EPISODE 27

[INTRODUCTION]

[0:00:00.7] JM: Modern software architectures often consist of containers that run services. Those containers scale up and down depending on the demand for those services. These large software systems also often use a technique known as event sourcing, where every change to the system's data model is kept in an event log. When an event on the log is processed, several different data stores might be updated in response.

In these architectures, containers are often interacting with each other. Multiple databases are responding to events in the event log to connect these systems together. Engineers can write small functions to pass data around. You might call the small connecting functions glue. Glue functions are a great use for a serverless tools such as AWS lambda or Google Cloud Functions. As these glue functions grow in popularity, there's an increased need for an open-source way to deploy serverless functions or otherwise notice functions as a service.

By the way, I know I'm talking about a lot of things that are some advanced concepts here. If you're not familiar with serverless or micro-services or Kubernetes or you event sourcing, we've done shows on all of these topics and feel free to turn off here and go check out our back catalog for these other subjects.

Sebastian Goasguen and works on Kubeless, a serverless execution tool built on top of Kubernetes. In this episode, we explore his take on the serverless on Kubernetes problem. This is a great companion episode to yesterday's interview with Soam Vasani about his tool; Fission.

Software Engineering Daily is looking for sponsors for Q3. If your company has a product or service or if you're hiring, Software Engineering Daily reaches 23,000 developers that are listening daily, so send me an email, jeff@softwareengineeringdaily.com , and I'd be happy to talk to you about sponsorship.

With that, let's get to this episode; event driven serverless with Sebastian Goasguen.

[SPONSOR MESSAGE]

[0:02:21.7] JM: Blockchains are a fundamental breakthrough in computer science and ConsenSys Academy is the place to learn about block chains. The ConsenSys Academy developer program is a free, highly selective, and carefully designed 10-week online curriculum where you will immerse yourself in blockchain development and earn a ConsenSys blockchain certification.

By completing the program, you will be eligible for immediate hire by ConsenSys; a leader in the blockchain space focused on the Ethereum platform. If you want to learn about blockchains and become a developer for blockchain technology, check out the ConsenSys Academy by going to softwareengineeringdaily.com/blockchain. The graduation ceremony is a spectacular all-expenses-paid trip to Dubai where you will meet block chain developers working on real-world solutions.

Dubai has announced ambitious plans to become the first city to run on blockchains by 2020. If you like the idea of immersing yourself in blockchain technology, graduating, and going to Dubai, check out softwareengineeringdaily.com/blockchain. Build your new career on the block chain with the ConsenSys Academy developer program. Applications are open from now until July 1st, so you want to apply soon. For more details and to apply now, go to softwareengineeringdaily.com/blockchain.

To learn more about ConsenSys and Ethereum, please visit Consensys.net, and sign up for the ConsenSys weekly newsletter for everything you need to know about the blockchain space. Thanks to ConsenSys for being a sponsor of Software Engineering Daily.

[INTERVIEW]

[0:04:18.2] JM: Sebastian Goasguen works at Bitnami and he is the creator of Kubeless. Sebastian, welcome to Software Engineering Daily.

[0:04:25.3] SG: Yeah. Thank you, Jeff.

[0:04:26.7] JM: You have written some books about Docker and CloudStack and you've been somebody who’s worked on cluster computing for a long time. Let's start with the little bit of history. Can you give me some perspective on your evolution as an engineer in the cluster computing area?

[0:04:49.8] SG: Yeah, definitely. That’s an interesting question. I’m going to try to be relatively brief. I’ve been around a little bit now, so I’ve seen several things in the industry. I spent a lot of time in academia. I enjoyed research. Interestingly, I’m actually an electronics engineer. I work a lot on things like radars and antennas and things like these. Nothing to do actually with computer science or not directly related.

Very quickly, I got into computational science, so now we’re talking super computers, clusters of servers, and the need to orchestrate all the servers to be able to do computation and solve very interesting problems.

Naturally, I went away from my natural tendencies of electronics, nanotechnology, things like these, and I got interested in making computing or utility. Really, cloud computing; how can we harness all the power of clusters and bring that to people in a very easy manner.

[0:06:06.4] JM: It’s been interesting to watch the adoption of industry, industry adopting all these distributed systems and things that used to be widely discussed in theoretical computer science community bringing these to implementation. Can you comment on how you have seen the differences between academia and industry evolve? From my point of view, it seems like they’ve basically converged.

[0:06:40.9] SG: Yeah, that’s actually a very good point and that’s why I left academia and I joined the industry, wanted to be more on the industry side. Cluster computing, super computers, is still going strong in academia. You got lots of super computer projects in the U.S., in Europe. You got things that are being driven by CERN, obviously, but you got all the super computer in the U.S.

