EPISODE 755

[INTRODUCTION]

[0:00:00.3] JM: When TensorFlow came out of Google, the machine learning community converged around it. TensorFlow is a framework for building machine learning models, but the life cycle of a machine learning model has a scope that is bigger than just creating a model. Machine learning developers also need to have a testing and deployment process for continuous delivery of those models.

The continuous delivery process for machine learning models is like the continuous delivery process for microservices, but it can be more complicated. A developer testing a model on their local machine is working with a smaller data set than what they will have access to when it is deployed. A machine learning engineer needs to be conscious of versioning and auditability.

Kubeflow is a machine learning toolkit for Kubernetes based on Google's internal machine learning pipelines. Google open sourced Kubernetes and TensorFlow and those projects have users on Amazon Web Services and Microsoft and many other cloud providers.

David Aronchick is the head of open source machine learning strategy at Microsoft. He joins the show to talk about the problems that Kubeflow solves for developers. He also talks about the evolving strategies for cloud providers. David was previously on the show when he worked at Google. In this episode, David provides some useful discussion about how open source software presents a great opportunity for the cloud providers to collaborate with each other in a positive some relationship.

It really has become an amazing environment where you have these open source projects that a vast quantity of software engineers are using, and then you have giant cloud providers that are contributing to these open source projects. It's interesting to see this in its early days, because this seems like a trend that's just going to continue and provide a lot of value, because if you think about just the level of software quality that we will see as we just have these titans of industry contributing to the same open source software projects, not just in the Kubernetes realm, but now in machine learning and in other verticals, it really just seems like a bright future for software development.

It was great talking to David. I hope you enjoyed the episode as well.

[SPONSOR MESSAGE]

[0:02:35.6] JM: Kubernetes can be difficult. Container networking, storage, disaster recovery, these are issues that you would rather not have to figure out alone. Mesosphere’s Kubernetes-as-a-Service provides single-click Kubernetes deployment with simple management, security features and high availability, to make your Kubernetes deployments easy.

You can find out more about Mesosphere’s Kubernetes-as-a-Service by going to softwareengineeringdaily.com/mesosphere. Mesosphere’s Kubernetes-as-a-Service heals itself when it detects a problem with the state of the cluster, so you don't have to worry about your cluster going down. They make it easy to install monitoring and logging and other tooling alongside your Kubernetes cluster.

With one-click install, there's additional tooling like Prometheus, Linkerd, Jenkins and any of the services in the service catalog. Mesosphere is built to make multi-cloud, hybrid cloud and edge computing easier. To find out how Mesosphere’s Kubernetes-as-a-Service can help you easily deploy Kubernetes, you can check out softwareengineeringdaily.com/mesosphere, and it would support Software Engineering Daily as well.

One reason I am a big fan of Mesosphere is that one of the founders, Ben Hindman, is one of the first people I interviewed about software engineering back when I was a host on Software Engineering Radio. He was so good and so generous with his explanations of various distributed systems concepts. This was back four, or five years ago when some of the applied distributed systems material was a little more scant in the marketplace. It was harder to find information about distributed systems in production, and he was one of the people that was evangelizing and talking about it and obviously building it in Apache Mesos.

I'm really happy to have Mesosphere as a sponsor. If you want to check out Mesosphere and support Software Engineering Daily, go to softwareengineeringdaily.com/mesosphere.

[INTERVIEW]

[0:04:54.6] JM: David Aronchick, you are the head of open source machine learning strategy at Microsoft. Welcome back to Software Engineering Daily.

[0:05:00.9] DA: Thank you so much for having me.

[0:05:02.1] JM: Today we're talking about some various open source projects and their relationship to machine learning and distributed computing. I want to start with TensorFlow, because you've spent a lot of time with the TensorFlow community. Before TensorFlow came out, there was a wide variety of machine learning tools that people used and there didn't seem to be much convergence. What did TensorFlow do differently that caused convergence around the project?

[0:05:31.6] DA: I think that's a really interesting question. To be honest, you still see an enormous amount of machine learning toolkits out there, whether or not they're straightforward solutions for various – not simple by any stretch, but solutions that can be solved in a single node, like things scikit-learn and NumPy and so on. People who understand R are certainly using R for that, or even writing raw Python, that's all great.

The thing that TensorFlow really came along to do was to standardize really complicated problems and make them much more straightforward to address them without having to change the way that people were designing and deploying systems. Now in just a few commands, you were able to do rich machine learning solutions, such as recurrent networks, convolutional networks and so on. Also run them in much more scalable ways using distributed nodes in the various tiered system that TensorFlow offers, without having to go and write your own orchestrator.

That was a really, really powerful solution. Such that even though there are phenomenal models in all those frameworks that I mentioned earlier, some of the most complicated frameworks, or most complicated models and solutions were really only possible to converge in a finite amount of time using this highly distributed system. That's what TensorFlow really unlocked.

That said, if you go out and you look at the machine learning usage today, 95% plus of it is in not TensorFlow. It just so happens that TensorFlow is exceptionally good at doing deep learning and that's where some of the most interesting machine learning is occurring today.

