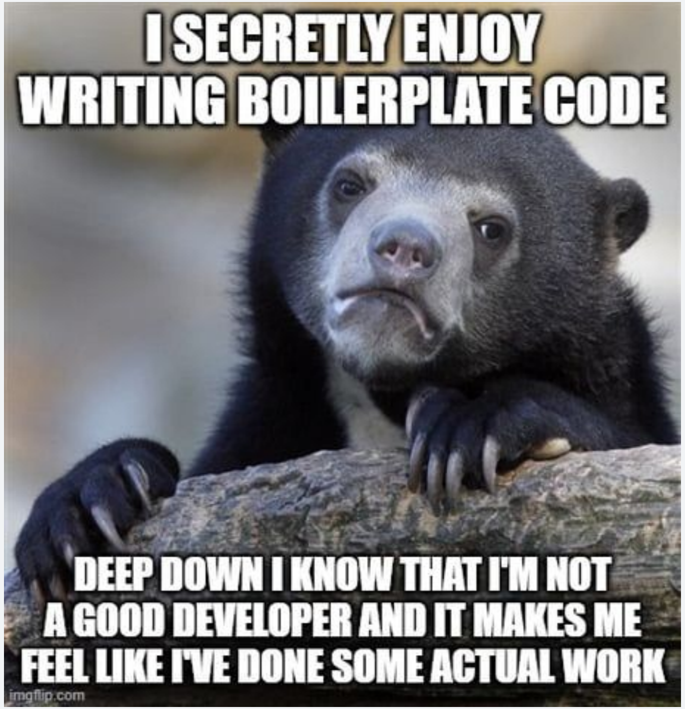
GoodData

WTF is an Analytics Lake

Building an open data service layer with Arrow, DuckDB and Semantic Layer

Raise your hands!

Who likes writing boilerplate code?



Building an Analytics Stack is Hard!

Developer Velocity

You want to build an analytics stack powering a BI/analytics platform

With every new **data service**, developers have to solve the same problems again and again. For instance:

- Server handling, interfaces
- Config management
- Routing, Load balancing

Often you spend **80%** of your time with boiler-plating

Developers Wish For …

Skeleton data service

- Implement only the service logic, e.g. pivoting

Examples

- Real and meaningful data services

Performance

- Have you ever seen a really responsive end-user experience?
- Example: Tableau server

Our Initial Motivation

GoodData 1.0

We built the first version of our analytics stack more than **15** years ago!

We combined it with large amounts of our own proprietary code

- with the technology available at that time

GoodData 2.0

We decided to rewrite the platform on greenfield **4** years ago

- Reuse open-source and SaaS services as much as possible
- We fell in love first with Apache Arrow and later with DuckDB
- Some prospects even ask for embedding custom data services into our platform

Anatomy of an Analytics Lake

Physical Execution

Usually it consists of:

- **Querying** the data source(s) or pre-aggregations
- **Post-processing** the data - e.g. pivoting, ML
- **Caching**

We call this GD-specific stack **FlexQuery**

It's powered by (GD-agnostic) engine called **Longbow**

- Built on top of Apache Arrow, DuckDB, and Pandas (TODO: Polars)
- We are considering an **open sourcing** of Longbow engine

The Importance of Semantic Layer

Business users

-> Semantic(logical) report definition - *SUM(amount) BY product_category*

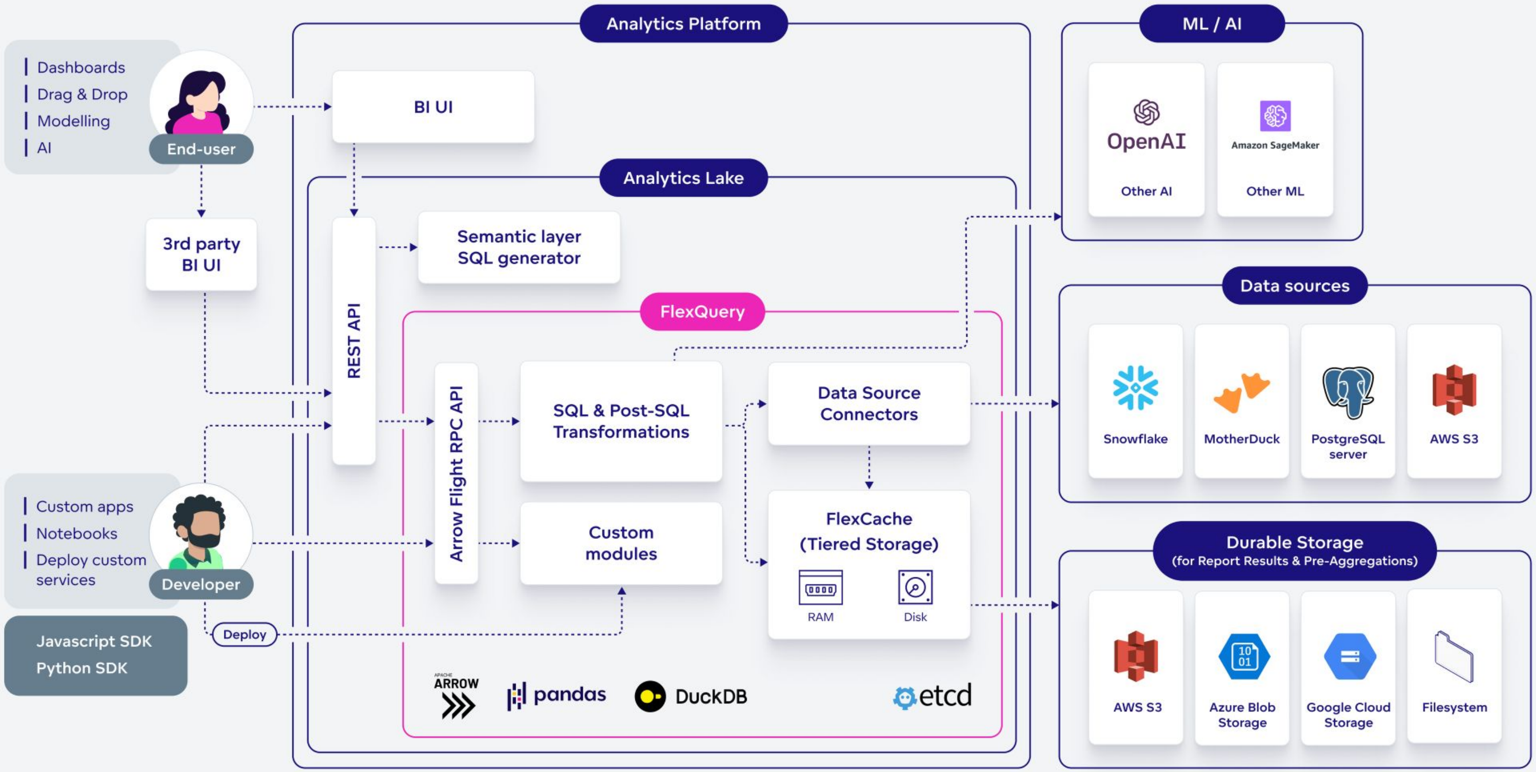
-> Semantic layer with mapping to data - *[SALES] -> [PRODUCTS]*

