EPISODE 1406

[INTRODUCTION]

[00:00:01] KP: Happy New Year software engineering daily listeners. We've got a great show about build tools lined up for today. But before that two quick announcements. First, we're looking for part time hosts for the show. If that's something interesting to you, or you're just curious about the details, please head over to softwareengineeringdaily.com/applying-to-host. There, we've got a short write up that just tells you a little bit about what we're looking for, how to apply, and what the commitment would be. I can tell you firsthand, it's a joy to interview software professionals. If that's something on your career agenda, check us out.

Similarly, if your company or the company you work at might be a great fit for our audience and you'd like to consider sponsoring Software Engineering Daily, we're now opening up our 2022 sponsorships. Please email sponsors@softwareengineeringdaily.com to get further details.

All right onto the show. Writing software is an absolute joy. Getting software to build is a chore, thus build systems emerged as a solution to automate this chore. At some point, software engineers either use or hear legends about make and make files. While perhaps being the historically best-known tool, a great deal of thought has gone into approaches to build systems since 1976 when make was introduced.

In today's episode, we focus on the build system called Pants. Pants is a scalable software build system. It can help you support all the modern challenges of a build system such as dependency resolution, testing linting, and packaging just to name a few. In this episode, I interview Benjy Weinberger, co-founder at Toolchain and contributor to the open source project Pants.

[INTERVIEW]

[00:01:48] KP: Benjy, welcome to software engineering daily.

[00:01:52] BW: Thanks for having me.

[00:01:54] KP: Well, before we get into the core section of the interview, can you start us off with just a little background and how you got into software in general?

[00:02:02] BW: Absolutely. So, I was born in the UK, but I grew up in Israel. And the tech industry is a big deal in Israel. When I went to university, the computer science was a very popular major that a lot of people were interested, and it has always been fairly obvious to me that was sort of science in general and mathematics and computer science in particular, were things that I would love and it turns out I was right.

[00:02:32] KP: And where'd you get your start professionally in the software industry?

[00:02:36] BW: So, I got my first job out of college at Check Point, which was in Tel Aviv, which was in the late ‘90s, which was the very prominent Israeli tech company at the time, and really cut my teeth there on low level network engineering and had a great time and learned a great deal.

[00:02:57] KP: So, you've been through a couple of corporate experiences. What made you make the jump from I guess, worker to founder?

[00:03:05] BW: Really good question. I had worked at so many companies, I moved to the US in the early 2000s. I ended up working at Google, which was a phenomenal experience. And Google was the first time I noticed a strong investment and effort on the part of a company in developer productivity. It was really a joyful experience working there and having all this incredible tooling. Obviously, everything is a work in progress. Not everything was incredible. But there was this focus on and devotion to developer productivity. And when I left Google, I went to other smaller companies, I discovered that the rest of the industry, that was not the norm. It was not obvious that a company would invest significantly in tools and developer productivity and best practices and in the tools to support those.

So, I started getting really interested in that space, and eventually realized this is what I want to do. And I have strong ideas and strong opinions about what needs to happen in that space. And no one is working on those, and realized that if I want to make a dent in it, then I need a

company that is the vehicle to do that. So, I'm one of those people that necessarily always dreamed of becoming an entrepreneur or starting a company. But it is very much my dream to be starting the company that I did start and working for Toolchain.

[00:04:31] KP: So, Toolchain is the company. Can you tell me about Pants and what the relationship is between Toolchain and Pants?

[00:04:37] BW: Absolutely. So, Toolchain is a company focused, as I mentioned, on developer tooling and on giving developers a scalable, fast, stable build and workflow experience. And our technology is built around Pants which is an open source project that is based on about the last 10 years of the efforts of myself and other people in that area. So, the Pants project actually came out of an internal project that John, my co-founder and I worked on originally internally at Twitter, forward for Twitter's uses. And it eventually got spun out of there because I had gone on to work at Foursquare and Foursquare has similar set of build needs. Both companies were using Scala and we're having real trouble scaling the standard Scala tool chain.

So, hands got spun out so that we could collaborate on it, those two companies could collaborate on it. And we learned a lot from that building that system. What we currently referred to as Pants is a new system related only by name and by shared experience with the old one because, whereas the old one was really designed around scholar use at those two companies, circa the early 2010s. Pants V2, as we unimaginatively call, it was rebuilt from the ground up to serve a much wider set of use cases at companies of all sizes and teams of all sizes with various different languages and different use cases and different verticals. And so, that's what Pants is. It is currently the repository of our open source efforts. It is this relatively new system that we launched last year, based on lessons we learned from several years of working on that holder one of the same name.

