EPISODE 1141

[INTRODUCTION]

[00:00:00] JM: Pachyderm is a system for data version control. Code has been version controlled for many years, but not data. In previous episodes with Joe Doliner, we explored the evolution of Pachyderm. In today's show, we talk about the state of the company in 2020 as well as Pachyderm Hub, an end-to-end machine learning and data lineage product.

[INTERVIEW]

[00:00:27] JM: Joe Doliner, welcome back to the show.

[00:00:30] JD: Thank you, Jeff. It's great to be here. This is I think my third tour of duty on Software Engineering Daily.

[00:00:35] JM: It is the third appearance. So the last episode, we talked about the updated version of Pachyderm. What's new? What has occurred since we last spoke?

[00:00:43] JD: So a bunch is new. The main thing that's new is that we spent, I guess is the last two years since that interview, building our SaaS offering, which is called Pach Hub, and that just launched a few weeks ago, and that's basically the same open source product that the people are using and loving, taking online. So you don't have to deal with any Kubernetes infrastructure, any object storage or anything like that. You just click a button on Pack Hub and you get a fully functioning Pachydern instance with all the enterprise features installed, and it's basically completely pay as you go. So you don't have to have any really upfront cost, you just sort of pay for the infrastructure as you use it.

[00:01:23] JM: For people who don't know what Pachyderm even is, version control for data, what else? Explain more about what Pachyderm actually is.

[00:01:31] JD: Yeah. Pachyderm is basically a complete big data stack. And so it's a full data storage system. It's in a distributed file system that we called PFS, the Pachyderm file system.

And that file system has all the normal features you'd expect from a distributed file system. It can store gigantic pieces of data. It can store lots of little files and stuff like that. What it has that most other distributed file systems don't have is that it's version controlled, as you just mentioned. And so that means that similar to Git, you commit data to it rather than writing data to it. And when you commit data to it, that turns into an immutable commit that you can reference using a hash. And so that allows you to do all of the things that you can do with code version control. It allows you to sort of travel in time and see where did this data come from. How has it changed overtime?

And the other really cool thing that it does is it has a processing layer on top of it that's entirely based on Kubernetes. And so that allows you to not just store your data, but process your data in a distributed fashion. And so to that, you create a Docker container that contains whatever code you want to put in there. People use it for a lot of machine learning stuff. They use it for image processing. They use it for genetic stuff. The sky is really the limit, because it's a Docker container. It's whatever open source cool tools you want to put in there. And Pachyderm will take that container. We will deploy that as a distributed pipeline that will just keep running as new data comes in. Keep your data up-to-date. And it tracks the lineage of that data as it flows through the system. So when new data comes out, you can always ask it, "Where did this data come from? What is the provenance of this data as we call it?" And that will just point you to other commits in the system to tell you what went into making it.

[00:03:15] JM: The version controlled data system, it's something that's been talked about for a long time. There's been a demand for this for a pretty long time. So given that, my sense is that this is just incredibly hard to build. What are some difficult engineering problems that you've had to overcome?

[00:03:29] JD: Yeah. It is incredibly hard to build. I mean, for anybody who's ever used Git, you've probably gotten yourself eventually in this situation where you accidentally commit a file that you didn't mean to. A lot of times, it's like a binary. Sometimes people accidentally commit VM images and stuff like that. Once you do that, that file is in the history. And so anytime anybody goes to check out that, the repo in the future, they're going to get that file unless you go back and modify history.

And so that only works for Git, because code is normally fairly small. It's normally pretty reasonable to keep it on your laptop. Even really big software projects don't have 100 gigabytes of code. And so the really challenge is basically scaling these algorithms up. The algorithms that allow you to do the snapshotting and the version control to just arbitrarily large datasets. And to give you sort of an idea of the flavor of the problems, imagine somebody wants to store a 20 terabyte file within Pachyderm, right? We store everything in object store on the base layer. And object stores can't hold a 20 terabyte file. So we're going to have to split this file up into some number of chunks. And so that's fine. We can do that. you can chunk up the file however you want. And then the real difficulty comes when somebody wants to modify just a little bit of this file, right? Because you could say, "Okay. They modify a little bit." Well, that's just one new chunk, right? So if we say we have like megabyte chunks, then that's only one megabyte of new storage.

But what if they make one of those chunks a little bit bigger? Right? And so now the chunks are slightly differently aligned. And so now all of the chunks are new. And so all of a sudden somebody inserts an extra byte into one of these chunks and we're storing another 10 terabytes of data just to store an extra copy of the file. And so you have to figure out how to have a system that is very able to handle all of these little changes in data and store a somewhat sane amount of data, basically store a diff, and being able to quickly access that data when you need it, right? So you can store everything as diffs. But then when somebody accesses a random commit, you have to reapply all of those diffs to get the actual result. And that can take a long time too. So it's a pretty deeply challenging computer science problem.

