EPISODE 650

[INTRODUCTION]

[0:00:00.3] JM: Continuous delivery is a way of releasing software without requiring software engineers to synchronize during a release. Over the last decade, continuous delivery workflows have evolved as the tools have changed. Jenkins was one of the first continuous delivery tools and is still in heavy use today. Netflix's open sourced Spinnaker has also been widely adopted.

As Kubernetes has grown in popularity, some engineers have developed a workflow around Kubernetes, together with git known as GitOps. GitOps treats git as the source of truth for deployments. Under GitOps, when a divergence occurs between your git repositories configuration files and the state of your production infrastructure, your infrastructure should automatically adjust its state to align with the state defined in git.

Alexis Richardson is the CEO of Weaveworks, which is a company that has built tooling around GitOps. Alexis joins the show to describe how GitOps works and explain how it compares to other methods for continuous delivery.

[SPONSOR MESSAGE]

[0:01:18.4] JM: At Software Engineering Daily, we have user data coming in from so many sources; mobile apps, podcast players, our website, and it's all to provide you our listener with the best possible experience. To do that, we need to answer key questions, like what content our listeners enjoy, what causes listeners to log out, or unsubscribe, or to share a podcast episode with their friends if they liked it? To answer these questions, we want to be able to use a variety of analytics tools, such as Mixpanel, Google Analytics and Optimizely.

If you have ever built a software product that has gone for any length of time, eventually you have to start answering questions around analytics and you start to realize there are a lot of analytics tools.

Segment allows us to gather customer data from anywhere and send that data to any analytics tool. It's the ultimate in analytics middleware. Segment is the customer data infrastructure that has saved us from writing duplicate code across all of the different platforms that we want to analyze.

Software Engineering Daily listeners can try Segment free for 90 days by entering SE Daily into the how-did-you-hear- about-us box at sign-up. If you don't have much customer data to analyze, Segment also has a free developer edition. But if you're looking to fully track and utilize all the customer data across your properties to make important customer-first decisions, definitely take advantage of this 90-day free trial exclusively for Software Engineering Daily listeners.

If you're using cloud apps such as MailChimp, Marketo, Intercom, Nexus, Zendesk, you can integrate with all of these different tools and centralize your customer data in one place with Segment. To get that free 90-day trial, sign up for Segment at segment.com and enter SE Daily in the how-did-you-hear-about- us box during signup.

Thanks again to Segment for sponsoring Software Engineering Daily and for producing a product that we needed.

[INTERVIEW]

[0:03:48.1] JM: Alexis Richardson, you are the co-founder and CEO of Weaveworks. Welcome to Software Engineering Daily.

[0:03:52.9] AR: Thank you very much, Jeff. Nice to be here.

[0:03:55.3] JM: We did a show a few years ago and that was in the midst of the container orchestrator wars and Kubernetes was coming to market and becoming increasingly popular. As the world of containers and container orchestrators really started to become more popular, when did people start to have patterns around continuous delivery and deployments around containers? Was there some centralization in those early days?

[0:04:25.3] AR: I mean, I've been involved with containers since 2014 through Weaveworks, and indirectly while I was at Pivotal we were doing obviously work on Cloud Foundry and through that used Alexey containers, obviously Docker came along around 2014 or 2013. It looked like a really impressive technology.

It took me a while to understand how that joined up with the worlds of CI/CD, continuous delivery. Well do which I'm by no means a personal expert and have been learning about a lot in the last few years. It's been really fascinating. I think that we could see a lot of continuous delivery being advertised with containers and I discovered that was principally because for many early uses of Docker, writing a system that generated a container and deployed it was considered to be a really good way of using them, and a lot of people even built their own CI systems that ran inside containers.

There was an early affinity in early use case for Docker was CI. We saw some companies come to market like Shippable based on that and circle CI I was an early container-centric system and we have container ship and code fresh. Clearly, containers have revived continuous integration and delivery.

I also have noticed that if you talk to customers, pretty much everybody who's trying to use containers gets a lot more value out of them if they do CI/CD, or at least through CI. The idea that you should have frequent automated correct deployments is one which goes hand-in-hand with the ability to generate code to deploy in the form of artifacts that can be run. Of course, CI/CD, CI and containers really, the first set of things that you encounter as you start your journey of containerization.

