Building a Data Platform on Apache Iceberg and Nessie

Jacopo Tagliabue
Gnarly Data Waves

Us and them

Apo

Ciro

Mattia

- Serial entrepreneurs: from inception to IPO through acquisition (founded Tooso in 2017, joined Coveo in 2019, IPO 2021).
- Led AI and MLOps at Coveo: 30+ research contributions, including Nature, NeurIPS, KDD, SIGIR, RecSys. ~2000 GitHub stars and >1M downloads with Open Source projects.

Backed by

innovation endeavors

South Park Commons

And by founders and executives at

aws

docker

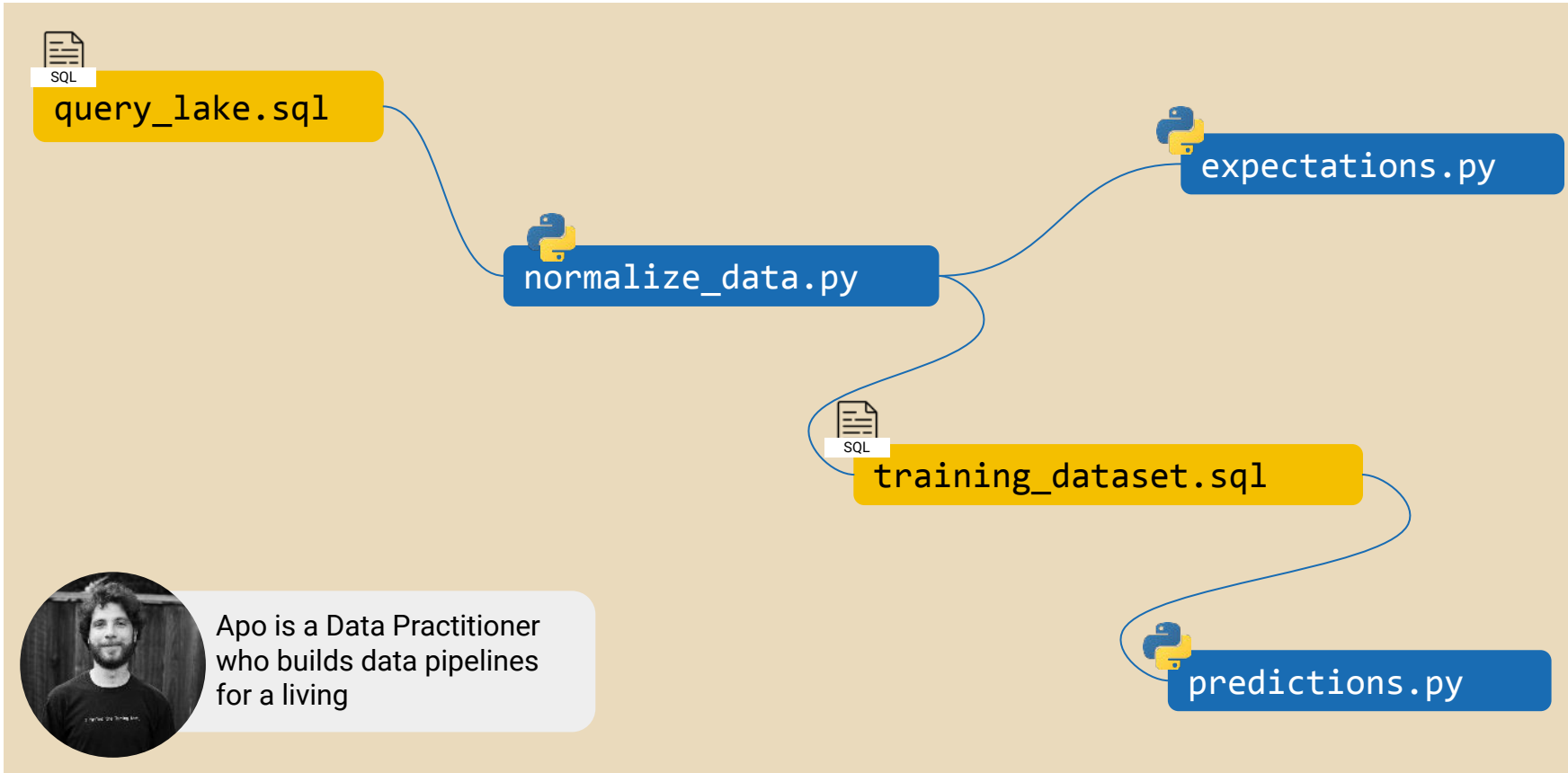
redis

VOLTRON DATA

CLOUDERA

Bauplan 101

A day in the life



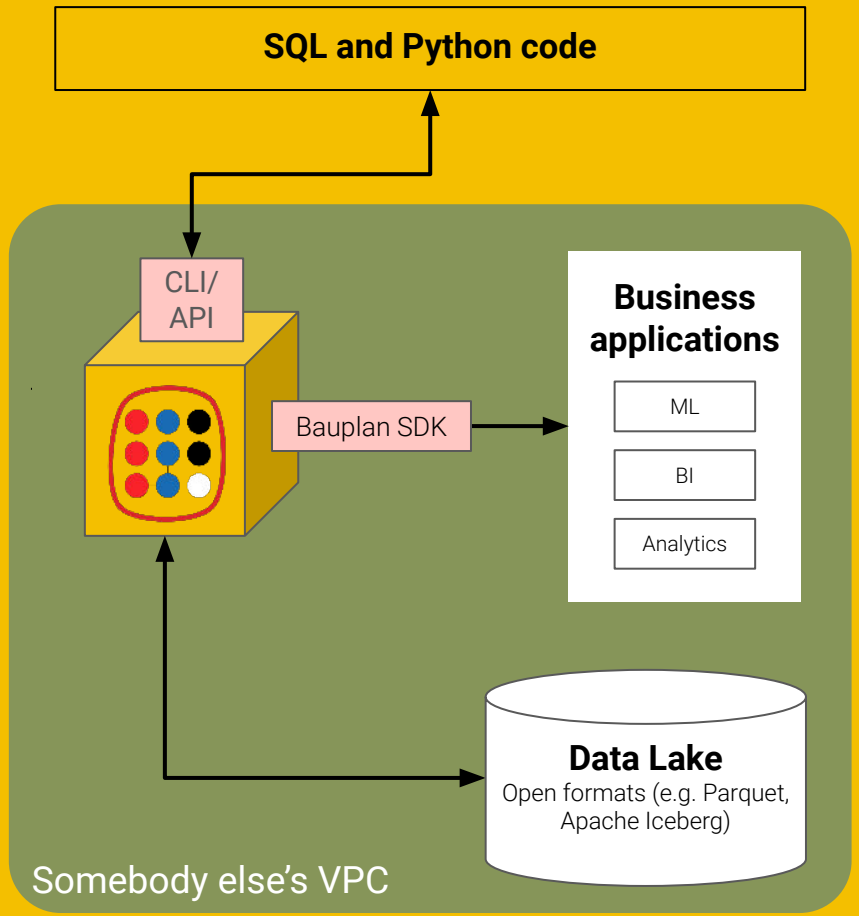
> Programs must be written for people to read, and only incidentally for machines to execute.
>
> H. Abelson

