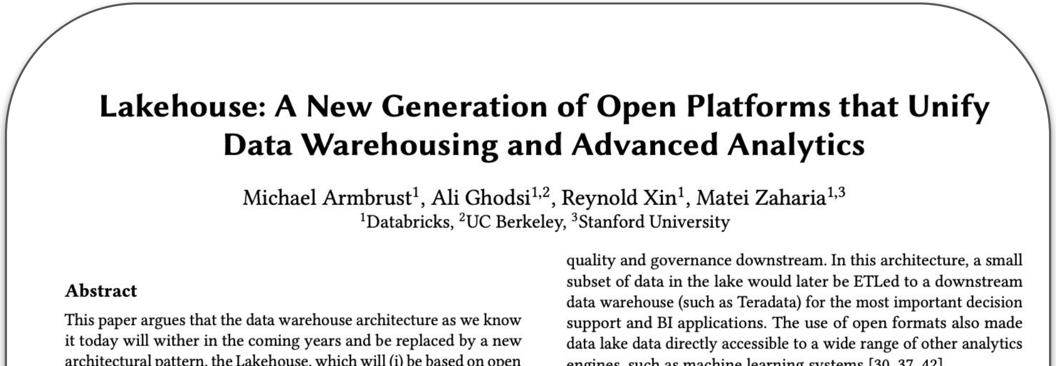
Building a serverless Data Lakehouse from spare parts

Jacopo Tagliabue, Ciro Greco and Luca Bigon

CDMS @ VLDB, Vancouver, 2023

Data Lake + Data Warehouse = Data Lakehouse

- **Centralization:** one layer for storage and governance.
- **Flexibility:** ETL, BI, data science, ML (Python+SQL)
- **Democratization:** lower the entry bar to do data work.



Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics

Michael Armbrust[1], Ali Ghodsi[1,2], Reynold Xin[1], Matei Zaharia[1,3]
[1]Databricks, [2]UC Berkeley, [3]Stanford University

Abstract

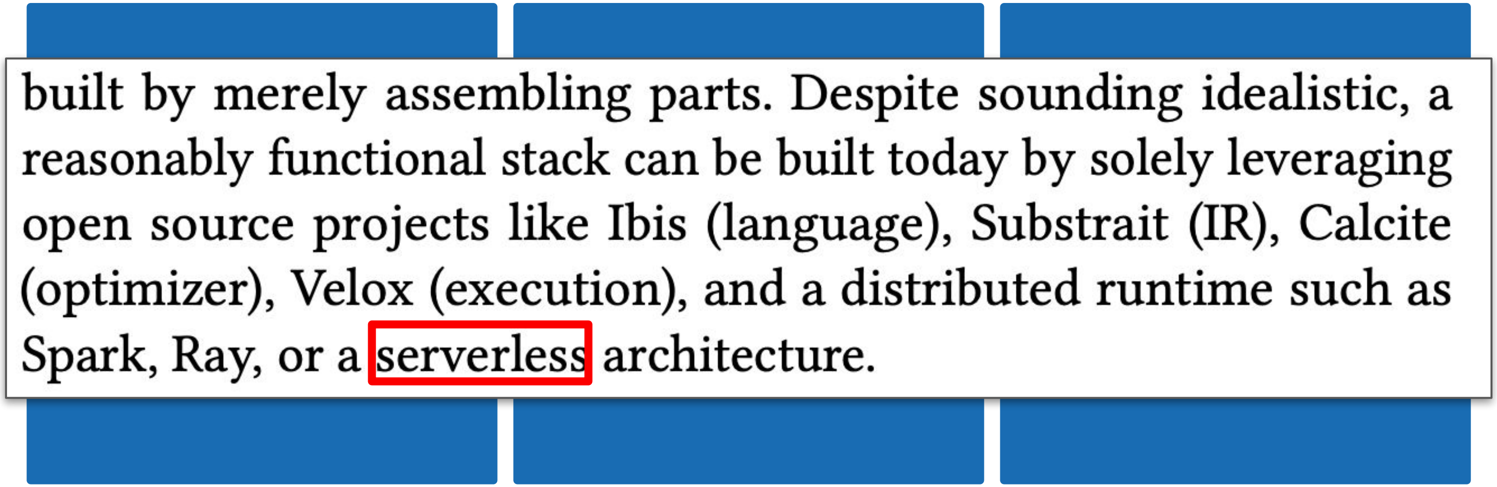
This paper argues that the data warehouse architecture as we know it today will wither in the coming years and be replaced by a new architectural pattern, the Lakehouse, which will (i) be based on open

quality and governance downstream. In this architecture, a small subset of data in the lake would later be ETLed to a downstream data warehouse (such as Teradata) for the most important decision support and BI applications. The use of open formats also made data lake data directly accessible to a wide range of other analytics engines, such as machine learning systems [30, 37, 42]

