1:14

Alexey

**This week we'll talk about MLOps and building machine learning platforms. We have a special guest today, Simon. Simon has been building ML platforms for over half a decade. Currently, he is a Lead MLOps Engineer at Transaction Monitoring, Netherlands (TMNL). I was always wondering what TMNL stands for and now I know. It is a worldwide unique initiative of the big five banks of the Netherlands. Next to his work at TMNL, Simon is also a university lecturer for data mining and data warehousing. Welcome to our event.**

1:56

Simon

Thanks a lot. Thanks for having me.

# Simon's background

2:00

Alexey

**Before we go into our main topic of MLOps and ML platforms, let's start with your background. Can you tell us about your career journey so far?**

2:08

Simon

Yeah. Actually, I started out my career while doing a Ph.D. – being a research and teaching associate, actually, to the Vienna University of Economics and Business. There I was doing some research in the area that was more focused on computational advertising. Machine learning was always applied to problems in the space of online advertising. Well, I wasn't a Ph.D. student for that long because, in the end, I found some very interesting challenges in practice. So I became a data scientist and worked for a consulting company in Vienna for quite some time. I started off as a data scientist and worked in a lot of interesting industrial AI use cases – visual inspection, predictive maintenance, all these classics. Then step by step.

Well, back then we also started to develop... This is also kind of how my MLOps journey started – we started developing a deployment and serving platform for our models that we built for our clients, simply because we actually found that this, back then, was a significant blocker when it comes to actually creating value with your machine learning models. That's also pretty much how my journey in the MLOps world started. Back then, at least, we were not aware of the name “MLOps” yet. Deployment and serving platform – that already sounded way too cutting edge for many companies that we were dealing with. After working in consulting, I moved to the Netherlands, where I'm still living.

There I joined bol.com, which is the largest e-commerce company here in Belgium and the Netherlands. I think even larger than Amazon, although Amazon also entered the market, I think two years ago. There, I was an expert machine learning engineer, a kind of Staff Engineer for Machine Learning. We dealt a lot with natural language processing, trying to understand our customers, building some transformer models, and working on GCP and Kubernetes. After that, that's a change to where I'm still at now – TMNL, Transaction Monitoring Netherlands. Here, I am the Lead Engineer for Machine Learning Operations. Actually, to be very accurate, I used to be the Lead Engineer for MLOps until two weeks ago. Now, my official title is Managing Engineering and Development – a bit of a different thing. MLOps is really, really still very close to my heart, but the focus in the data work has shifted a bit.

4:35

Alexey

**You're a manager.**

4:37

Simon

I'm more focused on the effectiveness and collaboration of various tech teams.

# What MLOps is and what it isn't

4:42

Alexey

**Which is also actually a part of MLOps. Maybe we can also talk about what MLOps is and what it is not. I'm really interested to hear your take on that. Because for me, MLOps is not only about the tools you use for deployment – it is much more. [Simon agrees] So maybe we can talk about this right now. What is MLOps in your opinion? You said that five years ago, you had no idea that the thing you were doing was called MLOps, you just call it deployment. But right now MLOps is more than that, right? So what is it?**

5:11

Simon

Yeah. I fully agree. Typically, when people hear the term MLOps, the things that always pop up in their brains are feature stores, experiment trackers, and model registries – that's what comes up. And that definitely is one part of MLOps. It's the tooling part – more the tech part, I guess. But at least I believe that doing machine learning operations successfully is a lot more than the tech part. It's actually... Well, the classic thing is that it's about people, processes, *and* technology.

The technology part, okay – feature stores, model registries – these pieces are what people are typically familiar with. But introducing machine learning operations successfully is a lot more than that. It requires processes, understanding how models are developed, how models will be developed, what are the actual use cases and the requirements we need to fulfill – this is actually where MLOps starts, understanding the processes behind model development, deployment, and serving. Because in the end, the tech part of MLOps is all about streamlining and automating exactly these processes.

Of course, the third part is people. Machine learning operations is still a fairly novel realm, and a lot of companies do have the challenge to think a bit about, “What skills do we actually need? What does it mean to build? What does MLOps mean?” And that might also mean something different for every company but, “What skills do we need to build a machine learning platform? What do we need to bring a model to production? To bring 100 models to production? And to operate them?” It's really a lot about people as well – people in terms of skills, but also people in terms of how they collaborate. So MLOps is a lot more than the tech part.

# Skills needed to build an ML platform that serves 100s of models

6:55

Alexey

**Yeah. To me, when I heard the term MLOps for the first time, I thought, “Okay, I've been doing this thing for so long.” But in my mind, it was mostly deployment. Like you, I think I followed a similar path. I started as a data scientist and then I noticed how difficult it is to deploy models – it's a significant blocker in the process of productionizing ML models. We were not really building a platform for doing this, but we were thinking, “How exactly can we make it simpler?”**

**When the term MLOps came about, I thought, “Okay, I'm such an experienced MLOps person. I was doing this before it was cool.” But then I started learning more about this. There are things like a feature store that I had no idea what that was. Or experiment tracking – I understood that this thing that we used was actually an experiment. So there is much more.**

**But, as you said, in addition to technologies, there are also people and processes. One thing you brought up was, “What skills do we need to build the platform to serve hundreds of models?” Did you find an answer to this question or are you still looking?”**

8:11

Simon

In principle, it's a difficult one, because it very much depends on the general organizational setup and fundamental beliefs of the organizations. At least in my world, and that's typically also the world I select, I believe in principle end-to-end, responsible teams for shipping products. That means, in my world, what I believe works well, is the building, the deployment, and the serving itself, being in the hands of what I call “streamline teams” teams that actually create value for the organization with their products.

