EPISODE 355

[INTRODUCTION]

[0:00:00.4] JM: Kubernetes is an orchestration system for managing containers. Since it was open-sourced by Google, Kubernetes has created a wave of innovation in the infrastructure technology space. Another recent innovation has been the serverless execution tools such as AWS Lambda and Google cloud functions. Serverless execution, otherwise known as, functions as a service, allows a developer to execute code against cloud servers without specifying which cloud servers they're executing on.

Serverless execution is a cheap and flexible resource that any large company wants to have access to, but AWS Lambda and the other popular serverless tools are closed-source and they're also only accessible if you're on a cloud provider. This led Soam Vasani to work on Fission, a serverless executor that sits on top of Kubernetes. If you've not heard about either Kubernetes or serverless, you can check out our previous episodes about either topic, which are linked to in the show notes, and if you're familiar with both of these topics, I think you'll enjoy this episode in which Soam explains the motivation for serverless on Kubernetes and the architecture of Fission.

We're going to do another episode coming soon about Cubeless, which is another one of these serverless on Kubernetes frameworks, and it's interesting to see people build on top of Kubernetes after so much talk of it being the distributed systems version of Linux. It's really showing to be of that level of importance and how it's an important building block that other people are creating tools on top of.

Software Engineering Daily is looking for sponsors for Q3. If your company has a product or a service or if you're hiring, Software Engineering Daily reaches 23,000 developers listening daily. I would love to hear from you. Send me an email, jeff@softwareengineeringdaily.com. Thanks for listening.

[SPONSOR MESSAGE]

[0:02:15.4] JM: Your application sits on layers of dynamic infrastructure and supporting services. Datadog brings you visibility into every part of your infrastructure, plus, APM for monitoring your application's performance. Dashboarding, collaboration tools, and alerts let you develop your own workflow for observability and incident response. Datadog integrates seamlessly with all of your apps and systems; from Slack, to Amazon web services, so you can get visibility in minutes.

Go to softwareengineeringdaily.com/datadog to get started with Datadog and get a free t-shirt. With observability, distributed tracing, and customizable visualizations, Datadog is loved and trusted by thousands of enterprises including Salesforce, PagerDuty, and Zendesk. If you haven't tried Datadog at your company or on your side project, go to softwareengineeringdaily.com/datadog to support Software Engineering Daily and get a free t-shirt.

Our deepest thanks to Datadog for being a new sponsor of Software Engineering Daily, it is only with the help of sponsors like you that this show is successful. Thanks again.

[INTERVIEW]

[0:03:39.4] JM: Soam Vasani is an engineer with Platform9 Systems. Soam, welcome to Software Engineering Daily.

[0:03:45.0] SV: Thank you so much for having me here.

[0:03:46.7] JM: Kubernetes is a management platform for containers, and we're going to talk about Kubernetes and talk about serverless. Let's give a little bit of an overview for those who are unfamiliar with Kubernetes. Why is it important?

[0:04:01.5] SV: To give some context, I think if you look at how people used to deploy software before the popularity of containers outside Google at least, there was a lot of language in stack-specific stuff and containers brought this uniformity whether it's Ruby or Pearl or Python or whatever, you can drop it in a container and that's uniformly deployed everyone. In fact, that's why that shipping container analogy was used.

Container solved that uniformity problem but now you needed to do all these stuff to manage container networking — It didn't actually solve how you would schedule a container on to a cluster of hosts and things like that. That's where Kubernetes comes in, and I think you can think of it as something that lets you view the cluster as a collection of fungible compute nodes, and one way of seeing it explained is that you never associate into a machine again. You get a cluster level API and you say, "Take this container image and deploy somewhere."

It's not just a container image, there's all sorts of stuff around it like how many replicas do want, how do you want to expose it to the world, and things like that. How do you want to update it.

[0:05:23.0] JM: Yeah. I think you'd agree with me that Kubernetes solved more problems than it introduced, but it did make some problems more important to consider. You talked about container networking. Kubernetes solved a lot of the deployment issues that perhaps Docker hadn't solved, a lot of the centralized management issues. What are those other difficulties? What are the newer difficulties that Kubernetes users had to encounter?

[0:05:56.3] SV: That's a good question. Let's see. I think this —

[0:05:59.5] JM: We're talking about networking.

[0:06:01.6] SV: Right. On a networking level, Kubernetes I think made a good decision which was all containers are — Well, containers are in parts in the Kubernetes world, but all parts are network addressable. In fact, they initially made this uniformity. Everything connect that's everything network space, but more recent versions have had more pluggable systems where you can isolate, where you can have some level of isolation.

To step back from networking a bit, I think one of the things that at least new Kubernetes users have to figure out is how to take this whole powerful, expressive language that Kubernetes gives you and how to apply it to your problem.

