



AZURE DATABRICKS

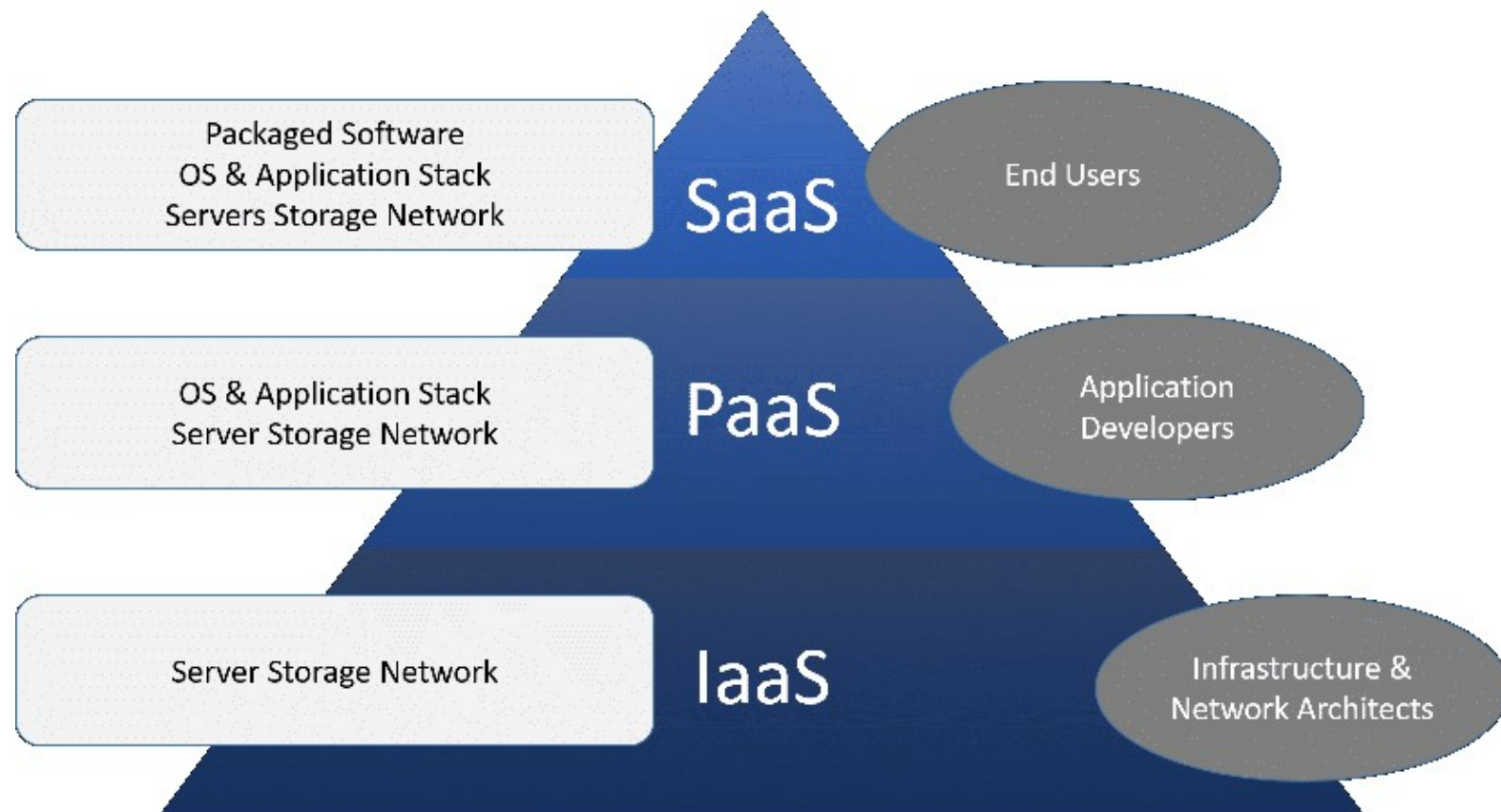
Data Analytics & Data Engineering on Azure Databricks

Cloud computing

- 'Cloud computing' is a term that indicates delivery of computing resources such as servers, storage, network etc. over the internet by a service provider and is accessible from anywhere and at any time.



Cloud computing service models





Microsoft Azure

Microsoft Azure

- Microsoft Azure is a cloud computing platform run by Microsoft, which offers access, management, and development of applications and services through global data centers.
- It provides a range of capabilities, including software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS).
- Microsoft Azure supports many programming languages, tools, and frameworks, including Microsoft-specific and third-party software and systems.

Microsoft Azure benefits

- Flexible
 - Move compute resources up and down as needed
- Open
 - Supports almost any OS, language, tool, or framework
- Reliable
 - 99.95% availability SLA and 24×7 tech support
- Global
 - Data housed in geo-synchronous data centers
- Economical
 - Only pay for what you use

Azure Pricing

- Azure pricing depends on many factors, including the type of service, the capacity required, the location, and the level of management.
- Azure offers a free tier, which provides free use of certain services for the first 12 months, and free forever usage of specific services.
- The majority of Azure services can be purchased under the “pay as you go” pricing model, which charges users based on actual usage. Additionally, Azure offers significant discounts for reserved instances (which require commitment of 1 or 3 years), and spot instances (virtual machines from Azure’s spare capacity, which can be terminated at short notice).

Azure Pricing – Free Tier

- Azure offers a free tier that lets companies use a variety of services free for 12 months (with a limited allowance), provides a credit for additional services in the first 30 days, and provides several services for free on an ongoing basis.
- **12 Months Free Use**
 - Azure provides the following services free for the first 12 months after creating an Azure account. Each service has a usage limit—for example, you can use a Windows Virtual Machine for 750 hours.

Compute Services Linux Virtual Machines Windows Virtual Machines	Storage Services Azure Managed Disks Azure Blob Storage Azure Files
Database Services Azure SQL Database (MS SQL Server) Cosmos DB (NoSQL)	AI and Analytics Services Computer Vision Text Analytics Translator Personalizer Language Understanding

Azure Pricing – Free Tier

Credit for first 30 days

- If you want to use a service that is not included in the free services, or if you exceed the service limits in the free tier, Azure provides a credit of \$200 which you can deduct from your first bill, during the first 30 days of usage.
- Beyond that, any further use of Azure services will be billed.

Azure charges after the free trial

- Certain Azure services are available free, within certain limitations, during the first 12 months.
- Azure also provides a \$200 credit for new customers. When the first 12 months end and/or when you finish using the \$200 credit, you will be charged for all Azure services you use.
- The only exception is services that are provided free forever

Free Azure Services

- The following services are always free, even after the first 12 months of use.
- Keep in mind that you are billed for use of other Azure resources you consume during the use of these services.
 - For example, when deploying containers using Azure Kubernetes Service (AKS), you are billed for the VMs/containers that are deployed by the service. But there is no charge for AKS itself.

Development Services Azure App Service DevTest Labs Azure DevOps	Serverless and Containers Azure Functions (free up to 1 million requests) Azure Kubernetes Service (AKS) Azure Container Instances Service Fabric
Messaging, Routing and Automation Event Grid Load Balancer Azure Automation	Networking Virtual Networks (VNets) Data transfer between VNets Inbound data transfer (unlimited) Outbound data transfer (up to 15GB)
Data Management and Search Data Factory Data Catalog Cognitive Search	Other Services Active Directory B2C Azure Security Center Azure Advisor

Azure pricing models

Microsoft offers three ways to pay for Azure VMs and other cloud resources:

- Pay as you go
 - Pay for services according to actual usage, billed per second, with no long-term commitment or upfront payments. You get complete flexibility to increase or decrease resources as needed. Azure VMs can be automatically scaled up and down using Azure's auto-scaling features.
 - Suitable for users who prefer flexibility and applications with volatile or short-term workloads.
- Reserved instances
 - Azure provides Reserved Virtual Machine Instances (RVMI)—virtual machines that are pre-purchased for one or three years in a specific region. Committing to reserved instances in advance grants a discount of up to 72% compared to pay-as-you-go prices.
- Spot instances
 - You can buy unused computing power at a discount of up to 90% compared to pay as you go prices. However, spot instances can be interrupted on short notice, so they are considered to be suitable only for workloads that can tolerate disruptions.
 - Suitable for distributed, fault tolerant applications, stateless applications, workloads that are not urgent, or are heavily parallelized.



Apache Spark

Apache Spark

- Apache Spark is an open-source, distributed processing system used for big data workloads.
- Apache Spark utilizes in-memory caching, and optimized query execution for fast analytic queries against data of any size.
- Apache Spark provides development APIs in Java, Scala, Python and R, and supports code reuse across multiple workloads—batch processing, interactive queries, real-time analytics, machine learning, and graph processing.



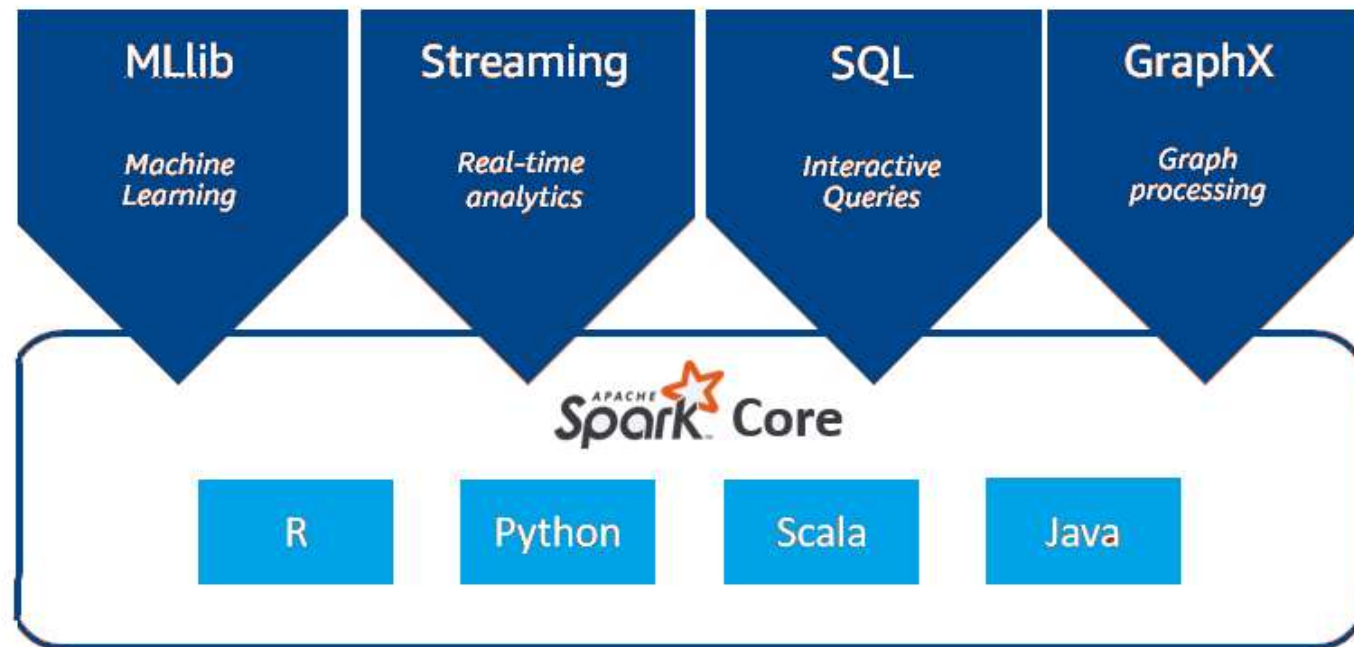
Apache Spark

- Hadoop MapReduce is a programming model for processing big data sets with a parallel, distributed algorithm. MapReduce uses sequential multi-step process to run a job. Because each step requires a disk read, and write, MapReduce jobs are slower due to the latency of disk I/O.
- Spark was created to address the limitations to MapReduce, by doing processing in-memory, reducing the number of steps in a job, and by reusing data across multiple parallel operations.
- With Spark, only one-step is needed where data is read into memory, operations performed, and the results written back—resulting in a much faster execution.

Apache Spark

- Spark also reuses data by using an in-memory cache to greatly speed up machine learning algorithms that repeatedly call a function on the same dataset.
- Data re-use is accomplished through the creation of DataFrames, an abstraction over RDD, which is a collection of objects that is cached in memory, and reused in multiple Spark operations. This dramatically lowers the latency making Spark multiple times faster than MapReduce, especially when doing machine learning, and interactive analytics.

