EPISODE 1024

[INTRODUCTION]

[00:00:00] JM: Lyft is a ridesharing company that generates a high-volume of data every day. This data includes ride history, pricing information, mapping, routing and financial transactions. The data is stored across a variety of databases, data lakes and queuing systems. It's processed at scale in order to generate machine learning models, reports and data applications.

Data workflows involve a set of interconnected systems such as Kubernetes, Spark, TensorFlow and Flink. In order for these systems to work harmoniously together, a workflow manager is often used to orchestrate them. A workflow platform let's a data engineer have a high-level view into how data moves through a system and it can be used to reason about retries and resource utilization and scalability.

Flyte is a data processing system built and open-sourced at Lyft. Allyson Gale and Ketan Umare work at Lyft and they join the show to discuss how Flyte works and why they needed to build a new workflow processing system where there are already tools such as Airflow. This is a great show about data engineering and a look into the modern data engineering process of a large company like Lyft.

[SPONSOR MESSAGE]

[00:01:25] JM: SAP is a company that touches $23 trillion of consumer purchases around the world. You've probably heard of SAP but you may not know about SAPs open source investments, such as SAP Data Intelligence. SAP Data Intelligence connects and transforms data to extract value from the distributed data landscape and embraces the best of open source technology.

SAP Data Intelligence brings together data orchestration, metadata management and powerful data pipelines with advanced machine learning enabling close collaboration between data scientists and the rest of your infrastructure teams.

SAP Data Intelligence runs on Kubernetes. SAP's contribution to Kubernetes includes the Gardener Project that helps to operate, monitor, manage and keep Kubernetes clusters alive and up-to-date. To learn more about SAP Data Intelligence, you can visit sap.com/sedaily. That's sap.com/sedaily

[INTERVIEW]

[00:02:36] JM: Allyson and Ketan, welcome to Software Engineering Daily.

[00:02:39] AG: Thanks for having us.

[00:02:41] KU: Thank you.

[00:02:42] JM: You both work at Lyft. Lyft has several different teams that require high-volume data processing, and some of these teams are building machine learning applications. Other teams, maybe they're just building complicated workflows involving large datasets and parallel computing. Allyson, you could start off by talking about some of the teams that need to do large scale data processing at Lyft?

[00:03:05] AG: Yeah, happy to do that. Yeah. When you open the Lyft app and you look at it, it comes across as a single app, but in reality it's a collection of services that all talk to each other. Some of those are machine learning services like pricing or ETA as well as kind of services that require a large scale compute, like the foundational map that everything is built off of.

Flyte actually services all of these. Most numbers that you see in the app are actually powered by Flyte. Pricing is actually my favorite example. You serve a price for Lyft. It actually requires many different kind of data endpoints. For example, you have to know how long the ride is. How long you expect it to take. What type of ride it is. Then also things like tolls or any other like regulations that might factor into the price.

The pricing team uses Flyte to actually do all of their model training. They pull all these inputs into their workflow and then eventually out pops the price. The other thing to highlight here is that all of these services are pretty much interconnected and interdependent. The ETA for the

ride will also affect the price. Then, again, which route we take them on will affect the ETA, which then affects the price. It's pretty neat. All these services basically have workflows on Flyte and they all interact together. So Flyte does a really good job at helping them kind of share different, I guess – Well, we call them task in Flyte. We'll get more into that later. But it helps them basically manage these dependencies as well as the compute itself.

[00:04:31] JM: What are the pain points of these kinds of applications that involve large scale data processing? The pain points of the people who are building these kinds of applications.

[00:04:43] AG: Yeah. So one of them is infrastructure itself. A lot of machine learning in managing infra, everything from the type of machine you're using, to actually like retrying something if it fails. The other thing is sharing in collaboration. As I mentioned, Lyft and just increasingly in the business world, a lot of these services are interdependent, interconnected. So one kind of principle is that you would only want to build something once. If we have a pricing model, the ETA team, again, estimate time of arrivals, if that team wants to use pricing in their own machine learning model, we don't have to rebuild that pricing model from scratch. In general, infra is a huge hurdle as well as kind of sharing and collaboration. Flyte kind of works to solve both of these.

[00:05:30] JM: The structure of a data team is an evolving subject. We have data engineers. We have data scientists. We have data wranglers. We have data analysts. Tell me about how you have both seen the structure of data teams change in your time at Lyft. Ketan, maybe you could go first.

[00:05:49] KU: Yeah. It's been a little more than four years when I started, the entire engineering team of Lyft was less than – About a hundred people. The data team itself was four or five people. The structure of that team has been very fluid. Initially, most of the engineering teams themselves would build the data processes themselves, how it evolves. But as time progressed, we centralized a bigger data team that would manage the ETL processes that would be the owners or the keepers of the fact and the dimension tables and would also create rule of dashboards for the execs and for the company as a whole. That became the primary function of the data team.

Eventually, and at that time, it was still doing machine learning and we had a lot of research which sat completely in a different place than the data team. They were [inaudible 00:06:45] with each of the teams themselves, and they were responsible for actually writing code that would be shipped to production at some point.

To that effect, most of the services initially at Lyft were written only in Python, because if you had to do anything with machine learning, we had to ship models that were written by research science whose preferred language was Python. We actually had services. So we made a decision as to what would be the best for engineers as well as for research science at that point.

