Applied Data Science in Azure with Microsoft Fabric and Azure Machine Learning

Enrico & Tomaž

Agenda

- Module 1 – Getting onboard with F&AML
- Module 2 – Storage / storing data and compute with F&AML
  - Exercise for F / AML
- Module 3 – Data Movement
  - Exercise for F / AML
- Module 4 – Data Exploration
  - Exercise for F / AML
- Module 5 - MLFlow
  - Exercise for F / AML
- Module 6 – E2E Solution
  - Exercise for F / AML

Outlook of the day:
9.00 – 12.00
12.00-13.00 - LunchTime
13.00 – 17.00
2x Coffee Time

Material

tomaztk/Fabric_Azure_Machine_Learning: Microsoft Fabric and Microsoft Azure Machine Learning (github.com)

tomaztk/Azure-Machine-Learning: Azure Machine Learning (github.com)

With you today – Enrico van de Laar

- Developer | data analyst
- Privacy Guru
- SQL Server, R, Python, C#
- 20years experience MSSQL, DEV, BI, DM
- Frequent community speaker, Author
- Privinity CEO
- MVP Alumni

PRIVINITY

MVP

With you today – Tomaž Kaštrun

- BI Developer | data analyst | data scientist
- SQL Server, SAS, R, Python, C#, SAP, SPSS
- 20years experience MSSQL, DEV, BI, DM
- Frequent community speaker
- Avid coffee drinker & bicycle junkie

http://tomaztsql.wordpress.com

tomaz.kastrun@gmail.com

@tomaz_tsql

/in/tomaztsql

http://github.com/tomaztk

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

Module 1

Starting with light general introduction to Microsoft Fabric and Azure Machine Learning Services (AML), and how you navigate these services.

- basic key concepts
- how they affect both offerings and subscriptions
- key differences and similarities between both services
- explore additional administrative settings

Basic Concepts

- **Subscription:** Pay-as-you-go, Capacity, ToC purchase
- **Azure ML Studio**: A drag-and-drop interface for building ML models.
- **Azure ML Designer**: Visual interface to create ML pipelines.
- **Notebooks**: Support for Jupyter Notebooks for code-based model development.
- **Automated ML (AutoML)**: Automatically trains and tunes models using hyperparameter tuning and model selection.
- **MLOps**: Supports integration with DevOps for model lifecycle management, version control, and continuous integration.

Basic Concepts – Data Storage

- **Azure Data Lake Storage (ADLS)**: A highly scalable data lake for big data analytics. It stores both structured and unstructured data in its native format.
- **Azure Blob Storage**: Object storage for data, images, videos, and other binary formats.
- **Azure SQL Database**: A relational database service with built-in machine learning capabilities.
- **Azure Cosmos DB**: A globally distributed, multi-model database ideal for real-time data and fast analytics.

- .

Basic Concepts – Cognitive services / Azure AI

- **Core Concept**: Pre-built APIs for AI capabilities without needing to build machine learning models from scratch.
- **Key Services**:**Vision**: Image recognition, object detection.
  - **Speech**: Speech-to-text, text-to-speech, translation.
  - **Language**: Text analysis, sentiment analysis, translation.
  - **Document**: Intelligence on documents, slicing, tokenization, embeddings
  - **Decision**: Personalization and anomaly detection.
  - **OpenAI**: Access to LLM and Vector index

Basic Concepts – Data movement, pipelines and orchestration

- **Azure Data Factory (ADF)**: A cloud ETL service that allows you to build data pipelines to move and transform data across services.
- **Azure Event Hubs**: A real-time data ingestion service for high-throughput data streaming and event-driven processing.
- **Azure Stream Analytics**: A real-time analytics service for processing data streams from IoT devices, social media, and more.

Basic Concepts – Model deployment / consumption

- **Azure Kubernetes Service (AKS)**: A managed Kubernetes service to deploy machine learning models at scale. Enables scaling, load balancing, and model versioning.
- **Azure App Service**: Another platform for deploying web apps, including those that host machine learning models.
- **Azure Functions**: Serverless compute to run lightweight machine learning models.

Basic Concepts – Model monitoring

- **Azure Monitor**: A comprehensive monitoring solution that includes logs, metrics, and alerts for deployed models and services.
- **ML Pipelines**: In Azure ML, these are reusable workflows for automating the training and deployment of models.
- **Model Versioning**: Keeping track of model versions and experiments within Azure ML or with MLOps.

Basic Concepts – Security

- **Role-Based Access Control (RBAC)**: Fine-grained access control to services and data.
- **Managed Identity**: Secure, managed identity for resources to authenticate and access other Azure services without storing credentials.
- **Data Encryption**: Built-in encryption for data at rest and in transit, ensuring security and compliance.

Basic Concepts – Inegration with other Azure services

- **Power BI**: Business intelligence tool for visualizing data and model results, tightly integrated with Azure.
- **Azure IoT Hub**: Integration for IoT devices to send data into Azure for real-time analytics and machine learning.
- **OpenAI**:

Offerings for Data Science

- Fabric
- Azure Machine Learning
- Databricks
- HD Insight

Feature / Product	Databricks	Microsoft Fabric	Azure Machine Learning	HDInsight
Primary Offering	Unified platform for data engineering, ML, and analytics	Data Fabric platform for data integration, analytics, and governance	Machine learning platform for building, training, and deploying models	Managed cloud service for big data analytics (Hadoop, Spark, etc.)
Data Engineering	5/5: Strong in ETL with Apache Spark, notebooks, and ML	4/5: Good integration but lacks Databricks' power	3/5: ML focused, limited for ETL	4/5: Built for big data, but not focused on ML
Machine Learning	5/5: Supports scalable ML with MLflow, AutoML, Spark MLlib	3/5: Limited advanced ML capabilities, better for data integration	5/5: ML-centric, strong AutoML, pipelines, distributed training	3/5: Supports ML but lacks advanced tools like Databricks
Analytics	5/5: Great for exploratory data analytics, Spark SQL	4/5: Strong reporting and Power BI integration	5/5: More ML-oriented, not ideal for interactive analytics	4/5: Focused on big data processing, not analytics per se
Programming Languages	5/5: Supports Python, SQL, Scala, R, Notebooks functionality	4/5: Primarily SQL and Power Query, Python, Spark	5/5: Python, R, Scala and SDK support for deep learning	5/5: Supports Python, Scala, and others for big data
Compute Scaling	5/5: Auto-scaling clusters, Pools, Photon, Multi	4/5: Elastic pools, easy scaling, but less compute-intensive	5/5: Scalable compute for distributed ML training, AK8s	4/5: Auto-scaling but more oriented to big data than ML
Integration	4/5: Integrates well with Azure, data lakes, MLflow	5/5: Best for Microsoft ecosystem, Power BI, Azure Synapse, DWH	5/5: Deep Azure ecosystem integration, supports Git, MLOps	4/5: Integrates well with Azure and other data services
Ease of Use	4/5: Excellent for data scientists, data engineers, but complex	5/5: User-friendly, low-code/no-code tools	4/5: Powerful, but setup can be complex for non-experts	3/5: More complex for beginners, focused on big data engineers
Personal experience				
Costs	5/5	4/5	3/5	4/5
Maturity	5/5 On Market almost 10y	3/5 Rapid development, changing focus	5/5 Mature with 10+Years	5/5 Mature

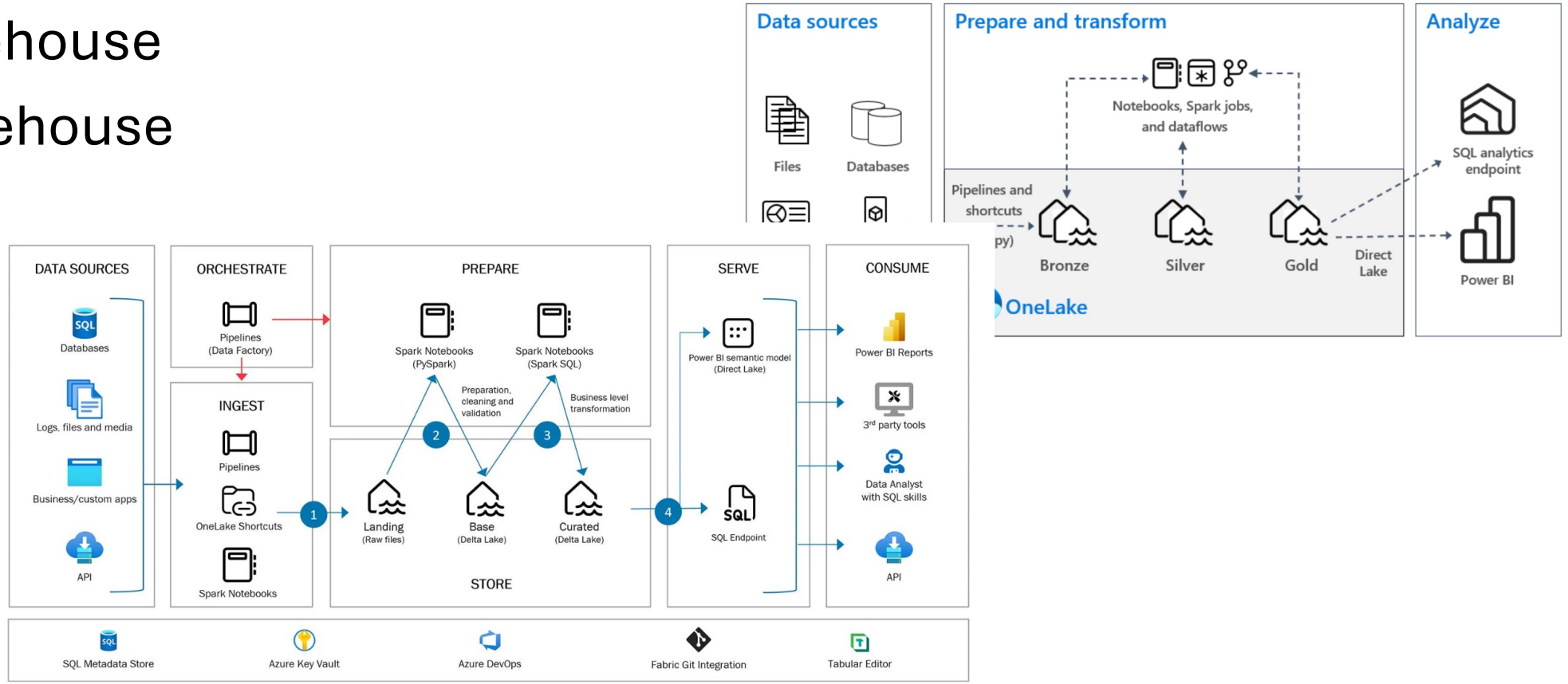
General concepts - Architecture

Workspaces (Delta Lake; ETL; Medallion; Environments; CI/CD)

Data Lakehouse

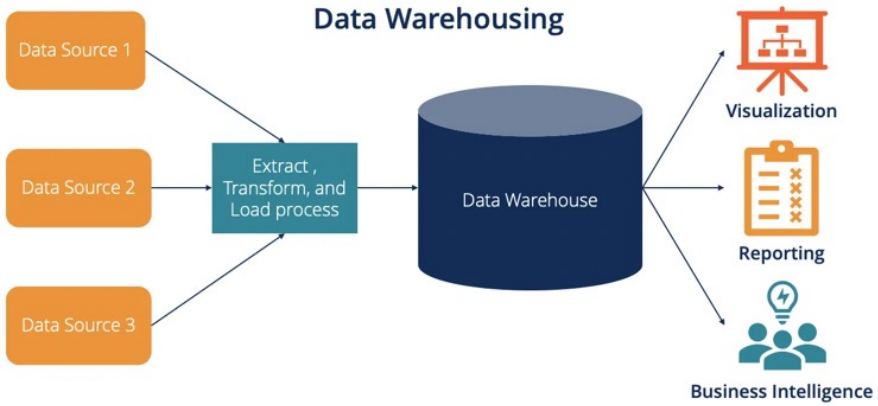
Data Warehouse

Spark



Data warehouse

- Unified data repository for storing large amounts of information from multiple sources
- Combines relational datasets from business apps, ERP, CRM, FICO
- Transactional data and master data
- Processes: extracting, clear storing, versioning, consuming



Data warehousing. Image source: https://corporatefinanceinstitute.com/

Data Warehouse PRO

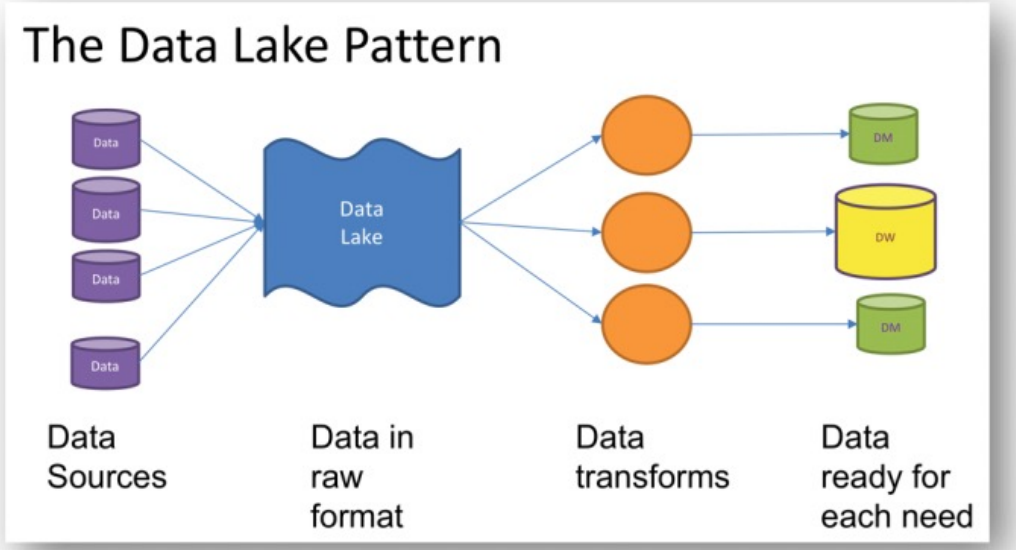
- Data canonization / standardization
- Improved data quality, consistency
- Improves decision-making business process and analytics

Data Warehouse CON

- Lack of flexibility
- “Struggles” with semi- and un-structured data
- High OPEX (implementation, CI/CD, maintenance)

Data Lake

- Centralized and highly flexible storage repository
- For structured, unstructured data; original / raw format
- Durable, cost-effective
- Schema on-read
- ETL for purposes and “ad-hoc”
- Machine learning, Shallow/Deep
- IoT devices, streaming data



The data lake pattern. Image source: datakitchen.io

Data Lake PRO

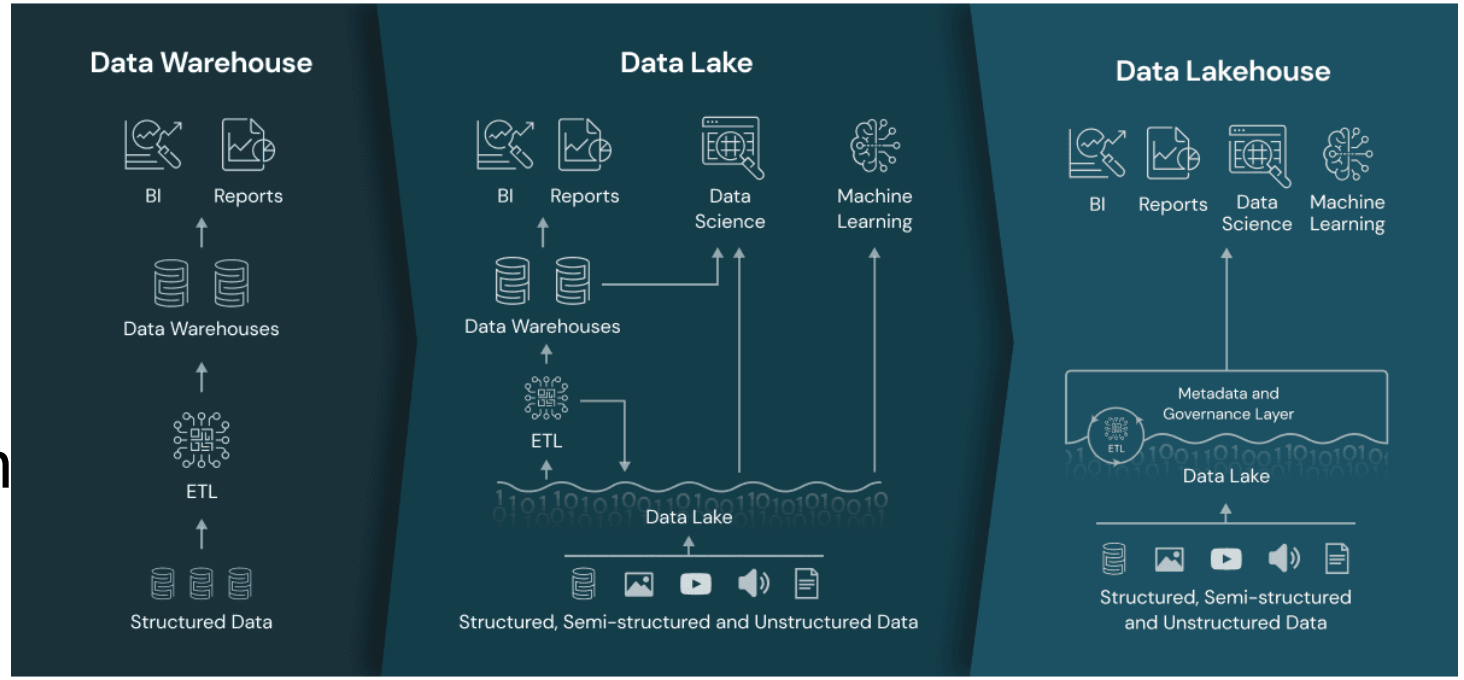
- Data consolidation (structured + unstructured)
- Improved flexibility
- Cost savings (object store)
- Improved and support for variety analytics, shallow and deep learning (machine learning)

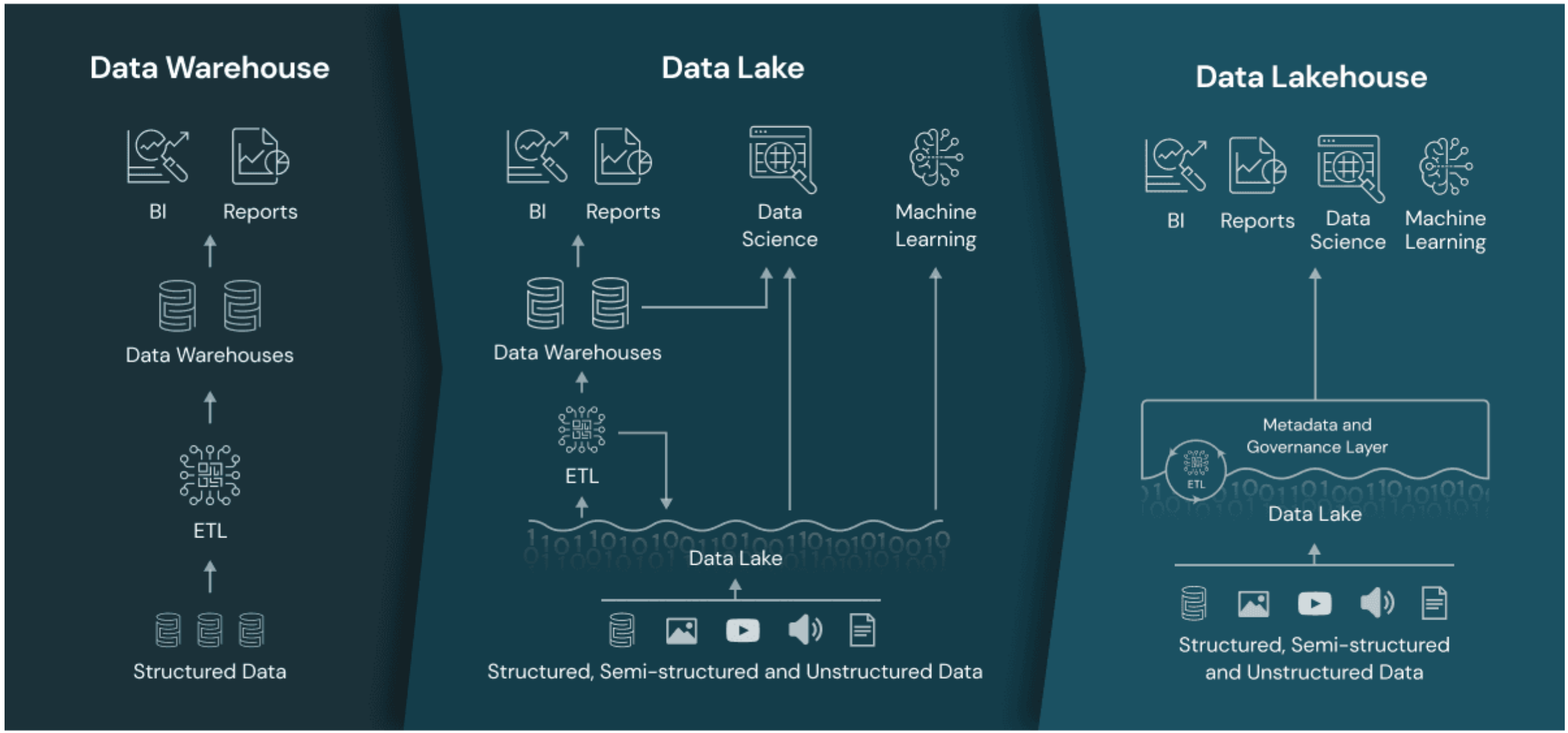
Data Lake CON

- Lack of management
- Enforcing data reliability
- Uniform security model (data formats)

