Modern Data Platforms: What Your Organization Needs to Know

forvis mazars

Meet the Presenters

Kyle Harman

Senior Manager

Analytics

kyle.harman@us.forvismazars.com

Philip Rago

Senior Consultant

Analytics

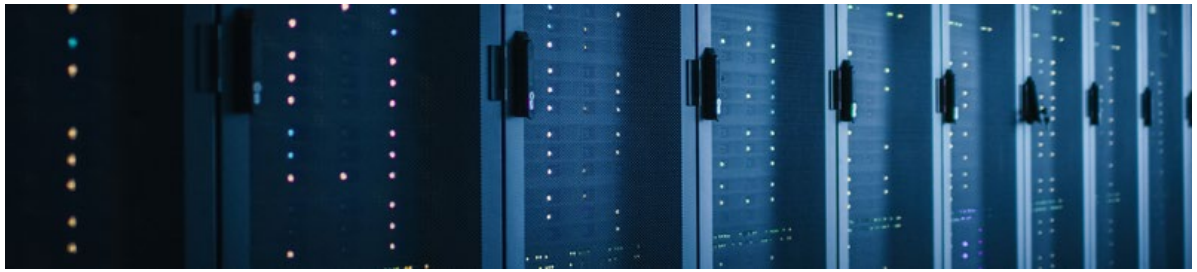
philip.rago@us.forvismazars.com

Agenda

1. Describe a modern data platform and the motivation for using one
2. Review common data platform components and tools
3. Review several data platforms and their use cases
4. Discuss how to choose the right data platform

forvis mazars

Motivation
How Platforms Differ From Databases



SQL Database Only

- Store data
- Transform data
- Administrate using SQL code and resource attributes
- Save SQL scripts

Data Platform

- Store data (SQL or lake)
- Transform data
- Administrate using a built-in web interface
- Store code w/ versioning
- Ingest data, including connectors
- Orchestrate data pipelines
- Leverage multiple languages, *e.g.*, SQL, python, scala, etc.
- Leverage AI/ML capabilities
- Scale compute dynamically

Data Platform Components

Ingestion

- How will you get your data into your data storage?
- What functions (connectors, transformations) does it need to perform?

Storage

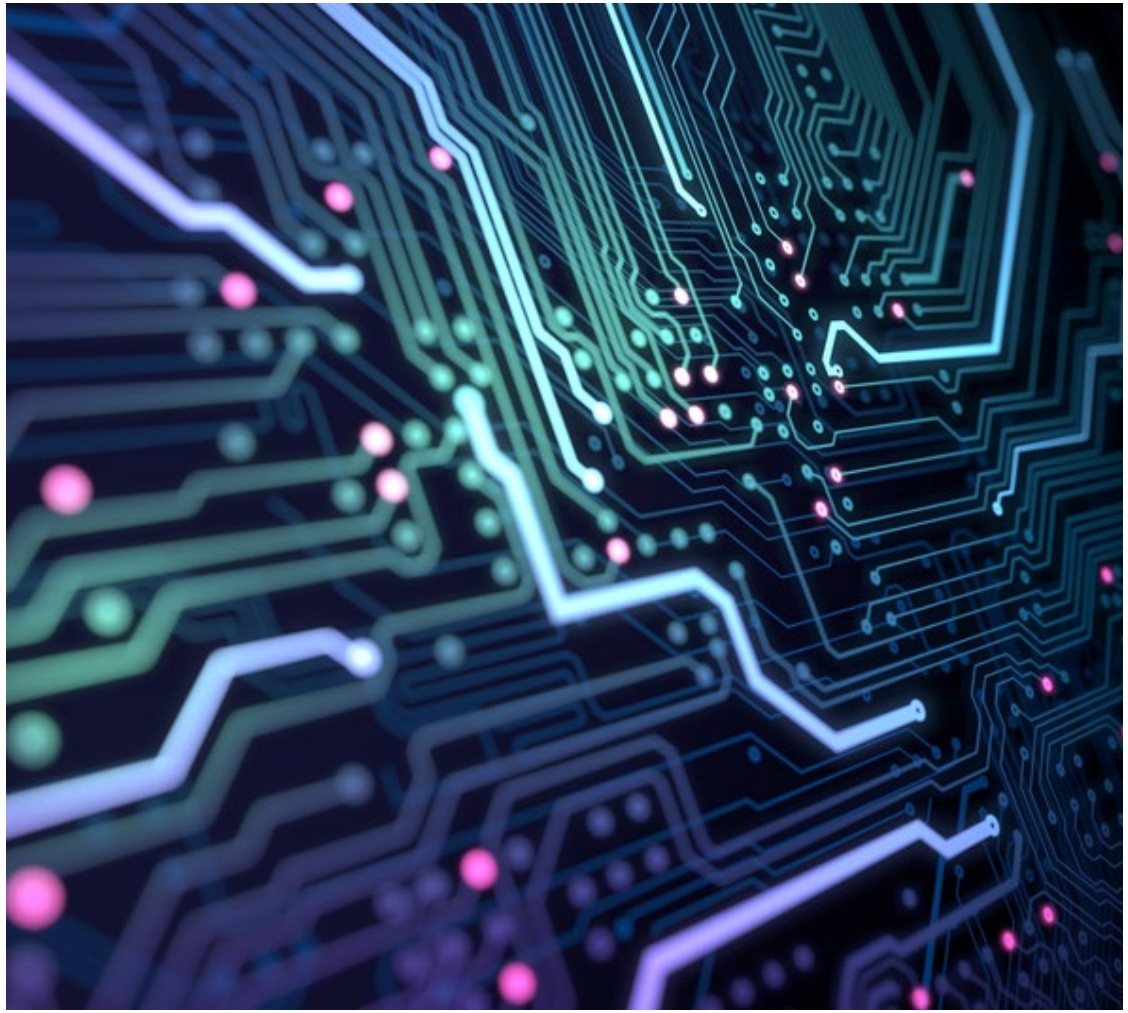
- Where will the data files be stored?
- What format will they be in, and what processing engines can read them?

Data Processing (Transformation & Analysis)

- What types of computation will the platform be doing? Do you need responsiveness? Power?
- Who will be using the platform, and what skills do they have?

Administration & Governance

- How will you organize, document, and secure your data?
- How will this system be administered, and by whom?



