EPISODE 1100

[INTRODUCTION]

[00:00:00] JM: Netflix runs all of its infrastructure on Amazon Web Services. This includes business logic, data infrastructure and machine learning. By tightly coupling itself to AWS, Netflix has been able to move faster and have strong defaults about engineering decisions. And today, AWS has such an expansive services that it can be used to build a platform for any kind of tooling that you would want out of your company.

Metaflow is an open source machine learning platform built on top of AWS, which allows engineers at Netflix to build directed acyclic graphs for training models. These DAGs, directed acyclic graphs, get deployed to AWS as step functions, a serverless orchestration platform in AWS.

Savin Goyal is a machine learning engineer with Netflix and he joins the show to talk about the machine learning challenges within Netflix and his experience working on Metaflow. We also talk about DAG systems such as AWS step functions and Airflow.

We are looking for writers for Software Engineering Daily. If you're interested in writing for us and potentially earning some money, send me an email. I'm also looking for companies to invest in. If you are working on a company and it's for developers or it's something related to infrastructure, send me an email, jeff@softwareengineeringdaily.com. I'd love to hear from you.

[SPONSOR MESSAGE]

[00:01:28] JM: Errors, and bugs, and crashes happen all the time across my software. Most often, these crashes have to do with obscure exceptions that come from React components, failing to render on the client device. Source maps and stack traces would be useful, but in many cases, I'm not able to identify the root cause because the error is occurring on a client device. It's not on my infrastructure. And what can I do about that? I can use Sentry. Sentry.io can quickly triage and resolve issues in real-time.

Visit sentry.io/signup and use code SEDAILY to sign up for two free months. You can simply choose the platform that you're integrating with. Sentry works for every major language and framework, from Rails, to C#, to Java, or React Native, and you just install the SDKs. You integrate it with your application. After that, errors will be caught by Sentry and you'll be alerted the moment that they occur so that you can triage and resolve and keep your users happy with functional applications. If you're building an application and you want to monitor your errors and your performance, try Sentry by visiting sentry.io/signup. And new users can use code SEDAILY for two free months.

Thank you for being a sponsor of Software Engineering Daily, Sentry, and you can go to sentry.io/signup and use code SEDAILY if you're curious about it.

[INTERVIEW]

[00:03:01] JM: Savin, welcome to the show.

[00:03:02] SG: Thank you for having me.

[00:03:04] JM: Netflix has lots of applications where machine learning is relevant, and I'd like to start by just talking about those at a high level. Can you describe some of the use cases for machine learning within Netflix?

[00:03:17] SG: Sure. Netflix recommendations is sort of are cron jobs. So anytime you log in to our service, all the shows that are recommended to you are powered by a constellation of machine learning algorithms. But besides that, Netflix also happens to be one of the biggest tool use. We spend a lot of time in energy, evaluating how should we program our content as one of the biggest subscription services. We also need to constantly be sure how are we tracking user churn and how are we processing payments, fighting fraud. There are significant applications around computer vision as well. We want to make sure that we are able to efficiently transfer all the media builds from our locations in AWS to our users' homes, as well as make sure that people are able to do automated content analysis, QA on our content. So there are wide variety of use cases, and as a result of that, we have a wide variety of talent internally at Netflix. We're focused on these problems.

[00:04:18] JM: Tell me about machine learning development flows. So there's this term of flow or a workflow. What are the steps within a machine learning development flow?

[00:04:29] SG: Sure. The workflow that's followed inside of Netflix, and I guess this would be true for most people in the community as well, is our data scientists, they are usually focused on optimizing certain business metric, solving a particular business problem. And usually their starting point is Jupyter Notebook if they are in the Python ecosystem, or maybe in our studio, IDE. Some sort of an IDE at the end of the day. They'll have some mechanism of accessing the data. In most cases, they'll usually run a Presto query or a Spark query to get access to their data. Then they'll use some off-the-shelf library, be it TensorFlow, PyTorch, XGBoost to certainly get their work done and trade on some of the earlier versions of their model and sort of like go through that.

Once they are sort of like happy with this approach, once they see some promising results, then they would ideally want to scale up their compute. In many instances, they might start off with a small set of data and they might now want to sort of like train their models on a much bigger set of data, or maybe train multiple different sets of models to figure out like which is the best performing model.

Then once they have built a model, the next concern is around productionization. And at Netflix internally, productionization, it's ultimately a spectrum. It means different things and different contexts. Many times, it will power an offline use case where the results of model scoring need to be written to a table, which let's say powering Tableau or a micro-strategy dashboard, or maybe is powering an internal UI, maybe somebody wants to sort of like take a look at how some of our various titles are performing on the servers, and we can have some of the metrics being driven by these models. A variety of different production use cases at the end of the day.