Also people who've been doing CI before with these sites that have Java and .net quickly adopt containers, because they can just extend the world of CI. That's on the one hand, but I think the game-changing thing in additional to immutable infrastructure is orchestration. The idea that you can have a tool which uses convergence through an orchestrator to control exactly what's running inside the system previously done in an ad hoc way, but now done systematically through tools like Kubernetes, of course Docker Swarm, ECS, mesosphere, marathon, I mean let's not forget all these technologies; Diego inside Cloud Foundry as well.

The concept of orchestration is one which is based on managing to a plan, trying to synchronize between the intended state and the observed state. In this case, what the orchestrator can see. That's something which I think is a mechanism for updating changes into your system and gives you a new way to deal with deployment. I think that's what I'm seeing in terms of CI and CD and orchestration and containers today.

[0:07:17.4] JM: When did convergence start to happen around the ideas that became GitOps?

[0:07:25.2] AR: GitOps is essentially the idea that you should manage a system by comparing its desired state has expressed through a declarative tool, like a set of declarative config files in the case of Kubernetes, may be other things for other systems. Compare the desired state with the observed state, and the observed state is what you can observe about the actual system.

In fact, there are four different things that are true at any given time; the desired state, the real state, the observed state and what's in the head of the operator. We can never know the actual state. What we do is we observe the state through observability tools, such as monitoring and alerts. Of course, that gives us a way building up a picture of what might be true about our actual system.

We have the observed state, our set of beliefs and we have the desired state, which is the source of truth written down in our repose, usually git, but other things too can be involved. Then when we see a deviation between those things, that's when we know there is a reason to check what's wrong and converge the desired and observe state. For example, somebody can make a pull request and that gets merged and that means that desired state gets updated and then that means that somebody needs to do a deployment, because the observable state says, “Hey, you haven't deployed this thing yet. The desired state has been updated, but the cluster hasn't, so let's go and do that.”

Alternatively, you can have system drift where the system drifts away from its desired state and the observers pick that up and go, “Hey, wait a minute. We seem to be different from what we intended to be.” You can imagine as time goes by, as the things that you can describe become a richer set of things like, alert me if this threshold is passed more than three times in five minutes on this metric that could be a rule. That policy statement could be something that you observe

for and compare, because it's something in your desired state that is desirable about the system. You can trigger a diff alert when the real system that you're observing drifts away from that observable rule.
It's a very simple concept. Then you use convergence to get away from the diverge state back on to a world where what you can observe and what you believe needs to be true, need to be the same. This is a very simple iteration of DevOps infrastructure as code, orchestration and observability coming together; those four things. DevOps infrastructure is declarative infrastructure as code. Observability and orchestration are the ingredients that you need to do GitOps.

Quite honestly, for us, calling it GitOps is just a way of reducing all of those complicated jumbly words to something that even my mom can understand. Is simply saying, "Hey, operations, git, let's drive to convergence." When I go home for family Christmas, I can be challenged about this by the family and they can understand what's going on. I'm sorry, I didn't mean to be sexist about my mom. This is for everybody. This is non-trivial stuff and I have found talking to many customers and it uses, "Hey, you're putting into a simpler more modern language something that we already have been trying to do and have been trying to get into the right words. It makes it much easier now for us to proceed confidently, because we feel like we can understand what we're trying to do."

[0:10:37.9] JM: Now we've explored a couple different perspectives on "continuous delivery." The the earlier phases that you outlined when you had the Jenkins and the code ship and the shippable companies that came out to help with continuous delivery, and then Kubernetes came out and changed the set of tools that we had available to us for continuous delivery. Your perspective is that GitOps is in some sense, the next iteration, or a new way of doing continuous delivery that when you think about how to do continuous delivery in the context of Kubernetes, GitOps is an alternative, perhaps a superior way of doing continuous delivery. Can you explain some reasons why it is a desirable way of doing continuous delivery over the previous models of the pipeline? I think, people think of more of it like this pipeline, your code is progressing through this pipeline from dev to test to staging to production.

[0:11:44.3] AR: Yeah. I mean, again this is where I need to take off my hat and hold it very close much authority to my chest and be very, very clear that everything that we're talking about

to a great extent, builds on, iterates from and extends and enhances core ideas that have been around since 1993 when Mark Burgess talked about CFEngine, were popularized by people at ThoughtWorks in the early 2000s, written up by people Jez Humble and Dave Farley with the continuous delivery book in 2011.
The ideas are basically there, but I think that there are some things that we are finding that we do that are worth highlighting. For example, we talk about CI Ops to express the idea that driving operations by using the CI is the source of truth, instead of the declarative description of the system, is something that can lead to anti-patterns. We see people who for example, try to push a group of changes into a running cluster directly from CI. Then if some of those do not succeed correctly, then it's very difficult to get the system back into a correct state and you have to start again.