Data Lakehouse

- Scalable data lakes
- Organized data managem
- Organized ACID for:
  - Transactional data (DWH)
  - Improved analytics (DWH)
  - Improving ML, DL (DWH, Data Lake)
- Metadata layer for data lakes
- Improved Query engine
- Optimized acces for data science




Data Warehouse
BI
Reports
Data Warehouses
ETL
Structured Data
Data Lake
BI
Reports
Data Science
Machine Learning
Data Warehouses
ETL
Data Lake
Structured, Semi-structured and Unstructured Data
Data Lakehouse
BI
Reports
Data Science
Machine Learning
Metadata and Governance Layer
ETL
Data Lake
Structured, Semi-structured and Unstructured Data

Data lakehouse PRO

- Reduced data reduncency
- Improved flexibility
- Cost effectiveness (combination of DWH and Data Lake)
- Improved data governance, data management, and security
- Support for variety of workloads (analytics, open data formats (API, parquet))
- Straightforward Machine Learnign

Data lakehouse CON

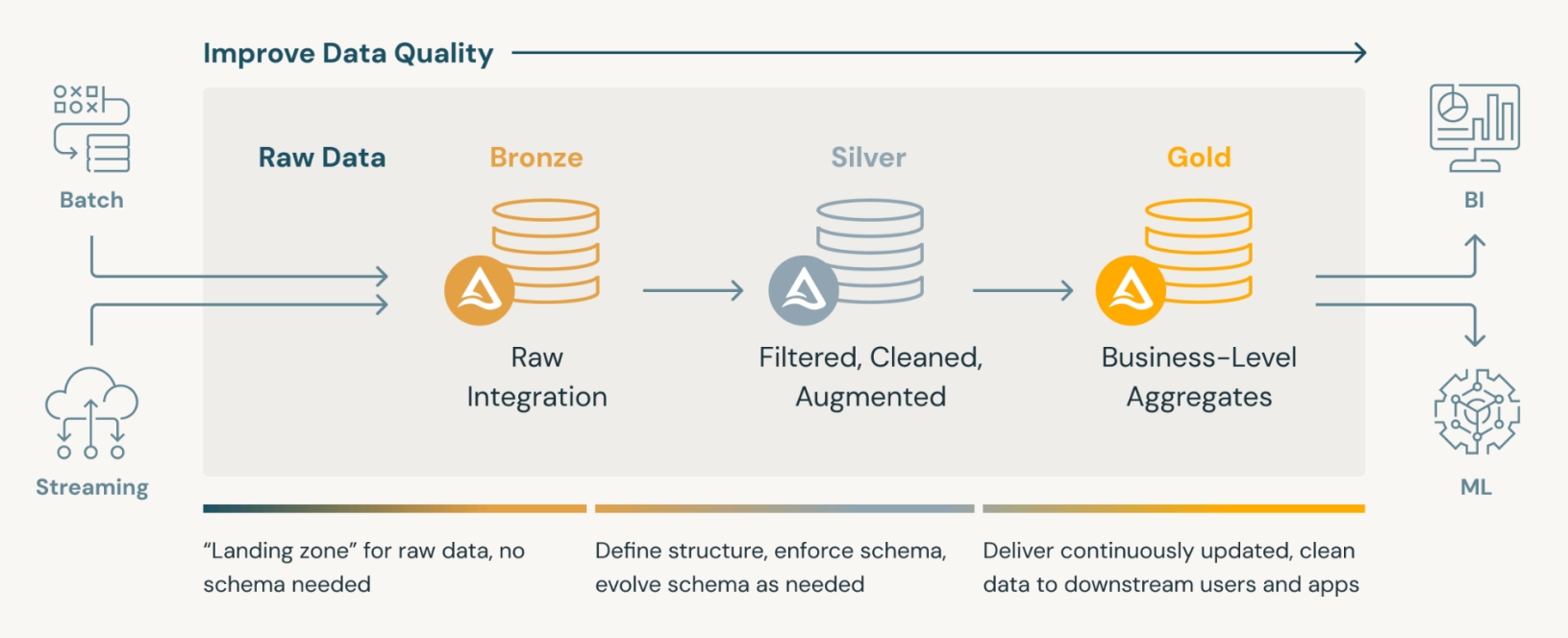
- Relatively new (concept)
- Improved, yet immature technology
- Offerings (Databricks)

Recap: DWH vs. DL vs. DLH

	Data Warehouse	Data Lake	Data Lakehouse
Storage Data Type	Works well with structured data	Works well with semi-structured and unstructured data	Can handle structured, semi-structured, and unstructured data
Purpose	Optimal for data analytics and business intelligence (BI) use-cases	Suitable for machine learning (ML) and artificial intelligence (AI) workloads	Suitable for both data analytics and machine learning workloads
Cost	Storage is costly and time-consuming	Storage is cost-effective, fast, and flexible	Storage is cost-effective, fast, and flexible
ACID Compliance	Records data in an ACID-compliant manner to ensure the highest levels of integrity	Non-ACID compliance: updates and deletes are complex operations	ACID-compliant to ensure consistency as multiple parties concurrently read or write data

General concepts – Data Lake

Medallion concept (Databricks)



ACID on Data Lake (files)

- ETL frameworks / ELT pipelines
- Assuring zones (architecture)
- Logging, storing, updating
- Delta Lake (Databricks)

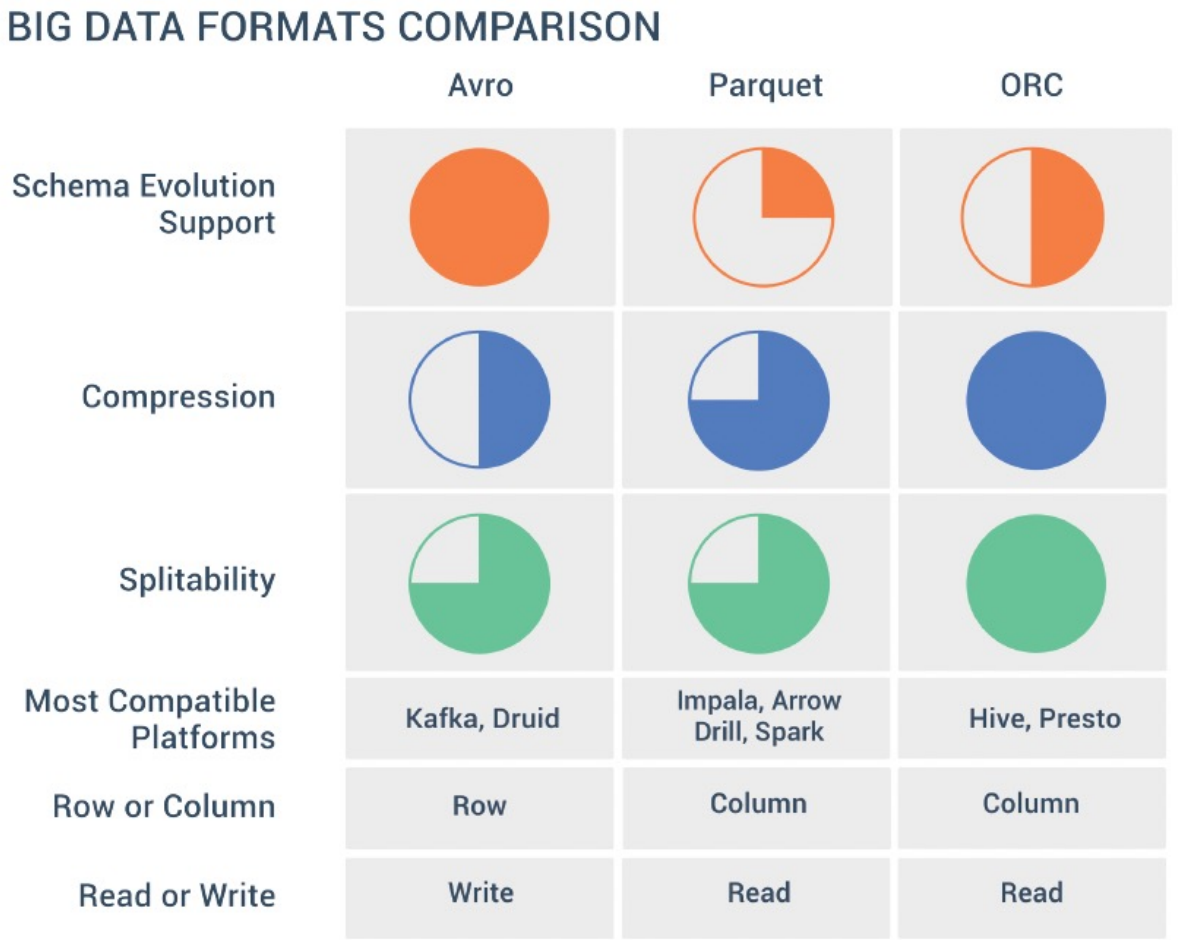
File Types in supported in Spark

- .CSV – delimited text file using comma, semicolon to separate values.
- .TXT – delimited text file.
- .JSON – text file that stores simple data structures and objects. Standard for data interchange
- .Parquet – column storage file format used by hadoop systes, such as Pig, Spark, and Hive. It has binary presentation, is cross platform.
- .AVRO – open source data serialization system by Apache Hadoop. stores serialized data in binary format and schema in JSON format.
- .ORC – open source columnar data storage format. Similar to Parquet. Is cross platform
- … and others

ORC – created by HortonWorks (feb 2013)

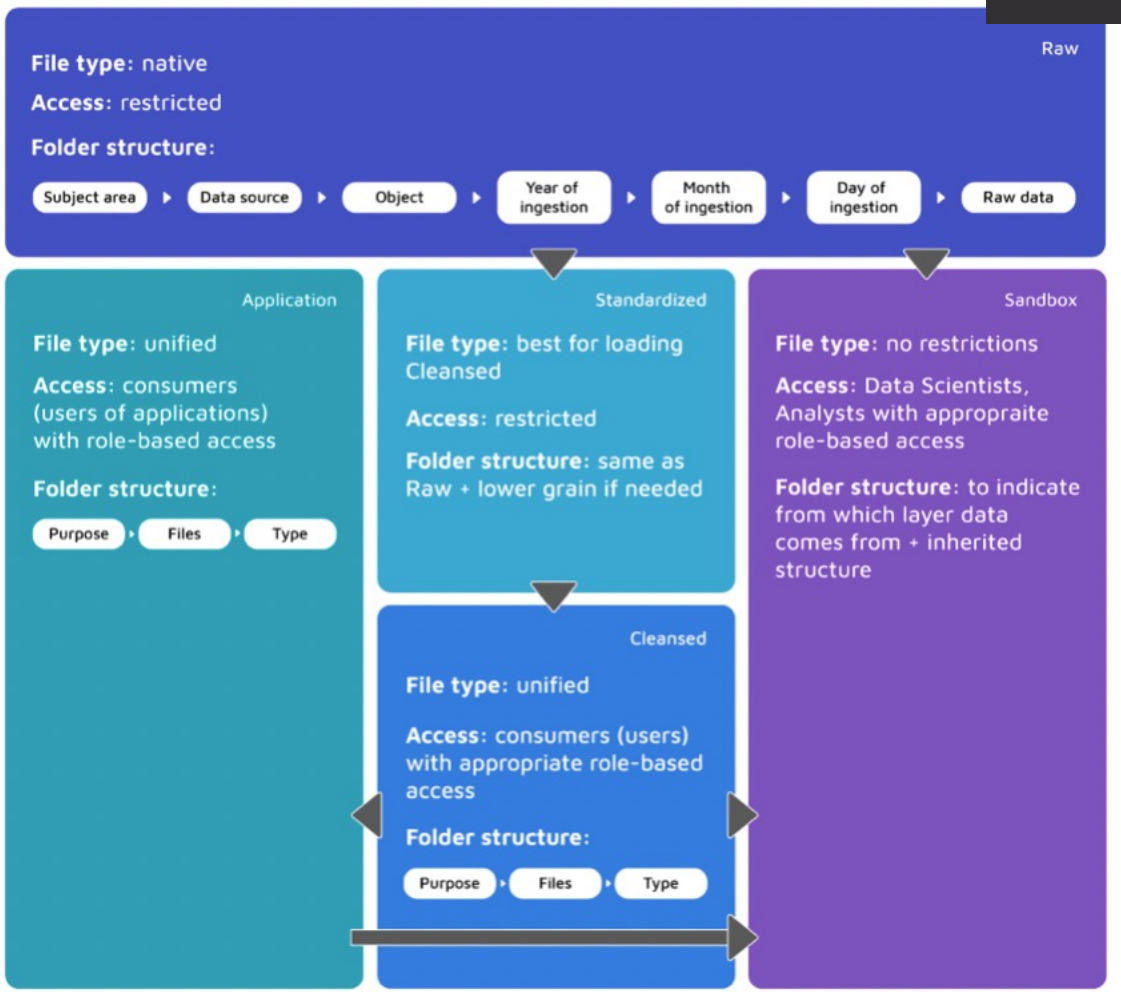
Apache Parquet – Created by Cloudera and twitter (may 2013)

Avro vs. Parquet vs. ORC



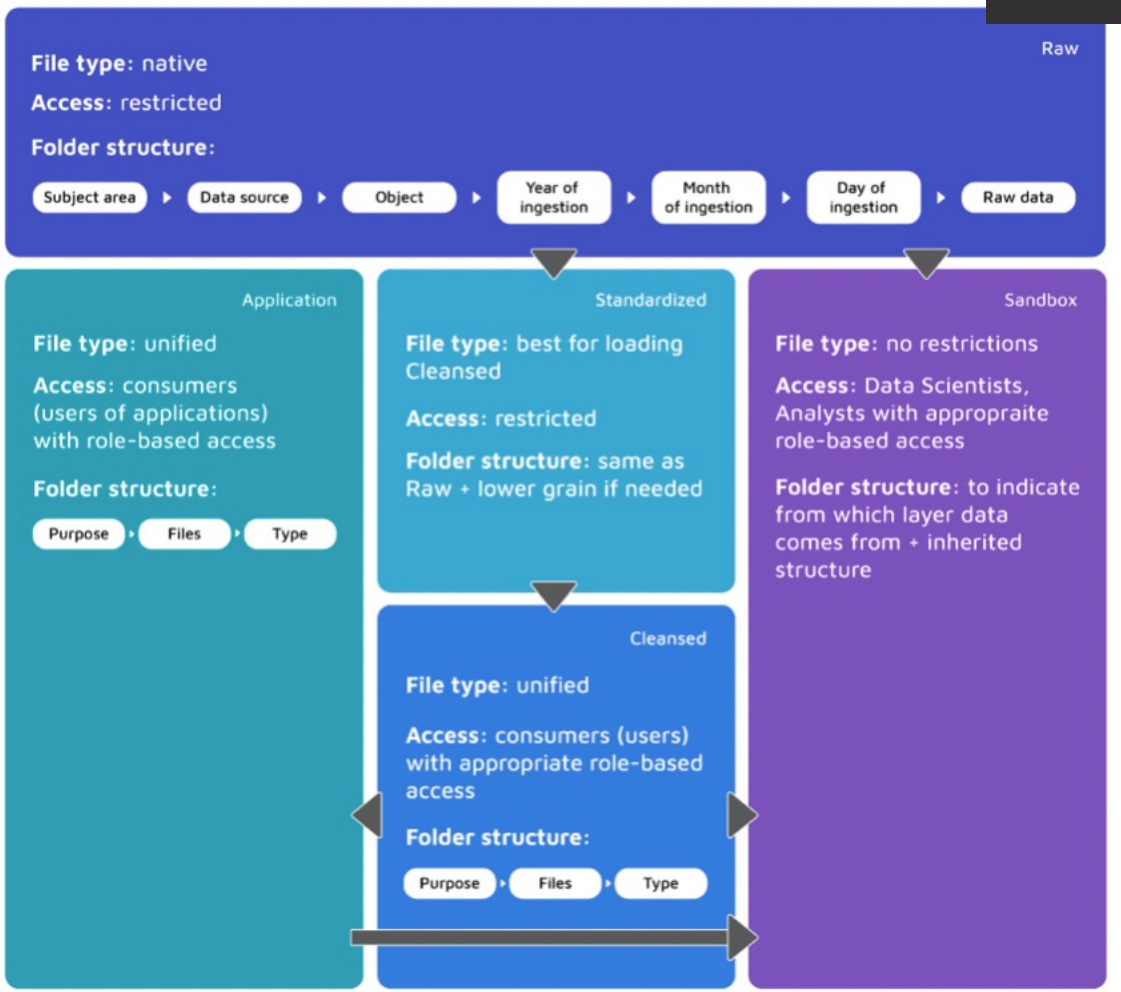
Organizing data Lake **layers**

- Data Lake is a “single” repository
- Separating them into layers
- Architecture of layers / zones
- Each layer represents additional data manipulation
- Layers:
  - Raw
  - Standardized
  - Cleaned
  - Application
  - Sandbox
- Alternatives:
  - Raw > Curated > Enriched
    - Sandbox, Machine Learning
  - Ingest > Curated > Conformed > Production
    - Sandbox, Personal, Reporting, Machine Learning
  - Bronze > Silver > Gold
    - Sanbox, Work, Machine Learning



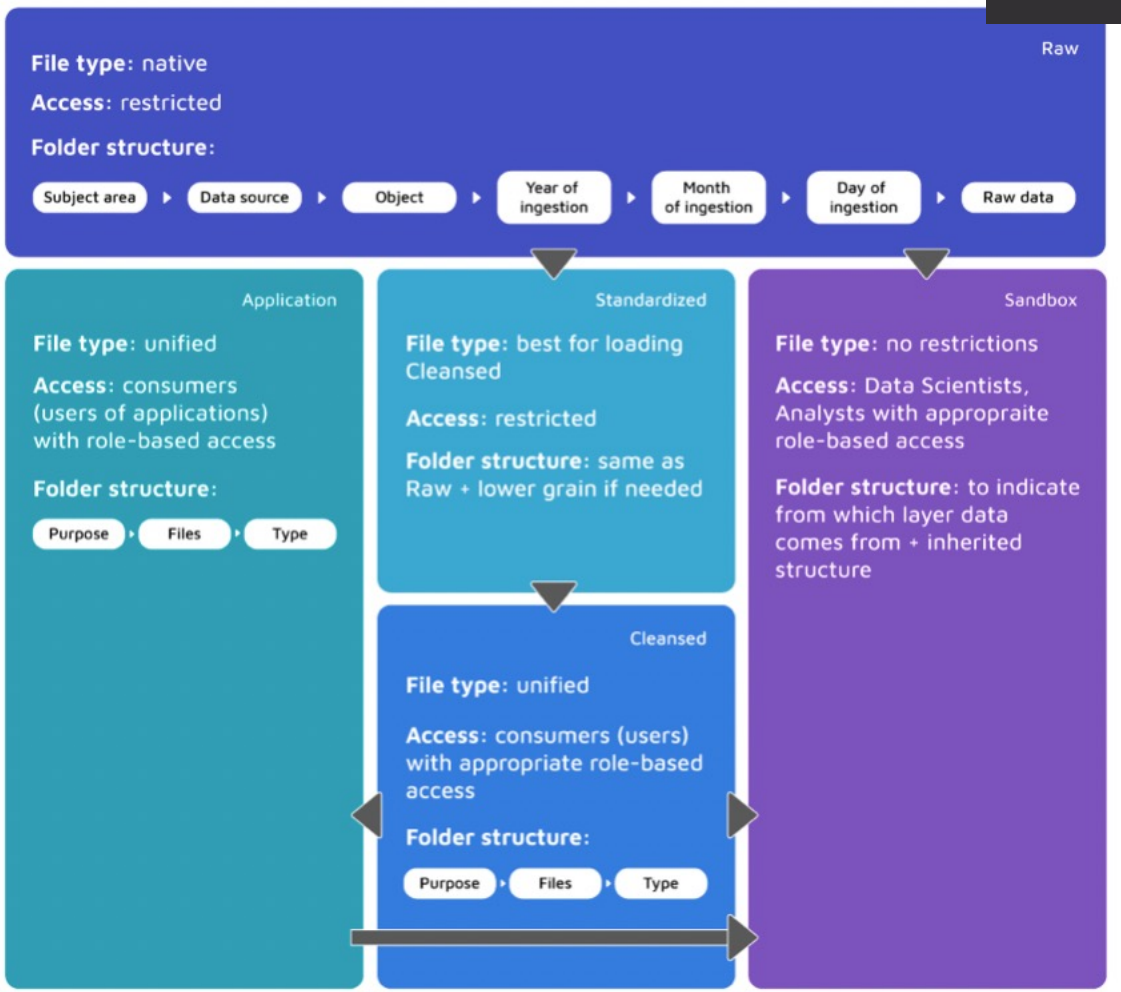
Organizing data Lake **folders**

- Folder structure within Layer
- Folder by purpose:
  - Area
  - Source
  - Object
  - >>**Key**<<
  - Raw data
- Folders by structure
  - Purpose
  - Files
  - Type



