Data Lineage with Apache Airflow using OpenLineage

Julien Le Dem and Willy Lulciuc, Datakin | July 2021
Agenda

The need for lineage metadata

OpenLineage and Marquez
- OpenLineage, an open standard for lineage collection
- Marquez, its reference implementation

Airflow observability with OpenLineage
The need for lineage metadata
Building a healthy data ecosystem
Today: limited context

DATA

- What is the data source?
- What is the schema?
- Who is the owner?
- How often is it updated?
- Where does it come from?
- Who is using it?
- What has changed?
Maslow’s Data hierarchy of needs

- New Opportunities
- Business Optimization
- Data Quality
- Data Freshness
- Data Availability
OpenLineage
Contributors
Purpose

To define an open standard for collection of lineage metadata from pipelines as they are running.
EXIF for data pipelines
Before OpenLineage
With OpenLineage

- Analysis Tools
- Schedulers
- Warehouses
- SQL Engines

OpenLineage

- Amundsen
- Data Hub
- MARQUEZ
- EGERIA
OpenLineage architecture

Producers

Backend

Consumers

Metadata and lineage collection standard

GraphDB client

Kafka client

HTTP client

GraphDB

Kafka topic

Graph DB

Connecting producers to the OpenLineage framework.

1. Pandas
2. Apache Spark
3. dbt
4. Apache Airflow
5. Prefect

Connecting the backend to the OpenLineage framework.

1. GraphDB client
2. Kafka client
3. HTTP client
4. ...
Data model

Built around core entities: Datasets, Jobs, and Runs

Defined as a JSONSchema spec

Consistent naming for:
Jobs (scheduler.job.task)
Datasets (instance.schema.table)
Protocol

Asynchronous events:
- Unique run ID for identifying a run and correlated events

Configurable backend:
- Kafka
- HTTP

Examples

**Run Start** event
- source code version
- run parameters

**Run Complete** event
- input dataset
- output dataset version and schema
Extensible

Facets are atomic pieces of metadata identified by a unique name that can be attached to core OpenLineage entities.

Decentralized

Prefixes in facet names allow the definition of Custom Facets that can be promoted to the spec at a later point.
Facet examples

Dataset:
- Stats
- Schema
- Version
- Column-level lineage

Job:
- Source code
- Dependencies
- Source control
- Query plan

Run:
- Scheduled time
- Batch ID
- Query profile
- Params
OMG the possibilities are endless

Dependency tracing
Root cause identification
Issue prioritization
Impact mapping
Precision backfills
Anomaly detection
Change management
Historical analysis
Compliance
Metadata Service

- Centralized metadata management
  - Sources
  - Datasets
  - Jobs
- Features
  - Data governance
  - Data lineage
  - Data discovery + exploration
Marquez: Data model

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

Dataset

Job
- BATCH
- STREAM
- SERVICE

Job Version

Run

Dataset Version

1 - 1

* - *

1 - 1
Design benefits

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

- **Backfilling**
  - Full / incremental processing
Airflow observability with OpenLineage
Airflow support for Marquez

- **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](https://github.com))
  - Metadata extractors
Capturing task-level metadata in a nutshell
Marquez Airflow Lib.

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

```python
from marquez_airflow import DAG
from airflow.operators.postgres_operator import PostgresOperator
...
```

```
room_bookings_7_days_dag.py
```
Marquez: Airflow

Airflow

Operator

```
airflow.operators.\texttt{PostgresOperator}
```

Extractor

```
marquez_airflow.extractors.\texttt{PostgresExtractor}
```

Marquez Airflow Library

Metadata
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
)
Operator Metadata

```
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>
</tbody>
</table>
Demo
Join the conversation

Open Lineage

github.com/openlineage
openlineage.slack.com
@openlineage
groups.google.com/g/openlineage

MARQUEZ

github.com/marquezproject
marquezproject.slack.com
@marquezproject