[0:07:22.3] JM: As you said, there are many machine learning jobs that we can do on a single node. My laptop is pretty powerful. My smartphone is pretty powerful. I could get a bunch of data about movies, for example, and maybe I have user ratings of those movies and I could build a recommendation system that will run on my laptop, it will run on my phone. I can have a system where I get additional data over time and that machine learning, now it's a “machine learning model,” it's updating over time, it's getting smarter over time.

This would work on a single node. There's a bunch of different frameworks that work for this. That's totally fine, right? Most of these machine learning jobs today, can most of them run on a single machine or where does it get to the point where machine learning workloads get complicated enough to need multiple machines?

[0:08:19.9] DA: Yeah. I think that's an excellent question as well. Ultimately, sadly, there's no direct answer that you say, “Okay, here's the clear line where you have to do one versus another.” The reality is no matter how great distributed systems are at running these very large workloads, nothing will be simpler to debug and run than a bunch of stuff running on a single machine.

When the first deep learning frameworks came out that really had breakthroughs around things like object detection and identification and so on, that was the early 2010s, I believe. The reality was is at the time, attaching 8 GPUs to a single node was effectively impossible. Certainly clouds didn't offer it and you had to do a whole bunch of work on hardware in order to accomplish that. Really at that time, the only way to achieve true parallel workloads with lots of GPU processing was to do this distributed architecture that worked across a whole bunch of machines.

To your point though now, every major cloud offers a VM with multiple GPUs attached and your laptop is two or three times more powerful than the one that you were running in 2010 for doing image processing and other tensor processing. You're now at the point where you can do a lot of things on that single machine. That said, so much of this is not about what happens on your single machine. We do see the vast majority of machine learning experimentation occurring on a laptop. There's no question about that, but a laptop no matter how great it is is not production ready. It's not repeatable. You can't hand it to another researcher in my organization. I can't monitor it or log it when it comes to production. I can't debug it in six months' time when the model has gone off the rails and figure out what changed about my data, or the environment, or something like that.

The reality is that no matter how great it gets in that single node, you're going to need to develop DevOps and infrastructure engineering practices around your machine learning environment in order to truly make it production ready, so that you will be able to debug it and you will be able to meet your business needs over time.

More importantly and this is something that we really were thinking about when we started Kubeflow was how do you extend beyond just training? Training is good. Obviously, there have been enormous breakthroughs there, but it's all other steps. The 90% of other work that's involved when it comes to machine learning, that really is going to be the next generation of stuff.

[0:11:13.6] JM: These parts of machine learning, there is the training process, there's also the serving process of a model, there's the updating of a model that has already been trained, that is already serving users, you've got all these things that are offshoots of these training and serving and updating, like you can do AB testing between different models, things like that. Describe how TensorFlow gets used in some of these different parts of the machine learning engineers' workflow.

[0:11:46.8] DA: Yeah. I think it's really an interesting situation, because when you go – a lot of people look at Google and look at what they did with TensorFlow and say, "Okay, this is the gold standard." Everyone is going to use what Google uses, because for better or worse, it is one of the top machine learning organizations in the world. They use it in Android, in their search engine and ads and so on and so forth.

It’s interesting, as I came over to Azure I was like, “Oh, Google's clearly the leader here.” I looked around every corner here and there's machine learning all over the place as well. What you find is that it's much less about a single framework; single the use of TensorFlow in order to do machine learning.

Again, it's not to say it's a deeply critical part of the process, but I highly encourage everyone to go out and read a paper by Google about TFX, that's a TensorFlow Extended. It describes all the systems that they use internally to actually build machine learning models. If you look at it, they have a wonderful diagram that talks about this and they talk about TensorFlow certainly of course, and they talk about the trainer and then also serving, but those are just two steps.

They also talk about how to do data ingestion, analysis, validation of the models, orchestration of the models as they roll out to production, make sure that the infrastructure is appropriate for them, they run properly in production, and how they do updating on-the-fly of the underlying training, making sure that the model is actually performing better than the existing one and roll it out with no change to the customer. All those systems are outside of the core of TensorFlow, as they should be.

In any properly orchestrated microservices environment, you want your systems to be loosely coupled together around a microservices orchestrator or organization and then use an external orchestrator that knows how to speak machine learning in order to do that. At Google internally, you would use Borg. More often than not, externally you would use a workflow orchestration system like Kubeflow running on Kubernetes.

[0:14:01.0] JM: Excellent point. You're describing all of these finer points of the machine learning usage and delivery process. Most people don't necessarily need that today. In the future, they will use it at some point. I mean, eventually people have migrated towards the “DevOps world.” I think in some ways, perhaps the machine learning world today is like the world of DevOps was, I don't know, five or six years ago. Do you think that's a fair analogy?

[0:14:36.7] DA: Yeah. I think it generally is, but I would really implore people not to think about this in the "future." I don't want to dismiss it. I get it. Setting up machine learning frameworks and pipelines are very hard. Let me give you a little analogy here. Let's say you were a financial engineer or a financial analyst working at a company and your boss came to you and said, "Hey, you know what? I'd like you to go build a model in Excel to decide what the next three regions we should expand to would be."

You would take a whole bunch of data; what the average population and GDP growth and average salary for those particular locations might be, all these various things that you were investigating. You'd build it into a model and it spit out something on the other side using Excel to go and do that. Maybe you decided to go forward with that. You made a presentation around it. You explained everything that was involved.

