**EPISODE 1082**

[INTRODUCTION]

**[00:00:00] JM:** Machine learning workflows have had a problem for a long time. Taking the model from the prototyping step and putting it into production is not an easy task. A data scientist who is developing a model is often working with different tools or a smaller data set or different hardware than the environment which that model will be deployed to. This problem existed at Uber just as it does at many other companies. Models were difficult to release. Iterations were complicated and collaboration between engineers could never reach a point that resembled a harmonious DevOps-like workflow. And to address these problems, Uber developed an internal system called Michelangelo.

Some of the engineers working on Michelangelo within Uber realized that there was a business opportunity in taking the Michelangelo work and turning it into a product company. Thus, tech Tecton was born. Tecton is a machine learning platform focused on solving the same problems that existed within Uber. Kevin Stumpf is the CTO at Tecton and he joins the show to talk about the machine learning problems of Uber as well as his current work at Tecton.

If you want to reach 30,000 unique engineers every day, consider sponsoring Software Engineering Daily. Whether you are hiring engineers or selling a product to engineers, Software Engineering Daily is a great place to reach talented engineers and you can send me an email, jeff@softwareengineeringdaily.com if you're curious about sponsoring the podcast, or forward it to your marketing team. We are also looking for writers and a videographer. If you're interested in working with us, you can also send me an email, jeff@softwareengineeringdaily.com.

[SPONSOR MESSAGE]

**[00:01:47] JM:** Over the last few months, I've started hearing about Retool. Every business needs internal tools, but if we’re being honest, I don't know of many engineers who really enjoy building internal tools. It can be hard to get engineering resources to build back-office applications and it’s definitely hard to get engineers excited about maintaining those back-office

 

applications. Companies like a Doordash, and Brex, and Amazon use Retool to build custom internal tools faster.

The idea is that internal tools mostly look the same. They're made out of tables, and dropdowns, and buttons, and text inputs. Retool gives you a drag-and-drop interface so engineers can build these internal UIs in hours, not days, and they can spend more time building features that customers will see. Retool connects to any database and API. For example, if you are pulling data from Postgres, you just write a SQL query. You drag a table on to the canvas.

If you want to try out Retool, you can go to retool.com/sedaily. That's R-E-T-O-O-L.com/sedaily, and you can even host Retool on-premise if you want to keep it ultra-secure. I've heard a lot of good things about Retool from engineers who I respect. So check it out at retool.com/sedaily.

[INTERVIEW]

**[00:03:24] JM:** Kevin, welcome to the show.

**[00:03:25] KS:** Thank you. Thanks for having me.

**[00:03:27] JM:** You work on Tecton. Tecton is a company that was born out of the machine learning problems and the solutions that you encountered at Uber. Tell me about the challenges that you found at Uber in making machine learning usable.

**[00:03:41] KS:** Yeah. In about 2015 when we started the ML platform called Michelangelo at Uber, there were a lot of dispirited efforts to get ML into production. Different data scientists were using various different frameworks and it was taking a really, really long time to access and find the right data, train the model and then eventually get it into production. As a result of a lot of the different productionization challenges, we built Michelangelo over there, which is the centralized platform that all the data scientists and software engineers could come to to go through the entire ML workflow end-to-end from training the model to deploying the model, serving it, monitoring it, etc., and to basically ease all those manual painful steps that you otherwise would have to go through if it didn't have the centralized platform. And I'm happy to dive into any of those if you'd be curious.

**[00:04:32] JM:** Well, one acute problem that we've certainly touched on previous episodes is that of the machine learning model creation process and then deployment and production processes, and it's often difficult to get these two things aligned because you have "data scientists" that help you get the V1 of the model built, but the DevOps workflow that you would want out of the iterative production and release process is not very cleanly defined even today. How did that affect machine learning workflows at Uber?

**[00:05:10] KS:** Yeah, that's exactly right. Initially, the way it would work is that – And that's how a lot of organizations still do it today. You have your data scientists that work in Jupyter Notebooks. They're pulling in raw data and they're transforming this raw data using their Python code and different Python libraries to turn it into features, then they train their model within their Jupyter Notebook and then they back test it and then they've got something that they're happy with, but it's not possible for them to then get this model and the data that they've trained and prepared in the Jupyter notebook into production, especially if you want to use it for really higher scale and low latency use cases. That's where they now have to partner up with data engineers with ML engineers who basically now take over their Jupyter Notebooks, look line by line at the code that the data scientists has implemented to fetch the raw data, transform it into features. Now, train a model against it and they would basically, from scratch, re-implement a lot of this transformation logic oftentimes even retrain the model using a different framework that's better suited for high scale production use cases and basically go through the entire workflow again.

What you can imagine is, in this world, the data scientist is empowered and has to talk to a lot of different teams, to a lot of different people until a model eventually makes it in production. Given that machine learning is a really iterative workflow where we really want to get something in production and tested really rapidly and quickly, but oftentimes can grind the data science productivity very quickly to a halt.

