EPISODE 668

[INTRODUCTION]

[0:00:00.3] JM: Engineering organizations can operate more efficiently by working with a continuous integration and continuous deployment workflow. Continuous integration is the process of automatically building and deploying code that gets pushed to a remote repository. Continuous deployment is the process of moving that code through a pipeline of environments; from dev, to test, to production. At each stage the engineers feel increasingly safe that the code will not break the user experience.

When a company adopts Kubernetes, the container orchestration solution, the workflow for deploying software within that company might need to be refactored. If the company starts to deploy containers in production and is managing those containers using Kubernetes, the company will also want to have a testing pipeline that emulates the production environment using containers in Kubernetes.

Sheroy Marker is the head of technology at ThoughtWorks Products where he works on GoCD, a continuous delivery tool. Sheroy joins the show to talk about how Kubernetes affects continuous delivery workflows and the process of building out Kubernetes integrations for GoCD. We also discussed the landscape of continuous delivery tools. Why are there so many continuous delivery tools? There is a question of how to choose a continuous delivery product if you are implementing continuous delivery.

Continuous delivery tooling is in some ways the space of monitoring, or logging, or analytics, there are lots of successful products in the market of monitoring and logging and analytics and they make different trade-offs in user experience, in how heavy the agent that runs on your host is. Continuous delivery tools are like that. There are different tools for different people and we talk about GoCD in particular, but we also address some of the other tools in the market like Jenkins me of the cloud offerings from AWS, or Google Cloud. Full disclosure, ThoughtWorks and GoCD are sponsors of Software Engineering Daily.

[SPONSOR MESSAGE]

[0:02:09.4] JM: Accenture is hiring software engineers and architects skilled in modern cloud native tech. If you're looking for a job, check out open opportunities at accenture.com/cloud-native-careers. That's accenture.com/cloud-native-careers. Working with over 90% of the Fortune 100 companies, Accenture is creating innovative cutting-edge applications for the cloud. They are the number one integrator for Amazon Web Services, Microsoft Azure, Google cloud platform and more.

Accenture innovators come from diverse backgrounds and cultures, and they work together to solve clients' most challenging problems. Accenture is committed to developing talent, empowering you with leading-edge technology and providing exceptional support to shape your future and work with a global collective that's shaping the future of technology.

Accenture's Technology Academy established with MIT is just one example of how they will equip you with the latest tech skills. That's why they've been recognized on Fortune 100's best companies to work for list for 10 consecutive years.

Grow your career while continuously learning, creating and applying new cloud solutions now. Apply for a job at Accenture today by going to accenture.com/cloud-native-careers. That's accenture.com/cloud-native-careers.

[INTERVIEW]

[0:03:47.7] JM: Sheroy Marker, you are the head of technology at ThoughtWorks Products. Welcome to Software Engineering Daily.

[0:03:51.9] SM: Hi Jeff. Thanks for having me.

[0:03:54.0] JM: It's great to have you. Today, we're going to talk about continuous integration, continuous delivery and some aspects of Kubernetes and how Kubernetes is changing those things. These are some pretty big topics, so I think we should start with basics. How do you define the terms continuous integration and continuous delivery, particularly in 2018, what does that term mean?

[0:04:16.7] SM: We try to stick to the definition of these terms in the continuous delivery book by Jez Humble. The CD book was written when Jez was the product manager for our continuous delivery product GoCD. In the book, continuous integration is defined as the practice where every time somebody commits a change, the entire application is built and a comprehensive set of automated tests is run against it. Continuous delivery is defined as the ability to get changes of all type including features, configurations, bug fixes, stuff like that into production safely and quickly and in a sustainable way.

CD, if you think about it from that perspective is a pretty essential component of any software delivery practice. Regardless of the target development environment and in 2018, you have a variety of modern infrastructure stacks, but regardless of your target infrastructure stack, you still have a design a CD workflow to get your software changes into production.

[0:05:22.3] JM: The CD, the higher level CD workflow where lots of changes are integrated into a software product continuously, does that term refer to, I don't know, like changes to my software design, or my A/B testing procedure, because continuous integration is the continuous integration of code via a pipeline. What's the difference there and where – is CD more about organizational procedures? Tell me more about the difference there.

[0:05:53.3] SM: The key difference is that CI, or continuous integration is a small component of CD and CD is a more larger, all-encompassing workflow. The way we think about CD is that it's an automated path to production. Regardless of what you're trying to get into production, whether it's software changes, or if you're trying to enable your teams to experiment quickly with A/B testing, you need to have a automated part to production that automates every aspect of the process of getting these changes into your production environment.