A platform then, I considered on the one hand as a way to streamline processes – how these streamlined teams, teams actually working on ML powered products – how they develop their models, deploy them, and serve them. So it's about building a platform to streamline their processes, but also to make the processes faster, to make them worry about them less, and reduce the cognitive load on these teams.

When you think about building this “platform,” which is then really not actually about developing the model, then the skills that I typically saw to be incredibly valuable is this mix of infrastructure and cloud knowledge. Because these days, in most organizations – in many, many organizations – you do build your platforms, your products, in some kind of (whatever) cloud, whether it's AWS, GCP, Azure, etc. So the infrastructure and cloud knowledge is definitely something that is incredibly important for building an ML platform.

9:53

Alexey

**Things like Kubernetes, that sort of thing?**

9:55

Simon

Kubernetes, Terraform – knowledge of, let's say, AWS services and how they can help you build what you want to build. Also, next to the infrastructure knowledge, it's really knowing how models are built – knowing your users, actually, your customers. Because as an ML platform team, your customers, your users, are typically internal data science teams, or at least product teams with some element of data science in them. Having an understanding for them is quite fundamental.

10:31

Alexey

**But do you mean that a platform engineer needs to know what finding a derivative in the functional space means?**

10:38

Simon

No, not at all.

10:40

Alexey

**So it's “XGBoost exists and whatever it gives – the model outputs some numbers.”**

10:47

Simon

Yeah, so pretty much the latter. I believe that to be a successful ML platform engineer or MLOps engineer as I sometimes call them, you have to know the data science workflow – how do data scientists actually work? You need to have an understanding that, yes, data science is an experimental discipline as well. There needs to be space and support – tooling support, process support – for doing experimentation, for example. All these are things that, if you come from a classic software engineering background – typically, this is something you have not quite seen or you don't quite understand why somebody would work in a notebook, right?

11:23

Alexey

**Yeah, for me, that was the first reaction when I saw a Jupyter Notebook. [Simon agrees] I was a Java developer and then somebody showed me a Jupyter Notebook and said, “Okay, this is how we do things.” And I'm like, “Oh, my God. Really? Where's my...” Back then I used Eclipse (IDE), so I was like, “Where is my IDE? What is this? It's awful.” [chuckles]**

11:48

Simon

Yeah, as an ML platform engineer, you need to understand why people actually choose this and how...

11:54

Alexey

**Why would you program in a browser? Right? [chuckles]**

11:57

Simon

Yeah. Yeah. So understanding your users, understanding how models should be deployed, what deployment patterns exist – these things matter. What doesn't matter for an ML platform engineer, typically, is what you said, “Why would I choose a root mean squared error over a mean squared error?” For an ML platform engineer, that is not quite important. It's important to understand that there are certain valuation metrics, perhaps – on that level.

12:29

Alexey

**These metrics exist, but I don't really need to know the difference between mean average error versus something else.**

12:43

Simon

Yeah, exactly. At least that level of knowledge should be sufficient to build tools that help your data scientists do things more effectively. Of course, the deeper your knowledge, the better. But there are hardly any unicorns, right? So you need to prioritize a bit. Typically, the two things are important, and the third thing is, obviously, for writing Terraform, you also will need to write some Python, for example. You will need to perhaps write some Java, depending on your context – so the classic software engineering part also is of importance.

13:25

Alexey

**So infrastructure and cloud are the first thing. Then knowing about the process of building models is the second thing. You do not need to know them in detail, but you need to know how the process starts, what things the data scientists do, and what the output is. That's the second thing? [Simon agrees] And then the third thing is being a software engineer.**

13:48

Simon

Correct. Yeah.

# Ranking the importance of skills

13:50

Alexey

**How important are each of these things? If you put them in order in terms of importance, what's the most important one? And what's the least important?**

13:58

Simon

Hmm. Good question. In principle–dodging the question a bit–I always believe that a team needs to have the sum of these skills. That needs to be right. In a team, you might have specialists who really have a big strength on the cloud engineering end. For them, for example, that's what they would add to the team and they might have close to zero knowledge of how models are built. And that is okay if there is another person who can augment it.

But if I had to rank them for one person, I would say infrastructure knowledge is number one, software engineering knowledge is number two, and understanding of how models (how data scientists actually work) is number three. Also, because I believe that's probably the thing that you can catch up on the easiest. Also your users (your customers) will let you know, hopefully, when you build stuff that just doesn't work for them.

14:53

Alexey

**Or have a data scientist on the team – somebody that was a data scientist, but now is more interested, let's say, in software engineering or platform engineering.**

15:02

Simon

Exactly. That's what we typically try to aim for also at TMNL currently. In the machine learning operations team, who are building the ML platform, that's also what we have been seeing is really, really effective – bringing a mix of specialists and a bit generalists together. Some specialists in cloud engineering and infrastructure, and some people who have been data scientists, and over time transition. Typically, the sum of these parts really makes a good ML platform team.

