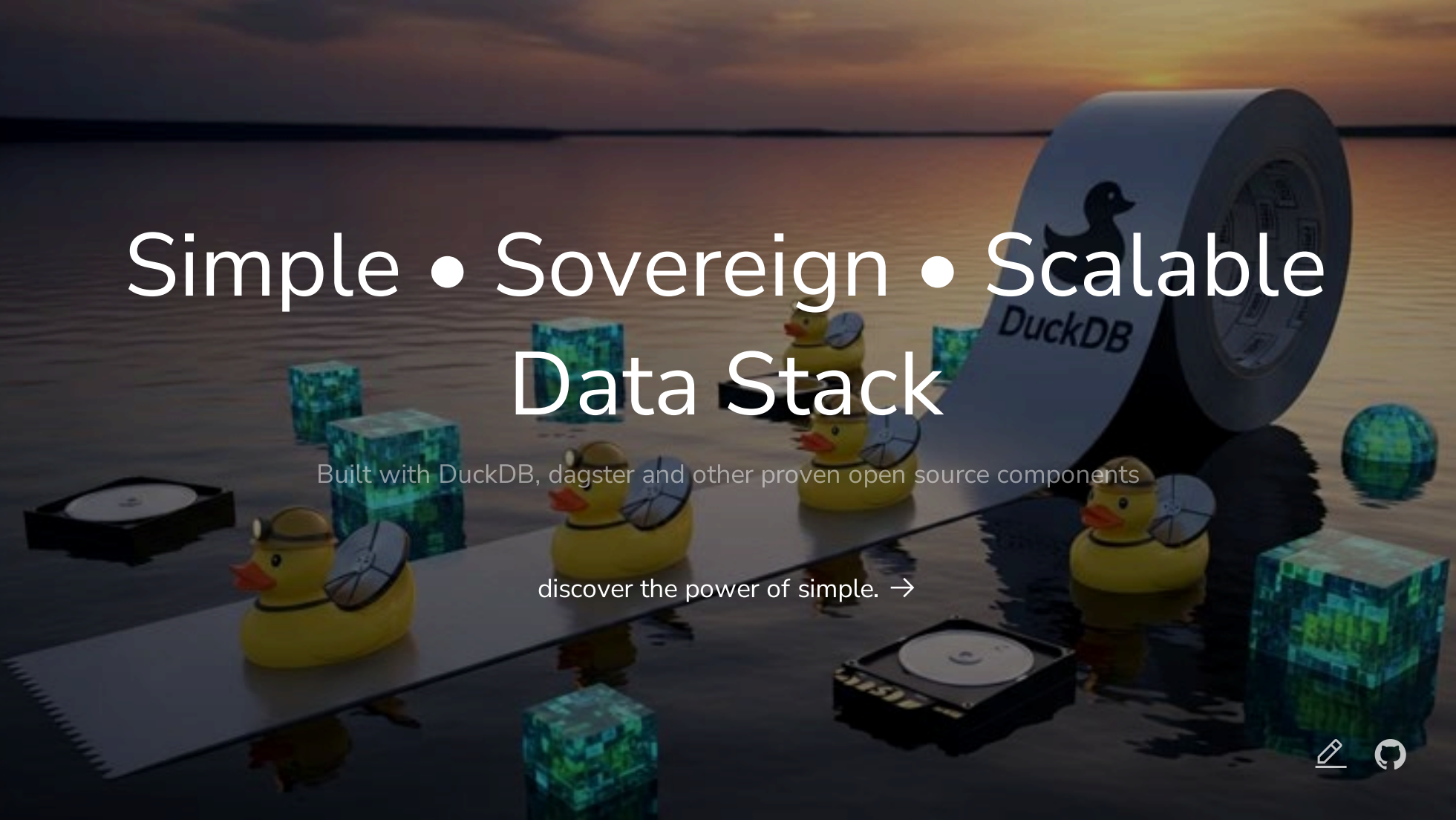


Simple • Sovereign • Scalable Data Stack

Built with DuckDB, dagster and other proven open source components

discover the power of simple. →

Georg Heiler

Solving challenges with data.

- Senior data expert @Magenta
- Research Software Engineer @ASCII & CSH
- Meetup organizer & frequent Speaker

geoheil.com, linkedin.com/in/geoheil

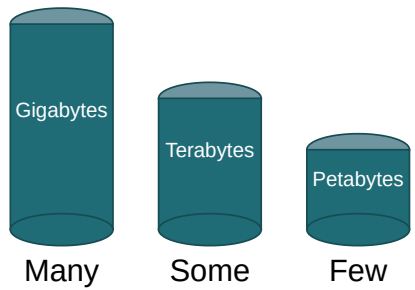
History

- Mainframe
- Data warehouse
- Big Data (Hadoop)
- SQL on large data (Hive, Spark)
- Cloud DWH (Snowflake, BigQuery)

- Cloud exit?
- SQL is there to stay!

Medium-sized data

- Much of the time we work with medium-sized data
- Too big for traditional data processing applications, too small to justify complex data infrastructure



data accumulates over time

Partitions

local data

Reduced latency
Enhanced control (privacy, quality)
Not just laptop; k8s is fine

Where does DuckDB fit?

	Transactional	Analytical
Embedded	SQLite RocksDB	**DuckDB**
Stand-alone	Oracle Postgres MongoDB	BigQuery Snowflake Starrocks

DuckDB

- Fast, in-process SQL OLAP database
- Vectorized query engine
- Real database (ACID, memory manager)
- Flexible IO: No ingest mandatory (virtualization?)
- Extension ecosystem
  - `spatial`
  - `duckpgq` (graphs)
  - `vectors` (embeddings)
  - `fts` (full text search)
- Innovative and convenient SQL syntax
- Query plan visualizer

DuckDB basics

```
duckdb
```

DuckDB basics

```
CREATE TABLE t1 (
  grp VARCHAR,
  val INT
);

INSERT INTO t1 (grp, val) VALUES
  ('a', 2),
  ('a', 1),
  ('b', 5),
  ('b', 4),
  ('a', 3),
  ('b', 6);

FROM t1;
```

grp	val
varchar	int32
a	2
a	1
b	5
b	4

DuckDB basics

```
-- find the tok-k largest values per group

-- traditional window-function based approach
SELECT array_agg(rs.val), rs.grp
FROM (SELECT val, grp, row_number() OVER (PARTITION BY grp ORDER BY val DESC) AS rid
      FROM t1
      ORDER BY val DESC) AS rs
WHERE rid < 4
GROUP BY rs.grp;

┌──────────────────┬─────────┐
│ array_agg(rs.val) │   grp   │
│     int32[]      │ varchar │
├──────────────────┼─────────┤
│ [3, 2, 1]        │ a       │
│ [6, 5, 4]        │ b       │
└──────────────────┴─────────┘
```

DuckDB basics

```
-- duckdb: shorter + more efficient
SELECT grp, max(val, 3) FROM t1 GROUP BY ALL;

┌─────────┬──────────────┐
│   grp   │ max(val, 3)  │
│ varchar │   int32[]    │
├─────────┼──────────────┤
│ a       │ [3, 2, 1]    │
│ b       │ [6, 5, 4]    │
└─────────┴──────────────┘

-- simpler
-- faster
```

DuckDB basics

```sql
-- direct IO, no prior ingestion needed
SELECT
    station_name,
    count(*) AS num_services
FROM 's3://duckdb-blobs/train_services.parquet'
GROUP BY ALL
ORDER BY num_services DESC
LIMIT 10;
```

DuckDB basics

```
INSTALL stochastic FROM community;

-- Generate synthetic dataset
CREATE TABLE synthetic_data AS
SELECT
    i,
    dist_normal_sample(100, 15) AS height_cm,
    dist_normal_sample(70, 10) AS weight_kg,
    dist_binomial_sample(1, 0.5) AS gender  -- 0 or 1
FROM range(1000) t(i);

-- Calculate z-scores
SELECT
    height_cm,
    (height_cm - dist_normal_mean(100, 15)) / dist_normal_stddev(100, 15) AS height_zscore
FROM synthetic_data;

-- Probability calculations
SELECT
    weight_kg,
    dist_normal_cdf(70, 10, weight_kg) AS percentile
FROM synthetic_data
LIMIT 10;
```

DuckDB basics

```
-- MERGE INTO
CREATE TABLE item(item_name varchar, count int);
INSERT INTO item
VALUES ('book', 3), ('computer', 5);

WITH buy(item_name, count) AS (VALUES ('book', 2), ('phone', 3)) MERGE INTO item
USING buy
ON (item.item_name = buy.item_name)
WHEN MATCHED THEN
    UPDATE
    SET count = item.count + buy.count
WHEN NOT MATCHED THEN
    INSERT;
FROM item;

┌─────────────┬─────────┐
│  item_name  │  count  │
│   varchar   │  int32  │
├─────────────┼─────────┤
│ book        │       5 │
│ computer    │       5 │
│ phone       │       3 │
└─────────────┴─────────┘
```