[00:05:47] JM: Are you reusing any kind of off-the-shelf file system that you've forked or taken advantage of?

[00:05:54] JD: Not really. No. We, actually, in very early versions of Pachderm, that was how we implemented it. We built it on top of Btrfs, which if you're familiar with Btrfs, OverlayFS. OverlayFS is a little bit different. But Btrfs and CFS are the two main ones that are just a standard file system like you can run on your machine, but it supports the snapshotting. And that worked well in a lot of ways. What was really difficult about that was running it within a containerized environment, which is the environment that we needed to get Pachyderm to deploy in. We had to do all of these sort of crazy hacks to get the right kernel modules and the right libraries installed within the container so we could access the file system. And it was bit of a permissioning nightmare. It was a bit of a versioning nightmare. So we eventually sort of decided that it made more sense to go the other way and have our base storage layer just be a dumb object store, which doesn't give us any of these versioning primitives to work with. But it's really, really good to work with in containers, because it's just an endpoint that you talk to and it's just there. And if your container goes down, the object store doesn't. And so you don't really have to worry about a lot of the distributed file system aspects of it.

And then on top of that, we built the versioning by basically taking the algorithms from Git and sort of rewriting those to work on top of object storage. And Git is actually doing a somewhat similar thing. When we first started on this, we sort of had the idea of maybe we should use Git. And it doesn't really work for large files except they've added – They had just added at the time, this was like six years ago, Git large file storage. And that was sort of a promising feature that didn't seem like it was quite there yet. There's been a lot of development on that with Git sort of doing a lot of the same things that we've been doing to get this to work on object stores and get this to work like for large files that need to be broken up and stuff like that. So we have a lot of prior art to derive from here, but we haven't actually been able to use any of the off-the-shelf libraries to do it.

[00:07:54] JM: There are some different primitives in the file system. You've got repo, commit, branches that are very similar to Git. Can you tell me more about the primitives that you've chosen and what you built upon them?

[00:08:06] JD: Yeah, absolutely. So it's very, very intentional that they are similar to Git. And you should – The goal is that somebody who is familiar with Git should basically be able to come in and just understand what these things mean. So a repo is exactly you what you think it is. It's sort of the top-level object that contains all of the other objects. The thing that's a little different pachyderm is that, normally – And this is actually I guess a bit of debate/flame war within the software development community of whether you want to have a mono repo or whether you want to have a bunch of small repos that are sort of special purpose. Pachyderm pushes you pretty hard toward the second model, and that's not really a comment on the validity of either model and software development. That's just what works in Pachyderm, is that you have a different repo for each of your datasets. And then pipelines get their own repo that corresponds to them that’s just their output repo and you consume things via repos.

And so those repose contain commits within them exact same meaning. As in Git, more or less algorithmically the same implementation is in Git as well. And where things get a little bit different is with branches, because in Git, a branch is basically like a label that's applied to a commit so you can have a human readable name for it rather than having to remember your commit hashes when you commit to something, which obviously would get really unwieldy really quickly. But they’re also sort of an object that you can push to, right? So you can push to a remote branch, and that takes your version of the branch and sort of fast forwards that, the remote version up. The inverse of that of course is pulling where you pull stuff back down.

What's different Pachyderm is that the branches actually are the things that define how the computation runs. So we like to draw a link between the master branch of an input repo and the master branch of an output repo that a pipeline is writing to. And the system sort of records that and knows that when one master branch moves, another one should move, and all of the downstream ones should move. And that sort of lays out the DAG, the directed acyclic graphs, for computation using these branches. So they're the only ones that have a little bit more smarts in them than the Git equivalent.

[00:10:15] JM: What is a pachyderm pipeline?

[00:10:17] JD: A Pachyderm pipeline is basically a Kubernetes service that gets deployed, and it contains your container with your user code that you want to run. Then it also contains information about what this code needs to run. So for example, if it's a machine learning pipeline, it might need a GPU. It needs a certain amount of memory and things like that. And we basically see it as a standing order to process data when it comes into a certain location.

So for example, if you were a company that has a problem with fraud and you want to build a new fraud model whenever new data becomes available, you would have a repo that contains a dump of your database and whatever other data needs to go into this fraud model. But dump of the database is normally sort of the first step in that. And you would have a pipeline that takes that dump and spins up some number of containers and runs it and outputs a fraud model that you can then push to your site, or you can deployed a model serving framework, or whatever you want to do. And you would write this up as just a little JSON spec that we call pipeline spec that you send into Pachyderm.

