EPISODE 371

[INTRODUCTION]

[0:00:00.8] JM: Software deployment evolves overtime. In the 90s a deployment might have meant issuing a new edition of your software via CD-ROM. Today a deployment is often a multistage process. A new software build will undergo automated unit tests and integration tests before being deployed to users. The deployment might only go out to a small percentage of total users initially with that percentage going up as the deployment process proves not to have bugs.

Avi Cavale is the CEO of Shippable, a platform for DevOps. In this episode we discussed deployments in the context of containers including a discussion of what has become easier; micro services feature flagging and continuous delivery. He also discussed the experience of building Shippable, which is a very interesting SAS business for developers and development teams.

Software Engineering Daily is looking for sponsors for Q3. If your company has a product or service or if you're hiring, Software Engineering Daily reaches 23,000 developers listening daily. Send me an email, jeff@softwareengineeringdaily.com if you have an interesting product or service that you'd like to get in the ears of developers. I hope you like this episode.

[SPONSOR MESSAGE]

[0:01:31.9] JM: Hosting this podcast is my full-time job, but I love to build software. I'm constantly writing down ideas for products; the user experience designs, the software architecture, and even the pricing models. Of course, someone needs to write the actual code for these products that I think about. For building and scaling my software products I use Toptal.

Toptal is the best place to find reasonably priced, extremely talented software engineers to build your projects from scratch or to skill your workforce. Get started today and receive a free pair of Apple AirPods after your first 20 hours of work by signing up at toptal.com/sedaily. There is none of the frustration of interviewing freelancers who aren't suitable for the job. 90% of Toptal clients hire the first expert that they're introduced to. The average time to getting matched with the top developer is less than 24 hours, so you can get your project started quickly. Go to toptal.com/sedaily to start working with an engineer who can build your project or who can add to the workforce of the company that you work at.

I've used Toptal to build three separate engineering projects and I genuinely love the product. Also, as a special offer to Software Engineering Daily listeners, Toptal will be sending out a pair of Apple AirPods to everyone who signs up through our link and engages in a minimum of 20 work hours. To check it out and start putting your ideas into action, go to toptal.com/sedaily.

If you're an engineer looking for freelance work, I also recommend Toptal. All of the developers I've worked with have been very happy with the platform. Thanks to Toptal for being a sponsor of Software Engineering Daily.

[INTERVIEW]

[0:03:34.9] JM: Avi Cavale is the CEO of Shippable. Avi, welcome to Software Engineering Daily.

[0:03:40.0] AC: Thank you very much for having me.

[0:03:42.0] JM: Today we're going to talk about modern software deployment and what that means. Engineers have been deploying software for a longtime. How do you define the term deployment?

[0:03:56.7] AC: The way I think of it is it doesn't matter who your end customer is. If you can't get your end customer to interact with whatever you're doing, then it's on a deployment. For me, deployment is all about whatever you built eventually has to make it into the hands of your end customer, whatever that medium of delivery is. That's what I call as deployment.

[0:04:20.3] JM: What is a modern deployment look like?

[0:04:24.6] AC: That’s a very complicated question. The way I think of it is today there is a lot of latency between when software gets completed to the point of time, where your customers are interacting with it. Modern software deployment or delivery is all about how you reduce that lead time, and the closer you are to completion with your users can actually start interacting with it, the closer you re trying to get to the utopia of what modern software deployment is all about.

[0:04:58.0] JM: What are the problems that can occur during a deployment?

[0:05:02.2] AC: I think there are many, and some of it is dependent on how your organizational structure is in terms of how your teams are organized in order to build software. Some of it is also the architecture and some of it is also could be what type of packaging you're using in order to get your software packaged.

All of these can have issues with trying to create deployment as you want to kind of make it into a general term. Some of the problems could be there's a lot of communication, high-fidelity communication is hard across organizations and especially if it is multi-time zone. All of that stuff can lead to a lot of complications.

Second thing is depending on how your architectural of the application is, you might end up introducing regression problems where some changes end up breaking some other things that were actually working. Trying to figure out that everything that was already shipped is still working the way it is supposed to work and your new changes are not adversely affecting anybody. All of that takes a lot of time. That could slow you down.

The third one is just the packaging technology. If you were deploying 10, 15 years , people were deploying it into physical machines and that would mean every time you need to redeploy, there's a ton of preparations that you need to do on a physical machine or to get it to deploy. That kind of moved into virtual machines, which alleviated some problems, but it was still a machine that you have to bring up.

Now, we are talking about containers, which makes it even more easier so that the lead time for preparation of those machines can reduce, and a lot of dependencies that you have can be back issues for your application itself. It’s all about reducing variability.

As we kind of looked even further, we are talking about functions being deployed that is operating system independent. It's various different things that cause these problems and it's sometimes usually a mix of all of these.

[0:07:08.6] JM: I'm glad you mentioned functions. We'll get there eventually. That's the functions as a service/serverless recent developments. We've done a lot of shows on this. Let's talk about the present first, which is more like Docker. How did containerization changed the deployment process?

[0:07:29.5] AC: I think the fundamental thing that's different is, in the past, application packaging happened in the last — Which means the actual binary bits or whatever that you — I call it the deployable unit. Wrapping of that into deployable unit was happening in the last. What that meant is there's a lot of variability between the point when that piece of software was actually developed to the point where it was actually packaged. That's where you have all these concept of it works on my machine but it doesn't work in an environment that we are trying to deploy to.