CI would be one initial stage of that process, where you build a software component and then usually follow that up with multiple automated test stages and deployment stages, and then eventually automated deployments to production. That end-to-end path is usually what a CD workflow would model.

[0:06:50.3] JM: A continuous integration would just be one point along that, like I have shipped my code to Github and there was an automated hook that integrated that code into at least stage one of the pipeline, that was one instance of continuous integration. Continuous delivery is more the encompassing pipeline of all of the stages of the code moving through that pipeline.

[0:07:13.2] SM: Exactly. Again, all the stages being highly automated with eliminating as many manual steps in the process. The reason that is important is so that you get a consistent end-to-end pipeline for changes. If you were to push say a new experiment, or a fix to a production issue, it's important to have a consistent part of production, which is fully automated.

[0:07:40.5] JM: There are a lot of CD tools. I was talking to an investor recently about winner-take-all markets in software, and he was talking to me about monitoring software. I was like, "Why would you invest in monitoring software, because it's not winner-take-all? You have all that – you have a super-crowded market, you've got New Relic and Datadog. Why would you want to be in any of this?" Then you pointed out that all of them are really big successes, so you don't necessarily have to have this monopolistic winner-take-all world.

It's not necessarily a bad sign that there are many monitoring tools, because different people have different monitoring practices. I think the same thing goes for CD tooling, because every software organization should be doing some continuous delivery – I mean, not to be prescriptive, but I wouldn't want to work at somewhere that didn't have continuous delivery somewhere on the roadmap. Maybe it's when we get to 10, or 20 people, but eventually you want something like that. What are the subjective decisions between different CD tools? What are the different workflows that different organizations have in continuous delivery?

[0:08:53.1] SM: Yeah, for sure. As we mentioned earlier, CD's pretty instrumental in your software delivery practice, but what we're seeing is there are multiple operational models. There is SaaS versus on-premises, and then there is the open source category of tools versus proprietary and license-based. There are products that cater to all of these different market segments, and then there are also competing philosophies around how to implement CD workflows.

Some tools are specifically designed exclusively for CI. You have the earlier versions of Jenkins in that category, you have SaaS tools, like Travis CI that fit in that category. Then you have some that are specifically designed to model deployments with pipelines for CD workflows, and some of these give you the capability of modeling end-to-end CD pipelines. It's really how you think about this and how much of this process you want to automate.

Also, there is a lot of innovation happening around tooling for Kubernetes. There are now a few CD tool vendors that are looking at solving just this problem, how do you do CD to Kubernetes? The good news is you have a large variety of options to pick from. The bad news is that you need to build a strong opinion on how you want to model your CD workflow and then figure out which is the right tool for you.

[0:10:15.7] JM: You mentioned Kubernetes there. How is Kubernetes affecting the workflows of continuous delivery?

[0:10:23.6] SM: Kubernetes is a deployment platform. It's a target environment. In that sense, it doesn't really change the CD workflow too much. You still need to model all the various aspects of a CD workflow as you did earlier. What changes now is the implementation of some of the stages. Kubernetes introduces a very mature declarative format for deployments and it also changes the packaging format now. The packaging format now is Docker. The output of CD stage is really a Docker image, and a Docker image is propagated through the pipeline. Then you use the Kubernetes semantics for deployments and stuff for that.

You still need to build your Docker images, you still need to test your application, whatever that means. You need to design your test strategy accordingly, and then you, you need to depict all of this on our end-to-end CD workflow. That workflow doesn't change a lot. The implementation details change a bit.

[0:11:24.4] JM: What kinds of implementation details?

[0:11:26.5] SM: Like I said earlier, one of the big changes now is the image or the artifact that you generate at the end of CI is mostly going to be Docker images. You are essentially propagating these images along the pipeline and designing test stages that insensate instances

of services or applications from these Docker images, and then eventually designing deployment books, so whatever that means in your context.

Another interesting development is that a platform like Kubernetes gives you the opportunity for creating on-the-fly environments using namespaces. We've seen our customers do some of that in early pre-production environments where you now no longer need to have long-running environments. You spin up environments on the fly to do ad hoc testing and then clean up the environment at the end of your test cycle.

These are opportunities that come with the Kubernetes environment. Some of the aspects of how you would test your applications change how you would interface with artifact repositories changes, and how you perform deployments and check the health of applications at the end of a deployment changes. Essentially, it starts using more Kubernetes concepts.

[0:12:42.4] JM: If I push a change to my code to my git hosting repository and that code gets built into a Docker container, as it's moving through the different stages, so if it's going from running unit tests, to running integration tests, to sitting and staging for a little bit, to accepting some subset of production traffic, to accepting all of traffic, is the same container making its way through that pipeline and are the pointers, the different environmental pointers changing, or does the container get rebuilt at each of those stages?

