# EPISODE 1440

[INTRODUCTION]

**[0:00:00:1] JM:** At Lyft, Ketan Umare worked on Flyte, an orchestration system for machine learning. Flyte provides reliability and APIs for machine learning workflows and is used at companies outside of Lyft, such as Spotify. Since leaving left, Ketan founded Union.ai, a company focused on productionizing Flyte as a service He joins the show to talk about the architecture and usage of Flyte, as well as how he is formulating a company around it.

If you're interested in sponsoring Software Engineering Daily, we reach over 250,000 engineers monthly. If you're interested, send us an email, sponsor@softwareengineeringdaily.com.

[INTERVIEW]

**[0:00:38.2] JM:** Ketan, welcome back to the show.

**[0:00:40:0] KU:** Hey, Jeffrey. Nice to see you again and hear from you.

**[0:00:46:0] JM:** Yeah. Last time, we talked about Flyte and the basics of how machine learning orchestration worked when you were at Lyft. Since then, you are working on a new company, which is Union.ai. I want to just start by talking about your mission, which is pretty ambitious, which is to build a user-focused operating system for the multi-cloud era. That sounds great. It's pretty ambitious beyond the core vision of Flyte, which is making machine learning workflows more workable. Can you explain how there's a throughline from a machine learning workflow engine to a user-focused operating system?

**[0:01:40:2] KU:** Great question. I guess, our website is still not really completely designed. Thank you for going through the stealth mode of that site. What we imply and why an operating system, we can think about what an operating system is in the traditional sense. You have a hardware, which is your processor and memory and lots of other peripherals. Potentially, all of them nicely synchronized and working together through this one unified interface, which is your operating system. Without that operating system, every single person out there would have to make sense of a lot of different things. Potentially, that slows down any development that it can really do once you have without this abstraction there.

In here, what you mean is that it's necessary to have the right set of abstractions to make forward progress, because without that, we actually keep on reinventing the wheel and doing many things ourselves.

Now, how does that relate to what we are trying to do? Flyte is not really a workflow engine only. It's a platform that essentially orchestrate machine learning. In machine learning, there can be many, many different things that – or machine learning pipeline involves multiple things of that. For example, we've got, I think last year, a lot of folks talked about feature engineering and feature certain systems and so on. What that essentially implies is that data is really critical. Data is a very difficult beast to conquer. We've spent probably 15 years trying to create multiple different systems and paradigms to conquer that make sense of the data itself. Now, we are up-leveling that further by using machine learning models on top of it to make further sense.

The data ecosystem itself is very challenging. It's lots of different players, lots of different fragmentation, different solutions, and all of them – Some of them fit together. Some of them don't work well with each other. Then now you move from the data to the ML ecosystem. In some way, I think they are very conjoined. The ML ecosystem also consists of modeling. The way you train a model potentially differs from how you process some data set, even though in my head, training a model is another transformation delivering on the data, or providing the data more. If you are classifying, then you're creating a classifier, which is more attractive kind of a transformation that results in – so that it can be played online to get those classifications.

Then once you have models, then you probably go and you want to run predictions, either offline, or you want to run predictions online, or in streaming. You may also want to retrain the model based on triggers, or things that you observe in production, and/or also observe the models as they are running in production and react to the observations.

If you see, these are very different pieces, all of them. There are lots of players already, like develops boom, I would say. There's a lot of players coming up and a lot of different things, different ways of solving a problem. All of them happening. In my opinion, that's more likely to benefit on devices and device drivers. We don't have a unified way of really bringing all of them together onto a single integrated platform. That just works for the user. Now, that was the ambitious goal. When we started it, what are we trying to do with the very, very simple term layman term in-site. We are building an operating system that allows you to plug and play different devices, bring things together and make it unified for the user.

That’s why the name of the company is also Union. Essentially, we don't think we will be building bringing every single piece within the platform ecosystem, but we do definitely want to work with the best of the breed, open source, or process solutions out there to bring them together to our customers in a very unifying way. That's why we actually called it the user operating system. Of course, some of it is going to get updated. We really do mean, that's what we're trying to build.