01

Data Platform Components – Ingestion

Data Ingestion

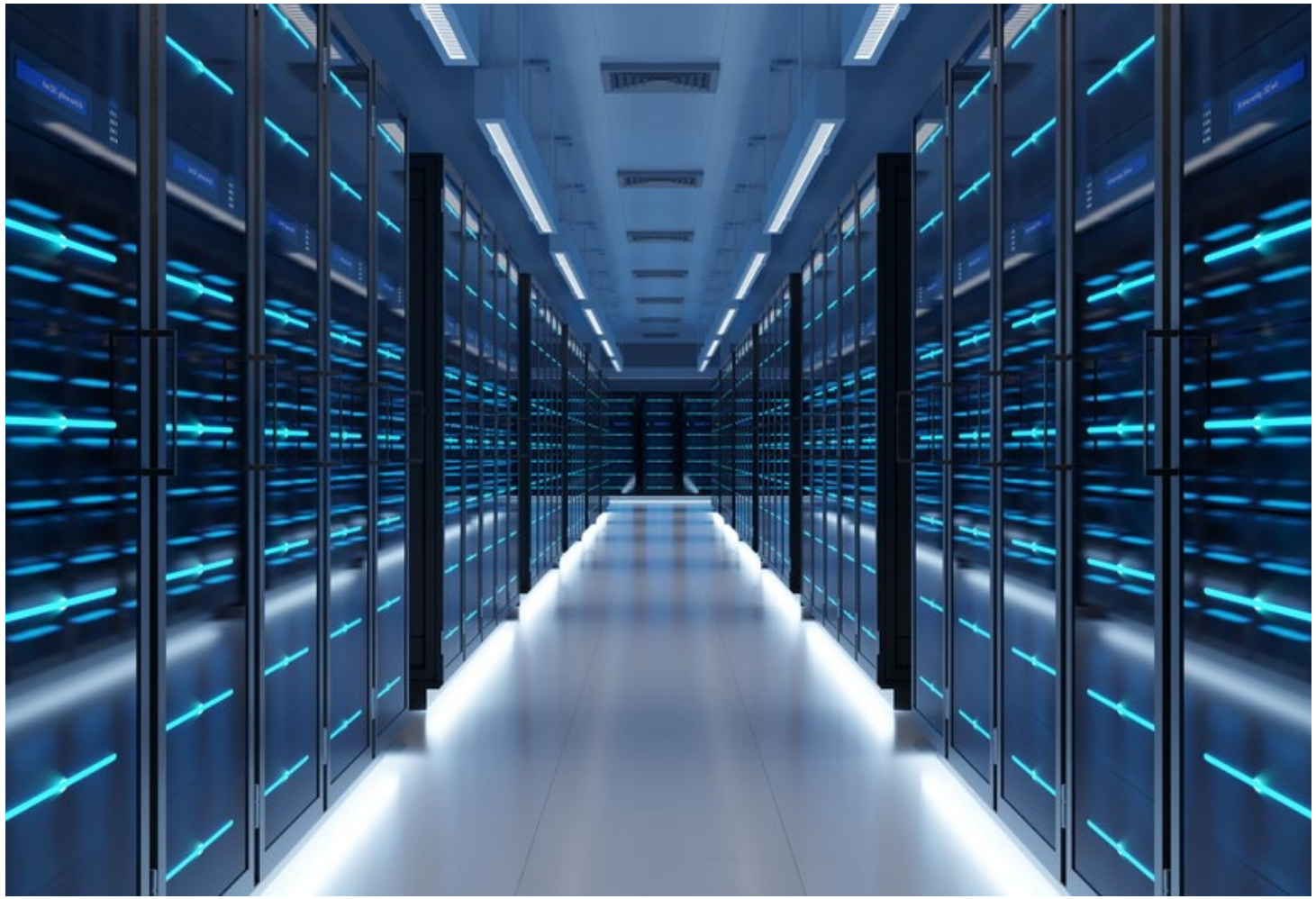
1 Data Sources

(external or internal)

2 Data Ingestion Tool Examples

- Data Factory
- CData
- App Functions
- Lakeflow
- Lambda/App
- Kafka
- Fivetran
- AWS Glue
- Snowpipe
- Delta Live

3 Lake or Database



Data Ingestion

- Data ingestion tools differ by:
  - What built-in connections they offer*
  - What built-in data transformations they perform*
  - How much data they can process
  - Real time vs. intermittent
  - User interface
  - Ability to leverage SQL, PySpark, or other code
  - PRICING
- You need **at least one** ingestion tool, and many systems leverage several.
- Some data tools, such as Databricks and Fabric, include their own ingestion tool.

* These built-ins are not required, but can be very handy.

Data Ingestion Examples
Application & Database Connectors

Platform Connectors

- Prebuilt connectors from your data source to your internal data system
- Specific to a certain set of technologies
- If a platform connector exists for the source and sink systems, generally an efficient method of building the integration

Data Pipeline Orchestrators

- Orchestration and workflow tools that coordinate multistep data movement processes
- Design complex workflows with dependencies, transformations, scheduling, and error handling
- Particularly strong for connecting on-premise data sources to cloud destinations

Third-Party Integration

- Standalone platforms that specialize in connecting data sources to destinations, independent of your data platform
- Offers hundreds of pre-built connectors that may not exist in platform-native options
- These tools specialize in data movement; they're not tied to any particular cloud ecosystem or data platform

Data Ingestion Examples
Considerations

	Platform Connectors	Data Pipeline Orchestrators	Third-Party Integration
Setup Time	■■□□□ Fast	■■■□□ Medium	■■□□□ Fast
Complexity	■■□□□ Low	■■■□□ Medium	■■□□□ Low
Flexibility	■□□□□ Low	■■■■□ High	■■■□□ Medium
Maintenance	■□□□□ Min	■■■□□ Moderate	■■□□□ Low
Cost	■■■□□ Pay/Use	■■□□□ Infra	■■■■□ License
Examples	Lakeflow Connect Snowflake Native	Fabric Data Factory (MS Fabric) Azure DF (Legacy)	Fivetran Airbyte
Best For	• Cloud-native • Quick setup • Managed SaaS	• Multistep • Dependencies • Hybrid/on-premise	• Coverage gaps • Cost control • Customization • Engineering teams

Ingestion Considerations
The Right Tool for the Job

Streaming & Change Data Capture

For real-time or near real-time data needs. Continuously capture changes from databases and event streams as they happen.

Examples: Snowpipe Streaming, Kafka, Debezium, Delta Live Tables

Serverless File Loaders

Auto-detect and incrementally load files from cloud object storage as they arrive. Pay only for what you use.

