Revisiting Analytical Storage with DuckLake

A simpler take on the lakehouse

May 7, 2026

Table of contents

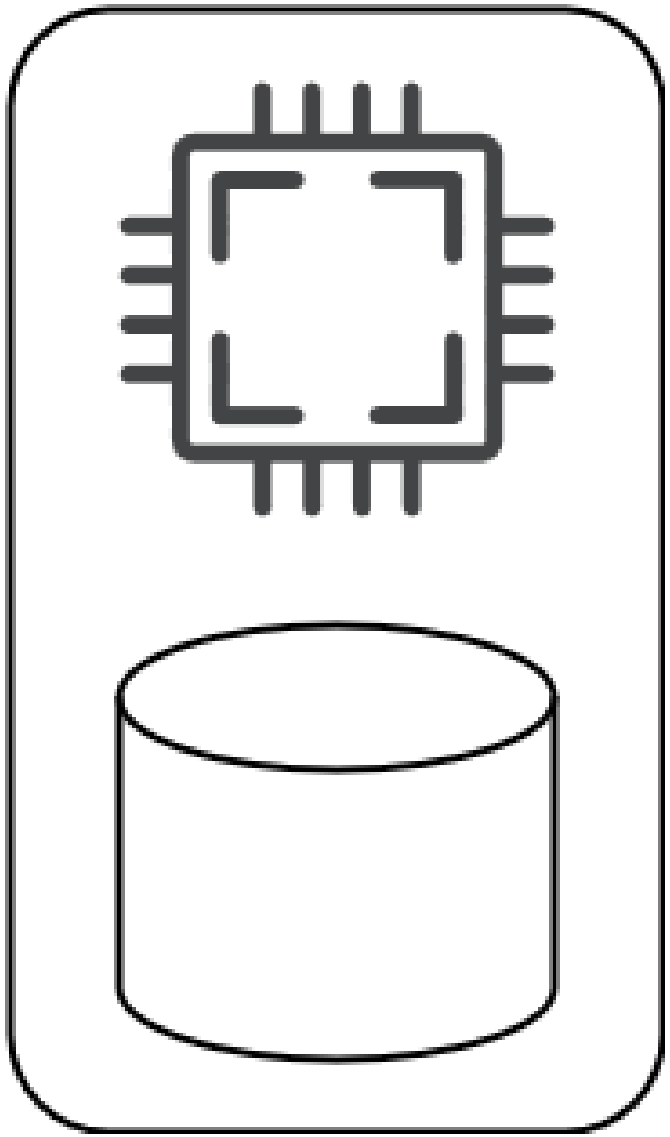
The historical evolution of analytical storage

From database → warehouse → lake → lakehouse

The transactional database (OLTP)

