PROFINIT

Databricks advanced

Martin Oharek

December 13th, 2023

Outline

Outline

PROFINIT

1. Data ingestion
2. Data processing & ETL
3. Data governance
4. ML development

Data ingestion

Data ingestion

PROFINIT

- Ingestion backed by Delta Lake

- Different ingestion scenarios:
  - Load data from (external) object storage
    - Data resides in S3 bucket, Azure Blob Storage,…
    - Both batch and streaming use cases
  - Load data from external data sources
    - JDBC connection to relational database systems (PostgreSQL, MySQL, SQL Server,…)
    - Other Databricks workspace
    - Use different Spark connectors to connect to bunch of different services
      - Cassandra, MongoDb, ElasticSearch, Google BigQuery, Snowflake,…
    - Streaming sources
      - Kafka, Kinesis,…

Data ingestion

PROFINIT

- Built-in support for different data formats in Spark
  - Delta Lake
  - Delta sharing
  - Avro
  - ORC
  - JSON
  - XML
  - TEXT
  - Binary
  - …

COPY INTO

PROFINIT

- Load data from file location into Delta Table
- Configurable file or directory filters from cloud storage, including S3, ADLS Gen2, ABFS, GCS, and Unity Catalog volumes
- Exactly-once (idempotent) file processing by default
- Target table schema inference, mapping, merging, and evolution

```
COPY INTO my_table
FROM '/path/to/files'
FILEFORMAT = <format>
FORMAT_OPTIONS ('mergeSchema' = 'true')
COPY_OPTIONS ('mergeSchema' = 'true');
```

COPY INTO

PROFINIT

- Set schema and load data into a Delta Lake table

```
CREATE TABLE default.loan_risks_upload (
  loan_id BIGINT,
  funded_amnt INT,
  paid_amnt DOUBLE,
  addr_state STRING
);

COPY INTO default.loan_risks_upload
FROM '/databricks-datasets/learning-spark-v2/loans/loan-risks.snappy.parquet'
FILEFORMAT = PARQUET;
```

Auto Loader

PROFINIT

- incrementally and efficiently processes new data files as they arrive in cloud storage
- S3, ADLS Gen2, GCS, ABFS, ADLS Gen1, DBFS
- Provides Structured Streaming source called **cloudFiles**
- Supports both one time (batch) upload of existing files and streaming uploads
- Contains all features from COPY INTO and even more

Auto Loader

PROFINIT

- Metadata is persisted in checkpoint location
- It can resume after error based on the information from checkpoint location
- Exactly-once guarantee
- Incremental ingestion support (start where it left off)
- Must provide schemaLocation to store inferred schema (or specify schema directly)

Auto Loader

PROFINIT

- Advantages of Auto Loader over regular Spark Structured Streaming directly on files?
  - **Scalability**
    - can discover billions of files efficiently. Backfills can be performed asynchronously to avoid wasting any compute resources
  - **Performance**
    - cost of discovering files with Auto Loader scales with the number of files that are being ingested instead of the number of directories that the files may land in
  - **Schema inference and evolution support**
  - **Cost**
    - uses native cloud APIs to get lists of files that exist in storage
    - file notification mode can help reduce your cloud costs further

Auto Loader

PROFINIT

- Any new columns will fail the stream and evolve the schema
- Parsing errors will go to _rescued_data

```
spark.readStream.format("cloudFiles") \
  .option("cloudFiles.format", "json") \
  .option("cloudFiles.schemaLocation", "<path-to-schema-location>") \
  .load("<path-to-source-data>") \
  .writeStream \
  .option("mergeSchema", "true") \
  .option("checkpointLocation", "<path-to-checkpoint>") \
  .start("<path_to_target")
```

Auto Loader

PROFINIT

- Well-known schema handling

```
spark.readStream.format("cloudFiles") \
  .schema(expected_schema) \
  .option("cloudFiles.format", "json") \
  # will collect all new fields as well as data type mismatches in _rescued_data
  .option("cloudFiles.schemaEvolutionMode", "rescue") \
  .load("<path-to-source-data>") \
  .writeStream \
  .option("checkpointLocation", "<path-to-checkpoint>") \
  .start("<path_to_target")
```

