CDC patterns in Apache Iceberg

Ryan Blue
Trino Fest – June 2023

Scan for an Iceberg cheat sheet for Spark or Trino

Tabular

Quick refresher

What’s Iceberg?

Iceberg is an **open standard** for tables with **SQL behavior**

What’s CDC?

Change **D**ata **C**apture:
As relational tables are modified, emit an update stream to keep copies in sync—capture changes to tables as they happen


Bernie
SRE TEAM
I am once again asking
you to stop your accidental DOS
attacks on our production database
imgflip.com

Bank example

Bank accounts

- Account ID and balance
- Updated by primary key
- Layout and order configured

Goal: Keep `accounts` up-to-date using incoming transaction data

```
-- example table
CREATE TABLE accounts (
    account_id bigint,
    balance decimal(12, 2))
PARTITIONED BY (
    bucket(4, account_id))

-- set primary key fields
ALTER TABLE accounts
SET IDENTIFIER FIELDS account_id

-- configure write order/distribution
ALTER TABLE accounts
WRITE DISTRIBUTED BY PARTITION
      LOCALLY ORDERED BY account_id
```

Transaction data

Double entry bookkeeping

- Each transfer updates 2 accounts
- Total deposits should not change (transactional consistency)

Transaction source is flexible

- Kafka or kinesis stream
- Upstream table

```
+----------------+-------------+---------+
| transaction_id | account_id  | amount  |
+----------------+-------------+---------+
|              1 |           9 |    -435 |
|              1 |           8 |     435 |
|              2 |           2 |    -863 |
|              2 |           4 |     863 |
|              3 |           6 |    -530 |
|            ... |         ... |     ... |
+----------------+-------------+---------+

-- bank deposits must be reliable!
SELECT
   sum(balance) AS total_deposits
FROM accounts
```

Why is CDC difficult?

Wants

- Direct writes — single table
- Accurate historical record
- Time travel to any point
- Consistent within and across tables
- High volume, low latency
- Read-optimized
- Write-optimized
- Schema evolution

Why is CDC difficult?

Wants

- Direct writes — single table
- Accurate historical record
- Time travel to any point
- Consistent within and across tables
- High volume, low latency
- Read-optimized
- Write-optimized
- Schema evolution

Problems

- Lower latency ⇒ more work
- Write amplification
- Batch writes — frequency
    - Double update problem
    - Transaction alignment/consistency
- Read requirements
    - Equality: delete *id=5*
    - Positional: delete *A.parquet, pos 11*

Storage trade–off

Direct writes

- One table, one write
- Increases write complexity
- Volume limit
- Double update problem

Change log table

- Historical record
- Time travel to any transaction
- Simple append-only writes
- High volume
- No direct reads, not optimized

Most important (and overlooked) decision

Change log pattern

Surprisingly effective with Trino!

- Track only changes
- Efficient writes, expensive reads
- Continuous time travel:
  `WHERE transaction_id < ID`

Tip: Handle UPSERT using SQL windows

```sql
-- store only account changes
CREATE TABLE account_updates (
    transaction_id bigint,
    account_id bigint,
    amount decimal(12, 2))
PARTITIONED BY (
    truncate(100000, transaction_id))

-- compute account value at query time
CREATE VIEW accounts AS
SELECT
    account_id,
    sum(amount) AS balance
FROM account_updates
```

MERGE pattern

- Direct write to an analytic table
- Uses position deletes (reads data!)
- Supports custom logic
  - Count duplicates
  - Consume any source data

```sql
-- squash multiple updates
WITH updates AS (
    SELECT
        account_id,
        sum(amount) AS amount
    FROM transactions
    GROUP BY account_id
)

MERGE INTO accounts a USING updates u
ON a.account_id = u.account_id
WHEN MATCHED THEN UPDATE
    SET a.balance = a.balance + u.amount
```

MERGE strategy trade–off

Lazy *merge-on-read*

- Write only updates
- Low write amplification
- Defer work to read or compaction
- Example table: creates up to 8 files

Supported in Spark and Trino

Eager *copy-on-write*

- Rewrite files as needed
- High write amplification
- Do work at write time for fast reads
- Example table: rewrites up to 4 files

Supported in Spark

Commit frequency trade-off

Faster

- Closer to real time
- Requires more maintenance
- Exacerbates the strategy trade-off!

Slower

- Higher latency for changes
- Reduces conflicts with services

```sql
-- squash multiple updates
WITH updates AS (
    SELECT
        account_id,
        sum(amount) AS amount
    FROM transactions
    GROUP BY account_id
)

MERGE INTO accounts a USING updates u
ON a.account_id = u.account_id
WHEN MATCHED THEN UPDATE
    SET a.balance = a.balance + u.amount
```

Flink UPSERT pattern

Update type trade-off

- Equality update
  - No reading needed
  - Cannot compact deltas
- Positional update
  - Requires locating rows
  - Can conflict with updates

Flink UPSERT is NOT recommended

- Inflexible
- Requires aggressive maintenance
- Doesn’t sort data for efficiency
- Worst pattern in practice

```
stack.pop()
```

Why is CDC difficult?

