EPISODE 524

[INTRODUCTION]

[0:00:00.3] JM: Earlier this year, we did several shows about Cloud Foundry, followed by several shows about Kubernetes. Both of these projects allow you to build scalable multi-node applications, but they serve different types of purposes for different types of users.

Cloud Foundry encompasses a large scope of the application experience larger than Kubernetes. Kubernetes is lower level and it's actually being used within newer versions of Cloud Foundry to give Cloud Foundry users access to the Kubernetes abstractions.

Recording these shows about Kubernetes and Cloud Foundry gave me a wide understanding of how infrastructure is managed across enterprises and how it has evolved. They were really helpful for me. Today's episode gives even more context about Cloud Foundry; how the project got started, how people use it and where Cloud Foundry is going.

Today's guest, Mike Dalessio is a VP of engineering on Pivotal Cloud Foundry. We had a great time talking about his work. Engineering leadership is a fine art and conversations with engineering leaders are consistently interesting. This was no exception. I think you're going to enjoy it.

[SPONSOR MESSAGE]

[0:01:22.9] JM: Sponsoring today's podcast is Datadog; a cloud-scale monitoring and analytics platform. Datadog integrates with more than 200 technologies, including Cloud Foundry, Kubernetes, Docker and Kafka, so you can get deep visibility into every layer of your applications and infrastructure in the cloud, on premises and containers, or wherever they run.

With rich dashboards, machine learning-powered alerts and distributed request tracing, Datadog helps teams resolve issues quickly and release new features faster. Start monitoring your dynamic cloud infrastructure today with a 14-day trial. Listeners of this podcast will also get a free t-shirt for trying Datadog.

Go to softwareengineeringdaily.com/datadog to get that free t-shirt and that 14-day free trial. That's softwareengineeringdaily.com/datadog.

[INTERVIEW]

[0:02:25.4] JM: Mike Dalessio, welcome to Software Engineering Daily.

[0:02:27.7] MD: It's great to be here. Thanks, Jeff.

[0:02:29.3] JM: You are a VP of engineering at Pivotal. You work on Cloud Foundry. I want to start with a little bit of history for how the company evolved. Pivotal is interesting and that it wasn't necessarily a startup that was founded from the ground floor. It was a germination that came out of several different companies coming together and realizing that there was a set of opportunities that wouldn't be best suited with those companies themselves. Could you just give a little bit of history for how the company Pivotal came to be?

[0:03:08.2] MD: Sure. I'll at least give you the version that I know. I started working at Pivotal as a consultant for Pivotal Labs, which is where Pivotal gets its name from. Pivotal Labs is where the core of how we build software came from. Pivotal Labs has some very strong opinions on how to build software. It's been called the reference implementation for extreme programming at times, and that actually influences a lot of how their company works today. That's pretty relevant.

In 2012 or so, I think EMC acquired Pivotal Labs, which is interesting if you think about it. EMC is a huge hardware storage manufacturer. Why would they buy a software consulting firm? Part of that is because we had been working with a subsidiary of EMC called Greenplum; working with them to build some of the tooling for their data center product. They were really happy with us.

EMC bought us and set us on a corner for a year and said, "Don't change anything. We'll figure out what to do." Over the course of that next year, we worked with some of the other companies in the EMC federation, notably VMware, the Cloud Foundry Project, worth noting that given

where I am now, a VP of engineering for Cloud Foundry, that it all started as a consulting gig for Pivotal Labs working with VMware on that project.
Adding a little bit, the fast forwarding to 2013 or so, all of the CEOs of various companies; VMware, EMC, Greenplum, Pivotal Labs at the time, decided – there was actually a really interesting investment hypothesis here, which is these companies can all combine to build a company that can change how people build software. This is actually the vision for Pivotal today is to transform how the world builds software; presumably for the better.

I like to joke that you might as well assume that we're transforming it to become worse. But we think we're doing it for the better. Now Pivotal is made up of product teams for Cloud Foundry, which is platforms of service that I think has been covered on a couple of previous podcasts. Greenplum, which is a massively parallel database used for data warehousing jump fire, which is this massive in-memory data
Grid and Pivotal Labs, which is still where a lot of Pivotal's services are provided to our customers through Pivotal Labs.

[0:05:35.3] JM: The reason I find the story of Pivotal interesting is because the narrative around how companies get built today is typically they start with one specific vertical, and then they get extremely good at that vertical and they expand horizontally.

Actually there's a case with Amazon starting with a bookstore and just – you can look back and you can trace back the history and obviously today, Amazon looks like a huge sprawling company, but if you look at each of the areas that they expanded to in retrospect, it's all systematic and it makes a whole lot of sense. They didn't stray too far from their core competency. It's just that their core competency expanded over time.

Pivotal is much different, because when it got started it was from day one this sprawling array of products that people used. There were synergies between the products, but there were also a lot of disjoint customers between them. I know you probably weren't in the room when perhaps the CEOs were deciding how this was going to unfold, but how does that affect the management structure and the strategic decisions at the high levels of the company?

[0:06:48.1] MD: It's a great question. We're definitely on a journey together at Pivotal to figure this out. The key bits being we have these diverse products, how do we make them work well together? I think we've got something that resonates with the market in terms of Spring plus Cloud Foundry. Spring wanting to be the developer framework of choice and have it be tuned to run really well on Cloud Foundry and make Cloud Foundry the best place to run Spring.

Incorporating the data, the vision for data products has been a longer journey. Over time the R&D teams have started to work more closely together. I think what's happening now is we're starting to express the vision of we want to transform how the world build software and we want to be the home for everyone's workloads, including your Big Data workloads. That's what 2018 looks like for us is really honing that vision and getting all these products to work together well in a really compelling product.