15:34

Alexey

**How many people should there be? At least two?**

15:39

Simon

Well, it depends a bit on your platform's availability requirements. For example, if you need to have people on standby and so on, then you need to factor that in.

15:54

Alexey

**You mean if we want to make sure that this platform is up and running all the time, then somebody will need to be on-call. [Simon agrees] Then if something happens during the night, they would get notified and they would wake up and fix the thing. One person cannot do this, so there should be at least three people.**

16:16

Simon

Absolutely. Even two people cannot really do this. It really depends on the load on your platform. How many teams and people use it? How business-critical actually is it? That, I think, defines a lot of the real headcount requirements. If you just think about building it – let's ignore any significant operational overhead that comes from just having it up and running 24/7 – four people, five people, six people are typically nice numbers for an engineering team. But you can also have a nice mix of skills. It's difficult to answer. It depends.

# The point where you should think about building an ML platform

16:52

Alexey

**At what point should I actually think about building? Let's say, six people – especially now, when everyone's budgets are kind of tight – does it actually make sense to build a team of six people while they can be doing something else? Maybe it's a good idea just to buy an existing platform? There are quite a few on the market. Right?**

17:14

Simon

Yeah. I think these days, there are not many companies who make, and should make, the choice to build an ML platform from scratch. I think usually what companies look into – companies that are not Uber or Amazon, and these these big tech companies – typically what normal companies look into is, “How do we buy tools from vendors, how do we integrate them into our landscape, and how do we make them work together?” Usually, it is more, “What do I buy?” And maybe some parts you build yourself. But even if you “buy” platforms or parts of platforms, there is a lot of integration effort – gluing things together and making it fit to your workflows – because these might be very different depending on your organization, depending on your use cases.

Even if people advertise end-to-end platforms, you can be quite sure that you will either need to adapt your processes or you will just need to bend some things to actually make it work for you. But to your original question, maybe – when should you start building a platform? Typically, there are a few “smells that you see,” which gives a bit of an indication that you should consider thinking about a platform. For example, you have a set of engineering teams that have data science somewhere in their products. Let's say you have five or ten teams, and a couple of these teams make products that are powered by some model.

Let's say there's a team that takes care of some recommendation engine, a team that takes care of some natural language processing. I'm also thinking now a bit about my ecommerce experience. If you then see the themes, reinventing the wheel and doing training, serving – all of these things – in very different ways without actually having a really good reason, that's usually the point where you should think about, “Hmm. Maybe a platform that helps me standardize some things and take away rebuilding things. That could then definitely make sense.” And typically, then you can also calculate the business case and see if it will pay off or not.

19:45

Alexey

**One thing you mentioned was that teams do things in a different way, and “teams” is literal here. It means that you have to have at least multiple teams in your organization in order for them to do things differently. Right? [Simon agrees] If you're a smaller company, you may just have one team.**

20:04

Simon

Yeah. I think in that case, it doesn't mean that you should not consider a platform. Because even for one team, at least some elements of a platform – for example, an experiment tracker, which is only one piece of a larger platform when you build or procure it – that is something that can be super important and a massive boost in effectiveness, even for one team. These things, especially if you go for some isolated pieces, typically there are SaaS offerings, where you do not need to worry about any maintenance whatsoever. That is something that comes with very, very little overhead – engineering overhead does cost something, obviously. But these are things that you should consider in any case, even if you have one team building things. When I talk about platforms, typically, in my mind, it's more comprehensive software and infrastructure that helps you do what you want, but at scale.

# The importance of processes in ML platforms

21:03

Alexey

**One thing we talked about was processes. We talked about skills that people need to have, but we also mentioned processes. Actually, one of the skills is understanding these processes and understanding how data scientists build models. Another thing you mentioned is, if you use an external platform – for instance, for building an external ML platform – the flow they have (the process they have) in mind when building this external platform might not be the same as the process you have. Then you would need to readjust (to re-do) your projects in a way that fits this platform. The processes here seem to be quite an important part. I'm wondering what these processes are. So can you maybe walk us through a simple process?**

21:57

Simon

Yeah. Simple process. As a simple process, usually, a data science workflow starts with pulling data. That's typically where the work of a data scientist starts. Let's say you want to do some exploration because you want to start building a new model – you want to train a new model. Your process (your workflow) would start with pulling data into an exploratory environment (into a notebook, for example). That's usually where it starts. So then you go on to... perhaps you need a cluster to actually do proper exploration and experimentation on that dataset? Well, again, a branch of your process will be, “Well, you actually have the need for a bit more powerful, scalable compute environment in an exploratory setting.” Part of your process is starting to branch off. Then you will train something – you need to evaluate it, you want to keep track of your experiments. That will also be a piece of your process – something that your platform should help you to do, that it should cater to – keeping track of your experiments, of your model training and evaluations.

