DAG lakehouse planning with an ephemeral and embedded graph database

Luca Bigon
Bauplan Labs
luca.bigon@bauplanlabs.com

Jacopo Tagliabue
Bauplan Labs
jacopo.tagliabue@bauplanlabs.com

Semih Salihoğlu
Kùzu Inc.
semih@kuzudb.com

```python
@bauplan.model()
@bauplan.python("3.10", pip={"pandas": "2.0"})
def cleaned_data(
    # reference to its parent DAG node
    data=bauplan.Model(
        "raw_data",
        columns=["c1", "c2", "c3"],
        filter="eventTime BETWEEN 2023-01-01 AND 2023-02-01"
    )
):
   # the body returns a dataframe after transformations
   return data.do_something()

@bauplan.model()
@bauplan.python("3.11", pip={"pandas": "1.5"})
def final_data(
    data=bauplan.Model("cleaned_data")
):
   return data.do_something()
```

Figure 1: A two nodes DAG in Bauplan.

ABSTRACT

Bauplan is a code-first lakehouse built by vertically integrating modular data components through APIs – catalog, I/O [3], runtime, Flight server etc. [5]. To shield users from the underlying complexity, Bauplan provides a declarative functional framework to express multi-language data pipelines over Iceberg tables (Fig. 1). The planner is a module taking as input user code, and producing a *logical plan* with the DAG topology (Fig. 2, top). The planner then maps declarative user instructions to platform operations, finalizing the *physical plan* (Fig. 2, bottom) needed by workers [4].

Characteristically, the planner needs to perform static inferences over DAGs (with opaque nodes). Similar to database planners, Bauplan's planner combines filters for efficient I/O scans, and validates column matching of adjacent nodes. Similar to FaaS planners, it unifies Python packages along the transitive dependency graph, and infers function ordering from their signature. *We present a graph-based planning module that uses an embedded graph database management system (GDBMS) – Kùzu [1] – in a novel way*:

1. User code is parsed and inserted into an ephemeral graph database in Kùzu, which represents both data and runtime entities.
2. We execute static checks and planning steps using Cypher queries (e.g. do all children functions have a parent?).
3. The final graph is serialized into Protobuf for execution by downstream workers, and then destroyed.

Our initial planner was a home-grown Python library with recursion for inference and static checks, which was both slow and error-prone. Instead, following the philosophy of composable data systems [2], we chose to utilize a GDBMS that gave us: (i) a high-level query language, simplifying our inference through recursive queries; (ii) optimized query execution leveraging multi-core hardware.

Given the on-demand nature of our workloads, we wish to move the embedded GDBMS in memory, further simplifying our infrastructure and speeding up queries. In collaboration with the Kùzu team, we developed an in-memory version of their database, so that we could leverage a new, ephemeral graph at every run: as a result, we currently create tens of thousands of ephemeral graph databases on-the-fly per day. The in-memory version provided optimized inference without infrastructure dependencies, updates to our build system, or changes in the life-cycle of user requests. Today, a single request may involve >500 Cypher statements, which are all executed with sub-second latency.

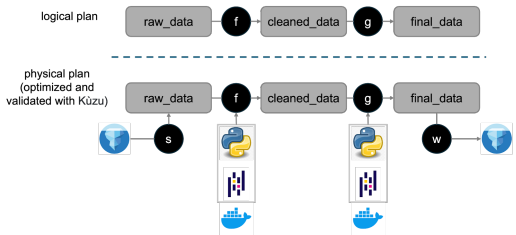

Figure 2: The logical plan is created by parsing user code, the physical plan is obtained running Cypher on Kùzu.

Our planner achieved a 20x speedup over the original solution, with composability also improving engineering efficiency and debuggability [2]: since the DAG plan is now expressed in a language-agnostic representation, it can be dumped, inspected, tested and visualized without depending on the rest of the distributed system. While our planning needs are lakehouse-oriented, we believe our solution to be of broader interest since graphs are a natural representation for many states in data systems.

VLDB Workshop Reference Format:
Luca Bigon, Jacopo Tagliabue, and Semih Salihoğlu. DAG lakehouse planning with an ephemeral and embedded graph database. VLDB 2025 Workshop: Third International Workshop on Composable Data Management Systems.

REFERENCES

[1] Xiyang Feng, Guodong Jin, Ziyi Chen, Chang Liu, and Semih Salihoğlu. 2023. Kùzu Graph Database Management System. In *The Conference on Innovative Data Systems Research*.
[2] Pedro Pedreira, Orri Erling, Konstantinos Karanasos, Scott Schneider, Wes McKinney, Satya R Valluri, Mohamed Zait, and Jacques Nadeau. 2023. The Composable Data Management System Manifesto. *PVLDB* 16, 10 (June 2023), 2679–2685.
[3] Jacopo Tagliabue Ryan Curtin. 2025. The Deconstructed Warehouse: An Ephemeral Query Engine Design for Apache Iceberg. *Proceedings of Workshops at the 51th International Conference on Very Large Data Bases* (2025).
[4] Jacopo Tagliabue, Tyler Caraza-Harter, and Ciro Greco. 2024. Bauplan: Zero-copy, Scale-up FaaS for Data Pipelines. In *WoSC*.
[5] Jacopo Tagliabue, Ciro Greco, and Luca Bigon. 2023. Building a Serverless Data Lakehouse from Spare Parts. *ArXiv* abs/2308.05368 (2023).

Proceedings of the VLDB Endowment. ISSN 2150-8097.