Apache Spark Workloads

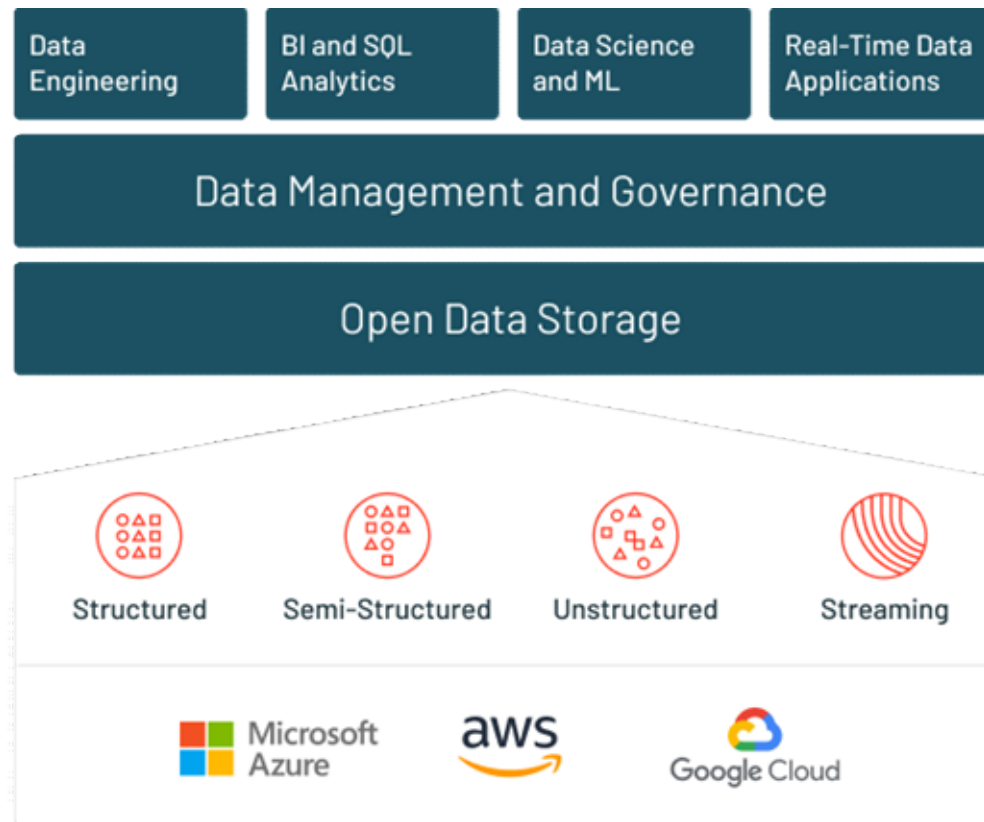




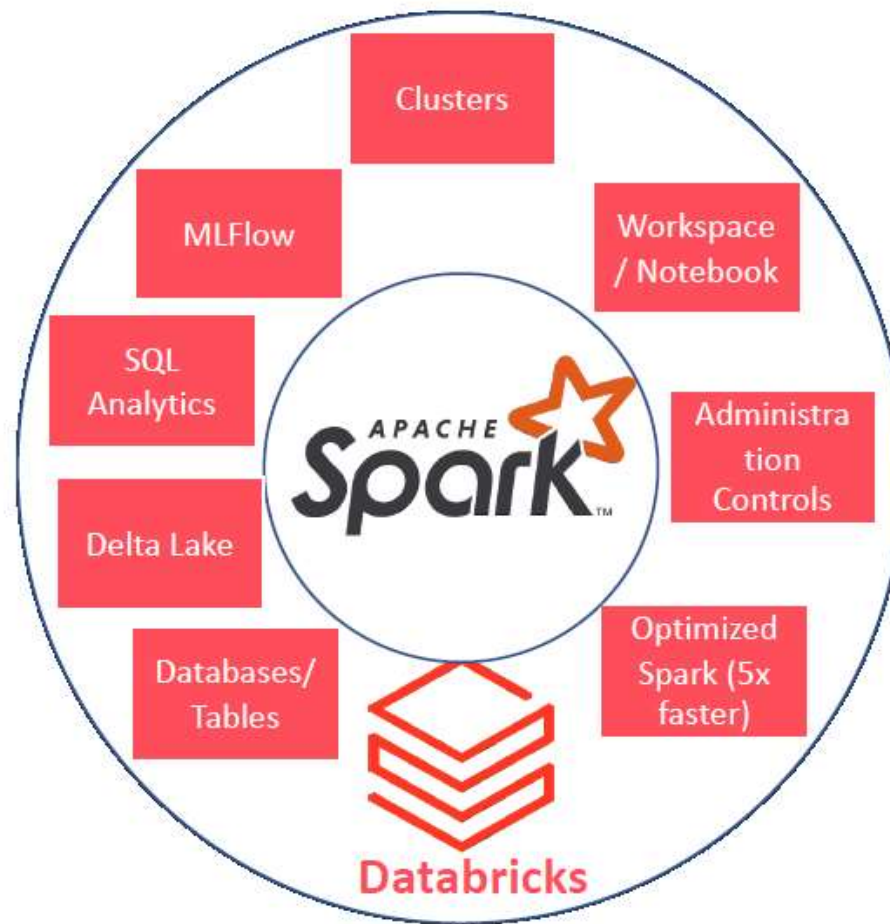
Azure Databricks

What is Databricks ?

- **Databricks** is a cloud-based platform for managing and analyzing large datasets using Apache Spark.



What is Databricks ?



Databricks features

- **Unified Workspace**

- Databricks provides a single platform for data scientists, engineers, and business analysts to work together and collaborate on data projects.

- **Scalability and Flexibility**

- Databricks is designed to be highly scalable so that they can easily handle large amounts of data.
- It can also be flexibly configured to support different workloads, such as batch processing, real-time streaming, or machine learning.
- This makes it a good choice for organizations that need to process data at different scales, or that have complex data pipelines.

- **Integrated Tools and Services**

- Databricks comes with a range of tools and services for working with big data, including support for various data formats, integration with popular data science libraries and frameworks, and tools for data preparation and analysis.
- This makes it easier to build and deploy data-driven applications, without having to worry about setting up and managing complex infrastructure.

- **Security and Compliance**

- Databricks offers a range of features to help ensure that data is handled securely and in compliance with relevant regulations.
- This includes support for encryption, role-based access control, and auditing, as well as integration with popular security and compliance tools.

Databricks Use Cases

- **Data Warehousing**

- Databricks can be used to store and manage large amounts of data from multiple sources, and provide fast and efficient access to the data for analysis and other purposes.

- **Data Preparation**

- Databricks provides tools and services for cleaning, transforming, and enriching data, making it easier to prepare data for analysis or other uses.

- **Data Analysis**

- Databricks offers a range of tools and services for exploring, visualizing, and analyzing data, so that users can gain insights and identify trends in the data.

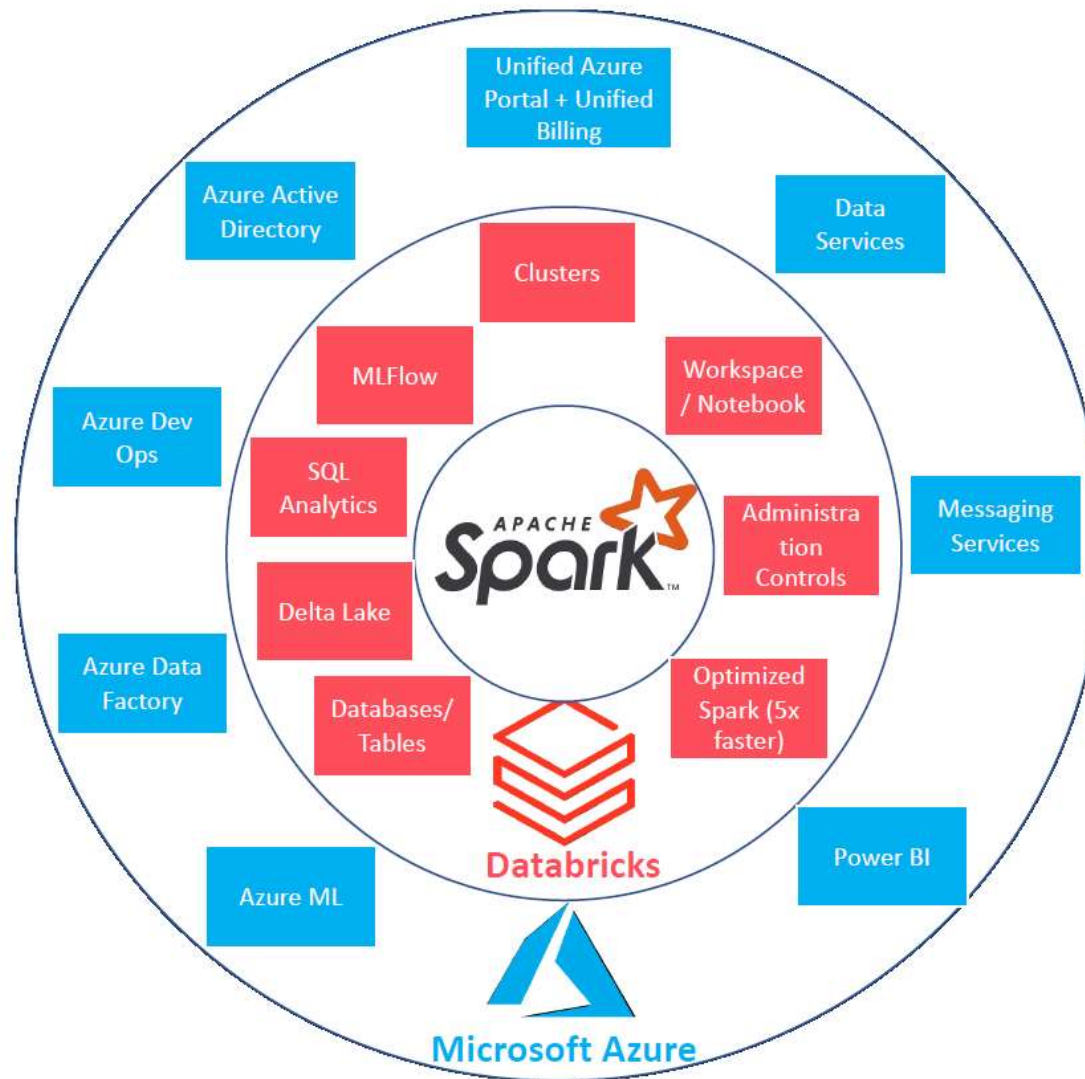
- **Machine Learning**

- Databricks supports popular machine learning libraries and frameworks, such as TensorFlow, Keras, and PyTorch, making it easier to build and train ML models on large datasets.

- **Real-time Data Processing**

- Databricks can be used to process streaming data in real-time so that users can take action based on the data as it arrives.

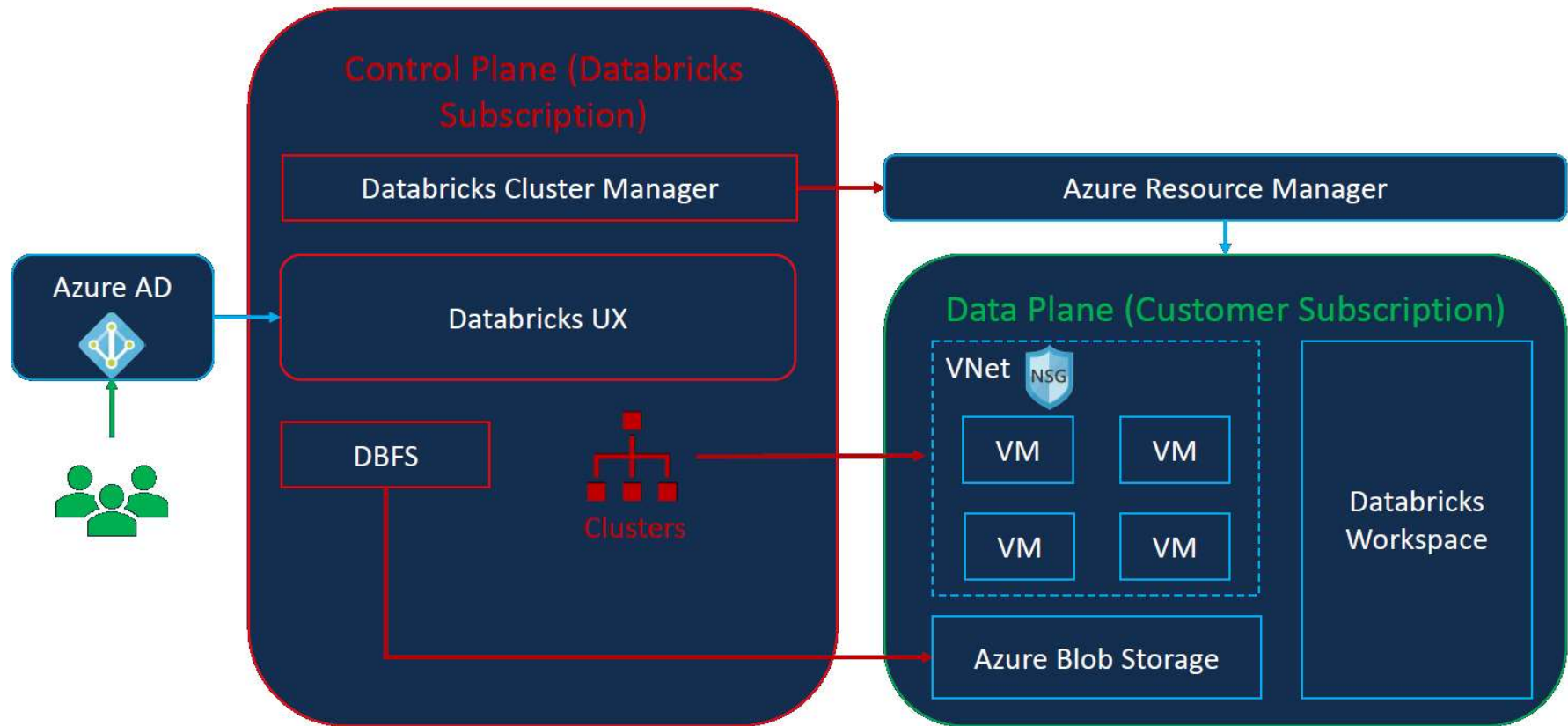
Azure Databricks



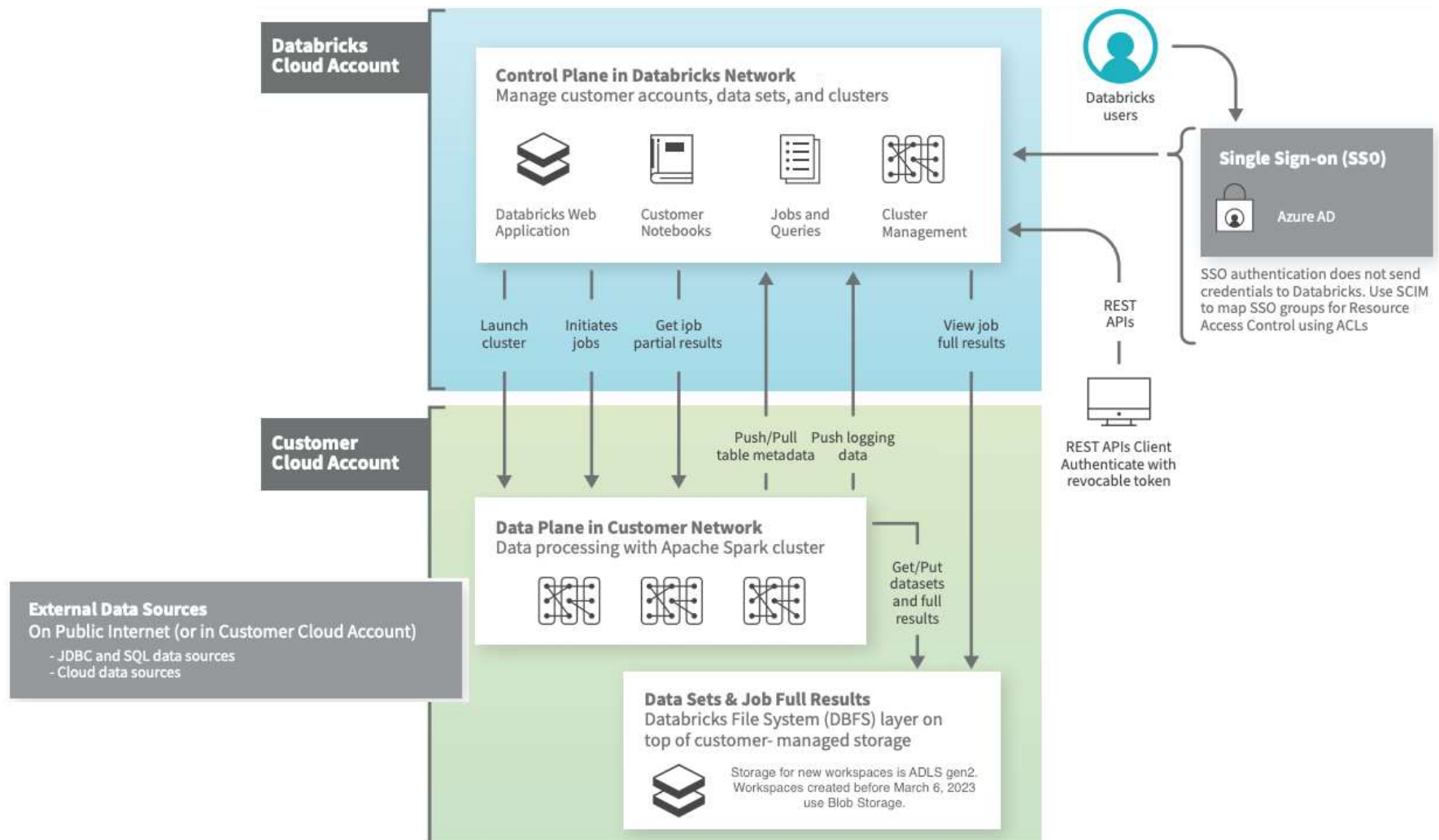
Azure Databricks architecture

- Azure Databricks is structured to enable secure cross-functional team collaboration while keeping a significant amount of backend services managed by Azure Databricks so you can stay focused on your data science, data analytics, and data engineering tasks.
- Azure Databricks operates out of a ***control plane*** and a ***data plane***.
- - **Control plane**
 - The control plane includes the backend services that Azure Databricks manages in its own Azure account. Notebook commands and many other workspace configurations are stored in the control plane and encrypted at rest.
 - **Data plane**
 - Your Azure account manages the data plane, and is where your data resides. This is also where data is processed. Use Azure Databricks connectors to connect clusters to external data sources outside of your Azure account to ingest data, or for storage. You can also ingest data from external streaming data sources, such as events data, streaming data, IoT data, and more.