Then, as the next step, you want a persistent model, and perhaps share it as well, and make it available to services – downstream processes. Now you need a model registry for that. The story goes on. How is that model going to be consumed by a downstream service? Does it even need to be consumed? Is it maybe only a batch job that actually runs this model? When we speak about processes, that's exactly it. Depending on your use case – depending on how your people built these models – that process will look different and you might have several processes, depending on your team and your use cases, again. So when I say you need to understand your process to build a platform (to think about tooling, even) this is exactly it. You need to understand the data science products are built in your company.

24:01

Alexey

**So for example, if most of our projects (like 78% of the ML projects) don't need to be up and running all the time – we just execute them in batch – then maybe we do not need to invest a lot of time in making a platform that can serve these models online? [Simon agrees] You should focus on batch first. Right?**

24:24

Simon

Exactly. That can be a very... I think what you said is already one step further. Prioritization. Because you could build this beautiful platform that does it all and serves every single thing that you *might* want to do. But that's not how you typically build. You want to build iteratively and incrementally when you build your platform. So you need to prioritize and if you see, as you said, that 70% of your models – and let's say the value that these models generate is equivalent to the quantity – that's typically what you want to build out first.

# Weighing your options with SaaS platforms

25:00

Alexey

**It's also important that you decide which platform to buy, because maybe not all the platforms support batch. I know I definitely saw a couple that do not support batch mode. They only support online, like web services. Then I was like, “Do I really need this? It's not really what I want.” Some of these platforms offer “batch” which is just sending a lot of requests to the online service.**

25:30

Simon

Yeah, that's a good one.

25:32

Alexey

**I'm thinking, like, “Do I really need that? Or maybe I need something else?”**

25:35

Simon

Yeah, it's a very good point. Even thinking of Amazon SageMaker – a very popular service, especially for companies already on AWS. The way that batch processing is recommended is pretty much what you said. They call it batch transform. What it does is spin up an endpoint, shoots, let's say, your complete batch run against it – and in batches, we often deal with large amounts of data – it shoots 100 gigabytes of data against it, and then it tears down the endpoint again. And that's batch mode. It's actually spinning up an endpoint, tearing it down. Whether that's really what you want, and whether it is cost-effective for you, and whether this is optimal from a performance perspective, is questionable, of course. That's what you really need to look into. A very good point that you made.

26:18

Alexey

**On paper, they have “batch transform,” but you into this, it's like, “Oh. That's not what I need.” This comes back to the second set of skills that you mentioned – you need to understand the process, (how exactly models are built) in order to understand that, “Okay, this actually makes a difference. If I want my batch jobs to be fast, then this platform does not work for me. I need to do something else. Maybe I need to build my own stuff with Spark or whatever.” Right? [Simon agrees] I see.**

26:58

Simon

And that might not be the killer argument to not to go for the platform, but you will need to build something custom or go a different route, perhaps. Bend it. For example, TMNL is heavily using SageMaker as well. Well, I need to bend some things and find other ways of how to do batch processing without using batch transform. But that also bites you a bit, because in that specific case, if you followed the recommended path, then you would still get some nice features down the line – automated bias unfairness detection out-of-the-box (Clarify they call it). That's what you do not get, or the integration is just so much harder, if you go for the non-batch-transform – the way that when you don't spin up an endpoint, shoot your data against it, and tear it down again. So there are downsides to then having to branch off and build custom stuff.

# The exploratory setup, experiment tracking, and model registry

27:59

Alexey

**Okay. To summarize, the process is: first is the exploration phase, where you need to pull some data. For that you need a data processing platform, where it can explore things quickly. It could be a data warehouse, or a Spark cluster.**

28:20

Simon

It could be, let's say, GCP of BigQuery and then you have some Colab notebook and you authenticate to BigQuery, write your SQL query, and the notebook pulls in your data. That would be an exploratory setup. Of course, you want to have enough infrastructure power behind your notebook so that you can actually do what you want to do, usually.

28:41

Alexey

**I think Databricks also offers this kind of stuff, right? [Simon agrees] I mean, in place of Spark, but...**

28:49

Simon

Yeah. Databricks, AWS has it – I think pretty much all the big cloud vendors. But the platform component about that is really giving your data scientists the ability to provision the resources that they need to do their job. Obviously, as a data scientist, you don't want to then configure and spin up via infrastructures, code your own cluster. But what you want to do is just click some buttons to spin up your cluster and connect. This is really the platform part – making it easy for people to do their work.

29:21

Alexey

**So they don't need to clone a Terraform repo or create an EMR cluster there, wait for some platform engineer to approve this, and then apply...**

29:34

Simon

You want to build a self-service capability so that it's easy for people and you don't need to worry about infrastructure as code and these things.

29:41

Alexey

**Okay, so that's the data exploration part, where we pull the data, we explore, and we see what we can actually do with this data. The second step is, once we did the initial exploration, we train and evaluate models. Then you mentioned that we need to experiment tracking tools, right? So that's another set of tools (or another tool) that we need in addition to the first one. Right?**

30:06

Simon

Yeah, I think an experiment tracker is something that most teams – specifically teams that at least use some evaluation metric to evaluate their models – could benefit from a lot. It's usually one of these low-hanging fruits, just to move from keeping track of your experiments in an Excel sheet to actually something that... Well, that works – that's scalable and also shared and transparent (to your team, at least).

