EPISODE 527

[INTRODUCTION]

[0:00:00.3] JM: Apache Spark is a system for processing large data sets in parallel. The core abstraction of Spark is the RDD, the Resilient Distributed Data set, which is a working set of data that sits in memory for fast iterative processing.

Matei Zaharia created Spark with two goals; to provide a composable, high-level set of APIs for performing distributed processing and to provide a unified engine for running complete apps. High-level APIs like Spark SQL and MLlib enable developers to build ambitious applications quickly.

A developer using Spark SQL can work interactively with a huge data set, which is a significant improvement on batch hive jobs running on Hadoop. A developer training a machine learning model can put the model through multiple steps in the training process without checkpointing the data to disk.

The second goal of Spark, a unified engine for running complete apps was the focus of my conversation with today's guest, Matei Zaharia. Matei is the CTO of Databricks, a company that was started to implement his vision for Spark and to build highly usable products on top of the Spark technology.

Databricks Delta is a project that combines a data warehouse, data lake and streaming system, all sitting on top of Amazon S3 and using Spark for processing. In our recent episodes about streaming, we explored some common streaming architectures. A large volume of data comes into the system and is stored in something like Apache Kafka.

Backend microservices and distributed streaming frameworks read that data and store it in databases and data lakes. A data warehouse allows for fast access to the large volumes of data, so that machine learning systems and business analyst can work with those data sets interactively.

The goal of Databricks Delta is to condense the streaming system, the data lake and the data warehouse into a single system that is easy to use. If you listen to the previous episodes, you will have an idea for the level of complexity that's involved in managing these as different systems.

For some companies that does make complete sense to manage your Kafka cluster, your Spark cluster, set of S3 buckets, a data warehouse like Amazon redshift, but we'd probably don't want all of that management to be the lowest barrier of entry. Databricks Delta will hopefully reduce that barrier of entry and make it easier for enterprises to set up large systems for processing their data.

A few quick notes before we get started, we just launched the software daily job board. To check it out, go to softwaredaily.com/jobs, where you can post jobs, you can apply to jobs. It's all free. If you're looking to hire, or if you're looking for a job yourself, I recommend checking it out. If you're looking for an internship, you can certainly use the job board to apply for an internship at Software Engineering Daily.

Also, meetups for Software Engineering Daily are being planned. Go to softwareengineeringdaily.com/meetup if you want to register for an upcoming meetup. In March, I'll be visiting Datadog in New York and HubSpot in Boston. In April, I'll be at TeleSign in LA. I look forward to seeing you there. I hope you like this episode with Matei Zaharia.

[SPONSOR MESSAGE]

[0:03:32.3] JM: Apps today are built on a wide range of back ends, from traditional databases like PostgreSQL to MongoDB and Elasticsearch, to file systems like S3. When it comes to analytics, the diversity and scale of these formats makes delivering data science and BI workloads very challenging. Building data pipelines seems like a never-ending job, as each new analytical tool requires designing from scratch.

There's a new open source project called Dremio that is designed to simplify analytics on all these sources. It's also designed to handle some of the hard work, like scaling performance of analytical jobs. Dremio is the team behind Apache Arrow, a new standard for end-memory columnar data analytics.

Arrow has been adapted across dozens of projects, like Pandas, to improve the improve the performance of analytical workloads on CPUs and GPUs. It's free and open source. It's designed for everyone from your laptop, to clusters of over 1,000 nodes.

Check out Dremio today at dremio.com/sedaily. Dremio solved hard engineering problems to build their platform, and you can hear about how it works under the hood by checking out our interviews with Dremio's CTO Jacques Nadeau, as well as the CEO Tomer Shiran. At dremio.com/sedaily you can find all the necessary resources to get started with Dremio for free.

I'm really excited about Dremio. The shows we did about it were really technical and really interesting. If you like those episodes, or you like Dremio itself, be sure to tweet @dremiohq and let them know you heard about it from Software Engineering Daily.

Thanks again to Dremio and check it out at dremio.com/sedaily to learn more.

[INTERVIEW]

[0:05:33.3] JM: Matei Zaharia, you're CTO at Databricks and the creator of Apache Spark. Thanks for making the time to come on Software Engineering Daily.

[0:05:40.1] MZ: Yeah. Thanks for having me.

[0:05:41.3] JM: We're talking about Spark and some of the new developments in the Spark ecosystem today. I want to start by just giving people a quick overview of what Spark is if they're not familiar with it, or if they use it but they're not really sure how it works under the hood. There is a variety of systems that we've seen for processing large data sets and what set Spark apart was the idea of the in-memory working set. Explain why the abstraction of the working set, the RDD was important.

[0:06:12.6] MZ: Yeah, for sure. Basically, when we started the Spark project, we were seeing many different distributed programming models for clusters including MapReduce, different programming models for graphs, such as Apache Giraffe, different programming models for streaming and so on.

We realized that inside a large application, people will want to basically combine different computations. To do so efficiently and with these different systems, there wasn't the concept of storing data in an efficient way to feed it into the next computation. You could run, for example a MapReduce job, but then the output of that would always go to a distributed file system.

Then maybe you would add another MapReduce job under result, or maybe you would run something like Giraffe or graph computation or something like that. We figured basically data sharing is essential if you want to compose software and every developer becomes productive by being able to compose lots of little pieces of software. We wanted to make that efficient.

