EPISODE 488

[INTRODUCTION]

[0:00:00.7] JM: Cloud Foundry is an open-source platform as a service for deploying and managing web applications. Cloud Foundry is widely used by enterprises who are running applications that are built using Spring, a popular web framework for Java applications, but developers also use Cloud Foundry to manage apps built in Ruby, Node and any other programming language. Cloud Foundry includes routing, message brokering, service discovery, authentication and other application level tooling for building and managing a distributed system. Some of the standard tooling in Cloud Foundry was adopted from Netflix open-source projects, such as Hystrix, which is the circuit breaker system; and Eureka, which is the service discovery server and client.

When a developer deploys their application to Cloud Foundry, the details of what is going on are mostly abstracted away, which is by design. When you're trying to ship code and iterate quickly for your organization, you don't want to think about how your application image is being deployed to underlying infrastructure. You don't want to think about whether you're deploying a container or a VM, but if you use Cloud Foundry enough, you might have become curious about how Cloud Foundry schedules and runs application code.

BOSH is a component of Cloud Foundry that sits between the infrastructure layer and the application layer. Cloud Foundry can be deployed to any cloud provider because of BOSH's well-defined interface. BOSH has the abstraction of a stem cell, which is a versioned operating system image wrapped in packaging for whatever infrastructure as a service is running underneath. With BOSH, whenever a VM gets deployed no your underlying infrastructure, that VM gets a BOSH agent. The agent communicates with the centralized component of BOSH called the director. This role of director is the leader of the distributed system.

Rupa Nandi is a director of engineering at Pivotal where she works on Cloud Foundry. In this episode we talked about scheduling an infrastructure, the relationship between Spring and Cloud Foundry and the impact of Kubernetes, which Cloud Foundry has integrated with so that users can run Kubernetes workloads on Cloud Foundry.

I interviewed Rupa at SpringOne Platform, a conference that is organized by Pivotal who, full disclosure, is a sponsor of Software Engineering Daily, and this week's episode are all conversations from that conference. Whether you like this format or don't like this format, I would love to get your feedback. We have some big developments coming for Software Engineering Daily in 2018 and we want to have a closer dialogue with the listeners. Please send me an email, jeff@softwareengineeringdaily.com or join our Slack channel, there's a link in the show notes to that Slack channel. We really want to know what you're thinking and what your feedback is, what you would like to hear more about, what you'd like to hear less about, who you are.

So let's get on with this episode, and thanks for listening.

[SPONSOR MESSAGE]

[0:03:18.0] JM: This episode of Software Engineering Daily is sponsored by Datadog. We know that monitoring could be a challenge. With so many services, apps and containers to track, it's harder than ever to understand application performance and to troubleshoot issues. Built by engineers, for engineers, Datadog is a platform that's specifically designed to provide full stack observability for modern applications. Datadog helps dev and ops teams easily see across all their servers, containers, apps and services to monitor performance and make data-driven decisions.

To get a free t-shirt and start your free trial of Datadog today, visit softwareengineeringdaily.com/datadog to get started. Datadog integrates seamlessly to gather metrics and events from more than 200 technologies, including AWS, Chef, Docker and Redis. With built-in dashboards, algorithmic alerts and end-to-end request tracing, Datadog helps teams monitor every layer of their stack in one place. But don't take our word for it, start a free trial today and Datadog will send you a free t-shirt.

Visit softwareengineeringdaily.com/datadog to get started.

[INTERVIEW]

[0:04:43.1] JM: Shatarupa Nandi is the director of engineering for Cloud Foundry. Shatarupa, welcome to Software Engineering Daily.

[0:04:49.5] SN: Thank you. I'm looking forward to it.

[0:04:51.5] JM: Yes, I want to start with a discussion of Spring and Cloud Foundry, because Spring is a framework for people to build web applications in Java. Cloud Foundry is a platform as a service for deploying and managing web applications, and when I was preparing for these interviews, one of the things I was trying to study was the interaction between Spring and Cloud Foundry, because these are two big products that Pivotal works on. They're open-source projects, but they also are productized. Can you describe the interaction between Spring and Cloud Foundry?

[0:05:27.0] SN: Yeah. From the very beginning we kind of wanted Cloud Foundry to be the best platform for running Spring applications. To tell you a little bit about Cloud Foundry briefly, the guiding vision for us has always been something that our VP of engineering said a while back, which is, "Here's my source code. Run it on the cloud. I don't care how." We want to make that true for all applications, but especially for Spring applications, and Spring kind of follows a similar philosophy for running Java apps. They want to be the framework that makes it really easy for you to get your Java apps up and running, and we've made several integrations which make running Spring apps on CF just as simple. We want to experience for you to be — You take your application, you go to the working director, run CF push and you have your application running on Cloud Foundry.