Also, sometimes they could be quite slow. Whereas, with the GitOps approach, we're also recommending the cluster orchestrator is what you should use to do the convergence inside the cluster. You should not use a build orchestrator such as Jenkins. Bless Jenkins, it's a wonderful tool for doing build tests and all kinds of other things that are CI-ish. It's not the right tool for driving conversions in a running cluster, because if it doesn't work correctly, you need to – doing a group of operations, idempotently the build orchestration tool often fails it, also creates questions about your security, you need to have apparatus to make sure that it's safe for Jenkins to access your cluster, which is possible, but can lead to fragile architectures, which once broken, are a paradise for security people to hack away at.

We talk about something called the immutability firewall; this idea that all the CI systems should do, the build orchestrator is all-powerful in build. It should be allowed to update images, labels, tags, config files and all kinds of settings in the repos. What it's doing is it's doing a destructive update on a set of immutable artifacts in the said repos. It's not allowed to touch the cluster, the running clusters directly.

The running clusters are instead orchestration by runtime orchestrators. Kubernetes is the prominent example of that. They inspect the descriptions in the repos and the runnable artifacts, which are immutable artifacts, and then they update themselves using tools like Weave Flux, which we made, which is a key GitOps tool. GitOp flux runs just as an operator inside Kubernetes. Well, it's actually doing is it's saying, you don't need a separate CD tool, you should let Kubernetes do your CD. It does that by running a small GitOps operator, which notifies Kubernetes when the desired state in the repose differs from the observed state in Kubernetes, and indeed, you can have this for other parts of the spec too.

Then it says, “Oh, if you go now. Now do your update.” What that happens is the running cluster will get the changes that it needs, will pull them into itself from the repos. That means that it can then update itself using the changes, but it has done so the whole workflow has occurred, without the build side and the run side talking directly to each other. Instead, they share a common bulletin board of repos, in which values can be updated, but they don't talk directly, so it's like a workflow way of thinking about updates, but there's no concept of release automation driven by the build system. Instead, the runtime orchestrator manages, releases and deployments and can also do things like progressive deployments and Canaries and blue greens and traffic management and all these other cool things that we would all be doing as standard in a few years’ time.

Do you see what I'm saying? The immutability firewall is a critical concept to understand. CI Ops is when operations are run by build, and GitOps is when the cluster drives its own operations based on noticing differences between the observed and observed – the observed and desired state.

[SPONSOR MESSAGE]

[0:16:15.9] JM: Today's episode of Software Engineering Daily is sponsored by Datadog; a monitoring platform for cloud scale infrastructure and applications. Datadog provides dashboarding, alerting, application performance monitoring and log management in one tightly integrated platform, so that you can get end-to-end visibility quickly.

It integrates seamlessly with AWS, so you can start monitoring EC2, RDS, ECS and all your other AWS services in minutes. Visualize key metrics, set alerts to identify anomalies and collaborate with your team to troubleshoot and fix issues fast. Try it yourself by starting a free 14-day trial today.

Listeners of this podcast will also receive a free Datadog t-shirt. Go to softwareengineeringdaily.com/datadog to get that fuzzy comfortable t-shirt. That's softwareengineeringdaily.com/datadog.

[INTERVIEW CONTINUED]

[0:17:22.6] JM: Let me ask a clarifying question. In a continuous delivery model where I'm using something like Jenkins, or some other continuous delivery tool, the way that I have used it before, or I used the last time I was doing continuous delivery was when I was at Amazon, so I was using an internal continuous delivery tool. I think, it might have been some fork of some other open-source project or something, but the way that my interaction went was I push to git and then there's some listener on the git repository that finds out, "Oh, there was a change to the repo," and so it spins up infrastructure and then it deploys that infrastructure and puts it in some phase of the pipeline. Maybe it's put in a testing environment to run some unit tests, and then if those unit tests pass then it gets automatically spun up in a staging environment, then I can do some manual tests in the staging environment, and then I can do a manual promotion to the production environment if I want to.