In Spark, we added built-in concept of keeping data sets in memory and a $5 in fashion and sharing them efficiently across computations. We were able to support these end-to-end workflows.

[0:07:32.3] JM: You also had a novel approach to resilience of these distributed data sets, the idea that instead of backing up a data set by checkpointing to disk, your backup was more implicit, because you knew that you could always recreate an intermediate data set in the event of a failure by rerunning some previous operations. Why was this a novel recovery mode?

[0:08:00.4] MZ: Yeah. Compared to what people were doing in these other systems, it's basically just a more efficient way to do fault recovery. It only works when you have deterministic computations that you can rerun, but in these programming models you were asking people to have deterministic computations anyway in order to do fault recovery, even within a single job.

Basically, instead of using something like a distributed file system where it doesn't know anything about how the data was computed then it just has to replicate it and save it to disk in order to make it reliable. In Spark, we actually track the computation that went into each piece of data. If we lose that, we're able to redo the computation just on that part of the input.

Then the other aspect that's pretty cool is because it's a parallel computation, if a node fails we can split up the work of recovering it across a 100 nodes and basically recover a 100 times faster than it could've taken that one – it would've taken to bring up one new node. Basically, you can recover quite quickly even if a small fraction of your nodes fail and you don't pay that extra cost of replicating data and storing it to disk.

[0:09:16.0] JM: This core idea of the RDD that provided this working set, as well as this new durability model, this has led to an ecosystem around Spark. The last time that you and I spoke was two and a half years ago, how has the ecosystem evolved in that time?

[0:09:34.7] MZ: Yeah. Quite a bit has happened since then. It's still a very active open source community of lots of different individuals are contributing to it. Probably some of the biggest changes and some of the libraries on top of Spark and some of these are included in Apache Spark as built-in libraries, but there are also many third-party external packages that work with that.

I think Spark SQL, the structure data library was just starting out back then and same with the data frame API on top of that, which is very popular now. It's a programmatic way to work with structured data. It's basically similar to building up SQL ways, but it's much easier to organize into large programs, and it enables the engine to do a lot of optimization underneath.

That's one of the major things that's come out now. Basically, majority of users are using that API in some form. The other big thing that started a little over a year ago is structured streaming, which is higher level streaming API, based again on structured data where we can understand then the data types and do optimization similar to what a database would do.

That's now each sort of general availability and it's a much easier way to write streaming computations. Basically, you don't need to think about streaming operators. You just write a batch computation and the engine will automatically turn it into an incremental computation to give you the same results. It will do all the job of converting it to the streaming operators for you.

[0:11:07.0] JM: I think something that has been really valuable for this ecosystem is that you've consistently provided a clear and concise vision for what things are going to look like in the near future, as well as what this is going to enable in the long-term future. You have both this clear pragmatic approach, but you also have an aspirational approach to the project, and you articulate this in your two core ideas that having this composable high-level APIs like Spark SQL, or MLlib for machine learning, these that should be very easy to use, but under the hood they're doing things that are revolutionary.

Then also this idea of building a unified engine for running complete apps. I think the set of APIs, that's a little easier to understand. Okay, you want to do distributed SQL across your RDD and that's very easy to understand. What do you mean by the unified engine? This is the other part of your vision for what Spark should become? What do you mean by the unified engine? What kind of apps could people build on the unified engine of Spark?

[0:12:17.3] MZ: Yeah. By that, we mean just an end-to-end engine or framework for distributed data analytics applications. Anything where you taken data in either in a static data set, or in a stream and process it in parallel and run different algorithms on it, then serve basically provide results to some other system.

The reason to look, or to try to design a unified framework for that is pretty simple. It's because all the real-world applications that we see have to combine many different types of processing algorithms and libraries. If you have a unified engine that understands all of them, you make it easier to build, you make it easier to manage and operate when someone has to put it in production. You also enable optimization across all these things to happen automatically.

It's the same reason why people really like using basically unified framework, such as Jango, or Ruby on Rails for web developer. It's just nice to have a one-stop shop where all the pieces fit together and you know that they're going to work together. In some ways, it's even more important in this data analytics space, because if you don't have unified tools, you often lose out a lot in performance or in ease of operation, because now you have to learn how to setup and manage two distributed and fast data between them efficiently. That's basically the goal.

[0:13:45.6] JM: Is there any analog, historical analog to that that you're inspired by, or is it more like you would like to have the first unified engine? Because I can't really think of anything that's like a Ruby on Rails for distributed computation.

[0:14:00.6] MZ: Yeah, it's a good question. I think because distributed computation, especially for a kind of this ad hoc data analytics has put in you, I think people haven't built something like that yet. This was probably one of the most interesting things we did and one of the most surprising at the time is that there were these dozens of different cluster programming models that people were proposing for different applications.

Even large companies like Google or Microsoft were already using these different frameworks internally, and we showed that you can actually get very good support for all these workloads across what's basically a pretty simple engine. It's basically MapReduce, plus these RDDs for sharing data.

I think, even though maybe it hadn't happen for this distributed computing, it did certainly happen for other types of computing that people did in the past.. You got this end-to-end application frameworks and it's continuing to happen. Today you have it for example, for client side Java script stuff, you have really powerful frameworks for that, for mobile applications and so on.

