BEST PRACTICES FOR ENRICHING YOUR DATA

INTRODUCTION TO THE DATABRICKS LAKEHOUSE PLATFORM

EASTERN ANALYTICS, INC.

Introduction to the Databricks Lakehouse Platform

Welcome everybody, to an **Introduction to the Databricks Lakehouse Platform**. Just a quick introduction to Eastern Analytics, for those of you who are familiar with us. We're a boutique consulting service who helps customers maximize the value of their data.

We ARE analytics people providing technology advice, solution architecture, data engineering, dashboards and visualizations, and staff augmentation. With over 25 years of experience, we have the functional knowledge and the technical skills to make sure your project is a success.

I'm Kerrilee Pietroski, Director of Communications at Eastern Analytics, and I'll be your moderator today. If you have any questions, please type them in the Q&A box in the lower right-hand corner. And if you're watching this on replay, you can always email your questions to support@easternanalytics.us.

Now, we do have an information-packed session today, so we'll try to get to as many questions as possible. But if we don't get to your question today, know that we'll be reaching out to everyone in the next couple of days to follow up. Feel free to call us as well at (781) 757-7036.

Our presenter today is Scott Pietroski, Eastern Analytics Managing Partner and Senior Solution Architect. For more than 25 years, he's been building out analytic platforms for corporations such as Bose, Adidas, Estee Lauder, and many more. He specializes in Microsoft Power BI, Azure, and Databricks. He's excited to share with you today's presentation.

Let's begin.

Hi, everybody. Welcome. I'm Scott Pietroski with Eastern Analytics. Today, we're going to talk about Databricks. We're going to do a Databricks intro. What is Databricks? We're going to talk about Databricks on Microsoft Azure. We're going to talk about the Databricks Lakehouse and Delta Lake, what they are and how they work together. We're going to talk about some easy-to-understand use cases. At the end, we'll have a Q&A.

It's important to note that today's presentation, it references projects that we've done at Eastern Analytics or people have done. Also, it references information from the Databricks partner program that we're a part of.

The Databricks Unified Data Management System. What is Databricks? In essence, Databricks is a giant toolkit to process your data. It sits on top of your data lake. What it does is it gives you the ability to ingest all these different kinds of data, structured data, unstructured data, semi structured data, and to put some structure to it, and then to be able to go and use current technologies against that. Think about writing SQL against your data, or think about the different steps you go through for data engineering and engineering your solutions, whether it's cleansing or harmonizing data. It also includes machine learning.

Introduction to the Databricks Lakehouse Platform

If we look at it as the three components, the first component is a **cloud database management system**, or think of it as SQL. It uses cheap BLOB storage, so inexpensive BLOB storage to store as much data as you need. It's designed to deal with massive amounts of data, but it also can work just typically as your regular enterprise data warehouse with normal scale to it. It doesn't have to be, it doesn't have to be big data. It supports basic SQL. When I say basic SQL, the SQL right now is maturing, so it's not as mature as, say, a T SQL where you have a whole bunch of different functions that you can call and different commands you can call. But it does have all of the basics that you need to be able to do all of your different CTEs and your select statements and things like that. It has most of the functionality of a T SQL. Lots of times actually you'll go looking for a specific function and you'll find it. Sometimes I've even found that it came out the month before. It's maturing rapidly. And what it does is, you can basically store inside of there, stuff in tables. So, information in tables, they're called Delta tables. Also, you can store stuff in views. And very importantly, it supports asset transaction capabilities.

So, think of those as one of the problems over the years has been when you are reading and writing in a file-based system, you'd end up with conflicts or locks. You had no way of making sure that information was completely written into a file before the process failed. With Delta tables, what they did was they gave you the ability, basically, they came up with a way to make sure that things work like you think of in a database with actual commits occurring, with change logs taking place, logging on those tables, and the ability to roll things back if something is bad, all in a file-based system.