Optimized for transactions,

not analytics



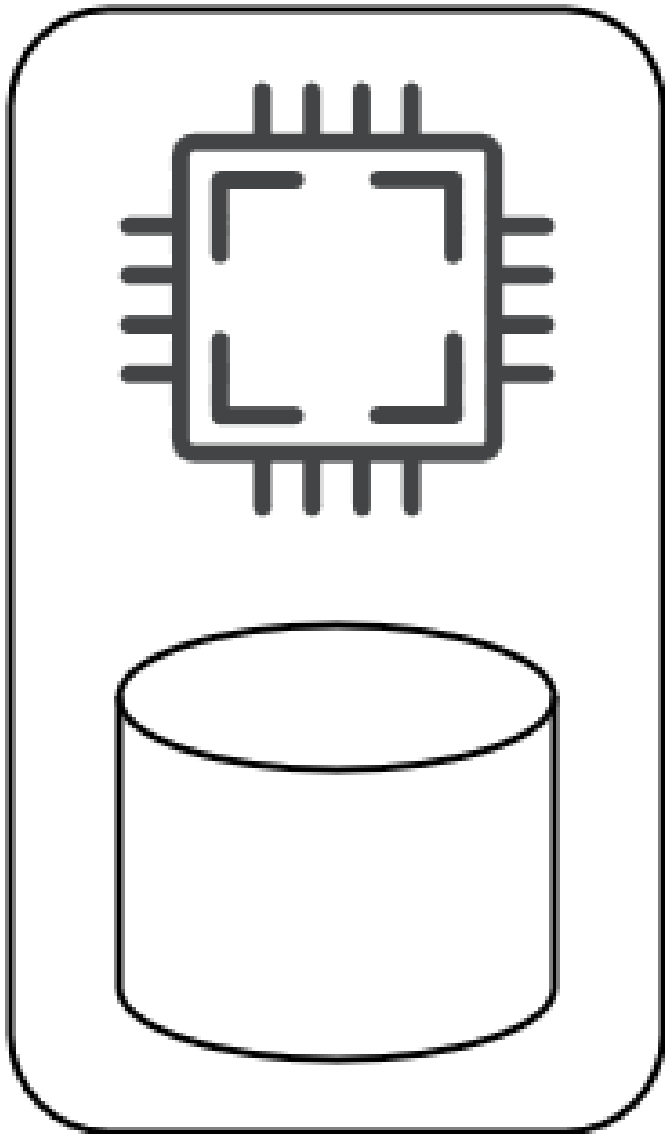
Transaction processing vs analytics

Property	Transaction processing systems (OLTP)	Analytic systems (OLAP)
Main read pattern	Small number of records per query, fetched by key	Aggregate over large number of records
Main write pattern	Random-access, low-latency writes from user input	Bulk import (ETL) or event stream
Primarily used by	End user/customer, via web application	Internal analyst, for decision support
What data represents	Latest state of data (current point in time)	History of events that happened over time
Dataset size	Gigabytes to terabytes	Terabytes to petabytes

M. Kleppmann, *Designing Data-Intensive Applications: The big ideas behind reliable, scalable, and Maintainable systems*. 2017. [Online].

The data warehouse (OLAP)