Azure Databricks architecture



Azure Databricks architecture



Getting started with Azure Databricks

Before you begin, do the following:

- You must have an Azure subscription that isn't a Free Trial Subscription. If you have a free account, complete the following steps:
 - Go to your profile and change your subscription to pay-as-you-go.
 - Remove the spending limit.
 - Request a quota increase for vCPUs in your region.
- Sign in to the Azure portal.
- You must be an Azure Contributor or Owner.

Reference URL: <https://learn.microsoft.com/en-us/azure/databricks/getting-started/>

Azure Databricks Workspace

- A workspace is an environment for accessing all of your Azure Databricks assets. A workspace organizes objects (notebooks, libraries, dashboards, and experiments) into folders and provides access to data objects and computational resources.
- Databricks recommends you deploy your first Azure Databricks workspace using the Azure portal. You can also deploy Azure Databricks with one of the following options:
 - Azure CLI
 - Powershell
 - Azure Resource Manager (ARM) template
 - An ARM template is a JavaScript Object Notation (JSON) file that defines the infrastructure and configuration for your project. The template uses declarative syntax, which lets you state what you intend to deploy without having to write the sequence of programming commands to create it.
 - Bicep
 - Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Lab 1 – Setup Azure account

Instruction document: L01-Setup-Azure-Account.txt

In this lab we do the following:

- Create a 'Pay as you go' Azure account.
- Login to Azure portal with your account.
- Check your subscription and register for a few 'Resource providers'
- Update the quotas for the compute service in 'East US' region

Lab 2 - Getting started with Azure Databricks

Instruction document: [L02-Getting-Started-With-AzureDatabricks.txt](#)

In this lab we do the following:

- Create and launch 'Azure Databricks' workspace
- Create a single node Databricks cluster
- Upload data files
- Create a new Databricks Notebook
- Create a Spark application using Azure Databricks Notebook
- Export and Import of Azure Databricks Notebooks
- Delete the Databricks cluster
- Delete Azure Databricks Workspace

Azure Resource Group

- Resource group is a container that holds resources for an Azure solution.
 - The resource group can include all the resources for the solution, or only those resources that you want to manage as a group.
 - You decide how you want to allocate resources to resource groups based on what makes the most sense for your organization.
 - Generally, add resources that share the same lifecycle to the same resource group so you can easily deploy, update, and delete them as a group.
- The resource group stores metadata about the resources.
 - Specifying a location for the resource group means specifying where that metadata is stored.
 - For compliance reasons, you may need to ensure that your data is stored in a particular region.

Azure Storage Account

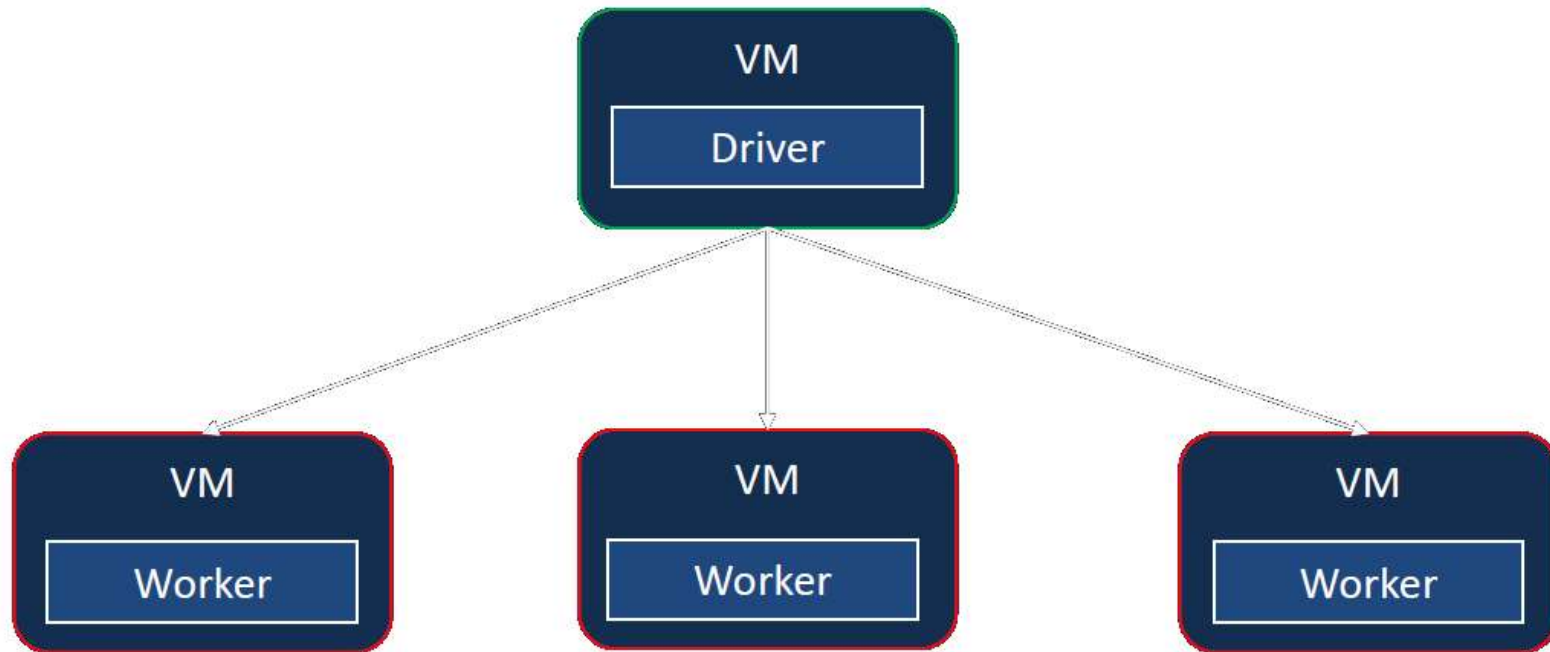
- An Azure storage account contains all of your Azure Storage data objects: blobs, files, queues, and tables. It provides a unique namespace for your Azure Storage data that's accessible from anywhere in the world over HTTP or HTTPS.
- Data in your storage account is durable and highly available, secure, and massively scalable.
- Types of storage accounts
 - Standard general-purpose v2
 - Premium block blobs
 - Premium file shares
 - Premium page blobs

For more details: <https://learn.microsoft.com/en-us/azure/storage/common/storage-account-overview#types-of-storage-accounts>

Databricks Cluster

- A Databricks cluster is a set of computation resources and configurations on which you run data engineering, data science, and data analytics workloads.
- You run these workloads as a set of commands in a **notebook** or as an **automated job**.
 - Databricks makes a distinction between all-purpose clusters and job clusters.
 - All-purpose clusters are used to analyze data collaboratively using interactive notebooks.
 - Job clusters are used to run fast and robust automated jobs.

Databricks Cluster



Multi-node Databricks Cluster

Databricks cluster types

There are two types of clusters in Databricks:

All Purpose	Job Cluster
Created manually	Created by Jobs
Persistent	Terminated at the end of the job
Suitable for interactive workloads	Suitable for automated workloads
Shared among many users	Isolated just for the job
Expensive to run	Cheaper to run

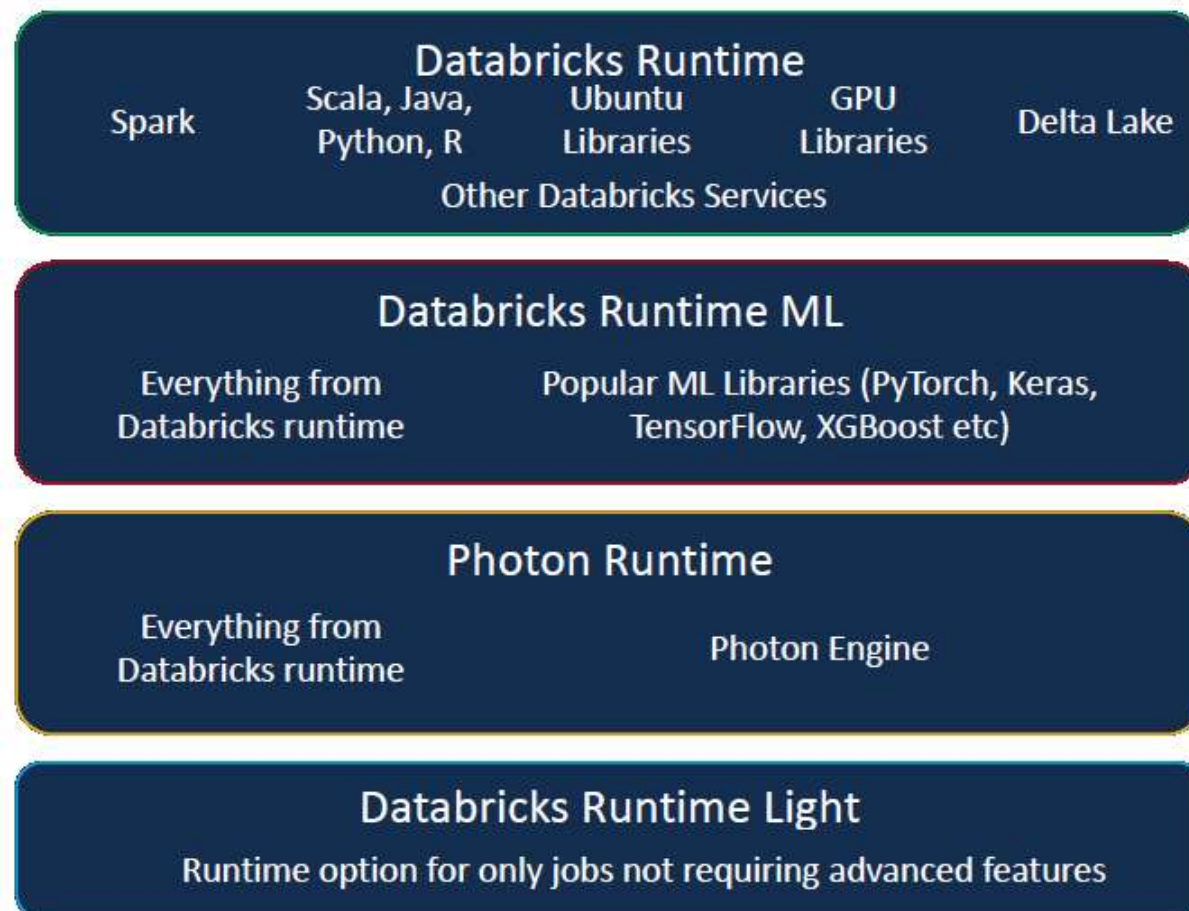
Databricks cluster access modes

Cluster access mode is a security feature that determines who can use a cluster and what data they can access via the cluster.

Access Mode	Visible to user	UC Support	Supported Languages	Notes
Single user	Always	Yes	Python, SQL, Scala, R	Can be assigned to and used by a single user. See single user limitations .
Shared	Always (Premium plan required)	Yes	Python (on Databricks Runtime 11.1 and above), SQL	Can be used by multiple users with data isolation among users. See shared limitations .
No Isolation Shared	Admins can hide this cluster type by enforcing user isolation in the admin settings page.	No	Python, SQL, Scala, R	There is a related account-level setting for No Isolation Shared clusters.
Custom	Hidden (For all new clusters)	No	Python, SQL, Scala, R	This option is shown only if you have existing clusters without a specified access mode.

Databricks runtime options

Databricks Runtime is the set of software artifacts that run on the clusters of machines managed by Databricks.



Compute policy

A compute policy defines limits on the attributes available during compute creation

Policy ?

Unrestricted | ▼

Unrestricted

Personal Compute

Power User Compute

Legacy Shared Compute

Performance

Databricks runtime version ?

Runtime: 13.3 LTS (Scala 2.12, Spark 3.4.1) | ▼

All-purpose compute

Job compute

SQL warehouses

Pools

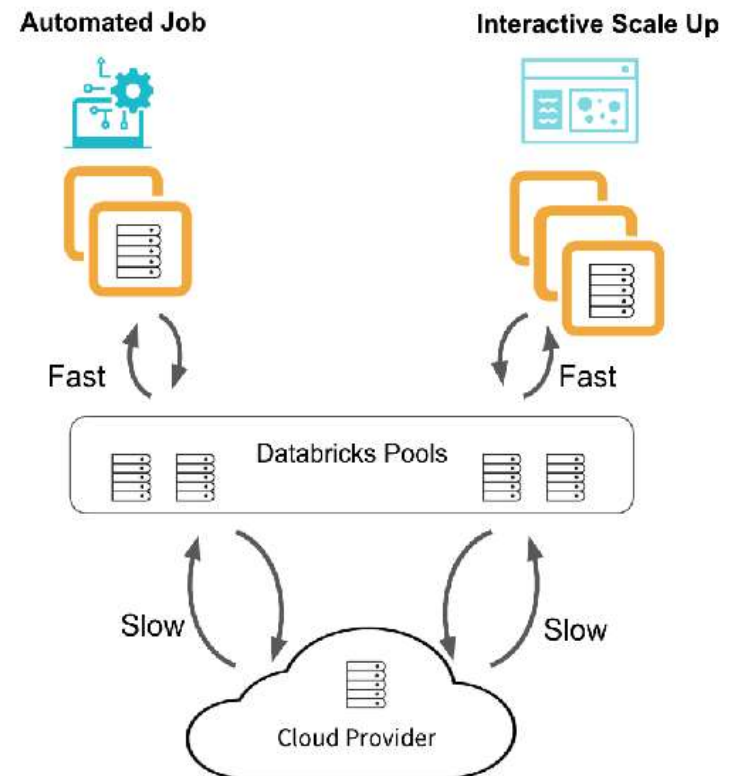
Policies ?

Q Filter policies

Name ↕	Definitions	Policy Family
Job Compute	9	Job Compute
Legacy Shared Compute	13	Legacy Shared Compute
Personal Compute	11	Personal Compute
Power User Compute	12	Power User Compute
Shared Compute ⓘ	13	Shared Compute

Cluster pools

- Azure Databricks cluster pools are a set of idle, ready-to-use instances.
- When cluster nodes are created using the idle instances, cluster start and auto-scaling times are reduced.
- If the pool has no idle instances, the pool expands by allocating a new instance from the instance provider in order to accommodate the cluster's request.
- When a cluster releases an instance, it returns to the pool and is free for another cluster to use. Only clusters attached to a pool can use that pool's idle instances.



Create pools based on workloads

- If your driver node and worker nodes have different requirements, create a different pool for each.
- You can minimize instance acquisition time by creating a pool for each instance type and Azure Databricks runtime your organization commonly uses.
 - For example, if most data engineering clusters use instance type A, data science clusters use instance type B, and analytics clusters use instance type C, create a pool with each instance type.
- Configure pools to use on-demand instances for jobs with short execution times and strict execution time requirements. Use on-demand instances to prevent acquired instances from being lost to a higher bidder on the spot market.
- Configure pools to use spot instances for clusters that support interactive development or jobs that prioritize cost savings over reliability.