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

BAUPLAN

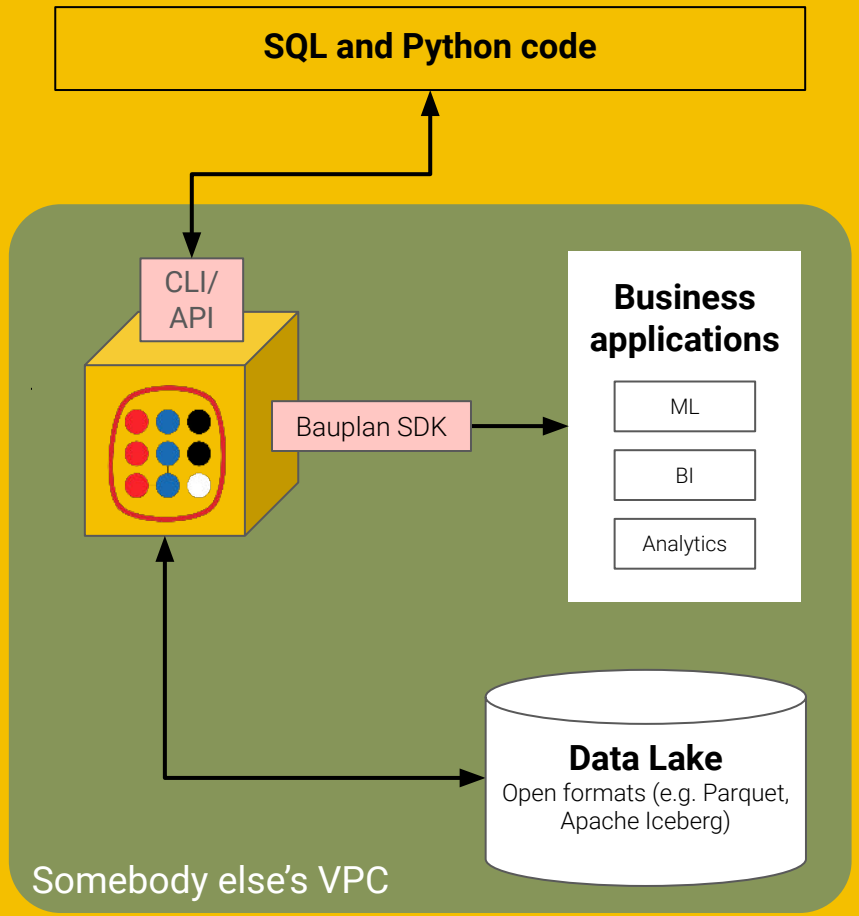
Serverless computing platform for data transformation pipelines.

We focus on mixed-language tabular transformations over data lakes.



BAUPLAN

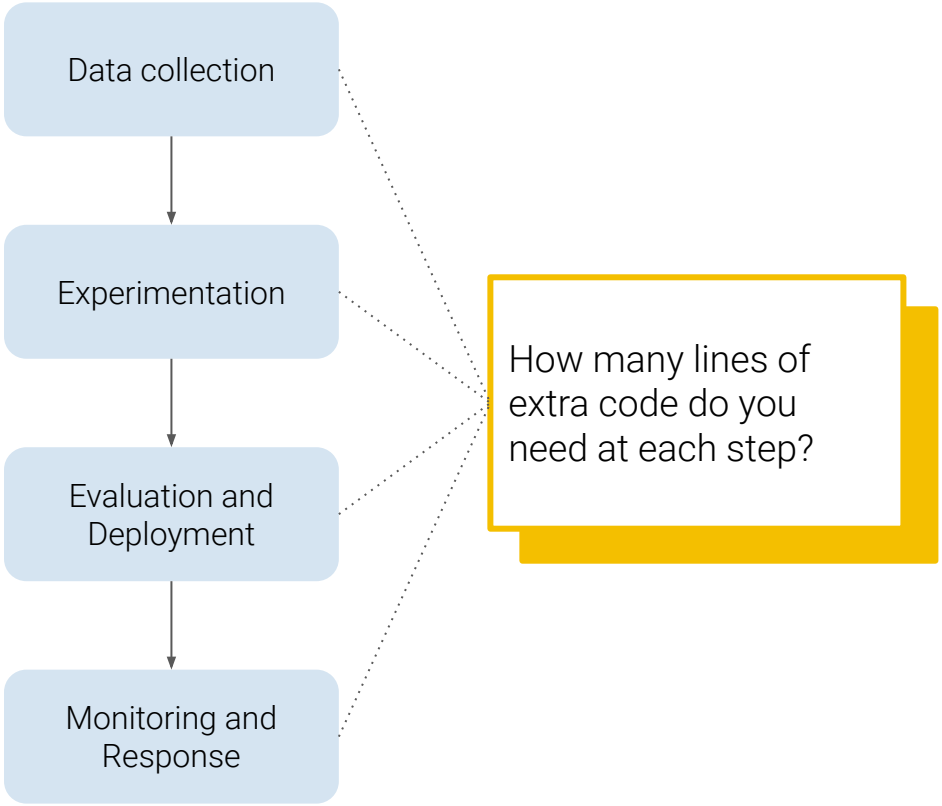
- SQL + Python.
- Tables, not files!
- DAGs have many nodes that need to be reused.
- Don’t move your data, we come to you.