[00:06:21] KP: So, I'm thinking about a company I was at where we were a big enough engineering team, that we had a dedicated change management engineer, how would a person like this take up Pants as a tool and effectively helped get my code into production?

[00:06:34] BW: So, it's very common that, when we talk to companies that are using Pants or looking to adopt it and helping them adopt it, we're talking to one person behind that person and

maybe tens or hundreds of engineers that that person is supporting. So, you ask the right question in terms of that dedicated engineer is usually the person doing all the work on behalf of their coworkers. So, the first thing you typically do is look at the documentation. We have a very detailed documentation site at pantsbuild.org, which contains a lot of getting started guides and explanations of how you use the system. And the other thing you do is hop on Slack. Because this is very much a community effort, and we are very enthusiastic about helping with adoption at various companies, for multiple reasons.

One is, obviously, we don't want people to drop off adoption because they feel like they weren't getting support they needed. But also, the more we learn about use cases, real world use cases and attempts to apply our technology to them, the better we can make that technology and the more suitable for wide use, we can make it. So, we really love hearing from people on Slack. There's a link in one of the early pages on that documentation site, I mentioned, there's links to a Slack workspace and you hop on there and you ask some questions.

But generally, the adaption, one thing that we really focused on in the design that was very different from earlier systems from the first version of Pants and very different from other systems that were design for a single company’s use cases is we needed the system to be really easy to adopt. Because adoption is not just something that happens once at that one company, but happens over and over hundreds and thousands of times at all these different teams and all these different companies. And so, unlike other systems, there’s a huge amount of automated assistance in adoption, so scripts that will set things up for you, the system itself knows how to generate its metadata, and it doesn't need very much metadata, because instead of relying on a lot of elaborate manually provided information, it uses a static analysis and looks at your code to derive almost all of that information.

So, one obvious example is you typically do not have to tell the system very much about your dependencies, because it can look at your input statements in your code and figure it out from there. And in another example, you don't have to restructure your code or modularize it in some way to meet the expectations of the system. A system can work with dependency tangles and spaghetti code and all sort of unpleasant, but sometimes inevitable facts of real-world code base architecture.

[00:09:25] KP: So, I've bumped into a handful of problems in my time with Python dependencies, and always – well, eventually found a way to solve it with some finagling and requirements dot txt. Can I still use that? Or do I still need that? Can you kind of automate that process away for me?

[00:09:42] BW: So, we try to work with existing tooling and existing configuration as much as possible. So, if you have a requirements dot txt, we will use it. But one thing Pants does because it has this very, it does static analysis and has this intimate knowledge of your dependencies, is that it can figure out sort of just what these necessary set of dependencies for a given operation is. So, instead of you having to, through fine tuning, splitting your requirements dot txt and sort of very carefully managing them, in order to make sure that a set of requirements is present, for a given operation, say running some tests or packaging your code for deployment. So, instead of having to do that manually, the system does it for you. Your requirements dot txt can now function as just a piece of metadata that says, “Here's a bunch of external requirements that my code may depend on.” And Pants will, at runtime, pluck out from that set, just the set that you truly need for this operation. So, it does a lot of that untangling for you.

[00:10:55] KP: Gotcha. The last time I had to set up a bigger size Python project was before I learned about Pants, and at the time, I was able to find somebody at a nice boilerplate somewhere on GitHub that I cloned and it was a somewhat opinionated way of doing all the linting and testing and that sort of stuff. I realized that Pants, that's only a subset of the features. But for someone who knows that boilerplate experience, what's a similar onboarding to get a project stood up?

[00:11:21] BW: So, are you talking about a new project, or you have some existing code that you want to adopt Pants into?

[00:11:28] KP: Well, both are interesting cases, let's do what I presume is the easier one first, the Greenfield project.

[00:11:33] BW: So, with a greenfield project, typically, you can structure it however you want. Pants is very agnostic to how you lay out your code. For example, we prefer to have unit tests

live alongside the code that they test. So foo.py is tested by foo_test.py in the same directory, and many tools don't support that, because they would just naturally bundle those tests in with your production code, they couldn't tell the difference. But Pants is smart about knowing well, this is a test, it's not part of your production code.