Similarly with Spring cloud services, you have it really easy to get several microservices up and running and have them connected to each other. We have other integrations, such as when you bind services, it's really easy for you to connect to your Spring application to databases. With all these things, we've made it so that Cloud Foundry is the best platform up there for running Spring services.

[0:06:51.2] JM: Got it. As a Cloud Foundry developer, what are the core abstractions I need to know about? Because I think there are plenty of people who are — Personally, I was a Spring developer for a while in the Java ecosystem. I didn't know anything about deploying applications. All I knew about was just developing applications. If I wanted to get started deploying something to Cloud Foundry, what are the core abstractions I need to know about?

[0:07:18.2] SN: Yeah. Kind of what you said that as a developer you know mostly about developing applications, and that's what we want to keep you focused on. One of the things that's pretty key to us on Cloud Foundry is that as you think about deploying your applications, the barrier to that is really low and you don't have to think about a lot of things. Today, we try to make the experience as much as you have your source code, you run CF Push, you know about the Cloud Foundry CLI and once you do that you see a single instance of your application up and running on CF, and that's the experience we want you to have coming into it.

I think what's also really cool is that deploying applications is day one, right? That's the first time you've got your app up and running, but the next part of that is what is day two look like? What is it look like for your application to have production traffic? What is it look like for your application to have users that increase overtime? How do you scale that application out?

The cool thing about Cloud Foundry is that the abstractions that you need to know to make that happen is also very similar to your day one where you don't necessarily need to know a lot more about CF Push, because we make it super simple for you to scale instances up. We have auto-scaling, which allows you to do that. We make it really easy for you to put store logs and query logs just by either using apps manager, which is a closed source GUI on top of the Cloud Foundry CLI, or with just a Cloud Foundry CLI which you're used to using as a developer. Even as you go into day two, the main tool you're still interacting with is the CF CLI.

[0:09:08.2] JM: There are a bunch of areas within Cloud Foundry that we could explore, we could talk about service discovery, we could talk about circuit breaking, we could talk about message passing, we could talk about databases and continuous deployment. Basically, the idea of Cloud Foundry is to take care of all of that within different components of the platform.

I'd like to start with a discussion of BOSH, because BOSH is the provisioning and deployment area. This is — I've discussed this in a couple of previous interviews. The place where is the interface between the cloud provider, like an infrastructure as a service provider, and Cloud

Foundry itself. So BOSH is this layer of infrastructure management. Can you explain what BOSH is doing?

[0:09:59.6] SN: Yeah. BOSH is — I really like talking about BOSH. I spent a lot of my Cloud Foundry engineering time on BOSH. Very high-level, BOSH is like a toolkit that helps you with release engineering and deployment and lifecycle management of distributed systems. Breaking that down, what we want your experience to be is as someone who writes a BOSH release, you have a way to get for you to put your opinions of how to run your software, how to start your software, how to monitor your software in scripts that then BOSH takes over and is able to run the software in a repeatable and consistent manner across different IaaS's, right? You don't have to worry about, "Do I run this in a t2.nano? What does that translate to on Azure? What does that mean on vSphere?" That's something BOSH takes care of. You as a service author, release author can just say, "Hey, I know that for running this monster node, I need X-gigs of RAM and so much persistent disc and two CPUs, and BOSH translates what that means in terms of different IaaS's. BOSH takes care of, if you have these networks on these different IaaS's, how should they be assigned to IPs, how should they talk to each other? And it brings your cluster up and running for you. This is your day one with BOSH. This is how you get your deployment out there.

The other thing that BOSH does goes back to a bit of our day two story, which is something that we focus a lot on in CF. Anything that we put out there, we wanted to have a very strong day two story. Similarly with BOSH, we focus a lot on, "Okay. Now that your software is up and running. What happens if there's a security vulnerability in the Linux kernel? What is that look like for your software? How do you redeploy all of your software and make sure it's a rolling deploy so you're not taking downtime and make sure that you have the newest version of the Kernel running within a certain amount of time?" We are constantly releasing new stem cells and providing ways for you to BOSH deployment and get your software up on the latest kernel.

The other really cool thing is that it does health monitoring for your cluster. BOSH is constantly keeping a watch on all the VMs that are part of your deployment and it's keeping an eye out on the processes running on those VMs and it has the ability to restart those processes and recreate VMs even if, for example, a data center goes down or an availability zone goes down.

Those are some other aspects where it helps you with not just getting your release out there, but also monitoring your release when it's out there and making sure that you have a good story for how do you update it overtime.

[0:12:55.1] JM: You mentioned the abstraction of a stem cell, which is a versioned operating system image that is wrapped in packaging for whatever IaaS you running underneath IaaS, infrastructure as a service. When we're talking about these operating system images that are getting wrapped, I guess with this — For example, if I'm running on AWS, is this like the abstraction of an AMI, like an Amazon Machine Image and this gets wrapped in a stem cell that we have the proper interfaces for Cloud Foundry to deal with?