A Lakehouse is more than the sum of its parts

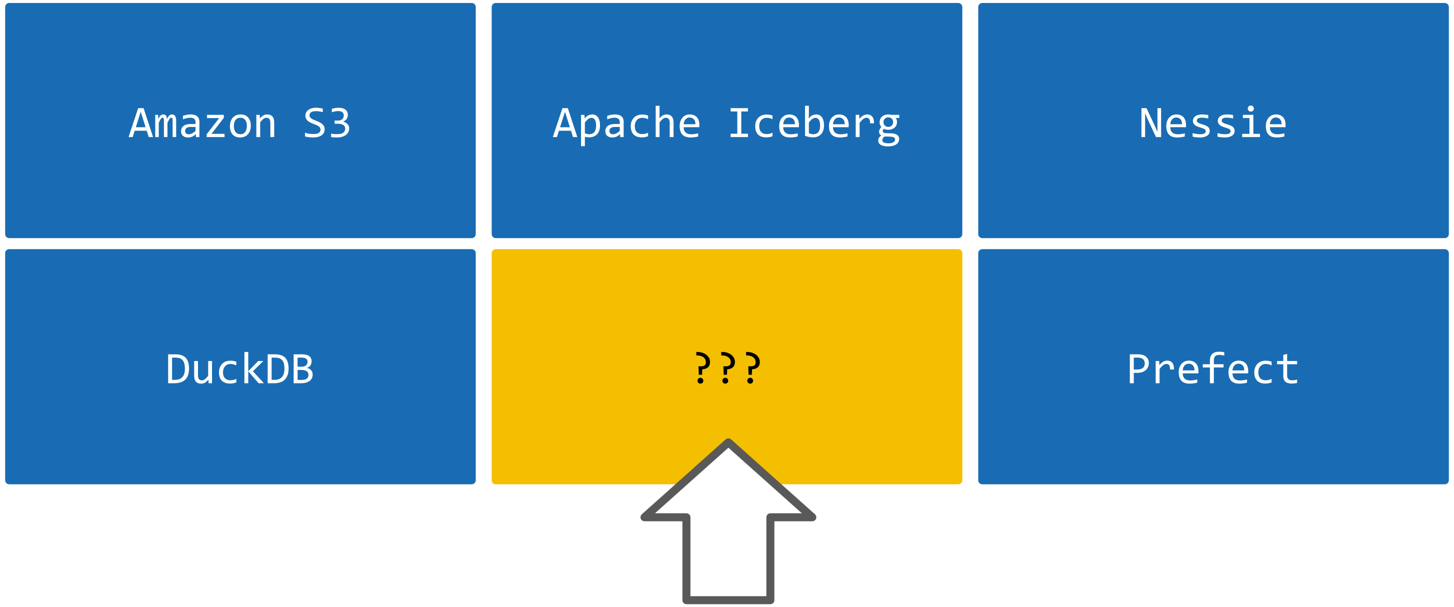
Object storage	Table format	Data catalog
SQL engine	Runtime	Orchestrator

A Lakehouse is more than the sum of its parts



built by merely assembling parts. Despite sounding idealistic, a reasonably functional stack can be built today by solely leveraging open source projects like Ibis (language), Substrait (IR), Calcite (optimizer), Velox (execution), and a distributed runtime such as Spark, Ray, or a serverless architecture.

