**Databricks - Introduction**

* What is Databricks?
  + It is a Multi cloud Lakehouse platform based on Apache Spark.
* What is Lakehouse?
  + It is a Unified Analytics platform where we can work Data Engineering Analytics and AI workloads in one platform.
  + It brings best features from Data Lake and Data warehouse.
    - Data lake 🡪 It brings Open source and ML Support.
    - Data warehouse 🡪 It brings Reliability, Strong Governance, Performance.
* Lakehouse Architecture (Three Layers)
  + Cloud Services
    - Databricks Available on multi cloud providers such as AWS, Azure, GCP.
* Runtime
  + Databricks uses Cloud provider to Build Cluster. When you are building a cluster, Databricks Runtime will be pre-installed.
  + Databricks Run time sit on top of Spark, Delta Lake and other libraries.
* Workspace
  + On top of Run time, we have a Workspace which contains Data Engineering, Data warehousing and Machine learning.
* How Databricks installed in Cloud provider?
  + There are two panels which accumulate databricks.
    - Data plane 🡪 (Storage and Compute)
    - Control Plane 🡪 (Databricks UI, Cluster Management, Notebooks and Workflows)
* Databricks File System (DBFS)
  + DBFS is a File System which helps to persist your data and files. It act as Structured layer on top of Storage Layer.
  + When you are placing the files in DBFS, Actually, It is going to store in underlying S3/Blop Storage/GCS. So, Suppose, If DBFS is corrupted or terminated, your files will be safe in Cloud Storage.
* Exploring WorkSpace
  + Workspace 🡪 Workspace tab is used to organize all the assets in your workspace.
  + Repos 🡪 For Git Integration
  + Data 🡪 To Manage the databases and tables.
  + Compute 🡪 To create and Manage the Clusters
  + Workflows 🡪 Deploy and Orchestrate the jobs.
* Notebooks
  + Notebook provides interactive environment for developing your code. you can collaborate these notebooks with your team members.
  + Before running this Notebook, attach this notebook to the cluster and run it.
  + Shift + Enter to Execute the commands in Notebook.
* Magic Commands (SQL, Python, Scala, R, run and md)
  + Magic Commands are Built in Commands which provides a output regardless of the Notebook Language chose.
  + Markdown Magic Command (%md)
    - It helps to format your text
* Run Magic Command (%run)
  + Import another notebook to current notebook.

- %run ./Includes/setup

- print(variable\_name)

* fs Magic Command

- helps to perform file system operations.

- %fs ls '/databricks-datasets'

* dbutils (Databricks Utilities)

- Other way to deal with file system operations. Also, you can interact with different services and tools like Credentials, file system, Secrets and widgets.

- dbutils.help()

- dbutils.fs.help()

- dbutils.fs.ls('/databricks-datasets')

* which one would be preferable to list of directories and files.?
  + dbutils will be more useful than fs magic command. you can use dbutils as part of Python code. So you can store output into variable.

- files = dbutils.fs.ls('/databricks-datasets')

- print(files)

- display(files)

* To Export your code, Go to File --> Export --> IPythonNotebook.
* To Export folder and notebooks --> Workspace --> folder --> Export --> DBC Archive (Databricks Cloud)
* To Undo changes, Click on last edit in 2 mints link.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

A screenshot of a computer

Description automatically generated

**Delta Lake:-**

* What is Delta Lake?
  + Delta lake is a Storage framework which provides transactional support to data lake.
  + It is not a data lake or data warehouse. Delta lake enables Building Lakehouse.
* Advantages of Delta Lake
  + provides ACID transactions to Data lake
  + Time Travel
    - * In Delta Lake, Every transaction on delta table is automatically versioned. So, you can able to restore at any specific Timestamp.
      * DESCRIBE HISTORY 🡪 see the history changes
      * Methods to query older version of data,

- select \* from table\_name TIMESTAMP AS OF "2022-01-01"

- select \* from table\_name VERSION AS OF 36

* to restore,

- RESTORE TABLE table\_name TO TIMESTAMP AS OF "2022-01-01"

- RESTORE TABLE table\_name TO VERSION AS OF 36

* We can compact small files,

- OPTIMIZE table\_name

* Cleaning up Un-used data file using Vacum

- VACUUM table\_name [retention period] --> Default 7 days

- Note that, Vacuum = No time travel

* Schema Enforcement and Evolution
  + Delta Lake supports schema enforcement to ensure that data written to the lake with a predefined schema.
  + It automatically handles schema evolution by reconciling schema changes with existing data.
* How is Delta Lake installed in databricks?
* When you are creating Databricks cluster, Delta Lake will be deployed as part of databricks Runtime.
* What is transactional Log (Delta log)?
* When you are creating Delta Lake tables, data will stored in a Storage layer as one or more data files in Parquet format. Along with that, there is a Transactional Log file also be stored.
* It is a Single Source of Truth. Whenever we are doing transaction on Delta table, new transactional Log (.json) will be created. It captured what transaction has been done on the delta table and What are the files get affected.
* Write/Read/Update/Delete operation in Delta Lake? (Insert/Update/Delete)
  + During Insert operation, Writer process will write data into Parquet file (1.parquet). Along with it will create Transactional Log file (000.json) into delta\_log folder.
  + During Read operation, Reader process will take Latest transactional log file from delta log folder, based on that It will read data files accordingly.
  + During Update/Delete operation, Writer process will copy parquet file (1.parquet) and make changes related to update operation and create a new parquet file (2.parquet). Also, it creates another transactional log file (001.json)
  + During Read operation, Reader process will take Latest transactional log file from delta log folder, based on that It will read data files accordingly.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* How to Retrieve Data from Databricks?
  + Query data directly from Databricks.