In contrast, if I understand you correctly, the GitOps model is you push to git and it's somewhat similar, but instead of having an external tool that is having to do the spinning up of the infrastructure, Kubernetes itself is aware of the relationship between itself and the git repository, and it can just make the changes, the declarative change to itself. Right.

[0:18:51.6] AR: Yes, correct. That means it can basically run as a transactional process. What I mean by that is that it can continue to iterate on attempts to update the system, 'til we have a correct state, usually by just having idempotent updates, but sometimes it just continues pressing on until it says, "Aha, now the observed state and desired state are the same, I can stop." This all happens without us needing to understand anything about the actual state in the cluster. All we do is let the cluster carry on until it's observably correct. That's a different thing.

If you draw it all on a whiteboard, it looks like a big workflow from repo, to bill, to test, to smoke test, to staging, all of these things. What's happening often is you're just changing the direction of the arrows, because you're saying, "Hey, Kubernetes. Hey, dev cluster, your image is ready for testing. Go get it from the imagery post.” It's not just git. You've got a set of repos. So long as they contain immutable artifacts, that's the main thing. Go get these things from the repo and it says, “Ah, yes. I’m ready to do testing. Now off I go.” It goes gets the images and it can run the tests. Do you see what I'm saying?

[0:19:55.5] JM: I think so. Help me understand why it's superior to the model, or how it differs from the model of pushing to git and having a deployment tool spin up the infrastructure.
[0:20:12.1] AR: For a start, the deployment of the spinning up the infrastructure is actually something you generally don't want to do, because it takes time. More often than not, if you're doing a 1,000 deployments a day on a 100 clusters, you don't want to be spinning up a 100,000 clusters to do that. What you'd like to do is ideally be making changes to an existing environment. Actually, the main change is usually going to be updating a dev cluster or a prod cluster, or some other cluster with the necessary pieces to run the tests, or do a canary in production, or do a production rollout, or some other stage rollout. In that sense, I'd say that your question while a good one is focusing on slightly the – not the biggest lump in the carpet, if you don't mind the expression.

The other key differences are when you let Kubernetes update itself, you're not asking another process, which is all-powerful like Jenkins build orchestration is a very comprehensive tool, to be given secure access to the cluster. You can see how that might be important if you had fears that whoever controlled Jenkins could control your production systems.

If you talk to people in the CI world, they will confirm that yes, this can be a concern with large infrastructures, or in indeed in highly compliant environments. We don't really want dev and prod to, sorry, the build and production runtimes to be too close together, to be able to talk to each other. It helps that we can have security in production that we can be confident in, and Kubernetes has a strong security model by running the tools to do update Kubernetes inside the cluster. You're essentially saying, “Kubernetes, you're responsible for the security of things that you update to yourself.”

Of course, this still leaves open is who gets to update the repos? If you have no security on your repositories and if you have automatic updates from your repositories to your production cluster, then of course, if somebody starts to make changes to your config repos, or your image repos, they may end up in production. I'm not pretending that security goes away as an issue, quite the opposite, but it draws the line in a slightly different place, which is generally easier to manage.

The other thing is that when you talk to people and you say, this is a way of knowing that you can do CI/CD in a joined-up way. Without ever letting the build orchestration tool have direct access to the running clusters, they generally go, “Aha. Yes, I'd really like that.” I think that that's a subtle shift away from the old way, but it's an important one that you let Kubernetes update itself.

[0:22:49.9] JM: I see. Okay, so I think I understand it now. Let's go through what happens when I make a git push. When I have a change to my application, maybe it's a photo-sharing app and I make some change to the backend; I create a new API for example, for liking a photo or something. Something that I want in my back-end and I want it to proceed through the dev test staging infrastructure, “pipeline,” whatever you want to call the series of environments that I want my code to proceed through, in the GitOps world, what's the series of events that happens? After I push my code change, what's going on the Kubernetes side? There's these tools like Kubediff, or Ansible diff, or Tera diff, or Weave Flux, which are figuring out what has changed in the git repository. Maybe you can just walk me through the GitOps a series of events that occurs.

[0:23:52.1] AR: Well first of all, I would say that whatever I described on this podcast, people should go to our website and read about it in the blogs, and also review our GitOps how-to’s and need-to-know guides, just because some of this stuff you need to read, you need to think about it a bit and look at different diagrams. Obviously, if you don't see the diagram, but you need that brings it all into focus, you should also e-mail us and say, “I was reading this blog and I did this bit. This piece did not make sense to me. Can you explain what you meant by this, or that, or can you give me a better diagram than this one?” We’ll definitely do that, because we want our diagrams and our stories to make perfect sense to people.

