codecentric

duckdb



DuckDB is a
fast
in-process
analytical
open source
portable
database system

from https://duckdb.org/

We start with a demo

```
duckdb --ui
```

Analyse local

```
SELECT *
FROM   read_json("employees.json");


SELECT department,
       avg(salary)
FROM   read_json("employees.json")
GROUP  BY all;
```

Analyse remote

```
SELECT *
FROM   read_parquet("https://…/oil-and-gas.parquet");


SELECT "reporting year"          AS year,
       sum("gas produced, mcf") AS gas
FROM   read_parquet("https://…/oil-and-gas.parquet" )
GROUP  BY "reporting year";


SELECT *
FROM   read_json("https://api.spacexdata.com/v3/launches")
```

Analyse blob

```
SET s3_region='eu-central-1';
SET s3_access_key_id='AKIA2RYWYEMUJVK3C4X4';
SET s3_secret_access_key='';

SELECT * EXCLUDE payload
FROM read_json("s3://test-data-matthias/github/*.json.gz")

SELECT actor.login, actor.url FROM
read_json("s3://test-data-matthias/github/*.json.gz")
```

Friendly SQL: Add - Ons & Full PostgreSQL

```
SELECT * EXCLUDE payload FROM
table

SELECT department, avg(salary)
FROM table GROUP  BY all

FROM table
SELECT * EXCLUDE payload
```

all columns not in a aggregate function

duckdb databases

```
ATTACH 'persisent.db';

CREATE TABLE raw_data AS SELECT * FROM
read_json("employees.json")

ATTACH 's3://…/ducky_catalog.ddb';

SELECT * FROM customers LIMIT 10
```

duckdb data sharing

```
ATTACH 's3://…/ducky_catalog.ddb';

-- the file itself is just roughly 268KB
SELECT sql FROM duckdb_views() where temporary=false;

ATTACH 'md:'
CREATE DATABASE cloud_ducky_catalog FROM ducky_catalog;
CREATE SHARE share_ducky_catalog FROM cloud_ducky_catalog
(ACCESS UNRESTRICTED, VISIBILITY HIDDEN, UPDATE AUTOMATIC);
```

Python: Seamless integration with pandas, polars & arrow

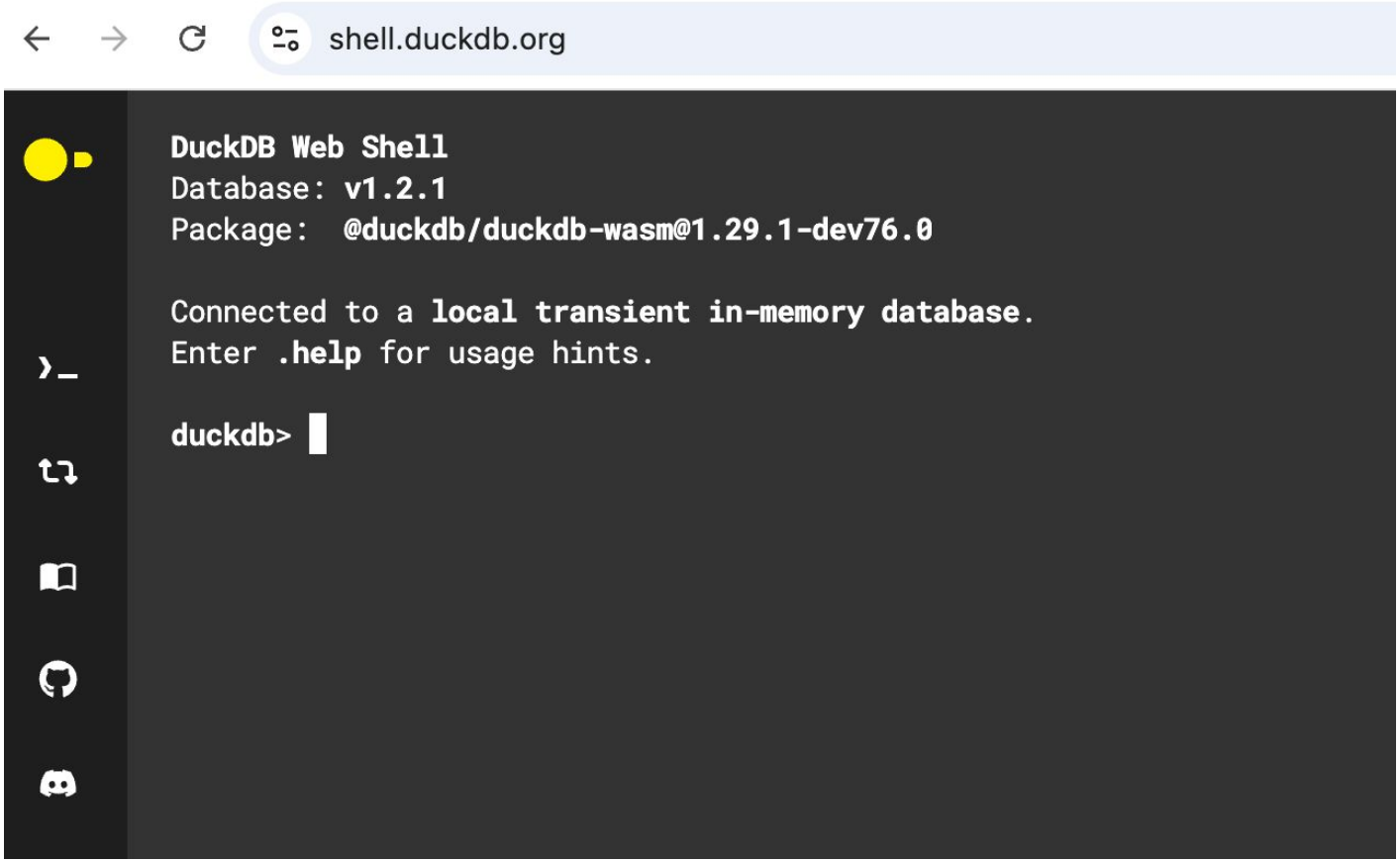
```
my_df = pandas.DataFrame.from_dict({"a": [42]})

results = duckdb.sql("SELECT * FROM my_df").df()

duckdb.sql("SELECT * FROM my_df").arrow()

duckdb.sql("SELECT * FROM my_df").pl()
```

Run in the browser



duckdb in Fabric

- use in python notebooks
- duckdb extension: onelake (read & limited write)

onelake.sql

```sql
-- Attach to your OneLake workspace and lakehouse
ATTACH '<your_workspace_name>/<your_lakehouse_name>.Lakehouse'
    AS <your_connection_name>
    (TYPE ONELAKE);

USE <your_connection_name>.<your_schema_name>; -- e.g., dbo

SHOW TABLES;
```

