Automating Airflow Backfills with Marquez
Hey!
I’m Willy Lulciuc
Software Engineer, Astronomer
Co-creator, Marquez
Committer, OpenLineage
@wslulciuc
AGENDA

01 Backfills (naive)
02 Intro to OpenLineage
03 Intro to Marquez
04 Backfills (take 2)
05 Future work
Let’s get booking!
<table>
<thead>
<tr>
<th>Location</th>
<th>Room size</th>
<th>When</th>
</tr>
</thead>
<tbody>
<tr>
<td>Salesforce Tower</td>
<td>1-4 seats</td>
<td>Thu, November 8</td>
</tr>
</tbody>
</table>
01 Location + floor
02 Open time slot
<table>
<thead>
<tr>
<th>Location + floor</th>
<th>Open time slot</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Salesforce Tower</td>
<td>12pm - 2pm</td>
<td>4pm - 6pm</td>
</tr>
</tbody>
</table>
01 Location + floor
02 Open time slot
03 Duration
04 Confirm
Which location has the most bookings?
Set[RoomBooking] ➞ LocationID
SELECT location, COUNT(*) AS bookings, booked_by
FROM room_bookings
GROUP BY bookings DESC
LIMIT 1
... but, DAGs fail and backfills are a thing
Backfilling

As your organization scales up, so will the amount of data and number of internal data sources. As data outages happen, they become more serious. **Backfilling** refers to the process of retroactively processing historical data. Having a central place to examine and understand DAG dependencies will make your organization more resilient to data outages.
DAG Failures

- Data quality
  - Data freshness (incomplete or missing data, etc)
  - Data schema change (column dropped, data type changed)
- Bad code
  - DAG crashes
- DAG dependencies
  - Upstream/downstream DAG failures
01 Backfills (naive)
Airflow backfill

$ airflow backfill \
    --start-date <START_DATE> \
    --end-date <END_DATE> \
    <DAG_ID>
Airflow execution_date

2022-01-01
YYYY-MM-DD

24hrs

2022-01-02
YYYY-MM-DD

24hrs

2022-01-03
YYYY-MM-DD

24hrs

start_date: 2022-01-02
execution_date: 2022-01-01

start_date: 2022-01-03
execution_date: 2022-01-02...
Data Quality Failures
Retries!

Let’s keep trying!
Upstream Dependency Failures
One Bad Datapoint
Bad Code Failures

Input Dataset → Job → Output Dataset
Downstream Failures
Backfilling is tough...

- How quickly can data quality issues be identified and explored?
- What alerting rules should be in place to notify downstream DAGs of possible upstream processing issues or failures?
- What effects (if any) would upstream DAGs have on downstream DAGs if dataset consumption was delayed?
... ugh, backfills shouldn’t be this hard!
Open Lineage
OpenLineage: Intro

Data Analysis Tools  Schedulers  Data Warehouse  SQL Engines

Amundsen  Apache Atlas  EGERIA
OpenLineage: Intro

- Data Analysis Tools
- Schedulers
- Data Warehouse
- SQL Engines

OpenLineage

- Amundsen
- Apache Atlas
- EGERIA
A **open standard** with a **specification** for collecting **lineage** metadata

- **Focuses on job-level** execution
  - Datasets
  - Jobs
  - Runs

- **Event-based** metadata collection

- **Extensible** model via facets
Metadata Service

- **Centralized metadata management**
  - Sources
  - Datasets
  - Jobs

- **Features**
  - Data governance
  - Data lineage
  - Data discovery + exploration

[Diagram of Marquez: Intro showing ETL, Batch, Stream, REST API, Core, Lineage, Search]
Marquez: Data model

- Source
  - MYSQL
  - POSTGRESQL
  - REDSHIFT
  - SNOWFLAKE
  - KAFKA
  - S3

- Dataset
- Dataset Version
- Job
- Job Version
- Run

- Source: 1
- Dataset: 1
- Dataset Version: *
- Job: *
- Job Version: *
- Run: 1

Types:
- BATCH
- STREAM
- SERVICE
Design benefits

- **Debugging**
  - What *job version(s)* produced and consumed *dataset version X*?

- **Backfilling**
  - Full / incremental processing
How is metadata collected?

- **Push-based** metadata collection
- REST API
- **OpenLineage** integrations
  - Airflow
  - Spark
  - dbt
Capturing lineage metadata with Marquez using OpenLineage in a nutshell
OpenLineage + Airflow
OpenLineage support for Airflow

- **Metadata**
  - Task lifecycle
  - Task parameters
  - Task runs linked to *versioned* code
  - Task inputs / outputs

- **Lineage**
  - Track inter-DAG dependencies

- **Built-in**
  - SQL parser
  - Link to code builder (**GitHub**)
  - Metadata extractors
OpenLineage Airflow Lib.

- Open source! 🏆
- Enables **global** task-level metadata collection
- Extends Airflow’s DAG class

```python
from openlineage.airflow import DAG
from airflow.operators.postgres_operator import PostgresOperator
...
```
OpenLineage: Airflow

Airflow

Operator

Example

airflow.operators.PostgresOperator

Extractor

openlineage.extractors.PostgresExtractor

Metadata

OpenLineage

Airflow Lib.
t1 = PostgresOperator(
    task_id='new_room_booking',
    postgres_conn_id='analyticsdb',
    sql='''
    INSERT INTO room_bookings VALUES(%s, %s, %s)
    ''',
    parameters=... # room booking
)
```
t1=PostgresOperator(
    task_id='new_room_booking',
    postgres_conn_id='analyticsdb',
    sql='''
        INSERT INTO room_bookings VALUES(%s, %s, %s)
    ''',
    parameters=... # room booking
)
t1=PostgresOperator(
    task_id='new_room_booking',
    postgres_conn_id='analyticsdb',
    sql='''
    INSERT INTO room_bookings VALUES(%s, %s, %s)
    '''
    parameters=...  # room booking
)
Managing inter-DAG dependencies

new_room_bookings_dag.py  top_room_bookings_dag.py
Managing inter-DAG dependencies

new_room_bookings_dag.py  public.room_bookings  top_room_bookings_dag.py

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>TS</th>
<th>ROOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>b648485,</td>
<td>1541501885,</td>
<td>9</td>
</tr>
<tr>
<td>b940314,</td>
<td>1541624285,</td>
<td>2</td>
</tr>
<tr>
<td>b648485,</td>
<td>1541710685,</td>
<td>4</td>
</tr>
<tr>
<td>1541501885</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1541624285</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1541710685</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
04 Backfills (take 2)
Fail Collaboratively

- **Global View**
  - Lineage metadata allows teams to look at failures across the organization, understanding the impact of the data outage

- **Coordinate**
  - Efforts aren’t duplicated

- **Empower**
  - Give teams the power to resolve data outages completely
Roadmap

- Column-level lineage support
- Job hierarchy and grouping
- Flink integration
Thanks! <o/>
Be cool, take the Airflow survey!

bit.ly/AirflowSurvey22
Questions?