There was always this thing, "It works on my machine. It works on old machine and then it works on some machines and eventually it will start working on all machines." That was a major problem in terms of how you deployed applications.

Docker, what it did is it allowed you to package the application first into an immutable image. What that meant is even if you deploy the same version of that image, or in Docker terms, the tag of that image, even 20 years from now you would get exactly the same state of that application. That meant that you get portability, hence the same application that can run on my laptop can be deployed to a [inaudible 0:08:48.7] inspected for test issues and then eventually pushed to maybe a pre-production and eventually to production.

It would also scale up and scale out very easily. You could increase the memory. You could increase the CPU. You could increase the number of instances of that particular container running. It gave folks a lot of flexibility in terms of how you manage an application, how you package an application, and how you deploy an application.

[0:09:14.1] JM: Oh, that sounds great. What is hard about deploying these containers? It sounds like I guess some great features out of it, but what is difficult about this process?

[0:09:25.2] AC: I think from a challenged perspective, what you're now doing is, in the past, the number of version of a particular package existed far fewer. Whereas with Docker, you're not creating immutable images way up during the software delivery chain. Now, if some of the folks kind of want to remove that problem, which is, "Hey, I'm getting these container image scroll and I want to do it later," but then you're really not getting immutability. It all is a balance between that too.

Finding that right balance between how many distinct versions do you want to maintain versus how far down the lane you want to actually clear the immutability is a hard decision.

[0:10:14.1] JM: We've been talking about process of deployment in the abstract. The more contemporary term is this continuous deployment. Describe what continuous deployment or continuous integration or continuous delivery, whatever continuous term you want to use. I think these all basically mean the same thing. it's like different people in the organization talk about it differently, but everybody is talking about pretty much the same thing. What is this continuous X?

[0:10:44.5] AC: Actually, I disagree on that one. They all actually mean very different things. I think that's one of the problems is that people use them interchangeably, and that's where the confusion starts. Eventually, they all kind of mean the same thing depending on the — Actually, they don't mean the same thing. It actually means whatever I think it means even if you use the word continuous deployment. I might think of it as something completely different than what you actually intended it to be.

One of the hardest problems what we have faced at Shippable is that we have tried to level set what these terms mean and so that if we can get everybody on the same taxonomy, then the conversations can be a lot better and more constructive.

I'll try and define them. Continuous integration is basically a development process, and what that means is on a single repository of code, if multiple developers are working on it, all the developer changes have to be as frequently as possible pushed to a common repository. Once you push it, you need to run a bunch of unit tests to make sure that your push or whatever your changes got merged into that particular repository didn't break. That's all continuous integration means.

Usually, you use a continuous integration server like Jenkins or like TeamCity, or in the cloud it could be CircleCI. Of course, Shippable has a CI server in the cloud too, or Travis. All of these guys, what they're doing is they're listening to changes on source control. The moment any change happens, we run a bunch of tests on top of it and make sure that the consistency of your code base is not broken. That's what CI typically means.

Now, a single application is not made up of a single repository. What ends up happening is multiple different repositories are source controlled blocks of place where you put your components source code. They all have to come together in order for you to create an application.

Now, what a lot of folks do is they build byte lines to kind of take once the CI Passes, I'm going to now deploy the latest version of this particular repository score into a dev environment so that I can run some functional testing on top of it as an application as a whole. That's kind of what we call as a pipeline.

A pipeline is basically a set of activities that run in sequence with a very clearly defined starting point and a very clearly defined endpoint and all of them happen in a continuous manner, in a continuous fashion. Those are pipelines.

Now, let's talk about continuous delivery now. What continuous delivery means is very very simple. It means that the tip of your source code, which is whatever was the last push of your main trunk or master branch is always deployable to production. It shouldn't be like, "I made a bunch of changes to my tip of my master, and then suddenly everything has broken and I can't deploy to production anymore." That state where you're in where your entire applications, how many ever repositories you have across the system. If the tip of all of it can be deployed to production if somebody chooses to, then that is what is continuous delivery.

Now, if your automatically deploying that tip to production on an ongoing basis, they keyword here is production, not any other environment. That is continuous deployment. Usually, if you want to reach continuous deployment state, it's almost — I call it the elixir of DevOps life, and that's what is continuous deployment. Everybody is trying to get there and there are very, very, very few companies that actually effectively do it today because it requires a lot of investment and a lot of time for you to be able to get that.

[SPONSOR MESSAGE]

[0:14:50.8] JM: VividCortex is the best way to improve your database performance, efficiency, and uptime. It's a cloud-hosted monitoring platform that eliminates your most critical visibility gap, providing insights at 1-second granularity into production database workload and query performance. It measures the execution and resource consumption of every statement and transaction, so you can proactively fix future database issues before they impact customers.

To learn more, visitvividcortex.com/sedaily and find out why companies like Github, DigitalOcean, and Yelp all use VividCortex to see deeper into their database performance. Learn more atvividcortex.com/sedaily, and get started today with VividCortex.

[INTERVIEW CONTINUED]

[0:15:54.1] JM: Okay. Let me just reiterate those because I think you kind of enlightened me a little bit. I've done so many shows on this but I guess I just have forgotten these definitions and they've gotten ambiguated from me doing so many shows about it. You're saying continuous integration is every time there's a push, you're constantly running tests against that build, that new build, and then maybe it goes from a test environment, it gets promoted to a staging environment and then maybe you have some other stuff that goes no in the staging environment before it gets approved at the prod environment. Then if it gets approved to the prod environment, if you just push it directly to prod — I'm sorry. If it gets directly delivered to production — No. No. No. Sorry.