DuckDB basics

```
-- MERGE INTO deletions
WITH sell(item_name, count) AS (VALUES ('book', 5), ('computer', 2))
    MERGE INTO item
USING sell
ON (item.item_name = sell.item_name)
WHEN MATCHED
    AND sell.count > item.count
    THEN ERROR CONCAT('Trying to sell ', sell.count, ' copies of item ',
      item.item_name, ' but we only have ', item.count)
    WHEN MATCHED AND sell.count = item.count THEN
DELETE
    WHEN MATCHED THEN
UPDATE SET count = item.count - sell.count
    WHEN NOT MATCHED THEN ERROR CONCAT('Trying to sell item ',
      sell.item_name, ', but we don''t have it in stock');
FROM item;
```

item_name	count
varchar	int32
computer	3
phone	3

DuckDB basics

```
INSTALL spatial;
LOAD spatial;

-- List the closest IC stations (as the crow flies)
CREATE TABLE stations AS FROM 's3://duckdb-blobs/stations.csv';

SELECT
    s1.name_long AS station1,
    s2.name_long AS station2,
    ST_Distance(
        ST_Point(s1.geo_lng, s1.geo_lat),
        ST_Point(s2.geo_lng, s2.geo_lat)
    ) * 111_139 AS distance
FROM stations s1, stations s2
WHERE s1.type LIKE '%Intercity%'
  AND s2.type LIKE '%Intercity%'
  AND s1.id < s2.id
ORDER BY distance ASC
LIMIT 3;
```

DuckDB basics

```
INSTALL vss;
LOAD vss;

CREATE TABLE my_vector_table (vec FLOAT[3]);
INSERT INTO my_vector_table
    SELECT array_value(a, b, c)
    FROM range(1, 10) ra(a), range(1, 10) rb(b), range(1, 10) rc(c);
CREATE INDEX my_hnsw_index ON my_vector_table USING HNSW (vec);

SELECT *
FROM my_vector_table
ORDER BY array_distance(vec, [1, 2, 3]::FLOAT[3])
LIMIT 3;


-- quick one-shot nearest neighbor searches.
SELECT min_by(my_vector_table, array_distance(vec, [1, 2, 3]::FLOAT[3]), 3 ORDER BY vec) AS result
FROM my_vector_table;
```

DuckDB basics

```
INSTALL fts;
LOAD fts;

CREATE TABLE documents
(
    document_identifier VARCHAR,
    text_content        VARCHAR,
    author              VARCHAR,
    doc_version         INTEGER
);
INSERT INTO documents
VALUES ('doc1',
        'The mallard is a dabbling duck that breeds throughout the temperate.',
        'Hannes Mühleisen',
        3),
       ('doc2',
        'The cat is a domestic species of small carnivorous mammal.',
        'Laurens Kuiper',
        2);
```

DuckDB basics

```
PRAGMA create_fts_index(
    'documents', 'document_identifier', 'text_content', 'author'
);

SELECT document_identifier, text_content, score
FROM (SELECT *,
             fts_main_documents.match_bm25(
                     document_identifier,
                     'Muhleisen',
                     fields := 'author'
             ) AS score
      FROM documents) sq
WHERE score IS NOT NULL
  AND doc_version > 2
ORDER BY score DESC;
```

document_identifier varchar	text_content varchar	score double
doc1	The mallard is a dabbling duck that breeds throughout the temperate.	0.3094700890003546

DuckDB basics

```
ATTACH 'https://github.com/Dtenwolde/duckpgq-docs/raw/refs/heads/main/datasets/snb.duckdb';

USE snb;
install duckpgq FROM community;
LOAD
duckpgq;

CREATE OR REPLACE
PROPERTY GRAPH snb
VERTEX TABLES (
  Person, Forum
)
EDGE TABLES (
  Person_knows_person     SOURCE KEY (Person1Id) REFERENCES Person (id)
                          DESTINATION KEY (Person2Id) REFERENCES Person (id)
                          LABEL knows,
  Forum_hasMember_Person  SOURCE KEY (ForumId) REFERENCES Forum (id)
                          DESTINATION KEY (PersonId) REFERENCES Person (id)
                          LABEL hasMember
);
```

DuckDB basics

```
-- find the shortest path from one person to all other persons
FROM GRAPH_TABLE (snb
  MATCH p = ANY SHORTEST (p1:person WHERE p1.id = 14)-[k:knows]->*(p2:person)
  COLUMNS (p1.id, p2.id as other_person_id, element_id(p), path_length(p))
);

┌──────────┬─────────────────┬──────────────────┬─────────────────┐
│    id    │ other_person_id │ element_id(p)    │ path_length(p)  │
│  int64   │      int64      │     int64[]      │      int64      │
├──────────┼─────────────────┼──────────────────┼─────────────────┤
│       14 │              14 │ [0]              │               0 │
│       14 │  10995116277782 │ [0, 0, 13]       │               1 │
│       14 │  24189255811081 │ [0, 1, 26]       │               1 │
│       14 │  24189255811109 │ [0, 1, 26, 61, 27] │             2 │
│       14 │  26388279066641 │ [0, 0, 13, 42, 29] │             2 │
```

DuckDB basics

```
-- query, modify and serve data with Apache Arrow Flight
-- like plane: you can bring back things, and leave stuff there

INSTALL airport FROM community;
LOAD airport;

-- Attach a database named 'hello'
ATTACH 'hello' (TYPE AIRPORT, location 'grpc+tls://hello-airport.query.farm');
-- Query the remote data
 SELECT * FROM hello.static.greetings WHERE language = 'Dutch';
```

DuckDB basics

```sql
-- Interact with real-time event systems, Websockets, Redis Pub/Sub

INSTALL radio FROM community;
LOAD radio;

-- Open a connection to a Bluesky websocket server
CALL radio_subscribe('wss://jetstream2.us-east.bsky.network/subscribe');
-- Block until a message is received for 10 seconds.
CALL radio_listen(true, interval '10 seconds');
-- Show messages
SELECT * FROM radio_received_messages();
```

subscription_id uint64	subscription_url varchar	message_id uint64	message_type received_message_t…	…	seen_count uint64	channel varchar	message blob
0	wss://jetstream2.u…	20070	message	…	1	NULL	{\x22did\x22:\x22d…
0	wss://jetstream2.u…	20071	message	…	1	NULL	{\x22did\x22:\x22d…
0	wss://jetstream2.u…	20072	message	…	1	NULL	{\x22did\x22:\x22d…
0	wss://jetstream2.u…	20073	message	…	1	NULL	{\x22did\x22:\x22d…
0	wss://jetstream2.u…	20074	message	…	1	NULL	{\x22did\x22:\x22d…

DuckDB basics

```sql
-- decoding the blob
WITH msgs AS (
    SELECT CAST(decode(message) AS JSON) AS j
    FROM radio_received_messages()
  )
  SELECT
    j->>'did'                          AS did,
    j->>'kind'                         AS kind,
    j->'commit'->>'cid'                AS cid,
    j->'time_us'                       AS time_us,
    j                                  AS full_json
  FROM msgs;
```

did varchar	kind varchar	cid varchar	time_us json	full_json json
did:plc:gzspoik7ex…	commit	bafyreibzq7urxiyso…	1759050634291941	{"did":"did:plc:gzspoik7ext7svhqylb6knhj","time_…
did:plc:4ulxz5k46g…	commit	bafyreibulbylvjloa…	1759050634292491	{"did":"did:plc:4ulxz5k46g3kk3ha5gtq7iyc","time_…
did:plc:fuyyz7t2xy…	commit	bafyreienqydw6ixak…	1759050634293361	{"did":"did:plc:fuyyz7t2xy6tf7726g7xckte","time_…

DuckDB basics

```
INSTALL tributary FROM community;
LOAD tributary;

-- Read all messages on a topic
SELECT * FROM tributary_scan_topic('test-topic', "bootstrap.servers" := 'kafka-cluster.query.farm:9092');
```

DuckDB basics

```
-- string similarity (fuzzy matching)
INSTALL rapidfuzz FROM community;
LOAD rapidfuzz;

SELECT rapidfuzz_ratio('hello world', 'helo wrld');
┌──────────────────────────────────────────────┐
│ rapidfuzz_ratio('hello world', 'helo wrld')  │
│                    double                    │
├──────────────────────────────────────────────┤
│                     90.0                     │
└──────────────────────────────────────────────┘

SELECT rapidfuzz_partial_ratio('hello world', 'world');
┌──────────────────────────────────────────────────┐
│ rapidfuzz_partial_ratio('hello world', 'world')  │
│                      double                      │
├──────────────────────────────────────────────────┤
│                       100.0                      │
└──────────────────────────────────────────────────┘
```

