EPISODE 1087

[INTRODUCTION]

[00:00:00] JM: Apache Airflow was first released in 2015, introducing the first popular open source solution to data pipeline orchestration. And during that time, Airflow has been widely adapted for dependency-based data workflows. A developer might orchestrate a pipeline with hundreds of tasks with dependencies between jobs in Spark, Hadoop and Snowflake. Since Airflow's creation, it has powered the data infrastructure at companies like Airbnb, Netflix and Lyft, and it's also been the center of Astronomer, a startup that helps enterprises build infrastructure around Airflow.

Airflow is used to construct DAGs. DAGs are directed acyclic graphs for managing data workflows. Maxime Beauchemin is the creator of Airflow, and Vikram Koka and Ash Berlin-Taylor work at Astronomer. They all join the show to talk about the state of Airflow, the purpose of the project, its use cases and the open source ecosystem. If you want to reach 30,000 unique engineers every day, consider sponsoring Software Engineering Daily. Whether you are hiring engineers or selling a product to engineers, Software Engineering Daily is a great place to reach talented engineers, and you can send me an email, jeff@softwareengineeringdaily.com if you're curious about sponsoring the podcast, or forward it to your marketing team. We are also looking for writers and a videographer. If you're interested in working with us, you can also send me an email, jeff@softwareengineeringdaily.com.

[SPONSOR MESSAGE]

[00:01:38] JM: JFrog SwampUp is an online user conference with more than 30 sessions from cross-industry expert, including Google, Microsoft, capital one, Adobe and more. You can get a unique look into the broad DevOps market, not just point solutions. There will be tracks for cloud native DevOps, enterprise DevOps, DevSecOps and digital transformation.

Participate in expert DevOps training classes across DevOps tools, security, CICD and more, and you could join from two time zones, June 23rd and 24th for the Americas, and June 30th and

July 1st for EMEA and APac to suit daylight hours across the globe. Go to softwareengineeringdaily.com/swampup to learn more and sign up

JFrog will be donating all conference registration proceeds to COVIT-19 research. Go to softwareengineeringdaily.com/swampup and check out SwampUp.

[INTERVIEW]

[00:02:42] JM: Guys, welcome to the show.

[00:02:44] VK: Thank you. It's good to be here.

[00:02:46] MB: Thank you, Jeff. It's good to be on the show again.

[00:02:48] JM: Yes. We're talking about Airflow today, and I'd like to start by just giving a little bit of background on Airflow and then we'll get caught up to the present and speculate on the future a bit. Max, you started Airflow back in 2015. What was Airflow originally used for?

[00:03:08] MB: The origin story is a little bit long, so I'm going to go quickly through it. But originally, so I joined Airbnb with the idea of coming and building this thing and open sourcing it, and I was coming at from Facebook at that time where, internally, at Facebook, they're building all sorts of internal tools and they had multiple schedulers. And there was clearly one that was ahead of the pack and out of all this experimentation on schedulers, this tool internally called Data Swarm, which is not open source and not viable to the public, was coming out as something that everyone wanted to use. It was really popular. I took some of the ideas from Data Swarm and some of the ideas from 10 the years at that time of ETL and kind of pre-data engineering, data engineering. Like data warehouse, like years and years of doing data warehouse architecture and ETL and I decided to go ahead and build this thing.

I think within a few months, we're up and running with like a very early version of Airflow, and it was all about – The first version was mostly about scheduling, data processing jobs in places like Hive, Spark, Dash Script, Python Scripts. Really, taking all of these little workloads and tasks and orchestrating them in DAGs and directed acyclic graphs. That was like the very kind of first, first kind of set of use cases for Airflow. We had a different DAGs for different – I call it data marks, or different subject area in the data warehouse. That grew very quickly to support all sort of other things. Airflow turned out as a really good distributed kind of cron scheduler, a batch scheduler.

Internally, at Airbnb, it grew to serve service all sorts of other use cases. So things like big data frameworks, database scrapes and export, AB testing, doing all sorts of like data-centric or data bit dashboard-centric kind of prep and munging. And then all sorts of themes that needed to schedule batch job kind of hopped on the bandwagon. Teams like doing pricing, fraud detection, data infra, maintenance, anomaly detection, metrics repository, future repository. It grew kind of over many years to support all sorts of data-centric use cases.

[00:05:37] JM: It's obviously been successful. It’s had lots and lots of use cases since being open sourced at lots of other companies. To talk through a bit the usage, Airflow centered around DAGs. Could you give an example of a typical DAG or a directed acyclic graph that might represent a data pipeline?

[00:05:58] MB: Right. I think like if you think of something like a batch scheduler, like cron. If you think the most basic of all, if you think of like what is like a typical cron job. There's no real good answer for it, because you can really use cron to schedule just about anything and it’s kind of the case with Airflow. But I think like Airflow caters to data engineers so much, because just the sheer amount of tasks that they need to kind of juggle with and the number of plates that they need to spin and keep spinning.

A very typical place to start is what I would call kind of the DBT use cases of like orchestrating a lot of like SQL queries, but quickly beyond that you want to schedule a handful of Spark jobs, or scraping some SaaS services or getting data from other sources. I think a typical DAG might be kind of a subject area of your data warehouse.