It sounds a little too ambitious, but it's like many engineers, we think that we are trying to abstract the complexity for the user. Just make operating systems that can be many layers in it. You cannot freckle it. You can have Linux build the next kernel, or you can have a higher layer, which is one, two, or Fedora, or whatever, which have a much more user interface layer on top of it. You can have even higher layers at the top of it. That's how we are thinking of the problem. Hopefully, that satisfies your question.

**[0:06:55:1] JM:** Yeah. One thing I want to ask is, you're interested in unifying the data and ML processes in the same platform. I feel like, the current status quo is some disconnected tools, ETL tools, reverse ETL tools, machine learning jobs in Pandas or scikit-learn, or Tensorflow, and then just Spark jobs, Airflow jobs, all these different things. It seems like a necessity for there to be disparate data and ML processes. Why does it make sense to unify them?

**[0:07:41:1] KU:** I don't mean to say we're trying to unify every single thing under the data ecosystem. It's very vast, right? It's from data ingestion, as in reverse ETL. That's not them. I think, the intention is to unify the processes that matter to a data scientist, or a machine learning engineer, which they do have. They do care about the data, now that data could be transformed after the data processing dedicated to a data warehouse and our data lake and lake house and all of these different terminologies that are coming up.

They care about the data from that point on and then apply machine learning models on top of it, which in turn, generates a lot more data than anything. They have a model, they throw that in production, which you have predictions on that generate a bunch of data, which maybe use actually internal. I've seen this at least in other companies now, which get to time use to generate another model.

This is a factor of layering, and there is interaction, transformations happening from data to model, to data to model. That's the portion that we really, really care about and concentrating on. If you squint, you could generalize some pieces to do other things like, ETL, reverse ETL and so on. The platform that we are willing does not want to really find those pieces specifically, because there's a lot of work to be done just in this – in the machine learning part of the world, which has personalized most importantly.

**[0:09:16:2] JM:** Let's give a review for people who haven't heard the previous episode we did about Flyte. What problem does Flyte solve, and how does it compare to the other workflow management tools that people are probably familiar with, like Airflow, for example?

**[0:09:40:0] KU:** Great question. I think, we did in our previous episode and we talked, or last time when we talked, we did touch upon – At that time, I was at Lyft and we started Flyte at Lyft, probably in 2017. It's much before many of the other orchestrators on the market existed, besides Airflow. We actually started off not by saying that we'll reinvent, unpair one. We said, "All right, we don't want to build this. This is a lot of infrastructure work."

I used to run a model training, or a model creation team. That team was responsible in delivering models that affected Lyft's metrics. We wanted to basically bring efficiency to our daily operations. We were like, "All right. How do I solve this problem?" We looked around. We found Airflow. We actually tried to really, really make it work and to a point that we actually heavily modified Airflow. I think, maybe a few days ago somebody was reminded me of a script I had written, which was called Better Airflow, which will take Airflow and add a few features into it, like dynamic triggering of attacks and blasting data into that and so on. It was all a set of bandages and duct tape being and massive amounts of blood, sweat and tears that actually went in to make it work, but it worked. We got it working and credit where it's due. We were able to deliver what we wanted at a very, very small scale with Airflow.

The problem really happened where it was not operationally feasible than it did, that monstrosity within it. That was one. Then second, many other teams looking at us that wanted to use what we had done. When we actually thought about it, we were like, okay, if there are so many things that are wanting to build and doing something similar to us, this seems to be a more general problem. That really led us that we do a first design of something that we think would solve the problem, not with any intention of implementing. It was more like, “Hey, this is the design. We think this is a solution. Hopefully, we can get some support from the entire team and somebody has to build this.”

Or, if they can think about an existing solution out there that we can use that solves this problem, let's go for it. The result was there was no such solution. Of course, we were a tiny company that had left through us, so we didn't have too many people building it. As a challenge, we built it and we were able to help many, many other product teams in the company and deliver value very quickly. I think, the learnings that we had was that the ML and the data engineer, the ML and the data scientists who used slide, essentially did not care about the infrastructure. The time for procuring infrastructure, like the typical – if you think about a typical DevOps journey, when you are thinking about writing a service. The way we do it is we write the design, system design of the service.