Then these models were created usually on a laptop or in some dataset separates in like one hosted notebook. Then the model itself as an artifact was given down to the service team and the service team would manage it and maintain it and deploy it and so on. This doesn't scale quite a bit. So what started happening is people started writing machine learning pipelines. Now you would call these as machine learning engineers or machine learning pipeline engineers.

In the earlier part, they were actually the engineers who are part of a team. Let's take an example of ETA. Within the ETA itself, we had a lot of machine learning models being built. So some of the engineers took the responsibility of becoming machine learning pipeline engineers. They worked with the research science to develop and deploy models and then there were some engineers who would just work on the frontend service and there are lot of things to be done for ETA itself, which are purely engineering people without the machine learning. It's something [inaudible 00:08:22]. This was a very natural [inaudible 00:08:24] that happened.

Eventually, I think, we realized that the fact tables and the dimension tables were getting too big to handle a lot of data and understand a lot of data and we needed expertise. We started hiring a bunch of data engineers who help the data analysts that now they're called data scientists to help with understanding and training models or extracting information from data that is useful to other parts of the company.

This is further evolving. These data engineers now actually are sitting next to the machine learning engineers and the engineering teams so that the data itself is available in the most usable form to each of these teams.

You absolutely hit the point. I think data – The structure of data engineering and machine learning engineering teams is completely fluid. I don't think we've found the right way yet, but we are constantly evolving, and I've seen this evolution over the last few years and I assume that it's going to evolve in the next couple of years too.

[00:09:34] AG: Yeah. Something else is when the user research kind of within Lyft talking to a bunch of machine learning engineers and data scientists. One phrase that I heard repeatedly was, "Okay, data scientists writes the model and then they throw it over the wall to the engineer to kind of write the production scale pipeline," and that phrase was just echoed almost verbatim across both functions.

One thing that we're seeing, I mean, that's like a very clear indication that there's this kind of hurdle or at least there was this hurdle between these functions. But what we're seeing too is like the lines between these functions are getting blurred, as Ketan mentioned. I mean, we do see kind of a convergence happening.

So kind of one, again, of like the predictions of Flyte is that overtime these functions really will blur. They kind of will be a singular function that just has different specializations. It might be more specialized in model creation of something like that. You might be slightly more specialized in the engineering side. But what you really want to do is get rid of that wall, and you can get rid of that wall through product. You can get rid of that wall through process. You can get rid of that wall through the definitions or the rules themselves. Flyte kind of plays a role in that by being a product that is intended to take down that wall, but then also to reflect eventually a convergence of these rules.

[00:10:46] KU: I just wanted to add more one thing. Actually, I think Ally absolutely hit one of the most interesting and important points for the genesis of Lyte. I was a lead on the team and the problem that we had was research science would come up with like an idea of a model and probably have a prototype on some sample set of data. To actually productionize it would take months. When the engineering team would come back and say that, "Hey, we would take months to actually productionize this," that would not be acceptable to research science, because you want – You've taught about a model. You want quick feedback for production, and

that was very, very hard to achieve. Many times what would happen then is that that goes to the first pointing, that infrastructure was the bottleneck.

The second problem was that there was this inherent thing that research science would create a model and then the model was finally deployed by the engineers themselves and then a PagerDuty rotation, I would say, was handled by the engineers that caused a lot of friction because in the middle of the night, I deploy the model, it caused a bunch of pages. I have no idea what to do. The only course of action would be rollback that model.

Now that may have for their consequences on the analysis of that model. It became actually harder and harder to manage specifically. In situations like this or like in cases when the model would be given to an engineering team, the engineering team would say, “Hey, I’m going to do the PagerDuty for this. So let me rewrite this model.” This translation was not accurate. You essentially ended up with sometimes losing some information while throwing over the wall, over this imaginary wall and that two friction between the teams. One of the core principles of Flyte was we will not try to segregate these teams, rather try to get them work together and collaborate more effectively.

[00:12:43] JM: Okay. So as we get into what Flyte actually is, my high-level understanding of it is this is a system that’s built on top of Kubernetes. It orchestrates your machine learning or data workflows, and I think we’re going to need to talk a little bit about what a workflow really entails and why it involves different teams. Why it involves this different handoff process. Could you just give me an overview for what a data workflow is in the context of Flyte?

[00:13:20] KU: Yeah. I think it’s better if you take an example, right? Let’s take an example of ETA. Again, we’ve talked a lot about ETA. So we’ll continue to talk about ETA. ETA essentially means estimated time of arrival. When you open up the app you see how long a car would take to reach you. This seems simple and naïve, but it’s actually very, very important for Lyft as a business, because based on – ETA here from Lyft’s point of view is actually a reference between two points. When we dispatch a car, we have to understand where the car is in reference to that user so that we can make better decisions. We can optimize our system better and actually eventually affect the bottom line.

ETA, as I said, is critical. But just to power that one simple number, you need to do a bunch of processes. For example, you need to know exactly where the location of a car is. Then you need to know how the road network is. Then you may need to know if there is traffic on the current road network. Then you may want to know how much time does it take actually towards that road network. Eventually you may want to also add that this driver actually drives differently from this other driver. So he may be faster than this other driver, and this is the time of their day. In this time of their day, even though the traffic is low, it usually takes more time because we see more red lights or stop signs. All of these things are impact of – That impacts a simple ETA. These individuals things calculated at using models.