At Airbnb, it could be customer experience data, for instance. It would create a DAG that would get data from all sorts of SaS services that we’d use there, like Zendesk, Genesis, likes call center data to getting information, in-app kind of information, and gather all these data and bring it to a central kind of data mart. That’s a typical use case.

Another one at Airbnb that we've worked on early on was called Core Data, and that was essentially creating a lot of like a certain number of fact tables and dimension tables for all of the core data at Airbnb, like bookings and users and things like that.

[00:07:39] JM: Has the usage of Airflow gone beyond the scope of what it was originally intended for?

[00:07:49] MB: Definitely. I think so much so, I think originally, you kind of scratch your own itch and then people defined what actually the thing can do, right? I think it's been definitely pushed in all sorts of direction. Some of it we kind of embraced and not only like allowed, but also embraced. I think in some cases, we had to build an immune system to make sure that Airflow could support some of these really intense use cases too.

[00:08:22] JM: Take me forward to today. If we talk about today – I just tweeted this question. You thought it was a reasonable place to start. This question of the fact that Airflow is really popular. It's used by a ton of people, but there are some competitors to it today. There are some new entrants. Why is there some competition? Why are there alternatives to airflow today?

[00:08:51] MB: Let me try to take a stab at this one, and I'm curious to hear the others too. But I think part of the success of Airflow is because the contract that it provides or the contract that it offers is really simple, right? It's like express your tasks using our operators and define your DAGs and we're going to go and run these tasks. There's a bit of a separation of concern here where Airflow doesn't know too much about what your actual data flow are doing, right? It's like contract is largely based around tasks. That's a fine abstraction and something that people can comprehend and live and work with very well.

I think a lot of people are looking at Airflow or something like Airflow and think there's so much more it could do potentially. It could know so much more about the metadata in your system. It could know more about the nature of the jobs that are taking place in the system. I think like the beauty of Airflow is like a lot of people are building on top of it, kind of second layer things and frameworks that do some of these things. Kind of on that base contract, they build upon the base contract to kind of jam in their own business logic or their own frameworks on top. I think people that are looking at this space think that there's an opportunity to maybe provide more guaranties in a way that you'd probably need to have more constraints kind of upfront, but provide more guaranties and maybe be more opinionated on things like how should we do data quality management and data quality assertions? How should we do certain things like type enforcement or inference? Airflow is a little bit agnostic to a lot of these things.

[00:10:37] VK: Well, if I could just chime into that, because this is such an interesting topic of discussion. One of the things I loved about Airflow as I was initially exposed to it was an expression of data pipelines as code, and that is one of the first time I had spoken with Max about it as well. I think it's like the evolution of anything is code in any particular programming domain or any particular domain overall. So there's a nature once there's a successful element of like at the base level of code, which is starting to be defined, and then it's successful. There's a natural level of like building abstractions on top as well as filling in the other elements of the software lifecycle.

I think that's completely natural with the evolution of what we're seeing with Airflow, is for certain abstractions on top, things like SQL or other things, or certain elements in the lifecycle, then people want to be able to add into it. I think that's a really good thing, because that also reflects the maturation of data engineering that's a practice and about different patterns within data engineering, and then particular elements of tools or lifecycles or flows which make those particular patterns easier across the broad data engineering spectrum as a whole.

I think that's – Just we've seen this consistently across the board in different software engineering domains, and I think that's really what we're seeing with data engineering as the domain using code as the base level infrastructure for solving problems in this particular domain.

[00:12:14] JM: Right. The infrastructure as code idea applied data pipelines might be one way to think about it. I'm still wondering. So, the newer data pipeline management things like Dagster. Nick has been on the show talking about Dagster, and Prefect is another one of these data workflow schedulers. How do these alternatives to Airflow compare?

[00:12:47] MB: Well, I think one clear line that they crossed that we don't cross directly is getting into the data flow space. You can kind of think of a workflow scheduler or an ETL tool as having the workflow editor, which is kind of place that where you define what are the tasks that are going to run and what are the dependencies across these tasks. Then there's the dataflow editor, which is kind of getting inside a task in defining what is this task going to do.

I think for Airflow, originally, we were really interested in solving the workflow part of the problem and letting other system, orchestrating other systems that are really good at doing distributed data processing, systems like originally MapReduce and Hive, but Spark and PySpark, and all of these other systems that are built that are super solid, distributed, fault-tolerant. We decided quite consciously to not get in that space for the reason that it's a very complex area and these systems are really specialized and it's kind of a slippery slope too. If you try to redesign Spark or something like Spark, there are so many so that you need to think about that we decided we're mostly out of context for Airflow originally.

I think some of these challenges are things around budgeting and fault-tolerance. If you do go and allow data scientists to spin up infinite number of task instantly, it's kind of great for them and they're going to be happy and that's probably what they're going to demand, but they can do a lot of damage just in terms of budget or just in terms of creating a lot of contention too. I think we decided to call out to the systems that do these things well and orchestrate them as supposed to making it super easy or supposed to start building a competitor to these systems also.

[00:14:43] JM: If we talk a little bit more about modern DAGs, let's talk about an Airflow DAG. If I am executing airflow dag, it's got a Hadoop job, A Spark job, Snowflake. You've got all these different systems that it's integrating with. How is Airflow communicating with these other data systems that make up a workflow?

[00:15:10] ABT: The primary thing that Airflow uses to speak to other services, your Hadoops, your Sparks, your interface as a DAG author is operators. So kind of operators are the tasks in your DAG, the nodes in your DAG. DAG being directed acyclic graphs. So you can branch it with that. Each node in that graph is a DAG, is an operator.