[0:07:43.5] JM: Let's get into talking about Cloud Foundry, because that's a topic that I have been asked to cover more often by the listeners, and I want to start with a little bit of the history, because this is a project that was started in 2009. It was an open source platform as a service. In the last, almost a decade, it's grown to become quite a popular project. Could you talk a little bit about how it evolved? Maybe give a little snippets on the early days of the Cloud Foundry Project as well.

[0:08:16.2] MD: Sure. I wasn't around for a lot of the really early days. This has been passed down through oral tradition. Originally there were really only two teams working on Cloud Foundry. There was a runtime team that was worried – their concerns were containerization, container orchestration, the external API calls that need to be made by developers.

Then there is the BOSH team, who is concerned with the infrastructure abstraction layer. What happened over the following – that was in the 2012-ish timeframe. What happened over the following years though is like cellular mitosis happened, where – the runtime team split into multiple teams. I think at the time it was like, there was a routing side and then there was the container management side of things.

Then those things were empowered to evolve their own pieces of the infrastructure independently. Over time, we got to the point where we are now, where there's something like I

want to say 30 or 35 different open source teams working on Cloud Foundry around the world. We have teams based in Europe or SAP as working on a BOSH CPI. We've got teams working on San Francisco, and pretty much every time zone in between. I can go into little more specifics about some of those changes if you want.
[0:09:32.2] JM: Maybe you could describe the early customers, the early users of Cloud Foundry why they moved onto it.

[0:09:41.8] MD: I think the big thing early on, the elevator pitch for Cloud Foundry was hey, this is like Heroku, except that you can run it in your own data center. This is attractive for a certain set of customers who are maybe in a regulated industry and aren't ready to move to the public cloud yet. Or maybe they just have their own governance policies around data.

Early on there was massive distrust of the public cloud in banking and insurance. They **[0:10:11.6]** to the public cloud. Cloud Foundry was a really attractive option just from the point of view of how do we operate a lot of applications, without having to go unto Heroku or other public clouds? That's why it was attractive initially.

[0:10:27.7] JM: Well, let's talk about the deployment of an app to Cloud Foundry. Let's say somebody's getting started with Cloud Foundry, what happens when they deploy their application to Cloud Foundry for the first time?

[0:10:41.6] MD: Great question. [Inaudible 0:10:43.3] who you interviewed has this haiku about Cloud Foundry that goes, "Here is my source code, run it in the cloud for me, I do not care how." This is essentially what cf push does. When I run cf push from my directory, it's going to upload all of my source code to Cloud Foundry and it goes through what's called a staging process. Staging uses buildpacks, which are very, very similar in concepts to buildpacks people might know from Heroku. The job is to turn that source code into a full running application.

That might involve compiling code, if it's a go application. It's definitely going to involve resolving dependencies. For Ruby, it's going to run bundle install. It will do all of your NPM, or tip things if you're in node or Python languages. Cloud Foundry comes with nine different buildpacks to support nine different language ecosystems. At the end of that where you have – so you have a full-blown running app, all the dependencies are resolved sitting in a droplet and ready to run on port 8080. Then that blob just gets dumped into a container and spun up. That's how your application actually runs on the platform and can scale up and down from there.

[0:12:01.1] JM: Perfect. Let's get into the technical details of how Cloud Foundry works. You've already described how to spin up an instance of an application on Cloud Foundry. Once that is spun up, there are some things like routing and authentication and application life cycle, messaging, stuff that we should discuss. Let's talk about routing.

[0:12:24.8] MD: Sure.

[0:12:25.1] JM: How is routing configured on Cloud Foundry?

[0:12:28.5] MD: I guess, maybe I would start with the cloud controller, which is like the brains of Cloud Foundry. It's where most of the state in the system is stored. It's the canonical single source of truth. The cloud controller has all the information about what routes should be up and running for the applications that are up.

Well, the way this generally happens and there is to be honest, this is really difficult for me to store all of the state of Cloud Foundry in my head, because it's a pretty complicated system at this point, and it's evolving really quickly. To the best of my abilities, how this works today is when the application has spun up, part of the metadata that goes along with that application is what route or routes do I want this available under, like what are the names?

When the application is up and running, Diego which is the container orchestrator, when it sees that a process is up and running and healthy, it will essentially send out a broadcast message to the routers saying, "This application is up and running at this IP, on this external port. Feel free to start sending traffic this way." Then the frontier of HTTP layer stuff and routers is then aware that any incoming request can be sent to that IP and port. Does that make sense?

[0:13:44.2] JM: Yes, absolutely. Then throughout the different applications that you have running across Cloud Foundry, you might have different levels of authentication. How does authentication and identity management work within Cloud Foundry?

[0:13:59.3] MD: That's a great question. There is an open source component called UAA, that's in Cloud Foundry, which can be used as a single sign-on server for anybody who's logging into the system acting as developer in the system. If you're an end-user, who is sitting in the application though, you're not going to use UAA.

We have a couple different ways that we can authenticate the connection all the way to the container. It's really only relatively recently that we've been able to extend encryption and container authentication all the way to the container. We used to use the go router as the SSL terminator, where it was done in front. You have a load balancer or something in front of the routing tier that would do that.

We do have the ability now to – This might not actually be in the Pivotal product yet, but this is definitely ongoing in the open source work. We're actually going to be experimenting using Envoy to terminate SSL connections in the container. We're actually going to be using Envoy as a sidecar process in those containers. Now what I'm saying is I'm not actually sure when that's going to get delivered. Take that with the greatest offer now. That's one way to do this.

[0:15:07.4] JM: Well, it's an exciting development nonetheless.

[0:15:08.9] MD: It is. Another way to do it is to in the headers, send information about the address that the user is trying to connect to. You can use SNI here to transmit some information through the container as well. There is a lot of activity in this area. It's really interesting.