In six months' time, maybe one of the regions wasn't doing so well and you might want to go back and investigate your Excel model, figure out what went wrong. Or maybe your model works perfectly and you wanted to figure out the next three places to expand to. You would go back and reuse your model, or maybe you'd update it based on the new requirements of the geographies.

Machine learning is very, very similar today. A data scientist might download a blob of data, import it into their local machine learning environment, run it through a data frame and do some analysis, manually exclude some outliers, manually do some feature engineering and convert whatever blanks to zeros, or something like that. Then they'll take that, they'll build a model, they'll test it and they'll want to roll it out to production. Or if not production, they'll at least want to get an answer that they can use internally and say, "Hey, you know what? The next three regions we should expand to is X, Y and Z.

In three or six months' time, you're going to need to reproduce that. You're going to need to debug why the regions didn't work properly, or maybe you've left the company and the next person is going to want to pick it up. How do they reproduce it? Things that exist on your laptop that you make business decisions from, those are business critical situations. If organizations don't do some work to remove it from that single laptop and put it into an environment where it can be reproduced and used, you're going to end up with pain. It may not be today, it may not be tomorrow. At some point, you're going to run into real trouble when you're trying to reproduce or examine exactly what was going on.

That's where machine learning is today. More often than not, people are building models of any kind in a local way. We really do need to inject some software engineering processes in here in order to make these things really things that your business can depend on.

[0:17:43.5] JM: I think that's a good point, because in the early stages of putting machine learning into an application, I think the way that it gets perceived within an organization is we've got the production engineering team over here and they're building “microservices” that are serving data. Then we've got this team over here that is the data science and machine learning team and they're building models. These models are these additive things that are like a layer over our business logic, our microservices.

The reality, or at least where things are going, or where things should be is that it's just one big blob of stuff that can be providing your users with value and you're going to need really good processes around all of it. If you treat the machine learning world a little bit differently and like we don't need these, like processes we don't need really good release strategies, like we don't need the productionization workflows really ironed out, so that we don't have this one engineer that's the hero engineer that is responsible for doing all the machine learning stuff, that has tons of institutional knowledge trapped in their brain, and if they leave our machine learning stuff is all going to fall over or become unreliable. That's not something that we want to deal with.

[0:19:12.4] DA: Yeah. I couldn't agree more. I mean, could you imagine going to any infrastructure engineer and saying, “Okay, here's this opaque blob of a thing that I've compiled. I'd like you to take it and run it across 14 machines. Oh, by the way, you're going to take on the SLA for making sure this thing has four 9s and there's no monitoring involved. Go to work.”

[0:19:37.4] JM: By the way, it's calculating the prices for our ride sharing app.

[0:19:41.9] DA: Yeah, exactly. You'd have to restrain them from jumping across a desk to throttle you in the neck. These things right now, God bless my data science brethren, but you're throwing a bunch of weight in a directory and you're asking me to move that into production, it’s just – that's Crazy Vance. There have to be some standard software engineering practices that we start to get all across the system in order to really bring machine learning to the next level.

[SPONSOR MESSAGE]

[0:20:19.7] JM: DigitalOcean is a reliable, easy-to-use cloud provider. I've used DigitalOcean for years, whenever I want to get an application off the ground quickly. I've always loved the focus on user experience, the great documentation and the simple user interface. More and more people are finding out about DigitalOcean and realizing that DigitalOcean is perfect for their application workloads.

This year, DigitalOcean is making that even easier with new node types. A $15 flexible droplet that can mix and match different configurations of CPU and RAM to get the perfect amount of resources for your application. There are also CPU-optimized droplets perfect for highly active frontend servers, or CICD workloads.

Running on the cloud can get expensive, which is why DigitalOcean makes it easy to choose the right size instance. The prices on standard instances have gone down too. You can check out all their new deals by going to do.co/sedaily. As a bonus to our listeners, you will get a $100 in credit to use over 60 days. That's a lot of money to experiment with.

You can make a $100 go pretty far on DigitalOcean. You can use the credit for hosting, or infrastructure and that includes load balancers, object storage, DigitalOcean spaces is a great new product that provides object storage, and of course computation. Get your free $100 credit at do.co/sedaily. Thanks to DigitalOcean for being a sponsor.

The co-founder of DigitalOcean Moisey Uretsky was one of the first people I interviewed and his interview was really inspirational for me, so I've always thought of DigitalOcean as a pretty inspirational company. Thank you, DigitalOcean.

[INTERVIEW CONTINUED]

[0:22:26.8] JM: Okay, so as we started to talk about Kubeflow, which is a project that uses TensorFlow and Kubernetes together. We should talk about why Kubernetes is useful with TensorFlow. I have done some shows where we have touched on distributed machine learning, things that are like, this model is so gigantic, we have to break it up into multiple machines, or we have so much training data that we need two parallelize the ingestion process of all this training data.

Now that's another area of distributed computing and machine learning that we could talk about, but I think we could probably start at a place that's a little bit more pedestrian, right? We couldn't just talk about with Kubernetes and TensorFlow together, we can start to move towards having this more harmonious release process. Is that right?