Organizing data Lake - **Keys**

- Keys are considered as “index”
- Keys help data organized
- Implement keys based on search
- Folder structure:
  - Year=2022
  - Year=2022&Month=03
  - GeoAreaID=100&Year=2022

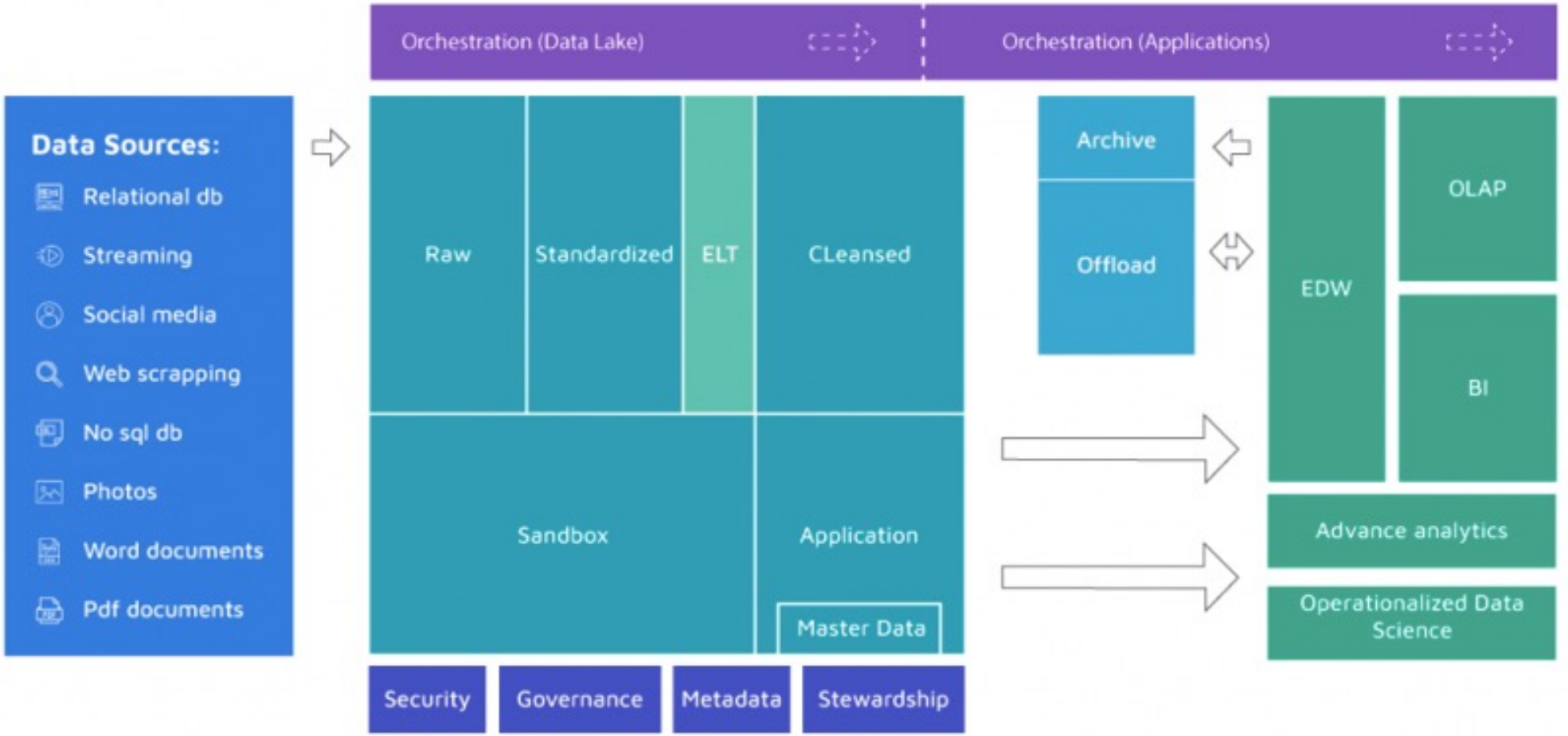


Files organization

Structure, type, access, granulation

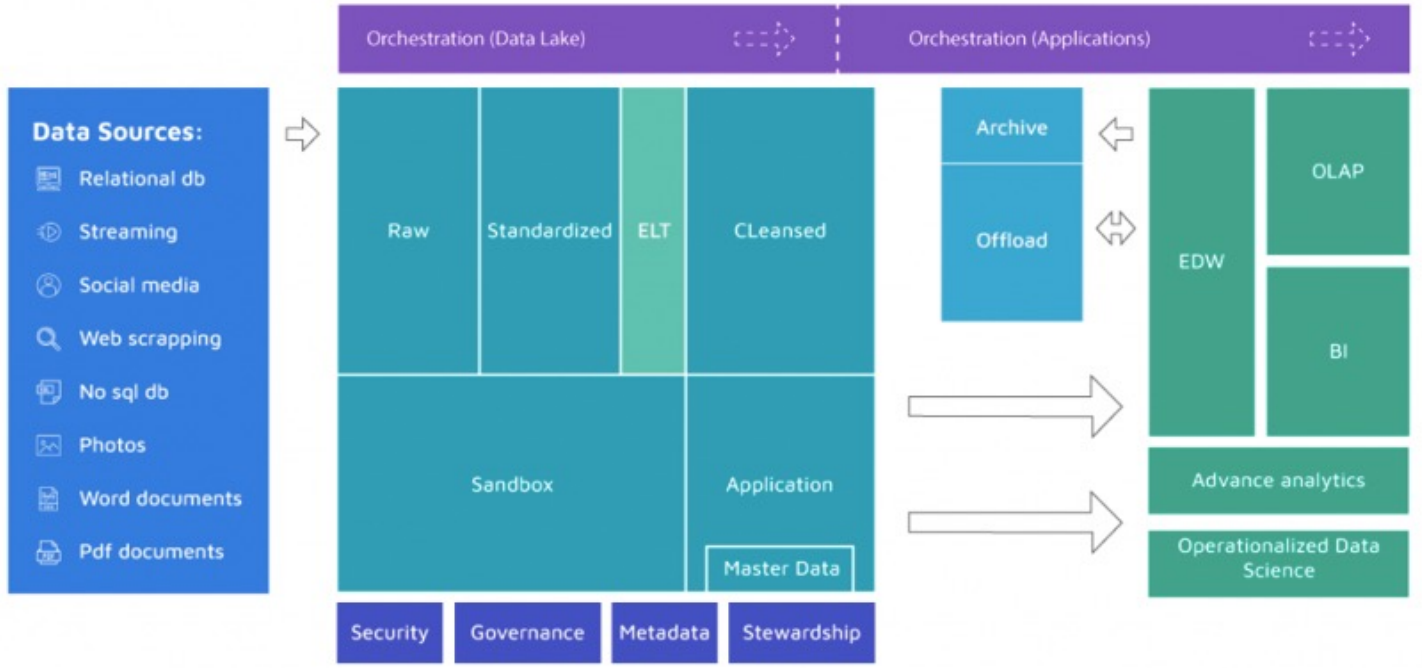
Folder structure	Data Types	Access control	Granulation (e.g.: stream data)
Raw layer	Original format	Data Engineers	As is
Standardized layer	Files aggregated, still original format	Data Engineers, Data Scientists	Aggregate to higher level (e.g. from sec -> min)
Cleaned layer	Sparsed (faster) format	Data Engineers, Data Scientists, Data Analysts	Another abstration layer above (e.g.: from sec -> hour)
Application layer	Custom to application	Data Engineers, Business people, Data Analysts	Higher granulation (e.g. Day, month, custom)

Data Lake Architecture



Data Lake Components

- Security
- Governance
- Metadata
- Stewardship
- Master data
- Archiving
- Orchestration

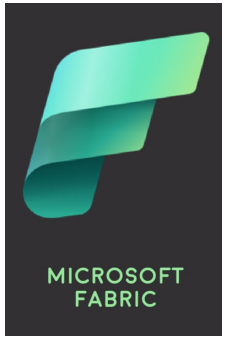


Architecture designs

Architecture designs can be influenced by:

- Data load patterns (hot/cold, real-time, stream, increment/full)
- Time partitioning
- Data type (relational, time series, blob)
- Subject areas and sources
- Security and roles
- Retention policies (temporary, permanent, time-fixed)
- Business impact (Core, Critical, High, Medium, Low)
- SLA and regional regulations
- GDPR and confidential classifications (public use, internal use, financial, Government, supplier/partner confidential)
- Workspaces (Fabric)

General concepts – Naming conventions



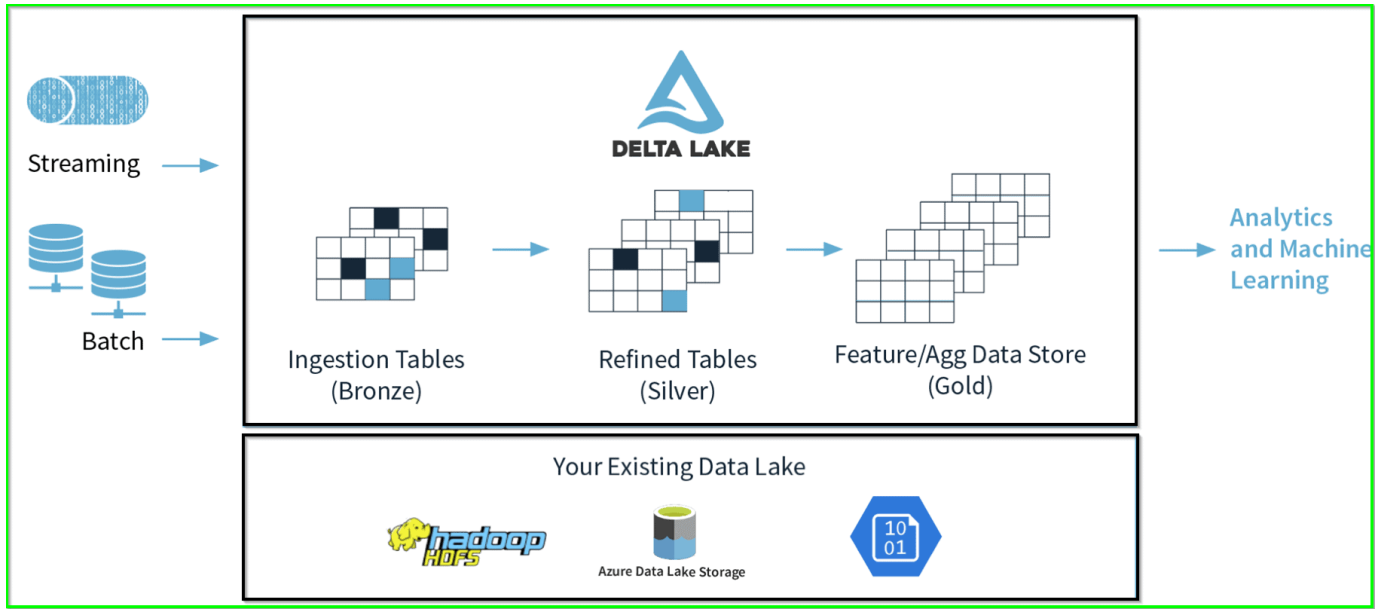
Keep in on organizational level

Granularity

RBAC access

Delta lake

- Open source data storage layer that unifies ACID transactic scalable metadata management and batch and/or streaming data processing. It also offers time travel and CDC, SCD, streaming upserts operations
- Sits on top of your existing Data Lake in tandem with Apache Spark APIs.

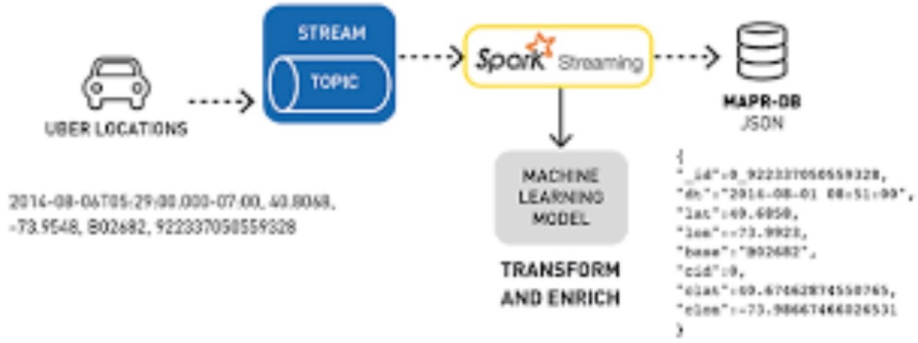


General concepts – Spark

- Fast, expressive, general-purpose in-memory cluster computing framework compatible with Apache Hadoop and built around speed, ease of use and streaming analytics
  Faster and easier than Hadoop MapReduce*
  Large community and 3rd party libraries
- Provides high-level APIs (Java, Scala, Python, R) Supports variety of workloads
  - interactive queries, streaming, machine learning and graph processing

Spark Use-Cases

- Logs processing (Uber)
- Event detection and real-time analysis
- Interactive analysis
- Latency reduction
- Advanced ad-targeting (Yahoo!)
- Recommendation systems (Netflix, Pinterest)
- Fraud detection
- Sentiment analysis (Twitter, X)

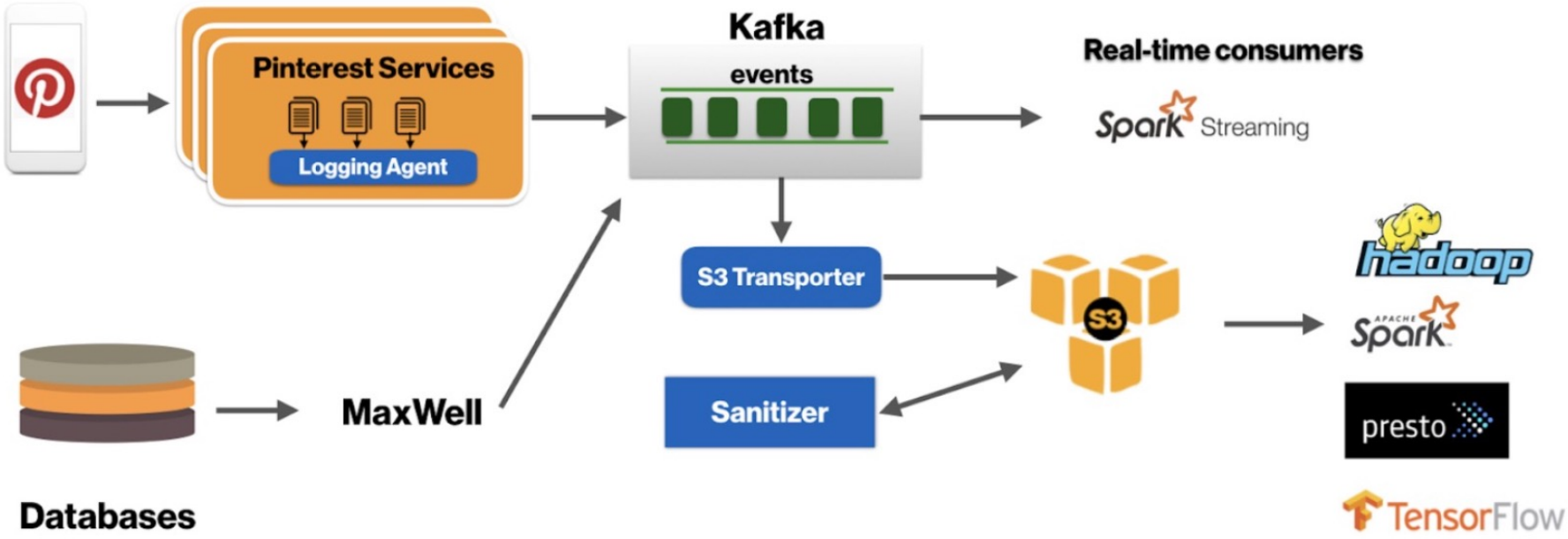


Spark Use-Cases

- Logs processing (Uber)
- Event detection and real-time analysis
- Interactive analysis
- Latency reduction
- Advanced ad-targeting (Yahoo!)
- Recommendation systems (Netflix, Pinterest)
- Fraud detection
- Sentiment analysis (Twitter)

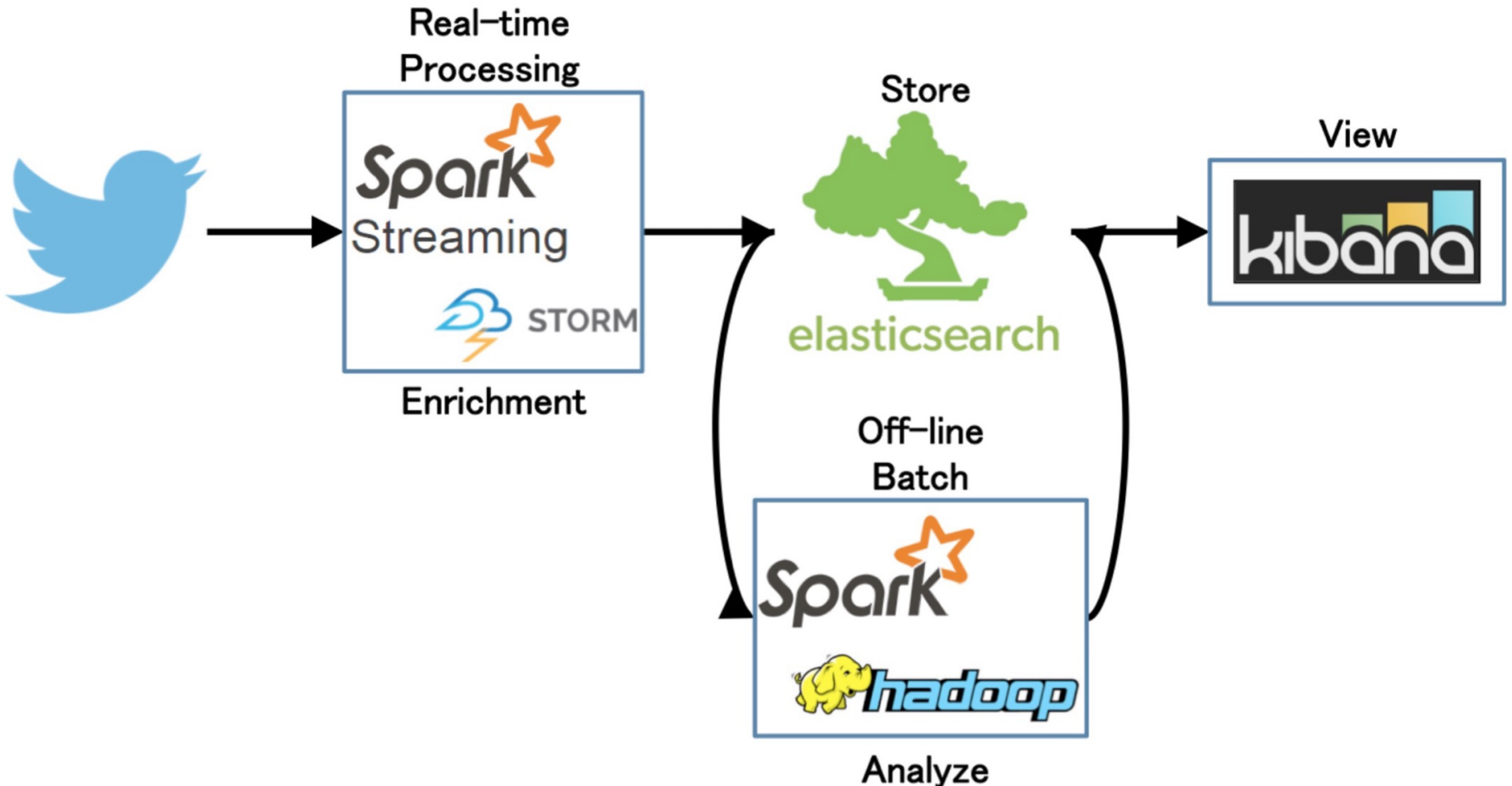
Spark Use-Cases

- Recommendation systems (Netflix, Pinterest)

