DATAPAO

DELTA LAKE & DELTA SHARING

Imre Greilich
Data Engineer

BUDAPEST DATA FORUM
2022

WHAT DO WE DO?

- we make data **understandable**, **accessible** and **useful**
- we deliver **solutions** to unsolvable problems
- we bring **order** to disorganized systems

DATAPAO HAS BEEN BUILT TO GUIDE YOU THROUGH THE WHOLE DATA JOURNEY

CONSULTING

We solve your toughest data challenges

- Data Engineering
- Data Science
- Cloud migration
- Pipeline development
- Query Optimization
- MLOps

EDUCATION

We train your internal data team

- Data Engineer and Data Scientist enablement
- Software Engineer transition to Data Engineer
- Data Platform and Data Cloud trainings

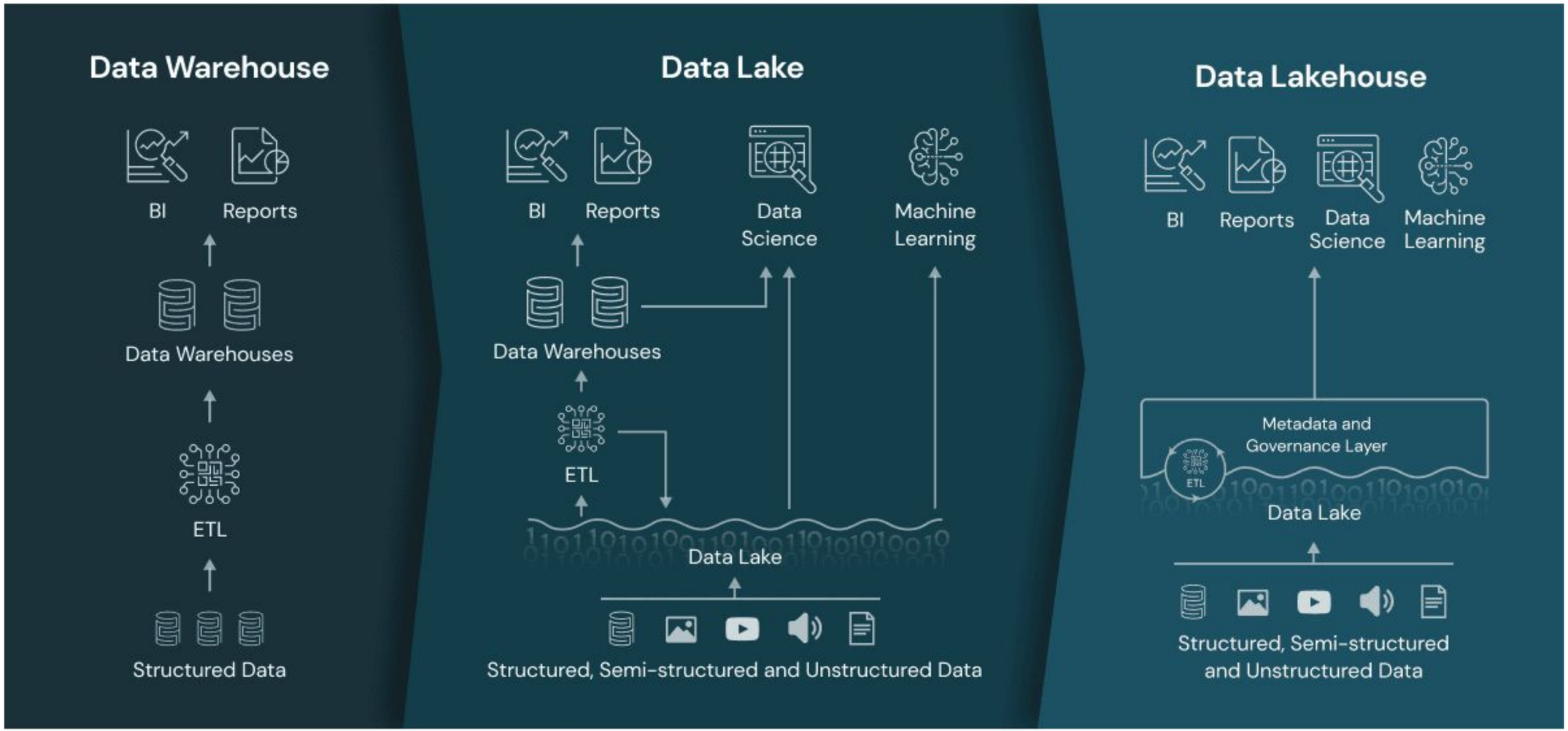
LABS

We pursue your industry's unanswered questions

- Cutting edge technologies
- Solution development
- Joint innovation efforts


DELTA LAKE

BRIEF STORY

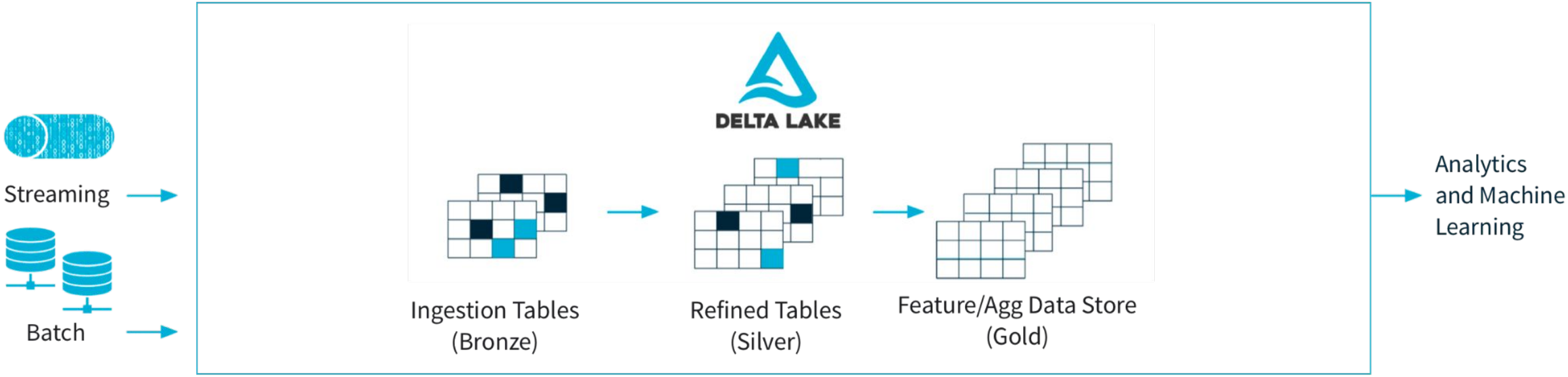

Data Warehouse
BI
Reports
Data Warehouses
ETL
Structured Data
Data Lake
BI
Reports
Data Science
Machine Learning
Data Warehouses
ETL
Data Lake
Structured, Semi-structured and Unstructured Data
Data Lakehouse
BI
Reports
Data Science
Machine Learning
Metadata and Governance Layer
ETL
Data Lake
Structured,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

WHAT IS DELTA LAKE?

- Open source storage layer built on parquet format
- Enables building a Lakehouse architecture
- Integrations with multiple compute engines
- APIs for Scala, Java, Rust, Ruby, and Python
- Simpler, faster, safe ETL

DELTA LAKE FEATURES

- ACID transactions
- Scalable metadata handling
- Time travel & Audit history
- Schema enforcement & Schema evolution
- Delete, update and merge support
- Streaming / batch unification



TRANSACTION LOG

- Ordered record of changes performed on the table
- Stored in the **_delta_log** folder
- Similar to the .git folder in a code repository
- Changes are stored in JSON files, once the transaction is “committed”
- Each JSON file contains the changes compared to the previous version

```
000000.json   Add 1.parquet
              Add 2.parquet
000001.json   Remove 1.parquet
              Remove 2.parquet
              Add 3.parquet
...
```

Parquet Table

```
# Review PARQUET_PATH folder
display(dbutils.fs.ls(PARQUET_PATH))
```