**[00:06:43] JM:** It seems that that issue has been around and it's existed at many, many companies. I imagine it has been a problem at Google before. Why hasn't this been solved? If it's been a problem at Google, it's been a problem at Facebook, we've had machine learning at these place for a long time, why is there still no workflow for easily fixing that productionization and release process?

**[00:07:09] KS:** If you look at the really advanced tech companies like Google, Facebook, Twitter, Airbnb, Uber and a handful of others, they actually have implemented solutions in order to solve exactly these types of problem. But those are pretty hard problems and oftentimes the solution also needs to be fairly deeply integrated with the existing stack of a company. You just saw all these different efforts coming up over the last couple of years within the different companies who then stood up those pretty heavyweight platforms that bring the DevOps principles and the DevOps workflow to the data scientists, to the software engineer who want to train ML models and get them into production.

But when you look at the open source world and/or generally outside of the large tech companies, what you do find by now is this trend towards ML ops and you do find that there is now new tooling and better platforms that allow you to more easily train models, manage those models, deploy those models and then serve those models even in production. So there has been a trend into this direction, but what we found, what's been largely missing to a big extent, is a system that complements this model workflow itself with a data workflow. Actually, preparing the data, delivering the data to the data scientists who trains the model and then delivering the data to the models that's actually running in production. Right now, half of those workflow, basically, everything that's related to the model itself, has gotten much, much, much better over the last couple of years, but the data later that the model layer depends on has not gotten as much attention yet.

**[00:08:50] JM:** I want to talk through some other issues, then we can start to talk about what you've actually built. There's this problem of a model siloization. I can imagine wanting to take a model for one application and easily port it to a similar application. When you are at Uber, was it possible to reuse models that were built for one purpose for another purpose?

**[00:09:14] KS:** Yeah, good question. Typically what he would find it is not that the models themselves are necessarily reused for other purposes, but what you do find is that the features of the underlying data that an ML model is being trained on and makes predictions against, that those would be reused very widely and then you would find the different business use cases that could benefit from an ML model. They share a pretty large subset of features with other models, but it's not 100% identically the exact same feature set and it's not the exact same

model training algorithm with the exact same hyperparameters. Sharing happens much more lower at the data levels.

**[00:09:50] JM:** Okay, so data sharing, rather than model sharing. Uber's Michelangelo project is at the roots of your company, Tecton. While you're at Uber, you worked on Michelangelo. Tell me about the architecture for Michelangelo.

**[00:10:08] KS:** Yes. Michelangelo is an end-to-end platform as I briefly mentioned that the data scientist and engineer comes to to select the data, turn it into features, train the model, deploy it to production and then run it in production. Under the hood, the system, the backend is mostly built on top of Java. All the data processing system is built with Java and the data processing itself is actually on the backside mostly happening using Spark that overruns itself. Then on the stream processing, Uber, or Michelangelo, uses Flink to process streaming jobs from Kafka data sources.

Then we would use Spark to train models under the hood, turn it into a Spark MLlib model and then we'd have a patched version of Spark that we would use for online surveying where we would have a Java service, the online prediction service, which is able to load the Spark model into memory and then serve really high scale and low latency predictions to the user.

Outside of that, Michelangelo under the hood also has a feature store, which is the system that serves historical feature data to training processes, specifically the spark training processes that I just mentioned, and that serves the data to the real-time serving component where the model is making real-time predictions in production. That feature store under the hood stores and serves feature data at low latency and high scale from Cassandra, and then it serves the historical batch data for training purposes from HDFS.

**[00:11:47] JM:** Michelangelo was built on top of Uber's data infrastructure. As you said, you've got Kafka. You got HDFS, Spark, Flink. There're a lot of pieces of data infrastructure there and we're going to be talking about taking Michelangelo and turning it into the company, Tecton. I'm wondering if you are trying to solve the data infrastructure problem as well, because obviously data infrastructure is a big complex, hairy problem. Are you trying to solve that as well or you

focused completely on the machine learning infrastructure problem? Is there a clean separation of concerns there?

**[00:12:24] KS:** We specifically focus on the data problems for machine learning and we explicitly do not focus on the modeling problems. Tecton does not help you, it does not train a model for you. It doesn't manage the model artifacts and it doesn't deploy the model itself to production either. What it does do is it does focus on the many data problems that you encounter when you try to train a model, when you try to discover features that you have available in your company when you want to make those features available in production. That separation of concerns we're focusing on. Nothing about the model, but everything about the data, and data itself specifically focused on the machine learning problems and not any other generic data problems and data infrastructure problems they would encounter in an organization.

**[00:13:09] JM:** Okay. What problem specifically is Michelangelo going to solve for me? If I am a company that has machine learning, I've been using the same old techniques of designing my models and productionizing them, what problems is Michelangelo solving?