In the system design, we say like, “Okay, this is our database.” This is our service. This is our SLA and so on. We do the entire gamut first. Then we go ahead, building things, procuring the infrastructure for it. Then, we actually make the design. Machine learning is you work it. You don't think about the infrastructures. You think about the problem. You think about the potential, potential that would solve it. Here, I’m talking about model architecture and so on. The data bits that you need to solve it. Then, you don't know if you're actually going to deliver this model in this specific way, because you have to try it out.

You try it out and it may not work. Then you try out another thing. It may not work, and another thing. Then it would work, and you want it immediately delivered. It's an inversion of the way we think in DevOps, where we think and develop. That needs to be encapsulated by the infrastructure itself, because the many, many data scientists and ML **[inaudible 0:13:56.9]**, they’re fantastic for object. Really smart. What they do is probably I can't do in my daily life, because my depth of understanding machine learning is not as much.

On the other hand, if you take the typical approach, we are expecting them to become great distributed systems engineers too, along with a amazing mathematical and machine learning capabilities there, which is just too much to ask for.

What we wanted was to create a streamlined experience that allows them to build, yet to be within the realms of good practices, so that we can have good standards, like on security of data. No malicious code running anywhere. There's always vulnerabilities that happen very quickly and so on. We realized that it's important to split the roles of infrastructure and machine learning. That means that user should not care about the infrastructure, and the infrastructure should be handled by folks who really care about infrastructure, or care about operations. This was a joke, another joke that I make is like in ML ops, I think it's two parts, there’s ML and there's ops and there are two people who care about – one-third of personality that cares about ops and others that cares about ML. In ML, actually bring those two together really. That’s the idea.

The way you bring that together is through the software, not through the practice. You write the software and you can split the two people on two sides of the boundary, where the ML folks can concentrate on the machine learning part of it. While the infrastructure folks can concentrate on the operations and aiding and scaling the planning part of it.

That's what happened is when we started working with Flyte, we essentially made this conscious decision that we will be the infrastructure game for the entire company. The entire company gets to use this as a platform. We realized that many people do not have one single unit of work. They have multiple units of work that they want to tie together into a flow. That's why we have workflow engine in there. It's not really just a workflow engine. It’s more of a workflow automation platform that we do have a bunch of steps. You can run one step individually, or you can write a bunch of steps together. You can run them on demand, or you can run them on a schedule and you can debug them. As a user, you don't really care about the infrastructure management.

It runs serverless from their point of view. Then there's an infrac team that actually manages that platform for you and that runs it for you. That is Flyte in summation. I quickly went over why we came up with this decision.

**[0:16:32:2] JM:** Can you describe the runtime model of Flyte? What does the application consist of? Is it a single node that's just sitting there, helping you schedule lots of other tasks, or is it a multi-node system?

**[0:16:46:2] KU:** Yeah. A great question. That was another thing that we – when we were working with Airflow. We didn't like the single point of failure style. We didn't like the coupling between the code and the scheduler and the workers, the deployment model of it. We wanted people to work on independent repositories. We didn't want the entire system to crumble when somebody imports data flow, or **[inaudible 0:17:14.2]** auditor. We didn't want us to dictate the versions of the Python that the entire company should use, because that really doesn't work.

The model is essentially a service, which is like a native to service in the company. All the functions that Flyte provides and offer through this simple service endpoint. It's not really simple, because there are so many endpoints on it. You can actually dynamically create a workflow just using an API. You don't even need to write code. You can trigger the execution of a workflow. You can observe workflow. You can get historical information. All of this to this one API and it's written in GRPC, rest, so you can actually invoke from any language that provides supports, as well as interact with it to your CLIs and things like that.

This is one of the gateways into the system and that is you're – it's the service. Once at runtime, the execution actually goes to a Kubernetes cluster. We have an engine that runs within Kubernetes cluster and it can run on – it can scale on two multiple nodes at no single point of failure. Kubernetes itself has a very stateful, on a very consistent state store, which is called ETCD. It's based on Raft. It's really scalable and durable. Even in the event of a multiple node failures, it works pretty well.