[0:15:06.8] JM: It's also interesting to note what happened with Kubernetes, where you saw a framework for flexible distributed systems orchestration and scheduling. They just focused on that layer specifically. By doing that, they gave birth to an entire ecosystem of providers that build higher level interfaces on top of that. You've got Rancher and Platform9 and OpenShift and a ton of other providers providing managed Kubernetes. I could certainly see something like that happening with Spark, or with – a collection of different big data models.

[0:15:48.2] MZ: Yeah. We all heard and see a lot of Spark packages. Basically third-party packages that you can plug into Spark, including things that interact with say the SQL engine, or the optimizer. For example, data sources that understand how to push part of the query into the data source, or machine learning operators that you can plug into a pipeline with other ones.

Certainly, within just the world of open source packages you can use, there are quite a few that you can combine in this way.

[0:16:17.2] JM: We've done some recent shows about streaming systems and I want to discuss streaming with you. A typical architecture for streaming that I've heard from people I talk to is you have lots of events that are being created by a web application, or a Fitbit, or anything. Those events get put on to stream, that's maybe Kafka or – They get buffered to a durable system like Kafka or Kinesis that can hold the stream abstraction. Then operations are performed along those streams using a stream processing system like Spark, or Flink, or Kafka streams. Is that consistent with the typical types of streaming application architectures that you're seeing?

[0:17:01.9] MZ: Yeah. This is definitely one common thing you see. I mean, I think the more interesting part at least to me is what happens after you've got the event stored reliably in a message bus? What are you actually trying to compute on top of them, and how do you make that really easy to build?

In Spark, that's one of the main things where we're trying to simplify with structured streaming. For example, a lot of people taken these events, they put them in a message bus like Kafka, or Kinesis for durability, but really what they want at the end is something they can query efficiently, some kind of historical data store, such as a data warehouse, or a Parquet file for – in some kind of data link for querying using Spark SQL or Hive.

One of the things that we try to do is make it really easy to get that reliable pipeline from the events coming in to whatever transformations you have to do to an efficient kind of index, or partition format.

Another example is people trying to update a machine learning model, so what they care about that is how do I publish a model and publish metrics and monitor it periodically? Basically, I think just this stream-to-stream, like let's take one Kafka stream and other map function and produce another one, is one step you can have towards that, but it's not usually the complete application. I think the real challenge is getting this end-to-end application to work that actually talks to external systems beyond that.

[SPONSOR MESSAGE]

[0:18:45.8] JM: I listen to a lot of podcast about technical content and the Google Cloud Platform Podcast is one of my favorites. The GCP podcast covers the technologies that Google Cloud is building, through interviews with the people building them. These are often unique Google Cloud services like BigQuery, AutoML and Firebase. I'm a big Firebase user, so I try to learn about how it works under the hood and I want to hear about new features that they're releasing.

I also listen to the GCP podcast prepare for episodes of Software Engineering Daily, because when I do shows about Google Cloud technologies, I'm doing research around them and I find that the GCP podcast covers topics before I do.

If you want to stay on the leading edge of what is being released at Google and how these technologies are built, check out gcppodcast.com. I've been a listener for a few years now and the content is consistently good. A few of my favorite recent episodes are the interview with Event Surf, who is one of the creators of TCP/IP. He's one of the fathers of the internet. Also the show about BigQuery was super useful. You can find those episodes and more by going to gcppodcast.com and follow GCP podcast on Twitter.

Thanks to Google Cloud Platform, the podcast for being a sponsor of Software Engineering Daily. Much appreciate it.

[INTERVIEW CONTINUED]

[0:20:19.9] JM: Well, let's break down the components of that end-to-end application that Spark can assist with. The Spark approach to streaming, structured streaming, this seeks to be simpler than the current model of separate systems. It's also different than the streaming system you've had for a while, which is Spark streaming. Can you explain how structured streaming works in contrast with Spark streaming?

[0:20:46.3] MZ: Yeah, definitely. In Spark streaming, as well as in other streaming APIs like Kafka streams, or Flink, or things like that, the user has to be explicitly use a bunch of streaming operators and set up – figure out how to run their computation in a continuous and incremental fashion.

For example, you can choose different types of windowing operators and set them up, choose different ways of managing the state and so on. Basically, the user is compiling these high-level thing they want to do, like for example maybe they just want to group and count the events by device type and time or something and they have to map it down to these streaming operators.

The way structure in streaming is different is that it tries to do that for you automatically. What you do is you give it a computation you want to run, you express it as a data frame computation on as if you had all the data in advance. Imagine you had a static data set with all the data received so far, you just say what you want to do, like aggregate stuff by device ID and the minute of that the event was generated or something like that.

Then it uses something very similar to a database query optimizer to come up with a plan of physical operators that do that, including the different forms of stateful management operators and things like checkpointing for file recovery and so on.

The main difference is the user doesn't need to know about the streaming operators and the different ways of configuring them in advance. They just need to know what query they want to express, and then the system will automatically choose them. It's basically if you know how to write a Spark job on static data, you already know how to write a job on streaming data. That's what I mean by higher level.