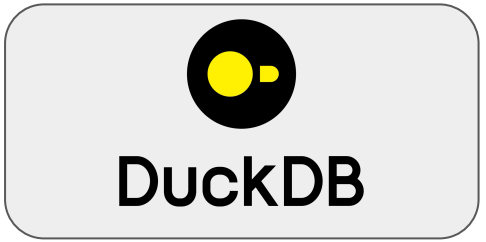
- There is also a SQL Server extension (r/w)

recommended LinkedIn follow: Mimoune Djouallah

Embedded & in process

in-process / embedded





client-server





transactional **analytical**

Portable

- no external dependencies
- a lot of clients
- every platform - linux, windows, mac



```
pip install duckdb
```




```
org.duckdb:duckdb_jdbc
```



```
install.packages("duckdb")
```




```
npm install duckdb
```

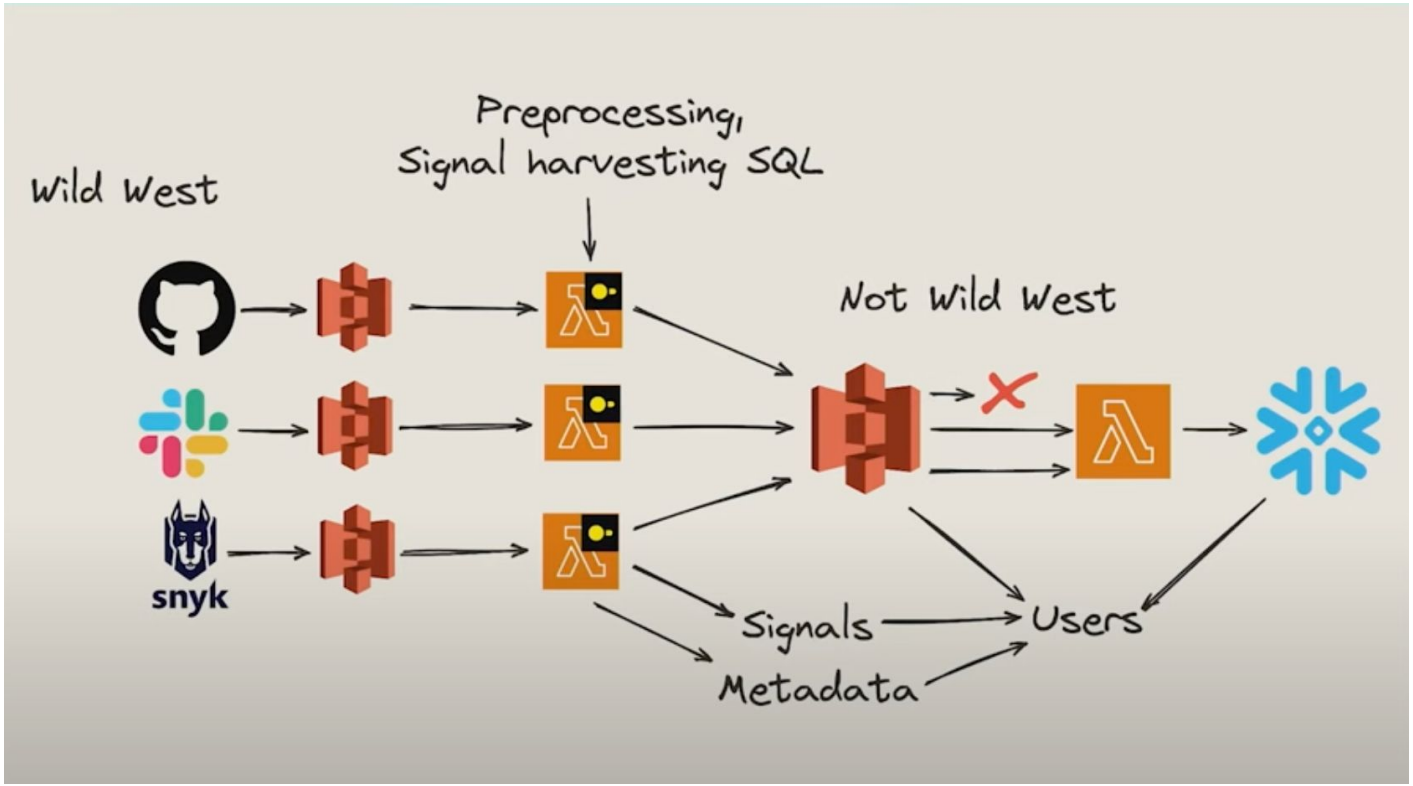


```
brew install duckdb
```



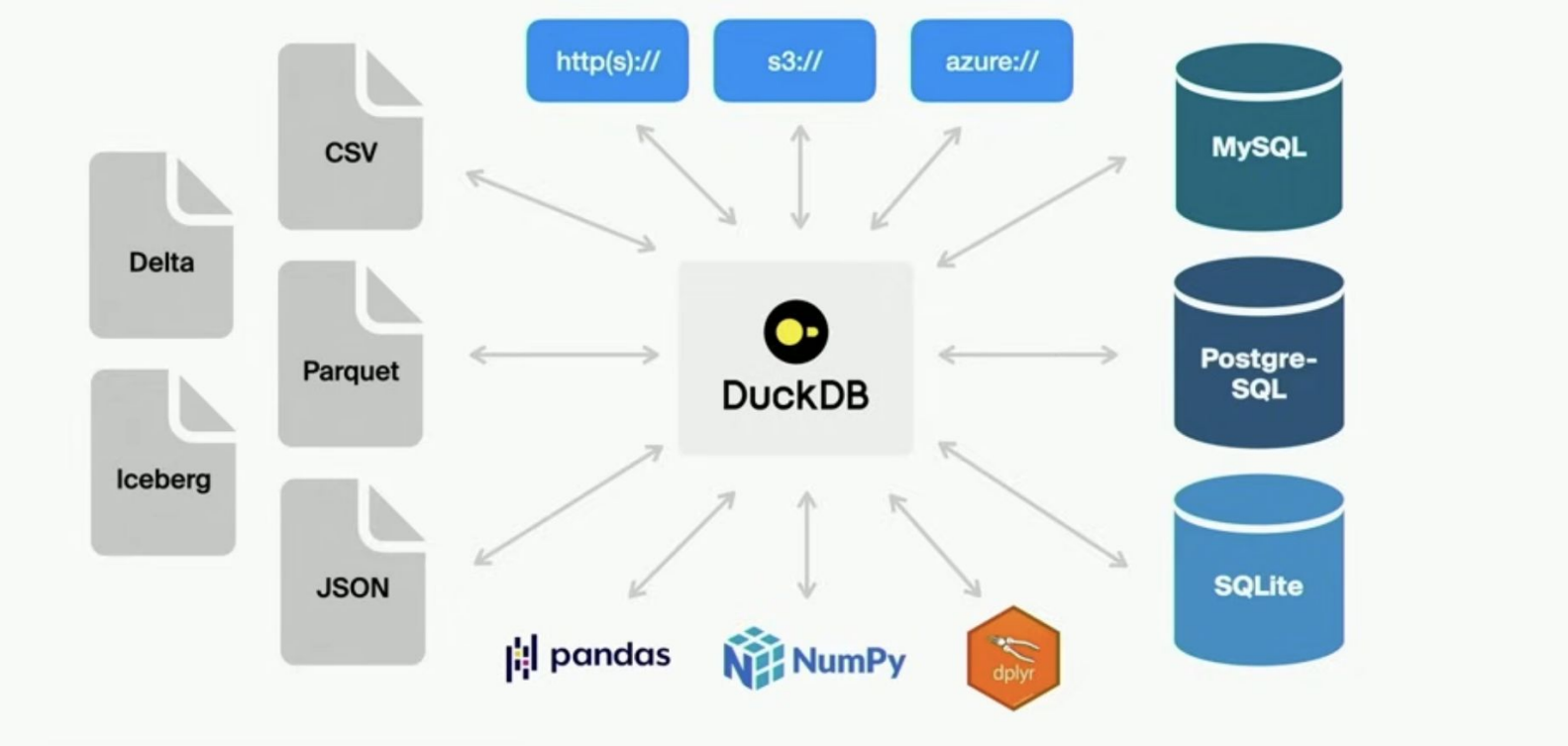
```
cargo add duckdb
```

Run it in lambda function - or a container



https://www.youtube.com/watch?v=TrmJilG4GXk

Flexible



DuckDB: Crunching Data Anywhere, From Laptops to Servers • Gabor Szarnyas • GOTO 2024
https://youtu.be/9Rdwh0rNaf0

Extensions

- data formats
- storage locations
- sql syntax
- types & functions
- addons like the ui