To describe it verbally, if I push a change into code, but this is what we have – this is what we do; we run a fairly large-scale SaaS on top of Kubernetes ourselves. It's called Weave Cloud; it gives you monitoring and deployment and other tools in it. It's dog food, because it's running on Kubernetes. It's a multi-zone, multi-stage SaaS on Amazon.

The normal operational practice for the team when making changes is to merge a pull request. That is used both for dev and test and for a production. In a classic production rollout, we'd basically say, “I'm going to push this towards the production cluster,” which means certain rules about how things are labeled and tagged in our git repos and the config files. Then the system that was responsible for updating Kubernetes will be watching the repos where those changes might be taking place, and that system is called reflux.

It all it is it's a, you can think about it as an operator that runs inside the Kubernetes clusters, scales up and down with Kubernetes, shares Kubernetes security model, inherits management for Kubernetes. Basically Kubernetes manages it for you, that's why it's following the operator pattern. It's just a little process per cluster that's saying, I know which repos I need to be watching and under which permissions and policies I should be updating myself.

It watches the repos, it watches the image repos, it watches the config repos and looks for changes, in places where it's been told to. Then when it sees a change like, “Hey, this image needs to be – not this number image, but this number plus 1, or here’s an image key that's changed from A to B, it's usually some hash, so more complicated than A and B of course,” but this is the image you should be running. It will say, “Ah, I should take this image and deploy it.” Then it will make sure that Kubernetes knows that that image needs to be running instead of the other one. Hey, look at these config files.

Kubernetes will do that and it will watch to see if it's correct. When it's correct, it will then write back saying, “Sorry, that's not quite true.” After Kubernetes has acknowledged that it should be updating, then it will tell flux and flux will write back into the repo, “I have notified the cluster that I'm up, that it needs to update and it just begun the update.” There will be a commit back into the shared repositories saying, this is now happening.

Something that we would like to do, but we don't currently do is have an automatic way of validating that the change once complete, has completed correctly. Because for example, you can have things like containers crash loop and things like that. You’d like to know that isn't happening and you'd like to know when the cluster is in the correct state, so you can say, “Hey, job done. You can go on to the next thing.”

We do have a partial mechanism for that, which is Kubediff, which is a way of alerting us if the desired and observable states are different by in certain areas. You can use that as a substitute for this notification. You can also use a couple of the Kubernetes commands to verify that things are in the correct state. That's the basic workflow. Does that make sense?

[0:27:32.0] JM: It does. What does this make easier in terms of deployments, and what frictions does it eliminate?

[0:27:39.4] AR: What happens in practice and this is where I'm going to point to. We have a customer called Cordova that has very kindly agreed to let their experience be a case study for all of this, so I talked about them quite a bit. They’re a San Francisco-based VC back startup with – they've now I think got four different IT teams, tech teams doing microservices. They were using GKE and Jenkins when they started talking to us, and they used Weave Cloud to interpose flux GitOps processes between Jenkins and GKE to work, to move from a CI-driven deployment system, to a Kubernetes-driven deployment system based on moving the orchestration of the deployment and the releases into Kubernetes using flux and Weave Cloud.

This is described in a blog post called High-Velocity CI/CD for Kubernetes that you can find on our site. Also, I've talked about it quite a bit in couple of online presentations that you can find on SlideShare, like the Continuous Life Cycle London 2018 presentation. Go look at those.

What they found was that by using this way of working, they were able to – previously, they had a CI-driven update system which would sometimes break and take about 20 or 30 minutes to complete, even on GKE. When they did changes, they would stop working and wait together as a team for the change to complete. Deployment would be a company-wide event, and it would happen once or twice a week.

They found by using our approach, which our approach didn't change how they use CI, it didn't change how these Kubernetes. What changed was they introduced this element to managed controlled automation, semi-automated updates in between those two things, so that flux that responsibility for doing the updates, instead of Jenkins which continue to be the same for build and test.

Through that, they found that they could reduce the time of a deployment down from minutes to seconds. They could get quick feedback on how deployments were working using some of the other tooling we provided for observing the system and the diffs and so on. They could – things were not correct. They could always roll back to a previous point in time. They could roll forward again after that and they found that this meant that they stopped worrying and thinking about deployment anymore.