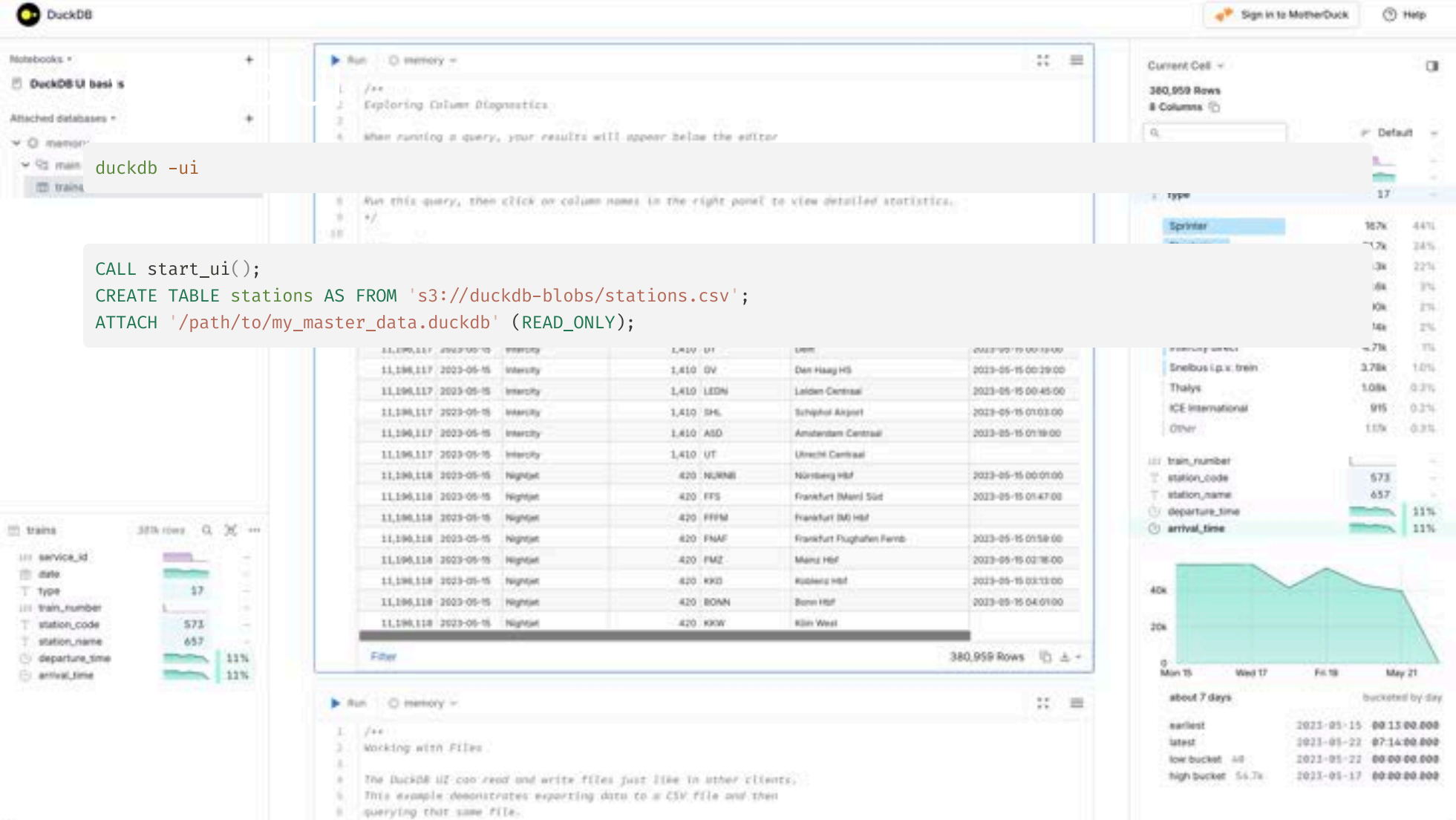
DuckDB basics

```
-- probabilistic data structures
CREATE OR REPLACE TABLE big AS
SELECT CAST(random() * 1_000_000_000 AS INTEGER) AS user_id,
       CASE
           WHEN i % 10 < 6 THEN 'A' -- 60%
           WHEN i % 10 < 9 THEN 'B' -- 30%
           ELSE 'C' -- 10%
           END                                AS category
FROM range(1_000_000_000) AS t(i);

SELECT COUNT(DISTINCT user_id)        AS exact_distinct,
       approx_count_distinct(user_id) AS approx_distinct,
       approx_quantile(user_id, 0.5)  AS p50_id,
       approx_quantile(user_id, 0.95) AS p95_id,
       approx_top_k(category, 2)      AS top2_categories
FROM big;

100% ▕████████████████████████████████████▏ (00:00:19.32 elapsed)
```

exact_distinct int64	approx_distinct int64	p50_id int32	p95_id int32	top2_categories varchar[]
632110399	650106546	500005515	950005930	[A, B]

Paradigm shift & limitations

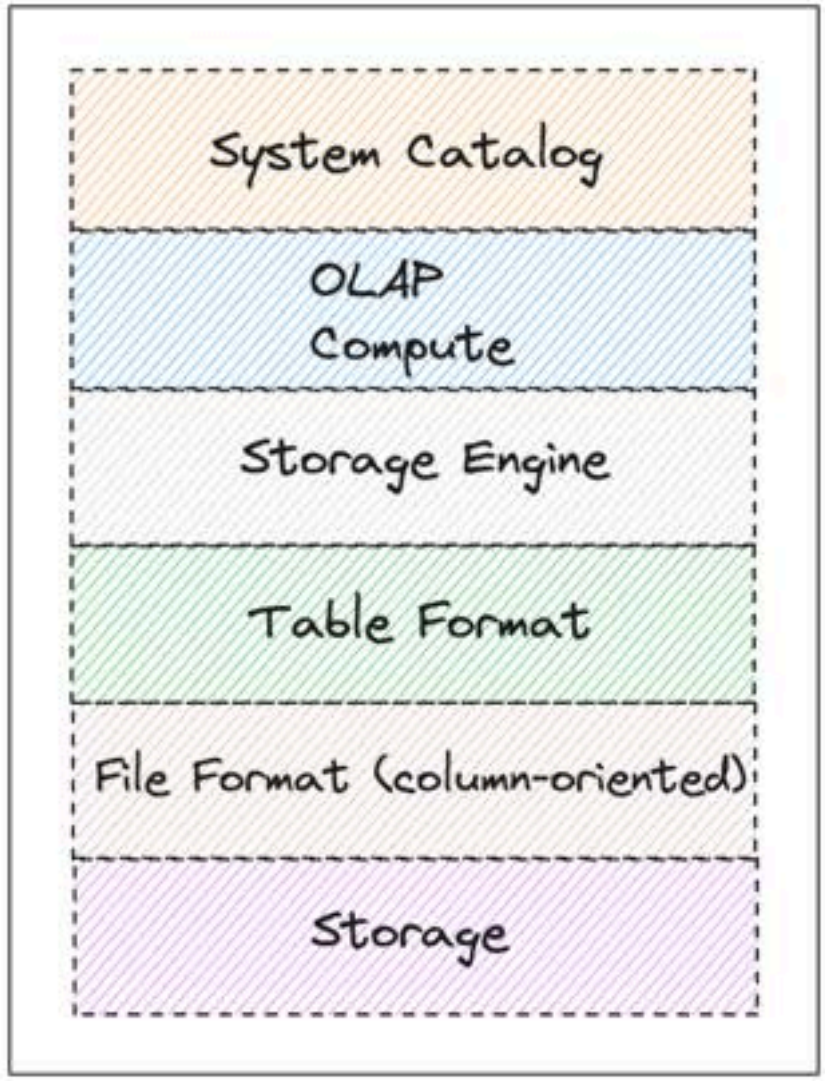
- In-process - No server: Warehouse per user
  - No user management
  - Access control via storage layer
  - No dynamic data masking
  - Easily hand a fully hydated database (with masked data) to 3rd parties
  - Possibility of AWS Lambda/Cloudflare Worker massive scale-out deployments
  - Not only a database - dynamic in-process transformation engine
- Arrow zero-copy integration with (data-science/AI) ecosystem + ADBC for fast BI
  - Bluesky near real time batched analytics of social media data
  - Arrow Flight (columnar streaming) -> first integration: Boilstream streaming ingestion
  - Columnar layout in memory & SIMD
- Latency reduction - client-side analytics (Rill data); even WASM support
- Single writer multiple readers (locking)