(3) Spark Jobs

	path
1	dbfs:/ml/loan_by_state_parquet/_SUCCESS
2	dbfs:/ml/loan_by_state_parquet/_committed_6800427028842607708
3	dbfs:/ml/loan_by_state_parquet/_started_6800427028842607708
4	dbfs:/ml/loan_by_state_parquet/part-00000-tid-6800427028842607708-2e96ca1c000.snappy.parquet

Delta Table

```
# Review DELTALAKE_PATH folder
display(dbutils.fs.ls(DELTALAKE_PATH))
```

(3) Spark Jobs

	path
1	dbfs:/ml/loan_by_state/_SUCCESS
2	dbfs:/ml/loan_by_state/_committed_830456464963115331
3	dbfs:/ml/loan_by_state/_delta_log/
4	dbfs:/ml/loan_by_state/_started_830456464963115331
	dbfs:/ml/loan_by_state/part-00000-tid-830456464963115331-a1d05b90-adfc-46ee-9692-a91a0

```
my_table/
  _delta_log/
    00000.json
    00001.json
  date=2019-01-01/
    file-1.parquet
```

Transaction Log
Table Versions
(Optional) Partition Directories
Data Files

A GLANCE INTO A SNAPSHOT OF THE TRANSACTION LOG

- Change metadata
- Add file
- Remove file
- Transaction identifiers
- Protocol evolution
- Commit info

```
{
  "commitInfo":{
    "timestamp":1515491537026,
    "userId":"100121",
    "userName":"michael@databricks.com",
    "operation":"INSERT",
    "operationParameters":{"mode":"Append","partitionBy":"[]"},
    "notebook":{
      "notebookId":"4443029",
      "notebookPath":"Users/michael@databricks.com/actions"},
      "clusterId":"1027-202406-pooh991"
  }
}
```

```
{
  "add": {
    "path":"date=2017-12-10/part-000...c000.gz.parquet",
    "partitionValues":{"date":"2017-12-10"},
    "size":841454,
    "modificationTime":1512909768000,
    "dataChange":true
    "stats":"{\"numRecords\":1,\"minValues\":{\"val..."
  }
}
```

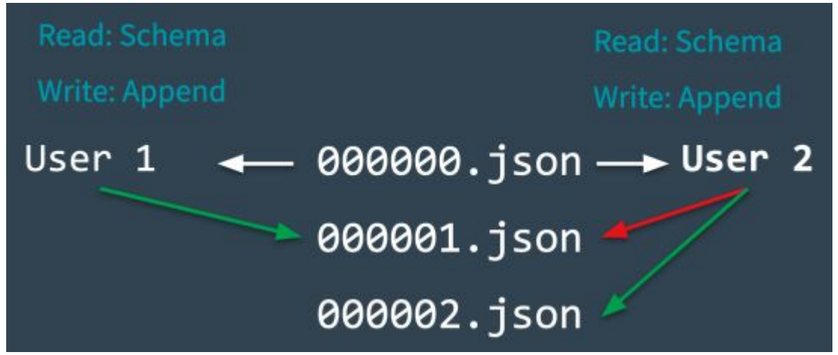
```
{
  "remove":{
    "path":"part-00001-9…..snappy.parquet",
    "deletionTimestamp":1515488792485,
    "dataChange":true
  }
}
```

OPTIMISTIC CONCURRENCY CONTROL

3 steps of a write operation

1. **Read** the latest version of the table
2. **Stage** changes by writing new data files
3. **Validate** - commit new snapshot or reject

	INSERT	UPDATE, DELETE, MERGE INTO	COMPACTION
INSERT	Cannot conflict		
UPDATE, DELETE, MERGE INTO	Can conflict	Can conflict	
COMPACTION	Cannot conflict	Can conflict	Can conflict