I was very involved in this and some of the problems was definitely scheduling, efficiency of using those clusters. Some problems were also around security running code from anonymous people, potentially running malicious code and things like these. Those experiences and areas of research led to the use of virtual machines in grids, virtual machines in clouds, in super computers. Of course, also, the use of containers for better scheduling, better packing of workload on computers.

All those themes of efficient scheduling, improved efficiency of the clusters, that's exactly what you find with things like Kubernetes which is about orchestrating containers but with an idea towards greater efficiency, packing more workloads on the server. It was also the same theme with virtual machines when VMWare came out with all their solutions which was about consolidation of servers. Definitely, you see the same themes over there.

[0:08:28.0] JM: We've done a lot of shows on Kubernetes. We've done a lot of shows about serverless as well, and I'm sure we're going to do a lot more on these. For people who are a little unfamiliar with these topics but maybe they've heard one or two shows, give the audience a quick refresher on what Kubernetes is, what serverless is, why these two technologies are important.

[0:08:51.7] SG: Sure. Kubernetes is, I would say, the next evolution in a way to manage your workloads. There were solutions like OpenStack and CloudStack which were virtual machine driven. They were very much about deploying your workloads on VM's, which would be a non-prem cloud solution. Now, there's this evolution, thanks to Docker, of actually using containers as a packaging format for the workload. Also, it's pushing towards breaking up monolithic application and embracing more of a micro-service applications, application mindset.

As you go towards containers, now you need a system to be able to manage all those micro-services instead of more bigger VM's, and Kubernetes is quickly becoming the de facto, almost standard to really orchestrate those containers at scale. It's not only an issue of scale. It's also an issue of what are the APIs that you can use with Kubernetes to basically architect your application and all full-tolerance and all configuration and so on.

You got Kubernetes on one side and I'm sure the people listening, I've heard your shows already on Kubernetes. One thing that's interesting with Kubernetes that it also allows you to build extremely new and exciting system. You could call them PASS, platform as a service solutions. Those PASS-like solutions are extremely easy to do on top of Kubernetes. Now, you have all of us that have been working on infrastructure for a long-time on trying to make computing a utility, trying to make infrastructure boring.

Now, we actually start going back towards things like PASS trying to actually concentrate on the application, again, going back to the applications. Now, we figure out that things like serverless are much more interesting and powerful so that we go back to actually concentrating on the logic that you have in your application. You concentrate on writing the little bits of logic that are critical to your system and you want to figure out how to deploy those little logical items.

Serverless is very much about going back to the roots of your application logic, concentrating on those logical units and deploying them as quickly as possible and building much more interesting distributed applications.

[0:11:57.9] JM: The way I have started to think about Kubernetes, like you said, it's a tool where you can easily build PASS on top of it. Ruby on Rails, the common way that people would explain Ruby on Rails is you can build Twitter in a day. You can build an application that makes a Twitter, like the entire functionality of Twitter in a day on Ruby on Rails. With Kubernetes, you might be able to say the same for Heroku. You could build a Heroku in a day, an entire full-fledged platform as a service on top of Kubernetes in a day, and that's what's so impressive about it. That's why we've seen such massive adoption, like the adoption of Kubernetes is — I think it's more aggressive than any other open-source technology in history. Maybe it's outstripped by Linux.

In terms of production use, Kubernetes is widespread. We've done shows with Meetup and several other companies who are deploying production Kubernetes clusters. Serverless, on the other hand, assuming we're talking about functions as a service. This idea where you write your code, you write some blob of code and it executes against a serverless cluster, which is there are some resource that get spun up and executes that code and then that transient resource just disappears somewhere or its get cached for a while or something.

That is a technology that a lot of people are talking about, but I have seen less production adoption of, except for things like image resizing or really well-defined atomic operations where you can very easily have this disposable container.

I've heard you talk about this. You encourage people to not dismiss serverless just because there is a lot of buzz and a lot of hype around it and perhaps there's not as much production use case quite yet. Remind listeners, what is the value of serverless despite the fact that we're not seeing a whole lot of production use of it today?

[0:14:10.5] SG: Yeah. First, I would like to go back to Kubernetes and it being easy to write a PASS-like system. It's extremely easy to write a POC, a prove of concept, but really a full-fledged PASS, it's going to be a little bit more complicated. I totally agree that all the primitives are there in Kubernetes to be able to build those systems.

