# EPISODE 519

[INTRODUCTION]

**[0:00:00.3] JM:** Pinterest is a visual feed of ideas, products, clothing and recipes. Millions of users browse Pinterest to find images and text that are tailored to their interests. Like most companies, Pinterest started with a large monolithic application that served all our requests. As Pinterest's engineering resources expanded, some of the architecture was broken up into microservices and dockrized, which made the system easier to reason about. To serve users with better feeds, Pinterest built a machine learning pipeline using Kafka, Spark and Presto. User events are generated from the frontend logged on to Kafka and aggregated to build machine learning models. These models are deployed into Docker containers much like the production microservices are.

Kinnary Jangla is a senior software engineer at Pinterest and she joins the show to talk about her experiences at the company; breaking up the monolith, architecting a machine learning pipeline and deploying those models into production. Kinnary is also writing a book on the topic and she's going to speaking at Strata Data Conference in San Jose, March 5th through 8th. I'm also going to be there, just walking around, hanging out and seeing talks. So if you're going to be attending the Strata Data Conference, let me know. You can also get a 20% discount on a ticket by entering the code PCSED.

With that, let's get to this episode of Software Engineering Daily.

[SPONSOR MESSAGE]

**[0:01:38.3] JM:** If you are on call and you get paged at 2 a.m., are you sure you have all the data you need at your fingertips? Are you worried that you're going to be surprised by things that you missed, errors or even security vulnerabilities because you don't have the right visibility into your application? You shouldn't be worried. You have worked hard to build an amazing modern application for your customers. You've been worrying over the details and dotting every I and crossing every T. You deserve an analytics tool that was built to those same standards, an analytics tool that will be there for you when you needed the most.

Sumo Logic is a cloud native machine data analytics service that helps you run and secure your modern application. If you are feeling the pain of managing your own log, event and performance metrics data, check out sumologic.com/sedaily.

Even if you have your tools already, it's worth checking out Sumo Logic and seeing if you can leverage your data even more effectively with real-time dashboards and monitoring and improved observability. To improve the uptime of your application and keep your day-to-day run time more secure, check out sumologic.com/sedaily for a free 30-day trial of Sumo Logic.

Find out how Sumo Logic can improve your productivity and your application observability whenever you run your applications. That's sumologic.com/sedaily.

Thank you to Sumo Logic for being a sponsor of software Engineering Daily.

[INTERVIEW]

**[0:03:21.0] JM:** Kinnary Jangla is a senior software engineer at Pinterest. Kinnary, welcome to Software Engineering Daily.

**[0:03:27.2] KJ:** Thanks, Jeff, for having me.

**[0:03:28.2] JM:** Yes, it's great to have you. We're going to talk about the machine learning infrastructure more so than the machine learning algorithms themselves in this conversation today. So in order to get us into the right frame of mind to talk about these data pipelines that are deployed at Pinterest, let's give people a feeling for what Pinterest is. Can you just explain the product and the sources of data that are going into the data pipeline?

**[0:04:02.8] KJ:** Okay. Sure. Yeah. So Pinterest is a visual discovery engine, and by that what I mean is that Pinterest help people find or explore new ideas, for example, cooking dinner, trying a new outfit, redecorating a home. Anything that expires you to do things that you love. So it's basically an online pin board for collecting these visual multimedia, also known as images. And since the last — I don't know, eight years, ever since images has started, people have been saving and labeling their ideas. They save these pins and classify them into collections that they would like to go back to in the future.

So that gives us a lot of insights on not only their interest but also what's upcoming. That's definitely one source of data. Now, another source of data for us is the user actions on these pins. So depending upon what the user likes, what the user clicks, what the user re-pins? Those are a lot of different user actions that we could use to basically get information about what the user wants, and in turn create a more personalized feed for the user.

**[0:05:23.1] JM:** This is the same thing that is true with Netflix or with Facebook where you have a feed of options available to you and based off of the ones that you choose, you will have better feed content available to you in the future, and I believe that the feed generation process is key to a lot of these different companies. It's key to their success, because essentially you need to provide the most engaging, the best content that you can, and otherwise the users aren't going to be as interested in the product. They're not going to ultimately stick around as much.

So go through the evolution of the feed a little bit. So the home feed when Pinterest first got started, what was the most basic implementation that Pinterest had when the home feed first got started?