So, we support that layout. But other people like to have parallel trees of sort of production code, and then a parallel tree containing test code and it supports that as well. I'd say the initial setup of pants in a greenfield project, or even for that matter in an existing project is the one thing you do have to tell Pants about is what your source routes are. Source routes which is the term we use for the points in your codebase to represent the roots of the package hierarchy. So, for many Python, smaller Python code bases, that is actually just the root of the repository, right? So, your import paths start from the root of the repository. But if in larger repos, you might have your Python code under src/python. So, you'd have to configure the system to say, “Well, these are my source routes.”

In general, there are some sensible defaults there. So actually, the examples I mentioned are all covered by the default. So, if your code lives under source Python, or source py, or is directly in the route, we will handle those cases. But if you have something unusual, that's your preference, or if your tests live somewhere else, you might have to add some configuration there. And that's really the only main thing you might have to say which versions of Python you want to use. So, you can constrain, sort of paths will go out to your system, find all the Pythons and pick one that matches the constraints in your configuration.

You might want to say, my repo, just don't try on running this on anything other than Python 3.7, or Python 3.7 and Python 3.8, or whatever your choices are. Beyond that, you run Pants itself, there's a script that you run one time that creates an empty config file, and then you run some stuff that like, adds some basic configuration. You check that into your repo, and then you run a Pants goal. So, commands in Pants are called goals, or something like Pants test, or Pants lint, those are called goals. And we have one called tailor, which is the only Pants related plan, we have allowed ourselves in the codebase. And tailor does kind of what it's the reason we kept it is it's a good pun, it will create what's called build files for you. You may be familiar with build files from other build systems, most build systems have some way of just pointing to cut sort of little metadata files that point to code and say, this is a Python library, these are some tests, this is a

binary, these are resource files that can be loaded, using package resources or runtime, that sort of thing. Those are generated for you again by inspecting the code. And so, onboarding in either case is basically you add a few lines to a config file, you run pants tailor, you check in the results, and you're pretty much off to the races at that point.

[00:14:46] KP: So, it's very appealing to me as a developer that I've got the freedom to structure my project however I want. Does that present any challenges for you to be able to package it that there seems to be a very open way I can configure it?

[00:14:59] BW: No. So again, the core of everything Pants does is the static analysis, and the very fine-grained understanding of the dependencies in your repo. So, we analyze those down to the file level, which is why you can intermingle tests, safely intermingle tests and production code. We need to do this for many reasons. So, one example is caching. One of the ways that Pants is fast is that it caches all the work that it does on a very fine-grained level. So, it's not sort of caching entire directories like a CI system would do. But it's caching individual tiny work units, even a simple build might have thousands of individual work units cached. And to do that correctly, we have to know about your dependencies.

Similarly, concurrency pants can say you have eight cores on your laptop, Pants can use all eight to run eight tests at the same time. Doing so requires knowing that two different pieces of work don't depend on each other, and that requires knowing about code dependencies. So, once we have all this fine-grained knowledge of your dependencies, then figuring out what needs to be packaged in order to run a binary. So, perhaps we'll detect that some file that contains, for example, the if name equals main stanza that looks like an entry point in Python. It will create a, what we call a binary target for that, and it can analyze all the dependencies of that thing. When you request to package it, it will package just those dependencies. The only complications if there are dependencies that could not be inferred from this static analysis. So, for example, loading code or resources at runtime by name, in which case, you just have to manually add a dependency into one of those build files and check it into the repo. But generally, the information that Pants needs to do its job also helps it do things like packaged correctly.

[00:16:55] KP: And what are some examples of common workflows that people will use Pants to build?

[00:17:00] BW: That's a great question. So, we see a lot of use cases. Again, so listeners will notice that we're talking about Python a lot, and that is because when we first launched this Pants V2 as we're unimaginatively calling it, we focused on Python, because Python has become so popular so quickly. And there aren't great solutions out there for medium, let alone large-scale Python development, that sort of standard Python tooling kind of assumes implicitly that there's just your entire repos building one binary.