Now, production. It's something in our industry, we tend to day, "Oh, yeah. I'm production. I'm production," and we're never quite sure what people mean by I'm production. I want to be always careful when we say we are production, because people see it in a different way.

Definitely, talking about adoption, starting with Docker. Docker was adopted extremely quickly. The pace of adoption of Docker was extremely quick. I think it was quick because developer really embraced it. The developer really saw the strength of Docker to be able to quickly get a Redis or get a MySQL, memcache running on their laptop for development.

Then once people indeed started focusing on production, they understood that they needed to move towards a real cluster. They needed a networking solution. They needed an orchestrator across machines and so on, and that's why there was a little bit of a lag between the adoption of Docker and then the adoption of Kubernetes. Even though Kubernetes, of course, arrived after Docker.

Now, what we're seeing, and I agree with you, is that Kubernetes is definitely caching up and the adoption of Kubernetes is quite impressive. Even out of Europe, which is very, I would say traditional, or they're very skeptical about new technologies, especially technologies that come out of Silicon Valley. Even in Europe, you see adoption of Kubernetes picking up extremely quickly.

Now, back to serverless and your question. Sorry, I went around here. Back to serverless. You do see people in production, iRobotics and so on. Even in Europe, you have people adopting serverless in production or which I would say AWS Lambda in production. If you look at solutions to do serverless, right now it's Lambda on AWS, Google cloud function is starting, but I still behind what AWS can do. You get Azure functions. Again, AWS is still in the lead. When we look at production, we actually have to look at AWS Lambda users.

The key for these guys is that they are already in the cloud and they're using the cloud services of AWS. They're using the S3 store. They're using the messaging system. They're using maybe like a vision API or EC2 or whatnot. They're using existing services. Now, they're trying to tie all those services together into an actual more complete application. They're becoming what Patrick Debois from the DevOps movement called service full. Instead of serverless, he mentioned, in fact, people are going service full. They're just writing little bits of logic to link the services.

The key for serverless is going to be a framework that allows us to write those little functions and deploy them, whatever that means underneath. It also means that we're going to need services, so cloud services, available to tie them together. The real key is going to, indeed, at those services and have those services emit events that can trigger functions.

People are in production, yes. You have a few already, but it's still very very early and that's why at Bitnami we're pushing Kubless. It's still very early. Very much an exploratory learning phase, and I think we can meet again in maybe two years and then we'll be talking about real production.

[SPONSOR MESSAGE]

[0:18:059.7] JM: Do you want the flexibility of a non-relational, key-value store, together with the query capabilities of SQL? Take a look at c-treeACE by FairCom. C-treeACE is a non-relational key-value store that offers ACID transactions complemented by a full SQL engine. C-treeACE offers simultaneous access to the data through non-relational and relational APIs.

Company's use c-treeACE to process ACID transactions through non-relational APIs for extreme performance while using the SQL APIs to connect third part aps or query the data for reports or business intelligence. C-treeACE is platform and hardware-agnostic and it's capable of being embedded, deployed on premises, or in the cloud.

FairCom has been around for decades powering applications that most people use daily. Whether you are listening to NRP, shipping a package through UPS, paying for gas at the pump, or swiping your VISA card in Europe, FairCom is powering through your day. Software Engineering Daily listeners can download an evaluation version of c-treeACE for free by going to softwareengineeringdaily.com/faircom.

Thanks to FairCom c-treeACE for being a new sponsor of Software Engineering Daily, and you can go to softwareengineeringdaily.com/faircom to check it out and support the show.

[INTERVIEW CONTINUED]

[0:20:37.2] JM: The architecture that you're describing is people are building applications with things like S3 and Firebase and maybe Heroku and maybe Amazon Kinesis, which is like an event queue, and maybe DynamoDB, which is a really rich database. You want to use these managed services because they take a lot of the frustration of running your own database. Nobody wants to run their own database anymore and be responsible for the uptime of the database. You'd much rather have AWS — Assuming your cost and sensitive, you would much rather just have it hosted and have AWS be responsible for it.

Of course, you need a way to have these services interact with each other, and so that's why we still are managing servers somewhat. We have servers that have lines of codes that say, "When this S3 bucket gets changed, do this other thing over here, and you've got to do glue code between S3 and some other service that you're using."

Serverless, in terms of AWS Lambda or Google Cloud Functions, it's extremely cheap. These functions are really cheap to run. What's the motivation for wanting to have a homegrown serverless on top of Kubernetes platform if it's so cheap to use these hosted services?