And once you do that, we basically take everything from there. Anytime new data comes in, we will orchestrate the containers to do it. We will get the data into them. We will make sure that the code runs successfully and retry it if need be. And then when they're done, we'll take that result and we'll do whatever else you want with it. And that can be sort of egress and get to some other location, or sometimes that can be spinning up more pipeline.

So pipelines are sort of chainable so that you can have like your training pipeline and then your pipeline that does scoring of the model or your pipeline that does some inferences based on things that you care about, stuff like that. It's the core sort of computation primitive in Pachyderm.

[00:12:05] JM: Can you explain more? What's the difference between building data pipelines from scratch in pachyderm versus importing a bunch of ad hoc scripts?

[00:12:14] JD: The big difference is the orchestration. In terms of what the actual code looks like, it's oftentimes exactly identical. And the code that you write on your laptop that just reads data off of disk and writes it to somewhere else on disk, that's the same code that's going to work on pachyderm. We expose the data to you right on disk in a special – Just like a folder called /PFS.

What's different is that all of the orchestration around, it gets completely automated within Pachyderm, and all of the lineage tracking just happens automatically. So this allows you to just say at a very high-level like, "When data comes in at this spot, I want all of this complicated processing to happen downstream and data come out of the spot." Then all you had to do really was say where you wanted the data come from and what the code was supposed to look like. And pachyderm will handle scaling up to more instances if you need it, retrying when things fail. When a node fails, it reschedules those workers and everything like that. That's really the difference, is like we're handling all of the orchestration of the light compute, the actual like pods and processes that need to be spun up to process this data. And we're were handling the orchestration of the data. We're making sure that the right data gets into your code so that you can process it.

[00:13:27] JM: Tell me more about Pack Hub.

[00:13:29] JD: Yeah, absolutely. So like I said, Pack Hub is the hosted version of this. And so it's really the same product experience. You're using Pachyderm. You're using the same tools. It's just that there is no worrying about how much, "Do I need more compute? Do I need to spin up more compute?" It automatically scales up and down the computer and we just charge you for what you've used.

It was a really interesting product to build, because Kubernetes sort of came along five years ago and "democratized" the ability to do this. It gave us sort of just a standard language to speak and a standard primitive for deploying workloads on EC2 instances, or whatever VM instances you want to use.

And so now that it's mature, we were able to build Pack Hub as a system that basically when someone spins up a Pachyderm cluster, the first thing we do is go and talk to GKE, which is Google's Kubernetes engine, and we spin up a new Kubernetes instance there. Users get their own completely clean Kubernetes environment. And then we deploy a bunch of things on it, and we're able to standardize that fairly well because, again, Kubernetes is this common language where you just put a manifest on there. In this case, it's written in YAML, and that just spins up all the right things for you.

And so that's basically what the product does. It's kind of – When you phrase it like that, it sounds kind of simple. But of course, there's all of the things that go into there of like how do you authenticate users? How do you share this cluster with other users? As the name suggests, it's meant as a tool for a team to collaborate on .So like once you spin up this cluster, you can then invite other people to join Pachyderm Hub and join your cluster, and we'll image all of the user information, the authentication and authorization and stuff like that. And you can all just collaborate on your data infrastructure and your data science pipelines that are exposed on Pack Hub to the team.

[00:15:28] JM: What have been the biggest engineering problems in building Pack Hub?

[00:15:33] JD: Oh wow! I think probably the biggest problem has just been figuring out how to do the infrastructure well and in a way that's going to be stable, because we're taking the primitives that exist in Google compute engine and we're attempting to build an API on top of them. And some of those APIs are really, really resilient, obviously. The object store API is designed to just be really, really scalable and stuff like that. Some of them aren't totally.

So to figure out building, we had to make a separate project. And if you haven't used GCP before, projects are sort of like the top-level object. And we had to spend a long time figuring out like is it acceptable to just make this many projects? What sort of quota is there attached to projects? Has Google sort of assume that a project is created every once in a while when I go into the console and I click a button on the web interface? Or is this API really resilient?

And of course, there's a million ways that these things can fail. Sometimes you get just a VM that isn't functioning as well as it's supposed to, and sometimes your networking just doesn't come up correctly. And so figuring out how to detect those and recover from them. That's probably been the hardest part of it.

The other part that's really, really hard is figuring out how to. And this is more of a software design than engineering problem. It's just figuring out how to sort of design things and design the UI in a way that's going to be really, really intuitive for users, because we sort of see this as like successive layers of abstraction, wherein like the GCP console or the AWS console is really trying to abstract all of the functionality of a data center into a web interface.