This is important. Really important, because the team was full of mostly machine learning people. Their business is very interesting. They have machine learning for helping sites that need marketing that's localized in different languages and idioms; these machine learning to support analysis and optimization of that language. Therefore, they want to spend their time writing machine learning, all the things that you do; test suites, training sets, looking at the actual customer data and building the things that make it useful to the end-customer, to their end-customer.

They didn't want to spend time becoming infrastructure experts, Jenkins experts, Kubernetes experts and CI/CD experts and CI Ops experts. By using our tool, they stopped working on that stuff and they spent much more time and effort just doing business logic and machine learning. They said one of the main benefits for us is we just stopped worrying about things that didn't matter anymore. They just worked.

Then they started also another big change happened, which is they started to do deployments much more frequently. They started spinning up clusters much more frequently and deploying to production much more frequently. That went hand in hand with making many, many smaller changes much more quickly.

Deployments stop being a company-wide effort and started being just part of every developers day-to-day activities. Wow, so now they've got four different teams; each different team is completely autonomous and the each individual developer can do a deployment within that team and they can roll it back or forward as they like, and they're constantly changing things all day long, and they're doing 30 to 50 deployments a day per team. A peak, this means that they can experiment and they start moving from continuous integration, to continuous deployment, to continuous experimentation.

The idea that they – the whole system is one which they can use to try out many different configurations of more with anything in the technology for customers is what where they're at now. Through that, they found that they started to see real productivity gains that they could measure, and this data is on our website. The time to fix bugs and the time to add features for customers and do issue requests all improved by a factor of around two, in some cases closer to three. Their uptime improved and that all went hand in glove with just a more productive system, the overall productivity of the team went up and up and up.

If you imagine that being applied in a team of 400 developers on 50 or 80 microservices, you can imagine that's going to be a very big monetary impact. The other thing that happened, which is just not something that we expected, it's because all the build activity and the coding activity and the deployment activity was being recorded in the repos, because the Weave Cloud tool and flux operator were actually writing things back in to git.

You actually had a complete audit trail of who made change to the system at any time, including metadata that was sometimes add to the commits, like why they did things. When it came time for the talks to compliance people to come round and have a look at what they were doing, the auditors were delighted to see everything was recorded in git, and they like, “That's great. You're done.” No need to go and spend a ton of money on an expensive configuration management database and [inaudible 0:33:20.7] consultants.

I think those are just astounding benefits; velocity, correctness, confidence, experimentation, stability, security, compliance and all that – just think about all of that just by adopting this approach.

[SPONSOR MESSAGE]

[0:33:43.2] JM: We know that effective learning is active learning. When you master concepts by solving fun and challenging problems yourself, that's active learning. With brilliant.org, you can actively solve problems and have an interactive experience that can enhance your computer science expertise. Brilliant offers foundational courses in algorithms and artificial neural networks, where you can get up to speed on AI. For people with some background in algorithms, there are advanced courses on machine learning and computer memory. Coming soon is an advanced algorithms course, which will help you prepare for technical interviews.

Go to brilliant.org/sedaily to create an account for free. You need a toolkit and a framework to tackle new problems. Brilliant.org has a well-designed, colorful user interface and each lesson is packed with guided problems, with explanations and interactive quizzes.

To support Software Engineering Daily and learn more about Brilliant, go to brilliant.org/sedaily and sign up for free. Also the first 200 people that go to brilliant.org/sedaily will get 20% off the annual premium subscription. If you're like me and you're always looking for new learning materials, especially for complicated subjects like machine learning, then you might enjoy checking out brilliant.org/sedaily.

Thank you to Brilliant for being a new sponsor.

[INTERVIEW CONTINUED]

[0:35:22.6] JM: Yeah, well you get version control by default on all of this stuff that you're putting, that controls your infrastructure and having a versioned history and/or basically treating the version history as the source of truth. I mean, that's just going to have a lot of advantages of like what you mentioned with the compliance related stuff. You could basically have time-stamped – a time-stamped history of when different things were in production win, so it gives you a great audit trail.

[0:35:58.5] AR: Totally.

[0:35:59.3] JM: What's the process of onboarding with GitOps, if somebody is on a different deployment system and they want to migrate to GitOps, they want to try out this model of doing things, what's the onboarding procedure?