[0:22:13.7] SG: Yeah, that's a very good question. There's been lots of talks about everybody going public cloud and then are people going hybrid cloud and so on. I don't really like to be caught up on definitions, what hybrid cloud means or what serverless means and so on. Definitely, what you're going to see is much more of a gray world where as you say, some enterprise are going to say, "Hey, that bit of service, that service actually much prefer having Google of AWS manage that service and use their DynamoDB or use their cloud SQL, whatever."

People are going to study their infrastructure and they're basically going to outsource some components to the cloud. Then that means that they're going to have some type of hybrid environment. They're still going to have some on prem and then some cloud services.

Those bits of glue code exactly as you mentioned, you're going to need to be able to use some glue code deployed in the cloud and then some glue code deployed on prem. As soon as you start looking at containers for some of your applications that are on prem and that you start deploying solutions like Kubernetes to manage those containers, it makes total sense to be able to have a serverless, a [FAZ] solution on prem. That's exactly what we're doing with Kubeless. We are saying, "Hey, you're going to embrace Kubernetes on prem for your new containerized workload. Best way, if you deploy Kubeless, then you'll be able to use [FAZ]. The CLI that we provide is compatible with Google Cloud Functions, so it's exactly the same thing."

We also provide a plugin for the serverless framework. If you're on AWS using Lambda and you are now familiar with the serverless framework, you can also Kubeless. I think there's definitely room for a hybrid type environment, and Kubeless there makes total sense.

[0:24:40.3] JM: There are people listening who might right now be like, "Well, but who cares about on-prem?" If we're talking about on-prem, we're just talking about banks and old healthcare companies and these companies that they don't actually need to be on-prem, they're just slow to the game. The reality is that there is going to be an increased uptake in more on-prem companies. For example, I talked to someone from the Microsoft Azure IoT team and he was talking about Azure Stack, which is an on-prem deployment of Azure Technologies. It's like, "Okay, why would you want to do that?"

The example that we talked about is let's say you've got a factory, and this factory has cameras all throughout the factory and you want to have machine learning model that are processing the data from those video cameras really quickly and making judgments in "real time", whatever that means, but just low latency essentially and you want to be able to like, "Oh, identify — There has been a chemical spill and we should close off this area or warn everybody to avoid this area."

If you want to get the latency really low, you actually probably want to do something on-prem. If you want to have this similar model to what you have in the cloud, in your on-prem system, then you might want to have a serverless architecture for writing the glue code for your big bulky on-prem system.

[0:26:26.6] SG: Yeah, the reality is like companies like Oracle and even Red Hat to some extents, which does most of its business for the enterprise which you can consider on-prem, Cisco, VMware, all these big companies still have a very healthy business. There is a move towards public cloud definitely with Google, AWS, Microsoft.

The end of the enterprise infrastructure and on-prem is not near. It may be in the future, but it's going to be a long long time before this is the case. Now, when you talk about the cloud and everybody is going to the cloud and people are late to the party, when you talk to people who may be not at the forefront of things, they still have a ton of legacy. They're still struggling with embracing some, even version control, embracing configuration management. A lot of the things that we do in our industry when we say, "Oh, yes, DevOps and config management and Git and so on, there are still a lot of people that are way behind that are dealing with lots of Java application that they need to run without configuration management and so on."

So a little bit of a long-winded story, but I think we have to be aware that there's definitely a huge huge market and a lot of people that are still in the enterprises as we say and not going to the cloud anytime soon.

[0:28:18.3] JM: Companies like Google — Let's take Google as an example. Google has the Google Borg, and this is the project that Kubernetes was based on. This is what Googlers still use internally to run their services. It's a cluster manager where you can deploy your application against these giant cluster manager that manages all the resources at Google. Do Googlers — Do you know if they have an internal serverless technology for writing their glue functions, or Amazon perhaps? These companies that are at the forefront of cloud management, are they actually using these types of services internally to improve their server usage?

[0:29:09.1] SG: It's a very question, and actually I never ask them. I actually don't know if they have basically an internal Lambda type function. I think it makes total sense and they probably have. Definitely, I know for a fact that in the case of Borg, different groups within Google have developed their own domain-specific language to describe application and then they've written lots of different types of wrapper on top of Borg to deploy application and even manage them in a certain way. I wouldn't be surprised if some of those PASS-like solution are actually very close to what we call now a FAS solution.

Again, sometimes we tend to get really caught up in naming. For me, the issue or what we're trying to do with serverless is really even forget about Kubernetes, forget about containers. Let's go back to the application. Let's go back to the code. You got some bits of codes on your computer. How do you run that in a distributed manner on remote resources?