```
INSTALL spatial;
LOAD spatial;
```

- features are core extensions and autoloaded
  - https
  - json
  - parquet
  - delta

Current state of duckdb (March 2026)

Inital Commit:	July 2018
v1.0:	June 2024
v1.4:	current & LTS
v1.5:	tomorrow :-)

“Build a correct system first, then make it fast”

Since 1.0: extending functionality & make it fast

The commercial background



foundation

- non profit
- owns IP



- development & consulting
- no VC



- Commercial SaaS offering
- VC backed
- Labs as co founder

What to do with it?



Local prototyping & exploration

01 - Analyse local

Run memory

```
1  SELECT *
2  FROM   read_json("/Users/matthiasniehoff/dev/git/duckdb/data/simple/employees.json");
3
```

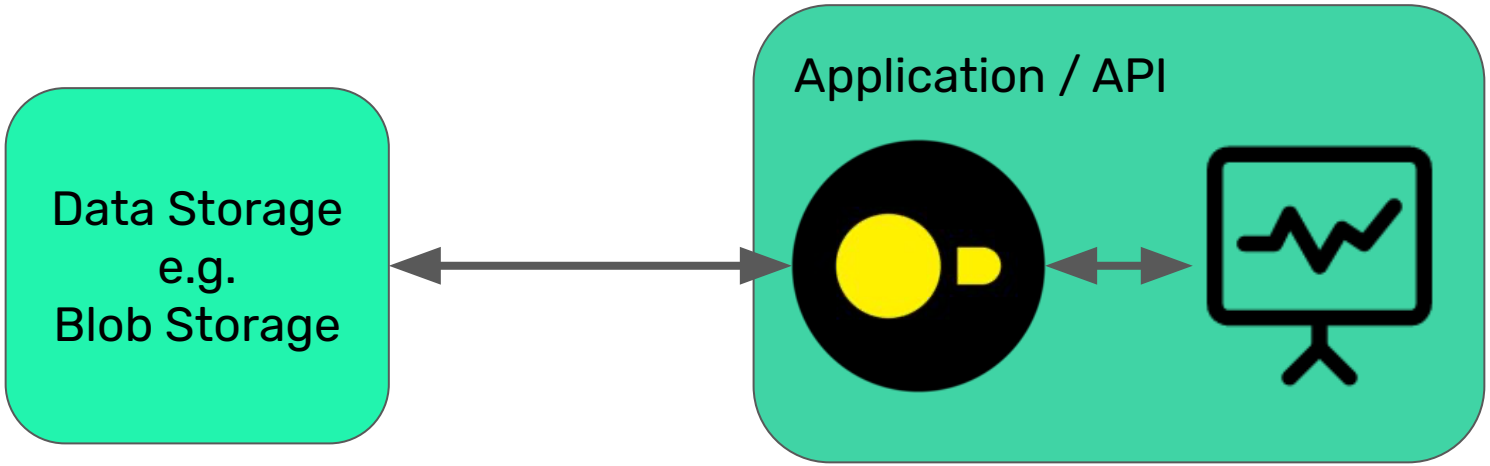
Run memory

```
1  SELECT department,
2         avg(salary)
3  FROM   read_json("/Users/matthiasniehoff/dev/git/duckdb/data/simple/employees.json")
4  GROUP  BY all;
```