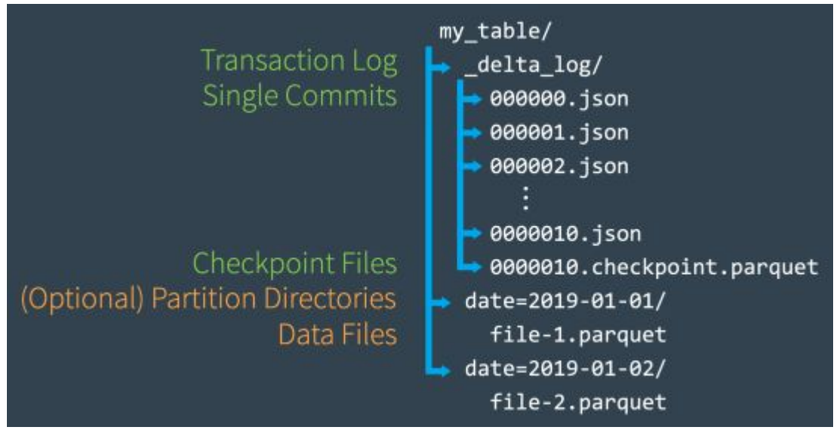
Avoid conflicts by partitioning

- E.g. partition by date, if you usually filter for date
- If the intervals don't overlap, then different partitions will be written → won't conflict

CHECKPOINT FILES

- After each 10 commits a checkpoint file is created
- Parquet format (not JSON)
- Contains the union of the preceding transactions
- Much faster to reproduce the table's state
- Also deletes old log files older than 30 days by default

delta.logRetentionDuration = "interval <interval>"



SCHEMA ENFORCEMENT & SCHEMA EVOLUTION

- Schema is enforced on the source dataset
- If the schema changes, and you want to propagate the update to the sink, you can overwrite the schema
- Adding new columns to the schema can be automated for the operation or for the whole session

- `write` or `writeStream` have `.option("mergeSchema", "true")`
- `spark.databricks.delta.schema.autoMerge.enabled` is `true`

Change a column type

Python

```python
spark.read.table(...) \
  .withColumn("birthDate", col("birthDate").cast("date")) \
  .write \
  .format("delta") \
  .mode("overwrite")
  .option("overwriteSchema", "true") \
  .saveAsTable(...)
```

Change a column name

Python

```python
spark.read.table(...) \
  .withColumnRenamed("dateOfBirth", "birthDate") \
  .write \
  .format("delta") \
  .mode("overwrite") \
  .option("overwriteSchema", "true") \
  .saveAsTable(...)
```


Getting started

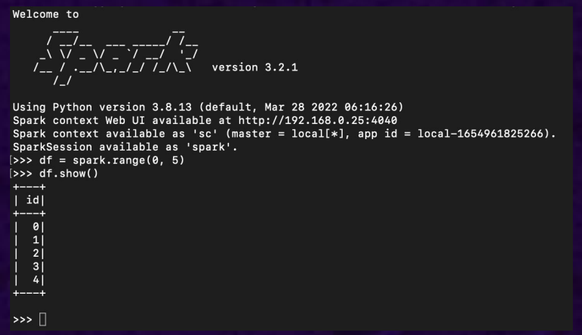
GETTING STARTED

PySpark Shell

- Check version compatibility

```
pip install pyspark

pyspark --packages io.delta:delta-core_2.12:1.2.1 --conf
"spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension" --conf
"spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog"
```



Databricks Community Edition

- Sign Up
- Insert profile details
- “Get started for free”
- “Get started with Community Edition”

Instead of using “parquet” format...

```
df.write.format("parquet").save("/data/test_parquet")
```

...simply use “delta”

```
df.write.format("delta").save("/data/test_delta")
```

BASIC SYNTAX - SQL

```sql
-- Create table in the metastore
CREATE TABLE people (
  id INT,
  firstName STRING,
  lastName STRING,
  ...
)
USING DELTA
PARTITIONED BY (...)


-- Create table in the metastore using existing data
CREATE TABLE people

USING DELTA

LOCATION '/tmp/delta/people'


-- Insert into a table with overwrite option
INSERT OVERWRITE TABLE default.people SELECT * FROM morePeople


-- Delete from a table
DELETE FROM people WHERE birthDate < '1955-01-01'


-- Update a table
UPDATE people SET gender = 'Female' WHERE gender = 'F';
```