It's almost a theoretical question, but it's very basic. Forget about the how it's being done. Indeed, forget about the servers, but forget about Kubernetes and forget about containers. You don't need to worry about Docker, but you care about your code. You care about debugging your code. You care about your code being able to serve all the request that it needs to and you care about your code being triggered properly and so on. How do you make that happen? How do you make that happen extremely quickly so that you can innovate quickly and so on? I'm sure that, internally, Google has probably several different systems to do this and AWS as well.

[0:31:19.0] JM: Now they could just use Google Cloud Functions of AWS Lambda internally if they want to.

[0:31:25.2] SG: Or Kubeless. I'm sure they're going to try Kubeless internally.

[0:31:31.0] JM: Or Kubeless, or one of the other serverless on Kubernetes implementations.

Let's talk about Kubeless itself. I think at this point we've motivated the serverless on Kubernetes discussion. I like having these discussions over and over and over again. We've rehashed so many of these topics about architecture on different episodes, but I think it's really useful because this is a lot of where things are going and I think it's really useful for people to just get refreshers over and over again on how these things work and why they're important. Now that we've done that, let's get an overview for the Kubeless architecture.

[0:32:12.3] SG: Yeah. You're totally right with rehashing those ideas, and that's very much what I keep in mind all the time. A lot of the things we do, they're just evolution. They're not new. We progress, things evolve. We have new technologies coming in, but we are still trying to do the same thing, which is run your code. The time from development production of your code should be extremely quickly, so it's like meantime to production, MTP, or something like that. That should be extremely quick.

Kubeless, the ideas was really that Kubernetes gives us all the primitives to be able to build a very exciting and useful system. The Kubernetes API is extremely powerful. That's one of the reasons why I jumped on Kubernetes. The part concept, the deployment concepts, which allows you to do rolling updates, rollbacks, the service concept for rooting your traffic properly to the right containers. There are lots of different API primitives in Kubernetes that are extremely powerful.

When you think about writing a system to do something like serverless, you can leverage all the strength of Kubernetes, and that's why we call Kubeless Kubernetes Native. What we're trying to do is write actually a very minimalistic system, so it's not going to take us just a day. It's going to take a while, but it's as minimalistic as possible because we reuse all the Kubernetes primitive. We reuse deployment, we reuse services, and we reuse especially the third-party resource to extend the Kubernetes API.

That was the main idea behind the architecture. Now, in details, Kubernetes has this concept of third-party resource, which is a way to extend the Kubernetes API. It's extremely powerful. IT means you're trying to build a new API for your application. You can actually use the Kubernetes API server itself by writing your new API resources, giving it a spec, and then Kubernetes suddenly is able to manage that new REST API endpoint.

It's extremely powerful concepts. We're using it in Kubeless, things like the etcd-operator by CoreOS. They're also using third-party resources. There's a new mechanism to do similar things that's coming up in the new version of Kubernetes which is API server aggregation. It's going to allow you to do same similar type of things. That's the first thing. We use third-part resource and then we reuse as many of the primitives as possible.

[0:35:19.2] JM: Let me stop you there, because this is one thing we don't go into enough on Software Engineering Daily is the deep internals of systems sometimes. We just don't go deep enough. You mentioned this third-party resource abstraction. I'm just a reporter on software and I don't actually use these things, so I don't have firsthand experience with third-party resource. What does that abstraction do within Kubernetes?

[0:35:47.0] SG: The main motivation, we call them TPR, third-party resource. The main motivation was to say that at one point the core of the Kubernetes API is going to be stable. If we've done our job or the group that's developing the Kubernetes API, if they've done their job properly, that course should be extremely stable and shouldn't need to be expanded.

There shouldn't be any discussion about, "Hey, we need this new API resource. We need this new thing, and then we need this new thing, and should it get into core, and, oh no, it shouldn't get into core." There are lots of those debates in open-source communities sometimes. Here, the idea was to remove that potential for debate by saying, "We're going to have very strong core, stable." Then if you want to extend the API, you're going to write your API extension using this system, which is called a third-party resource.

What it is exactly, it's another Kubernetes API primitive. You write a manifest to define a third-party resource. Then what Kubernetes does when you create that third-party resource, Kubernetes now creates on the fly a new REST endpoint. Suddenly, you start talking to the

Kubernetes API server and there is a brand new API endpoint that's available and you can do HTTP get and post and so on. Now, maybe we're getting into too many details. I do think it's important.

[0:37:34.7] JM: Yeah.

[0:37:36.6] SG: Once you've done that, you haven't done much actually. You've just let Kubernetes create that new endpoint. What you need to do is write a controller. You need to write some piece of code that now is going to monitor these new endpoint. When things happen in this new endpoint, then you're going to do some actions.