[0:15:24.5] JM: Yes, definitely. Of course, across any application you've got storage, you've got databases. How are blobs of data and block storage for databases, how are those different types of storage managed across Cloud Foundry?

[0:15:46.4] MD: Well, there's a couple of different things we can do today. I'll start with maybe the most recent and most interesting bit of this, which is persistent storage for applications. Diego for a long time require that you had a pure cloud native tool factor application running, which meant no local storage would be persistent in the container.

You should be using a database to do any of your stay for work. There's a feature in Diego now spearheaded by a team of people at Dell EMC to introduce persistent storage to the applications running, and this is done over in NFS binding. It's not doing a block storage binding to the application, so we want to make sure that multiple applications can bind to this and great to it, etc. One way to deal with storage, at least on the application layer if you need it locally is to use this persistence mechanism in Diego that's available now.

[0:16:40.9] JM: I see.

[0:16:42.3] MD: If you're going to be spinning up databases though, so most of the database is at least that are part of Pivotal's Cloud Foundry are going to be BOSH-deployed on some level, and a lot of these are even going to be on demand databases that get spun up. As a developer, if I want a new database, I have the option of connecting to a service broker and saying I would like a database, which will be created in a multi-tenant database. I'll get a slice of that.

You can also spin up an on-demand database. One of the things that Pivotal sells as part of our commercial offering is the ability to spin up on MySQL database, MySQL cluster HIA on-demand and use that for my application or set of applications, which is pretty interesting. In either case, this database is going to be deployed on top of BOSH. BOSH has all of these infrastructure primitives available to deal with block storage. If you're on Amazon, you'll get EBS and it will manage all of your disks for you. It's almost like magic. It's pretty good.

[SPONSOR MESSAGE]

[0:17:45.5] JM: When you're building an application, you needed to be fast, secure and always evolving. With Kubernetes engine on Google Cloud platform, developers can deploy fully managed containerized apps quickly and easily. Google has been running production workloads in containers for over 15 years. Google builds the best of what they learn into Kubernetes, which is the industry-leading, open source container orchestrator.

Kubernetes engine combines automatic scaling, updates and reliable self-healing infrastructure with open source flexibility to cut down development cycles and speed up time to market. To learn more about Kubernetes engine, visit g.co/getgke. That's g.co/G-E-T-G-K-E. G.co/getgke.

Thanks to Google Cloud for sponsoring Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:18:50.5] JM: We went over this a little bit in my conversation with Rupa about BOSH, which negotiates this relationship between the Cloud Foundry instance and whatever cloud provider it's running on, so there is this contract that is consistent across any cloud provider that you are on top of, that the BOSH – I guess, maybe you could refresh us a little bit on that – the discussion of the BOSH. I think it's the CPI. Is it the cloud provider interface, the interface between Cloud Foundry and a given cloud provider?

[0:19:23.1] MD: That's right. You have a great memory. Yeah. The CPI is relatively simple. It's just got a couple of primitives in there to spin up a VM, spin down a VM, give me a disk, resize a disk or throw away a disk. Using these primitives, you can spin up and run with everyone on those VMs.

[0:19:41.4] JM: Right. Every VM has this Diego cell that manages when an app starts or stops and also manages a VM's container. Can you talk a little bit more about the Diego aspect of things, I think that's relevant for application execution.

[0:19:59.3] MD: Yeah, exactly right. There are a lot of different kinds of VMs that have to run inside of a Cloud Foundry instance. The bulk of the application work is done with Diego. There is what we call a Diego cell. The Diego cell has a couple of processing running on there to manage the local containers and do health checks and make sure that auctions happen appropriately.

The really interesting get here is that that is where the application containers will end up running. A big deployment, there could be a couple of a hundred Diego cells. We have one customer in particular I know has a large – just a development environment that has something like 350 Diego cells on it. We've actually done scaling tests up to 1,200 Diego cells running 250,000 applications.

Yeah, it's interesting. There is a really great paper about this that you can download from I think the open source Cloud Foundry website with some of those details around it. The Diego cell itself is where the magic happens.

[0:21:01.5] JM: Cool. You also touched a little bit earlier on the aspect of service brokers. Service brokers help link applications to services such as databases. Talk a little bit more about service broker. If I'm a Cloud Foundry developer, what do I need to know about a service broker?

[0:21:19.4] MD: Yeah. The easiest explanation that I can think of is that it's simply an abstraction layer for a database or another service. The primitives in this abstraction are I would like a database please. Then it will come back with like, here is an IP address you can connect to. Then you might say, "Well, I would like some privileges on the system and it will come back with a username and a password for you."

The information that it gives back to you is going to be very generic. Here is how to connect, here is what you authenticate with. The service broker's job is to translate those very general requests into something very specific. To MySQL database, it's actually behind the scenes going to create you a new database. It's going to create a new user. It's going to set a random password for you and give that back.

This abstraction is general and off that – you can use it for things that aren't databases. One of the interesting things about PKS, which is Pivotal's new Kubernetes distribution is you can actually run these Kubernetes clusters on demand. As a developer can say, "I would like to have a Kubernetes cluster please." It will spin up a cluster. "Then give me back the information I need to then use Kub control to connect and run on that cluster."

That's an interesting idea that you can – you're spinning up a service. We use this ambiguous abstract word service to represent. It could be a database, it could be a message bus, it could be a Kubernetes cluster.

[0:22:46.9] JM: When you say that term 'message bus,' when you talk about a message bus running with Cloud Foundry, how do people use messaging systems within – they've got a big –

like if I'm a bank, I've got certain messages that I want to publish and have any consumer that wants to subscribe to that message be able to subscribe to it.

When you have multiple – when you want to do multi-cast messaging – that's a common pattern. It's the pub-sub pattern. You've got a message bus and you've got publishers and subscribers. How do people implement those systems within deployment of Cloud Foundry?