We use that. We piggyback on that and we store our state information within the ETCD itself. The engine essentially gets a workflow, like a DAG. Or, it's not really a DAG. It's more than a DAG, but from simplistic point of view, it's like a DAG that you can execute. Then, it tries to execute it in an event loop. An event loop here is essentially, every time it gets a DAG, it tries to make as much progress as it can, till it can make no more progress, and then it gives control back to the scheduler.

Progress here is like creating an instance of the work, expected work. An example could be starting a part in Kubernetes, or running a container, or scheduling some work in an external system, like AWS batch, or Sagemaker, or things like that. Then, once it ensures that a progress has been made, it reschedules on the scheduler, finds the next bit to do, makes more progress and then use network.

This way of event loops is actually, there are many storage systems. There are many, many interesting systems that been written by vent loops. The scheduler is not really doing a lot of work. It's essentially just scheduling. Doing it this way, allows it to be – You can really kill the scheduler any point in time and restart it, and it can just come back and start working.

The only real thing that actually affects the scale of the scheduler is if that is a slower right to port that you get from some storage to sub-system. To scale from that, we actually do sharding within the storage subsystem and so on. We have seen that even one process can run about 10,000 concurrent workflows. Flyte allows you to scale that across with multiple nodes within a cluster, and so you can run about a 100,000 or so without any problem.

Then, I think Kubernetes itself would start having problems. That's where Flyte allows you to run multiple Kubernetes clusters and browse between Kubernetes clusters. The idea there was, because you are – if you are an infrastructure team, you don't want your users – You may sometimes have to isolate your users, because there's a very critical use case and they are like, "No, no. We want to run every 10 minutes and we don't care. We are ready to pay the cost for it. You want to dedicate a cluster for them. You might want to isolate their failure node. In other cases, you may want to swap out clusters on the fly, because you do routine maintenance on your clusters, that even in Kubernetes. We have seen cases running up to patch the node. You have to patch even the API servers and so on. Flyte allows you to manage all of this transparently to the users.

It's a vertically integrated workflow automation platform that does not really have a single point of failure. It's scalable to multiple Kubernetes clusters. Instead of a node, or 10 nodes, it can run on thousands of nodes. At Lyft, we had about 5,000 nodes in our cluster that would run on a Spark jobs at Lyft's customers. All of machine learning training, as well as many, many data processing jobs. Some were offloaded to external systems as well.

**[0:22:02:0] JM:** To get a better-grounded understanding of what problems Flyte solved at Lyft relative to older systems, can you describe a particular machine learning job that ran on Flyte, or runs on Flyte and how it's utilizing Flyte?

**[0:22:24:1] KU:** Oh, yeah. Yeah. Let's take an example. Maybe I'll go through two examples, slightly different ones. One of them is, I guess, a team that I was leading. We'll talk about that. It's open, because I've worked on it the past. It's ETAs. From the end-user point of view, when you open up your app, you get a number, which is the ETA, estimated time of arrival, in the Lyft app, I mean. This is critical, because it actually is important to be accurate, or close to accurate. Otherwise, the user loses trust in the system. Then the second part to that is that when you also get that, you get an estimated time to destination, which is used to drive the price for your right. Both of these are critical from the user point of view. Both of these are models that are trained on Flyte, or at least, at some level.

It's not just one model. It's a bunch of models that actually play in together, but I'll talk about one model. It's trying to guess the traffic that's currently, that Lyft can observe currently on the road. Essentially, trying to estimate the road traffic situation, using the drivers who are driving around, because we are getting head props from them. We know that on this road, these cars are taking two times longer than the previous – than the default average on that road. That can give us that there is a slowdown. Now, if I say that if I route the person through this road, then instead of the default 10-minute, it might take – it's being on 30 minutes, depends on the model.