For us, with Kubeless, what it means is that we create a new endpoint which we call function.IO. Now, we have a new endpoint that's managed by the Kubernetes API server which manages functions. The Kubeless controller just watches all the time that endpoint. When you create a function, it says, "Oh, a function was created. Let me spin up a container using the Kubernetes resources." I spin up a container and I inject the code, that glue code. I inject that glue code inside that container. Now, suddenly your function is exposed. That's what Kubeless is. It's a definition of a third-party resource to extend the Kubernetes API and then a function controller, so we called it the Kubeless Controller, to be able to launch the containers and inject the function code into them.

[SPONSOR MESSAGE]

[0:39:06.7] JM: For more than 30 years, DNS has been one of the fundamental protocols of the internet. Yet, despite its accepted importance, it has never quite gotten the due that it deserves. Today's dynamic applications, hybrid clouds and volatile internet, demand that you rethink the strategic value and importance of your DNS choices.

Oracle Dyn provides DNS that is as dynamic and intelligent as your applications. Dyn DNS gets your users to the right cloud service, the right CDN, or the right datacenter using intelligent response to steer traffic based on business policies as well as real time internet conditions, like the security and the performance of the network path.

Dyn maps all internet pathways every 24 seconds via more than 500 million traceroutes. This is the equivalent of seven light years of distance, or 1.7 billion times around the circumference of the earth. With over 10 years of experience supporting the likes of Netflix, Twitter, Zappos, Etsy, and Salesforce, Dyn can scale to meet the demand of the largest web applications.

Get started with a free 30-day trial for your application by going to dyn.com/sedaily. After the free trial, Dyn's developer plans start at just $7 a month for world-class DNS. Rethink DNS, go to dyn.com/sedaily to learn more and get your free trial of Dyn DNS.

[INTERVIEW CONTINUED]

[0:40:57.4] JM: If my application writers — What if they're just writing tons and tons of functions as a service against my Kubeless cluster. What's going on there? Is it the controller that gets hammered with all those requests and the controller scales up or does the third-party resource scale up? I'm trying to understand the purpose of the third-party resource abstraction versus the controller abstraction that's watching the third-party resource.

[0:41:29.5] SG: What happens for us from a Kubless standpoint is that we didn't have to write our own API server. We're not writing our own API server. We're not writing our own scheduler. We're letting Kubernetes itself handle that. When all your developers start creating function, they hit the Kubernetes API server, and it's the responsibility of the Kubernetes API server to be able to handle all those requests properly. If you hit any limitations, you hit the limitations of Kubernetes itself. We know how well Kubernetes is scaling now, so we are very confident that the scale is there.

Now, all the storage, like the functions, where are they stored. They're stored in the etcd key-value store used by Kubernetes itself. We don't have to set up a separate database. We don't have to handle the state on ourselves. Basically, Kubeless delegates all of these to Kubernetes. That means that we just have the controller and the controller is just like watching the resources and then performing actions.

[0:42:44.9] JM: Can you give me a little more clarity on what the purpose — I understand the third-party resource is this way of extending Kubernetes to do your domain-specific things. I would just love to know in a little more detail, just how — What is the importance of a third-party resource here, because I'm hearing you say, "Okay, you set up a controller through the third-party resource or using the third-party resource. I'm just trying to understand what the purpose of the third-party resource is relative to that controller.

[0:43:13.5] SG: If you compare with the Fission, which is another serverless solution that runs on Kubernetes. When you deploy Fission, you deploy its own efficient API server, and the developers, they're going to talk to the Fission API. The difference with Kubeless is that you're going to talk to the Kubernetes API itself, which with Kubeless, we'll know what a function is. That's one of the biggest difference.

Then once your function is created, we use Kubernetes to actually decide where that function is going to run in the cluster. We use the Kubernetes scheduler. Whereas in Fission, they have their own scheduler on top of it, and that's why we call it Kubernetes Native.

[0:44:11.1] JM: In the Fission world, they have a pool of containers, a resource pool of containers that just sits around, and if you've got — Let's say you've written an image resizing function in JavaScript, they have a pool of containers that are suited to running JavaScript and when the cluster gets a request for the image resizing, they schedule one of those containers against that function. You're saying the different between that and Kubeless is that in Kubeless you've got the function, whatever language its written in is in etcd and when you get a request for that function, the entire scheduling of the container being allocated and the code being loaded into it, all of that is handled by Kubernetes on the fly. Is that accurate?

