you will see that the only thing you need to provide is a container image. And actually, in the past two years, we've even worked on making sure we can easily build that container image from a Git repository, or even deploy your local source code into a container image and then gets deployed to Cloud Run. So is in addition to the fundamental capability of auto scaled containers without VMs, there is also a developer productivity value that Cloud Run is bringing to table. And AWS has been answering to that by releasing a new a new product, which is called App Runner.

But, really, like if we come back to Cloud Run, that it is really a piece of infrastructure as well as the value of productivity that you get when you use Cloud Run. And we know that we are doing well here because we get tremendous feedback from the community online, the developer community that when they use Cloud Run, they deploy and they don't think about it anymore. But also, internally, we measure the satisfaction of the product in absolute and also relative to

other Google Cloud products. And Cloud Run really is always at the top of the satisfaction of our customers.

[00:21:38] JM: Your description of the AWS stack reminds me of why we need multiple big cloud providers, because the AWS design is so definitively AWS-y, that it's like I don't like the fact – I wouldn't like a world where that was the only option, because basically the AWS stack is like, basically, we're going to like Amazonize all these different abstractions and make them basically firewalled from you in terms of like maximal configurability. And I just feel like – Or even at a minimum, we're going to offer you the Amazon flavored containerization experience, which maybe that's what you want if you're fully bought into the AWS like event-based, like event operating system for your distributed architecture. Maybe that's what you want. That's fine. But like I also want the option to have something that's built off an open source reference implementation, plays nicer with a wider array of paradigms, which is kind of how I view the Google ecosystem. I don't know. Maybe you could tell me like ideologically how you're thinking about this stuff.

[00:22:50] SG: So I can tell you that from the early phases of the product design of Cloud Run, we took as a guiding principle openness and portability, because we've been hearing the fact that customers felt locked in when they were using App Engine or Cloud Functions. And that is true. For the last 10 years, App Engine customers, they have benefited from proprietary SDKs that they cannot get outside of App Engine. So on Cloud Run, we said, “Okay, you know what? What you're going to deploy to Cloud Run is going to be a standard container image, like a Docker image, or an OCI container image. So first, the deployment artifact has nothing proprietary. There is no proprietary API to implement. There is no SDK you have to use. Not even a list of languages or language versions. No, no, no, as long as it's a container that respects a very small contract, which is that it has to listen on a certain port, then you're good to deploy it to Cloud Run.

So you first have the portability of the container. And by the way, I think the industry has evolved to standardize on the container. And that's a good thing. Like regarding cloud provider portability, today, if you have containerized your workloads, it would be much more portable between clouds than it was before containers, right? I think this is an amazing thing that the industry has agreed upon the packaging format, right? And then we are building many tools

around this packaging format, the build tools, the CI/CD tools, the security scanning tools. They all have standardized on the container image. So that's already a tremendous win for the industry. And that's why very early in Cloud Run we said, “You know what? It's a no brainer. We are going to accept container images as the deployment artifacts.”

But we haven't stopped here. As you said, we also wonder, “Hey, it's not only about the container. It's also about all of the config around that container. And maybe it's also about the behavior of the platform. And so that's why we went the extra mile and we actually started the Knative project to let customers know that in case they want to take out, they can take out their containers. But it can also take out into a reference implementation that will basically do the same on their own infrastructure. So that was a very strong message that we invested in very early as we were developing Cloud Run. And lastly, I think, to step back and not only talk about Cloud Run, I think the cloud provider portability really improved things to Kubernetes, right? We have many customers who really value the fact that the Kubernetes control plane is available on AWS, is available on Google Cloud, is available on Azure, can be deployed on-premises. And the control plane will be stand out. The resource model will be stand out up. And so that I think is also a win for the industry, right? The discovery of containers as the packaging format, but also the discovery of this joint API specification, which is the Kubernetes style API, which we call Kubernetes resource model. I think this is pretty strong that the industry has agreed to support these across clouds.

[00:26:15] JM: Not to take you off the ball, but just to come back to WebAssembly real quick. Is there any chance that the WebAssembly module is the next container as like the deployable artifact of choice?

[00:26:29] SG: Yes and no. Yes in the sense that you’re write, that Wasm binary is very portable. The Wasm defines the standard binary format that executes on every platform. Well, that reminds us a lot about containers, right? That being said, the fundamental difference is that to containerize an existing software, you basically have to put all of its dependencies into a container, and that's good to go, especially if this software was already Linux-based. To transform into Wasm binary an existing software, most of the time, you cannot do it without like massive changes in its code base or its dependencies, right?