[0:13:20.8] SM: That's a very good question. That this is a very important consideration for CD pipelines. So it is pretty essential that the same container is propagated through the pipeline. This is something you should expect out of your CD tooling, is called provenance of artifacts. Once an artifact is generated, that artifact should be propagated across various stages of your CD pipeline. If you test a certain artifact in staging, that is the artifact that should go to production, and likewise if something is running in production, it is important to trace it back when the artifact was built and what changes went into it.

Short answer, it should be the same artifact, or the same Docker image that's deployed to each environment, and it's also important for you to be able to trace back to see what changes went in there.

[0:14:11.2] JM: The large enterprises that are buying into Kubernetes that are in the process of getting on to Kubernetes, it's so interesting to talk to some of these companies, because a lot of them are – their organization is maybe it's a 100,000-person organization and they've got thousands of engineers and different engineering teams and subsets of the engineering teams are in the cloud, some of them are on-premises, and some of them are adopting Kubernetes, some of them are not, some of them have different Kubernetes toolings, some of them have different cloud providers, and some of them are on CD and some of them are not. There's a wide variability.

Over time, different teams merge. The composition of architecture changes. Maybe you have two Kubernetes clusters in disparate areas of a team and they want to merge over time. Can you tell me a little bit more about the experience that a large enterprise is going through today as they are doing all these different things at once? They're moving into Kubernetes, they're moving into the cloud, they're getting onto CD and it's all heterogeneous and maybe there's not a central orchestrator, or maybe there is. I'm sure you've spent some time talking to customers. Get me inside the head of one of these large enterprises.

[0:15:29.9] SM: What we are seeing with larger enterprises is that their move to Kubernetes, or their adoption of containers and container schedulers is usually attached to some modernization activities. This could take the shape of say a new micro-services migration project, or some greenfield development work that's building new stuff.

It's usually in a smaller, more isolated part of the organization. Then once they achieve some success with that, they start rolling it out to the larger organization. That's one model we've seen. We've seen another pattern where organizations start with just a cluster per environment, so you have say a production environment which is starting to run a small subset of services and then the entire org is expected to start moving their services into these environments.

It really depends on like you said, it really depends on the size of the organization, the system architecture, the maturity of the CD process. We see a wide variety in terms of how organizations start to adapt, or adopt this technology. We have seen a lot of cases where there is a small group that's trying something new, and then sets the stage for what the rest of the organization does. They almost lay the path forward.

From a CD perspective, again a lot of these large organizations try to standardize the technology that they use for CD, just so that they can set up templates and workflows for how the entire standardized workforce, or how the entire organization adopts this. Then you do see some variations in tooling based on regional preferences and team preferences and stuff like that.

Usually, when you have these disparate CD tooling, usually what we see is the interface becomes the artifact that you generate. As long as you're generating an artifact with standardized naming conventions and stuff like that and then deposit it into an artifact repository, you then have the final stage which is deployment pipelines that then pick up these artifacts and then do deployments to production environments. There's a wide variety on how large organizations go about adopting this technology.

[SPONSOR MESSAGE]

[0:18:04.2] JM: This episode of Software Engineering Daily is sponsored by Datadog. Datadog integrates seamlessly with more than 200 technologies, including Kubernetes and Docker, so you can monitor your entire container cluster in one place. Datadog's new live container view provides insights into your container's health, resource consumption and deployment in real-time.

Filter to a specific Docker image, or drill down by Kubernetes service to get fine-grained visibility into your container infrastructure. Start monitoring your container workload today with a 14-day free trial and Datadog will send you a free t-shirt. Go to softwareengineeringdaily.com/datadog to try it out. That's softwareengineeringdaily.com/datadog to try it out and get a free t-shirt.

Thank you, Datadog.

[INTERVIEW CONTINUED]

[0:19:01.5] JM: With GoCD, you personally have been working on the Kubernetes integrations. I'm wondering what your experience has been like with that. What are the touch points that you

had to build off of, or what that integration process was like and what your perspective of the current Kubernetes ecosystem is.

[0:19:24.0] SM: We found that getting started with Kubernetes was simple and straightforward. As we started digging deeper and had to get into some of the more advanced aspects of Kubernetes, things started getting a little complex.

Running production workloads on Kubernetes currently starts getting a little complex in a few areas. The few aspects of Kubernetes that we had to dig a little deeper into things around ingress routing, so the out-of-the-box ingress controllers aren't really stable enough for large production workloads. Usually, we see our customers having to supplement the ingress routing with some edge routing and external load balancing.