[0:45:09.0] SG: Yes, that's accurate. What happens, the reason why the Fission guys — I know them quite a bit. The reason why they created a pool of containers is to reduce the startup time of the function, because it's very important in serverless. If your containers are already running and you just inject the little bit of code, the startup time is extremely short. The latency from the time that you create the function to the time that you can actually call the function is extremely short.

For us, in Kubeless, the startup right now is longer, indeed. It's longer, because we have to start the containers. There is scheduling time from Kubernetes, and then you have to start the containers. We opted to delay the optimization on the startup time to actually concentrate on the functionality. We have the plugin with the serverless framework. We have the compatibility with the Google Cloud Functions. We handle dependencies. Then we basically we go through all the AWS Lambda examples. You mentioned image resizing and then Kinesis Stream and things like these. We are methodically going through all those examples that AWS Lambda showcases and we're trying to hammer them out and enable that functionality.

If you go to the Kubeless GitHub repo, you'll see a short screencast of an image resizer using the Minio object store and then using Kubeless with a 50 line of Python.

[0:46:52.9] JM: With database, we have gone to this point where there's so many different databases based on whatever model of consistency you need, whatever a model of availability you need. Do you need ACID transactionality, do you need high throughput. It almost sounds like if we're moving to a world where you've got most of your glue code managed to buy "serverless", some of those glue code operations are more latency-sensitive than others. Is this is a rich enough gradient that we're going to see a real variety of scheduling tools for serverless functionality?

[0:47:41.6] SG: I'm not a database expert. I'm more of a cluster computing guy.

[0:47:46.8] JM: Sure. Just to be clear. The analogy I was trying to draw was that there's a lot of richness in — Originally, I was like, "Okay, we'll just put everything in a SQL database," but then we realized, "Okay, there's all these tradeoffs we can have." Here, between Kubeless and Fission, I see essentially the tradeoff that you have is the cold start problem and how aggressively you're solving the cold start problem. Fission solves the cold start problem really aggressively by essentially preloading the containers with some information and that makes the latency less sensitive.

With Kubeless, you're saying, "Well, we want to have these very — If we're going to solve cold start problems, then we want to solve them with individual solutions for individual problems like image resizing or whatever it is.

[0:48:37.5] SG: No. That's not exactly what I was going at — Is that we want to concentrate on enabling the application use cases because that's what Bitnami does. Bitnami has been doing application packaging and delivery for a long time. We concentrate on the apps. We want to enable the different application scenarios that people are interested to solve using serverless.

We're going through all those application scenarios, data processing, data streaming, image resizing, mobile backend. We're going through all those different scenarios and figuring out what do we need to build into Kubeless to enable those scenarios for the apps.

Now, our thinking is that the startup time. Since we're Kubernetes Native, we actually offload the problem of the startup time to Kubernetes. What we want to do is basically say, "Okay, now we can do Kinesis Steam." But let's say there is a problem with the startup time. Now we actually go back to Kubernetes, upstream to the project, and we say, "Hey, when we are scheduling this in such and such conditions, the scheduler is really bad. Can we do something?"

Now, we can actually write a serverless optimized Kubernetes scheduler because we can tune that. Basically, we feed those requirements back to the Kubernetes community to make sure that Kubernetes can actually handle that. That's our entire philosophy around developing Kubeless.

The one big thing also that's different with Fission, and we haven't talked about it at all. Even for serverless in general, it's very important, is events. All of these is only useful if the services that we try to glue together emit events. Right now, there's lots of debates about coming up with some standard around events, around interface to events and things like these. That's going to be a big challenge for the serverless community.

The one thing we did with Kubeless early on is that we wanted to enable events. When you install Kubeless, you also get a Kafka broker, which is deployed inside Kubernetes. It's a standard Kubernetes StatefulSet Kafka and ZooKeeper deployment. That means that now, any of your services that emit events, if they can emit those events into Kafka, your functions can be triggered by events, which is a totally different scenarios than triggering the iHTTP. We get that out of the box. It plays an interesting role with what you're saying about database, because there are lots of scenarios. People are talking to us and they're saying, "Well, actually, my transaction in my database, I would like all my database events to go into a Kafka broker.

What we're working on is actually figuring out a way to ingest all those events from databases and other things into our Kafka set up in Kubeless so that they can be triggered by a function. Now, you make an insert in MySQL and that triggers a 50 line of code Python function.

[0:52:13.1] JM: Okay, that's actually really interesting. We did a show with Neha Narkhede from Confluent a while ago and she was part of the founding team of Kafka. We talked about event sourcing. We talked about event sourcing on Kafka. Event sourcing or CQRS. I'm not exactly sure which one is referred to specifically. Basically, the idea is every time there is a change to anything across your applications data, you put that in a Kafka queue, or some kind of queue, and you have different consumers of that data change.