[0:23:22.2] DA: Yeah. The way to think about Kubeflow is it takes all the best of a cloud native architecture, which is to say you're able to declaratively describe the artifacts that you roll out; in this case, containers and service endpoints and deployments. You can hand that to Kubernetes and Kubernetes will roll it out in a way that you could be assured will either land, or it'll give you a very clear answer as to why. Most of the time, that rollout is a single YAML file, something that anyone can read.

Kubeflow is a platform that allows you to use that deployment methodology with a whole variety of machine learning tools. I talked about many of the components in the TensorFlow extended paper, but Kubeflow is definitely not just TensorFlow. There is PyTorch and Chainer and MXNet, XGBoost and so on, all of which for training.

In the serving category, we have Seldon and TensorFlow serving, we have TensorRT from NVIDIA. We use rich tools for doing things like hyper parameter optimization through a community contributed tool called Katib based on Google's Vizier technology. We have tools for doing analysis of your data through things like TensorFlow, data validation and TensorFlow model analysis to ensure that you're not targeting just outliers.

I think there are over 25 different packages all available via Kubeflow, all deployable with that very clear cloud native framework involved. Now as of November, we have not just the packages that you deploy, but we also have a brand new workflow orchestration based on the

Argo technology from the Argo project recently acquired by Intuit! that allows you to do Kubeflow pipelines.

Now you can take your rich end-to-end machine learning pipeline based on whatever technology you like and string it together in a way that gives you an end-to-end clear path for doing builds and deployments of your new models as data gets updated. Now you've started to really coordinate and configure between these various things in such a way that even if you're not doing true distributed training, or data processing, even if each of these things is just a single container that you're deploying, you're deploying it in a declarative way, so you can specify exactly the right version and exactly the right libraries.

Know that because this component needs Python 2 and that component needs Python 3, you don't have to worry about them stomping on each other's dependencies. That's what containers are for. They give you an isolated environment to roll that out and still have support going forward.

[0:26:22.8] JM: TensorFlow is this framework for making machine learning applications that require resources. It has distributed application development that require resources and Kubernetes is the provider of those resources. It makes sense that these two projects would have something intermediating them, like Kubeflow.

[0:26:48.4] DA: Yeah. One of the things that – when we started the Kubeflow project in December of 2017, we actually based it off a technology that had come out of the TensorFlow repo already built for running TensorFlow on Kubernetes. The reason that people needed that was because setting up and running TensorFlow is not – just TensorFlow is not the easiest thing in the world. There are three different layers of infrastructure; you have master nodes, you have parameter nodes and your have worker nodes. Each of which can scale independently, each of which need rich ways to communicate back and forth to each other.

If machine one has your master node on it and machine two has a parameter node and machine three has a parameter node and machine four and five each have worker nodes, if machine three dies and you bring up machine six with a brand new parameter node, today you would have to have your own name service and resolution and make sure that things were able to recover.

By deploying this using Kubernetes and Kubeflow, you now get a rich orchestration system that abstracts away having to understand all of the nitty-gritty of your various infrastructure. You took this formerly very complicated way of rolling things out and gave people a much simpler declarative way to do it. The reality is that pattern for doing even single node deployment, but especially for distributed nodes of declaring what you want to roll out, whether or not it's TensorFlow or PyTorch or Chainer or MXNet and having Kubernetes handle the resources and all the wiring between them makes it so that your ML engineers can now operate at one level higher and focus on things that are much more interesting to your business.

With a lot of the work around things like Kubeflow pipelines and Jupiter integration and so on, your data scientists are also able to operate at even one level higher than that. That's really what we're trying to go for here, how do we give people clear layers and let the systems do the things that they are exceptionally well at?

[0:29:04.9] JM: Let's go deeper into this example of Kubeflow pipelines. Kubeflow pipelines allow the developer to define complex machine learning workflows. What is a Kubeflow pipeline?

[0:29:20.0] DA: Kubeflow, or the Argo project was started to make workflow orchestration on Kubernetes something that was really cloud native. Taking an arbitrary example today where we wanted to use a machine learning example. We'll say you wanted to create a website that had thumbnails on it, right? Or hosted images and each image would have a thumbnail associated with it. Today, maybe you would have a page that would upload, allow upload of an image, that image gets written to your lob, then you'd have a second process that took that image and recognized that an image had been uploaded there, it took the image, it down-scaled it, so that it was the appropriate resolution and cropped it and so on so it made a thumbnail and then it wrote that to a different file or directory.

Then when it was done and verified that the process had completed correctly, it updated something, for example a database that hey, here's the location of this thumbnail and now the entire image is ready to store. That's a very, very basic workflow orchestration that people have written all over the place.

It turns out that machine learning has very, very similar problems. Let's take an image example. The first thing you might want to do is you'll upload that image and maybe you discover that you don't actually need colors to do object detection in that. Let's downscale it to be grayscale. Then you're going to use some automated process to that pixels in that image and convert them into features that your system can read in as part of your machine learning framework.

Then you'll have your model, look at wherever those features have been stored so that it can do an inference against that image and get you an actual answer. That's a standard machine learning pipeline. In that particular case, today before the advent of some of these workflow orchestration tools, you had to wire all that together, which was notoriously error-prone.

