EPISODE 674

[INTRODUCTION]

[00:00:00] JM: Continuous integration and delivery allows teams to move faster by allowing developers to ship code independently of each other. A multistage continuous delivery pipeline might consist of development, staging, testing and production. At each of these stages, a new piece of code undergoes additional tests so that when the code finally makes it to production, the developers can be certain that it won't break the rest of the project.

In a company, the different engineers working on a software project are given the permissions to ship code through a continuous delivery pipeline. Employees at a company have a strong incentive not to push buggy code to production, but what about open source contributors? What does the ideal continuous delivery workflow look like for an open source project?

Abel Wang works on Azure Pipelines, a continuous integration and delivery tool for Microsoft. Azure Pipelines is designed to work with open source projects as well as companies. Abel joins the show to talk about using continuous integration and delivery within open source, and the process of designing a CI/CD tool that can work in any language and any environment.

Full disclosure; Microsoft is a sponsor of Software Engineering Daily.

[SPONSOR MESSAGE]

[00:01:29] JM: Your audience is most likely global. Your customers are everywhere. They're in different countries speaking different languages. For your product or service to reach these new markets, you'll need a reliable solution to localize your digital content quickly. Transifex is a SaaS based localization and translation platform that easily integrates with your Agile development process.

Your software, your websites, your games, apps, video subtitles and more can all be translated with Transifex. You can use Transifex with in-house translation teams, language service providers. You can even crowd source your translations. If you're a developer who is ready to

reach a global audience, check out Transifex. You can visit transifex.com/sedaily and sign up for a free 15-day trial.

With Transifex, source content and translations are automatically synced to a global content repository that's accessible at any time. Translators work on live content within the development cycle, eliminating the need for freezes or batched translations. Whether you are translating a website, a game, a mobile app or even video subtitles, Transifex gives developers the powerful tools needed to manage the software localization process.

Sign up for a free 15 day trial and support Software Engineering Daily by going to transifex.com/sedaily. That's transifex.com/sedaily.

[INTERVIEW]

[00:03:17] JM: Abel Wang, you are a senior cloud developer advocate specializing in DevOps and Azure. Welcome to Software Engineering Daily.

[00:03:26] AW: Hey, thank you so much.

[00:03:27] JM: I want to talk to you about some various subjects under the purview of DevOps and continuous integration. I'd like your perspective on the general landscape of continuous integration. So I've talked to a lot of different companies who are at various stages in their continuous integration rollout, kind of their test coverage. Where is it typical enterprise in its CI coverage? How many enterprises are out there, or does the typical enterprise have continuous integration deployed?

[00:04:02] AW: So that's a really good question, and the answer like everything our industry is kind of it depends, right? So if you look at the very basic, most enterprises have some type of build setup. So if you check in the code, guess what it's coming to kick off a build? So hurrah for that. We're at that point where almost everybody has that and that's a not a problem.

I'm so old I remember back in my day when I first started writing code, that wasn't even a thing, right? When you check in code, it didn't automatically just kick off a build. Then it really starts varying pretty broadly from one company to a next depending on how far along they are on their

DevOps journey. Even if you look at Microsoft itself, between from one group to another group, we have vastly different capabilities and how much we've adopted DevOps practices.

So the industry is kind of like that as well, but thankfully most people have build. Testing is still big problem, or it's not even a problem. We know what the fix is. It's just people haven't done them, like write massive amounts of unit tests that you can put into your CI/CD pipelines. But for the most part, builds are in place. The deployment, it starts getting a little bit sketchy, or testing is sketchy as well.

[00:05:14] JM: So with build itself is kind of useful even if you just are building your application into what like a staging environment or a testing environment where you can maybe do manual tests over it before you promote it to production.

[00:05:28] AW: Oh yeah. Builds are super, super important. That's like the very first step. I mean, I check in code and if the bill does nothing more than just compile or not, that right there can give me immediate feedback as a developer. I check in my code, wait a couple of minutes for the build to kickoff and finish and then I immediately know, "Did I break the build or not?" Once again, old as dirt, right?

I remember back in my day, we would check code in and we would only nightly builds, which means I wouldn't know that I broke the official build until the next morning, when disaster strikes. But now, I can get that immediately feedback. Even if your build, all it does is just compile, hurrah for that. That's a win. That's like step one.