**[0:06:14.8] KJ:** Sure. I'm going to tell you a little bit about what exactly the home feed is, because I understand a lot of people are not usually aware of that. So the home feed is basically the landing page where users first get to when you try to refresh Pinterest, or which is your homepage, which shows the feed that Pinterest generates for you.

So going into a little bit of the history, six years ago or even before that, 2012 and before, what Pinterest is they had a chronological feed of pins depending upon the users you follow and the boards you follow. That's pretty straightforward. You basically get content from whatever the user follows, whoever the user follows, etc. That's good for small community, which is oriented around the same interests and even demographics if you may.

However though, it's hard to explore content. It's hard to explore interest. For instance, understanding that if this user likes baking, there's a potential that this user might also like ice cream making for instance. So that didn't happen in 2012. So we were not really adopting to the user behavior.

Now in 2013 we launched recommendations that Pinterest picks for you. So these were called picked for you recommendations. This is basically based on your existing re-pins. Whatever you like, you save it to your board, the collections that you make. Those are the recommendations that happened in 2013.

In 2014 it evolved slightly more. We introduced something called the concept of a smart feed, because after you give user recommendations, how do you understand what is more important to the user, because the first two to three pages of your feed are primed real estate. So a relevance came to the picture, and so we focused on ordering the feed based on relevance to you.

In 2015 we added more to this. We took more user actions into consideration by taking click through rate, re-pins, that improving relevance, adding localization to it, adding more machine learning based models, tree based regression models to this, so on and so forth. Then in 2016, we set it reacting. We became more reactive to this. So whatever, if the user didn't like something or the user likes something more, we started predicting what the user would like, but also help the user understand a lot about their own selves. Now we're just doing more and more of that.

**[0:08:39.2] JM:** I've seen a couple high-level approaches to how you do recommendations. So Pinterest, building the feed, you're essentially building a recommendation engine, "Here are a bunch of links and pins that you, the user, may be interested in. We are recommending them to you. Do you like them?" And the two ways, from a high-level, that I've seen of generating recommendations are collaborative filtering and content-based recommendations. So the content-based recommendations might be you have clicked on 10 links about baking in the past, and so we're going to give you some more links about baking. That would be the content-based thing.

The collaborative filtering based thing is you click on 10 links and there's another set of people who have also clicked on those same 10 links. Those people have also clicked on an 11th link. So Pinterest would decide to deliver you that 11th link as well. Is there an approach that you take either collaborative or content-based recommendations or is it just a mishmash of different approaches?

**[0:09:47.7] KJ:** Yes, exactly. It's the later. It's basically — There's also something that we predict. If you have in the past like something that we've recommended for you that the user you follow have liked, and we're going to show you more of that. There's a higher probability that you're bound to re-pin something if you've re-pinned what people you follow have liked in the past. Yeah, we definitely take history into account and we kind of mix it up and we try to decide whether the content that we recommend to you is more important to you than the collaborative recommendations that we're giving to you.

**[0:10:22.1] JM:** Even if it was something as simple as a basic collaborative filtering system or a basic content-based recommendation system, the scale at which Pinterest is at in terms of its number of users would make this a complex problem, even if the machine learning algorithms themselves were simple, and that's why we're going to spend most of the conversation today talking about how you wrangle that data, how you deploy new way of dealing with that data. So I think in the earliest days, Pinterest was a monolithic system. It was just a big, I believe a Rails monolith. Maybe you could talk about the early deployment of the Pinterest application itself kind of before it really had a lot of machine learning and modularity associated with it, and give a description for how it's evolved since then and how different services have sprung up around the monolith.

**[0:11:18.6] KJ:** Sure. Cool. In the beginning, obviously there's a big, huge monolithic service. I didn't work at Pinterest at the time, so I can't be very specific on how things went. But just understanding about the history of that happened, we did break that down into a lot of different microservices. But at the time — So I think when you talk about micro-service, you're talking about independence and when that doesn't — When your services are not divided into independent modules, it kind of becomes this one giant deployment process where your scalability is basically not truly divided into — You don't understand what exactly do you need to scale, because you've just scaled the whole service, takes more resources you don't understand. There's a higher likelihood that the whole service will crash together.

So with the evolution of Pinterest, we basically divided that into different services for our ML algorithms, our ad systems, our search, all of them, our different services now. So different components, different verticals of Pinterest are now different services. And what that basically did, it made everything very independent. We have different service owners for all these services, deployment units are very small, smaller rather that release of one services does not block due to any unfinished work in another service. It's basically continuous delivery. That's become much easier. Scalability has become much easier, because you get to scale the services that are bottlenecks only. Technology stacks are very independent. There's no long-term commitment to a specific stack. So it's become a lot more modular, can use multi-framework applications, so on and so forth.

