Beyond Testing:
How to Build Circuit Breakers with Airflow

Prateek Chawla

Agenda

- The Rise of Data Downtime
- The Data Reliability Workflow
- Introduction to Circuit Breaking
- Implementing Circuit Breakers with Airflow

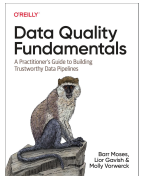
Prateek Chawla, Founding Engineer

MONTE CARLO

Barracuda

$236m in funding by backers of the world's best enterprise companies, including **Accel, GGV, Redpoint, ICONIQ, and IVP**

Creator of the **Data Observability category**



Adopted

by 100s of customers across all industries and sizes

affirm, CNN, FOX, GoodRx, HubSpot, jetBlue, PagerDuty, asics, PEPSICO, auth0

The impact of data downtime

~70

high severity events each year per every 1k tables [1]

30-50%

data engineering time spent on fire drills [2]

$15M

avg. annual financial cost for companies resulting from poor data quality [3]

1. Benchmark data based on Monte Carlo customer production deployments
2. Monte Carlo market research and customer-reported benchmarks & NYT
3. Gartner (2018)

Detection, resolution, and schema management workflows are the issue

Data sources

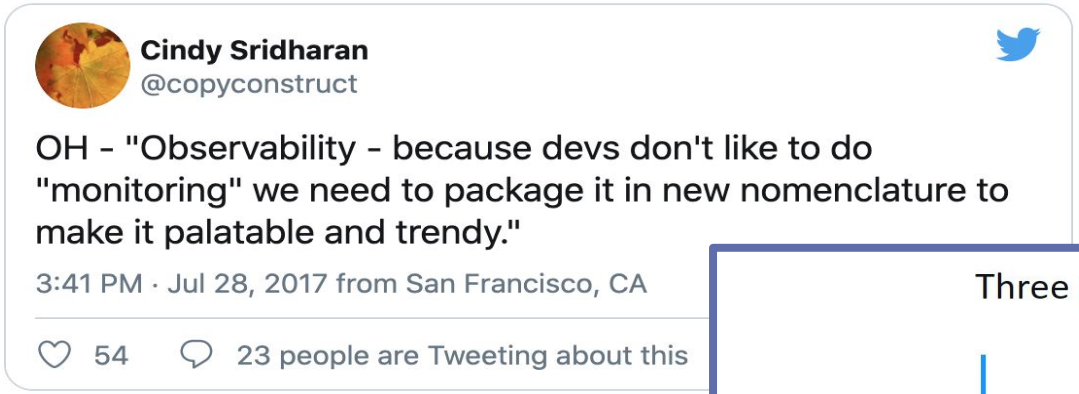
Data warehouses and lakes

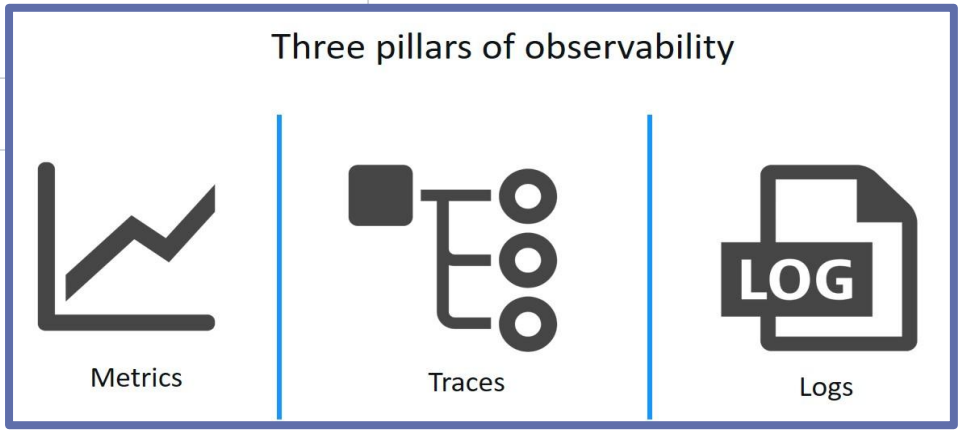
BI tools and data products

Good news: data downtime looks similar at all companies

- Is this data up-to-date?
- Why does this data size look off?
- Isn’t this value suspiciously high?
- Why are there so many nulls?
- Why do we have duplicate IDs?
- What reports will I break with this schema update?
- Why are there 0s on tiles that usually show 100s?
- …

Three Pillars of Observability Engineering





The Solution? Detect. Resolve. Prevent.

Detect

- ML-powered anomaly detection
- Rule-based detection
- Targeted alerts to impacted owners & downstream users

Resolve

- Automated field-level lineage
- Impact radius assessment
- Code, data, and operational diagnostics

Prevent

- Auto-generated and on-demand insights
- Schema change notifications
- **Automated circuit breakers**