Now, one thing that you would notice in this entire cycle is the amount of time that our data scientist spend on an actual machine learning, and this was pretty much true before we started working on Metaflow. It's pretty low. A lot of their concerns are concerns in the engineering domain. So how do I scale up my model training? How do I scale out and train multiple models in parallel? How do I productionize this model? Whatever that definition might be. How do I get access to the data efficiently?

Metaflow ultimately at the end of the day, our focus with that product was to ensure that our data finders are able to focus more on data science concerns and we are able to abstract out some of the engineering bottlenecks that they were facing on their day to day work.

[00:07:11] JM: The definitions of roles are blurry in the field of data science. There're data scientists, machine learning engineers, there are data ops or DevOps, or ML ops teams. And it's not necessarily well-defined. Can you give a holistic view of the kinds of roles that exist in these teams and how they collaborate on building and maintaining machine learning models?

[00:07:37] SG: Sure. Netflix is rather unique in the sense that it has a culture of freedom and responsibility, and [inaudible 00:07:46] to that is that our data scientists are, in other words, people who practice machine learning or data science. They have the utmost freedom to use whatever tools sort of like get their job done. But then they sort of like also have the responsibility to see their project end-to-end. That means starting from a prototype phase and sort of like owning the entire process till the very end to productionization step, which means that there is no strict handover that happens where let's say a data scientist might be responsible for prototyping an idea and then somebody gets involved and sort of like taking it till the very end and then maybe interfacing in some amount over the software engineering team.

So we don't have those strictly defined roles at the end of the day. Internally at Netflix, the one single job title that we have is senior data finders. These are people who have really strong experience, really strong academic background in machine learning and data science and it's their responsibility to sort of like, at the end of the day, deliver solution to the business problem.

[00:08:49] JM: What are the aspects that are tricky about getting a machine learning project from the data science, "I'm just a Jupyter Notebook tinkerer," to the production process? This is something we've covered relentlessly on the show, of course. But just tell me about some of the problems and solutions to getting ML models into production.

[00:09:12] SG: Sure. I guess like one of the big problem that people have is this distinction between what is development and what is production. Let's take a dummy example here. Let's say there's a data scientist and they're working on machine learning problem say around they want to predict how many viewers a particular piece of content will get once it's live on our service. Then they'll have access to some raw data that they are going to use to train their model within our data warehouse. The number one concern that they'll have is how do I get quick and efficient access to this data? And once they have access to this data, they want to make sure that they are able to reliably to run feature engineering on top of this dataset.

Now, this dataset could be humungous and there are tools, like Presto, Spark, that people can use to sort of like then do some sort of distributed feature engineering or feature processing. But then at times, it's really helpful if you sort of like get this one big data frame in your Python process that then you can interactively slice and dice. Many times the compute resources that are directly available to the data scientist might not really be up to the job at that point in time. It might just be really efficient if say they had access to a bigger laptop or a bigger cloud instance. That's like one big piece of model like that. We observe that a lot of users were facing internally where they had to very rapidly switch between different instances to sort of like get their job done.

Now once you have access to this data, then the next step is that you need to train a model. Model training in general, again, might have different requirements in terms of resources, maybe. You need to train your model using a bunch of GPUs, and the instance that you're currently on doesn't have those GPUs available to you. Then how do you actually package your code and move your compute to a different instance as well as move your data to a different stance, train your model and then get that model back? That's, again, one of the big bottle necks that people have to sort of like wrestle with.

Now, let's say you sort of like figured out a solution to this problem as well. Now at the end of the day, machine learning is a fairly iterative process. We try out like a bunch of ideas. Some of the ideas pan out. Some of those don't. You need to have some running log to track these experiments. At times, it's also sort of like left at the discretion of the end user around how do they want to do this experimentation tracking? And that can become cumbersome really, really quickly. I guess a solution that's sort of like very comprehensively deals with like all of these concerns. We have a really long way in removing some of the engineering bottlenecks that the data scientists raise at the end of the day.

Now yet another aspect that is not really often talked about for machine learning use cases is around this notion of reproducibility. Now, yes, like people have been focusing on reproducibility from the stand point of code and data, but when you look at the kind of machine learning that's practiced in one of these companies, there's a big element of infrastructure involved as well. Your infrastructure is also rapidly evolving and it's not just the code and the data that's evolving. So you need to take a comprehensive view where you're able to snapshot the actual environment, the user code, all the dependencies as well as the data to sort of like guarantee a stronger notion of reproducibility. Wherein somebody else, let's say, a week, a month, a year down the line, can very reliably take your code, regenerate people, use the exact same set of results and then iteratively build on top of that.

So these are like – We spoke about like a bunch of different issues. So were the issues that we saw data scientists internally at Netflix were facing, that was sort of like one of the big motivations for the team to sort of like start building this end-to-end framework that sort of like abstracts away these concerns of all the operational best practices that we had learned over the years.

[00:13:26] JM: This led to the creation of Metaflow, which is a platform built within Netflix to manage machine learning. What is Metaflow? What are the problems that it addresses?