Next step is to take those bits and let's go ahead and run all of our unit tests, because that can once again let me know [inaudible 00:06:20] with that unit test, first line of defense and our best line of defense for quality. Those can happen just right before you deploy your bits anywhere. This just right on the build machine. Once that's done, it'd be great to pick those bits up, deploy it into some type of environment where you can start running other types of tests.

[00:06:39] JM: So one challenge of getting this CI DevOps journey going that I have seen is these tests. The question of how you get test coverage over your legacy spaghetti, like your legacy monolith spaghetti code where the original person that ran it that wrote that code 10

years ago is no longer with the company, and nobody knows how to work this code. Is it possible to get test coverage over these legacy balls of mud?

[00:07:12] AW: That's a very painful subject, and the short answer is sort of, kind of, but – So we also have massive amounts of legacy code at Microsoft as you can. Just piles and piles of code, and anything that does not have unit tests we consider that legacy code. How do we maintain quality for that? Nobody is going to want to and no management team is going to give you the time and money to go back to working code and retrofit unit tests into everything, right? It's impossible.

Especially considering if it is all spaghetti code, in order for you to write unit tests, you have to make sure your code is testable, right? So we kind of have a moving forward policy, which is, "All right. So the old stuff we can't really write good unit test around. But anything new that we do, we better write really good unit tests around and make sure our code is testable. Anytime we have to go back and fix a bug, we better be able to refactor whatever we need to touch, tweak it, so that we can write those unit tests around it to give us that type of coverage. But otherwise, yeah, now one is going to give us the time to go back and re-architect a working application to make it testable.

[00:08:26] JM: Which is too bad, because I think it's really hard to get to that world of DevOps dreams without having test – I mean, I guess you can do it, you can at least do it with the newer applications that are written at a legacy enterprise. I mean, newer legacy enterprises, they are always writing new applications so at least they could stand up of greenfield application. In that greenfield, they can at least set a shining example for the rest of the enterprise to maybe work towards.

[00:08:58] AW: Oh, yeah. Absolutely. Even legacy apps, you can have the moving forward policy, which is you don't get the benefit of having all of the unit test and the coverage in the pipelines. So you're still going to have bottlenecks when it comes to testing. But moving forward, as you start building more and more functionality, that can really, really jumpstart your project.

We ran into this problem pretty extensively at Microsoft. For a long time, the way that we would maintain quality was through end-to-end functional test. But the problem with that is in the DevOps world that we live, where we're constantly trying to push new bits out into production,

we just don't have the time to really run these full end-to-end functional tests. That could take weeks, maybe months, even a year or so depending on how big your software is.

So because of that, we started moving away from functional testing and we moved to – Well, first we tried doing automated UI testing, and that – Well, we thought we'd found the magic bullet and we spent a lot of money writing automated UI testing and our thought process says, "Well, great. Now we can check a new code, deploy it into an environment, run these automated UI test. We might have to wait a little bit for them to run, but when they're done, hurrah for that. Now we know if the quality is still good or not."

But what we found out is trying to do automated UI testing at the range that we do, it became very difficult. The automated UI tests were so freaking fragile. There may have never been one time where I've seen all the automated UI test [inaudible 00:10:34]. So some of them would break when you run it, same tests, same bits, exact same environment, and they would magically pass next time and other automated UI tests would fail.

Because they were so fragile, and that's just part of the automated technology, right? That's just part of what you get when you write automated UI test. So we made a concerted effort to move away from functional testing and automated UI testing to unit testing. So that has become insanely important for us. Like we were saying earlier, you can't really go back in time to fix everything, but you can start moving forward. Anything new that you do, let's do this so that we can start iterating at a faster rate.

[00:11:16] JM: I want to talk to about open source as well, because I have a good understanding of how DevOps and continuous integration, how these things work at enterprises. How they work in startups? I have less of an understanding of how it works in open source. How well do the practices of DevOps, and site reliability engineering, and continues integration, do those map to open source?

[00:11:45] AW: They map, and they actually map surprisingly well. There are some challenges with open source, right? For instance, a typical open source project, it might need to support multiple platforms, right? It might need to support Linux. It might need to support Windows. It might need to support even on Macs. What that means is now you're going to have to have three different build systems, maybe three different billings, three different configurations, and

that's one thing that actual pipelines does really well. We will give you build agents on Mac, Windows, and Linus. So all three platforms we support just right out of the box.

Another interesting challenge with open source projects is there is often a lot of chatter going on, a lot of check-ins that are happening, a lot of pull requests, which means a lot of builds get kicked off as well. So because of that price constraints, it might need to happen.