And if you've ever gone into those consoles, it shows that that's how much they're trying to obstruct in there, because there's just all of these little nooks and crannies of the UI that are really confusing. When you need that feature, when you're actually in the situation that it was designed for, then it starts to make sense. You're like, "Oh! Okay, this is the button that I need to turn on this VPN peering so that I can use this from within by VPN." But when you don't, you don't need that at all.

And so we've been trying to sort of like shave that down into like a layer of abstraction that's just Pachyderm and figure out how to make this usable for users when they don't have sort of the escape hatch of like, “Can you go in and just check like is GCP reporting any errors with your network on these machines? Do you see connectivity there?”

We didn't want to expose all that to users, because once you do, it both makes the product worse, right? Because you have to deal with those types of things that people shouldn't really have to deal with to just to data science. But it also makes it a lot harder for us to secure it. It makes it a lot harder for us to know, “Okay, this person can't access somebody else's Pachyderm cluster. They can't sort of like borg their own Pachyderm cluster by inputting the wrong things here.”

So I think that's – When I sort of like look at the things that we spent the most time talking about, that was probably it. Because there were just a million conversations, so we realized actually like the user can sort of escalate these permissions and they can delete all of these things in their cluster and in someone else's cluster. And then what do we do? And the only thing we can do in that case is like, “Okay. Well, we have to cut off their ability to do that.” but then how do we let them do the things that they need that to do?

[00:19:01] JM: Are there any other ways that you’ve tried to prevent the user from shooting themselves in the foot?

[00:19:07] JD: There's a lot, and there's a lot in sort of the core Pachyderm product. And then there's a lot in hub on top of that. When we think about what some of the most interesting ones are. One of the sort of big things in our ethos with this, and this is a thing that we've lifted from the UNIX ethos, is that if you make a product totally impossible to shoot yourself in the foot with, then you also cut off the ability to do a lot of clever things that the users are going to want to do.

And so that's why I think like the first form of this question that people normally encounter with programming is like, “Well, why doesn't the language stop me from creating infinite loops?” And one, that’s sort of – There's a the famous proof of why that's kind of impossible to do or why it's impossible to have a program that detects infinite loops. But also, you could do that by putting some sort of a limitation on loops or something like that, but then there'd be a limitation and it be really, really hard for you to actually do the things that you want to be able to do with loops. Every language allows you basically write infinite loops, and a lot of times you want to do that.

With Pachyderm, that’s sort of like it's not that that simple of a case. But there are a lot of ways where you can do things like sort of manually moving around branches is one good example. And most of the time, users just want to commit to the master branch and output to the master branch, and they don't really want to mess with their branches. But there're a lot of clever things that you can do by moving around those branches. And so we don't prevent you from doing that.

A really good example from Pack Hub is it comes from via the scheduling primitive. So we use Google's auto provisioning and auto scale up within Pack Hub. And it's a really amazing feature, because it means that you can send a manifest that just says like, “This pipeline needs a GPU, and it needs 10 GB of memory, and it needs one CPU core.” And we just send that request to Kubernetes and say, “Hey, this is what we need.” And it talks to GCP and basically it says like, “Okay, I've got a machine that has enough memory, but it doesn't have the GPU. And I've got a machine that has the GPU, but it doesn't have enough memory. And so I've concluded that I need a new machine.” And it will go into the GCP console. I mean, it's not literally going to the GCP console. It’s using the API. And it will spit up that new machine for you. And it will take that machine and it will process your code. And then when your code is done processing, it'll scale down that machine, and you have automatically had to pay for only the amount of it that you use.

Scheduling has a ton of sort of like bad behaviors that can accidentally crop up where you can get sort of into these feedback loops where this finishing now triggers some more things that scale up even more stuff, and then they scale up even more stuff. And some types of scheduling can basically scale up based on CPU usage, which in Pachyderm, your pods are almost always using all of their CPU, because it's not like their web servers that are getting requests, and sometimes they're not getting that many requests. They've just got this data to process, and they're going to process it as fast as they can.

And so scheduling was something that we really had to think well about both just writing good documentation so users understood what the behaviors were going to be and then putting on like guardrails and warnings that are basically saying like, “This is how much this is going to scale up," and it's going to be forever unless you sort of like tweak these things so that these things will be able to spin down and things like that.