[0:13:31.9] SN: Yeah. It is similar to an AMI. It is a machine image. It's typically — There's like Linux stem cells. There are Windows stem cells, so they're platform specific. However, what we try to do is that as much as possible, these images are the same across different IaaS's.

The Linux kernel, if you picked an Ubuntu particular version stem cell, it's going to be very similar environments on Azure versus on AWS versus on vSphere, and then the other cool thing is that on the stem cell we've done a lot of security hardening. We've played around with Linux configurations to make sure that this is up to the security standards of enterprise software. So you can feel comfortable running your software on top of it.

[0:14:25.0] JM: Okay. When in the lifecycle of a deployment does a stem cell get created?

[0:14:32.5] SN: A stem cell is usually created and published by either people in the community of the BOSH team, and so this is something we publish on BOSH.io, which is our open source website for hosting most BOSH artifacts and BOSH docs and it's just a place to get started. There is a whole slew of stem cells for all of our supported IaaS's and platforms on there. Some of them created by Pivotal, some of them created by the community.

Now, when you use the BOSH CLI to upload your stem cell to the BOSH directory, which we'll talk about in a little bit. Effectively, BOSH director is kind of the like centralized brain of BOSH. Once you upload the stem cell, you can reference what stem cell you want in your deployment manifest, and your deployment manifest effectively specifies the deployment topology for your software. How many VMs do you want? What's running on each VM? What VM is talking to which other VM? That kind of thing is all specified in your deployment manifest, and you also specify what stem cell version.

The first thing that BOSH does when it deploys is it boots up AVM with the stem cell image that you specified. From then on it goes on to put the BOSH agent on there and put your bits on there as well.

[0:15:55.5] JM: Okay. Tell me where I'm getting this wrong. Let's say I'm deploying my Cloud Foundry platform to Azure, for example. When I deploy that, I get an instance of BOSH which at the first deployment is just going to be the director mostly, and the director is the centralized manger of the other stem cells that I'm going to eventually deploy. I give to the director a stem cell image that I'm going to say — Or a collection of stem cell images and I say, "Hey, whenever I'm spinning up a new service on this Cloud Foundry platform, I want you to look in your directory of different image templates and spin up an instance of a server that is from the appropriate instance that you are hosting director," and the director knows how to do that. Am I articulating things correctly?

[0:16:52.5] SN: Yeah. It's pretty close. Usually, BOSH is the one that deploys a Cloud Foundry platform. You would start off with a BOSH director, and there's a couple of ways of getting that. There's a CLI to get a BOSH director, or you can use operations manager, which is a closed source GUI. You have a BOSH director.

The next thing you do to a BOSH director is upload a stem cell. Typically, we publish some stem cells and those stem and say, "Hold up two different security audits." They automatically are approved by a lot of enterprises. You upload the stem cell on to the BOSH director.

The next thing you do is you upload your release. Let's say you wanted to deploy Hadoop. You upload a Hadoop BOSH release, which you can either write yourself or you can get from community releases that are out there on BOSH.io, so you can just download it from there. Now, your BOSH director has a stem cell, so it has an operating system on which you're going to run and it has the software you want to run, which his your Hadoop release.

The next thing you do is you write a deployment manifest. This is where you specify, “Okay. This is my master node. These are the bits that should run on that. These are all the different Hadoop workers. This is what should run on that.”

Then with that deployment manifest, you can run a BOSH deploy command. The BOSH deploy command translates your deployment manifest to IaaS’s concepts. This is where it will create a new VM in your IaaS’s of your choice, and the first thing that comes up on that VM, that VM gets booted with is the stem cell image that you’ve uploaded. Once the stem cell is up there, you have a VM which is a basic Linux kernel running. That’s when BOSH puts on the BOSH agent. The BOSH agent is a small binary that lives on every single BOSH-deployed VM and its job is to constantly ping back to the director. It’s constantly talking back to a director and saying things like, “I’m healthy,” or “here are the processes I’m running.” It takes commands from the director which say like, “Start this. Stop this. Run this script now.” Things like that.

Once the agent is up, it then gets commands from the director to be like, “Okay. This VM is going to be my master. Run these bits on it.” It starts running that and monitoring that. Once the whole fleet of VMs has finished getting all their bits as well as started all the processes and all the agents say everything is healthy, that’s when BOSH director says, “This deploy is successful,” and you’re able to route to your VMs and so on.

[0:19:39.1] JM: Okay. Can a stem cell be a container as well or it can only be a VM?