-> Physical execution plan (SQL) - *SELECT SUM(amount) FROM JOIN GROUP BY*

We use Apache Calcite for translation of business requests to SQL

- Thanks to the community, especially to Julian Hyde ;-)

Analytics Lake = FlexQuery + Semantic Layer



Why we chose Apache Arrow

Arrow format

Language-independent

Columnar memory format

Efficient for

- Analytic operations on modern hardware (CPUs, GPUs)
- Communication of services over the network

Zero-copy reads!

But You Get Much More Out of the Box!

I/O operations with the data - disk, object storages, ...

Convertors - CSV, Parquet, from JDBC, ...

Streaming computation on top of Arrow data - Acero engine

Flight RPC - API blueprint and infrastructure

Arrow Database Connectivity (ADBC) - query external data sources

Easy to Integrate

Arrow is becoming de facto standard

- Even big players like Snowflake(JDBC, 2020) are adopting it

Other technologies we found valuable and integrated:

- Transform
  - Pandas (2.0), Polars
  - DuckDB
- Federate
  - DuckDB

What we set out to build

FlexCache - tiered storage

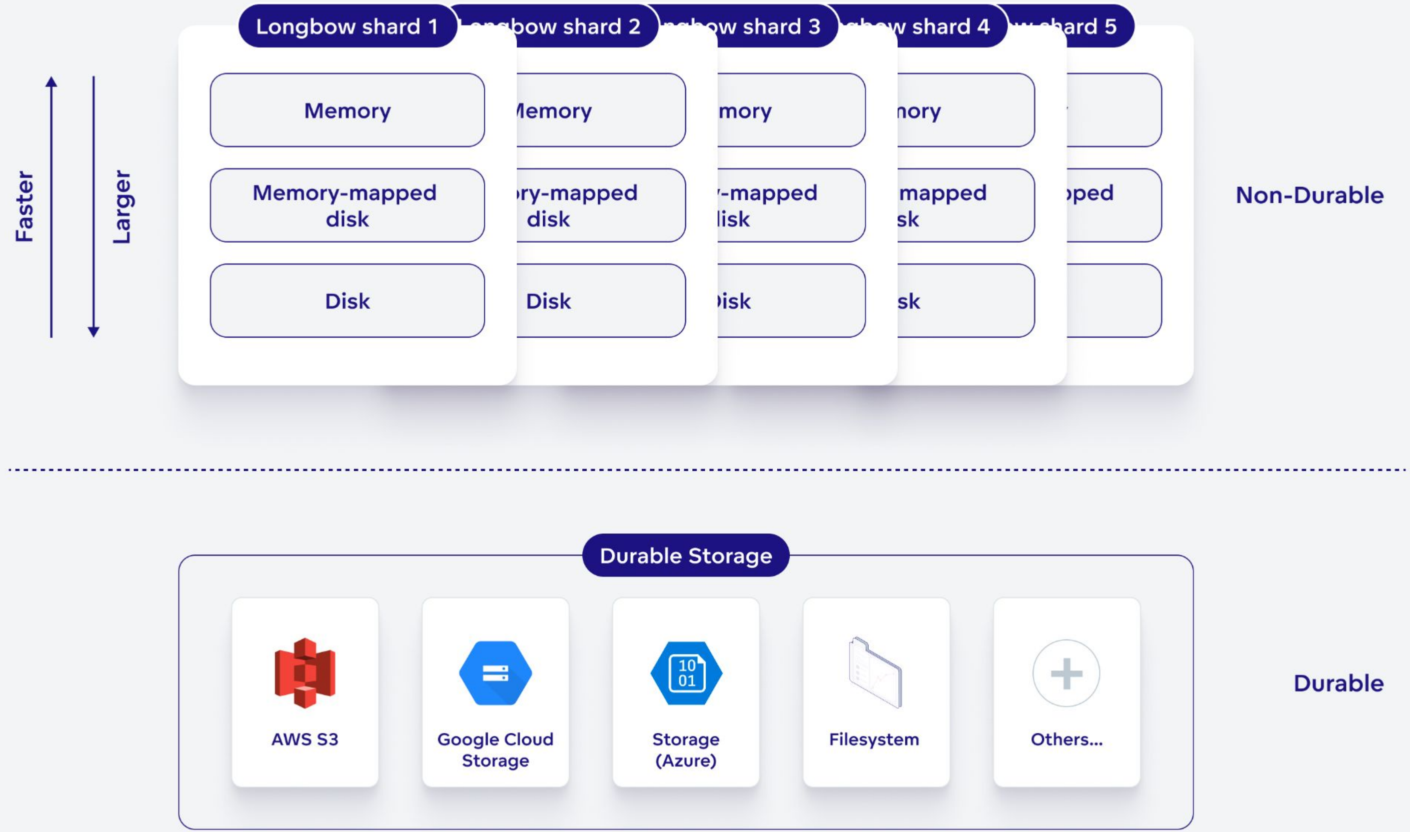
Memory <-> Disk <-> Object Storage (S3, ABS, GCS)