[00:13:37] SG: Sure. Netflix has been doing recommendations for a really long time. I guess it's like over 15 years that we're sort of like we've done our recommendation algorithms. And we have a very bespoke set of infrastructure that's sort of like handles those concerns.

Now roughly four, five years ago as we started getting into more and more additional content as we sort of went global, we started coming across a whole bunch of different problems related to like how we value content, how we process payments, how we do customer service, where people were eager to apply machine learning to sort of like gain that last [inaudible 00:14:20]. And all of these problems, they're very different from one another. They require really diverse set of skillsets, very diverse set of tooling. And it wasn't tenable for us to sort of like have a bespoke set of infrastructure for each of these problems.

Now, yes, at times some of these problems can become super important that they deserve some sort of like bespoke infrastructure, but then that's sort of like the 20% of the problem set, 80% of the problems still seem to share common enough concerns that you can sort of like build this end-to-end platform that can sort of like quickly get people productive when they're like working on these problems.

So that's when roughly 3 years ago we decided that, "Okay, the infrastructure that we had built for the recommendations side of business, that couldn't really apply one-on-one to many of these problems, and it might make sense to spin up a product, spin up a team that can tackle this wide diversity of problems in a very cohesive manner. The result of that was Metaflow.

And then last year in December at AWS re:Invent, we actually open sourced Metaflow as well. So now it's an open source project and it's available to audience outside of Netflix as well.

[00:15:39] JM: What kinds of projects is Metaflow a fit for? What's an example of a machine learning project? What would the spec be for a project that would make sense for Metaflow?

[00:15:51] SG: Metaflow by itself, it doesn't really take any opinions on the kind of machine learning that people are trying to do. So I'm part of the machine learning infrastructure team, and even though machine learning is in the name of my team, we very rarely take opinions around the kind of machine learning that our users should be doing. What we take strong opinion on is how the infrastructure should be arranged. For example, some of the concerns that I spoke about before around moving your compute from your laptop to a cloud instance. Storing data in a cloud data store. How to do that and how to sort of like abstract that away? Those are the problems that we sort of like focus on. These are concerns that all machine learning problems take care like have at the end of the day as a commonality.

Now, to answer specifically the kind of problems that Metaflow is a good fit for. So right now using Metaflow, you can train your models in a batch mode. If at the end of the, you are training, you have some set of data that is lying in a data store or in some data vault. Using Metaflow, you can very easily get access to that using whatever libraries you're using currently outside of Metaflow, and you can bundle in your training logic. You can scale it up, scale it out on a fleet of cloud instances. Then at the end of the day, Metaflow will log and store all of the artifacts and the intermediate states of your model training runs, including the model that you train.

It's sort of like then provides you a very convenient ability to inspect and debug that model as well as inject that model into any other process. So, say, you want to run batch inference or real-time inference on your model. Then whatever that service is, you can very easily just inject your model into that process using Metaflow.

[00:17:39] JM: What was the impetus for Metaflow getting started? I mean, there's a lot of machine learning tooling out there. So somebody must have had to say, "Okay, we've got to actually build Metaflow. We got to build something to solve some distinct set of problems where solutions did not exist." What were those solutions or – Sorry. What were those problems and what was the distinctive solution that Metaflow is built to solve?

[00:18:04] SG: Sure. We started building Metaflow middle of 2017, and the ML ecosystem, surely, was very much mature at that point in time. But the ML infrastructure ecosystem was still very much in its infancy. So some of the other projects that you see in the ecosystem, like Kubflow and MLflow, they were either just starting out or they just simply didn't exist.

So it didn't really have very many options honestly and don't look just like looking at solutions that we could buy and sort of building something by ourselves. And at the same time, the other sort of like problem – Not really a problem, but inside that we had was – So Netflix has been using AWS for well over a decade now. So we had built a lot of operational expertise in terms of how do I interact with AWS? How to scale out our compute on AWS? How to deal with, say, S3? And we wanted to sort of like bake-in those operational best practices in a library so that our end users won't have to worry about any of those by themselves. So this was also a great opportunity for us to like build something that relies on all the knowledge that we have gathered over the last 10 years or so.

[00:19:22] JM: And this brings us to AWS. Netflix took the, at the time, unconventional decision to go all-in on AWS many years ago at this point. And that has treated Netflix really well, because it’s almost like the whole idea around blessed programming languages, where you make a strong decision within an organization to restrict the number of programming languages within an organization, and constraint ends up helping the organization make decisions more quickly and allow for engineering mobility and so on.

This was the case – This has been the case with AWS when Netflix strongly moved on to AWS and it’s just continue to do that. That extends to Metaflow. So Metaflow is an open source framework, but it has a tight coupling with AWS. So why is the tight coupling to AWS useful for a machine learning framework?

[00:20:18] SG: I wouldn’t say that we are tightly coupled to AWS. So when we were open sourcing Metaflow at that point in time, because we had a good amount of operational expertise with AWS. We chose integrating with AWS as our very first cloud integration. But the architecture of Metaflow is very much vendor-agnostic. So we already have people who have ported Metaflow to work on top of Google Cloud, for example.