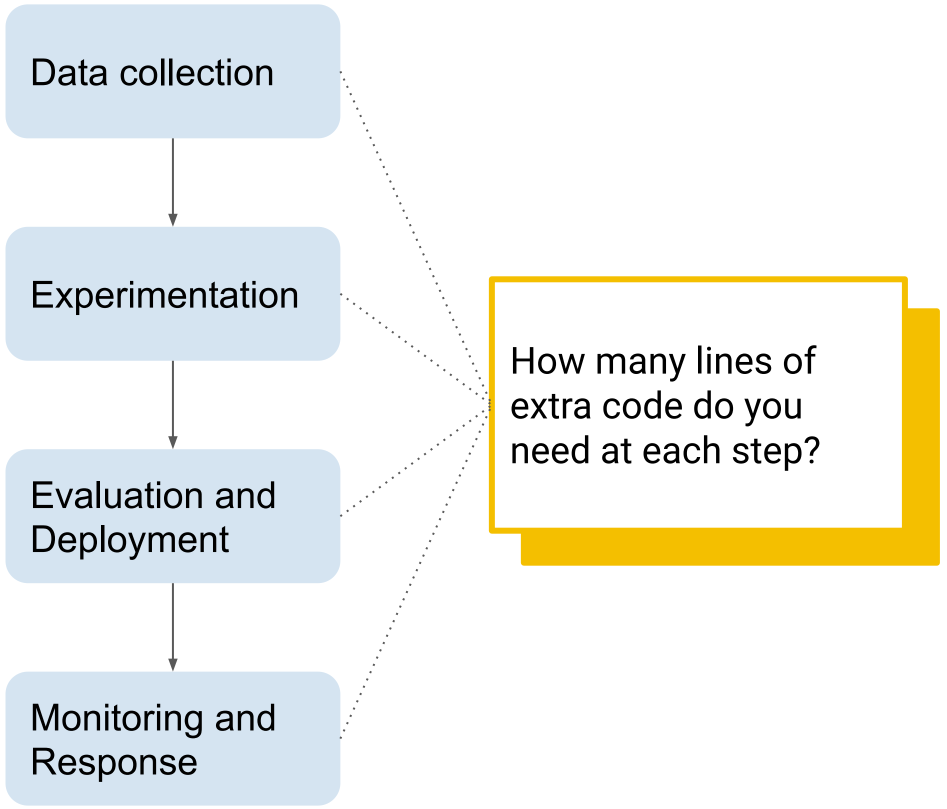
Building a Lakehouse from spare parts…except for one



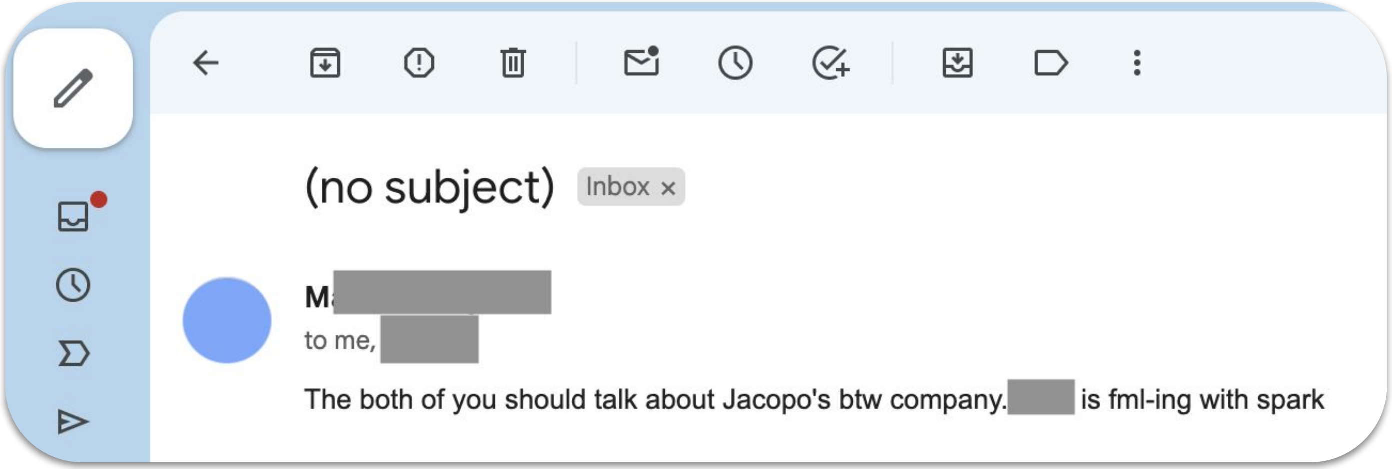
Goal #1: minimize infrastructure

- Terrible use of data scientists' time/skills.
- Unnecessarily long production cycles.

👇 Data development cycle


Data collection
Experimentation
Evaluation and Deployment
Monitoring and Response
How many lines of extra code do you need at each step?