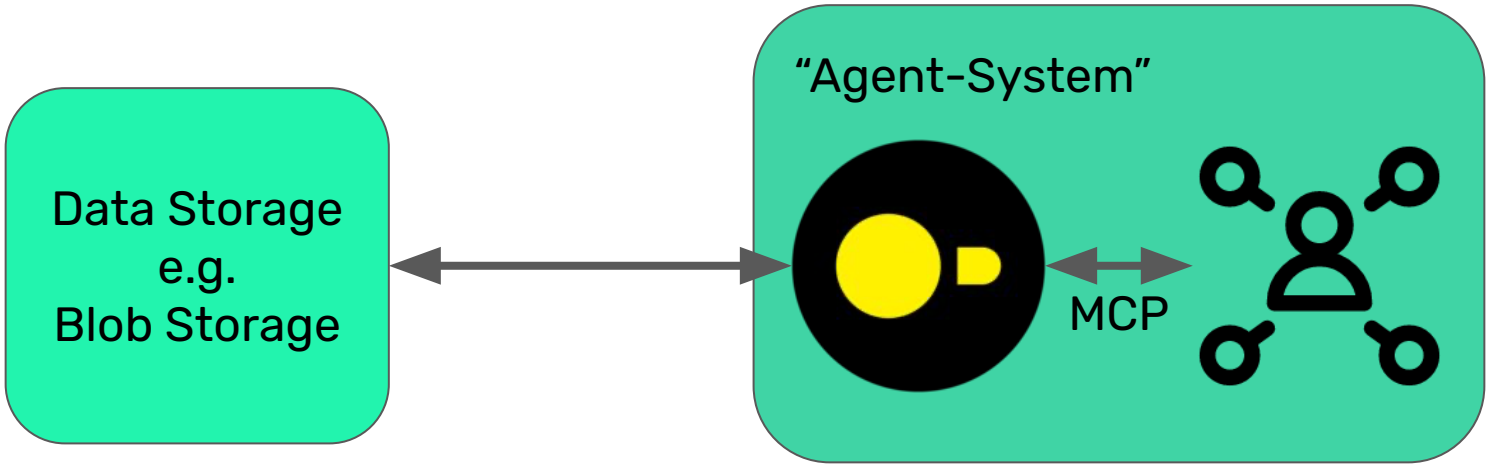
No config - No dba - Open standards

Data Applications / API

The query engine for a fast user experience



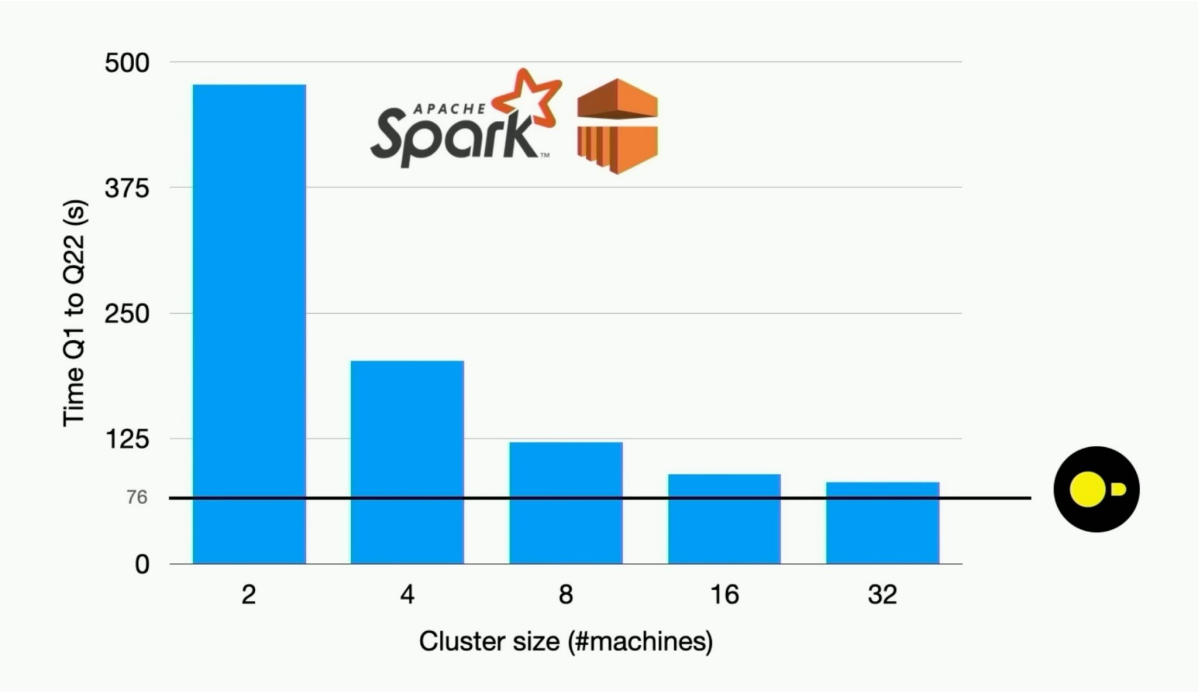
Data Access for LLM



https://github.com/motherduckdb/mcp-server-motherduck

Reducing costs

- move compute towards data
- short lived instances e.g. per user
- replace proprietary compute
- replace distributed compute



https://youtu.be/9Rdwh0rNaf0?feature=shared&t=1712

Use in / for data processing pipelines*

- faster, leaner pipelines
- replace existing code, e.g. pandas



* limited write support for e.g. open table formats

Limitations

- multi user support limited
  - either one user read/write
  - or multiple users read
- not for write heavy workloads
- write support for lakehouse formats still early
- per default: single noded - only scale up vertically
- you still need a runtime :-)

What are the implications on data & software architecture?

Why TPC Is Not Enough: An Analysis of the Amazon Redshift Fleet

Table 8: Table Size Distribution: *Shows the distribution of an estimate of the number of rows stored in non-empty tables across the fleet and for the TPC benchmarks.*

Rowcount Bucket	Fleet	TPC-H	TPC-DS
$(10^0, 10^1]$	12.3%	12.5%	-
$(10^1, 10^2]$	10.8%	12.5%	25.0%
$(10^2, 10^3]$	12.0%	-	-
$(10^3, 10^4]$	12.9%	-	16.6%
$(10^4, 10^5]$	14.2%	-	12.5%
$(10^5, 10^6]$	13.5%	-	4.2%
$(10^6, 10^7]$	11.4%	-	4.2%
$(10^7, 10^8]$	7.6%	12.5%	8.3%
$(10^8, 10^9]$	3.5%	25.0%	12.5%
$\geq 10^9$	1.8%	37.5%	16.7%

https://assets.amazon.science/24/3b/04b31ef64c83acf98fe3fdca9107/why-tpc-is-not-enough-an-analysis-of-the-amazon-redshift-fleet.pdf