To compute these models, you need a bunch of data. As I said, let's go just look at one of the models to compute traffic. To compute traffic, we need to understand the road network. We get the road network from something called Open Street Maps, which is an open source mapping system with a lot of data. You download the data. You extra data – From the data, you extract the road network. On the road network now at the same time we analyze all the cars that are moving in the Lyft system and we understand that on this road network, the current movement or the traffic patterns in these specific road areas is X.

Using that, we actually build a routing engine model that gets deployed and then served. At the same time, on the traffic itself, we may have biases, because if there is a sudden accident in one place, we may shoot up the ETA. That's very undesirable, because it may not be accurate because outliers happen. How do you smooth out? So we have a layer on top that actually smooths out this basic model using some sort of historical trend analysis.

For that, we have now been analyzing the traffic for the last 6 months, 1 year, and applying it to this already – Of the computer traffic at the moment. You see that there are two pipelines that run and power one simple computation, and both these pipelines have to happen at some frequency. Their traffic computation probably happens at every 10-minute interval or so. We started filter and analysis happened every week or so, gets submitted all the time.

Along with that, they also use different sets of data. One used an open source dataset. The other used the internal computer dataset and that creates – The workflow here means you take one data just like we talked initially when we were talking. Before, you have a workflow, you talk,

then you edit and then you post-process or then you launch your podcast. Same thing the workflow for machine learning engineers.

[00:17:16] AG: Yeah. I'll also add a little bit to that, and since I'm a product manager, I love analogies. So I'm going to use a little bit of analogy here. I like to kind of describe tasks and workflows in kind of Flyte's overall process here kind of like cooking a meal. If you imagine you're cooking a recipe, we can kind of consider that like a single workflow and you would have kind of different tasks as part of that. So if you're making pasta, you have to boil the pasta. You kind of have to strain it afterwards. You have to make your sauce. Maybe you're making a side salad as part of that. All these individual tasks went to cooking that meal.

Even actually probably a better analogy here is to consider this like the sauce itself as a single workflow. You'd like maybe chop the meat if you're doing that. Chop the mushrooms. You have to boil the sauce in the stove. You have to kind of combine it all together and then out comes this amazing sauce at the end. We imagine that as kind of like maybe something, like our mapping team putting together the overall view of the map at Lyft.

Then since you asked about kind of teams in the handoff process, you can imagine now that we have a whole kitchen of chefs and everyone's creating their individual portion of that meal. There's a certain like amount of communication that needs to go into this. You have a whole team that's just focused on the paste and creating that. A whole thing that's focused on the sauce. A lot of these kind of are dependent on one another. You don't want to cook the pasta too soon, like once the sauce is made, you might want to then start boiling the pasta. That gets us into things like kind of event that's triggering space on workflows.

Every time our mapping team produces a new map of Lyft, we want the ETA team and the pricing team to know that since their services depend on that map. It also kind of – Just to abuse the analogy a little bit more. You can get into things like versioning. If you're creating this Italian meal, you might at some point decide to update your sauce. Maybe you're putting more garlic in it now or something like that. That's important for the other teams to know. They might also change what they make or how the salad is done or something depending on that new flavor being added in.

Flyte kind of has to be aware of everything that everyone is making, also make it really easy for them to kind of maybe start their workflow when someone else's workflow finishes and then also be aware where there's updates happening to all these individual portions as well. Hopefully that helps a little bit.

[SPONSOR MESSAGE]

[00:19:30] JM: I've recently started working with X-Team. X-Team is a company that can help you scale your team with new engineers. X-Team has been helping me out with softwaredaily.com and they have thousands of proven developers in over 50 countries ready to join your team and they can provide an immediate positive impact and lets you get back to focusing on what's most important, which is moving your team forward.

X-Team is able to support a wide range of needs. If you need DevOps, or mobile engineers, or backend architecture, or ecommerce, or frontend development, X-Team can help you with what you need. They've got a full-range of technologists who can help with AWS, and Go lang, and Shopify, and JavaScript, and Java. Whatever your engineering team needs to get to the points of scale that you want to get to, X-Team can help you grow your team. They offer flexible options if you're looking to grow your team efficiently, and their model allows for seamless integration with companies and teams of all sizes. Whether you're a gigantic company like Riot Games, or Coinbase, or Google, or if you're a tiny company like Software Daily. You can get help with the technologies that you need. If you're interested, you can go to x-team.com/sedaily. That's x-team.com/sedaily to learn about getting some help with your engineering projects from X-Team.

Thank you to X-Team for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[00:21:14] JM: Okay. So now that we've got an example and we've talked through this from the point of view of an analogy also, let's talk about the actual interconnected systems. Maybe you've got a data warehouse. You've got some blob storage. You've got Spark. You've got TensorFlow. You've got all these different interconnected systems that need to have their share

of work along a data pipeline. How are these all getting instrumented together and communicating with one another?