BASIC SYNTAX - PYTHON

```python
# Create table in the metastore
DeltaTable.createIfNotExists(spark) \
  .tableName("people") \
  .addColumn("id", "INT") \
  .addColumn("firstName", "STRING") \
  ...
  .execute()

# Append data to a table
df.write.format("delta").mode("append").save("/tmp/delta/people")
df.write.format("delta").mode("append").saveAsTable("people")

# Delete from a delta table
deltaTable = DeltaTable.forPath(spark, '/tmp/delta/people')
deltaTable.delete("birthDate < '1955-01-01'")

# Update data in a delta table
deltaTable.update(condition = "gender = 'F'", set = { "gender": "'Female'" })
```

MERGE - SQL

```
MERGE INTO logs
USING newDedupedLogs
ON logs.uniqueId = newDedupedLogs.uniqueId AND logs.date > current_date() - INTERVAL 7 DAYS
WHEN NOT MATCHED AND newDedupedLogs.date > current_date() - INTERVAL 7 DAYS
   THEN INSERT *
```

MERGE - PYTHON

```
deltaTable.alias("logs").merge(
    newDedupedLogs.alias("newDedupedLogs"),
    "logs.uniqueId = newDedupedLogs.uniqueId AND logs.date > current_date() - INTERVAL 7 DAYS"  ) \
  .whenNotMatchedInsertAll( "newDedupedLogs.date > current_date() - INTERVAL 7 DAYS" ) \
  .execute()
```

Utility commands

VACUUM

- Removes files no longer referenced by a Delta table (i. e. log files were removed)
- Default retention period is 7 days

DESCRIBE HISTORY

- Shows audit logs for the delta table
- Perfect input for Time Travel, statistics and performance optimization
- Operations e.g.: write, create table, streaming update, delete, optimize, truncate, merge, convert, restore

version	timestamp	userId	userName	operation	operationParameters	job	notebook
2	2022-06-12T13:40:45.000+0000	7123948404436735	.com	WRITE	{"mode": "Append", "partitionBy": "[]"}	null	{"notebookId": "3517172034562804"}
1	2022-06-12T13:40:36.000+0000	7123948404436735	.com	WRITE	{"mode": "Append", "partitionBy": "[]"}	null	{"notebookId": "3517172034562804"}
0	2022-06-11T15:45:50.000+0000	7123948404436735	.com	WRITE	{"mode": "ErrorIfExists", "partitionBy": "	null	{"notebookId": "3517172034562804"}

clusterId	readVersion	isolationLevel	isBlindAppend	operationMetrics	userMetadata	engineInfo
0612-133509-21xthguf	1	WriteSerializable	true	{"numFiles": "8", "numOutputRows": "140000", "numOutputBytes": "565361"}	null	Databricks-Runtime/10.4.x-scala2.12
0612-133509-21xthguf	0	WriteSerializable	true	{"numFiles": "8", "numOutputRows": "1490", "numOutputBytes": "10900"}	null	Databricks-Runtime/10.4.x-scala2.12
0611-151821-9bmsdxvg	null	WriteSerializable	true	{"numFiles": "5", "numOutputRows": "5", "numOutputBytes": "3000"}	null	Databricks-Runtime/10.4.x-scala2.12

UTILITY COMMANDS

TIME TRAVEL

- Query earlier versions of a delta table
- Restore to an earlier version (e.g. after an accidental delete/update, or a buggy pipeline run)
- Reproduce experiments