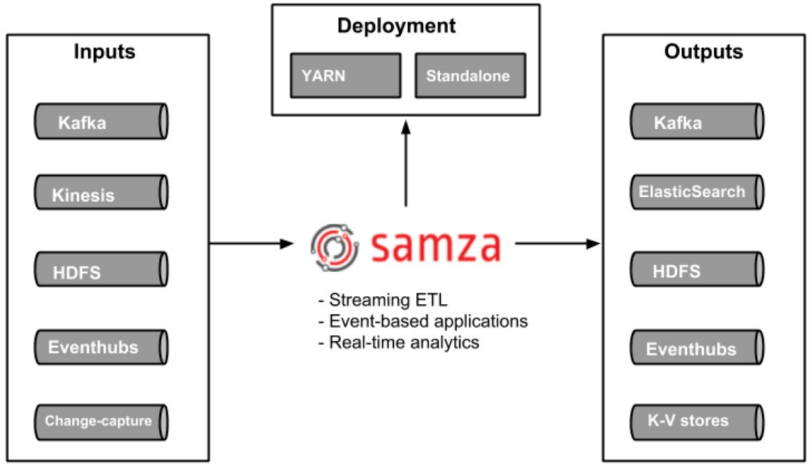


Spark Use-Cases

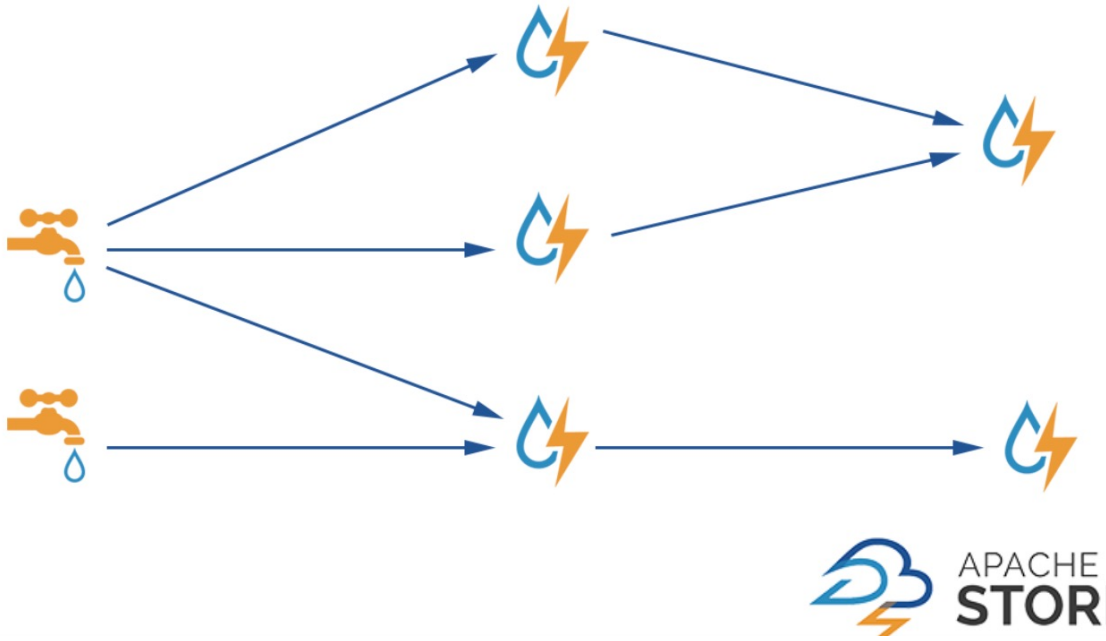
- Sentiment analysis (Twitter, X)

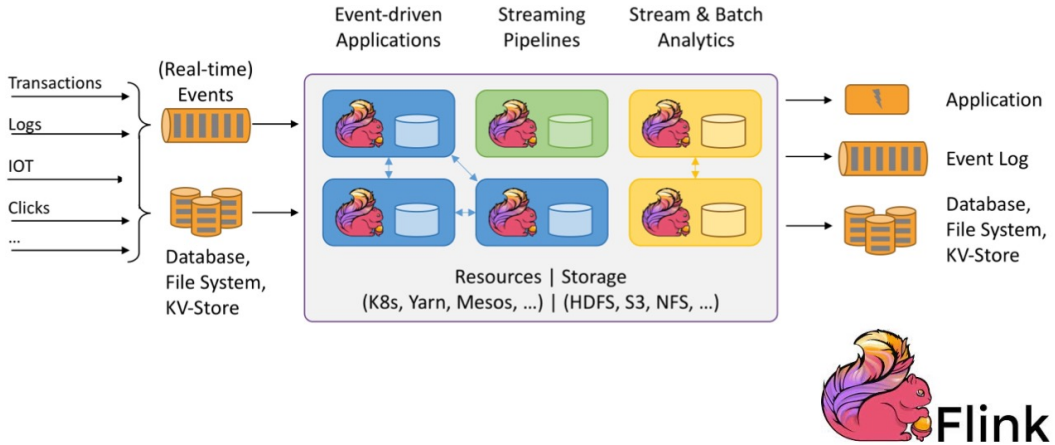


Apache Spark “real-time”



- Apache Samza (library/framework)
- Apache Storm (real-time stream processin
- Apache Flink (native streaming support for all workloads)





Apache Spark 3.x.x

- Latest stable release – v3.5.3. (Sep 24, 2024)
- Latest preview release – v4.0 (Sep 26, 2024)
- Improvements over Spark 2 (v2.4.1)
  - Python 2 deprecated
  - Adaptive execution of Spark SQL (merging intermediate results among workers)
  - Dynamic partition pruning
  - Support for deep learning (GPU support for Nvidia, AMD, Intel)
  - Better Kubernetes integration
  - Graph features (Morpheus as extension of Cypher, neo4j support)
  - ACID transactions (for Delta Lake storage)
  - Apache Arrow data format integration (columnar format for analytics)

Hadoop MapReduce vs. Apache Spark

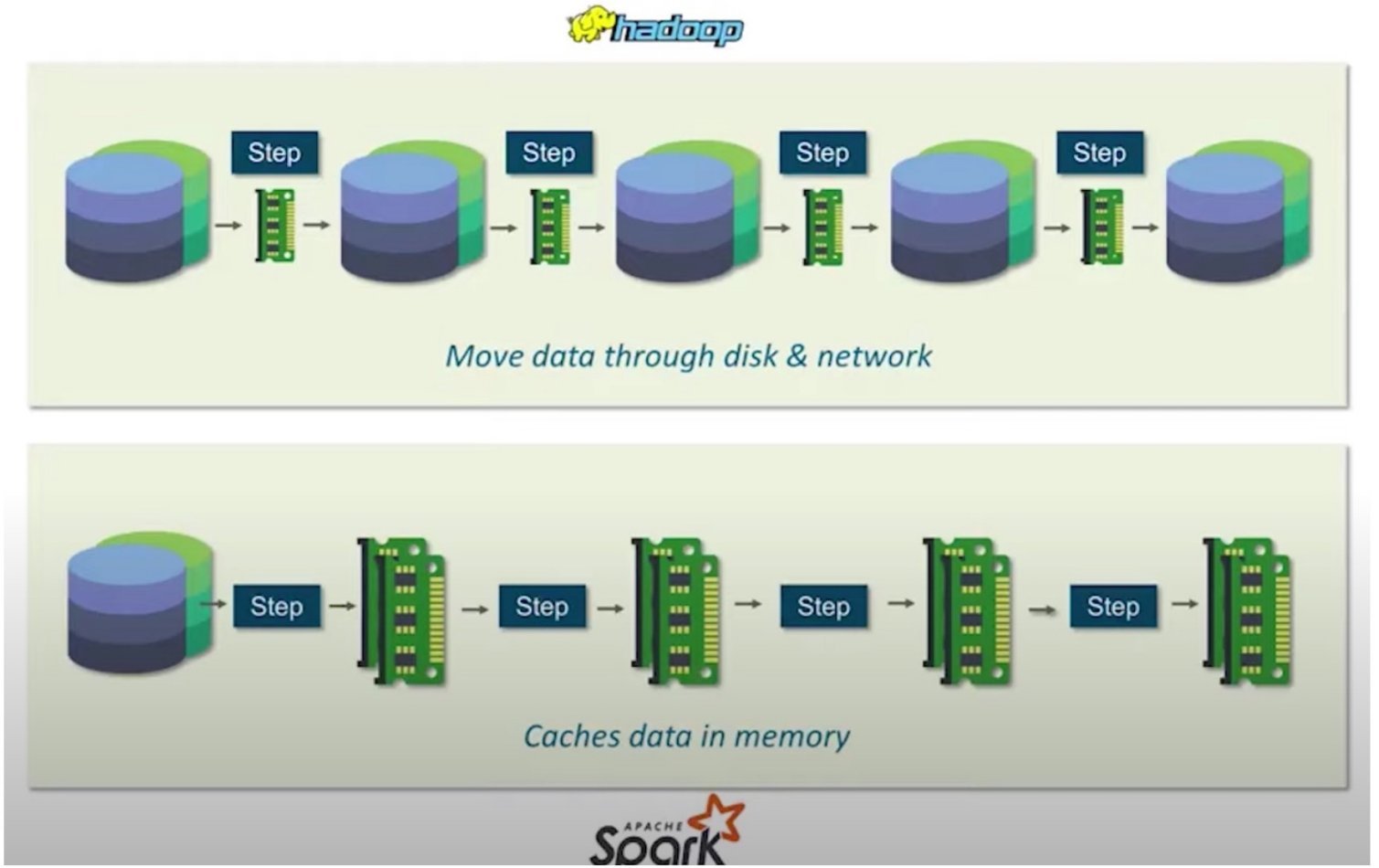
Big data frameworks
- Performance
- Ease of use
- Costs
- Data processing
- Fault tolerance
- Security

Hadoop

Archival data analysis

Spark

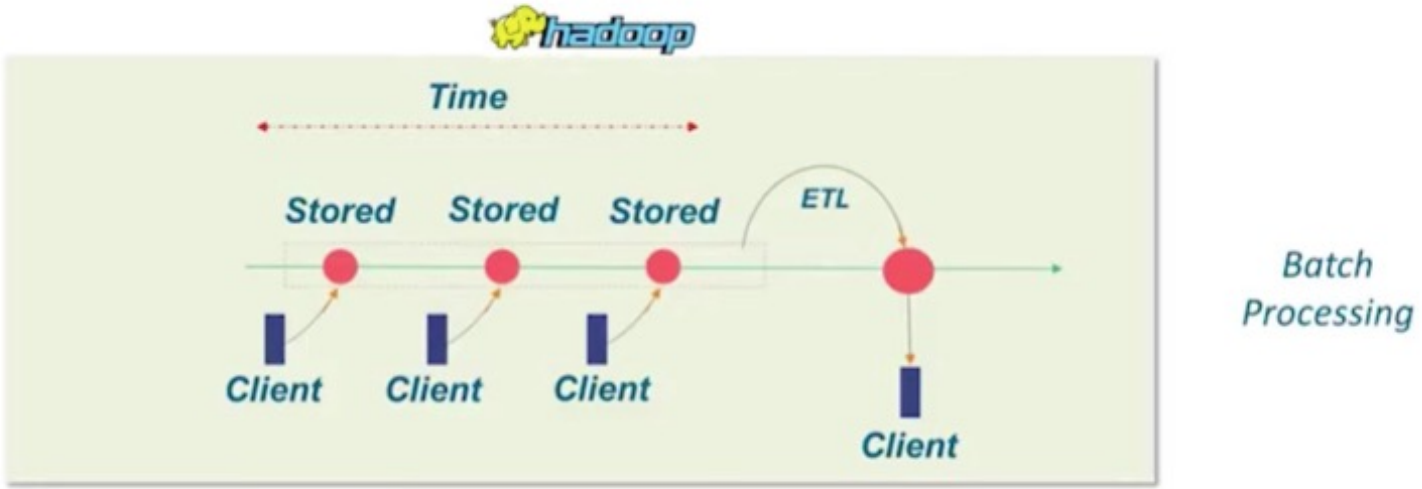
Real-time data analysis



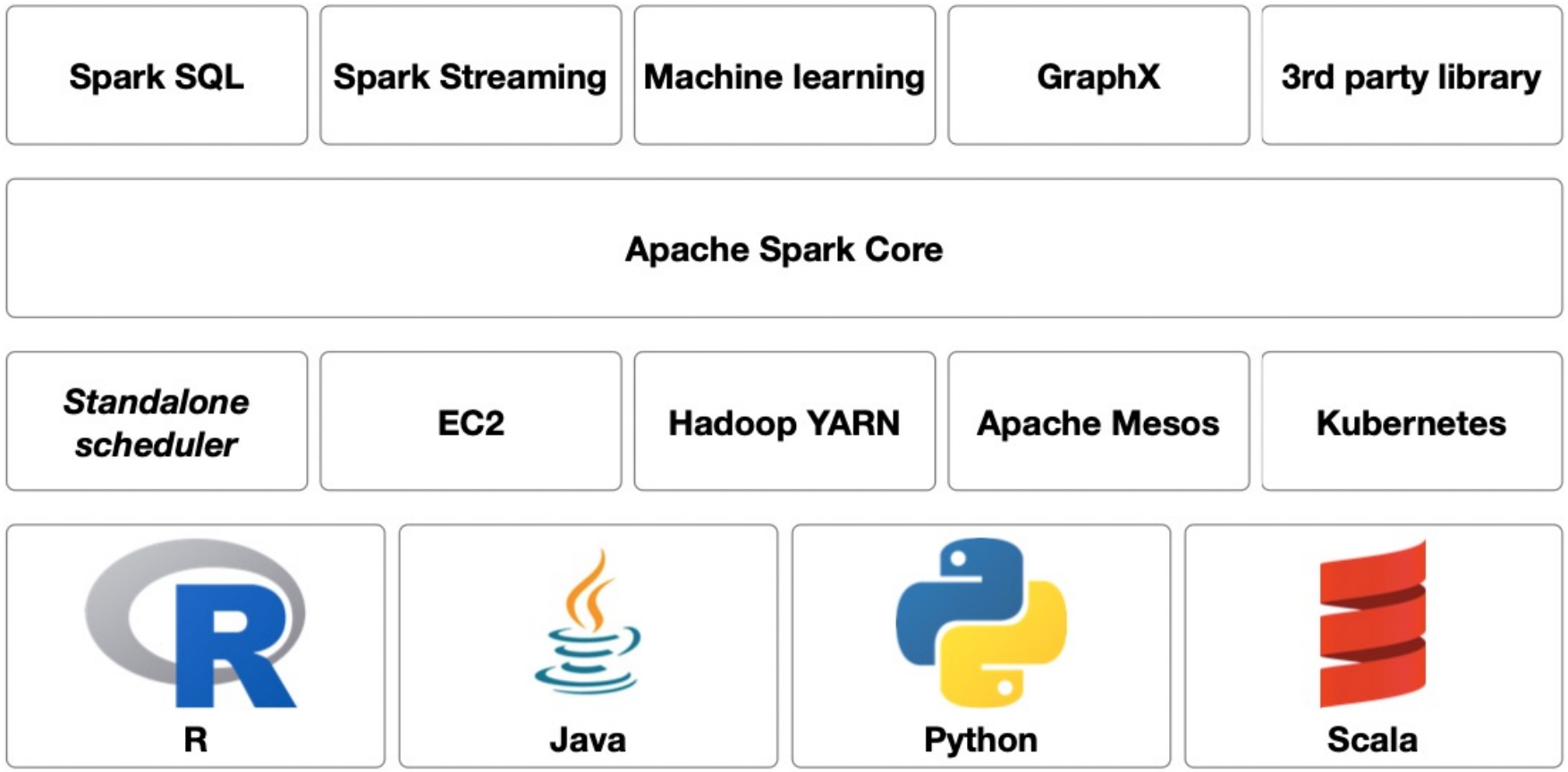
Hadoop MapReduce vs. Apache Spark

Hadoop MapReduce vs. Apache Spark

Hadoop MapReduce vs. Apache Spark



Apache Eco-System

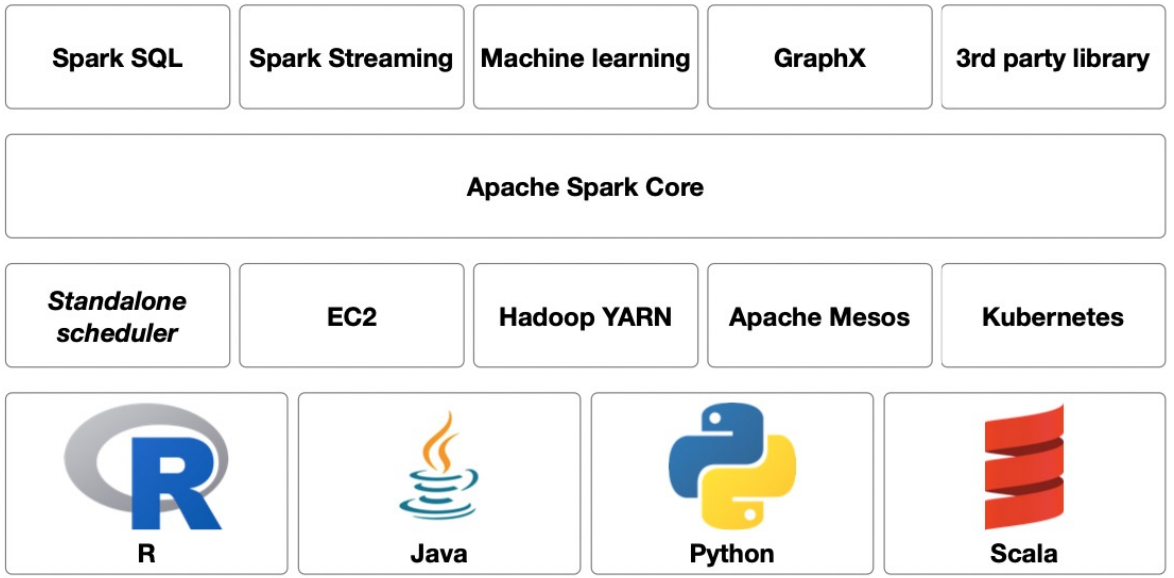


Spark Core functionalities

Core functionalities
- task scheduling
- memory management
- fault recovery
- storage systems interaction

etc.

Basic data structure definitions/abstractions
- Resilient Distributed Data sets (RDDs)
  main Spark data structure
- Directed Acyclic Graph (DAG)



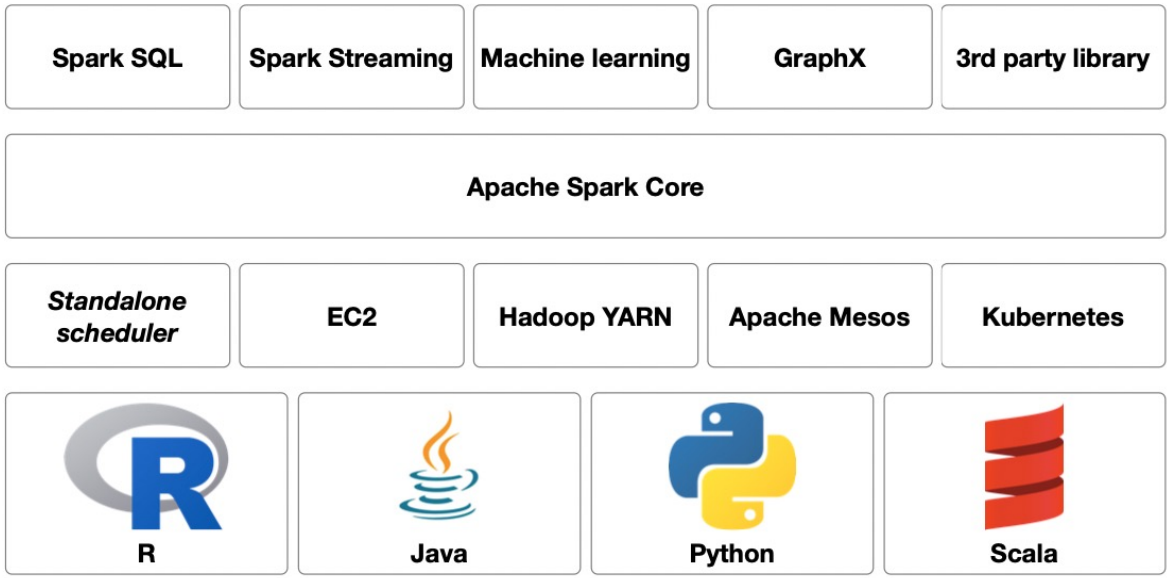
Ecosystem: Spark SQL

Structured data manipulation
- Data Frames definition

Table-like data representation
- RDDs extension
- Schema definition

SQL queries execution

Native support for schema-based data
- Hive, Paquet, JSON, CSV



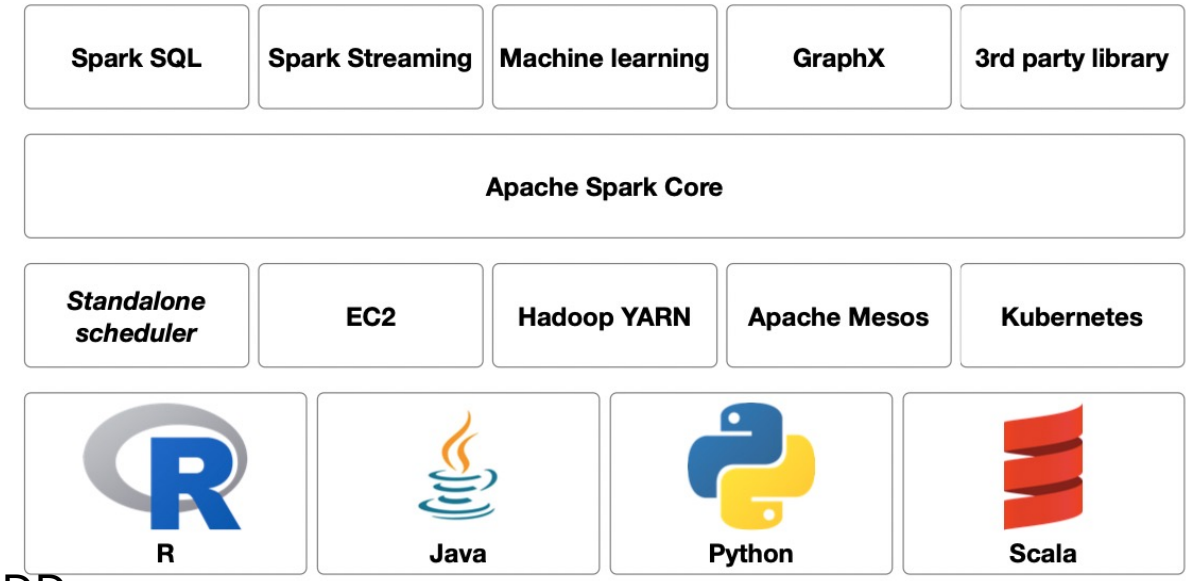
Ecosystem: Spark Streaming

Data analysis of streaming data
- e.g. tweets, log messages, SCADA

Features of stream processing
- High-throughput
- Fault-tolerant
- End-to-end
- Exactly once