So, we have a lot of Python use cases, data science is a big one. Various DevOps workflows are another. And also, just web applications, such as, building flask, and Django apps. So, we have a lot of workflows around. For example, Django, microservices is a really interesting one. The standard Django tooling assumes it's supports things like multiple databases, multiple binaries, but it's not really quite built for that without additional tooling around it and Pants can provide that tooling. So, you can package a what we call a PEX file. So, PEX is a format that we've developed that it's short for Python executable, and essentially bundles all of your first party Python code, along with all of its third-party requirements into a single executable zip file that you can just deploy to a server and run, and it expands itself finds local suitable Python interpreter and then runs itself.

For example, one of those can run a production Django binary, via the sort of unicorn entry point, G unicorn entry point, and you can build many of those in out of a single repo and they can share common settings. And there are all sorts of interesting practices around that. And so, one of the things that enables we use this internally at our company and other companies use Pants for this as well is Django microservices.

[00:19:05] KP: So, how do you know the entry point for the, I guess, top level of the micro services? Like whatever runs flask? How does the PEX file know where to start?

[00:19:14] BW: So, there'll be a little bit of config in the build file, you would say. This is the code that's bundled and here’s the one line of config for it that is different than the other binaries and all that config contains is when sage unicorn starts up, here is the server file that it loads.

[00:19:33] KP: Gotcha. So, I love a vision of situation which I can push my software when I merge a pull request to master, workflow could get triggered, the PEX file could be built, shipped to an environment, all that kind of stuff. How far can I go with Pants alone? And do I need other tooling to really complete the delivery?

[00:19:54] BW: Well, one of the features that we have added recently, thanks to a one of our contributors is Docker support. So, you can actually have pants build a Docker image. But the single build command will first build any binaries that need to be embedded into that Docker image. So, the Docker image has a dependency on say, for example, one of these PEX files. And so, when you say build the Docker image paths will first understand that it needs to ensure that the underlying binary is built. So, you can run a single Pants command that will do everything on the Python side and then embed the resulting Python binary into a Docker image.

In fact, because of this PEX format, one thing you can have with your many services is the same underlying Docker image that then deployment images can differ only by this one single different PEX file. So instead of for example, needing multiple images which builds different virtual environments, out of different requirements dot txt, here everything is packaged into this one single file that gets executed. So, you can get as far as Pants building a Docker image that contains all the binaries. And you don't need to do anything manual there.

Another example is, you can build cloud functions like GCP Cloud Functions, or AWS lambdas, that run your Python code. And so, Pants will build all the way to the zip file that you need to then deploy. There is not much support currently for deploying. There is some for publishing. So, you can, for example, push a Docker image. But right now, Pants kind of mostly stops where there is something ready to be published or published, but not deployed. Because deployment is a, very large, very interesting, very elaborate area in its own right. And there are a ton of CI/CD platforms out there that handle that very well. But again, Pants is generally a workflow, a developer workflow system. It is designed for making all of the iterative steps that developers perform while they're developing, fast and stable. And then also, when those types of steps like running tests are recapitulated in CI, Pants is very good at making that robust and fast as well.

So, if someone wanted the deployment capabilities built into Pants, they could absolutely do that, because it is extensible. Pants is fairly, it understands a sludge set of these goals that I mentioned. So, things like type checking, compiling, resolving third party dependencies, running tests, linting, formatting, packaging, you can add your own, and people do.

[00:22:46] KP: So, the static code analysis that's done in the background allows you to build these, you get insight about the versioning and dependencies. But it seems like there's also an opportunity here to maybe give me some feedback as a developer. Are there any insights you can surface about my code base?

[00:23:03] BW: Yes. We support linting, and formatting sort of out of the box. Pants sort of understands that, for example, linters, don't modify the source code. So, you can run all the linters in parallel, but formatters do, you have to run them sequentially. It sort of has some intrinsic understanding of the semantics of these different operations. And so, you can add custom linters that will do things like inspect your code in various custom ways.

One area that we're looking at right now, we have an open ticket for and we're very interested in supporting it as our most requested feature, is validating dependencies. So, we know all about your dependencies. But right now, we essentially let you depend on anything, that we don't say this import statement is invalid, because there are rules in your code base that don't allow this package to depend on this package. So, we will pretty soon be adding that support, because as I mentioned, it has been asked for a lot. That I think, especially in larger code bases provides an extra level of sanity. It helps prevent getting into circular dependencies, and it helps prevent spaghetti code, and it helps keep make sure that the way we think about this problem is the data science stack problem, because it's sort of common manifestation of it is you just wanted to bring in one utility function from somewhere in the code base, and now your micro service depends on the entire data science stack. That sort of thing that we would like to have the tool much more proactively prevent, and so that is an upcoming feature that we will presumably add.