We had to dig a little deeper into the Kubernetes networking stack. Kubernetes uses overlay networks for its networking and that starts becoming a little complex when you start digging a little deeper. Those are some of the aspects that we had to look into. Persistence was another one. Managing persistence, especially for data stores and Kubernetes is particularly hard. The Kubernetes concepts don't really lend themselves well to running data stores in Kubernetes.

Those are some of the aspects of Kubernetes that once we started operationalizing our product on Kubernetes, we had to dig a little deeper. Getting started is definitely quite simple. From what we've seen is Kubernetes makes it much simpler for developers to package their applications and define deployments declaratively, but it doesn't necessarily make operations of large clusters any less complex.

[0:21:06.0] JM: If you want to deploy a application like GoCD if you're running your own Kubernetes cluster, can you do that with helm, or what's the deployment process for getting tool like GoCD, or GoCD specifically deployed to your Kubernetes cluster?

[0:21:24.0] SM: Yeah, that's another good question. We definitely recommend using helm for deployments. Helm has become the de facto standard for packaging applications for Kubernetes. It gives you a good templating mechanism, it gives you a good way to define all aspects of a deployment. It also has this concept of releases, so you can track all the releases

that have been made on a Kubernetes cluster and roll back to a previous version or not and stuff like that. We are definitely seeing a lot of traction for helm and we definitely recommend that package your application as a hen chart and that's what we've done with GoCD as well.

[0:22:04.1] JM: I want to talk about some of the different continuous delivery tools and get your perspective on how they might contrast, because I think there's probably listeners who are evaluating that maybe they work at one of these large enterprises and they're evaluating what CD tool to go with, or maybe they're in a completely greenfield environment and they're evaluating what CD tool to go with. Let's start with Jenkins. Many people use Jenkins as their CD solution with Kubernetes. What are the pros and cons of a Jenkins-based workflow?

[0:22:36.6] SM: Jenkins again, has a few varieties now. Let's stick to the Vanilla Jenkins. What we've seen is that for a lot of our customers who are already used to Jenkins and are already using it for other types of workloads, they tend to extend Jenkins and use it in the Kubernetes scenario. Jenkins was initially built as a general-purpose automation tool and you need plugins to introduce some of the CD capabilities in Jenkins and this includes pipelines.

Sometimes the experience of using these plugins can be a little challenging, and sometimes the plugins are not very well maintained. In a way, by extending Jenkins, you don't really have to bring in a new CD tool just for Kubernetes and Kubernetes could be a smaller part of your entire infrastructure landscape. That's a positive for sure. Then there are limits to how much you can extend Jenkins and still have a very usable workflow.

[0:23:39.9] JM: There are also cloud providers that have CD tools now. Google has something, AWS has something. Is there an advantage too tightly coupling your CD workflow to the cloud provider you use?

[0:23:54.1] SM: There are advantages and disadvantages. We feel that there are more disadvantages and advantages. Some of the advantages obviously are that you are starting to use native services from the cloud provider, but what we've seen is some of these services are still not very mature. If you look at the AWS cloud bill for example, it still has some ways to go with how it integrates with other AWS services.

The other thing is we feel that CD tooling should be completely independent of your deployment workflow and it should let you orchestrate pretty much any deployment workflow. When you start buying into vendor-specific CD tooling, it's hard to do that. Then the advantages are that you're not maintaining another set of CD tooling, you're just using native services off of a cloud provider.

[0:24:52.0] JM: GoCD is the continuous delivery tool that you work on from ThoughtWorks. What's the backstory for it? How did it get started and how has it evolved over time?

[0:25:03.8] SM: GoCD was initially built by a small team in ThoughtWorks that was starting to build a product. In 2006, ThoughtWorkers Jez Humble, Dan North and Chris Read presented a white paper called The Software Production Line at an AGILE conference. Then following that, Jez Humble and David Farley started writing this book called *Continuous Delivery*. While writing this book, they were conscious of the fact that they were recommending practices in the book that didn't really have any tool support.

Just around that time, ThoughtWorks decided to form a dedicated product division. At that time, it was called ThoughtWorks Studios. Jez Humble started building a CD platform as a part of this organization that later became GoCD. Just an interesting trivia is that the product actually launched before the book was published, so the first use of the term continuous delivery pipeline was in the context of the GoCD product.

[0:26:08.6] JM: I guess, we could talk through some of the technical features of it. If I push a new commit of code to a repository, what happens? What needs to happen in order to get that code tested and step through the CI pipeline?