#1: Minimize infrastructure


(no subject)

M[redacted]
to me, [redacted]

The both of you should talk about Jacopo's btw company.[redacted] is fml-ing with spark

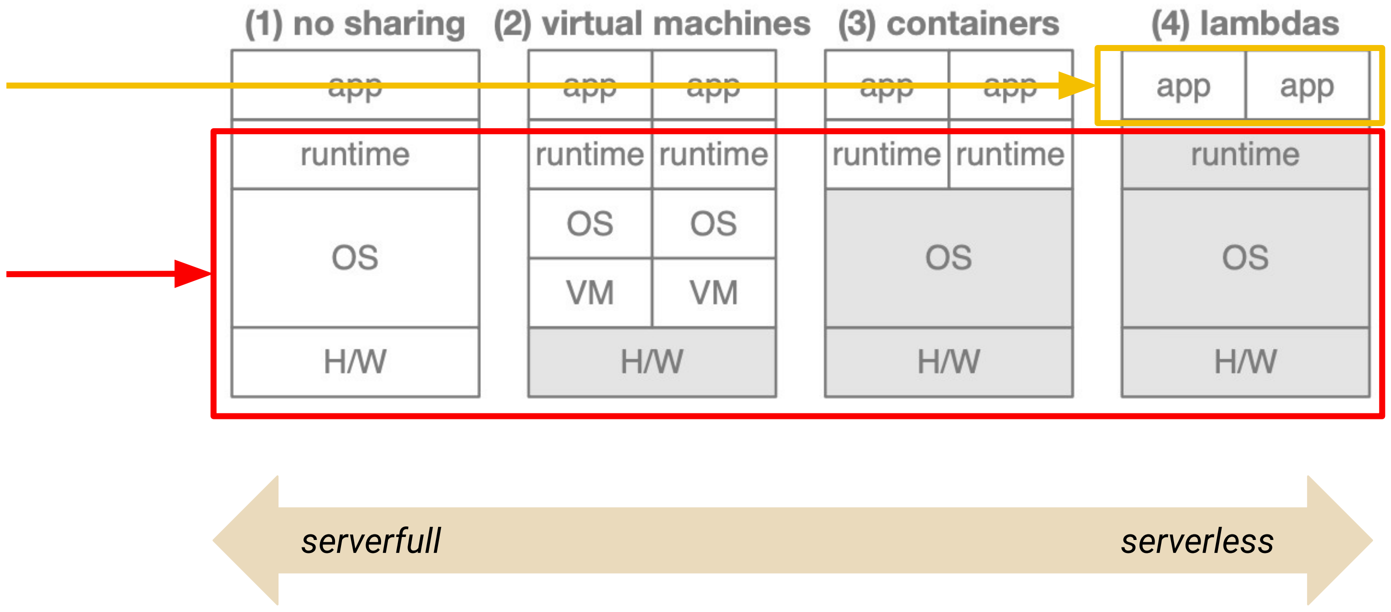
*“FML-ING WITH SPARK”**

** Tagliabue (2023), personal communication*

#1: Minimize infrastructure

We care about this

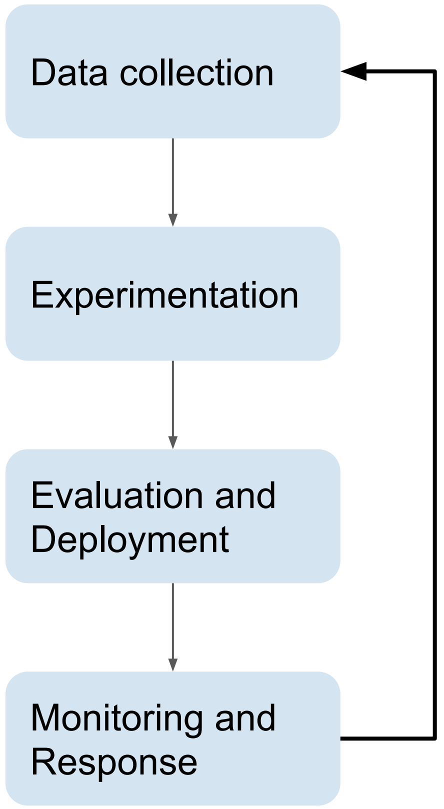
We do **NOT** care about this



Goal #2: minimize loop time

- Data development requires to loop **over production data**. ≠ backend or frontend development.