[0:19:45.1] SN: A stem cell is just an image. So you could have it be a container by using an infrastructure that deploys it to a container. If you use the Docker cloud provider, what it’ll do is it takes your stem cell image and deploys it to a container, and so you have a Docker container running the Linux stem cell image. However, if you chose to use like the AWS cloud provider, then you would end up with that stem cell image running on AVM. It depends on what you want to back it with.

[SPONSOR MESSAGE]

[0:20:26.2] JM: Simplify continuous delivery GoCD, the on-premise open-source continuous delivery tool by ThoughtWorks. With GoCD, you can easily model complex deployment workflows using pipelines and you can visualize them end-to-end with its value stream map. You get complete visibility into and control of your company's deployments.

At gocd.org/sedaily, find out how to bring continuous delivery to your teams. Say goodbye to deployment panic and hello to consistent, predictable deliveries. Visit gocd.org/sedaily to learn more about GoCD. Commercial support and enterprise add-ons, including disaster recovery, are available.

Thanks to GoCD for being a continued sponsor of Software Engineering Daily.
[INTERVIEW CONTINUED]

[0:21:26.9] JM: We mentioned this centralized component called the director. What's the fault tolerance model for the director? If my director dies, what happens?

[0:21:37.9] SN: Yeah, this is a really good question. This is something we've given a lot of thought to. Something that's coming on our road map is a way to make the director itself be more highly available. Today, when your director dies, it has a database, and so it's pretty easy for you to like backup and restore from the database and you have a low meantime to recovery for the director itself. However, the way BOSH is architected, when the director dies, it doesn't mean it affects any of your existing deployments.

Your existing deployments continue to run as usual and are available. It does mean that you can't make any changes to them or you can't make new deployments until the BOSH director is back up, which is typically has a pretty low time to recovery. However, we are working on stories to make BOSH director itself be more highly available.

[0:22:37.6] JM: Sorry. Who restarts the director when it goes down?

[0:22:41.7] SN: There is a BOSH CLI command with which you can bring up the director. BOSH has bootstraps itself, and so you can either do that with the BOSH CLI or with operations manager.

[0:22:54.0] JM: Okay. You have to do it manually or —

[0:22:57.7] SN: Yeah.

[0:22:58.2] JM: Okay. Interesting. When somebody is deploying BOSH, do they do anything in particular to, I guess, have alerting around that director? If I'm a cloud provider and I'm offering Cloud Foundry — If I'm Azure, I'm offering Cloud Foundry as a service. Do I have to do any particular monitoring around the director? How does that work?

[0:23:26.3] SN: Yeah. We have a product called health watch, and that is a separate closed source product, and what that does is it constantly monitors the BOSH director API and it looks for is the director API up and available, and if not, it alerts you.

A lot of other folks who are not in the open-source community who are not using health watch, I think they have Datadog or something in that setup to just ping the director API and make sure it's up and running.

[0:23:58.5] JM: Sure. Make sense. The BOSH agent is something that gets deployed on all of the VMs that get deployed by BOSH. BOSH spins up these VMs and puts agents on them and then the agents communicate between these random VMs that are services and the director. What does the agent do? Can you talk a little bit more about the communication pattern between an agent and the centralized commander?

[0:24:30.0] SN: Yeah.

[0:24:31.6] JM: Sorry. Director, not commander.

[0:24:33.7] SN: The BOSH agent, it communicates with director over a message bus. It has like dedicated message bus that it talks over. Typically, the command set BOSH agent receives. We try to keep that pretty simple so that BOSH agent doesn't have a ton of inherent complexity in it. Usually, it's things like start this, stop this, or heartbeat.

Most of the time the traffic is heart beating. When you want to run something, it just gives you something like run the script, and agent finds it on the file system and runs it.

[0:25:15.6] JM: Okay. Keeping with our theme of talking about the lower level aspects of Cloud Foundry, hopefully we'll have time to talk about some high-level stuff, but when Kubernetes came out, what did you see is the opportunities for synergies between Kubernetes and the pre-existing Cloud Foundry platform?

[0:25:36.9] SN: Yeah. I think we've always wanted Kubernetes to be like an and game, so we want it to be Kubernetes and BOSH, and I can talk a lot about how BOSH and Kubernetes play really well together.

With Kubernetes today, a lot of the advantages that you get is once you have your Kubernetes cluster up and running, and getting to that point of deploying Kubernetes has often been — People have had to had their own scripts, or Terraform or something there. This is where BOSH really comes in and helps. Once you're able to encapsulate your opinions and the community's opinions around how to run a Kubernetes cluster and you have a BOSH release for it, from then on, BOSH takes over and you have a repeatable and consistent way of deploying Kubernetes. You have a way in which you can now specify IaaS's agnostic concepts in your release or in your deployment manifest, and you have a really good way of getting Kubernetes to not just Google, but also AWS and Azure and vSphere and all of the other platforms.

