Machine Learning (and Data Analytics) with Azure Databricks

Tomaž Kaštrun, Data Platform MVP

Boston Business Intelligence Meetup
20.09.2022

About (3.0.1)

- BI Developer and data analyst
- SQL Server, SAS, R, Python, C#, SAP, SPSS
- 20years experience MSSQL, DEV, BI, DM
- Frequent community speaker
- Avid coffee drinker & bicycle junkie
- I do a lot of weather prediction

http://tomaztsql.wordpress.com

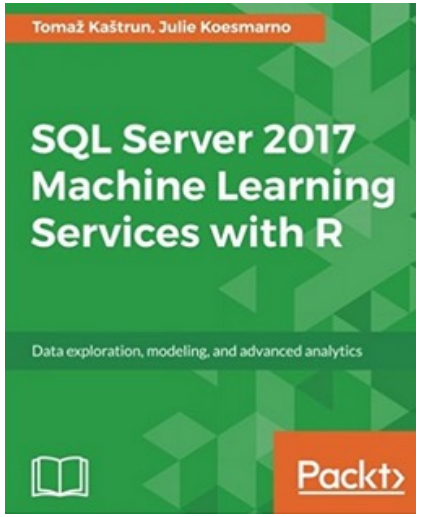
tomaz.kastrun@gmail.com

@tomaz_tsql

/in/tomaztsql

http://github.com/tomaztk

https://mvp.microsoft.com/PublicProfile/5002196



Material for this session: https://github.com/tomaztk/Azure-Databricks

What is Azure Databrick?

A fast, easy and collaborative Apache Spark based analytics platform optimized for Azure.

Benefits:

- Designed in collaboration with Apache Spark founders
- One-click set up; streamlined workflows
- Interactive workspace that enables collaboration between data scientists, data engineers, and business analysts
- Native Integration with Azure Services (HDFS / Blob storage, Azure DW, Power BI, Functions, ADF gen2,...)
- Integrated Azure security and identity management (AD integration, compliance, enterprise-grade SLAs)

What is Azure Databrick?

... furthermore

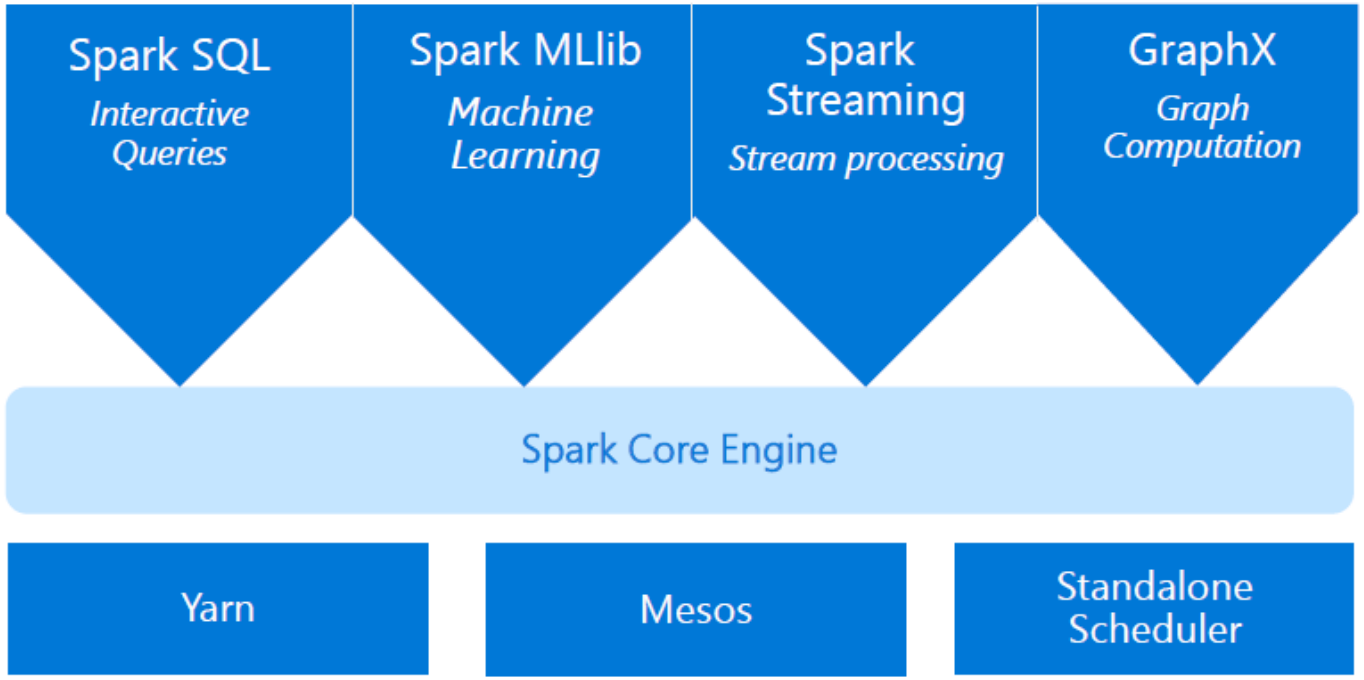
- is unified data analytics platform
- including discussion of how open-source projects including Apache Spark(TM)
- Delta Lake for data engineers
- MLflow and Koalas for data scientists
- Delivers Data Science Workspace using and operating with Spark clusters
- Provides great collaboration with notebook
- Brings Delta Lake for data orchestration and time travel
- Brings Databricks SQL (Analytics)

Why Spark?