A new user — Let's say you've never heard of Kubernetes, now you have to learn at least may be half a dozen to a dozen concepts before you can start using it, like you have to know — You have to know containers, images, how to create those images, how to host them in the registry somewhere, how to use image text to manage your versions, and that's just before you start with Kubernetes, then you need to know what a deployment is and then you presumably want to expose this software out to the world somehow so you need to learn how ingress works and you want to write up that ingress to your deployment. You can learn the concept of label switch, which is really cool. All of these concepts are extremely powerful and useful, but I think there’s a significant learning curve to Kubernetes, and that's one of its big challenges right now I’d say.

It created — I think it's very complete in a sense, like you can take almost any distributed systems problem and express it using Kubernetes, I think. But to learn how to do that takes some time especially for the new user, and there are use cases that are pretty simple where a user might think, “What do I need to learn so many concepts just to do something simple?”

[0:08:04.2] JM: Okay. We've done a lot of shows about Kubernetes, so if people are not familiar with Kubernetes and they’re still little bit confused, now might be a good time to stop this episode and go listen to some of the previous episodes about Kubernetes, because we’re going to talk about serverless now and then we'll get into serverless and Kubernetes and the interactions between the two. How do you describe serverless or functions as a service to people who have not heard of this paradigm?

[0:08:33.0] SV: I like functions as a service as the term more, because it more specifically describes what we are doing. I think functions as a services as doing two related things. One is that it tries to give you a short-lived stateless compute without having to deal with turning your function into a service and figuring out how to deploy and scale it. It also gives you the ability to run that function only on demand. In some sense, either in a the billing sense or in a resource utilization sense, that function — it makes that function free when it's not being used.

If you’re talking about AWS Lamda, then we’re actually talking about billing only when the function is being used. If you’re talking about something like Fission, then we’re talking about memory and CPU usage only while the function is being used. The aggregate leads you to design your cluster capacity more as a function of how much your functions are being used rather than how many functions you've deployed. It's usage versus deployment size. Essentially, that's another way of saying it's trying to drive up your cluster utilization.

[0:09:50.2] JM: When I run a function on AWS Lambda or Google cloud functions, I'm writing a piece of code and it's deploying against these clusters on Amazon or Google cloud. What's actually going on there?

[0:10:08.7] SV: We don't know for sure, but from the level at which they've publicly spoken about it, at least for Lambda, they talk about having these, again, a pool off some compute resource. It's some kind of isolation environment, either a container with various knob tweaked or a VM. I'm actually not are sure how exactly Lambda runs. But there's a pool of those and one of those is then chosen and a function is loaded into that — Into that entity, a container, a virtual machine, something like that. Then the request is routed into that function, so that function has some kind of wrapper for the language that's being supported. I think Lambda supports JavaScript, Python natively, so each of those probably has some kind of wrapper and then the JavaScript or Python function is called with the arguments that were sent in by either the request or an event.

[SPONSOR MESSAGE]

[0:11:19.4] JM: For years, when I started building a new app, I would use MongoDB. Now, I use MongoDB Atlas. MongoDB Atlas is the easiest way to use MongoDB in the cloud. It's never been easier to hit the ground running. MongoDB Atlas is the only database as a service from the engineers who built MongoDB. The dashboard is simple and intuitive, but it provides all the functionality that you need. The customer service is staffed by people who can respond to your technical questions about Mongo.

With continuous back-up, VPC peering, monitoring, and security features, MongoDB Atlas gives you everything you need from MongoDB in an easy-to-use service. You could forget about needing to patch your Mongo instances and keep it up-to-date, because Atlas automatically updates its version. Check you mongodb.com/sedaily to get started with MongoDB Atlas and get $10 credit for free. Even if you're already running MongoDB in the cloud, Atlas makes migrating your deployment from another cloud service provider trivial with its live import feature.

Get started with a free three-node replica set, no credit card is required. As an inclusive offer for Software Engineering Daily listeners, use code "sedaily" for $10 credit when you're ready to scale up. Go to mongodb.com/sedaily to check it out. Thanks to MongoDB for being a repeat sponsor of Software Engineering Daily. It means a whole lot to us.

[INTERVIEW CONTINUED]

[0:13:21.5] JM: As I understand it, the way that people tend to think about AWS Lambda is Amazon has this big cluster of servers, and at any given time, they've got many of these servers that are allocated towards doing something whether it's EC2 or S3 or it's serving some other purpose. At any given time, they've got excess capacity and something like AWS Lambda can just absorb this excess capacity and be used to run functions. Whether that's true or not, it makes sense to think of it that way because that helps to motivate why it's so cheap. These things are super cheap. There's the cost efficiency of running against the serverless architecture, but there's also the manageability and the scalability advantages to running against these serverless paradigms. Why is serverless useful? To what degree is there cost reduction and to what degree does this actually help developers manage their code?

[0:14:35.0] SV: Those are two great questions. One is about cost and the other is about manageability, right? Let's dive into the costs argument a little bit. The cost of Lambda itself, the benefits come from two sources. One is that your service, it depends on your utilization. If you have very varied utilization and you have long periods where your service isn't used at all, then that's where you're getting billed zero and that's where a lot of your savings come from.