[0:26:27.4] SM: GoCD does exactly what you tell it to do. GoCD lets you define CD workflows and it gives you constructs to be able to do that, so you can model something called a pipeline and a pipeline could have sequential stages, and then there is a parallel aspect to it. It gives you all the constructs to define a workflow as you need one in your current context. It builds in CD concepts, like artifact propagation and provenance of artifacts and fanning out of pipelines when you need to run something in parallel and then fanning back in and then keeping the context of the artifact that was generated.

What GoCD focus is on is the CD workflow constructs. Then you can plug in pretty much anything into these workflow constructs. If you can script it out, you can run it in the context of a GoCD pipeline.

[0:27:26.7] JM: What's an example of some scripting that I might want to run in a GoCD pipeline?
[0:27:31.6] SM: Very typical scripting involves kicking off testing products for running stuff like automation tests, or doing deployments to various environments, verification stages, stuff like that. Pretty much anything that you can execute in a script, you can plug into a GoCD job.

[0:27:54.1] JM: Dependency management is something that's important in a continuous integration tool. I think because if you're deploying a new piece of code, then it might like other pieces of code might be dependent on it, or you might be dependent on other pieces of code, so you have to maintain a healthy dependency graph as you are pushing out new code in a continuous fashion. Describe how dependency management should ideally work in continuous delivery?

[0:28:23.0] SM: This is another area where different products take different stances and different opinions on how these things should be modeled. With GoCD, we recommend that you model bills for dependencies quite explicitly. What I mean by that is say you have a software component that depends on a lower-level library. In GoCD, you would define an upstream bill pipeline for the lower-level library that could then trigger the build pipeline for any software component that uses it.

There are various patterns in which you can set up these dependent bills, really based on how you want to react to changes in your dependency graph. Now there are some other tools that use the features of build tools like Maven, to discover dependency graphs and react to changes in dependencies. It really depends on how you want to react to changes in this graph.

[0:29:21.9] JM: You have spent a lot of time understanding how Kubernetes works in order to get GoCD working with Kubernetes. I guess, give me a little bit more color on where you think Kubernetes is going and why it's important to organizations.

[0:29:39.2] SM: Kubernetes provides a very mature runtime environment, especially for containerized workloads. Containerization gives you a few benefits. It introduces a new packaging format as we've said earlier. It lets you package applications and its dependencies and then use that in a consistent manner across environments. It also lets you pack your hardware more efficiently.

If you're using, if you're trying to gain benefit from some of these containerization efficiencies, your next step is going to be adopting an orchestrator like Kubernetes, because Kubernetes makes production workloads viable for containers. It plays a pretty crucial role in the modern infrastructure stack from that perspective.

Like we mentioned earlier, there are a few complexities and running workloads on Kubernetes. What we're wondering is if the next step for Kubernetes is that you develop more user-focused platforms on top of Kubernetes and use Kubernetes as a infrastructure platform to do that. Almost like a more higher order PaaS. Definitely, we're seeing Kubernetes play a very important role in the tech strategy for large organizations who are trying to make use of containers in production.

[0:31:02.1] JM: With those higher-level PaaS products, you have OpenShift, you have Rancher, you have Platform9, you have several others, how do you expect they will integrate with the CD products, or will they pull in CD support into the PaaS product?

[0:31:23.1] SM: It's hard to actually predict the future in that way, but what we are seeing is that a lot of these PaaS products are starting to define and build CD capability into the PaaS platform. Again, that starts becoming slightly more platform-specific and slightly more vendor-specific. If you have simple standard workflows, then maybe that model works for you and maybe that's a good fit.

Again, what we're seeing is with Docker and Kubernetes, another by-product is that CD workflows are becoming simpler and more standard. If Docker is a standard unit of packaging and Kubernetes deployment is just a declarative thing that you define close to your application and then you just pass it on to Kubernetes and Kubernetes sucks in images from a registry and

stuff like that, now this process is starting to look pretty standard. You can replicate this across various components in your organization.

The CD workflow itself starts becoming standard and less complex. It should be easy for you to define products and tooling for these standardized workflows in the context of a higher-order PaaS, and that's what we're starting to see.

[0:32:37.6] JM: Do you need to integrate with those higher order PaaSes, or does the Kubernetes support take care of that integration for you?

[0:32:44.9] SM: You have to interface with the high order PaaSes. They usually come with SDKs and APIs. You interface with the PaaS with whatever interface it provides you. With PaaSes, like OpenShift for example, they also provide you CD capably out of the box. Whatever that offering looks like, you would interface with that CD offering to get your changes into OpenShift for example.

[0:33:12.2] JM: In your time working with Kubernetes, was there anything that took you a while to understand? Were there any tricky things? I think there are people that are listening that are just undergoing that process right now of learning Kubernetes.