I think one last thing that I just thought of is that one problem with trying to manage builds with open source projects is, because a lot of times people are working on these at odd hours and stuff, it takes time for people to a pull request and to review the code. Because of that, it might not. It depends on the project. It might not get merging to master as fast as in other places, and because of that, the code can drift and then you start getting merge more complex. So those are some type of challenges that you get with open source projects.

[SPONSOR MESSAGE]

[00:13:23] JM: This episode of Software Engineering Daily is sponsored by Datadog. Datadog integrates seamlessly with container like Docker and Kubernetes so you can monitor your entire container cluster in real-time. See across all of your servers, containers, apps and services in one place with powerful visualizations, sophisticated alerting, distributed tracing and APM. Now, Datadog has application performance monitoring for Java.

Start monitoring your micro-services today with a free trial. As a bonus, Datadog will send you a free t-shirt. You can get both of those things by going to softwareengineeringdaily.com/datadog. That’s softwareengineeringdaily.com/datadog.

Thank you, Datadog.

[INTERVIEW CONTINUED]

[00:14:18] JM: So I have the an open source project that I've been working on for a while, Software Daily. So we've got like a website and some mobile apps that people can use to access our old content. It's like kind of a mobile friendly way that people access the content and

various people contribute to the open source project. We have been trying to get continuous integration up and running for a while.

So our stack is – It's like we've got a node app that sort of the backend API. We've got android app. We've got iOS app. We've got a web frontend. So I've seen these challenges of trying to get continuous integration for open source code up and running, and it's been significant, mostly because you have to have this authority. Somebody has to be in charge of approving stuff that gets integrated and pushed into production.

If you're at a company, if you're at a company, then by the very nature of the fact that you're at the company, your incentives are aligned with that company. But here we have a production application that's open source. The database itself is not open source – Well, the database, the contents of the database are not known, but other than that, it's an open source consumer product.

So it's somebody could walk in and push malicious code if we have this beautiful continuous integration experience. So how do you overcome that problem of like if you really need – I mean, I guess you could put previous stages in and have an approval stage at the end. I don't know. What prescription would you give to me? What should my continuous integration set up look like?

[00:16:05] AW: So there should be somebody that owns the project, right? They should be the ones that either approve or not approve a pull request. Part of the pull request process is before – Not anybody can just go and merge to master, right? So before they merge into master, they have to go through a pull request where whoever's checking that code is going to get looked at, make sure it looks good, make sure it's following all the best practices. Maybe you should even build in some checks and balances within the build process itself, and then only if the pull request passes, that's when the code merges into master and then the bits gets built and flow all the way out into production. So there should be some gatekeeping before the code even gets merged into master.

[00:16:47] JM: I feel like if this kind of project is actually the exception for open source, because most open source projects are more like infrastructure, like node.js, or Kafka, or something. What kinds of open source projects should have continuous integration?

[00:17:03] AW: I think all of them need to. So it depends on what you mean by continuous integration, right? Either you're talking about continuous integration continuous deployment, where something actually gets deployed into a production environment, or does it mean you build something and the end result is maybe a zip file that's put on to a share somewhere.

[00:17:21] JM: I suppose I should talk more generally, like what should the pipeline of acceptance into an open source project look like?

[00:17:30] AW: Should definitely go through a pull request. So before I can even merge into master, I kick off a pull request. The owners of the master branch, they will review my code. Make sure everything looks good, and if they think everything looks good, that code gets merged into master. Once it gets merged into master, that should kickoff a build that compiles the bits. It should run all the unit tests associated with the changes. If everything looks good, then it should flow through the release pipeline, whatever release means.

[00:17:58] JM: So what are some of the problems that open source projects encounter when they're trying to have a pipeline such as the ones that you've just described?

[00:18:08] AW: The biggest one that I see is that the pull request merges, sometimes they can be difficult. The main reason for that is that the code is drifting really fast, or I shouldn't say drifting, but their code is changing really fast. So if whoever owns the master branch, they're not completing pull requests fast enough. A lot of these pull request – The code is going to be out of date. They need to remerge from master to make sure everything looks good. I mean, just so all the code is caught up. So because of that, merge conflicts can happen. But that's really the big one.

[00:18:41] JM: So what else do we need out of a continuous integration, or continuous deployment, a DevOps style set up for an open source project? For example, do we need some kind of observability? Do we need some monitoring stuff to have an open source deployment system work properly?