So, if we take a look at the SQL portion of it, this is just a screenshot of the SQL panel here. Everything that people are familiar with. So, if you're thinking of SSMS, you get basically your tables over in the left. In this case, I selected a table with the different fields. You've got your SQL panel. Your SQL panel, you can write what you would usually write in SQL, and of course, your results panel down below. Now, we've got stuff information that's stored in workspaces. We can save queries off. This dashboarding piece actually has to do with presenting data returned by a query. Now, if you're working in a data science environment where you're grinding through large amounts of data and you don't want to have a front end on it, you can use the dashboards piece here. If we look at the data piece, the data piece actually goes down and you can look at the information inside of a table. So, more information about a specific table, think of it that way. And the Secret Warehouse, if you think of this similar to Snowflake, so this is really your computes that you can connect to from the outside world to Databricks, like a Power BI.

If we take another view here, this view here is a view of a table itself and then the information about the table. So, we have the schema here. You can view data for it. You can view the details. If it's a view, you can see the SQL that makes up the view. You can look at your grants and you can look at additional history and lineage. And then Insights is something that's new that has to do with looking at access, who's been accessing it, what queries, what notebooks have been accessing your table.

Introduction to the Databricks Lakehouse Platform

It's also, along with the SQL capabilities, which is the database system, it's designed around **data engineering.** When we look at it from a mature standpoint, we say, 'Well, this GUI really isn't great. It does what it needs to. It's functional, but it doesn't have a lot of bells and whistles to it,' and that's because it really is designed to be able to do anything. Because it's designed to do anything in the data engineering space, all of the work is done inside of notebooks, so you code everything. It's designed to support streaming and batch ingestion. It has change data capture, for if you're streaming through or dealing with huge quantities of data. It allows for massive scaling on the compute side. Think of it as any cloud application where basically you're spinning up your computes and sizing them or scaling them appropriately. Then it also supports multiple languages. It supports Python, Scala, R, and SQL. The majority of the different commands, like the SQL commands, or the majority of the commands are available in all the different languages. It's very flexible based upon your skill set that your team already has.

This is just a quick look at opening up the data engineering tab here. The workspace stores all of your notebooks and your artifacts for working with. Repose is a connection to Git. So, you can do your pushes and pulls. Basically, you have your recent items you've worked on. You have access to your tables here in data, in your compute. And workflows inside of Databricks, a workflow is basically a pipeline. This is just an example of one of the commands inside of a workbook. You can see here it's written in Python, it's using Spark. It's written in Python and it's returning a data frame. It's your typical programming for a Python approach.

From a **data science, ML plat**form. It's important that... This includes Auto ML. Most people are familiar with Auto ML at this point. Now, if you're not, think of it as it's a scripted way of you providing input in the source of the training set. The system will then go through and figure out which algorithms are the most accurate based upon your feature selection. Then after that, once it determines which algorithms are the most accurate, it will then also start doing hyper parameter tuning and try to refine it and make it as accurate as it can be in an automated fashion to help you get along the road of picking out, basically building your model and finalizing it and pushing it to production.

The computes, they have specific computes that include ML libraries. So, for the computes with the ML libraries, it's already installed on there. It comes with the typical scikit learn, Numpy pandas, Tensorflow, all of the different libraries, not all of them, but most of the standard ones as a starting point on your compute.

If you look here, this is the Machine Learning tab. In the Machine Learning tab, we have the typical stuff we talked about, workspaces and repose, and you can get to your data and your computes. But it also works with MLflow. Anybody who's familiar with machine learning, an industry standard, or quickly becoming the standard, is something called MLflow, which is an open source method or sets of methods and programs and libraries that are used to catalog. When you go through an experiment, you basically instantiate MLflow and you then commit your artifacts and your statistics into MLflow. It's a way of logging all the information about that experiment. In this case, we have an experiment selected, and these are the artifacts that end up getting logged with it. From a machine learning standpoint, you've got your model stored as a pickle file, you've got your YAMLs, your environmental variables, YAMLs for Conda. It gives you everything you need so that you can then publish that model if you're ready to actually use it.

As a feature store, feature store is unique to Databricks. I haven't seen it anywhere else. Think of it as a way of storing your data engineering, basically your feature engineering features, almost like a master data table, except it's locked down. But it gives you the ability to standardize your features so that you can use them across different applications and across different models. It has a model repository, which you can use to save off your models and publish your models too, or save them into the repository, then from there published to something out to a serving endpoint for real time machine learning.