[00:24:49] KP: What are your options there? I mean, at some point, I do have to import NumPy and Pandas, right?

[00:24:53] BW: You do, if your code needs to, but there are many cases where it doesn't, but people do because convenient or because it gets pulled in accidentally because, you know, it depended on some utils library and that utils library happens to also have some data science related utilities in it. And so, those cause that library to pull in NumPy. So, you can end up depending on things you didn't intend to. I think we've all had that experience. So, just maintaining some sort of sanity, and rule checking around dependencies is an area of codebase hygiene and how a tool like pants can help you with Kobe's hygiene that we're very interested in exploring.

[00:25:38] KP: Interesting. Yeah, I mean, unlike language like Go, which is strict about it, I've seen one of the sloppiest behaviors of Python. Engineers who worked for me is just massive copy and paste of everything they've ever had to import at the top of a notebook or file. So, being able to wrap that up, I think maybe the reason people are sloppy like that is it's not always apparent what the problem is, at least until later on the development cycle. Could you highlight some of the reasons we don't want to continue to practice like that?

[00:26:05] BW: Well, there are kind of the hard problems around code base architecture and code base management are reasoning about changes and reasoning about dependencies. And where those two get really hard is the combination of the two of them. So, how changes flow through your dependencies is the hardest problem, it is the thing that causes – is the hardest problems to do manually, it is the hardest problems to do automatically, and it is what makes code bases brittle and hard to update. Because the greater the surface area of your code, the harder it is to change it and to reason about how your changes affect the consumers of your code.

So, keeping the surface areas small, for a given amount of volume, if you allow me this geometric analogy, is really, really important for growing the code base in and continuing to be able to work in it in an effective way. The sort of allowing incidental dependencies, those become part of the surface area of your code, and then you have to reason about how those are affecting the consumers of your code until the end of time. And so, keeping your code as tight as possible, really helps with scaling it and still being able to up work effectively and rapidly in that code base.

[00:27:28] KP: Do you have any access to some sort of metrics or statistics about developer productivity? I know I've spent a lot of time waiting for things to install and stuff like that, where, I'm sure the caching might have saved me many hours of work hours, right? Is there any way you can get some sort of grasp on like a percentage improvement or something like that that's quantifiable about the benefits of adoption?

[00:27:52] BW: Yes. So, we've talked a lot about pants in the open source system, Toolchain also offers SaaS products that greatly enhance the effectiveness of Pants. One of those is this analytics and inspection UI that lets you look at the history of your bills. So, if you think about how bills and workflows work today, you run things in a terminal, and the history of them gets lost in the scroll back. And there is no wider context, right? So, when builds gets slow, or there are bottlenecks and problems, you're kind of left to your intuition to figure out what went wrong, or what is happening over time. With our UI that we're calling build sense, you can see the entire history, not just of your own bills, but of your organization's bills and that includes both desktop builds and CI builds.

So, in this UI, you can compare in very fine detail, performance of builds over time, so you can look at it and to see where things are getting slower, where bottlenecks are, where things are breaking, if you know, a particular test is failing, inconsistently, you can see context in sort of where it fails and where it succeeds. Now, we still have a lot of work to do on this, of course, there's a huge amount of interesting work to be done in automatically surfacing a lot of this stuff. But I think we have been interacting with our build systems on the command line since make, which was from the ‘70s. We haven't fundamentally changed that paradigm. But we have browsers now, right? We have these brilliant viewports into data that allow us to view his sort of time series data in a structured and sensible way. And since builds are so complex, as I mentioned, even simple ones might have thousands of steps in them, having a structured UI that can show you sort of surface useful information, I think is really a game changer and is really, really helpful in finding the problems with your build and problems in your codebase.

[00:30:02] KP: When I think about companies I've worked at or consulted for, the format is something like this would end up being run by like a DevOps group. Maybe the developers would be looking at it to get some insight about the code, but it'd be more of a build team at a

larger company. Is that a common model? Or do you see other structures and ways people utilize the tools?

[00:30:22] BW: I think in the smaller companies, it's individual engineers, I think in the larger companies, it is this sort of build team or engineering effectiveness team. They have different names in different places. One of the things we want to do is to be that team for a large number of companies that don't want to or cannot invest in that area, or as much as they should. I mean, if you are some company that has some expertise in whatever brilliant product that you're building, you should focus on that, and you should be able to have a really good build experience without having to invest too much of your own time and resources into it.