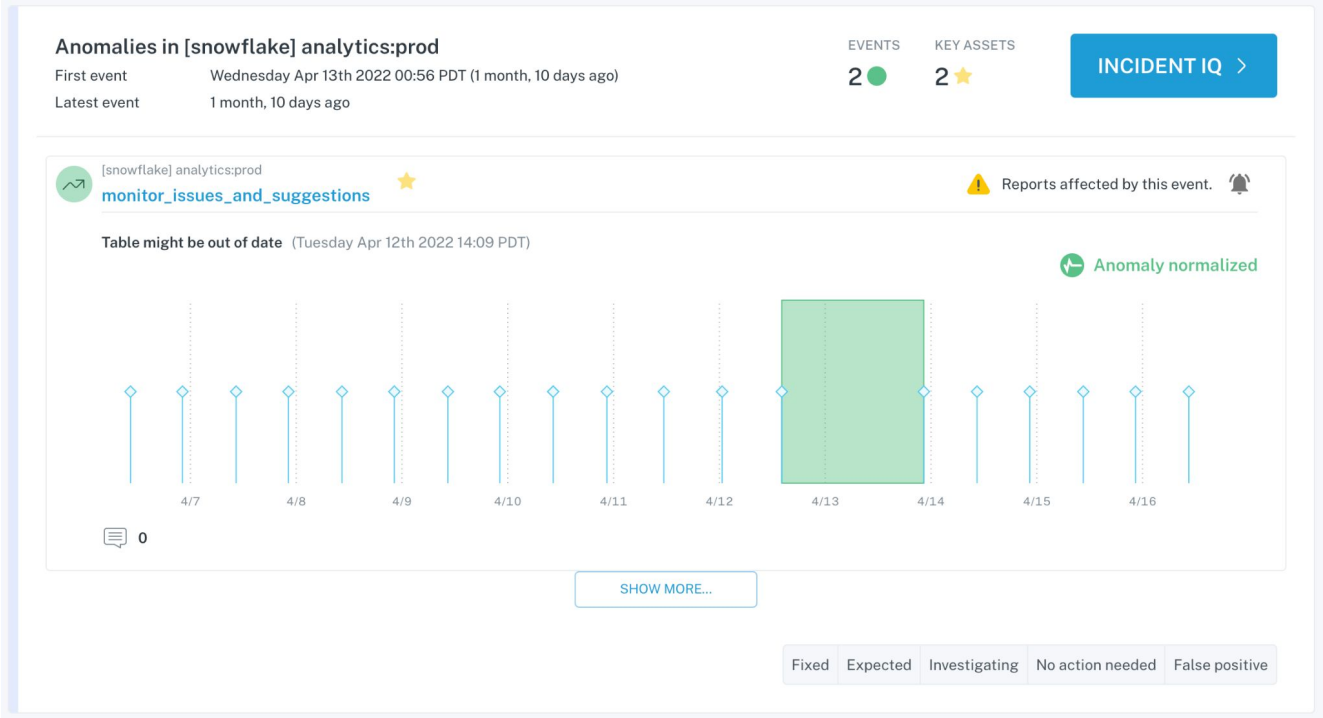
DATA OBSERVABILITY PILLARS

Freshness | Distribution | Volume | Schema | Lineage

The 5 Pillars of Data Observability

DATA OBSERVABILITY PILLARS

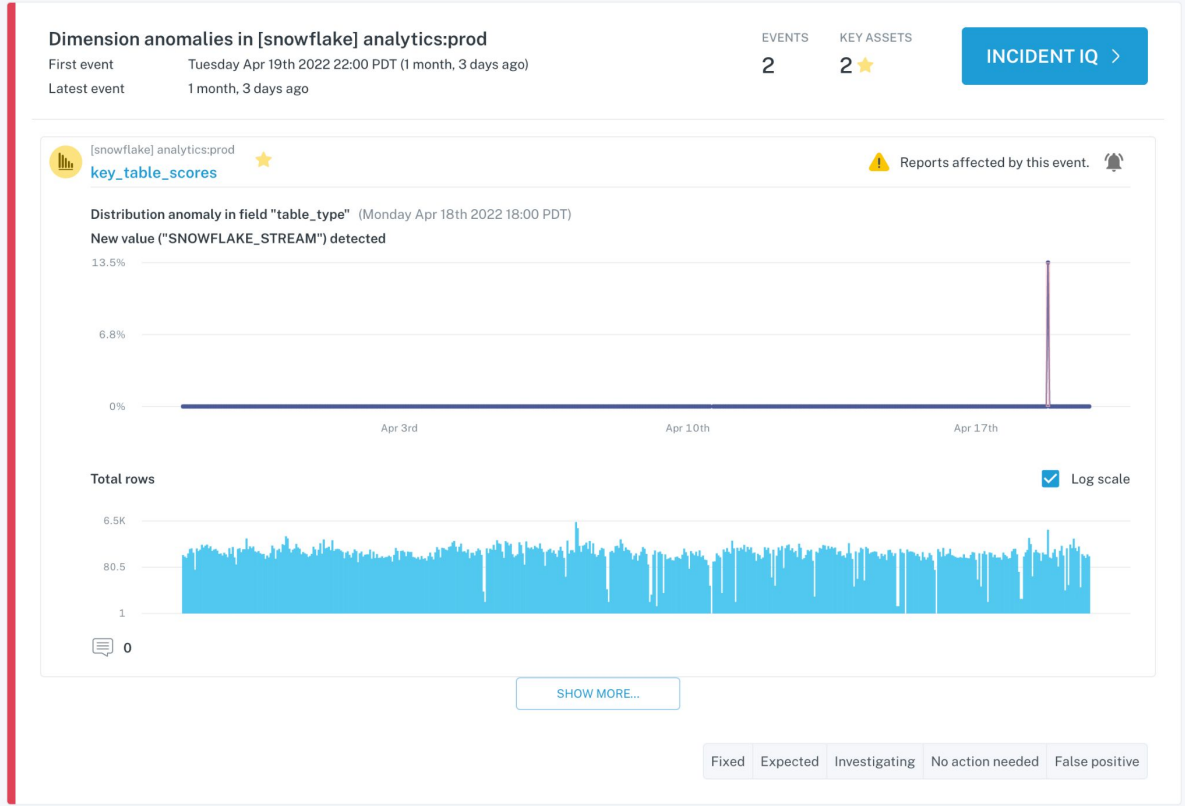
- **Freshness**
- Distribution
- Volume
- Schema
- Lineage