If we talk a little bit about **Databricks on Microsoft Azure, how does it work out on Microsoft Azure**? Databricks, of course, you're worried about **data security**. Databricks is built for the cloud, which is nice. It exists in all three major clouds. With Databricks, your data stays within your own storage account, and that's important. Think of it like it's a service. It's a data processing service. But what it does is that data that you then process through gets stored back in your own storage account.

There's something called the **Unity Catalog**. The Unity Catalog is a governance tool built by Databricks. What it does is you set it up at your account level. Think of it as you might have a bunch of databases, data brick workspaces with different schemas set up within them. A schema is a database, basically. You end up having multiple schemas spread out across all these workspaces. Well, with Unity Catalog, it gives you the ability to basically control access to them in one centralized location. Then it also basically is where your logs are stored for accessing that data. You have all your logs across your entire data estate stored inside of one location to be able to see who is doing what to it. It's important to say that Unity Catalog must be implemented. I say that because you may have workspaces in place now if you're already working with Databricks. If you decide to use Unity Catalog, because it becomes this overriding authority to your different workspaces, you may have to make some enhancements or migrate your code to it. It's not a difficult migration, but there are adjustments that you need to make whenever you access data within your code or your SQL to make sure now that it accounts for the Unity Catalog in place.

For **enforcement**, access controls are enforced everywhere. When we talk about the Unity Catalog, that's what I was referring to about having to possibly port your code, is that when you instantiate the Unity Catalog, it then controls and checks in the Unity Catalog definition who can access what, whether it's in a program or whether it's on the front end on end or the SQL side. If we look here, I'm just going to minimize the little window here for a minute.

If we look here, so this is basically Databricks and the way it exists out in the cloud. You have a control pane up here. This is the stuff basically. So, it's the Databricks application itself. This is out inside of a Databricks account. This is your Databricks account. But basically, inside the Databricks at the account level, so above your workspaces, you've got your database web application, which is what you see in the GUI to run Databricks. It also stores your notebooks, so, all of the code you write, and it stores the queries.

Now, if we look down below, this is in your customer storage account. The important thing about security, at least as an organization, people are typically worried about where does my data reside and where does it go? Well, with Databricks, it resides inside your own Azure storage account that you have complete control over. When you then run it, they're your own clusters that are basically pulling the data out of your storage account, up into the clusters, executing the code that you have stored off in your Databricks application, executing that code, and returning the result sets back to your own storage account. Your data, the entire time, is stored out inside of your own storage account. You even have the possibility that if you want to go to Databricks and say, I also want my notebooks and my queries stored in my own storage account. I think behind the scenes, they can flip a switch and it'll be pointing somewhere.

From the standpoint of the Unity Catalog, basically, we're looking at the Unity Catalog, which has the user management globally amongst all your Databricks workspaces, and the meta store, which stores everything about your catalogs and everything below it, along with the logs as to who's accessed what. That's this Meta store here, and these could be separate workspaces here. Your catalog with your different schemas, which are databases and tables, you can control all that access from the Unity Catalog.

So, if we talk a little bit about the **Databricks Lakehouse, why?** Well, why do we need a Lakehouse? Lots of people might not need a Lakehouse, but moving forward, lots of people will. If we think about how Analytics, it's now called Analytics. Originally, it was called enterprise data warehousing. It then became business intelligence. Now, here we are in the analytics.

Now, **data warehousing** has been around since the late '80s. It's designed to store structured data. Think of it as your relational database. You used to have them in Oracle, maybe SAP Hana, inside of any number of them. It's structured data where you clean it, you transform it, and you aggregate it, and it's used for BI reporting.

Around 2010, the **data lakes** came around, and the data lakes were basically just really cheap storage that you could just basically put all your data there. You'll hear them referenced to as data swamps. That's because most of the time they did become a swamp. Everybody just threw everything in there and it was a mess. Just because you had everything in there wasn't useful. What people ended up doing was they had to write programs to take it out of the lake and process what was in the lake. Then they pushed it into a relational database like a warehouse to actually consume it because then it had some structure.

Now we have a **Lakehouse**. Now what a Lakehouse is, is it combines the data lake that you have, which is where you're just dumping all of your data into your raw storage, and it then combines it with data warehouse capabilities on top. It's really one environment that the Lakehouse sits on top of your lake and it is your data warehouse.

