EPISODE 510

[INTRODUCTION]

[0:00:00.3] JM: On this show, we spent a lot of time talking about continuous integration, continues delivery, data engineering, microservices. These are technologies that have been widely talked about for the last 5 to 10 years and that means they are easy to adopt for startups that got founded in the last 5 to 10 years, but not necessarily for older enterprises.

Within a large enterprise, it can be challenging to make significant changes to how technology is used. Many of the listeners might even take it for granted that your source code is going to be in Git, but if you work in an enterprise that started building software in 1981, you might be moving source code around on FTP servers or floppy disks. The difficulty of changing the technology within a large enterprise gets compounded by culture. Culture develops around specific technologies. That's one interpretation of Conway's Law. That's the way that an organization uses software informs an organization's communication structure. So this is no surprise if your organization manages code using FTP servers and floppy disks. It's going to slow down your innovation, and the software that you produce is going to reflect your communication culture as well, because it won't be able to update as quickly.

Zhamak Dehgani is an engineer at the ThoughtWorks where she consults with enterprises to modernize their software and culture. Zhamak works on a blueprint that describes specific steps that an enterprise can take towards modernizing. Things like continuous integration, building a data pipeline, building a system of experimentation. In some ways, this conversation fits nicely with our shows about dev ops that we had a year ago, a few years ago. I guess we had a ton of shows about dev ops, and it was great to talk to her about the conversations that she has with these large enterprises, because I know there are a lot of people who are listening to this show that work at a large enterprise that are doing their best to modernize, and I know it's a lot of work to modernize a large organization that has certain practices that have been around for a long time.

Full disclosure; ThoughtWorks, where Zhamak works is a sponsor of Software Engineering Daily. To find all of our shows about dev ops and enterprise reinvention, you can download our

apps, Software Engineering Daily apps for iOS or android. These apps have all 650 of our episodes in a searchable format. We have recommendations and categories and related links and discussions around the episodes. It's all free and also open-source, and if you're interested in getting involved in our open- source community, we have lots of people working on the project. We do our best to be friendly and inviting to new people coming in looking for their first open-source project, so no matter how large or small a contribution you want to make, we have quite an interesting platform and an interesting group of people that are really nice. You can find that project at github.com/softwareengineeringdaily. You can join our Slack, and you can send me an email at any time, jeff@softwareengineeringdaily.com. It'd be great to hear from you.

With that, let's get to this interview.

[SPONSOR MESSAGE]

[0:03:26.0] JM: Your company needs to build a new app, but you don't have the spare engineering resources. There are some technical people in your company who have time to build apps, but they're not engineers. They don't know JavaScript or iOS or android, that's where OutSystems comes in. OutSystems is a platform for building low code apps. As an enterprise grows, it needs more and more apps to support different types of customers and internal employee use cases.

Do you need to build an app for inventory management? Does your bank need a simple mobile app for mobile banking transactions? Do you need an app for visualizing your customer data? OutSystems has everything that you need to build, release and update your apps without needing an expert engineer. If you are an engineer, you will be massively productive with OutSystems.

Find out how to get started with low code apps today at outsystems.com/sedaily. There are videos showing how to use the OutSystems development platform and testimonials from enterprises like FICO, Mercedes Benz and Safeway.

I love to see new people exposed to software engineering. That's exactly what OutSystems does. OutSystems enables you to quickly build web and mobile applications whether you are an engineer or not.

Check out how to build low code apps by going to outsystems.com/sedaily. Thank you to OutSystems for being a new sponsor of Software Engineering Daily, and you're building something that's really cool and very much needed in the world.

Thank you, OutSystems.

[INTERVIEW]

[0:05:17.8] JM: Zhamak Dehgani works with ThoughWorks. Zhamak, welcome to Software Engineering Daily.

[0:05:22.9] ZD: Thank you, Jeff. Thanks for having me.

[0:05:25.0] JM: Absolutely. So 2017 is wrapping up, it will be 2018 when this airs, and you've been doing software engineering with ThoughtWorks for a while, and a lot of what ThoughtWorks does is work with larger enterprises to help them figure out how to migrate their software. So what is it like to be an enterprise going through a large-scale software migration these days?

[0:05:51.3] ZD: Yeah, good question. I mean, all of the enterprises I come across, they are going into some form of a migration. We can talk about the motivation as why that's happening. A lot of them — There was a really good reports from HBR about more than 50% of the executive and board members feel that more than half of the revenue is going to be threatened by the digital disruptors. So there is a real threat to industrial kind of organization, the industrial area organization with the digital disruption, and that's triggering kind of migration not only the technology, technologies to meet the business strategy, but also business strategy.