**[00:13:27] KS:** Do you mean Michelangelo or Tecton?

**[00:13:29] JM:** I should say Tecton. Sure.

**[00:13:32] KS:** Tecton, at the end of the day, it's a data platform for machine learning. It gives you the ability to centrally manage all of your organization's features that are relevant for machine learning purposes. Then it gives you the ability to provide those feature values to the data scientist who trains a model to the automated training pipeline that retrains a model on a cron job basis or any other trigger, and then it provides those feature values to a model that is running in production that needs these feature values at high scale and low latency.

It also allows data scientists to then share and reuse features across different use cases. It allows you to actually define features in code rather than in a Jupyter Notebook that may not be centrally managed and tracked by the organization. It allows you to centrally, in a git repository, manage the metadata and the definition of your organization's features. What does the data

transformation actually look like that turns your raw data into a feature that can be consumed by a model or by a training algorithm?

We give you that system where you can define those features as code, check them into a git repository and then using a normal or a typical CICD pipeline, deploy those updated feature definitions to Tecton, which then under the hood spins up, if necessary, data processing jobs that now run those transformations that you've defined and makes the output of those jobs those feature values available for your training systems as well as your production systems where you model is running in production.

**[00:15:08] JM:** You've mentioned this word feature a few times. Could you just define what a feature is?

**[00:15:14] KS:** Yeah, definitely. Features are really the lifeblood of ML applications. They're typically highly curated data or information. It's almost like this hierarchy from you start with data and then you clean it and you process it and then you turn it into information. You typically get just much higher density quality of information out of data, which you turn into a feature. At the end of the day, it's still data. It's still information, but just of a different level of quality, of a different level of expectations. You can trust it. It's cleaned, it's excluded outliers. It oftentimes aggregates the raw data to produce the final feature value.

To give you a more specific example, imagine you had an event stream that tracks the number of Uber Eats orders that have been coming in. A feature would now be, say, the trailing 30-minute order account for a given restaurant, or the restaurant, or the cuisine that a certain user is most likely to order again in the future. Basically, anything that takes raw data, cleans it, applies human intuition, human understanding to it to then turn it into an extremely high quality signals of information that ML algorithms are able to then train on and detect patterns and to eventually make a business use case prediction.

**[00:16:38] JM:** Okay. What is required to manage the data that we're going to be using as features?

**[00:16:46] KS:** Typically, you need to do two things. You need to be able to access the raw data. You need to process it according to the processing definition of the user, where the user would say, “Hey, these are the steps that I want you to go through in order to clean the data, remove any outliers, aggregate it, and then turn it eventually into this highly curated feature and the feature values that I then want to make available for my training systems or my serving systems.

**[00:17:16] JM:** What kinds of infrastructure do we need for storing the features once they've been refined?

**[00:17:24] KS:** If you have production use cases that require high scale and low latency serving, then you typically need a keyvalue store that's really optimized for high scale and low latency. Tecton, under the hood, when we’re deploying in AWS environments, we use DynamoDB to store and serve those feature values from. Then we use a data lake under the hood to S3 on AWS deployments where we store all the historical feature values that are extremely relevant and important if you train models on historical data ranges.

[SPONSOR MESSAGE]

**[00:18:03] JM:** There are two ways to add analytics to your application, you can build them yourself with basic charts and dashboards using free open source charting libraries, or you can use a comprehensive analytics platform from a partner that you trust. If you've tried to build it yourself, you know that free actually is not so free. There are hidden costs like time, and maintenance, and technical debt, and those hidden costs can really add up.

Check out Logi Analytics. Logi Analytics is developer grade embedded analytics solutions and they make it easy to create branded dashboards and report the scale within your own application. You can stop wasting time piecing together analytics and allow yourself to focus on your core application. You can go to logianalytics.com/sedaily and you can get a demo to see what is possible with Logi today. Go to logianalytics.com/sedaily. That L-O-G-I-analytics.com/sedaily.

[INTERVIEW CONTINUED]

**[00:19:17] JM:** Let me see if I can regurgitate what I understand to be features and how those features are used. A machine learning model is going to be this set of multidimensional data points. You've got a bunch of data points that are sitting throughout some multidimensional space. When you train a machine learning model, you're drawing some kind of gradient across those different data points. You're training a gradient across those data points. Those data points each have their features associated with them. Once you have that trained model, you have a trained gradient, then whenever some query comes in, that's a new set of data points. You can figure out where that set of data points fits relative to that gradient that you have defined. Am I explaining the features like how features are seen in that modeling space correctly?

**[00:20:12] KS:** I think so on. One thing I would add here is that, really, those different points that you've talked about that are in this space around which you draw a gradient or some curve, typically, the different dimensions of those data points are the features. If you just imagine like an X-Y Cartesian system, then you would have just for any given data point in there, you would just have two features, the X value and the Y value. But typically, you have data points of much higher dimensionality than just two dimensions as in this case, but it's exactly right. Every single row in your training dataset would map to one of those points in this space that you draw the gradient around, and then every single point can be of N dimensionality, and N in this case would map to the N different features on which you train the model.