[0:23:29.0] MD: Great question. I'll use RabbitMQ as a case that's relatively top of mind for me. Again, I'm going to as a developer say, "I would like to have a RabbitMQ cluster, please." There are different plans set up, where it's maybe it sets a three-node cluster and each of the nodes has half a gig of memory or something.

I can fine-tune what size cluster I want that somebody attributes around that. Then once it's up and running, I will have the permissions to both publish and subscribe to whatever gets put on that message bus. It's sized appropriately for my application, the number of subscribers and publishers that I have. Behind the scenes, that's actually being deployed by BOSH on demand to new VMs. It's not running within the Diego application container system. It's actually running on VMs that are spun up on demand.

[0:24:18.6] JM: Could you provide some common use cases for why people would want to use a pub-sub system?

[0:24:23.9] MD: Sure. The Spring team has a framework called Spring Cloud Dataflow. The idea here is that you might be running in internet of things, you might be pulling streaming data off of a number of devices. Cloud Foundry supports TCP routing in addition to HTP routing, so you can have these devices be streaming data in through the go routers to your applications running.

Maybe that all just gets dumped straight into a message bus, and then Spring Cloud Dataflow allows you to set up essentially a pipeline of factors that will handle this data and process it and maybe it's you're subscribing to one-name space and then you're going to write back to the same cluster in a different name space and chaining all these data together. Spring Cloud

Dataflow has a lot of really great primitives for developers to do that really easily. That's all built on top of Rabbit.

[0:25:15.7] JM: I've been doing a lot of shows recently about Kubernetes. One of the things that I've come to understand about that system is the importance of configuration management, and configuration management can mean all kinds of things, like how many instances of a service do you want to be standing up? What are the IP addresses that you want to route different data points to – You use configuration for all kinds of things. Is there a global configuration system within Cloud Foundry? Or could you just talk about how config management works?

[0:25:50.8] MD: Sure. I think there's a couple of different levels of the system that end up getting expressed through YAML config for the most part. On the infrastructure layer, you're describing essentially the topology of Cloud Foundry that's being expressed through BOSH config file.

Essentially, I have a config file. It's going to get uploaded to the BOSH director, and the BOSH director is going to item potently, make sure that the state of the universe matches what's in that config. That in it of itself is like a really large configuration, because you're describing what are the types and sizes of VMs that I want to use. This is my cloud configuration. There is my actual little description of the jobs, the processes I want it running on each of these nodes.

Then there's also the set of credentials I have. All these get merged together, because they all have a different rate of change and are stored in different places, that actually all gets merged together by the BOSH director, who then makes sure that you have actually paved your architecture the way you – your infrastructure, the way you need it to be.

Then for each application, the application developer can have his own application manifest, and this is going to describe the metadata around the application, so things like how many instances to I want to have running? What are the routes I want to associate with it? How much memory do I need to associate with this? How much local disk does the process need?

That configures under the control of the actual application developer who is deploying to Cloud Foundry. Those are two examples of where configuration is important. In my application when that's running needs configuration, the Spring team has what's called a config server, so I can store my config in a Git repo and setup a hook, so that every time I commit a change to that config, the application gets the config pushed to it. This is really useful, especially when you're trying to do a lot of config as code and iterating aptly.

[0:27:49.3] JM: The Spring is the Java framework that many people use to build their applications and oftentimes they're using Cloud Foundry as the runtime system for scaling and managing those Spring applications. From a developer on a day-to-day basis, maybe I'm not interacting with Cloud Foundry so much, except when I'm pushing my code. It's mostly Spring that I'm interacting with. Maybe the Cloud Foundry might be more the purview of the operations person, but that might depend from company to company.
In any case, I know as a developer I did a lot of work on Spring Systems when I was working at various enterprises. I did find it to be something of a complex beast to understand. I went through I think three different companies where I worked at Spring, without fully really understanding what was going on with Spring.

When I heard about Spring Boot, I thought this makes a whole lot of sense. It's a simplified, stripped-down version of Spring. Can you describe how the Spring Boot project got started?

[0:28:59.8] MD: I'm not sure I can actually do the story justice. I came to the Spring ecosystem relatively late in life. I think I had the same experiences that you did, which is like this is just a huge sprawling framework ecosystem and like where do you start? I had moral objections to writing Java code and XML before Spring Boot came along.

I think the short version is probably that Spring Boot was specifically written to address some of those concerns, right? Like hey, let's write Java in Java. Let's make configuration easy. Let's make things auto-wire. Let's do good things by default.

That came out of a lot of what was happening I think in non-Java ecosystems that point to **[0:29:42.5]** for a long time made its bones by just being like, "Hey, we're going to do the smart thing by default, but you can configure it however you want."

The Spring team intended Spring Boot to address that. I think that's been really, really successful. I work with a lot of the Pivotal Labs' consultants still who do quite a bit of Java and Spring Boot work these days. These are people who all used to use Rails and they were very happy in the Rails ecosystem. Their response to Spring Boot is like, "Hey, you know what? This is actually pretty good."

Java makes something as just difficult by its very nature. Spring Boot is really easy to use. When you hear that from people who are steeped in a lot of the – let's call it typeless fun languages like Ruby and Rails, that's says a lot. That says a lot about how far Spring has come and how far that ecosystem and community has grown.

[0:30:38.4] JM: When I was just talking [inaudible 0:30:39.7], I realized although I know that Cloud Foundry is widely used in enterprises and if you look at a bank, or a company that makes canned goods or something like that, these giant enterprises, oftentimes they are running on Cloud Foundry. I actually have not much of an idea how different – are different teams using the same Cloud Foundry instance? Are they using different Cloud Foundry instances? How are resources shared between them?