I think the other great thing is what it looks like to upgrade your Kubernetes platform. Once you have your Kubernetes cluster and you want to keep it updated, because Kubernetes has a very fast pace of delivering features if you want to keep it updated. BOSH already has good ways of doing rolling upgrades. So you can have zero downtime on your running apps on the cluster while you're still updating your Kubernetes cluster, and that's where I think BOSH really has a strong story with Kubernetes in terms of operators running Kubernetes.

In terms of like application developers on Kubernetes, I think what we've tried to focus on is providing a single control plane for all of the Kubernetes clusters that are running as well as providing plans and quotas and so on. So you just have better way of monitoring usage on each of those clusters. These are some of the things that we've often heard folks on the Kubernetes community want as they are operating and providing Kubernetes as a service to a slew of people.

[0:27:59.0] JM: Right. Yeah, when I was talking to [inaudible 0:28:01.9], he was saying there are certain workloads that you as a company might not want to run on Spring, or might not want to run on Cloud Foundry, and you could run these on Kubernetes. You would have a deployment where if you've got your Cloud Foundry already deployed to Azure, that means that you have BOSH deployed to Azure, and BOSH can be the — Again, can serve as the middleware between the cloud provider's raw infrastructure as a service, and now instead of orchestrating resources for cloud foundry, it will orchestrate resources for Kubernetes, and you can run whatever you want on Kubernetes and have BOSH taking care of things that are difficult to implement in Kubernetes, such as high availability.

The raw things that you're going to want out of any Kubernetes cluster, but are not easy to roll your own for today. Right. Examples of things that you would want to run on Kubernetes that might not be a good fit for Cloud Foundry, if I'm like a bank, for example, what would I want to run on? If I'm saying, "Okay. I've got Cloud Foundry. It runs most of my workloads, but there are some banking related things that are not going to run on Cloud Foundry." What are some examples of things I'd want to run in Kubernetes instead?

[0:29:28.3] SN: Yeah. Today, on Cloud Foundry and with the apps going cloud-native, I think we often make the assumption that most apps are 12-factor apps and they don't have any particular state aside from in the database, and that's a great world. Developers really enjoy the world, but there's a lot of legacy applications, which are not yet quite there or even newer applications which really need access to the file system while they're running, and they don't have the ability to become 12-factor apps at the moment.

These apps, it's really difficult to run them on Cloud Foundry, and this is where Kubernetes is hugely powerful, because you have the ability to run your own Docker images on there and you can choose to put what you'd like in those Docker images.

I think a main part of us bringing Kubernetes into the platform allows users to bring more workloads into the platform. Earlier, for you to move some of your legacy banking applications on to Cloud Foundry, you would have to rewrite them to be 12-factor applications. Now, you can kind of take them in their existing form as long as you're able to create trusted Docker containers out of them, which a lot of enterprises are investing time into doing that. You'll be able to bring these workloads to the Cloud Foundry platform and leverage some of the good stuff that BOSH gives you while also keeping the advantages you'd get from running it on Kubernetes.

[0:31:16.2] JM: What is that term 12-factor apps mean?

[0:31:19.1] SN: 12-factor applications, it's something that came out I think from the Rails community and it refers to principles you follow to make your application easy to iterate on and easy to deploy in multiple different environments. One of the big parts of that, and I think often the part that's most difficult to accomplish is to not keep a local persistent state and always rely on your app can get restarted at any point and all of your persistent state is in a database somewhere where you can recover from really easily.

[SPONSOR MESSAGE]

[0:32:07.9] 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 email, 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 improve themselves.

To find out more about sponsoring the show, you can send me an email or tell your marketing director to send me an email, 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 email at jeff@softwareengineeringdaily.com. Thank you.

[INTERVIEW CONTINUED]

[0:33:35.2] JM: So in the Kubernetes environment, I can mount a file system, I guess and actually write to that file system, and if the Kubernetes node dies, then I can still have that file system around, and Cloud Foundry doesn't give me that same type of abstraction, I guess.

[0:33:59.8] SN: Yeah.

[0:34:00.2] JM: Okay. Interesting. One of these reasons to have your own — To have a Kubernetes provider, like a container service. There are all these — Amazon has a container service. Google has a container service. They all have container services that handle your Kubernetes cluster. One of the reasons to do this is because high availability Kubernetes is difficult to do on your own. Why is that? Explain what makes high-availability Kubernetes difficult.

[0:34:32.5] SN: One of the things that makes HA Kubernetes really difficult is running etcd in HA mode. Running a multi-node etcd cluster and keeping it from having significant — Or keeping it from having downtime, especially when upgrading the cluster. It's really hard.

A bit part of that is because of the consensus mechanism that etcd has and how it has its own decision making around when the cluster is healthy, around at what time can you turn the cluster off without losing data and keeping that both available and consistent at the same time is hard. It's a CAP theorem.