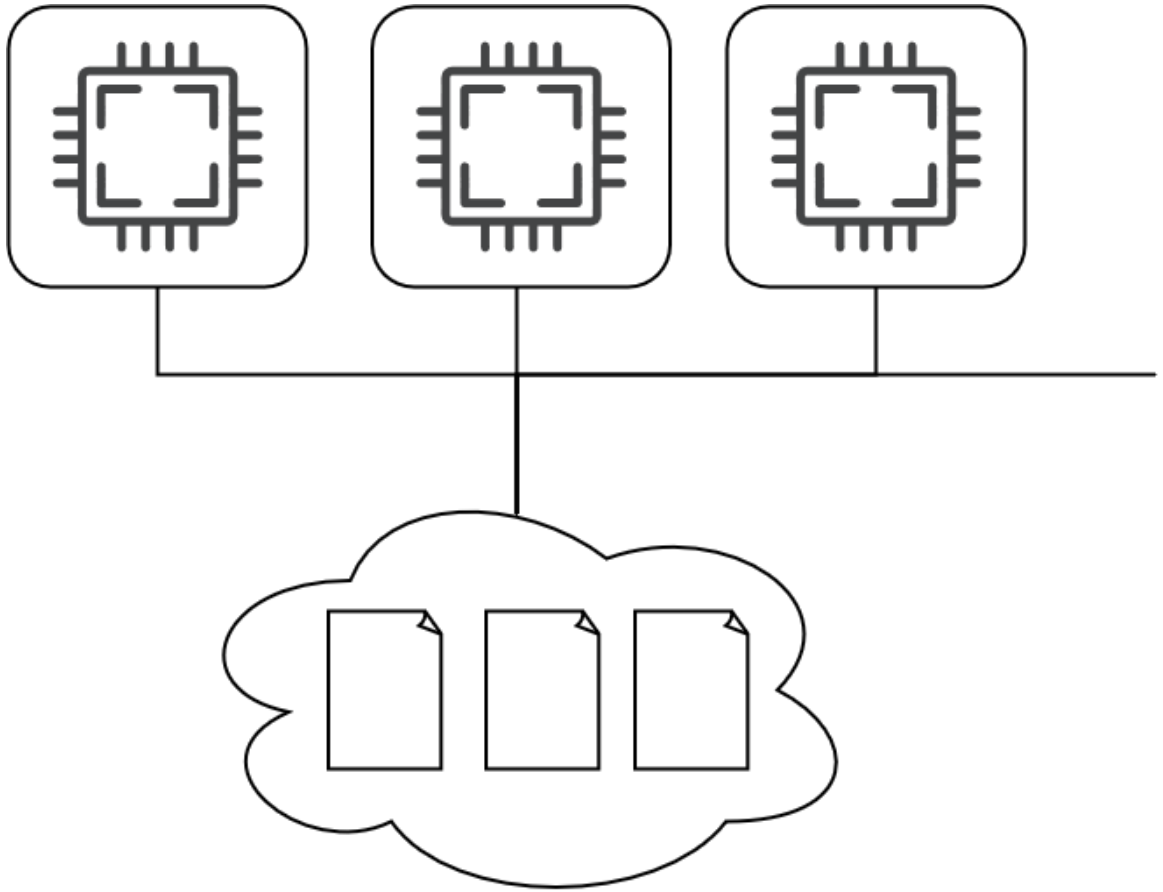
Storage & compute combined



The data lake

Weak governance => can become a "data swamp"