You can try this math. Go to Amazon's pricing page for EC2 and their pricing page for Lambda, and let's say there is the cheapest instance of Lambda is 128 megs, so find the pricing for an EC2 VM and imagine that you're running a Lambda 24/7. The price for that does not compare favorably to the EC2 VM. It is quite a bit more expensive, except for the free tier that Lambda gives you. That's where there are two cost-saving things from AWS Lambda, and we're diving into the details of AWS pricing a bit, but what I'm trying to get at is that the cost savings for Lambda come from being able to not pay when you're not using the service, but they don't work well for you if your service is being used all the time. It's more useful when you have unpredictable usage.

The other thing you pointed out is excess capacity. So there's things like part instances, and on Google cloud they have something out preemtable instances, and that seems to be a way where they try to sell excess capacity in the form of virtual machines, and there's been a lot talk of running Kubernetes on such virtual machines, and that may give you the price benefits for that excess capacity for Kubernetes, and then because those instances can actually get killed, they fit well into the model of running short-lived stateless things, like Fission on it. I wandered a bit, but to answer cost point, I think there is a cost benefit to the hosted serverless things, like Lambda and Google cloud, but you have to do the math for your use case before figuring out how much of a cost benefit there might be. Often, there just isn't a cost benefit.

Secondly, you talked about manageability, right? This is a really growing field. I think as usage increases for serverless, in particular on Lambda, there's been this whole bunch of new products coming up around AWS Lambda and ecosystem of both open-source on hosted services that will help you manage things from versioning and deployment, to monitoring. I think serverless sort of moves your abstraction layer, and you've sort of created some manageability problems for others. Now, for example, if you get rate limited by AWS, you need some sort of monitoring to know why, or you may have functions itself scaling very well, but other parts of the infrastructure is slowing down, so you need some kind of deep monitoring and observation tools data infrastructure to figure out where things are slow if they become slow. Does that make sense?

[0:18:00.2] SV: Yes, it does make sense. We had Mike Roberts on the show and he has written a lot about serverless architectures, and by that I mean how you structure your application such that you can use AWS lambda. He talked about some patterns, like using an API Gateway and these other things that are abstractions that exist from the AWS point of view. This is something build, an AWS abstraction, the API Gateway, there are other things.

No matter if you're on AWS and you're using Lambda or you're on Google cloud and using Google cloud functions or you're on Azure and you're using Azure cloud functions, all of these things have the same property where when you make a request, your code gets spun up on some server thing somewhere, whether it's a server or a container. It gets run, and then that transient entity, that transient container or VM gets spun down.

The question that I have for you before we dive into talking about fission, when people are building on these hosted serverless platforms, like Azure cloud functions or AWS Lambda, how do they manage long-running state, because usually when I'm on Facebook or I'm on Gmail, I leave my Gmail window open in some tab and then I go off and do something else and then I come back to it and I want to be able to continue using my Gmail. The server-side picture of that, it probably looks like a container that's spun up and that's indexed my emails and can serve me information quite quickly.

If Gmail were trying to serve to me using serverless architecture, it's hard to imagine how exactly that would work, what the persistence model would look like, because the actual compute node is getting spun up and then spun down as soon as it has nothing to do. How do people typically treat persistence when they're building on these platforms?

[0:20:14.7] SV: One the big lessons of the whole cloud native event and Twelve-Factor App stuff is that when developing your application, you separate the things that need to persist either in memory or on stable storage from the business logic that actually operates on that state, and the big advantage of separating essentially compute from state is that you can scale those things separately, you can do ruling updates without losing state. Really, stuff like Kubernetes, deployments, and replicate sites and the ruling updates that they support, they kind of depend on this idea of separating out your state.

If you need persistent long-lived memory state, then you simply keep it separately from your serverless functions. For example, you might run Resis either in a long-lived container or in a virtual machine, or you use PostgreS in a VM and it has its own internal caching and queries can be pretty fast, again, depending on use case. The core ideas that you separate persistent state out of your compute.

[0:21:32.2] JM: Okay. Now that we've talked about Kubernetes a bit, we've talked about serverless a bit. Let's talk about serverless on Kubernetes. Explain what your project; Fission, is.

[0:21:45.3] SV: Fission lets you run functions as a service on Kubernetes. We actually started it about, let's say, it was in November 2016 that we first open-sourced it. The reason we started it was that Platform9 to Kubernetes to new users, from the enterprise and open-stack world and things like that, and they were having difficulty with the learning curve for using Kubernetes. We wanted to give them something where they can bring a piece of code, and on their first day of having a Kubernetes cluster, make a useful run on it.

Really, one big goal of ours was making the initial use-case use of Kubernetes really simple, and there's also the idea of basically making resource allocation more efficient. As we've talked about it extensively, the idea of invoking functions only when there is demand for that function.

What fission does is, basically, it uses a Kubernetes APIs. It hides the idea of containers from the user, and Fission gives users only three concepts. One is a function. That's a piece of code in any of the supported languages; there's JavaScript, Python, C#, Go, Java. I might be missing one or two. Then there's the concept of environment, which is the language specific part of that functions invocation, of the functions execution, and there's the concept of triggers which is how a function is invoked. You can have an HTTP trigger, so you can try a function to an HTTP request or you have a timer, and you can also invoke functions from events generated from Kubernetes itself. You can use the function to watch, for example, new services. Anytime a new Kubernetes service is created, that function would be invoked.