I think we are at a very interesting point in terms of how the technology has evolved and where we are in terms of applying the new ways of doing software, building software that we see

enterprises kind of adopting, but it's exciting, it's frightening. That change always has a bit of resistance from the immune system of the environments, but I think it’s an exciting time to go through technology transformation.

[0:07:04.1] JM: When you sit down with an enterprise that has an out-of-date technology stack, which is a lot of enterprises, even newer companies. Your technology stack goes out of date in a year and you're always updating something, but how do you decide what to prioritize when you get a client that is like a really large enterprise and a lot of their surface area is out-of-date. What do you prioritize?

[0:07:27.6] ZD: Yeah, absolutely. I think, for us, we always think about the customer value, the mission of the enterprise. What are they actually trying to do? When we do technology strategy or migration strategy, we always look at their business strategy and what they have in the pipeline and we start with those. So we start with the domain within the enterprise that will have a high impact — The technology change will have a high impact on their customers. You start with a domain that it may be kind of more ready in terms of the transformation and we go from there.

Usually, the transformation, you try to localize it to kind of a domain and starts building from there. We think about — Even when we think about organizational-wide platform change, everything from your delivery infrastructure, your compute engine, your data structure, infrastructure, your services APIs, we always start with recognizing what capabilities you need to change going through your customer journey, going through your business product backlog and pick a thin slice to drive that sort of transformation as opposed to kind of a bottom-up transformation that at the end of the day may not deliver any value to anybody.

[0:08:43.8] JM: So it's not necessarily around what is the most important aspect of our business, and let's re-factor the technology there. It actually might be the opposite. It might be what is the least important. What can we do that is the easiest? How can we perhaps get a win under our belt and prove to the rest of the organization that this is possible?

[0:09:06.6] ZD: Yeah, absolutely. That's definitely a case, but it also depends on where we come and talk to the organization in their transformation journey. Sometimes there is a new

leader that is pushing for the change and he or she is looking for a very visible and high impact, kind of low hanging fruit to show what can be done and get trust from the rest of organization and push that agenda further.

Sometimes we join the organization when they are way through some form of a platform change or technology change and they're asking us to come in and help them within their journey. So they've already vote in into the idea. They've probably have some metrics at the executive level in terms of the volume of the traffic that needs to go through their new platform and they just now need help to rule that out.

Yeah, it depends where you are, but if you're really early in your transformation journey, you really want to buy, win the organization trust and show value quickly.

[0:10:10.4] JM: There are both cultural barriers and technological barriers. Do you take those separately? Do you try to reformat the culture and then reformat the technology separately or do you try to bundle those together somehow and reconfigure how an organization works all at once?

[0:10:29.3] ZD: Yeah, absolutely. Very good question. I think the engineering, organization, the culture, how the team gets set up, it's all very intertwined with technology. It's hard to do one without the other. We've recognized that in our practices. So, for instance, we don't go in to just prescribe a technology strategy and leave. We think that the culture change happens with showing how things are done by coaching through doing and demonstration of how to deliver value, how to deliver kind of software, make the organizational change, the team structure change incrementally, gradually by forming the new teams [inaudible 0:11:13.7] new structure and go from there. But it's absolutely — Everything is very closely tightly coupled. It's hard to do one without the other. The million-dollar question is that when you're dealing with a large organization, how do you scale that type of change?

[0:11:29.8] JM: So what's the answer to that? How do you scale that change? By the way, when you start with an enterprise, are you sitting down with one specific person at the company or do you engage with the entire team altogether? How do you sow the seeds for what will eventually hopefully be a scalable change in the enterprise?

[0:11:49.4] ZD: Absolutely. We sit at all levels. I think the change requires grassroots movement and top-down executive support. You really need support from top and you need support and buy-in at the grassroots and engineers level. So once we get engaged. First, we really want to understand the landscape we're in, the dynamic between different parts of the organization, the maturity of the technology they have, the capabilities of the developers. We do a quite thorough assessment of where we're going in and we'd be respectful of the constraints that the organization have and the challenges that they have.

We talk at the executive level, at the CTO, CIO level all the way down pairing with the developers and seeing the pain that they're going through, the friction that they have day-to-day to be able to deliver software. We talk also to the business side of the enterprise. These days, CIOs are more business focused than being just the tech heads, functional heads of organizations. Talking to the business to see what's their vision, where they want to go and how technology can support and accelerate that vision. We kind of engage at all levels.

[0:13:06.0] JM: ThoughtWorks has a blueprint for the steps in the different areas where you can add value to modernize an enterprise, and I want to go through some of these aspects of the blueprint and the area that I actually hear companies start with the lot is — First of all, sometimes I hear companies don't even have like a version control system in place. They'll be sending files to each other over FTP or saving files on to discs and then sharing the discs with each other and stuff like that, and that's probably a huge problem that you would want to start with if a company had that sort of issue, but I think we'll skip that one and maybe just say assuming a company has version control, I think that's probably the first step you need to get to a CD system, continuous integration, continuous deployment system. How do you ease an enterprise on to a continuous deployment system?