This situation changes every 10 minutes, or every 3 minutes, maybe every minute. We also observe that changing too rapidly doesn't really improve the signal. Changing at a good cadence, does improve the signal. What we did is we got a lot of data in real-time, which was from these drivers, which we can order to road signals. Using that, we generated the model. Here, the model is essentially waits on the road network and the bias adjust model that would adjust, or based on the variance of the bias that we had in the system. A baseline model that would help make sure that our decisions are never – do not deviate from a norm standard. These three models together, were trained on Flyte.

One of them was trained every 10 minutes. One of them is trained at a monthly, or a weekly frequency. Another one is trained at every day frequency. Together they get deployed to a service in the end.

Another critical part of this was when I'm about to make a change to this model, how do I verify that my model is correct? What is good? You can of course, on AB test, but that's too late. This is like taking a bar example and giving it to your students to verify it if it works or not. I's not really what you want to do with your customers all the time. Flyte also runs the simulations for making sure that the model is good enough before deploying to production.

Now, granted that these simulations are simulations, so they will never catch all situations, but they do catch some really, really bad off of a – heavily and characterized, or completely outlier-like models. The simulation works. It was like, okay, take the last three months of data, take the first month as a training, the next two months as the validation and run the simulation with all these three models being deployed and using different capacities. We used to run the simulation. That was also another reason why we built Flyte in the way we did it, when we had a lot of ad hoc, large-scale jobs running.

Simulations are extremely large and time-taking. This is an example. Another example is Lyft trained, like builds its own maps now, for example. Then we have another user, black shark, which is creating a digital twin of the world maps. They use a lot of machine learning to essentially extrude the buildings and the road network and create a three-dimensional artifact from the satellite view of the geography.

Lyft does not do that. Lyft users often treat maps and some other things and collects them and uses machine learning to actually match things together and create a world map. Not a world map, but just – US and Canada map. Both of these are extremely data processing-centric. They may use Spark. They use a lot of Python and Java processing, but they also use deep learning models in them. This fusion of data and ML, that's really shaped in this, another use case learning. Folks use Flyte and actually get a lot of value, because these teams consist of – They do consist of experts in Spark, experts in machine learning, experts in geospatial engineering and so on. Then Flyte allowed that, and Flyte brings all of this together to build a cohesive outcome, which is essentially a map in this case.

**[0:27:56:0] JM:** When you look at productizing Flyte into a service that is bendable as a product, does it look just like the platform that you wanted to make available to engineers inside Lyft, or is there some alternative route to deployment and monetization that you're taking?

**[0:28:24:0] KU:** Fantastic question. Some of them may be under stealth. I think, it would be hard to disclose everything. I think, it can be done in multiple layers. There are three parts to this problem. One of the parts, and as I said, we wanted to separate out the infrastructure folks from the folks who actually do the real work of writing the business logic. This works great with larger companies, and that's what is happening at the moment with the adoption of Flyte. It's adopted by all the companies, the talk leaders and folks who have a decent engineering strength. That's because running, even though Flyte actually is a very reliable software and it's battle tested, you still have to run Kubernetes. You have to have knowledge of then, sometime of Kubernetes, even if you're using GIFs, there's some knowledge that you need to have.

As a company offering, we actually want to help our users achieve their goals without really having to worry about the infrastructure. Union, essentially, is poised to be their infrastructure partners in that sense. That's one part, definitely a part of the play. Second is I think, certain things to do in open source and Flyte is – We pride ourselves on being completely open source. We actually donated to the Next Foundation just last week. It was graduated to adopt the other project, in the Next Foundation. Certain things to do in open source are extremely hard, just because the scale characteristics are – sorry, the deployment characteristics of fake how you would build certain systems.

We think in Union, we can actually be certain systems that are really hard to build in open source, just because nobody want – global deployment, and you want like, multi-region availability and so on. Those things that just went back to building open source. We will offer those at additional services. Some aspects are around sharing. Anybody who has a Flyte cluster can get more out of them. Whether they are using the open source, or the Union variant, they can get more out of it. That's the thought angle that we think we have to offer. I think, this way – I don't know what the monetization is that we would do once over here, but essentially, we think we can add value to all our users and new users of Flyte and Union platform.