I think that's probably the biggest thing that we spent thinking about. The user can shoot themselves in the foot, because I think everybody's biggest fear with using cloud infrastructure is that like one day you wake up and you look at the bill and you're like, "Wow! That's like $10,000 more than I thought it was going to be," and it's because you've accidentally been running GPU's doing nothing. Or worse, your account got hacked and somebody's been mining Bitcoins with those GPU's, which is another thing that we spent a while thinking about, is like eventually somebody's going to try to mine Bitcoins on Pack Hub. That's sort of like the christening moment when you put up a SaaS services, is like have hackers try to mine Bitcoins on it. And what do we do? What do we tell the person when they're like, "My account was compromised and people been mining Bitcoins on it." Can we offer them a refund and stuff like that?

[00:23:24] JM: The process of getting an open source project to be a SaaS product, I'd like to know more about that, because you got to stand everything up for the user and orchestrate everything. Tell me more.

[00:23:37] JD: Yeah. That is basically the experience and the question that I've been getting to see firsthand for the last two years. And it's really, really interesting, because before you actually try to do it, you sort of just think that writing the core open source product is the hard part. And then after that, it's going to be like, "Okay, there is a well-known way to turn this into a SaaS product, and we just need to go ahead and do that."

Of course, as you can probably guess, as soon as you start actually doing it, you realize like, "Holy crap! This is actually so tough," and there's so many decisions. I think that's really – The essence of the difficulty is just all of these little things that you don't really think of as decisions until you actually start trying to implement it and get it out the door and you start having to answer the question of like, "Okay. Well, what does the model for users look like? Who has the privilege to do what? How do you organize them?" You need to have some concept of organizations. You need to have some concept of how people join those organizations and what their permissions are.

Those seem really simple when you don't have to think about the details. That as soon as you do, you realize there's a million ways to do that. Similarly, you have to sort of think about like what is the backend architecture for this look like? And you have this tradeoff between – In some cases, you would like to have everybody just on exactly the same infrastructure, because that's cheaper for you on the backend and you can sort of exploit militant multi-tenancy. Then you start to get into all of these problems of like, “Well, would you do about noisy neighbors?” The person that's using a ton of resources, and then there's some poor guy who just happens to have gotten scheduled on the same node as that guy. And so his Pachyderm cluster isn't working at all. And what do you do about when certain pieces of infrastructure die and you have to reschedule all of those? Is that an outage that affects one user? Is that an outage that affects all of those users? How do sort of like steel yourself against those? And also, how do you do security? Multi-tenancy makes security a lot harder.

Kubernetes and Docker containers help some with that, but they don't completely solve that problem. And so those are problems that you don't have it all with an open source project, because with an open source project and an enterprise product that we sell on top of that, it's just we give you the code and you deploy that on your own infrastructure. And so you get to make the decisions of who else is using this infrastructure. How do we keep it secure? And things like that. Suddenly, we’re responsible for all those.

I will say one of the main reasons that we wanted to build a SaaS product is that while of these things are challenges, they’re challenges that people have also running Pachyderm in production. And so building a SaaS product also allowed us to really level up our ability to run the product in production, because we have just now at least say to people like, “Look, here's how we do it in Hub, and it works pretty well. And it's been well-tested and all of the hub clusters are using it so you can at least know that this is a good way.” Whereas before, the best we can say is like, “Well, this is what we've done in our tests, and this is what we’ve seen other people do in production.” But we could be quite as confident in that information.

[00:26:47] JM: Are there some other places where you stub your toe in trying to productionize it?

[00:26:54] JD: Yeah. Stubbing our toe is probably a pretty good description of the entire process. I think one of the big tubs toes we had in the beginning was we really wanted to figure out a way to sort of merge the functionality with Pack Hub with our on-premises deployments and have some way. Because Pack Hub also is a really good way for people to connect to each other and a good way for you to basically say like, “Okay, here's something that I'm just – Here is like a pipeline or a dataset that I'm putting out into the world.” And we didn't want to completely silo off our on-prem deployments from the functionality within Hub. And so what we wound up doing was building this sort of like weird networking layer that would allow us to sort of proxy things from locally deployed clusters into cloud clusters and vice versa. And it really didn't work that well. And the basic reason for that was just that it was sort of ill-conceived. And I sort of realized from that that most of the reason that SaaS works so well is because you can control and unify all of that type of stuff. You have all of these other challenges to solve, but you get the great benefit that all of these clusters are running on the same networking stacks. They’re running on the same cloud provider. They’re running with the same everything.

And so that makes things a lot easier. And we were really, as you say, stubbing our toe by doing the exact opposite and saying, “Okay. Well, but we also want to include all of these other clusters in this functionality, and that we get none of those benefits and have even more challenges.” And we lost probably a good 4 to 5 months working on that problem before we realized like, “Okay, to we need to take a step back and do sort of what everybody else does with SaaS and just build a SaaS product,” and the fact that this is going to create these two silos is just something that we’ll live with for now and sort of figure out how to bridge the gap in other ways. But bridging it at this sort of infrastructure layer is just not working.