Just by learning three concepts, the user can immediately start doing something useful. There's a command line tool and a UI as well and they can write a function, map to it to an environment so they can say, "Okay, the JavaScript functions runs in the JavaScript environment, and then they can map that function to some trigger and it'll only be invoked when the trigger is —When that trigger is activated.

[0:24:03.2] JM: Here, it's worth coming back to the two things that we discussed; cost and manageability. I think particularly manageability, because that is the thrust of why you would want to run Fission on top of your Kubernetes. We talked at the beginning about some of the frustrations of running a Kubernetes cluster. What fission does is instead of having the AWS Lambda or Google cloud functions world where when you execute your code against the giant capacity that AWS Lambda has available and then they do all the scheduling and you don't know it doesn't it's a black box, Fission allows you to do this with your own semantics or at least you know what the semantics of Fission are or at least you can look into it if you want.

You have more clarity on how the — When you write your function to be executed serverlessly, you know how that scheduling works. Give us some more of the motivation for why we would want to self-host disability, because coming back to the cost, if it's so cheap to run our functions as a service on AWS Lambda, and probably they're going to give us a pretty good SLA even if we don't know exactly what's going on under the hood, at least get a good SLA. At least we know it's going take a certain amount of time. Why would we want to self-host the ability to have a serverless architecture built on top of Kubernetes?

[0:25:42.8] JM: That's a totally fair question. Like you said, you get a lot of visibility. Since the whole stack is open-sourced, Fission to Kubernetes to Linux, you can you can examine and monitor at every level when a function is slow.

Secondly, to look at Lambda itself, they do have rate limiting and stuff like that. In particular, rate limiting can be pretty interesting from a user's point of view. The third thing is, again, the cost. If you're using your functions enough, if you're running them continuously, then in fact it's much cheaper to run functions as a service as a pattern on top of EC2 or compute instances from any of the clouds. The other sort of higher-level point is that if you run function as a service on an open-source stack of products, basically Fission and Kubernetes, then you've made something that's portable and you've essentially made the cloud very easy to switch, so you've reduced your lock-in in some sense to any particular cloud. Basically, it's portability, open-source, and depending on use case cost.

[0:27:06.0] JM: Right. Okay. You've successfully convinced me that there is a motivation to doing this, so let's diagnose how you do it. In order to run these functions as a service on a Kubernetes cluster, you need to always keep a pool of running generic pods. These are pods where — By the way, for those who are not familiar with Kubernetes lingo, a pod is an abstraction around a container or contain multiple containers I believe. It's just the abstraction of a Kubernetes unit.

These generic pods, if I write some function that I want to be executed as a function as a service against my serverless on Kubernetes cluster, the Fission, my Fission architecture, my Fission cluster, that's going to get scheduled on one of these generic pods. Tell me more about these generic pods. What are the requirements for these pods?

[0:28:06.4] SV: First of all, why do they exist? To go back to that idea of a fast rate, we want to activate a function on demand because we don't want to use resources when it's not running. However, users want to pretend that the function is always there, so that you don't have to pay much of a penalty for invoking that function, especially if there's a human being waiting for that function. The moment you take a noticeable amount of time, people feel that the app is slow or something like that.

We need to satisfy both goals. We need to invoke the function only when it's in demand and we need to make sure that we can invoke it really quickly when it is required. First, we actually tried a simpler approach which is when does a request put the function in a container image, and then call Kubernetes to invoke that container image and run it as a container on the cluster somewhere.

This, at best, takes a few seconds. While that's good enough for some use cases, it's not good enough for anything that's interactive where you have a person waiting for that function. The next approach was, "Okay, what's the minimum we can do at runtime when a function is invoked?"

We said, "Okay, we need we need something that's already running and scheduled on Kubernetes and we want to be able to drop a function into that at runtime." Now, those running things need to be language specific because they have to dynamically lowered either a Node.JS jazz or a Python or a function of some sort.

For each supported language, we have we have these environments, and the environment basically causes a pool of these parts to be created, and the environment is just a container image that knows two things. It knows how to dynamically lower a function and it knows how to route HTTP request into the function.

Let's say you add JavaScript to your Fission cluster, you add the JavaScript environment, or rather you enable the JavaScript environment on your fission cluster. Then Fission goes and creates a pool of a few of these genetic parts. That part is in a state where it's just waiting for a request from Fission to lower the given function and to route the request into it. That pool of parts is eagerly created when you add that environment to Fission. When you invoke the function, that pool is always available.

[0:30:41.9] JM: As we said, a pod is an abstraction that can have multiple containers in it. Are there are there any sidecar containers the you want to put in that pod along with the container that’s just going to handle the execution of that serverless function?