An operator, when it comes down to it, is a Python class with an execute function. It's slightly more complex than that, but that's basically it. It's a class with a function that Airflow knows to call. That then uses Python libraries to speak to Hive, Hadoop, Spark, whether that is the built-in libraries to go make a network request or kind of built-in publish client libraries. PyHive, doing some Hadoop stuff, things like that. It's just kind of we try to build on existing libraries whenever we can. That's kind of one of the strengths of Airflow, is we've got all these hundreds of operators for all kind of like common data processing and data manipulation tasks from simple things to SQL queries, run this thing, do a transform, take this file from S3. The list goes on and on, to EMR, Dataproc jobs, and Databricks and kind of – Yeah, list longer than I can even remember.

[00:16:36] MB: If I can chime into it, an operator is a little bit like a task factory. It's a more or less a class that you instantiate to get a task object. Part of the value proposition with Airflow is that it already knows. It has already all these abstraction to talk to all these external systems and it makes it really easy for someone to say to orchestrate these things and say, "I'm going to instantiate a Hive operator and just pass it to SQL. I'm going to instantiate a Spark operator and just pass it or reference to my entry point, to my Spark cluster." Then you simply have to say like, "What are your tasks and what are the dependencies between them?" Essentially, that provides you with a DAG object.

The other value proposition too for people who are less familiar with Airflow is the web server that makes it really easy for you to see what's going on, to see the status of your job and see the status of your DAG. You'll see very clearly which job ran where? Whether it succeeded or failed. Whether it retried? You'll see kind of your graph representation of your DAG visually and different views to really understand like what's happening with the execution of my task and my DAG.

[00:17:52] JM: The DAG is calling out to scripts and calling out to APIs?

[00:18:02] MB: Mostly, it's an API. It's not necessarily a script. When you define a DAG, you're in a Python module where you instantiate an Airflow DAG object and a certain number of operators to create tasks. A typical Airflow DAG definition module will look like you'll specify a DAG object, specify a certain number of task where you'll pass in your relevant parameters.

Something like, “Hey, I want to instantiate a –” It could be a snowflake job and I'm in a past it a connection ID to what is a reference to what is the way that you connect to your Snowflake instance with some SQL to run. Then you could create as many as these tasks as you want and define in which order to run those. Which ones are kind of parent or dependence and dependencies or upstream and downstream tasks.

As you do that, what I was saying earlier, that Airflow does not necessarily too much about your data. It knows about your tasks. Really, it is the king of understanding what your tasks are and in what order to run them. But as a data engineer, you know very clearly this task A is probably loading some data in the table, and then this task B is following up on task A and knows where the data should be, because it's deterministic.

Yeah, the DAG code is really instantiating these operator objects. You can define your own pretty easily and they can be Python operators, Hive operators, Snowflake operators. You can orchestrate your whole workflow. Once you commit that, the Airflow scheduler and Airflow cluster will start playing this symphony basically of orchestrating all your tasks and making sure that everything runs in place. As an operator, you can easily see in the web server where you’re at. What’s succeeding? What’s moving forward? How long it took? All the information that you need as a data engineer to monitor your workflows.

[00:20:05] VK: I think one – If I could chime in. I think one additional element which Airflow does really well is this notion of separation of concerns. I think the DAG itself fundamentally has effectively the application logic of what needs to get done and what sequence and what dependencies. Then there is the importing or referencing off the connection information about how to connect to this particular source or how to connect to a particular database or how to connect to something like Salesforce? That is completely separate so that that can be pulled in at run time. That can be potentially changed by a different person who manages credentials or different element in the organization. Then finally, resources are, again, can be managed completely separately.

I think one of the things which Airflow does really well is that separation of concerns from purely the space of the application developer or DAG developer versus the infrastructure concerns of how do I connect to this particular system? How are these credentials stored and managed, potentially, centrally or for like security concerns? Finally, how are these resourced and managed from a resourcing standpoint and keeping them independent of the application logic?

[SPONSOR MESSAGE]

[00:21:26] 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.

[INTERVIEW CONTINUED]

[00:23:15] JM: When we're talking about an Airflow DAG, we're obviously talking about a distributed system that's plugging into a lot of other systems. I can imagine this being somewhat difficult to set up and configure. Is there's anything that's particularly hard about setting up an Airflow cluster on cloud resources?

[00:23:35] MB: I could take this one as I've been an operator of the Airflow cluster early on at Airbnb and I quickly kind of needed some help to operate the cluster and all the connection. When you're at that federation level or this orchestration level, you'll typically see to all the problems that exist in the whole like data platform, data infrastructure too. It can be fairly challenging to manage such a system. You also have all sorts of use cases and people using the system in all sorts of different ways too.

I think, back in the days, we didn't have Kubernetes. We didn't have like Helm. I think it was harder where you'd have to start from probably like EC2 plane if you're on AWS at the time, and there's a certain number of components that you need if you want to run airflow in a distributed kind of way where you have multiple workers and you have a scheduler, you have a web server, you have a metadata database, you have a communication layer typically using Redis so that the scheduler can talk to the workers. It's a fairly involved thing to manage a distributed system. Nowadays, we've seen the rise of different ways to run Airflow within Kubernetes, for instance, that that makes things somewhat easier.