Wants

- Direct writes — single table
- Accurate historical record
- Time travel to any point
- Consistent within and across tables
- High volume, low latency
- Read-optimized
- Write-optimized
- Schema evolution

Why is CDC difficult?

Wants

- Direct writes — single table
- Accurate historical record
- Time travel to any point
- Consistent within and across tables
- High volume, low latency
- Read-optimized
- Write-optimized
- Schema evolution

Reasons to use the change log pattern

Why is CDC difficult?

Wants

- Direct writes — single table
- Accurate historical record
- Time travel to any point
- Consistent within and across tables
- High volume, low latency
- Read-optimized
- Write-optimized
- Schema evolution

Reasons to use the MERGE pattern

- Use eager rewrites by default (copy-on-write)
- Use lazy rewrites for frequent updates

Hybrid pattern: MERGE + change log

Best of both patterns

- Land updates in change log table
  - Optimized for writes
  - Historical record, time travel
- Periodically MERGE
  - Simple reads
  - Separates concerns
- Optional view for read efficiency
  - Low data latency
  - Infrequent MERGE

Worst of both patterns

- Eager/lazy strategy trade-off
- Commit frequency trade-off
- Complex pipeline

Future work

Branches and tags

- Maintain change log in a branch
- Tag periodic MERGE results
- Use views to apply latest changes