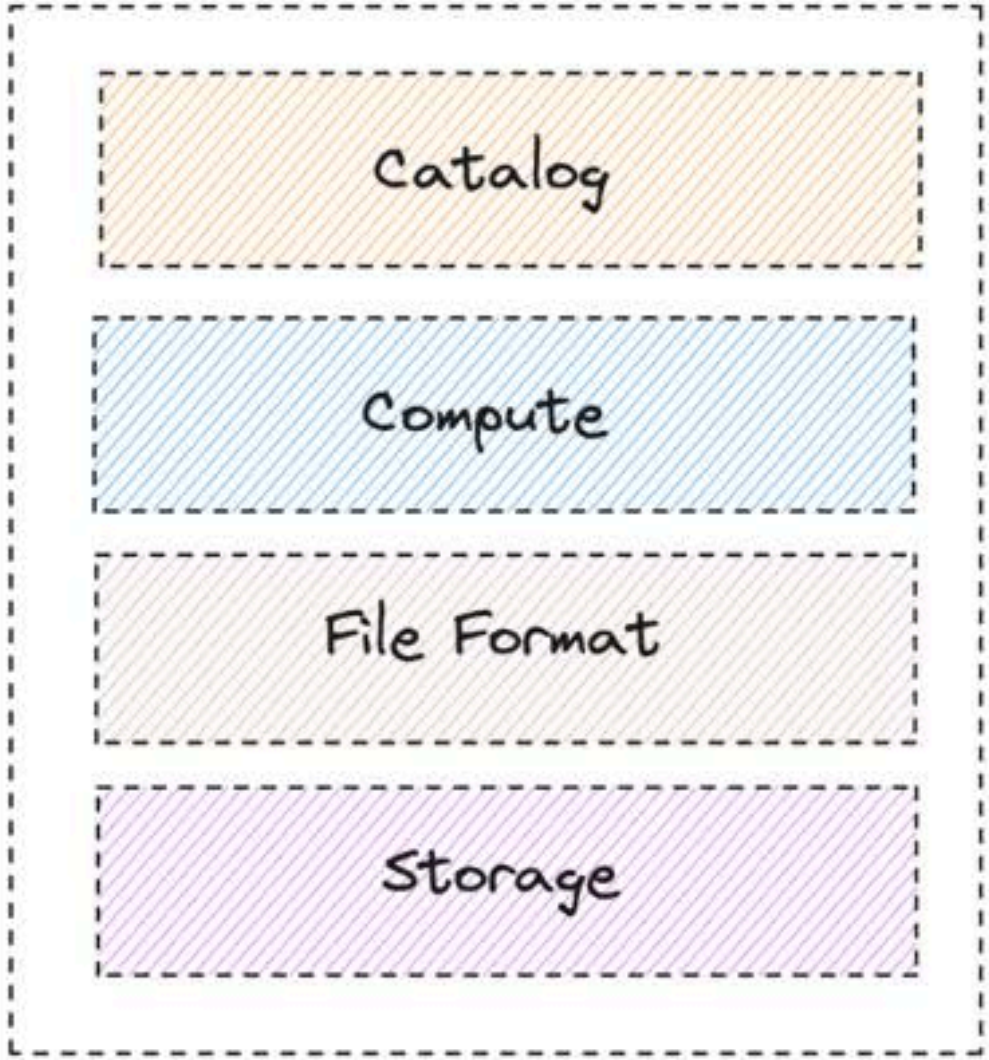

DuckDB
Sign in to MotherDuck
Help
Notebooks
DuckDB UI basics
Attached databases
memory
main
trains
Run
memory
/**
Exploring Column Diagnostics
When running a query, your results will appear below the editor
Run this query, then click on column names in the right panel to view detailed statistics.
*/
duckdb -ui
CALL start_ui();
CREATE TABLE stations AS FROM 's3://duckdb-blobs/stations.csv';
ATTACH '/path/to/my_master_data.duckdb' (READ_ONLY);
11,196,117 2023-05-15 Intercity 1,410 GV Den Haag HS 2023-05-15 00:29:00
11,196,117 2023-05-15 Intercity 1,410 LEDN Leiden Centraal 2023-05-15 00:45:00
11,196,117 2023-05-15 Intercity 1,410 SHL Schiphol Airport 2023-05-15 01:03:00
11,196,117 2023-05-15 Intercity 1,410 ASD Amsterdam Centraal 2023-05-15 01:19:00
11,196,117 2023-05-15 Intercity 1,410 UT Utrecht Centraal
11,196,118 2023-05-15 Nightjet 420 NURNB Nürnberg Hbf 2023-05-15 00:01:00
11,196,118 2023-05-15 Nightjet 420 FFS Frankfurt (Main) Süd 2023-05-15 01:47:00
11,196,118 2023-05-15 Nightjet 420 FFFM Frankfurt (M) Hbf
11,196,118 2023-05-15 Nightjet 420 FNAF Frankfurt Flughafen Fernb. 2023-05-15 01:59:00
11,196,118 2023-05-15 Nightjet 420 FMZ Mainz Hbf 2023-05-15 02:18:00
11,196,118 2023-05-15 Nightjet 420 KKD Koblenz Hbf 2023-05-15 03:13:00
11,196,118 2023-05-15 Nightjet 420 BONN Bonn Hbf 2023-05-15 04:01:00
11,196,118 2023-05-15 Nightjet 420 KKW Köln West
Filter
380,959 Rows
trains
service_id
date
type
train_number
station_code
station_name
departure_time
arrival_time
Run
memory
/**
Working with Files
The DuckDB UI can read and write files just like in other clients.
This example demonstrates exporting data to a CSV file and then
querying that same file.
Current Cell
380,959 Rows
8 Columns
Default
Sprinter
Snelbus i.p.v. trein
Thalys
ICE International
Other
train_number
station_code
station_name
departure_time
arrival_time
40k
20k
0
Mon 15
Wed 17
Fri 19
May 21
about 7 days
bucketed by day
earliest
latest
low bucket
high bucket

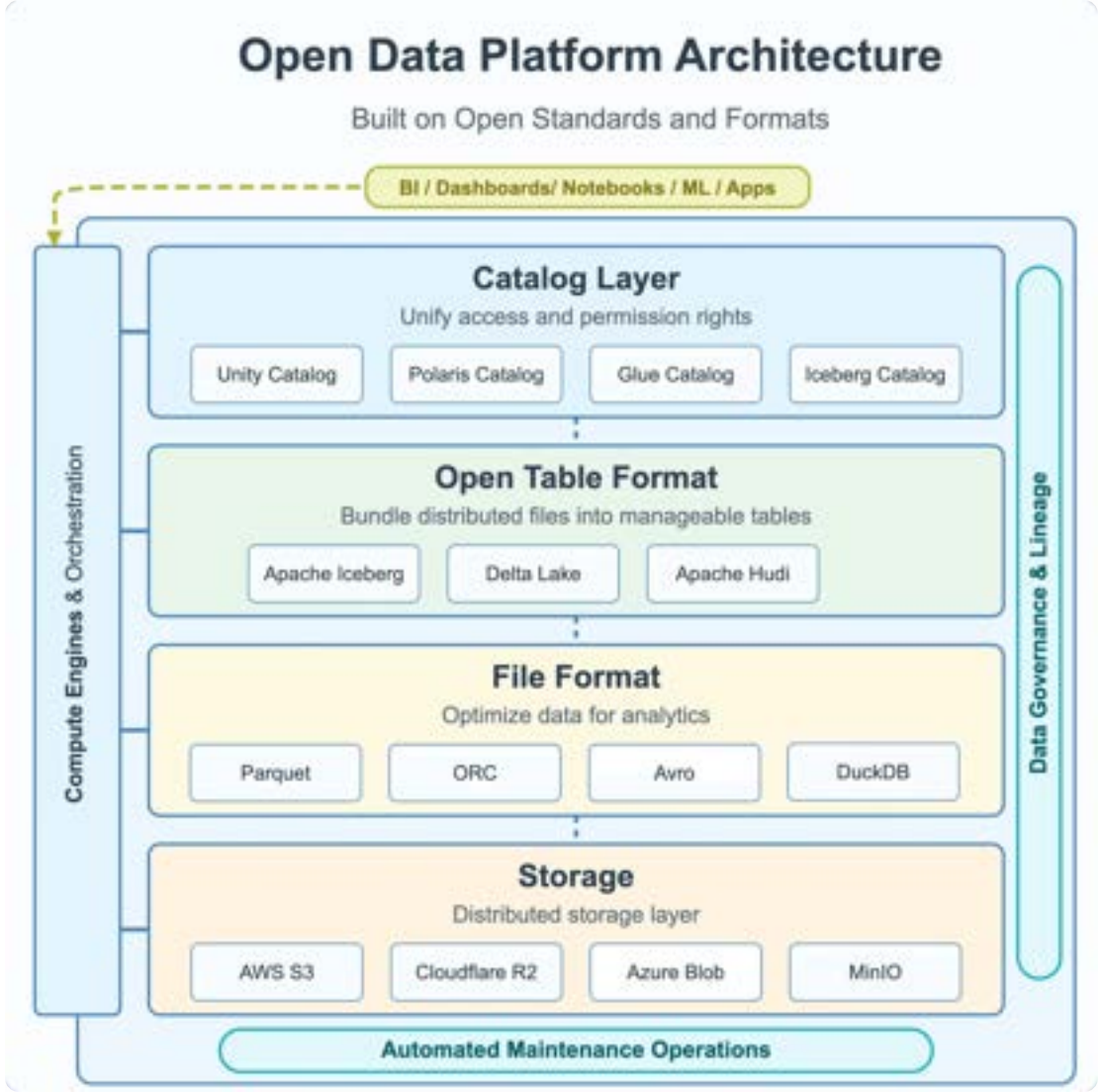
pg_duckdb

- Remember: powerful in-process transformation engine but limitation of single writer
- Combine the strengths of postgres (proven, IAM) as a serving layer with the power of duckdb

```
-- remember DuckDB solo
-- CREATE TABLE stations AS FROM 's3://duckdb-blobs/s

-- combination
CREATE TABLE pg_stations AS
select *
from duckdb.query('SELECT * FROM ''s3://duckdb-blobs/

-- postgres
select *
from pg_stations
limit 10;
```


System Catalog
OLAP
Compute
Storage Engine
Table Format
File Format (column-oriented)
Storage