[0:33:27.1] SM: Right. Like we said, getting started is simple. Getting your initial bills going on Kubernetes and getting simple deployment going as is pretty straightforward. Kubernetes provides a very good object model that's easy to reason about and make sense of. It provides you pretty good APIs to interact with Kubernetes and a really good CLI.

Quick to get started. Once you start running actual production workloads to start planning to move workloads into production, the few areas you should think about are networking. Kubernetes provides a specification called the container native interface that lets you integrate the Kubernetes platform with the underlying network infrastructure. There are a lot of software-defined network providers based on CNI for Kubernetes.

If you've never deployed to Kubernetes before, making sense of these software-defined networking options and how to get development teams to use them effectively can get a little daunting.

Security is a little complex. There are many moving parts in a Kubernetes cluster, and this creates a large attack surface. Getting a robust security configuration is a good understanding of the Kubernetes role-based access control mechanisms. At times, you need to supplement the Kubernetes secret management with something like vault to manage secrets for infrastructure components specially, so that you're not exposing something like your etcd store outside the Kubernetes cluster.
Finally, persistence is hard. Managing any persistence needs some good understanding of this capability with Kubernetes. Like I said earlier, the Kubernetes concepts don't really lend themselves well to running persistence and specially stuff like data stores in Kubernetes. While it can be done, it's much easier to just use hosted services, like RDS for from Amazon if you're running a data store on Amazon, or even just set up clusters and physical hardware outside of Kubernetes specifically for data stores. That's something to watch out for. You don't really need to run everything on Kubernetes. You can run your service stacks on Kubernetes and then your persistence stores outside. Overall, running production workload still involves some complexity and you still need to dig a little deeper into the Kubernetes platform.

[SPONSOR MESSAGE]

[0:36:04.1] JM: We are all looking for a dream job. Thanks to the internet, it's gotten easier to get matched up with an ideal job. Vettery is an online hiring marketplace that connects highly qualified jobseekers with inspiring companies. Once you have been vetted and accepted to Vettery, companies reach out directly to you, because they know you are a high-quality candidate. The Vettery matching algorithm shows off your profile to hiring managers looking for someone with your skills, your experience and your preferences. Because you've been vetted and you're a highly qualified candidate, you should be able to find something that suits your preferences.

To check out Vettery and apply, go to vettery.com/sedaily for more information. Vettery is completely free for job seekers. There's 4,000 growing companies, from startups to large

corporations that have partnered with Vettery and will have a direct connection to access your profile. There are fulltime jobs, contract roles, remote job listings with a variety of technical roles in all industries, and you can sign up on vettery.com/sedaily and get a $500 bonus if you accept a job through Vettery.

Get started on your new career path today. Get connected to a network of 4,000 companies and get vetted and accepted to Vettery by going to vettery.com/sedaily. Thank you to Vettery for being a new sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:37:48.2] JM: A couple things you mentioned there; container networking, persistence and security. I've heard the issues mentioned as well. Also auto-scalability has been that something, but that's a higher level concern. Tell me how those security networking and persistence, how has that affected your work building an application that is integrating closely with Kubernetes?

[0:38:13.9] SM: We had to look at the persistence options provided by the various cloud providers. GoCD supports Kubernetes deployments on GKE and the Amazon Cloud and on [inaudible 0:38:28.4]. Each of these vendors provide different persistence mechanisms for Kubernetes clusters. When you are designing persistence of say, the GoCD database, we've had to take these variations into account.

The Kubernetes role-based access control is pretty robust and pretty granular. You have to design your access control for various services in your application in-line with the role-based access control. Like I said earlier, you might want to supplement what Kubernetes has with an external secret store like vault at times.

From a networking perspective, there are various overlay network providers and you want to choose one that works well for you. We've seen Cool, OSS, Flannel is definitely a leader in this space, so you should really look at getting it in early, testing it and seeing how, especially if you have multiple teams running their workflows on your Kubernetes cluster. It's important to see what their interface, or their interaction with these overlay networks would look like. How does

application isolation look like in conjunction with these overlay networks? Those are some of the things we have to look into.

[0:39:42.1] JM: Okay, this is a term I've heard before, but I have not ever drilled into it. Overlay network, what does that mean?

[0:39:48.9] SM: An overlay network is a software-defined network, which gives you high-level abstraction over the physical network. Kubernetes comes with its own default networking stack called KubeNet. That is pretty limited. For example on say Amazon, on AWS, you can only define 50 IPs on your KubeNet network. That's pretty limited, so you need to extend that capability and you usually do that by using an overlay network, which is really software-defined networking that essentially wraps all of your packets with additional information and defines routing rules and isolation rules on the Kubernetes environment. Essentially, it's a higher level abstraction over the network layer.