Arrow tables

Policies

- Various rules for promoting, demoting
- Enforce limits, isolate tenants, define replicas
- Leverage for pricing

```
cache_tier.in_memory = {
  type = "memory", max_flight_size = '512M', upload_spill_to = "mapped_disk",
  move_after = 60, move_to = "mapped_disk", spill_to = "parking_lot", priority = 10
}
cache_tier.mapped_disk = {
  type = "disk_mapped", max_flight_size = '2048M', upload_spill_to = "parking_lot", spill_to = "parking_lot"
}
cache_tier.parking_lot = {
  type = "disk", promote_after_hits = 10, promote_hits_window = 5, promote_to = "*best_fit"
}
```



Faster
Larger
Longbow shard 1
Memory
Memory-mapped disk
Disk
Non-Durable
Durable Storage
AWS S3
Google Cloud Storage
Storage (Azure)
Filesystem
Others...
Durable

Data Source Connectors

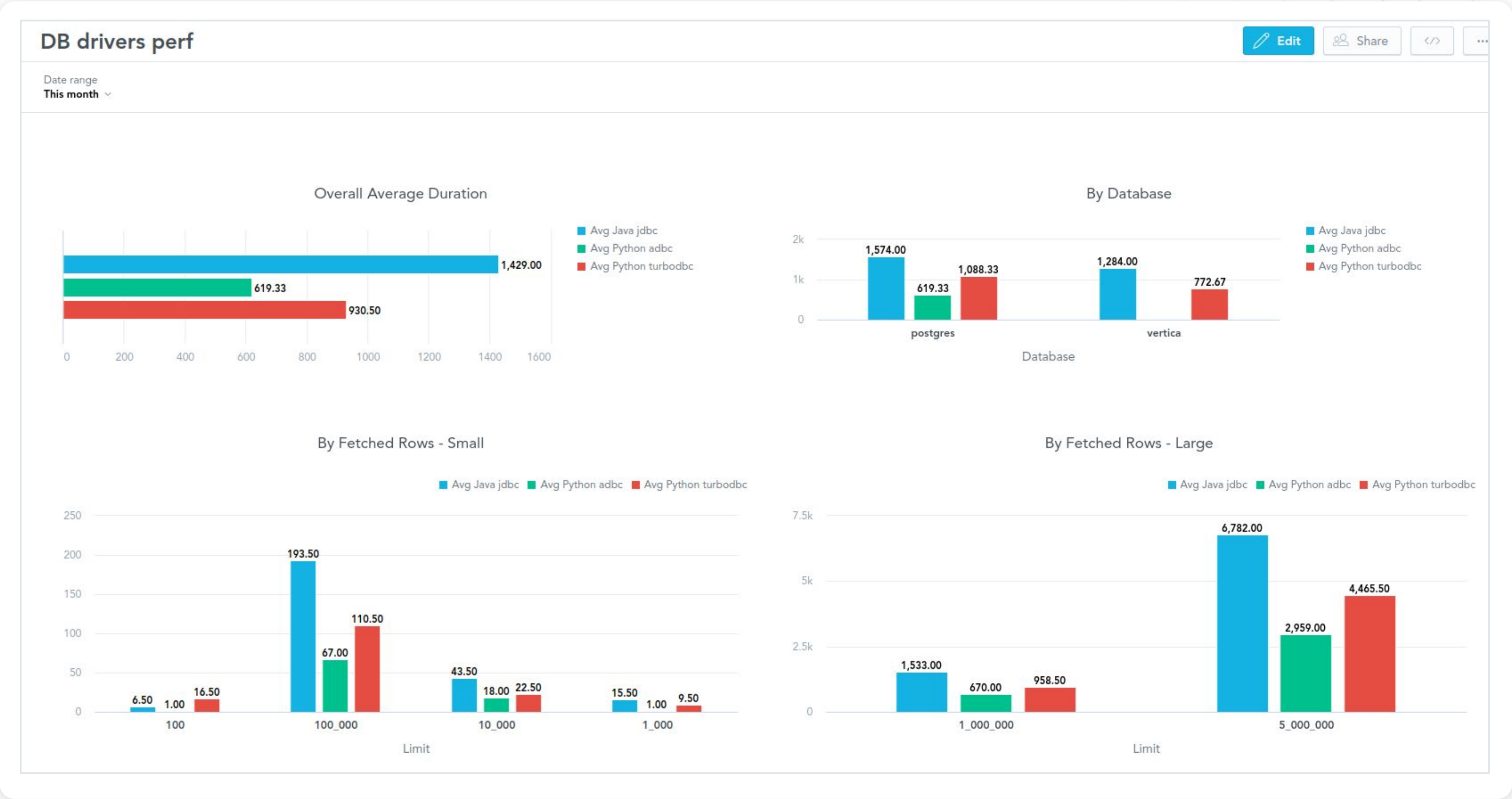
TurboODBC -> ADBC

- Fallback to JDBC is possible

Flexible deployment / Limit enforcement

- Max connections, timeouts, ...