By using something like Argo and specifically, I'll get to Kubeflow pipelines in a second, by using something like Argo, you're doing that in a cloud native way. You're using the native constructs that Kubernetes provides to say, "All right, here. I'd like you to watch this container. When this container finishes, I'd like you to kick off this next step. If this container has an error, I'd like you to fire this event and so on." Instead of having to write all that manually, you're able to do it using native – cloud native concepts.

Okay, so that's all great. Except if you're a data scientist, you really don't care about cloud native concepts. I don't mean that dismissively. It's just you're probably working at the Python level or an R or whatever language makes sense to you. You really want to operate it one higher level up and say, "Hey, you know what? Just do your downscale, or do your copy from directory A to directory B and so on."

Kubeflow pipelines lets you write rich machine learning pipelines that are cloud native under the hood, they understand all the core concepts of running cloud natively inside Kubernetes, but it lets you do that in Python. It lets you operate at a level that makes sense to you, rather than what makes sense to Kubernetes. You might write an entire rich Kubeflow pipeline without even realizing you're ever running on top of Kubernetes at all. You're just writing Python like you always did, except this Python is checkable, it's lintable, it is strongly typed if you'd like it to be, it has rich metadata information to allow you to take further action on it and do all the kinds of machine learning engineering things we were talking about earlier, without asking you to learn a brand new language in a brand new paradigm.

[0:33:20.5] JM: Okay. I'd like to continue through some higher-level examples that help us illustrate how Kubeflow can be used. You mentioned a term earlier, the term feature engineering. Can you define what feature engineering is and how it fits into a machine learning developer’s workflow? Then we can talk about how Kubeflow assists in that.

[0:33:44.2] DA: Yeah, absolutely. Feature engineering, it's a really simple problem that somehow we've turned into this crazy world of abstract terms and so on. Let's take a really, really, really basic example. Let's say you were going to do something around NLP, that's Natural Language Processing. What you would want to do is figure out the sentiment of a particular paragraph, okay?

Today, you would get a paragraph of information made up of I don't know, 78 words. You're looking for whether or not this paragraph, this review says you're happy or sad about this particular thing. An incredibly basic way to do it would be to create the so called bag of words. This is basic feature engineering. When you look through every word in that paragraph and you say how many times were they repeated, and then you get this list. Basically, you add it up and you say, “Well, if the word happy is in there more than the word sad, we'll call that a happy review. If you do the opposite, then we'll call it a sad review.”

That's basic feature engineering. You take your data that is input in one format and you convert it into something else where you say, “Okay, I've counted it all up and the word happy appears here three times and the word sad here appears here zero times. Therefore, it's a happy review. We'll call this happy and that'll be the output of our machine learning model.”

Like I said, that's very, very basic but the reality is obviously that's not going to be very good. If it contained the word grotesque, that would obviously fail, but probably would be something that I would want to flag as a sad review. You're going to need something more complicated. In this case, there's a natural language processing technique called developing things bigrams or trigrams. This is generalized to the term engrams.

A bigram is when you take two words that are side by side and join them together and say, “Okay, this is very happy or very sad or not happy or something like that,” and then you start to get a little bit more formulation around that. Then as you might imagine, trigrams starts to get more and more. Then you get more and more complicated as well when you start to get things like trigrams, then you have that many more permutations and it becomes more complicated.

Feature engineering is the process of taking your raw data and translating into something that your machine learning model can actually operate on and balancing how much of that engineering you do, such that you're actually able to get answers out without taxing your system to the nth degree.

To extend that then into Kubeflow, with cube flow pipelines you can build a very easy step that does your feature engineering for you. Now Kubeflow doesn't offer feature engineering natively, but it makes it very easy to take whatever feature engineering you like and use it in a pipeline format. I gave you a trivial NLP example there. There are many, many algorithms across many different domains for taking raw data and transforming into something your model can either be trained or make inferences on.

[0:37:06.4] JM: What specifically is Kubeflow giving you there that you wouldn't have with just a regular TensorFlow?

[0:37:14.4] DA: TensorFlow doesn't offer any native feature engineering today. It expects that you handle the feature engineering separately. Even if it did, what you would want to do is you'd want to break up your feature engineering from your training step, or from your inference step, because that's just good practices. You want each of your microservices to do one thing and one thing only. Obviously, that's an idealized state. If you can get there, that's really what you're going for. In this case, like I said I've given you a pretty basic example, but instead of throwing everything into a single blob because you've been able to use Kubeflow pipelines and in step one, you do your feature engineering, take from your paragraph and turn it into a bunch of features that your model can process on.

Step two is take those features and then come out with answers, or train your model, whatever it might be. You've broken it up and that gives you the opportunity to say, “Okay, hey. You know what? This thing failed last night. Let's go debug which step it failed in and we can dig in there.” Or maybe it didn't fail, but you need to update something. You're no longer using English, you’re using French, and so you're going to need to update the way your features get engineered. Or maybe you want to add a second step in there and you say, “Okay, after I do my initial feature engineering step, I'm going to want to do statistics generation on it, or outlier exclusion,” and say, “Hey, you know what? I want to remove stop words, for example, like a, and and the.” Because again, the more that you can streamline your data, the faster and often more accurate your model and inference will be.