Goal #1

Minimize infrastructure

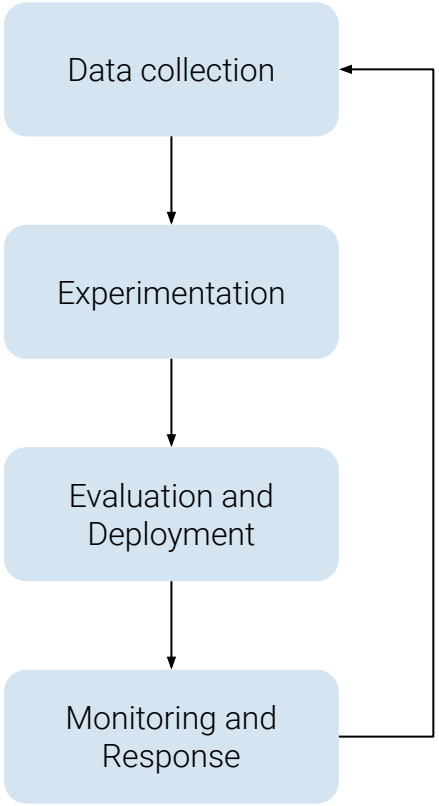
👇 Data development cycle



Goal #2

Reduce the feedback loop time

👇 Data development cycle



How long do you wait to do a full loop?

How do we move with production data fast while *not* breaking things?

From lake to lakehouse

People don’t think in files

```
file schema:    hive_schema
--------------------------------------------------------------------------------
advertiser_nbr: OPTIONAL BINARY O:UTF8 R:0 D:1
order_nbr:      OPTIONAL BINARY O:UTF8 R:0 D:1
creative_nbr:   OPTIONAL BINARY O:UTF8 R:0 D:1
creative_sz:    OPTIONAL BINARY O:UTF8 R:0 D:1
make:           OPTIONAL BINARY O:UTF8 R:0 D:1
model:          OPTIONAL BINARY O:UTF8 R:0 D:1

row group 1:    RC:18610100 TS:138756396
--------------------------------------------------------------------------------
advertiser_nbr:  BINARY SNAPPY DO:0 FPO:4 SZ:19022560/24391353/1.28 VC:18610100 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
order_nbr:       BINARY SNAPPY DO:0 FPO:19022564 SZ:25490086/26539053/1.04 VC:18610100 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
creative_nbr:    BINARY SNAPPY DO:0 FPO:44512650 SZ:31361154/32545183/1.04 VC:18610100 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
creative_sz:     BINARY SNAPPY DO:0 FPO:75873804 SZ:16105784/16316166/1.01 VC:18610100 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
make:            BINARY SNAPPY DO:0 FPO:91979588 SZ:15595856/15737301/1.01 VC:18610100 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
model:           BINARY SNAPPY DO:0 FPO:107575444 SZ:23178066/23227340/1.00 VC:18610100 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE

row group 2:    RC:18890100 TS:140042021
--------------------------------------------------------------------------------
advertiser_nbr:  BINARY SNAPPY DO:0 FPO:134217728 SZ:18478358/24001580/1.30 VC:18890100 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
order_nbr:       BINARY SNAPPY DO:0 FPO:152696086 SZ:25055736/26174098/1.04 VC:18890100 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
creative_nbr:    BINARY SNAPPY DO:0 FPO:177751822 SZ:31236535/33058784/1.06 VC:18890100 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
creative_sz:     BINARY SNAPPY DO:0 FPO:208988357 SZ:16114299/16548274/1.03 VC:18890100 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
make:            BINARY SNAPPY DO:0 FPO:225102656 SZ:15978228/16139637/1.01 VC:18890100 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
model:           BINARY SNAPPY DO:0 FPO:241080884 SZ:24063766/24119648/1.00 VC:18890100 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE

row group 3:    RC:2042113 TS:14236969
--------------------------------------------------------------------------------
advertiser_nbr:  BINARY SNAPPY DO:0 FPO:268435456 SZ:1903853/2442711/1.28 VC:2042113 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
order_nbr:       BINARY SNAPPY DO:0 FPO:270339309 SZ:2496674/2516632/1.01 VC:2042113 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
creative_nbr:    BINARY SNAPPY DO:0 FPO:272835983 SZ:3155336/3447472/1.09 VC:2042113 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
creative_sz:     BINARY SNAPPY DO:0 FPO:275991319 SZ:1696448/1753627/1.03 VC:2042113 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
make:            BINARY SNAPPY DO:0 FPO:277687767 SZ:1619586/1623939/1.00 VC:2042113 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
model:           BINARY SNAPPY DO:0 FPO:279307353 SZ:2424547/2452588/1.01 VC:2042113 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
```

```
##fileformat=
##fileDate=20
##source=myIm
##reference=f
##contig=<ID=
##phasing=par
##INFO=<ID=NS
##INFO=<ID=DP
##INFO=<ID=AF
##INFO=<ID=AA
##INFO=<ID=DB
##INFO=<ID=H2
##FILTER=<ID=
##FILTER=<ID=
##FORMAT=<ID=
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Read Depth">
##FORMAT=<ID=HQ,Number=2,Type=Integer,Description="Haplotype Quality">
```