[00:18:58] AW: So there's almost no distinction between open source and if you're going to do this from the enterprise. DevOps best practices should be followed regardless of if it's open

source or not. But all those things matter, right? Security matters. Quality definitely matters. Usually, the order that it follows is, first, we just want to make sure things build. Then after things build, we want to make sure that our bits can be deployed somewhere in some automated fashion, and whatever that deployment means. It doesn't matter. It could be zip file on a shared drive, or it could be deployed into web servers behind somewhere up in Azure, wherever.

Then after that, people start realizing, “Oh, we need to add quality into our pipelines, because we’re just pushing bugs out at a really rapid rate. We can't keep up with testing. What are we going to do?” So then you start learning about letting writing your code into testable manner so that you can add unit test that test everything and you put that into your pipeline.

At some point in time, then people start realizing, “Ooh! Database changes. How are we going to do that?” So we need to do database DevOps and you need to add those into your pipeline as well. Then people start realizing security. That's kind of a big thing. Security with open source projects, that’s a huge thing.

So then maybe we need to add things like code scanning to see if we’re using the most up-to-date open source packages and things like that. So you do like security scanning within your pipeline as well and you start slowly building in all the stuff that you need.

When people first start, I don't tell them, “Just try to do everything all at once,” because that’s just way too much stuff to do. I just say, “Okay, let's pick one thing and start from there,” and then we slowly build upon that until we have a pipeline that's very, very functional.

[00:20:43] JM: How do you get people to write tests in open source projects? Is there a problem of incentives there where people just want to write new code, they don’t want to write the tests?

[00:20:52] AW: That's a problem everywhere.

[00:20:54] JM: Oh, yeah. That's true.

[00:20:55] AW: Nobody really wants to write tests, but it's so freaking important, right? I'm involved in a couple of open source projects where we don't allow a pull request unless it also includes the test for it. It's just simple. So if you don't have the test for it, guess what? Your pull request will never get merged into master. It's that important. Quality doesn't come for free, right? Quality comes at a price, and the price that we pay is the time that it takes us to write code that's testable and to write those unit tests as well.

[00:21:24] JM: So you've worked on a variety of DevOps tools of Microsoft. What kinds of stuff are you working on today?

[00:21:31] AW: S today, I'm working on the Azure DevOps Suite, and specifically Azure Pipelines. So this is a suite of DevOps tools that literally you can use the suite of tools to do everything that you need to take an idea and turn that idea into a working piece of software in the hands of your end users for any language targeting any platform.

So we're talking everything, right? From work item tracking, to build and release, to testing. Just everything that you can imagine, that's what the Azure DevOps Suite can do. Azure Pipeline, that is a specific product just for build and release.

[00:22:13] JM: There are a lot of continuous integration tools that are on the market. Why are there so many of those?

[00:22:21] AW: Because there has been a hole for a really long time for good tools, good build tools, good release tools. Some of them have been – I don't want to say they're bad, but there are some better than others, right? Because there've been so many – There's been a hole in this for so long, a lot of tools have come up to try to fix this problem that we have.

[00:22:43] JM: What's the differentiator? Because I've seen like a bunch of these different tools, and some of them work with certain workflows. Some of them are highly opinionated. Some of them are less opinionated, and it seems like one of those areas of software where it's not a winner take all market, there's a ton of different products that are successful, but they all make different trade-offs, different subjective trade-offs. What are the subjective trade-offs? What are the differentiators for Azure pipelines?

[00:23:12] AW: Azure Pipelines from the beginning, it has been designed to be from the ground up totally customizable so that you can make it do whatever you need to for any language targeting any platform. I say this over and over and over and over again, because it seems like people don't believe me until I sit down in front of them and literally show them. This is not just for the Microsoft stack. It doesn't just work with .net. It doesn't just work with Java and Linux. It can work with anything. So if you want to do Java on Linus, guess what? You can do that easily. If you want to build mobile apps on Macs, Guess what? You can do that easily. If you have in your build and release pipeline, you want to deploy this behind your firewall, guess what? You can do that really easily with Azure Pipelines, right? So it's a couple of things, it's the flexibility and it's the power of Azure Pipelines that is unprecedented.

[00:24:05] JM: So when you think about that broad array of different platforms and languages that people could potentially want to run on their system, if you're designing a continuous delivery tool, you need to be able to run every single one of those configurations on a container, on a VM, on bare metal. What makes it hard to build a continuous delivery tool that runs in a – Of different environments that can run a variety of different languages?