[00:21:51] KU: In the example of the real-time traffic case without giving out too many details, one of the first steps was, as I said, download the Open Streets Map data. These are in a specific format that Open Street Map publishes and there are source tools that we use to analyze and extract data, all of that format. That's written as a team or an individual who knows how to extract data and he writes that code. Usually, in Flyte, you can write this code in Python, for example, or any other language. Flyte is language-agnostic, but the language of choice at Lyft is Python. Flyte comes with a toolkit called Flyte Kit, which allows you to write, to express a function very easily. It's essentially like writing a Python function and adding into it something that says, "Hey, this is a Python function. I need X-gigabytes of memory and Y number of CPU, and I need to – If there is anything, I need X-inputs and I produce Y as an output."

In the case of road network, you may take the date of the map update as the input and the location of where to find that file as the input. The output might be the road network, which might be some sort of a binary format. Now, once you have that, you write another function that would be a Spark function that actually takes the road network and maybe chops it up into pieces or create a much more linearized model of the road network. This can be done in Spark and Spark would be much better because of the amount of data that you may want to handle.

In Flyte, again, writing a Spark function is just like writing a Python function. You just annotate it saying that it's a Spark function. In this case, I don't specifically need CPUs and memory. I need a number of Spark course to be X. Number of Spark executers to be Y and so on. Then you just write the code as if you're writing on either the Spark REPL or you're writing PySpark code. You write that and you again take some input. The input to this case was the road network. Maybe something else that you want to apply to the road network and the output is the modified road network.

Now, let's say on top of this you want to train the model, but just the bias on this road network. You've mentioned TensorFlow. Maybe we could use a CNN model or we could use a [inaudible 00:24:30], which is another popular black-box framework to do [inaudible 00:24:34] boosted decision trees. You could use a model based on [inaudible 00:24:38]. The input into that would

be the road network and the output the trained model. You would do the training as that function again.

A cool thing [inaudible 00:24:50] or even into full-length models, they are very black-box models. They usually depend on the type of model, but [inaudible 00:24:57] specifically can be run for any sort of data and it produces a model without too much help from the actual engineer and understanding how the model is structured.

In this case, it could be a library function that says, “Train this [inaudible 00:25:16] model given this data,” and it produces the model.

The last bit could be, “Hey, now I have the model. I have the road network with the traffic. Let’s make sure that it actually works well by using some historical data.” Let’s take a historical set of rides from Lyft. In those scenarios, if we had this model, what would have been the ETA, for example, that we would have predicted and how would that have deployed from the actual ETAs that we saw?” We actually gave out an ETA and then the ride was taken. Then we actually saw the ETA as an exposed fact. So we can now use this dataset to validate there are models actually improving the system.

These are five steps that I mentioned, or four steps. You can steam them together into one workflow or a pipeline and some of them could be written by specific engineers, as I said. We put out all of the data that a road inspection would be done by one engineer. The Spark stuff would be done by somebody who knows Spark better and the [inaudible 00:26:21] stuff can be actually just be a library function that we have written as the Flyte team and we are offering it too for the company to use and it’s optimized for that. Then [inaudible 00:26:31] hyper-parameter optimization and so on that you can offer as common functions within Flyte.

The last bit could be written by the data scientist, in this case, who’s trying to make sure that the new model makes sense. To trigger this, he just has to specify, he or she just has to specify the input date of the OSM dataset and the range of the Lyft rides to use and validation set and maybe the range of rides to use for computing the traffic. That’s about it. The same functions can run together in harmony to create a model and also tell you the accuracy of this created model.

[00:27:13] AG: Yeah. Something else on top of that to highlight is that we designed Flyte to be super extensible. For example, you mentioned TensorFlow and Spark, but you can think of other execution engines that you might want to use and then as well as if you didn't want to use Python to write your tasks. You could also extend Flyte to be for any other language as well. It makes it super easy to kind of adapt to your needs.

[00:27:37] JM: We're talking about a system that's written in Python. It's used to orchestrate different systems together. It can help us with data dependencies. Sounds a lot like Airflow, which was originally made by Airbnb by an engineer there named Max. That was many years ago. Why create a brand-new system? Why did you create Flyte when we already can take Airflow off-the-shelf?

[00:28:08] KU: Very good question. When I started off leading ETA, we did used Airflow in the beginning. It did help us in some way. We hacked up a lot of the Airflow to make it work for our use cases. Then we delivered something. It got successful and people wanted to use it more. At that point is when we thought about if we could extend Airflow to make it work for all these various use cases, and we found that it wouldn't work. So we had to move to Flyte.

Let me tell you some basic difference. One thing I want to correct is Flyte is actually written in Go lang. The entire engine, the backend and everything is written in Go lang. The interaction model that you see most common, which is called FLyte Kit, is written in Python. The biggest difference between Airflow and Flyte is from an idea point of view, Flyte is completely data-aware. It means it understands what's the data that's flowing through the system, while Airflow is a pure task dependency system. That means you as a user has to think about that task A, comes after task B, comes after task C and you arrange them in that way.

In Flyte, you never have to think about it that way. You just have to say, "My code is written so where I have a function whose data I consume – It produces some data that I consume in some function B whose data I produce and I consume in function C." Now, what is the structure of this DAG? It's a DAG, again, because it's called directed acyclic graph, but it is auto-computer from the code that's written by the users.