Etcd focuses a lot on consistency, which makes it hard for upgrades, makes it hard to just keep up and running. This has been a big challenge for us and with the etcd BOSH release, we've focused multiple years of effort, I'd say, into us trying to get an HA etcd running, and that's a big part of the value proposition of a Kubernetes BOSH releases. You get some of these learnings and these opinions that we've had by running etcd as HA.

Additionally, BOSH has a first-class concept of availability zones. These are availability zones that are different from your infrastructure availability zones. You can choose to tie them together, but it's a first-class concept in BOSH. What you can do today with a Kubernetes release is you can choose to stripe your master and worker and etcd nodes across different availability zones.

BOSH will take care of — If you have 3AZ's to stripe your Kubernetes deployments such that the masters are in different AZ's, and if one goes down, you still have another one up and running.

Aside from multi-AZ, it also gives you the ability to run it across multiple datacenters, across multiple clouds even if you wanted it to. That's, I think a big part of the proposition of what used to make running Kubernetes in HA hard that is made a lot easier by BOSH.

[0:37:03.0] JM: Yeah. Okay. Etcd, for example — What does etcd accomplish in a Kubernetes cluster? What does it do for you?

[0:37:09.3] SN: Etcd is the main data store that backs a Kubernetes API. Anything about what is running on what container, where, where it's scheduled, what IP, how to reach it. All of that information is in the etcd. If the etcd goes down, the Kubernetes API would very soon take downtime.

[0:37:29.5] JM: Okay. Oh, I see. If I'm deploying my own Kubernetes, I need to make sure that etcd is running all the time. Yeah, okay. Etcd is not highly available out of the box. What's hard about making etcd — I know we're getting really into the weeds here, but I'm just curious, because I've seen this question a number of different places. What's hard about making etcd highly available? What do you have to do in order to make it highly available?

[0:38:03.9] SN: That's a really good question. We are still learning a lot on that front, but I think the main learning for us has been that etcd has a consensus mechanism called Raft, and with Raft there is very particular points at which it says that, "Okay. This data is now on all three of the nodes, or above half where it can replicate to all three, right?" It has its own mechanism for when it would say that the data is in a non-corrupt state and that this data is going to be replicatable. For you to make decisions about the cluster while it itself is making decisions about the state of its own data consistency, there is often conflict in those two things.

I think the big thing that etcd tries to accomplish is both to be consistent and to be portioned tolerant by using Raft, which means you have to compromise an availability. That what makes it challenging to make etcd HA.

[0:39:18.7] JM: Okay. I'm just going to keep going on low-level stuff, because one of your colleagues told me to ask you about this. Container networking is a popular topic these days. What are the improvements that have been made in container networking in the last three years? Maybe you could even just discuss what is that term even mean, container networking? What does that term mean?

[0:39:42.2] SN: This is a really exciting part of my job. Container networking as a field today, it's new and it's what everyone wants to be involved in. I think that it started off by just us or the industry coming on to containers, and very soon realized that, "Hey, if I have these different containers running on these VMs and I have these microservices kind of an architecture, why do I keep having to go out to a router or to some kind of gateway to communicate? Why couldn't I just go container to container? From there, container networking came about and it was a really exciting field with a lot of developments, a lot of different players in the field. I think some of the things that I personally have been following and have been exciting to watch I the standardization of the CNI or the container network interface.

Pivotal has been involved or Cloud Foundry has been involved a lot in that and our part, like contributors to CNF and to the container network interface. We have our own implementation of the CNI on Cloud Foundry called Silk, and Silk lays down an overlay network and allows you to go container to container.

This is another place actually that we work very closely with Spring on. One thing that I've learned overtime about container to container networking is that just the networking part by itself doesn't bring as much value as service discovery, because in the end what you want to know is not just what container do I need — Now just now to talk between the containers, but also what container do I need to talk to. That's where our service discovery comes in and that's where Spring cloud services provide some of that for you.

While the platform Cloud Foundry provides this container to container networking and the ability to talk there, what Spring cloud brings in this service discover piece. That's been one thing that has been new and exciting in the container to container networking world.

Additionally, we've also made it so that you don't have to stick with the Silk CNI. As you're aware, there's a lot of other folks who also provide CNI, like NSXT provides a CNI, Calico provides. There're a lot of others, and we want to make it so that you can bring your own CNI to the platform. We're working on an NSXT integration, which hopefully very soon you'll be able to use NSXT in place of Silk CNI if you want it to. That's another part of what makes container networking challenging, I think, is that not only do you care about how do you communicate between the containers. You care about where the other container is, but also should you be allowed to talk to that container.