Yes, it is easy for something that is purely code written in a certain language. But as soon as you want to create a Wasm binary out of something existing that has a lot of dependencies on a lot of language modules or even operating system modules, then you cannot like package this into Wasm. No, no, that's not how it works. You basically have to compile the code into a Wasm binary. And if the tools aren't there to compile this code into Wasm, it will be very hard.

[00:27:44] JM: Yeah, but that said, you could imagine a world where all that tooling is taken care of, right? And somehow we just magically get everything compiled down to Wasm and it's deployable anywhere, right? I mean, it seems intuitively – Intuitively, if you could invent a world where this could all become magically a possibility without any work, it seems like a better deployable artifact than a container to me.

[00:28:06] SG: So, yes to what you said, which is that I mentioned earlier that the Wasm ecosystem is still very early. If you give it a try, you will see that it's quite hard to compile to Wasm your favorite language. But I totally expect that to change over time. And you're right, that the Wasm binary, the fact that browser vendors have agreed to support it. Just this, just on the browser side, the fact that we had a binary format that all browsers can execute is amazing. Because now, we talked about audio and video editing in the browser, and then now that's possible. And that wasn't possible before with only JavaScript.

And one step further, now you can – Wasm and Node.js. So Node.js, as you might know, has a lot of native dependencies. Well, we could replace most of these with Wasm. And in that case, they would become very much portable and not rely on any native operating system capabilities, or native packages, right? And FFmpeg for transcoding a video, today, the Node.js FFmpeg module probably uses FFmpeg binary that you get from your operating system package. Well, tomorrow, I hope that there will be a Node.jspure JavaScript plus Wasm FFmpeg package that doesn't rely on any operating system package.

So you see how it has escaped from the browser just with Node.js. And as I mentioned earlier, there is also now the fact that Wasm by itself is a portable format that can be used to extend Envoy to run anywhere. So that's a very interesting nascent ecosystem that we are seeing here. But to come back to containers, I think if you have files and software today, the easiest is to package into a container. What is a container? It's basically a zip file of – It's basically layers

that are zip files and we all agreed on what they contain and how to describe them. And this is much easier to create than creating your Wasm binary.

[00:30:19] JM: I mean, am I thinking idealistically? Or am I thinking – Like is it not our correct feature to even think about this? Because like maybe the container is just good enough, and like let's do everything with containers going forward. And like WebAssembly, basically, should be relegated to making the browser work better and more dynamically. And like let's just not even think about the WebAssembly module as a deployable artifact yet. I don't know. It seems to be like red herring. Like we've already got enough like bandwidth that people are spending on improving the deployable artifact by making it a container. Let's not even think about – Like I just sort of think this whole CloudFlare worker thing, like let's deploy WebAssembly. It seems like it's we're not ready for that yet. I don't know. Am I crazy?

[00:31:07] SG: You're right to say that this is a very interesting packaging format that has very interesting capabilities, because contrary to containers, which rely on a runtime, Wasm can be executed in a much more lighter way and potentially having better characteristics, like cold starts, to come back to the cold start. However, let's face it, like most customers are still struggling to modernize their businesses. And most customers are still in that journey of migrating from on-premises to VMs. Or even from VMs to containers. It's not going to play well if you go and meet a customer and tell them, “Oh, you have to rewrite all of your code base in order to compile to a WebAssembly binary.” It's much easier to say, “You know what? Write is Docker file and you will have a container out of your existing software.” That is a much easier story to tell when it comes to modernizing applications.

Now, I totally agree with you. If you are starting from scratch, and you mentioned CloudFlare workers, and you want to do some lightweight processing at the edge and you are not shy of writing it from scratch, yeah, taking a look at Wasm can be interesting. But most software will need to be migrated in some ways. And this is where the challenge come if you are looking at migrating them to Wasm. But yeah, it's a very interesting tech to watch.

I just said earlier that I think the industry has agreed that containers are the deployable artifact. And this is what we should leverage. Like many customers are still not using containers when it comes to packaging and deploying and releasing software.

[00:32:59] JM: And what are your biggest problems these days? So do you have like – I assume like modern cloud, like Google Cloud has all kinds of interesting problems. There's like marketing problems. There's go to market problems. There're implementation problems. There're engineering problems. I shouldn't even call them problems. Maybe just like tasks to be done. But what are the sort of like highest order impact things that you're concerning your time with these days?

[00:33:25] SG: That's a good question. So I should say, first, that for what it is designed to do, Cloud Run is doing very well. So we see great adoption numbers, great growth for this product. But as I mentioned earlier, it is designed to do auto scaled requests or event services. And that's actually a huge setback. That's just a small fraction of what our customers want to do in the cloud. Not every service has to be auto scaled. Some services actually won't need to be manually scaled, and then you pull some data instead of receiving events or receiving requests.