**[0:30:55:1] JM:** When you take a existing set of disconnected workflow operations and move them on to Flyte, what are you getting out of it exactly? How is it improving your usability?

**[0:31:18:0] KU:** Yeah. Fantastic question. Actually, somebody asked me this, or asked this question in Slack yesterday, or the day before yesterday. The question was very simple. It was like, “Okay, I wrote all of these tasks in the workflow. What's the benefit? Why am I doing this?” The answer, and this was the simpler part of the question. Then the other question that you have is like, disparate parts and connecting them. Let's just answer why to break a large problem into smaller pieces.

One, it's a great way of documenting your software, like just to break it's large pieces. We usually break it into functions. It improves reusability. It improves shareability dramatically. At runtime, what it really improves is failure – improves resiliency, because let's say, you have three parts, three things that you want to do in sequence. If you fail at the second part in a traditional normal piece of software, you start from the first part again and after. There's no other way. You press a piece of software. You start with the first part, then you go to the second part, then you go to the third part. Now, you can solve that by doing checkpoints. This is why a checkpoint is created.

Checkpoints are not trivial to implement and correctly even more so. Workflow engines are essentially a way of checkpointing your progress. Because in workflow, it means you are guaranteed to never go back in time. You only make forward progress. in case of the failure, let’s say the failure was transient and you can retry a few times and the failure goes away. This can be easily accomplished by using something like Flyte. You can just put a simple decorator on it and say like, “Oh, I'm okay with failures. Retry that.” Another piece is, there are two aspects to this. Now, some of these failures are just infrastructure failures. They don't even retry on that. Now, you've lost progress on it and there was no other way, recourse, but for Flyte, to say, that this workflow cannot make any more progress. Sorry, you have to stop it.

This can be like, let's say, eight of this goes down. A crazy situation, but actually, they made of this in Norwegian goes out. Then, when you want to recover, you have to restart it, because you've lost all progress, even all kinds of things that you've done. Flyte essentially allows you to fully recover from a previous execution. Again, step to a point where the last failure occurred and drive everything ahead from that point on in, which is for critical application, this is a required sense of reliability and resiliency that's required.

Then, the third aspect is that you treating and specifically in machine learning, you have the three steps again. Step one, you ran with a play that you do, just long and it's costing you a ton, maybe on Snowflake, and you now have the result. But your model had a bug, and you fix that bug. Now, should you lead on that query to refresh that data? You shouldn’t. You already just fetched it, like maybe a day ago, or 10 minutes ago.

Flyte memorizes results of previous executions. This is an open feature, of course. When you open, you have to tell Flyte that this particular execution is deterministic, hermetic, and does not have side effects. It'll minimize the results of that execution. Every time it sees that execution anywhere in the platform, now, it's not just you. Maybe there are four other people working around the same query. You minimize those results and reuse the results from that one single execution across everybody. Now, this has huge cross-selling advantages, as well as performance and efficiency benefits, especially they take model of – a new model of trying out new things. Another aspect that you said is like, what about disparate systems?

Disparate systems communicate in different way, then I’ll give you an example. Most data scientists are used to writing Pandas data rates. Sometimes for data processing, you have to use Spark. Spark uses its own concept of data frames. Then sometimes, you want to actually take that Spark data frame and send it to a model, which is Pandas dataframe-based model, which is on one single machine, and that's good enough. For most model in one machine with a couple of GPUs is pretty good for many models to get started with.

How do you translate from a Spark data frame to a Pandas dataframe? Yes, there exists technologies like Arrow and there exists a way to hand off and to parquet as an intermediate format and then load it back in. This is all complicated for most users. They really have never dealt with it. Have never thought about it. In Flyte, we have canonicalized this into a schema, into a type of schema system. If Sparks generates a dataframe, you can easily consume it into a Python dataframe, or even in Java independently, without thinking what how the translation is happening, where the data is going to, what's the intelligent representation, and so on. We take care of everything within the underneath there.