👇 Data development cycle



How long do you wait to do a full loop?

#2: Loop time (a true story)

t_1: I have a function doing data sciency-y stuff.

```
def handler(event, context):
    start = time.time()
    import pandas as pd
    # DATA SCIENCE HERE
    return {
        "metadata": {
            "eventId":
            str(uuid.uuid4()),
            "time_in_ms":
            int((time.time() - start) *
            1000.0)
        },
        "versions": {
            'pandas': pd.__version__
        }
    }
```

#2: Loop time (a true story)

t$_1$: I have a function doing data sciency-y stuff.

t$_2$: I realize I need to do some scraping with Selenium.

```
def handler(event, context):
        start = time.time()
        import pandas as pd
        # DATA SCIENCE HERE
        return {
                "metadata": {
                        "eventId":
                        str(uuid.uuid4()),
                        "time_in_ms":
                        int((time.time() - start) *
                        1000.0)
                },
                "versions": {
                        'pandas': pd.__version__
                }
        }
```

#2: Loop time (a true story)

t$_1$: I have a function doing data sciency-y stuff.

t$_2$: I realize I need to do some scraping with Selenium.

t$_3$: I want to run my function with the new package.

```
def handler(event, context):
    start = time.time()
    import pandas as pd
    # DATA SCIENCE HERE
    return {
        "metadata": {
            "eventId":
            str(uuid.uuid4()),
            "time_in_ms":
            int((time.time() - start) *
            1000.0)
        },
        "versions": {
            'pandas': pd.__version__
        }
    }
```

#2: Loop time (a true story)

AWS Lambda

1. Update requirements.txt
2. CLI: `serverless deploy`
   a. Update container
   b. Update ECR
   c. Update Cloud formation
3. Invoke the function

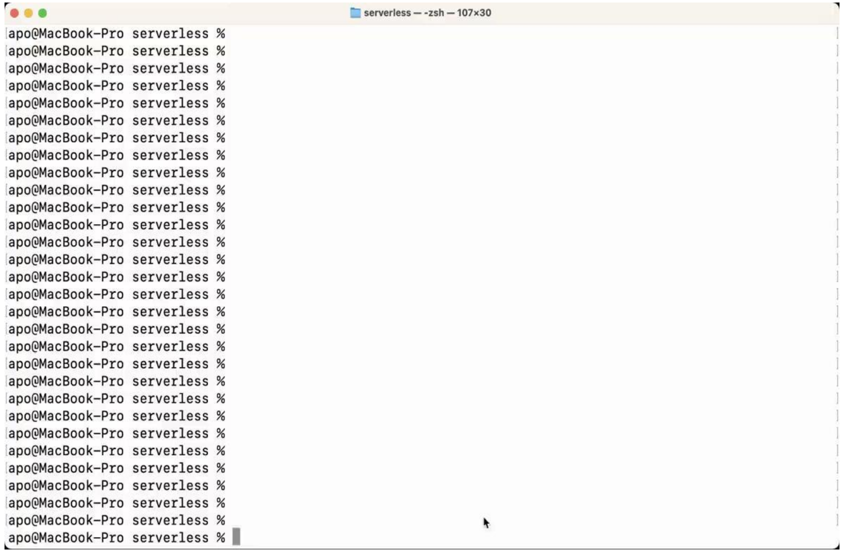
Bauplan serverless

1. Update the function
2. CLI: `bauplan run`
   a. Connect to cloud
   b. Build function
3. Invoke the function

#2: Loop time (a true story)