[0:16:47.9] AC: Let me try to reiterate it in a quick way.

[0:16:52.1] JM: I'm so bad at learning this terminology.

[0:16:54.7] AC: I know. I know. That's one of the biggest challenges with everybody. You're not alone in this space. Continuous integration is, usually, software is developed by multiple developers, and all of that software goes into what we call as a repository, which is where all the source code is maintained.

Now, what happens usually is developers take a copy of that source code on to their machine and they make a bunch of changes and something starts to work. Now, what continuous integration dictates or at least encourages you to do is to keep pushing all those changes as frequently as possible into your repository. If 10 developers are working and if you can — Usually, 10, 15 years ago, we used to call it the Friday afternoon flush, which is developers are working on a bunch of changes for 5, 6 days, and then everybody on Friday afternoon at 4:00 will end up pushing their change at the same time into this repository.

Nobody cares about what other people have changed, because they're all coming together, and that basically means you end up overstepping on somebody else's work very very easily, and that's where continuous integration tries to kind of encourage you to do it more frequently than doing it in one big push.

The way they're verified is by running some what we call as unit tests. The moment that change is pushed so that you've not broken anything. You never allow multiple people to push to the repository if the unit test have not been passing from the prior person's push. You always do it very very seemingly. That's basically what continuous integration is.

Continuous delivery means you are making sure the edge of your source code is always deployable to production. That means you've ran everything and you know that this thing works but it has become either an operations decision or business decision to decide when to push this production.

[0:19:00.5] JM: Right. Okay. In continuous deployment is where you say, "No. There's no business decision. We just always ship it. As soon as it's at the master, we just ship it."

[0:19:12.9] AC: Yes. There could be — That basically means production is already deployed. That doesn't mean users are still using it. You are using various different techniques, like feature based routing and all of that stuff in order to route traffic into it. As far as I'm concerned, from a developer's perspective, that core has already hit the production servers.

Now, the ultimate goal of what you're really trying to do with DevOps is putting the decision of when users will use this feature into the hands of the business and taking it away from the hands of the operations or the developers.

[0:19:56.2] JM: Okay. I see. This is where we get to this term feature flags. Why don't you define that term?

[0:20:04.3] AC: Traditionally, the way we did development back in the early 2000s, late '90s, was all about what we called as environment-based deployment, which is you had a Dev server, you had a test server, or a test environment and then the pre-production environment and a production environment. What we did is we released a particular version into Dev, run a bunch of test, whatever Dev tests were, and then eventually pushed it to test, run a bunch of user Dev acceptance tests and all of that stuff. Push it to production and then pre-production and ran a bunch of security audits and all that stuff. Eventually, it made it to production.

The moment it made it to production, everybody who are our customers or who's using the product gets all the changes at the exact same time. It's kind of like a block deployment. It's just moving one block over to the next one, and the next one, and the next one. That's how traditionally software was being deployed.

The problem with that is your test matrix and all of that stuff is significantly large. For you to be able to say, "I want to be able to deploy this to production," usually ended up taking multiple months, sometimes multiple quarters to get it out there.

Now, some of the modern companies, like I can name Google, Facebook, Amazon, these kind of the usual suspect, what they've really done is they have said, "Look, all the features should be able to be deployed to production whenever they're done." What we do is we actually use the user profile or the person who's actually logging in or using a product, we done on certain flags on their profile that tell our infrastructure what part of the software should be used in order to serve whatever request they're making. That was is a feature flag.

What you end up doing is every single feature that you paired comes with a flag and if that flag is present on the user's profile, that person will end up using their flag. You could potentially — Let's say if you're using Facebook Messenger, I could add a new button to that Facebook Messenger which kind of could allow me to actually have a video conference with you, but that video conference flag doesn't exist in my profile so my Messenger will not show it, whereas you have it on your profile and it shows it. Even though we both are hitting the same exact server on which the Messenger is instantiated.

[0:22:32.0] JM: How do these different feature flags typically map to new functionality? Is it like you have a tight coupling between a feature flag and a micro-service or is it something more granular than that?

[0:22:48.7] AC: It's a little bit more granular than that. It's actually within a micro-service. You can kind of say if the flag is A, then run this piece of code. If the flag is B, then run this other piece of code. It could be forking at micro-service level. It could be forking at a service level, or it could be even forking at a function level.

[0:23:10.5] JM: How are people typically specifying that? Is it like annotation? Give me some description of the end-to-end engineering that goes into — If I'm a developer or I'm releasing a new feature or I want to put it under a feature flag, what am I doing?

[0:23:26.9] AC: I think at the end of the day, it all becomes part of the release manager or the person who owns the service, how they're actually trying to drive. What a developer is typically kind of saying is, "We've built this new feature. We have called this —" There is some taxonomy and nomenclature that is standardized across the entire system. Then developer uses that nomenclature or that team which is building it uses that nomenclature, and that get pushed as a service discover aspect where the release manager will get the latest notification of that service discovery which kind of lists out all the different feature flags that have been added as part of that particular service.