With that, whenever you talk about microservices for a fast-based company like Pinterest, a mid-size company like Pinterest where you don't really want to depend upon — There are so many moving pieces rather. Microservices makes things much easier. However, you're going to deal with problems of a distributed system, like a lot of more effort, a lot more deployment cycles, a lot more monitoring, you're dependent upon maintaining life cycles of a lot more services. There's increased config management, a lot more delivery pipelines. Microservices also communicate over a certain network. Basically, there's a higher likelihood that something will go wrong compared to a programmatic API call. You're thinking about performance hits due to HTTP overhead. Migration to different services gets very tricky.

There's a lot of pros and cons that came along with dividing this monolith into microservices. However though, I like I mentioned, for a company like Pinterest, which is super-fast-based and we want to make sure that modules are pretty independent, people are not dependent on each other, code is not dependent on each other a lot, microservices has become really handy.

[SPONSOR MESSAGE]

**[0:14:08.8] JM:** Apps today are built on a wide range of backends, from traditional databases like PostgreS, to MongoDB and Elasticsearch, to file systems, like S3. When it comes to analytics, the diversity and scale of these formats makes delivering data science and BI workloads very challenging. Building data pipelines seems like a never ending job as each new analytical tool requires designing from scratch.

There's a new open-source project called Dremio that is designed to simplify analytics on all these sources. It's also designed to handle some of the hard work, like scaling performance of analytical jobs. Dremio is the team behind Apache Arrow, a new standard for in-memory columnar data analytics. Arrow has been adopted across dozens of projects, like Pandas, to improve the performance of analytical workloads on CPUs and GPUs. It's free and open-source. It's designed for everyone, from your laptop, to clusters of over 1,000 nodes.

Check out Dremio today at dremio.com/sedaily. Dremio solved hard engineering problems to build their platform, and you can hear about how it works under the hood by checking out our interviews with Dremio CTO, Jacques Nadeau as well, as the CEO, Tomer Shiran. And at dremio.com/sedaily, you can find all the necessary resources to get started with Dremio for free.

I'm really excited about Dremio. The shows we did about it were really technical and really interesting. If you like those episodes or you like Dremio itself, be sure to tweet @dremiohq and let them know you heard about it from Software Engineering Daily.

Thanks again to Dremio, and check it out at dremio.com/sedaily to learn more.

[INTERVIEW CONTINUED]

**[0:16:10.1] JM:** So a user navigates to their homepage and there might be a hundred different services that get called in order to assemble the elements of their home feed. Maybe you have got one service that's collecting images, one service that's serving ads. You've got a bunch of different services interacting with one another, and also once the user has landed on the page and the user is interacting with various elements, they're clicking on stuff, they're commenting on stuff, data needs to make its way from the user's browser to the services that are recording the information about the user's interaction. They need to make their way to those backend services. They need to make their way into the data pipeline at Pinterest so that that data can be analyzed and used to build better recommendations in the future for that user.

So now that we've talked through the deployment process and kind of the microservices architecture, tell me a little bit about how data that is generated from user interactions with Pinterest, how does that make its way into a data pipeline?

**[0:17:23.9] KJ:** Sure. At Pinterest, the way we ingest data, we have two data pipelines. We have an hourly data pipeline and we have a daily pipeline. It's basically hourly streaming process and daily as a batch process, and both these pipelines, they persist online data through Kafka and Spark. So we use Kafka for — It's basically a buffer. So it's a sub-buffer at Pinterest where we have topics, and these topics retain for two days. After that they got deleted.

So Kafka serves as a buffer between our online and offline data systems, after which we have clients which read off of these Kafka topics and persist these logs in sequence files in S3,which are streaming and batch jobs basically pick up and ultimately store these events in an HTFS cluster.

So for example, one of the examples of our hourly data reporting pipeline in the ads team at Pinterest is powered by Spark. And so once we have all these data, we basically use something like Presto for data exploration and analysis and we ultimately — All the user analytics data is ultimately stored using a combination of Hive and Spark, but it's all a huge HTFS lake.

**[0:18:37.3] JM:** Okay. So it's HTFS and the S3 stuff. You said there's a period of time where the events gets stored into S3. Is that just temporary? Does it eventually get thrown away?