AWS Lambda

Bauplan serverless

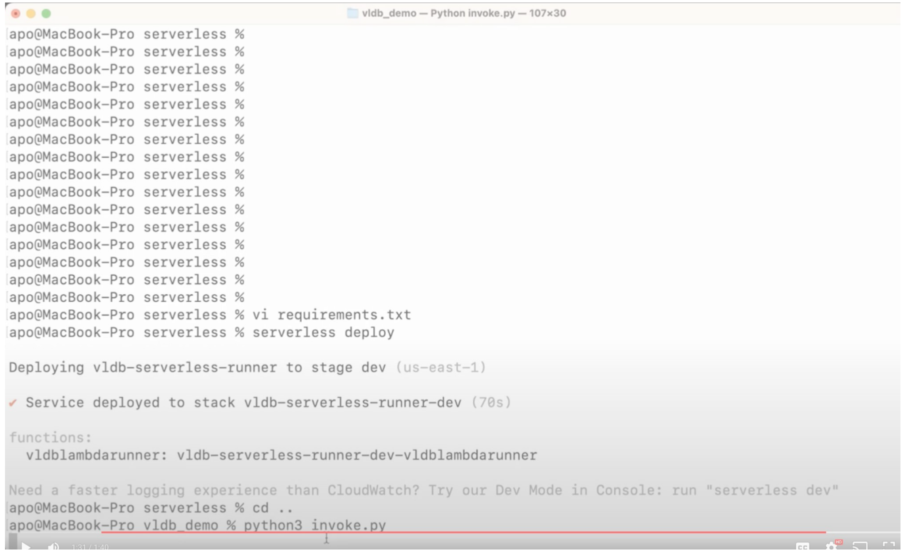


```
serverless — -zsh — 107×30
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
```

Feedback loop: 70s

#2: Loop time (a true story)

AWS Lambda

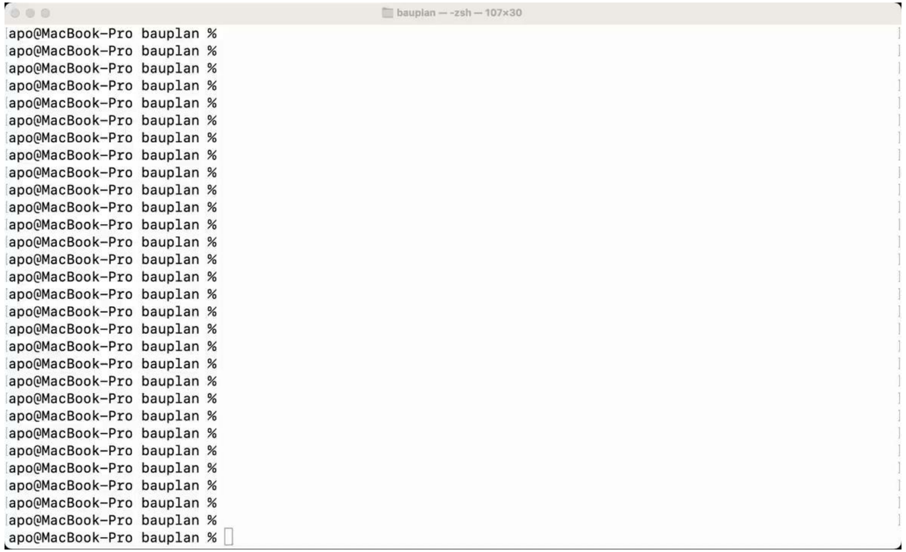


vldb_demo — Python invoke.py — 107×30

```
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless %
apo@MacBook-Pro serverless % vi requirements.txt
apo@MacBook-Pro serverless % serverless deploy

Deploying vldb-serverless-runner to stage dev (us-east-1)

✔ Service deployed to stack vldb-serverless-runner-dev (70s)

functions:
  vldblambdarunner: vldb-serverless-runner-dev-vldblambdarunner

Need a faster logging experience than CloudWatch? Try our Dev Mode in Console: run "serverless dev"
apo@MacBook-Pro serverless % cd ..
apo@MacBook-Pro vldb_demo % python3 invoke.py
```