[0:14:01.7] ZD: Sure. You have to do it while you are delivering software. We can just practice continuous delivery without delivering something that needs to be continuously delivered. We usually pick, let's say, a feature, and then we go, "Okay. Let's put in the infrastructure that you need to be able to continuously deliver those features. Version control, we've already talked about that.

We do have some sort of a maturity model, the continuous delivery maturity model, where we look at the different aspects of the continuous delivery source control being one, continuous integration being another, configuration management, another aspect of that. So we go through that maturity model or different aspects that need to be implemented and we will have people who have capabilities to build the infrastructure to set up the infrastructure for the teams and slowly build the pipeline. You may start with a smaller pipeline that runs that code that you had. You need to get your code under test. So start with some functional test or some unit test. Get the code under test at the test of the pipeline, then you go into, "Okay, be able to release it to different environments. Build the deployment scripts to be able to release that."

Building that whole infrastructure through delivering value through the pipeline and setting that up and be able to kind of show and prove as you go along. We will have people who are capable in setting up the delivery infrastructure, setting up the CD agents, building some startup kits for developers to be able to get up and running more quickly, because what we want to also avoid is having a central team who's now responsible centrally for your build pipelines. That's a very slippery slope to getting into other friction points very soon after. So we want the teams to own the pipelines themselves to own the agents that they're running with the right support from the delivery infrastructure team.

[INTERVIEW CONTINUED]

[0:17:13.3] JM: There are anti-patterns that enterprises can stumble on to using if they don't have the right best practices around continuous integration or continuous deployment. One of the enterprise-wide problems that I've read about is the idea of having this enterprise-wide integration test environment that's like a staging environment, but the problem is that everybody in the organization shares the same "staging environment". Why is that problematic? What's the problem with an enterprise-wide staging environment?

[0:17:48.0] ZD: Yeah. I think it's just fundamentally anything central to me is a bottleneck. It's going to cause some bottleneck for somebody along the way, whether it's the central CI server or central staging environment. Usually, with the staging environments, you have different teams that are pushing code into that bringing the services up or down. They have different data states and what you — Very soon you get yourself into is that you have dependency to the rest of services, and the way you have written your test or your test structure, you're doing this sort of kind of wide integration testing that assumes that data that you need is there, the state is there, all the downstream from services that you have are up and running and this volatile environment that everybody's changing and touching wouldn't give you the known state that you rely on. So it's unrealistic.

So what happens is you have to kind of schedule now access to these central environment so that you can have a known state to start integration testing from, and from there you can imagine that you will be on the queue and you only get a very short time slice to access that kind of server and all the resources that need to be set up.

Also, it's very expensive. Usually, when you want to mirror the production environment, all the systems that you depend on to run these kind of wide integration test, sometimes they cost millions of dollars. Just having copies of it running could be expensive and maintaining that could be expensive for the enterprise.

[0:19:24.1] JM: Yeah. So you're saying that the alternate model is you have some image of the production environment and you find a way to spin up your own personalized fork of that image

that has your own changes on it and then you can run your continuous integration tests on that forked image with your new build.

[0:19:45.9] ZD: Yeah, that's to start with. I think, more importantly, I would say let's move away from this idea of integration testing, and integration testing just before release. So move to the idea of more kind of localized contract testing. It's been a few years that we've been practicing consumer-driven contract testing exactly for this purpose, so that every service or every part of the code can be independently tested and released with a good level of confidence by running the test that are written by the consumers of your application or your service and you run those as part of your pipeline against your service in a localized fashion. You don't have these large number of integration testing that would need integrated environments.

Consumer-driven contract is an approach you can use, and also on the other side, you can run these kind of synthetic transactions and test in the production environment and focus on time to recovery. So if something's not working, be able to recover quickly, so recovery versus kind of prevention.

[0:21:02.1] JM: What are some other CI/CD anti-patterns that people have as they're trying to migrate their enterprise to a place that uses some of the modern software technologies?

[0:21:15.6] ZD: We actually have, I think, few of them on our tech radar in the [inaudible 0:21:21.1] ring, which means try to avoid. One of them, if I recall correctly, is called CI theatre. So the idea is that the teams kind of set up the CI server and they have some tests possibly running, but although there is a CI agents running and there is a pipeline, the code doesn't integrate frequently. There is a large gap between the check-ins and commits. The code might be on feature branches that are not being integrated into the trunk. The other pattern is having a central CI server that everybody's using and having a central team to look after and to create the pipelines, which we create a bottleneck.