30:32

Alexey

**Then the next thing is persisting the model – making it available for downstream usage. You also mentioned that we need a thing called a model registry. I know that the experiment taking and model registry are usually the same tool, for example, MLflow. Right?**

30:48

Simon

Yeah, very often they go hand-in-hand.

30:50

Alexey

**I know Weights & Biases or many other platforms, also – AWS, SageMaker, GCP I'm sure also has it. Azure has it. Right?**

30:58

Simon

Yeah, it very often comes in a package, especially experiment tracker, model registry, metadata store, meta data tracker and store – that is something that, when you look at MLOps tooling vendors, it's something that you very often see packaged in one SaaS offering.

# What comes after deployment?

31:15

Alexey

**Then we kind of finished the training phase, and then we go to the deployment phase. We need to make sure that somebody can consume the output of this model. Then we talked about deployment – we need to understand if we want to serve this online thing, serve as a web service, or it should be a batch job. We also talked a bit about the tools. You mentioned that it's possible to do with SageMaker. I think I brought up Spark. So there are a bunch of tools like that. After deployment, there is something else, right? It's not the end of the process yet?**

31:51

Simon

Yeah. I think even deployment typically depends a bit on how opinionated you want to be as an ML platform. That's also a piece that you could build, and you should consider building for your teams – reusable, centralized, managed deployment pipelines – especially if you have some narrow use cases. Let's say two or three use cases that you do very, very often. If the patterns that the models follow are pretty much the same, then you should even consider building and managing centralized deployment pipelines. Even that is something that you could take away from your data science focus teams. It's not always a good thing – it doesn't always make sense. It's something to always carefully weigh between flexibility in the teams and what you, as a platform, push out. That could be something.

After deployment? Well, it's about serving. Serving, at the principal decision, is always, “Well, is it batch? Am I just going to load the model in some batch job – in some, let's say, Spark job? Do some pre-processing and run it and store my predictions in some table?” For example, that's an option. I think there, it's usually not so different from your training infrastructure. Typically you would choose some workflow orchestrator: Airflow, SageMaker, SageMaker pipelines (if you want to be in that ecosystem). Typically, it's a similar tooling choice, at least as you will also do for training, when it comes to orchestrating what you actually want to do.

In the end, performing model inference in a batch job is not that different compared to model training in a batch job. It's usually a sequence of jobs – data loading, pre-processing, feature engineering, training/inference, and then just your output artifacts are different. On the one hand, you have a model as an output artifact, and you would store it in the model registry, whereas in the batch inference job, you will usually have data, predictions, whatever, as output and store it somewhere.

# Stitching tools together to create an ML platform

34:01

Alexey

**When we talk about building a platform, do we actually mean that, “Okay, let's create an experiment tracker from scratch.” Or “Let's create a serving infrastructure from scratch based on PaScal (or whatever).” Or is it that here, we mean more like, “Okay, what are the tools that are available there? Let's see how we can take these tools, see if these tools fit, whatever requirements we have, and how we can stitch together these tools into something meaningful at the end.”?**

34:28

Simon

Yeah, the latter. Again, the former... I cannot really not think of a reason why you would build your own experiment tracker. I'm certain that there are good reasons in some very, very niche use cases. But these tools have become a bit of a commodity even. There are lots of these tools out there, from open source to self-hosted open source solutions, to fully-managed SaaS solutions – pretty much everything you can think of. I think they're are very few reasons why you would really build that experiment tracker yourself. Usually, it's really about getting the right tools, integrating them, making them easily consumable, and making them fit to your data science workflow.

35:14

Alexey

**It sounds like it's not a difficult job. But I think it's actually the opposite, right? [chuckles] [Simon agrees] You still need to connect these tools somehow and make this a seamless experience.**

35:26

Simon

Yeah. I think it's a common misconception that people have, when you think, “Well, such an ML platform? I mean, I'm just gonna buy SageMaker (or I'm just gonna buy Vertex AI).” Yeah, it's not so easy, usually. Well, buying it is very easy. The company is happily gonna take your money and give you access to their compute infrastructure. But again, as you mentioned, the devil is really in, “Does this really support what I want to do? Does it support what I want to do given certain constraints (most companies have constraints) meaning data governance, security, specific types of models?”

Nowadays, when you think about large language models, for example, it's not trivial to fit some special needs of large language models into existing ML platforms. I think what you can see, specifically based on this, what you could see in the last one or two months, is that so many vendors in the MLOps space have pushed out really, really nice updates to their platforms (to their tools) that would allow you better handling of large language models. And large language models now are one example.

Usually, as an organization, you would have just some niche – some weird, weird stuff that's not default, and therefore not as easily and nicely supported. Or another thing is – which we are specifically investing a lot of time and energy in these days – improving developer experience. It's not nice for a data scientist to interact with raw Amazon SageMaker. It's a lot of overhead. You need to think about BPCs, you need to think about encryption, and these things. You should not need to think about this as a data scientist.