[00:24:58] JM: Got it. If I set up my airflow cluster, what are some of the ongoing support issues that I'm going to encounter? What are the kinds of failure cases and difficult management issues that might come up?

[00:25:14] : One thing that's very clear is that whenever any system on your data platform is going to fail, whether it's Spark, Hive, MapReduce, the name node, you're going to know very quickly as people will just come to the Airflow channel on your Slack or whatever communication mechanism you use in your organization and say like, "Airflow is not working," and in a lot of cases, it's some other system that is having some sort of issue.

I think it's typical to see a lot of the issues kind of bubble that way. I think like a classic thing would be running out of worker slots where maybe you have a certain number of workers and you're able to schedule – I don't know. It could be like at thousand jobs in parallel and at some point maybe there's abuse, maybe there's like just a lot going on the cluster. Sometimes it could be like Hive is overwhelmed or Spark is overwhelmed. So, jobs are starting to queue up and people might reach out to say, "Why is my job not running?" The answer is often out of the control of Airflow itself. But sometimes the solution is to change your infrastructure to be a little bit more elastic, or to put kind of check mechanism in place.

Airflow has got different mechanisms to limit parallelism, which is part of that immune system that we've built over time that we should talk a little bit more about. But sometimes the solution is to parameterize your immune system and limit the parallelization that any specific DAG or pool of resources is able to use.

[00:26:51] ABT: Plus, 9 times out of 10, it's DNS, because it's always a DNS problem.

[00:26:56] JM: And what are some other aspects of tuning I might need to do for my Airflow cluster? Would I need to – Situation where I'm going to need to manipulate the CPU or the memory that's allocated to the different areas of the DAG execution?

[00:27:13] ABT: Yeah. It very much depends on what your tasks are doing. If you are calling out to third-party services, your Hadoops, your Sparks, you need to have network requests, then that level of tuning isn't really probably get an issue. People can and do process data inside Airflow, but it's still not the most common ways used.

The most common things that you're going to run into will be kind of – Yeah, like ton of concurrency limits. Airflow has a number of different concurrency settings. You can have an overall number, like maximum number of tasks that you want your cluster to run. It's got a concept called pools where you can limit the amounts of tasks using third-party resource. Kind of attribute pools so you can say, "I only want, at most, 10 jobs running in Hadoop at once," or you could use that for a third-party API or don't have a load and make sure only one job. So you create a pool of 1. You can insure that each DAG doesn't run too many tasks at once. I think the default is something like 16 or maybe 32. Each DAG will only run 16 tasks at once. If you have a lot of – A big DAG, it won't overwhelm all the other ones.

Kind of like the default out of the box, try to get a sensible behavior for common use cases, but as you start to push it either vertically or horizontally, lots of DAGs or one very big DAG, you're going to need to look at those limits and the concurrency settings and kind of workout what makes sense for your workflow.

[00:28:46] MB: There are so many configuration points with Airflow, which can be a challenge for an Airflow administrator or someone setting up a cluster. The defaults are sensible, but over time, I think we grew to have a lot of these configuration parameters that can be tweaked by the administrator and making sense of everything that's in your Airflow, that CFG file, and any kind of finding the right value for your organization at any point in time can be challenging.

I think in the context of running on something like Kubernetes too, it kind of changes the game around resource containment or task containment. It's this idea that any given task should be limited in how much resource it is given. For the health of the system, you need to make sure that a data scientist cannot go and create some sort of like R job that will allocate as many CPUs as there are on the box, which we've seen before. You'll see R operate in some mode that will look like, "Hey, how many CPUs are on this box? All right. I'm going to do like CPU number -1 and just parallelize my stuff."

We've seen that early on at Airbnb and we're like, "Who the heck is doing this?" and introduced C groups really early on. That's probably 2015, I think, when we introduce some sort of support for C groups, which are a feature of the Linux kernel that allows you to define like the CPU and memory resources that are allocated to any given process in a system.

We did that, but I think like since we've been working on Kubernetes, we get all the nice properties that come out of Kubernetes too. So we can define clearly like what is the image that we want to use and what kind of resource you want to allocate to any given tasks? Then made it so much easier to operate Airflow so you don't have to go on the lower level configuration to do that.

There're other challenges that we've seen around things like impersonation. You might want to impersonate a certain user in different contexts. That's something that – Some tricks that Airflow has learned to play kind of over the years.

[00:31:00] JM: Cool. Ash and Vic, Vikram, you guys work at Astronomer, and this is a company that is built around Airflow, and you're working with people who have enterprise use cases for Airflow. What are the common enterprise use cases that you've seen and how do they compare to the kind of startup origin stories, the Airbnbs and Netflix kind of use cases that Airflow was originally used for?

[00:31:35] VK: I'd say that one of the things we've seen is people really wanting to expand the use of Airflow, because typically it starts with a data engineer having downloaded Airflow, using it for their data pipelines and then expanding it to a team and then saying, "Hey, there's another team pretty much very similar to how Max said it evolved at Airbnb, wanting to use it maybe for ETL, maybe for ML pipelines," and more and more teams started to adapt it.

One of the things we've actually discovered during this political process is that different teams to some cases have different needs and we want to make the experience for them not only from development standpoint, which we think Airflow does really well, but also from the deployment and the observability and the resource management standpoint, which are some of the key enterprise needs in being able to deploy Airflow at scale.