Fixed fields

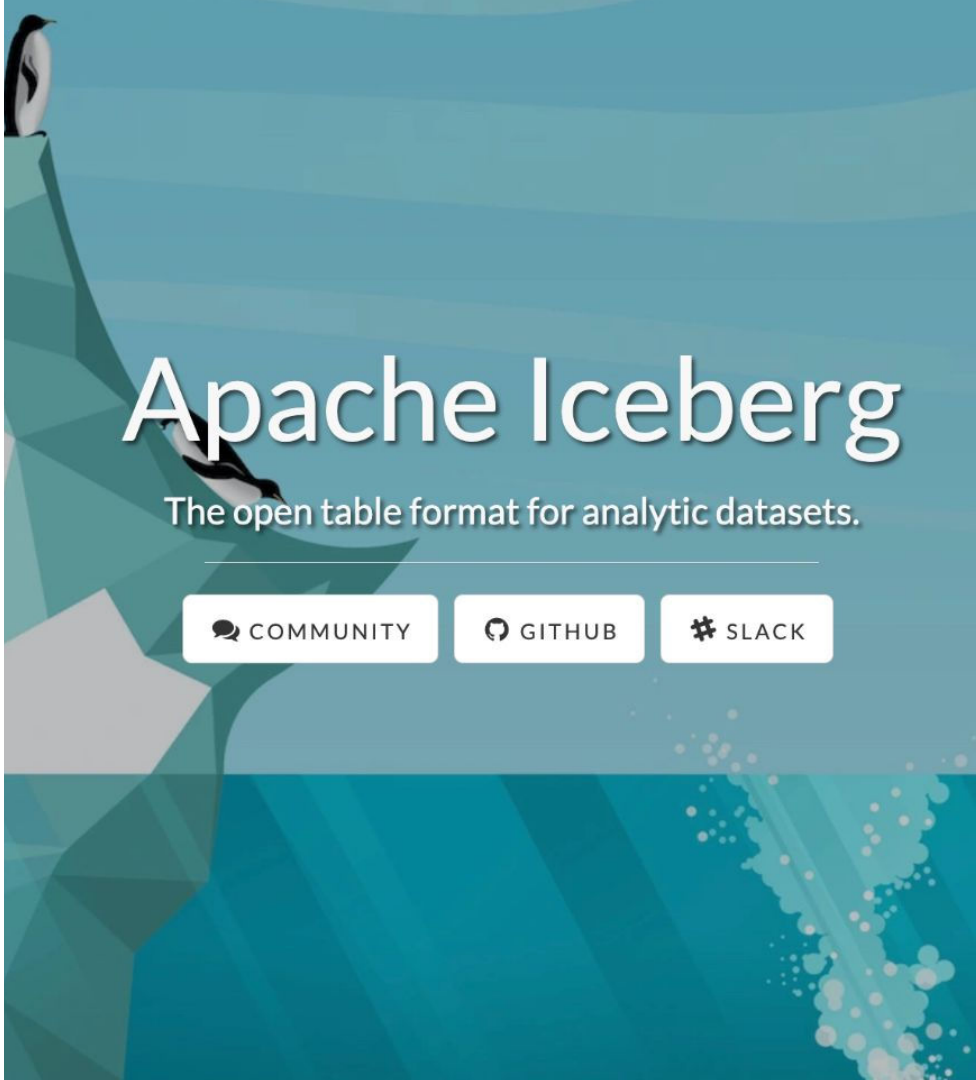
Optional: FORMAT field specifying data type + Per-sample genotype data