Examples: Auto Loader, Snowpipe

Custom Functions & Code

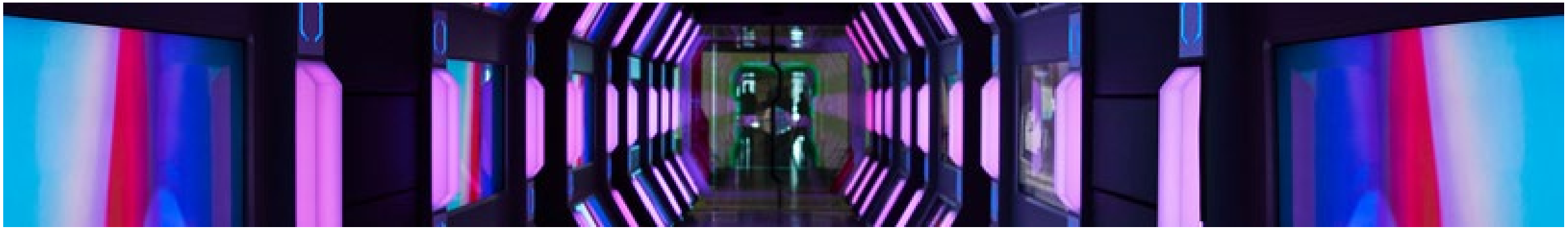
Maximum flexibility for edge cases, complex transformations, or unique business logic. Write custom code that runs serverless.

Examples: Azure Functions, AWS Lambda, Google Cloud Functions

& Many More ...

Message queues, API gateways, IoT ingestion pipelines, reverse ETL tools, database replication utilities, and domain-specific connectors.

The ecosystem continues to evolve with new patterns and tools emerging regularly.



forvis mazars

Ingestion Considerations

Where does your data live?

- Cloud Blob Storage
- Software Products, *e.g.*, CRM, ERP
- Databases (Cloud or On-Prem)
- Event Streams

How fresh does the data need to be?

- Real-time: sub-seconds
- Near-real-time: seconds – minutes
- Hourly or Daily

How technical is your team?

- Analysts vs. Engineers
- Code vs. GUI

Data Ingestion Tools
Summary

- Data ingestion tools are a key component that pulls data into your ecosystem.
- Many tools exist, and it is common to leverage more than one.
- Start with a simple, inexpensive tool that pulls in most of the data, and then deal with edge cases later.

02

Data Platform Components – Data Storage

Data Storage

- Data is stored in files. Examples:
  - Microsoft SQL -> .mdf
  - SAS -> .sas7bdat
  - Python, R, Fabric, Databricks, etc. -> .parquet, .orc, .csv, and others
- How data is accessed varies between systems.
  - In many cases (*e.g.*, SQL), the data is meant to be stored and interacted with through the processing engine.
  - In other systems the compute and storage are separate, and storage and access are separate decisions.
  - So, for instance, you can have a folder of .parquet files and Databricks, Synapse Analytics, or both to interact with them.
- Deciding on data storage type:
  - What systems do I want to be able to access my data? (Data Lake or other?)
  - What types of operations will be done on the data (columnar vs. row, schema-on-read vs. on-write)
  - What will the data be used for? (transactional systems, reporting, analysis)

forvis mazars

Defining Data & Delta Lakes

Data Lake

- A highly scalable data repository that can be efficiently but securely accessed.
- There can be any or no file organization.
- There can be any or no rules around how data is written to or read from the lake.
- Any file types are accepted.

Delta Lake / Iceberg

- A type of data lake.
- Requires a hierarchical folder system.
- Typically stores relational data.
- Files are stored in a high-performance file type such as .parquet or .orc.
- Software is used to organize the data and generate metadata.

If someone says “data lake,” you don’t know much about it except that it’s a scalable repository that may have files in it.

If someone says “delta lake,” you know it contains high-performance files containing relational data and can be efficiently interacted with by modern data platforms like Databricks and Fabric.

Note that both Data and Delta lakes are their **own entity** and exist without software that accesses them.

Data Lake vs. Traditional Database

Should you use a lake (+ lakehouse) solution with separate data and compute, or a traditional solution where the data is stored with the software?

- **Advantages of leveraging a data lake:**
  - Data is portable between data software
  - The expensive part (the compute) can be reduced and the data still exists
  - Effectively infinite storage
  - Superior access and governance controls
- **Advantages of a traditional (data + software combination) solution:**
  - Simpler?
  - Some use cases are better met with specific database types, *e.g.*, NoSQL backend for applications

forvis mazars

Data Storage Tools
Summary

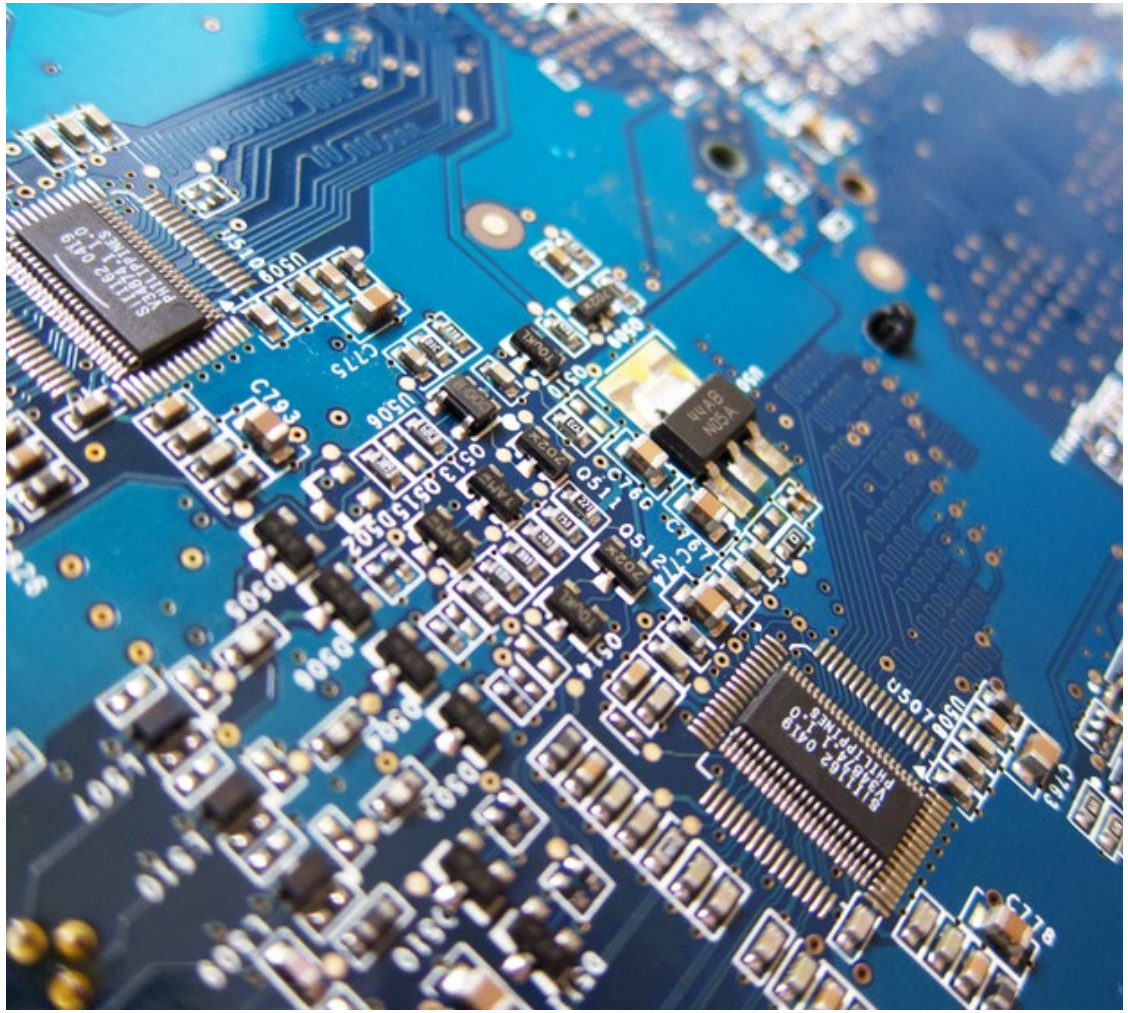
- Traditionally data storage was (largely) dictated by the data software.
- A data (or delta) lake is its own entity and can be accessed by multiple types of data software.
- Unless you have a compelling reason not to, your first choice should be a delta lake and data processing software that meets your needs.