I'm sure there are listeners out there that work at large companies that use Cloud Foundry and would love to know more about some common patterns. Can you describe how a Cloud Foundry, I guess a cluster is it shared? Or are there different clusters within their organization? How does a giant enterprise like a bank deploy Cloud Foundry and let other people within the enterprise stand up their own applications on top of it, or other instances of Cloud Foundry? I guess, how is it used within large enterprises?

[0:31:43.4] MD: I don't think there is any one way that everyone does this. There are a couple of common patterns though. One I can think of is there is really large banking customer. Has a Cloud Foundry instance, we call it a foundation; that's probably an overloaded word, but it's a single deployment of Cloud Foundry running for each of their business units. This means that in all practical reality, they have 40 or 50 Cloud Foundry deployments that they need to manage worldwide and follow the sun model.

There are other banking customers that definitely go for the, we're going to go big on one or two Cloud Foundry deployments and everyone is going to be multi-tenant on these. They optimize for operational simplicity. Most everybody is running multiple foundation simply for disaster recovery though. Even if you're going for one big shared Cloud Foundry instance, you're probably going to have at least two running in two different availability zones.

These are people who are usually running in their own data center. 80% of Pivotal's customer at least are running on top of VMware, vSphere, running in their own data centers. They'll have two data centers that they'll treat as availability zone. If one of them goes down, the applications are still running on the other one.

Then maybe even have a third one setup as disaster recovery somewhere else at a very different physical location. This is going to vary from company to company around what their uptime requirements are, what their meantime to recovery is. There's a lot of variables here.

[0:33:19.7] JM: Yeah. You mentioned VMware, vSphere. VMware is a company I need to do more shows on, because it's a company that was in its – it was in its most prominent days before I got into software. Do you have much of understanding of the history of how VMware made its way into enterprises and the effect that I guess VMware has on enterprises today?

[0:33:44.1] MD: I don't think I can do the history of it justice, but I can tell you a little bit about where it is today, which is they have hundreds of thousands of installations worldwide in enterprises large and small. Their penetration in large and medium-sized businesses is just unbelievably high.

Their main product, which I called vSphere earlier but is really like ESXI is the product, which is VM management. It manages the disk, it manages the network, it manages the VMs for you. It does everything. It's just an incredible piece of software. It's providing all of these infrastructure abstractions for like your – it could be just for half a rack of machines, and I've got ESXI running on it.

This really is a powerful abstraction. Before VMware came along and did this, people were managing physical hardware. The density of what they're being used for was really low, it was really expensive and vSphere and ESXI coming along changed all that in a very fundamental way and made this much, much easier to manage.

[0:34:49.3] JM: Yeah. When you say density, you mean they had these physical servers and you might be utilizing 20% of that server and then VMware allowed you to essentially split that server into a number of servers, and then this was taken even further with Docker and other containerization technologies that just split up those VMs even further, but is interesting to look back and just think about the amount of waste that was in it. No getting around it back then, but there was just a lot of waste.

[0:35:26.6] MD: Yeah, there's a ton of waste. In addition, there was just really long lead times when you needed anything. I remember the days when I wanted to run a demo and you had to put a ticket in with IT and ask for machine and have this back and forth that would last in days, where they would say, "How do you want your file system structured? How much memory do you need on this? How many NIP cards do you need?" Those were really the battle days. It took weeks to get anything done.
VMware having this terrific product that allows companies to just spin up a VM on demand of whatever config you want shortens that time somewhat. Then having a platform as a service on top of that, where application developers can just be like, "I don't care how much memory, or how much disk I have. Just put it on the cloud for me. I don't really care how it happens," shrinks that feedback loop even more.

[0:36:15.5] JM: I think I talked to Rupa a little bit about, but the Spring Cloud Netflix set of tool, so I've been reading about the Netflix open source tools for a while. They've got Hystrix, which is a circuit breaker. They've got Eureka, which is a service discovery system. I always wondered who else works on the Netflix open source stack. What are the other companies that consume this technology? As I was doing the research for these Cloud Foundry shows, I found that many people who are in the Cloud Foundry ecosystem take advantage of these things. Do you have much knowledge of how that developed?

[0:36:59.4] MD: I have a little bit of knowledge accidentally. I was pulled in as the facilitator for the Spring Cloud services inception. When that team first came together and said, "Hey, there is the thing here where we can take all of these Netflix, OSS, these building blocks and components and we can smash it together with some of the other Spring components and create this thing called Spring Cloud Services.

I happen to be in the room as a witness to all that, which is really great. I think the idea here is similar to Kubernetes. Kubernetes provides all these really great primitives building blocks for building a platform. The Netflix OSS stuff are really great building blocks, but there is high-level of complexity to get those to work together to actually accomplish what your goal is. Spring Cloud Services was about taking those components, making them easy to use. Being able to reason about them and deploy them on Cloud Foundry with a minimum of fuss.

[0:38:00.4] JM: Something like circuit breaking, so before there was a system Hystrix, which does circuit breaking and controls interactions between services and for example circuit, the circuit breaker pattern is if two services are communicating with each other and then there is another service downstream of that, like let's say there's a service A calls service B, service B calls service C, service C calls service D.

If service D has some latency and service C is waiting on it, you might get this chain of blocking that happens among these different services. Hystrix is a circuit breaker, which maybe provides a timeout for that last service of its experiencing latency, it times out and prevents this chain of waiting, this cascading failure problem. Prior to something like Hystrix, where people just writing their own circuit breaker and logic randomly in their applications?

[0:38:59.0] MD: That's right. When they remembered to write the circuit breaking logic in their applications, they would do it. I think it's very common once a upon a time when people were building microsevices for weird behavior to emerge from the system, because something would be hung and you'd have to wait for network timeouts to kick in.