Then I guess, one aspect of it is basically like converting your computation into streaming operators, but the other aspect that's a bit more subtle, but is also important is basically ensuring end-to-end consistency and reliability across the input streams and the output system. Figuring out, like if I'm supposed to periodically upload my results into say MySQL, how do I know if I cached where I left off? How do I update them transactionally and so on? That's the other thing that's built into the structured streaming IO components, like the input sources and those things.

That's something that we often found was a big pain point for people, is like they'll set up a streaming job. It runs fine, and then if something crashes, they have to manually figure out what was left over and how to recover. If they want to change the computation, or if they discover a bug, they need to go back and manually clean things up. We tried to simplify that as well.

[0:23:36.7] JM: The word structured, structured streaming in contrast to just streaming, I believe that this refers to the fact that you're assuming that the objects in the stream have a structured schema. Is that right?

[0:23:49.0] MZ: That's exactly – yeah. It's the same word we use for the data frame and SQL APIs of Spark. We call them the structured APIs. It's a little bit confusing maybe in some ways, but that's the word we chose.
The main point is that they have a schema we know, but also it's a limited data model, so the type of objects you store in them, there is like integers, strings, structures, arrays, there is like a limited form of objects. As opposed to the RDD API in Spark where you are storing arbitrary Java objects.

Because the format of them is limited and is known to the engine in advance, it's possible to do more optimizations and automatic management, like you can store them in memory more efficiently, you can use like a columnar format for example and so on.

[0:24:40.8] MZ: That's again in contrast to what you just said, in contrast to –

[0:24:45.9] MZ: The RDDs.

[0:24:47.4] JM: The RDD. The Spark streaming world, where it's just a Java object and it doesn't have – does that also allow for better data sharing between different programming languages?

[0:25:00.2] MZ: It can actually. Yeah, that's a good point. You can actually mix together user-defined functions in Java and Python and so on, because that is a clear way to translate the same object into all the languages.

Actually, a lot of people do that. I think the most common thing actually is if you want to write your job in Python, but have some fast user-defined functions in Java, this is a really nice way to connect and do.

[0:25:27.0] JM: Do you use Apache Arrow for that data interchange?

[0:25:29.0] MZ: Yeah. Actually in the next release of Spark, which being voted on now, so hopefully coming out very soon, it will be using Apache on how to move data from Spark into Python and back.

[0:25:41.3] JM: I saw your benchmarks of structured streaming versus Kafka streams and Flink. You just articulated one of the – you gave an overview for how those might compare to those systems, but could you drill in a little bit further why – Or we could even start higher level, how do these different streaming systems compare to one another and how did that inform your approach to structure stream? I know you've given the overview of that. Maybe we could just drill down even further.

[0:26:12.1] MZ: Kafka streams and Flink and more of these – basically, as I mentioned before, systems where you explicitly setup some streaming operators as the user of the system and decide which ones you're going to put together and how to organize your computation. Depending on the – in Kafka streams, I believe that the data model is just java object, so each operator produces some Java objects that you can serialize them somehow into a format and pass them to the next one.

In Flink, I think it can also have these more structured data types we had to understand and see internals if you want to use that. One of the main differences is this explicit programming. The other difference though and probably the main reason that we do better in benchmarks is just the way they do job execution. Both of these systems are based on pushing around individual objects one at a time across the different operators and even across the network.

That can be good if you want to get ultra-low latency, but the overall support you get isn't super great, because you're paying all these overhead for control logic around each record of data that you're passing through. Then Kafka streams for example, the way you pass them through is between different operators is by writing to Kafka, while you're also paying the overhead of application and reliable storage on disk, which becomes even higher.

In structured streaming, we built on basically the rest of the Spark engine, which can operate on batches of objects and can do code generation underneath. You can get much higher support basically because you're doing more CPU-efficient stuff. You're also spending less time doing IO and coordination around each record.

In terms of support, we're able to do – right now we're able to do several times better than all the other engines that we tested, which is awesome to see because we didn't do any optimization specifically for streaming. This is just the same existing engine. When you use that for this, it just happens – it just gets a higher support.

One reason I think it's like it's really important to look at this is because in streaming in particular, you want it on your application 24/7. Instead of having 20 nodes writing 24/7, if you had 5 nodes, it's a significant cost-savings and it's also a significant improvement to reliability, because with four times more nodes, basically you're going to have four times more frequent failures and maintenance events and stuff like that.

We think that maximizing support is really important, at least for the kinds of big data applications that we target in these distributed engines. Basically, if you can the same thing with fewer nodes, it's usually a winner.

[0:28:56.5] JM: All right. Well, I'm glad we've covered some of the overview like topics, because I want to dive into delta, which is a different project that you're working on at Databricks. The current model of a typical organization with a lot of data involves some different systems that we've discussed already. You've got a streaming system, like Apache Kafka or Kinesis that is – or I guess I should say message bus that's maybe a little bit less ambiguous for the sake of this conversation.

[0:29:26.3] MZ: Yeah. I think that's a good point. Yeah.

[0:29:28.0] JM: You got a message bus, you've got a data lake, which might be HDFS, or S3, and you've got a data warehousing system on top. Explain the roles of these different components.