**[0:18:48.0] KJ:** It's temporary in the sense of it's temporary for months, for a few months. Yes, it eventually gets thrown away, but like I said, recommendations, you want to store the data for two to three months at least.

**[0:18:57.5] JM:** What about the Kafka data? What's the expiration date? When do you garbage collect events that are sitting in Kafka?

**[0:19:05.5] KJ:** Every two days. Our attention policy is of two days, and all records get deleted after that.

**[0:19:10.2] JM:**So if I'm a user, I go to Pinterest, I'm clicking around. I'm making comments and stuff. Those events are going to hit — They're going to be propagated to a backend service. The backend service is going to write those events to Kafka. Could you zoom in a little bit on that process? Do you have different services that are recording different events, or do you have a single service that kind of records all of user's sessions. How is that partitioned in Kafka? Give me a description for that process.

**[0:19:38.8] KJ:** Yeah. We need different services that basically ingest this data, different data also, like user — For instance, user analytics is taken by a different service. Then we have everything — So all the data about a certain user, right? That's ingested by a different service. All the actions that a user takes, that's ingested by a different service, and then for instance the ads platform, all the advertisement data is ingested by a different service.

What we have is we have jobs. We have different jobs that ingest this data and write to different Kafka topics. Also, we have different clusters for these different services where all these data gets stored. Then ultimately, like I mentioned, these Kafka topics —

**[0:20:19.3] JM:** Different Kafka clusters?

**[0:20:19.7] KJ:** Different Kafka clusters. Exactly. We have these different Kafka topics where this data is written to, and ultimately our different jobs consume data from these Kafka topics to ultimately write to S3.

**[0:20:31.2] JM:** What would be — Like how many different Kafka clusters do you have? You have one for the ads team, one for the feed team. How does that work?

**[0:20:39.5] KJ:** I actually don't know the answer to that question, because I'm not very involved with that part, but, yeah, I'm pretty sure that we have different services that write to different Kafka clusters.

**[0:20:48.5] JM:** Interesting. You have some standardization around the ways that events are written regardless of what Kafka cluster you're using, or do you have a standardized event schema throughout the company?

**[0:21:03.1] KJ:** Yup.

**[0:21:04.0] JM:** Okay. How do you design an event scheme and evangelize it within the company, or does it matter? Do the events just vary from team to team, and if a team owns a specific type of event, they get to design their own scheme and it doesn't really matter what the other team schemas are.

**[0:21:22.2] KJ:** Yeah, and that's exactly also one of the problems. Different teams design their own schemas. I think that every team is supposed to be also involving stakeholders, but then again if you think about the future, it's very hard to tell, sometimes, two years another, another service might want to use your data as well, right?

In that sense it kind of becomes interesting, and that's why it's one of the problems at Pinterest is also accessibility of data, where these older legacy storage methods don't really help you retrieve data very quickly.

**[0:21:54.0] JM:** Do you have a scheme registry system?

**[0:21:57.6] KJ:** Not that I know of. I mean, I'm sure, but I don't know about it.

**[0:22:00.9] JM:** Okay. All right. That's fine. So just to poke around on this a little bit further. When somebody clicks on a pin, for example, and it makes a call to a backend service that the user has clicked on a pin. Does it fire a response to that click and then whoever that wrote that service is also responsible for logging an event for Kafka to be analyzed in the future or is there also an operational element to the Kafka cluster? Is it also like when somebody clicks on a pin, you're logging an event to Kafka and some operational database is going to need to read from that cluster and — For example, update a search index or update something in the billing team, something that's different than analytics.

**[0:22:46.6] KJ:** Oh, it's definitely used for operation data as well.

**[0:22:49.3] JM:** Okay.

**[0:22:49.8] KJ:** Yeah, because a lot of our recommendations come from the data, right?

**[0:22:52.9] JM:** Right. I would think of that as more as analytics as supposed to let's say updating a search index. Like if you're updating a search index, that would be — Well, I guess you could consider that analytics as well.

Anyway, so you mentioned Spark and Presto. Give me a little bit more of an understanding for how events are pulled off of Kafka and processed for other purposes. Like I think, for example, Spark, you're probably aggregating events off of Kafka and then doing things with those events. Could you talk about that?

**[0:23:28.2] KJ:** Sure. So like I mentioned to you earlier, we have two kind of pipelines. We have daily and an hourly pipeline. Both these different jobs take data from Kafka differently. There is the stream processing jobs basically are more real-time computations, which takes smallish windows of real-time data from Kafka. Obviously, it's all asynchronized. Basically that develops hourly data reporting pipeline.