COPY INTO vs Auto Loader

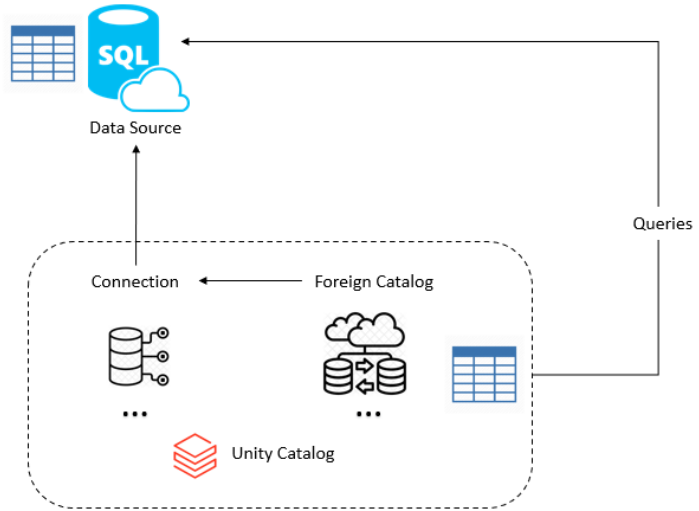
PROFINIT

- Files in order of
  - thousands -> COPY INTO
  - millions and more -> Auto Loader
- Frequently evolving schema -> Auto Loader
- Loading a subset of re-uploaded files -> easier with COPY INTO

External data sources

PROFINIT

- Lakehouse federation
  - Access data in external systems with external compute
  - No data ingestion to Databricks
  - Still can leverage Unity Catalog lineage features



External data sources

PROFINIT

- JDBC connectors
- Might need to install JDBC driver on the cluster

```
driver = "org.postgresql.Driver"

database_host = "<database-host-url>"
database_port = "5432" # update if you use a non-default port
database_name = "<database-name>"
table = "<table-name>"
user = "<username>"
password = "<password>"

url = f"jdbc:postgresql://{database_host}:{database_port}/{database_name}"

remote_table = (spark.read
  .format("jdbc")
  .option("driver", driver)
  .option("url", url)
  .option("dbtable", table)
  .option("user", user)
  .option("password", password)
  .load()
)
```

External data sources

PROFINIT

- Other Spark connectors (example Snowflake)
- Also might require installation of different libraries

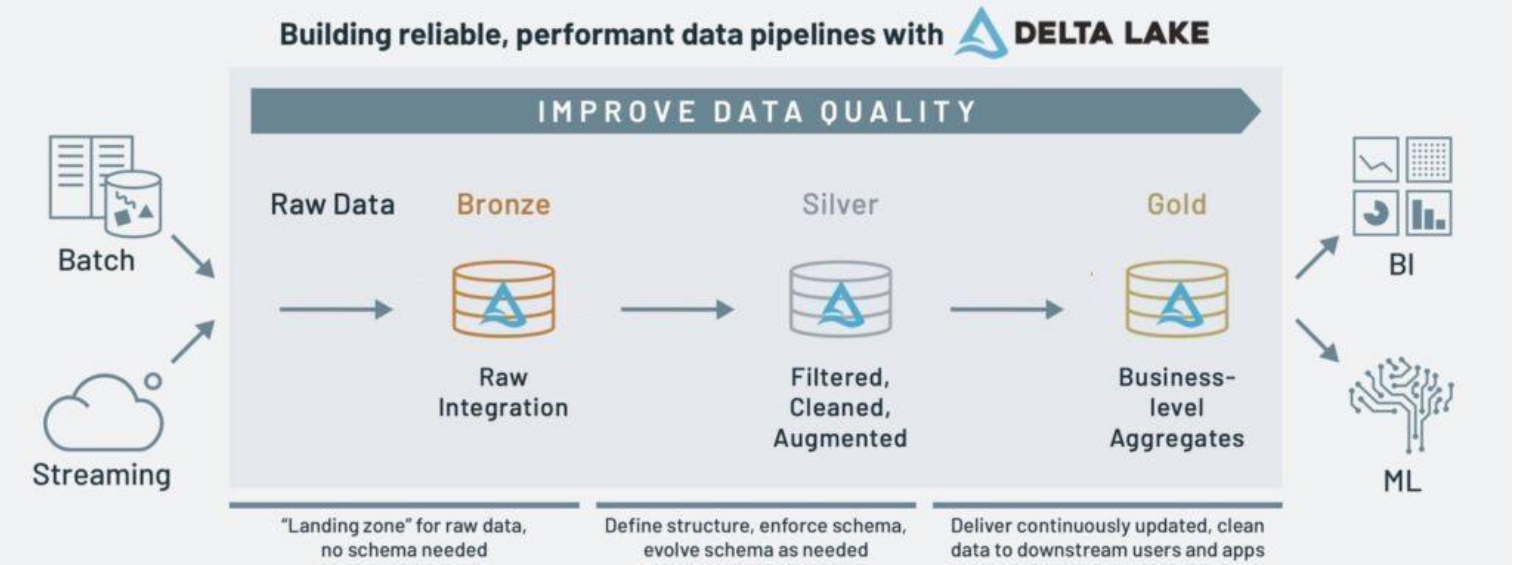
```
snowflake_table = (spark.read
  .format("snowflake")
  .option("host", "hostname")
  .option("port", "port") # Optional - will use default port 443 if not specified.
  .option("user", "username")
  .option("password", "password")
  .option("sfWarehouse", "warehouse_name")
  .option("database", "database_name")
  .option("schema", "schema_name") # Optional - will use default schema "public" if not specified.
  .option("dbtable", "table_name")
  .load()
)
```