Catalog
Compute
File Format
Storage


Open Data Platform Architecture
Built on Open Standards and Formats
BI / Dashboards/ Notebooks / ML / Apps
Compute Engines & Orchestration
Catalog Layer
Unify access and permission rights
Unity Catalog
Polaris Catalog
Glue Catalog
Iceberg Catalog
Open Table Format
Bundle distributed files into manageable tables
Apache Iceberg
Delta Lake
Apache Hudi
File Format
Optimize data for analytics
Parquet
ORC
Avro
DuckDB
Storage
Distributed storage layer
AWS S3
Cloudflare R2
Azure Blob
MinIO
Data Governance & Lineage
Automated Maintenance Operations

Iceberg catalog

- S3 until recent - no strong consistency
- Read-after-write consistency was missing
- Impact: Add layer to handle transactions

> TLDR: Add a database in disguise

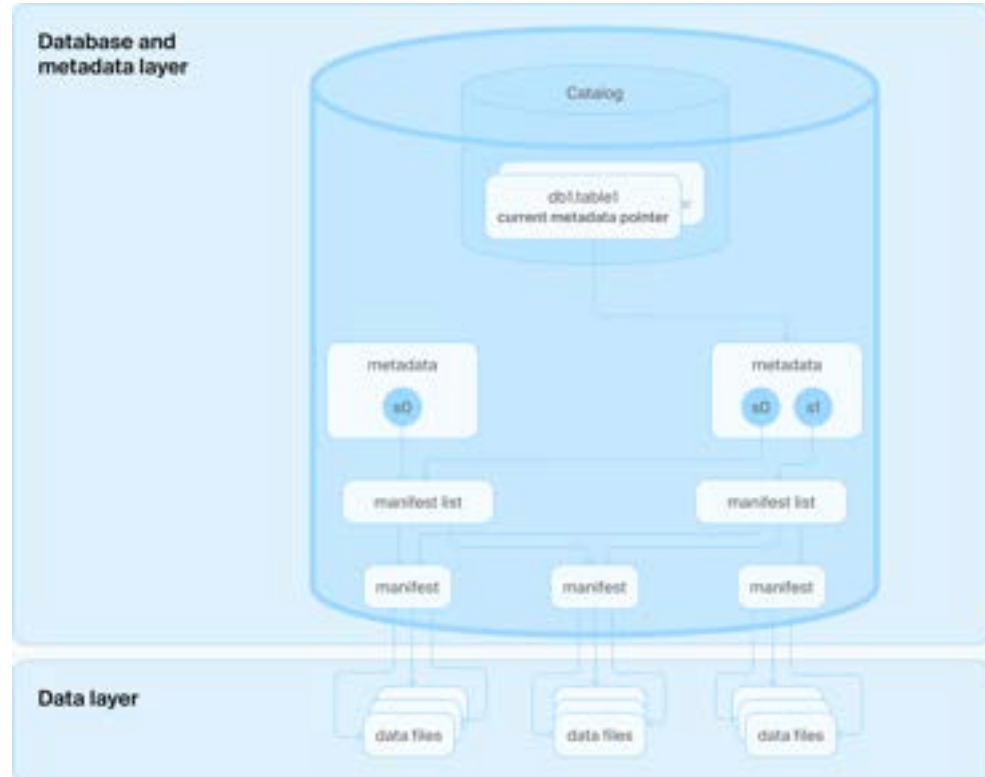
Less features + custom API; no SQL.

- No global transaction.
- No neat SQL filters.
- Lots of **complexity**

Problem: Table format was specifically designed to not require a database

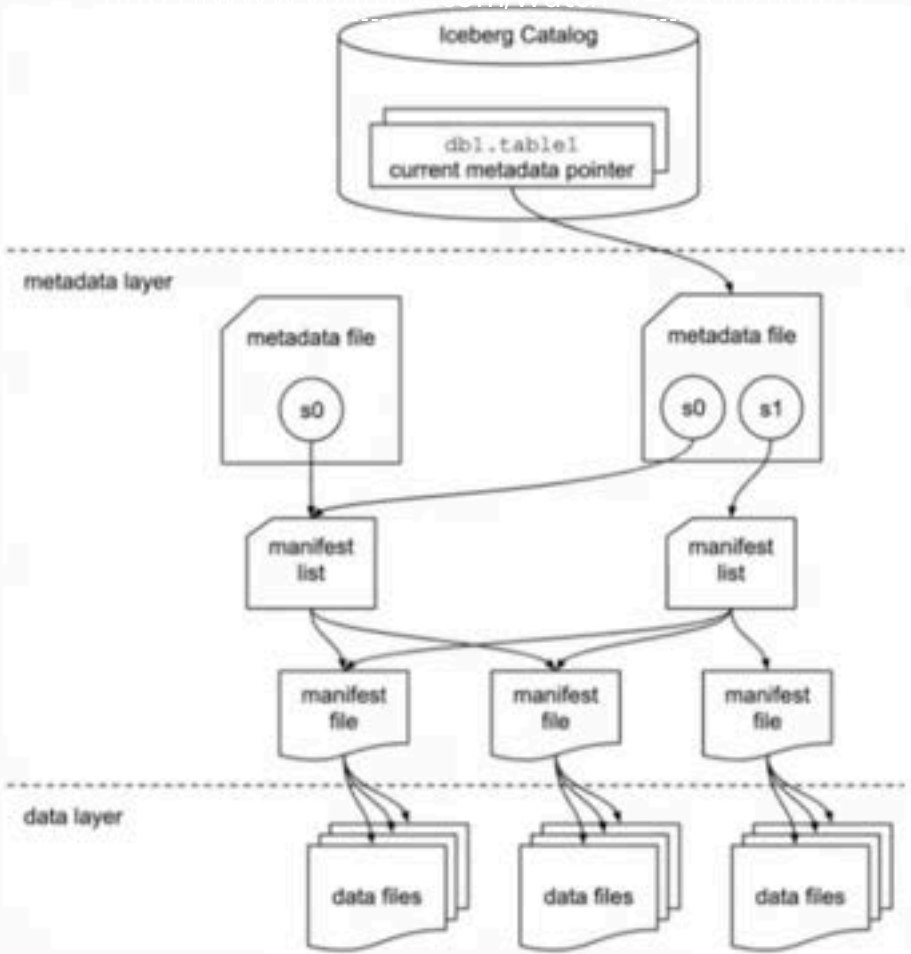
Ducklake

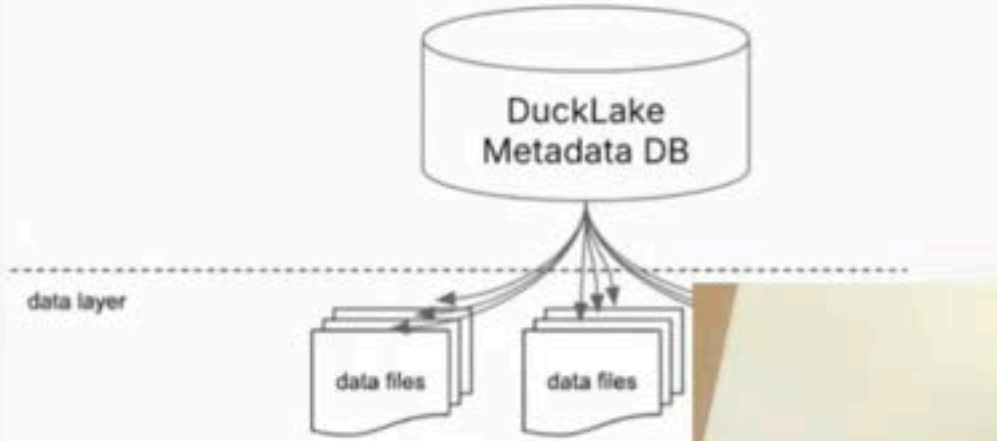
- Database and SQL first design
- Open standard
- Flexible backends
  - Catalog (postgres, sqlite, duckdb, mysql)
  - Storage (local fs, s3 compatible, ...)
- Real ACID transactions (not just for a single table)
- Reduce complexity (increase performance - inlining)
- Bridge gap to table formats
- Simple encryption of data



> Most cloud data warehouses internally have a similar architecture!