Then you'd go through this, enter the process of being like, "Oh, we need to make sure we put a timeout in there," and then you have to add a lot of complexity to the application, like accidental complexity to make sure that you are handling network timeouts correctly, or you're first of all detecting it and then you're then responding to properly. Hystrix just makes this just really easy to consume. It's like, if A else B. That just simplifies the whole system of building these complex microservice architectures.

[0:39:45.8] JM: I take it the same goes for load balancing.

[0:39:49.5] MD: Load balancing how? Like load balancing on the front of Cloud Foundry?

[0:39:52.3] JM: Well, in the sense of I guess the clients I had load balancer from Netflix Ribbon –

[0:39:57.9] MD: Sure. Yeah.

[0:39:58.5] JM: People writing that client side load balancing logic in their application code, as opposed to being able to specify it through config in something that the import.

[0:40:08.7] MD: Right, exactly. Yeah, I once worked at a company, we'll go nameless, where there is a team who spent probably two or three months arguing about the right way to do load balancing. Is it random? Are we going to find certain requests to certain IPs? At the end of the day, it's really just – it's come out of the behavior. You just want to have something that works mostly off the shelf that you can config and Ribbon does that.

[SPONSOR MESSAGE]

[0:40:44.1] JM: Your enterprise produces lots of data, but you aren't capturing as much as you would like. You aren't storing it in the right place and you don't have the proper tools to run complex queries against your data.

MapR is a converged data platform that runs across any cloud. MapR provides storage, analytics and machine learning engines. Use the MapR operational database and event streams to capture your data. Use the MapR analytics and machine learning engines to analyze your data in batch, or interactively across any cloud, on premise, or at the edge.

MapR's technology is trusted by major industries like Audi, which uses MapR to accelerate deep learning in autonomous driving applications. MapR also powers Aadhaar, the world's largest biometric database, which adds 20 million biometrics per day.

To learn more about how MapR can solve problems for your enterprise, go to softwareengineeringdaily.com/mapr to white papers, videos and e-books. MapR can leverage the high volumes of data produced within your company. Whether you're an oil company like Anadarko, or a major FinTech provider like Kabbage, who uses MapR to automate loan risk, and has 3 billion dollars of automated loans to date.

Go to softwareengineeringdaily.com/mapr to find out how MapR can help your business take full advantage of its data. Thanks to MapR for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:42:26.6] JM: What do you think of – since we talked about Envoy a little bit earlier, which is this service proxy that runs in a sidecar next to your application containers, that's a bit of a decoupling of the same stuff, like load balancing and circuit breaking decouples it out of the application code and into a container on its own, as opposed to the Spring model of importing Eureka, or importing Hystrix and having it run within your application, although it's partitioned because it's in a library on its own; you have its own config files. Architecturally, is that cleaner to you, or do you think it matters?

[0:43:08.1] MD: I can see it both ways. I think some people are really, really offended by the fact that there is this separate process you've got to communicate with. The advantage from my perspective though is that you don't need to have 17 libraries for 17 languages, or frameworks to take advantage of something.

A really great example of this; Steeltoe, which is the dotnet framework that corresponds to Spring Cloud services; so think I'm a dotnet C# developer, I want to use all these Netflix OSS. How do I take advantage of it? That ends up being like a relatively thin layer of calls out to the Netflix OSS. Either on [inaudible 0:43:48.0] dash you have this really bulky library to do all these things, because essentially like hey, there is a thing over there that I'm going to talk to that's got the business logic.

It's like a single responsibility principle writ large across your architecture. For me, I find that much easier to reason about than having to worry about like, is this problem in the network is the problem in my library is the problem on the server side library. Just being able to reason about like Envoy as an example is like I'm just going to connect to one place, and it's going to figure out, can I in the service mesh context, it's going to figure out for me, do I have the permissions to talk that other thing? Where is that other thing? If there is more than one, how is that being a load balance?

I no longer need to think about any of that. All I'm doing is making that one connection to one thing and everything else just happens by magic. That's really lovely, and I think that's much easier to reason about and is worth the additional complexity of having to do the separate process.

[0:44:43.2] JM: To make sure I understood what you said correctly, you're saying that people in the dotnet world, when the Netflix open source set of libraries came out like Ribbon for load balancing and Hystrix for circuit breaking, you didn't really have access to that if you're in the dotnet world, because these are all Java implementations, so you're saying that what people in the dotnet world did was spin up a server that ran these Eureka, Ribbon, HystrixJava services and then they put some kind of C# shim or API layer between them so that the C# developers didn't have to learn Java, is that right? They had their own server set up that was running Java?

[0:45:26.0] MD: Yeah, first order approximation right? That's exactly what happened, right? I'm running Spring Cloud services on my Cloud Foundry instance, but I have a dotnet app. My dotnet app can use all of these things. It can talk to Eureka. It can do service discovery, just as if it was a Java application. That's the advantage you get from having these things separated out from being just a Java library.

[0:45:48.0] JM: Do you see a lot of C# customers running C# on Cloud Foundry?

[0:45:52.3] MD: We do. We have quite a few. Pivotal has quite a few customers that are still in the windows world, or still running dotnet. I think this is a really – is a largely ignored populous. This demographic is completely being ignored by many of the open source projects. I know that Kubernetes for example has a SID Windows. They are working with Microsoft.

I think that that's not quite ready for production use yet, and I'm sure somebody in the internet can correct me about that, but my impression from lurking on some of the lists and in the SIG meetings is that it's not quite ready for production yet. That's a huge missed opportunity in my opinion, right there on really large enterprises who are still running a lot of Windows workloads.