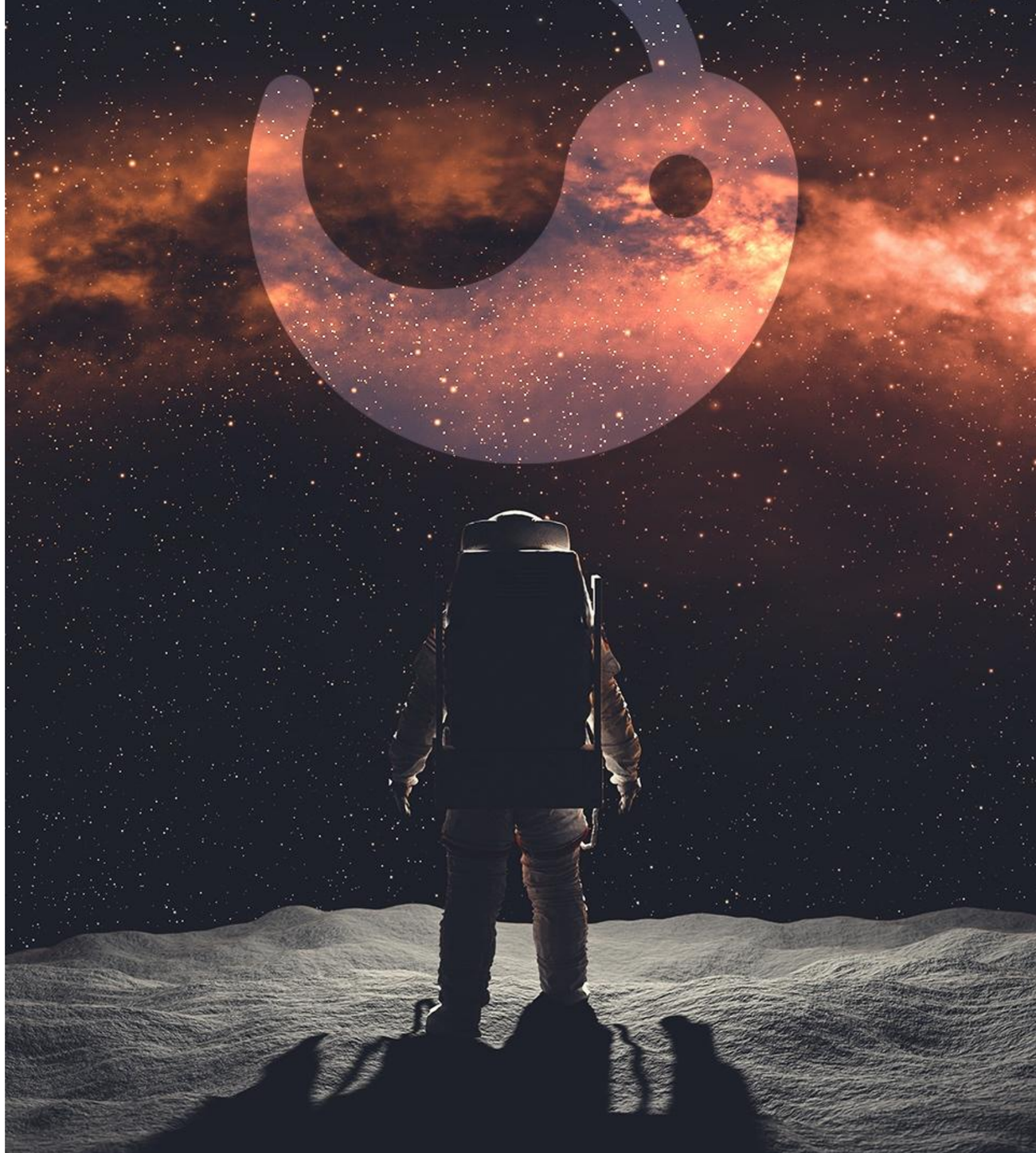
```
RESTORE TABLE db.target_table TO VERSION AS OF <version>
RESTORE TABLE delta.`/data/target/` TO TIMESTAMP AS OF <timestamp>
```

CLONE

- Shallow clone
- Deep clone

OPTIMIZE

- Improves query speed by coalescing small files into larger ones
- Optional Z-order to physically order the records
- The data will not change



UTILITY COMMANDS

CONVERT A PARQUET TABLE TO DELTA

- SQL: CONVERT TO DELTA parquet.`<path-to-table>`
- Python: DeltaTable.convertToDelta()

CONVERT A DELTA TABLE TO PARQUET

- Run VACUUM to delete data from previous versions
- Delete the _delta_log directory