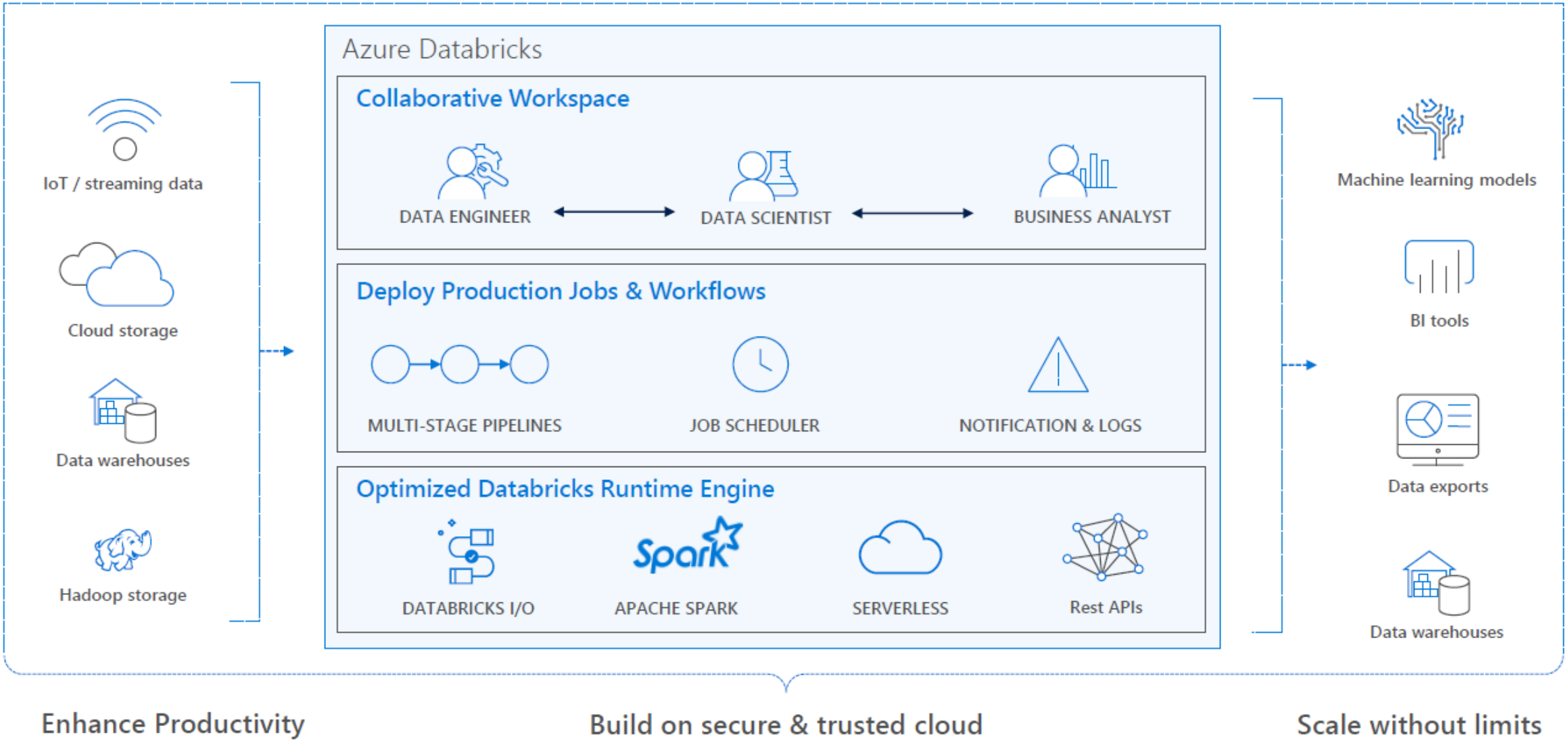
- Open-source data processing engine
- Build on philosopfy of speed, ease of use, RDD files and analytics
- 100+ times faster than Hadoop
- Highly extensible with support for scala, Java, R, Python and packages for Spark SQL, GraphX, data streaming and ML (Machine learning libraries)
- Connect to prefered storage

Spark unifies

- Batch Processing
- Interactive SQL
- Real-time processing
- Machine Learning
- Deep Learning
- Graph Processing