All of these really are anti-patterns, because they're against the fundamental concept of continuously integrating your code all the way from your commit to production and delivering to production. I think these are a couple of ones that are on the tech radar.

[0:22:21.3] JM: Got it. I'll put those in the show notes. What about security? What are the security issues that an enterprise might need to address as they're updating their platform?

[0:22:34.5] ZD: Sure. Security is usually function that kind of getting involved in the development late, quite late, and it's about, "I'm going to stop you from putting your code that might have some vulnerabilities right before you just release your code." They get involved quite late in the process and their function is really stopping code to go to production to possibly make the environment unstable or insecure, and that is really changing.

For us, this shift would be security concerns get considered much, much earlier in your development process. So as you write stories and develop the functional capabilities of the application you're building, you also think about the threat model. You think about the security stories and consider them just as part of your — The feature set. Security would get involved quiet earlier and their function would be how can I enable you to build security into your application? In an enterprise, traditionally, security has been built at the perimeter. So we have some sort of a firewall and some form of a zoning and out [inaudible 0:23:53.7] zones, we put some sort of a security in place, and within it, everything's fine and secure, so you don't have to worry, and that's completely shifting, because with kind of more microservices architecture and distributed architecture you're moving into a zero trust environment that you always — You don't trust, you always validate. So security gets pushed from the perimeter to every service and every application that you're building. So security features that you need to authenticating and authorizing the calls coming in has to be built into your infrastructure for developers to kind of seamlessly use it, and that kind of leads into kind of interesting patterns of implementing that and providing the infrastructure to support that and yet have a frictionless seamless experience for developers.

[0:24:42.0] JM: What if an enterprise has some significant overall architectural problems? Like you hear this referred to as the big ball of mud, for example. Just this monolithic situation where there's lots of tight couplings and it's really hard to figure out where to even get started. How should an enterprise approach to that issue?

[0:25:04.6] ZD: Yeah. Both big monoliths, and it could be systems that are 30, 40 years old or they could be systems that are 10 or 15 years old and they have the same characteristics.

Usually, the enterprise — And that one of the problems that they have and that the other end of it is that they have too many system that do the same thing. I remember a CTO of a company that I worked with, every time I went to his room, he had this giant kind of diagrams of all the systems in the enterprise which had — I don't know, hundreds of tiny boxes and each of those tiny boxes did something really, really important, and probably 10 of them did almost the same thing, but not quite the same things. The entropy that you see within the [inaudible 0:25:47.7] entropies within the enterprises is quite complex.

So on one end you have the ball of mud, which is these kind of large systems that are hard to break, and the other end you see this kind of fragmented technology that they kind of do the same thing through by acquisition, through acquisition or building new technology, but not quite finishing what they were doing. So you have the new system, an old system that kind of run at the same time. These are both the two ends of challenges that we see at the enterprise.

Where you get started? You get started with where it matters, and by that I mean you are either — Usually, enterprise wants to change the technology, because they want to go faster. They already have the scale to some extent. They have found ways of scaling the organization, but they need to go faster, and you start with, "Okay. What changes are coming down the pipe from kind of the business and product point of view and what capabilities are getting changed more often?" You can talk to the business, find out the capabilities that they want to change. You can also do the other end, you can do social code analysis on the, let's say, monolith, to see what parts of the code are changing more to find out where it matters to start extracting that.

Then with extraction, again, you have to be very careful, because people — There are people in the organization that are very biased to the existing code, because it's the code that they put their hearts into. It's their baby, and you're coming and telling them, "Your baby is ugly and it has to change." You need to kind of inject some new blood, a new thinking into existing organization, take advantage of the knowledge, that this experience is already there, but bringing some new thinking to decide whether extraction and the reuse of the code out of this bowl of mud is the right approach or is it that rewrite the right approach. I can tell you that most of cases, rewrite is probably the right approach unless you have a very complex IP that you need to retain, and you kind of rinse and repeat the process. By finding the next thing that matters or build the test

around it, put all the good software engineering and continuous delivery practices for that part of good, so it can be independently released, and you redo that.

One thing that really matters in terms of — I mean, all the things that I'm kind of describing are evolutionary change. They're not kind of revolutionary change, but if you're doing evolutionary change, we have to be very conscious of what is the smallest unit of change that can get me closer to the goal and the vision that I have? And that's something I've have seen in the migration strategies. We start with something, we say, "Okay. We want to," let's say, extracts capability X. Let's say it's a big monolith. We say authentication of the user. The security is the first thing we need to get right, so let's build an authentication service authorization service outside.