Yeah, we want to make sure that at the end of the day, the end user, our data scientists, they don’t have to worry about any of the concerns that are introduced by, let’s say, using AWS, or GCP, or Azure, or the other cloud vendors. To them, they are just writing code in an idiomatic language, whether it’s Python or R. And then Metaflow takes care of actually understanding their code and orchestrating that on top of, say, AWS, or GCP, or any of the other cloud providers. Yeah, that’s the strategy that we have been following.

We started with AWS because that’s what we use internally at Netflix. That’s what we have most amount of experience and expertise. But there are people who have made it work with GCP. And going forward as well, it’s something that’s on our roadmap to have more and more vendor integrations.

[00:21:39] JM: A point of comparison to Metaflow might be Airflow, the distributed workflow scheduling system that’s often used for data engineering jobs. How does Metaflow compare to Airflow?

[00:21:55] SG: So Airflow is what you would term as a production create scheduler in the sense that you have your ETL pipeline and now you want your ETL pipeline to run autonomously, say, at the stroke of midnight every day or when some data is available that pipeline should be triggered. Once you sort of like created your ETL pipeline and you're happy with like how it's running, at that point in time, it makes sense to sort of like port it to an Airflow or, say, AWS Step Function or Luigi, for example. But what is severely lacking in some of these tooling is the local experience. On my laptop, I want to make sure that my workflow runs perfectly well and then I should be able to deploy it, and that's sort of like what we can refer to as productionizing this workflow on top of, say, an Airflow or an AWS Step Functions.

Now, Metaflow has this notion of a DAG and it bundles in a local scheduler. So when you're writing your code, build Metaflow on your laptop at that point in time, Metaflow's local scheduler will be responsible for executing your code, executing the nodes of the DAG, and you can mix and match. You can have certain nodes of the DAG to run on your laptop, certain nodes run on the cloud with specific resources that you've already specified.

Then once you are happy with the end-to-end execution of your workflow, then at that point in time, Metaflow allows you to compile your DAG into a specification that one of these production period schedulers understand. As a matter of fact, in early July, we are going to release our integration with AWS Step Functions, which is a production scheduler available in AWS, wherein you can take your Metaflow workflow, and with just one command line argument, you can deploy that on top of AWS Step Functions.

Now, step functions like some of the nice things about it is it's a highly available DAG scheduler, and it allows for running workflows which can span, say, a year as well. And the operational footprint is really, really small. So you get like really, really nice qualities when, say, you're running on one of these [inaudible 00:24:18] schedulers.

Internally, at Netflix, we chose to build one ourselves. So it's called Meson. And there are a bunch of talks online around the specific infrastructure as well as the architecture details on that one. And our internal users, what they do is they'll essentially write their Metaflow workflow in this DAG, and then with just one single command line argument, they're able to export that as a Meson workflow. And this Meson executor/scheduler, has a good amount of feature set around, say, how to do alerting. How to trigger when certain other workflows finish? How to resume your workflows? You can set a variety of triggers based on time of the day and so on and so forth.

So you get like all of those nice feature set when you are on top of one of these production create schedulers. And then you also get all the lineage tracking and the data code and environment snapshotting using Metaflow. To   our end users now, they don't really have to concern themselves with understanding the programing model that Meson provides or understanding the programming model that AWS Step Functions provide. They just write their code in Metaflow and then Metaflow takes care of interfacing with any of these production schedulers.

Now coming back to the question that you had around comparing Metaflow with Airflow, I would say that these are very two orthogonal products and they are not competing against one another. On the contrary, they are supposed to work well with one another. The happy situation would be that you use Metaflow for your local prototyping. And once you're happy with the results, then at that point in time you can very simply just export your workflow on to Airflow.

[SPONSOR MESSAGE]

[00:26:10] JM: You probably do not enjoy searching for a job. Engineers don't like sacrificing their time to do phone screens, and we don't like doing whiteboard problems and working on tedious take home projects. Everyone knows the software hiring process is not perfect. But what's the alternative? Triplebyte is the alternative.

Triplebyte is a platform for finding a great software job faster. Triplebyte works with 400+ tech companies, including Dropbox, Adobe, Coursera and Cruise Automation. Triplebyte improves the hiring process by saving you time and fast-tracking you to final interviews. At triplebyte.com/sedaily, you can start your process by taking a quiz, and after the quiz you get interviewed by Triplebyte if you pass that quiz. If you pass that interview, you make it straight to multiple onsite interviews. If you take a job, you get an additional $1,000 signing bonus from Triplebyte because you use the link triplebyte.com/sedaily.

That $1,000 is nice, but you might be making much more since those multiple onsite interviews would put you in a great position to potentially get multiple offers, and then you could figure out what your salary actually should be. Triplebyte does not look at candidate's backgrounds, like resumes and where they've worked and where they went to school. Triplebyte only cares about whether someone can code. So I'm a huge fan of that aspect of their model. This means that they work with lots of people from nontraditional and unusual backgrounds.