[00:24:37] AW: I think that's the problem right there, right? Because it's a variety of different platforms and a variety of different languages where you need to do vastly different things. How would you build a system that solves all of that? It's easy to build like a CI/CD tool for a specific language targeting a specific platform that you're deploying to. It's much tougher to build a generic system that can do everything on everything and still make it easily customizable, highly configurable, etc., etc.

[00:25:09] JM: So what's an example of some continuous delivery environments that are dramatically different that if you're developing a tool that works on any environment, kind of would illustrate like how dramatically different systems are?

[00:25:22] AW: So if I'm deploying to, let's say a cloud, like Azure, that's going to be – And I'm using something like Azure Pipelines, that's going to be vastly different than if I'm trying to deploy behind your firewall on to bare metal. Those are two very different things.

So because of that, well, Azure Pipelines actually literally doesn't care. It can do either one. Some of the steps that it does is going to be a little bit different. So what Azure pipelines does is

it's basically just a task runner, right? It will do one task after another, after another, after another, and these tasks can be written in node or they can be written in PowerShell.

So now what that means is these tasks can run on a Windows environment, or they can run on a Linux, or even a Mac environment. If this is node or if it's PowerShell, what that means is anything that you can do from the command line, you can get this build and release system to do as well.

So, out-of-the-box, Azure Pipelines is going to come with hundreds of tasks that lets you do all sorts of stuff. If what you need to do doesn't exist out-of-the-box, it's not a big deal, because if you jump into the marketplace – I think our partners have now created over 700 building build and release tasks that you can just download, drag and drop onto your screen and just start using them.

So that's how we accomplish, how are we able to deploy into all these different systems. We also provide for you build agents that run on Mac, Windows and Linux. So in theory, you can have a full CI/CD pipeline with no equipment whatsoever of your own. You don't have to have a build machine. You don't have to have deployment machines. We can take care of all that for you. Of course, unless you deploy behind your firewall, then you can't use our host editions.

[00:27:12] JM: What are some of those things in the marketplace where you would want to have tasks? Could you give more example of tasks that you would want to run in your continuous delivery pipeline?

[00:27:25] AW: Sure. One that we talked about earlier is like code scanning, right? If I'm using a lot of open source projects, one of the things that I worry about is the security of those open source projects. So I would want to scan my code, figure out what open source projects I'm using, libraries that I'm using, what version they're at and what vulnerabilities do these specific libraries have and am I touching those vulnerabilities?

If I could have a task that scans my code and spits me out a report that does that, that would be freaking awesome. So I can jump to the marketplace, download in WhiteSource Bolt, and viola, that's it. I drag it into pipeline, say, "I want this to run after I download my source code," and I point it at the directory and hurrah for that. I'm done.

[00:28:14] JM: You. You could also do stuff like chaos testing. We've done a number shows on chaos testing, and I guess that some people treat chaos testing is what something they would just do on a drill occasionally. But the ideal world is that you have chaos testing in your continuous delivery pipeline.

[00:28:31] AW: It should be part of your pipeline. Everything should just be part of your pipeline. So anytime I check in code, guess what? It's going to run through all those steps, verify everything and hopefully in as automated fashion as makes sense.

[00:28:43] JM: Yeah. So I guess you can also write stuff to – For example, delete a testing database and then instantiate a new testing database so that you have a new testing run entirely from scratch.

[00:28:57] AW: Yup, exactly.

[00:28:59] JM: What about open source? Why does this fit well into open source workflows?

[00:29:04] AW: So open source workflows, they need to have build and release pipelines just like in the enterprise. There is really no difference. They really need to adopt DevOps best practices just like the enterprise need to adopt DevOps best practices. Why do they need to adopt DevOps best practices? Because the faster that they can iterate and provide higher quality, guess what? Everybody wins when that happens.

Open source does have a lot of challenges, right? One of the biggest ones is the sheer amount of builds that potentially – Builds and releases that potentially could get ticked off in a project. In Azure Pipelines, we're really catered to open source projects. So if you're an open source project, we will literally give you 10 parallel pipelines completely for free that you can just use. If you're an open source project, guess what? You get those for free. If you have more than 10 parallel pipelines, that's a massive project you're on. But even if you do have more than 10, we will still give you more pipelines. You just have to call us to ask.

[00:30:02] JM: So these different pipelines, how do you see different pipelines partitioned? Is it like per project, like any given – I guess it would be per repo in a different project? Is it like a pipeline per repos at a typical deployment strategy?