By scale, meaning that their number of deployments could be starting at around 20 deployments in some cases, going up to 100 deployments or more within an enterprise for different teams. The ML team might actually have a particular deployment, which is maybe based on scikit and they have an image of Airflow based on that. They are running a lot of ML jobs, which have got a certain set of resource constraints and resource needs. A different team might be based on ETL jobs and they've got a very predictable workflow. It's every hour or daily jobs as well, and some combination of a reckoning element which does not or cannot be impacted by the experimentation workflows, which are also run on airflow, but be good to be able to segregate and manage them independently.

Some of the key things which we see is, and I'm sure Ash is going to chime in as well, is around reliability, is around observability, ease of deployment, and really that separation of concern, which brought up earlier about the needs of a large number of data scientists and data engineers just needing to deploy to actually alter and then deploy their particular data pipelines. But then be able to manage them at scale without them trampling over each other because of different teams' needs at different times.

[00:33:54] ABT: Yeah, absolutely, and kind of like use cases. It varies as there are as many data scientists in the world. There's the traditional kind of batch jobs, absolutely. There're more machine learning things. There are people running AB or split testing of machine learning models. Some delivery companies I know are trying to – They are. They're training and running their machine learning models to give you an estimate of when your food delivery is going to arrive, because I think it's very predictable based on loads. They've got machine learning models which they're running to Airlow. They're based on the volume and stuff. They [inaudible 00:34:34] this every five minutes of peak and kind of farm it out and make it available so that when you place an order through the app, it tells you your food is going to arrive X, and they're running that through Airflow and they kind of got fitness scores coming out and promoting models and they're doing that all through Airflow. Yeah, it's anything you want with data, which is not a very good answer, but it's kind of – If you're doing something with data and you want to do it repeatedly and in a controllable and observable fashion, that's what people are doing in Airflow.

[00:35:08] JM: With Astronomer, the company that you're building around Airflow, how does the deployment experience compare to if I was just a self-deploy on AWS? If I was just a configure everything myself? What's the difference?

[00:35:28] ABT: Sure. You run AWS yourself, Airflow on AWS yourself. You need to ensure that you get all your DAGs and all your Python dependencies in-sync on all the nodes in approximately the same time. There's a little bit of leeway there. But every node in an Airflow cluster, which could be as small as a single node or it could as large as 50 nodes per cluster. There's no [inaudible 00:35:57] limit, but that's the biggest I've heard of recently. Every node needs the same DAG files and anything executing those DAGs needs the Python dependencies in store.

Airflow, as an open source project, doesn't say how you should do that. It's up to you to build your own deployment mechanism to get those DAGs available. That's one of the core things we've done at Astronomer, is we said, "This is how we think you should deploy. We think you should bake your DAGs and your dependencies into a Docker image and push that out."

We built a whole set of tooling. So deploying to your Airflow deployment on an Astronomer cluster is as simple as a Docker push. Then we handle everything else behind the scenes. Making sure all rolls out, gets to upgrade all the nodes. Make sure they're all run in the same version and then report metrics about that.

[00:36:59] JM: Okay. We'll talk about the modern Airflow ecosystem. The use cases of proliferated, and I imagine there must be some I guess growing pains that happen as the project has gotten mature. What are the maybe weak spots or areas of improvement or technical debt that have emerged over time across the Airflow project?

[00:37:28] ABT: Yeah. I kind of like – Generally, one of the hardest parts of running an open source project is saying no to things. One of the advantages of an open source product is you get thousands of contributors just sending you a pull request going, "Hey, it'll be great if you did this." And sometimes the answer has to be, "No. That's not Airflow's job." That's kind of a general comment.

Some of the technical debt left in Airflow, probably the big one will be availability of the scheduler. Right now, the scheduler is a single point. You should not run more than one currently. With Kubernetes, that basically equates to a deployment with max one. If it dies, Kubernetes will be started. Your tasks aren't run in your scheduler. So the impact of your scheduler being unavailable for a node restart or a pod restart is past, don't scheduled for a minute or two. But that's not great. That's kind of one of the big things, biggest bits of technical debt in Airflow, which was it hasn't impacted its availability to do its job, but it would be nice to fix it. That's one of the big things we're working on kind of the next month we started Astronomer. The Airflow team is working on making the scheduler highly available so that you can run more than scheduler.

[00:39:00] MB: Yeah, I can speak about the tech debt too, because I think like sometimes, in a way that software evolves, and I started too and took some shortcuts early on. Part of the beauty of the open source community too is to get things right over time. When you have a massively successful open source project that's used at hundreds, if not thousands of places and run under like a good governance model like The Apache Software Foundation, it enables all sorts of people to jump in and contribute and fix the things that needs to be addressed, improved, patched. There's so much that comes out of the distributed governance model.

From security patches, keeping things optimized and running well, things like making the scheduler a distributed part of the system too that you can run in parallel. That's coming through now. I think, originally, that REST API was not super well-defined too when I started the project, and that's getting super well-addressed by the community with people that are honestly like more rigorous and better than me at these things and really making it right.

I think what's interesting too is the idea too of like thinking about like what makes an open source project successful. Sometimes part of it is like having a strong architecture you can build upon, but sometimes it's doing the right thing at the right time too. The way that software evolves too, it enables people to go and re-architecture the bits and pieces that need to be re-architectured.