Introduction to the Databricks Lakehouse Platform

The major advantage includes functionality that's built in to ingest all of this raw data in your lake and to process through massive amounts of data if you have massive amounts. But we have customers where it's not massive amounts, where it might be 300 million rows on different sources. That might be a big source, 20 million rows on another one. We're still building it out inside of Databricks, and we're building it out inside of Databricks because of the possibility of what Databricks brings with it.

This is just a quick look at the Lakehouse. We've got our data lake down on the bottom. That's all the raw data. Your data lake could be S 3 buckets. It could be in Azure or Google Cloud. We have the **Delta Lake** on top, which are tables. I'm going to talk about those in a moment. Those are **Delta tables**. We have these tables on top, which add structure to it, to the lake. After that, we have the Unity Catalog, which sits on top of those Delta tables to control access to them. Then we have the different purposes up above. Data warehousing, we've got basically data science and machine learning. Data engineering for moving all of the data through there. It might not be for reporting, it might just be for processing data. It also supports data streaming. We could have data flying in on the backside right through a stream, all the way through Databricks and back to the source, which is something that's happening in places like Comcast. Or if you go to some place like a CVS where you go to the cash register, you enter in your information, next thing you know, you have 14 coupons that spit out. Well, those 14 coupons took all the information from the register, the POS system, sent it through Databricks, and it went back to the register again to tell you what coupons you should need.

If we look a little bit about the **Databricks Lakehouse architecture**. The architecture itself. It's built for ingestion, so we have our raw storage on the front. Here we've got basically these different methods that they have here. Programmatically, you can use a copy into command, which is used in batch, and it'll make sure that it never processes the same file twice to send it into your tables. You've also got auto loader, which is something which runs as a stream, basically, and it will automatically, when something lands, move it in. Rather than batch, it'll automatically move the data into your tables. We also have structured streaming, which has to do with maybe you had a Kafka on the front and you connected this directly to that stream, so there is no landing zone and the data just flows right through.

Now, Databricks is built on what they call the medallion approach. When it comes to processing data and their tables, what they do is they say, Okay, well, you want to take a layered staging approach, which comes from the data warehousing days, a layered staging approach, which basically means that your first set of tables, your Bronze tables is all the raw data. Whatever landed here, plus maybe some tracking fields and things, the data stored here just as it came in.

Silver is where you end up doing all of your processing. You basically, as you load the data from Bronze into Silver, in this layer, you go through and you transform your data, you cleanse it, you might do some harmonizations, whatever you need to do to make that data useful.

And then the last layer is what people would call their Mart layer. It's basically your aggregated data. That's the data that you end up making available to your front end.

So, you may have 300 million records come through to your silver layer. Your bronze might have 400 million. Your silver has the most common version of that, which then reduces it to 300 million. And your Gold level aggregated level might be down to 100 million. But it's consumed through a SQL endpoint. So, it's called the data warehouse here. But what this really is is this is a SQL endpoint and it's a SQL engine that sits on top of the tables that you're populating. When you then access, you can then access this SQL endpoint through Power BI or Tableau or whatever your front end of choice is.

We also have different things for the platform down below. Unity Catalog, we talked about for governance. There's things called workflows, which basically think of them as pipelines. You use those inside of Databricks. It connects to Git for the repositories. Delta Live tables are something that you actually use inside of your workflows. Think of them as steps in the pipeline. Then on top of that, on the top use case, we have machine learning capabilities which we talked about earlier. You can code your notebooks inside of Python, R. It uses Spark for the parallel processing. You do your development inside your notebooks. From there, you can then store your stuff inside of MLflow for the versions of your models, and then you can productionize those and consume them and either do batch inferencing or real time inferencing through an endpoint.

The way they built this with the functionality is you can really process a lot of data, standardize it, add structure to it, and then use it for whatever use case you need, whether it's ML, whether it's BI, or no matter what the use is. You can get data out of these tables as well for any other reason.

Real quickly, we'll talk about basically the Delta Lake. What is a **Delta Lake and Delta Live tables**? A Delta Lake is basically a set of Delta tables. Delta Lake is open source, it's an open source concept. What it is, is a Delta table is the concept that they came up with to be able to use files for storage. Think of their parquet files. A Delta table, they used file for storage. When you go and create a Delta table, what you're really doing is creating a folder inside your BLOB storage, or inside BLOBs, it's really a namespace within that structure. But you're creating a folder within that area. Then basically what they're doing is, when you do updates to that table, Delta tables are automatically figuring out basically what the current state of that table is. Then when you run a SQL statement on it, it gives you the current state out of all the files it has inside that directory.