- select \* from json.`path/filename.json`

* CTAS: Registering tables from Files

- CREATE TABLE table\_name as select \* from file\_format.`path/to/file`

- CTAS statements automatically infer Schema information from query results.

- Does not support Manual Schema Declaration

- Useful for external data Ingestion with Well-defined Schema Such as Parquet files and Schema

- CTAS Statements Does not support file options.

* Create Delta Table with Location pointed

- CREATE TABLE table\_name (col\_name1, col\_type1,..)

USING CSV

OPTIONS(header = "true", delimiter = ";")

LOCATION = path

* Create External table (Not a Delta Table)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Structured Streaming**

* What is Data Stream.?
  + DataStream can be any data source that grows over time. For Example,
    - Directory of data
    - Table Data
    - Messaging Queue
* Methods to process Data Stream data.?
  + Full Load: Traditional Approach (Full Load) - Reprocess the entire source data each time.
  + Delta Approach: Write Custom logic to process only those new data added since last update.
    - Here, we can use Structured Streaming to achieve this goal.
* Structured Streaming
  + Structured Streaming is a Stream processing engine which is built on top of Spark-SQL.
  + Structured Streaming treats a stream of data as an infinite table called a DataFrame or Dataset which allow us to perform SQL queries, DataFrame operations, and machine learning algorithm.
  + Concepts
    - Watermarking
      * Watermarking is a technique which defines the threshold for late data, specifying how long system should wait for late events to arrive before emitting the output.
* Triggers
  + Triggers define the policy for executing outputs.
  + Trigger intervals.

|  |  |  |
| --- | --- | --- |
| Trigger (Mode) | Method Call | Behaviour |
| Unspecified |  | Default: Processing Time (500ms) |
| Fixed Interval | .trigger(  processingTime="5 minutes") | Process data in micro batches at the User Specified intervals. |
| Triggered batch | .trigger(once=True) | Process all available data in a Single batch, then stop. |
| Triggered Micro batches | .trigger(availableNow=True) | Process all available data in multiple micro batches, then stop. |

* Output Modes:
  + Complete mode (all new and updated results are written to the sink)
  + Append mode (only new results are written to the sink)
  + Update mode (only updated results are written to the sink).
* Unbounded table:-
  + Structured Streaming treats a stream of data as an infinite table called a DataFrame or Dataset which allow us to perform SQL queries, DataFrame operations, and machine learning algorithm.
  + How to read data from Data Streams:- (Read Stream helps to read Delta table)
    - Using Spark readStream method to Read delta table, which allows to process the whole data present in the delta table. As well new data arrived later.

- streamDF = spark.readStream.table("Input\_table")

* How to write data into Sink
  + streamDF.writeStream.trigger(processingTime="2 minutes").outputMode("append").option("checkPointLocation", "/path").table("ouput\_table")
* Checkpointing
  + Databricks creating Checkpointing location for storing the current state of your streaming jobs to cloud storage. It can be used by streaming engine to check/Track the progress of your streaming progressing.
  + Separate Checkpoint location is required for every Streaming writes to ensure processing Guarantees.
  + Checkpointing Can not be shared between other Streams.
  + Structured Streaming provides two Guarantees.
    - Fault Tolerance using Checkpointing.
    - Exactly Once Guarantee
* Unsupported Operations:-
  + Some operations are not supported by Streaming Dataframe,

- Sorting ()

- De-duplication

* + These unsupported operations can be achieved by Some Advanced methods,

- Windowing

- Watermarking

**Incremental Data processing**

* Methods to process Incremental data.
  + COPY INTO
  + Auto Loader
* Copy Into
  + Copy into Command used to load file from file location to a Delta table. Each time, if you are running the same command, it will load only new files from the source location. Skip old files.
  + Suitable for thousands of files.
  + Less Efficient when Data grows.