37:20

Alexey

**Yeah, sometimes I really question the design choices that the SageMaker team made at some point. [Simon agrees] Why would I need Lambda in front of a SageMaker endpoint? Why would I see the CSV data in my request instead of JSON? [Simon agrees] Things like that. Okay. Some things look pretty arbitrary. [Simon agrees] It's not something data scientists will use at the end, so you need to make some tooling around that to make it easier to use.**

37:51

Simon

Yeah, that just takes away the unnecessary complexity and introduces some opinionated things. For example, if you want data scientists to use a specific KMS key to encrypt their data at rest, this is something that you would abstract away completely in a thin layer on top of SageMaker. Then data scientists don't need to worry. They just can be sure the data will be appropriately encrypted.

38:20

Alexey

**You said “thin layer,” how thin is this layer? Is that something one developer can do in one week? Or is it something that you need a team to work on for half a year?**

38:31

Simon

Specifically the example that I gave now is something that...

38:34

Alexey

**Just in general – around an existing platform.**

38:40

Simon

I believe that the layers around an existing platform should be as thin as possible. It always depends on what you really want to achieve. There is one side where you basically say, “I want my models to be built independent of whatever they are running on.” Meaning, “If I want to migrate from SageMaker to Vertex AI, for example, I do not want to have to change my models, but I'm going to change the platform piece. I'm going to change the interaction patterns with these models.”

If you want to achieve that, then your platform will naturally need to be a tad thicker, compared to when you just say, “Well, I trust that we are going to stay on SageMaker for many, many years to come. What I want to do is improve the developer experience.” Then a fairly thin layer – something that is really a matter of months to develop – can be sufficient.

39:41

Alexey

**A matter of months, so it's still not like you buy the SageMaker platform and you're good to go. You still need to put in some effort for it to be usable.**

39:54

Simon

I would definitely say so. Again, it always depends on your company. I might also be quite biased because currently, I'm in the Fintech space and when it comes to financial stuff, there are a lot of regulations. There are fairly strong (for good reason) requirements and everything that's security-, compliance-, auditability-related – of course, that raises the bar significantly. If you deal with, let's say, some IoT-generated data (machine data) where, if you lose that data, you've just lost the data, but no person is affected whatsoever, then you might have a lot fewer restrictions – a lot less requirements on many things. That will naturally translate to different choices when building your platform.

# Keeping data governance in mind when building a platform

40:43

Alexey

**You also worked in algorithmic advertising, right? [Simon agrees] And you also worked in e-commerce. [Simon agrees] I guess the requirements there are less strict compared to Fintech?**

40:57

Simon

Depends on the use case as well. Even in e-commerce, you can have quite sensitive use cases. Think of fraud detection. Customer data. Fraud detection – if you detect a case of fraud, and you ban the person from your platform, well, you need to be able to show why this happened. You basically need to be auditable. That means you need to show for a certain period of time exactly why this decision was made and what happened.

There are certain requirements, typically, around being able to explain your model and being able to ensure your model is not biased and that it's fair. That can even mean e-commerce – with sensitive use cases, even there, that's going to be more challenging. However, if this is a fraction of your use cases, (which it probably is in e-commerce) then you will probably not build your platform for that. You will build your platform for 90% of use cases.

44:56

Alexey

**So what to do with these cases – with data governance, with security, with auditability? I don't know how much SageMaker offers in this disregard. I guess you still need (especially for a bank) to build something on your own, right? Or are there actually tools that you can just take and adapt to your use case?**

42:48

Simon

Yeah. I think, especially for SageMaker, it definitely makes a lot of things easier. Just thinking of emitting and storing of metadata – what specific image your job used, what data (what inputs) it consumed, what outputs it wrote. It makes the tracking of these things, storing it persistently as well, and connecting your metadata over various pipeline runs – it makes it fairly easy. There's still some glue code to be written if you want to be able to, let's say, visualize that or have it nicely connected in a kind of data model. You will then still need to do a lot of glue work. Also, when it comes to reproducibility, actually. Let's say you want to reproduce the results of a model that you ran three years ago. In theory, of course, your model is stored in the model registry and it's going to stay there for a couple of years if you don't delete it. But what about all the other stuff?

43:20

Alexey

**The code is there and you know exactly which version of the code will be used. [Simon agrees] You can go back in time and... [chuckles]**

43:28

Simon

Exactly. Which version of the data was used three years ago. Right. Yes, SageMaker does help you for some things. But you need to think this process through end-to-end and make sure that this works. And that's what SageMaker doesn't do for you.

43:42

Alexey

**What actually is data governance when we talk about this specific case – when we're talking about ML platforms? Because data governance is a very large topic and we already had a couple of podcast episodes about that. When it comes to an ML platform, is there any specific part of the data governance framework that is most important for platform engineers?**

44:05

Simon

That's a good question. Usually, when you look at MLOps tooling, the first touchpoint you typically have with consuming data from a data warehouse (data lakehouse or whatever) is that tools – such as Weights & Biases, Neptune, Comet ML – usually have some sort of data tracking functionality included. What that means is actually fairly different, depending on the tool. Some tools really focus on logging, and storing metadata around your query, for example – what was the query you used to fetch data?

Whereas some other tools go even beyond that and say, “Well, basically you log the entire artifact.” By artifacts, I mean the data you actually used. That means basically, you would say, “Well, I want to store that artifact – all the data that I consumed for a specific model run, I want to store it in some other cloud storage – cloud storage provisioned by that vendor, for example, or cloud storage that I have (some S3 bucket, for example).” That's typically where it starts. There are already different approaches emerging in how you actually keep track of your data.