[0:31:00.3] SV: Yes, we do. A part is basically collocated set of containers, and the advantage of them being collocated is that you can share temporary file system space for example. In Fission, we retry to divide the language specific in the language independent parts, and to make the language specific parts really small, we have a Fission-specific and language independent sidecar with every environment. That sidecar knows how to get a function from Fission's function store and make it accessible to the language specific container using a shared volume and then to book that language specific container to lower the function.

[SPONSOR MESSAGE]

[0:31:53.4] JM: Artificial intelligence is dramatically evolving the way that our world works, and to make AI easier and faster, we need new kinds of hardware and software, which is why Intel acquired Nervana Systems and its platform for deep learning.

Intel Nervana is hiring engineers to help develop a full stack for AI from chip design to software frameworks. Go to softwareengineeringdaily.com/intel to apply for an opening on the team. To learn more about the company, check out the interviews that I've conducted with its engineers. Those are also available at softwareengineeringdaily.com/intel. Come build the future with Intel Nervana. Go to softwareengineeringdaily.com/intel to apply now.

[INTERVIEW CONTINUED]

[0:32:51.4] JM: Yeah, that makes sense. Quick question; let's say I've got my Kubernetes cluster and I've got certain things that I want to continue to do on Kubernetes. I want to have these certain long-running containers across my application, but I now have a new feature that I am building. It's an image resizing feature. This is the classic example that's used for serverless. If I want to get an image resized, serverless is great for that because an image resizing is a well-formed stateless thing, "Okay, I need to resize my image," so a container gets spun up to do the image resizing. It gets done. I get my resized image back and then that container gets spun down or garbage collected somehow.

If I am creating — if I want to create my serverless area of my cluster or of my Kubernetes — Yeah, of my Kubernetes cluster, my Fission area, that does things like image resizing, what's the interaction between the Fission area of my Kubernetes cluster and the rest of my Kubernetes cluster?

[0:34:04.5] SV: That's a great question, and it really brings up the — It sort of brings home the fact that you can mix together microservices and functions as a service patterns easily on Kubernetes. You don't have to have a separate area as such for functions, and you can invoke functions today by HTTP and pretty shortly through a variety of message queues.

Is your question around how you invoke functions from regular services?

[0:34:37.6] JM: It's more around what is the sharing. I've got capacity of X for my entire Kubernetes cluster. If I wanted partition off some of that capacity for Fission, and Fission takes up why, then how am I moving resources between those two resource spaces and how much does Fission take care of it. It sounds like they play pretty nice together.

[0:35:07.5] SV: Yeah. Again, that's the advantage of living on top of Kubernetes, we get to use all of the primitives that they provide. For example, Kubernetes provides name spaces, and you can run all of your functions in one namespace and provide an upper limit for CPU and memory, and that will make sure that your functions never cross that limit, and so your other resources can't have the remaining cluster resources.

To recap a bit, Kubernetes allows you to specify CPU requests and limits. I think, on a per part basis on a per namespace basis, it lets you specify a total quota. You can organizes this anywhere you want. You're going to have your apps and the names — Each app in the namespace or you can divide your namespace according to how your company's organization is and you can then provide resource allocation limits using name spaces.

[0:36:02.6] JM: Okay. I've heard of the cold start problem on serverless systems, and this describes the problem where if I make a request to AWS Lambda or Google cloud functions and it needs to spin up a container containing my code somewhere on the cluster, it takes some time to spin up that container and execute my request. How does the cold start problem manifest in Fission?

[0:36:33.9] SV: Right. We have exactly the same problem. When a request hasn't come in for a given function for a while, the first time it comes in, we have to invoke this function, and that's where that pool of genetic parts is useful. Sufficient deals with cold start by — Let me break up the components of Fission a little bit. There's a router which is where HTTP requests come in, and there's a pool manager which is what manages this spool off genetic parts and knows how to load a function into a part.

The request comes in to the router, the router checks if there's already a part for that function. Since this is a cold start, there's nothing existing yet. It sends a request to the pool manager to create a part for that function that's been requested. The pool manager then chooses a part from that genetic pool. It removes it from the pool and loads a function into it by actually calling that sidecar that we talked about earlier. Once that part is ready, it returns control to the router which then caches that parts address and forwards the HTTP request. It just create an HTTP proxy and forwards the request to that part.

That process, depending on the language and the size of function, takes on the order of 100 milliseconds. Sometimes it can be a lot faster depending on, again, the contents of the function. Then that part is cached for a while, for a few minutes, and the router is keeping track of how often that — Or when the last time was that the part was invoked, that the function was invoked, and if a function hasn't been invoke for a while, then that part is just —The next invocation of that would be a cold start again.

[0:38:28.9] JM: Okay. I want to talk about schedulers a little bit. Kubernetes has a built-in scheduler that — Actually, I don't know much about it, but I know that if I make a request to Kubernetes for a container, that request is somehow going to get scheduled against the resources that I have available and my request is going to get matched up with a blob of resources that will turn into my container for me.

Can you talk more about how the built-in scheduler for Kubernetes works and how that lines up with the scheduling requirements that you have for Fission?