This is not dotnet core can run on Linux new applications. These are like applications that have been around forever. Big banks probably tens of thousands of these applications still running, and that prevents them from moving to a lot of the newer platforms. One of the things that I'm proudest of in Cloud Foundry is how far we've pushed our Windows support. We can run really large, really legacy Windows applications in Cloud Foundry.

We spin up a Windows VM, we run Diego on it, it attaches and talks to the rest of the Diego cluster that's Linux VMs. The same CF push experience exists for all of those developers. They can just push their dotnet apps and they just run in Cloud Foundry, just like Linux developers have been doing for years.

This is an enormous mark an opportunity first of all and Pivotal is doing great and our customers are super happy with it. I think in open source communities, because Windows is perceived as being a difficult operating system to work with, it's expensive to work with and Microsoft is this big faceless corporation that I don't know how to work with. This is what open source contributors are saying to themselves, it's very easy to just say, "Well, that problem is too hard, I'm going to ignore it. I'm just going to focus on the Linux thing, because that's fun and it's easy. I can get meantime to happiness is really low there. Meantime the dopamine."

[0:48:06.0] JM: Yeah, the command line works just like MacBook. How did people in C# world end up on Cloud Foundry? Because I thought there was some – I haven't done any shows on this, but I thought there were some dotnet server system that was like Cloud Foundry, like a dotnet or Windows server thing. I don't really know that ecosystem very well, but what are people choosing between when they're deploying their distributed multi-tenant C# applications, they're choosing between Cloud Foundry and some Microsoft product, or not?

[0:48:43.3] MD: There has been and there are cloud products that do support Windows. Aprenda is one. Someone from Aprenda is actually the SIG lead for SIG Windows in the Kubernetes community. CenturyLink had a product many years ago, which is actually what Cloud Foundry's Windows support is based on, called CenturyLink Fog.

There have been cloud solutions for Windows applications for a long time, but I think that what I've seen personally from actually visiting with customers is a lot of people are still just running Windows server and manually provisioning these applications onto servers and trying to figure out at the operator level. Then when that VM or that hardware dies, they have to scramble to figure out well, let's fin up a new VM, let's make sure that we've provisioned the software onto that VM and it's running well. It's still a lot of toil still involved in that for most companies that I've seen.

[0:49:42.1] JM: Fascinating. Every time I do one of these shows about an area that's even slightly unfamiliar to me, I just – it's like uncovering all of these information that I had no idea exist and there must be thousands of customers that are like that. I sometimes wonder what kind of software does an oil refinery run? There must be all these domain-specific software that they run, that I've never heard of and it's always interesting going down the rabbit hole, those different areas.

I mean, I worked at a trading company very briefly and I did see some Windows applications there, because it was like some bank like systems there, tickers and stock information and so on. Certainly since moving to Silicon Valley, I just do not see any of that. I'm just inundated in the – I guess, the cloud native web 2.0 world, but is actually like 5% of workloads on the internet.

[0:50:40.7] MD: Yeah, for sure. Interesting story about oil rigs is we've actually had customers approach us about miniaturizing Cloud Foundry, so that they can run it in quarter of a rack on an oil rig. That's actually driven a lot of really interesting innovation. We do have a small footprint Cloud Foundry distribution that we can use that spins up, I think it's six VMs. We can fit in a whole Cloud Foundry to six VMs.

That came about from these oil rigs essentially having racks running on them that were essentially like, if that hardware went then somebody had to get on a boat to bring new hardware out there. Giving them the same options around VMs moving around between hardware. Thanks to vSphere and vMotion and having Cloud Foundry running on that, so the apps have a level of HI as well. That's something you're going to need on an oil rig it turns out.

[0:51:33.7] JM: Yeah. Pivotal has its own cloud and I always love to think about the competitive advantages and disadvantages of different cloud providers and how they differentiate from each other. Could you talk about the strategy that Pivotal is architecting for how to build a robust cloud biz?

When I say cloud, I mean like hosting, because I think much of the money that Pivotal makes from Cloud Foundry is through training and support and I think there is some kind of deal that Cloud Foundry has with customers who are running – even if you're running on AWS, or on Azure you make a deal with Pivotal to help with support and deployments and stuff, but that's different than what Pivotal is developing with their own cloud. Maybe you could talk about the strategic decisions that are being made around building a successful cloud business.

[0:52:32.3] MD: Sure. I'll try. The story of our public cloud product, which we call Pivotal Web Services, or P-Dubs for short, the story there actually starts with really long sale cycles with our customers. Two things I could think of; one is just like, "Hey, how long does it take an enterprise to buy software?" Just in general. You're talking six to 12 months probably is a good place to start.

What that means is is often a really long feedback loop between us building a future into the platform and giving feedback from anybody. There might be a couple people who upgrade in a timely fashion. For the most part, it was really hard to get feedback on whether we were improving the quality of product.

The second aspect of enterprise sales is if I'm a developer or I'm running a business unit at a large company, like I want to use Cloud Foundry. We signed a contract. I want to start using it immediately, but it's going to take my IT and operations team another two months to get around to actually spinning up this cluster, because they're waiting on CRUD and hardware or whatever else. They want to use Cloud Foundry right now.

P-Dubs started as a way to solve both of these problems. Number one, to give our customers an option to run their applications on day one. Okay, we have a public cloud. You could just connect in, you could push your applications there. Maybe that's only good enough for staging given your governance requirements, but it was good enough and it helped our customers get up and running on day one.

It solved the other problem as well, which is that we became our own customer zero, where we were the first consumers of our own software and we were running it in production at scale with paying customers, with SLOs. This is one of the quality of Cloud Foundry turned to the corner. Even if you asked other companies involved in the open source product, running Cloud Foundry, leading edge Cloud Foundry on P-Dubs is incredibly valuable because that is usually our first indication that something is wrong, or something is good when we make an architectural change, or any other kind of a change.