No transactional guarantees

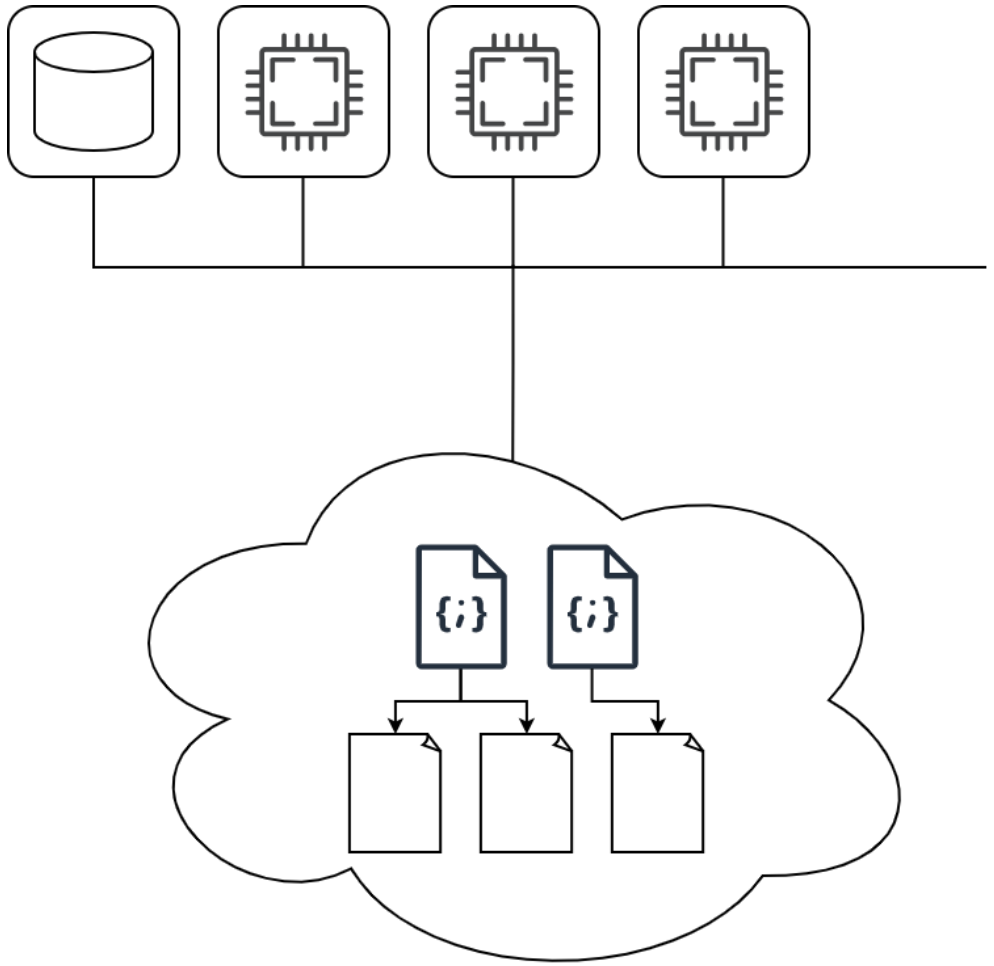


The data lakehouse

Example:

- Iceberg (+ Polaris)
- Delta Lake (+ Unity)
- DuckLake

Bring order to the datalake using metadata

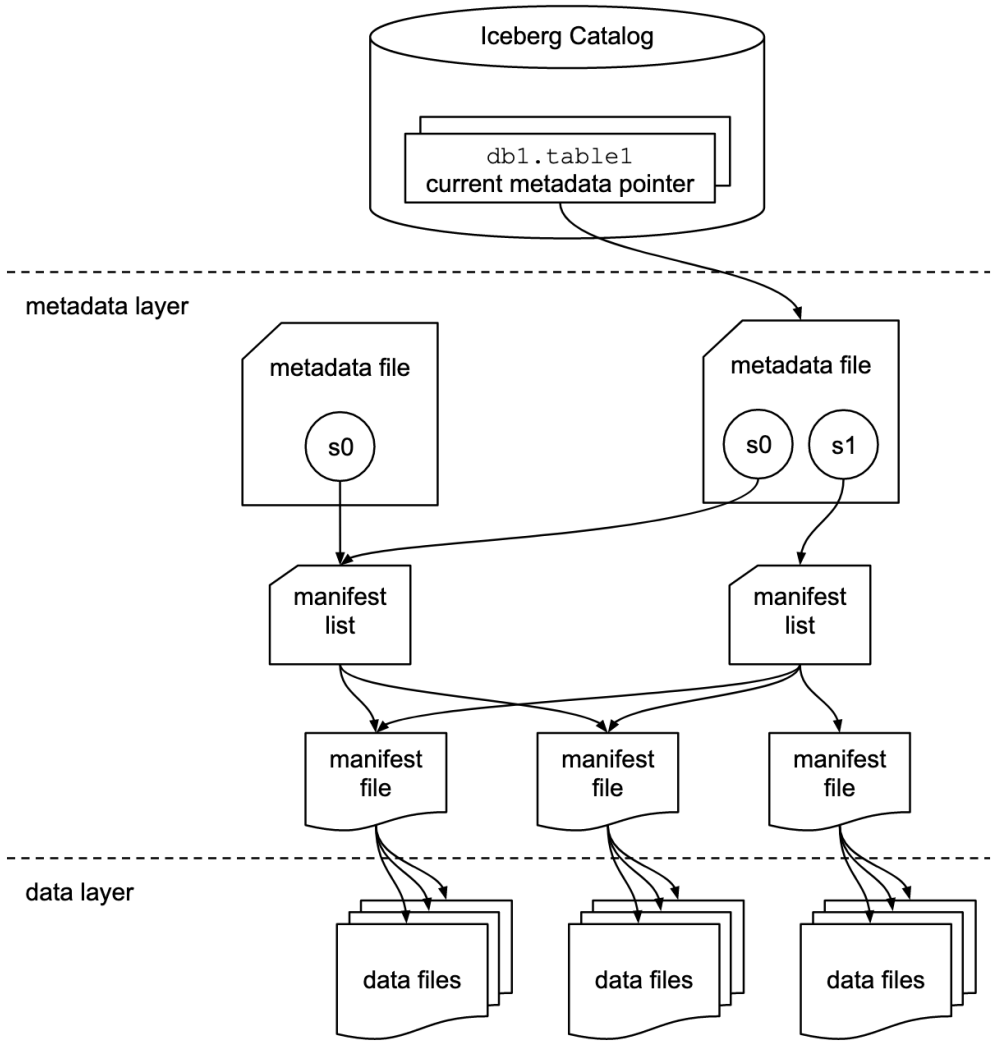


DuckLake: Rethinking the Lakehouse Architecture

Metadata as a database problem

The Iceberg / Delta model

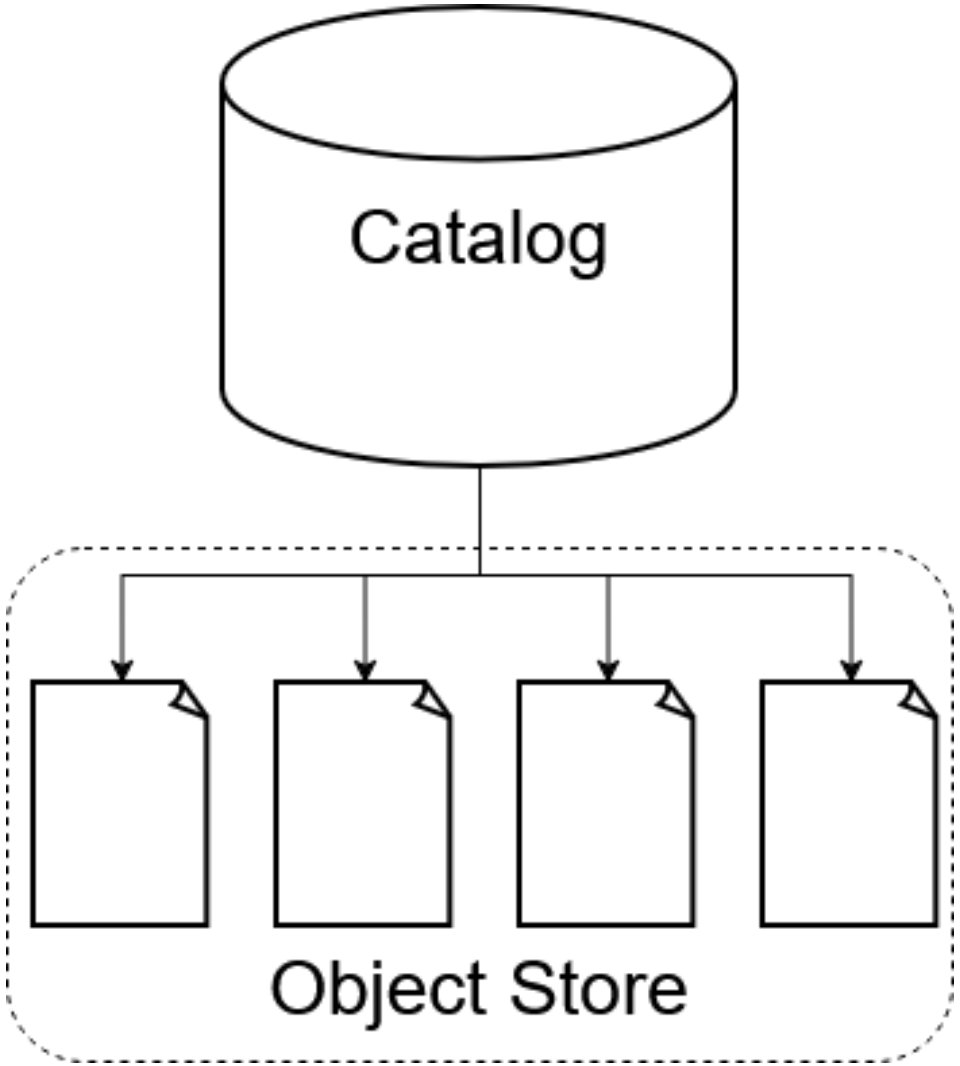
- Requires read / write of many small metadata files
- Increased overhead in query planning
- Increased operational complexity