[0:39:06.8] SV: I'm not an expert on Kubernetes scheduling. Yeah, sorry.

[0:39:10.3] JM: Not a problem.

[0:39:12.8] SV: I don't want to hand wave my way through something that's a lot more precise.

[0:39:19.1] JM: To me, that's almost interesting. It's interesting that you don't need to be an expert on Kubernetes scheduling to build a serverless environment on-topic of it, because serverless is this big scheduling problem, because like AWS Lambda for example. They've got tons of requests for serverless execution coming in at any given time and they've got to schedule those requests against different blobs of resources around their gigantic network of data centers. How does that relate to the scheduling the you have to build for Fission?

[0:39:58.2] SV: We actually completely —Again, this comes down to scheduling the size of the pool. There's a few problems in this area and some of them are scheduling. Let's just list out the questions that affect the scheduling of a request on to some physical compute resource. There's the size of the genetic pool. There's the scheduling of the genetic pool. There's a question of how we choose a part from the pool for a function. Just today, we had a bunch of questions from a user about federation, and we haven't even really started about Federation, so I'm going to keep it to one cluster for a moment.

Then there's stuff like auto scaling when you have lots of load for one function. How do you manage that? To answer all of those questions, first, the size of the pool. Today in Fission, it's static, which obviously only works for a certain rate of incoming requests, incoming cold start rather. It works well enough for a certain size of deployment and you can tweak that static size of the pool. What we will eventually have to work on dynamically changing the size of that pool. It's not directly a scheduling problem, but that something we have to do.

Secondly; the scheduling off the genetic parts onto nodes. That's something that we leave entirely to Kubernetes, and don't really have to do anything function specific there. One thing that's a bit often unsolved problem for Fission on Kubernetes is that because we keep this pool of functions so that cold start is fast, they need to have their CPU and memory usage defined before a function comes in. If a function has a different CPU and memory requirement from what the part was scheduled with, we can't change that parts requirement. There's a complicated set of tasks there we're doing that.

Generally, the idea is that Kubernetes does the scheduling off genetic parts on to the cluster, and then Fission schedules functions into that pool. To schedule functions into that pool, we simply choose randomly, because we don't keep too many parts active that are idle. The assumptions is that Kubernetes has chosen a good place already for any function that might run in those parts. As long as we uniformly spread out functions over the pool, we are okay. Testing on smallish clusters has shown that this is fine. There's a lot more testing to do. We haven't merged our auto-scaling into the the main release yet. There, there's another —

For auto-scaling, we no longer use the pool. We actually create a deployment. Kubernetes lets you move parts from one deployment to another by changing labels. You can orphan a part from a deployment and adopt it into another one. Then Kubernetes own auto-scaling logic can handle the auto-scaling of a function part once it's in a deployment. Auto-scaling work in Fission is basically letting Kubernetes auto-scaler do the right thing with the function.

The reason I haven't gone a whole lot into Kubernetes on schedulers is just a question of maturity. We haven't yet gone to a large-scale testing and high-performance stuff, but that’s something we will do eventually.

[0:43:34.3] JM: Certainly. I did a show about schedulers with Adrian Cockcroft a while ago, and that was a great episode. I really enjoyed talking to Adrian. Of courses, he’s a legendary figure and he just had a lot to say about scheduling and a lot of historical context, but it's a problem that never gets solved. Scheduling a set of resources against a set of tasks is just an impossibly complex problem. That's why Netflix has their own scheduler, why Kubernetes has its own scheduler, why your CPU has its own scheduler, and scheduling is just a thing that has to get solved at every layer of the stack in different ways.

[0:44:18.5] SV: Yeah. True. We might be — Since your function invocations in general are maybe an order of magnitude or more than regular part scheduling's, we might be loading the scheduler quite a bit. That something we’ll have to test and see how it works.

On the other hand, functions being PR compute and not caring where they get scheduled for the most part is helpful, so they won't have a lot of — Functions usually won't have node affinity or things like that. It’s mostly just saying, “Find me some CPU and memory to run this function with.”

[0:44:56.0] JM: What applications have you written that use Fission?

[0:44:59.8] SV: At Platform9, we’re using it for a few internal applications. There’s a couple of parts that are used in our dev ops channel for managing some of our CICD infrastructure. Someone wrote a bot where you can just pause all tests or restart them if you're doing something to the underlying infrastructure, and they didn't want to stand up a whole services for this bot, so they quickly wrote a function, wired it up to Slack Webhooks and they were on their way.

We also use it for Webhooks, again, from our CICD infrastructure and creating these dashboards that are used internally. We’re hosting some external facing apps for our customers using Fission and those are, again — Those contains some interactive forms, so it's important that we are able to start a function on time also. Again, the cultural problem shows up there.

We've had users tell us a lot about how they're using — We have many users who are interested in the Kubernetes watch functionality. Kubernetes has something called a watch, which is where you can specify a set of resources, bots, services, deployments, anything like that, and you can watch them, which means you get an alert anytime something in that set is added, removed, or modified.