Data processing & ETL

Data processing & ETL

PROFINIT

- Medallion architecture



Data processing & ETL

PROFINIT

- Can leverage „regular“ solution
  - Implement ETL logic in plain scripts, notebooks,..
  - Use Databricks workflows and job clusters to create tasks and job DAGs
  - Orchestrate jobs to populate data entities defined in your logic

- Or use managed „Delta Live Tables“

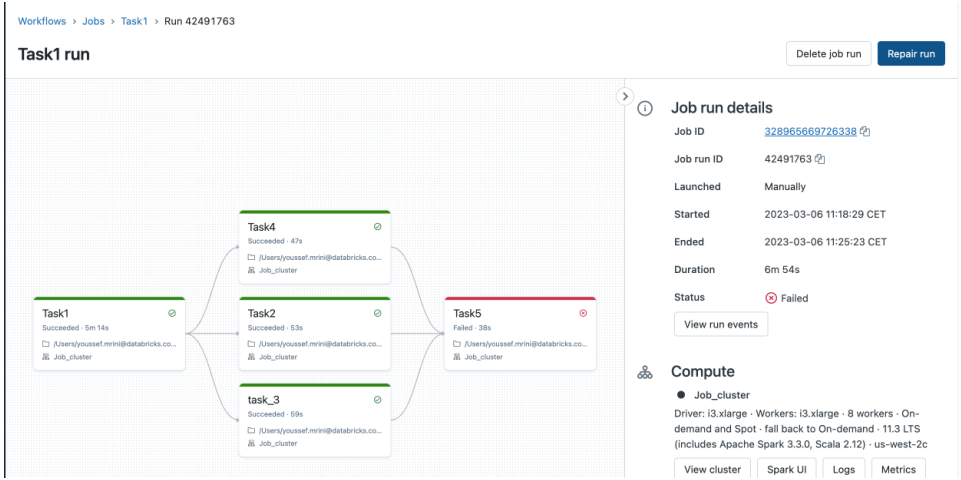
Databricks workflows

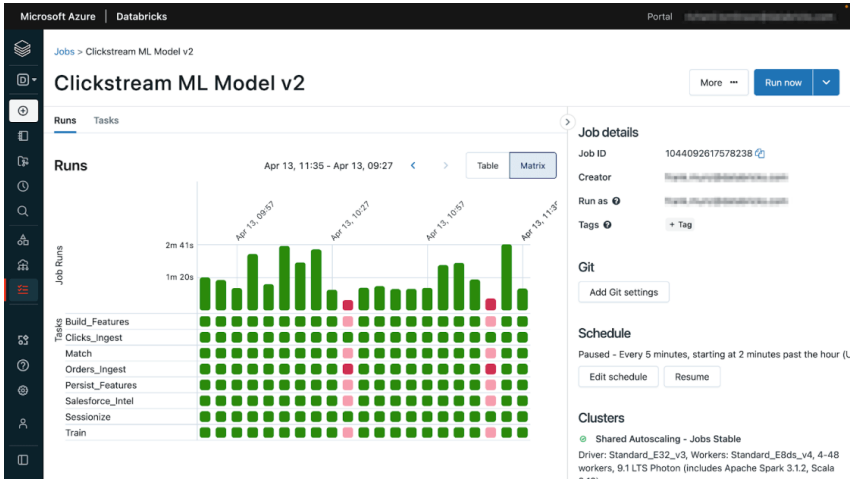
PROFINIT

- Orchestrates data processing, machine learning, and analytics pipelines on Databricks
- A workspace is limited to 1000 concurrent task runs
- Number of jobs a workspace can create in an hour is limited to 10000

Databricks workflows

PROFINIT





Delta Live Tables

PROFINIT

- Declarative framework for building reliable, maintainable, and testable data processing pipelines
- Delta Live Tables manage 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

PROFINIT

Dataset type	How are records processed through defined queries?
Streaming table	Each record is processed exactly once. This assumes an append-only source.
Materialized views	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).
Views	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.