What happens is we don't fully complete that unit of migration, and by that I mean changing the consumers that are using the existing application as well as changing the application itself. So what you end up with is essentially two patterns. The old way of doing some — That is still part of your organization or part of the application is doing the old way of authorization, authentication, and there's part that are doing the new way. Are we really closer to the final goal of reducing the tech debt to be able to go faster and reducing the cost of maintenance? Probably not. We now ended up with two. So we added probably four more boxes on that CTOs kind of system diagram. Yeah, it's interesting, but it takes a lot of, I guess, discipline practice and collaboration across products and engineering and the support of the leaders, leadership.

[0:29:42.0] JM: It sounds like you've seen a lot of monoliths and big balls of mud.

[0:29:46.0] ZD: Yeah, almost everywhere.

[0:29:47.8] JM: Yes.

[0:29:49.0] ZD: It's very easy. It's a path of least resistance.

[0:29:51.9] JM: It is.

[0:29:52.5] ZD: You go in, you want to add a feature. Should I go build this as this [inaudible 0:29:56.9] or should I just band-aid and add a line of code, a case, a new function? It's a path of resistance that gets really sticky after a while.

[0:30:08.6] JM: So describe, I think, two ways of dealing with it. You could break off functionality into some new service or perhaps you can do a complete rewrite, and there're different gradations of both of those. What about the approach of setting up a façade in front of the monolith and basically having a different API into the same code base, or maybe you could have like a GraphQL server that interfaces with the monolith and basically makes the API surface a lot easier for engineers to deal with while also satisfying the legacy consumers who want to access that API directly.

[0:30:49.0] ZD: Yeah, that's a very good point. I think that's a very good first step, and we see a lot of clients that they go as far as that step. So with façades, absolutely. As you said, you can kind of start building that new APIs and user interfaces that make the consumer's life much easier. You can, yeah, separate your kind of legacy from your new modern way of building the applications. The challenge there is that you can go so far with that approach, and by that I mean very quickly you need to either modify capability, add a new feature that unfortunately is locked-in that monolith. The first, the reason that you want to migrate from the monolith, because the monolith is hard to test and hard to change and doesn't lend itself to continuous delivery.

So you're very much back to square one of I need to change my monolith and it's now going to take 5 months or 4 months and that many dollars to do it, and your digital team is being kind of stuck with that dependency. So though it's a very good first start because you can isolate your channel applications, your mobile applications, your consumer-facing applications, and it's a good way of kind of really finding out some of the limitations that the legacy system have. For instance, a lot of the telcos go through this process, because they have an ecommerce site that they want to be able to sell different products and packages, and iPhone comes out and the sales of iPhones brings out the whole site down, because even though that you have APIs and façade APIs that you built around the monolith, the monolith is not built. The legacy systems is really not built to take the throughput and the load that the system goes under. So you have to

now reinforce and protect those façades and build throttling in place between the legacy and so on.

The next step is, as an example, in the telco industry, almost every telco wants to bundle different packages, different data mobile, broadband satellite and create different offers, but they want to be able to experiment with those and the marketers want to be able to define different packages and put it out there and see how consumers respond.

To do that, it's very hard to do that at the API level. It gets really messy very quickly. So you have to change those core systems that needs the kind of — That change needs to continue, and there are kind of patterns of extending that façade to become these autonomous wobbles in a way that you can try to build, extract capability and [inaudible 0:33:31.1] capability outside of the legacy. It's absolutely a great first step.

[0:33:35.5] JM: There are so many organizations who they started building their systems, like you said, 20 years ago or 30 years ago at a time where we didn't have this mental model of saving all of our data, putting it in a data warehouse and being able to extract insights from it, because I think companies were so busy just trying to handle the day-to-day operations. They didn't even think about saving that exhaust data and trying to run map produce jobs over it or anything, because we didn't really have those technologies. Nobody was thinking about that, but now you fast-forward to 2017, and we start to see a company that is, let's say, a manufacture of toilet seats even. If you've got a factory that is running software that's manufacturing toilet seats, you actually have lots of interesting data. You have data that you could use to save money further down the line if you're actually saving that data. So many of these companies, they're thinking about, "How do we re-architect our system so that we can aggregate this data and we can make it available to other people inside of the organization as a data lake?" They're just like totally unprepared to take advantage of this this data that they have. They don't have it available. They don't have teams that are available to do data science on it. What can these companies do to start to think about building that data pipeline?

[0:35:02.7] ZD: Yeah, absolutely. That's a super important point. I think just stepping back for a moment and thinking about [inaudible 0:35:10.0] to the technology side, but I think what you mentioned, it's such an important point, because with the change in industry, with the digital

natives coming and threatening the big enterprise, enterprises can be in a very favorable position if they can unlock their data and assets and really create and join a business echo system.

What we see with a lot of enterprises, they're thinking about their business models that what can I unlock in terms of data insights and assets that I have that gives them me a competitive advantage to a new kind of digital native start up that is really disrupting my business and how can I create a business ecosystem outside of my company, inside of my company, but also outside of my company. So using this data, I can go into partnership or go into a kind of a bigger network of economy and business with syndicates. Unlocking that data is super crucial to really shape your business for future.