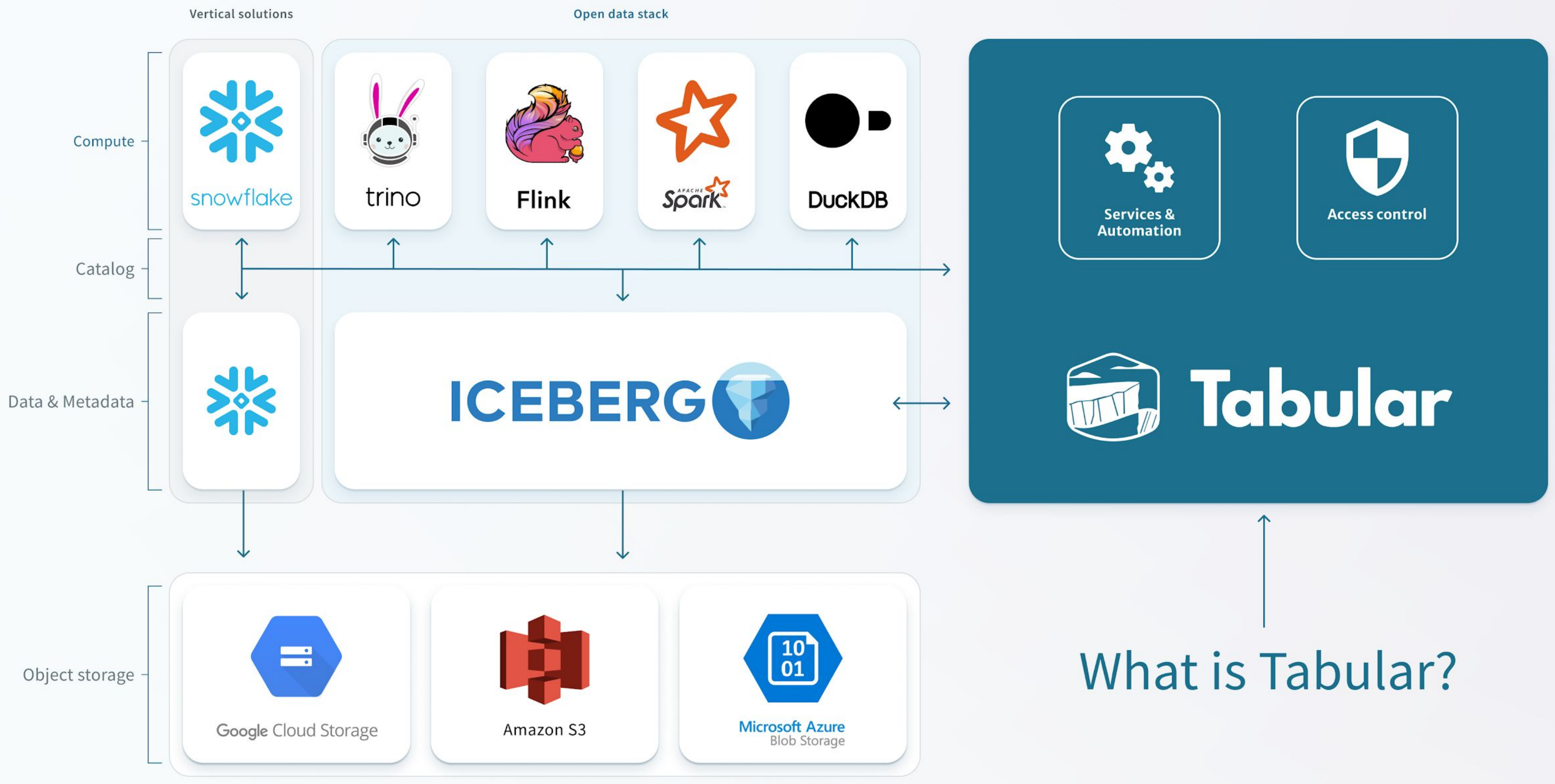
New patterns

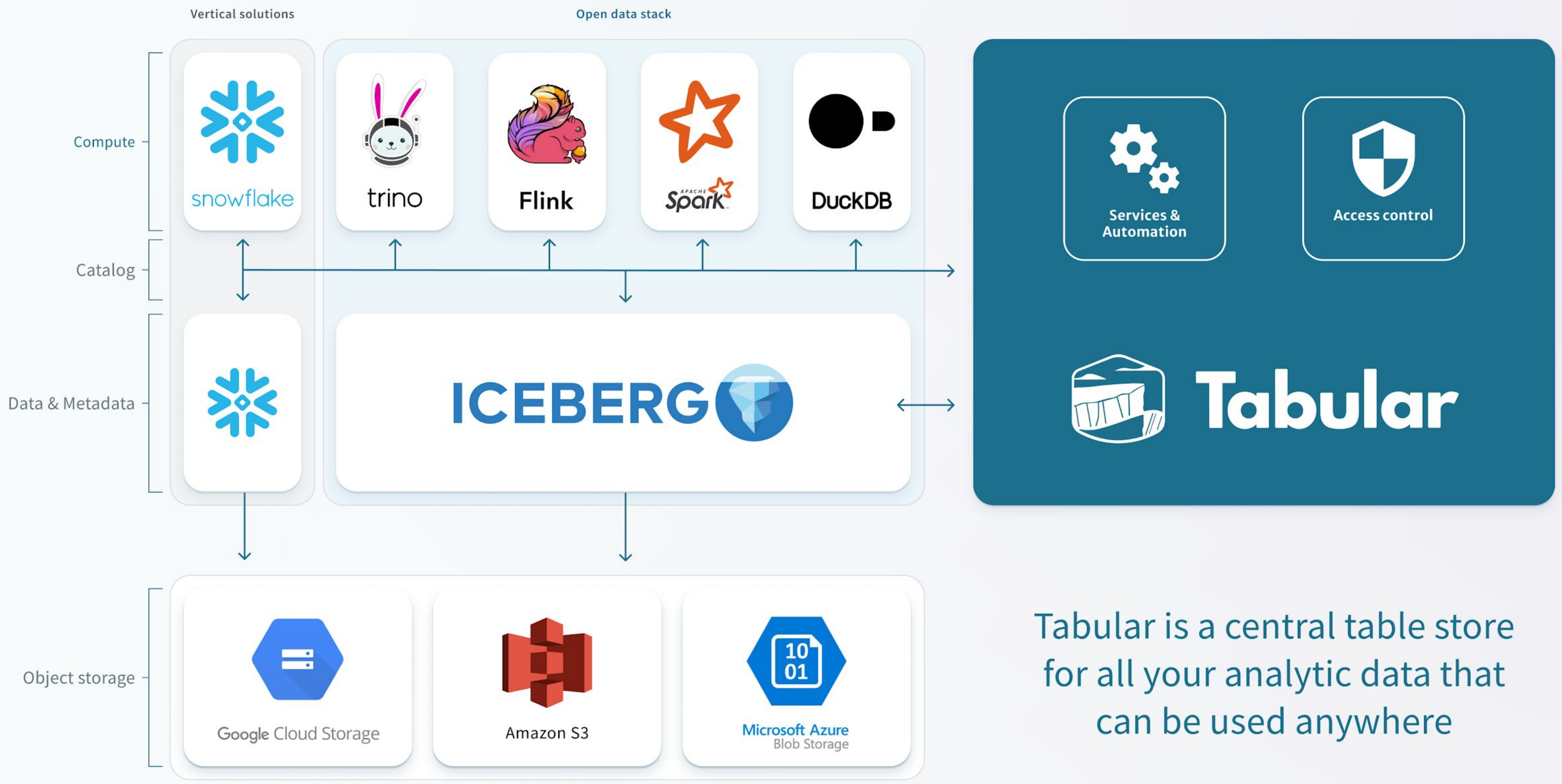
- LSM patterns
  - Equality updates with sorted data

Questions?

Thanks for attending!

app.tabular.io/signup


Vertical solutions
Open data stack
Compute
Catalog
Data & Metadata
Object storage
snowflake
trino
Flink
Spark
DuckDB
ICEBERG
Google Cloud Storage
Amazon S3
Microsoft Azure
Blob Storage
Services & Automation
Access control
Tabular
What is Tabular?



Vertical solutions
Open data stack
Compute
Catalog
Data & Metadata
Object storage
snowflake
trino
Flink
Apache Spark
DuckDB
ICEBERG
Google Cloud Storage
Amazon S3
Microsoft Azure
Blob Storage
Services & Automation
Access control
Tabular
Tabular is a central table store for all your analytic data that can be used anywhere