[0:29:42.2] MZ: Yeah, sure. Yeah, so basically what we noticed with all the different organizations setting up data pipelines today is they use these three different types of system. The message bus is a system that allows low-latency ingest and it also allows basically keeping track of what data has been processed and consuming in a streaming fashion.

If you setup something like Kinesis or Kafka, you can set up different consumers that each know what offsets to end inside each partition of the data. That's kind of its all. It's coordinating across applications and also just providing low-latency ingest. That's one type of system, but it's not really meant for long-term storage of lots of data. It becomes really expensive compared to other systems.

Then you've got the data lake. A good example of this is S3, or the Hadoop file system. This is something designed for high capacity, very low-cost storage. Everyone pretty much uses some kind of data lake for their long-term storage, because it's just the least expensive option. In this system, you can put in objects, you can get them out, but it doesn't really provide any notion of subscribing to a set of objects or any kind of indexing or any structure over them. It's not always efficient to do stuff with these objects, but it is very low-cost.

Then finally, you've got data warehouses and many organizations, basically they take their data from a message bus, they put it – they do a few transformations or cleaning, they put it in a data lake and then they periodically run jobs that take a subset of the data and move it into a data warehouse to enable fast interactive queries.

The main difference between a data warehouse and a data lake is that it does indexing of the data, so it stores at an efficient way. It builds other data structures that allow fast lookup and it also provides higher level structure and reliable transactions across. You can have someone updating a table, while someone else is querying and then so on.

Whereas, in the data lake and in the message bus system, it's a wild west, if different people are doing stuff at the same time, you'll see this inconsistent snapshots of the data. These are the three systems. You see they have different characteristics. Basically, a data warehouse has the best query performance, but has a pretty high cost. The data lake has the lowest cost per byte and the message bus has low latency, but it doesn't have high capacity for a long-term storage.

[0:32:18.5] JM: In this typical model of the data flowing through an organization, where did Spark fit for the last two and a half years?

[0:32:29.2] MZ: Yeah. Spark itself is primarily focusing on the computation part. It doesn't really store data a long-term. I mean, it will store data short-term within a computation as we discuss. Spark can be used for several things. It can be used for the streaming jobs that take in events maybe from a message bus and then transform them and either they immediately publish a result or they load the data into a data lake, it can also be used for the jobs on top of the data lake itself, if you want to compute something on it directly, or if you want to take a subset of the data and push it into a data warehouse.
Using Spark SQL and using some of the more efficient storage formats, such as Apache Parquet, you can also begin using Spark as a data warehouse on top of the data lake data. Because you can store your data in one of these more efficient formats and then have faster queries. It still lacks full on indexing and it still lacks a reliable way to do transactions, because the data lake is basically a file system, or a key value store where anyone could be changing the data underneath you at any time.

[0:33:39.4] JM: For some companies, it's going to make sense to have these different components. If I am Netflix, then I've got to do proficiency in technical infrastructure that it's probably okay for me to have a data lake and a queueing system and data warehouse and I want to tweak these things and experiment with them and have duplication of different components in some places.

For many organizations, this is going to be way too complex. We don't have the engineering resources for this. What do you see is the tradeoffs between having these three different components separated versus having them consolidated with some simpler API, some simpler ways of interacting with them and getting data through them.

[0:34:27.1] MZ: Yeah. I think in general, if you want very high degree of control over your data management and storage, you may want to use these many different specialized systems. Really, their issue that so far there wasn't an obvious way to simplify these architectures and to consolidate them, because each of these systems had its own tradeoffs.

If you just like the data lake for its cost-efficiency, you couldn't get reliable transactions across it and you couldn't get high performance indexing and this ability for consumers to be notified when new data arrives. You had to combine it with these other systems if you wanted those performance characteristics.

I think many organizations if they had a way to simplify this, they would use that. One of the fundamental issues with having more of these systems is also added in – be on adding complexity, which maybe you can manage through a lot of engineering. It also adds delays and latency in terms of moving data between one and the other. Anything you can do that it moves on over these steps, will let you actually get access to the latest data faster and show it to people that actually writing applications on it.
I think if possible, I think people would love a system that combines the best of these, but of course you have to make sure that it actually delivers that for the right applications and that it actually gets the acceptable performer and same cost and reliability that you expect.

[0:35:56.8] JM: Which gets us to Delta, which you're working on. This project Databricks Delta, this is a combination of a data streaming system, a data lake – I'm sorry, I should say message bus. Message bus, a data lake and a data warehouse. Explain how Delta works.

[0:36:15.2] MZ: Yeah, great. Delta is essentially the first data management or storage product that Databricks has built. It's pretty exciting to get that released. Initially, Databricks has primarily been focused on computation using Apache Spark. Our model was hey, use whatever storage system you want, use multiple storage systems and we'll just give you a great environment to run computations across them, which is Apache Spark plus all the tooling we have around it to make it easier to use and more performant. That was the initial model.

Then we saw that basically every customer was struggling with the same issues in terms of setting up a data pipeline, which is how do I reliably move stuff between a message bus, data lake, data warehouse? Some customers had just two of those things, but it's still the same issue. That turned out to be basically a major point of friction for everyone, even before they begin to use Databricks.

We even had this issue internally with our own data pipeline. We wrote the data pipeline several times in order to make it really full proof and reliable to get data from the different events, or as we monitor into formats that we can then actually use to make decisions in the company.