Using the pool for your cluster

Clusters are attached to the pool. You can specify a different pool for the driver node and worker nodes, or use the same pool for both.

☒ Multi node ☐ Single node

Access mode ?

Single user access ?

Single user

Kanakaraju Yarakaraju (kanakaraj...)

Performance

Databricks runtime version ?

Runtime: 12.2 LTS (Scala 2.12, Spark 3.3.2)

☐ Use Photon Acceleration ?

Worker type ?

Min workers

Max workers

CTS Demo Pool

Standard_F4

2

8

Lab 3 – All purpose cluster, job cluster & instance pool

Instruction document: [L03-Allpurpose-and-Job-Clusters-Pools.txt](#)

In this lab we do the following:

- Spin up an all-purpose compute cluster
- Create a notebook
- Create a job and attach the notebook to it
- Run the job using the all-purpose compute cluster
- Modify the job to run it using job cluster
- Create an instance pool
- Create an all-purpose compute cluster in the pool
- Run the job using the all-purpose compute using the pool.

Databricks Databases & Tables

- Databricks Database is a collection of tables.
- Databricks Table is a collection of structured data.
 - We can cache, filter, and perform any operations supported by Spark DataFrames on Azure Databricks tables and query tables with Spark APIs and Spark SQL.
 - There are two types of tables.
 1. Global Table
 - A global table is available across all clusters.
 - Databricks registers global tables to either the Hive metastore or an external metastore.
 - These are persistent tables (i.e. not temporary tables)
 2. Local Table
 - A local table is not accessible from other clusters
 - Not registered in the Hive metastore. (i.e. not created in any database)
 - This is also known as a temporary view.

Databricks File System (DBFS)

- DBFS is a distributed file system mounted into a Databricks workspace and available on Databricks clusters.
- DBFS is an abstraction on top of scalable object storage that maps file-system calls to native cloud storage API calls.
- DBFS Root is the default storage for a Databricks workspace created during workspace deployment.
 - DBFS Root contains a folder called “**FileStore**” into which we can upload files from Databricks UI.
 - DBFS Root contains two folders
 - `dbfs:/databricks-datasets` → Has sample datasets
 - `dbfs:/databricks-results` → Stores results of the queries executed.
 - DBFS Root is deleted when the workspace is deleted. Hence, it is advised not to store any customer data here. Instead, store customer data in Azure Blob storage or ADLS.

What can you do with DBFS?

- DBFS provides convenience by mapping cloud object storage URIs to relative paths.
- Allows you to interact with object storage using directory and file semantics instead of cloud-specific API commands.
- Allows you to mount cloud object storage locations so that you can map storage credentials to paths in the Databricks workspace.
- Simplifies the process of persisting files to object storage, allowing virtual machines and attached volume storage to be safely deleted on cluster termination.
- Provides a convenient location for storing init scripts, JARs, libraries, and configurations for cluster initialization.

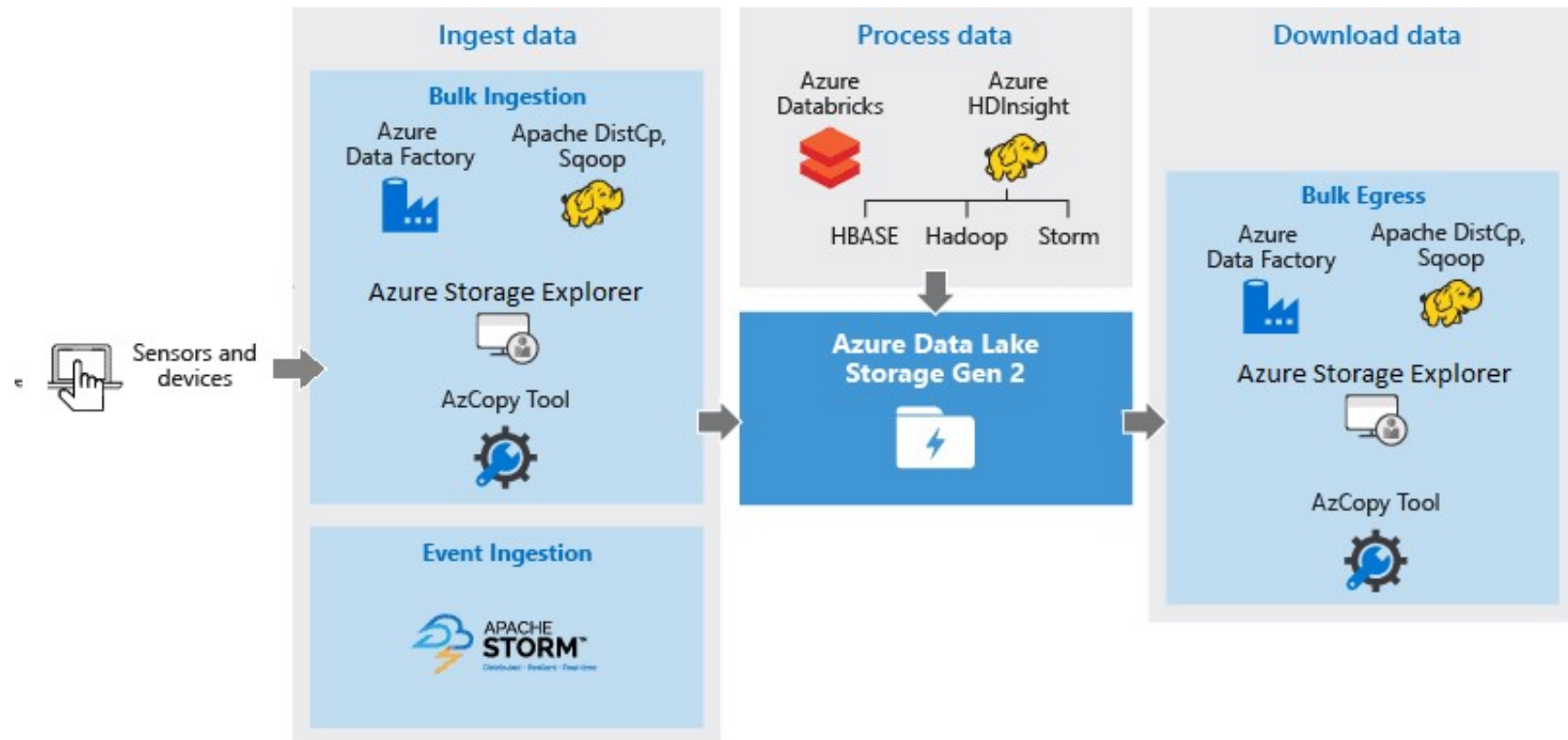
Azure Blob Storage

- **Azure Blob Storage** is an **object storage solution for the cloud**. It is optimized for storing massive amounts of unstructured data, such as text or binary data in a single hierarchy, also known as a flat namespace.
- Blob Storage can manage thousands of simultaneous uploads, enormous amounts of video data, constantly growing log files, and can be reached from anywhere with an internet connection via HTTP/HTTPS.
- Blobs aren't limited to common file formats. A blob could contain gigabytes of binary data streamed from a scientific instrument, an encrypted message for another application, or data in a custom format for an application. Azure takes care of the physical storage needs on your behalf.

Azure Data Lake Storage (ADLS)

- ADLS is a comprehensive, scalable, and cost-effective data lake solution for high-performance big data analytics built into Azure.
- ADLS Gen1 is an enterprise-wide hyper-scale repository for big data analytic workloads. It enables you to capture data of any size, type, and ingestion speed in one single place for operational and exploratory analytics.
- ADLS Gen2 is a set of capabilities dedicated to big data analytics, built on Azure Blob. It converges the capabilities of ADLS Gen1 with Azure Blob storage.
 - **ADLS Gen2 = Azure Blob Storage + ADLS Gen1**
- ADLS Gen2 provides file system semantics, file-level security, and scale, which are inherited from ADLS Gen1. All these capabilities are built on Blob storage resulting in low cost, tiered access, high security, high availability, and durability.

Azure Data Lake Storage (ADLS)



Accessing ADLS from Databricks

There are different authentication methods to access the data in the ADLS from Databricks workspace.

- Access Data Lake using Access Keys
- Access Data Lake using Shared Access Signature (SAS) token
- Access Data Lake using Service Principal
- Access Data Lake using Cluster scoped authentication
- Access Data Lake using Azure AD credential pass-through

Reference: <https://learn.microsoft.com/en-us/azure/databricks/storage/azure-storage>

Lab 4 – Access ADLS using access token

Instruction document: L04-ADLS-Gen2-Access-Patterns.txt

In this lab we do the following:

- Open the storage account and upload sample data into a container
- Disable 'soft delete' for blobs and containers
- Spin an 'All purpose' cluster in your workspace
- Get the 'Access key' of your storage account.
- Set the spark configuration for access token
- Access the ADLS container files from notebook using access keys

Lab 4b – Access ADLS using SAS token

Instruction document: L04-ADLS-Gen2-Access-Patterns.txt

In this lab we do the following:

- Open the storage account and upload sample data into a container
- Disable 'soft delete' for blobs and containers
- Spin an 'All purpose' cluster in your workspace
- Generate the SAS (shared access signature) token for your container
- Set the spark configuration for SAS token
- Access the ADLS container files from notebook using SAS token

Lab 4c – Access ADLS using service principal

Instruction document: L04-ADLS-Gen2-Access-Patterns.txt

In this lab we do the following:

- Open the storage account and upload sample data into a container
- Disable 'soft delete' for blobs and containers
- Spin an 'All purpose' cluster in your workspace
- Register Microsoft Entra ID Application / Service Principal
- Generate a secret/password for the Application
- Assign Role 'Storage Blob Data Contributor' to the Data Lake.
- Access the ADLS container files from notebook using Service principal

Lab 4d – Access ADLS using cluster scoped credentials

Instruction document: L04-ADLS-Gen2-Access-Patterns.txt

In this lab we do the following:

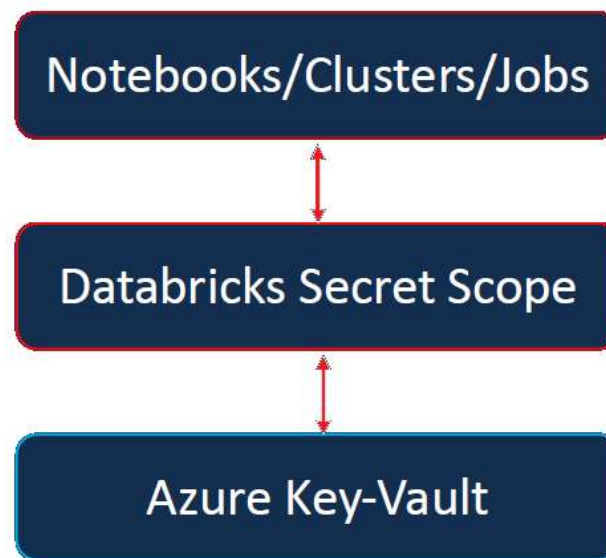
- Open the storage account and upload sample data into a container
- Disable 'soft delete' for blobs and containers
- Spin an 'All purpose' cluster in your workspace
- Edit the cluster to add the access-key in Spark configurations.
- Access the ADLS container files without requiring any access configurations in the notebook.

Databricks Secret Scope

- A secret scope is collection of secrets identified by a name.
 - Managing secrets begins with creating a secret scope.
- Secret scopes help store the credentials securely and reference them in notebooks, clusters and jobs when required.
- Two Types:
 - Databricks backed secret scope
 - Azure key-vault backed secret scope
- A Databricks-backed secret scope is stored in an encrypted database owned and managed by Databricks.
 - This can be used only using Databricks CLI or Secret API

Azure Key Vault

- Azure Key Vault is a cloud service for securely storing and accessing secrets.
 - A secret is anything that you want to tightly control access to, such as API keys, passwords, certificates, or cryptographic keys.



Lab 5 – Secure ASDL access using Secret scope

Instruction document: [L05-Securing-secrets-using-Secret-Scope.txt](#)

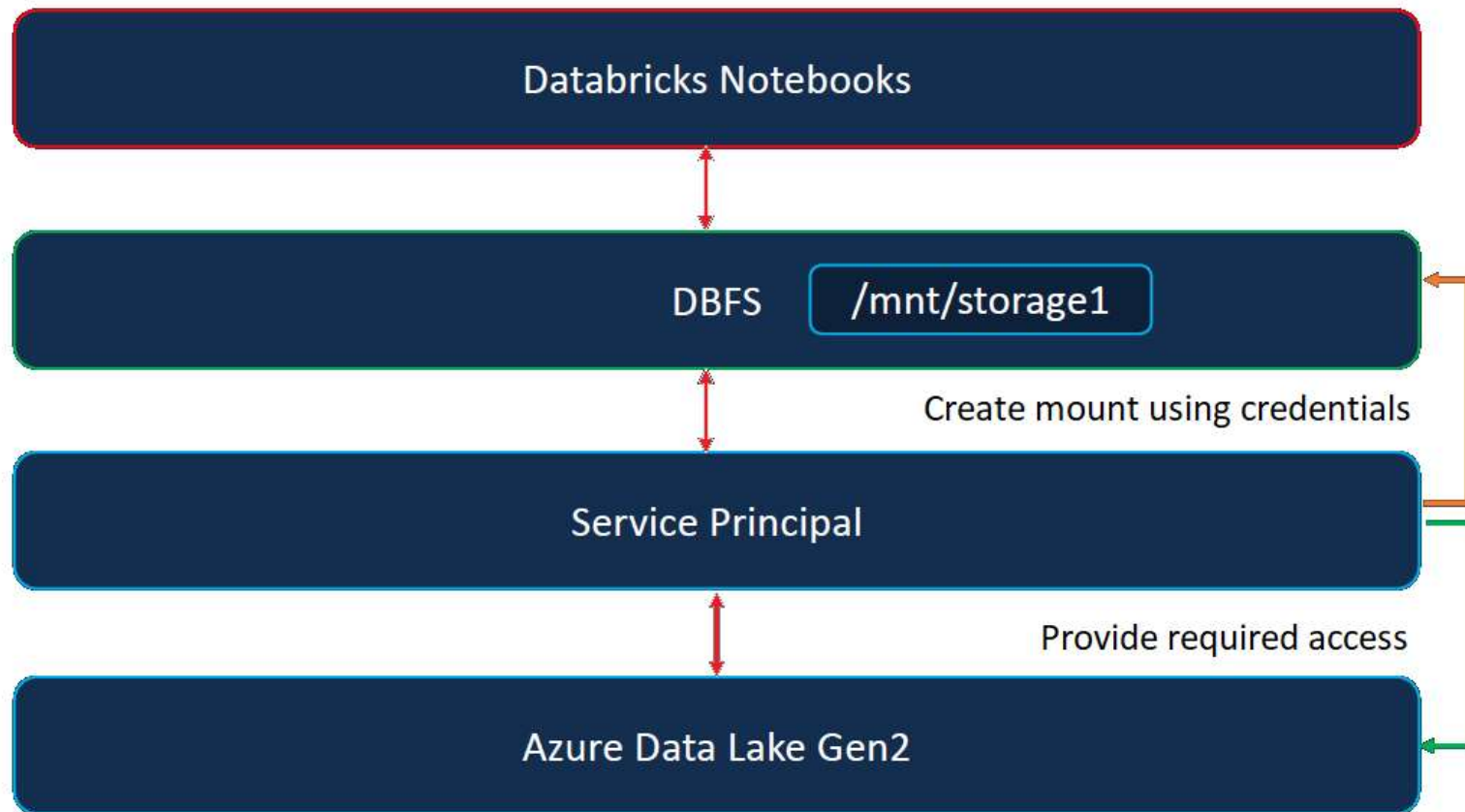
In this lab we do the following:

- Understand Azure Key Vault service to store secrets
- Create a secret for access-key in Azure Key vault
- Create a Databricks secret scope
- Link the secret scope to the key vault
- Understand **dbutils.secrets** utility
- Refactor the Lab-9 code to replace the hard-coded secret values using secrets.