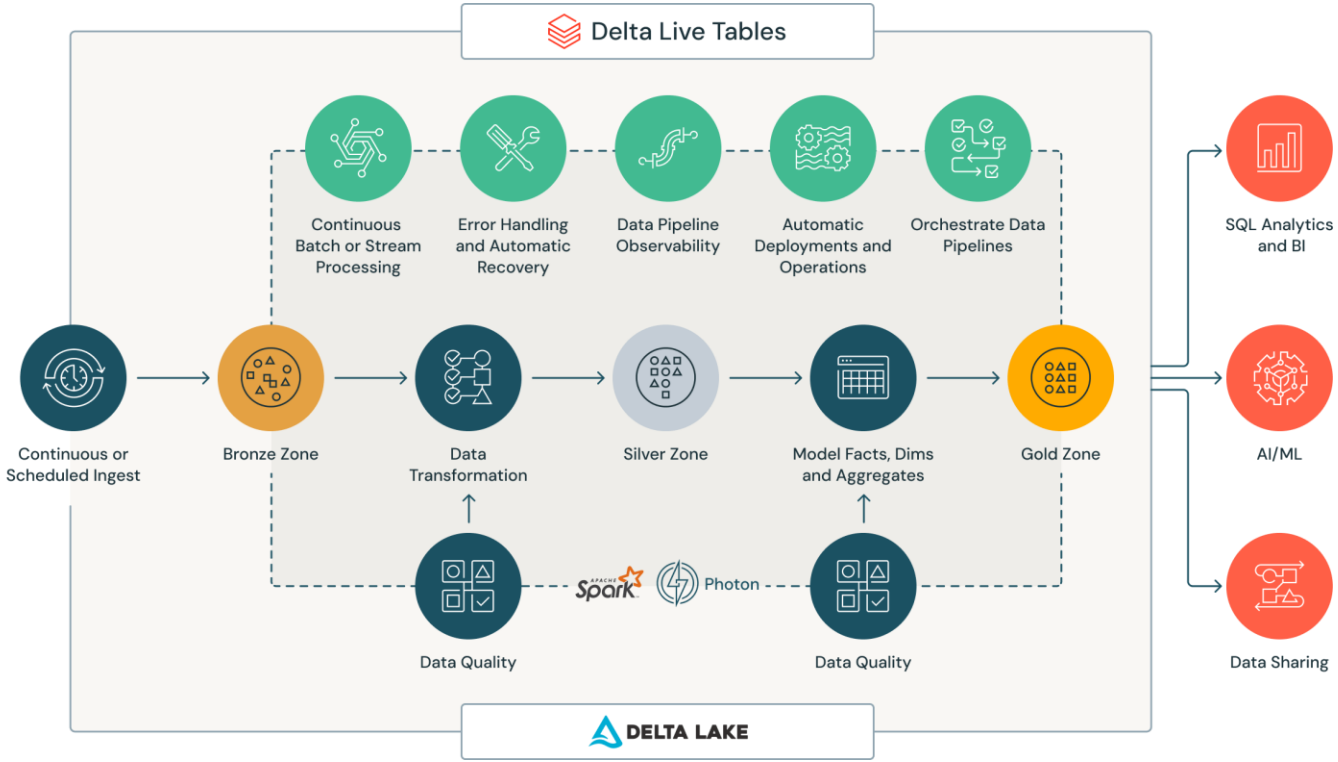
Delta Live Tables

PROFINIT

- Streaming table
  - Delta table with extra support for streaming or incremental data processing
- Materialized view
  - implements materialized views as Delta tables, but abstracts away complexities associated with efficient application of updates
  - refreshed according to the update schedule of the pipeline
- View
  - compute results from source datasets as they are queried, leveraging caching optimizations when available. Delta Live Tables does not publish views to the catalog

Delta Live Tables

PROFINIT



Delta Live Tables

PROFINIT

- https://github.com/databrickslabs/dlt-meta

```
@dlt.table(
  comment="Wikipedia clickstream data cleaned and prepared for analysis."
)
@dlt.expect("valid_current_page_title", "current_page_title IS NOT NULL")
@dlt.expect_or_fail("valid_count", "click_count > 0")
def clickstream_prepared():
  return (
    dlt.read("clickstream_raw")
      .withColumn("click_count", expr("CAST(n AS INT)"))
      .withColumnRenamed("curr_title", "current_page_title")
      .withColumnRenamed("prev_title", "previous_page_title")
      .select("current_page_title", "click_count", "previous_page_title")
  )
```

Optimization

PROFINIT

- Improve data layout of Delta Tables, improving read performance
  - OPTIMIZE
    - Reduce number of files present
    - Optimize a subset of data or colocate data by column. If you do not specify colocation, bin-packing optimization is performed
    - Bin-packing is idempotent (aims to produce evenly-balanced data files with respect to their size on disk)
  - ZORDER indexes
    - Data skipping technique for improved read performance
    - Data skipping information is collected automatically when you write data into a Delta table
    - Colocate related information in the same set of files
    - Usually run on a columns that are expected to be used frequently in query predicates and that has high cardinality
    - Not idempotent in general

Optimization

PROFINIT

```
OPTIMIZE events

OPTIMIZE events WHERE date >= '2017-01-01'

OPTIMIZE events
WHERE date >= current_timestamp() - INTERVAL 1 day
ZORDER BY (eventType)
```

Optimization

PROFINIT

- VACUUM command
  - removes all files from the table directory that are not managed by Delta, as well as data files that are no longer in the latest state of the transaction log for the table and are older than a retention threshold
  - omits directories starting with underscore (such as _delta_log)
  - files are lost after VACUUM, so you lose ability to time travel back to older versions

```
VACUUM table_name [RETAIN num HOURS] [DRY RUN]
```

Optimization

PROFINIT

- NOTE: To improve performance when writing to Delta Table, it might be beneficial to omit some columns for statistics collection, especially when it contains huge string values