**[00:21:04] JM:** In the case of a food delivery recommendation system, let's say the user has some collection of preferences that you figured out over time and your machine learning backend might need to answer a query that the query is based on these preferences that the user has issued in the past. Based on those features, we want to find the recommendations to return to them. Assuming this is a a reasonable example, could you describe how that feature query would be answered by the Tecton feature storage system?

**[00:21:41] KS:** Yeah, definitely. I'll take a step back, and let's say there is a feature that is defined, which is, "Hey, over the last one week and over the last one day, what is the type of cuisine?" Say, Thai food or Mediterranean that a given user has most frequently purchased.

Those may be to features that are extremely relevant to predict what type of restaurants you want to recommend to a user now.

In production, when the Uber Eats app wants to make a recommendation, wants to come up with a list of restaurants that the user is most likely to choose from, what that system would do together with Tecton is it would know that, “Hey, here is user Jeff. Jeff's user ID is XYZ, and I now need these two or I need a set up feature values for Jeff that I can make a prediction against. Specifically, I want to have, “Hey, what’s the most favored cuisine that this user has ordered over the last one day and the last one week?” It then sends a query to Tecton and says, “Hey, give me these two feature values as of right now for user X, Y, Z Jeff.” Then Tecton responds with a feature vector that contains, say, in the last one day your most frequently ordered Thai. In the last one week, Mediterranean, and now you've got this feature vector that you can one-to-one pass into your model that now makes a prediction and recommends what list of restaurants do you now want to surface to the user. Specifically, the way you would actually do this is that you would have a list of restaurants and you would rank them according to a score that this model associates to the payer of Jeff in any given restaurant based on those feature values.

**[00:23:22] JM:** Okay. Tell me a little bit more about how the feature storage system would be used in a query for the purpose of this recommendation system.

**[00:23:35] KS:** In this case you would have, say, a Java backend that is making the prediction that is then served to the Uber Eats app, and this Java service is owned by the user, by the Tecton user, who is responsible for the backend service. This one would now integrate with Tecton and make a GRPC request to Tecton’s feature serving API and literally just request those feature values for your user ID and then receive the feature values and then you and your Java service that makes a prediction would just pass feature values to the model, make the prediction and then serve it back to the application. In production, that interface to Tecton is a very simple one. It’s really just a GRPC request, “Hey, give me feature values. The most recently computed feature values for a given user and for a given set of features.”

**[00:24:26] JM:** Let’s actually go back a little bit further. How is the training process done, and like where does the training take place relative to the serving layer?

**[00:24:39] KS:** The training happens typically much earlier in the process before you have your model deployed in production, and here you would have a data scientists who's in a Jupyter Notebook or a Databricks notebook or any other environment that they typically process their data and train their models. Here, what they would do or the way they would interact with Tecton is they would pip install the Tecton Python SDK and then import their Python SDK and then ask Tecton to provide historical feature values. Again, for Jeff, give me all the cuisines he's most frequently ordered from in the last one year and always give me the most frequently purchased cuisine over a one week period and a one day period for any given point in time in the last one year.

Then from the Tecton SDK, you would now get a data frame back that you can now start training your model on. Now, the data scientist is back on their own. They've got the training dataset. They can train the model. They can evaluate it, and then eventually they hopefully have a model that they're happy with, and then they would go back to Tecton and say, "Hey, these features, I know they're of really high quality. I now want to make them available in production." You literally basically just hit a button, so to say, through the CICD pipeline to indicate to Tecton that those feature values should now be made available in production for low latency and high scale serving.

**[00:26:07] JM:** This process of selecting features, how does that take place? How do you identify which features are actually going to provide signal to a model?

**[00:26:18] KS:** Yeah, that's a good question. Honestly, it's a lot of human intuition that goes in here where you would imagine, as a data scientist, "Hey, this is the type of signal that I'd imagine could be really correlated with the outcome that I'm trying to predict." This is typically when you start with, and you see how far that gets you.

Depending on the amount of data that you have available and the number of features that you can choose from, it may even be scalable to try a kitchen sink approach and just throw everything at the problem and let the model training algorithm figure it out and try various different permutations and subset of all the features that you have available to train a model against and then see what the model performance looks like.

**[00:27:00] JM:** I can imagine that over time you would want to change or want to research more what impacts different features would have on different models, and this kind illustrates the iterative process we're going to have with developing our machine learning systems. Tell me more about the feature selection process over time and what are some of the problems that can emerge from a poorly managed feature selection process?

**[00:27:27] KS:** Yeah, definitely. A couple of the problems that can definitely do occur are that you may have actually features as part of your – That you use in your model training system or your model that actually degrade the performance of your model, or they don't actually provide any signal to what you're trying to predict, but you're not picking up on the fact that they're providing very limited signal and you're still behind the scenes productionizing this feature and making it available to the model as it's trying to make predictions. So you're basically just unnecessarily burning money because it does cost money to process the data and the raw data and turn into features.