The 5 Pillars of Data Observability

DATA OBSERVABILITY PILLARS

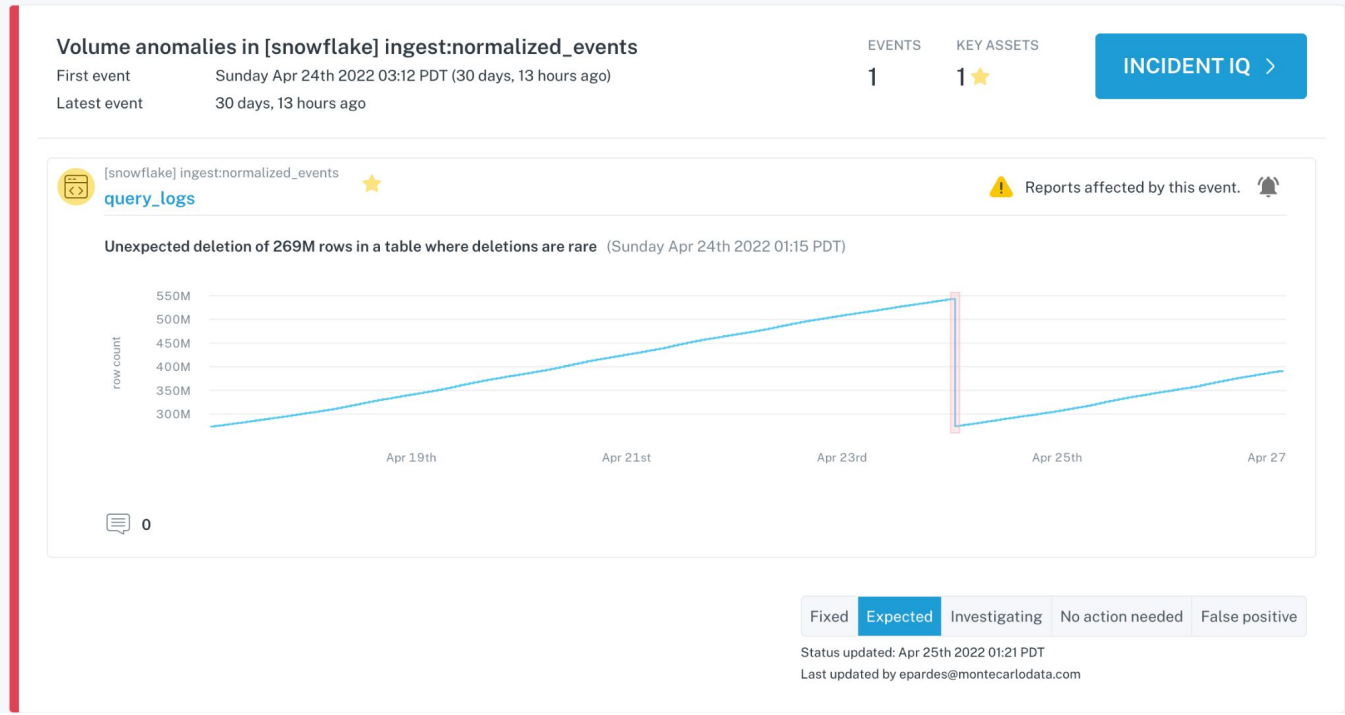
- Freshness
- **Distribution**
- Volumn
- Schema
- Lineage



The 5 Pillars of Data Observability

DATA OBSERVABILITY PILLARS

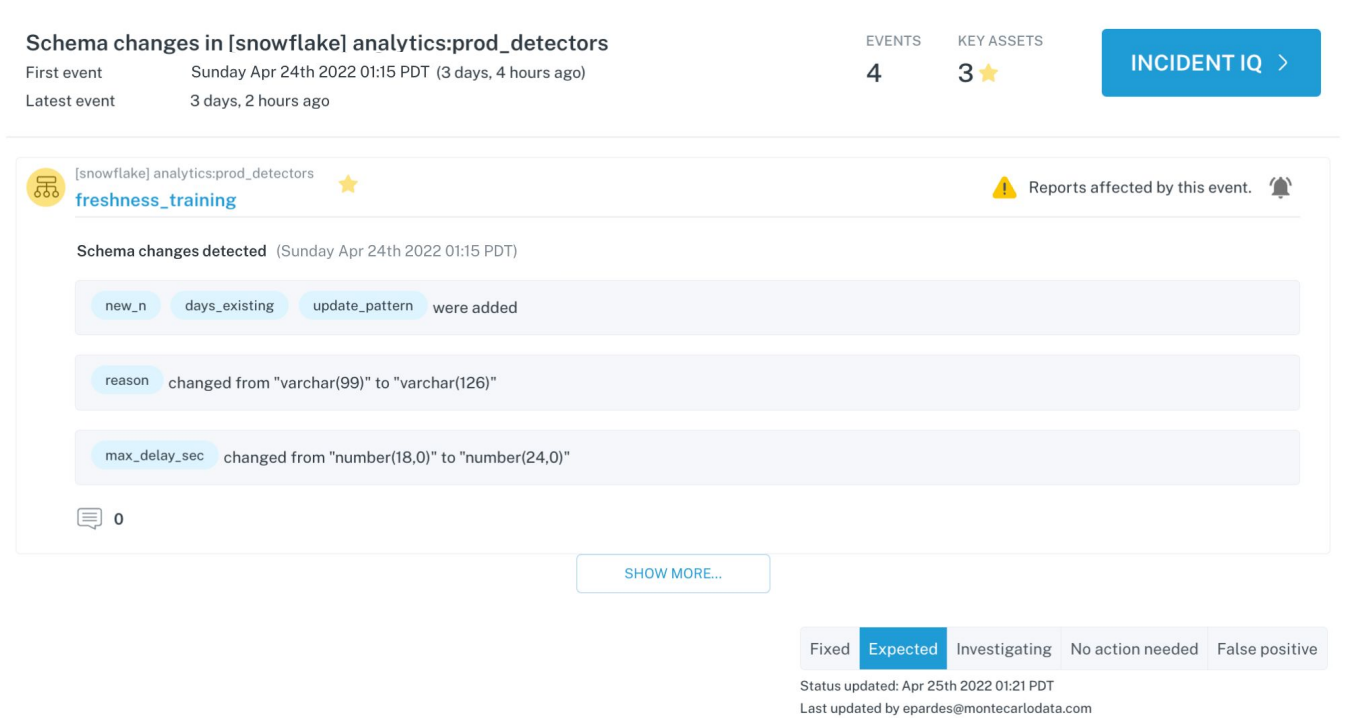
- Freshness
- Distribution
- **Volume**
- Schema
- Lineage