[0:36:13.5] AR: The easiest way to try things out is to be using an existing Kubernetes cluster and then go to Weave Cloud and sign up and use a free trial and follow the instructions to get the deployment piece, because Weave Cloud also gives you all the monitoring and observability that you need in order to manage these deployment and do monitored deployments better. Just to begin with, actually you just want to be trying out some basic automated and manual flows around GitOps. If you follow the instructions, you can get GitOps setup pretty easily of your cluster, and there's a happy, friendly, healthy Slack channel where you can go and get help at Weave. There's a general channel that also especially is Flux channels if people want to do hacking on the open source and stuff.

You can also, if you want to go down the open source route, you can set up Weave Flux yourself; the instructions to that are on Github. By doing that, you probably don't get a nice GUI, an audit trail won't be quite so easy to understand and many other things. I would recommend honestly to try Weave Cloud, but go for it whatever you like, or whatever you want to do.

There's also other tools that are in the Github space. There's a tool called Gitkube from a company called Hasura, which runs a build system, a custom build system inside Kubernetes to do and git like update. I use git push for updating dev clusters that's quite neat. Although, not everybody wants to see as a new build system. There's Jenkins X, which takes, is a new codebase written in Go by some people at the CloudBees team that's optimized for doing all kinds of neat developer and GitOps workflows around Jenkins, and I think for many people that might be a good thing to try out.

I saw a few days ago Kenzan and CTO talking about Spinnaker as a good ops tool recommended practices there. Spinnaker and Weave Flux are somewhat similar. One of the key differences is that Weave Flux his Kubernetes native, whereas the Spinnaker mapping onto Kubernetes is a bit of a kludge, but it is still a tool you can use.

[0:38:13.3] JM: Is that because Spinnaker is JVM-based?

[0:38:16.7] AR: Well, Spinnaker is written for a world of virtual machines, and its fundamental unit of deployment is a load balanced group of VMs on Amazon, in a security group, which is a very good thing if that's what you're deploying to. It's slightly different from what Kubernetes is doing, so actually mapping the Spinnaker objects confirms into Kubernetes is just a bit of impedance mismatch there.

Yeah, it's not because it’s – Yeah. Spinnaker is largely written in Java, so it's as more apparatus to install and deploy and manage yourself, whereas with flux, it's just a tiny little thing that Kubernetes runs for you. Spinnaker was written before Kubernetes. You can also run Spinnaker inside Kubernetes, but that's like running a rhino on top of an elephant.

[0:38:59.6] JM: I'm very curious about the company building process. There's a lot of listeners that are looking for ideas of companies to build, and I think a lot of people are looking in the Kubernetes ecosystem. I've been to a couple Kube Cons at this point and I find this area of business to be so fascinating, because you've got this open source project that has tremendous momentum, you've got cloud providers that are all – they've all converged on this project and they're providing their support to it, and there's clearly a lot of opportunity.

What has been your experience over the last four years of building a company in the cloud native ecosystem? Specifically, what have been the difficulties and what does it make you realize about the opportunities in building cloud infrastructure businesses?
[0:39:49.2] AR: Wow, that's a real far side chat question.

[0:39:52.2] JM: That's right.

[0:39:53.9] AR: I don't know. I tend to think of a technology startup as being a bit like a boat and the market is being a bit like an ocean, and the customers is being harbors and ports in the storm and you can imagine that there is such a thing as stormy weather. It's all about tacking towards where the winds are strongest and where you can meet the biggest customers and the most customers at the right time.

When they're ready to talk, I mean, I would say that one of the things that's most fascinating about the evolution of cloud native is a combination of very rapid change with a very large number of people who want to engage at different times. When we started out, the problems were how do I network two containers? Docker was the king of the market. Now, it's much more multipolar, which I think is healthy. Docker is doing well, but so are a bunch of other people too.

Customers have much more pragmatic and obvious problems around how do I really use this? How do I’d use things in production? How do I [inaudible 0:40:52.1] things at scale? You've got to be patient and be ready for those customers to come when the time is right. We are seeing people now come with substantial problems to solve and budgets to solve them, with that a year ago were sitting on their hands waiting for things to happen. You can imagine how big a difference that makes.

Yeah, I think also you've got to make sure that you can last the journey. The market seemed exciting three, or four years ago, but was very, very early, and I think some people were not ready for a journey of four years just to get to this point. I think we've got a good number of years left in this market.