Coming to the technology side of it, that's a problem with all organizations, like customer loyalty, brand, information about your customers are the most important assets that you have, and I can tell you the number of times that we talked to a CTO and they would say, "We have no idea how to get to this information." If you have an information about our customers, we probably have to go talk to developer and he would dig around and — Or she, and find the information that we want.

We self-serve kind of data engineering practices. It's the foundation for it. Be able to build APIs to expose the data as the first step. Be able to — Because a lot of the organizations run kind of back jobs that go to different databases and extract the data out. They are kind of working in an off-line mode. They work in a batch mode. So we want to move to more of the stream-based exposure of the data, and not just exposure of the state of the data, exposure of the events as they occur on the system. So it's very, in a none kind of invasive way, how can we change even that monolith that when change a to its data store happening, it can trigger publishing of that as a rich business domain on a data pipeline. Definitely, you will need that data engineering practice and capability brought to the organization to set up the self-serve data engineering pipelines. You need to have the good domain design so that the data that you are exposing, it's abstracted well, it's secure, access to it is secure and really allows the consumers to process it easily.

Yeah, self-served data engineering and using that technology and opening your legacy for extension. We've had these open close principle in object-oriented programming forever. Let's apply that to the architecture. How can I open — Instead of changing my monolith for mortification, how can I open it for extensions? By opening that is, how can I expose the data in a real-time fashion into the data pipeline so that the modern applications of consumers that I have built, they can consume it, they can infer state from it or they can infer insights from it, or just store it in a data lake where different teams can come and make sense of it.

[SPONSOR MESSAGE]

[0:38:59.8] JM: Azure Container Service simplifies the deployment, management and operations of Kubernetes. Eliminate the complicated planning and deployment of fully orchestrated containerized applications with Kubernetes. You can quickly provision clusters to be up and running in no time while simplifying your monitoring and cluster management through auto upgrades and a built-in operations console. Avoid being locked into any one vendor or resource. You can continue to work with the tools that you already know, such as Helm and move applications to any Kubernetes deployment.

Integrate with your choice of container registry, including Azure container registry. Also, quickly and efficiently scale to maximize your resource utilization without having to take your applications off-line. Isolate your application from infrastructure failures and transparently scale the underlying infrastructure to meet growing demands, all while increasing the security, reliability and availability of critical business workloads with Azure.

Check out the Azure Container Service at aka.ms/acs. That's aka.ms/acs, and the link is in the show notes. Thank you to Azure Container Service for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:40:28.2] JM: The data pipeline gets us more value out of those existing assets, and I think the same could be said of an experimentation system. If you have a way to experiment against the way that your architecture has been set, your data, if you can A-B test against different user

sets, you can find out a whole lot about where your company should be headed. I think this can be really valuable for companies that are trying to figure out exactly how to innovate or in what direction they should be innovating. Described the value of an experimentation system and what an experimentation system looks like for a large enterprise.

[0:41:10.5] ZD: Sure. I couldn't agree with you more. Absolutely. It's really interesting. Like a lot of enterprises go through these costly slow and time-consuming process of approving features, because building every feature is so costly. You have to bring a team together. A lot of the enterprise work in a project-base, so you have to bring a team together, you have to justify a business case that this is going to work. This is going to have the return on investment, spend a whole bunch of money, put the feature out there. Sometimes never even measure that it worked or didn't, because you've already spent a lot of money on it and you went through six months, three months, whatever cycle of approval process. In some cases, do monitor what happened, but the feedback cycle is just so late and so costly that changing direction would be very difficult.

So experimentation is the foundation for kind of continuously adjusting the direction we're going. Continuously adjusting where we invest money, where we kind of subtract money and don't invest money. Experimentation could be a different level. So the technology for it could be as simple as, "I am able to reroute a portion of my traffic. I'm able to —" Let's just step back for a minute. Able to, first of all, put a piece of software out there that it might be temporary, it might be permanent, but I can quickly make a change in the existing behavior of system by maybe extending it, adding a new piece, new service, new application and be able to route part of my traffic, a segment of my customers to this service and be able to measure the impact of that towards the goal that I had. Whether it was bottom line, whether it was the number of the customers that clicked on that particular feature or purchased the product, and most importantly, be able to either discard [inaudible 0:43:12.1] of the feature that I had, because it didn't work and I need to iterate over it or double down on it and push it all the way to production and make it kind of production ready.