After seeing that for a while, we asked ourselves, we already saw people were excited to use S3 and then data formats like Parquet as a data warehouse as well. They were saying, "Well, look. If I can just have my data in S3 and have acceptable performance for those data warehouse queries, then I don't need to setup a separate system and populate it and make sure that's reliable.

We decided to embrace this and also extended to supporting the message bus component. Basically what Delta is is it's a data management system that uses S3 underneath for their storage. Delta itself, it tracks a set of tables in S3 and it also tracks input streams including being able to let people listen to each stream and being able to load data incrementally and notify these consumers and so on.

In contrast to just using S3 by itself, it adds the concept of transactions and of indexing in top of it. As data flows into a Delta table, you can have transactions, you can make sure that consumers don't see the data until it's all in there and it's ready to use. You can also have indexing. Over time, for example you can compact files into a single big file, you can sort them, you can partition them, you can build different types of indexes on them and you can start getting similar performance to what you would get in a data warehouse.

It's a really cool design overall, because basically you just have the brains of a data warehouse and of a message bus, which is just keeping track of this meta data, but for the actual storage we use S3. You get this super low-cost storage arbitrary scale, whatever scale Amazon S3 provides, or Azure block store or whichever storage system you're using. At the same time, you get all the control logic of these other systems.

[0:39:28.8] JM: Not to mention, you don't have to worry about running an HDFS cluster.

[0:39:32.9] MZ: Yeah, exactly. I think if you're in one of the public clouds, you could already – A lot of users they don't run HDFS anyway because they just use S3 or a block storage. Yeah, definitely you don't have to manage it yourself. We think it's pretty exciting. Basically maybe the surprising thing about it is that you can get very good performance for both the streaming side and the warehousing side, the queries.

Once you do that it's really attractive to just use S3 underneath. It's probably going to have higher availability than any other system you would stand up on your own. Unless, you invest a lot of work into that. It's always up. You can scale up, you can read thousands of streams at a time from it if you want. Basically, you don't have to worry about the long-term storage.

[SPONSOR MESSAGE]

[0:40:33.8] JM: QCon.ai is a software conference for full-stack developers looking to uncover the real-world patterns, practices and use cases for applying artificial intelligence and machine learning in engineering.
Come to QCon.ai in San Francisco from April 9th to 11th, 2018 and see talks from companies like Instacart, Uber, Coinbase and Stripe. These companies have built and deployed state of the art machine learning models, and they've come to QCon to share their developments.

The keynote of QCon.ai is Matt Ranney, a Senior Staff Engineer at Uber ATG, which is the autonomous driving unit at Uber. He's an amazing speaker. He was on SE Daily in the past. If you want to preview for what he is like, then you can check out that episode that I did in conversation with him.

I've been to QCon three times myself and it's a fantastic conference. What I love about QCon is the high bar for quality, quality in terms of speakers, content and peer sharing, as well as the food and the general atmosphere. QCon is one of my favorite conferences. If you haven't been to a QCon before, make QCon.ai your first.

Register at qcon.ai and use promo code SE DAILY for $100 off your ticket. That's qcon.ai and you can use promo code SE DAILY for a $100 off. Thanks to QCon for being a sponsor of SE Daily. Check out Qcon.ai to see a fantastic cutting-edge conference.

[INTERVIEW CONTINUED]

[0:42:20.1] JM: Contrast this with the model of the different components. If I've got for example, this system of Fitbit. Let's say I've got a small Fitbit-like company and all the users are simultaneously reporting data about their location, let's say every 10 seconds they send – or let's make it 1 second. Every 1 second, their GPS location is being communicated to my Delta cluster or system, whatever you want to describe it as. Give me an idea for the path that a piece of data would take from the user's device into Delta, to a business analyst, or a machine learning application?

[0:43:04.5] MZ: Sure. Let's look at the simple case where you're uploading events from these devices let's say every few minutes, then you want them to get into a format that's very efficient to query. For example, your downstream application or your analyst might ask questions like, "How many events of a specific type happened for users in California in this time range?"

If you didn't have a system like Delta, you would probably have to set something up involving basically a couple of storage systems and some streaming jobs to move data between them. For example, you would receive these events into a queue and then you'd have a job that periodically reads stuff on the queue and then writes something like an Apache Parquet file, or Hive table where the data is sorted and partitioned in a way that makes that possible to do fast queries. Then you need to make sure that job is up and running and so on.

If you use Delta, you can upload these events into it basically by just putting a file into S3 and a specific location. Then over time the system, as it receives these, it will move them into a format that's partitioned and that's indexed in the right way. It will consolidate the files and so on.

When someone does a query over this table, they'll see the data that's been consolidated into an efficient form and they'll also see maybe the latest files that haven't been compacted and [inaudible 0:44:29.3] yet and the query will work across them. This will happen in a transactional way where if we're in the middle of taking a bunch of files and sorting them and producing a new Parquet file, we're not going to see the intermediate results of that.

The system knows which files are ready to use and which ones are not. Basically, what you'll get is you just upload stuff into a folder in S3 and then you run your queries on the other end and you get this fast performance without having to manually decide how to set up that job to translate them.