03

Data Platform Components – Data Processing

Data Processing Engine

- Data systems employ software (“engine”) that, when leveraged by processors (“compute”), acts on the data.
- The engine defines how the data files are interacted with, what kinds of processing power is needed, how that processing is leveraged, and what kind of inputs are required to process data.
- There are many different engines, some of which are proprietary.
- In the following slides we’re going to review common data engines by breaking them into three categories:
  - SQL Databases
  - Lakehouses
  - Other



SQL Databases

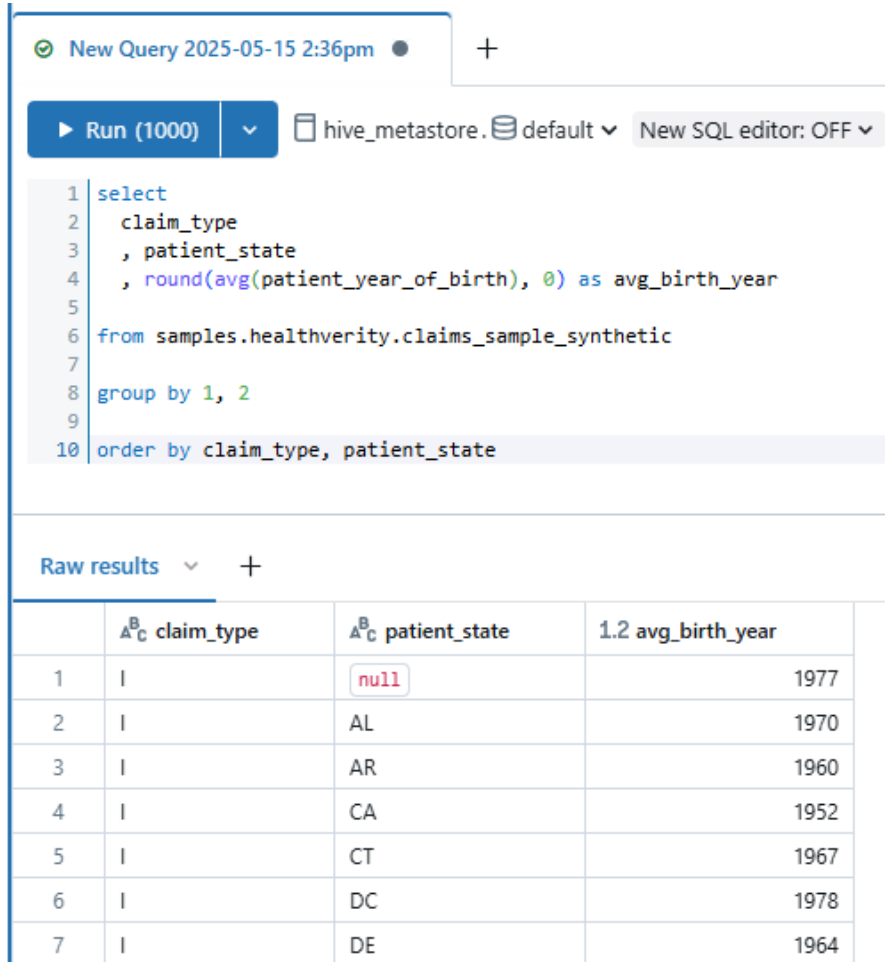
- A SQL Database is a database whose method of interacting with the data is through SQL.
- “SQL” stands for Structured Query Language, and was developed in the ’70s.
- Therefore, SQL Databases are a class of data systems, with many examples.
- Traditionally, the SQL engine and data were tightly coupled.
- Typically, the user brings their own user interface.

Example SQL Database Types

SQL Database	Pricing	Processing Engine	Notes
Microsoft SQL Server	Cost of infrastructure used	Microsoft SQL Server Engine	Strong MS support, license cost
Databricks SQL Server	By compute and data storage	Spark	Optional capability of Databricks
Postgres	Cost of infrastructure used	PostgreSQL	Open Source, widely used
MySQL	Cost of infrastructure used	Varies (*e.g.*, InnoDB)	Open Source, widely used

SQL Example Use

- SQL is a simple but powerful language used to manipulate relational data.
- The language allows for simple, quick queries, as well as complex joins and nested instructions.
- Only SQL can be used on a SQL server.



Source: Databricks

SQL Server Pros, Cons, Use Cases

1

SQL Server Pros

- Time-tested relational database system
- Well understood and documented administration
- SQL easy to learn (moderate to master)
- Vast resources for learning and debugging SQL
- Adequate speed for pipelines and analysis

2

SQL Server Cons

- Can only leverage SQL (no Python)
- No AI/ML capabilities
- Not data/delta lake friendly
- Limited scaling ability
- Inadequate response times for most front-end applications

3

Use Cases

- Data exploration and analysis
- Organization's secondary data system
- Supporting data pipelines for an application
- In rare circumstances, as an application back-end

forvis mazars

Lakehouse Systems

- Often called “lakehouses” because they are designed to work with a data/delta lake.
- Compute and data storage separate and interchangeable.
- Are often “All-in-One” solutions that include data ingestion tools, scalable data processing, data warehouse functionality, and administrative support.
- Support data engineering and data science capabilities.