- Per node or per data source type

Streaming to FlexCache


DB drivers perf
Edit
Share
Date range
This month
Overall Average Duration
Avg Java jdbc
Avg Python adbc
Avg Python turbodbc
1,429.00
619.33
930.50
0
200
400
600
800
1000
1200
1400
1600
By Database
2k
1k
0
1,574.00
619.33
1,088.33
1,284.00
772.67
postgres
vertica
Database
By Fetched Rows - Small
250
200
150
100
50
0
6.50
1.00
16.50
193.50
67.00
110.50
43.50
18.00
22.50
15.50
1.00
9.50
100
100_000
10_000
1_000
Limit
By Fetched Rows - Large
7.5k
5k
2.5k
0
1,533.00
670.00
958.50
6,782.00
2,959.00
4,465.50
1_000_000
5_000_000
Limit

Result Post-Processing

Dataframe operations

- Even easier to implement new operations than modules - dataframe IN/OUT
- Pivoting, sorting, paging for BI
- Three ML algorithms in Beta, created in a few days

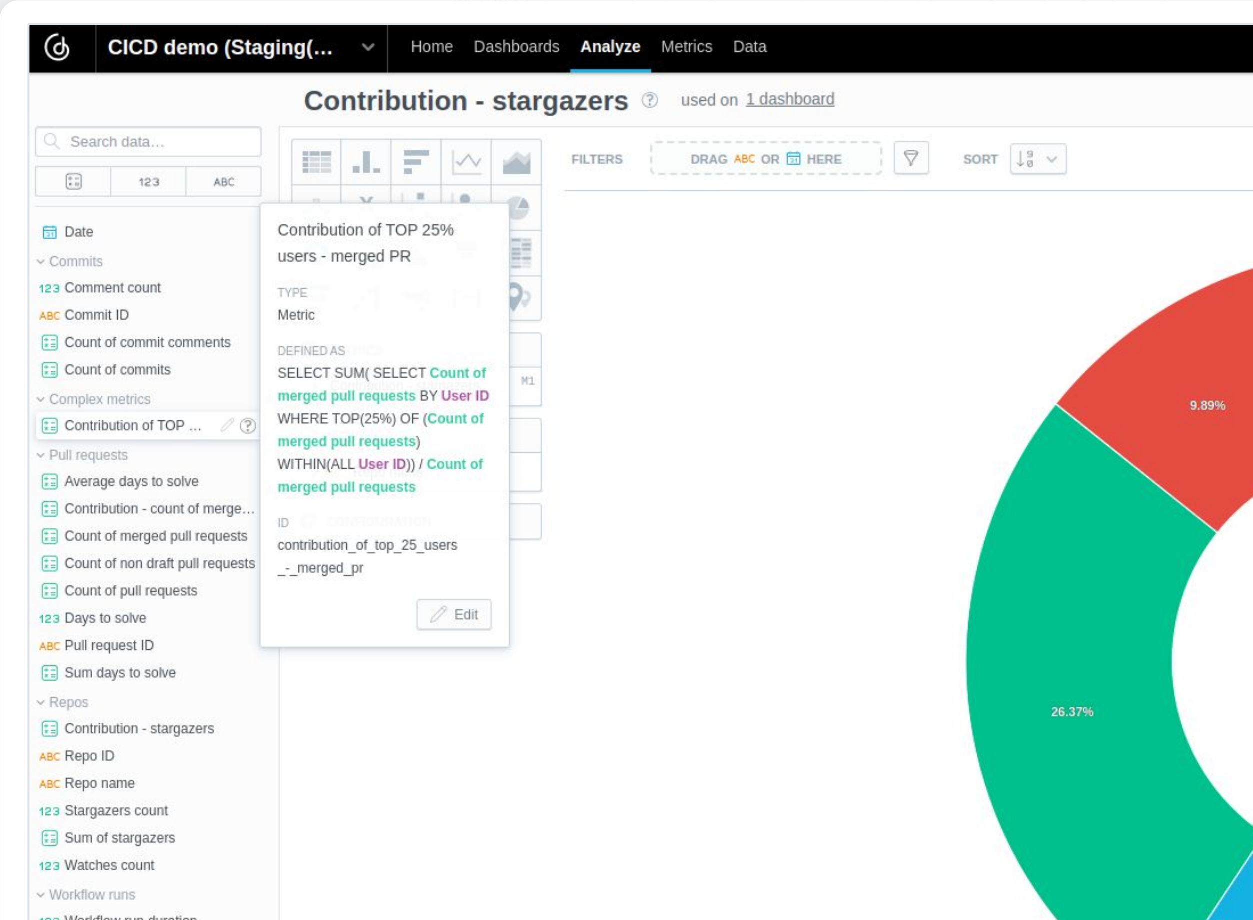
Federation

- Query hybrid data sources, stream to FlexCache
- Object stores, databases
- Query FlexCache with DuckDB