[0:45:03.3] JM: What's the user experience for programming against Delta? Like whether I'm a business analyst or a machine learning algorithm writer.

[0:45:13.3] MZ: Yeah. Good question. Delta is just exposed as a data source in Spark, which is similar to how you expose files in HDFS, or Hive or Apache Cassandra, or systems like that. All you have to do is when you create your data frame or your streaming job in Spark, you say the input source is Delta and you give the Delta a location basically in S3, why this table is being managed.

Then you just saw a normal Spark commands after. Same thing if you had analysts using SQL for example, you would define a table that's backed by Delta using just a SQL command, like create external table and then you would add in SQL queries over it. That's the way we're exposing it. Any kind of Spark application using the Spark APIs can read consistent snapshots of data from Delta.

[0:46:05.3] JM: If I understand this project correctly, you are consolidating the message bus with the data lake, is that right?

[0:46:13.2] MZ: Yeah. With the data warehousing bits after, with basically maintaining the data in a form that's efficient way.

[0:46:19.8] JM: Right. Okay. Yeah, because I was thinking of data warehouse as the business application on top of it, but that's not really what the data warehouse is. The data warehouse is the thing that powers the business application.

[0:46:29.9] MZ: Yeah, it's the storage part. That's what I mean by it. Yeah, I just mean the storage part of it. Yes.

[0:46:36.6] JM: Right. Okay, so one question and I know I asked you this over e-mail, but I feel like we should go over this again, because people use their message bus in more ways than just as a system of data lake ingress basically. People use their message, their Kafka bus as a way to shuttle logic between different user-facing applications for example. It's a pub-sub system where you subscribe to topics and your publish topics. I guess, my question is does Delta provide that pub-sub functionality as well?

[0:47:14.7] MZ: Yeah, that's a great question. You can use Delta to publish and subscribe to data coming into different tables. We have users that are doing this already. They're setting up these pipelines that have multiple streaming jobs and one job reduces data and then the other one consumes it.

The data source for delta when he use that in a streaming job, it supports keeping track of what data it processed so far and being notified just on new data. It can be very good for being – for analytics jobs. The one tradeoff compared to using like a pure message bus is because Delta is backed by S3, it probably has slightly higher latency than say just an in-memory message bus.

On the other hand, it is highly reliable and it is geo-replicated and stuff like that. If you wanted something with like super low millisecond latency across a bunch of applications, like say some microservices, it probably wouldn't be the right thing to use. If you wanted something where you're running a bunch of analytics applications and you need to reliably transform data into a given format and have it come out the other end, then you need to have many workers, many machines working in parallel to do this, then it's probably a good fit for doing that.

[0:48:35.3] JM: I see. What kind of feedback have you gotten from people for how they're using Delta?

[0:48:41.0] MZ: Yeah, we've seen very positive feedback so far, because again, it's driven by exactly the kind of issues that people have when they just begin using S3 and different systems around it like S3 plus Kinesis, or just S3 plus Apache Spark to do some computations. Basically, without Delta there's a lot of trouble with okay, how exactly do I manage the data in S3? How do I make it so that when one of my jobs is uploading a result, it doesn't mess up the files that are being used by another job to do interactive queries and so on.

You get all these weird consistency to deal with and all these work to make it reliable, which comes up again whenever anything breaks – if you have to update one of your jobs or something, you need to go back. Users really appreciate that. It simplifies this.

I'll give just one example of a Databricks customer that we talked about when we announced Delta, this is a very large Fortune 100 company that using Delta and Databricks and Spark for information security. They have their security team, like many other companies collects events from all kinds of devices on their networks, from operating systems, different types of intrusion detection systems and so on and it wants to collect these events and basically build some kind of warehouse where analysts can search through the logs and can run programs that automatically identify certain cases that identify possible security attacks.

Before using Delta, they had basically a large data lake based on Hadoop, where they would collect all the data, so they could store all of it. It wasn't efficient to query. Then they had a bunch of data warehouses, and the data warehouses had to be much smaller, because they're just very expensive and some of them also had scaling limits.

They could only have about two weeks' worth of data in the warehouses where the analyst can do really quick queries. Then if you ever had a query where you need to go farther back, like you discovered you say a compromised use accounted, you want to see what did they do in the past six months, you have to go to the data lake, write it in a different programming model and run this kind of slow computation.

The whole system also had a bunch of complex jobs to move data through and it actually took about six months for a team of 20 people to build the whole system, which is quite a bit of an investment, you can imagine is difficult to change.

We tried using Delta for the same use case and basically just using the built-in features of Delta for indexing and using S3 as the data lake, we were able to get comparable performance to those data warehouses for the whole range of historical data. Many multiple petabytes of data can live in S3. It can be updated reliably in real-time as new events happen. Then analyst can run the queries directly against everything and get very good performance and not have to worry about moving them into a separate warehouse and dealing with this like two-week limit.

They were really excited to see that. They were also excited, because just by using Spark they're able to run other types of computation, like different machine learning computations as well. I think it's an awesome example of where you need – you really want the massive scale and the low cost of something like S3, because in the security world you want to keep as much data as possible for as longest time as possible to identify threats.

You also need this very fast indexing and the agile environment, where potentially hundreds of engineers and analysts can each setup their own queries and their own jobs and work on a clean version of the data and track different types of events.