Replacing a whole lot of stuff with a database





Ducklake basics

```
duckdb
```

Ducklake basics

```
INSTALL ducklake;
```

Ducklake basics

```
ATTACH 'ducklake:metadata.ducklake' AS my_ducklake;
CREATE TABLE my_ducklake.demo
(
    i INTEGER
);
INSERT INTO my_ducklake.demo
VALUES (42),
       (43);
FROM my_ducklake.demo;

┌─────────┐
│    i    │
│  int32  │
├─────────┤
│      42 │
│      43 │
└─────────┘

```

Ducklake basics

```
DELETE
FROM my_ducklake.demo
WHERE i = 43;
FROM my_ducklake.demo;

```

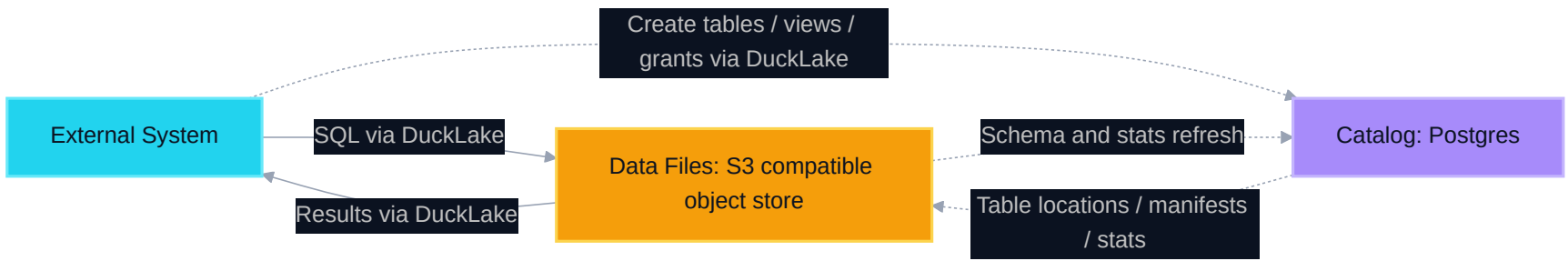
i
int32
42

Ducklake basics

```
BEGIN TRANSACTION;
DELETE
FROM my_ducklake.demo;

FROM my_ducklake.demo;
┌─────────┐
│    i    │
│  int32  │
├─────────┤
│ 0 rows  │
└─────────┘

ROLLBACK;
FROM my_ducklake.demo;
┌─────────┐
│    i    │
│  int32  │
├─────────┤
│   42    │
└─────────┘

```

Ducklake: Example architecture

- Catalog: Postgres
- Storage: S3 compatible (MinIO) for parquet files
- All locally in docker with SSL + DNS



Ducklake demo

```
-- setup ducklake
INSTALL ducklake;
INSTALL postgres;
INSTALL httpfs;
LOAD httpfs;
```

Ducklake demo

```
-- define secrets
CREATE OR REPLACE SECRET duckpond_secret_catalog (
    TYPE postgres,
    HOST 'db.duckpond.data-engineering.expert',
    PORT 5433,
    DATABASE ducklake_catalog,
    USER getenv('CATALOG_DB_USER'),
    PASSWORD getenv('CATALOG_DB_PASSWORD')
);

CREATE OR REPLACE SECRET duckpond_secret_storage (
  TYPE S3,
  KEY_ID     getenv('OBJECT_STORE_ROOT_USER'),
  SECRET     getenv('OBJECT_STORE_ROOT_PASSWORD'),
  ENDPOINT   'objectstore.duckpond.data-engineering.expert',
  URL_STYLE  'path',
  REGION getenv('OBJECT_STORE_REGION'),
  USE_SSL    true,
  SCOPE      's3://my-duck-lake'
);
```

Ducklake demo

```
-- conect
CREATE OR REPLACE SECRET duckpond_ducklake (
    TYPE DUCKLAKE,
    METADATA_PATH '',
    DATA_PATH 's3://my-duck-lake/',
    METADATA_PARAMETERS MAP {'TYPE': 'postgres', 'SECRET': 'duckpond_secret_catalog'}
);
ATTACH 'ducklake:duckpond_ducklake' AS duckpond;

USE duckpond;

-- inlining (PG not supported yet)
-- https://github.com/duckdb/ducklake/issues/293
-- ATTACH 'ducklake:my_ducklake.duckdb' AS local_lake (DATA_INLINING_ROW_LIMIT 10);
```

Ducklake demo

```
CREATE TABLE nl_train_stations AS FROM 'https://blobs.duckdb.org/nl_stations.csv';

SELECT * FROM (SHOW ALL TABLES) WHERE  schema == 'main';
```

database	schema	name	column_names	column_types
varchar	varchar	varchar	varchar[]	varchar[]
duckpond	main	nl_train_stations	[id, code, uic, na…	[BIGINT, VARCHAR, BIGINT, VARCHAR, VARCHAR, VARCHAR, VARCHAR…

Ducklake demo

```
mc ls duckpond/my-duck-lake/main/nl_train_stations
# 1 file

[2025-09-29 19:53:52 CEST]  41KiB STANDARD ducklake-0199969c-0a24-7b90-a7fd-562c7a2e4d69.parquet
```

Ducklake demo

```
UPDATE nl_train_stations SET name_long='Johan Cruijff ArenA' WHERE code = 'ASB';
SELECT name_long FROM nl_train_stations WHERE code = 'ASB';

┌──────────────────┐
│    name_long     │
│     varchar      │
├──────────────────┤
│ Johan Cruijff ArenA │
└──────────────────┘
```

Ducklake demo

```
mc ls duckpond/my-duck-lake/main/nl_train_stations
# More files current, one delta, one deletion vector

[2025-09-29 19:53:52 CEST]  41KiB STANDARD ducklake-0199969c-0a24-7b90-a7fd-562c7a2e4d69.parquet
[2025-09-29 19:56:58 CEST] 1.8KiB STANDARD ducklake-0199969e-df9a-74ab-9f40-9542612bbc0e.parquet
[2025-09-29 19:56:58 CEST]   790B STANDARD ducklake-0199969e-dfb7-7267-b817-ff61d6edaba4-delete.parquet
```

Ducklake demo

```
FROM duckpond.snapshots();
```

snapshot_id int64	snapshot_time timestamp with time zone	schema_version int64	changes map(varchar, varchar[])
0	2025-08-31 21:16:09.133+02	0	{schemas_created=[main]}
1	2025-08-31 21:21:39.603+02	1	{tables_created=[main.nl_train_stations], tables_inserted_into=[1
2	2025-08-31 21:23:16.028+02	1	{tables_inserted_into=[1], tables_deleted_from=[1]}

```
--
```

Ducklake demo

```
SELECT name_long FROM nl_train_stations AT (VERSION ⇒ 1) WHERE code = 'ASB';
```

name_long
varchar
Amsterdam Bijlmer ArenA

```
SELECT name_long FROM nl_train_stations AT (VERSION ⇒ 2) WHERE code = 'ASB';
```

name_long
varchar
Johan Cruijff ArenA

Ducklake demo

```
-- inlining into RDBMS benefits from RDBMS - no small files problem for small data
-- currently does not work with Postgres catalog
CREATE TABLE duckpond.tbl(col INTEGER);
-- inserting 3 rows, data is inlined
INSERT INTO duckpond.tbl VALUES (1), (2), (3);
```

Ducklake demo

```
mc ls duckpond/my-duck-lake/main/
# No new table created! No parquet file! all in RDBMS!
```

Ducklake demo

```
INSERT INTO duckpond.tbl FROM range(100);
```

Ducklake demo

```
mc ls duckpond/my-duck-lake/main/tbl
# Now data resides in parquet files
```

Ducklake demo

```sql
-- clean up small files
CALL duckpond.merge_adjacent_files();

-- delete old snapshots (a) first expire (b) perform deletion
CALL ducklake_expire_snapshots('duckpond', versions => [1]);
UPDATE nl_train_stations SET name_long='Johan Cruijff ArenAxx' WHERE code = 'ASB';
CALL ducklake_expire_snapshots('duckpond', versions => [2]);
FROM duckpond.snapshots();
CALL ducklake_cleanup_old_files('duckpond', cleanup_all => true, dry_run => true);
CALL ducklake_cleanup_old_files('duckpond', cleanup_all => true);

-- CALL duckpond.ducklake_expire_snapshots(older_than => now() - INTERVAL '7 days');

FROM ducklake_list_files('duckpond', 'nl_train_stations');
```

SQL automation

SQL focused orchestration (we touch on real orchestration later)

- Bring software engineering practices to SQL
- Reusable, tested, documented, modular SQL

dbt (bought by fivetran)

- Simple, built around jinja templates
- New dbt-fusion with advanced features in the making

```
{% set payment_methods = ["a", "b"] %}
select
  order_id,
  {% for payment_method in payment_methods %}
  sum(case when payment_method = '{{payment_method}}'
    then amount end),
  {% endfor %}
  sum(amount) as total_amount
from app_data.payments
group by 1
```

sqlmesh (bought by fivetran)

- Extends dbt-core functionality
- Operates on AST (semantic validation)
- Stateful execution (backfills, branch deltas)
- Engine-agnostic SQL transpilation (sqlglot)
- Column-level lineage
- Multi-repository support
- Less need for a full orchestration system
- Multi-engine mode
- New ecosystem

Automating Ducklake with dbt

- Set secrets + catalog & storage connection details
- Attach the desired ducklake catalog(s)
- TLDR: fairly straight forward

profiles.yaml

```yaml
dev:
  type: duckdb
  extensions:
    - ducklake
  path: 'ducklake:local_dev.db'
  threads: 4
  attach:
    - path: "ducklake:my_ducklake.ducklake"
      alias: minidemo
      options:
        data_path: "./my_data_dir/minidemo"
```

dbt-project.yaml

```yaml
dbt_ducklake_demo:
  minidemo:
    +database: minidemo
    +materialized: view
    example_a:
      +schema: example_a
```

dbt model versioning (detail)

https://docs.getdbt.com/docs/mesh/govern/model-versions

Models: `my_model_v1.sql`

schema.yaml

```yaml
models:
  - name: dim_customers
    latest_version: 1
    ...
    versions:
      - v: 1
        deprecation_date: 2025-10-01
      - v: 2
        # Removed a column -- this is the breaking change!
        columns:
          # This means: use the 'columns' list from above, but exclude country_name
          - include: all
            exclude: [country_name]
```

https://github.com/dbt-labs/dbt-core/issues/7442 For versioned models, automatically create view/clone of latest version in unsuffixed database location