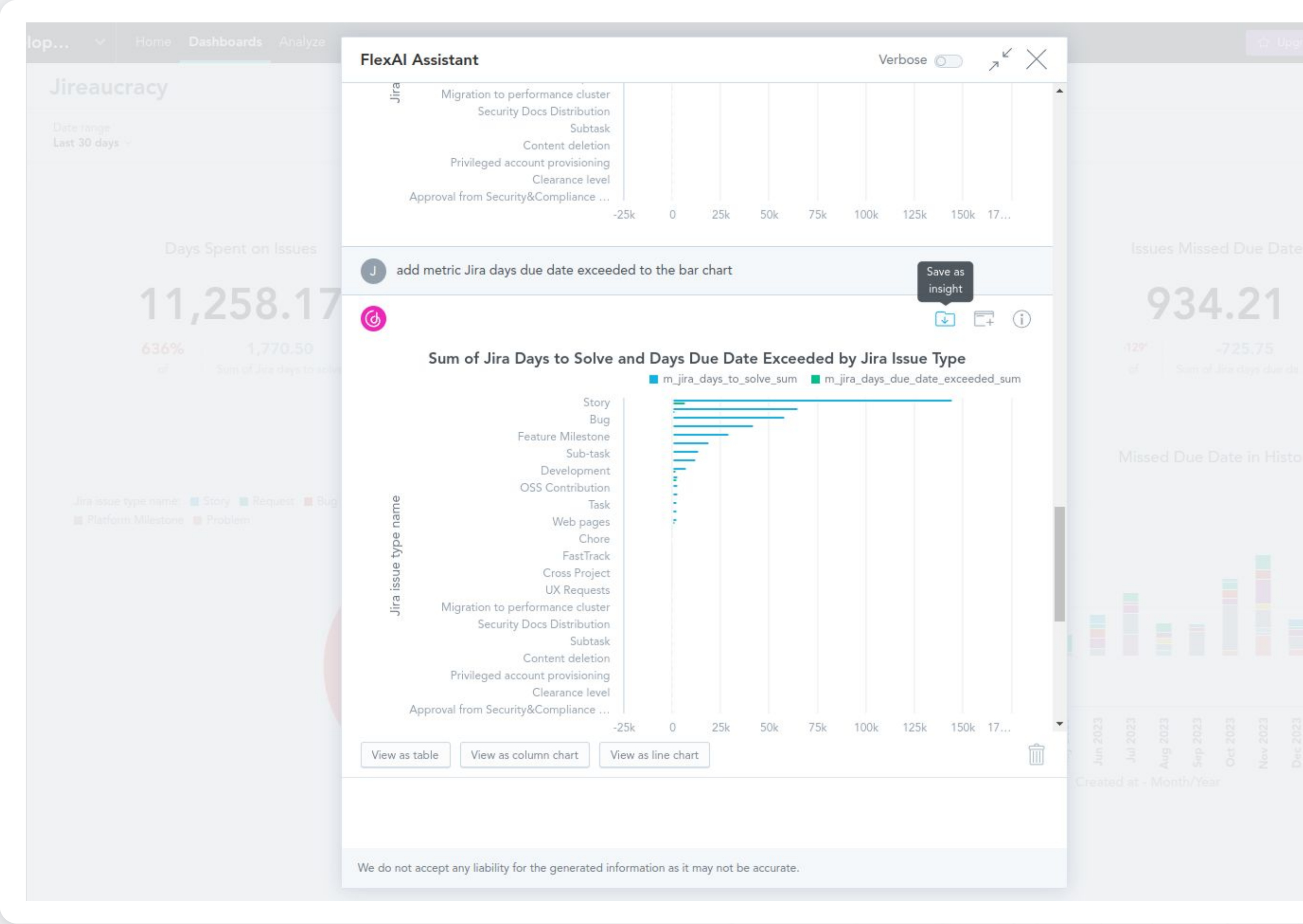
Real UX/DX

From Developers to Developers

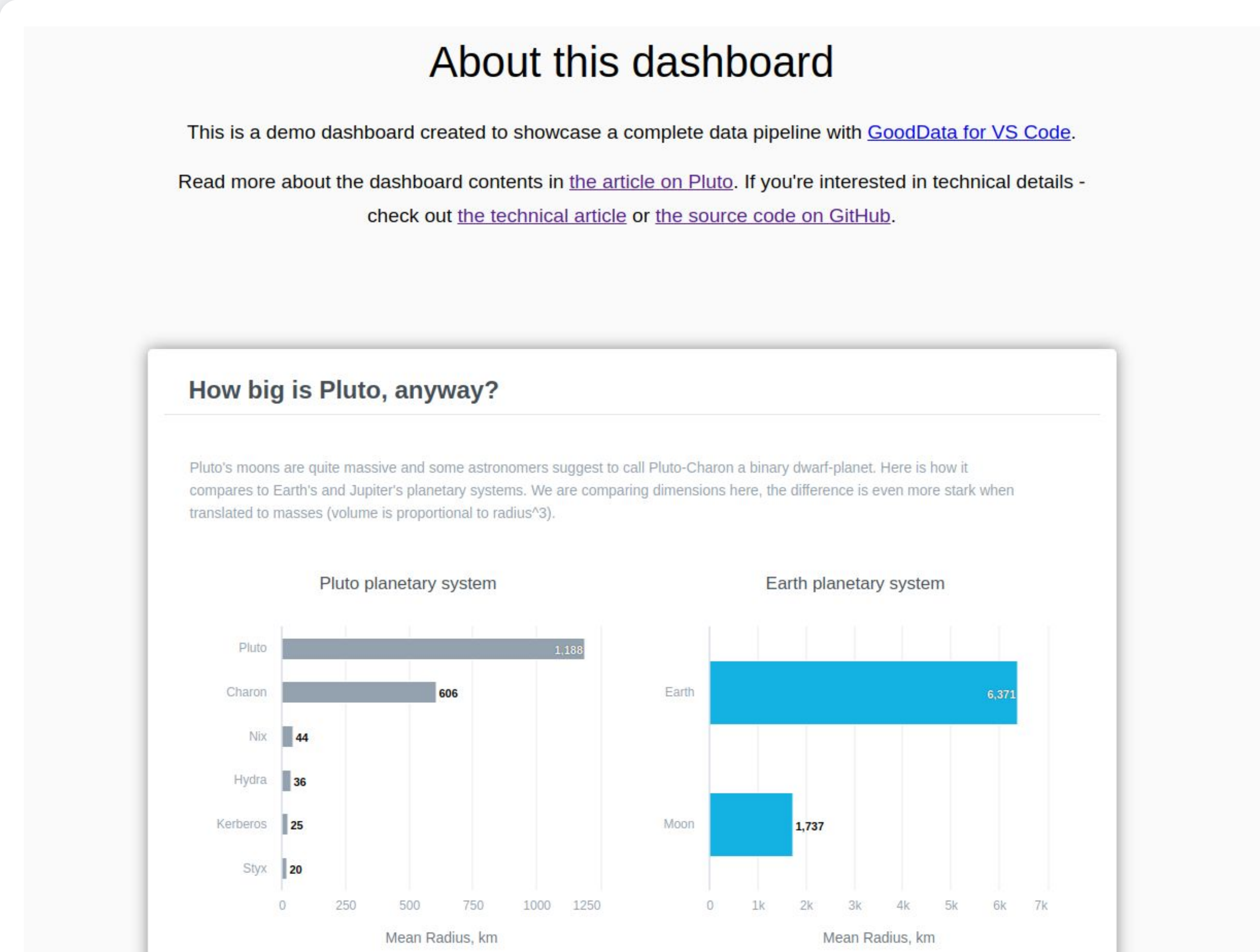
Self-service drag&drop - freeing hands of developers