All these things that I mentioned, they sound very simple, but structure and the infrastructure that you have to put in place to support that is a fundamental change that needs to happen across the board. It needs to happen in your infrastructure, and I've have had firsthand

experience of these that we've been able to — Let's say, one of the unlocking capabilities that you want, you need to have a good set of APIs, APIs that people who want to do that experimentation can easily discover it, can use it. It doesn't need to talk to 10 different teams to figure out what this API does. It's simple, well-documented, it's self-served.

Once you are there, you need to have your delivery infrastructure set up in a way that, again, in a self-served way, I can push my application to production and I don't have to go through a long release cycle. I don't have to wait for the host to be set up. The networking and the routing, again, I need to be able to configure that ideally in a self-served way. So the team that is doing the experiment needs to have all these capabilities in kind of an as a product essentially available to them to be able to go through that experimentation. Need to have a way of then be able to capture the metrics, be able to visualize the metrics and make decisions.

It's really sad, because sometimes we go through a lot of hurdles and do experimentations and we actually find out that little — Very, very small change could make such a big difference in the bottom line, but we don't have the capability to make that a production ready code that can scale and can consume all traffic. Even the experimentation is successful, it gets kind of discarded and it goes back to the production code release train, which defeat the purpose of the fast feedback loop.

[0:45:10.7] JM: I want to shift to talking about some specific technology, some more modern technologies, and starting just with cloud. So if we take the large enterprise, like a toilet seat manufacture. How readily our large enterprises adopting cloud? Because they've got 20 or 30 years of infrastructure experience with their own servers. Are they adopting cloud? Do they do anything with the cloud?

[0:45:39.8] ZD: Yeah. I mean, it's interesting. We have shift kind of in the —At least in the mentality and the approach from enterprises a few years on the track — Years ago, they were questioning why cloud and why us and what we should put on the cloud to saying, "Why not cloud?"

So we see cloud first across a lot of the companies that we worked with, and their approach is, "Let's just start with the new pieces of the software." So all the new capabilities, they start

building kind of the cloud native applications and put it on a cloud. What we see is also a shift from kind of application runtimes, the cloud foundry kind of application runtimes and building tool factors wraps, which is great if you write new applications, but it doesn't really work that well when you have legacy systems that just haven't built that way.

So moving from kind of application runtimes in cloud to container runtimes on cloud, so Kubernetes was a big theme on our radar on this edition of the radar, because the company see more value to work at that level of abstraction so that they can containerize their legacy systems, have more control over the configuration of their environments and dependencies to the legacies. That's another shift that we see, but it's really — It's cloud first is one of the top three kind of initiatives within the enterprises.

[0:47:09.8] JM: So you're seeing clients adopt Kubernetes?

[0:47:12.9] ZD: Yeah. That seems to — Kubernetes seem to have kind of got the right level of abstraction and the right level of control and, obviously, the evangelism behind it and the developer support and the community support. We've seen clients are leaning towards that much more easily, and I mean it's clear where the industry is heading as well with all the kind of the modular architecture that the Kubernetes has and it allows the cloud providers to kind of adopt it and plug into it, plug that into their infrastructure. We see kind of managed Kubernetes. [inaudible 0:47:47.9] just recently in Reivenent, announced their own. We have used JKE for quite a while. Yeah, and Pivotal, I think, they announced a container runtime. So big players and big cloud providers, they provide the managed Kubernetes, which takes away the overhead and the cost of installing and managing the cluster. For enterprises, this seems to — And especially with the legacy system, it seems to kind of give them the right level of control that they need for the older pieces of the software.

[0:48:17.0] JM: And they're typically choosing to go with the managed Kubernetes offering, or do you know if they — What they prefer? Do they roll their own or do they go with the managed Kubernetes managed provider? Too early to tell?

[0:48:30.0] ZD: Yeah, too early to tell, and it really depends on the maturity of the organization. What we encourages is consume first, build next. So invest your energy and effort in money

instead of building bare metal, building features that matter to the mission of your organization. Managed Kubernetes as supposed to hand rolled versions, because there is overhead in managing that.

[0:48:54.9] JM: Of course. There's another topic of discussion that I saw on, I think, the most recent ThoughtWorks radar, which was that a lot of people are using Kafka to re-create some of the patterns of the enterprise service bus, and we've done some shows about this, this event sourcing pattern where you publish all of these events on to Kafka and Kafka becomes your event streaming source of truth. Anybody that's curious about that can check out those episodes. But what was the enterprise service bus pattern? How does that relate to Kafka?

[0:49:31.1] ZD: Sure. I think it's more the organization side of it. So [inaudible 0:49:35.1], your organization reflect your system's architecture and vice versa. So in this case, we see kind of the Kafka becomes your backbone, and rightly so, it becomes a backbone of your integration across your services, but then what happens is you end up with a central team that is managing that, is managing kind of all of the infrastructure around it, the topics and the channels that you need to create and ownership of the kind of the data and the schema of the events and so on. Then, in itself, it's become kind of a monolith and essentially controlled really important part of your architecture.