[0:40:37.7] JM: Okay. The other thing you mentioned a couple times was in the context of securities, you might want to use vault, or something like vault for secret management. What's the example of when you would need to use another secret management tool to do permissioning, or – there's role-based access control within Kubernetes and I guess that's one form of doing permissioning. I guess, secret management is a little bit different than permissioning, but tell me a little bit more about that security question.

[0:41:08.3] SM: Sure. If you had to expose secrets about your application, or even secrets relating to infrastructure components, you could use the Kubernetes secret management capability, or you could externalize these secrets in a more robust store like vault. Then use vault in conjunction with Kubernetes deployments to make secrets available to applications at runtime. It's just a more robust way of managing these secrets.

[0:41:39.2] JM: Okay. Persistence, first of all, what are you persisting? If you're building a continuous delivery tool on top of Kubernetes and you need to evaluate different persistence products per cloud provider, I guess it's like what, the relational database that describes your pipelines for example?

[0:41:57.3] SM: Yeah, yeah. It's the relational database. It's the definition of pipelines. We have all pipeline definitions stored on local persistence. Some of the considerations for us were, so how do you make a persistent volume available to the GoCD application? The different cloud vendors provide you different assistant volume options. You need to be aware of these and you need to know which ones to pick.

Some of the considerations are do you want some of this configuration to live after a container has been destroyed and recreated, or and some of it just more volatile and can go away? With more persistent information, you need to have persistent volumes that can live, out live, or contain a restart for example. Kubernetes, auto scales, pods in reaction to a pod outages, or so, you should be able to restore your persistent volumes on a Kubernetes cluster. How you do it really differs per cloud provider.

[0:43:05.6] JM: Tell me more about that restore process. I guess, I didn't quite understand what you were saying there. If some failure can occur, I need you do a restore. What are you talking about?

[0:43:15.6] SM: You can have scenarios where you bring down pods, or a pod gets recreated, for example. If you're running say the GoCD server on a Kubernetes pod, if that pod was to be destroyed and recreated, either because the pod was unhealthy, or because you're just bringing the Kubernetes application down and restarting it, the pod has no persistent storage attached to it. Usually, you would attach or put something called a persistent volume, which is external to a pod. The persistent volume options that you have are different per cloud provider essentially. Does that answer your question?

[0:43:57.8] JM: Yeah. Yeah, it does. I think I want to move on to talking a little bit about the future. We've talked about the past and the present of GoCD. I guess, we've talked a little bit about the future in terms of perhaps organizations will move more and more towards PaaS and that will affect the architecture, or at least the delivery mechanism of GoCD, and how people want to consume a continuous delivery tool. What are the other things you think will change around CICD in the future?

[0:44:30.3] SM: We feel that CD has to move in step with how software engineering practices move. If there is a move towards smaller, more distributed services, then CD will have to move in step with that. At the moment, we're seeing a lot of adoption of containers and schedulers and this introduces the – also introduce an opportunity to simplify and standardize the CD process, so hopefully CD tools and CD tooling will also become more standardized. We'll have to wait and watch and see how these modern infrastructure stacks evolve and CD tooling would have to evolve with them, essentially.

[0:45:11.5] JM: What are the next steps for the engineering team specifically at GoCD? What kinds of things are you working on right now?

[0:45:17.9] SM: Our current focus is on making improvements through the product across the board. This includes improving integration with cloud providers and making it easier for our customers to use GoCD elastic agents. Elastic agents let you scale build agents, or build workers on GoCD in reaction to build workload. We're also actively working on improving the developer experience of GoCD.

This could take the form of better SDKs, or a CLI and an improved experience in controlling GoCD pipelines using code. We have existing capability in these areas and we're looking at extending this. Lastly, we're continuing to improve the user experience of GoCD. We talked about developer experience, but we're also looking at the user experience. We want to make it easier for users to understand GoCD concepts. Essentially, we're working on across-the-board improvements of the product currently.

[0:46:18.9] JM: I think the speed of getting your tests run, that can depend on how parallelizable your tests are, right?

[0:46:28.8] SM: This is again a philosophically consideration. What we believe is that CD tooling should let you run any amount of parallelization in your CD workflow, and GoCD already enables you to do that. Now how parallelizable your test suites are really is a function of how they are implemented, and if there are dependencies between tests, or test suites and stuff like that.

Usually we've seen that test suites, especially the more longer-lived ones tend to get into an issue of dependencies between tests. Some tests do data set up, which another test in a suite depends on, so you can't really run them and parallel. Sometimes it's not intentional. It's just a byproduct of just QA teams working on test suites constantly.