Now you can stick that in between it. Previously where it went A to B, now it might go A to C to B. That's obviously trivial to do with something where you have that loosely coupled orchestrated system like Kubeflow pipelines provides.

[0:39:09.3] JM: The point here is that you want to have the different sections of your machine learning pipeline broken into different modules that can be moved around, scaled up independently, just for the same reasons that you would have microservices broken up in a microservices architecture.

[0:39:33.0] DA: Yeah. You really hit the nail on the head. The world is moving towards microservices, because of all the benefits that you see; independently updatable, transparent, debuggable, independently scalable. These are all huge benefits that microservices cloud native architecture will provide. Machine learning is no different.

The key is though as you move to those environments, you want to make sure to have a platform that layers over the top, or helps you with that orchestration, because otherwise you're going to be taking on a brand new challenge. I remember when I first started with Kubernetes, I built a simple app for myself and all I wanted to do was tail the log of what was going on to make sure that things were operating properly. Well, tail on my machine lets me attach myself to the log and just watch it until I cancel it.

When that log is being run across four different machines in 10 different pods, that becomes a challenge. Either I figure out a way to build distributed tail, which people have done, or I use my platform. In this case, it was Kubernetes to aggregate all those logs together and give me a single end point. That's why really relying on your platform to move from that single node monolithic architecture to a microservices architecture for something like machine learning is so important.

[SPONSOR MESSAGE]

[0:41:06.8] JM: Today's sponsor is Datadog, a real-time monitoring platform that unifies metrics, logs and distributed request traces from your cloud containers and orchestration software. Track the health and performance of your dynamic containers, your apps and your services with rich visualizations and machine learning-driven alerts.

Datadog’s new cluster agent streamlines data collection from large container clusters and allows you to auto scale Kubernetes workloads based on any metric you're already collecting with Datadog. To start monitoring your container clusters, sign up for a free trial today and Datadog will send you a free t-shirt. That is a free t-shirt.

Visit softwareengineeringdaily.com/datadog to get started. Thank you to Datadog. Please visit softwareengineeringdaily.com/datadog.

[INTERVIEW CONTINUED]

[0:42:10.1] JM: What about the challenge of versioning different models, or AB testing two different versions of a model at the same time? How does Kubeflow help with those set of problems?

[0:42:23.7] DA: That's a really interesting question. I think there's a – we don't want to try and tackle everything at once, but that's certainly something we've heard very specifically, because many times people will want not even just two, but many different versions of models and production to make sure that things aren't drifting and customers are always being exposed to the best that you have to offer.

Because Kubeflow uses Kubernetes to do all its core infrastructure and deployments and things like that, we're also able to use all the functionality that exists in the Kubernetes ecosystem to build rich deployment methodologies like that. That means that you can use something where that's already built in natively with something like Seldon, which offers many, many rich deployment tools, like single-arm band and multi-arm band and so on, excuse me, multi-arm band and then so on.

Or there's currently some work right now in Kubeflow to integrate with Istio, which gives you declarative ways to roll out rich, multi-endpoint traffic routing and manage the way that you route your customer’s traffic to do things like exponential rollout, or blue-green rollout and things like that.

The story that that I say over and over again is Kubeflow provides the platform to do all these things. It's really the richness of the cloud native and Kubernetes ecosystem that enables folks to do many rich things on top of Kubeflow.

[0:43:54.7] JM: Yeah. The Istio idea is – that sounds pretty promising for doing the routing control. I talked to Dan Kahn when I was in China, the guy that runs the cloud native computing foundation, or was the director of it, and I asked him what open source project he would like to see somebody start and somebody donate to the cloud native computing foundation. He said, a continuous integration tool that is Kubernetes native would be something that's really desirable.

It's surprising we haven't seen something like that quite yet. It would be really cool to see something like that and then see it adapted to machine learning, or the Kubeflow, Kubernetes machine learning model release control process. Are you surprised we haven't seen some continuous delivery tooling built natively into Kubernetes?

[0:44:54.8] DA: Yeah. I think that's interesting. I don't –

[0:44:58.0] JM: I guess it's not your area of the –

[0:44:59.8] DA: No, no, no, no. I mean, as someone who loves Kubernetes, I have a lot of thoughts around this. I think my take would be I don't ever expect – I think, there's no there's been a lot of talk about how Kubernetes is starting to get boring. That's really a great sign. What you want is the platform to be stable and reliable and it have a defined upgrade strategy and so on and so forth.

I think on top of that, you do see a lot of great tools that are built for Kubernetes to do native CICD. I will slightly disagree with fan's point. I fully expect people to continue to deploy rich CICD tools that understand Kubernetes and containers. I'm not sure I would like it to be part of Kubernetes. I just like it to be similar to Istio or Knative or who knows? All the various frameworks that are built on top of Kubernetes, to have some first-class support for Kubernetes. I think there are quite a few that do that. Weaveworks/flux is an example of that that does understand CICD and Kubernetes natively, but isn't part of the Kubernetes project. It's external.