To get started, just go to triplebyte.com/sedaily and take a quiz to get started. There's very little risk and you might find yourself in a great position getting multiple onsite interviews from just one quiz and a Triplebyte interview. Go to triplebyte.com/sedaily to try it out.

Thank you to Triplebyte.

[INTERVIEW CONTINUED]

[00:28:28] JM: It is worth revisiting this AWS question, because it seems like if you just used the AWS Step Functions without having an entire layer on top of it, it might be just as useful. Why do you need this translation layer of Metaflow where you specify things in Metaflow and then it gets translated into a step function workflow?

[00:28:54] SG: Good question. So not if you look at AWS Step Functions, it's a general purpose DAG orchestrator of sorts, right? Machine learning workflows ultimately at the end of the day, they have very significantly different requirements as compared to, say, an ETL workload, right? In machine learning workload, you want to make sure that you have some notion of being able to access different set of resources. So an example could be that, say, I have three steps in my workflow. The first step fetches some data, does some feature engineering. The second step trains a model. The third step verifies that the model has actually seen and like deploys that model [inaudible 00:29:35].

Now, in the very first step where you're doing feature engineering, maybe you just need an instance that has a good amount of memory, let's say, 400 gigs of RAM to be able to allow you to sort of like filter your dataset and come up with the actual set of features that you need to train your model upon. The model training step might only need access to a GPU instance that sort of like allows you to train that model and you just need that instance for that specific duration.

Then the last step where you are just, let's say, running some lightweight tests on your model, maybe you just need a very small tiny instance. Now, mixing and matching these instances is something that can get super tricky if you sort of like start building something on top of Airflow or AWS Step Functions. It will necessarily like need a layer of [inaudible 00:30:26] on top of any of these schedulers.

Now, also machine learning workflows, they are very iterative in nature. So you need to make sure that like every single execution that you're doing, whether you're doing it locally or whether you're doing it on top of AWS Step Functions. They are versioned and you have access to all the internal state. And the AWS Step Functions runtime by default would not provide you any of these niceties. And another factor at play is that not all machine learning workflows might actually end up on AWS Step Functions. So an example here could be that, say, a data scientist is working on a model and they are just happy training that model on their local workstation. Or maybe when they actually need access to a GPU, they are still kicking off their workflow execution from their laptop and not really having the need to actually deploy it on top of any Airflow or AWS Step Functions.

Metaflow has been open sources for like close to six months now, and it's only now that we're actually releasing our workflow orchestration integration with AWS Step Functions. For the last six months, we actually had this big gap in our offering. Honestly, looking at how people in the open source community are using Metaflow so far, I think that majority of the use cases are still in the world where people just want a workflow manager that they can launch from their own laptop.

[00:31:58] JM: There have been a number of these types of open source platforms for doing machine learning development. The closest one that comes to mind is Flyte, from Lyft. Have you looked much at Flyte and do you have a perspective on how the problem solved with Flyte compare to what you've solved with Metaflow?

[00:32:18] SG: So, now the perspective that we have with Metaflow is, for our data scientist, like the concerns that I've spoken about previously, all of those concerns are not impossible concerns. People were getting their job done before Metaflow existed. So what we wanted to enable was making sure that the life of our data scientist was easier than before.

For that, we chose to not really sort of like reinvent the world. For example, when it comes to our data store, we rely on S3 or – Our eventual GCP integration will rely on GCS for that. When it came to compute, we rely on Amazon ECS and we haves similar integration that work for Kubernetes as well. Then for our orchestrator layer, even though we bundle with a local scheduler, the expectation is that when you actually move your workflow into production. We're using a production create scheduler. We didn't really want to reinvent the world, but we essentially wanted to provide this glue layer of sorts that made it easy for people to interact with all of these components that have been very well architected.

Now, I'm not very intimately familiar with Flyte, but from what I have read and heard about it, Flyte is a very Kubernetes-centric ecosystem and they do also have some integrations with Amazon ECS as well. But then they're positioning themselves more as an ETL workflow orchestrator. If, say, your company has a requirement or an ETL orchestrator, then yes. At that point in time, maybe you would be comparing against an Airflow or a Flyte or an AWS Step Functions. Similarly as Metaflow is orthogonal to an Airflow or an AWS Step Functions, it's equally orthogonal to Flyte. Theoretically, you can take a Metaflow workflow and you can compile it down such that it can now execute on your Flyte installation.

[00:34:13] JM: Makes sense. Do you think the fundamental machine learning problems at Netflix are significantly different from those at Lyft?