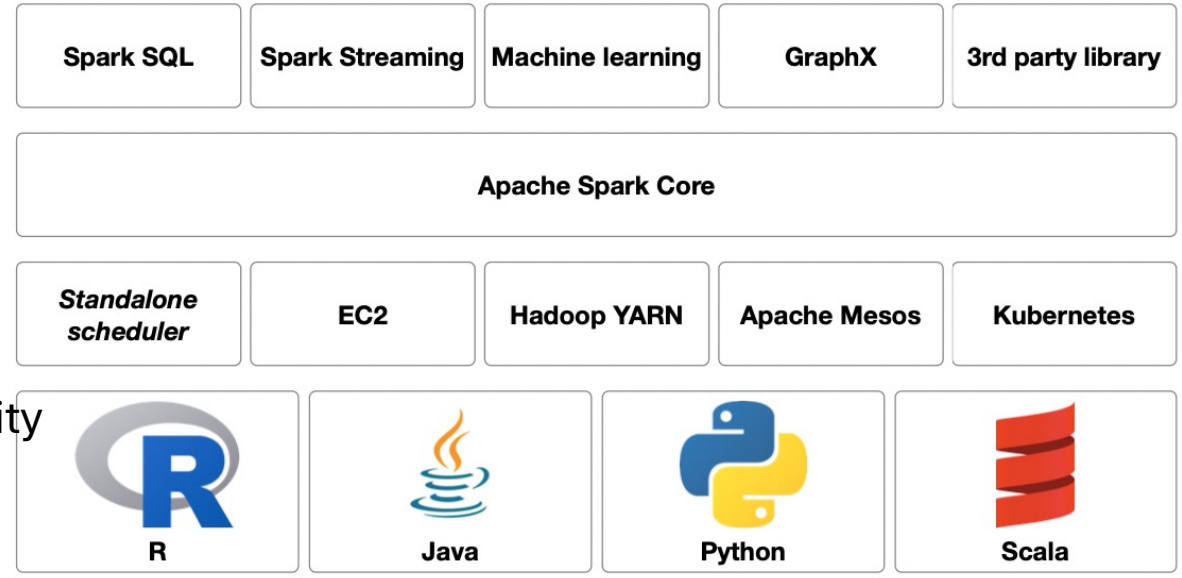
High-level abstraction of a discretized stream
- Stream represented as a sequence of RDD

With Spark 2.3.x and above continuous processing
- End-to-end low latency (< 1ms)



Ecosystem: Spark MLlib for Machine Learning

- Common ML functionalities
  - ML Algorithms
    - common learning algorithms such as classification, regression, clustering, and collaborative filtering
  - Featurization
    - feature extraction, transformation, dimensionality reduction, and selection
  - Pipelines
    - tools for constructing, evaluating, and tuning ML Pipelines
  - Persistence
    - saving and load algorithms, models, and Pipelines
  - Utilities
    - linear algebra, statistics, data handling, etc.
- Two APIs
  - RDD-based API (*spark.mllib package*)
  - Spark 2.0+, DataFrame-based API (*spark.ml package*)
- Methods scale out across the cluster by default



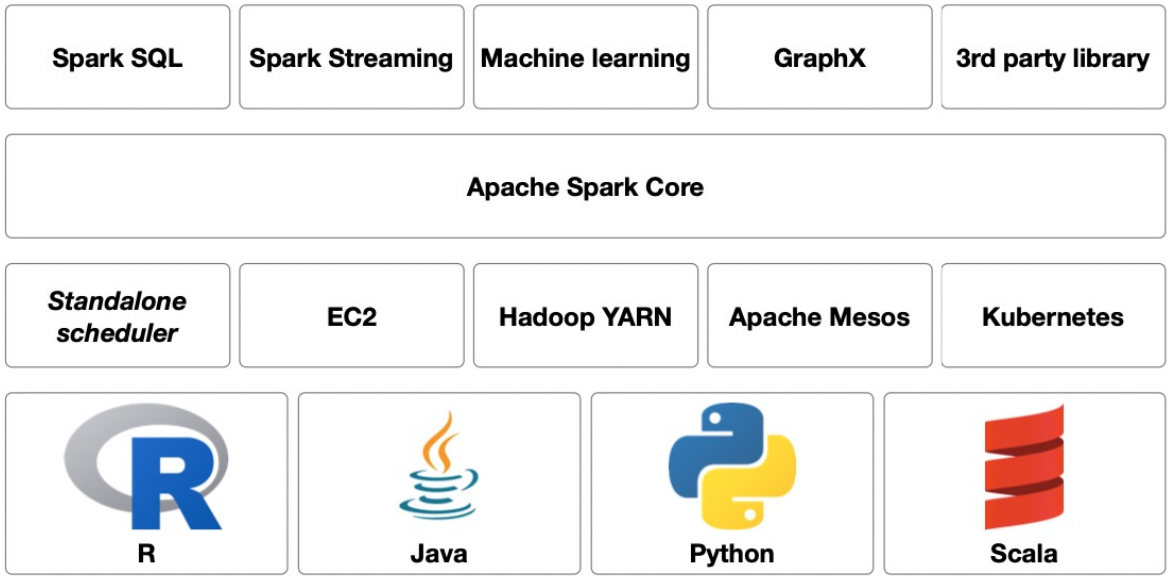
Ecosystem: GraphX

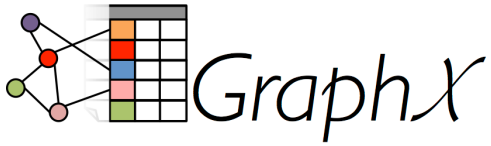
Support for graphs and graph-parallel computation

- Extension of RDDs (Graph)
- Direct multigraph with properties on vertices and edges

Graph computation operators

- subgraph, joinVertices, and aggregateMessages, etc.
- Pregel API support





Spark Modes

Spark operates in 4 different modes:

- **Standalone Mode:** Here all processes run within the same JVM process.
- **Standalone Cluster Mode:** In this mode, it uses the Job-Scheduling framework in-built in Spark.
- **Apache Mesos:** In this mode, the work nodes run on various machines, but the driver runs only in the master node.
- **Hadoop YARN:** In this mode, the drivers run inside the application's master node and is handled by YARN on the Cluster.

Spark Context

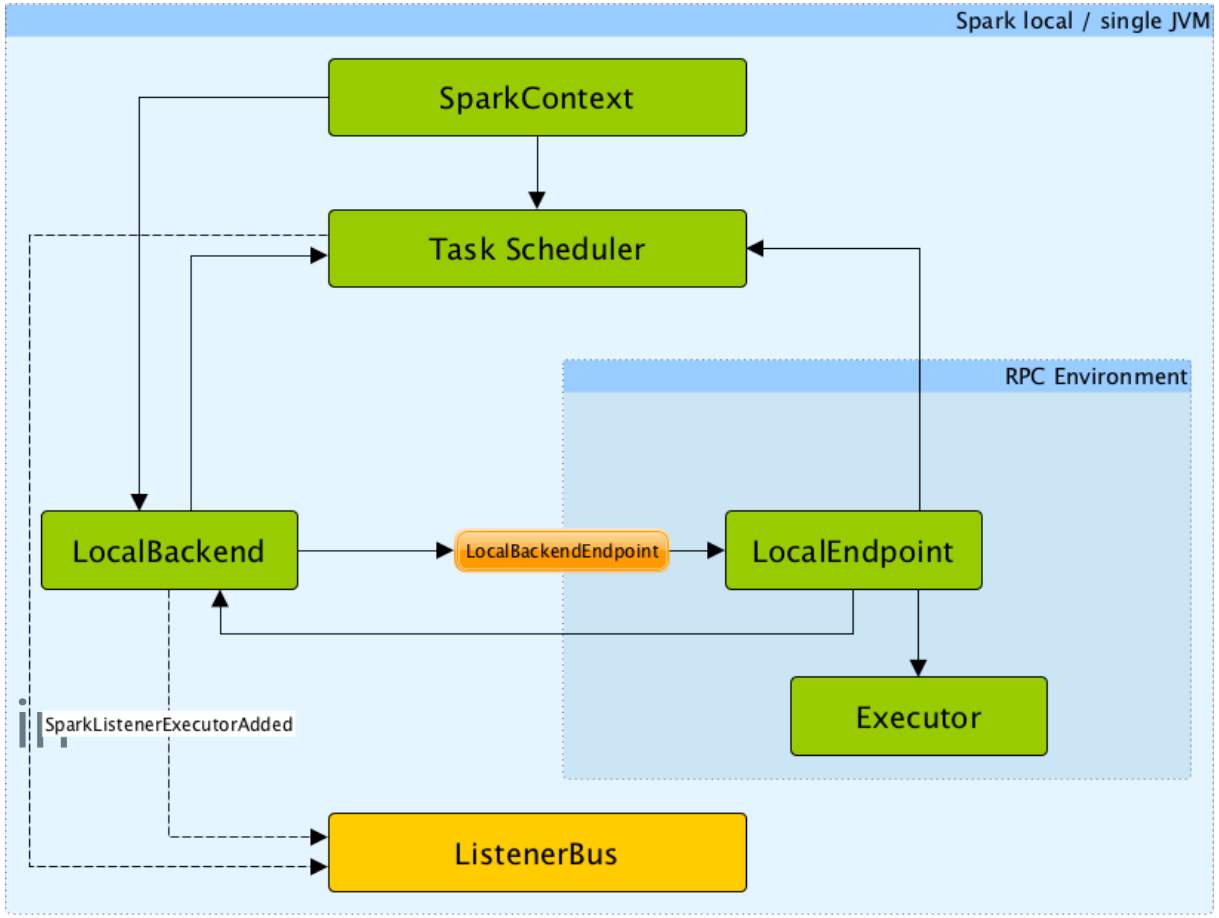
- driver Spark application which communicates the user commands to the Spark Workers
- `SparkContext` is an object, which coordinates with the cluster manager about the resources required for execution and the actual tasks that need to be executed
- Submitting spark applications: Submitting Applications - Spark 3.2.1 Documentation (apache.org)

Execution modes

- Local mode
  - „Pseudo-cluster“ ad-hoc setup using script
- Cluster mode
  - Running via cluster manager
- Interactive mode
  - Direct manipulation in a shell (*pyspark*, *spark-shell*)

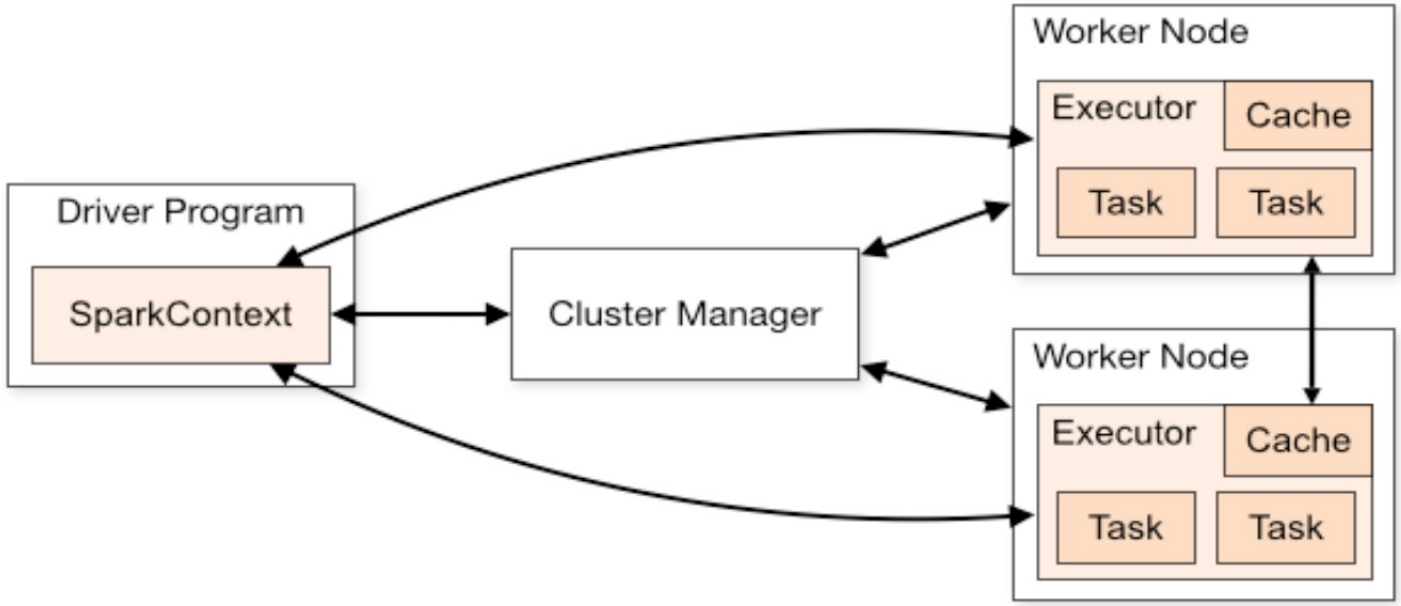
Spark execution modes Local mode

- Non-distributed single-JVM deployment mode
- Spark library spawns (in a JVM)
  - driver
  - scheduler
  - master
  - executor
- Parallelism is the number of threads defined by a parameter N a spark master URL
  - *local[N]*

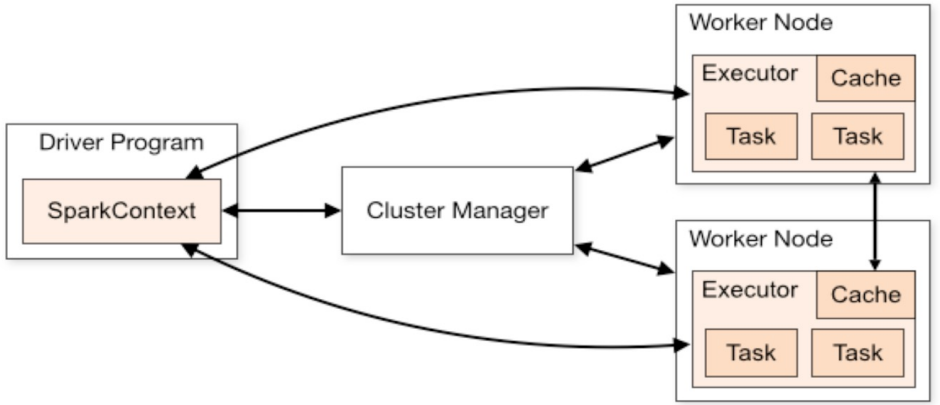


Spark execution modes Cluster mode

- Deployment on a private cluster
  - Apache Mesos
  - Hadoop YARN
  - Kubernetes
  - Standalone mode, …

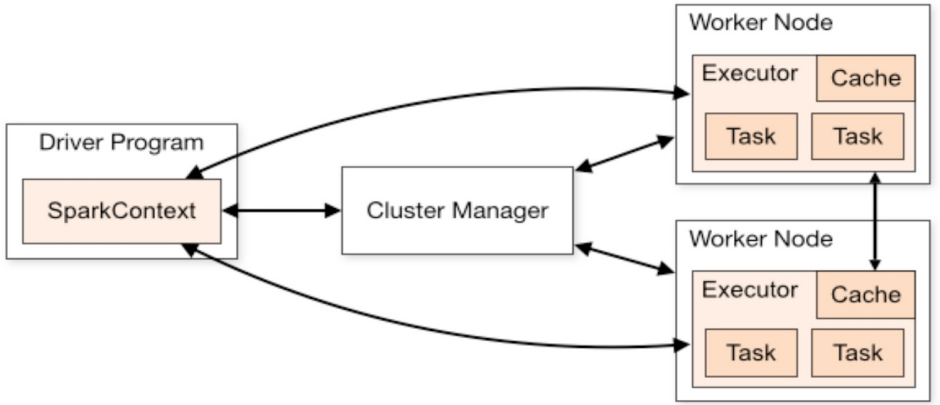


Spark execution modes Cluster mode



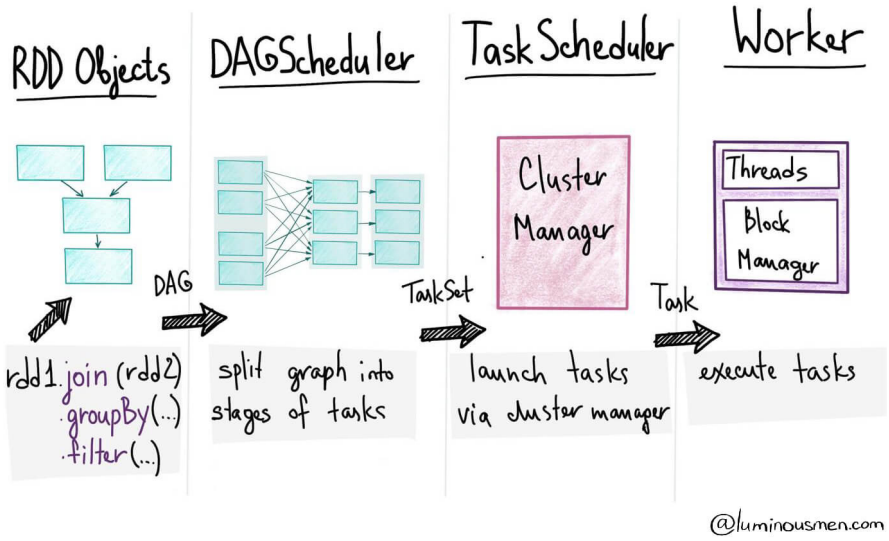
- Components
  - Worker
    - Node in a cluster, managed by an executor
    - Executor manages computation, storage and caching
  - Cluster manager
    - Allocates resources via SparkContext with Driver program
  - Driver program
    - A program holding SparkContext and main code to execute in Spark
    - Sends application code to executors to execute
    - Listens to incoming connections from executors

Spark execution modes Cluster mode



- Deploy modes (*standalone clusters*)
  - Client mode (default)
    - Driver runs in the same process as client that submits the app
  - Cluster mode
    - Driver launched from a worker process
    - Client process exits immediately after application submission

Spark execution process



1. Data preparation/import
   - RDDs creation – i.e. parallel dataset with partitions
2. Transformations/actions definition*
   - Creation of tasks (units of work) sent to one executor
   - Job is a set of tasks executed by an action*
3. Creation of a directed acyclic graph (DAG)
   - Contains a graph of RDD operations
   - Defition of stages – set of tasks to be executed in parallel (i.e. at a partition level)
4. Execution of a program (application)

Shell vs.Cluster mode

- Shell Is interactive
- Cluster is for connecting into private network with several machines
- In cluster mode, it’s called submitting applications

Submitting applications

Pseudo code

./bin/spark-submit ****
--class <*main-class*> ****
--master <*master-url*> ****
--deploy-mode <*deploy-mode*> ****
--conf <*key*>=<*value*> ****
... # other options <application-jar> ****
[application-arguments]

Applications: Locally, in Standalone cluster

```
# Run application locally on 8 cores
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master local[8] \
  /path/to/examples.jar \
  100
```

```
# Run a Python application on a Spark standalone cluster
./bin/spark-submit \
  --master spark://207.184.161.138:7077 \
  examples/src/main/python/pi.py \
  1000
```

Applications: Mesos and Kubernetes

```
# Run on a Mesos cluster in cluster deploy mode with supervise
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master mesos://207.184.161.138:7077 \
  --deploy-mode cluster \
  --supervise \
  --executor-memory 20G \
  --total-executor-cores 100 \
  http://path/to/examples.jar \
  1000
```

```
# Run on a Kubernetes cluster in cluster deploy mode
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master k8s://xx.yy.zz.ww:443 \
  --deploy-mode cluster \
  --executor-memory 20G \
  --num-executors 50 \
  http://path/to/examples.jar \
  1000
```

Module 2

Covering data storage and compute offerings.

For Microsoft Fabric we will cover the outstanding OneLake data store, learn how it works, what is is capable of and how to get navigate.