The way this DAG is computed is not because you arbitrarily put the tasks one after the other. It's because the consumption, because there is a producer-consumer relationship between who creates the data and who consumes the data. For that to happen, you actually have to understand that there is data flow between the spaces. That's the other big difference in Flyte where we actually had to model a complete type system to understand that a function can produce an integer and a string and a list of integers or a parquet file or whatever and/or model and they are all different things. Each of the downstream functions can consume each one of them separately or together and that creates a different set of dependencies.

For example, if function A producers two integers and there are two functions that consume one integer each, both of them can go in parallel after function A. To achieve that, we created, as I said, a type system. The other thing that we saw was a big problem with Airflow when we were using it was that if we write a new pipeline in Airflow and let's say that pipeline uses TensorFlow. The moment you import TensorFlow, the scheduler imports TensorFlow. The workers input TensorFlow. That means the version of TensorFlow has to be same across all of them, or you do tricks like [inaudible 00:31:19] importing of TensorFlow and so on. That would cause a bunch a problems in the scheduler and the worker.

Besides that, there was no – Also, when we would deploy a new version of the pipeline in Airflow, it actually just overwrites the existing version of the pipeline. Let's say that pipeline is running in production at the moment and you deploy, the behavior is unknown. This was an anomaly for us because what we wanted to do was have a pipeline that runs in production and constantly deploy a new version just like we do with services without actually affecting the executions that were already in progress. That means after this epoch, every new execution is the new version of the pipeline.

We also wanted to go back in time and execute all the versions of the pipeline. To achieve all of that, we created a spec-based DAG system. You write code in Python in Flyte Kit or in Go lang in some other Flyte Go kit and it compiles down into a pure specification. This specification says that here's a structure of the DAG and Flyte just records that structure. Now when you want to execute, it uses this specification to rehydrate your code using containers and executes it in Kubernetes.

It's a little different model from Airflow. It also allows building much more dynamic and hermetic systems, and by hermetic here I mean that you can now have a full version trace. We can have a complete history of every execution. You can have a complete history of every update that was done to a pipeline, and all of these can be done very easily without actually recording the code itself because it's recording the intermediate specification that's generated by the compilation system that's part of like git.

This also allows to write different languages, and that's why Ally was saying that we are not really language dependent. We could write it in Go if tomorrow that becomes hard for machine learning, or we could write it in Java because that's already hard. That's one of the other reasons why we decided to move away from Airflow. Of course, we are a container space. That means every execution is done in container that gets some isolation. Prevents noisy neighbors problems and we use Kubernetes. We are Kubernetes-native. Though Flyte doesn't need to only use Kubernetes, but that was a decision we made a couple of years ago that we are currently implementation. We're going to absolutely be Kubernetes-native. Every execution happens at Kubernetes and you get the entire power of Kubernetes given toy you, where in Airflow, that's a very different way of thinking from the way Airflow thinks about tasks and dependencies.

I don't know if that answers – Roundabout answer, but it's a bigger discussion than just probably a talk.

[00:34:22] JM: Was there a point at which you were encountering chronic problems in executing these data workflows where you just said, "Okay, that's it. We're going to implement a brand-new system. We are fed up with this." Can you take me to that pain point?

[00:34:41] KU: Yes. This is 2016, I want to say. I don't actually remember the time correctly. It's funny how you forget the pain points. I was a new lead on the ETA team and one of our engineers had actually written all the pipelines for the ETA team on his laptop and he would run it manually one after the other, really, the steps of the workflow. The first thing that I did when I joined the team was like, "This cannot scale." With all respect to the engineer, it was not ETA knowledge-wise. He was one guy running the entire team.

When I joined the team we're like, "Hey, we need to improve this process," and I was responsible for productionizing all of these. We looked around and we're like, "How do we automate a workflow and what's the state-of-the-art in the open source community today?" Amazingly there was just very few solutions. One of them was Airflow. We had Airflow installation at the company and the ETL's repo.

We went to them and like, "Hey, want to run this as part of your installation." They're like, "Nope. We are already –" They were running only on like one or two machine and [inaudible 00:36:00] underwater. As I said, they were four people. It was a hard time for them. We had no other choice but to create our own Airflow cluster. We created and I spent about a weekend to create my own Airflow cluster and we moved all the engineer's work and some other stuff into Airflow.

We spent about like a month doing all sorts of weird acts. We allowed Airflow to essentially pass data between tasks in some way. Doing all of that, we were able to deliver the model that we wanted to deliver. Don't get me wrong. Airflow is a great concept. It works really well. Flyte actually is very thankful to Airflow, and Max is a great friend of mine.

We did get a lot of influence form Airflow's idea or operators and like things, extensibility and so on. But where we really got affected was that we wanted to launch these models dynamically. The user really wanted to like say, "I want to run this model right now and here are the set up inputs." In Airflow, it's all time-driven. People would pick, run things on a schedule, which is great, but the moment you want to run something on-demand, Airflow doesn't really allows you to do that very well.

The other thing where we started having problems is we had huge models and deployed the first Airflow cluster that was about 50 or 60 machines at Lyft within this team. As the models were computed, one execution would affect the other execution and they started thrashing each other, and that's because there were no containers or things like that. We had to use various Airflow intrinsics to improve that situation, and we did quite a bit with pulling and special queues and so on. But because of lack of isolation, at some point this would come and affect us. Even within one team, this is great. It works fine.