[00:28:57] JM: So we barely talked about Pachyderm usage. Can you tell me what’s your feedback loop today between companies using Pachyderm and yourself?

[00:29:06] JD: Yeah, absolutely. I mean, the feedback loop today is companies – We have a user Slack channel, and that's got a little over 2000 people in it right now. And so that's a pretty constant stream. I sort of see that as like the most base layer. Normally, the first time anybody hears about Pachyderm and we talk to them, that's where we talk to them. And from there, that leads to just a number of good conversations of like someone post a question, and I'll jump in and answer it, but then other people within the community will start talking there and we’ll just get hopefully a resolution for the original question, but also then a lot of good feedback on what we can do to make things better.

As we get more involved with customers, we will upgrade them to their own private Slack channels when it sort of gets to a point where they're telling, they like want to give us information like logs and stuff that's maybe a little bit sensitive. And DMing stuff around in Slack gets really unwieldy when you're trying to have a team support people, because like you can't really share something out of a DM. So we normally make people a private channel when it's like we find ourselves copying a bunch of things around in DMs.

And so then we've got a private Slack channel. And then we just use all of the sort of like standard telecommunication tool. So we'll have regular Zoom calls with our users. We're trying to do more to sort of like highlight users use cases and sort of do like little interviews with users that we can put up on the site that both help us to just understand their usage of the product and challenges better. And then also to just like help them to explain to the outside world like, "Here's how we're using Pachyderm," that's both good for us to show like, "Here's what usage of Pachyderm looks like," but also good for them in a lot of cases, because they are able to sort of recruit other people from within the Pachyderm ecosystem that want to work at their companies.

[00:30:57] JM: And can you give you me an example of a problem that has been encountered by a user that was relayed to you that lead to a solution?

[00:31:05] JD: Yeah, absolutely. I think one of the best stories for this that took – It took a lot of back and forth for users for me to really understand the problem was spout. We had sort of this consistent back and forth with users where they wanted to basically make a pipeline that didn't take any inputs and just outputted stuff. And that to me – I really didn't understand that use case for a while, because that to me is just inputting data into Pachyderm. And you can just use the API and you just input data, and a pipeline is something that's supposed to take data and process it. It's not supposed to just output data. So it took a lot of back-and-forth with customers for me to basically realize that the reason that people didn't like just using the API in this case is that they wanted the resilience that pipelines offer them, right?

You can you can access the API. You can access it from a web console. You can access it from the command line. You can script that. But if you want to turn that into a resilient service, say something that is listening to a Kafka stream and outputting that into Pachyderm, you need to go and like build a Kubernetes service from scratch. To me, I always thought – I tell a lot of people like you can just build a community service to do this, and it's really not that bad. And it took me a while to realize that in a lot of cases, the people who wanted to do this didn't even actually have direct access to Kubernetes. They were just they just access the Pachyderm API.

And so once I sort of got that and started thinking in terms of like, “Okay, we need to have some sort of a special type of pipelines that just basically send data out into the world. They don't consume data from the world at all.” And that led to the concept of a spout pipeline. And the analogy there is it’s spouting data out into the world like a spout of water or something like that. And once we built that, that was – You can you can sort of like tell really quickly when you've gotten it when the idea for a feature is right, when it's an idea that people actually want, because we released that. And within a month, almost all of our customers were using it somewhere. It was immediately one of our most highly trafficked features in terms of like people finding bugs in it much more quickly. People like having further requests for what they wanted to do with that type of stuff. And it took maybe a year of hearing that feedback from customers. Maybe I wasn't listening is much as I should have, but I never – At no point was I thinking like this is just a stupid request. People just shouldn't want this. It was more I just didn't understand exactly what it was that they wanted and sort of couldn't understand how it was different from the solutions that I thought was more that we just needed to sort of evangelize the solutions that I sort of solved this problem, rather than building something. And then when we eventually figured out what to build, it really worked well.

[00:34:00] JM: So spouts are like data transport facilities for – Like if you want to ship Kafka data to your Pachyderm cluster, or if you want to ship like a change log from DynamoDB or something.

[00:34:14] JD: Any data that sort of takes the form of like there's an endpoint that you can kind of subscribe to, right? So you can do it for Twitter feeds. Although that the Twitter API keeps changing and they keep messing with that, so that one works less and less well. But in theory, you can do it for like SQS, is I think the Google equivalent. You can subscribe to changes like on an object storage bucket. It's basically going back to the Hadoop ecosystem, which is sort of like a lot of the original inspiration for Pachyderm. You have a lot of special-purpose services within there whose job is basically to just be a connector between two things.