The CD tooling really should get out of your way and it should let you run things in as parallel a manner as you want. Then it comes down to the design and the craft of building test suites that are highly parallel.

[0:47:29.1] JM: To close off, I'd like to get some advice for people who are in the process of getting, going with continuous delivery. I think there are a lot of organizations out there who do feel like it's in the early days for them. What advice do you give to people when they're just getting started with continuous delivery? How to get going, how to build momentum and how to change the culture in an organization?

[0:47:55.9] SM: I think organizations should, especially when you're getting started with CD, you should really focus on enabling your teams to move fast and experiment quickly. You need to put in processes and automation that enables teams to deliver value quicker. This really means automation all the way from infrastructure to planning tools and delivery tools. Then beyond that, you also need to think about improving the culture of automation in an organization.

Getting people to think about automation first and a plan for moving away from snowflake infrastructure that you may already have. Adopt more modern technology that lets you move fast and automate more. This is what we're seeing organizations who are ahead in the game already doing. This is what gives them a big differentiation. Essentially, the CD workflow is easy to implement when you have a highly automated process and you have a culture of automation in your organization.

[0:49:00.9] JM: I thought of one other thing I wanted to ask you about. I've done some shows around machine learning deployments. There seems to be something about machine learning models and I guess, data science workflows that require specific, like domain-specific continuous delivery tooling for machine learning models. Have you talked to any customers

about that issue, or have you looked at the issue of machine learning model deployment and is that something that is in the scope for you?

[0:49:35.5] SM: We aren't focused on that specific area at the moment. We have seen one of our customers take GoCD and use it in that manner, and essentially again, the CD tooling really provides you the workflow to do what you need to do in that context. Then as long as the deployment of machine learning models, or training models of whatever you're doing in that context is something that you can script out and include in a larger workflow, there should really be no reason you shouldn't be able to do it with any CD tooling, especially GoCD.

We have seen one of our customers use it in that manner, but we're not looking at building specific extensions to GoCD to supplement the machine learning operational model in anywhere.

[0:50:25.3] JM: Yeah. Man, that space is weird, because I think it's hard test machine learning models. It's also hard to test whether you're talking about the inference process, or the training process. The training process, you want very specific hardware for, so it's like, do you take that hardware and deploy it in a continuous delivery pipeline as well, or maybe you don't do continuous delivery for changes to your model training and you only do continuous delivery for new models and you just validate? Then there's the data set question, like do we need to validate it on some test data set, or do we need to validate it against production data. I don't know, these questions are interesting.

[0:51:12.7] SM: Yeah. As you said, right? It really comes down to what you're validating, or what you're trying to validate in this process. Any changes to models for example, would have to be validated in some form of the other. I'm not sure if training is a part of that validation, or what else you would do to validate a change to a model, or how often these models change. Essentially, if you are running either the training processes on special hardware, or any validation on a set of hardware, what's really important is that you should be able to involve that hardware in your CD workflow.

If you can create, say build workers, or build agents on that hardware and then have a process where the straining process, or the validation process can feedback and provide some feedback

to the CD workflow in terms of success failure, then that's very easy to involve into CD workflow. It really depends on what you're trying to do and what you're trying to validate and what feedback loops you're looking for, because if you training machine learning models that can be quite a time-consuming process. I'm not sure if you want that on your critical path to production, whatever that means in a machine learning context.

[0:52:30.8] JM: Sheroy Marker, it's been really great talking to you. I appreciate you come on the show and talking about Kubernetes and CICD.

[0:52:36.1] SM: Thanks a lot. Thanks for having me.

[END OF INTERVIEW]

[0:52:40.6] JM: GoCD is a continuous delivery tool created by ThoughtWorks. It's open source and free to use and GoCD has all the features you need for continuous delivery. Model your deployment pipelines without installing any plugins. Use the value stream map to visualize your end-to-end workflow. If you use Kubernetes, GoCD is a natural fit to add continuous delivery to your project.

With GoCD running on Kubernetes, you define your build workflow and let GoCD provision and scale your infrastructure on the fly. GoCD agents use Kubernetes to scale as needed. Check out gocd.org/sedaily and learn about how you can get started. GoCD was built with the learnings of the ThoughtWorks engineering team, who have talked about building the product in previous episodes of Software Engineering Daily, and it's great to see the continued progress on GoCD with the new Kubernetes integrations.

You can check it out for yourself at gocd.org/sedaily. Thank you so much to ThoughtWorks for being a long-time sponsor of Software Engineering Daily. We're proud to have ThoughtWorks and GoCD as sponsors of the show.

[END]