A lot of the early hype around containers made it sound like this is cloud 2.0, or web 3.0 if you prefer. The cloud's market grew very fast. I mean, one of the things that's truly remarkable about Amazon is they instantly created an ecosystem for companies, like New Relic and Twilio and others to build real businesses just on top of Amazon. I mean, that's really testimony to the incredible power of that market; Heroku as well.
Whereas in containers, it's just not a clean enough, simple enough change that a market has been created, and then on top of that, everyone builds their new castle and makes money from that. That's why we're all still helping enterprises to do the spade work, to chop the wood and carry the water in the Kubernetes team's parlance, just to get basic stuff done. I'd say that it's been a journey requiring patience, strong waterproof clothing and a lot of food.

[0:42:34.7] JM: Is that to say there is – there's some, I don't want to say hand-holding, but you have to – if you're building a product in this space, maybe you have to be willing to do something that looks a little bit like consulting to help people onboard to the product and eventually, maybe someday a lot of these products will look like the one-click spin up your infrastructure magically, like on AWS, like what AWS has, that simplicity. Today, so just a little bit more working directly with the customers?

[0:43:10.6] AR: What can I say? Yes, is the answer to your question. I think that consulting is very important. I prefer to use the word helping customers for things they need to do. The number of people who are okay, I've got this thing that I downloaded that helps me install Kubernetes, but I want to know if I can run it on VMware, or something else is quite high still. There's people who are using GKE and EKS now and AKS happily, but have questions about load balancers and backups and training and applications and what the hell was this CI/CD thing anyway? Blah, blah, blah.

You can imagine that just giving them a product and saying, “Just use this,” is going to frustrate them. You know what I mean? It’s like, “Did you just tell me to F off?” Whereas, if you say, “I know, I understand where you're going, but let's have a longer conversation about how are you going to get there?” Then, they very, very much appreciate that. I hope they do anyway. It's very important to us that they do. There is a bit of consulting.

As a company, we try to keep – we are a product company, so it's important to us that our revenues are recurring subscriptions, and we think very, very carefully about how to give customers value primarily around that. That is not to say that you can do it with excluding consulting, because then, you shoot itself in the foot, and then your customer is disappointed. Yes, it's a mix.

[0:44:34.2] JM: Very interesting. Okay, well to wrap-up, what's in the near future for Weaveworks? What are you focused on building right now?

[0:44:41.0] AR: We are very focused on building out more features that help customers to do GitOps well. Really understanding okay, this GitOps, it's an idea, it's a set of best practices and patterns, which iterate from infrastructure – declarative infrastructure as code, immutable infrastructure, orchestration and DevOps. How do those come together and observability, right? Those things all come together; observability, orchestration, DevOps, declarative infrastructure as code and immutable infrastructure. I've got to keep saying this, otherwise I’ll forget.

That's all complicated. How we make it simple? Which pieces of it are actually useful to customers? Which pieces do people find easy and obvious, and which pieces to people find hard? How do we focus in on the things that are most important? We're seeing a lot of stuff around staging, progressive deployment, smoke tests, introducing observability and measurement and metrics and monitoring into the deployment processes much more intimately. That's very important to customers.

We're also seeing a lot of stuff around not just applying this at the level of applications and services, but applying this within the platform itself. A lot of customers are saying, "Give me an operating model. How do I operate apps and services on and around Kubernetes? Help me do that." We're trying to turn that into features, roadmap, exciting things for customers to make them delighted. That's really what's going on now.

We're also doing a lot of work with our main partners; Amazon, Google, Pivotal especially. Very excited to do things with them and how they go to use Kubernetes both in the cloud and on-premise. There's also all kinds of cool stuff landing in terms of stuff you can run on top of Kubernetes. Kube flow for machine learning, Istio, service mesh, serverless service mesh, serverless mesh. You can imagine all kinds of crazy combinations. While we don't necessarily support it all ourselves, it's one of the things that drives customers to adopt, so we like to stay close to that and help enterprises and small companies be successful with it. That's a stuff that you'll see us tweeting about and so on.

[0:46:51.6] JM: Very cool. Well Alexis, thank you for coming on the show and describing GitOps, and also giving some reflections on the business building process. I really appreciate it.
[0:47:00.2] AR: Yup. My pleasure.

[END OF INTERVIEW]

[0:47:04.5] 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]