Single Node Data Processing Engines



pandas

single node
not bigger than
memory

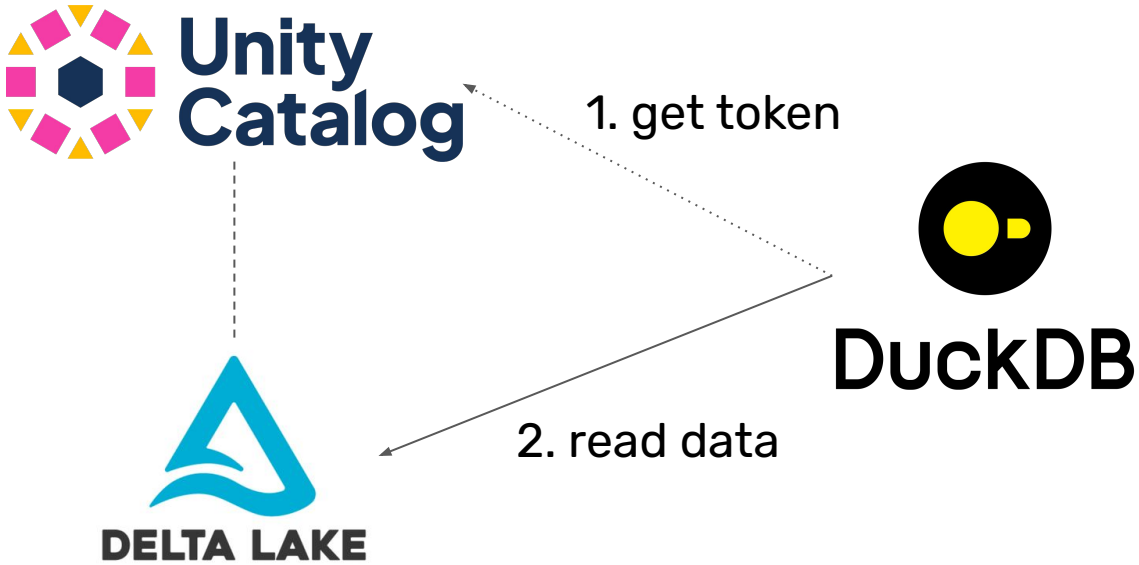
DuckDB

single node
bigger than
memory

PySpark

multi node
linear
scaling

Alternative Query Engine (for reading!)

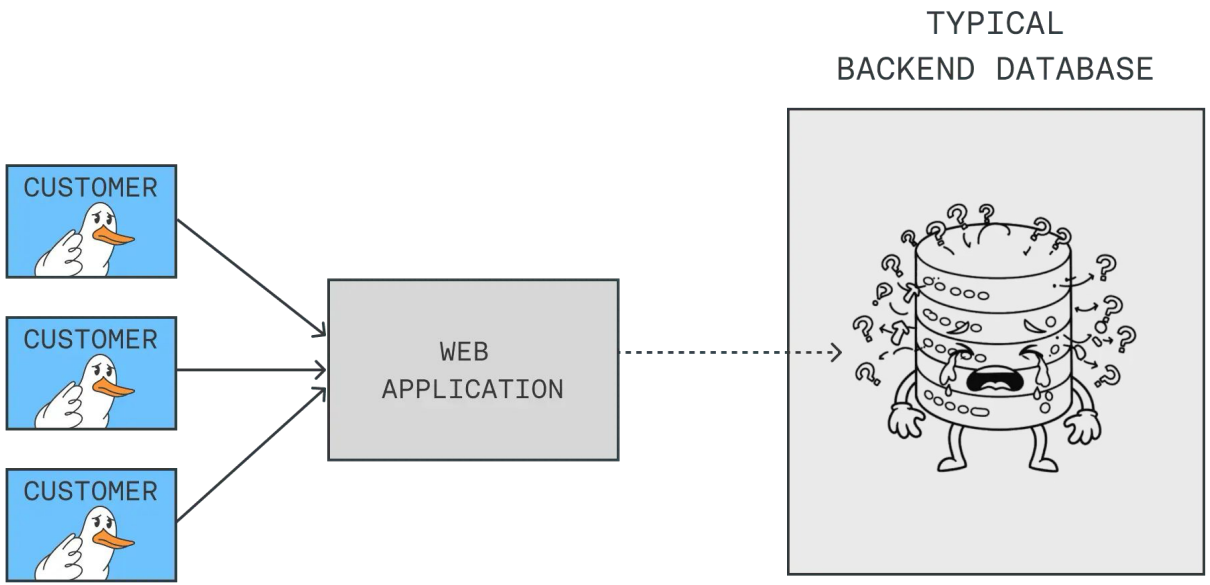


https://www.codecentric.de/en/knowledge-hub/blog/access-databricks-unitycatalog-from-duckdb