Usually when we cut in feature hits open source, before it's even in Pivotal's on-prem product, PCF, it's going to be up and running on P-Dubs usually within a few days. We immediately get operational feedback from is this causing CPU spikes, or are there network latencies or processes dying? What's going on in the system and we can analyze it?

We share a lot of these data with the open source teams that are working on these components. It really helped us tighten up our feedback loops, quality hockey stick at this point, and that upgrade and installation process I think if you ask most of our customers over the last two years has improved dramatically. That's all being driven by us running P-Dubs.

That's the origin story of P-Dubs. Now it's like, "Oh, great. We have a lot of paying customers who are running some of them production workloads in there." How do we want to build our cloud business out of that? I think that's a really interesting question that we're still exploring around how to increase this business, how to describe the value to customers.

I think it's still majority enterprise customers who are on P-Dubs. We're not attracting the startups like Heroku does. Interesting question is does Pivotal want to go there? I think we're still really pondering where we want to spend our engineering investment at this point.

[0:55:56.0] JM: Well, one place that it does seem like Pivotal has decided to spend some engineering investment is Kubernetes. I think, I'd like to close off the conversation with a short discussion of that. People who are already on Cloud Foundry, there are some workloads that they may want to deploy to Kubernetes either now or in the future, how do you see that evolving?

Well, how are people going to choose workloads that – how are they going to choose which workloads to run on Cloud Foundry, which to run on Kubernetes and how do you think Pivotal will position itself to take advantage of the expanse in workloads runtimes to Kubernetes, as well as Cloud Foundry?

[0:56:40.2] MD: Kubernetes is really attractive platform for things that are not cloud-native applications. Just to give listeners a little bit of context, Cloud Foundry does run – if you have an OCI image, Docker image, you can actually run that on Cloud Foundry. We can spin it out, we can run it, we can operate it. It's great.

You're limited within that context of exactly how your application is going to run. A top of the mind example, something that would be just perfect for Kubernetes; we have some large customers who are using a third-party application for some of their own internal monitoring and metrics. The system is actually its own cluster. It wants to spin up three instances of itself. It wants to have these things talk directly to each other. I don't know if it uses raft or whatever, but that's got some protocol for determining who is the primary and who are the followers.

Cramming that into the Cloud Foundry abstraction is really, really difficult today. Because essentially square peg and a round hole, because they know they want exactly three of these. If you have a cloud-native app, if you ask for three, you might get two, you might get four depending on are VMs rolling, did an extra process start up? It's like eventually consistent, eventually you'll have three. You might have two or four. These applications have been able to discover each other very easily. There is probably an aspect of writing a local state disk.

For all these reasons, it doesn't really fit into the Cloud Foundry abstraction. It fits really well into like a stateful set on Kubernetes. I would like to run three of these things. I want to make sure that they can talk to each other. I can configure all of that myself.

We're finding more and more vendors really want to distribute their software through helm charts, or something to get it up and running on Kubernetes. For some extent, this is a huge ecosystem play, where we want to be working with our partners. They want to be able to provide their software so that it runs on Kubernetes and we want to be able to run that side-by-side with all the Cloud Foundry applications that are probably going to be talking to it. Does that make sense?

[0:58:51.1] JM: It does. This helm charts stuff like, I've been looking at this recently and talking to some folks who worked on helm and who – like Brendan Burns. Talking to Brendan Burns about the future of application distribution, and he put this idea in my head that I've talked to some other Kubernetes people about and validated that people will want to distribute applications through helm, almost like the distributed applications across the AWS app store, for example.

You could find an easy installation for confluent version of Kafka on the AWS app store. If you wanted to do that on Azure, you would have to go onto the Azure app store and find a confluent version of Kafka, and that's because the runtimes of the VMs differ from cloud-to-cloud, but Kubernetes create a standardized system where people could install helm charts, which describe a distributed application. Am I crazy, or does that start to look like an app store? Or is it just a package manager?

[1:00:07.1] MD: You're not crazy. How I am currently thinking about it is that Kubernetes, if it hasn't already will soon become the predominant abstraction for infrastructure. Everyone is going to have hot and cold running Kubernetes, whatever cloud they happen to be on, whether it's Amazon, Google, Azure, or vSphere in their own data sensor. To that extent, things like helm providing easy way for people to trade applications, to distribute applications.

It's like, look at the package management, it's a little bit of like app store. There is something there. My personal take on it is that it needs to mature a little bit more, but there's definitely a thing there.

[1:00:48.4] JM: Yeah, agreed. Okay, well I think that's a good place to stop. Mike, thanks for coming on the show. It's a wide-ranging, entertaining conversation. I really enjoyed talking to you.

[1:00:55.7] MD: I really enjoyed it too. Thanks for keeping it interesting.

[END OF INTERVIEW]

[1:01:01.0] JM: If you are building a product for software engineers, or you are hiring software engineers, Software Engineering Daily is accepting sponsorships for 2018. Send me an e-mail jeff@softwareengineeringdaily.com if you're interested.

With 23,000 people listening Monday through Friday and the content being fairly selective for a technical listener, Software Engineering Daily is a great way to reach top engineers. I know that the listeners of Software Engineering Daily are great engineers, because I talk to them all the time. I hear from CTOs, CEOs, Directors of engineering who listen to the show regularly. I also hear about many newer, hungry software engineers who are looking to level up quickly and prove themselves.

To find out more about sponsoring the show, you can send me an e-mail or tell your marketing director to send me an e-mail jeff@softwareengineeringdaily.com. If you're a listener to the show, thank you so much for supporting it through your audienceship. That is quite enough, but if you're interested in taking your support of the show to the next level, then look at sponsoring the show through your company.

Send me an e-mail at jeff@softwareengineeringdaily.com. Thank you.

[END]