The release manager can decide how to package it. It could be — I don't think people are managing it at each individual feature. The way they typically manage it is in X-Box Live, and I was working on — We used to always have people who belong to either the green bucket or the blue bucket or the purple bucket and we would just decide all the users in the purple bucket will end up keeping this feature first.

The way we do that was it was all throttled and we will always have X-percentage of people sitting in the purple bucket. Typically, these guys are early adopters and so we throttle it to 2% of our traffic and then eventually blue bucket will hit 30% of our traffic. Green bucket is it hits 100% of our traffic.

[0:24:55.7] JM: Is that still how companies are doing that user partitioning, like figuring out the subgroups to whom they want to deploy new features? It's still just like these broad bases or has it gotten — Has it changed?

[0:25:11.8] AC: I think it depends on the sophistication of the company. A lot of folks are getting a lot more granular. If you kind of think about Facebook, they might be segmenting their customers based on age groups, male versus female, left versus right. All kinds of stuff. At X-Box, we didn't have — I'm talking — I left Microsoft like 6, 7 years ago, so I'm talking like 10 years ago. This is kind of how we did it.

I'm sure that with more of like the databases getting larger and larger and the ability to process more data faster and faster, I think people going at a much much more granular buckets.

Usually, also, it's region specific. I think one of the broader best thing that people — The broader brush that people use is just I want to release it to a particular zip code or a particular state or that kind of stuff.

[0:26:07.9] JM: Companies that are trying to automate their deployments and get to the DevOps Nervana where they have continuous integration, continuous delivery and continuous deployment, they often have several disjoint teams throughout the company. There might dependencies on each other. Is this ever an issue with getting to a fully automated deployment where you have different teams that are relying on each other or does this typically — Is that an issue?

[0:26:43.8] AC: This is the biggest issue. What we believe is that if you kind of look at — This is kind of — I don't want it to be like a Shippable bitch here, this is exactly what we do. I'm just disclaiming it right away, but I'll try to speak to the concept as much as I can.

If you look at every industry in this world, eventually you have to get all of these things automated and talking to each other without human involvement. You want high-fidelity, tool chain collaboration is what you're looking for.

Today, everything that DevOps talks about is all about like what I call it as cultural cooperation. It's not really collaboration, because collaboration is two people working on a common thing to make it successful. Cooperation is more about one person supporting the other person to make that person's goal successful.

We are still in the cultural cooperation world and we need to move to tool chain collaboration where all of these tools are all talking to each other. What's happening today is all of these — I call it DevOps activities are all getting automated in siloes. What that is doing is it's creating islands of automation, and a simple island of automation is, "Hey, I have a CI server that is doing CI, but that doesn't talk to my release automation server."

Another thing could be, "I have a server which builds Amazon machine images using a tool, and anytime it builds it, none of my environment is where that a new image has been deployed which I need to actually put and pretty much redeploy all my test servers with the new image, because that is going to make it to production at some point of time."

All of these automation is all happening in siloes. What you really need is assembly lines, and there's a fundamental difference between pipelines and assembly lines. Pipelines is a series of activities that are all sequenced in serial and has to begin and end around the same time. Say, a simple thing could be, "Somebody pushed something to source code. I need to do a build, I need to do a Docker image, and then I need to run some unit testing," and eventually I push to a Docker hub. That entire thing is a Dev pipeline.

Now, the next pipeline could be a deployment which is I not got a new image that should trigger a deployment, but that deployment should not occur until the three other components that are also needed for this deployment are also finished. Now we start creating these complex dependency workflows, and that's what is an assembly line.

Where we believe that the next step of DevOps need to go towards is this concept of assembly lines where you're connecting not just development activities, you also connect security operations activities, network operation activities. Pretty much the entire thing into an end-to-end view so that you can kind of manage your entire software delivery as an assembly line process.

That is what is where we need to actually go where all these tools are talking to each other. The most important thing is, today, people are kind of building it, but they're doing point to point integrations. That basically means for this particular application, I have somehow connected my CI server to my deployment server. That doesn't scale, because now I'm talking 50 other applications, hundred other applications in my IT portfolio and they all aren't [inaudible 0:30:18.2] doing point to point integrations for all of these stuff.

What works for one will not work for another. You need to have some kind of a platform that does this for you, and building these assembly lines should not take too much time. It needs to be super simple to build and it needs to be as declarative as possible and so that you're not writing a bunch of if, then, else kind of code. It should be more of like saying, "I want this to connect to this." That's a declaration as supposed to writing code to actually make them connect.

Then most importantly, you need to have end to end visibility with high-fidelity telemetry around it. We need to know everything that's happening so that these islands of automation, if the bottlenecks exist in between the islands of automation, you can actually go and optimize it.

That's basically the next wave of DevOps. Then once you kind of get these assembly lines, now you can get with the telemetry continuous improvement which is basically what the Toyota production system was all about, and we need to build that for software engineering.

[0:31:16.4] JM: You're describing a platform that pulls together these different places where it's nice to have integrations, like CICD, feature flagging. You certainly want a lot of observability. You want to have monitoring around your deployment process, because that's how you figure out if something's gone wrong, or it's one way to figure out if something's gotten wrong especially if you don't have matrix, like hooks, to rollback a deployment automatically if something goes wrong. You want to be able to have observability over it. Am I describing that —

[0:31:55.6] AC: Yeah, it's very very similar. Those are all very valid points. It's all about — You want to get more comfortable about continuous deployment if you can reward a wrong thing with high confidence. That's the first step. It's kind of like skiing. You are scared of skiing or learning how to ski until you figure out how to get up in the most effective way.