This is a nice use-case to wire up Fission to. Normally you write your own controller using the goal Kubernetes client, and that’s a certain amount of work and you have to learn how to do that, and then you would run the service on Kubernetes. With Fission, you can just write one command and say, “Okay, watch the set of bots in so and so namespace and invoke so-and-so function on the contents of that bot whenever it changes.”

The resource gets serialized into JSON and the function gets called with that JSON, and you can do things like, “Oh, if a new service is launched with a certain annotation, you can go wire it up to your own infrastructure’s load balancer or something like that.” You can do custom behavior on Kubernetes resources.

[0:47:06.8] JM: The example you talked about are, like the first one you gave with the chat bots that are able to execute some simple CICD stuff, and the example I gave of the image resizing, these are fairly small compute problem. But I can easily imagine wanting to spin up a serverless function to, for example, serve a machine learning model, and maybe I've just got a few requests that I need to serve to this this machine learning model and it gets discarded. Have you talked to anybody that — Whether you want to talk about machine learning or some other heavier service. What are the heavier services, the bigger, bulkier services that you've talked to people about running on serverless?

[0:47:57.3] SV: There's interest in running video encoding on Fission, for example. That's, again, a computer-heavy task. The idea is that you — There are some storage system, S3, or, really, any storage system where you dump a large video file into it and then — It's similar to the image resizing case really. It's the same thing, but for video. When the video lines in the storage system, you invoke the video encoding function and you get a video that's suitable for various kinds of devices and screen resolutions that's poured back into the storage system.

There's interest in that dependent on our auto-scaling landing. We're pre-beta, which is why all our use-cases are kind of glue and automation kind of stuff for now. In fact, that's kind of how these frameworks start, because those are the areas where you're willing to experiment. It's internal. If the chart misbehaves, then you can go fix it because it's an internal. We expose some of these to customers, and that's carefully managed cluster and things like that, of course.

We come with a don't use it in production label for now. Once we have some of these cool features, like auto-scaling, and we can talk more about a roadmap in a bit. There' a few features that need to be in there before we can do things that are heavier in computer, like machine learning, video encoding, stuff like that. Mainly, it's auto-scaling and workflows around functions. How we manage function upgrades and so on.

[0:49:38.7] JM: There are some other serverless on Kubernetes implementations out there. Have you talk to the other people that are building serverless on Kubernetes implementation? Have you seen any interesting design decisions that they've done differently?

[0:49:52.0] SV: I haven't spoken a whole lot. There's two that I know off that's function from Red Hat and there's Cubeless from Sebastian [inaudible 0:50:01.9]. I've spoken to Sebastian early on. Something they did differently is that they chose not to have a pool of genetic functions, which allows them to — Basically, they create a part whenever a function is invoked. I think that gives them — That's a different trade-off you. You lose the cold start time, but you can specify a part at startup time. I guess it's easier to do things like specifying CPU limits when you don't have a part that’s already running.

[0:50:34.3] JM: That seems like actually a good thing that there are two different approaches that are — Because that's a distinctly different approach with the distinctly different scheduler strategy you might have. You could imagine having a Kubernetes cluster where you have both Cubeless and Fission.

[0:50:54.9] SV: Well, maybe.

[0:50:57.3] JM: Maybe? Who knows?

[0:50:58.6] SV: I think the execution strategy is one part of the whole fast system, right? In fact, we've talked a bit about — We go a little bit back and forth on this in Fission and we’ve talked about, really, just having a pluggable execution strategy. If your function is such that it does not care about the cold start time, but does care a whole lot about features that you can only get by starting a part on demand, then we could just map that function to be invoked as a new part instead of using the pool.

The reason we might want to do that — The other features of Fission, like language environments and the various triggers that are going to be supported, we don't want users to have to say, “Okay, I can either use this language on this trigger and I can get—” Users will have to make strange choices about over what features are available to them.

I think the execution strategy is one part of a fast and it's plausible that we’ll implement multiple strategies. What else? I think they also bundled Kafka — If you’re working on event queue integration as well, and we can talk quite a lot about events. In fact, we just support a HTTP, time triggers based on ground strings and Kubernetes watches, but we’re working on support of message queue calls NATS. NATS is designed as this easy to deploy and operate with tunable persistence settings at least once message queue.

It sounds like a very nice set of properties, and we're working on asynchronous invocation of functions using that. Really, these different deployments have different message queues that they use that are completely dependent on the use-case. For example, heavy users of AWS probably is S&S and SQS, because S3 plugs into it very well. We'll be adding support for a few different message queues as well.

I've spoken to some of the other implementations early on. We tried to see if there was a way to work on one project instead of multiple, but we didn't seem to find one. That's where we are. I think —In general, as you go higher in the abstraction levels of the stack, you get people who do things differently who have different opinions and users end up having to use — Users kind of have to make the choice of what high-level framework to use.

I think this happens a lot — for example, as you go all the way up in the stack, there's 10,000 JavaScript frameworks, and each of them has certain good properties, and I think ultimately users will take their use-case and try to map it into each of these frameworks and see which one fits.