[0:46:21.9] JM: We had a show recently with Netflix about how Netflix uses Jupiter notebooks for machine learning and just data science jobs and a ton of other things. I was blown away, because I guess Jupiter notebooks is something that had escaped my – I guess, I didn't realize how much impact they were having in the world of machine learning. Can you explain how Jupiter notebooks fit into the workflow of a machine learning engineer?

[0:46:51.6] DA: Absolutely. Jupiter has just absolutely caught fire. God bless them that the team out of Berkeley is just doing absolutely groundbreaking work and making it so easy for folks to engage and use machine learning. Jupiter though is really designed to be an exploration tool, like Visual Studio code, or any other IDE you might use.

It's designed to be the development time exploration, where you can execute something, you can look at something and see whether or not it's debugging properly and so on. It may not be quite ready for production as it stands. It's not intended to be. The Netflix folks have a wonderful project called Paper Mill, which takes your notebook and strips out all the stuff that isn't production ready, and packages it in such a way that now you can run that notebook in a more distributed way and move on.

That's really what you'll see out of platforms over and over again. You'll see people being able to use a Jupiter notebook as a way to explore the various permutations and how quickly a model converges and so on, but not really to do your production grade training, or deployment, or anything like that.

I think what you'll see is people will take those notebooks and figure out a way to compile them at scale, such that the artifacts change very little, or they change in a programmatic way when they move out to production. Jupiter as a notebook and a concept is fantastic. It's just a matter of bridging that exploration step with the real production-ready step.

[0:48:38.1] JM: Can you talk in more detail about how Jupiter notebooks, or I guess I should say Jupiter hub is built into Kubeflow?

[0:48:47.0] DA: Yeah, absolutely. What we saw with Kubeflow was that on a single cluster, people would want multiple scientists to be able to log in and do their own exploration. That's really what Kubernetes is designed to do through things like namespaces and multi-tenancy and so on.

In partnership with the folks at Jupiter, we were able to include Jupiter hub as part of Kubeflow, so now you can make a deployment on Kubernetes using Kubeflow of Jupiter hub. That makes it very easy to roll out. Then once you get inside that, you are able to select the particular version of the image or kernel or whatever has been blessed by your IT admin or so on. As a data scientist, you're now presented with a Jupiter notebook very, very quickly without having to think about dependencies, or is my version of Python cracked, or is this given the right credentials and will run inside this environment and so on.

All I need to know as a data scientist is I go to a web page, I log in and I'm presented with the familiar notebook that I would understand, even if I was running on my local laptop. I should say that Kubeflow does run inside a local laptop environments as well. When you start to get to larger clusters, particularly with precious resources like very large machines, or GPUs, you're able to use something Jupiter hub to share a cluster while still giving each data scientist their own unique environment.

[0:50:19.9] JM: Okay, so you have moved from Google working on Kubernetes and TensorFlow there to now working at Microsoft today. I think this is an interesting story of continuing work on open source projects, despite moving from one cloud provider to another. I think this is happening in an environment where we really are seeing more collaboration and I think positive communications between companies that could be cast in a certain light as being bitter rivals.

I think it is less, frankly speaking from all the coverage I do and talking to people, I think there is less bitter rivalry today and more of a realization that this is just a gigantic market and it's not like we're in a dogfight for some limited set of enterprise contracts right now. It's more the entire world is going through an adoption of cloud-related technologies, whether it is by a cloud provider or they're just modernizing their own infrastructure.

I think there's just a general feeling that this is really good for global business. You see it reflected in I think somebody like yourself where you're moving from one giant company to another, but it feels like, I don't know. My impression talking to you is you are – it's not like you have to adjust your mentality and your personality to at least to a stark a degree in going from one of these big corporations to another, which is something that would have been very different five or ten years ago, I think. Can you tell me about what it's been like to go from Google to Microsoft?

[0:52:08.3] DA: Yeah. That's such an interesting point. I think that there's no question that we're still fiercest competitors in many ways, whether or not you're Google or Amazon or Microsoft. We certainly all have opinions about how to help customers get to the cloud and revolutionize and transform their data centers.

Like you said, this really is a new world, where folks like Azure are so deeply committed to open source, that they hire real notables from the community to come in and help them guide the way that we do open source here and collaborate where it makes sense. Microsoft and Azure are one of the top contributors to Kubernetes. Obviously, we're continuing our work with Kubeflow with me here. They contributed Onyx and Onyx runtime to public use and there are contributors to PyTorch and so on. The amount of open work that's going on here at Microsoft and Azure is just absolutely tremendous.

To your point, I think that every major cloud provider has really realized that it's not like it was 10, 20, 30 years ago where every software package needs to be incompatible with every other company's software package. There's just a lot of opportunity to collaborate on common problems and then differentiate with services and tools and data centers where it makes sense.

Something near and dear to my heart around Kubernetes, that's a perfect example where people saw the opportunity, saw the need to develop a production-ready container orchestration system and said, “You know what? I don't think there's an opportunity for us to differentiate here.” Or it's actually just more beneficial to the entire industry for us to collaborate on a platform that everyone is choosing is the industry leader. If we can help that platform be better and then give them a very easy way to run on our cloud of choice, regardless of what it is, we're going to help you do that. I think you really are seeing that across the board.

There's no question that folks will offer differentiated services that plug into these open platforms, but more than anything I think with open source software today, you are seeing a very broad sense of collaboration and really solving the common problems that let us all go on to the next level.