Azure Databricks

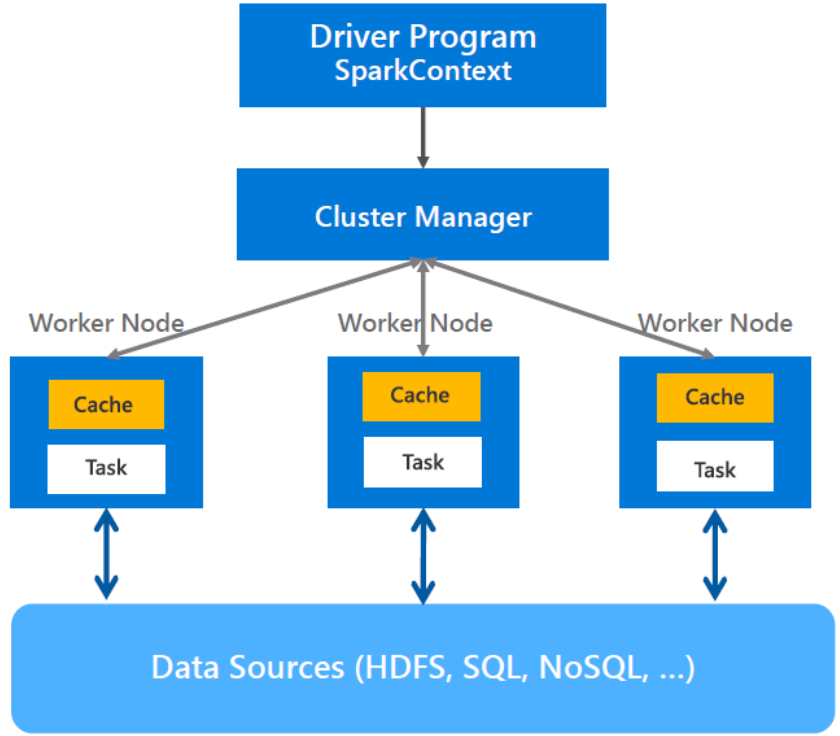


Essentials for ML Projects

- Clusters, DBFS, GIT, Runtimes, Spark 3.0
- Data integration and orchestration with Delta Lake
- AutoML, ML Flow
- Databricks SQL (Lakehouse)

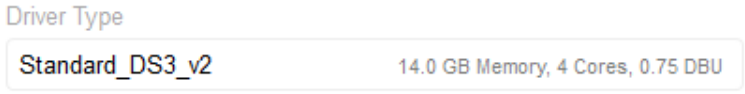
Scalling - Cluster Architecture

- ‚Driver‘ runs the user‘s main function and executes various parallel operations on workers nodes,
- The results of the operations are collected by the driver
- The worker node reads and write data from/to data sources
- Worker node cache (delta caching / IO) transforms data in Memory as RDDs (Resilient Data Sets)
- Worker nodes and the Driver node execute as VMs in the cloud
- RDD variables: broadcasted and accumulated variables

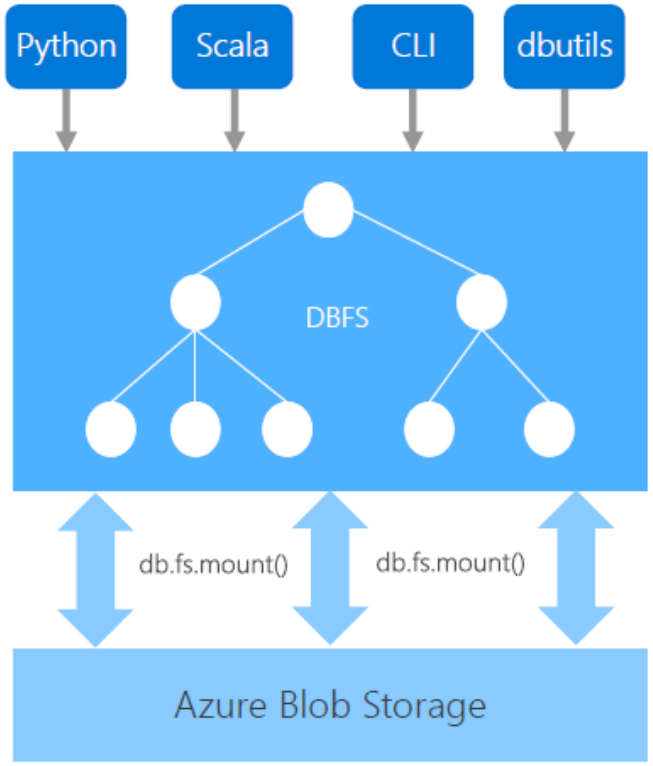


6 items

NAME	TYPE	STATUS	RESOURCE GROUP	LOCATION
1ee746f0cc3943f7ac9e54f5c711d410	Virtual machine	Running	databricks-rg-TK_Bricks-bckhauk2i6uxy	West Europe
30b5d3d88e064a0b88fad9f174af3bbe	Virtual machine	Running	databricks-rg-TK_Bricks-bckhauk2i6uxy	West Europe
384224d8de0d442aa40c237ef8c6bd04	Virtual machine	Running	databricks-rg-TK_Bricks-bckhauk2i6uxy	West Europe
9dfd4985382442689ccef4598d75e181	Virtual machine	Running	databricks-rg-TK_Bricks-bckhauk2i6uxy	West Europe
b03cbe5bb72c426a9a552572c9050889	Virtual machine	Running	databricks-rg-TK_Bricks-bckhauk2i6uxy	West Europe
ca943ca757c945a780a3be9199bc4f9c	Virtual machine	Running	databricks-rg-TK_Bricks-bckhauk2i6uxy	West Europe



Scalling - DBFS



- Azure Storage buckets can be mounted in DBFS (distributed file System) that is a layer over Azure Blob Storage and can be directly accessed Without specifying the storage keys
- DBFS mounts are created using dbutils.fs.mout()
- Azure storage data can be cached locally on each of the workers nodes
- Python and Scala can access both via DBFS CLI
- Data always persists in Azure Blob Storage and is never lost after cluster termination
- DBFS comes preinstalled on Spark clusters in Databricks