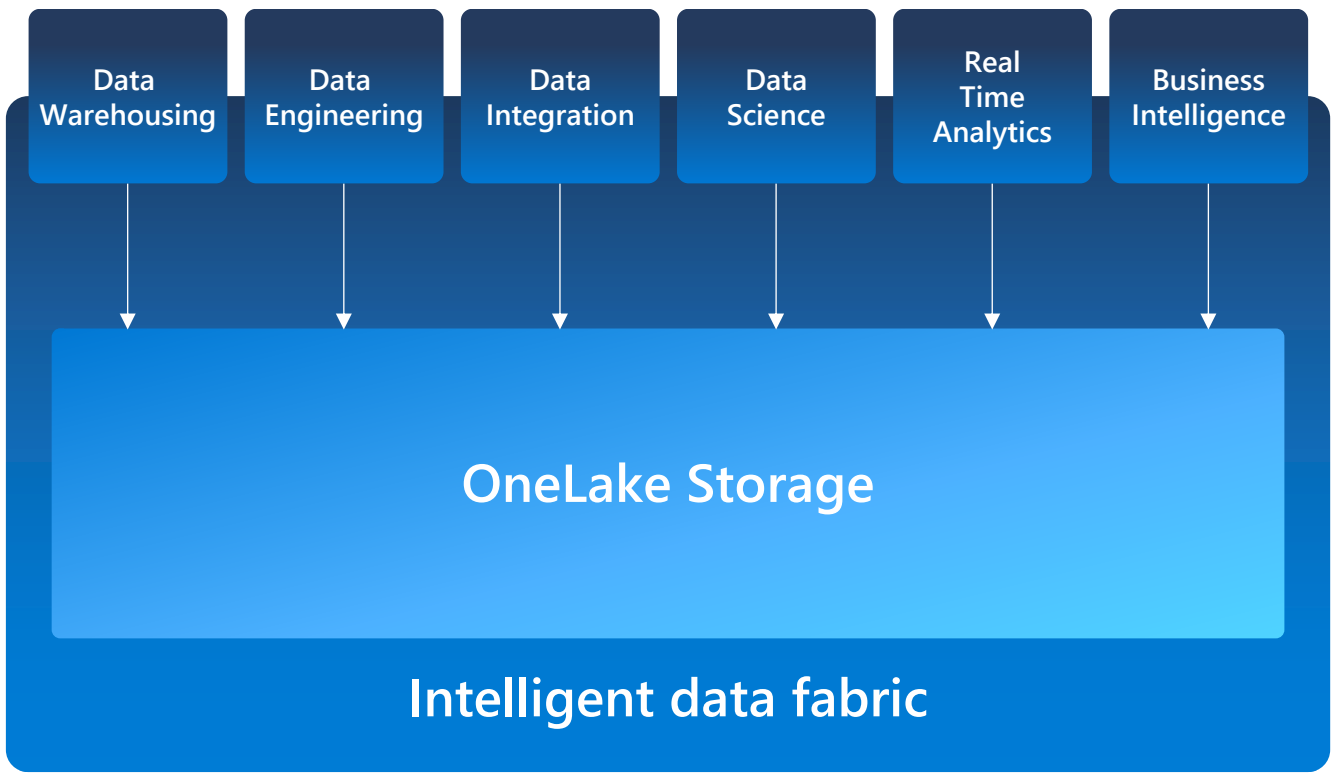
In Azure Machine Learning we will look into different Azure data store options like ADSL Gen2, and learn about creating datastores and datasets.

Concepts

- Distributed Computation
- One Lake
- Compute

OneLake for all Data

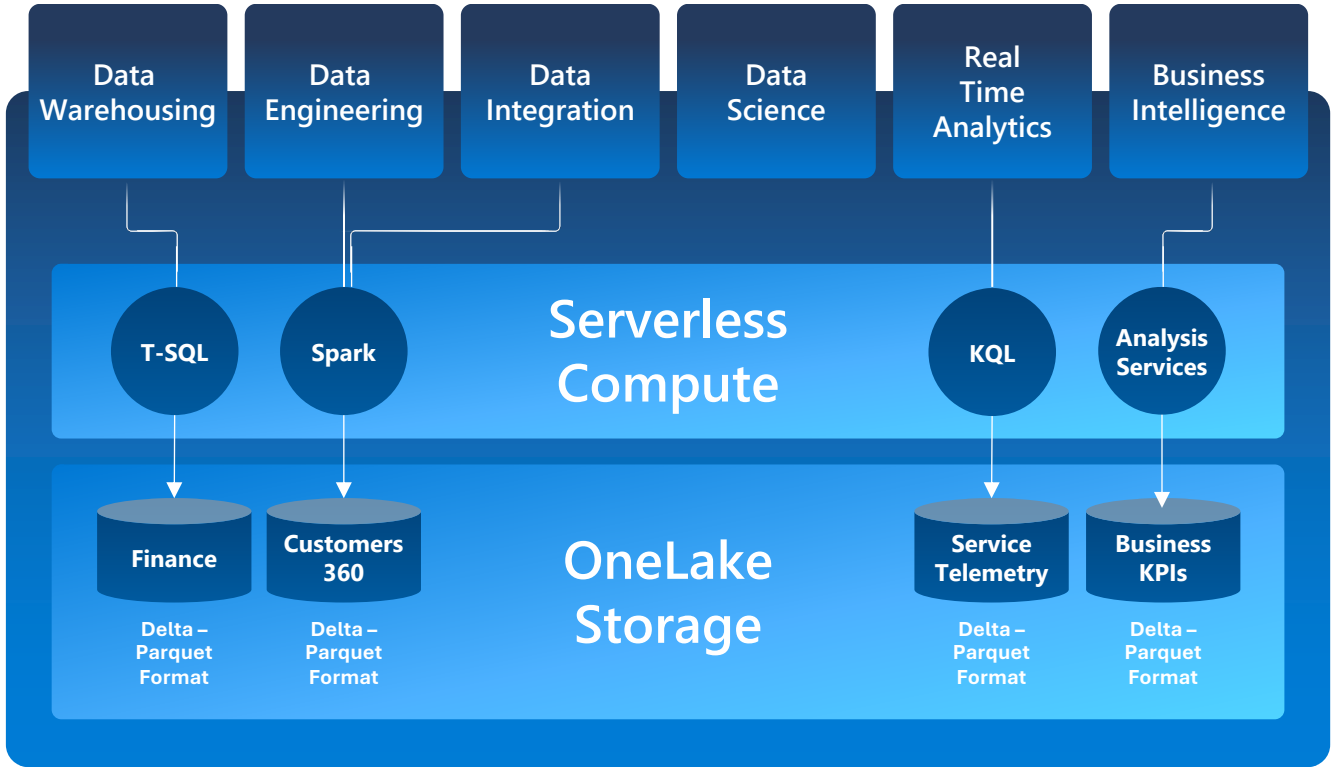
"The OneDrive for Data"



- A single SaaS lake for the whole organization
- Provisioned automatically with the tenant
- All workloads automatically store their data in the OneLake workspace folders
- All the data is organized in an intuitive hierarchical namespace
- The data in OneLake is automatically indexed for discovery, MIP labels, lineage, PII scans, sharing, governance and compliance

One Copy for all computes

Real separation of compute and storage



All the compute engines store their data automatically in OneLake

The data is stored in a single common format

Delta – Parquet, an open standards format, is the storage format for all tabular data in Fabric

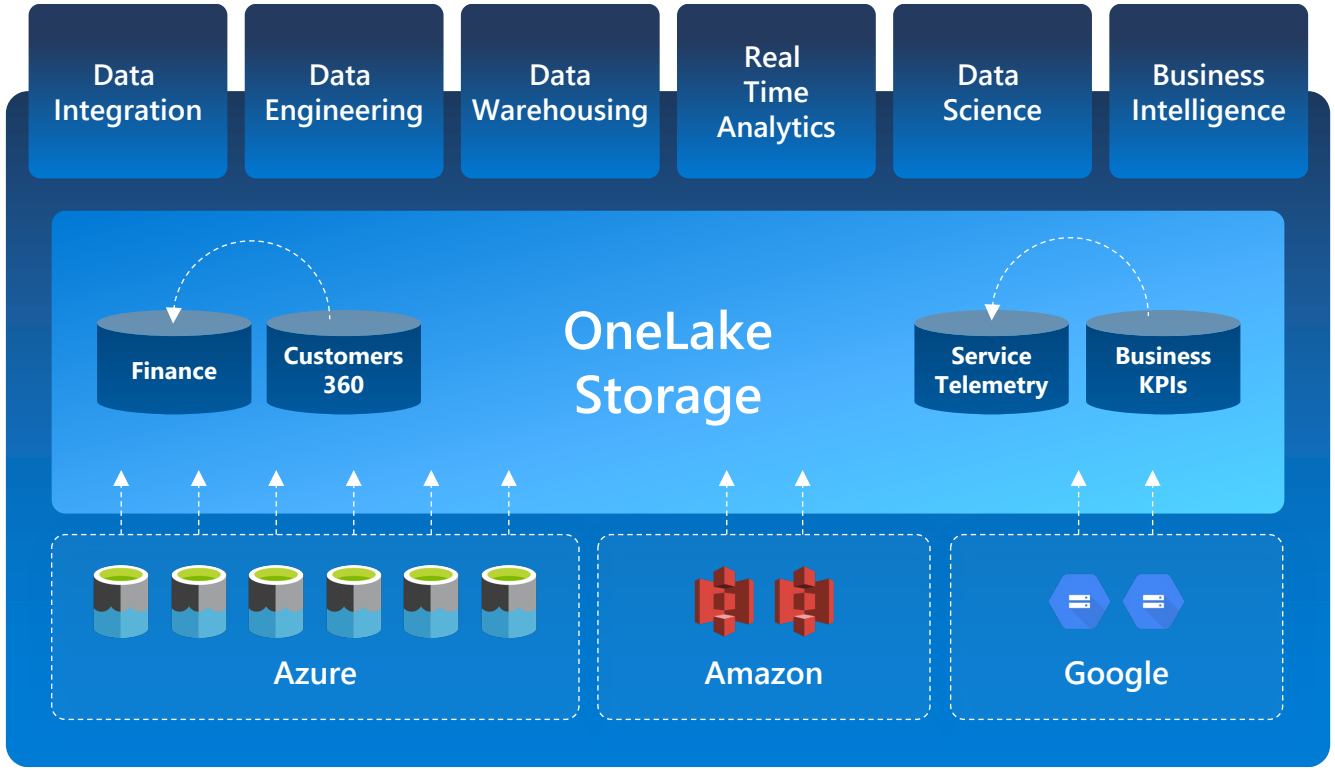
Once data is stored in the lake, it is directly accessible by all the engines without needing any import/export

All the compute engines have been fully optimized to work with Delta Parquet as their native format

Shared universal security model is enforced across all the engines

Taking One Copy to the next level

Shortcuts



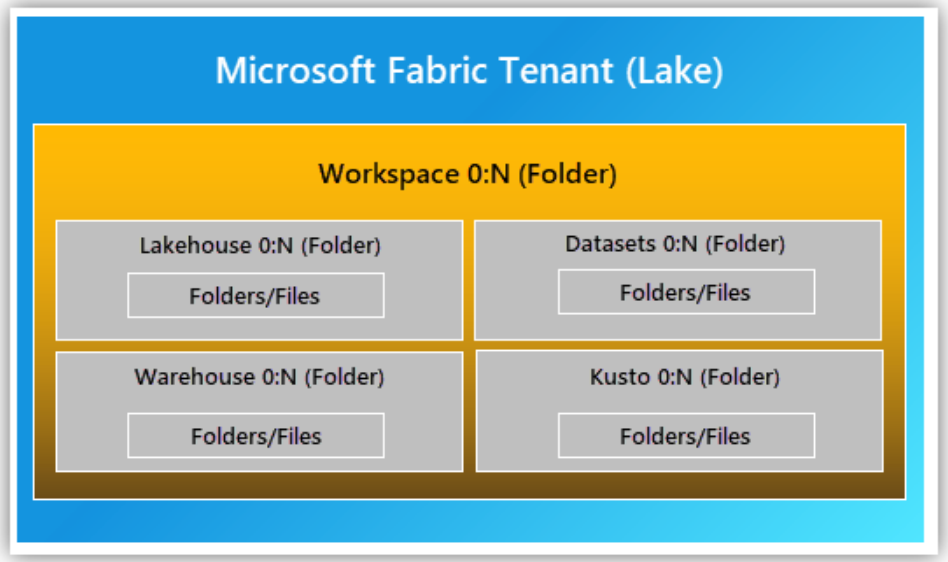
Sharing data in OneLake is as easy as sharing files in OneDrive, removing the needs for data duplication

With shortcuts, data throughout OneLake can be composed together without any data movement

Shortcuts also allow instant linking of data already existing in Azure and in other clouds, without any data duplication and movement, making OneLake the first multi-cloud data lake

With support for industry standard APIs, OneLake data can be directly accessed by any application or service

OneLake organisation



- is hierarhical and there is no

need to up-front provisioning

- one per tenant and single file

system namespace across users and regions

- divided into manageble containers

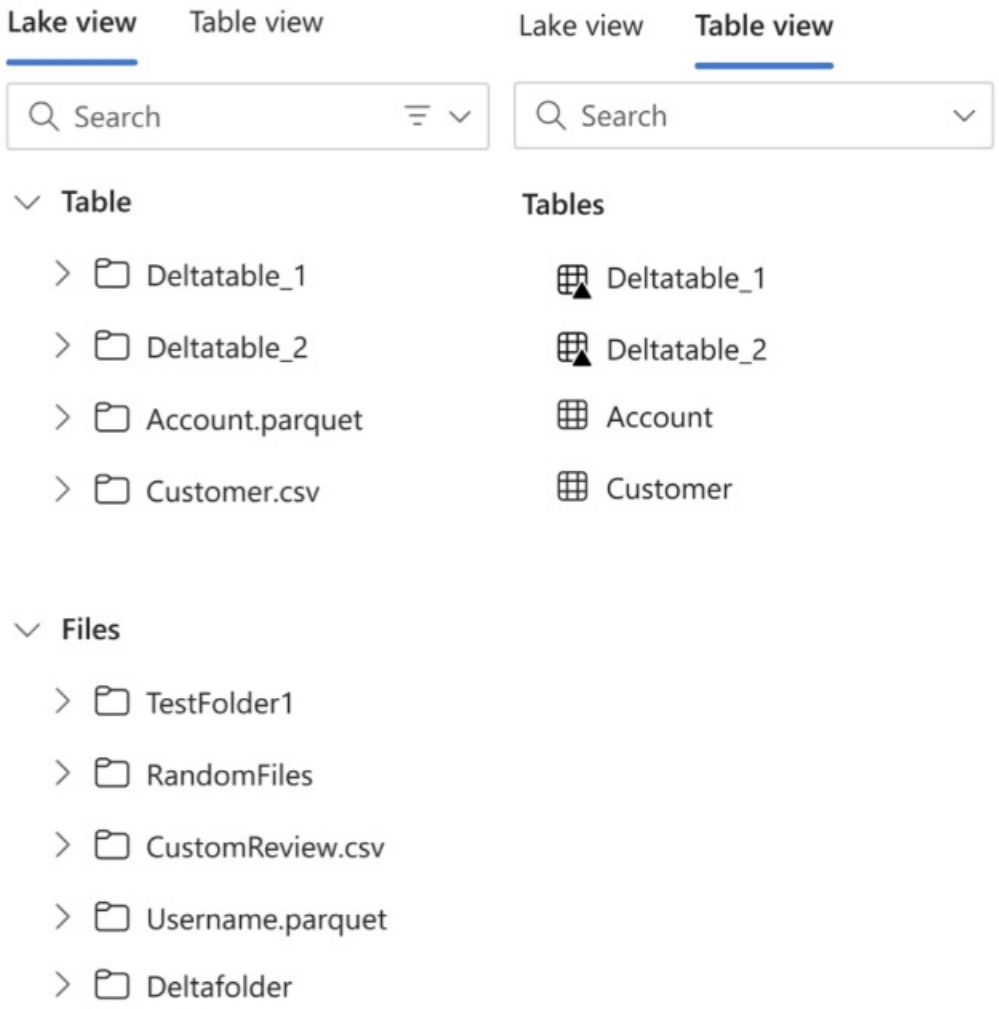
- user can create n-number of workspaces within tenant (WS can be understood as folder)

- every developer / business unit can create WS and ingest, process and analyse data, automatically the compute will be prewired to this WS

Use Shortcut feature to access data in Azure Data Lake storage without migration

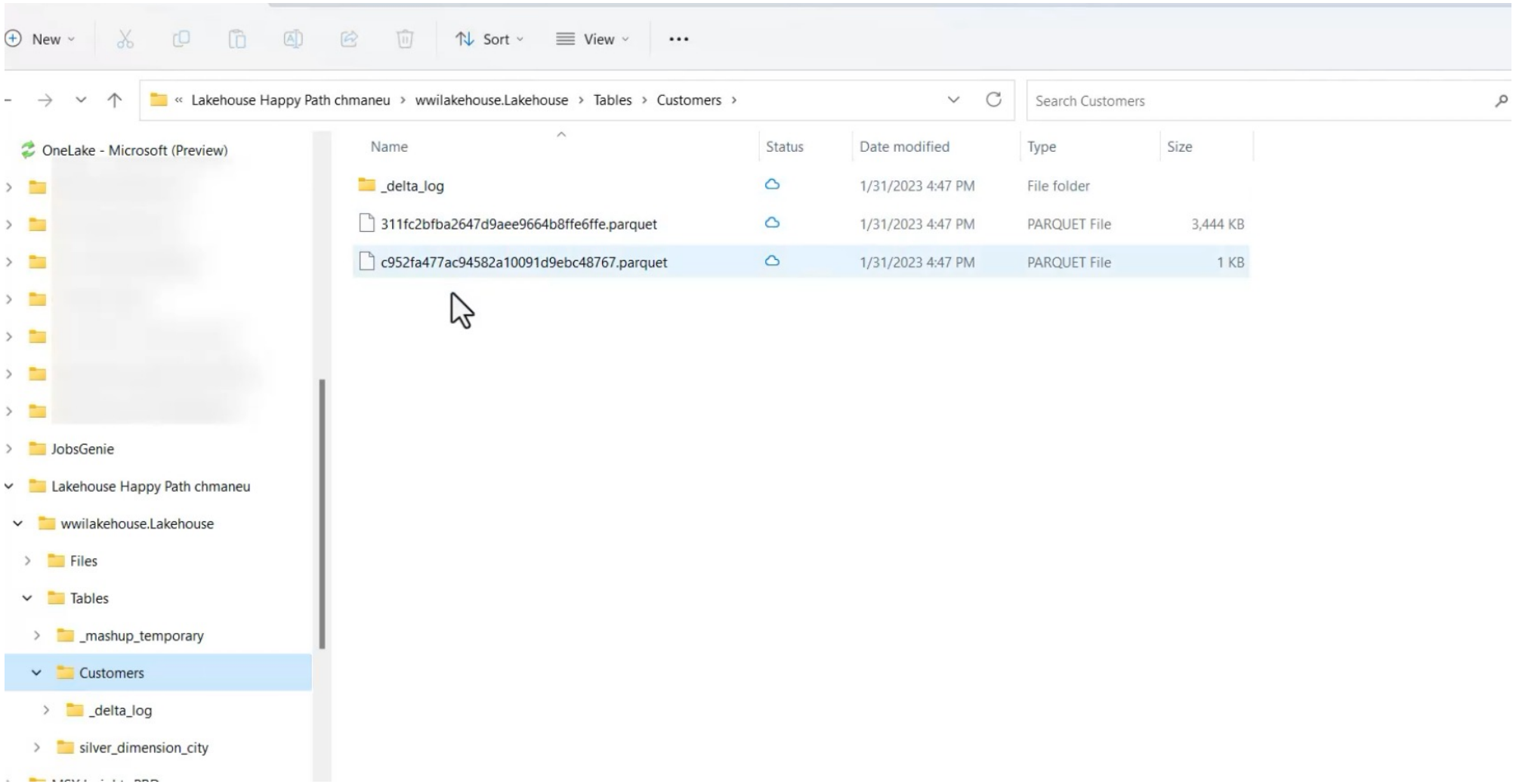
Data Engineering - Data storage

Working with both files and tables



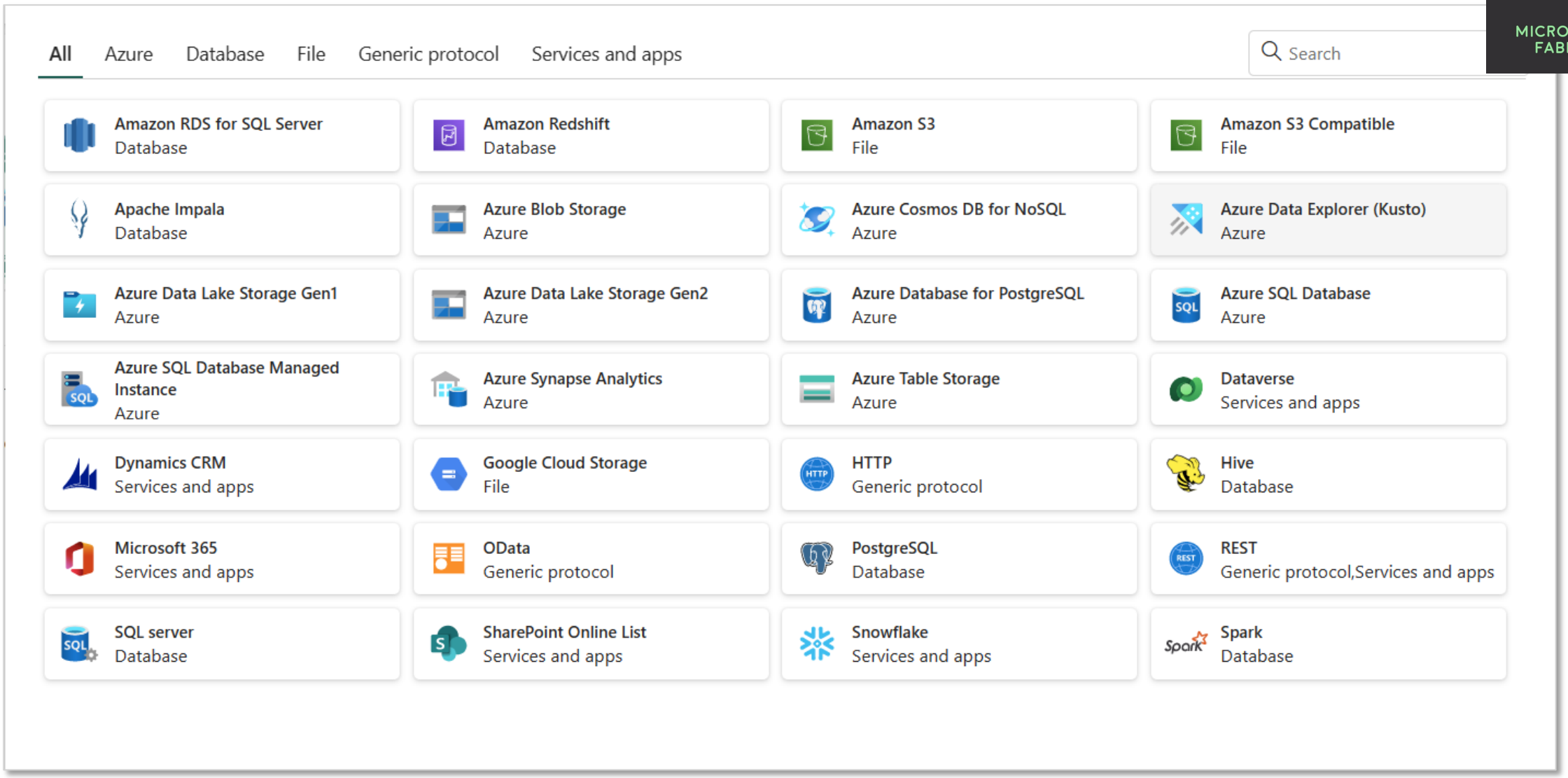
Data Engineering - Data storage

Browse your OneLake from Windows Explorer



Data Engineering - Data storage

Copy data from a wide range of services...