The 5 Pillars of Data Observability

DATA OBSERVABILITY PILLARS

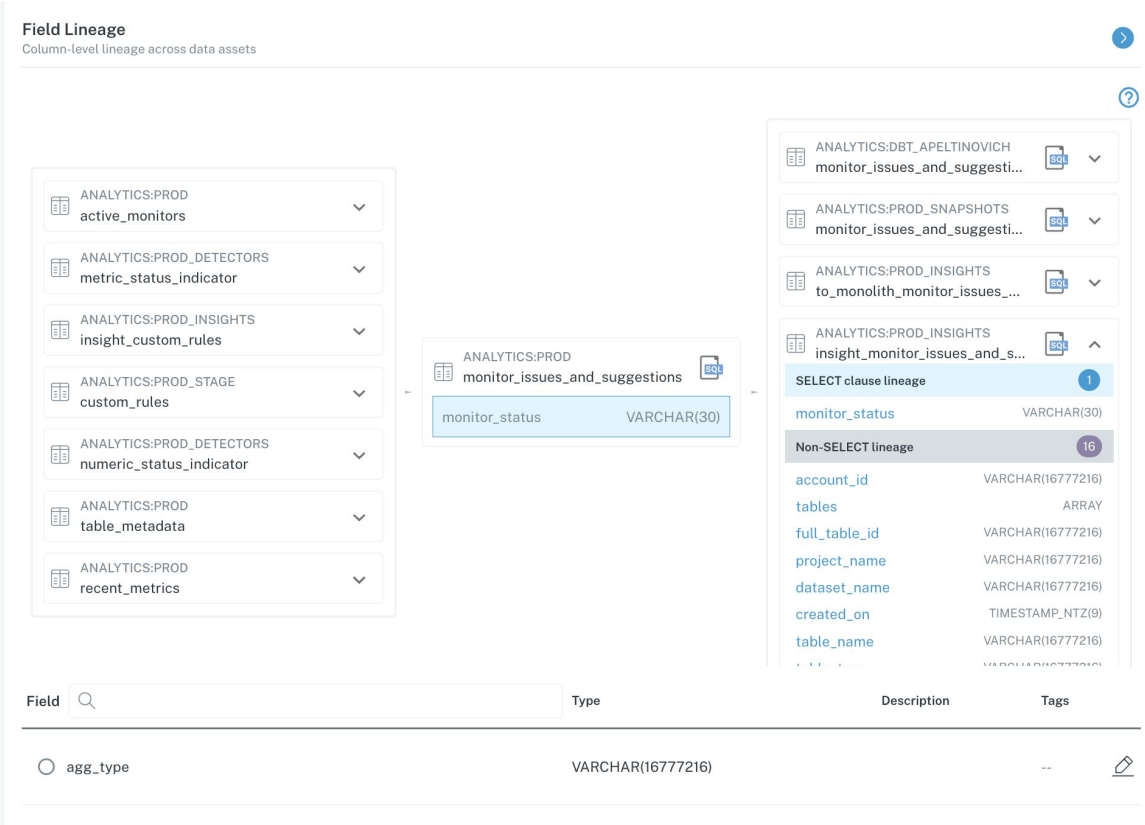
- Freshness
- Distribution
- Volume
- **Schema**
- Lineage