Lakehouse	Engine	Pricing	Notes
Databricks	Spark	By compute usage + cloud storage	Mature system with extensive history and support
Fabric		Monthly (with compute limit) + cloud storage	Offered through Microsoft Azure, a recent entry in the field. Replacing Synapse Analytics
Synapse Analytics		By compute usage + cloud storage	Offered through Azure, an older offering still in use but with limited support from Microsoft
Google BigQuery	Proprietary + Spark	Compute + storage	Offered through GCP, a proprietary system with a spark add-on
Snowflake	Proprietary	By “units” and storage cost	Data stored on Snowflake infra; more vendor involvement

Defining Data & Delta Lakes

Databricks SQL Example Use

22 hours ago (2s)

```
%sql

select
  claim_type
  , patient_state
  , round(avg(patient_year_of_birth), 0) as avg_birth_year

from samples.healthverity.claims_sample_synthetic

group by 1, 2

order by claim_type, patient_state
```

(2) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [claim_type: string, patient_state: st

Table +

	claim_type	patient_state	avg_birth_year
1	I	null	1977
2	I	AL	1970
3	I	AR	1960
4	I	CA	1952
5	I	CT	1967
6	I	DC	1978
7	I	DE	1964

Pyspark Example Use

21 hours ago (1s)

```
from pyspark.sql.functions import round
df = spark.read.table("samples.healthverity.claims_sample_synthetic")
df_agg = df.groupBy(['claim_type', 'patient_state'])\
    .agg({"patient_year_of_birth":"avg"})\
    .withColumnRenamed("avg(patient_year_of_birth)", "avg_birth_year")
df_agg = df_agg.withColumn('avg_birth_year', round(df_agg.avg_birth_year, 0))
df_agg = df_agg.sort('claim_type', 'patient_state')
df_agg.display()
```

(2) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [hvid: string, claim_id: string ... 33 more fields]

df_agg: pyspark.sql.dataframe.DataFrame = [claim_type: string, patient_state: string ... 1 more fi

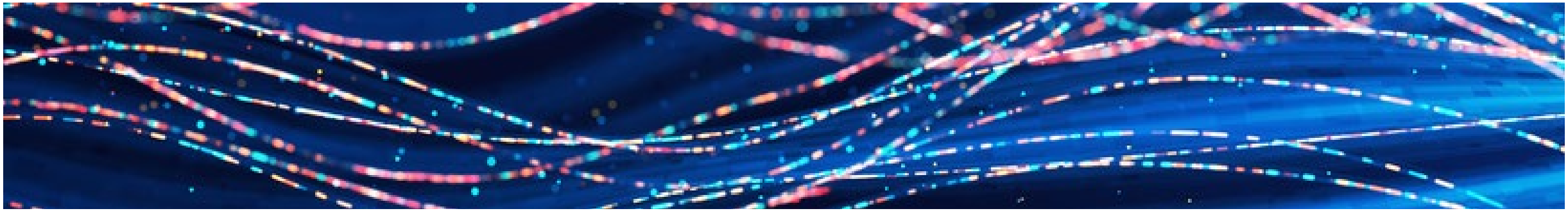
Table +

	claim_type	patient_state	avg_birth_year
1	I	null	1977
2	I	AL	1970
3	I	AR	1960
4	I	CA	1952
5	I	CT	1967
6	I	DC	1978
7	I	DE	1964

Source: Databricks

forvis mazars

Lakehouse Pros, Cons, Use Cases



1 Lakehouse Pros

- Core system is mature and well understood
- Massively scalable, capable of huge workloads
- Pay for the compute you use
- Leverage multiple language choices (*e.g.*, SQL)
- Works with many file types

2 Lakehouse Cons

- The flexibility requires more admin knowledge
- Scaling compute rewards efficiency and punishes sloppy design/code
- Some types of errors can be frustrating to debug

3 Use Cases

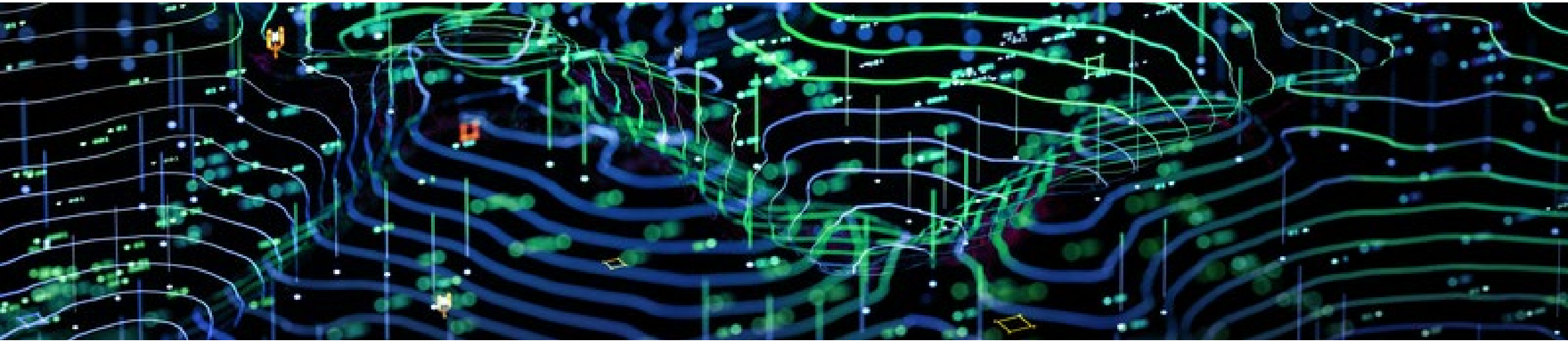
- Organization's primary data system, including data cataloging and management
- Data exploration and analysis
- Supporting data pipelines for an application
- Stats, Machine Learning, and AI implementation

forvis mazars

Other Data Engines

Data Engine	Description	Use Case	Notes
ML Studio / Sagemaker	Standalone machine learning platforms	Enable ML capabilities (*e.g.*, combine with SQL database)	Data tools geared towards machine learning
Alteryx	GUI-driven data processing tool	Prototyping, simple workflows that don't require automation	Useful for less technical users with strong business expertise; can later move pipeline to the data platform
SAS (traditional)	Data and analytics system	Data transformation, statistical analysis, advanced analytics	Thwarts every modern software pattern, leading to unwieldly, unmaintainable code bases. Still strong in a few very specific statistical areas
Python Pandas/ Polars	Execute Python code on a server or serverless; bare-bones	Very simple data processing for an application	Lacks important data processing attributes such as ACID compliance
NoSQL (*e.g.*, MongoDB)	Non-tabular data storage and retrieval, limited transformations; exceptionally fast response time	Application back-end, niche dictionary storage/retrieval	Often the only data storage system that can meet users demanding needs