[00:40:34] VK: One of the thing I'd like to actually add to this is something you brought up earlier, Jeff, which is about configuration of CPU and memory. One of the things which I think is the big focus right now is just auto scaling. We really want people not to think about it and say, "Here's some broad constraints, because for machine learning team, this might be the broad constraints."

For an ETL team who's running production jobs or for experimentation, for AB testing team, these are the constraints. But within that, just let things auto scale without needing any configuration upfront and let the system automatically scale the kindles constraints so that people don't have to think about it. Again, that's part of the separation of concerns, which is that for a DAG developer or like whether it's a data scientist or a data engineer or any of the particular role within the company, that they should be able to just focus on what they're actually trying to accomplish without necessarily worrying about the infrastructure concerns and hold they will actually be deployed as long as they're within reason.

The auto scaling and not having to worry about constraints within a reasonable element is one key element we're working on. The second key thing which goes with auto scaling is, again, those guardrails, which is when are you actually getting to a break point before it actually breaks. Coming in from like a little bit in IoT background, that’s kind of the Holy Grail, right? You want to actually be able to do prevent it and maintenance and be able to address issues before it actually breaks. That's where the observability and being able to predict and saying, “Here’s where the guardrails and this is what needs to be tweaked or changed,” is kind of a key element as part of making this continues to grow and adaption within a much larger set of engineers within enterprises and around the world.

[00:42:25] MB: Yeah. About this topic, I think it comes down to like do you want to be on-call for that system or not, or do you want to rely on the expert that are specialized in operating this system and tuning it, configuring it, keeping it secure, providing SLAs on it so you can just rely on the professionals to do it as supposed to struggling and figuring out like when you're going to upgrade the system or patch it. That that part of operating distributed systems at scale and enterprises is very easy to underestimate how much work that takes, and it's nice when you can find a service that's reliable and people – You can trust to do that better than yourself.

[SPONSOR MESSAGE]

[00:43:16] JM: When the New Yorker magazine asked Mark Zuckerberg how he gets his news. He said the one news source he definitively follows is Techmeme. For more than two years and nearly 700 episodes, the Techmeme Ride Home Podcast has been one of Silicon Valley's favorites. The Techmeme Ride Home Podcast is daily. It's only 15 to 20 minutes long and it's every day. By 5 PM Eastern, it has all the latest tech news, but it's more than just headlines. You could get a robot to read you the headlines.

The Techmeme Ride Home Podcast is all about the context around the latest news of the day. It's top stories, the top posts and tweets and conversations about those stories as well as behind-the-scenes analysis. Techmeme Ride Home is like TL DR as a service. The folks at Techmeme are online all day reading everything so they can catch you up. Search your podcast player today for Ride Home and subscribe to the Techmeme Ride Home Podcast.

[INTERVIEW CONTINUED]

[00:44:18] JM: How is fault tolerance model for Airflow today? Has it been improved or are there any issues that can – Any failure scenarios that can cause problematic situations, like data quality issues, or some kind of problem like that?

[00:44:40] MB: Part of the foundation for that is that we ask people to build idempotent tasks, right? If it in your Airflow DAG you have a task that will fix itself if you rerun it, then that makes it much more easier to operate. I think it's pretty much a standard in data processing in general that your task should be idempotent. So that means if a task fails half way through for whatever reason or right before the end or right at the beginning, the Airflow scheduler knows it has not succeeded and can reschedule it.

At that point, another worker, maybe that worker died, maybe a grenade exploded in the AWS data center and there were some casualties and your servers are not there anymore. If you have an Airflow worker up and the scheduler is up, the task will rerun and reprocess itself within what you define in terms of like retry policies.

In Airflow, you can define how many times a task should be retry. How long you should wait between those retries. Whether it's like a linear amount of time or giving it a little bit more time as there's more failure, kind of exponential backup type of scenario. As long as you have a worker up and a scheduler up, the core component of your cluster, things will work. Off course, if you don't have your metadata database or your Redis kind of message queue up, things will come to a halt, right?

We need for all the vital organs to be up and functioning. I think like, typically, the things that we rely on that are kind of single point of failure, like an RDS database or just like a metadata database, it's pretty common for systems like Airflow to rely on such a system and these systems have failover mechanism or they have their own ways to provide guarantees about being up.

[00:46:41] JM: Max, are there any mistakes you made, original sins, in Airflow's architecture that pervade to this day?

[00:46:49] MB: Well, I think so, but I think like, as I was saying before, is like these shortcuts, maybe you can call them mistakes. But I think I've taken some shortcuts early on, like one thing was like, say, for the CLI to not talk to a Rest API and just talk directly to the database, for instance. That's like a design flaw and I knew when I wrote that CLI that I should be writing a REST API first and then have the CLI talk to the REST API instead directly to a database.

But all these things are getting addressed today and that enabled us to build a relevant project earlier that delivered more value and grew a community, and these things can be changed as the project evolves.

One side question or like – I'm going to talk about. One thing I wanted to talk about too is this idea that in software, I think we tend to really value architecture as kind of a beauty, a grander, like architecture is really important. I think it is, but I think we sometimes forget how important it is to have like a battle-tested mature system that has been used in a lot of different settings and environment by different groups of people and set of people for that assistant to have evolved and reflected all these needs over time.