  - By setting table property **delta.dataSkippingNumIndexedCols**

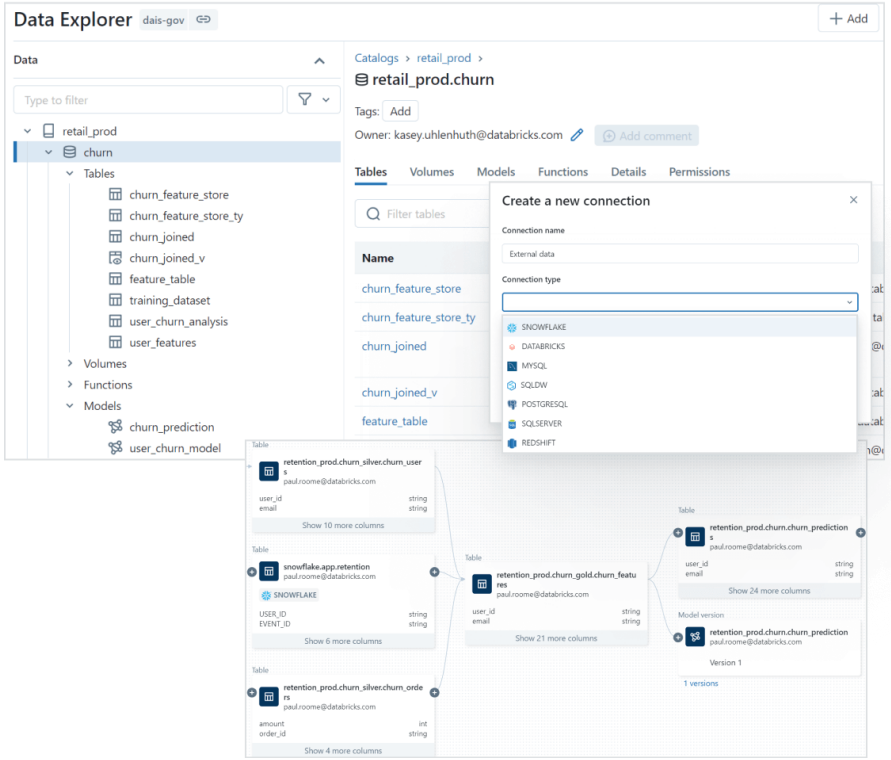
Data governance

Data governance

PROFINIT

Unity catalog

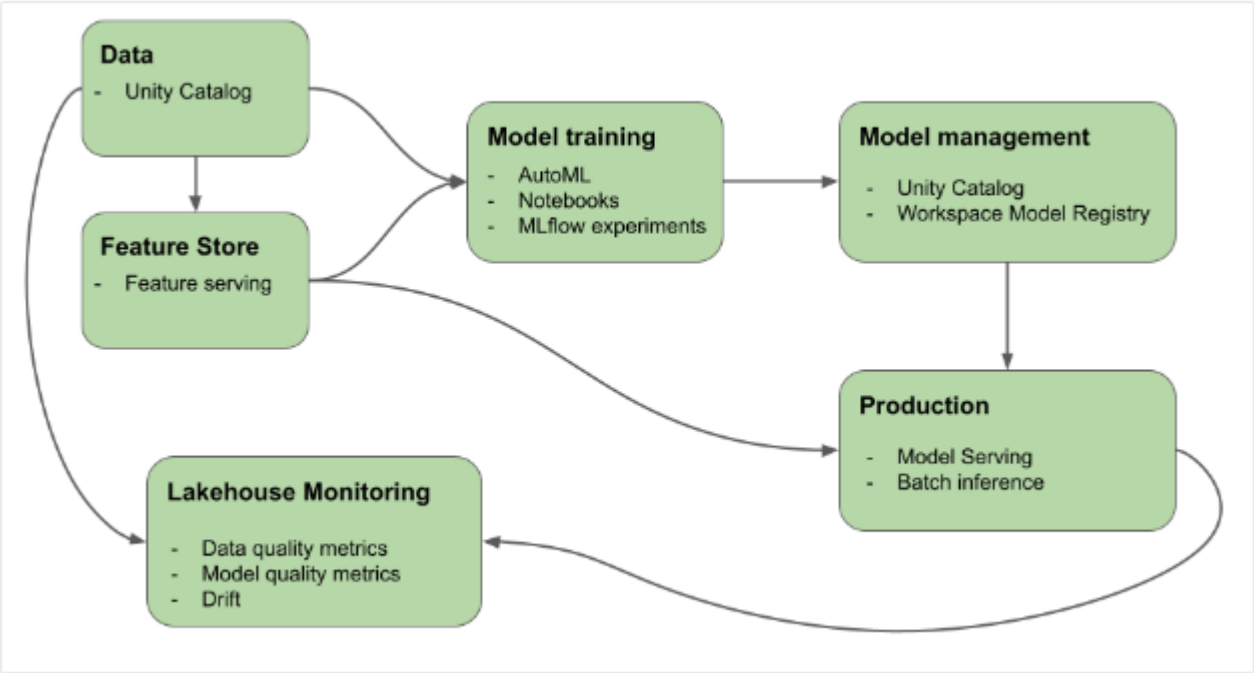
- Governance solution for data + AI assets
- Centralized metadata layer
- Abstraction over cloud-native role management
- Ability to set low-level permissions on UC objects
- Not only data lineage between tables
  - Notebooks, ML models
  - Workflows, DLT
  - Dashboards
  - …