45:24

Alexey

**I think if you use MLflow, then you kinda need to arrive at the need of storing the data yourself. So you would put the data to S3, and then maybe you would keep a pointer to this data. [Simon agrees] While in Weights & Biases (which I think has this feature, I'm not sure about others), you can just log the entire... Yeah, all of them have that. You can just say, “Okay, this is the data. Keep it.” You don't need to implement this yourself. [Simon agrees] That's pretty cool.**

45:50

Simon

Exactly. Yeah. While it's pretty cool, if you're dealing with smaller datasets – completely fine, right? You're just going to copy that, I don't know, 10, 15, 100 megabyte dataset. Well, if your models run on tens or hundreds of gigabytes of data, this actually becomes difficult to use. And not only difficult to use, because obviously, it's a cost. Especially if you're using some proprietary storage of that vendor – if you don't want to upload like 50 gigabytes of data every time you train your model. That hurts. But not only that, but also managing that data appropriately becomes a challenge.

46:31

Alexey

**It could be personal data, right?**

46:32

Simon

Exactly. What if a GDPR request comes and that you need to delete a specific person from your data? Well, good luck doing this. If you log all your data every time you train a model, it's gonna be extremely hard to find that person and reliably delete it. So you really need to think about, “How do you manage your data to be compliant with certain regulations?” as well, especially if you do things like logging (duplicating, basically) your dataset every time you run your model?

# What comes first – the model or the platform?

47:08

Alexey

**We have a couple of questions. The first question is kind of a chicken and egg problem. The question is, “Sometimes I encounter problems when trying to build a whole pipeline from scratch with no models built yet. How do you deal with drafting the infrastructure?” Also, maybe this is a continuation, “Do you even need to think about building the platform before building the model? What should come first?”**

47:35

Simon

Yeah. Good question. I believe that always, especially if you want to do this in a profit-oriented organization, there needs to be a business case first. I think there are hardly any organizations that would say “Yes, please build that platform because at some point we're going to build some models.” It's typically very hard. It's beautiful if you can do this, because it's a full green field project and it's going to be a lot of fun. But it's very hard to argue, usually. Usually how it goes is – you would have a set of models already running, generating some kind of business value, and then you would look at, “What would it have saved us, on the one hand, if we had a platform?” And you want to project (look a bit into the future) and think, “How many models do we actually expect to have in a year (in two years, in five years)? And what will this mean if we don't build a platform? Are we actually scalable in our efforts?” That's usually how you would start thinking about the platform. But usually, the models come first. Otherwise, it's going to be difficult to argue. And also to build, actually.

48:45

Alexey

**It's the business case, right? [Simon agrees] Can we do this in parallel? For example, we just started the data science initiative in our company, we know that we will have a lot of use cases in the future, and there is one business case that we selected as the most promising one. Do we need to maybe try to start building the platform in parallel to the case? Or first, develop the case, develop the model, and see how to deploy (to start building) the platform for this specific case?**

49:19

Simon

I think there are some pieces of a platform that already make a lot of sense with one model. I mentioned it before – an experiment tracker is a classic thing. That is something, I think, that is going to pay off no matter what size you are**.** Even for a single model or for a single team that is definitely something that I believe you should consider, at least if you have a couple of data scientists. That will make sense. There are other parts of the platform where it's going to be very difficult to build the platform in a targeted way if you do not have a good picture of what it should cater to.

It's basically trying to build a product but you don't really know your customer yet. Or your customer doesn't even know himself yet – he doesn't even know, “What am I gonna want? I know that now I need to open this store. But tomorrow maybe I need to close it again. I don't know yet.” So usually, you would want to have at least a user, a customer, or a customer base, that kind of knows which use cases are coming up so you can build an architecture around it. If you don't have anything, then it's a lot about guessing (estimating) what's going to come.

It can work, but you might be building things that are really just not going to bring that much value to your customer or they're just never going to be used. I think every person who builds a platform has experienced this – thinking way too far ahead, and you're building something that's going to maybe be used two years or three years down the line.

50:57

Alexey

**So the summary here would be to wait with heavily investing into the platform, before you have at least a handful of use cases. Right? Then you see what's common in these use cases and how you can abstract away some stuff on there to the platform?**

51:15

Simon

Absolutely, I think that's very, very well summarized. Naturally, it does not mean that you should not build abstractions. If it makes your life easier as a team – if you build some abstraction on top of SageMaker – well, do it. If it makes the lives of three data scientists easier, do it. You may call it a platform, and it may be just the starting point, actually, of something bigger.

# Do MLOps engineers need to have deep knowledge of how models work?

51:41

Alexey

**Thank you. Another question. “It seems that MLOps is biased more towards software engineering. Do we still need to invest time in learning state-of-the-art models? Or we just take whatever is there, such as what Hugging Face or another framework offers and not bother with learning SOTA?”**

55:07

Simon

Ooh, that's a good question. If I understand it correctly, what you're saying is that there are quite some Hugging Face as a model platform, and also a kind of a vendor (a service provider) that makes your life fairly easy already. So the question, as far as I understand is, “Do we even need to worry that much about MLOps itself if there are some...”