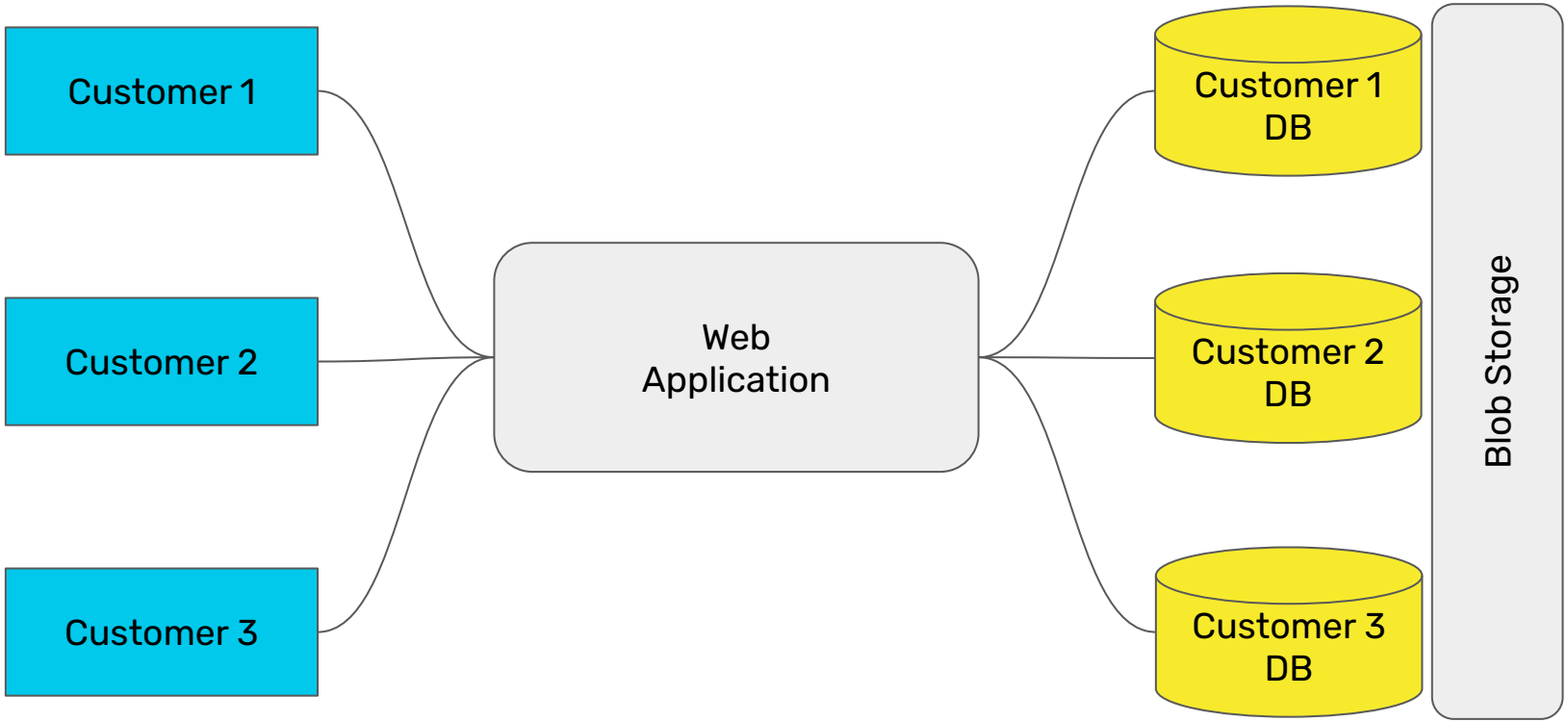
Simpler data driven applications



Scaling Analytical Applications



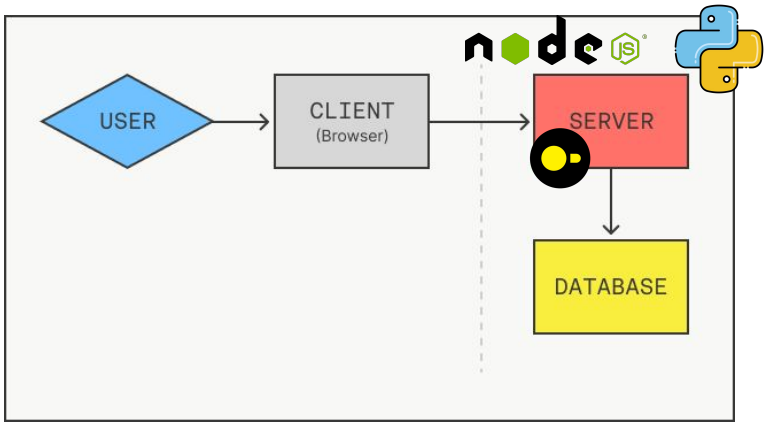
Customer Facing Analytics



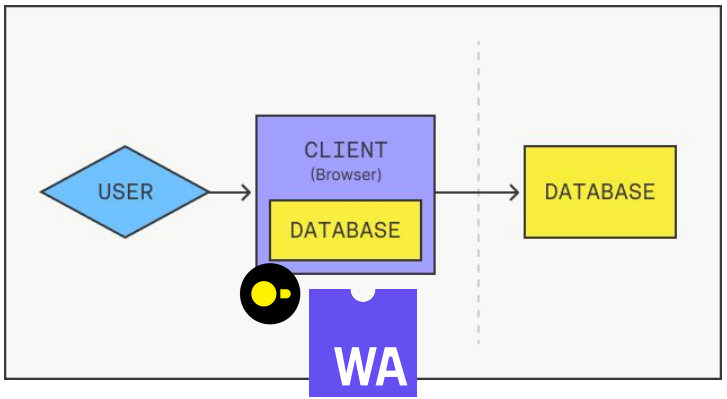
https://motherduck.com/blog/scaling-duckdb-with-ducklings/

Going Beyond 3 Tier Data Architectures" ?