Databricks Mounts

- Databricks allows you to mount Azure Blob storage or ADLS as a mount point (/mnt) in DBFS by specifying the required credentials.
 - Once mounted, we can access the data without requiring credentials.
 - We can access files using file semantics rather than storage URLs (e.g. /mnt/storage1)
 - As the files are stored to object storage (e.g. Azure Blob), you get all the benefits of cloud object storage.
- Databricks Mount is the recommended solution for accessing Azure Storage until the introduction of Unity Catalog.

Databricks Mounts



Lab 6 – Mount ADLS on Azure Databricks

Instruction document: [L06-Mount-ADLS-using-Service-Principal.txt](#)

In this lab we do the following:

- Launch a new Databricks Workspace
- Register an Azure Active Directory Application
- Create Databricks Secret for AD Application Client Secret
- Create ADLS storage account
- Assign IAM Role on Storage Account to Azure AD Application
- Create 'ADLS File System' and upload data
- Mount ADLS Storage Account on to Azure Databricks cluster
- Run PySpark code on the mounted datasets
- Un-mount the mount point



Introduction to Delta Lake

Data Warehouse, Data Lake & Data Lakehouse Concepts

Relational databases

- In 1970, EF Codd introduced the concept of looking at data as logical relations, independent of the physical data storage. This logical relation between data entities became known as a *database model* or *schema*. Codd's writings led to the birth of the **relational database**.
- Relational databases and their underlying SQL language became the standard storage technology for enterprise applications throughout the 1980s and 1990s. One of the main reasons behind this popularity was that relational databases offered a concept called **transactions**.
- A database transaction is a sequence of operations on a database that satisfies the ACID properties, ACID is an acronym which stands for four properties:
 - Atomicity
 - Consistency
 - Isolation
 - Durability

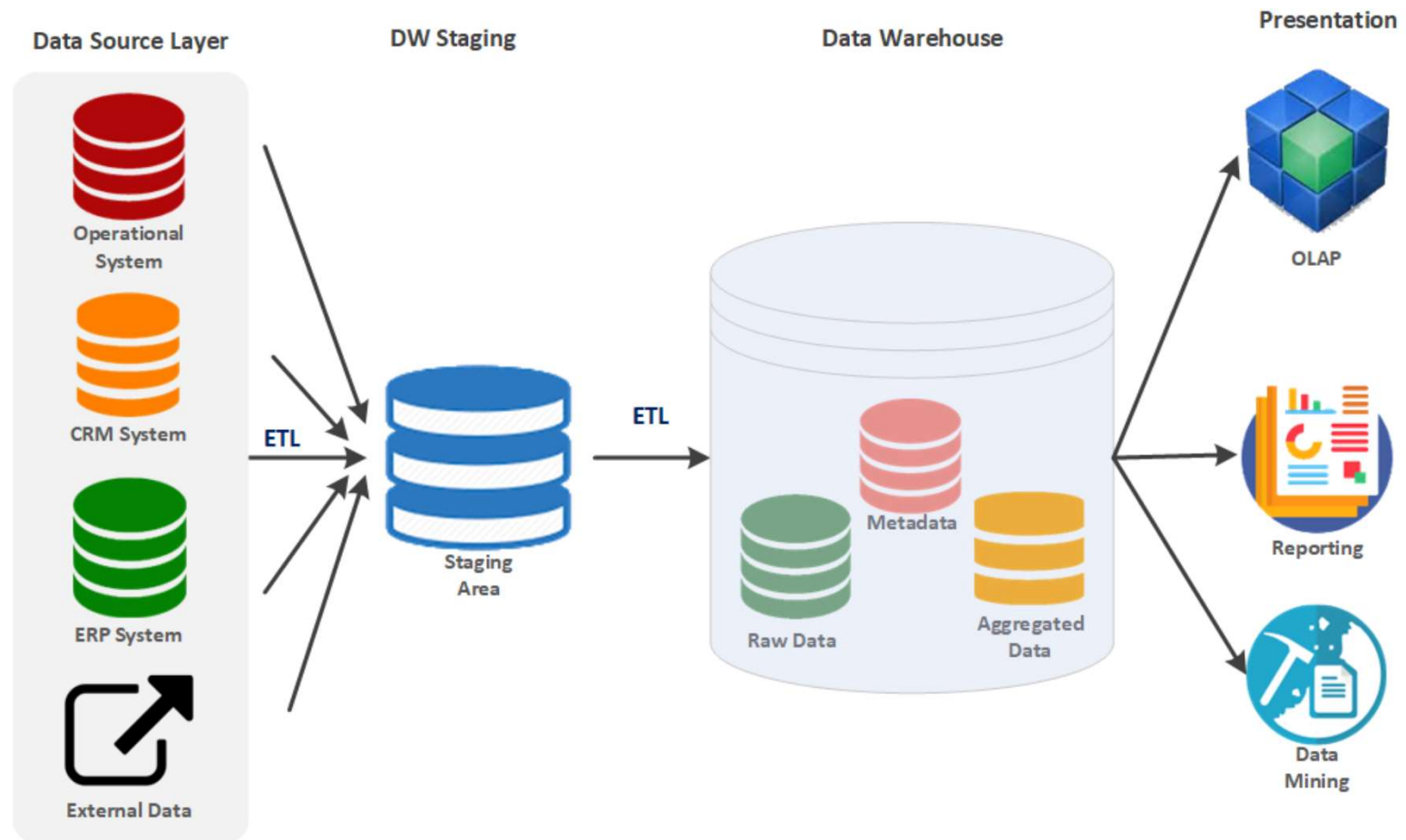
ACID properties

- **Atomicity** ensures that all changes made to the database are executed as a single operation.
 - For example, when I use my online banking to transfer money, the atomicity property will guarantee that the operation will only succeed when the money is deducted from one account and added to the other. The complete operation will either succeed or fail as a unit.
- **Consistency** property guarantees that database transitions from one consistent state of the transaction to another consistent state at the end of the transaction.
 - The transfer of the money would only happen if my savings account had sufficient funds. If not, the transaction would fail, and the balances would stay in their original, consistent state
- **Isolation** ensures that concurrent operations that are happening within the database are not affecting each other.
 - This property ensures that when multiple transactions are executed concurrently, their operations do not interfere with each other
- **Durability** refers to the persistence of committed transactions. It guarantees that once a transaction is completed successfully, it will result in a permanent state even in the event of a system failure.
 - Durability will ensure that updates made to both my savings and checking account are persistent and can survive a potential system failure.

Data warehouse

- Enterprise applications were using the RDBMS technology very effectively. Flagship products such as SAP and Salesforce would collect and maintain massive amounts of data.
- Enterprise applications would store the data in their own, proprietary formats, leading to the rise of ***data silos***. These data silos were owned and controlled by one department or business unit.
- Over time, organizations recognized the need to develop an **Enterprise view across these different data silos**, leading to the rise of ***data warehouses***.
- **Data warehouse** is a **central relational repository** of integrated, historical data from **multiple data sources** that presents a single integrated, historical view of the business with a **unified schema, covering all perspectives of the enterprise**.

Data warehouse

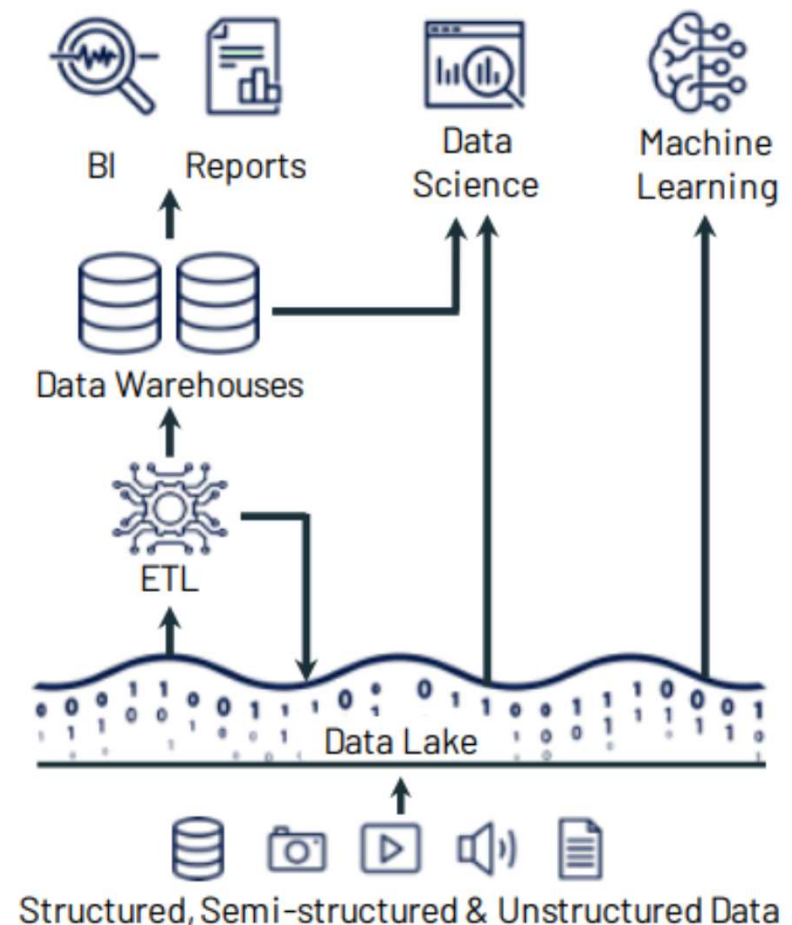


The 'Big Data' problem

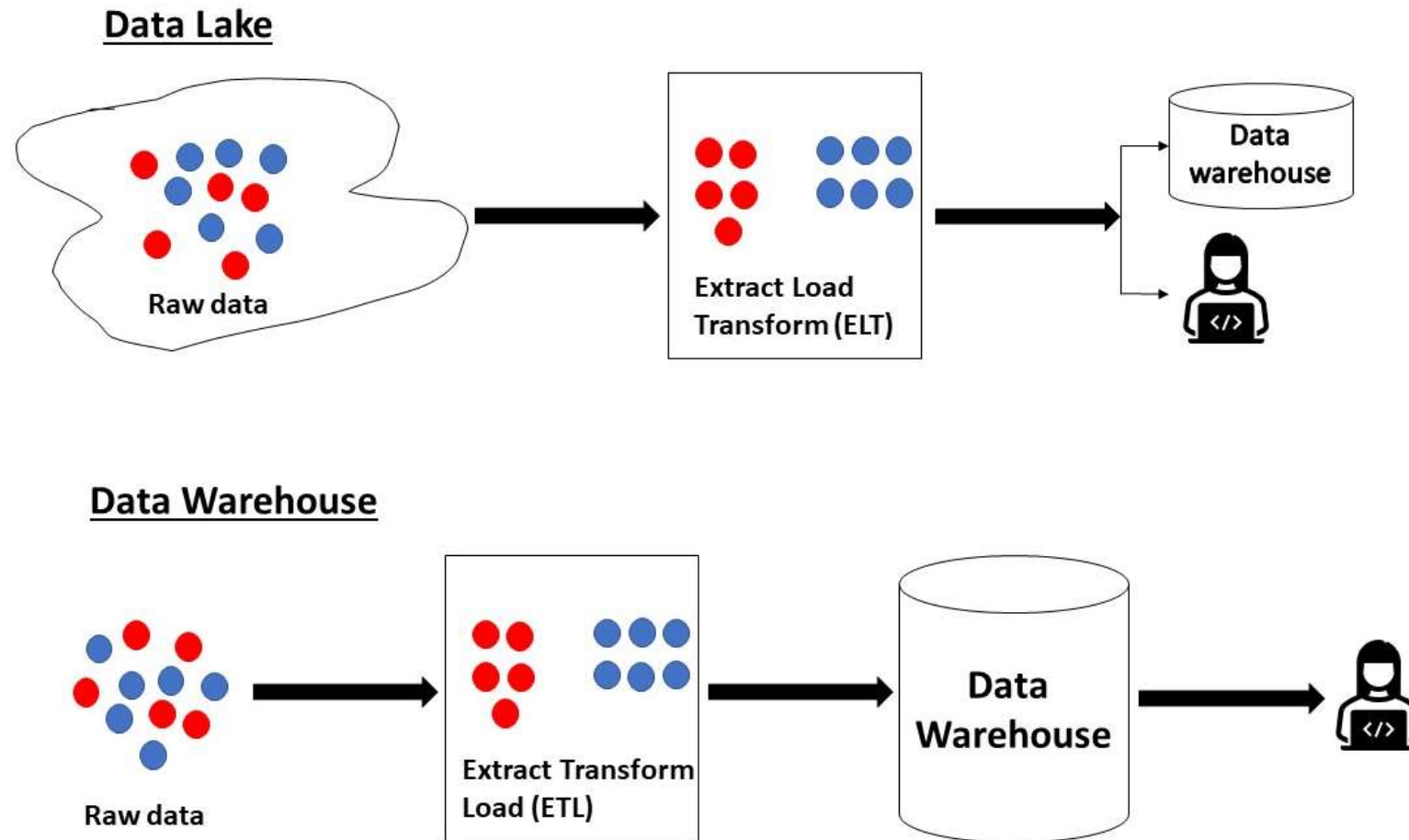
- The fast rise of the internet, social media and the availability of multi-media devices such as smart phones disrupted the traditional data landscape, giving rise to the term **big data**.
- Big data is defined as data that arrives in ever higher **volumes**, with more **velocity**, and a greater **variety** of formats and has higher **veracity**. These are known as the four Vs of data.
- Data Warehouses have a hard time addressing these four Vs.

Data Lake

- A **Data Lake** is a *cost-effective central repository to store structured, semi-structured or unstructured data at any scale, in the form of files and blobs.*
- The initial data lakes and big data solutions were built with on-premises clusters, based upon the Apache **Hadoop** open-source set of frameworks. Hadoop was used to efficiently store and process large datasets ranging in GBs to PBs data.
- Starting in 2015, **cloud data lakes** such as **S3, ADLS, and GCS** started replacing HDFS. These cloud-based storage systems have superior SLAs, offer geo-replication and most importantly, offer extremely low cost with the option to utilize even lower-cost cold storage for archival purposes.



Data Lake & Data Warehouse



Benefits of Data Lake

- A data lake architecture enables the consolidation of an organization's data assets into one central location.
- They use open formats, such as Parquet and Avro. These formats enable easy interoperability, especially in our open-source ecosystem.
- Data lakes are deployed on mature cloud storage sub-systems, allowing them to benefit from the scalability, monitoring, ease of deployment, and the low storage costs associated with these systems.
- Unlike data warehouses, data lakes support all data types, including semi-structured and unstructured data, enabling workloads such as media processing.
- Because of their high throughput ingress channels, they are very well suited for streaming use cases, such as the ingestion of IoT sensor data, media streaming, or Web clickstreams.