The daily pipeline that we have, it's much batch processing. It computes big and complex data and it basically takes the data of the entire day over, and if you want some examples of that, like I mentioned to you earlier as well, in our ads team at Pinterest, we have an hourly data reporting pipeline, which is powered by Spark. We kind of like Spark, because it provides us the SQL API, which is good for the ETO Jobs, etc. I'm not sure if that answers your question.

**[0:24:19.5] JM:** Okay. Yeah. I mean, could you go a little bit deeper in terms of why Spark is a useful tool for that? Because I think people use a variety of streaming frameworks to essentially process large volumes of data for exactly this purpose, where you've got a bunch of events that have been queued up. Let's say you've got a bunch of events about users clicking on ads and they're queued up in Kafka in this pub/sub system and maybe you have a streaming job that is going to grab all of those events and process which users click on, which ads, for the purposes of reporting. Why is Spark a good tool for that and how does Spark work? How does Spark process those, let's say, a hundred or a thousand events?

**[0:25:07.7] KJ:** I'm honestly not a Spark expert. I just know that we use Spark at Pinterest.

**[0:25:12.5] JM:** Oh, okay.

**[0:25:12.9] KJ:** Yeah, I'm not a Spark expert at all, and as I understand why we use Spark, is that the fact that it's pretty flexible. It gives us the APIs we need for our extract, transform, load jobs. It also allows us to code in a low level RDD code if you want, you now, resilient distributed dataset if we need the flexibility. So that's as much as I know about why we use Spark, but probably somebody from the serving system's team might be able to tell you more on that.

**[0:25:39.1] JM:** Totally fair. I should be asking you more about deployments, because I think that's really what you're an expert in, is basically the infrastructure and deployment aspect of this. We've been talking about a lot of different pieces of infrastructure, like the services, the Kafka pipeline, the Spark system, Presto. What is involved in getting these services deployed and keeping the uptime and reliability there and setting up a continuous delivery?

**[0:26:11.3] KJ:** Sure. That's a great question. So like I mentioned to you earlier, we divided our entire monolith into a bunch of microservices, and the way it works at Pinterest is that each service has its own service owner, and then we have an SRE owner, which is dedicated to a team which owns these services. And each service has its own release method. However though, they all run off of Teletron, and Teletron is an in-house tool. It's the Pinterest deployment system, which we've turned to open-source.

Basically, the process of — The release process, the deployment process is basically you make your change, you commit it, you push it, you land it even, and then there's a test job and a [inaudible 0:26:54.3] job that goes off, and if both of them pass, the build gets pushed to — There's another job that publishes this build to Teletron. There's yet another job which basically takes this build and uploads it to S3. Now, when it's on Teletron, different services have their own deploy processes. Different services can also use the same build that was just ready.

Then some services also auto-deploy whenever the next build it up. Most of the services deploy to a canary host. That's essentially for testing that everything looks great, they monitor it for 20 minutes, or they're supposed to monitor it for 20 minutes before they move to production. Obviously, the whole monitoring production step is there as well.

Now, back in the day when all the services were — I'm going to you a little bit about how we migrated to Docker as well. How our deployments moved to Docker-based as well.

**[0:27:44.9] JM:** Please do.

**[0:27:45.9] KJ:** Back in the day when we did not have these services on Docker, there was a big possibility that you deploy a certain build on a certain host, and it was possible that the environment would persist even after deploying another build over it. That would totally mess things up, break things, etc.

With Docker, what has happened is that Docker becomes the new environment. In that, it serves a stabilization tool. That's one thing, right? The builds that you deploy on hosts don't mess up with the environment, because Docker becomes that. That's one thing.

The other thing that Docker has helped us do is that we have these bunch of Jenkins jobs that run scripts, which consume a lot of space because of all the temp files, and that ends up making the host super slow and so on and so forth. So with Docker, we got rid of that as well, because once your job is done, you shut this down and all their changes are boom, gone. In that sense, the deployment process have become much more stable when we've introduced Docker at Pinterest.

**[0:28:52.2] JM:** I believe that the way that the services communicate with each other is via Thrift. Is that accurate? Thrift is a serialization protocol. Why did you choose Thrift?