Thus, it becomes a substrate that allows you to connect pieces, even if there cross-disparate systems. Yeah. Then there was this one more aspect actually to it, which is automatic parallelism. When you generate some data set that you're consuming across three functions, even if you write them linearly, let's say, Python, they will get executed linearly in Python. In Flyte, it will see that this dataset is ready and now it's consumed by these three different functions. They can run all these three functions entirely. Flyte analyze them and run them and distribute it, setting on multiple nodes. That gives you efficiency, as well as performance benefits.

Centralizing the platform of using Flyte, now again, if you are a data scientist in a company, if you have about tens of data scientist in that company, centralizing on one platform has a huge advantage. You can run GPUs on that one centralized cluster. GPUs are extremely expensive and reuse GPUs across all the data scientists, because Flyte is a multi-talent central holistic platform. It's very easy to like, “Oh, this guy uses a GPU right now, so I'm going to give him the GPU, part of the GPU. That person who uses the GPU then, that person of the GPU. You can just have one GPU and use across the day through – Flyte will also manage that pressure and queuing and all of that for you.

You can just manage your cost. You can put a budget on the Flyte center cluster and it can manage your cost for machine learning. Along with that, you get a Spark machine, then interruptible machines and Flyte also manages them very well. That actually has an effect of reducing your total spend on training machine learning. I think, the number one problem and actually, once you start really training a lot of models, these costs becomes really, really expensive.

**[0:38:27:1] JM:** Can you tell me about how the API for using Flyte has evolved? If I'm interfacing with Flyte, what exactly are my specifying to the platform and where are the points of integration?

**[0:38:45:0] KU:** I think that’s a fantastic question. You've done your homework. Last time when we talked, the API for Flyte was also Python-based. We also have Java and it's called a base API, but I'm going to call it the Python-based API, because that's actually the most commonly used one. The API was written in 2017. Remember, I told you that we actually built it for Airflow, and then we moved on to this called the step functions and different problems and we moved on. But the API did not change, and was written before Python 3 was a thing. It had a lot of boilerplate and folks had to get used to writing code in Flyte way. We realized that's not what users really do. Most Python engineers know Python and they write Python for it.

We did not want to get into the way of these folks until early 2021, we released a new version of the Flyte Python SDK, on Flyte kit. This API essentially is – still, as if you’re writing pure Python. This is just, if you remove any of the couple Flyte decorators that you add, it is a regular Python code and you can just run it as if it’s Python. Then, if you add those decorators, it becomes into a Flyte workflow. Including the input and output types, you use the Python depending system to essentially tell Flyte about what your intentions are about the data. For example, you say, “I want an end in a string in a Pandas data,” Flyte will automatically understand what do you want from it. Once you register with the system, you can actually generate – We generate a line form for you that's getting indeed, or in a string kind of file, which is a parquet file that contains your Pandas dataframe equivalent. All, this is automatically written in Python.

You do not try to get away from Python. It just provide that. Similarly, in Java and starlights, where you need to write in Java and starlight code. Now, the other aspect in Flyte is extensions. Let's say, I want to run a distributed training job using API. For a regular person, this may seem very daunting if they’ve never used API before this. With Flyte, you essentially BIP install Flyte kit plugins API. Of course, you have to have a back-end plugin installed in your Flyte deployment.

Once you do that, anybody in the team can write an API job, just like writing Python code and a context is given to you, which has the API set up for you. It's in the same Python code. You now have MPI running. You can, of course, run all of this locally, because it's all still a python code. You can run it locally. Of course, you want to get multiple machines to as if to train on distributed model, but you will be able to simulate the MPI as a single-node process. Then when you train it on remote, you can scale up to hundreds of millions, whatever you require for your training job.

The goal of Flyte kick, which is the SDK for writing Flyte jobs is essentially, to simplify the user interface. Just make it as simple as possible for users to write their Python code, and let us try and do the hard work of understanding and converting it to a distributed system.

**[0:42:16:0] JM:** Do you have any reflections on running Flyte intelligently? I'm sure you're learning more about this as you're turning into a service.