The first time I learned skiing, it took me 15 minutes every time I fell to figure out how to get up. Once I figured out how to get up, then I could actually learn skiing much faster. It's the same thing with continuous deployment, is people are afraid to do continues deployment because it takes them too much of an effort to rollback in case something that was not fully ready got deployed. You got to have all of that stuff in place.

It's not just about CICD. It's also about integrating all aspects of your assembly line. There are dependencies from the network team. If they're close to a particular board, your application will not be able to communicate. All of that activities across your entire idea organization, no matter how small or how big it is, all need to be connected to each other. Today, that entire connection is happening with either email, Slack, or some sort of a spreadsheet or something like that, and that's why I call is cultural cooperation.

It's kind of like if I look at in my past life at Microsoft, if I needed to know what is the status of my image processing API, what is the version that's running my test environment. There was no way for me to get that. There was no service discovery. There was nothing. I had to actually call the team that does it to tell them, “Hey, what is the version that you’re using that we can actually integrate into that particular version.”

If I know the version, now, I don’t know the endpoint. I don’t know what is the URL which I have to connect for that particular test environment. All of these needs to automated. Service discovery is one of the building blocks of trying to get to this assembly line view of the world.

[SPONSOR MESSAGE]

[0:34:09.8] JM: Your application sits on layers of dynamic infrastructure and supporting services. Datadog brings you visibility into every part of your infrastructure, plus, APM for monitoring your application’s performance. Dashboarding, collaboration tools, and alerts let you develop your own workflow for observability and incident response. Datadog integrates seamlessly with all of your apps and systems; from Slack, to Amazon web services, so you can get visibility in minutes.

Go to softwareengineeringdaily.com/datadog to get started with Datadog and get a free t-shirt. With observability, distributed tracing, and customizable visualizations, Datadog is loved and trusted by thousands of enterprises including Salesforce, PagerDuty, and Zendesk. If you haven’t tried Datadog at your company or on your side project, go to softwareengineeringdaily.com/datadog to support Software Engineering Daily and get a free t-shirt.

Our deepest thanks to Datadog for being a new sponsor of Software Engineering Daily, it is only with the help of sponsors like you that this show is successful. Thanks again.

[INTERVIEW CONTINUED]

[0:35:36.7] JM: Okay. You talked about these different integration points that might not make the most sense. Email or Slack, you probably don’t want these places to be the complete hub for doing your deployment process. Maybe you want a Slack bot that you can integrate, you can interact with to help do deployment, but do you really want to pull everything into Slack, like it’s an operating system. Probably now.

You're saying that is there is enough integrations that are around these deployment process that you want a platform which is Shippable, that's what you're working on. Explain —

[0:36:19.6] AC: It's not just — Slack helps you in the eventing process. There's really nothing wrong in using Slack, but what Slack doesn't give you is the state. It doesn't have — It's not a database where every single deployment, every single version is actually maintained so that it can be used across your organization.

For example, if I just asked somebody if they're using a large enterprise — We ask this to a lot of our customers, it's like saying, "If you want to know what is the security approved —" Let's assume they're on Amazon. If they want to know what is the security approved Amazon machine image for your enterprise to deploy a web server into production. There is no system that can tell you what that is. The only way you can do it is you have to call the network security operations or to have a meeting or you have to either ping them on some sort of a communication tool, like Slack or whatever you're using inside your enterprise to say, "Hey, Bob, what is the AMI for a web server for Ubuntu 1404?".

That is not going to get you to continuous deployment. You have to be able to have systems that maintain all of these information and you have to be able to not just get it when you want it. It also is the other way where let's assume, now, Bob wants to do a patch on this particular web server because of certain security aspects right now. He should not be having to hunt down the entire idea organization to figure out which are all the teams that are using 1404 Ubuntu as a web server.

I should be able to have entire assembly line where, "Hey, we are changing the AMI-X to AMI-Y and everybody who is using X trigger their workflows so that we can actually make sure that the latest security batches are already tested and fully ready to deploy." If you're doing continuous deployment, once that AMI goes through the testing and everything, it eventually will also get pushed to production. If you're doing continuous delivery, you know that the particular change, everything is ready. Now, there is a human who come in and clicks a button which actually pushes it to production.

[0:38:29.5] JM: The people who you're onboarding with Shippable, are they typically customers who don't have a CICD pipeline in place yet?

[0:38:40.8] AC: Usually, most of the people who we talked to, they already have a CI solution. What they are not able to do is get this end-to-end workflow happening, and that's where the biggest challenge is. Of course, if you don't have a CI system, the Shippable platform actually comes with it, but we are not really forcing any of these customers because most of the people have invested in one or two of these automation systems already and you don't want them to be throwing all of that work out just because there's a new kid in town. What you really want to do is to kind of say, "Hey, it works with whatever you have and it paves the way for what you might need in the future."

[0:39:24.9] JM: If I'm a company that has a CircleCI, I've got CircleCI for my continuous integration. I've got DataDog for my metrics, and I want to have better visibility and rollback ability in one place. I want to have the continuous integration of my CircleCI pipeline that I've already setup and I want to have the metrics of my DataDog that I've already set up because I want to be able to respond to those metrics in the same place that I see those metrics. That's one of the places where your company Shippable would be practical, because it centralizes both those things in one place.