[00:30:19] AW: Typically, yes, but not all the time. It depends on how people have – It depends on how they have configured their repos and what have you. It sure would be nice if every repo was just a deployable unit. Unfortunately, not everyone has done that. I am on a project right now where our repo holds like about seven deployable units. So that one repo now has seven parallel pipelines associated to it. So it depends. It depends on how people set things up.

[00:30:50] JM: Now, I feel like open source development is something that's in the very early days of tooling. One thing that makes me say that is I did a show a while ago about the open source community around Linux, and it sounded like – I mean, first of all, the state-of-the-art is GiotHu, basically, for large scale open source management and collaboration. For the largest project in the world, which is Linux, GitHub kind of doesn't work for them, because it's too big of a project, and we'll probably see more and more projects that are as big as Linux in the future.

What are the other gaps in tooling that you think exists around the open source software management space?

[00:31:40] AW: So the big problem is with this, is just how big these repos potentially get, the sheer number of lines of code, or the sheer number of files that they have. When you have a massive amount of files, git starts performing badly. There's just no way to go around it, right? It turns really, really, really slow.

But the funny thing is that at Microsoft we actually came up with a new file system, a new virtual file system for git specifically to handle these massive project files, right? So, internally, we have already switched over to git for our source control system for just about everything, including Windows. The code base for Windows is massive. It dwarfs Linux. I'm even embarrassed to say how large it is. It's just so massive, and using our virtual file system, it becomes usable. So that used to be a big hole. Not so much anymore, and I know this is going to be implemented in GitHub soon as well.

[00:32:40] JM: Wow! Okay. So this is like the thing where – Why don't you just explain? What is a virtual file system?

[00:32:46] AW: So git keeps up a copy of absolutely everything right on your hard drive. So because of that, if you have hundreds of thousands of files, if you have gigs and gigs and gigs and gigs and gigs of source code and stuff, the way that it tries to track all the changes, it needs to traverse all the files, it needs to touch things. That takes a really, really long time.

So if we have a virtual file system, the short of it is we don't have all the files. We just have markers in place, because of that, we can move around and do things much faster. So that's kind of waving my hands a lot and simplifying things tremendously.

But in a nutshell, that's what we're doing, so that now we can still use git, still use all of our git commands, but we're able to work with massive file systems of source code and everything still performs in a healthy fashion.

[00:33:34] JM: No. I think that's a pretty good description. So I think if you have the – Let's say you had the virtual file system for the Linux code base on your computer, if you just double-click the file, that file might take a second to materialize, because it's actually pulling down the file over the network, and the only thing you have on your computer is actually some metadata and the title of the file.

[00:33:59] AW: Right.

[00:33:59] JM: So what about collaboration, and community, and discussions, and stuffs? For example, if a build fails, you want to be able to maybe tag somebody in response to that build failing. It seems like there's a lot of opportunities for better ways of collaborating across CI/CD workloads, or across your various code interactions. Do you see opportunities for better collaboration?

[00:34:28] AW: Absolutely. If you've looked at not just inside of Azure Pipelines, although it is in Azure Pipelines, when we do a pull request – So we literally use our own tools. We dog food our own tools in-house. So there is a ton of collaboration, and where that happens the most is through our pull request, right? So every time we do a pull request, the first thing that it does is it

will actually take the pull request and it kickoff a build. Because it kicks off a build, if there are problems, we'll be able to see it immediately and it surfaces up much quicker than trying to build it after you merge.

During that pull request, there is a lot of chatter going back and forth between whoever's doing the code review and whoever owns the code and also with the build system as well, because the build system, not only can it compile everything, but if there are problems where it breaks, it will actually put that into the pull request as well.

If you bring in other vendors' tools, there are tools that will literally go in there, scan your code, and in your pull request, it will automatically add their suggestions in as well. So not only can he have really good interactions with humans, but you can have really nice interactions with the automated system as well.

[00:35:45] JM: There's also the component of the IDE. So VS Code is probably the most popular IDE today. What's the ideal format for – How should an IDE integrate with a continuous delivery pipeline?

[00:36:00] AW: You know what? The IDE really should worry about is just checking the code in, because that's the switch that does absolutely everything. I check my code in, and guess what? Everything, the build automatically happens, tests automatically get run, bits automatically get picked up and just sent through the department pipeline. That's all the IDE needs to do.

[00:36:20] JM: So what were some of the other features of Azure Pipelines that were typically difficult to implement?