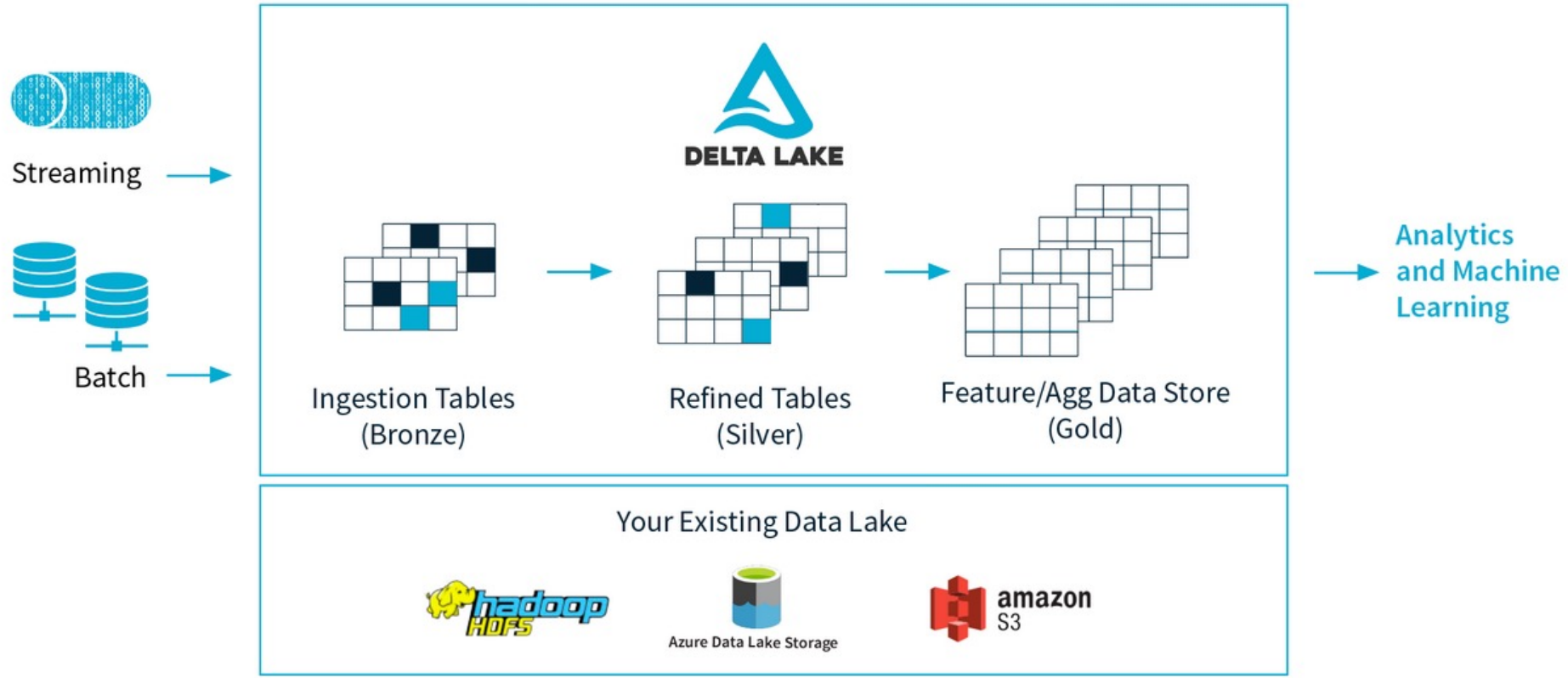
Cmd 2

```
# File location and type
file_location = "/FileStore/tables/AAPL.csv"
file_type = "csv"

# CSV options
infer_schema = "false"
first_row_is_header = "true"
delimiter = ","

# The applied options are for CSV files. For other file type
df = spark.read.format(file_type) \
  .option("inferSchema", infer_schema) \
```

```
# With this registered as a temp view, it will only be available to th
# Once saved, this table will persist across cluster restarts as well
# To do so, choose your table name and uncomment the bottom line.

permanent_table_name = "AAPL_csv"

# df.write.format("parquet").saveAsTable(permanent_table_name)
```

Shift+Enter to run shortcuts

Delta Lake



- Introduces storage layer for ACID operations on data lakes
- Open source storage layer for data reliability in data lakes
- Fully compatible with Apacke Spark APIs

MLFlow

Design, integrate and reproduce your Machine learning models, experiments,artifacts and solutions. Keep your code sane, reproducable and accessable to data scientist, machine learning engineers, data analysts and other departments.

Will help you with:

- Keep track of your experiments -> **ML Flow Tracking**
- Standardize your way for storing models, packages -> **ML Registry**