Challenges with Data Lake

As data lakes become more popular and widely used, organizations started to recognize some challenges with traditional data lakes.

- While the underlying cloud storage is relatively inexpensive, building and maintaining an effective data lake requires expert skills.
- While it is easy to ingest data into the data lake in its raw form, transforming the data into a form that can deliver business values can be very expensive.
- Traditional data lakes have low query performance, so they cannot be used for interactive queries.
 - As a result, the organization's data teams must still transform and load the data into something like a data warehouse, resulting in an extended time to value.
 - This resulted in a data lake + warehouse architecture. This architecture continues to be dominant in the industry
- Data lakes do not offer any kind of transactional guarantees.

Challenges with Data Lake

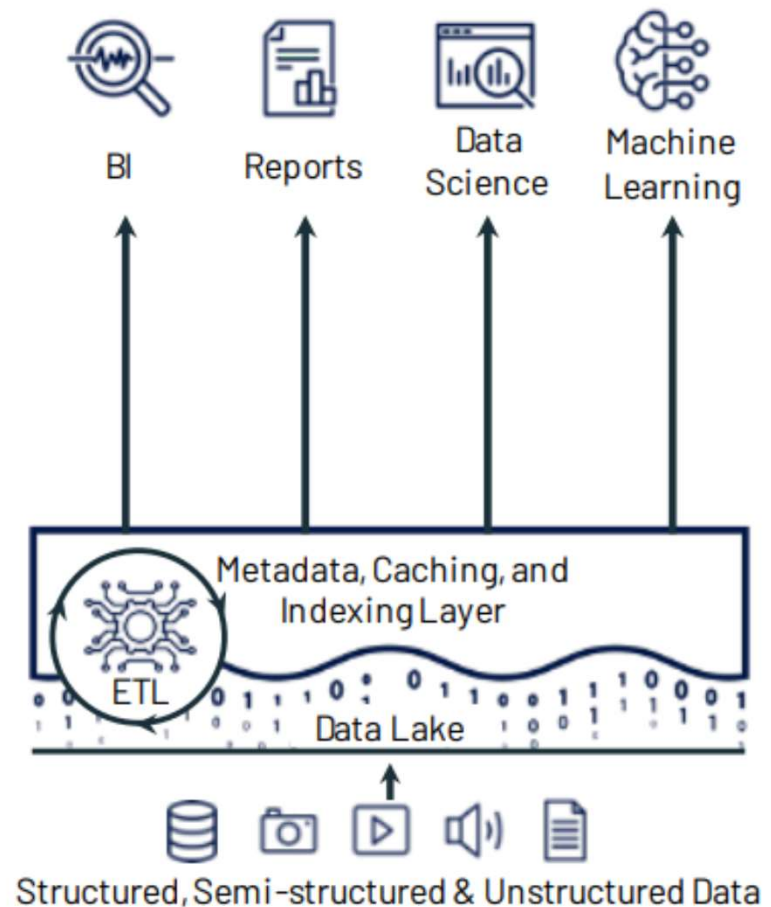
- Data lakes have typically used a “schema on read” strategy, where data can be ingested in any format without schema enforcement. This lack of schema enforcement can result in data quality issues, allowing the pristine data lake to become a “data swamp”.
 - Data files can only be appended to, leading to expensive re-writes of previously written data to make a simple update.
- **Lakehouse** combines the strengths, and addresses the weaknesses of both ‘data warehouse’ and ‘data lake’ technologies.

Data Lakehouse

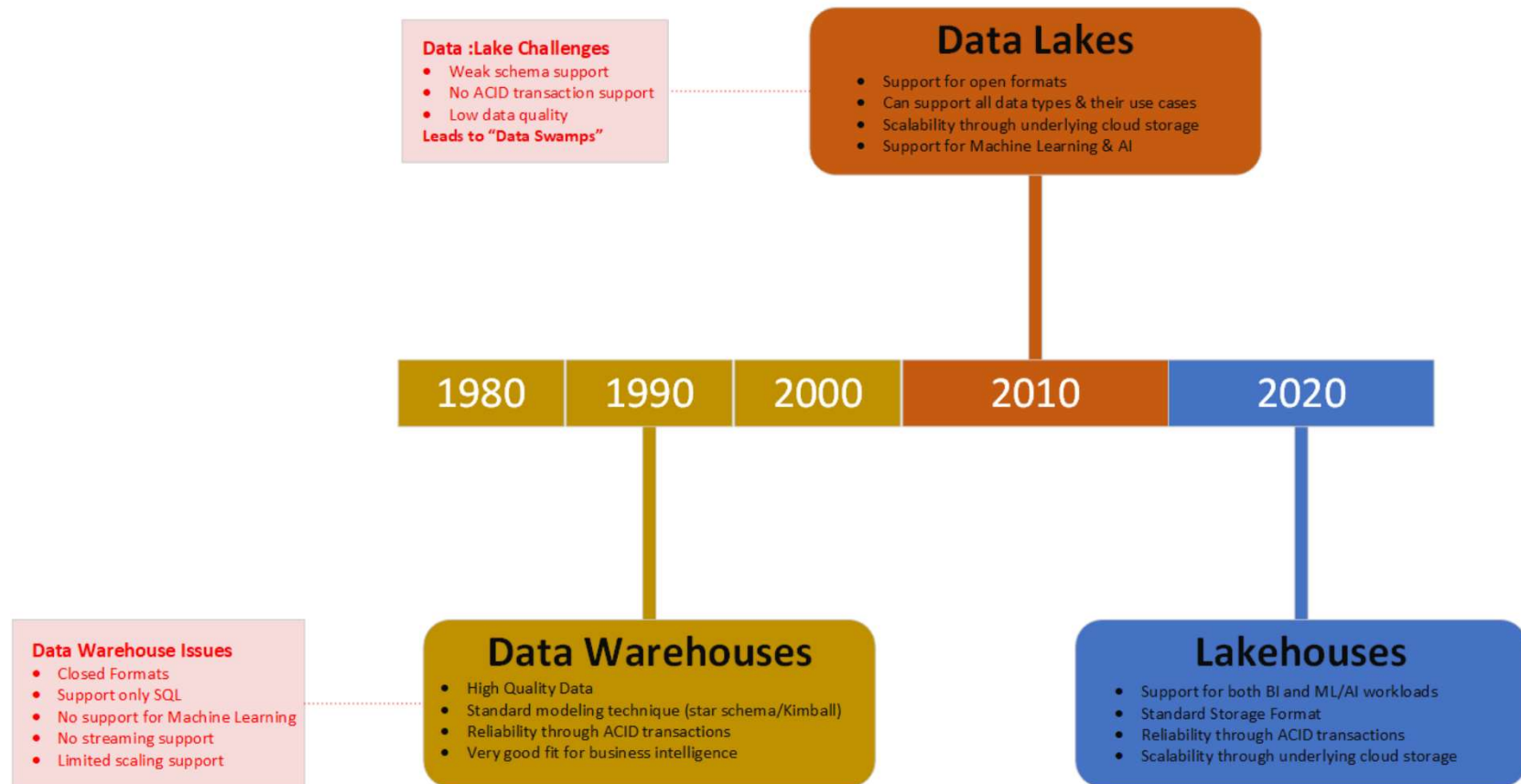
- **Lakehouse** as a system which merges the flexibility, low cost, and scale of a data lake with the data management and ACID transactions of data warehouses, addressing the limitations of both.
- Like *data lakes*, the Lakehouse architecture leverages the low-cost cloud storage systems, with the inherent flexibility and horizontal scalability that comes with those systems.
- It also continues to leverage the data formats like Parquet and AVRO, which enable the integration with ML and AI systems.
- Like *data warehouses*, a data Lakehouse will enable ACID transactions on data assets stored on these low-cost storage systems, enabling the reliability that used to be the exclusive domain of relational database systems.

Data Lakehouse

- With the Lakehouse architecture, we no longer need to have a copy of our data in the data lake, and another copy in some type of data warehouse storage.
- We can serve up our data directly from the Data Lake with comparable performance to a classical data warehouse.



Evolution



Delta Lake

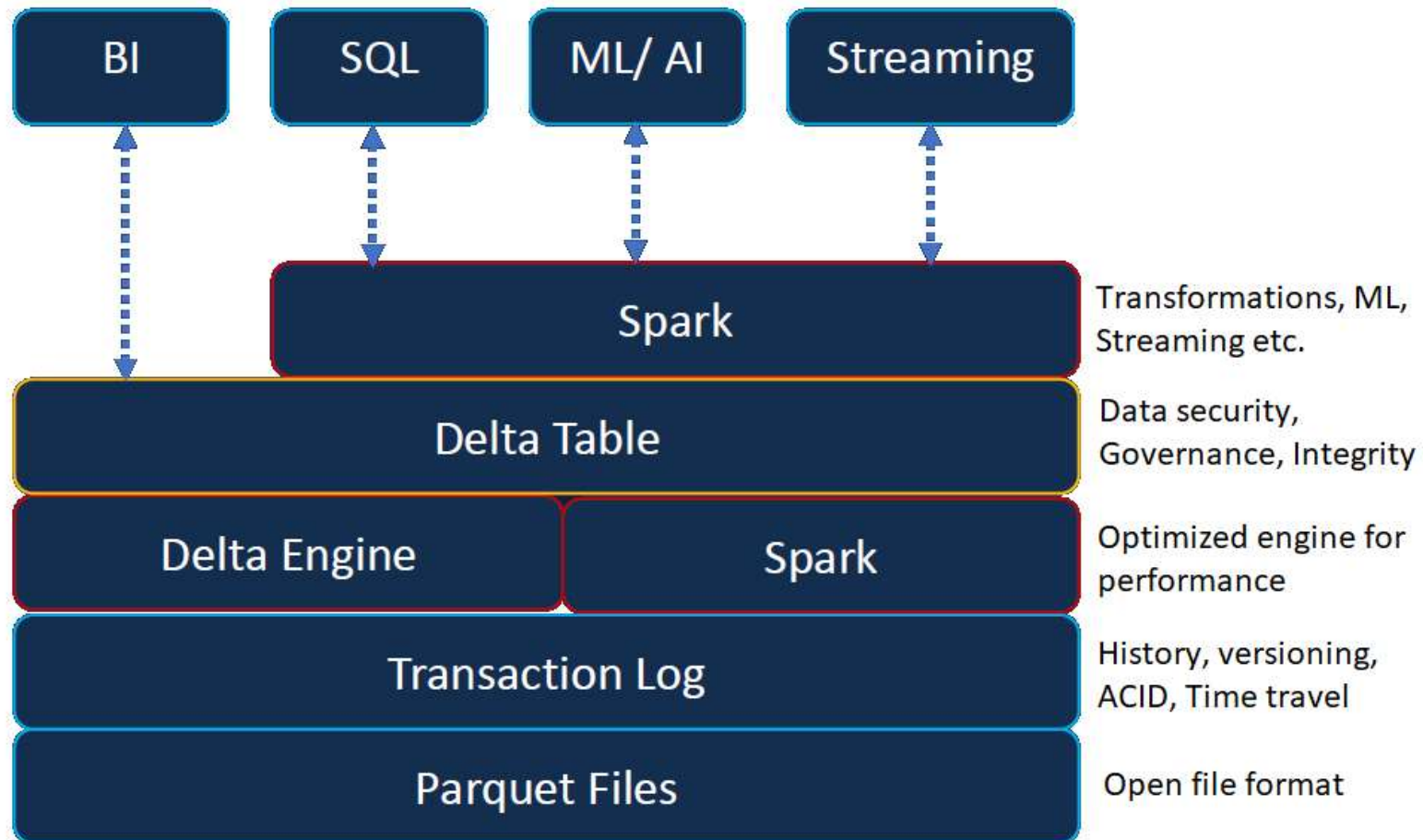
- Delta Lake (*delta.io*) is an open-source storage framework that enables building a Lakehouse architecture with compute engines including Spark, PrestoDB, Flink, Trino, and Hive and APIs for Scala, Java, Rust, and Python.
- Delta Lake is a file-based metadata, caching and indexing layer on top of a data lake storage that provides an abstraction to serve ACID transactions (CRUD operations – INSERT, UPDATE, DELETE and MERGE/UPSERT).
- Delta Lake provides ACID transactions, scalable meta data handling, a unified process model that spans batch and streaming, full audit history, and support for SQL DML statements.
- It can run on existing data lakes and is fully compatible with several processing engines, including Apache Spark.

Delta Lake Features

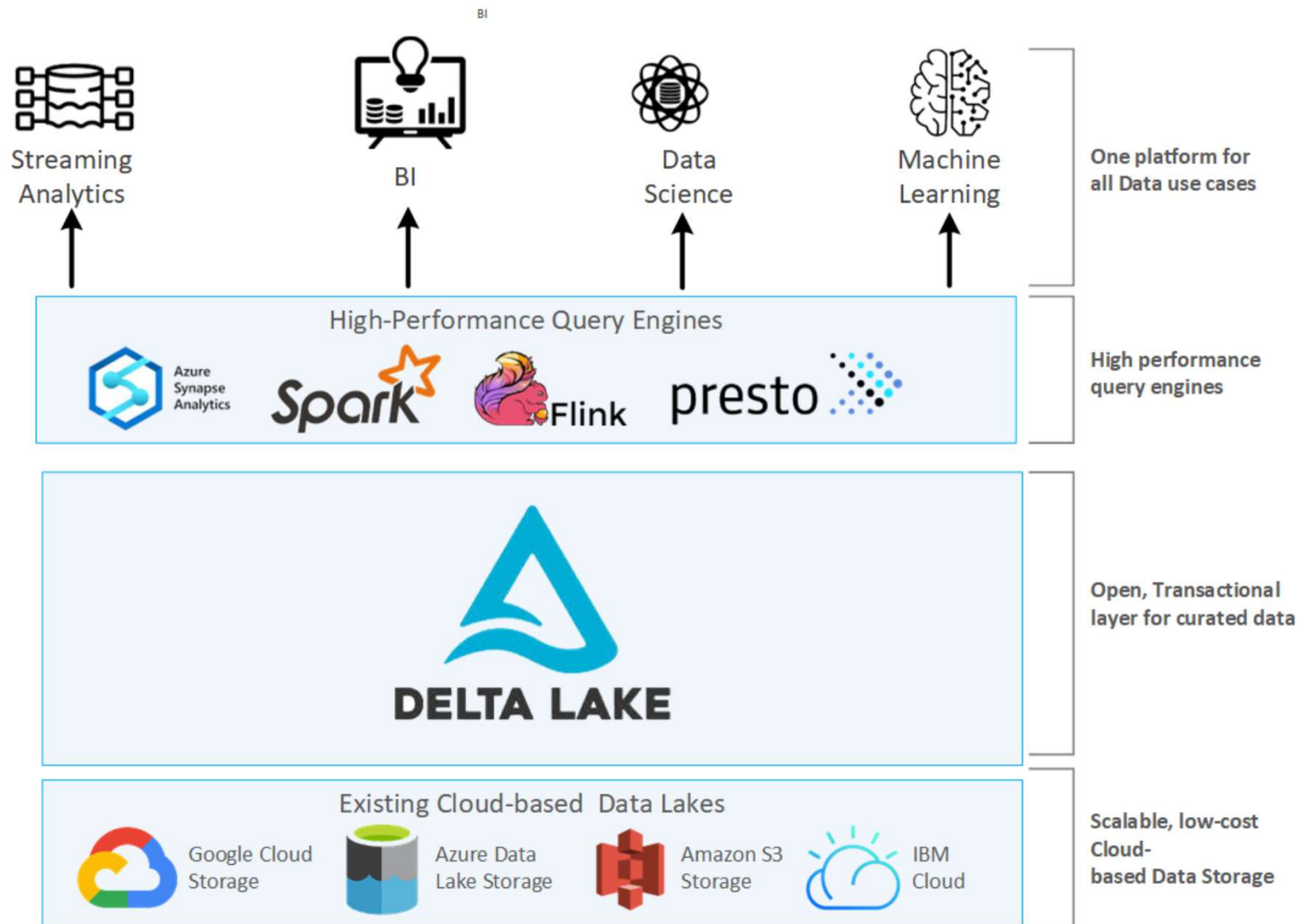
Features of Delta lake are:

- Transactional ACID guarantees
- Full DML support
- Audit History
- Unification of batch and streaming into one processing model
- Schema enforcement and evolution
- Rich metadata support and scaling

How Delta Lake works?