You may also oftentimes just train this model ones based on a set of features then put the model into production and basically forget about it and not continue to monitor the model as it's in production, not monitor the features. What oftentimes can happen is that the world actually changes and that your model could do much, much better if you retrain the model in an updated set of features. Maybe you change the features entirely or you just really retrain the model based on the most recent feature values. Oftentimes, that just doesn't happen. Models are in production, they're making poor predictions. They could make good predictions if you retrain them, but you're missing it, or there is new data that you're actually collecting in the company now from a new completely new data source that could provide an extremely high value signal to the model and you're just not picking up on that fact and you're not incorporating it into your model. So you're leaving model performance unnecessarily on the table.

**[00:29:03] JM:** Could we talk a little bit more about the usage of Tecton? Let's say I'm a brand-new company or I'm a company that's built some machine learning models and I'm going to integrate with Tecton and I've already got a data lake. I've already got a data warehousing system. Maybe I've already got some TensorFlow models I've built. What am I doing to integrate with Tecton?

**[00:29:25] KS:** Yeah. To integrate with us, you can either choose Tecton's SaaS deployment model or Tecton's VPC deployment model. The Tecton VPC deploy model, you have Tecton just deploy the entire Tecton stack into your, say, AWS account and then you need to connect it to your existing infrastructure and then what you would do is you now will define a feature repository, which defines the various different features that you want to make available to the data scientists. You can either tell Tecton, "Hey, I already got it. I already pre-computed and I'm already computing these feature values. Please just ingest them into Tecton. I just want to use Tecton to make them centrally available to all the different data scientists. I also want to use Tecton to make those feature values available for low latency serving a production."

You would then just point Tecton in this feature repository, add those data sources that you have available, your data lake, your Kafka streams, wherever you have your data. Then if you want to now use those feature values for training purposes or for serving purposes, you would then, as I mentioned earlier, pip install the Tecton SDK. Configure it such that it knows where to find the Tecton backend and then use the SDK to actually fetch feature values for training purposes. Then in your production system where you're actually making the real-time predictions, that's where you would integrate with our GRPC endpoint to fetch feature values for low latency serving.

**[00:30:56] JM:** The GRPC endpoint is between the Tecton-like API? The API backend and the serving layer? I guess there's like an endpoint that you might be hitting as like a mobile app is hitting the Tecton endpoint, and then that endpoint is talking over GRPC to this feature server?

**[00:31:20] KS:** Yeah. Typically, it wouldn't be the mobile app directly that talks to Tecton, but the mobile app would talk to the backend service of the company that provides the mobile app, and then that backend service would now integrate with Tecton's GRPC service to fetch the feature values and then make a prediction that they then conserve back to the mobile app.

**[00:31:38] JM:** Right. W what is the architecture of that feature store system?

**[00:31:43] KS:** Yeah. The architecture of Tecton as a whole really is I would separate it into the control plane and the data plane. In the control plane, we have of Postgres database that holds

all of the different feature definitions and their metadata. What are actually the features that I have on my company? What are the entities that are associated with? Who's responsible for them? What is the data transformation look like?

Then we have an orchestrator, which manages those feature definitions and is aware of, “Hey, which ones of those features do I need to make available only for training? Which ones do I need to make available for serving?” It’s basically the control tower of Tecton that is aware of, “Hey, what do my users expect me to be able to deliver for training and serving?” Then it’s responsible for spinning up data processing jobs to now run these data pipelines and make those feature values available. Then there is a backend service, which is responsible to support the DevOps workflow of Tecton, whereas I mentioned, we believe that features should be treated as code. The feature definitions are all defined on files in a git backed repository and you would use Tecton’s CLI, similar to Terraform, apply Tecton, apply those feature definitions to Tecton’s backend. So we've got a backend service that handles that.

Then on the data plane, we talked already about the storage of the features. We've got the online future store and the offline future store. The online feature has to be like keyvalue store. In our case, it’s Dynamo and AWS. The offline one is a data lake. In our case, it’s S3. Then the data plane also consists of these materialization jobs that actually run the data processing according to the feature definition. Those typically most cases today with Tecton run on spark, either your Databricks Spark or EMR, and they make the feature values available and store them in the offline and online feature store. ‘

Then we've got the feature service, which exposes this GRPC interface that we've talked about a few times, which makes those feature values available to the model and it can optionally also run online or real-time transformations on top of the data that it's providing to the user. Then the main interfaces that you really have to interact with those different components of the Tecton architecture are the web UI to actually browse and find and discover new features. Then this Python SDK that I mentioned, which serves the metadata, but also the feature values that it would use for training purposes to the data scientists in their notebook, the CLI that you used to rollout via like CICD your future definitions to Tecton, and then the feature service GRPC interface.