[00:36:26] AW: The biggest one is let's support every single platform, not just the Microsoft stack, and I say this over and over again, because everyone thinks, "Oh, Azure Pipelines. It must be a Microsoft tool. It must only work for Azure." No. It works for everything. So that was the big thing. Let's get this to work for everything in any platform whatsoever. Make this as open as possible.

[00:36:48] JM: What was the process of getting to all that coverage? You must've had a test across all the VM's, and containers, and cloud providers, and bare-metal formats, and stuff. How did you check off all the boxes for all those different runtimes?

[00:37:03] AW: So it gets – There is no good answer to this one, because it gets pretty hairy. Especially with a lot of open-source projects too, not just what we were doing. At some point, you do need to do integration tests, and at some point you do need to test whatever platform that you potentially could run on.

We chose Windows Linux and Mac as, “Okay. Our agents will run on these supported, and anything else not supported.” So we kind of hedged our bet that. Because we’re a tasked-based system that basically is just you can run commands from the command line, it's a little bit easier to test and say like, “I have this fabulous GUI that now needs to run on all these different platforms and test it to make sure that everything is still working correctly.” That's a much more difficult thing.

But, yes. We do have to spin up a lot of instances and test on a lot of different things to make sure everything is still working.

[SPONSOR MESSAGE]

[00:37:57] JM: Data holds an incredible amount of value, but extracting value from data is difficult, especially for non-technical non-analyst users. As software builders, you have a unique opportunity to unlock the value of data to users through your product or service. Jaspersoft offers embeddable reports, dashboards and data visualizations that developers love.

Give users intuitive access to data in the ideal place for them to take action within your application. To check out Jaspersoft, go to softwareengineering daily.com/jaspersoft and find out how easy it is to embed reporting and analytics into your application.

Jaspersoft is great for admin dashboards or for helping your customers make data-driven decisions within your product, because it is not just your company that wants analytics. It's also your customers that want analytics.

Jaspersoft is made by TIBCO, the software company with two decades of experience in analytics and event processing. In a recent episode of Software Engineering Daily , we discussed the past, present and future of TIBCO as well as the development of Jaspersoft. In the meantime, check out Jaspersoft for yourself at softwareengineeringdaily.com/jaspersoft.

Thanks to Jaspersoft from being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[00:39:29] JM: Fascinating. In terms of the product development, what were some of the lessons that you took away from interacting with different companies that were using Azure Pipelines, or I guess if you were just using it internally during the development process? What were the takeaways as people started to use it and how things changed once they used it?

[00:39:51] AW: That's a good question. All sorts of things have changed once people start using it and it touches real hands and doing real task loads. One change that has happened is there has been a trend in our industry to do pipeline as code, where you can define your pipeline using like a YAML file for instance, or maybe a JSON file or whatever, but you define your pipeline as code. You check it into repo right alongside your code. For the longest time I thought that was just simply silly, ridiculous. Why would I need this? I mean, this is – I guess, in theory, this seemed kind of cool to me. But from a practical standpoint, I didn't know why I needed it. Until we started using Azure Pipelines, and I realized that one of the early things that I was doing was I couldn't get – Well, the code required a change in the build, but I couldn't go through my pull request, because my pull request is going to queue up a build, but the build still didn't have my build changes yet. But I couldn’t make the changes to the build pipeline, because if I did that, someone else's pull request is going to break as well.

But if I had pipeline as code, if the pipeline was defined in my repo, in my branch, then that wouldn’t be a problem. That was a big evolution for us as well to move. Yes, we do – Our build engine is a task-based build engine, but it can be described as a YAML file or it can be defined using the visual editing tool as well. So that was a huge shift that we made.

[00:41:19] JM: Now, earlier in the show, you were talking about how many teams at Microsoft have been adopting various DevOps practices over the last couple of years. What have they found to be most difficult in that process?

[00:41:37] AW: If you ask different groups, they will tell you different things. Everybody is at different parts of that journey. For us, specifically, it was how do you deal with the bottleneck of testing and still deploy a quality product? Because it's easy to say, "Well, just write unit tests," but I can write a million unit tests that doesn't really test anything, right? It totally depends on not only do you have to write unit tests, you have to write good tests. But to really shift from functional testing to unit testing and still be able to maintain quality, that's kind of a huge thing. That's a massively huge thing.

[00:42:12] JM: Functional testing, meaning end-to-end test.

[00:42:15] AW: Yeah.

[00:42:15] JM: So the problem with functional end-to-end tests is what? It's just to black boxy?