ML development

ML development

PROFINIT



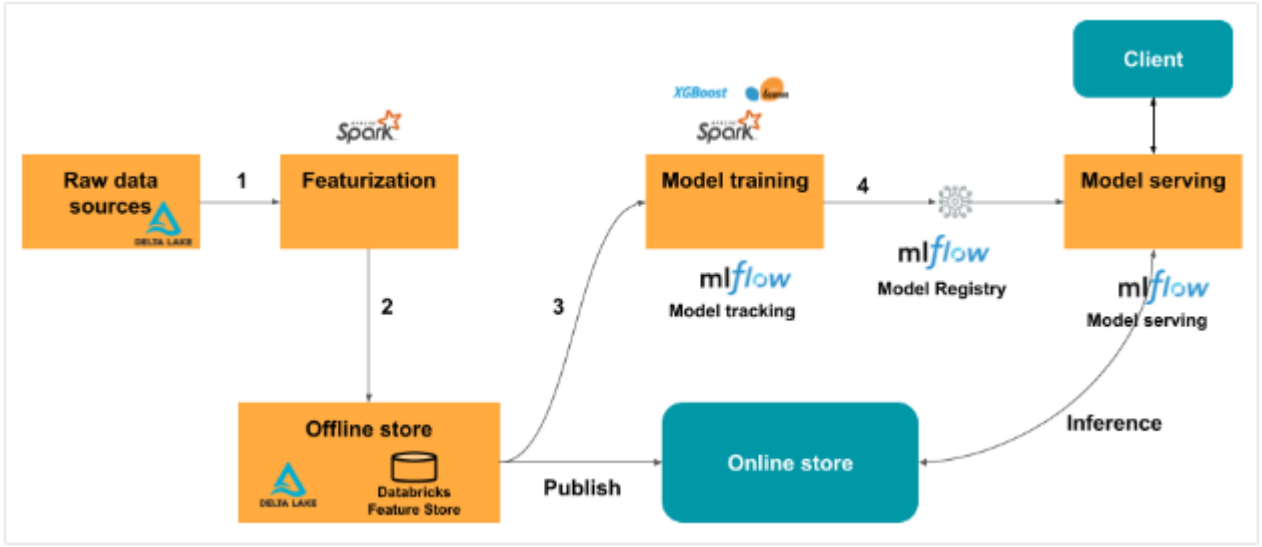
ML development

PROFINIT

- MLFlow
  - Managed ML development lifecycle
  - Logging of ML models, evaluation metrics, dependencies, other artifacts,…
- AutoML
  - Kick-off for ML projects
  - Can automatically run ML training on your data sources
  - Gives you notebooks for exploratory analysis and model training, ordered by model performance

ML development

PROFINIT



ML development

PROFINIT

- Model serving
  - exposes MLflow machine learning models as scalable REST API endpoints
  - Highly-available, low latency
  - Scalable
  - Serverless
- Lakehouse monitoring
  - Monitor the statistical properties and quality of the data in all of the tables
  - Data quality, data drift monitoring
  - Monitoring of model‘s performance

PROFINIT

Thanks for attention!

Profinit EU, s.r.o.
Tychonova 2, 160 00 Praha 6

Tel.: + 420 224 316 016, web: www.profinit.eu

LINKEDIN
linkedin.com/company/profinit

TWITTER
@profinit_EU

FACEBOOK
facebook.com/Profinit.EU

YOUTUBE
Profinit EU, s.r.o.