I guess what we're saying is that that's perfectly fine to use Kafka architecture and streaming on top of it as the backbone of the integration if you have pub subtype integration patterns, but leave the control of the events and the topics and the management side of publishing and consuming of those events to the domains, to the services that own those domains and to the teams that own those domains. Distributed kind of ownership of that infrastructure and giving control to the teams that define those rich events that is a specific to their domain.

[0:51:05.1] JM: Do you see it as an anti-pattern, this idea that we're putting a whole lot of centralization into that central Kafka cluster? We're publishing all these events to the central Kafka cluster. It sounds almost like the shared staging environment that we discussed earlier, that central dependency.

[0:51:22.9] ZD: Yeah, absolutely. It really depends how the distribution and how the communication across the organization works. You may have still essential cluster of Kafka or any other piece of infrastructure. It might be an API gateway that you decide to use configure in a cluster configuration. But how much of the control you expose as APIs as a, again, self-consumed way of changing the configuration to the teams themselves.

I think it's a tension. It's a tension between some piece of infrastructure that needs to have some sort of a central deployment model versus that piece of structure still releasing control in a secure and reliable way through APIs, through automation to the teams themselves and get that kind of architecture right and get that tension right. It's hard.

[0:52:29.1] JM: What about logging and monitoring? If we're talking about these large-scale enterprises, and I think logging and monitoring is maybe another one of those things that the company doesn't have, because it's sort of like the data lake. These companies were established before there was a really good monitoring and logging tools, and then now maybe they want to update their infrastructure to use Splunk or Prometheus or the Elk stack and they don't know what they should adopted. They don't know how they should adopt it. What advice do you have to enterprises that are trying to get better observability out of their stack?

[0:53:09.8] ZD: Yeah. The first one, learn from experience, is that to not build a piece of software without debugability and logging built into it. Actually, some of the big enterprises that I worked with, it's interesting. When they've moved architecture from the existing kind of monolith that they had or layered architecture to microservices, one of the first capabilities that architects think about is how I'm going to get observer in in place. What's sort of libraries or site cars, the new flavor of doing this I need to put in place so that with the first piece of code that goes into production I can get reports out, I can get information about the health of the system and all the different metrics that indicates the health out in a distributed fashion.

My, I guess, advice would be one of the first things you think about as part of your migration is distributed debugability and logging as a way of debugging, because the old way of debugging code is not going to work anymore in a distributed fashion. Really, the technology of choice, anything that lends itself to a modern distributed architecture.

This year, I think it's a year of service mesh, personally, in favor of the shifts that happened, because previously we would have been building service harnesses or [inaudible 0:54:37.9] that implemented the libraries for you getting this — The kind of observability metrics out. Now we can kind of extract that from the application itself into the sidecar and deal with kind of polyglot environments much better with less kind of cost.

So yeah, think about what plugs, what technology plugs well into that distributed logging. Adopt standards, adopt open tracing as a de facto of standard for instrumentation for distributed instrumentation of the code.

Prometheus, it plugs well right now with the kind of implementation of the service mesh that we have, and really I would think have empathy for the developers. So make their life for the developers as easy as possible to both expose information that they need, but also consume. So how — Let's say they they're using Splunk and the developers should be the main consumers of those dashboards knowing and learning about the health of their system. So how can they automate creating those dashboards the same way that they write code, they can have the right configuration and the automated deployment of the congregation for the dashboard, so the both — The production and the consumption and of the states in terms of debugability is important.

[0:55:59.2] JM: Zhamak, thank you for coming on Software Engineering Daily. It's been great talking to you.

[0:56:02.8] ZD: Same here. Thank you so much for having me, Jeff.

[END OF INTERVIEW]

[0:56:09.2] JM: GoCD is an open source continuous delivery server built by ThoughtWorks. GoCD provides continuous delivery out of the box with its built-in pipelines, advanced traceability and value stream visualization. With GoCD you can easily model, orchestrate and visualize complex workflows from end-to-end. GoCD supports modern infrastructure with elastic, on-demand agents and cloud deployments. The plugin ecosystem ensures that GoCD will work well within your own unique environment.

To learn more about GoCD, visit gocd.org/sedaily. That's gocd.org/sedaily. It's free to use and there's professional support and enterprise add-ons that are available from ThoughtWorks. You can find it at gocd.org/sedaily.

If you want to hear more about GoCD and the other projects that ThoughtWorks is working on, listen back to our old episodes with the ThoughtWorks team who have built the product. You can search for ThoughtWorks on Software Engineering Daily.

Thanks to ThoughtWorks for continuing to sponsor Software Engineering Daily and for building GoCD.

[END]