[0:54:41.1] JM: Okay. Something that seems very strange to me about the world of machine learning is if I was – I'm not writing much software these days, but I talk to a lot of developers. I hear a lot more excitement about building things in TensorFlow, than I do about utilizing these pre-baked machine learning APIs, or these guided machine learning cloud experiences. I don't really know what the cloud providers are offering in complete detail, but I know that there's like speech APIs and text analysis APIs and video recognition APIs. I hear less excitement about these APIs, than I do around something like a TensorFlow, when it seems the APIs can be a little bit higher level and perhaps higher leverage. Do you think there is a lack of excitement relative to what people should be excited about when it comes to machine learning APIs?

[0:55:44.1] DA: Well, I like to say all the time in machine learning, it very much feels people are operating with 1970s technology. That's not to be dismissive. It's just it's very, very new. No one has even really come along and invented C yet. As great as things like TensorFlow and PyTorch and so on are, they're still very, very raw and require a lot of understanding of the various components of machine learning.

If I'm a software engineer and look, all I need is I have this audio file and the audio file has some speech in it and I just need the words out of it. Can you just give me the words? Then an API works wonderfully. There's no question about it. Even if writing my own TensorFlow model might get me, I don't know, 5% or 10% better performance, that may not be worth the additional headache in the months necessary to build and debug my own system, versus just having an API.

The reality is that I think there will always be people in a variety of different levels, because I just don't believe there will ever be an API that lets Waymo process a petabyte of video information that they get off a single car driving around. It's just too much and too complicated and impossible to unpack into the right frameworks or formats or whatever. They will always need to write their own models and likely they'll need to write many models that cascade and join together.

I think that we should look at this as an opportunity of rich excitement across the board. Your average developer more than likely will operate at an API level, whether or not it's a hosted API like the ones you described, or using machine learning APIs that are built into native languages, like ml.net that extends C-sharp, and so you don't have to think about doing complicated things. Or extensions for Python or Ruby or whatever your language of choice is.

Then there will absolutely be people doing groundbreaking research using next generation deep learning that need to build their own models. Even with them, I remain very, very hopeful that we get to a new world where no one forgets that this is TensorFlow or PyTorch or so on running under the hood. The idea that I'm going to have to figure out the right optimization function, or use atom as my tool, instead of this other one, so on and so forth. It's just crazy that we're asking people to understand the size of the RAM on their video card in order to make sure their tensors are the correct size and shape. It's just too much. These are things that long time ago in regular computer engineering, we understood that compilers were way better at and we should get there with machine learning as well.

[0:58:52.7] JM: Right. Okay. The intuition is right, that things should get much easier. Whether there's any shortcut to be taken today is an open question. Okay, so to wrap up, again you joined Microsoft recently. This this vision of making things a lot easier that you've just laid out, this is in your wheelhouse. You joined to help with the open source machine learning strategy. Tell me more about what you're going to be doing. How are you architecting a strategy that is going to help bring these things to fruition, at least from the Microsoft point of view?

[0:59:26.7] DA: Yeah. I think that it really touches on exactly the question that you had earlier. Azure does an enormous amount of machine learning. Internally, we have all these services to help with machine learning. When I say help with machine learning, I really mean the end-to-end process, all those steps and pipelines and things that we talked about earlier.

I think what we'd really like to do is help open source projects have a good home on Azure, help them based on our own experience, bring that experience to these open source projects and help them operate better. Then also ideally offer APIs and things that enrich those projects when they get here.

What I’m doing here is trying to think about what the best ways are to accomplish all of those. To help the rich ML communities do what they're already doing faster, better, easier. To help them when they come to Azure have a great experience. Then make sure that no matter what a customer uses, whether or not it's a hosted solution or an open source solution, make sure that it's easy, it's fast and they're able to get to what their business needs are very quickly.

I'm only two months in, but every time I turn a corner I find an entire new group that has been doing really groundbreaking research here and we're really excited to start bringing that out to the public soon.

[1:00:51.1] JM: David, thanks for coming back on the show. It's been really fun talking to you.

[1:00:53.7] DA: Me too. Thank you so much for having me back.

[END OF INTERVIEW]

[1:00:59.2] JM: GoCD is a continuous delivery tool created by ThoughtWorks. It’s open source and free to use and GoCD has all the features you need for continuous delivery. Model your deployment pipelines without installing any plugins. Use the value stream map to visualize your end-to-end workflow. If you use Kubernetes, GoCD is a natural fit to add continuous delivery to your project.

With GoCD running on Kubernetes, you define your build workflow and let GoCD provision and scale your infrastructure on the fly. GoCD agents use Kubernetes to scale as needed. Check out gocd.org/sedaily and learn about how you can get started. GoCD was built with the learnings of the ThoughtWorks engineering team, who have talked about building the product in previous episodes of Software Engineering Daily, and it’s great to see the continued progress on GoCD with the new Kubernetes integrations.

You can check it out for yourself at gocd.org/sedaily. Thank you so much to ThoughtWorks for being a long-time sponsor of Software Engineering Daily. We’re proud to have ThoughtWorks and GoCD as sponsors of the show.

[END]