So what we see here is that Cloud Run is great, receives great new usage, as well as satisfaction scores for what it is designed to do. And what we are looking at now is how can we expand what Cloud Run can do. And things like maybe looking at non-request-driven workloads. Or today in Cloud Run, the CPU is allocated to a container only when it is processing at least one request. But some workloads don't play well with that. Some workloads expect to have CPU as long as the container instance exists, right? So removing those limitations, as well as changing how to use Cloud Run can actually expand the Cloudr Run addressable markets and make it grow even more. So that that's basically what I'm spending a lot of time on. Over the last two years, probably we've really so many things that we are, basically, again, removing the limits of the Cloud Run of two years ago. So a lot of investment has gone into connectivity with VPC networks, or even making a cloud on service internal only. Like when we shipped Cloud Run, the endpoint was protected by IAM, but was still exposed on the Internet. Now, you can completely disconnect the endpoint of the service from the Internet.

We've increased the instance sizes, going from one CPU when we launched, to four and soon even more. Same for memory, we now go up to 16 gigs of RAM. So basically, expanding the limits of Cloud Run as well as expanding what it can do is what I'm spending a lot of time on in order to expand the addressable workloads that you can deploy to Cloud Run, because the

alternative is then, okay, now you have to go back on VMs. And the vision that that I have for Cloud Run is that the same way we've seen physical hardware disappear from our day to day life. And today we only focus on VMs. We don't think about the physical hardware anymore. And we might see the VM disappear from our day to day life and in order to focus only on the container. And that's the promise of containers without VMs, containers without infrastructure, where the unit becomes the container and not the virtual machine anymore.

[00:36:44] JM: And can I just ask, what's your contract, or what kind of services, or can you offer any kind of relationship to the Firebase team, for example? So I love Firebase, right? And the way I see Firebase is like what if Google had its own Heroku? And it does. It's Firebase. It's not exactly Heroku. It's sort of a different approach. It's like what if this thing that we're focused on is actually the database? What if we like look at the database as the zone of data gravity and then we've build infrastructure solutions that are attached to that? I don't think it's necessarily like the best way to look at infrastructure. But it's a way. And I think it's actually really useful for newer programmers, because it gives them a sort of an authoritative, like Rails-like experience as a cloud provider. That's why Firebase is so beautiful. It has safe on-ramps, right? Safe on-ramps to real infrastructure basically, because if you need to get away from the Firebase pricing structure, there's a way to do that. It's not easy. But there's a roadmap to doing it. So I just wonder like when you're looking at building basically the next generation of Google Cloud infrastructure, do you talk to the Firebase team? Or do you just sort of like partition?

[00:37:51] SG: Yes, we talk a lot. Actually, we integrate with Firebase. So let me talk about the integration, and then I'm going to talk more broadly about the Firebase use cases. Today, Cloud Run is about creating services that are container-based. But if you want to do a website, or a web app, then you don't only need the server-side compute. You also need all of the client side JavaScript and maybe the caching layer that Cloud Run doesn't provide. So to get that, you can integrate Firebase hosting with Cloud Run and make sure that Firebase hosting will serve your static content, your JavaScript, your static assets. There will be served by the globally-distributed Firebase infrastructure. And some requests, maybe the ones going to your API, will pass through and be sent to Cloud Run, which itself will auto scale depending on the number of requests. So this integration already exists and existed from day one. And that is how we can nicely address the use case of a website or a web app that needs both very rich clients in JavaScript, HTML, CSS, as well as very powerful backend with any language. It can be Rails, it

can be Python, it can be Java on Cloud Run. So using the two Firebase hosting in Cloud Run works very, very well today. And that's what I use personally for my side projects. I put Cloud Run behind Firebase, because the CDN can also cache the requests sent to Cloud Run. So that means that maybe my API will have a Git endpoint. And this endpoint can be cached on Firebase hosting CDN. So that the next request that comes in won't even reach my container because it can be returned by the globally distributed cache. So that is how Firebase hosting can integrate with risk cloud run.

Now, Firebase as a product suite, because it's a product suite. There are many things in Firebase. I must say, it is more targeting the mobile and web developers. And the great thing about Firebase is that it has allowed mobile developers to store data in the cloud without even needing to write any server. You don't need to create an API. In Rails, for example, if you want to store data from mobile apps into the Firebase real time database, or Firestore, that's great. That's something very new actually. And similarly, for web developers, Firebase offers Firebase hosting, which is a great way to host progressive web apps and just web pages if you want to. And the on-ramp is that, well, a Firebase project is a Google Cloud project. When you use Cloud Functions for Firebase, actually you are using Cloud functions. When you use cloud storage for Firebase, you are using Google Cloud Storage. And if you open up your GCP project, you see all of those things in the Cloud Console. So it's basically a shared backend that presented for a specific target audience and specific use cases of mobile and web development.