[00:34:21] SG: I'm not entirely familiar with the machine learning challenges that Lyft is trying to solve, where it would be really hard for me to comment on that. But I guess like one luxury that my team has at Netflix is that Netflix has a very wide diversity of problems that they're trying to solve. So like we have people who are building their models in Python. We have people who are building their models in R, people doing work in Scala. We have folks who are working on problems in computer vision and classical statistics, NLP. I mean, you name it, and it's very lightly that we have folks at Netflix who are using any of these tools and techniques and areas of machine learning.

What that provides us is an excellent bird's eye view of all the diverse set of problems that these people face. And that's why for Metaflow, we are trying to make sure that we are not being opinionated about the top layer of the machine learning stack, which is essentially what sort of machine learning library do you intend to use. What sort of code do you actually want to embed in your machine learning workflow? The opinions that we take are very much at the very lower level of the stack in terms of how do we efficiently store your data in S3? How do we bundle your code and make sure that it executes on Amazon ECS or our container orchestration system that we use internally at Netflix called Titus, and you get the exact same set of results that you were getting on your own laptop.

So those are the concerns that we take care of. We take care of versioning by default. We make sure that all the specifics around model operations, or these are like managed, change management. Somebody depends on the output of your workflow and how do we coordinate those changes and all. Bake all of that into Metaflow while still affording the entire flexibility to our data scientist so that they can take care of any of the diverse business problems that they are focused on.

[00:36:19] JM: Let's go a little bit deeper into a flow. So single flow. Maybe you could give an example of a flow and talk about flows, steps and tasks within Metaflow.

[00:36:31] SG: Let's take a dummy example where you are, let's say –

[00:36:35] JM: Maybe recommendations.

[00:36:37] SG: Maybe. Yeah. Let's take recommendations, right? So you have some input dataset that's stored somewhere, and this dataset could number in like terabytes. But let's assume some huge data. Now, the first thing that you would want to do in your workflow is you have some way of accessing this dataset. Maybe you want to select some small partition of this dataset. Then the next step would be that now you want to train a model or, say, a bunch of models. And then your final step notionally would be that if you were training one model, then you would want to make sure that, okay, that model is sane. If let's say you were training a bunch of models, then you would want to pick the best model, all of those.

Essentially, now, what you end up at is this multi-step workflow, which looks pretty much like a directed acyclic graph. So you have this first step, which is fetching the model, doing some feature engineering. Then you have yet another step where you can be training one or more models in parallel maybe. And then the final step, where you're just confirming the results, making the best model if you were training multiple models and storing it somewhere so that then it can be picked by some other process for consumption.

Now, in Metaflow, each of these nodes are essentially steps. You'll write your workflow as you would write any [inaudible 00:37:59] class, and you would be sub-classing this flow step class that Metaflow provides. Every single function in your class is going to be annotated with this decorator known as Step. And this will sort of like indicate to Metaflow that, "Hey, this is my step of the workflow." Every single function will also have the special method, self.next, which will essentially tell Metaflow that, "Okay. Once this step is done, you need to execute this next step."

And we also provide some tooling. So for example, let's say you want to train a thousand different models. So you can specify a list of parameters, and we have this construct call for each, and you can say that, "Hey, the second step, I actually need multiple instances of the second step to execute, and the cardinality of this model training step needs to be dictated by the cardinality of this list of parameters that I'm parsing. And every single instance of this model training step should get one value from this list.

An example here could be that, "Hey, I have three hyperparameters and I want to train my model or three instances of my model with each of the value of those hyperparameters. So my code still stays the same and I'll just like point it to this list. And now in my workflow, Metaflow will first launch a job which will fetch the data from the data warehouse. Do feature engineering in idiomatic Python code using any and all the libraries that the users are already familiar with without actually placing any sort of constraints on that. Then Metaflow will launch, say, in this example, three parallel jobs on behalf of the user to train three models in parallel. And then in the final step, Metaflow will then parse the user. All the output of all of these three different steps or three different model training steps, so that then the user can compare and contrast these three different models and pick the best performing one.

Now, to point out like one big difference with, say, workflow orchestrators like Airflow and AWS Step Functions, usually what happens whenever you have a DAG orchestrator, state transfer is sort of like left as an exercise to the end user. The user is then responsible, let's say, at the end of the my first step, I created a data frame. Now, my model training step needs access to that data frame. Then the user is responsible for storing that data frame somewhere and then getting that data frame back in the model training step. Metaflow essentially extracts all of that away. So the user isn't every even thinking that there's a backing data store in S3 where all of these data is being persisted.

We also have a bunch of other decorator. So one other decorator that's of note here is the at resources decorator, where every single step of the workflow, you can specify what sort of resources you want. Say my model training step needs every single model training run might need two GPUs. So I can just annotate my model training step at resources GPU equals two. Then what Metaflow will do is it will run your feature engineering step, say, on your laptop. Then it will [inaudible 00:41:09] compute three instances on the cloud each of them with two GPUs where Metaflow will execute your model training code. And then it will bring back those results to your laptop so that the last step can execute on your laptop. And you can mix and match. All of your steps can run on to cloud. One of them can run on the cloud.