Data Processing Engines
Summary



- SQL databases and/or a lakehouse act as the backbone of an organization’s data platform computational capabilities.
- Mixing & matching resources can add value as long as cost and complexity are managed.
- While a SQL server may or may not be used, the SQL language is almost always available to interact with data.
- While SQL databases are still well-tested workhorses, try to avoid making them the first stop for your data, as their inflexibility can lead to challenges in the future.

04

Data Platform Components – Administration

Data Platform Admin

Administering the data platform happens in three layers:

1. **Infrastructure**: Either on prem or cloud IaaS. Typically command line management of how the hardware and operating system are made available to the software. In a cloud deployment it is not unusual to avoid the need to manage the infrastructure.
2. **Software**: Either installed on your infrastructure, or cloud PaaS/SaaS tools. This involves choosing, deploying, and managing the software that is used, including maintenance, networking, and configuration.
3. **Database/Platform**: Each data tool will require user authentication, connections to other tools or data, and configurations that are required for the software to work.

How administration differs between products/components:

1. Cloud native / Hardware install
2. UI vs. No UI
3. Data catalog functionality

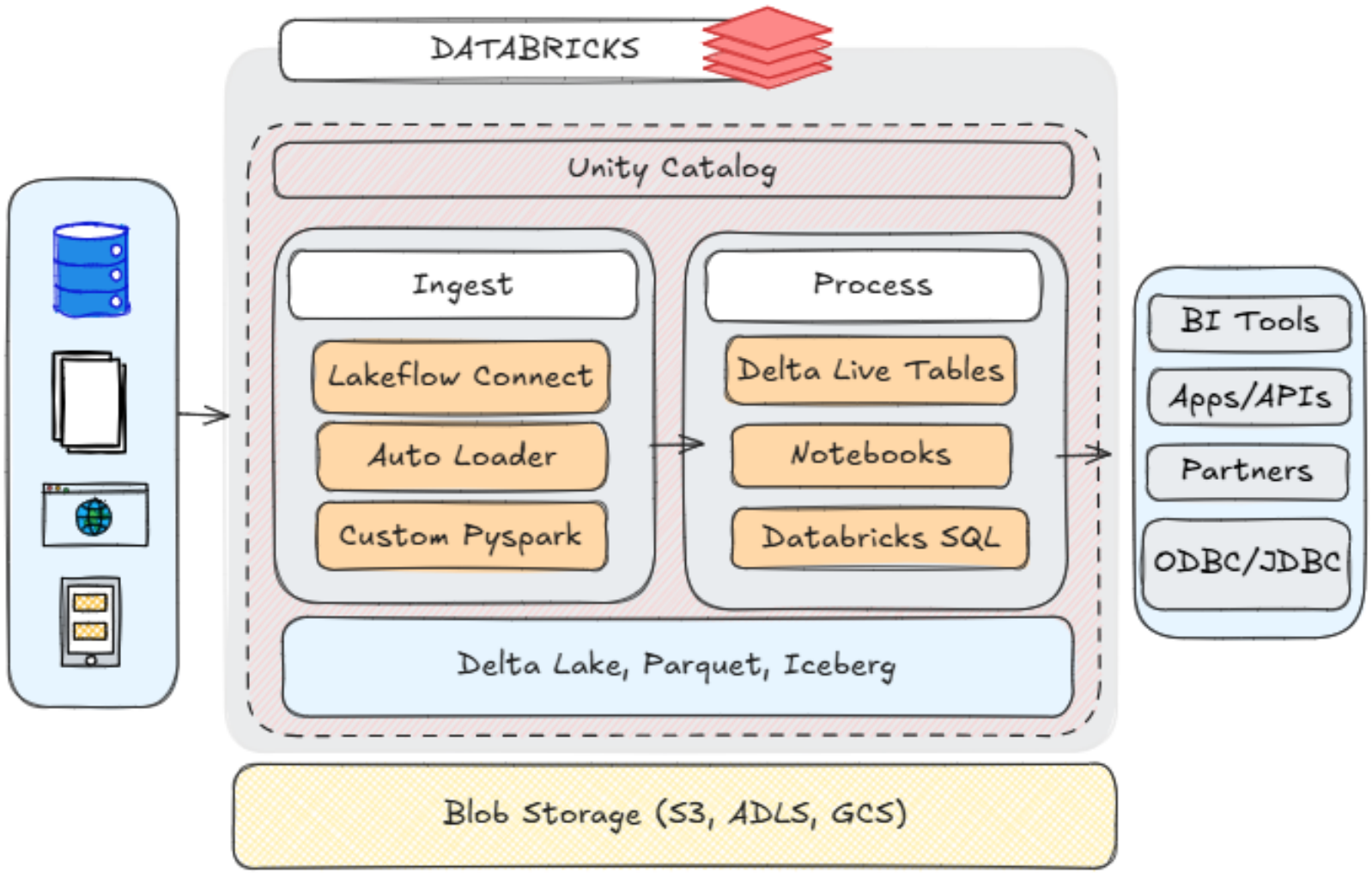
Some systems discussed in this presentation are more similar to use than to administer, *e.g.*, Fabric vs. Databricks.