55:31

Alexey

**Maybe the question is, “Do we need to worry about learning about these models – the internals of these models?” Or we can just take whatever Hugging Face offers and, as a platform team, maybe we don't even need to worry about it. But maybe as a larger organization, some teams might want to learn more about state-of-the-art things. How does it affect our office platform engineers?**

55:57

Simon

I think it's definitely a good point, I think for an ML platform engineer (for an MLOps person who builds a platform), it's not really going to matter whether... Often, it's not going to matter which exact type of model you want to run. However, there are definitely cases where this matters. Again, I'm going to fall back to the example of large language models. If your models reach an extent, or are in a way, that place specific requirements on your platform, on your infrastructure, for example, or on your deployment flow, or on your evaluation flow – it's especially interesting for large language models, again – then you, as a platform engineer, definitely should think about these aspects. You shouldn't necessarily think about, “How does this model work exactly?” But really, “Would this model run on my platform? Or why would it not?” That will help you evolve your platform into directions that make it potentially future-proof, if these use cases will become relevant for your organization.

# Is API design important for MLOps?

54:15

Alexey

**Thank you. Another question, “How important is API design for MLOps?”**

54:22

Simon

Well, API design is... It depends a bit, again, on what you want to run. From an ML platform perspective. It depends a bit on if you want to abstract that away. What's definitely important is for the team who wants to deploy this model, is that it needs to serve predictions to a set of consumers, then you, as that team, need to think about what that API should look like and how you would evolve it over time as well. From a platform perspective? There, it depends a bit.

I would say it's not something that you typically care about that much as a platform – as a producer of data and teams that build and deploy and serve models. They are producers of data. They should worry about, “How do I make and keep this consumable?” As a platform, I would think about, “How do I make it easy to deploy it and serve it via an API?” But what that API specifically looks at is typically not something that the platform engineer would probably look into.

55:29

Alexey

**For me, at some point, it became important. The case was when we wanted to log predictions in such a way that it's unified across all the use cases. For example, imagine that you have a model for churn prediction. You send a request to the platform, the platform replies with the prediction, and you save these predictions, you save the incoming request, and you have saved the response in, let's say, some database (in some log storage). You want to run some analytics on top of that.**

**In order to do that, you want the logs to be unified across all the use cases. So the churn prediction use case, the lead scoring use case – all the other use cases should follow a similar schema in order for you to be able to analyze this later and maybe do some monitoring, some analytics, etc. But yeah, this is maybe a specific one. You don't always think about this until you need to do this. But sometimes, yeah, it's important.**

56:33

Simon

I think it's a good point. I would imagine in that role you were part of the team that was building these models. Yes?

56:43

Alexey

**Yeah. I was kind of a part of all the teams. My role was to overlook the entire process. I connect the platform team and the users of the platform.**

57:03

Simon

Okay, yeah. Makes a lot of sense, especially in that position, I think – is such an interface position.

57:12

Alexey

**Okay. Well, this could be wrapping up. Maybe the last question for you is – we talked about building a platform, the skills we need, and I know that you have written a lot of stuff. Actually, the questions we prepared were questions about stuff you wrote, but we never had a chance to get to them**

57:28

Simon

Never touched upon it. Too many things to talk about.

# Simon's recommendations for furthering MLOps knowledge

57:32

Alexey

**So maybe you can recommend some further reading, if people want to learn more. It could be from you or from other people. Maybe there are already books about this stuff, maybe there are good courses, or good videos, good talks about this topic.**

57:49

Simon

Yeah. Well, of course, as you said, there are some good books about MLOps and machine learning engineering. What's important to notice is that MLOps is a term that, I think, is not yet not very well defined. For example, when I speak about MLOps, it's usually from a platform perspective – I'm thinking about ML platforms. For other people, it's very different. For them, machine learning operations is basically everything from deployment onward. That's also a bit of the case with the books. There are some books, for example, Designing Machine Learning Systems is a very popular one. It's also a very good one. But it does not take very much from the platform perspective, actually. It's really more around building that model. Nevertheless, it's a great book.

I also particularly like Practical MLOps, also a book by Noah Gift and his co-author (Alfredo Deza). What I liked about that one is that it's, like the name says, actually very practical. That also means that it's really going to show you how to build things on the three big cloud providers. That, I believe, is a great, great asset to have. That also brings me to my overall recommendation – books are great, but putting things to practice – building your pet projects – is what I really recommend.

I think that's also why I love DataTalks.Club (the MLOps Zoomcamp, specifically) because that's what you guys are doing and that's really where you learn. That's also not only where you learn, but also how you can showcase your skills to a future employer, for example. So it brings together a lot of these things. Don't get too stuck in books – start building. That's what I would recommend. But that's also the type of learner that I am.

59:39

Alexey

**Yeah, thank you, Simon. Thanks a lot, everyone, for joining us today. Thanks, Simon, for joining us today too, and sharing all your expertise. That's all we have for now. Enjoy the rest of your day and the rest of the week. See you soon.**

59:55

Simon

Thanks a lot.

59:56

Alexey

**Bye. Now I will need to go outside and it's so hot. I hope my brain does not melt. Hopefully we will still be able to talk after.**