Another example here could be that, say, when you started running your workflow, when you started iterating, you were iterating on a very small piece of dataset. Because, say, you had 16 gigs of RAM on your laptop and you can only fit so much data. And now you want to run on a much larger set of data. Let's say now you need access to 400 gigs of RAM. Then at that point in time, you can very simply just drop in this declarator and you're ready for a step. And then Metaflow, instead of running the compute on your laptop, it largely run the compute on the remote instance and it will pipe all the logs to your laptop.

So in a sense, to the end user, it would just seem that their laptops suddenly went from having 16 GB of RAM to 400 gigs of RAM. And that's really like super helpful for our users when they are trading and prototyping, because now they are not limited by the fact that they just have very finite set of resources, the fact that it can very easily and simply interface with the cloud without any changes to their code. That unlocks a lot of potential.

[00:42:28] JM: Right. You've talked about model parallelism here and the idea of training multiple models to test and get a sense of which features are most relevant to a machine learning task. Tell me a little bit more about the necessity of parallel model training and how that can be made as easily as possible through a framework.

[00:42:56] SG: So scale is a very important aspect. And as I said before, we provide a whole bunch of primitives for users to sort of like vertically or horizontally scale out their compute. If you look at how the cloud ecosystem has evolved over the past few years, when AWS launched in its very infancy the instances, like the peak memory that you could get was somewhere of the order of like one or two gigs. These days, you can very easily get terabytes of RAM.

Increasingly, more and more problems that earlier would need people to think in sort of like a MapReduce fashion can now be done on a single machine. The overall cardinality of problems that can be solved in that manner, it's only going up as days proceed. So the very first recommendation that we have to our users of Metaflow is that before actually thinking about data parallelism or model parallelism, just see, like if your work can be done very simply by getting access to a much bigger instance, and at times that's the more simplistic approach. And if let's say you go to a distributed first approach, no matter how simple it is, it's going to introduce a lot of entropy and debugging [inaudible 00:44:16].

And then at times, you simply just cannot do with a bigger instance. Like a use case here would be when you want to train multiple models for a particular use case, and you can very simply, using some of the primitives that we provide, [inaudible 00:44:34] compute to multiple container instances. At Netflix, we have use cases where people [inaudible 00:44:41] compute to like upwards of 10,000 containers in one go.

Then the third piece is that, "Okay. Now I actually want to do distributed model training." For those use cases, we provide very easy on RAM to say, for example, AWS SageMaker. You can get access to distributed TensorFlow, and Metaflow will then move your compute on top of

Sagemaker and then pull back the results down so that then the rest of your workflow can proceed as is. That's sort of like the viewpoint that we have taken so far.

[00:45:12] JM: And the metadata that might get kept at each step in a flow, I'd like to describe this in a more detail, because there is this stateful component to machine learning training. Can you tell me about how the metadata within a flow is managed?

[00:45:37] SG: When we talk about metadata within a flow, you have two pieces of that, right? One is, say, I'm training a TensorFlow model and then there's just some TensorFlow related metadata that needs to be processed. What Metaflow does by default is it will take all the outputs of the user code and it will store that in S3 in perpetuity in a content data store. And it will allow the user to access that at any later point within a notebook, for example.

So an example here would be that, internally, our users, they train their models using TensorFlow within Metaflow, and then all of the information is stored in S3 and then they can use tools like TensorBoard and point TensorBoard to that S3 location to very easily and simply visualize their models. All the TensorFlow specific metadata is available to them, and Metaflow places no assumptions or opinions on the kind of data that can be stored.

Now, the other piece of metadata that Metaflow tracks and stores is around the flow execution. Every single time you execute your flow, we'll assign it a new ID and we'll execute the flow in an isolated username space. We ship with a metadata service where we'll track all of these pieces of metadata so that at any point in the future, you can go in and you can be like, "Hey, I have this training run. I need to know what was the model that was generated by 360th training run of my training flow X, Y, Z."

And then because we have persisted all of these information in our metadata service as well as stored the actual data in S3, we can very easily provide that user the results that they seek. That's how a lot of our inferencing use cases are also powered at Netflix, where a user will use Metaflow to train their model. And then we also provide as a function service platform where they'll very simply write some Pythonic functions referring back to the artifacts or the models that were produced by some of their prior training grounds. And then our function as a service platform will just take that function and host it as a microservice frontend.

[00:47:54] JM: interesting. Could we talk more about the underlying AWS systems that would be called by one of these step function workflows? So in other words, you've mentioned like data warehousing and S3 or course. But I just like to know more about the AWS services that you've settled on for different components of implementing one of these metaflow flows.

[00:48:23] SG: Nominally, what metaflow needs is a place to store the data. We rely on S3 as our data layer. Then we need a place to compute or execute the user code. So our current integration is with AWS batch, which is essentially one way to visualize AWS batch is that it's a job queue in front of Amazon ECS.