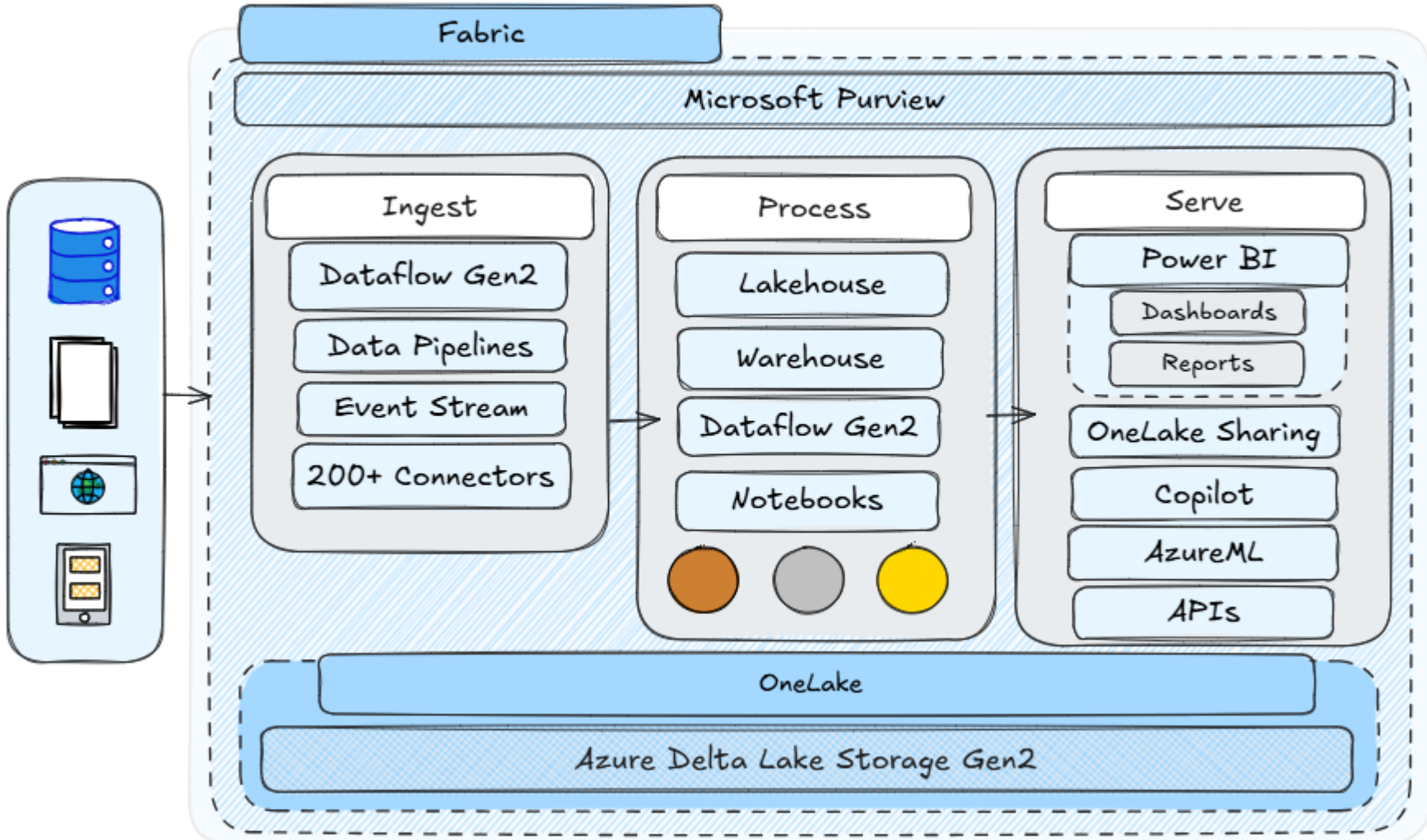
Data Governance

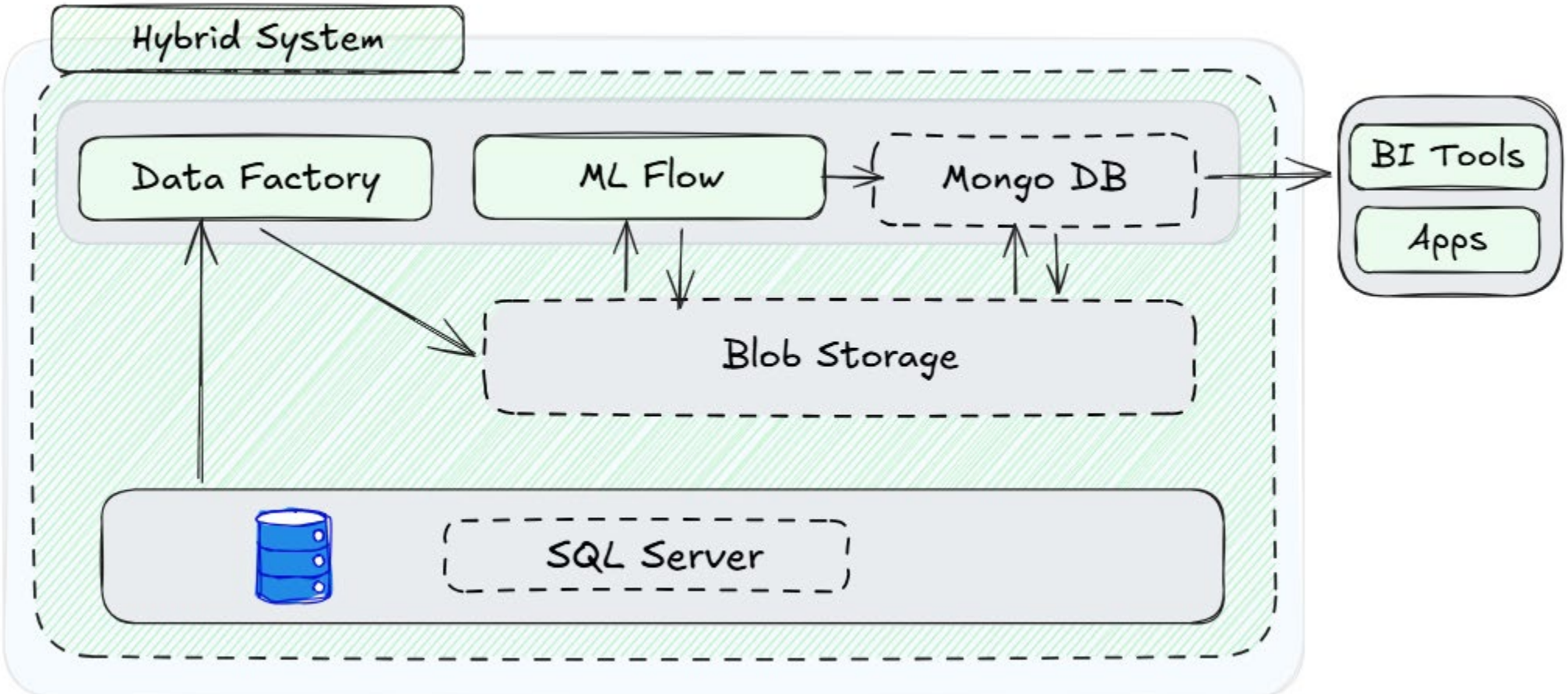
- Data Governance seeks to ensure appropriate quality, security, and availability of data used by an organization and its partners.
- Is largely a political and organizational endeavor, implemented by defining data ownership, creating processes, and extensive documentation.
- However, security and compliance can be strengthened through technical implementation such as data catalogs and data administrative controls.
- A good data catalog will:
  - Be directly integrated with the data,
  - Identify the source, purpose, and owner of the data,
  - Provide straightforward access controls that allows minimum required access to data.
- Databricks, Fabric, and Snowflake all come with some or all technical tools required for good data governance implementation.