The DuckLake model

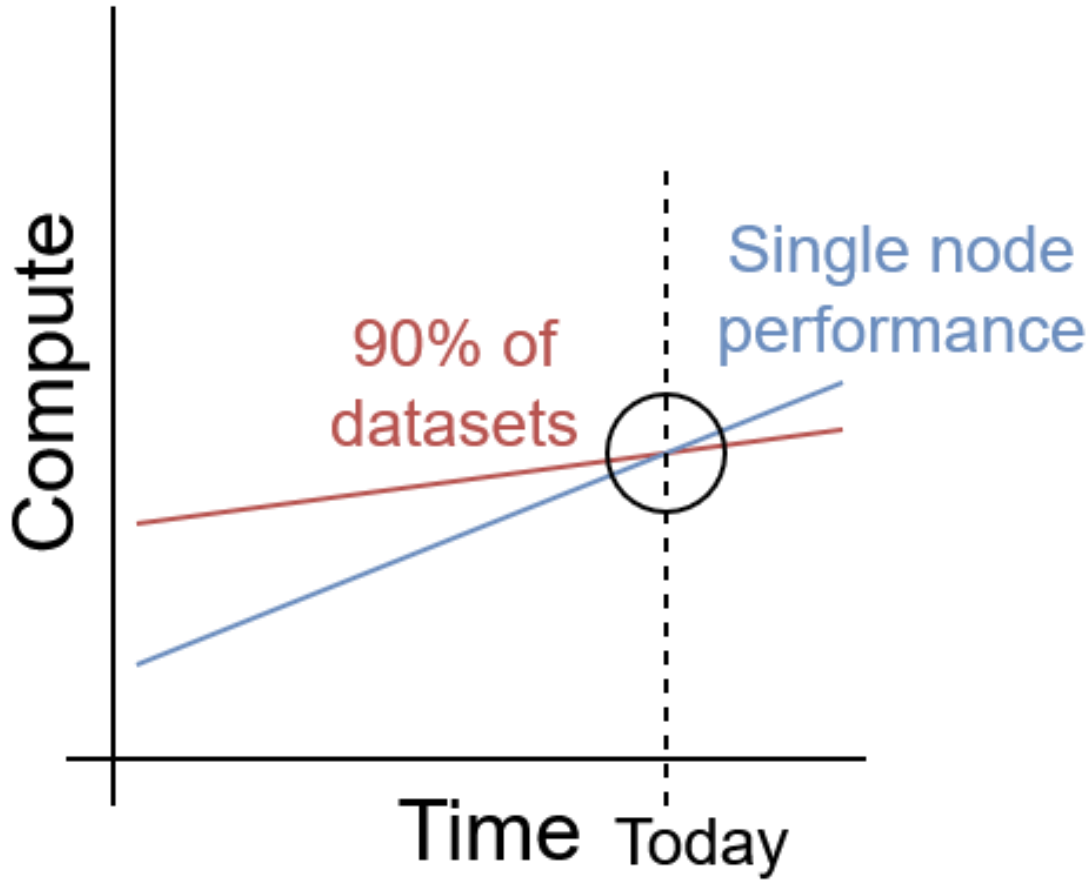
- Leverages catalog DB, which is already part of the stack
- Reduced operational complexity
- Increased performance
- Like Snowflake / BigQuery