Delta Lake Architecture



Databricks Delta Tables

- A Delta table is a managed table that uses the Databricks Delta Optimistic Concurrency Control feature to provide full ACID transactions.
- Databricks Delta tables support all common operations, such as:
 - CRUD (create, read, update, delete)
 - Upsert (update or insert)
 - Merge (upsert a set of records)
 - Delete from the table where ... (delete based on predicate)
- In addition, Databricks Delta tables support the following:
 - Time travel – rollback data as of any point in time (based on timestamp or version)
 - Uncommitted reads - see changes that have not yet been committed
 - Optimistic concurrency control - prevents dirty reads and writes



ELT using Spark SQL

Querying Files

- In addition to using DataFrameReader API, Spark also supports querying data files directly.
 - Self-describing formats like Parquet & JSON can be read using schema inference.
 - Non self-describing formats like CSV & Text are read as plain text
 - We can also read data files using binary format.

SELECT * FROM file_format.`/path/to/file`

SELECT * FROM **json**.`/path/file_name.json`

SELECT * FROM **parquet**.`/path/file_name.parquet`

Querying Files as Raw Data

- Extract text files as raw strings
 - Text-based files (JSON, CSV, TSV, and TXT formats)
 - **SELECT * FROM text. `/path/to/file`**
- Extract files as raw bytes
 - Images or unstructured data
 - **SELECT * FROM binaryFile. `/path/to/file`**

Creating tables from files using CTAS

- We can directly create tables in DELTA format from self-describing file formats like Parquet and JSON using CTAS syntax.
 - The table schema is automatically inferred based on query results
 - We can not declare schema manually.
 - Useful for external data ingestion with well-defined schema
 - Do Not support file options

```
CREATE TABLE table_name  
AS  
SELECT * FROM file_format.`/path/to/file`
```

External tables with custom schema

- We can create external non-delta tables with manual schema declaration and file options

```
CREATE TABLE users (id INT, name STRING, age INT)
USING CSV
OPTIONS (header = "true", delimiter = "\t")
LOCATION = '/FileStore/data/users.csv'
```

```
CREATE TABLE user (id INT, name STRING, age INT)
USING JDBC
OPTIONS (url = .., dbtable=.., user=.., password=..)
```

- The table is not in DELTA format, hence we can not get the benefits and performance of delta format

Overwrite and append tables



5

```
1  -- Create a new table or overwrite an existing table
2  CREATE OR REPLACE TABLE orders AS
3  SELECT * FROM parquet.`<data-path>`
```



6

```
1  -- Overwrite the data of an existing table
2  INSERT OVERWRITE orders
3  SELECT * FROM parquet.`<data-path>`
```



7

```
1  -- append data to an existing table
2  INSERT INTO orders
3  SELECT * FROM parquet.`${dataset.bookstore}/orders-new`
```

Merging tables

- MERGE command allows us to merge two tables where in we can perform insert, update and delete operations on the source table from target tables data based on join columns and other conditions.

```
CREATE OR REPLACE TEMP VIEW customers_updates AS
SELECT * FROM json.`${dataset.bookstore}/customers-json-new`;

MERGE INTO customers c
USING customers_updates u
ON c.customer_id = u.customer_id
WHEN MATCHED AND c.email IS NULL AND u.email IS NOT NULL THEN
| UPDATE SET email = u.email, updated = u.updated
WHEN NOT MATCHED THEN INSERT *
```

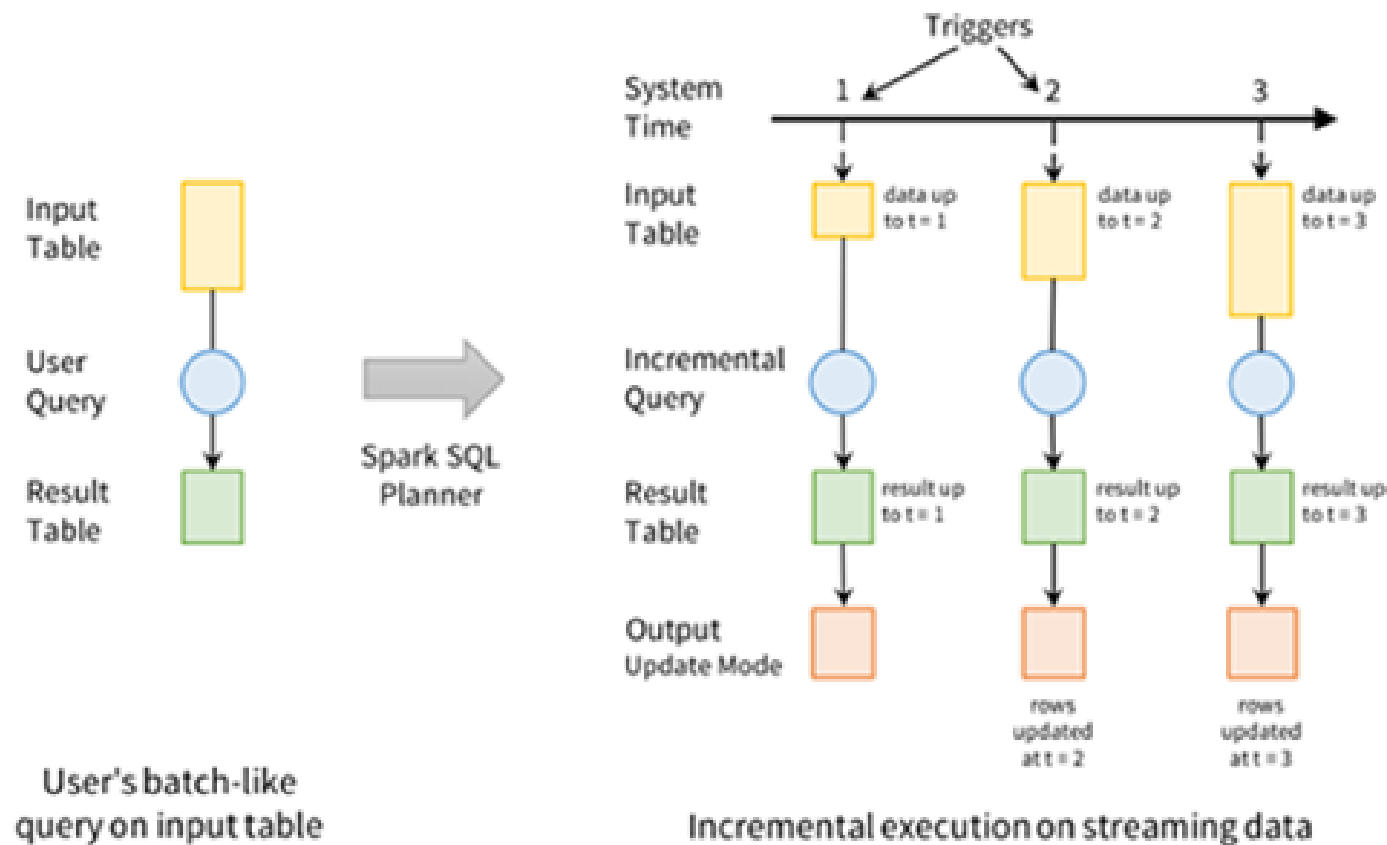


Structured Streaming & Auto Loader

Spark Structured Streaming

- Structured Streaming is a high-level API for stream processing in Spark (version 2.2 onwards) . It is a near-real time processing engine that offers end-to-end fault tolerance with exactly-once processing guarantees using familiar Spark APIs.
- Structured Streaming lets you express computation on streaming data in the same way you express a batch computation on static data. The Structured Streaming engine performs the computation incrementally and continuously updates the result as streaming data arrives.
- The best thing about Structured Streaming is that it allows you to rapidly and quickly get value out of streaming systems with virtually no code changes. It also makes it easy to reason about because you can write your batch job as a way to prototype it and then you can convert it to a streaming job.

Spark Structured Streaming

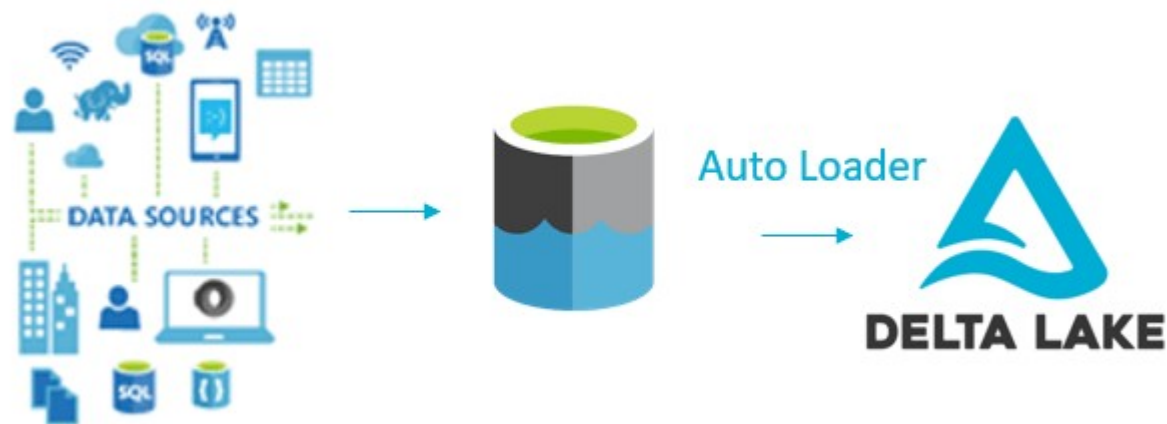


Structured Streaming Processing Model

Users express queries using a batch API; Spark incrementalizes them to run on streams

Databricks Auto Loader

- **Autoloader** is an Optimized File Source that automatically performs incremental data loads from your Cloud storage as it arrives into the Delta Lake Tables.
- Autoloader presents a **new Structured Streaming Source** called **cloudFiles**.
- With the DBFS paths or direct paths to the data source as the input, it automatically sets up file notifications that subscribe to file events to handle new files on arrival with an option to process the existing ones in the directory.



Databricks Auto Loader



Auto Loader file detection modes

Databricks Autoloader supports two methods to detect new files in your Cloud storage namely:

- **Directory Listing:**

- This approach is useful for cases when only a few files are required to be streamed regularly. Here, the new files are recognized from listing the input directory.
- With just access to your Cloud Storage data, you can swiftly enable your Databricks Autoloader Streams. From the beginning, Databricks Autoloader automatically detects if the input directory is good for Incremental Listing.
- You have the option to explicitly choose between the Incremental Listing or Full Directory Listing by setting `cloudFiles.useIncrementalListing` as true or false.

- **File Notification**

- Using the Cloud services like Azure Event Grid and Queue Storage services, AWS SNS and SQS or GCS Notifications, and Google Cloud Pub/Sub services, it subscribes to file events in the input directory.

Key Features of Databricks Autoloader

Using Autoloader can simplify your Data Ingestion process providing the following benefits to you:

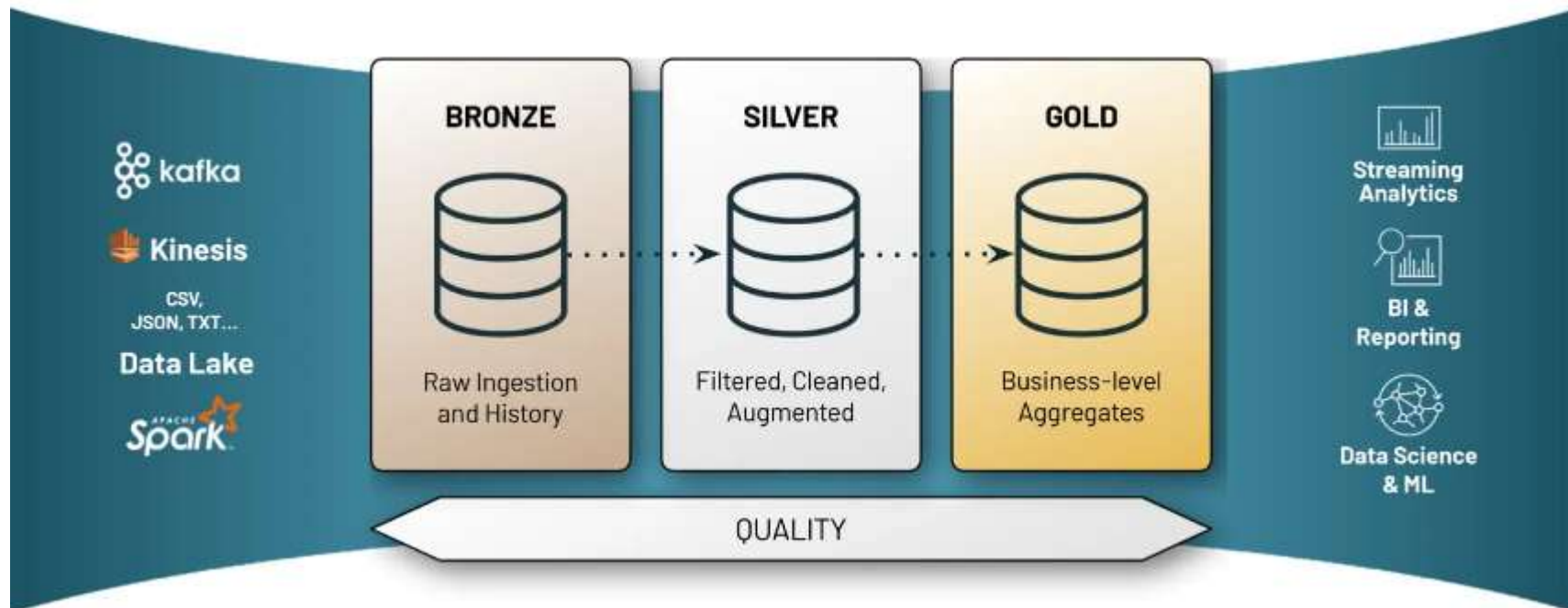
- **Scalability:** Databricks Autoloader can track billions of files by leveraging the Cloud Services and RockDB without the need to list all the files in your directory.
- **Cost-Effective:** The notification mode for file detection eliminates the need for a directory list, thereby reducing costs. The cost of file discovery also directly depends on the number of files that are ingested rather than the number of directories in which the files arrive.
- **Ease of Use:** The Databricks Autoloader sets up the notification mode and message queue services to perform Incremental Data Load. You also don't require to track files or manage any state information on what files have arrived.
- **Schema Inference and Evolution:** For cases when there are schema drifts such as new columns, Databricks Autoloader will manage it and notify you whenever schema changes.