Exercise module 2 - Fabric

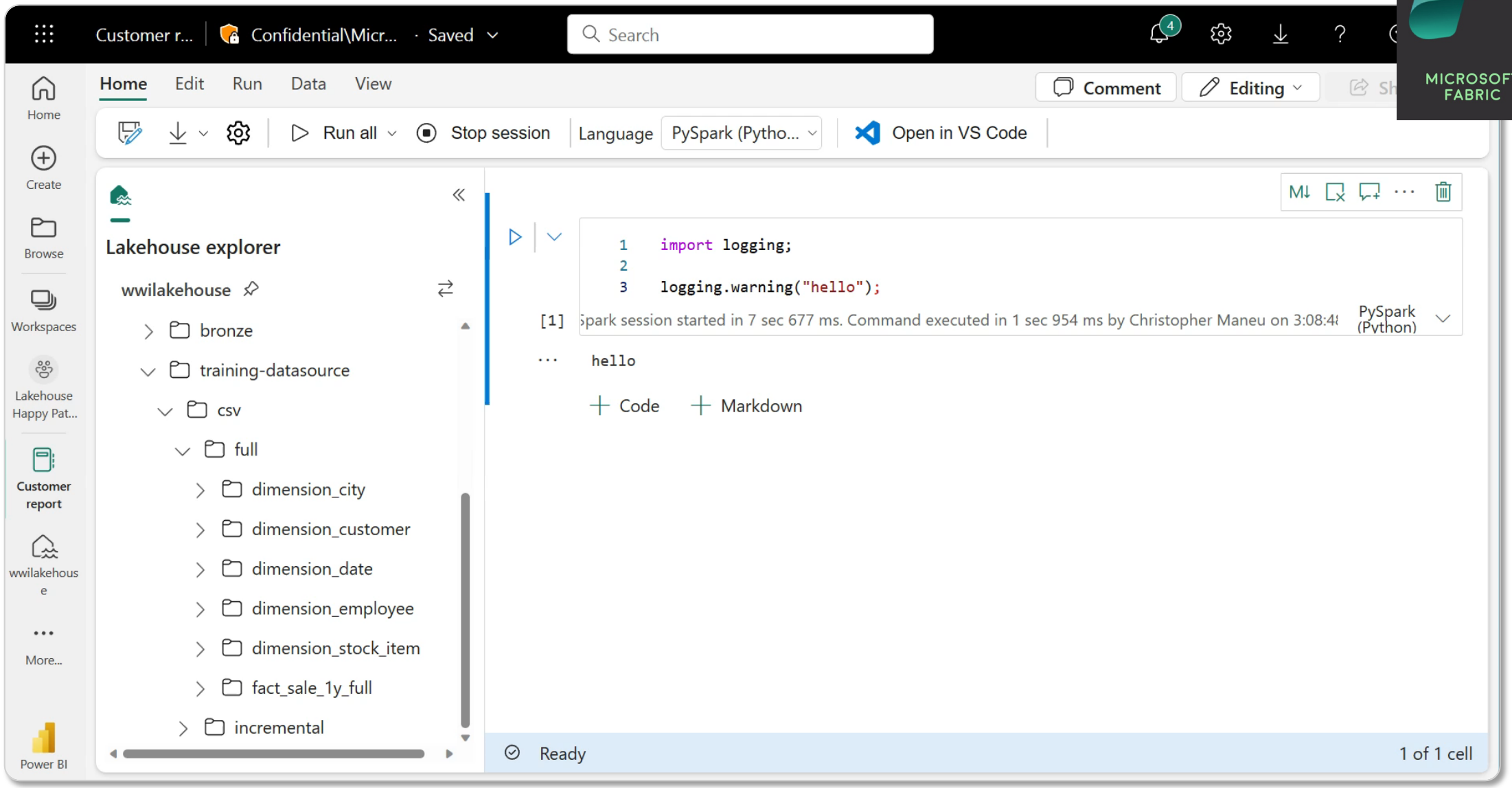
- Exercise 1: Creating a Lakehouse and Writing Data to a Delta Table
- Exercise 2: Creating a Spark Cluster for Distributed Computation
- Exercise 3: Partitioning Large Datasets in Storage for Optimized Compute Performance

Module 3

- Data Movement, data pipelines and data transformation
- Explore the usage of notebooks with Python or PySpark
- Microsoft Fabric: the UI-based and code-based data engineering tasks and explore the use of shortcuts and data lakes
- Azure Machine Learning: designer and learn how to build pipelines using the Python SDK

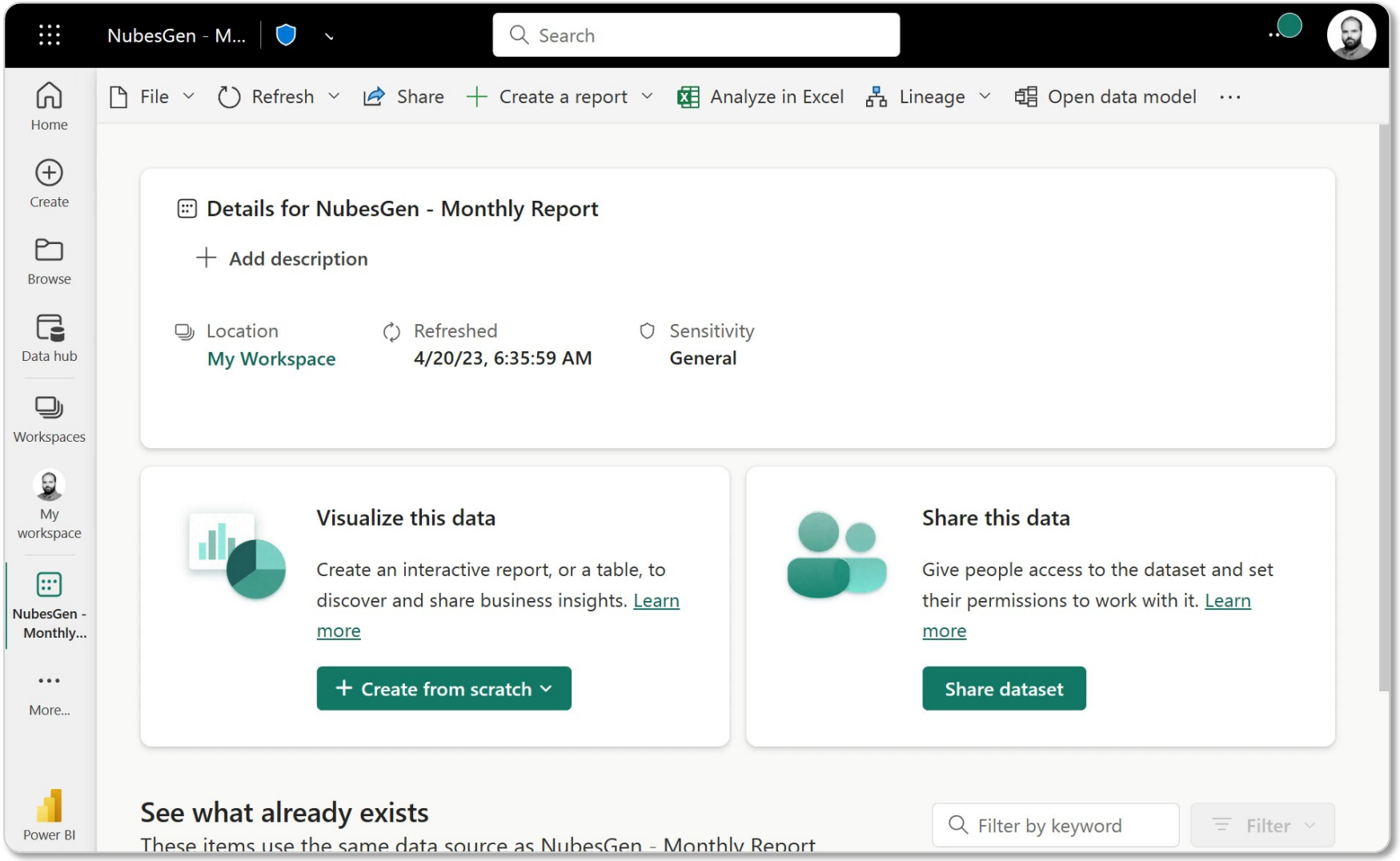
Data Engineering - Data delivery

Access lakehouse data from Notebooks and Spark Jobs



Data Engineering - Data delivery

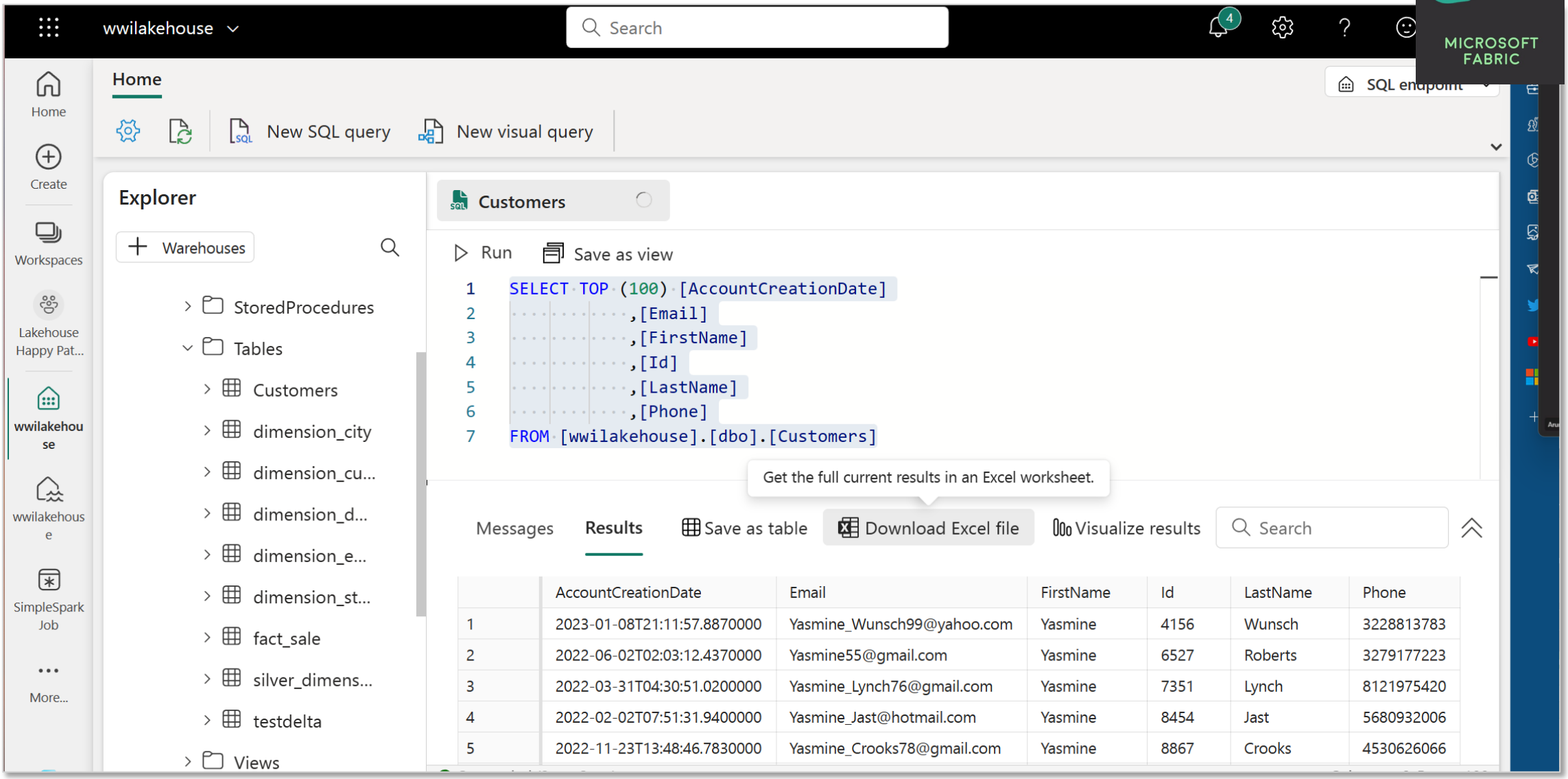
Expose Datasets ready for Power BI



MICROSOFT FABRIC

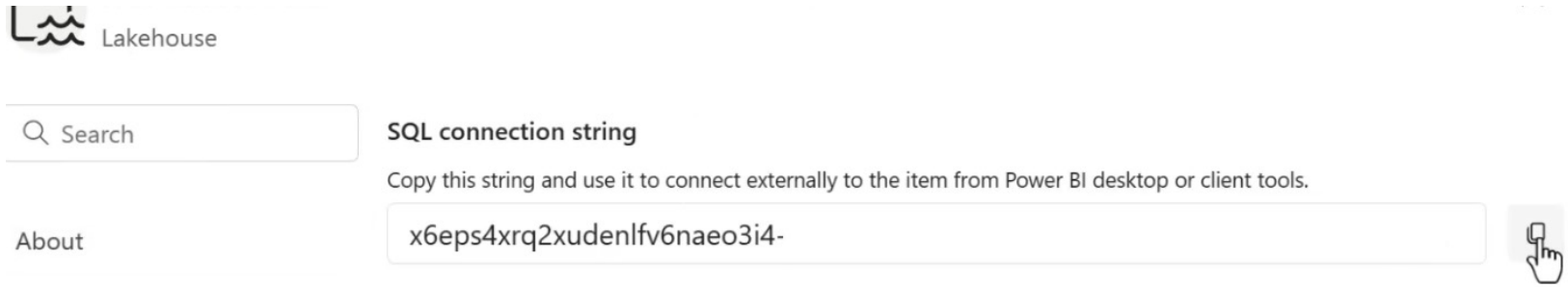
Data Engineering - Data delivery

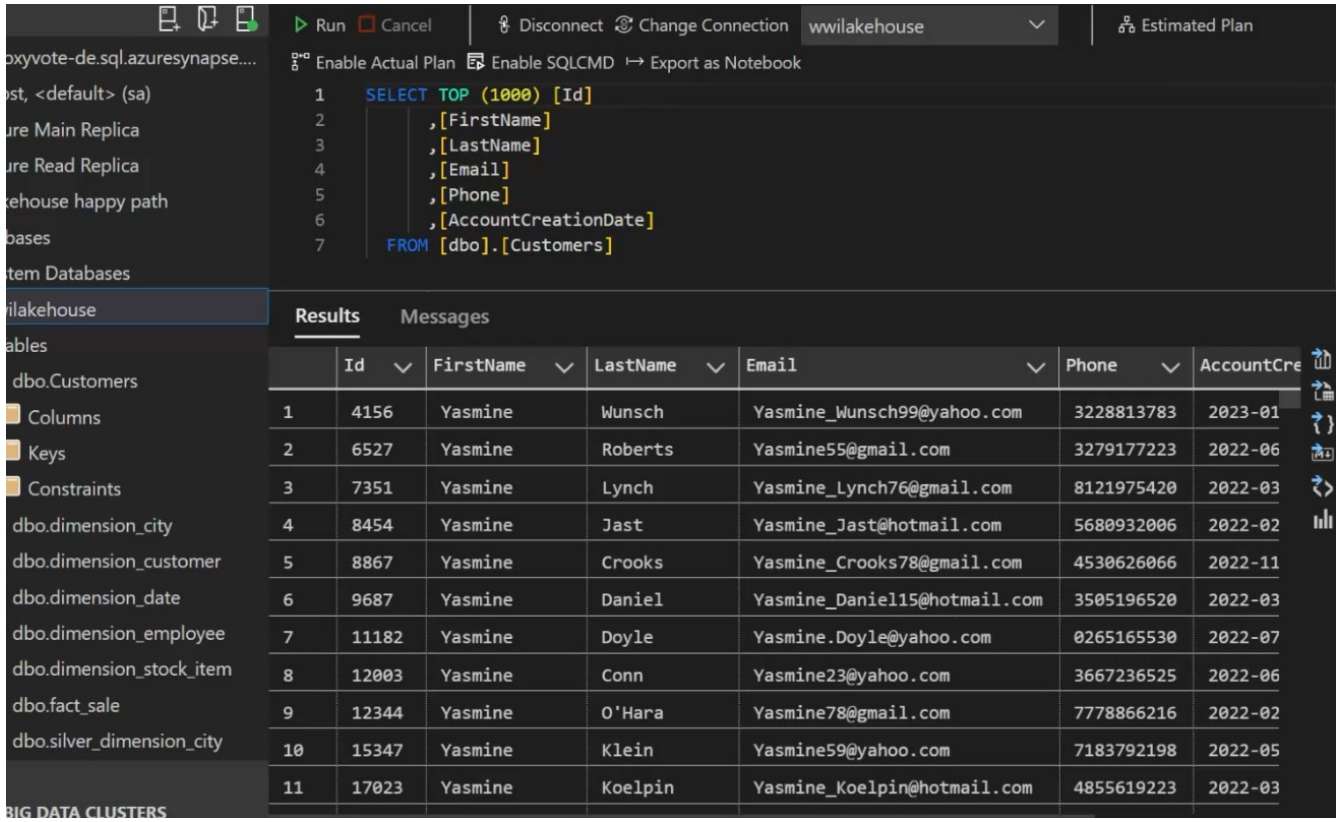
Use Lakehouse data from Excel



Data Engineering - Data delivery

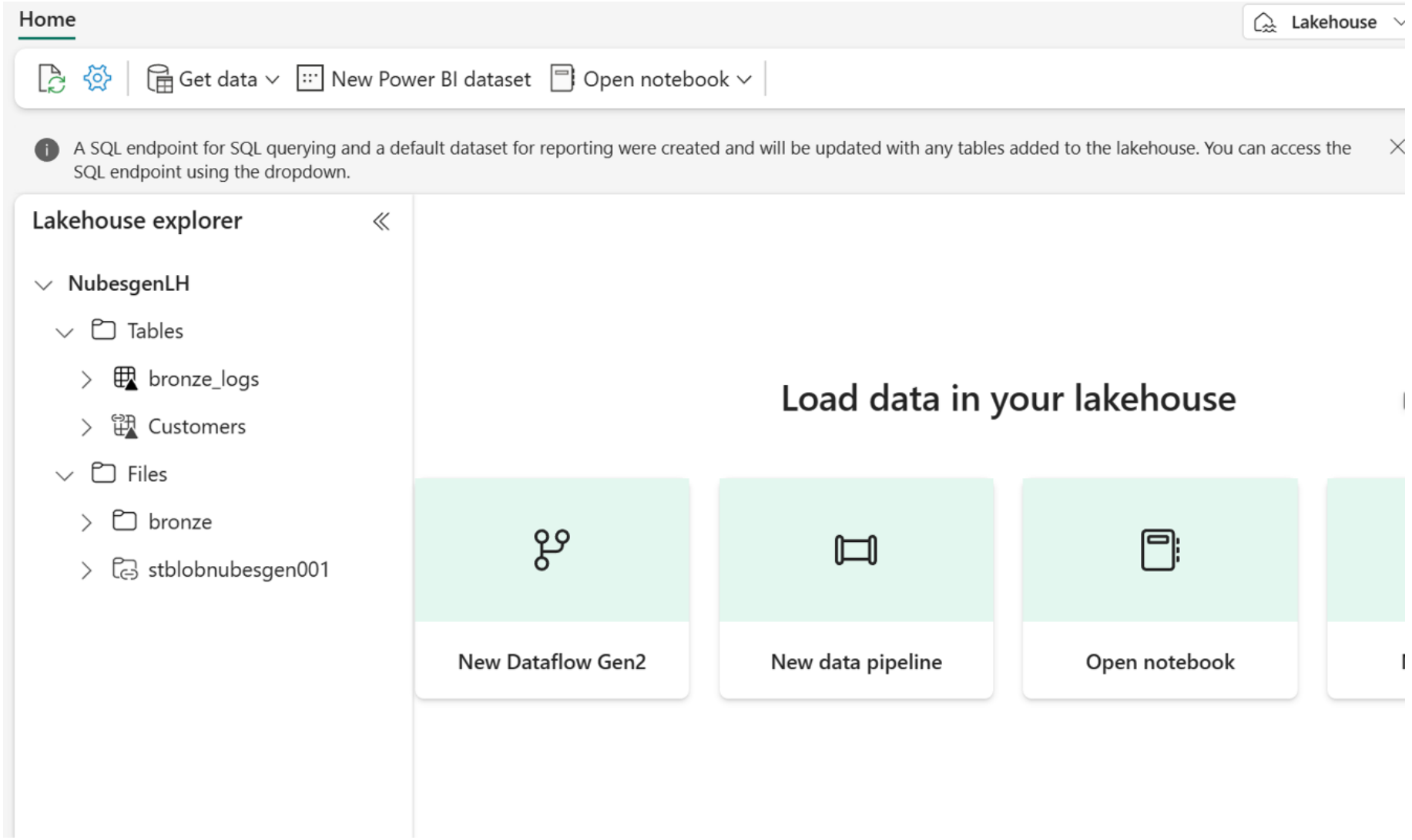
Access your Lakehouse data from any SQL Client