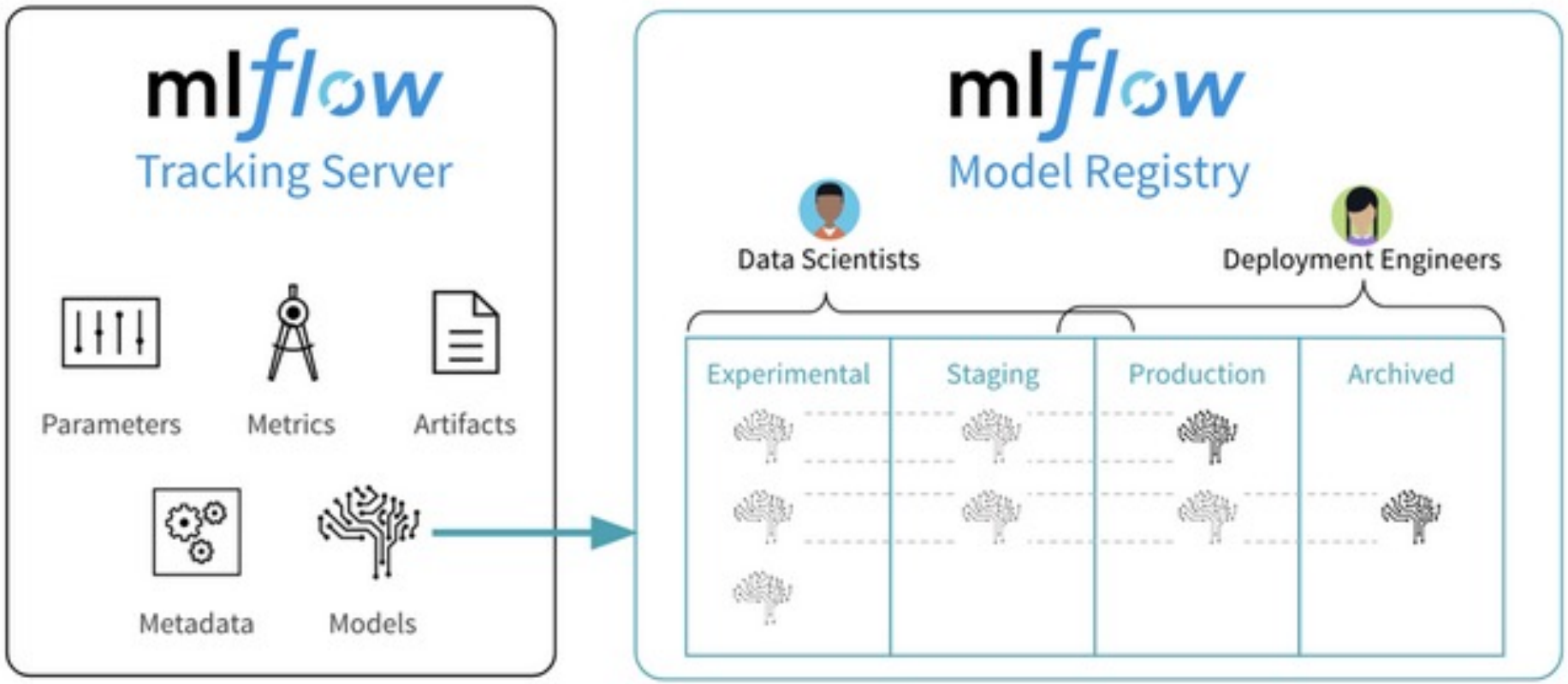
ML flow

- Creating a special notebook/git branch for each experiment is not the solution.
- In intensive experiments, scientists cannot track which model/settings achieved the best results.
- ML Flow Side bar: keep the quick iterations of development/parameter tuning/feature engineering and the corresponding results.

ML Experiment Tracking

- Tracking code development in GIT is easy.
- Tracking intensive data science experiment that includes model is almost impossible using GIT only.
- We need to go back and ask what model/settings achieved the best results.

ML Flow

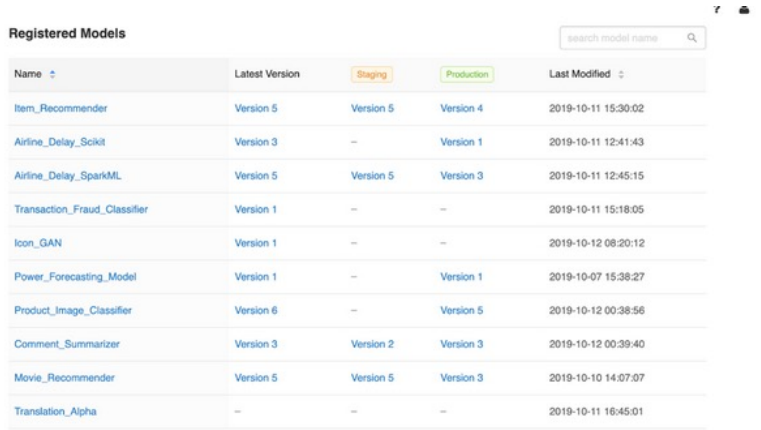


Source: https://databricks.com/

ML flow contains

Each ML experiment contains:

- **Source**: Name of the notebook.
- **Version**: Notebook revision.
- **Start & end time**: Start and end time of the run.
- **Parameters**: Key-value model parameters.
- **Tags**: Key-value run metadata.
- **Metrics**: Key-value model evaluation metrics.
- **Artifacts**: Output files in any format.



Analytics (#1)

Spark Machine Learning (ML)

- Offers a set of parallelized machine learning Algorithms (MMLSpark, Spark ML, Deep Learning, SparkR)
- Supports Model selection (hyperparameter tuning) using Cross Validation and Train-Validation split
- Offers parametrization of Notebook jobs for
- Supports Java, Scala or Pythong apps using Dataframe-Based API (current versioin Spark 2.4.0).
- Spark Mllib comes preisntalled on Azure Bricks
- Supports Scikit-Learn, XGBoosts, H20.ai and many others

```scala
import org.apache.spark.ml.linalg.{Matrix, Vectors}
import org.apache.spark.ml.stat.Correlation
import org.apache.spark.sql.Row

val data = Seq(
  Vectors.sparse(4, Seq((0, 1.0), (3, -2.0))),
  Vectors.dense(4.0, 5.0, 0.0, 3.0),
  Vectors.dense(6.0, 7.0, 0.0, 8.0),
  Vectors.sparse(4, Seq((0, 9.0), (3, 1.0)))
)

val df = data.map(Tuple1.apply).toDF("features")
val Row(coeff1: Matrix) = Correlation.corr(df, "features").head
println(s"Pearson correlation matrix:\n $coeff1")

val Row(coeff2: Matrix) = Correlation.corr(df, "features", "spearman").head
println(s"Spearman correlation matrix:\n $coeff2")
```

```python
from pyspark.ml.linalg import Vectors
from pyspark.ml.stat import Correlation

data = [(Vectors.sparse(4, [(0, 1.0), (3, -2.0)]),),
        (Vectors.dense([4.0, 5.0, 0.0, 3.0]),),
        (Vectors.dense([6.0, 7.0, 0.0, 8.0]),),
        (Vectors.sparse(4, [(0, 9.0), (3, 1.0)]),)]
df = spark.createDataFrame(data, ["features"])

r1 = Correlation.corr(df, "features").head()
print("Pearson correlation matrix:\n" + str(r1[0]))

r2 = Correlation.corr(df, "features", "spearman").head()
print("Spearman correlation matrix:\n" + str(r2[0]))
```