**[0:29:05.0] KJ:** I love Thrift. To begin with, first off, I think the fact that it has much less overhead, because it uses binary format compared to SOAP, for instance. That's great when — I think it's very clean. I'm not a big fan of XMLs, so Thrift really is a super clean library. It also generates both the client and server code including the structures who pass. All you end up dealing with is handlers. That's great. You have persistent connections. You're not dealing with TCP, HTTP handshakes. Yeah, for all those reasons, I think it's a great cross-language serialization tool.

[SPONSOR MESSAGE]

**[0:29:47.2] JM:** Amazon Redshift powers the analytics of your business, and intermix.io powers the analytics of your Redshift. Your dashboards are loading slowly, your queries are getting stuck, your business intelligence tools are choking on data. The problem could be with how you are managing your Redshift cluster. Intermix.io gives you the tools that you need to analyze your Amazon Redshift performance and improve the toolchain of everyone downstream from your data warehouse.

The team at Intermix has seen so many red shift clusters that they are confident that they can solve whatever performance issues you are having. Go to intermix.io/sedaily to get a-30 day free trial of Intermix. Intermix.io gives you performance analytics for Amazon Redshift. Intermix collects all your Redshift logs and makes it easy to figure out what's wrong so that you can take action all in a nice intuitive dashboard.

The alternative is doing that yourself, running a bunch of scripts to get your diagnostic data and then figuring out how to visualize and manage it. What a nightmare and a waste of time. Intermix is used by Postmates, Typeform, Udemy and other data teams who need insights into their Redshift cluster.

Go to intermix.io/sedaily to try out your free 30-day trial of Intermix and get your Redshift cluster under better analytics. Thanks to Intermix for being a new sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

**[0:31:33.0] JM:** So part of this discussion is about the fact that you have migrated this — Well, you have taken part in the migration. Maybe you aren't there for the entire thing, but gone from Pinterest being really monolithic and not very smart, not having a whole lot of machine learning, to a system where it's been Dockerized, you've got microservices and you also have this data pipeline. And my understanding is that this is a very gradual process. It's not a Big Bang process, and that there is an importance in how you strategize about this and how you create short-term goals as well as long-term goals in order to gradually get the entire service — The entire company rowing in the same direction. Do you have any suggestions around the management of that process?

**[0:32:27.0] KJ:** Sure. Yeah. This is the way that I look at — This also reminds me of how would you migrate legacy systems as well. So when it comes to migration I think about it in two ways. One is the classic you lift and you shift it. You just directly move it. The other one is a refactoring. You refactor the entire architecture.

Now, when you follow the first path where you're basically just lifting and shifting, there are a lot of automation software that you could use for image migration or data migration or whatever. However though, I do think that when you refactor or when you gradually migrate, it's a great time to think about a lot of optimizations around scalability, for instance. Great time to think about efficiency and a fantastic time to think about data, how your data is accessed, how your data is utilized. Are you using block storage? Should you be moving to object storage service, for instance?

I think when you're thinking any kind of migration, whether it's gradual or not, I think it's a great time to think about all the kind of optimizations that you've always wanted to make that looked tedious to you. It's basically like moving a home, right? You trash all the bad stuff you don't use. You get new furniture. You think about making your house more functional. So I think that's a great analogy when thinking about migrating systems.

**[0:33:45.6] JM:** You did talk about data access being a big issue at Pinterest, and the way I've seen data access as a problem manifest in other companies, is you have different teams that are generating events or other forms of data that are being thrown into a data lake, and you have analysts throughout the company as well as machine learning model builders. So you have analysts that are perhaps doing exploratory analysis of different parts of the system and you have machine learning engineering that are productionizing the realizations of those analysts. Those maybe the same person in some cases. You may have a machine learning person who does exploratory analysis and then turns that into a model. But in any case, it's important to have this data readily available and in a form that's easy to pull from. There are some tools, like Looker, for example, that have been — Entire companies have been started around this problem of data access. Can you talk a little bit about how data teams at Pinterest are organized and what are the tools that they're using and what are the process that they're using to explore data and then productionize those?

**[0:34:57.2] KJ:** Definitely. Sure. So at Pinterest we have a centralized data team. Under that, we have three kinds of teams. There's a data engineering team. There's a data science team, and there's a product analytics team. Now under the data engineering team, that's the team that deals with everything data, everything Kafka, everything streaming. Under the data eng team, we further have a big data platform team that deals with supporting Kafka, streaming, in-house Hadoop, our jobs, maintaining it, scaling the clusters, building tools that allow other teams to query big data platforms. Then we have another team, another engineering team, that is the data analytics team, which basically is responsible for building internal-facing data tools for building the experimentation framework, for building — There is a tool called Pinalytics, which is basically analytics for experiments. So also building a [inaudible 0:35:51.0] of internal tool for queries, ETL workflows and so on and so forth.