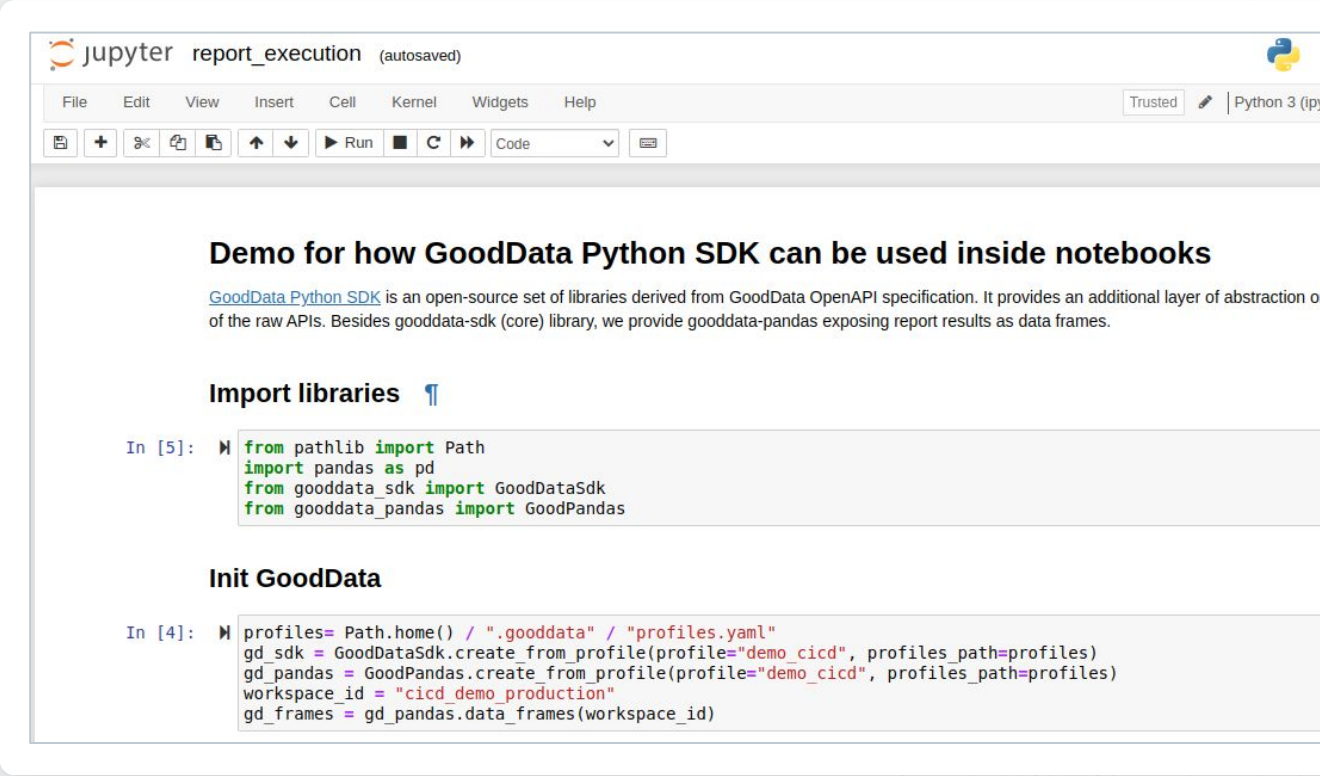
Must not forget about AI chat bot



DX - Analytics Lake with UI SDK



DX - Analytics Lake with Python SDK - connect



DX - Analytics Lake with Python SDK
- query stored report

jupyter report_execution Last Checkpoint: 49 seconds ago

File Edit View Run Kernel Settings Help

Markdown Open in... Python 3 (i

9	sum_days_to_solve	Sum days to solve
10	sum_stargazers	Sum of stargazers

Execute Already Defined Report

[8]:
```
insight_df = gd_frames.for_insight('contribution_of_top_25_users_per_repository')
insight_df
```

[8]:

		contribution_of_top_25_users_-_merged_pr
closed_at.month	**repo_name**	
2019-09-01	**gooddata-ui-sdk**	0.500000
2019-10-01	**gooddata-ui-sdk**	0.357143
2019-11-01	**gooddata-ui-sdk**	0.850000
2019-12-01	**gooddata-ui-sdk**	0.680000
2020-01-01	**gooddata-ui-sdk**	0.411765
...	**...**	...
2024-02-01	**gooddata-dashboard-plugins**	1.000000
	gooddata-python-sdk	0.750000
	gooddata-ui-sdk	0.615385
2024-03-01	**gooddata-python-sdk**	1.000000
	gooddata-ui-sdk	0.750000

121 rows × 1 columns

DX - Analytics Lake with Python SDK - custom report

Execute a Custom Report

In [22]:

```python
from gooddata_sdk import ExecutionDefinition, Attribute, SimpleMetric, ObjId, RelativeDateFilter
exec_def = ExecutionDefinition(
    attributes=[
        Attribute(local_id="created_at", label="created_at.month"),
        Attribute(local_id="repo_name", label="repo_name"),
    ],
    metrics=[
        SimpleMetric(local_id="count_commits", item=ObjId(id="commit_id", type="label"), aggregation="COUNT"),
        SimpleMetric(local_id="count_pull_requests", item=ObjId(id="count_pull_requests", type="metric")),
    ],
    filters=[RelativeDateFilter(dataset=ObjId(id="created_at", type="dataset"), granularity="YEAR", from_shift=-
    dimensions=[["created_at"], ["repo_name", "measureGroup"]],
)
df, df_metadata = gd_frames.for_exec_def(exec_def=exec_def)