It does also have the ability because it has a transaction log stored with it, it gives you the ability to do something called time travel. Time travel just means that you can say, I loaded 100 records into that table today and it really messed it up. Can I get back to where I was yesterday? And the answer with time travel is yes. You can say, do a select all from this table where the date or as of yesterday. And when you say as of yesterday, it gives you the date of the view from yesterday.

The Delta Lake follows a medallion approach we talked about, and you can access it with SQL. This is just a picture of the Delta Lake. But what it really is, is it's all in BLOB storage. So, all these tables, your Bronze, Silver, Gold layers are stored as BLOB data inside of a storage account. And Delta technology is taking care of making sure that it actually acts like a table.

Introduction to the Databricks Lakehouse Platform

Delta Live tables are basically your pipelines. They help you move the data through your different medallion layers. They are pipelines, so they tie together notebooks. Notebooks actually do the movement. Your Delta Live table is the thing that makes it flow. You code them in notebooks. They support both streaming and batch, so you can run them periodically or you can run them as a stream. Now, you can look at it also and look at the lineage of the flow. And you can also go in and it uses grade expectations, which is the data quality mechanism instead of libraries where you can put additional rules inside of your notebooks on the different tables and you can monitor data quality as it flows through.

A simple use case, we're down to the **use cases** now. For a simple use case, this is a use case that actually we're doing in a customer now. It uses the **Unity Catalog** for governance. It uses **Auto Loader** for loading data in. It does transformations. The transformations are done via notebooks and more importantly, something called **DBT**, which is basically it's a tool, it's a partner of Databricks. It's Data Build Tool is what it stands for. But what it really comes down to is it's a SQL scripting tool which adds structure to your SQL scripting so that you can deliver it and manage it. Last but not least on the front end is we have **Power BI**.

So, if we look at the use case here, this is similar to the diagram we showed you earlier. I just put in here the pieces that this particular customer was using. Now, inside of here, we've got our BLOB storage, which is used as a landing zone. Files then end up there. We use a data factory to put them there, people at it, put them there. They end up in the landing zone. From there, we then ingest the data. For the most part, we're using auto loader so that whenever something lands, automatically gets moved to the Bronze table of whatever that source is. From there, we then do a run, a DBT run, which is basically using DBT, we're then scripting out the definition of the build of silver and gold. Bronze is something that's defined by the source. When it lands, the rest of this is defined with DBT and it stored procedures and scripting.

These tables are being persisted. Because of that, they are actual tables in there. We can basically from that on our gold, we're writing queries on it via SQL endpoint and Power BI dashboards. We use Unity Catalog for governance. We use workflows to trigger DBT runs. And from there, we've also were connected to Git for any work.

Our last or second use case is for data science and ML. That same customer has a separate group that's doing data science. And that separate group, basically they're running **experiments**. You code them inside a notebook, you're writing them in Python and in Scala, mostly Python. They are using the **scalable computes,** Databricks, and those computers are coming preconfigured with the ML libraries, scikit learn, Tensorflow. They use **MLflow** to store their models. And then basically, the **models** are stored inside the repository and they're versioned. And every once in a while, they come up with a better version and they publish those models to a serving endpoint. They're not using the feature store. They just started experimenting with it. But what it does do is the **feature stor**e, I don't show it here once again, is it kind of locks down your features.

Introduction to the Databricks Lakehouse Platform

Companies spend a lot of time doing feature engineering. When you do feature engineering, think of it as maybe it's a customer. You've got a customer ID, you've got all these demographics about the customer, but you also have a bunch of purchase patterns for that customer and for those demographic groups. They go through a lot of time figuring out and engineering different features so that when that customer's information flows through a model, that the features are consistent across that. The data might flow through four models, and those four models may need consistent features, whether you're for vectorization and a whole bunch of things like that. They use a feature store for that.