And then the third component, if people need access to that, is some way of offloading the execution of their flow from their laptop to the cloud, and that's where we use AWS Step Functions. So that's again yet another managed service.

Then we also have this metadata service, which is [inaudible 00:49:10] app on top of RDS. This metadata service is an optional component. So if let's say you intend to collaborate with other people and give other people access to the models that you have built. Then this metadata service is going to act as a lookup service to know where actually that model is stored in S3. But besides that, there are no other components to Metaflow. Metaflow at the end of the day ships as a Python library. So if you have an S3 bucket, if you have an AWS batch job queue, then you can very simply just pip install Metaflow, configure it with this information and you can get going and you would pretty much have the exact same setup that data scientists at Netflix have for their production deployments.

[00:50:00] JM: I'd like to know more about what are the pain points that you are still experiencing even after building Metaflow. What do you think is on the horizon for what else you might build into the platform?

[00:50:12] SG: Sure. I think we've like barely scratched the surface in terms of machine learning infrastructure. One problem area or one area that we are focusing on recently is around this notion of model ops. Collaboration is a big, big thing in software engineering [inaudible 00:50:31], right? Software engineering has made big strides with the notion of version control.

But we still lack that functionality in the domain machine learning systems. Now that we have enabled people to be self-sufficient in building these workflows end-to-end, very quickly you sort of like find yourself in a situation where different data scientists, they'll write workflows which depend on other workflows or which depend on the outputs of other workflows.

In the domain of, let's say, like ETLs, the notion of failure is limited to catastrophic failures, where if something is not working, then your pipeline fails. While for machine learning systems, failures are really hard to triage and even harder to identify in the first place. Your reads are going to be off by a little bit, but you wouldn't necessarily know if that's expected because of changes in your data or if, let's say, some underlying dependency changed under your [inaudible 00:51:31].

Building tools to identify these sort of like problems and prevent this from happening, that's sort of like the big focus for my team. So we already have some solution out there. For example, we have this very tight integration with [inaudible 00:51:50], which sort of like allows to create this encapsulated closure of the user's compute, which has the user code, the data, the dependencies, and then we ship around this closure and we make sure that the execution characteristics of this closure, they stay immutable.

But we still need to do a lot of work where the users sort of like take an action, changes their workflow, and that can cause issues in downstream workflows or downstream applications. And we want to sort of like provide appropriate tooling for our user base so that they can prevent and identify these issues and sort of like get ahead of this.

[00:52:27] JM: Yeah. I'd like to kind of close off with just a high-level question, which is what I think about machine learning models and developing a feedback loop between the user and the developers on the machine learning model, you would always like to have some mechanism for getting feedback from the user, from knowing what is working and what is not working within a machine learning model. Because obviously you can always have features that you're going to be training the model around, but there's sometimes where the features that you might put plugin to your model are not really telling you enough information about whether or not a model is working. Are there any opportunities you see for developing better mechanisms? For getting feedback from a user?

For example, in a passive experience, like Netflix. When I'm sitting there, I'm watching the film and you can't really tell how much I'm liking the film unless I do something explicit like stopping it, but that's sort of a coarse-grained input mechanism. I'm wondering about finer-grained input mechanisms, if you have any ideas around that.

[00:53:39] SG: It's a journey in the sense that if you look at, say, 10 years ago, the contract between data scientists and software engineering teams was this notion of an idea, where data scientists would prototype an idea and then they would be machine learning engineers, and other software engineers will take that idea and sort of like build a machine learning workflow and a machine learning system around it.

The problem that people used to run into was that software engineers, they have a different vocabulary; and data scientist, they have a different vocabulary. So often, like these two worlds won't really understand what are the characteristics of a well-functioning system. Then slowly we moved to a world where the contract shifted to a model where the data scientist were responsible for building and training a model, and then that model was embedded into a production system. In that scenario, again, there was this sort of like a dichotomy of sorts, where the data scientist has built a model. But now they don't necessarily have control over how their model is being scored. I mean, they might still be responsible for the business logic, but they don't have complete control over the actual input. How is that input being generated? Do you have any sort of like skew in terms of the features that are being passed into your model at inference time?

Now, with Metaflow and with some of the other tools in the ecosystem, the contract has changed to being an API, where the data scientist at the end of the day is responsible for delivering an API to the software engineering teams. And the software engineering teams can then – Are completely abstracted away from many of the machine learning concerns.

I guess the next frontier is that, "Okay. How do we sort of like get more and more control or more and more flexibility to the data scientists so that they are in control over identifying what is the actual input that is being passed to their model? Is it well-formed or not? It's a really hard problem. I know there are plenty of individuals, plenty of companies who are hard at work trying to find a solution to that. And it's something that we are also keeping an eye open and something that will very soon [inaudible 00:55:46]

[00:55:50] JM: Savin, thank you for coming on the show. It's been a real pleasure talking to you.

[00:55:53] SG: Thank you.

[END]