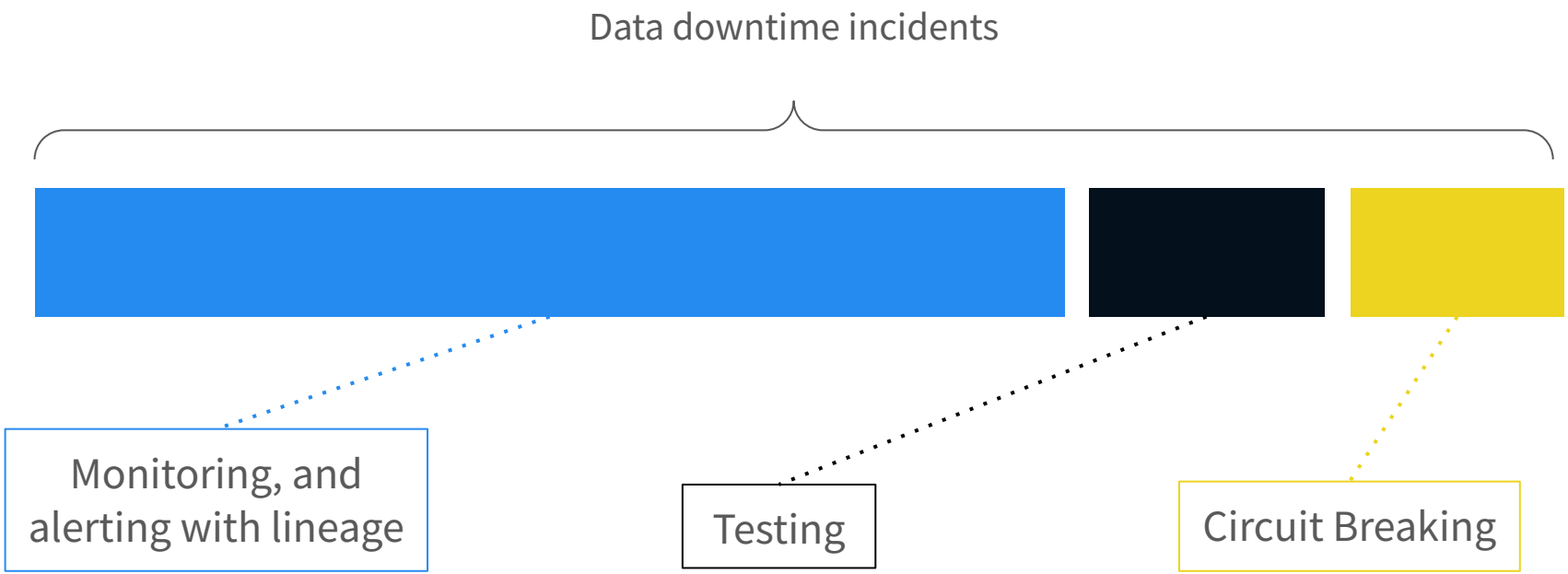
The 5 Pillars of Data Observability

DATA OBSERVABILITY PILLARS

- Freshness
- Distribution
- Volume
- Schema
- **Lineage**



Common use cases & technologies



Aside: The primary challenge with Airflow monitoring

- Airflow has plenty of interfaces and mechanisms to help with monitoring and logging, but it is not data-aware out of the box.

So what is a circuit breaker, really?

- Interrupts current flow to protect equipment if an issue occurs.
- No electricity (an outage) is a better than fire.
- Unlike a fuse, can be reset to resume normal operation.

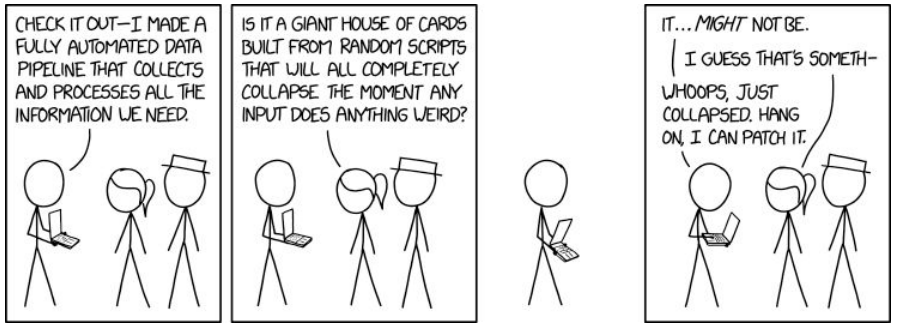
https://en.wikipedia.org/wiki/Circuit_breaker

Circuit breakers vs testing

- Circuit breakers should be the **most important test cases**.
- Circuit breakers create a **guarantee** that any insights produced from this dataset meet your expectations and acceptable thresholds.
- Circuit breakers **should be used sparingly**. Testing should be prolific.
- Circuit breakers are **always automated**. No manual verification of results.
- If a circuit is open, teams should be **immediately paged** to resolve.
- Circuit breakers are **always run in production**. Can (and should) also be be used in dev and stage.

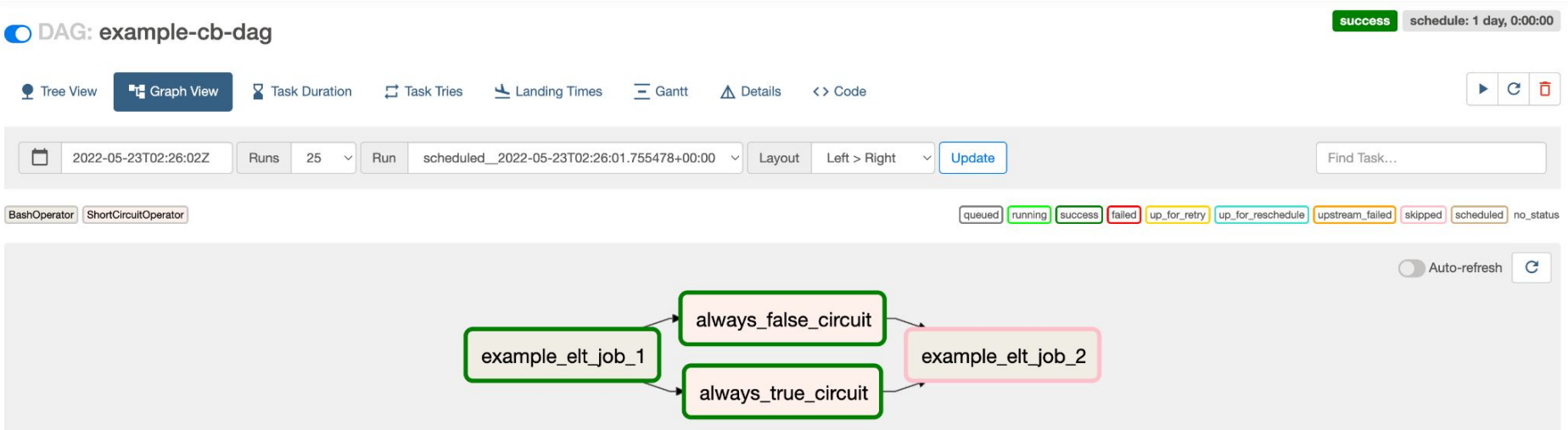
Challenges with circuit breakers

- Circuit breakers provide an excellent way to **prevent data issues** from occurring in the first place.
- However, circuit breakers can also **wreak havoc on pipelines** when delayed jobs set off a chain reaction of **data failures** downstream.
- Keep the following in mind when considering how you implement circuit breakers:
  - Select only rules where you have a good understanding of how often incidents are triggered.
  - Start small, by implementing circuit breakers in a handful of Airflow DAGs, with a lot of visibility to catch issues.
  - Select rules where the underlying SQL executes quickly to avoid long query timeouts.
  - Understand where and when to fail open.



https://xkcd.com/2054/

Now, let's build a circuit breaker!