Policy becomes a really important part of container networking and NSXT tries to approach some of that with its own implementations of firewall rules and so on. Similarly on CF, we have our own policy server and application service groups which address that. Kind of as you talk about all three of these; container networking, policy and service discovery, I think the immediate next thing especially in today's industry is to talk about Istio and the idea of a service mesh, right?

[0:43:08.8] JM: Yes.

[0:43:09.8] SN: That's something we've actually been following along and we're really excited about, because we do have all three of these and in some preliminary state on CF and we really want to take it to the next level by getting Istio integrated as well into CF.

[0:43:24.8] JM: Okay. All three of these, meaning container networking, servicing proxying and service mesh, or what do you —

[0:43:31.0] SN: Container networking, service discovery and policies, so policies between applications today.

[0:43:38.8] JM: Okay. Something in there I don't quite understand is what's the different between the container network — We did a show about this. I should be retaining the information better. We did a show about container networking, with somebody from Red Hat, who I think was leading the CNI efforts, the container networking interface efforts. What is the difference between the container networking interface layer and the service proxy layer, or is there a difference?

When you're talking about Istio and Envoy, Envoy is the service proxy that Istio has built on. Does Envoy use the container networking interface? Or am I mixed up on something?

[0:44:18.9] SN: The container network interface is something that's really low level. Today, what happens is when, Garden, which is the part of Cloud Foundry that creates a new container, when Garden brings up a new container, it talks to the container network interface to configure networking on that container. That's where the CNI plays a role. It's like at the very beginning, at the container creation time.

Envoy is something that is on the container and is collocated on there as a sidecar just throughout its life cycle. So any outgoing traffic from this container goes through an Envoy if you choose your IP table rule set, Envoy access a proxy. What this gives you is that from your perspective as a local app developer, when you're developing our app, testing your app, all you need to do is figure out how it's interacting with Envoy. Envoy just takes over and acts as the network for you when you're developing your app and so on.

Aside from give you a lot of features like TLS or mutual TLS or load balancing, draining of applications. Aside from that, I think just the simplicity that it brings when you're developing an application where you no longer have to think about what the network means, what about latency, what about mutual TLS. All of these things is just abstracted away by Envoy and I think that's what's really, really huge about the idea of a service proxy or about Envoy.

[0:46:02.5] JM: This stuff is all relevant to Pivotal, because you're talking about allowing Cloud Foundry customers to also deploy a Kubernetes alongside their Cloud Foundry clusters and you want to be able to have BOSH, I assume, setup the correct container networking or the correct, I guess, service proxying. Am I understanding that correctly?

[0:46:34.8] SN: BOSH is usually a layer of abstraction below this layer. Typically, what BOSH will do is BOSH brings up VMs on which the Cloud Foundry platform is running. Then within Cloud Foundry, there is a thing called Diego, which is a container scheduler, and then there's

Kubernetes, which is they're kind of side by side. Diego is where you can run CFI applications and you've been able to run CFI applications all this while. Then Kubernetes is the CF container runtime, which his new. What we want is these Envoys, in each of the Diego cells, which is a Diego VM, there is a bunch of containers. Each of these containers will have Envoys on them similar to how containers on Kubernetes workers will have Envoys on them.

[0:47:30.2] JM: Wow! Okay. Interesting. When I was talking to [inaudible 0:47:37.1] about managing Cloud Foundry — Well, I guess managing Pivotal more broadly for him. One of the things he mentioned was the granularity of the testing infrastructure. This is really highly critical infrastructure and Cloud Foundry runs in a bazillion different environments, right? It runs on premise, it runs on all of the different cloud providers. How do you test this software?

[0:48:08.4] SN: This is a really good question. I would spend a lot of my time thinking about it. As an engineering on a CF team, I think you spend about 40% to 60% thinking about and developing tests at different layers. One very interesting that we do is tester and development. So for every line of production code we write, we try to write a failing test first and then write this code to make it go green.

Initially from that, we also have integration test coverage, all the way up to performance test, load tests, and these are all just testing individual components. Once a team has its own concourse pipelines with all these kind of component, like tests, that test the component green, we then have teams dedicated to building Cloud Foundry and running it on different IaaS's. We have teams that builds pipelines which deploy our latest release candidates to all the different IaaS's we support. Then we run CF acceptance test on it.

We've invested a lot of time in building acceptance tests at every level, so we've built CF acceptance text, we've build BOSH acceptance test, we've build container network acceptance test. I think aside from giving the teams confidence that the bits they've build are functioning, what this does that's pretty cool is that it runs a smoke test in a customer environment.

We ship it as part of our product and we encourage customers to run it, because it gives you confidence in knowing that you've configured your environment correct since all the functionality that we verify and test against doing development is now working on your environment.