So I know like I've used Secor before, and I think that that was basically just like from Kafka to push things into HDFS. That's how I remember it at least. It may have evolved and changed since then. And we sort of always thought, with Pachyderm, because we have this containerized model, we really shouldn't have to be as focused. We shouldn't have to have these special-purpose services that connect things. We basically have what we think of as omni adapters on both ends, because they're containers. And so you can put a container at the front of your DAG, and that can talk to whatever you want, because you can put whatever you want in it and you can put one at the end that's sort of egresses to whatever you want.

And so it's now not an uncommon architecture in Pachyderm to have like a spout at the top of your DAG that consumes data off of a Kafka stream or something like that. And that gets pushed into Pachyderm, and then you all sorts of crazy things happen in between. You can train models on this. You can do inferences on this. And then at the end, you have something that emits events back to that same Kafka and you're basically just allow yourself to have some free-form manipulation within your Kafka stream of like, "All right. Here's where I can basically run a script in a distributed fashion that transforms all of my Kafka events and turns them into different events."

[00:36:10] JM: How does having version controlled data science change the workflows for data scientists?

[00:36:18] JD: Having version controlled data I believe massively impacts the workflow looks like. And I can give you one really poignant example of that. It's what we're allowed to do so. As I've mentioned on previous podcasts, I worked at Airbnb on their data infrastructure team. And that was where a lot of my early experience of what it's like to do this stuff in production came from. And one of the things that we had at Airbnb, which is very, very standard within the Hadoop ecosystem and it's just a standard way to do things, but nevertheless had a pretty negative effect on the data science workflows, is that we ran our entire pipeline on all of the data from scratch every single day.

And so like the top level of our pipeline was a snapshot of our production databases that was taken every single night. And when we took that snapshot at midnight every single night, that was when we would kick off our pipeline. And that would run for about nine hours processing all of the data that was in that database even though only 1% of that data has actually changed since the night before. Most users are the same as they were last time. Most listings are the same as they were last time.

And so what that meant for data scientistsis that you really only get one chance per day to run your code at scale, right? So if you're trying to debug some problem that came up last night, you do all the work you can to get your code in shape. You ship it off that night, and then you come in the morning and hopefully it runs successfully. And if not, you get another shot the next night.

And what sucks about this, of course, is that your feedback loop in any development task, it really, really determines how quickly you can get it done. And so if it's something simple, then this just adds what should have been an hour to test your code, is actually now a day. If this is something more complicated, it can easily turn what should've been a day into like multiple weeks.

Where I'm going with this in terms of version control is that the really cool thing about having your data version controlled is that you can tell if you've already processed something. So what the same workflow looks like in Pachyderm is that when you push a new version of your code, we can run that immediately, and we can run that only on the chunks of data that the last piece of code failed on. So if you failed on 1% of the data and another 1% is new since the last time you ran this pipeline, then we're just going to run on those two little chunks of data, the new data and the data that failed last time. And so your feedback loop comes down monumentally, because it's really, really costly as more and more data piles up to have to run it on all the historical data.

This also sort of leads to this hybrid in Pachyderm between streaming workloads and batch workloads. Most of the time, those are seen as just sort of completely different pieces of infrastructure that run them different methods of writing the code. They're just completely separate. In Pachyderm, there is absolutely no difference. And so we get to tell people when they write their first pipeline. And when you run it for the first time, then it just runs on all the data because it hasn't seen any of the data. And then they ask, “Okay. Now, how do I make the streaming? I want to process a little bit of new data.” And we get to tell them, “You already did.” And you go and commit a little bit of extra data, and it just magically runs on just that new data and outputs a result as if it had run on the entire thing. And that's really all coming from the version control system. That’s just because we have this hash of the data and we know exactly what it used to look like. And every time we run something, we hash the code along with it and we get this just ID for like, “Here's a result of running this.” And if the ID exists, we already ran it and we just use those results.

The other thing workflow-wise that version control really allows you to do that’s new is it allows the lineage system to be much more robust. So there's a lot of systems now. As far as I know, we were one of one pretty early ones. But now it's very common in the Hadoop ecosystem, in like Kubeflow. All of these systems have a way of tracking this data came from here and was processed and went to here. And that's the basic description of what lineage does. But version control makes it a lot more robust, because if you don't have version control, then sure you know where the data came from. But is the data that you're going to find they are actually the same as what it looked like when this job was actually processed?

And so this is a big problem that when you're just referencing like here's an object storage bucket and we read a file that was at this path, well maybe the file got updated. And so now your lineage is actually lying to you when you go to ask the question of why does this output look like this? You're going to get misled by the fact that there's some data there that isn't what was actually processed. It’s just sort of claiming to be what was processed.