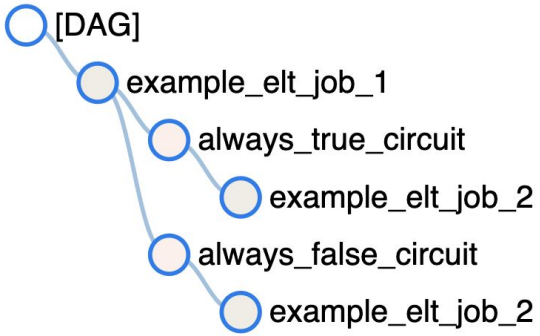
Using the ShortCircuitOperator

Airflow DAGs Security Browse Admin Docs

DAG: example-cb-dag

Tree View | Graph View | Task Duration | Task Tries | Landing Times | Gantt | Details | <> Code

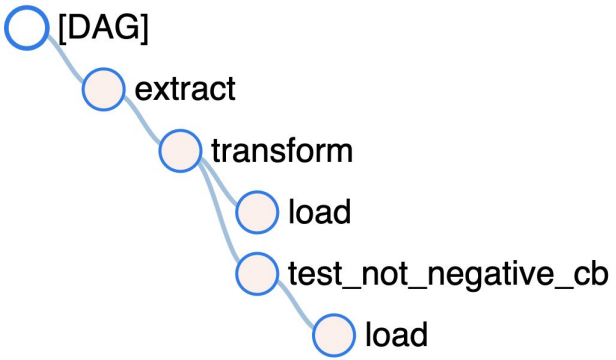
```python
from datetime import datetime, timedelta

from airflow import DAG
from airflow.operators.bash import BashOperator
from airflow.operators.python import ShortCircuitOperator

with DAG('example-cb-dag', start_date=datetime(2022, 5, 22), catchup=False, schedule_interval=timedelta(1)) as dag:
    example_elt_job_1 = BashOperator(
        task_id='example_elt_job_1',
        bash_command='echo I am transforming a very important table!',
    )

    always_true_circuit = ShortCircuitOperator(
        task_id='always_true_circuit',
        python_callable=lambda: True  # Replace with business logic
    )

    always_false_circuit = ShortCircuitOperator(
        task_id='always_false_circuit',
        python_callable=lambda: False  # Replace with business logic
    )

    example_elt_job_2 = BashOperator(
        task_id='example_elt_job_2',
        bash_command='echo I am building a very important dashboard from the table created in example_elt_job_1!',
        trigger_rule='none_failed'
    )

    example_elt_job_1 >> [always_true_circuit, always_false_circuit] >> example_elt_job_2
```