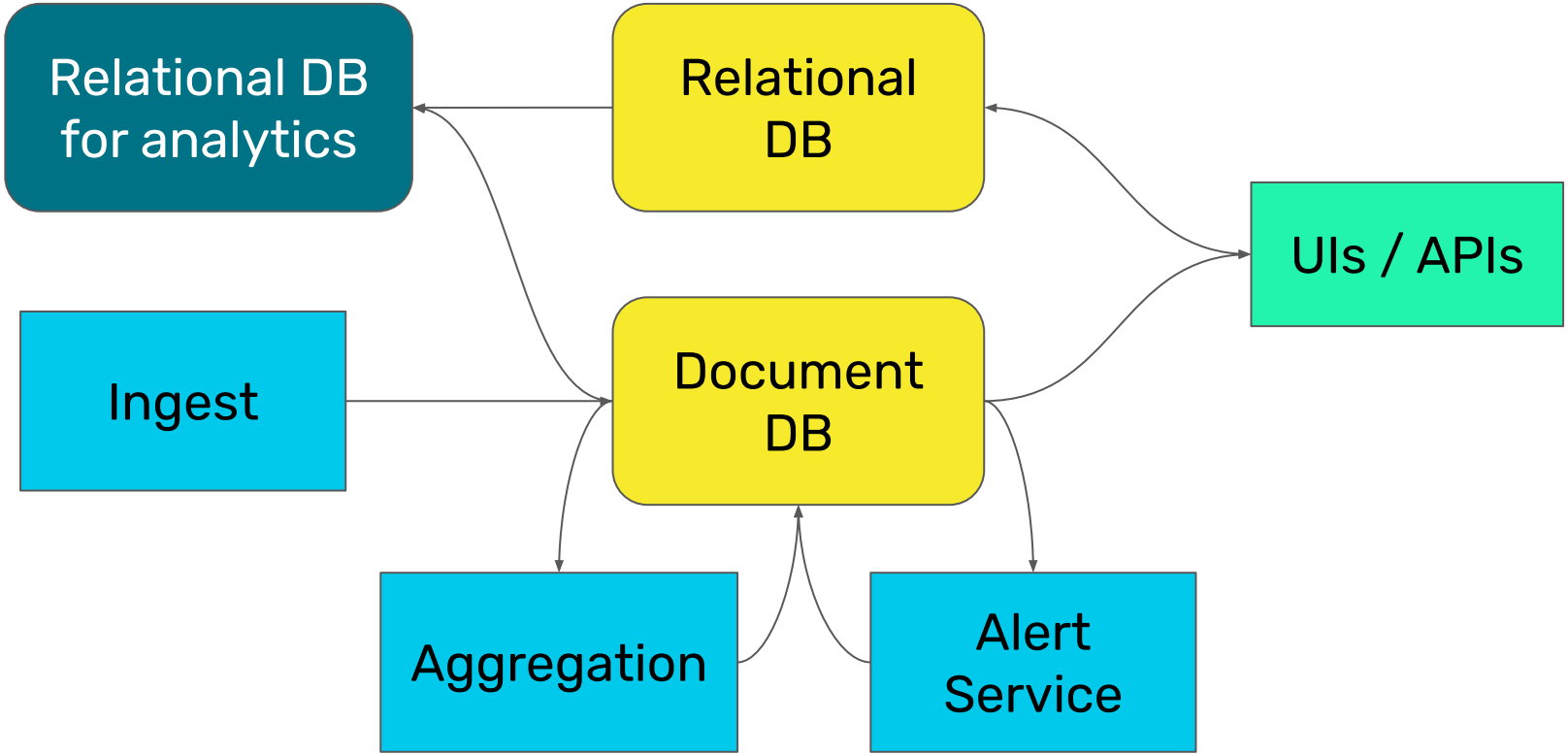
traditional 3-tier



client side computing

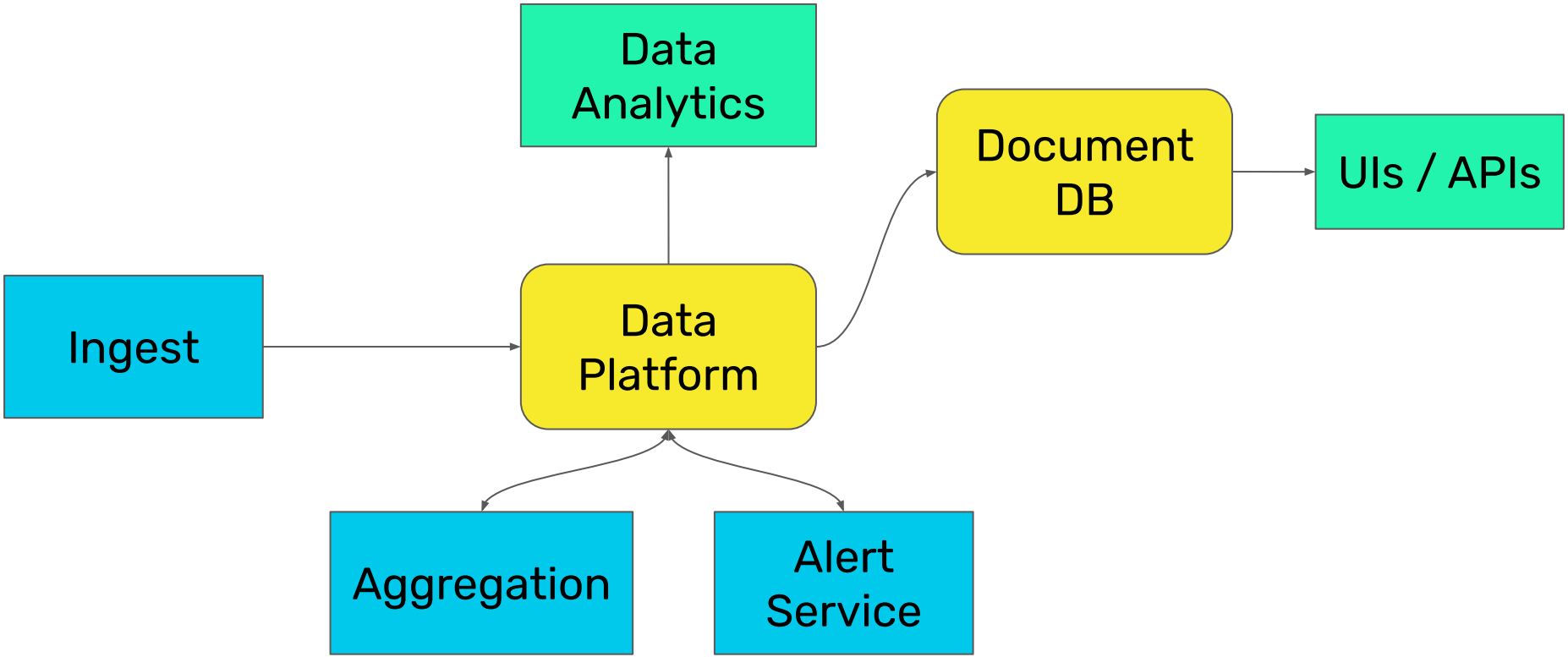


What we had

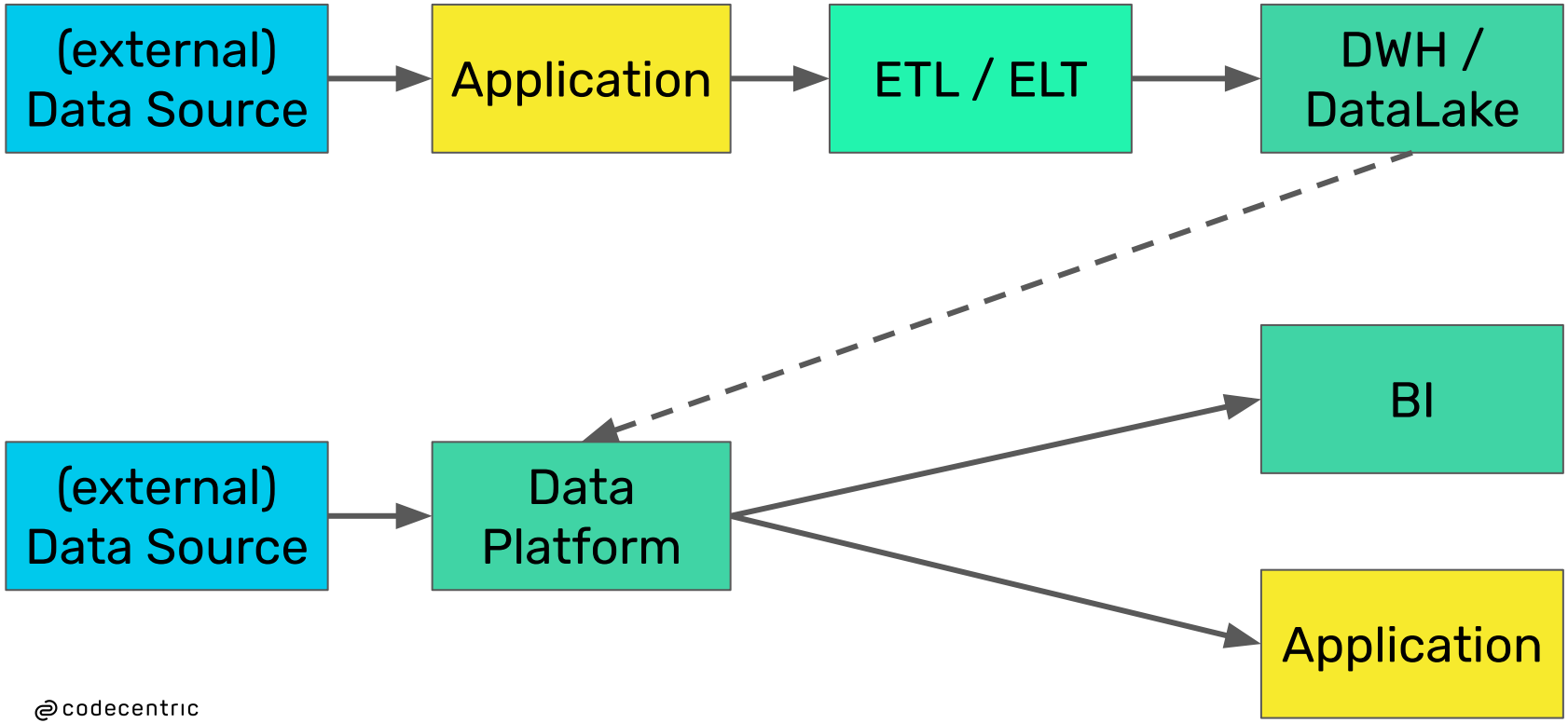

Relational DB for analytics
Relational DB
UIs / APIs
Ingest
Document DB
Aggregation
Alert Service