[0:40:11.8] AC: Exactly. The other thing also is let's say that I want to go towards a simple situation where I have secrets that need to be encrypted right now. I don't have a solution for it. There is clear text happening between CircleCI servers to whatever deployment endpoints I'm trying to push to. What you want is some sort of an encryption mechanism.

Now, Shippable comes with that as built-in. Now, if you already have an encryption system, you can connect to it within your enterprise. If you don't, we already have inbuilt into it, just the same way as CI is inbuilt, that is encryption inbuilt, that is state management that's inbuilt, that is service discovery that it's inbuilt, continuous delivery. All of that stuff is inbuilt, but you can't replace them out with whatever your enterprise uses as modules.

[0:41:03.2] JM: You use Vault for that secret sharing stuff, right?

[0:41:05.7] AC: Yes, we use a community edition of Vault.

[0:41:08.0] JM: Yeah. We did a show about Vault. Anybody who's curious about that secret sharing stuff can listen to that Vault episode. Can you describe — Because I find this idea of Shippable interesting, this platform where you want all these different integration points. I don't know another company that's doing that. Can you describe how some of the people who use Shippable, what they're doing with it or some of the — Give me some of the more extreme use cases. What are some of the weirder use cases where maybe customers are surprising you with how they're using Shippable.

[0:41:45.1] AC: I think one customer comes to my mind, Accolade. I think they are a series B or series C kind of startup funded by Andreessen and a bunch of guys. They're a healthcare company. They actually have created this concept of templatized assembly lines. Every single micro-service that they create, a central team ended up creating a templatized assembly lines on how to take that micro-service into production.

Now, what that template does is it looks at the assembly — The micro-service name itself, the repository name, and if you run a simple command, it will replace all those commands and then create an assembly line for you in less than 30 seconds. Then suddenly, as a developer of that particular micro-service, I can now push it to a test environment and even take it into production without having to learn how continuous deployment, continuous integration, any of these things work, because I just got a template that my IT team gave it to me, or the central DevOps team.

In that way, what I've seen, I have sometimes looked at their pipelines. They show it to me as like, "Here is what we have done." They have over 1,700 individual activities that have all been connected together into this one single view where if I was the CTO or the CIO of Accolade, I could just look at one single view and say, "Hey, is my entire organization on track? Do I see any red boxes anywhere?"

It's kind of like if you look at Toyota's concept of Jidoka, which is automation with a human touch, where the automation system should alert the human is something is wrong, not the other way around. This is exactly what that this. It's just that they've built this massive view of their entire application stack and they're all connected as assembly lines. Getting a micro-service on to this platform is less than a couple of minutes for them for a developer. The developers absolutely love it.

[0:43:45.9] JM: I'd like to talk a little bit more about the broader topic of deployment and software delivery. I think the word that has the most excitement around it today in this topic might be Kubernetes or at least it's one of the important words. Kubernetes is, of course, the container orchestration platform that was open sourced by Google and it's really gotten a lot of excitement around it. How did Kubernetes change CI or how did it just change the deployment process from your point of view?

[0:44:24.7] AC: I don't think it changed the CI world as much. What it actually did is if you'd really want to do future-based flags and all of that stuff, which Google has been doing for, I think, a decade almost, a decade or more than that, you need a platform like Kubernetes.

What they've done is a container is a deployable unit, but you need to be able to create isolation between environments. Let's say I have a simple application. Let's call it an API application. It just has a bunch of APIs. What you can actually do with Kubernetes is if I have an API container that I want to deploy, I could have one cluster of Kubernetes which has a whole bunch of VMs that are sitting under it. As far as a developer is concerned, he or she is completely abstracted from that.

Now, on Kubernetes cluster, I can actually create a name space which actually allows me to say, "This is my Dev name space, this is desk name space, and this is my production name space." It gives you the isolation between all of these name spaces so that containers in my Dev name space cannot talk to containers in a desk name space, which is the only reason why you have different environments. The beauty of this is now I have a same common machine that is running. It could be running 20 VMs. Some parts of that VM could be running development workloads. Some of part of the VM could be running desk workloads, and some part of the VM could be running production workloads. That isolation, it gives you. Even if you want to take it to the next level, you could also use labels on top of it. Within that name space, you can actually create routing between containers using labels.

You can kind of say, “You cannot allow container with a label prod —” If a container has a label prod, then only containers with the prod labels in it should be able to talk to it. You can start clearing all these virtual isolations on top of a common infrastructure platform. What you’re really done is they completely decoupled the infrastructure from the operations of the application. The moment you do that, then the folks who’ve focused on the infrastructure are purely focused on what is my CPU utilization, what is my memory utilization, what is my storage utilization. That kind of stuff. At any given point of time, let’s say I have 10 VMs running under the Kubernetes cluster, I can make it 50, or make it 20, because I need a lot of demand that’s happening. I don’t need to redeploy any of the applications because the orchestration layer will take care of it and rebalance across all of these VMS. That kind of flexibility is just unbelievable what you can get by having an orchestration platform like Kubernetes, and it’s definitely one of the more popular ones that people use.