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO	FORMAT	NA00001	NA00002	NA00003
20	14370	rs6054257	G	A	29	PASS	NS=3;DP=14;AF=0.5;DB;H2	GT:GQ:DP:HQ	0|0:48:1:51,51	1|0:48:8:51,51	1/1:43:5:.,.
20	17330	.	T	A	3	q10	NS=3;DP=11;AF=0.017	GT:GQ:DP:HQ	0|0:49:3:58,50	0|1:3:5:65,3	0/0:41:3
20	1110696	rs6040355	A	G,T	67	PASS	NS=2;DP=10;AF=0.333,0.667;AA=T;DB	GT:GQ:DP:HQ	1|2:21:6:23,27	2|1:2:0:18,2	2/2:35:4
20	1230237	.	T	.	47	PASS	NS=3;DP=13;AA=T	GT:GQ:DP:HQ	0|0:54:7:56,60	0|0:48:4:51,51	0/0:61:2
20	1234567	microsat1	GTC	G,GTCT	50	PASS	NS=3;DP=9;AA=G	GT:GQ:DP	0/1:35:4	0/2:17:2	1/1:40:3

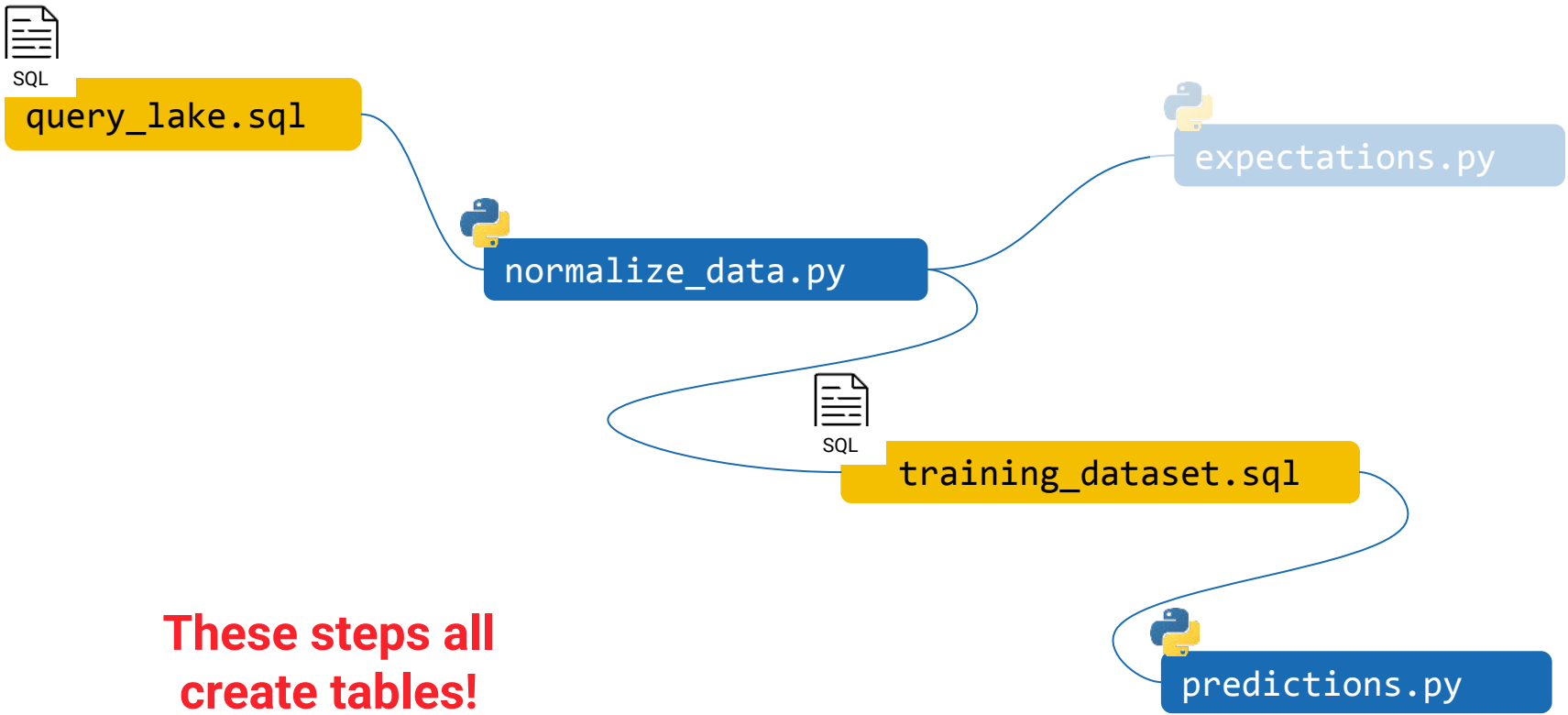
They think in tables

We use Iceberg as our open format

- Large user base.
- Large contribution base.
- Increasing Python compatibility.



From single tables to DAGs

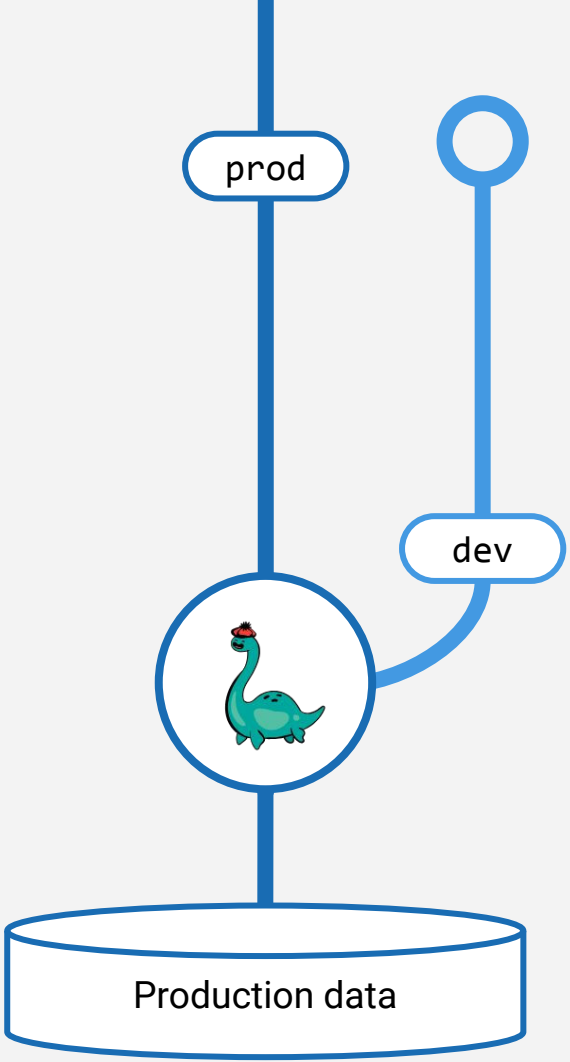


These steps all create tables!

Iterating on a DAG

We use Nessie for DAG git-semantics.

- Version DAGs, not tables.
- Work on production data.
- Move fast, but please don't break things!