Now, we are always working closely with the Firebase team to make sure that those integrations work better. The on-ramp from Firebase to Google Cloud works great. So yeah, we still have work to do. But things already worked quite well together. And people talk to each other. I would correct the fact that Firebase is the Heroku of Google. I think Firebase is much more around static web apps, progressive web apps than server-side code. Like Heroku is about – By default, Heroku was I can run my Rails application without worrying about the infrastructure. Well, actually that is Cloud Run. That is App Engine. What Firebase is providing is the mobile and web hosting solutions as well as an entire suite of products around monetization and advertisement.

[00:42:03] JM: I mean, I won't get into a semantics debate with you. I have my own framing of Firebase. I really like Firebase. I mean, I think they're basically like the same things. It's like

Firebase is just a different window into the same world that you're getting, which is just great. Maybe like Firebase is what Google App Engine should have been all along, or something like that. I don't know, whatever marketing you want to use. I love Firebase. And it's like literally the only thing – There're a few things that I could potentially use instead of Firebase to build a project. I like Supabase. I think that's really cool. I wish Google would acquire Supabase and just make it really clear who wants to own the Firebase space the most. But probably that's not even possible. Render.com I think is really interesting.

But anyway, as far as I can tell, like Firebase is still the gold standard for deployments of like people who want to be hipsters and developers at the same time. And then like Cloud Run is sort of like the next generation for serious businesses. I don't know. I like the way Google does things. It's really interesting seeing the dynamics of the Amazon Google Azure ecosystem. Do you think of this like competitive mindset when it comes to building cloud stuff? Or do you just like think for more of a first principles, like let's learn from what we did at Google sort of thing?

[00:43:23] SG: I must say, if we take a look at Cloud Run, we took a developer first approach and not looking at the competition. Actually, the thing that we launched two years ago when we met, the two of us, did not have a clear competitor. Like Cloud Run, this idea of super easy auto scaled containers without infrastructure. Yes, Fargate existed, but Fargate was basically a backend for a container management system, like EKS or ECS. And Cloud Run was much simpler than that. Cloud Run was much more targeting the developer.

So actually, to do this product development, we really iterated on listening to our user base, which was developers, and kept iterating, iterating on what we believed was a great experience and trying to validate as much depth as we could by doing user studies, gathering feedback from a large group of testers, without necessarily looking at what was as well. Of course, one might take inspiration from great things you can see elsewhere. And Firebase was one of my inspiration. Like if you look at the Google Cloud Run command line, it is quite smooth. Instead of creating errors when you're missing an argument, it's actually going to prompt you to provide the arguments. It's showing pretty well-crafted progress indicators. So all of this, I took this inspiration from the Firebase command line, which is amazing in my opinion. But in terms of competitiveness, for Cloud Run specifically, it was really developed as a way to say, “Okay, you know what? We believe customers like Function as a Service, customers like Cloud Functions,

but Cloud Functions comes at great, great, great limitations." Like when you use cloud functions, you cannot pick your own language. You cannot have multiple requests arriving on the same instance at the same time. And we said, "You know what? Let's reinvent this a little bit. And let's see what could be if we took a container first approach on the same infrastructure as the one that is powering cloud functions."

[00:45:36] JM: Anyway, so much more we could have gotten in depth in. I know we need to wrap up. Anything else you want to talk about, like in the last couple minutes?

[00:45:42] SG: No. Thanks, Jeff, for the interview. You were probably the first person to interview me after the launch of Cloud Run. Literally, I launched in the morning, and in the afternoon we are sitting together. I just want to send to the audience the message that for the past two years, we've been very, very hard at work delivering new features to Cloud Run. So if they had checked it out two years ago, I invite them to check it out again today, because a lot of things have changed. A lot of improvements have landed. A lot of fixes have landed. And yeah, we are always looking for feedback on how could Cloud Run help you better with your businesses. So like maybe you have the workload that doesn't fit on Cloud Run. As I said during the interview, let me know, because I'm always looking at making Cloud Run a better fit for more workloads. So give it a try. We know it has a great developer experience, because developers tell us this, because we measure the satisfaction. And so we are pretty sure that you're going to like it if you give it a try.

[00:46:44] JM: Steren, thank you for coming back on the show. Consider yourself the holder of an outstanding invitation, because I love talking to you about this stuff.

[00:46:52] SG: Thank you, Jeff. It was great talking about Wasm, talking about many other things during this interview. And Thanks for the invite, Jeff.

[END]