[0:47:16.2] JM: What are the other projects in the CNCF — Because you specified, the CNCF is the Cloud Native Computing Foundation, which is this open source foundation that was kind of created to how Kubernetes and associated projects or related projects, and it’s got things like Prometheus. Just other lower level technologies that are associated with Kubernetes. If you could look in the CNCF and understand what’s going on in the Kubernetes ecosystem, is there anything particular that you see in the CNCF that’s exciting to you?

[0:47:55.0] AC: There’s a lot of technology that’s coming out. I just heard recently about a company that is doing some crazy stuff in terms of — I mean crazy, awesome stuff in terms of container security and they’re still in stealth mode, but they have done something. I think it’s just awesome in what they’re trying to do, because if you really think about what they’re really trying to do is like if you think about how hackers thing of it, you need to have an endpoint that are listening to you so that they can actually use brute force to actually get it.

What these guys are doing is because container technologies are portable, they’re constantly moving the containers around. In the Unix world, there used to be shadow file where all the password, encrypted passwords were being stored and it was so difficult to hack that because the shadow file will keep moving around and only the operating system knew where he shadow file was.

If you kind of take the same concept, if we can make that happen to applications, and because all of the routing, all of the stuff is virtual, these things are all moving around. I cannot get more than — You could even do it at five-second interval if you really want it, and that basically means in five seconds, if I can't figure out how to hack it, I no longer have access to it because it's moved to somewhere else.

Those are kind of the cool stuff that I'm seeing where some of the things that we could not do before — Think about trying to do that on a VM, just imaging of VM will take you an hour to do. Let alone, moving it around, copying it to another place and all that stuff. This is like really really cool stuff that's happening. I don't think it's all public yet, but I think these are kind of things that I look at and say, "That's an amazing idea and how it actually works."

[0:49:33.3] JM: That security containerization thing, you'd be moving around middle service endpoints. You wouldn't moving around the endpoint that the user is accessing.

[0:49:46.9] AC: You can. If you think about it, most of the containers even if you are doing something, you're always behind a load balancer. Yes, your load balancer endpoint is there, but let's assume that your underlying provider to that load balance that is there, it doesn't accept any more new connections. It's waiting to finish off and drain off whenever existing connections are already there. You want to be able to access it anymore. Even though that's live and it's finishing what it's doing. You can actually do it even at the edge level if you really want to do it.

[0:50:20.0] JM: Okay. You mentioned earlier, deployment is going the way of services. What did you mean by that statement? I'm sorry. Not service — Functions. Deployment is going the way of functions. Right now we're doing we're deploying services, or containers, we'll be deploying functions.

[0:50:36.9] AC: If you really kind of think about it, every programming language is good at doing something and it's not so good at doing something. It could be a simple thing like some applications are very good in managing, sorting, and arrays and that kind of stuff. Those kind of programming languages tend to use that kind of applications. Some of them could be using — Be very good at doing some sort of regression analysis, like R and things like that, but not all language are good at all the things that are out there.

If you kind of agree to that as a concept, then what if you could actually pipe the results of one operation into another operation, because that's exactly what we are doing even when we are moving within the same functions. We're just having memory that's being supplied and we are just piping those memory. If I have a Node.JS application which is very good at asynchronous operations, but my async operation is about sorting, or it's about regression analysis. What I could potentially do is call a bunch of R functions and then wait for the results to come back and then combine them it all together and return back a result just because Node.JS is very good at async and how it actually operates.

If you kind of think about that kind of a thing, then what are you going to actually potentially do? Another simple example could be digital image processing. You might want asynchronous operations, but you also want to do some sort of image processing for which you could use certain specific tools to actually do it for you. If you kind of think about that kind of an architecture, you now longer bound to one single process that needs to execute. Hence, I can actually create a program, which is partially Node.JS, partially R, and partially some image processing package that I have and connect them all together using a serverless Lambda kind of workflow.

If I do that, then I get unbelievable performance because I'm getting the best of breed of everything. The architectures are moving to that kind of a world. Then more importantly, you could scale these things so as independently. So maybe my digital image processing requires 3X the amount of resources than my sorting function. Hence, I can change how much resources does the image processing function uses versus how much resources I need for my sorting.

It's all about making it more efficient. It's no longer about finding that lowest common denominator that works for me. It's more of like saying, "What are the highest common factors of all these things?" And I want to take all of that and combine it and create an application that is at the highest common factor as supposed to lowest common denominator.

[0:53:30.6] JM: It's a very clear vision you're providing. All the different cloud providers are working on their serverless strategy, you've got Azure, Cloud Functions, AWS Lambda, Google Cloud Functions. Do you see any notable difference between the approaches? I guess talk about the overall serverless landscape and how you see — I actually really like your vision, because basically you're describing a vision where you go from micro-services to maybe micro functions, or just functions or however you want to describe it or you just write code in a bunch of different languages and it gets magically deployed and all of the different functions are broken up into their own little functions as a service. To me, today, that feels like a very far off vision. Are there any companies or people who you know of that who seem to have this mapped out or they're working on the stuff that's going to get us there sometimes soon or are we still just very far from anywhere resembling what you described?

[0:54:45.3] AC: If you kind of say, "Do you have an application that is 100% written in this kind of an architecture? How far do you think it will be before we get something like that?" I think we are quite from that. I don't think there is an application that is going to be 100% using this kind of an architecture in the near time, anytime in the near time soon.