No metadata files, only a catalog database



Distributed vs single-node compute

VS



DuckLake in practice

Ecosystem integration and the early-adopter trade-off

Easy setup, local & cloud

```
INSTALL ducklake;
INSTALL postgres;

CREATE SECRET pg_secret(
  TYPE 'postgres',
  HOST 'localhost',
  DATABASE 'ducklake_catalog',
  USER 'ducklake_user',
  PASSWORD **********
);

CREATE SECRET (
  TYPE 'ducklake',
  METADATA_PATH '',
  DATA_PATH '/path/to/data/dir',
  METADATA_PARAMETERS MAP {'TYPE': 'postgres','SECRET':
'pg_secret'}
);

ATTACH 'ducklake:' AS my_ducklake;
USE my_ducklake;
```

```
INSTALL ducklake;
INSTALL postgres;
INSTALL aws;
INSTALL httpfs;

CREATE SECRET s3_secret (
  TYPE s3,
  PROVIDER credential_chain,
  CHAIN config,
  PROFILE development,
  REGION 'eu-central-1'
);

CREATE SECRET pg_secret(
  TYPE 'postgres',
  HOST 'remote-host.rds.amazonaws.com',
  DATABASE 'ducklake_catalog',
  USER 'ducklake_user',
  PASSWORD **********
);

CREATE SECRET (
  TYPE 'ducklake',
  METADATA_PATH '',
  DATA_PATH 's3://path/to/data/on/s3',
  METADATA_PARAMETERS MAP {'TYPE': 'postgres','SECRET':
'pg_secret'}
);

ATTACH 'ducklake:s3_secret' AS my_ducklake;
USE my_ducklake;
```

Ecosystem integration: dbt

Overall: **good**

Some rare minor workarounds required e.g.,

- SCD2 snapshots need dbt-duckdb macro edit + tmp schema

profiles.yml

```yaml
my_dbt_profile:
  target: dev
  outputs:
    local:
      type: duckdb
      extensions:
        - httpfs
        - aws
        - ducklake
        - postgres
      secrets:
        - type: postgres
          name: catalog_credentials
          host: "{{ env_var('DL_HOST')}}"
          port: "{{ env_var('DL_PORT')}}"
          user: "{{ env_var('DL_USER')}}"
          password: "{{ env_var('DL_PASSWORD') }}"
          database: "{{ env_var('DL_DATABASE') }}"
        - type: s3
          provider: credential_chain
          validation: "exists"
          refresh: auto
      attach:
        - path: "ducklake:postgres:"
          alias: catalog
          options:
            meta_secret: catalog_credentials
            data_path: "{{ env_var('DL_DATA_PATH') }}"
      database: catalog
      schema: main
      threads: 16
```

Ecosystem integration: dagster

Overall: **very good**

Some rare minor workarounds required e.g.,

- Subclassing of DuckDBResource and connection definition

profiles.yml

```python
import dagster as dg
from dagster_duckdb import DuckDBResource
from duckdb import DuckDBPyConnection, connect


class DuckLakeResource(DuckDBResource):
    """Get a python connection to a DuckLake."""

    @classmethod
    @contextmanager
    def get_connection(cls) -> Generator[DuckDBPyConnection]:
      ducklake = connect()
      ducklake.sql("""
        ATTACH 'ducklake:postgres:dbname=ducklake_catalog host=localhost' AS my_ducklake
        (DATA_PATH 'data_files/');
        USE my_ducklake;
      """)

      try:
          yield ducklake
      finally:
          ducklake.close()
```

Ducklake’s biggest problem

Early-adopter tax

Works where DuckDB works, but often not without adjustments

- DBT
- Dagster
- Metabase

Ducklake inconveniences

RBAC

- No DuckLake built-in abstraction
- Is delegated to catalog database & object store
- Fix planned for v2.0

Git-like branching

- Currently missing (planned for v2.0)

DuckLake

Conclusion

The current state of DuckLake

The current state of DuckLake

Promising project in its early stages

- FOSS
- “Multiplayer” DuckDB
- Simple Lakehouse
- For prod & quick local testing
- More hands-on than fully managed service

DuckLake

Thank you for listening