Analytics (#2)

MLlib

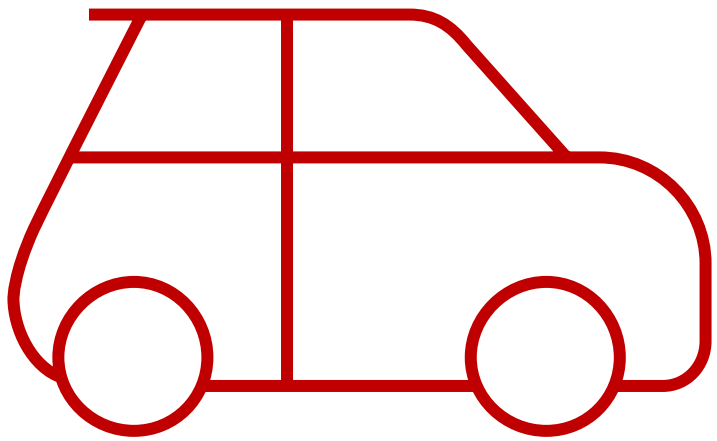
- Supports pipelines for tuning practical machine learning models on top of API dataframes
- Mllib Supports also RDD-based API based functions
- Classifications and regression
- Clustering
- Collaborative filtering (recommender systems), frequent pattern mining (association rules with FP-Growth or with PrefixSpan)
- Model selection and tuning
- Feature extraction and transformation
- Dimensionality reduction
- Evaluation metrices
- PMML model exports

AutoML

With Databricks

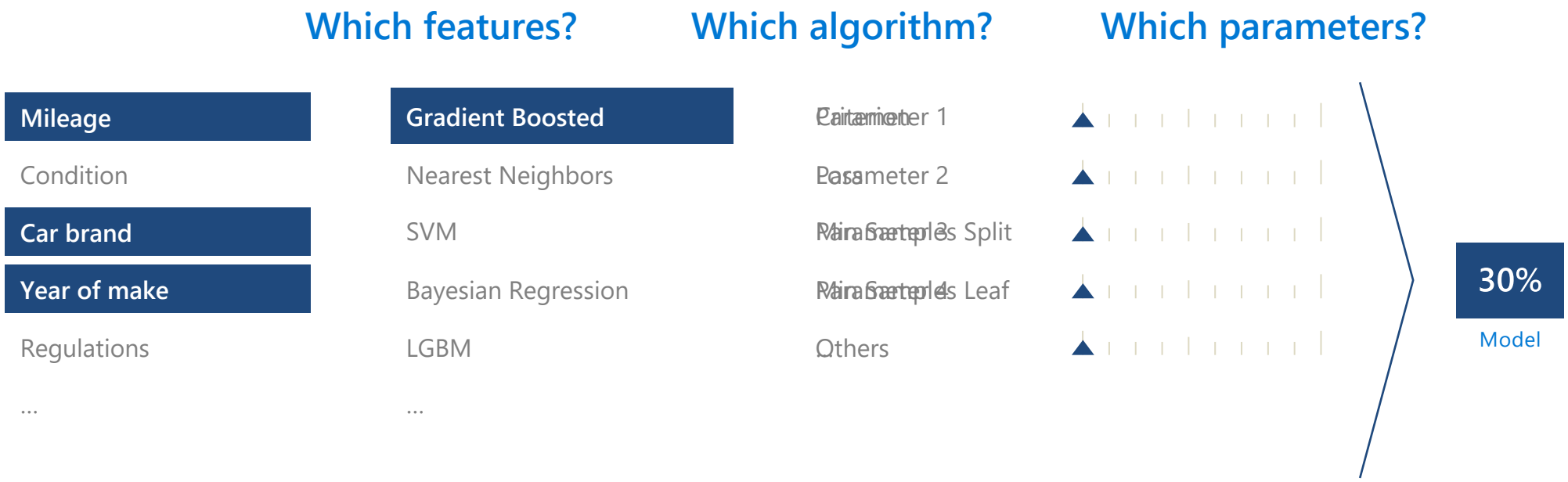
- Enables to quickly generate predictive models
- Shortens the initial decision making proces of finding the optimal algorithm
- Low code approach and quick jump-start experience
- Reproducable notebooks
- Great set of ML packages available

Auto ML



How much is this car worth?