Feedback loop: 70s

Bauplan serverless



bauplan — -zsh — 107×30

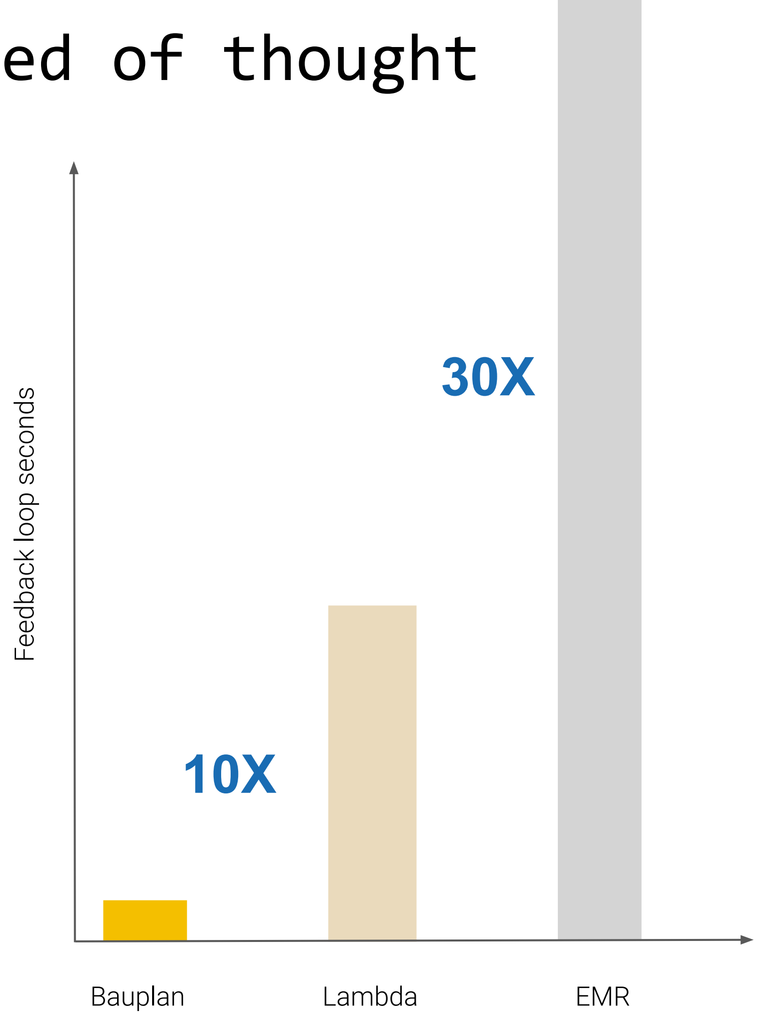
```
apo@MacBook-Pro bauplan %
apo@MacBook-Pro bauplan %
apo@MacBook-Pro bauplan %
apo@MacBook-Pro bauplan %
apo@MacBook-Pro bauplan %
apo@MacBook-Pro bauplan %
apo@MacBook-Pro bauplan %
apo@MacBook-Pro bauplan %
apo@MacBook-Pro bauplan %
apo@MacBook-Pro bauplan %
apo@MacBook-Pro bauplan %
apo@MacBook-Pro bauplan %
apo@MacBook-Pro bauplan %
apo@MacBook-Pro bauplan %
apo@MacBook-Pro bauplan %
apo@MacBook-Pro bauplan %
apo@MacBook-Pro bauplan %
apo@MacBook-Pro bauplan %
apo@MacBook-Pro bauplan %
apo@MacBook-Pro bauplan %
apo@MacBook-Pro bauplan %
apo@MacBook-Pro bauplan %
apo@MacBook-Pro bauplan %
apo@MacBook-Pro bauplan %
apo@MacBook-Pro bauplan %
apo@MacBook-Pro bauplan %
apo@MacBook-Pro bauplan %
apo@MacBook-Pro bauplan %
apo@MacBook-Pro bauplan %
apo@MacBook-Pro bauplan %
```

Feedback loop: 7s

Data development at the speed of thought

- No Docker build.
- No registry upload.
- Company-wide smart cache.

Faster than local!



Open source to the rescue?

OS serverless is built around micro-services use cases:

- many small, concurrent functions;
- full isolation;
- small latency, small individual throughput.

OpenWhisk

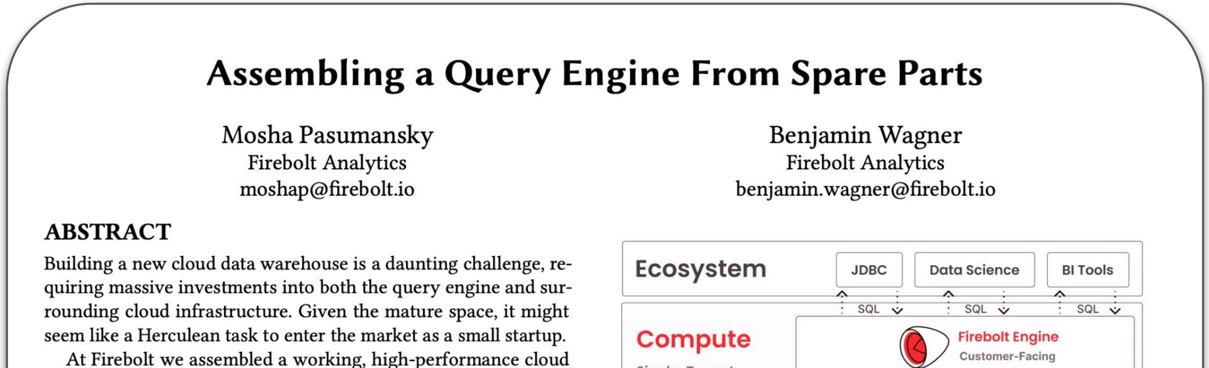
OpenFaas

OpenLambda

Open source to the rescue?