DELTA LAKE ROADMAP

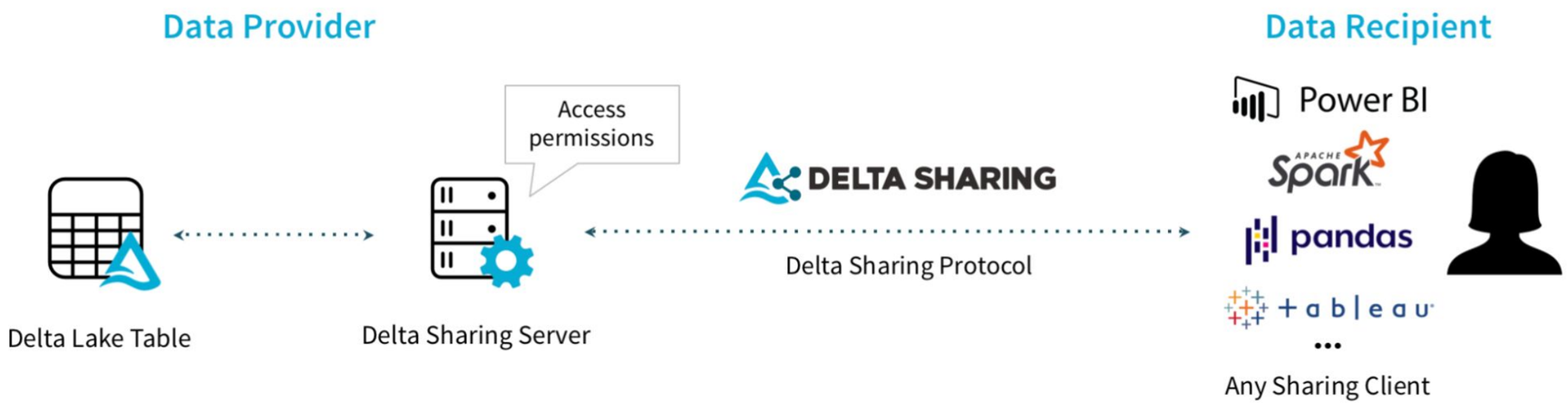
DELTA LAKE ROADMAP

FEATURE	DESCRIPTION	TARGET
OPTIMIZE	Table optimize is an operation to rearrange the data and/or metadata to speed up queries and/or reduce the metadata size	released (1.2)
File skipping using column stats	This is a performance optimization that aims at speeding up queries that contain filters (WHERE clauses) on non-partitionBy columns.	released (1.2)
RESTORE	Rollback to a previous version of a Delta table using Python, Scala, and/or SQL APIs.	released (1.2)
OPTIMIZE ZORDER	Data clustering via multi-column locality-preserving space-filling curves with offline sorting.	2022 Q3/Q4
CLONE	Clones a source Delta table to a target destination at a specific version. A clone can be either deep or shallow: deep clones copy over the data from the source and shallow clones do not.	2022 Q3
Change Data Feed	The Delta change data feed represents row-level changes between versions of a Delta table. When enabled on a Delta table, the runtime records “change events” for all the data written into the table.	2022 Q3