- COPY INTO table\_name

FROM 'path/to/files'

FILEFORMAT = <format>

FORMAT\_OPTIONS (<format\_options>)

COPY\_OPTIONS(<copy\_options>)

- Example:-

COPY INTO table\_name

FROM 'path/to/file'

FILEFORMAT = csv

FORMAT\_OPTIONS ('delimiter' = '|', 'header' = 'true')

COPY\_OPTIONS ('mergeSchema' = 'true')

* Auto Loader
  + It uses Structured Streaming in Spark which can process the new data files as they arrived in storage location.
  + Can process billions of files.
  + More Efficient when data grows.
  + Autoloader ensures the data file is processed Exactly once. - Exactly-once guarantees

- Auto Loader + Schema

- spark.readStream.format("cloudFiles")

.option("cloudFiles.format", <source\_format>)

.option("cloudFiles.schemaLocation", <schema\_directory>)

.load('path/to/files')

.writeStream.option("checkPointLocation", <checkpoint\_directory>)

.option("mergeSchema", "true")

. table(<table\_name>)

**Multi-hop Architecture in Lake house (Medallion Architecture)**

* It is a Design pattern for storing data into lake house with multi layered approach. which improving the Structure and Quality of data as it flows from Each layer.
* It consists of Three layers

- Bronze

- Contains raw data ingested from Various data Sources.

- Examples - JSON files, Operational data sources, Kafka streams.

- Silver

- Provides more refined view of our data.

- For Example - Data can be cleansed, filtered at this level. Also, Joins fields from various bronze table to reach our desired records.

- Gold

- It provides Business Level Aggregation used to Reporting, Dashboard and Meachine learning.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

How to do CDC in Databricks?

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Delta Live Tables (DLT)**

What is Delta live Tables?

* Delta table helps to store data in tables, whereas Delta Live Tables helps to describe how data flows between these tables declaratively.
* Delta Live Tables is a data pipeline (declarative) framework that manages many delta tables, by creating them and keeping them up to date.
* In short, Delta tables is a data format while Delta Live Tables is a data pipeline framework.
* Delta live tables only suitable for Streaming projects. Not for batch projects.

Benefits

* Dashboards are automatically refreshed when you are using Delta live tables in a pipeline.
* DLT helps to do automatic Orchestration, Monitoring, Error handling. We can see the data flows from Silver to Gold with Live.
* No code is required for Read and write into different layers.

Why/When we go for Delta Live tables?

* Use case I
  + If you have a complex pipeline and you want to visually see the data moving from your source tables in bronze layer to the final aggregated tables in your gold layer, with number of rows written per table.
  + Today, Unity Catalog helps to do trackability (Data Lineage), but for projects that don't have UC enabled yet, DLTs offer to do Data Lineage to capture dependencies between your tables and views.
* Use Case II:-
  + Normally, when you are using DLT, CDC will be very fast. DLT is very useful when you have to handle many merges, updates and deletes (CRUD operations) in your data flow from source to final tables.
  + When using DLT, they are auto compacted and are auto-optimized by default using the Delta format. Auto compaction simply means compacting small files within Delta table partitions to automatically reduce small file problems.
  + When developing DLT with Python, the @dlt.table decorator is used to create a Delta Live Table. But in SQL, just have to add the 'LIVE' word or 'STREAMING LIVE' words and basta -> CREATE LIVE TABLE X as SELECT \* FROM db.Table, and it works for your materialised views.

Coding Example: - Ingest the raw data into a table (Bronze layer)

A screenshot of a computer

Description automatically generated

Clean and prepare data(Silver Layer)

A screen shot of a computer code

Description automatically generated

Top baby names 2021 (Gold Layer)

A screenshot of a computer

Description automatically generated

Limitation of Delta live tables

* Databricks workspace is limited to 100 concurrent pipeline updates.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Unity Catalog**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* Unity Catalog provides centralized data governance for access control, auditing, lineage, and Cataloging capabilities across all the workspaces.

A screenshot of a login page

Description automatically generated

How Unity Catalog govern access to data and Assets in cloud storage?

* Storage credentials 🡪 you can store credentials in Unity catalog.
* Store external locations in Unit catalog.
* Catalog the data stored in S3 in Unity catalog.

Unity Catalog object model (hierarchy)

Metastore 🡪 top-level container for metadata. Each metastore exposes a three-level namespace (catalog.schema.table) that organizes your data.

Catalog 🡪 First layer, used to organize your data assets.

Schema 🡪 Second layer, Contains tables and Views.

Tables, Views, and Volumes 🡪 Lower level. Volumes provide governance for non-tabular data.A diagram of a company

Description automatically generated

A screenshot of a computer

Description automatically generated

Types of Tables

* Managed tables
  + Managed tables life cycle is managed by Databricks. Managed tables always use the [Delta](https://docs.databricks.com/en/delta/index.html) table format.
  + managed tables are stored in the root storage location that you configure when you create a metastore.
  + When a managed table is dropped, its underlying data is deleted from your cloud tenant within 30 days.
* External tables
  + External table data lifecycle and file layout are not managed by Unity Catalog
  + When you drop an external table, Unity Catalog does not delete the underlying data.
* Views and Dynamic Views
  + We can create [dynamic views](https://docs.databricks.com/en/data-governance/unity-catalog/index.html) to enable row- and column-level permissions.
  + Also, we do data masking.

Storage credentials and external locations

* To manage access to the underlying cloud storage for external tables and Storage, Unity Catalog provides these object types:
  + Storage credentials
  + External locations contain a reference to a storage credential and a cloud storage path.