What happened is with all the success of this one team, other teams wanted to use this infrastructure as we had done once going to ETL saying that I want to run my stuff on your Airflow cluster. Somebody else came to me and, “Can I run it on your cluster?” I was like, “No. I get paged all the time. If I run one more thing, I will die.”

At that point we decided to just take the step back and think about what would the ideal system be. This did not say that we would rewrite everything. This just said what the ideal system should be and if we could retrofit all of these in Airflow. The answer to that question came out as no. Then we looked around and we’re like, “Okay, do we find any other system?” We looked Luigi. We looked at a couple. None of those systems fit that requirement, and we saw in evolution that all happening in the machine learning space, which are like, “Okay, more collaboration, more sharing, more understanding of the data, more memoization, cataloguing.” All of the things are the needs of the future and we don’t have a system that fits that need.

Under a lot of pain, we actually decided to embark on a journey to build it, and this was early 2017. We used the first paper we wrote and we actually – A month or a month and a half, we had Flyte 1 operational, and that was I think August or September of that year, because till then we were just debating whether to use Airflow or not and we were just trying to support our current infrastructure.

In September we launched Flyte 1 with having only about a month of effort, and it was hugely successful. It was like one team adapted it. They were able to deliver a model after about a year. First time they were able to deliver a model in a year. Once that success happened, other teams came on and their ramp was phenomenal. The moment we saw that, we were like, “Oh shit! We’ve built something in a month, and people really like it. I think there is something here. Let’s really formalize and build a platform that take those principles and the learnings and actually delivers on our future goals. That’s what Flyte do internally. Externally, we just call it Flyte.

We have actually gone through an evolution. Complete evolution. That’s why it took between 2-1/2 in the making and between running in production for a couple of years. It’s not that we thought about this overnight and we build it. We really spent a lot of time and we had to debate and fight it internally within ourselves to make sure that this is the right approach.

[SPONSOR MESSAGE]

[00:40:49] JM: If you can avoid it, you don't want to manage a database, and that's why MongoDB made MongoDB Atlas, a global cloud database service that runs on AWS, GCP and Azure. You can deploy a fully-managed MongoDB database in minutes with just a few clicks or API calls and MongoDB Atlas automates deployment and updates and scaling and more so that you can focus on your application instead of taking care of your database. You can get started for free at mongodb.com/atlas, and if you're already managing a MongoDB deployment, Atlas has a live migration service so that you can migrate your database easily and with minimal downtime, then get back to what matters. Stop managing your database and start using MongoDB Atlas. Go to mongodb.com/atlas.

[INTERVIEW CONTINUED]

[00:41:48] JM: Was there anything about the fact that you could build – In 2016, if I remember the timeline correct, that's basically shortly after the container orchestration wars had ended. The world had settled on Kubernetes and we could all agree that like, "Okay, this is the thin we're all going to build distributed systems on. Now that we've got this figured out, what else can we build on top of that?" That's kind of – If you compare the pre- Kubernetes world to the post- Kubernetes world, pre- Kubernetes world, "Okay, let's all just use Airflow." But post-Kubernetes world you say, "Well, now that we have Kubernetes, should we reimagine this entire thing on top of Kubernetes?" Was there something about Kubernetes that made this whole flight system way easier to build?

[00:42:38] KU: Flyte 1 was not really Kubernetes dependent. It was container. We actually think the real difference happened with containers just that we could execute containers in isolation. But as we moved on, and that was 2017. As I said, 2017. Mid-2017 to about end of 2017. By that time Kubernetes was maturing, really. I think even though 2016, the container wars ended or the orchestration war ended between Kubernetes and Mesos and so on, I think Kubernetes was still nascent in early 2017.

But by early 2018, in our team internally we started talking. We're like, "I think Kubernetes is going to win, and big win." It's going to change the way we think about many of our systems. We took Flyte 1, which is learning how we think we could create a platform for users that we don't really need to think about machines and how it works for machine learning and data. We were actually searching for the right target platform.

In 2018, Kubernetes happened – Or for us, we discovered Kubernetes. That influenced the complete design of Flyte 2, which is what you see outside. The actual architecture of Flyte is very similar – The open source Flyte is very similar to Kubernetes alert, and that was because of the wins in Kubernetes. But the actual problems that we are trying to solve are because of our [inaudible 00:44:15] with Airflow and our learnings from trying to serve people in data and machine learning. I think maybe I answered your question that, yes, Kubernetes did affect us, but it's not the reason why we created Flyte. It did improve the design of Flyte I would say.

[00:44:33] JM: There are some different architectural components of Flyte that I'd like to maybe walk through a different perspective of an example you've already given. We have these things like there's the Flyte admin service. There's the Flyte properller. I'd like to go through the architecture of Flyte through the eyes of an example. Maybe you could revisit the ETA example and talk about how this is actually executing on Flyte in terms of the architectural components of Flyte.

[00:45:04] KU: Great question. Just for the users or listeners for your podcast, if they want to get a visual sense of all of these, there is a document on our docs in flyte.org and there are a couple of pictures. Our documentation is a little sparse. We're improving it all the time, but I think these two questions are in there. But let me – As I said, as a user of Flyte, you'll write your code in Python using Flyte Kit and you build a container that contains your code. Now, just touching back into the concepts, each Flyte workflow consists of tasks. Each task is nothing, but you can think of it like a Python function. We can have – In Flyte, we can have one container per function. It could be a different container. It could be the same container. It doesn't matter. You build a container or a set of container images. You upload them to a Docker registry and it could be your own private registry like ECR. It could be GCR, ACR or it could be Docker hub.