The web application is actually a data platform with an UI

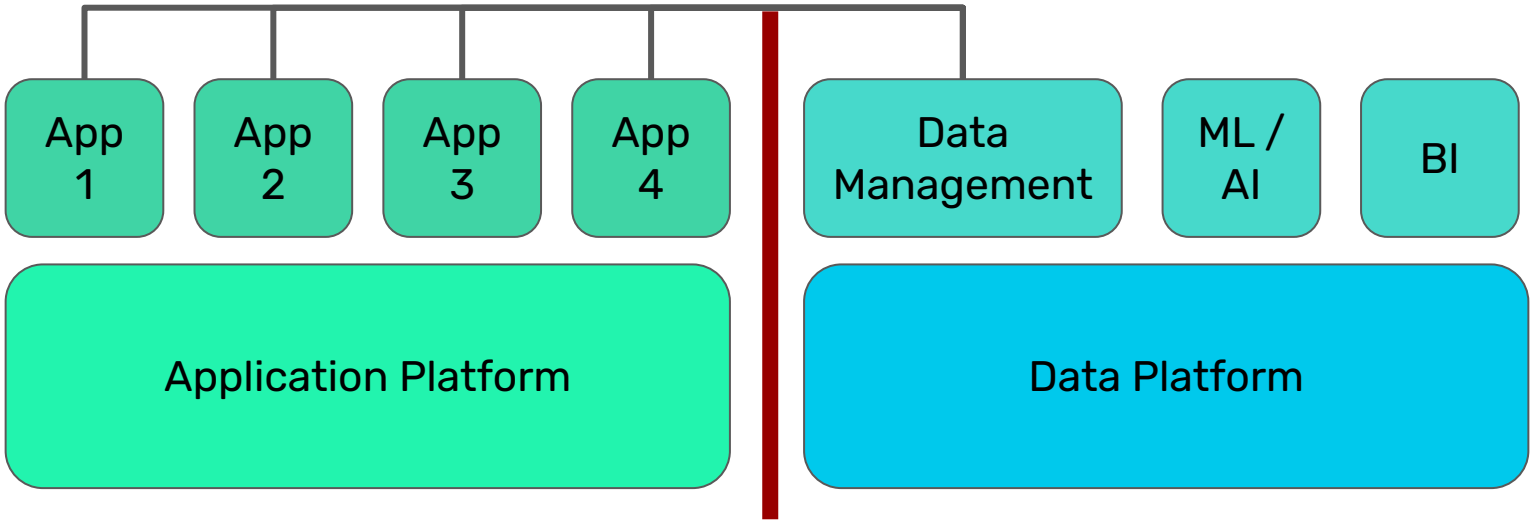
What we have now



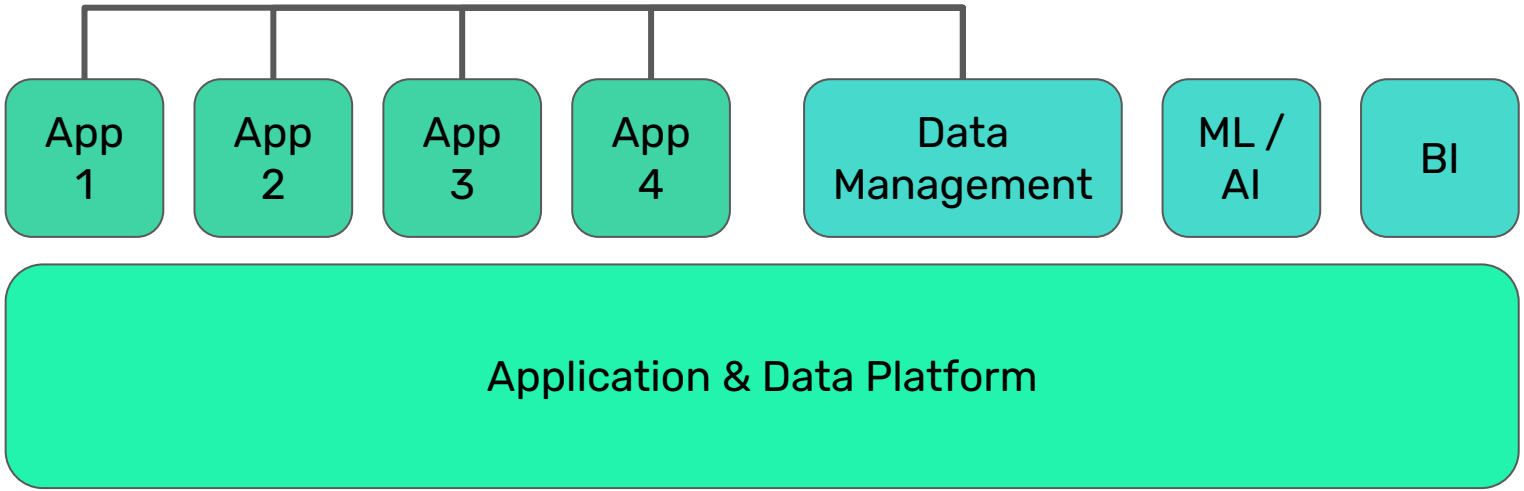
Shift left in data flows



Why build different platforms for data & application?



One platform for application and data



When data analytics & application become one

- Less friction between operational and analytics
- Define and encapsulate responsibilities
- Optimized data representation

- Know and use the tooling available on both sides
- Engineering best practices also apply to data




Takeaways

Conclusion

- Great for exploratory data analysis
- Really easy to use
- Huge momentum
- Will impact & simplify data architectures
- Already in use for “data driven applications”
- Closes gap between data & application / agents



We support you with your data projects

Feel free to contact us!

codecentric AG
Hochstraße 11
42697 Solingen

Matthias Niehoff
Principal Data Architect
matthias.niehoff@codecentric.de
www.codecentric.de

codecentric



Session Feedback



This session



Whole event