**[0:42:30:0] KU:** Great question.

**[0:42:31.9] JM:** In terms of the infrastructure, like what the deployment model is.

**[0:42:37:0] KU:** I want to talk about how we are doing it in Union at the moment, but we'll talk about all the learnings, because we've been running it for a while. There are lots of users now who've been using Flyte. I think, folks, or companies that use Flyte are often looking for a solution that is reliable, resilient and secure by design. Security is often an afterthought in many, many systems. Flyte, it's not an afterthought. It was a first level thought. That makes it harder for certain users to use it, because you have to use an IM node for every single execution, for example, for AWS. Or you have to have the right set of secrets, and there is just no way – We don't allow secrets to be passed in plain text ever in the system.

If you have secrets, you have to pass through a proper secret channel. These are extremely critical as a business, because we have users relying on it. In the back-end, when you're running it, you now have to run a Kubernetes cluster and potentially, have to run a secret manager like Walt, or you can use a hosting service from this specific manager or whatever. You can use Kubernetes-based operators to do certain things. For example, you can run, spin up a Spark cluster dynamically within Kubernetes using Flyte. Or, you can offload those jobs to something like Databricks. So that if you use Kubernetes to run all your jobs, then as an organization, it's better that you have some Kubernetes DevOps expertise to essentially maintain the scale and the capabilities of what it is.

There are cases in which it doesn't – it's not designed for really, really large-scale number of our creations and so on. Flyte provides a lot of tools to essentially allow that. Let’s say, if you have 10 users in your company using it at the same time. If all of them hammer you with thousands of jobs, it's going to cause a problem on the device cluster. Because of our years of experience with Kubernetes, we haven’t called out this into Flyte back-end. We do automatically handle back pressure. It will do resource pulling. It will try to queue up things and so on. At some level, to understand all of this, you definitely need to understand the problems of Kubernetes, in order to adapt rehabilitation and then there will be supports in our documentation. Please, bring it up in the Slack channel, in the open-source Slack channel.

Another thing is we highly recommend users one stake code, or some skill to have more than one Kubernetes cluster within the Flyte logical cluster. That is because if one Kubernetes cluster goes down, you do not want to lose availability for your users. At Lyft, we use to run with 13 clusters across 5 to 10,000 nodes. Yeah, so these were the learnings we have. We ship with a lot of observability dashboards, lots of tools to give up things. Those are probably more suitable for DevOps and shop owners.

**[0:45:54:0] JM:** Cool. Well, as we wrap up, do you have any final reflections, anything else you want to add about the engineering of Flyte, or just other things you've learned about company building, or –

**[0:46:06.8] KU:** Oh, company building, a lot. I think, probably, I would love to write my blog one day as an immigrant, starting a company and stumbling across different facts, and building a culture and how to really be intrusive and completely honest company, because those are the values that I believe in. I would loved to. I think, I would be writing a blog at some point. Don’t quote me, but I definitely want to.

From a engineering point of view, I think we realized a few things and we have a long-term roadmap for Flyte and we see a lot of things. We are working on layers that actually help certain types of users to further explore and express their ideas much more succinctly and much more clearly and easily. Stay tuned. Something’s coming soon. We’ve realized that many people have tried to push the boundaries with Flyte and we’ve been extremely focused on correctness, reproducibility and reliability. While there are some other players that were not really focused on those parts, but focused on purely on the user experience.

What I’ve realize is user experience is the key. It’s the key to everything. We are working heavily on the user experience portions of it, and we’re working with some users who are trying to push the boundaries of Flyte, and essentially, trying to run second workloads that are fully reproducible, that’s fully lineage-driven. We are working on those really interesting engineering challenges of building up almost real-time serverless platform that scales to user’s need, without touching it from the user site.

Yeah. There's a lot to build here. This is a fantastic opportunity. If anybody who's listening and we are hiring in all sorts of engineering roles. Would love to talk to you. Join the Slack channel. Say hi to me. Send me an email.

**[0:48:13:0] JM:** Awesome. Well, Ketan, thank you so much for coming on the show. It's been a real pleasure.

**[0:48:16.0] KU:** Yeah, same here. Thank you, Jeff.

[END]