**[00:34:24] JM:** Forgive me if you've covered this already, but the process of indexing all of those potential features in a company's data. If we're talking Uber, we've got riders, drivers, cars. We've got all these different entities and then in each entity we've got things like the color of the car and make and model of the car. In each of the drivers, we've got the name, age, driving history, number of rides that they've done. There's like so values and these might be in heterogeneous data sources. I can imagine, integrating with this vast heterogeneity of data and indexing all of these things as features, it's kind of a big data normalization problem, right?

**[00:35:15] KS:** Yeah. Data normalization definitely does come in here and you would typically also not bring all of an organization's data into something like a future store. You would really only bring features into Tecto that you believe are beneficial to your ML systems. There is a vast amount of data in an organization that typically has nothing to do with ML and the no ML system would ever be benefiting from. But then you're right. Once you've discovered, "Hey, this is the set of raw data that I want to turn into features." Now, it does become normalization problem, then also a feature organization problem. How do you even now make those features later on discoverable by a data scientist? How do you help them find the right ones for the right business use case? How do you associate a feature with, say, an entity like a rider or a driver or a car that makes it easier to discover it? How do you associate it with a team to make it clear who actually owns a feature? Who's responsible for it when it breaks? Or how do you otherwise group it in a way where you can actually define scopes? Because some features, you may not want to share with everyone else in the company. They may just be experimental features or that may only be usable by a subset of an organization.

[SPONSOR MESSAGE]

**[00:36:37] JM:** Vettery makes it easier to find a job. If you are listening to this podcast, you are probably serious about software. You are continually learning and updating your skills, which means you are staying competitive in the job market. Vettery is for people like you. Vettery is an online hiring marketplace that connects highly qualified workers with top companies. Workers and companies on the platform are vetted, and this vetting process keeps the whole market high-quality.

Access is exclusive and you can apply to find a job through Vettery by going to vettery.com/sedaily. That's V-E-T-T-E-R-Y.com/sedaily. Once you are accepted to Vettery, you have access to a modern hiring process. You can set preferences for location, experience level, salary requirements and other parameters so that you only get job opportunities that appeal to you. If you have the right skills, you have access to a better hiring process. You have access to Vettery. So check out vetter.com/sedaily and get $300 signup bonus if you accept a job through Vettery. Vettery is changing the way that people hire and the way that people get hired. Check out vetter.com/sedaily an get a $300 signup bonus if you accept a job through Vettery.

Thanks to Vettery for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

**[00:38:14] JM:** Let's talk a little bit more about the features. The implementation of the feature store, with Michelangelo, there was a dual database solution. You had these offline and online access to features. I think you mentioned that for Tecton as well, you have the offline features that you store in S3 in the online features you store in Postgres, I believe?

**[00:38:36] KS:** In Dynamo. Yeah.

**[00:38:38] JM:** In Dynamo. Right. Okay. Can you tell me more about like the – Actually, this is pretty easy. The offline line feature store, those are just features that are not currently in use in the model that you've deployed?

**[00:38:50] KS:** The offline feature, what actually typically hold the historical states, the historical values of a feature. You need this, because often times when you train an ML model, you want to time travel. You want to know, "Hey, what is the feature value look like at any given point of time in the past?" That is vastly more information that you need to store and make available than you would ever need to make available in production. Because in production, when you have these models that make a prediction right now, you don't care what the future value looked like two years ago. You only care about what does the feature look like right now. The most recent feature values are made available in Dynamo and then the offline feature store stores

way larger quantities of data because it makes all the different features available there and it stores their historical values so that you can fetch those and train your models on them.

**[00:39:42] JM:** Okay. I see. The offline store is for historical values the you might use for retraining the model. The online store is just for the ad hoc query of the model.

**[00:39:52] KS:** Yeah, you got it.

**[00:39:52] JM:** Okay. I understand it. As far is interacting with the actual data that is in your data lake, are you using Spark, or Flink, or how is the data from your data lake being pulled in for training?

**[00:40:10] KS:** Yes. We use spark. Typically, you would use Tecton's SDK actually in a notebook environment in which you are sitting on top of a Spark driver where you would typically already be using PySpark to interact with a Spark cluster, say, your Databricks notebook, or an EMR notebook, or whatever it is. Then if you now import Tecton, Tecton now just actually leverages the Spark context that you work with as part of your PySpark code to under the hood make those feature values available to you for training in the form actually of just a Spark data frame that you're ready familiar with if you were just manually writing PySpark code to generate Spark data frames.

**[00:40:53] JM:** Okay. What about TensorFlow? Does TensorFlow fit into this workflow for training your models?

**[00:41:00] KS:** It does.  Yes. TensorFlow would typically be the next step. You would first use Tecton to fetch the feature values and generate a data frame on which you want to train a model. Now you would go over to TensorFlow to actually run your training and produce a model artifact and then use that model artifact to deploy it, say, to TensorFlow serving or any other system that you use to serve your model.