What you will see a lot more often is that there are long poles in a lot of different things. For example, in Shippable rate. We have to show you console logs to exactly tell what's happening on the server side so that users can see this on the UI. For that, there is a lot of pre-processing that we need to do before the UI can actually display, and that is usually very rare in terms of sometimes an application that we're running runs, maybe outputs 10,000 lines of console. Sometimes they output 10 million lines of console.

When you kind of start thinking in these kind of problems, we initially — And we were in the world of like, okay we all had to be on a single homogenous kind of platform. We had to scale everything to the worst case scenario because you could get 10 million consoles.

Today, what we have done is we have just isolated just the console processing out into a separate function and I can easily scale that up and down. You will architectures like that that will come very very soon where a lot of folks, if they're not already doing it, they're already thinking about it is kind of take their long poles and isolate it out into these kind of functional programming.

In terms of different providers and what they are making different available is that I think both — I don't know about Google enough. I haven't dug in enough, but I know for sure that you can't take Lambda and run it on your processes. You have to be in the Amazon Cloud. I think Azure functions, you could actually run it as your own stuff on premises too. That's one clear differentiation where anytime in the near future, your hybrid model is going to remain — I don't think it's going to go off anytime soon the next. I think it's next 10, 20 years, it's going to be around.

If you have that kind of a scenario, Azure has probably a better story there because their functions can run both in Azure Cloud as well as on premises if you're running open door server.

[0:57:09.9] JM: What else is in the different cloud providers race is interesting? I've done shows about Google container engine and I've looked at ECS and Azure Container Engine. It seems like this is kind of the platform as a service for containers. What do you think of the platform, the containerize platform as a service space? It seems like every company — I don't know. Maybe not every company. Some companies want to operate their own Kubernetes cluster, but it seems better. How much adoption are you seeing in the container engine space as suppose to — This Kubernetes as a service basically.

[0:57:54.4] AC: Yeah. You're talking about Google container service which runs actually Kubernetes underlying it.

[0:57:59.5] JM: Well, and I think Azure container engine does that too.

[0:58:03.0] AC: Yeah. I think Azure does both. It does Mesos as well as Kubernetes, and also it does Swarm. You can pick which one you want. Amazon, I don't know what they use internally, but it's custom. I think it's two things. One is what we are seeing in our customer base, and this is where I think we just announced our server product just last week and we were forced to do it. We were purely a SAS kind of offering and what we started is even though a lot of enterprises are adopting cloud, the way they are adopting it is a logical extension to their data center.

It's all virtual private clouds where they are basically saying, "Hey, my data center has subnet," let's call it 10.0.10.0, and I'll also add one more subnet and connect that and that secondary subnet is actually running on Amazon. Even though it's running in the cloud it's not really like they're running on a public cloud. They're actually running on private, completely firewalled environment. If you're in that kind of an environment and that's how you want to run your infrastructure, which a lot of companies that I know, even though they are startups, they actually run it this way. It's predominantly because they don't want noisy neighbors.

They don't want their data to be compromised because the SAS platform on which they are on did not have the right processes, whatever it is. It's a very hard thing for a lot of folks. Because of that, I think they will run private networks like this. If that is the case, then these cloud provided container services is really not that attractive for these folks. Most of these guys are running a private instance of Kubernetes as a server themselves and managing all the operations. Or maybe Mesos or whatever they want to use.

I think we see that a lot more, especially among midmarket and enterprises, we see that quite a bit, because if you really think about it, one of the biggest barriers for me when I was kind of thinking about Shippable was people need to give me access to their source code, and I know that we are doing everything possible to keep it secured. At the end of the day, still, they're giving me access to their source code. Will they actually ever do that?

That's where I think if I can go look at our customer size and the kind of maturity of our customer is changing, most of these guys are running either GitHub enterprise or big buckets server or GitLab enterprise or one of these server product. The moment they say that, Shippable SAS is completely a non-startup for them because they want — If they're saying the source code has to be on a server, then they want the whole assembly line for DevOps, all of that stuff should be also provided to them as a server.

I think that's where I see, still — As much as I've been in SAS, I still see that for the next 10, 20 years. That's going to continue to exist and we'll have to see whether that will ever change over to pure SAS or not.

[1:01:17.9] JM: Avi, thanks for coming on Software Engineering Daily. It's been great talking to you about deployment —

[1:01:21.3] AC: Thank you very much for the opportunity.

[1:01:23.1] JM: Yeah, it's been really fun. Okay. Cool. Thanks.

[1:01:25.8] AC: Cool. Thank you. Bye-bye.

[END OF EPISODE]

[1:01:31.0] JM: Spring is a season of growth and change. Have you been thinking you'd be happier at a new job? If you're dreaming about a new job and have been waiting for the right time to make a move, go to hire.com/sedaily today.

Hired makes finding work enjoyable. Hired uses an algorithmic job-matching tool in combination with a talent advocate who will walk you through the process of finding a better job. Maybe you want more flexible hours, or more money, or remote work. Maybe you work at Zillow, or Squarespace, or Postmates, or some of the other top technology companies that are desperately looking for engineers on Hired. You and your skills are in high demand. You listen to a software engineering podcast in your spare time, so you're clearly passionate about technology.

Check out hired.com/sedaily to get a special offer for Software Engineering Daily listeners. A $600 signing bonus from Hired when you find that great job that gives you the respect and the salary that you deserve as a talented engineer. I love Hired because it puts you in charge.

Go to hired.com/sedaily, and thanks to Hired for being a continued long-running sponsor of Software Engineering Daily.

[END]