[0:53:54.6] JM: In the JavaScript case, we had a ton of frameworks and then React came out, and React has become the most popular one. But it's interesting, because there were lots of frameworks before. For all we know, we're only at the beginning of the serverless on Kubernetes of frameworks. Maybe there'll be many more to come. Who knows how deep this space goes? It's really hard to tell at this point.

[0:54:22.6] SV: Yeah, it's an early space for sure, and there's a lot more ahead than what's already been created. I totally expect there to be more.

[0:54:32.0] JM: Yeah, it's a wrap up. You work at Platform9, and Platform9, we didn't talk about that much, but that is a higher level control plane on top of Kubernetes. I think you can use it with other platform as a service things maybe, or it's maybe just Kubernetes.

[0:54:47.8] SV: Right. It actually started as an open-stack controlled plain. It started three-ish years ago and the idea was that people have all these on-prem deployments of hardware and they find it really hard to manage all that stuff. Platform9 created this sort of slick controlled blame. You just drop one agent onto your machines and you can run virtual machines using the UI or the CLIs and it deploys open-stack.

For about a year and a half, we've also been deploying Kubernetes for users both on-prem and in the cloud. The ideas more or less the same that they have some computer resources, and you we want to let users use their computer resources either as virtual machines or containers, and there's lots of ways in which those things interact. For example, authorization. You want the same set of user's credentials to be usable with both open-stack on Kubernetes APIs, so that's something Platform9 does, and it also gives you a single UI that lets you look at usage in both open-stack and Kubernetes deployment.

[0:56:01.9] JM: If I'm a user of Platform9 and I'm using it to interface with my Kubernetes cluster, why is Fission important to me?

[0:56:09.9] SV: Fission itself works on any Kubernetes cluster, and we've somewhat deliberately made the choice not to have anything Platform9 specific in it. It's open-source and we're not trying to tie Platform9 to Fission. Any Kubernetes user, we hope, will find Fission useful for a set of use cases that fit well into the fast model.

Now, we'll do some things that make it easy for users to use the stuff on their Platform9-managed managed Kubernetes cluster. We have in the Platform9 UI a little web CLI, which gives you the kubectl Kubernetes CLI and the helm CLI for installing packages on to your Kubernetes cluster. Then you can just use that to set up Fission.

Today, again, we don't tie fission into the product in anyway, but you can just tell them installed Fission and we could drop the fission CLI also into that web-based CLI. You have an easy really quick way to get started with your cluster.

[0:57:14.0] JM: Yeah, cool. Soam, I want to thank you for coming on Software Engineering Daily. We're up against time. It's been really enjoyable talking to you.

[0:57:22.5] SV: Yeah, thanks so much for having me.

[0:57:23.4] JM: You may be want to close off by saying what's on the roadmap, what's on the horizon for Fission.

[0:57:29.3] SV: Yes. We have a few areas for the Fission roadmap. I'll try to keep a sharp, but broadly the areas are development workflows, versioning functions, versioning a group of functions, doing rolling upgrades, things like that, managing testing. Then there's the area off composition of functions and making more complex systems using functions, doing workflow, synchronous and asynchronous composition of functions.

Again, versioning a set of functions and while a workflow is proceeding through those functions. Then there's features around the ingress into functions, so authentication, authorization, stuff like that. There's whole ops space around functions, so better observability, tracing, integration, exception tracking, stuff like that.

We track our roadmap. I'm going to redirect you to our GitHub project to look more deeply into our roadmap, it's at github.com/fission/fission.

[0:58:27.9] JM: Cool. All right. Soam, thanks for coming on the show. It's been a pleasure talking to you.

[0:58:32.3] SV: Thanks, Jeff. Thanks so much for having me. Great talking to you too.

[0:58:35.3] JM: Yeah, great conversation. Okay, cook.

[END OF INTERVIEW]

[0:58:44.4] JM: You have a full time engineering job. You work on back-end systems of front-end web development, but the device that you interact with the most is your smartphone and you want to know how to program it. You could wade through online resources and create your own curriculum from the tutorials and the code snippets that you find online, but there is a more efficient option than teaching yourself.

If you want to learn mobile development from great instructors for free, check out CodePath. CodePath is an 8-week iOS and android development class for professional engineers who are looking to build a new skill. CodePath has free evening classes for dedicated experienced engineers and designers. I could personally vouch for the effectiveness of the CodePath program because I just hired someone full-time from CodePath to work on my company Adforprize. He was a talented engineer before he joined CodePath, but the free classes that CodePath offered him allowed him to develop a new skill, which was mobile development.

With that in mind, if you're looking for talented mobile developers for your company, CodePath is also something you should check out. Whether you're an engineer who's looking to retrain as a mobile developer or if you're looking to hire mobile engineers, go to codepath.com to learn more. You can also listen to my interview with Nathan Esquenazi of CodePath to learn more, and thanks to the team at CodePath for sponsoring Software Engineering Daily and for providing a platform that is useful to the software community.

[END]