We'd like to help move companies away from having to constitute and put significant resources into creating their own teams with all and those teams have to gain this expertise, and rather, have it sort of move shift left more on to the individual developers who are now empowered through tooling to achieve a much greater productivity gains on their own.

[00:31:30] KP: Makes sense. So, then there's the SaaS offering, are there services and solutions that are at the enterprise level on top of that, that Toolchain offers?

[00:31:38] BW: Yes, so I'd mentioned build sense, which is the UI components. But the other aspect is the remote caching and remote execution components. Where, as I mentioned, Pants relies heavily on caching and concurrency to provide significant speed ups. And that makes sense, because when you think about work, there's really two ways of speeding things up. There's doing less work, and there's doing more work at the same time. Those are really the only two ways. And so caching means doing less work because it's already been done. And concurrency means do more work at the same time, because it is not interdependent.

Now, if you're running Pants on your laptop, as I mentioned, maybe you get eight-way concurrency or on a CI machine, and maybe only two way, because very often, CI machines are somewhat underpowered, and you have your local cache, which will make sure you don't repeat work that has been done before on your machine. But with our SaaS product, you can wire your builds, both in CI and on desktop, into a shared cache. Now, you're in a situation where if work has been done before, on any machine, by any user, it can be shared. So, the benefits increase

dramatically in terms of caching. And similarly, for remote execution, which is something we are still working on is not ready for use yet, instead of the sort of two way to eight-way concurrency that you might get on the cores on your machine, and even that is somewhat limited because there still may be contention on resources like the file system, you might get concurrency on many dozens, hundreds potentially, of cores, running in our build cloud. And so, the SaaS product will take the caching and concurrency abilities of pants, but really turbocharge them for much, much greater performance.

[00:33:35] KP: We touched briefly on the serverless use case. I'd love to revisit it. I know that many people trying to Deploy serverlessly will hit problems, like the fact that it's such a bare bones environment, or that there's no file system or even the fact that the zip file has to be under a certain limit. Are the extra challenges that pants has to overcome in order to help make serverless delivery?

[00:33:57] BW: We don't specifically have solutions for those except that manually getting the say, to take AWS, AWS lambda as the canonical example here, getting the format exactly right. When you try to upload a file is finicky, and accidentally uploading way too much data is way too much code is a common problem. And so, what Pants really solves that is it knows exactly what needs to be in that bundle and no more.

So, for example, you have a large requirements dot txt, and you cannot upload that entire the requirement all of those requirements into your cloud function because it's just too big. Now you'd have to manually start slicing and dicing it to figure out what exactly needs to go in that cloud function. Pants will figure that out for you, which means it's very practical to have it manage many dozens of different cloud functions and each one will be slim. Each one will contain just what it exactly what it needs to contain. You're guaranteed they will all be in the right format and the cloud function will run correctly the first time, because you didn't have to manually sort of package something in exactly the right way.

Obviously, with Cloud Functions, there are challenges. A lot of the challenges I've encountered with them have been at runtime around keeping them warm. There are some well-known challenges there. But when it comes to packaging the function and creating the metadata

around it, having this done consistently for you, in a way, which is guaranteed to be the most minimal, is very helpful.

[00:35:32] KP: Absolutely. When you see new customers coming in at Toolchain, I'm curious if you see any patterns in the level of maturity, the applications at? Is this something they're adopting early on in development, or something that's being adopted, perhaps a bit too late in an emergency,

[00:35:49] BW: We see a bit of both, as you'd imagine. Usually, I wouldn't say an emergency, but we usually see people come in when there's a need when they've noticed a need. Traditionally, with built systems, that point has always been a bit too late. Because at that point, your code base may not be amenable to some of these tools. And so, one of the big design points we had when we built this second version of Pants, was to make adoption relatively easy. And so, we had to make it so that you could adopt even if your code base was a little gnarly, because you hadn't been using a tool like this before.