Model Creation Is Typically Time-Consuming



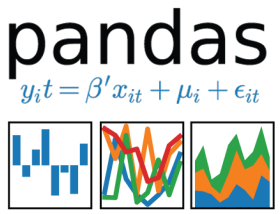
Koalas

Pure Python library

Aims at providing the pandas API on top of Apache Spark:

- unifies the two ecosystems with a familiar API
- seamless transition between small and large data

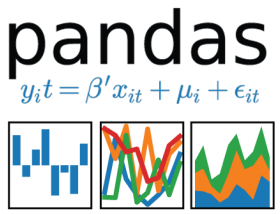
Short example of pandas vs. Spark




```
import pandas as pd
df = pd.read_csv("my_data.csv")

df.columns = ['x', 'y', 'z1']
df['x2'] = df.x * df.x
```

```
df = (spark.read
    .option("inferSchema", "true")
    .option("comment", True)
    .csv("my_data.csv"))
df = df.toDF('x', 'y', 'z1')
df = df.withColumn('x2', df.x*df.x)
```

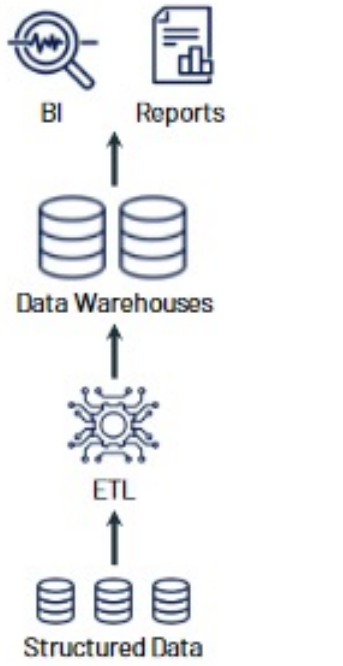
Short example of pandas vs. Spark



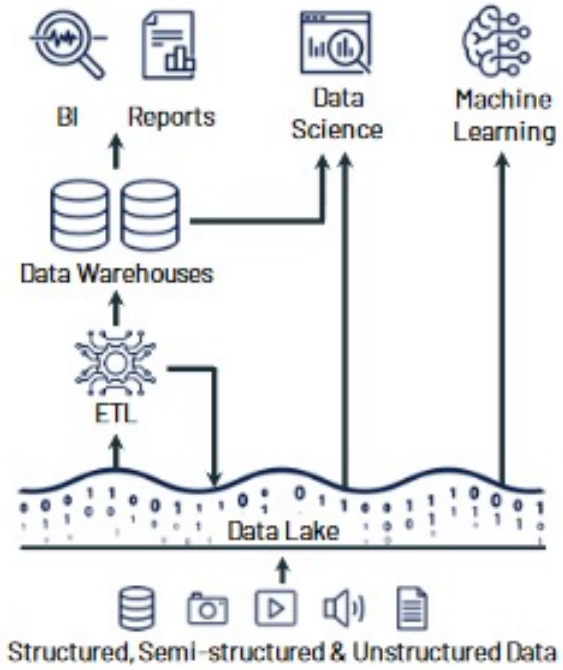

```
import pandas as pd
df = pd.read_csv("my_data.csv")

df.columns = ['x', 'y', 'z1']
df['x2'] = df.x * df.x
```

```
import databricks.koalas as ks
df = ks.read_csv("my_data.csv")
df.columns = ['x', 'y', 'z1']
df['x2'] = df.x * df.x
```

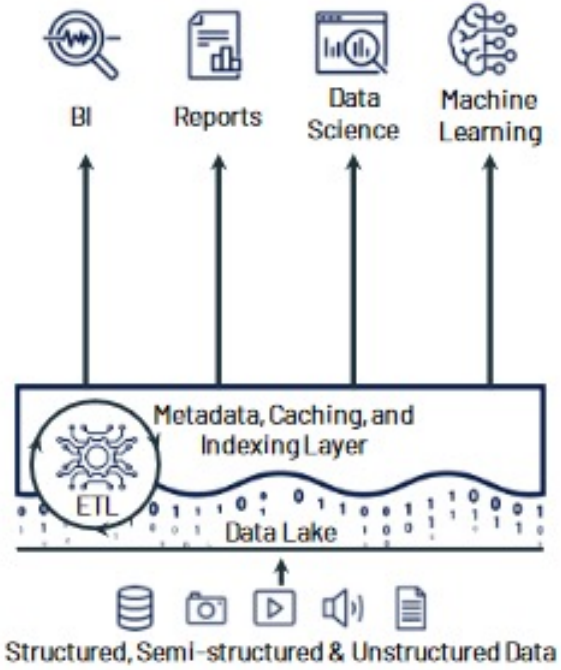
Lakehouse architecture



(a) First-generation platforms.



(b) Current two-tier architectures.



(c) Lakehouse platforms.