DELTA SHARING

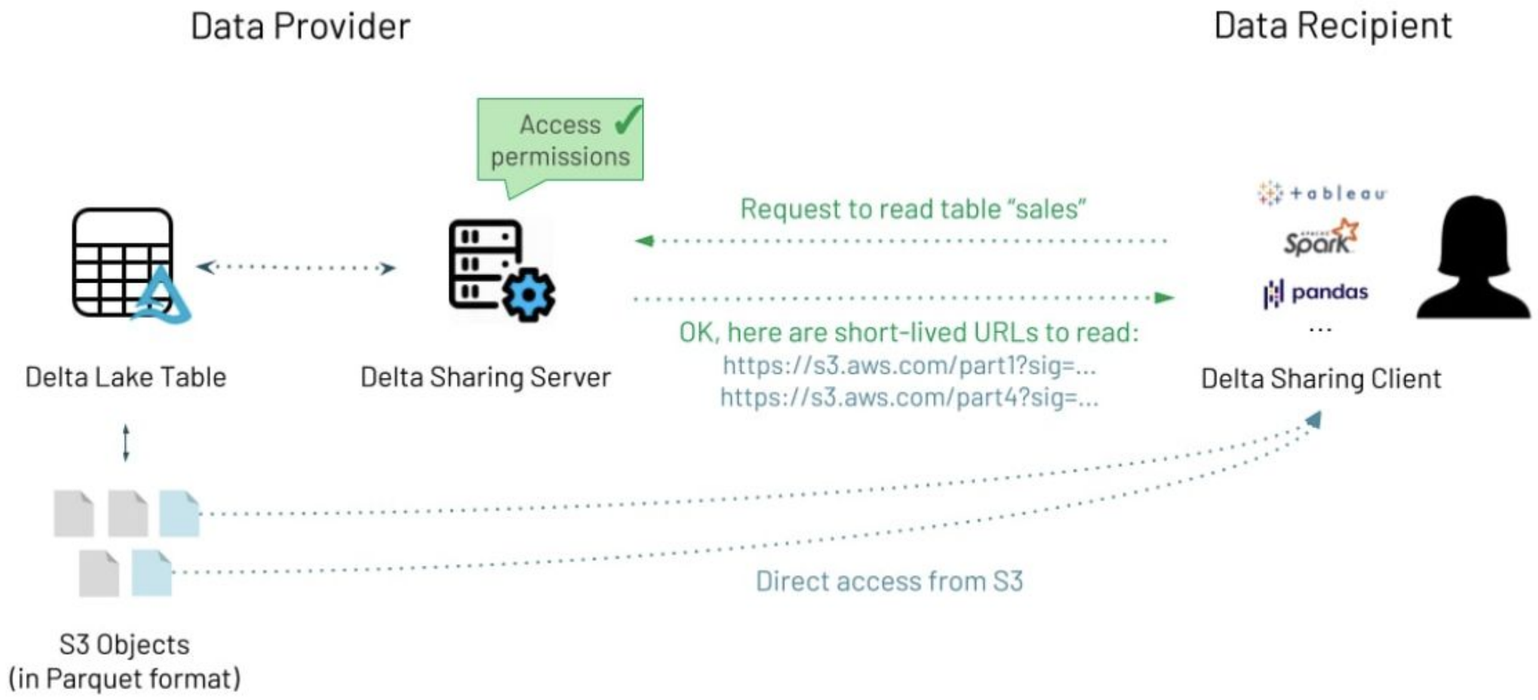
OVERVIEW

- Open REST protocol for secure real-time exchange of large datasets
- Share data with other organizations regardless of computing platform
- Direct connection without copying
- Strong security, auditing and governance



HOW DOES IT WORK

1. The recipient's client authenticates to the sharing server
2. The server verifies the client's privileges and determines which subset to send back
3. The server generates short-lived URLs that allow to read the parquet files directly. Transfer can happen in parallel at massive bandwidth



SETTING UP IN DATABRICKS

I already have a table that I want to share	select * from vaccine_data.vaccinations
Create a share	create share vaccine_data
Add some tables to the share	alter share vaccine_data add table vaccine_data.vaccinations; alter share vaccine_data add table vaccine_data.distributors;
Check if the tables are correctly added (shows the tables added to the share)	describe share vaccine_data
Create a recipient (returns the activation link for the recipient)	create recipient cdc
Send the link to the partner, who can use it to download a credential file	Download Credential File
Grant access to the recipient	grant select on share vaccine_data to recipient cdc

READING TABLES WITH PYTHON

Import the library (first install it)	import delta_sharing
List all available tables at that share	delta_sharing.SharingClient(share_file_location).list_all_tables()
Load one table to a Spark DataFrame	df = delta_sharing \ .load_as_spark(share_file_location + "#vaccine_data.vaccine_data.vaccinations")
Load one table to a Pandas DataFrame	df = delta_sharing \ .load_as_pandas(share_file_location + "#vaccine_data.vaccine_data.vaccinations")

ROADMAP

- Sharing other objects than tables:
  - Data streams
  - ML models (MLflow)
  - Views
  - Arbitrary files
- Time-based sharing permissions (0.3.0)

WE MAKE DATA YOUR SUPERPOWER

IMRE GREILICH
DATA ENGINEER

+36-20-211-8484
imre.greilich@datapao.com
datapao.com

DATAPAO