In Airflow 2.X - Using the TaskFlow API paradigm

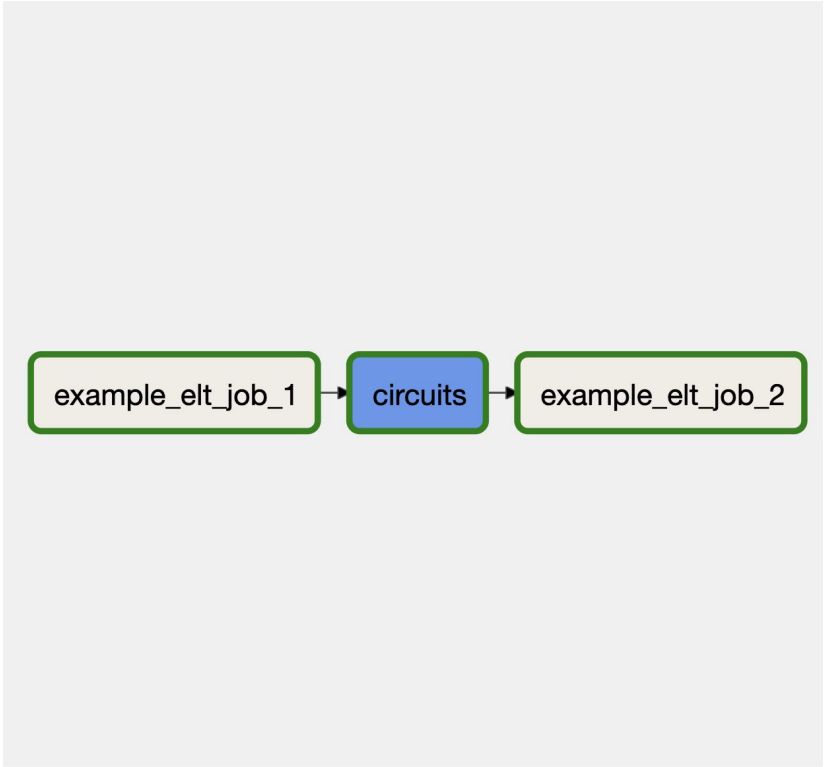
Airflow DAGs Security Browse Admin Docs

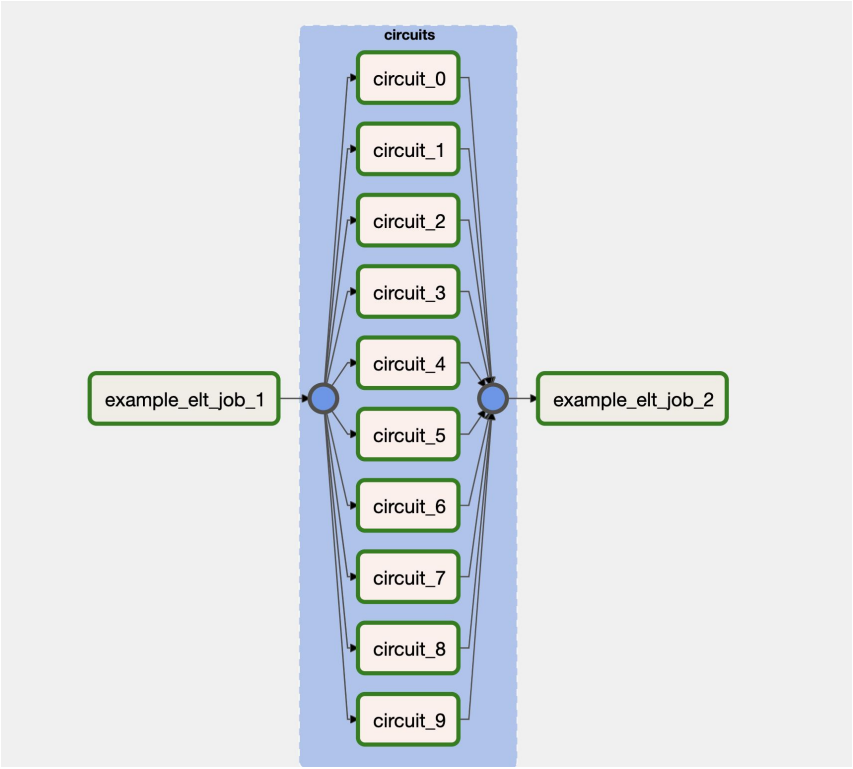
```python
import json
from typing import Dict

import pendulum
from airflow.decorators import dag, task
from airflow.operators.python import ShortCircuitOperator


def test_not_negative(**context) -> bool:
    items_in_usd = context['ti'].xcom_pull(task_ids='transform')
    for item in items_in_usd.values():
        if item < 0:
            return False
    return True


@dag(schedule_interval=None, start_date=pendulum.datetime(2022, 5, 22, tz='UTC'), catchup=False)
def example_cb_etl():
    @task()
    def extract() -> Dict:
        return json.loads('{"item_1": 42, "item_2": 24}')  # items in dollars

    @task()
    def transform(items_in_usd: Dict) -> Dict:
        usd_to_euro_conversion_rate = -0.937478  # bad conversion rate.

        return {name: round(cost_in_usd * usd_to_euro_conversion_rate, 2) for name, cost_in_usd in items_in_usd.items()}

    test_not_negative_cb = ShortCircuitOperator(
        task_id='test_not_negative_cb',
        provide_context=True,
        python_callable=test_not_negative
    )

    @task()
    def load(items_in_euros: Dict) -> None:
        for name, cost_in_euros in items_in_euros.items():
            print(f"Item {name} is {cost_in_euros} euros.")

    transformed_items_in_euros = transform(extract())
    transformed_items_in_euros >> test_not_negative_cb >> load(transformed_items_in_euros)


example_cb_etl_dag = example_cb_etl()
```



MC

In Airflow 2.X - Using Task Groups to improve manageability





Suggestions when adding circuit breakers to your pipeline

- **Do not** limit yourself to one type of operator.
  - You can use any operator, not just the `ShortCircuitOperator` to create a custom circuit breaker.
  - You can leverage many tools like DBT, Great Expectations, or Monte Carlo 🙂 as a circuit breaker.
- **Try** raising an `AirflowSkipException` instead of an `AirflowException` when a circuit is closed.
  - This increases visibility and prevents automatic retries.
- **Do not** merge multiple circuits into one operator.
  - This makes it harder to trace exactly what issue or threshold tripped the circuit.
- **Do not** merge a circuit breaker into the operator performing the task to evaluate.
  - This makes it more difficult to determine if the job failed or if the circuit was tripped.
- **Include** a mechanism to bypass or skip a circuit breaker.
- **Generalize** repeated patterns as plugins or custom operators.

Practical thresholds: Using freshness in circuit breakers

- A common pattern to circuit break on is to check if your table was actually updated.
- Many warehouse and lakes expose this information via a metastore or log.
  - In Snowflake:
    `SELECT CONVERT_TIMEZONE('UTC', last_altered) last_altered FROM information_schema.tables ...`
  - With Delta Lake:
    `DESCRIBE DETAIL ...` or via the `deltaLog.snapshot`
- It is worth combining this metric with volume and tracking changes and patterns over time.

Practical thresholds: Using custom logic in circuit breakers

- Breach when row count is greater than 0.

Row count match

Two tables have the same number of rows.

SQL

```
WITH t1
     AS (SELECT Count(*)
         FROM   {{table_1}}),
     t2
     AS (SELECT Count(*)
         FROM   {{table_2}}),
     t3
     AS (SELECT (SELECT *
                 FROM   t1) - (SELECT *
                               FROM   t2) AS diff_count)
SELECT diff_count
FROM   t3
WHERE  diff_count != 0
```

Identical tables

Two tables are identical record for record.

SQL

```
(SELECT *
 FROM   {{table_1}}
 {{except_operator}}
 SELECT *
 FROM   {{table_2}})
UNION ALL
(SELECT *
 FROM   {{table_2}}
 {{except_operator}}
 SELECT *
 FROM   {{table_1}})
```

Referential integrity

All IDs in a child table must be present in a parent table.

SQL

```
WITH child
     AS (SELECT {{column}} AS id
         FROM   {{table_1}}),
     parent
     AS (SELECT {{column}} AS id
         FROM   {{table_2}})
SELECT *
FROM   child
       LEFT JOIN parent using (id)
WHERE  child.id IS NOT NULL
       AND parent.id IS NULL
```

MC

https://docs.getmontecarlo.com/docs/creating-sql-rules#common-sql-rules

What happens AFTER a circuit breaker is triggered?