If you have a change to a user, a user's profile, you're going to need to update several different databases. You need to update the user database — If a user goes on to their profile page on Facebook and makes a change to their user profile, you're going to need to change the user database. You're also going to need to change the elastic search cluster that powers the search overall of your users and all of your other data. Maybe you need to change some other databases.

The question is, "Okay, if we want to be able to trigger multiple database changes based off of one user operation, how do we do that? The answer is you put the event on a queue and you have listeners that are somehow wired to listen for that type of event. Then those listeners process that event and make the change to the respective databases that need to get changed. This takes a lot of resources, because you've got tons of database changes in coming. Basically, what you're saying is, "This is a perfect application for serverless because you've got all these quick database changes that need to be made and it's kind of an atomic event and you probably want customized ways of making each of these changes."

Am I describing the situation accurately?

[0:54:16.8] SG: Yeah. Perfect. A big thing for serverless is event-driven based application. In itself, it's not new. You bring enterprise service bus type application and you have similar things. The key here is that those complex scenarios which need a fairly complex infrastructure. Now, it's becoming extremely easy to deploy. You set up a Kubernetes cluster. You turn on Kafka. You add Kubeless on to it. Now, suddenly, you start writing these glue code and that complex scenario that you just mentioned, now you can actually concentrate on it and write it extremely quickly. You get back to the basis of what we're all trying to do, which is an extremely quick pace of innovation, writing software, deploying software extremely quickly and being able to focus on those complex scenarios and doing it relatively easily.

[0:55:31.1] JM: As we begin to wrap up, we started off kind of talking about, "Okay, why isn't there a lot of serverless in production, or how are people using serverless today." Bitnami, it seems like is a great example of how you might use serverless. The little bit of your history is you started a company called Skipbox and then Skipbox joined Bitnami largely because teams like Bitnami is a company that can really make use of the Kubeless technology that you are working on at Skipbox.

Give a little history for why Skipbox joined Bitnami and what you're doing there now. What Bitnami is and how Kubeless fits into that.

[0:56:17.7] SG: When we started, I discussed a little bit about my background in computational science. Definitely, I saw a point where lots of things is being done in academia, but the industry is definitely pushing extremely hard on new technologies and so on, so I decided to leave academia, get on the industry. I worked on cloud systems. When Docker came around, containers — We had been using containers for a while. It wasn't new, but I was very intrigued by the fact that Docker really picked on. That's why I started writing the Docker cookbook. That's why, recently, you heard me talk about do not dismiss. Do not dismiss new tech, otherwise you may get left behind.

I didn't want to dismiss Docker. Right now, I do not want to dismiss serverless. Some people are going to do it naturally, but if you don't dismiss it and you actually try to understand what this is about, you'll realize that this is a continues evolution of what we're trying to do. The infrastructure is now boring. Cloud is really good. We have Kubernetes. We know how to do containers. Now, let's get back to applications and let's go back to writing applications, operating them, deploying them.

When we talk about applications, that's where Bitnami comes in. We had written Kubeless as part of Skipbox. We wrote additional Kubernetes software, like Compose, like a mobile app called Cabin and different other things. We started developing Kubeless late in 2016. We were talking with Bitnami and focusing on applications. That's where the two of us, we got together and we really hit a nerve is that I saw as part of Skipbox, I saw that this was all about applications trying to make them extremely easy to deploy, operate them and so on.

Bitnami totally now has embraced Kubernetes, has the new platform to deploy applications on. Bitnami tries to support their users in that entire journey of native installer if you're bare metal, VM if you do VMs, cloud if you use cloud. Bitnami can package and deliver applications for all those formats. Now, Bitnami is a strong contributor to helm charts, which is like the Kubernetes package manager. Serverless, it's a little bit in the future. There are already some users, but it's an advance packaging and deployment mechanism.

We want to be well-aware of what are people trying to solve and so on and we think we have a solution here to help the community embrace serverless and be able to build that continuum for applications. Don't leave anybody behind. It's not politics. Don't leave anybody behind. You want to do VM, we can help you, and then we can help you go through Kubernetes and then potentially up to serverless. That's where Bitnami and Skipbox came together and now we're very exciting and pushing further.

[0:59:50.0] JM: All right, Sebastian. It's been great having you on Software Engineering Daily. Thanks for coming on.

[0:59:53.5] SG: Thanks, Jeff.

[END OF INTERVIEW]

[0:59:57.7] 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.

[END]