# use result ID from computation above and generate dataframe just from it
df_from_result_id, df_metadata_from_result_id = gd_frames.for_exec_result_id(
    result_id=df_metadata.execution_response.result_id,
)
df_from_result_id
```

Out[22]:

Repo name	gooddata-dashboard-plugins		gooddata-public-demos		gooddata-python-sdk		gooddata-ui-sdk	
	count_commits	Count of pull requests	count_commits	Count of pull requests	count_commits	Count of pull requests	count_commits	Count of pull requests
Created at - Month/Year								
2022-01	18.0	4.0	NaN	NaN	65	22	164	113
2022-02	3.0	1.0	NaN	NaN	17	5	166	92
2022-03	NaN	1.0	NaN	NaN	8	7	301	126

DX - FlexQuery - TPC-H Q4

```sql
select o_orderpriority, count(*) as order_count
from orders
where
  o_orderdate >= date '1993-07-01'
  and o_orderdate < date '1993-07-01' + interval '3' month
  and exists (
    select 1 from lineitem where l_orderkey = o_orderkey and l_commitdate < l_receiptdate
  )
group by o_orderpriority order by o_orderpriority
```

DX - FlexQuery - Federation

```
tpch4_exec = SqlQuery(
    sql=tpch_q4,
    tables=(
        TableData(
            table_name="orders",
            data=ConnectorQuery(
                payload=orjson.dumps(
                    {"type": "parquet-file", "path": "tpch/orders.parquet",
                     "columns": ["o_orderpriority", "o_orderdate", "o_orderkey"]}
                ),
                sink_method=qc.SinkToFlightPath(flight_path="org1/tenant1/tpch/datasets/orders"),
            ).to_flight_descriptor(ds_id="my-s3"),
        ),
        TableData(
            table_name="lineitem",
            data=ConnectorQuery(
                payload=SqlPayload(sql='SELECT "l_orderkey", "l_commitdate", "l_receiptdate" FROM lineitem'),
                sink_method=qc.SinkToFlightPath(flight_path="org1/tenant1/tpch/datasets/lineitem"),
            ).to_flight_descriptor("postgres")
        ),
    ),
    sink_method=qc.SinkToFlightPath(flight_path="org1/tenant1/tpch/reports/4.result"),
```