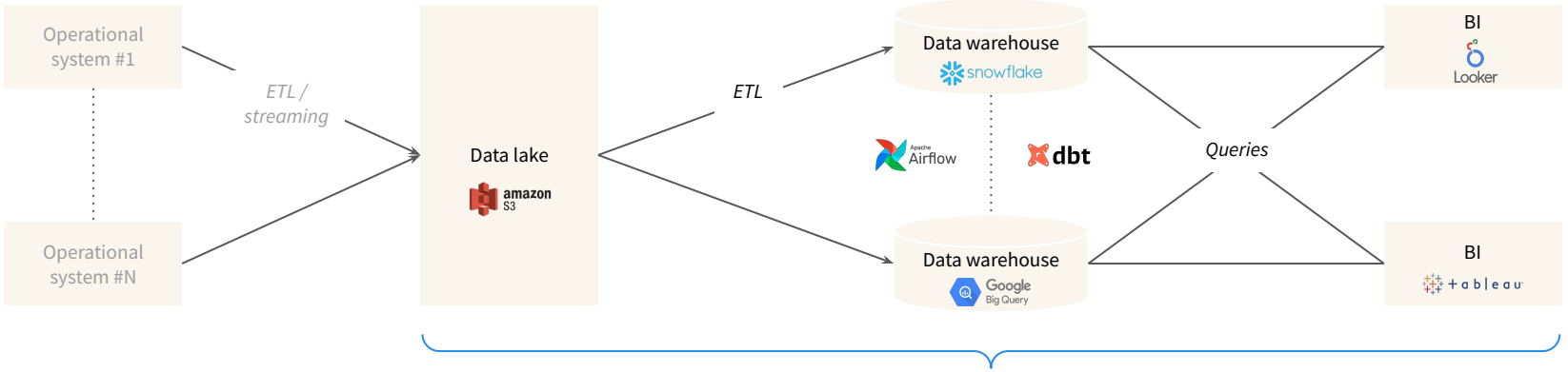
- Each data incident is different, but the following **7** steps are good starting point:
    1. **Review** table and field lineage for all impacted downstream assets and **notify** stakeholders.
    2. **Review** logs and any documentation for the circuit that tripped.
    3. **Review** logs and any documentation for the tasks(s) being evaluated.
    4. **Review** field lineage for all upstream transformations for table(s) that could be the root cause.
    5. **Check** for recent changes to the pipeline, like new data sources, streams, or modified queries.
    6. **Sample** table for anomalies and use any tools to review historical versions of the model.
    7. **Review** for any historical incidents or schema changes on the affected table(s).

Check out our Airflow Provider and SDK

https://pypi.org/project/airflow-mcd/

https://pypi.org/project/pycarlo/

A complete data observability platform enables your team to detect, resolve, and prevent bad data altogether

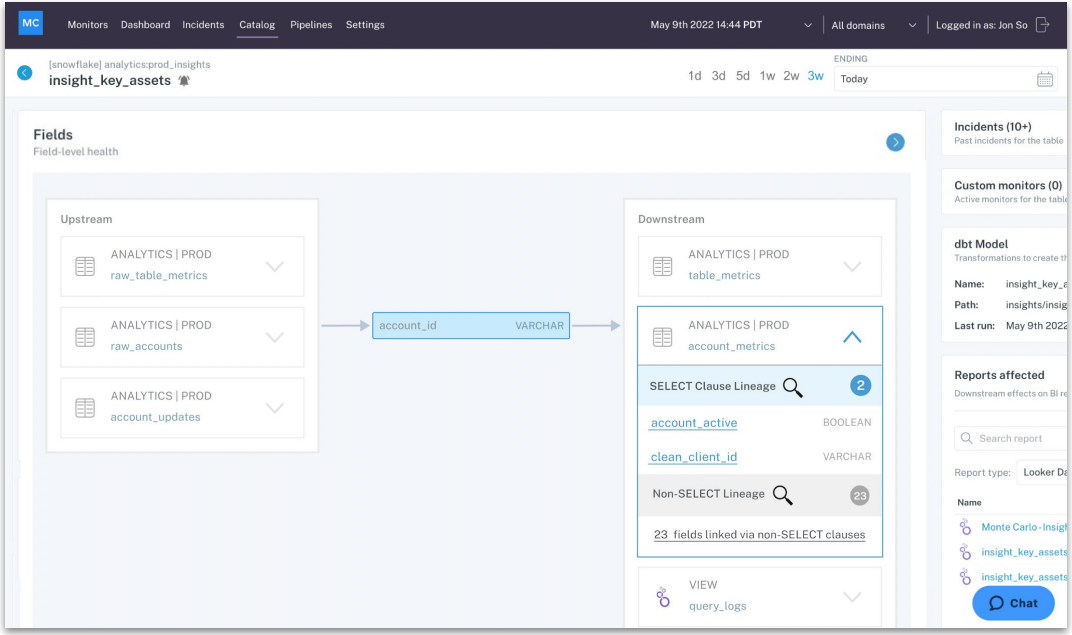


Monte Carlo securely collects metadata, metrics, query logs, and statistics to enable:

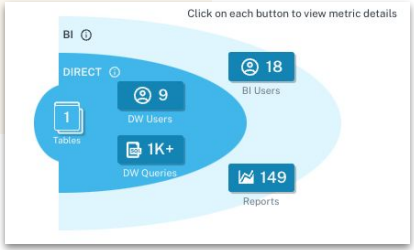
Detect	Resolve	Prevent
ML-powered data monitoring	Automated field-level lineage	On-demand reports & insights

Triage, find root cause, and manage incident workflows all in one platform

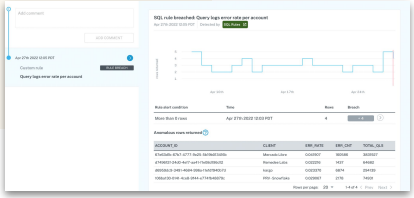
Automated field-level lineage



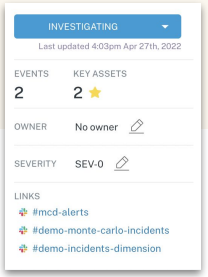
Triage incidents



Find root cause



Manage workflows



Key takeaways

- Data downtime has a measurable **impact**.
- The 5 pillars of data observability can give you a **holistic view of data health.**
- Circuit breaking is **powerful** and can be **easy to implement**, but it is only one tool and must be combined with others like monitoring, alerting, lineage, and testing for maximum effectiveness.
- Circuit breakers are like testing on steroids, so you have to **be careful** to prevent unintended data downtime.
- You have a **lot of options** for how to instrument circuit breakers and what **quality or integrity thresholds** you can use. Experiment and have fun!

Thank you!

www.montecarlodata.com/blog

Prateek Chawla: pchawla@montecarlodata.com