We don't comment on it. It's something you do decide which one you want to use. Once you build that, we have another tool that allows you to compile your workflows into a specification and upload them into Flyte admin service. Flyte admin service is the control plane of Flyte. It actually stores all these compile specification for the tasks and the workflows in its own inventory system. The system is catalogued using a SQL database, but the actual blob data or the large data is stored in a blog store like S3.

Most of these specifications are really lightweight in terms of like a large workflow with 500 steps. The specification is probably a few hundred to maybe sometimes less than a kilobyte to a few hundred kilobytes in the worst case. So now you can store the history of all the changes in a workflow structure over a period of time in very little amount of data. Now that's done by Flyte admin.

Flyte admin just stores under a cloud [inaudible 00:47:19]. Flyte admin drives, Flyte console and Flyte CLI. Console is where the users go and they can look at the history of all of the previous [inaudible 00:47:31] workflows or they can create launch a new execution. Let's say I want to start a new execution. You select the workflow and a launch plan. We don't have to get into details of that, and you say launch. Because we understand we have a type system, we understand that this workflow to launch needs three inputs, and one of them is defaulted. Let's assume those three inputs are. In the example of ETA, in the traffic example that we talked about, we said there would be three inputs that the user gves. One of them was the range of data for which we have to get the rides, the range of data for which you actually get the locations data from which we extract the traffic. Let's say the third one is the location from where you want to get the OSM, Open Street Maps data. These are the three inputs.

Let's say the Open Street Map data is always defaulted because there's just one place that we could get it. When you hit launch on the launch console in Flyte, it will say, "Yeah, you need to provide the ranges for where I can get – From where I can get the data for the rides and for the locations," and this form is created automatically and the user selects two dates. It's launched and that causes an execution.

What's happening is that execution request is sent to Flyte admin. Flyte admin goes through its inventory, finds the right workflow to execute. Pulls that out, compiles it again, and this compiling

is a second time compiling that it converts to an executable format, and this executable format is now targeted to Flyte propeller, which is a Kubernetes extension.

Kubernetes has this interesting concept of custom resource definitions. Like if you are aware of any Kubernetes resources like pods or services or things like that, Flyte workflow is an extension that was added to Kubernetes if you install Flyte. It behaves just like a pod. If you can interact with it using a YAML or you can interact with it using kubectl, right? If you're familiar with that.

What Flyte admin does is converts that intermediate representation of the workflow, combines the inputs that were provided by the user and converts it to a Flyte workflow Kubernetes extension format and then just basically submits it to Kubernetes. Now when it goes to Kubernetes, kub API understands that, "Oh, I have a thing called a Flyte workflow. Let me see if there is a handler for this guy," and that handler is Flyte propeller. Then it hands it over to Flyte propeller and Flyte propellers that's handling the workflow. We'll not get into the details of how that runs, but eventually, Flyte propeller drives the workflow to completion. As it is driving things to completion, it is sending back the event to Flyte admin saying that, "Hey, you told me about this workflow. This workflow has completed X, Y, Z and so on," and eventually says, "Oh, it's done."

Flyte propeller itself is just the execution engine that runs through the workflow or the graph. But for each of the steps within the workflow, we have a concept of Flyte plugins. You can actually add these backend plugins that extend the capability of the Flyte. For example, I want to teach Flyte how to run Spark. You write a plugin that knows how to run Spark on Kubernetes or maybe Spark on EMR or Spark or Kubl or so on. In our case, we run Spark on Kubernetes. So when Flyte propeller gets to a step that wans Spark, it just hands that control over to the plugin which says, "Okay. I know how to run Spark. It runs Spark on Kubernetes. Give back the control to propeller," and then propeller sends back all the information back to admin. This makes the system completely decoupled because you could potentially go into place by propeller. Flyte admin doesn't care about it.

As long as it knows how to compile and target a format, it can do that. You can also go and replace Flyte admin with your own admin service. Flyte doesn't care about. There's also a third

service that we didn't talk about. It's called the catalogue or the memoization service. Flyte propeller, when it's actually executing is actually talking to catalogue memoization service and recording all the inputs and outputs from every task, whether it has previously seen and does replacing it. That's also replaceable. You could actually go and bring in your catalogue service as long as you can follow the same interface.

[00:52:16] AG: Yeah. Actually, I just want to mention data catalog service. I think it's worth highlighting it a little bit in addition to admin and propeller because it has the potential to save teams and then therefore companies a lot of time and money and just make things more efficient, where basically if you have a task in Flyte, the data catalogue will be intimately aware of its inputs and outputs and when that task was last run.

If it's a sort of task that doesn't need to be run every time or if you ran it with – Let's say you ran it on Tuesday, this workflow, and so it computed some output for that task and then you're going to rerun it again later on that day because you're iterating or just like the next day is part of some other workflow triggering, you can mark that task as cachable, or in Flyte terminology, discoverable, and therefore the data catalog, if it sees that you recently ran this task with the exact same inputs, it will just automatically send over that pre-computed output so it won't rerun that task, and therefore your workflow executes a ton faster and you also save the computation cost that you would have done otherwise.