Automating Ducklake with sqlmesh

- Could only get the minimal variant: Ducklake catalog in duckdb to work - postgres failed with db/schema not found
- TLS 1.3 challenges (had to disable SSL) for state connection
- TLDR: Simple things (also S3) and switching state backend work smoothly; catalog change to pg failed

config.yaml

```yaml
devlocal:
  connection:
    type: duckdb
    catalogs:
      my_lakehouse:
        type: ducklake
        path: data/catalog.ducklake
        data_path: data/storage/
    extensions:
      - ducklake
  state_connection:
    type: duckdb
    database: 'data/sqlmesh_state.db'
```

Ingesting data with dlt

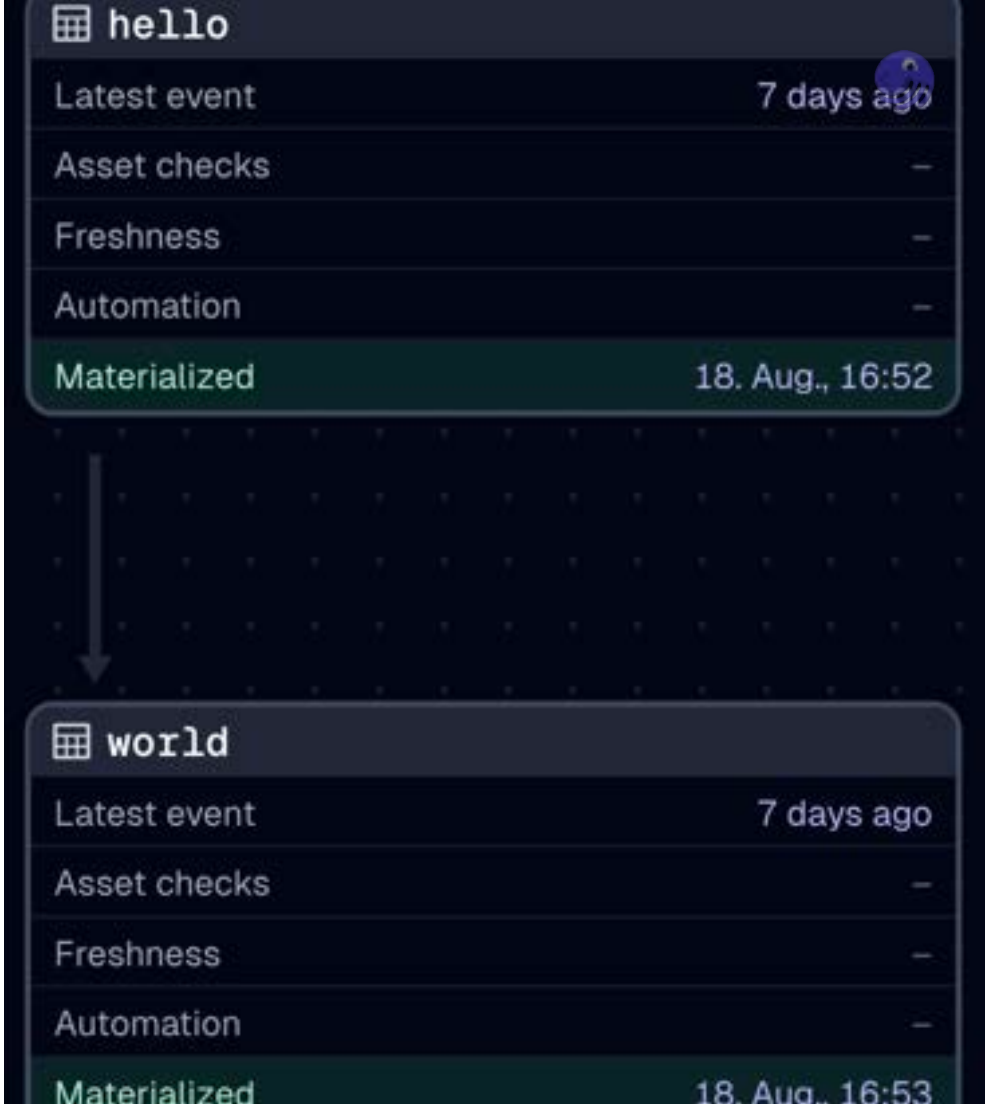
- Many ready-made connectors
  - REST API (including pagination)
  - SaaS (google analytics, stripe, salesforce, hubspot, shopify, zendesk, github, ...)
  - Databases
  - File systems
- Somewhat similar to Fivetran - but free (remember: recent price changes)
- Simple means of loading (update, append, replace, incremental loading)
- LLM integration (MCP server)
- FYI: sling is a similar solution

```
pokemon_source = rest_api_source(
    {
        "client": {
            "base_url": "https://pokeapi.co/api/v2/",
            "paginator": "json_link",
        },
        "resource_defaults": {
            "endpoint": {
                "params": {
                    "limit": 1000,
                },
            },
            "write_disposition": "replace",
        },
        "resources": [
            {
                "name": "pokemon",
                "primary_key": "name",
                "write_disposition": "merge",
            },
            "berry",
            "location",
        ],
    }
)
```

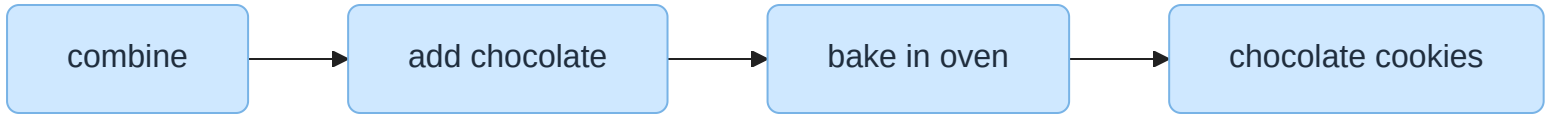
Dagster & asset graph

- Like a calculator for crunching numbers
- Graph allows computer to reason about data dependencies
- Rapid iteration: Just edit code. No need to wait for XYZ SaaS service
- Break down tool and department silos
- Ducklake integration `dagster-ducklake`

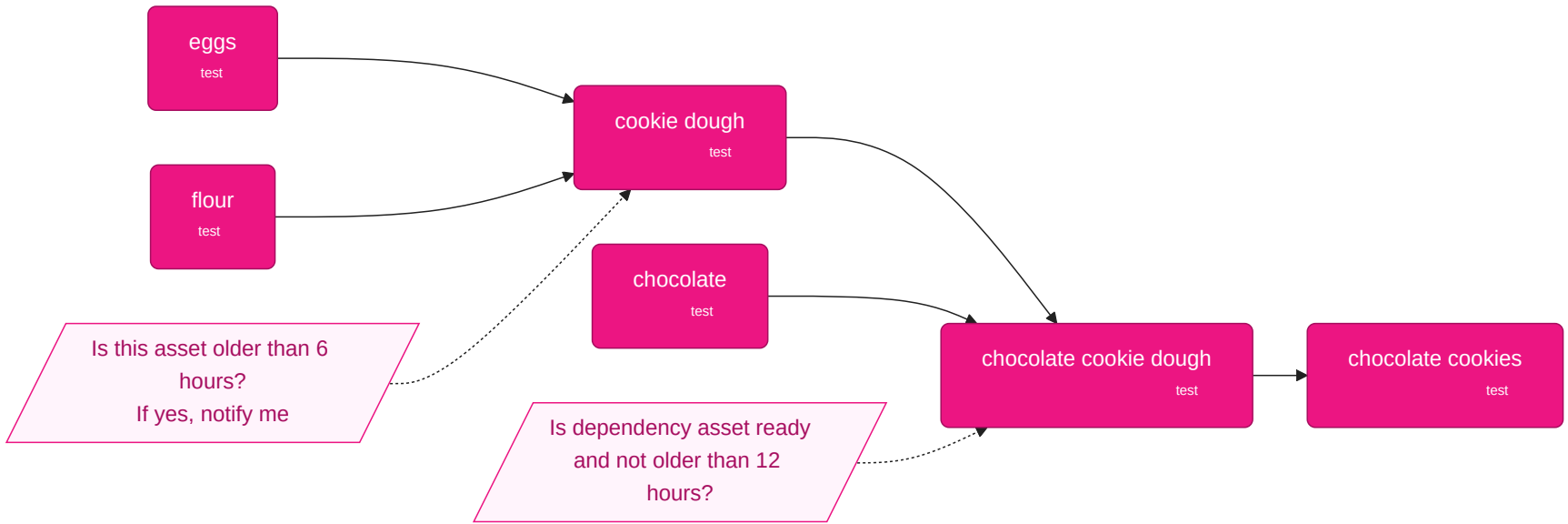
```python
import dagster as dg

@dg.asset
def hello(context: dg.AssetExecutionContext):
    context.log.info("Hello!")

@dg.asset(deps=[hello])
def world(context: dg.AssetExecutionContext):
    context.log.info("World!")
```



Task-based orchestrator



Asset-based orchestration



Dependency handling

- Conda support is a must
- Conda itself is slow and dated (even with mamba)

`pixi` as the solution

- Fast
- Lockfiles
- Multi-environment handling (dev, prod)
- Pip and conda support
- Neatly packaging of environments
- Easy bootstrap
- Task runner

Why a data catalog?