So, we do see a lot of people coming in where the code base has grown, the requirements of the requirements dot txt has grown to a point where it's actually hard to manage, where suddenly they're looking at, previously, maybe there was one big monolithic binary, many companies and many systems start out with just one server binary that does everything, and then they start splitting it up into smaller services. Maybe not fully micro services, but multiple services. And then that's when people start to realize that the existing tooling doesn't really cover that use case very well and Pants does. So, we see a lot of that of like, companies and teams who've gotten past the initial, maybe first year or two of the code base and now they're looking ahead to what should we be using for the next five years.

[00:37:18] KP: Is the goal Of pants to become the de facto standard for building Python Software?

[00:37:25] BW: Yes, absolutely. For Python, and for other languages. So, we are now, as I mentioned, we started with Python, because the need there was the most dire, but we've had users really ask for, when will you have support for Java, when will you have support for Go. So, we are now in the process of launching support for Go, Java and Scala. Because these are the

languages that have been among the most requested by existing users. Because commonly, repos contain multiple languages.

One angle there is that Python is obviously very commonly used for data science applications. And Scala has some great uses in that world as well. And so, we see Python and Scala used together in the same repo. And we want to bring a lot of those benefits of ease of adoption, ergonomic tooling, requiring very minimal boilerplate and working with your code base as it is, and working with the concepts with your users already familiar. We wanted to bring those on to other languages. So ultimately, we would like to be the right choice for all the companies, in all languages that can't or don't want to invest building up a team to develop their own internal tooling. If you're Microsoft, or Amazon or Google or Facebook, sure, you've probably got 30 people working on this problem internally, and building a system that is perfectly designed for just your repo. But if you're not those companies, we want to be there for you.

[00:38:57] KP: This seems like an area that a lot of companies spin their wheels trying to reinvent something on. Do you often find that people are pleased to finally arrive at pants when they discover it? Or what's the typical path?

[00:39:10] BW: We do find that people are pleased to. And one underappreciated aspect of that journey is, even when you have custom needs and custom build steps, and we all know, most companies of any appreciable size do, you still want to build them around a system that takes care of things like caching and concurrency for you. So, Pants has this very robust plugin API. And when you write your custom workflow steps against that API, you get the benefits of caching and concurrency that just sort of fall out of the API. You don't have to do anything special to make that happen. The API just sort of takes care of the safety that those features require.

So, even companies that are doing fairly sophisticated things on their own in using scripts come to realize that, “Oh, it's better if we do this within a framework like Pants where we it's still a lot of custom logic. But the custom logic is embedded in this larger system.” So, we don't have to worry about where the inputs into that custom logic come from. We don't have to worry about where the outputs flow to. We don't have to worry about how do we speed this up through caching and concurrency. The system takes care of all of this for you. So, we do see even

people who then have to invest in building their own plugins saying, “Well, this is way better than having to write all of my logic, including all of that infrastructural side from scratch.”

[00:40:39] KP: Makes sense. Where can listeners learn more about the project?

[00:40:45] BW: So, the best place to go is pantsbuild.org. That's our documentation site. There's a lot of information there about what the system is, how you can use it, and sort of how to get started. It also points to some very useful resources. Among those are a bunch of example repo. So, some little toy repos that show here's how, here's like a very basic Hello World Python example. Here's a very basic Django example. Here's a very basic Go example, things like that.

And then also, perhaps most importantly, is there are links there to our Slack workspace, where we really love to hear from users. As I mentioned, people should not be shy about hopping on there and expressing interest, because we want to help people adopt where we can. And if for whatever reason we can't, or there are needs that past doesn't yet meet, we need to find out about them. Because the purpose of the system, as I mentioned, if you have a use case, then Pants wants to solve it that I can't imagine a situation where we say, no, that's just not something we're interested in. If you know, if people have some substantial use case, then we want to help solve it, whether it's something we develop in core pants ourselves, if it's something that we think many organizations would care about, or we guide you to, as I mentioned, building a plugin that is perhaps more custom for your organization, while letting the Pants framework take care of so much of the infrastructure part of that.

We've had cases where people have done that second thing and then ended up upstreaming it into core Pants because it turned out to be useful enough for that. So, just to give one example, someone has been working on support for generating Sphinx documentation. And we're now inquiring with them about upstreaming that support which is currently in that custom plugin into core Pants because it seems like that would be useful for many people. And so, where where things are potentially generalizable. We're very open to upstreaming them into the main system.

[00:42:50] KP: Very cool. I look forward to following the project. Benjy, thanks again for coming on Software Engineering Daily.

[00:42:56] BW: Thanks for having me. It's been wonderful.

[END]