05

Example Data Platforms







06

Summary & Final Thoughts

Choosing the Right Data Platform Components

Ingestion

- Start with a general-purpose data ingestion tool, if possible.
- Use more specialized tools sparingly for efficiency gains.

Storage

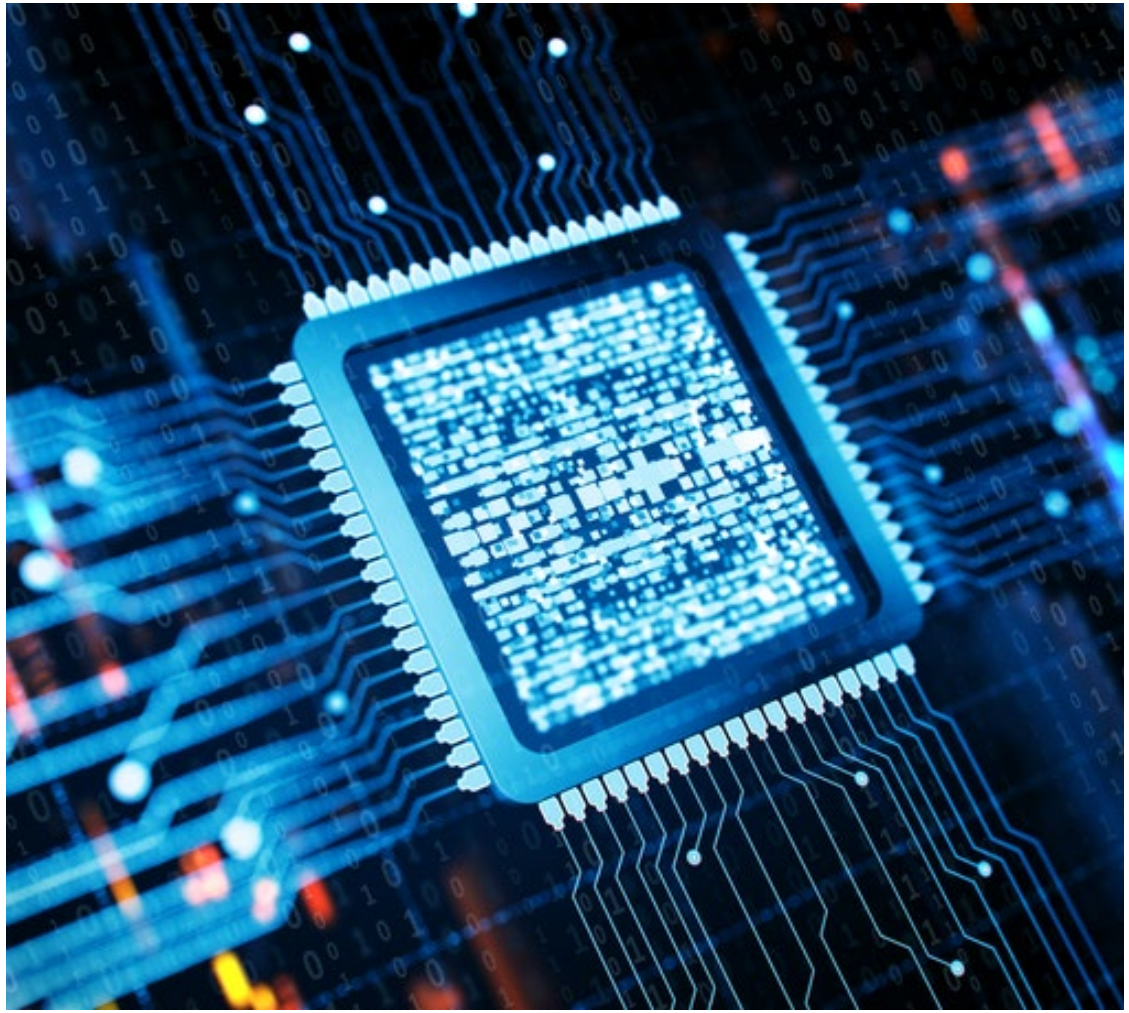
- Start with a lake, even if you plan to leverage SQL as your main tool.
- Look for flexibility and long-term sustainability.

Data Processing

- Consider a lakehouse first to enable advanced data and analytics.
- Add SQL/NoSQL and other tools as needed.

Administration & Governance

- Get to know the tools you've chosen and their admin capabilities.
- Have a governance plan built out to start and leverage built-in tools to assist with ownership and compliance.



Forvis Mazars supports clients end-to-end in four main areas in a way that fits their organization's data and technology maturity.

Data Strategy, Governance, & Management

Define your organization's data strategy & governance rules:

- Benchmark data maturity & assess internal capabilities
- Design & implement a data governance model (data structure, stakeholders, policies, & MDM/lineage)
- Catalog, characterize, & prioritize use cases
- Assess data quality, integrity, & completeness

Data Transformation & Architecture

Transform & organize your organization's data:

- Define data access & collection process (disparate data sources, pipeline)
- Connect, merge, reconcile, cleanse, & enrich data, as necessary
- Design & set up data quality monitoring platform
- Design & deploy data infrastructure (DWH, data lake(s), big data, cloud)

This presentation's focus

Data Analytics & Data Science (ML/AI)

Leverage analytics & data science:

- Develop & deploy priority, high-impact machine learning models
- Enable descriptive, predictive, & prescriptive capabilities (ML/AI, GenAI, NLP, etc.)
- Implement a continuous integration/continuous development model to enable a fast scale-up process

Data Reporting, Visualization, & Custom App Deployment

Engage business functions to understand data reporting & visualization needs:

- Design & implement data visualization, reporting, & dashboarding tools
- Provide meaningful visibility into key areas & improve decision making across the organization
- Develop & deploy custom app solutions aligned with client workflows and compliance needs

What data maturity enables

Questions?

Contact

Forvis Mazars

Kyle Harman
Senior Manager, Analytics
205.568.8170
kyle.harman@us.forvismazars.com

Philip Rago
Senior Consultant, Analytics
707.888.2496
philip.rago@us.forvismazars.com