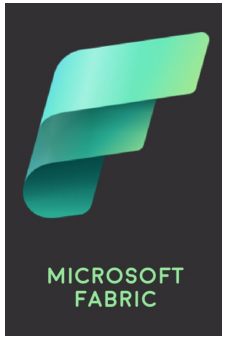


Data Engineering - Data delivery

Organize your data through workspaces & shortcuts



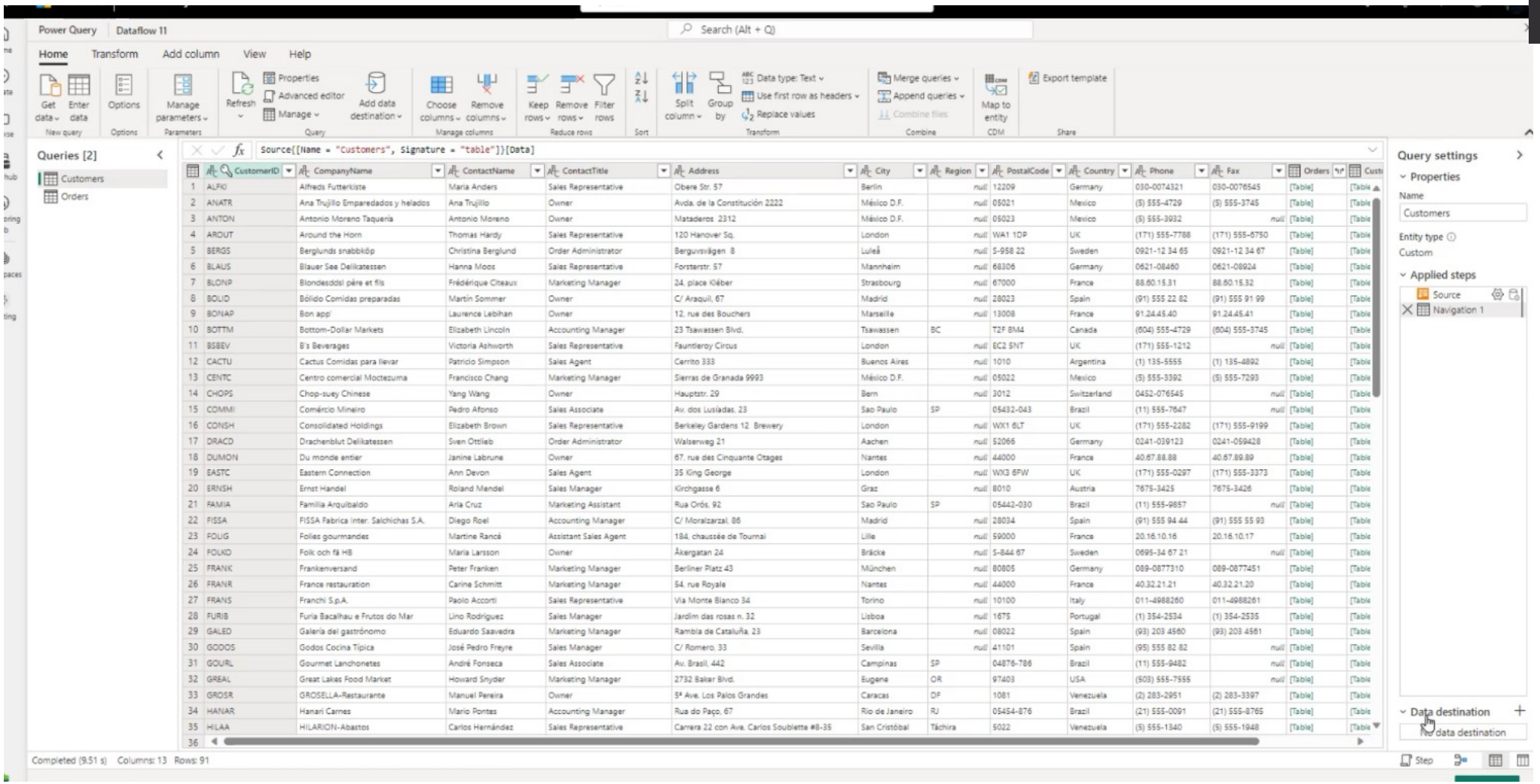
Data Factory



- enables fast data copy and movement to your DataLake and Warehouse
- helps you get the data, prepare the data and do the orchestration
- Scalable, governed via Purview, low-code, AI enabled ☺
- Two high-level features in Data factory:
    - data flow
    - data pipelines

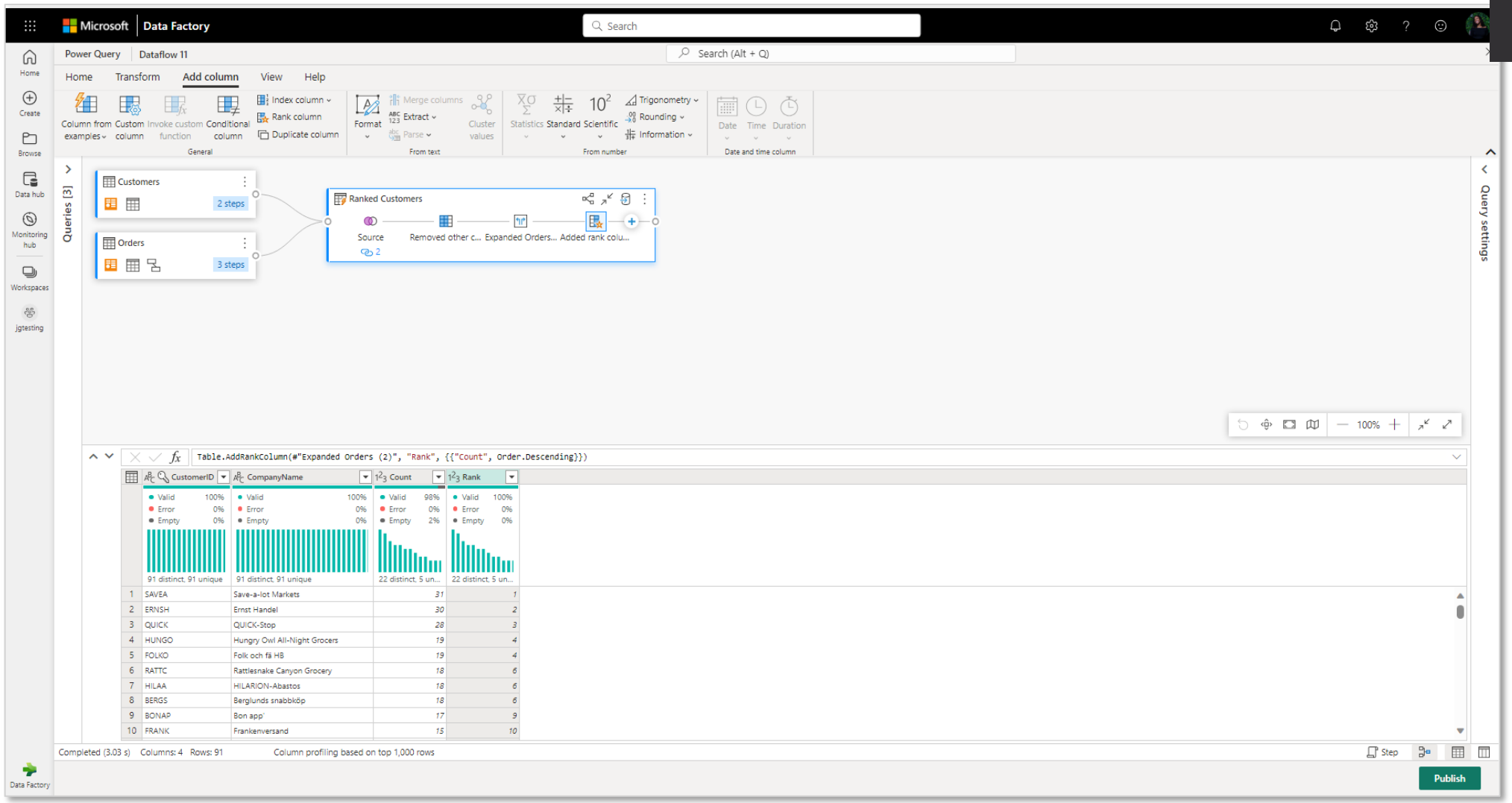
Data Factory

Creating a Dataflow for data ingestion



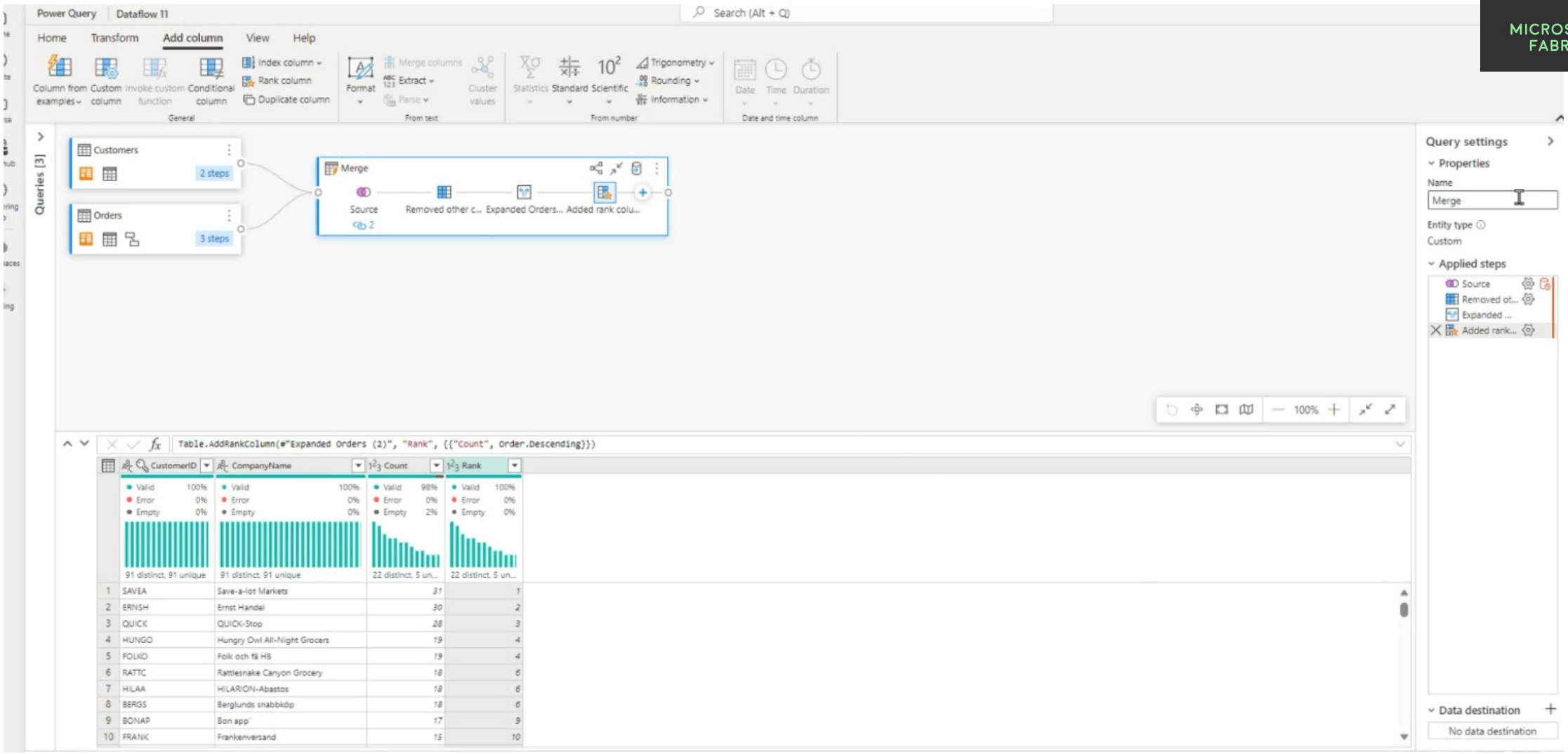
Data Factory

Transform data with Power Query



Data Factory

Aggregation and Merge Operations with Power Query



Data Factory

Building workflows with pipelines

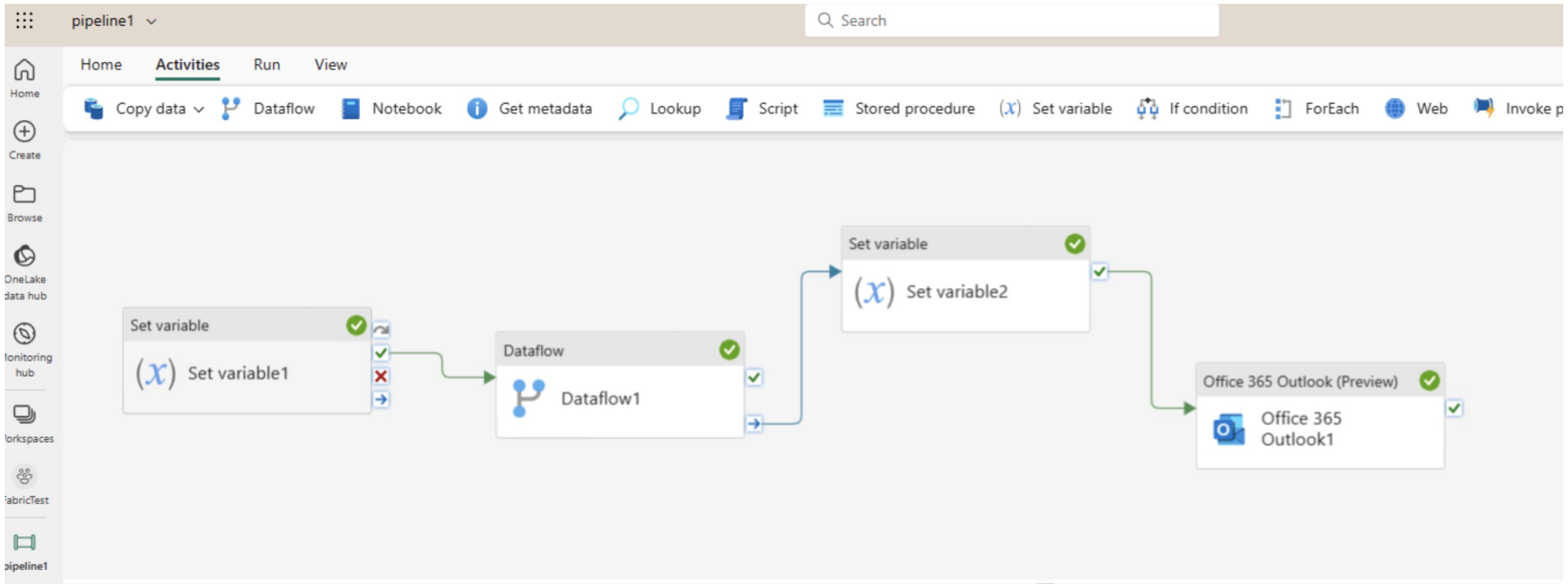
Datasets can be from any source

Activity produces dataset

Dataset consumes activity

Pipelines are a sequence of activities

Activities, datasets, and dataflows can be controlled within the pipeline



Exercise module 3 - Fabric

- Exercise 1: Data Movement Using Shortcuts from One Lakehouse to Another
- Exercise 2: Creating a Data Pipeline for Extract-Transform-Load (ETL) in Microsoft Fabric
- Exercise 3: Data Transformation and Aggregation in a Data Lake Using PySpark

Module 4

- Focusing on data analysis using notebooks
- Data analysis and exploration with Fabric and Azure Machine Learning
- build prepare and build a machine learning model and
- run experiments

EDA

- Exploring data with Fabric and AML
- Data Analysis
- Model creation
- Model training

Exercise module 4 - Fabric

- Exercise 1: Data Analysis and Exploration Using PySpark
- Exercise 2: Data Preparation for Machine Learning Model
- Exercise 3: Build and Train a Machine Learning Model and Run Experiments

Module 5

- Test the predictions and perform inference.
- Fabric: job scheduling and run jobs using MLFlow to model retrain
- Azure Machine Learning: job scheduling and run jobs using MLFlow to model retrain
- expose endpoints for consumption.

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:

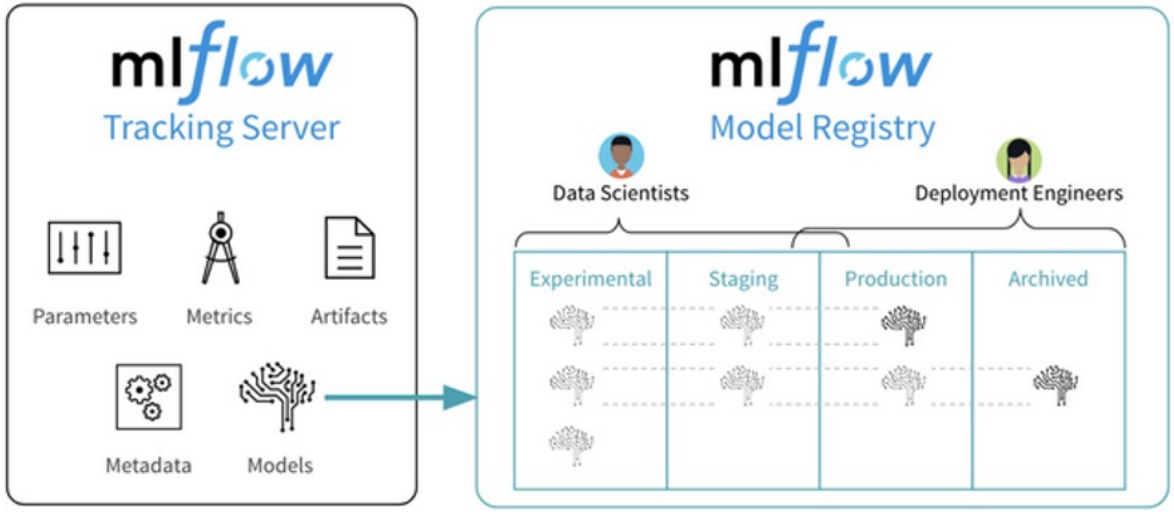
- Keep track of your experiments -> **ML Flow Tracking**
- Standardize your way for storing models, packages -> **ML Registry**

Tracking

- Tracking code development in GIT is easy.

- Tracking intensive data science experiment that includes model is almost impossible using GIT only.

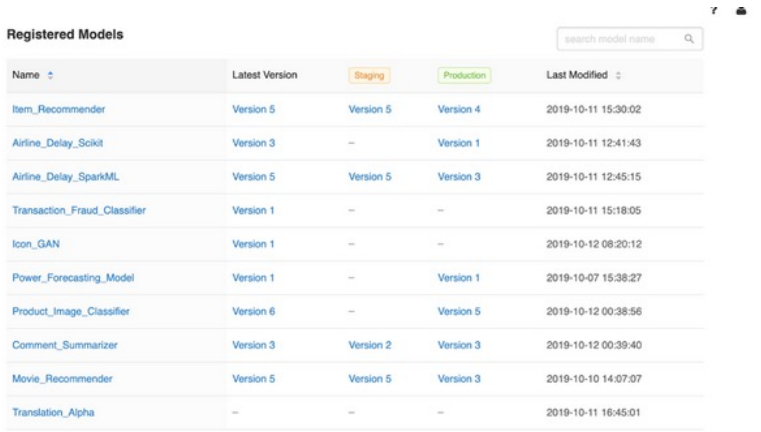
- We need to go back and ask what model/settings achieved the best results.



Experiments

Each ML experiment contains:

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



Exercise module 5 - Fabric

- Exercise 1: Test Model Predictions and Perform Inference Using PySpark
- Exercise 2: Job Scheduling and Running Jobs for Model Inference
- Exercise 3: Model Retraining Using MLflow and Job Scheduling

Module 6

- Creating an end-to-end integrated solution.
- Various options of visualizing the results of our analysis and model predictions.

E2E in steps

Step 1: Data Preparation and Reading Data from the Lakehouse and Data Lake

Step 2: Exploratory Data Analysis (EDA)

Step 3: Feature Engineering

Step 4: Build and Train the Model

Step 5: Model Evaluation

Step 6: Model Inference and API Implementation

Exercise module 6 - Fabric

- Exercise 1: Go through E2E solution