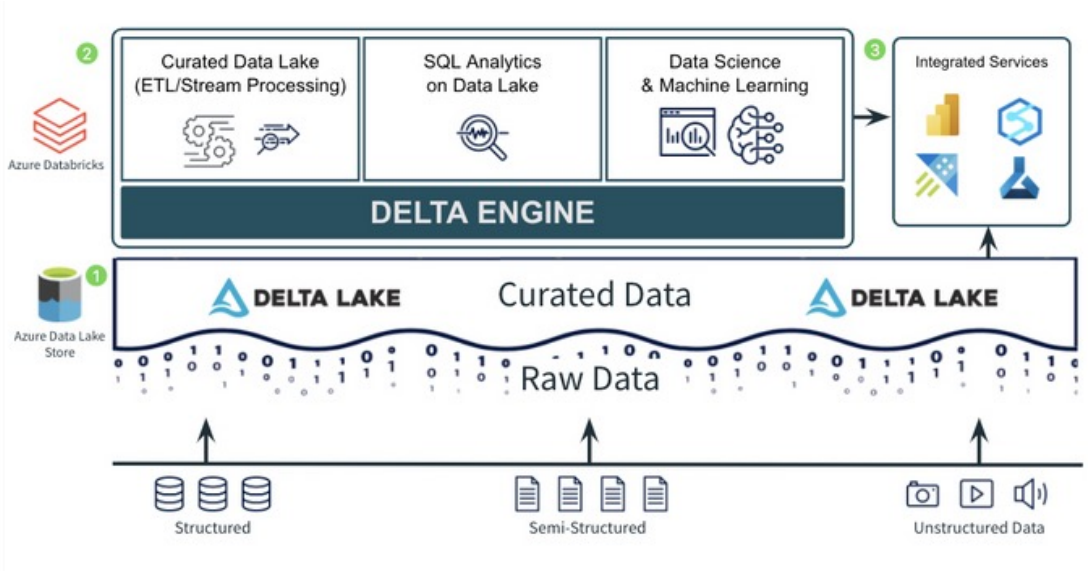
Source: http://cidrdb.org/cidr2021/papers/cidr2021_paper17.pdf

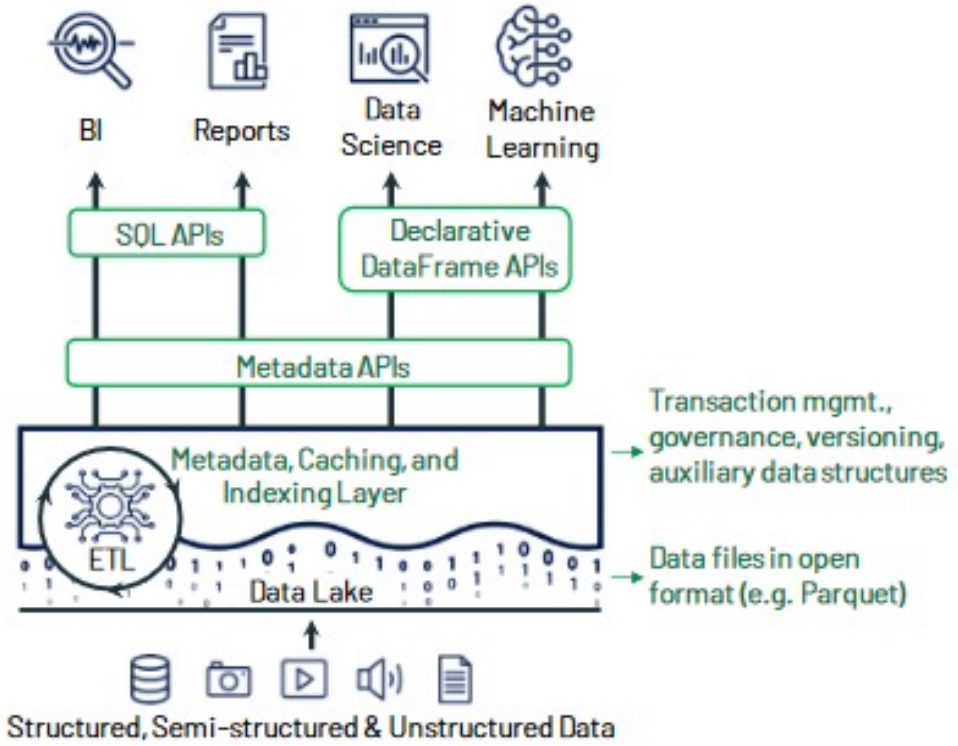
This paper argues that the data warehouse architecture as we know it today will wither in the coming years and be replaced by a new architectural pattern, the Lakehouse, which will (i) be based on open direct-access data formats, such as Apache Parquet, (ii) have first-class support for machine learning and data science, and (iii) offer state-of-the-art performance. Lakehouses can help address several major challenges with data warehouses, including data staleness, reliability, total cost of ownership, data lock-in, and limited use-case support. We discuss how the industry is already moving toward Lakehouses and how this shift may affect work in data management. We also report results from a Lakehouse system using Parquet that is competitive with popular cloud data warehouses on TPC-DS.

Lakehouse architecture

Metadata Layer for Data Management
SQL Performance:
- Caching
- Auxiliry Data
- Data Layout

Performance results (with C++ on Delta Engine)
Access for advanced analytics (Spark DataFrames)





Databricks SQL (Analytics)

Build on lakehouse architecture;

simple platform to store and manage all of your data

Support your analytics and gives AI support with different use cases

Offers:

- Querying all data in lakehouse (scalable and optimized SQL workloads)
- Works great with delta lake to support ACID transactions
- Supports multiple concurrent users
- Easy working with data, data browser and quick visualization
- Query Editor, rich dashboards, supports endpoints (thank you!!!) to connect to Power BI / Tableau / Looker

Demo

Consuming Azure DataBricks for ML

Thank you


http://tomaztsql.wordpress.com


tomaz.kastrun@gmail.com


@tomaz_tsql


/in/tomaztsql


http://github.com/tomaztk


https://mvp.microsoft.com/PublicProfile/5002196

Material for this session: https://github.com/tomaztk/Azure-Databricks