I think like this maturity sometimes is not necessarily ready from a design standpoint or from like some maybe bug fixes or patches. It might not look beautiful in the code, but like they are things that are part of battle testing. They're like scars from like proving the value and delivering on goals that matter to people too.

The exercise of like what's beautiful architecturally and kind of a perfect system versus like what is kind of battle-tested systems. I think sometimes we don't give enough credit to that second part too. You see these – I remember these systems internal to Facebook that systems that maybe there's been some papers on or some conversations and podcasts on, but systems like Scuba, Data Swarm, Hive, these assistant works so battle-tested. They're like tanks. You try to take them down and you couldn't just because they had so much load over time.

I think Airflow is kind of backtracking in some architectural decision, things like having a really centralized REST API that works really well. Things like having a distributed scheduler and all these things, have happened or are happening today. There's just a slightly different way to build software where maybe you start by delivering value and then you optimize to find the right like compromises and the right positioning as you move forward.

Airflow today has – I think on our GitBub page, there's been like 300+ organization that sent us a pull request to say like, “Hey, we use this thing in production,” which is really the tip of the iceberg. In reality, we do have thousands of companies using this in production that are committed. The nature of a federating system like Airflow is very sticky too. So we know at places like Lyft and Airbnb, where I was before, that they will be running Airflow in a decade, certainly, despite having built something like Flyt at Lyft and having some more specialized alternative for ML and other places.

I think this long-term vision is really important too or we an Apache Software Foundation project that is governed by many organizations, including Airbnb, kind of the mother company, but places like Astronomer and Google that are heavily invested in the project. And then hundreds of other organization, and there's something that's really nice about this. The Apache Software Foundation has got some – There’re differently like pros and cons to this model of governance, but like one thing that's really nice about it is that it's a meritocracy and it's very welcoming and people from all sorts of places can come together and govern and move that project forward.

[00:51:18] JM: All right. Well, we’ll begin to wind down. One question I wanted to ask. We did this show recently about a project called Cadence, which is a work – More like a workflow orchestration system rather than a data workflow orchestration system. Cadence is used for long-lived microservice workflows, for example, where you might have something that might take days to execute or it's almost like a long-lived user session that might exist across multiple services. It's an interesting system, and it just made me wonder why Airflow is only used for data workflows. Why isn't Airflow used for maybe complex multiservice kind of like state management things, like issues where you might have state managed across multiple services and you're doing something involving those multiple services and perhaps an interactive user. Why is that? Why is Airflow only used for data pipelines?

[00:52:25] ABT: Mostly because that's where it came from. That’s sort of all the docs are written about. You absolutely can do some of those, but it’s not kind of a use case that we’ve kind of put front and center in our docs. It’s not obvious at first site without getting familiar with the code how you’d have to wire it all up together and how you’d have to do the communication.

Yeah, ultimately, kind of Airflow – Airflow doesn’t care what it’s tasked of doing. Yes, it’s has all these built-in operators and kind of connections for dealing with common data processing tasks. But it can run a batch script. It can run Python code. Because it can run batch, it can run absolutely anything you want. It’s kind of Airflow doesn’t actually care what it’s running. You absolutely can do that. It’s not where Airflow came from, so it’s not obvious or [inaudible 00:53:18] how you could use it for this kind of workflow.

Cadence is a really interesting product. I checked it out after the episode, and the kind of the big difference between an Airflow workflow and a Cadence workflow is, in Airflow, you don’t have to change your code to work with Airflow. You can just run your existing code you want. Whereas Cadence, you have to make sure there’s all kinds of caveats, which it’s got pros and cons. Because of that, they’ve got all kinds of flexibility about retries and stuff like that, but you have to embed cadence really deeply in your application and in your workflow. If it’s in Go or Java, which is the two class they’ve got or had, you have to improve on the interfaces and you have to use their APIs, and it’s very tightly coupled into your application. Where something like Airflow, it kinds of stands back and just go, “I’ll run what you tell me,” and you can do what you’d like once Airflow kicks it off.

[00:54:14] MB: I think the term workflow can be deceptive too. We think that workflow orchestration is pretty generic. It sounds like we’re in the same space, but I think like fundamentally we’re in fairly different places between Airflow and cadence. Cadence, I’m reading from their docs page, but it talks about being kind of a platform to build distributed applications and it's definitely not the case with airflow.

Airflow, you can think of as a distributed scheduler that is really smart and where you can put intricate rules around complex dependencies and what happens with failures and you can have enormous DAGS with hundreds of task that depend on each other and they're all batch jobs, essentially. It’s just a really smart distributed CRON where cadence seems more like an infrastructure of like a framework kind of layer to build distributed applications on top off and maybe communication layers across system. So maybe a workflow for them is like atomic transactions, or transaction across systems maybe. But I don't know enough about the project.

[00:55:23] JM: No, that’s right. All right, guys. Well, one last question. Any changes you anticipate in the distributed workflow space in the next few years?

[00:55:34] VK: I would actually just going to piggyback on Max’s comment just a minute ago. By that, you do mean the distributed workflow for data pipeline?

[00:55:42] JM: Yes. Yes.