[0:52:32.4] JM: With the model of having the three separate components; the data warehouse, the message bus and the data lake, you've got a – the API for the ingress is probably you're writing to a topic on that message bus. What's the ingress API for Delta and how do those events make their way from the ingress point to the data warehouse?

[0:52:59.7] MZ: If you just want to use Delta by itself, you can do ingress by uploading files into S3 in a specific location. There's a nice rest API you can get from S3, or from – it also works with the Azure data storage. With either of those, you can use your rest API to upload a file. It's an API endpoint that's available across all of the cloud data centers in the world, then it's like highly available and you can just put in your data.

If you'd like, you can also ingest data into Delta using a Spark application. For example, you can have your message bus, maybe using the message bus for something else as well. Then you can have a structured streaming job that reads from Kinesis and writes into Delta. That's also supported.

[0:53:47.1] JM: I'm glad we've gone through Delta, really interesting project. I want to ask you now a little bit about some of the, I think future-leaning topics. There's two things I want to approach with you and maybe we can do them simultaneously. One is deep learning and the other is the cloud versus edge computing.

These are two really big topics. They are somewhat related, because as we have more and more deep learning workloads you can imagine more and more data is being shuttled between devices at the edge and devices in the cloud. The discussion of how data makes it from the edge to the cloud has been a repeated topic in some of the shows that I've done recently where you have all of these devices that sit in between your – for example, your smartphone, or your laptop and the cloud. Maybe you're going to have connected cars, you're going to have drones flying around outside.

I guess, you can take these two topics separately or together. How do you think deep learning and the discussion of cloud versus edge is going to affect your approach to distributed systems in the next 5 to 10 years?

[0:55:01.8] MZ: Yeah, these are great questions. Maybe I'll start by talking about deep learning and then I'll talk a little bit about cloud versus edge as well. Yeah, actually I've been working a bunch on deep learning topics, both at Databricks and also in my research at Stanford with my PhD students and collaborators there.

Basically, it's a really exciting tool to have in the machine learning toolkit, because it finally gives us a pretty reliable way to work with unstructured data such as images and audio that was way difficult to featurize and work with before.
I think there is a lot to be done to make it really usable and robust in a lot of applications. I guess, there are a few different projects in there. One of the things that we're looking at that Databricks is just higher level APIs for using deep learning. For example, if I just want to build

an image classifier, today I have to go around and Google for implementations of different models and try all of them and make sure I have tuned them correctly for my data and so on, then take the best one for my data.

Is there any way we can make that faster and just give you a simple API to trial of those and give you the best one? That's one of the things, or the use cases that we've been working on. In terms of the more research aspects, there are still many interesting limitations of deep learning. One limitation is from – basically from a performance standpoint, the more sophisticated deep learning models, the ones that get very good performance in terms of accuracy are also very computationally intensive.

There is the question of how can you efficiently run that model on lots of data, or an edge device that doesn't have a lot of computing power? One of the projects I'm working on at Stanford is actually a visual query engine that can automatically optimize inference for these models. It's called no scope, and it basically figures out when can it use a smaller model to get an answer, when does it need to use a larger model because the example or the input is difficult to classify. That's one interesting area.

The other really interesting area where I don't really have a solution, but I just think people will face is actually security, and to some extent debugging as well. It turns out that every deep learning model out there is amenable to these adversarial inputs, where you can change a few pixels in an image, or a few words and a piece of text and you can make it give a wildly incorrect answer.

It's a fascinating phenomenon. It just seems to be inherent to the nature of these models, that because they have so many parameters and they're so nonlinear, there is going to be the small inputs. I think people will have to solve this for a lot of real applications if they actually want to use deep learning there.

I think there's lots of exciting stuff. I'm personally most interested in these issues of productionizing it and making it easy to build applications. I'm less interested in just developing better deep learning models.

There are a lot of people doing that as well. It seems that we have a good understanding of how to do that, but we have less understanding of how could I actually have a production application that I can make reasonable sort of assessments of that I'm going to deploy that use as this.

[0:58:23.6] JM: Definitely. Okay, well we can save cloud versus edge for the next time perhaps. A bit off more than I could chew with that question. Okay, well Matei thanks for coming on the show. It's been really great talking to you.

[0:58:33.9] MZ: Yeah, no worries. Thanks for having me. Yeah, happy to chat more anytime if you'd like. These are really good questions.

[END OF INTERVIEW]

[0:58:42.6] JM: Today's podcast is sponsored by Datadog, a cloud scale monitoring platform for infrastructure and applications. In Datadog's new container orchestration report, Kubernetes holds a 41% share of Docker environments, a number that's rising fast.

As more companies adopt containers and turn to Kubernetes to manage those containers, they need a comprehensive monitoring platform that's built for dynamic modern infrastructure. Datadog integrates seamlessly with more than 200 technologies, including Kubernetes and Docker, so that you can monitor your entire container infrastructure in one place.

With Datadog's new live container view, you can see every container's health, resource consumption and running processes in real-time. See for yourself by starting a free trial and get a free Datadog t-shirt at softwareengineeringdaily.com/datadog. That's softwareengineeringdaily.com/datadog.

Thank you, Datadog.

[END]