Basically, if you're always kind of repeating or repeatedly running the same sort of workflow, it doesn't always need to recomputed every task and it's super, super valuable. It helps the teams that we service speed up their iterations a lot faster.

[00:53:30] JM: Okay.

[00:53:30] KU: Our use case for that is like debugging, like one step. Like your workflow is 15 steps long, 15 steps fails. You don't want to pay the penalty of those 14-step which would have taken hours to days in our data processing and machine learning case. You just fix that last step. We write the code. It's not really like resuming the existing workflow. It's running a new version of the workflow. Only one task was updated and Flyte figures that out and it runs

through the 14 steps in like a few seconds or one second and then you just jump with the 15th step and that's essentially preventing, first, saving money for the users and saving time.

[00:54:11] JM: Great. Okay. To draw to a close I know we're up against time, but maybe you could each go through your predictions for the near future of data engineering. I've done a bunch of shows recently on data engineering and there is a lot of change going on and a lot of different ways of doing things. I just love to get both of your perspectives on changes that we're going to see in the near future in terms of infrastructure, open source tools and team structure.

[00:54:42] KU: Yeah. I think we have created an arbitrary divide between machine learning and data processing or data. This is old terminology called ETLs, extraction, transformation and loading. It's the same as machine learning pipelines. You extract the data. You transform it to a model and you finally load it to a service, right?

It's like if I squint and I look at both of them, I see both of them to be two arms of the same structure. Eventually, I feel like data – Or a successful team is where data ML work in cohesion and together and try to deliver the output for the company.

The other bit is that there is really – Like machine learning is really penetrating almost all parts, all acts of data that traditional transformations are no longer existing also. We are using machine learning increasingly to understand data better, and that in turn generates more data that creates new models that generates more data. This is a virtuous cycle.

One of the aspects of machine learning is it seems to be a very hard thing to start using, but it's very interesting to see the trend is that there are many, many models for which you really don't need to understand a lot of machine learning. What you need to understand is your data and you need to completely be well-versed with your data and use that data well with [inaudible 00:56:15] of models and that generates value for the business.

I think there is going to be some grouping or coming together for various teams to – Or alignment rather. Let me say this this way. I think there is going to be more alignment in the future in terms of data and machine learning, and I think machine learning is going to penetrate all parts of data and I think a successful is which brings these two arms of the same beast and

makes them work together. I think that's why you need a unified platform to solve both of these problems.

[00:56:53] AG: Yeah. I'll mostly reiterate things that Ketan already said, but I think one important thing to distress here is that for any modern business these days, you have to have a good wrangle on your data in order to be successful at all and we're seeing this across all sorts of different dimensions like, classically, tech companies as well as companies that are kind of now entering tech in order to survive.

I think it's also really important to remember that machine learning is fundamentally a data problem. Data is 80% right now of the hassle of machine learning, whether it's wrangling and cleaning the data itself or the infrastructure. With that, it's important to remember that machine learning is, basically, it's a solution type for employing data at scale. It's really a vertical, but it is so powerful that we tend to talk about it like it's completely separate and it's just this other thing. But really, ML and data are kind of synonymous when you look at it. You have a bunch of data. You need to use that data effectively for your business. Machine learning is one of those tools for which you can kind of employ that data.

As Ketan mentioned, we really see this space converging. I think that we're talking about machine learning separately right now because it does require kind of new tools, a different way of looking at data, and therefore there's like all these kind of emerging – All these emerging offerings in the space. But the more that we can kind of unify these two concepts and these two spaces and really think of it as one, the more powerful that is.

I think because of that, we do see kind of the need for a single platform both from like a product perspective from the process perspective of data is machine learning. Then also, again, from kind of the roles that individuals play also converging. So, yeah.

[00:58:32] JM: Okay. Thank you both. It's been really great talking, and I'm very interested in following Flyte closely.

[00:58:38] AG: Thank you.

[00:58:38] KU: Thank you, Jeff.

[END OF INTERVIEW]

[00:58:49] JM: When I'm building a new product, G2i is the company that I call on to help me find a developer who can build the first version of my product. G2i is a hiring platform run by engineers that matches you with React, React Native, GraphQL and mobile engineers who you can trust. Whether you are a new company building your first product, like me, or an established company that wants additional engineering help, G2i has the talent that you need to accomplish your goals.

Go to softwareengineeringdaily.com/g2i to learn more about what G2i has to offer. We've also done several shows with the people who run G2i, Gabe Greenberg, and the rest of his team. These are engineers who know about the React ecosystem, about the mobile ecosystem, about GraphQL, React Native. They know their stuff and they run a great organization.

In my personal experience, G2i has linked me up with experienced engineers that can fit my budget, and the G2i staff are friendly and easy to work with. They know how product development works. They can help you find the perfect engineer for your stack, and you can go to softwareengineeringdaily.com/g2i to learn more about G2i.

Thank you to G2i for being a great supporter of Software Engineering Daily both as listeners and also as people who have contributed code that have helped me out in my projects. So if you want to get some additional help for your engineering projects, go to softwareengineeringdaily.com/g2i.

[END]