[00:55:46] VK: One key thing which I think we are seeing a lot of is an explosion in data pipelines from the perspective of a lot constituents wanting to be able to easily, and by that, I really do mean very easily author and deploy pipelines for a very large variety of needs without necessarily needing to know anything about [inaudible 00:56:08] where it’s run with a conflict, being that for different kind of pipelines or specialized processing now for machine learning and how do you take advantage of that. But the next step I think is really around the entire lifecycle, which is how do people integrate this into their CICD pipelines. How do people make this observable at the very end so that you can say, “Hey, look. I automatically need to scale. I need to deploy.” The entire set of processes which happened as part of any mature, even like a DevOps lifecycle or like how software development evolved in the lifecycle of all the DevOps. That same lifecycle with respect to data pipelines evolving with respect to of data operations and that smooth transition from somebody saying, “I have an idea, or I have this business need. How do I author something? How does that whole process happen seamlessly without having any hiccups and their interrupts?” Longer term, being able to be notified of this within these particular gaps.

Now, within hat, we’re covering things like the equivalent of GitHub from like a source code aspect, covering any equivalent of a deploy and like zero downtime deploy and like different micro-deploys over a period of time. Within that, you're starting to think about observability and versioning and, “Hey, this predictive change caused this.”

Starting to think about how do you store your infrastructure, whether using [inaudible 00:57:35], or resource constraints of whether they're automatically getting deployed. Within that, you're starting to think about security scanning for vulnerabilities and data security and data privacy as being an increasing development. I think that entire lifecycle, which we've seen in software applications and that lifecycle as applied to data pipelines is one key thing which I'd see like in a very, very quick change in that entire maturation of the data engineering practice.

[00:58:07] ABT: Yeah, the biggest change I think that's coming in to data, engineering data science, data processing space is because of all the data [inaudible 00:58:14]. Kind of GDPR and the right to be forgotten. If you have taken some customers' data and then you've processed it once and then like four levels down, it's kind of [inaudible 00:58:24] multiple different levels of using that data and consuming and transforming it. If someone says please go delete my data, how do you know which datasets that data fed into? Kind of [inaudible 00:58:37] and traceability around where your data has gone and where it's come from. That's going to be a big thing. GDPR in Europe and California has similar laws, parsed or parsing. I think that's going to be a big thing that's coming, which is what are you doing with your data and being able to show what you've done with your data. That's particularly important for regulated industry. So kind of like finance and healthcare. They also need that. I think that's going to be a big one, which is more people are going to have to pay attention to what they're doing with their data and being able to prove what they're not doing with their data.

[00:59:17] MB: One thing I'm really curious to see evolve to is what I call kind of layer 2 solutions to Airflow, right? There's an interesting analogy if you look at Bitcoin and lightning network where you have Bitcoin as really foundational and that things can be built on top of that blockchain, like things that might be faster, more opinionated that provide a different set of constraint and guarantees.

I think like we're definitely going to start seeing more open source project that are like these computation framework that are more opinionated that solve more specific business, specific use cases. Things like you know AB testing framework should data the data quality related things. Everything related to the data ops or the ML infrastructure movement. Feature repositories, metric repositories. Those are things that every company, every big companies with data professional, data team that have these teams are building today, but that are filled with business logic and have not converged. I think there are still things that different companies, different individual thinks differently about what a feature repository that can serve like an ML and for a platform looks, or does what is the metric repository look like for AB testing. I think we're already starting to see certain level of people kind of sharing their use cases or maybe doing reference implementations and sharing that code. But when I created Airflow, originally, I really wanted for these like layer two solutions and framework to start emerging at pieces of open source software that would be potentially powered by Airflow. I'm really curious to see that come through.

I think we'll see more tools too around like metadata management, or right now all these metadata that we have, and metadata is kind of an unclear term and there operational metadata. There's lineage metadata. There's kind of this graph of data object in your company that is very complex. Seeing tools like Airflow [inaudible 01:01:25] work a little bit better together and like kind of the data ecosystem to evolve to share more metadata. I think the theme of the next 5 to 10 years is probably like metadata management and metadata sharing across the ecosystem of tools.

[01:01:46] VK: One thing I would add, add to what Max just said, is from a commercial standpoint, we're definitely seeing commercial software solution providers wanting to embed airflow into their offering. It's kind of like the layer 2, not necessarily source solutions, but layer 2 commercial solutions and building Airflow and building on top of it for particular business vertical needs. That's what's trying to happen in the commercial side. I'll absolutely expect that I agree happening on the open source side as well.

[01:02:20] JM: Cool. Well, guys, thank you so much for this long and detailed conversation about Airflow. It's an amazing project. And Max, nice work on creating it, and Vic and Ash, I am impressed by what you guys have done with Astronomer.

[01:02:33] MB: Thank you so much, Jeff.

[01:02:33] VK: Thank you very much.

[01:02:34] MB: Thanks for having us.

[01:02:36] JM: All right, guys. Have a great rest of your day.

[END OF INTERVIEW]

[01:02:46] JM: Today's episode is sponsored by Datadog, a cloud scale monitoring service that provides comprehensive visibility into cloud, hybrid and multi-cloud environments with over 250 integrations. Datadog unifies your metrics, your logs and your distributed request traces in one platform so that you can investigate and troubleshoot issues across every layer of your stack. Use Datadog's rich customizable dashboards and algorithmic alert to ensure redundancy across multi-cloud deployments and monitor cloud migrations in real-time. Start a free trial today and Datadog will send you a T-shirt. You can visit softwareengineeringdaily.com/datadog for more details. That's softwareengineeringdaily.com/datadog and you will get a free T-shirt for trying out Datadog.

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

[END]