- Business overview, glossary, tags; Approval workflows; Data contracts
- Lineage, sample data, test results; anomaly detection
- Some OSS vendors below; there are more commercial offerings; choose wisely

Open Metadta

- Easy deployment, UI + API
- Metadata must be pushed, others read API
- Fully redacted search results based on RBAC
- Neat direct sample data (with PII redaction)
- MCP & ontology in OSS

Datahub

- Needs kafka, Neat CLI (GIT integration)
- Stream metadata changes (push and pull), Event-driven integration
- Direct dbt import (no catalog scan)
- Workflows and UI filters only in paid version

Process automation: Small Language Models

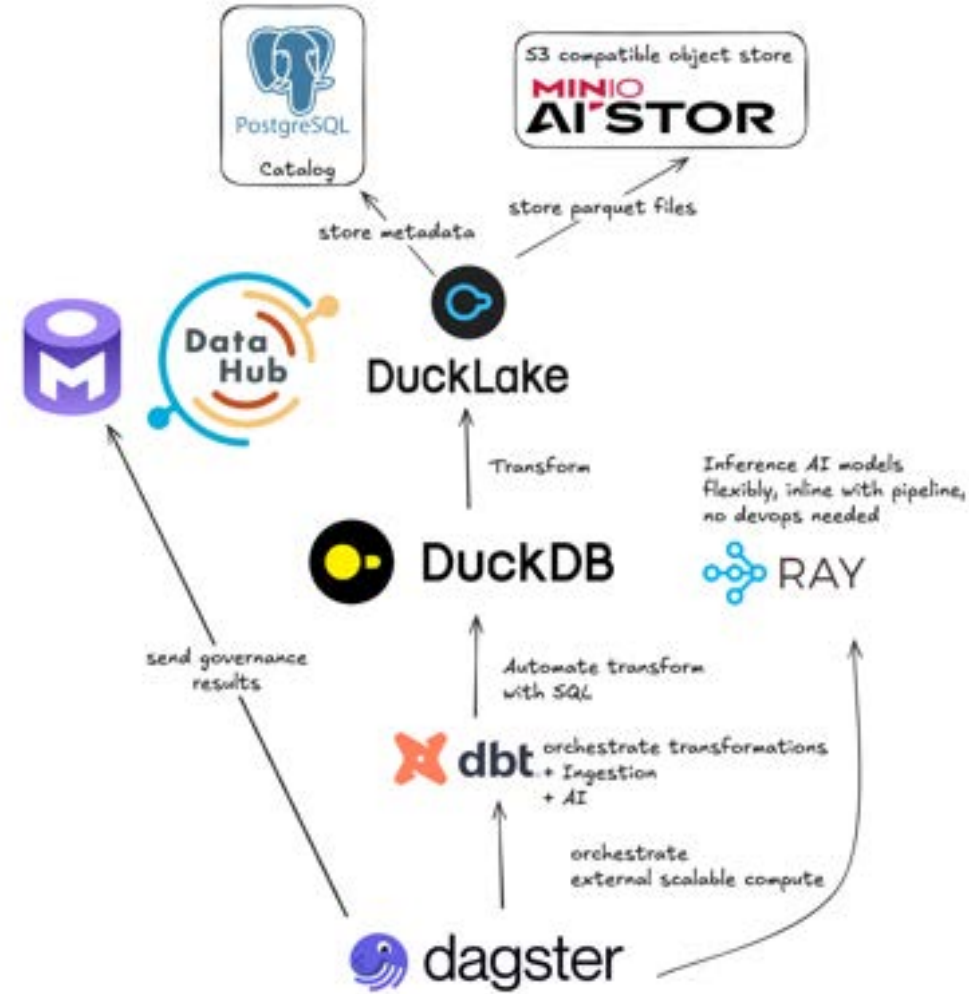
- No Human to chat and correct
- Cost and energy efficient
- Minimal latency
- Many small AI enabled steps with context engineering for repeatable reliable results

Ray

- Easily scaling arbitrary computation graphs
- Minimal devops support needed
- Seamless scaling: From laptop to cluster, no code rewrite.
- Elastic & cost-efficient: Scale up/down with demand, minimize idle GPUs.
- Batch and API serving support

Combined showcase

- Ducklake (postgres catalog, MinIO as object storage; fixing the rug-pull with open-maxio-ui)
- Transformation engine: DuckDB
- SQL automation - (dbt)
- Orchestration: dagster
- Dependency management: pixi
- Data catalog: open metadata
- Ingestion:
  - custom python scripts
  - dlt for reusable connectors and a smooth dagster integration
- Sec-Ops (sops, age)
- External compute: ray



Takeaway

- Ducklake is simple, powerful and flexible
- Ecosystem maturity (integrations) needs more time
- Asset based orchestration solves challenges:
  - Inside of datavault for more speed
  - Outside/around DV for AI/ML and ingestion
  - Not stuck on a single execution engine
  - Encapsulate complexity: Easy vibe-coding or onboarding of less technical users
  - Advanced: Environment-re-targeting possible to save money

Local-first data ecosystem is possible, and has advantages

Closing summary

Take back control: Reduce entropy

Systems should scale as needed:

- Down to a single developer`s notebook
- As big as necessary, but with simplicity

You should be in control

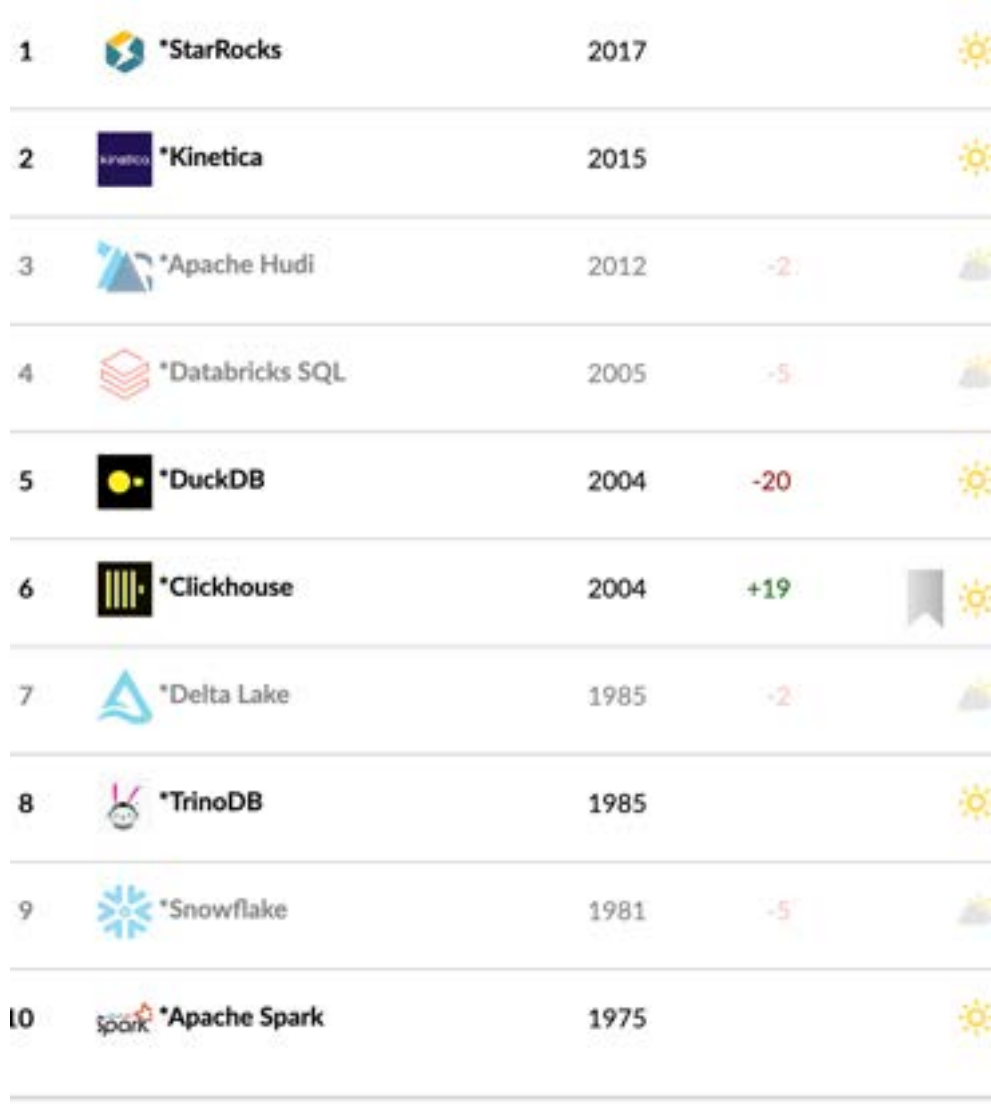
geoheil.com

Relative comparison

of analytical engines

- ELO score: Relative not absolute
- Robustness: Continously improve ranking, robust to cherry picking

See more at: https://georgheiler.com/post/elo-data-challenge/



Security

- Sops (multiple backends supported: age, various key stores)
- Age (pgp like without the mess)

Simple: `.env` file with secrets

- Encrypt
- Push to git - or your kms store of choice