When you have version control, the things that you reference are immutable snapshots. So the worst that can happen is that they can be deleted and garbage collected, and then you won't be able to see what was there. But the system will never lie to you. The system will never say this is the data that was processed, when actually it isn't, because it has to match that hash that you have.

[00:41:39] JM: Had there been any ways in which Pachyderm usage has surprised you from the user base?

[00:41:45] JD: Definitely. I mean, to be honest, it's not a surprise anymore. But when we started this, machine learning was nowhere near what it is today. And so the fact that people were going to want to run these huge long-running jobs with GPUs that crank through and just like train these machine learning models, we had that in our mind as maybe a use case at some point. We didn't think that it would become the use case, and that really just sort of reflects the history of the company of like 2014 was when we started working on this. And that was like right when the like early deep learning papers were coming out and were showing amazing results. And it’s sort of – We didn't really see the hype train coming and what that would ultimately become.

So that's probably – It being used for machine learning in the of itself isn't that big of a surprise. But how big that was and how much of our usage wound up being in that. That was a pretty big surprise. The other thing that we, of course, didn't see coming at all was the sort of data ethics movement that has evolved. And lineage is probably the best example of this surprise, of when we implemented lineage, we saw it 100% as a feature that is useful for doing data science, because you're constantly in this situation where you want to ask where did this data come from? Why does it look like it looks like right now? If this model looks to be spitting out garbage, is it because the data that went into it is garbage? Or is it because the code did something wrong?

If it looks really good, it's like, “Well, what was so good about this dataset and stuff like that?” We never could have predicted that people were going to go and pass laws, like the right to an explanation, which is basically a right that you have. I think this is under the GPR. I think it's in Europe. I'm not sure if we have an equivalent thing here in the US, where you can basically go to a company that's made, say, an algorithmic decision on whether or not you should receive a loan. And you can ask them, “Hey, what's all of the information that went in to this decision?” And that's of course to sort of keep them honest on like, “Well, did you just do this because of my race, or my gender, or something like that?” But when that law was passed and we saw that, we’re like – It felt as if the government had sort of legislated the feature that we had built like three years ago essentially, because lineage information is just the thing for that. That gives you just the ability to hardcore say within a system like, “Here's everything that like was exposed within this container. We know that it is hermetically sealed. This is the only thing that could've gotten in there to build this model." And that's just – Again, that's a total surprise and accident, because we never thought that that was going to be something that was going to be legislated.

The other thing that's really, really fun about Pachyderm is because we have this very general, like containerized model for the computations, we learn about all of these cool tools that we've never heard of, because they don't run really well on anything else. But you can always put them in a container and get them to run on Pachyderm. And so there's all of these cool like physics toolkits and genetics toolkits and like tools for doing financial modeling and image processing and stuff like that that I never really would've encountered, because I don't do the specific type of work that requires the tools. And so I'm never going to find the tool. And then once I do find it and someone mentions like, "Oh yeah, we're running this in Pachyderm," and I look it up. I'm like, "Wow! There's this whole sub-community of people who are doing this really cool open source development on this tool for like .01% of the population that happens to be interested in this type of physics," and stuff like that.

We always had sort of believed that that was going to be the type of usage we would get because of the system we were building, but actually getting to experience the individual examples of this and see what those specific toolkits are is a pretty fun part of running Pachyderm.

[00:45:54] JM: Well, we're winding to a close. Is there anything else you want to add about Pachyderm?

[00:45:59] JD: I don't know. You've asked such great, great questions about Pachyderm. I mean, I guess the last thing I'll just say is I would love for any users to use Hub and give us feedback in our Slack channel or via email, if you want to put my email in the episode notes or anything like that.

Pachyderm we think is really the best way to do high-powered big data stuff, high-powered data science and stuff like that. We think it's a great platform for that that both fits into the modern stack in terms of you can just deploy it straight on Kubernetes. And it also gives you features that we don't think you're going to find anywhere else. The version control and lineage tracking we think is something that we've thought about a lot more than anywhere else. And ours is a system that was sort of built from the ground up to support that in a really clean way rather than it's sort of being added on on top afterward. And so yeah, I think the only thing I want to say is try Pachyderm if you're interested.

[00:46:58] JM: Joe, thanks for coming on the show. It's been a real pleasure talking to you.

[00:47:01] JD: Yeah, absolutely. Thanks for having me, Jeff. It's always fun to talk with you about what's going on on Pachyderm.

[00:47:06] JM: I'm sure there will be more in a year or so.

[00:47:09] JD: Yeah, absolutely. Absolutely.

[END]