[00:42:21] AW: It just takes too long. It just takes forever. So if I'm to do functional testing, that could take weeks, maybe months. Easily it can, right? If we're trying to deploy every two weeks, how are you going to get your functional testing done in time? We couldn't keep up, and that was one of our bottlenecks. So then it was figuring out, "Okay, if that's the bottleneck, then we need to shift the left somehow so that we can incorporate this as part of the pipeline." That's when we stumbled upon using unit testing doing really, really good unit tests.

[00:42:55] JM: When you're thinking about the addressable market for a new continuous delivery tool and you look out at the different enterprises out there, is it possible to convince other companies to – Or companies that have already adopted a CI/CD tool to switch to a newer continuous delivery tool, or do you think the market of people who have who have not even adopted a continuous delivery tool yet is big enough that you can could just kind of go after greenfield opportunities?

[00:43:26] AW: Greenfield opportunities, that's easy, right? If you do a side-by-side comparison between actual pipelines versus any other CI/CD tool, I feel very confident that Azure Pipelines will just be a shining star. So that greenfield does fantastic.

For stuff that's already has a CI/CD system in place, that's a much tougher win, and it really depends on the pain point. I don't even necessarily recommend people switch if they have a CI/CD system that works for them. I'm not speaking as a Microsoft personnel. I'm speaking just as a DevOps person in general. If you have a system that works for that? Do that. But if you're noticing that there are holes in your CI/CD system or that it's painful to get some certain things done, then maybe it's time to look at other tools that can do things easier for you. Once again, it's not something where you have to shift everything over all at once either. You can move things piecemeal, little bit by little bit, project by project as well.

[00:44:27] JM: When you talk to various companies, because you've been involved in DevOps products for a while, where did they prioritize CI/CD? You have CI/CD. You have containerization. You have monitoring. You have micro-services, perhaps distributed tracing, perhaps service mesh. Where in the sequence of moving to DevOps do people prioritize CI/CD?

[00:44:57] AW: CICD is like the backbone of DevOps. You have to have some type of build system that packages everything up so it's ready to be deployed. You have to have some type of automated system that will pick up those bits and deploy them somehow. What those things are, what should actually deploy into? Whether you're creating a container image, or whether you're deploying an Azure function, that can vary depending on the architecture of your app. But you have to have CI/CD no matter what technologies you're picking.

[00:45:29] JM: What's the ideal process of a project being built and turned into some kind of artifact and then deployed gradually, or perhaps like A/B tested? Do you have an ideal sequence of stages that it would follow in an ideal pipeline?

[00:45:47] AW: It really depends like everything else in our industry, right? But, yes. The very first thing it needs to do is let's get the latest code from source control. Let's compile everything. Let's run all of our tests, all of our unit test, the tests that can be run. On top of that, let's scan our code for security. These are just important things that have to happen.

Then if all that passes, then let's go ahead and pick those bits up and start deploying it, whatever that means. Whether it's just deploying the app, deploying the database, deploying the mobile pieces, you can do all that in parallel if you need to, and each one of those technologies, there's different things that you need to do for your deployments.

[00:46:26] JM: To wrap up, since we've mostly been talking about CI/CD in the context of open source projects, one thing I wonder about is why there are not more open source SaaS companies. Because if your SaaS company and you have – Like if your proprietary advantages, you've got a database of data, for example, like if you're data company and your advantage is your database. It generally makes sense to open source your code. Why not? The only disadvantage would be somebody could potentially stand up the same SaaS and then start with a database from scratch. But the advantage of course is that you get more people contributing to your project. Do you think that we'll seem more open source SaaS companies in the future?

[00:47:12] AW: I do think so. We have been ingrained in this idea that our source code has to be too super top secret forever and we have to monetize everything. But even looking at a company like Microsoft, where we traditionally did not play in the open source world, we are open sourcing more and more and more stuff. I think I agree with you. It makes sense that open source for SaaS, that we're going to see more of that.

[00:47:40] JM: Fascinating. Well, Abel Wang, thank you for coming on Software Engineering Daily. It's been great talking to you.

[00:47:44] AW: Hey, thank you so much for having me.

[END OF INTERVIEW]

[00:47:49] 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 offline. 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.

To learn more about Azure Container Service and other Azure services as well as receive a free e-book by Brendan Burns, go to aka.ms/sedaily. Brendan Burns is the creator of Kubernetes and his e-book is about some of the distributed systems design lessons that he has learned building Kubernetes. That e-book is available at aka.ms/sedaily.

[END]