**[00:41:27] JM:** Got it. Okay. What about – After I've trained the first version of my model and I want to ensure that it's working properly. I can do monitoring. I can monitor the features and see how they're changing over time. Can you explain what feature monitoring is and why it's useful?

**[00:41:47] KS:** Yes, definitely. If you have a model that's running in production, it's typically not the model itself that breaks. We're pretty good now at running microsservices and knowing when they're down. However, what does breakthrough with ML is the data. The features may actually be breaking. So your model may just go off the deep end now because it's now seeing feature values that it's never seen before that's never drained on before, and there a variety of different things that can go wrong. It's very easy that, for instance, there is a complete data outage where some upstream raw data source is just not producing any feature or any raw data anymore that your model depends on, or it may actually produce data, but the data may be completely wrong. It may not match the schema expectations anymore and look completely different and looks so different that the models can't handle it and predict something that doesn't make any sense. Or, as I mentioned once before, it's also very often the case that the world just changes and you would then notice that features start to drift slightly or more aggressively, but not because anything is really wrong, anything is broken. The data is correct. It's clean. It's accurate, but the world has just changed and your model, in this case, really just would need to be retrained in order to now pick up on the new state of the world.

**[00:43:07] JM:** You monitor the features through just looking for dramatic outliers or do you just look for some deviation over time, some kind of drift of the query direction?

**[00:43:21] KS:** Yes. You would typically define the expectations of a feature, and like what is the type that I'm expecting? What's the range of values that I would expect for a numeric value? Then it's fairly easy to pick up on that and say, "Hey, now, suddenly, the value is not within the expected range anymore," or it's easy then also for us to say, "Hey, I typically see a thousand data points per minute coming from this data source. Right now, I see none. So something's probably off," or you can notice that, yes, now the distribution, the statistical distribution of your features start to slowly look different, or you can very easily just imagine that the min of like a normal distribution is now shifting or the standard deviation of it is starting to shift and start to – Very interestingly, actually in production now, start to look very different to what those statistics looked like in the training dataset that you originally trained on. That's really what you're curious about as a data scientist. When do those statistics on features change in production to an extent that they are very different to what I trained my model on in the first place?

**[00:44:26] KS:** With Michelangelo, there was a domain-specific language for interacting with features. Does Tecton also have a DSL?

**[00:44:35] KS:** We do indeed. Yes. If you look at features, there is oftentimes a tradeoff between, say, flexibility and ease of use, if you will. We do support a very simple to use DSL for very common feature types. Say, time window aggregations, like trailing 24-hour [inaudible 00:44:56] and stuff like that make it very easy to onboard those types of features, but the DSL is limited.

If you can't fit your use case into that DSL, you can just describe your feature transformations using general PySpark code that Tecton can then run on your behalf, or if you really want to, you can also just run those feature transformations outside of Tecton and make the pre-computed feature values available to Tecton where Tecton does run any transformation for you. You're then covering the spectrum off really super easy to use and not so flexible to not super easy to use, but very flexible. Then for online transformations, for features that you're actually calculating at prediction time and that you're not pre-computing, Tecton allows you to describe those types of feature transformations simply using Python.

**[00:45:48] JM:** We've been talking about the high-level and low-level issues of machine learning operations, and now that you've been out of Uber, you've been developing your own company and you've been interacting with a lot of other companies, I'm sure you have some customers. Tell me about what you've learned about the broader landscape, the broader market of companies that are dealing with various machine learning operational issues, DevOps for machine learning? What have you learned from that class of companies and problems?

**[00:46:24] KS:** Yeah, definitely. We have learned that this need for DevOps for data really and DevOps for machine learning is very prevalent and a lot of companies are struggling with it. You do see some good tooling out there that helps you with some of the DevOps aspects for data, for instance, DataOps and the folks at DataKitchen do a really, really good job with that stuff over there. But, generally, we still need to do more in order to enable DevOps for data along the entire stack, and then not just on the data side, but ML specifically, enabling DevOps truly end-to-end for the model development and model deployment workflow. That still needs to be handled much more seamlessly end-to-end where you're not just solving the model problem as I

mentioned very earlier in the podcast, which you can to a very good extent now do with good ML ops platforms, but you still need good systems to solve the data problems and provide these DevOps workflows on the data layer for machine learning.

**[00:47:35] JM:** What about scalability? Are there any scalability issues with making machine learning operational?

**[00:47:43] KS:** There are technical and then organizational scaling issues. On the organizational scaling issues, it's typically – Like it's fairly easy to get ML, to get up and running with ML if you just have a small team, you have a handful of data scientists. It's very easy for them to talk to each other and know what models they've trained, know what features they have. But eventually, once you have tens of data scientists, especially data scientists, they are disparate across different organizations in a company or different teams in the organization, then that becomes very, very hard to scale and that's typically where these centralized platforms that provide best practices in some sort of standardization can really help you scale those data science efforts.