This is also an important part of contribution model, so say tomorrow I talk with BOSH about the cloud provider interface or CPI and we wanted to be so that we are always expanding the number of IaaS's we support, and so we want people from the community to write CPIs and to maintain them with us. An important part of what we ship along with our CPIs is the certification suite for it. If you come in tomorrow and wanted to write a CPI for your own IaaS, you would have an API in docs to develop against, but once you're done developing, you can run the certification suite and gain confidence in whether the behavior will be the same and the experience will be the same regardless if it's someone's using a well vetted AWS CPI or if they're using your own CPI.

[0:50:55.0] JM: Yeah. It makes sense. It's pretty interesting. Zooming out, I know we're up against time and we literally only talked about like low-level orchestration. I intended for this show to be Cloud Foundry, but we talked more about BOSH. The thing is — So you're director of engineering for Cloud Foundry. Cloud Foundry is, like I said at the beginning, it's like all these different things, like message bus and circuit breaking and all these other services that are inside of it. Can you talk a bit about management? Like how do you keep up to date with all of those different components of Cloud Foundry and how do you delegate things?

Also, how do you manage cross-cutting concerns? When you have some component of the system that needs to do something in the circuit breaking and also needs to do something in the message passing area, how do you orchestrate the different teams to work together? Tell me about management, five minutes or less.

[0:51:56.5] SN: This is probably the most difficult part of my job.

[0:52:00.9] JM: People.

[0:52:01.0] SN: More than the people. Just keeping up with everything. There is so much going no. We are distributed across so many different locations and everyone is working on really interesting stuff. My email is always overflowing. Just keeping track of what people are working on and where I should divert my attention to, whereas where I can just gloss over the details is always something that — It’s the toughest part of my job and it’s something I continue to iterate on.

I think, for me, I try to apply as much as possible like our principles of build, measure, learn and iterate to management. Especially as an engineering director who’s been in this role for less than a year, I have been trying to do a combination of spending some time peering in with the teams and actually solving engineering problems with them. During that time, I tend to stay away from more coordination and so on, versus sometimes I am completely focused on like closer to our release cycle. I’m completely focused on, “Are these features integrating well together? What is it look like for our timelines? Where are the blockers? I’m wearing my coordination hat a lot.

Versus at other times when it comes to — Because a big part of my job is also coaching and building the next set of leaders, so coaching and working with anchors, coaching and working with managers. Usually, it’s very difficult for me to describe whether I spend what parts of my day doing each of those. I think for me, the best way of thinking about it has been when I’m focused on something, I focus most of my time on that thing and I just wear different hats and switch context differently depending on what signal I get.

[0:53:58.7] JM: You serialize it. It’s not multitasking. It’s serialized.

[0:54:02.2] SN: For me, I think serialized works better than multitasking. I think, for me, usually there is one thing that I’m very focused on and I’m paying active attention to and one thing on the backburner. Typically, having more than two things is I’ve seen leads to no progress on either front. Usually, I’m very quick in terms of making decisions of what those things are or switching those things out if needed. But I try at a time to focus on one thing primarily and one thing on the backburner.

[0:54:38.4] JM: Okay. All right. What’s the one thing on the backburner while you’re having this conversation? You don’t have to answer that.

[0:54:48.0] SN: Well, we’re really close to our release right now. We try to maintain every three months release cadence and we are coming up on that. A lot of the things that I’ve been thinking about right now are related to what's blocking the release, how can we make it, how can we make it move forward? What teams need help? What teams are done shipping their artifacts and what's next for them.

[0:55:12.1] JM: Okay. All right. Well, Rupa Nandi, thanks for coming on Software Engineering Daily.

[0:55:15.4] SN: Thank you. It was fun.

[END OF INTERVIEW]

[0:55:18.5] JM: You are building a cloud-native application and you need to pick a cloud service provider. Maybe you're just starting out with a new app, but you have dreams of scaling into the next giant unicorn. Maybe your business had been using on-premise servers and you want to start moving some of your infrastructure to a secure cloud provider that you can trust. Maybe you're already in the cloud, but you want to go multi-cloud for added resilience.

IBM Cloud gives you all the tools you need to build cloud-native applications. Use IBM Cloud Container Service to easily manage the deployment of your Docker containers. For serverless applications, use IBM Cloud Functions for low-cost, event-driven scalability. If you like to work with a fully-managed platform as a service, IBM Cloud Foundry gives you a cloud operating system to control your distributed application.

IBM Cloud is built on top of open-source tools and it integrates with all the third-party services that you need to build, deploy and manage your application. To start building with AI, IoT, data and mobile services today, go to softwareengineeringdaily.com/ibm and get started with countless tutorials and SDKs. You can start building apps for free and try numerous cloud services with no time restrictions. Try it out at softwareengineeringdaily.com/ibm.

Thanks again to IBM for being a new sponsor. We really appreciate it.

[END]