Then there's another team which is the ML platform team, which basically builds a centralized machine learning platform that's basically used across the company. Then we have this smaller team called human computation team, which is all about human judgment and all the mechanical turks, etc. That's our data eng team.

Then we have the products analytics team, which is basically all the data analysts, which are mapped to a specific product teams. These guys help engineers and the product managers understand the opportunity size. They analyze the data in the form of running SQL queries, and so on and so forth. Then we have third team, which is the data science team. Now, the data science team is less about operations. It's more about thinking about the high level problem domain, defining the problem domain. It has a more of a research component to it, and so bigger in scope, and it has a broader range of tools. So these guys have a lot more software engineering skills than analytics.

However though, I think the good part is that every team knows which data analysts you're going to work with, which data scientists you're going to work with, and data engineering team is basically across the company. So anything related to data engineering, you know which team to go to. So I wouldn't think it's that big of a problem, honestly.

Okay. So now let me go a little bit into what kind of work do the data scientists at Pinterest do, right? They do a lot of exploratory, experimental analysis, mostly build and train models with a fixed objective. There are three types of projects that data scientists at Pinterest do. One is the very traditional kind of work where they build predictive models, which are usually smaller in scale. They're internal models, not very user-facing, because they don't have an entire engineering team to support. So however, the experiment around optimizing certain features, for instance predicting how likely are certain keywords to be useful to the team, for example.

Then there are these other set of projects, which are towards research and analysis where you're defining new concepts. You're defining, for instance, variety on the feed. You're defining how do you measure it. You're defining statistical properties. You're trying to provide as much product information as you can. You're trying to understand how computationally feasible that's going to be. You're also trying to observe whether a certain thing is going to affect retention, for instance. If it's not, then do you really care about working on it, so on and so forth it. They also design and run a bunch of experiments.

Then there is a third team, which does more of internally-driven work. Nobody asks them to do it, but it's really important. For instance, anomaly detection. For instance, you see a week to week drop of 2% on a certain metric. So you're really deciding, should you freak out about this? How much do you take this seriously? Is it alarming or not? So on and so forth. That is what data scientists at Pinterest do.

Now, I did want to also touch a third question on that. I don't know if you asked or not, but I do want to go into that, is that data access at Pinterest is a huge problem. We have a lot of data, which is a great problem to have. However, what data to trust becomes a big problem. I think legacy storage methods have not really proven to be very optimal. So tools like Presto, that comes very handy. That helps us faster. That helps us to query data a lot faster than — This huge data lake takes a long time to query anyway. So Presto kind of comes very hand there.

**[0:39:37.7] JM:** So are you referring to HDFS as the legacy storage technique?

**[0:39:42.3] KJ:** Yes, exactly.

**[0:39:43.4] JM:** Okay. Right. Presto does what?

**[0:39:46.3] KJ:** Yeah, it just helps you query the data a lot faster, to explore the data.

**[0:39:50.8] JM:** Okay.

**[0:39:51.5] KJ:** Yeah.

**[0:39:52.0] JM:** How do you look at the overall process of the different teams working together and figure out places where you can improve the feedback loop? For example, I hear this problem a lot that getting models updated quickly and the testing process for models and the deployment process for models can be sometimes hard to do. Have there been any improvements that you've made overtime that would be worthwhile to share?

**[0:40:20.1] KJ:** Sure. There's one thing that I really want to talk about, because I worked on it. So one of the things is that being an ML engineer, it becomes — You want to be able to experiment quickly. So that in turn ties into how do you run your experiment on your local machine and try to see whether you're going to get the results from a certain model or not, from a certain experiment or not. That's a challenge, or at least that was a challenge when I joined Pinterest, where debugging was very tedious, because for any ML, any part of a code that does any kind of ML, you're basically — You have hundred dependencies, you have dissimulated databases that you want to run. One micro-service depends upon the other, and in order to get this whole piece working, everything behind it has to work very smoothly.