Then on the technical side, you do again need a central essential platform to help you scale these operations, because if you, say, didn't have a system like Tecton or any other data platform for ML that helps you with the DevOps type problems of getting your data into production, then you would typically do as you would just manually hand-create different data pipelines over and over again. They wouldn't be centrally managed and you would just have disparate data pipelines that somebody somehow needs to manually monitor and somehow keep an eye on to make sure that they're not going off the deep end, and that becomes very quickly very, very unscalable too.

Then, of course, if you talk about operational ML or ML that's running a production at very, very high scale and with low latency requirements, then it's is extremely important that your system is built in a way where it can scale out horizontally fairly easily and it's build to support these high scale and low latency types of use cases.

**[00:49:28] JM:** Of course, right. The difference between operational machine learning and analytical machine learning, like I think of operational is more the dashboard use case, whereas

analytical machine learning is more of an offline and I guess the query sizes is often smaller or less rapid, right? Could you just distinguish between operational and analytical machine learning a little bit more?

**[00:49:52] KS:** Yes, definitely. Yeah. When we talk about analytical ML or analytical data analysis, we really think more about BI, data analysts, who are manually analyzing data oftentimes still large batches of data. They ride SQL query that process a lot of data to generate insights. Oftentimes, they're manually generated insights or they actually do use machine learning to guide them a little bit in the process where, say, you want to make churn predictions for customers or predict who do things like predictive maintenance. That's where ML models can still help the business analyst.

Also on the analytical side, typically, that does feed into the dashboards that does feeds into the reports that are nightly or weekly generated and that help business owners make decisions about the company about the different products, but it's all typically batch. It's typically mostly human decision-making and it doesn't have low latency requirements.

On the operational ML side, it's typically automated decisions, automated decisions that do impact the customer experience like an Uber Eats app or does that automate business processes that need to happen very rapidly very quickly in a way where a human could never make those types of decisions and it's a scale and at latency requirements where you just need machines to make these automated types of decisions.

**[00:51:18] JM:** Those workloads, are they so distinct that you need different architectures or different kinds of frameworks for addressing operational machine learning versus analytical machine learning?

**[00:51:30] KS:** Yes. There is a good set of problems that you only have in the operational ML world where you have models out in production, high scale, low latency, and that's where you need different components in order to cater to these types of use cases. Again, you need something like a DynamoDB or a keyvalue store that can actually serve at low latency and at high scale and can make available only the most recent future values. You typically wouldn't need a component like that for real-time serving if all you care about is analytical ML.

That said, there is still a good set of shared components. Even with analytical ML, you still need or you still typically want a central repository that you can go to that you can look at where you can discover features or you can define or discover metrics that are useful for your analytical ML use cases. You just don't need this ability typically to productionize those feature values for low latency serving and you don't need the real-time monitoring either. Of course, that comes with that.

**[00:52:37] JM:** All right. Well, as we begin to wrap up, we really focused early on in this problem of collaboration between the machine learning development team and the machine learning release team. Could you just revisit that problem? Explain why Tecton solves the problem of throwing the machine learning model over the wall.

**[00:52:58] KS:** Yes, definitely. Tecton does empower the data scientists to get their features into production end-to-end without having to hand their work over to an ML engineer who now has to re-implement their future definitions. With Tecton, as a data scientist, you know that if I define a feature in Tecton, it's production-ready. It's really as simple as basically clicking a button to get this feature all the way productionized, and I don't have to beg somebody else for their attention to actually re-implement my work so that I eventually can use it in production. It's all about this empowerment of the data scientist to have end-to-end ownership over their work and not be bogged down by having to go through too many slow human interactions.

**[00:53:49] JM:** Okay. Well, we're pretty much at the end of our time. Can you just share anymore lessons about taking an open source project and putting it into a company? I mean, this has been quite a journey for you, I imagine. Just give me any lessons and tell me some quick anecdotes about turning an in-company open source project into a company.

**[00:54:16] KS:** Yeah. Michelangelo is actually not open source.

**[00:54:20] JM:** Oh! Right. Right. Right. Right. Okay.

**[00:54:21] KS:** But it's build with a lot of open source components under the hood, like Spark, Cassandra and a bunch of others, but we did back in the day look closely at, "Hey, should we try

to open source in one way or another?” But we ended up not doing it back in the day. I don't have too many lessons to share in that.

**[00:54:38] JM:** Okay. You basically built everything. You rebuilt everything from scratch for Tecton.

**[00:54:42] KS:** Correct. Exactly. Yes.

**[00:54:42] JM:** Right. Right. Right. Okay, cool. Well, Kevin, anything else you want to add about building the company or lessons from machine learning?

**[00:54:49] KS:** We got more lessons about machine learning that y9ou can find on tecton.ai. We’re of course also hiring. If anybody's interested in helping enterprises use machine learning at scale in production, I’d love it if they reach out to me.

**[00:55:03] JM:** Cool. Kevin, well, thank you very much for coming on the show. It’s been pleasure.

**[00:55:06] KS:** Thank you.

[END]