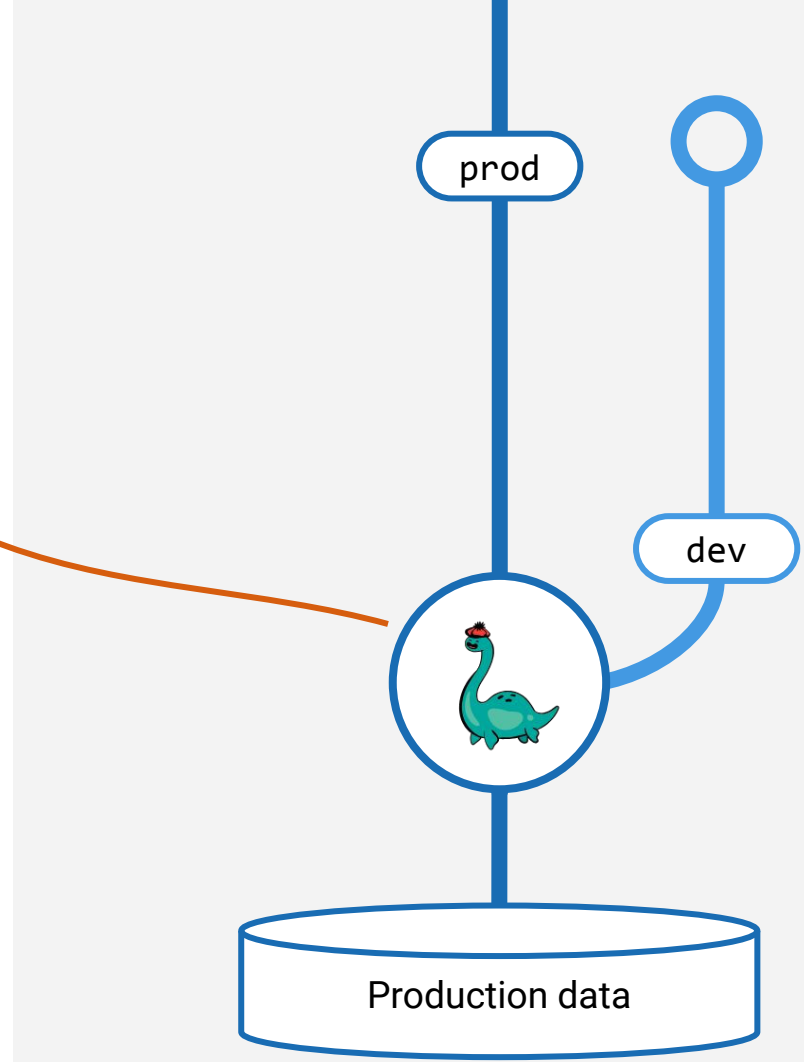
Debugging on a DAG

Versioning

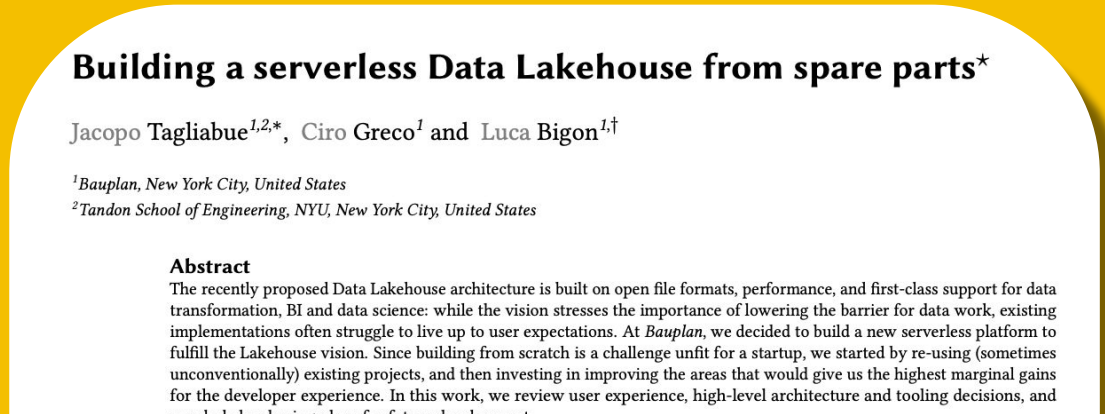
Code + Data

```
$ bauplan run --run=89186b10-6d2b-11ee-b962-0242ac120002

Running run 89186b10-6d2b-11ee-b962-0242ac120002 ...
└── booting
──────────────────────────────────────────────────────
──────── 100%
    └── table-1
──────────────────────────────────────────────────────
── 100%
       └── table-2
```



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



Building a serverless Data Lakehouse from spare parts*

Jacopo Tagliabue[1,2,*], Ciro Greco[1] and Luca Bigon[1,†]

[1]Bauplan, New York City, United States
[2]Tandon School of Engineering, NYU, New York City, United States

Abstract

The recently proposed Data Lakehouse architecture is built on open file formats, performance, and first-class support for data transformation, BI and data science: while the vision stresses the importance of lowering the barrier for data work, existing implementations often struggle to live up to user expectations. At *Bauplan*, we decided to build a new serverless platform to fulfill the Lakehouse vision. Since building from scratch is a challenge unfit for a startup, we started by re-using (sometimes unconventionally) existing projects, and then investing in improving the areas that would give us the highest marginal gains for the developer experience. In this work, we review user experience, high-level architecture and tooling decisions, and

BAUPLAN