Delta Live Tables (DLT)

Medallion Architectural Pattern

A medallion (aka multi-hop) architecture is a data design pattern used to logically organize data in a lakehouse, with the goal of incrementally and progressively improving the structure and quality of data as it flows through each layer of the architecture (from Bronze \Rightarrow Silver \Rightarrow Gold layer tables).



Delta Live Tables

- Delta Live Tables is a **declarative framework for building data processing pipelines** that are reliable, maintainable, and testable.
 - You define the transformations to perform on your data and Delta Live Tables manages task orchestration, cluster management, monitoring, data quality, and error handling.
 - Instead of defining your data pipelines using a series of separate Apache Spark tasks, you define streaming tables and materialized views that the system should create and keep up to date.
 - Delta Live Tables manages how your data is transformed based on queries you define for each processing step. You can also enforce data quality with Delta Live Tables expectations, which allow you to define expected data quality and specify how to handle records that fail those expectations.

Why Delta Live Tables ?

- Delta Live Tables (DLT) makes it easy to build and manage reliable batch and streaming data pipelines that deliver high-quality data on the Databricks Lakehouse Platform.
- DLT helps data engineering teams simplify ETL development and management with declarative pipeline development, automatic data testing, and deep visibility for monitoring and recovery.
- Features
 - Easily build and maintain data pipelines
 - Automatic data quality testing
 - Cost-effective streaming through efficient compute auto-scaling
 - Deep visibility for pipeline monitoring and observability

Delta Live Tables datasets

Dataset type	How are records processed through defined queries?
Streaming table <small>STREAMING LIVE TABLE</small>	Each record is processed exactly once. This assumes an append-only source.
Materialized view <small>LIVE TABLE</small>	Records are processed as required to return accurate results for the current data state. Materialized views should be used for data sources with updates, deletions, or aggregations, and for change data capture processing (CDC).
View	Records are processed each time the view is queried. Use views for intermediate transformations and data quality checks that should not be published to public datasets.

DLT datasets – Streaming Table

- A streaming table is a Delta table with extra support for streaming or incremental data processing. They allow you to process a growing dataset, handling each row only once.
 - Because most datasets grow continuously over time, streaming tables are good for most ingestion workloads.
 - Streaming tables are optimal for pipelines that require data freshness and low latency.
 - Streaming tables can also be useful for massive scale transformations, as results can be incrementally calculated as new data arrives, keeping results up to date without needing to fully recompute all source data with each update.
- Streaming tables are designed for data sources that are append-only.

DLT datasets – Materialized View

- A materialized view (or live table) is a view where the results have been pre-computed.
- Materialized views are refreshed according to the update schedule of the pipeline in which they're contained.
- Materialized views can handle any changes in the input. Each time the pipeline updates, query results are recalculated to reflect changes in upstream datasets that might have occurred because of compliance, corrections, aggregations, or general CDC.
- Delta Live Tables implements materialized views as Delta tables, but abstracts away complexities associated with efficient application of updates, allowing users to focus on writing queries.

DLT datasets – View

- All views in Databricks compute results from source datasets as they are queried, leveraging caching optimizations when available.
- Delta Live Tables does not publish views to the catalog, so views can be referenced only within the pipeline in which they are defined.
- Views are useful as intermediate queries that should not be exposed to end users or systems.
- Databricks recommends using views to enforce data quality constraints or transform and enrich datasets that drive multiple downstream queries.

DLT Pipeline

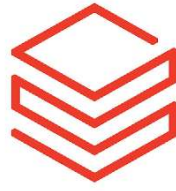
- A **pipeline** is the main unit used to configure and run data processing workflows with Delta Live Tables.
- A pipeline contains materialized views and streaming tables declared in Python or SQL source files.
- Delta Live Tables infers the dependencies between these tables, ensuring updates occur in the correct order. For each dataset, Delta Live Tables compares the current state with the desired state and proceeds to create or update datasets using efficient processing methods.
- The settings of Delta Live Tables pipelines fall into two broad categories:
 - Configurations that define a collection of notebooks or files (known as source code or libraries) that use Delta Live Tables syntax to declare datasets.
 - Configurations that control pipeline infrastructure, how updates are processed, and how tables are saved in the workspace.

Monitor and enforce data quality

- You can use ***expectations*** to specify data quality controls on the contents of a dataset.
 - Unlike a CHECK constraint in a traditional database which prevents adding any records that fail the constraint, expectations provide flexibility when processing data that fails data quality requirements.
 - This flexibility allows you to process and store data that you expect to be messy and data that must meet strict quality requirements.

Some important points to note

- DLT notebooks can not be run interactively. They are intended to be attached to a DLT pipeline. You can attach more than one notebook to a pipeline.
- Only Python and SQL are supported. Each notebook can have only one language – either python or SQL. We can not mix both
- We can not use most magic commands like %run
- Requires a premium Databricks workspace
- Automatically generated two clusters – execution and maintenance clusters.

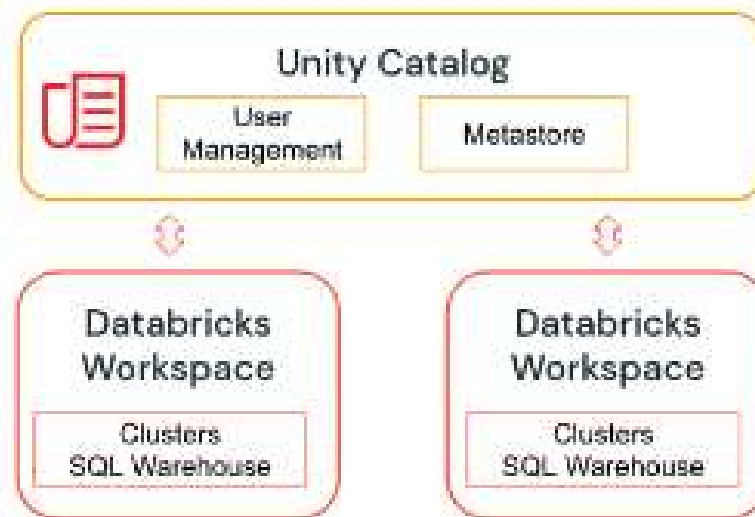


Unity Catalog

Unity Catalog

Unity Catalog is a unified governance solution for data and AI assets on the Databricks Lakehouse.

Unity Catalog provides centralized access control, auditing, lineage, and data discovery capabilities across Databricks workspaces.



- **Unity Catalog is supported on clusters that run Databricks Runtime 11.3 LTS or above.**

Unity Catalog features

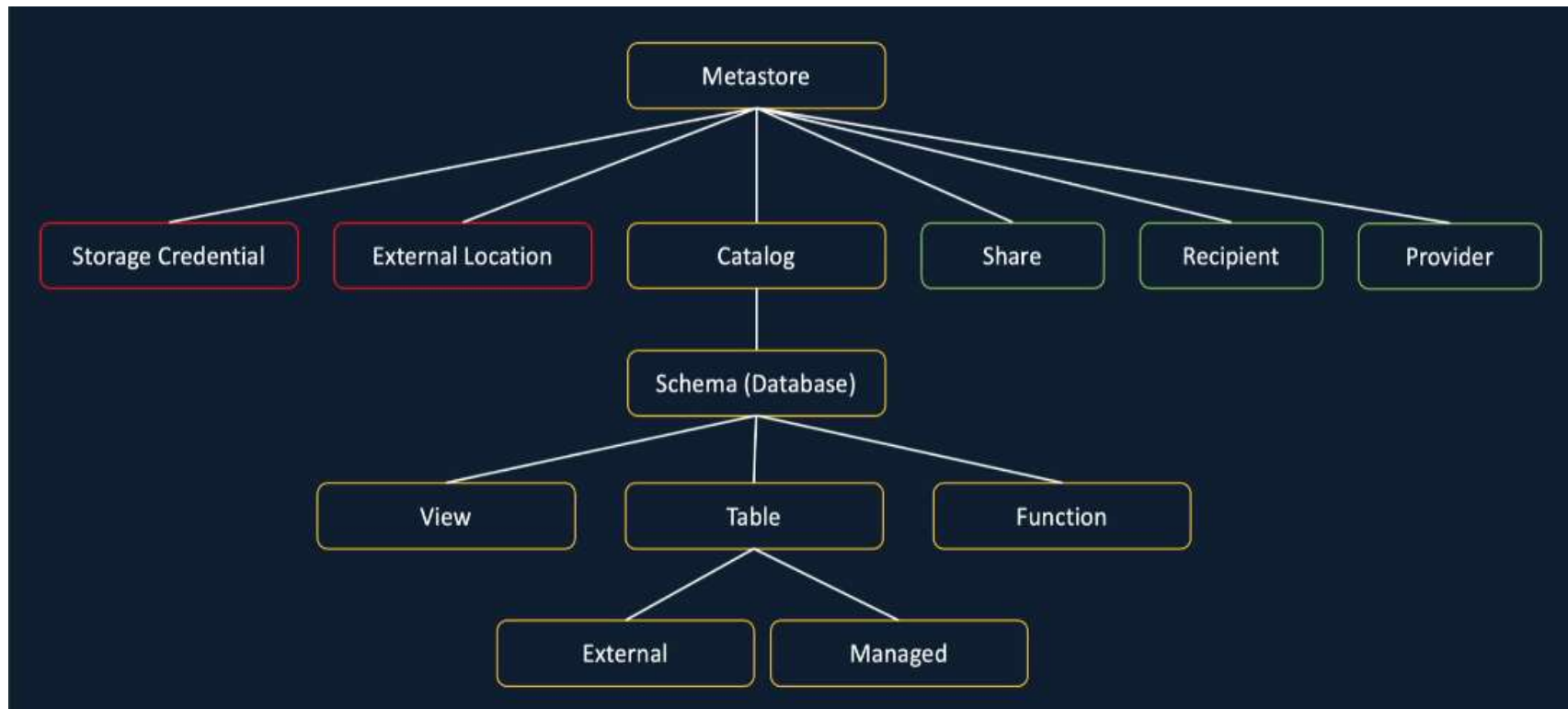
- **Define once, secure everywhere:** Unity Catalog offers a single place to administer data access policies that apply across all workspaces.
- **Standards-compliant security model:** Unity Catalog's security model is based on standard ANSI SQL and allows administrators to grant permissions in their existing data lake using familiar syntax, at the level of catalogs, databases (also called schemas), tables, and views.
- **Built-in auditing and lineage:** Unity Catalog automatically captures user-level audit logs that record access to your data. Unity Catalog also captures lineage data that tracks how data assets are created and used across all languages.
- **Data discovery:** Unity Catalog lets you tag and document data assets, and provides a search interface to help data consumers find data.

Unity Catalog object model

In Unity Catalog, the hierarchy of primary data objects flows from metastore to table or volume:

- **Metastore:** The top-level container for metadata. Each metastore exposes a three-level namespace (*catalog.schema.table*) that organizes your data.
- **Catalog:** The first layer of the object hierarchy, used to organize your data assets.
- **Schema:** Also known as databases, schemas are the second layer of the object hierarchy and contain tables and views.
- **Volume:** Volumes sit alongside tables and views at the lowest level of the object hierarchy and provide governance for non-tabular data.
- **Table:** At the lowest level in the object hierarchy are tables and views.

Unity Catalog object model



Metastore

- A metastore is the top-level container of objects in Unity Catalog. It stores metadata about data assets (tables and views) and the permissions that govern access to them.
- Databricks account admins should create one metastore per region in which they operate and assign them to Databricks workspaces in the same region.
- For a workspace to use Unity Catalog, it must have a Unity Catalog metastore attached.
- Each metastore is configured with a managed storage location in an S3 bucket in your AWS account.

Managed Storage

- When an account admin creates a metastore, they must associate a storage location in an S3 bucket in your AWS account to use as managed storage.
- Unity Catalog also allows users to associate managed storage locations with catalogs and schemas.
- Managed storage has the following properties:
 - Managed tables and managed volumes store data and metadata files in managed storage.
 - Managed storage cannot overlap with external tables, external volumes, or other managed storage.

Catalog

- A catalog is the first layer of Unity Catalog's three-level namespace. It's used to organize your data assets.
- Users can see all catalogs on which they have been assigned the **USE CATALOG** data permission.
- All users have the **USE CATALOG** permission on the **main** catalog. The main catalog is intended for organizations that are just getting started with Unity Catalog. As you add users and data, you should add catalogs to maintain a data hierarchy that enables efficient control over access.

Schema

- A schema (also called a database) is the second layer of Unity Catalog's three-level namespace.
- A schema organizes tables and views. Users can see all schemas on which they have been assigned the **USE SCHEMA** permission, along with the **USE CATALOG** permission on the schema's parent catalog.
- To access or list a table or view in a schema, users must also have **SELECT** permission on the table or view.

Tables

- A table resides in the third layer of Unity Catalog's three-level namespace.
 - To create a table, users must have CREATE and USE SCHEMA permissions on the schema, and they must have the USE CATALOG permission on its parent catalog.
 - To query a table, users must have the SELECT permission on the table, the USE SCHEMA permission on its parent schema, and the USE CATALOG permission on its parent catalog.
- A table can be managed or external.
 - **Managed Table** : Managed tables are the default way to create tables in Unity Catalog. Unity Catalog manages the lifecycle and file layout for these tables. You should not use tools outside of Databricks to manipulate files in these tables directly. Managed tables **always use the Delta table format**.
 - **External Table**: External tables are tables whose data lifecycle and file layout are not managed by Unity Catalog. Use external tables to register large amounts of existing data in Unity Catalog, or if you require direct access to the data using tools outside of Databricks clusters or Databricks SQL warehouses. When you drop an external table, Unity Catalog does not delete the underlying data. External tables can use the following file formats: DELTA, CSV, JSON, AVRO, PARQUET, ORC, TEXT.

How to set up Unity Catalog for my organization?

To set up Unity Catalog for your organization, you do the following:

- Configure an S3 bucket and IAM role that Unity Catalog can use to store and access data in your AWS account.
- Create a metastore for each region in which your organization operates.
- Attach workspaces to the metastore.
 - Each workspace will have the same view of the data you manage in Unity Catalog.
- If you have a new account, add users, groups, and service principals to your Databricks account. Create and grant access to catalogs, schemas, and tables.

Create a cluster or SQL warehouse

Before you can start creating tables and assigning permissions, you need to create a compute resource to run your table-creation and permission-assignment workloads.

Tables defined in Unity Catalog are protected by fine-grained access controls. To ensure that access controls are enforced, Unity Catalog requires compute resources to conform to a secure configuration. Non-conforming compute resources cannot access tables in Unity Catalog.

Databricks provides two kinds of compute resources:

- **Clusters**, which are used for executing commands in Databricks notebooks and running jobs.
- **SQL warehouses**, which are used for executing queries in SQL Editor.

You can use either of these compute resources to work with Unity Catalog.

Thank You