DX - Call FlexQuery

```
q = qc.QuiverClient("grpc://localhost:16004")

with q.flight_descriptor(tpch4_exec.to_flight_descriptor()) as stream:
    result: pyarrow.Table = stream.read_all()
    result.sort_by( sorting: "order_count", descending=True)
    # noinspection PyArgumentList
    df = result.to_pandas(split_blocks=True, self_destruct=True)
    st.bar_chart(data=df, x="o_orderpriority", y="order_count")
```

A Taste of the Future

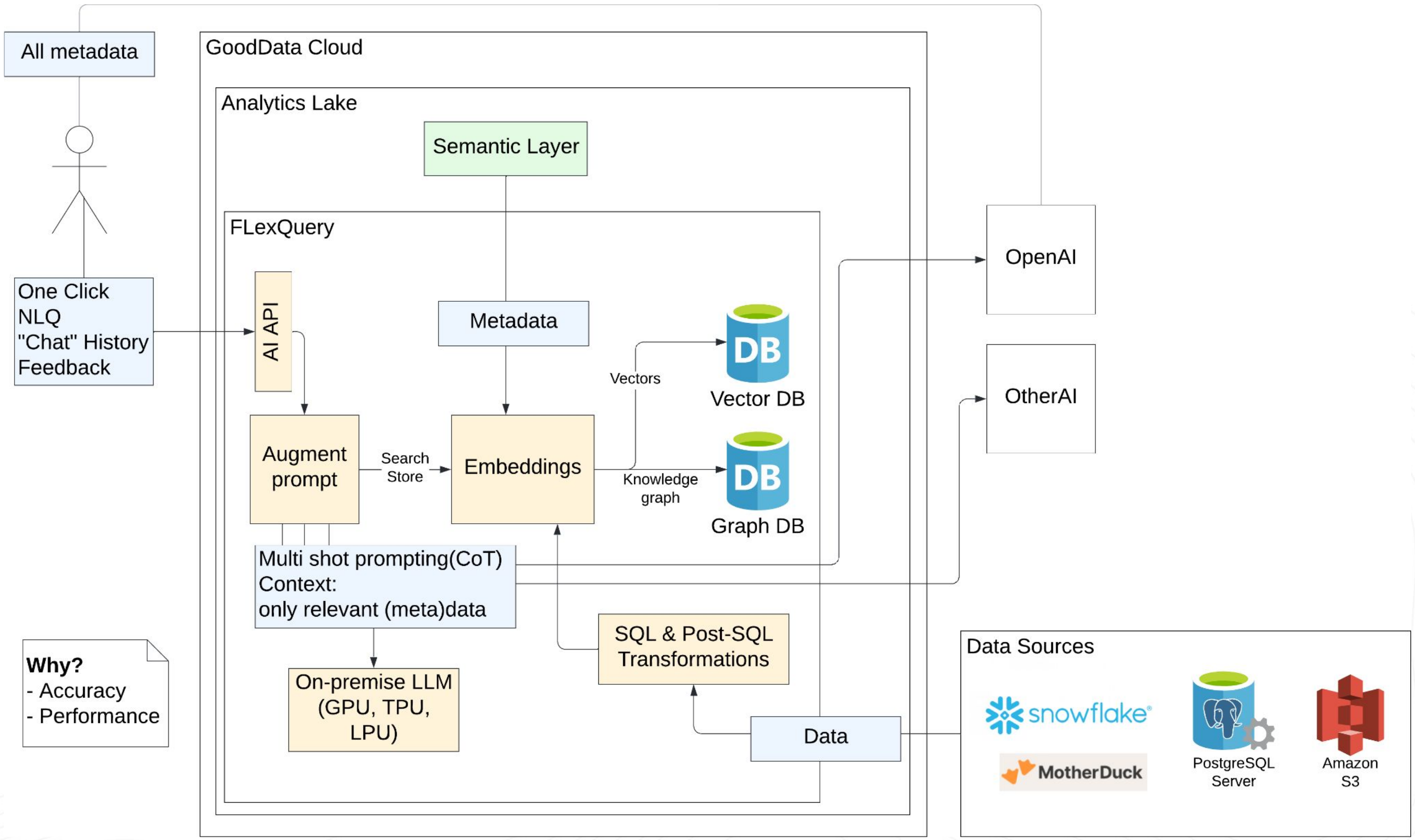
DX - AI Explain - Using FlexCache

```
AIExplainAction(
    source_data="org1/tenant1/tpch/reports/4.result",
    explain_parameters={
        "max_sample_size": 1000,
        "method": "OpenAIWithRAG",
    },
    gpt_cache=True
)
```

DX - AI Explain - Chain of Commands

```
a = AIExplainAction(
    source_data=SqlQuery(
        sql=tpch_q4,
        tables=(
            ...
        ),
        sink_method=qc.SinkToFlightPath(
            flight_path="org1/tenant1/tpch/reports/4.result", skip_if_exists=True
        ),
    ),
    explain_parameters={
        "max_sample_size": 1000,
        "method": "OpenAIWithRAG",
    },
    gpt_cache=True
)
```

AI and Analytics Lake Architecture



Try it out

Live app for fun

Blueprint repository

Meet with us

- Meet the team
- Get a demo
- Pick up swag

Set time with us

Visit our booth - ####

Admins Wish For ...

Consistent and meaningful metrics

Flexible deployment

Easy (auto)scaling

Base

Node - Arrow Flight server

Modules - config for loading modules to nodes

Shared features, e.g. requests cancellation

Tooling - CLI, operations(metrics)

Deployment options - bare metal, VM, K8S

Clients - Python, Java(Kotlin)