So getting all these services on your dev op or your local machine and having them up and running in order to test your model becomes extremely challenging. So this is also where it ended up using Docker where I've put all these services, each service has its own Docker environment and all you end up doing is use Docker compose, run all these services together, and there's this list of services. You only run the ones that you want. You have these services which run simulated databases as well. Debugging became a lot more easier. Hence, experimentation became a lot more quicker. Because now instead of actually going to experiment dashboard, creating your experiment, waiting for two weeks to look at the data, and then in turn going and modifying your features and whatnot. Now you can basically do all that on your dev op and it's a lot more faster, the pace — You get feedback very quickly.

**[0:42:03.8] JM:** I imagine that makes onboarding a lot easier too.

**[0:42:06.5] KJ:** Yes, definitely.

**[0:42:07.4] JM:** So you are writing a book about this exact topic that we've discussing, building and deploying machine learning deployment. What are some of the elements of the book that we have not discussed in our conversation yet?

**[0:42:21.6] KJ:** My inspiration for writing this book was that when I was on onboarding, when I was ramping up on Docker or when I was ramping up on how to use Docker for helping my team debug things a lot quicker, there was no place where I could find anything that was structured in a way that I could go from basics to semi-advanced, to advance topics. And I think that's where I got inspired to write this book where given the fact that I do think debugging using Docker is extremely interesting.

However, what exactly is Docker and why are we using Docker given other alternatives? Why does it exist? What are the use cases? What are the different use cases enterprises can use Docker for? That's a start of the book. After that, I also intend to write about basics of Docker, what is Docker compose and how do you shut down a container? How do you debug when you're using a container — Inside a container? Then how do you club these microservices and then go ahead and debug your entire app. So that's how my book is going to be structured. I'm looking forward to a lot of feedback from people to understand whether it provides a structure that I was looking for when I was ramping up on Docker.

**[0:43:32.7] JM:** This is more about the service deployment system rather than the — I guess even if you're talking about machine learning deployments, your machine learning models are going to sit in a Docker container, right?

**[0:43:45.1] KJ:** Exactly.

**[0:43:45.6] JM:** At Pinterest, are you looking at Kubernetes at all?

**[0:43:48.2] KJ:** Yes. We do have a cluster right now. We are in the experimentation phase of — We're not using it in production, but we do have a Kubernetes cluster for testing.

**[0:43:57.9] JM:** Cool. What are you focused on now at Pinterest? What are the big initiatives around machine learning and Docker and deployments?

**[0:44:04.2] KJ:** Sure. So as far as ML goes, so far Pinterest has been using machine learning to — The discovery team uses it to provide recommendations, related contents, predicts the likelihood that a person will pin content. We definitely want to keep doing all these. We want to just get better at it, improve our relevance.

We have another team called the visual discovery team, which is a lot into deep learning algorithms. They do objective recognition, etc. So we want to improve on that as well. We want to monetize our ad performance, our relevance prediction using ML as well. Our growth team uses ML to build intelligent models to determine which emails to send and prevent churn. That's something we want to continue doing better.

However though, I think that one of our major goals around ML in 2018 is to improve our spam detection and our fraud detection, and I think that's something that we're really looking forward to accomplish in this year using machine learning.

**[0:45:06.5] JM:** That problem is so hard. Whatever company you are, it seems like it's a never ending battle.

**[0:45:13.5] KJ:** Definitely.

**[0:45:13.8] JM:** Kinnery Jangla, thank you so much for coming on Software Engineering Daily. It's been really good understanding the pipelines and the machine learning deployments, the microservices deployments at Pinterest.

**[0:45:25.2] KJ:** That's, Jeffrey, for having me. I had fun.

[END OF INTERVIEW]

**[0:45:30.6] JM:** If you are building a product for software engineers or you are hiring software engineers, Software Engineering Daily is accepting sponsorships for 2018. Send me an email, jeff@softwareengineeringdaily.com if you're interested.

With 23,000 people listening Monday through Friday and the content being fairly selective for a technical listener, Software Engineering Daily is a great way to reach top engineers. I know that the listeners of Software Engineering Daily are great engineers because I talked to them all the time. I hear from CTOs, CEOs, directors of engineering who listen to the show regularly. I also hear about many newer hungry software engineers who are looking to level up quickly and prove themselves, and to find out more about sponsoring the show, you can send me an email or tell your marketing director to send me an email, jeff@softwareengineering.com.

If you're listening to the show, thank you so much for supporting it through your audienceship. That is quite enough, but if you're interested in taking your support of the show to the next level, then look at sponsoring the show through your company. So send me an email at jeff@softwareengineeringdaily.com.

Thank you.

[END]