We need:

- heterogenous functions;
- runtime isolation, but data sharing;
- medium latency, very high individual throughput.



Assembling a Query Engine From Spare Parts

Mosha Pasumansky
Firebolt Analytics
moshap@firebolt.io

Benjamin Wagner
Firebolt Analytics
benjamin.wagner@firebolt.io

ABSTRACT

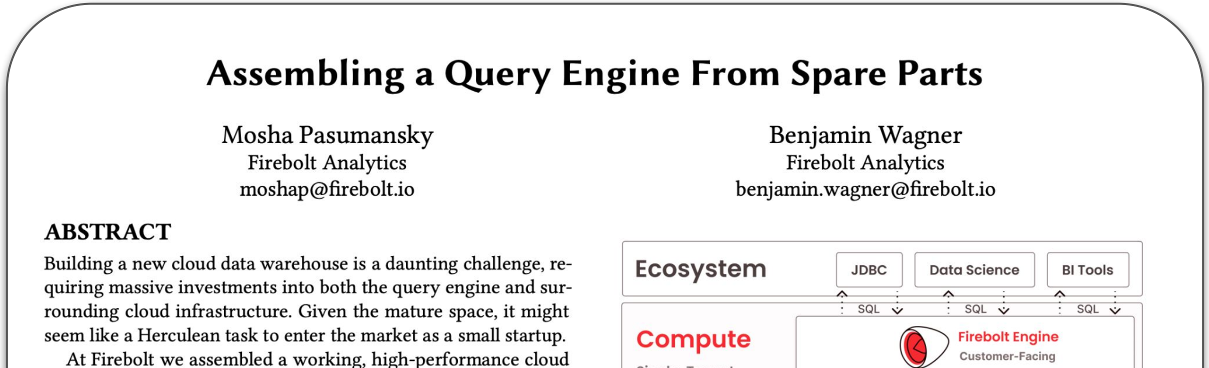
Building a new cloud data warehouse is a daunting challenge, requiring massive investments into both the query engine and surrounding cloud infrastructure. Given the mature space, it might seem like a Herculean task to enter the market as a small startup.
At Firebolt we assembled a working, high-performance cloud

Open source to the rescue?

We need:

- heterogenous functions;
- runtime isolation, but data sharing;
- medium latency, very high individual throughput

Invest in differentiating features, assemble the rest from “spare parts”!



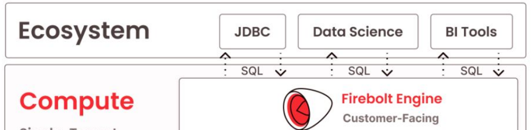
Assembling a Query Engine From Spare Parts

Mosha Pasumansky
Firebolt Analytics
moshap@firebolt.io

Benjamin Wagner
Firebolt Analytics
benjamin.wagner@firebolt.io

ABSTRACT

Building a new cloud data warehouse is a daunting challenge, requiring massive investments into both the query engine and surrounding cloud infrastructure. Given the mature space, it might seem like a Herculean task to enter the market as a small startup.
At Firebolt we assembled a working, high-performance cloud



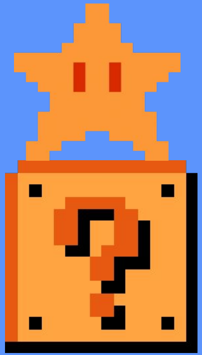
Programs must be written for people to read, and only incidentally for machines to execute

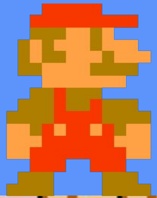
- H. Abelson

Pipelines must be written for people to read, and only incidentally for cloud to execute

- Bauplan

Want to stay up-to-date, collaborate or just chat? Reach out or check bauplanlabs.com!





BAUPLAN