But anyways, this customer is using some information from their Delta tables for their data science group. But they're also sometimes going straight after BLOB storage for additional information. They have a separate set of tables that they use for Delta tables for their data science group, especially in Dev, because the tables multiply. They keep ingesting all sorts of data. In the end, they're using their machine learning, and they can consume that machine learning, whether it's in normal data flows for the existing data warehouse. Just as often it's also maybe forecasting data that they end up consuming. They pump the forecasting data back into tables and then they consume it through a SQL endpoint to get it to Power BI.

I know I just covered a whole bunch of information. If anybody has any **questions**, of course, outside of this call, you can always reach us. We can be reached at Analytics-People.com or Eastern-Analytics.us. You can call me directly. I'm brave enough to have my phone number on the bottom of the screen here (781) 757-7036. You can call me directly or text me or however you want. Let me see, we may have some questions here.

I know I had a separate window for questions. I'm just scrolling through them here. We did have a question about whether or not we'll have a **webinar recording**, which I don't blame you because I covered a lot. Yes, we will have a recording and we can send that out to you.

The next question is, I **mentioned that you have to code everything. Can I elaborate on that**? Yes. So Databricks, in a lot of the mature enterprise data warehouse space, you basically draw your data flows out. You build your structure at the beginning, you draw your data flow out, you maybe insert little snipp of code along your data flow to do your transformations. Now, this tool itself is basically all code, so you do have to connect to your source inside of your notebook. You connect all your script files, you connect to your sources. From there, you then pull it into a data frame, you manipulate it, do whatever you need to. You can use SQL in there, and then as well as Python or whatever your language choice is. And then from there, you end up writing your results back to a table, but it's done in code. So, lots of times people come up with standard templates for different kinds of notebooks.

Let's see. **Do you have to use Unity Catalog**? You do not have to use Unity Catalog. Unity Catalog comes into play when you start really saying, 'Okay, how big is the use case for this? Is it going to be a lot of use cases for Databricks?' People tend to start using it, and once they start using it, the use cases just keep piling on. So, once you start get it growing and your environment grows, that's when you start to run into a government's problem. From there, you don't have to, but you will have to migrate to it if you don't now and you want to use that later.

Introduction to the Databricks Lakehouse Platform

Do we have to use multiple workspaces in a system landscape? No, you don't have to use multiple workspaces. Usually, people segregate the use of data breaks by maybe functional area or areas of the business. They usually break up their workspaces that way so that people aren't stepping on each other.

Then last question, and actually, I think I have another question up here. It might be the same one.

I'm a Python programmer. Can I write everything in Python? Yes, you can. You can go through and you can write everything in Python. If you want it to take advantage of Spark, which is the parallel processing, you're going to want to use Spark commands, which are variations of Python.

Let's see, another question. **Ml models built in vendor specific source are the easily transferable to open source framework?** If you think about your model, it really depends. Let me read that again just to make sure. Ml models built in vendor specific source are the easily transferable to open source framework. The traditional way of dealing with models is you end up dealing with models in pickle files. When I first heard of that, I was like, 'What the heck is a pickle file?' But when you end up exporting your model and maybe your vectorizations, right? Maybe you're doing an LPN TDF, whatever you're exporting, you can pull those into MLflow.

So, you might have to write something to import it, right? You might have to write a Python program to basically import that file and then save it inside of ML flow. But programmatically, you should be able to do that just because if you were going to code that and you trained a model, you'd still have an object inside your code that stored basically that model, and that would be what you'd be logging into ML flow.

Programmatically, you can pull in your models and things in code, pull them in and put you put them into MLflow if you then want to publish them or consume them inside of Databricks. Yes, you can do that. I would say it's really just more a programming exercise. You're dealing with it in Spark, so you're using these models, you're basically training them, and in this case, it might not be trained, but you're consuming them and you're consuming them inside of a programming environment which really has everything you need to consume them. It's more just a matter of instantiating, pulling that model and connecting to it appropriately, whether it's NML flow or not, and then running your predictions on it.

That's it. Okay, terrific. Well, thank you, everybody, for coming. If you have any questions, we would love to hear from you. We're always there. We provide our expertise, whether it's us doing a project or whether it's just somebody, you might just need somebody with a specific expertise to come in and help you for a while. That's what we do. Thank you very much. Bye bye.