

Data Lineage

With OpenLineage and Airflow

Ross Turk
Michael Robinson

ASTRONOMER



Agenda

Intro & data lineage strategies	10m
OpenLineage design principles	15m
<ul style="list-style-type: none">• Architecture & data model• Lifecycle of a Job Run	
Exercise 1: Setting up Marquez	20m
<ul style="list-style-type: none">• Deploying with docker-compose• Exploring the seed data	
Exercise 2: The Lineage API	25m
<ul style="list-style-type: none">• Using curl• Using python	
The basics of OpenLineage with Airflow	20m
<ul style="list-style-type: none">• Operators & extractors• Configuration	
Exercise 3: Setting up Airflow	20m
Running our pipeline	30m
<ul style="list-style-type: none">• Reviewing example DAGs• Running & verifying the output	

Prerequisites

8GB of RAM is recommended for the exercises in this workshop.

1. Install Docker Desktop:
<https://www.docker.com/get-started/>
2. Install Git (if it's not already installed):
<https://git-scm.com/downloads>
3. Configure Docker Desktop:
At least **4GB RAM** should be allocated for containers in the Resources section

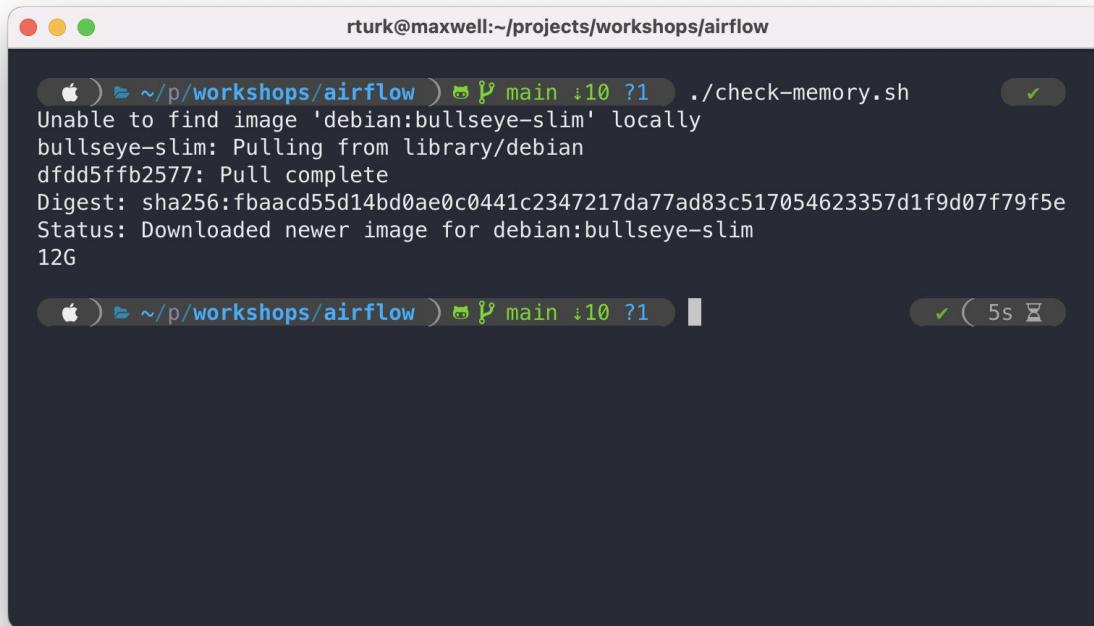
Checking out the workshop project

The screenshot shows a macOS terminal window with three tabs. The top tab is titled "rturk@maxwell:~/projects/workshops/airflow". The bottom tab is titled "rturk@maxwell:~/p/workshops/airflow)". The current tab is the middle one, showing the command "git clone git@github.com:OpenLineage/workshops.git". The output of the command is displayed, showing the progress of cloning the repository from GitHub. The progress bar at the end of the output is green, indicating success. The terminal has a dark theme with light-colored text and icons.

```
rturk@maxwell:~/projects/workshops/airflow
git clone git@github.com:OpenLineage/workshops.git
Cloning into 'workshops'...
remote: Enumerating objects: 81, done.
remote: Counting objects: 100% (81/81), done.
remote: Compressing objects: 100% (45/45), done.
remote: Total 81 (delta 31), reused 75 (delta 25), pack-reused 0
Receiving objects: 100% (81/81), 11.89 KiB | 5.94 MiB/s, done.
Resolving deltas: 100% (31/31), done.

rturk@maxwell:~/p/workshops/airflow)意识形态 main
```

Verifying available memory



A screenshot of a macOS terminal window titled "rturk@maxwell:~/projects/workshops/airflow". The window shows the command `./check-memory.sh` being run. The output indicates that the image 'debian:bullseye-slim' was pulled from the library/debian repository, with a digest of sha256:fbaacd55d14bd0ae0c0441c2347217da77ad83c517054623357d1f9d07f79f5e and a status message stating 'Downloaded newer image for debian:bullseye-slim'. A memory usage of 12G is also shown. The terminal has three tabs open, and the current tab is highlighted with a green checkmark icon.

```
rturk@maxwell:~/projects/workshops/airflow
( ) ~ /p/workshops/airflow ) ⌘ ⌘ main ↵ 10 ?1 ./check-memory.sh
Unable to find image 'debian:bullseye-slim' locally
bullseye-slim: Pulling from library/debian
dfdd5ffb2577: Pull complete
Digest: sha256:fbaacd55d14bd0ae0c0441c2347217da77ad83c517054623357d1f9d07f79f5e
Status: Downloaded newer image for debian:bullseye-slim
12G

( ) ~ /p/workshops/airflow ) ⌘ ⌘ main ↵ 10 ?1
```

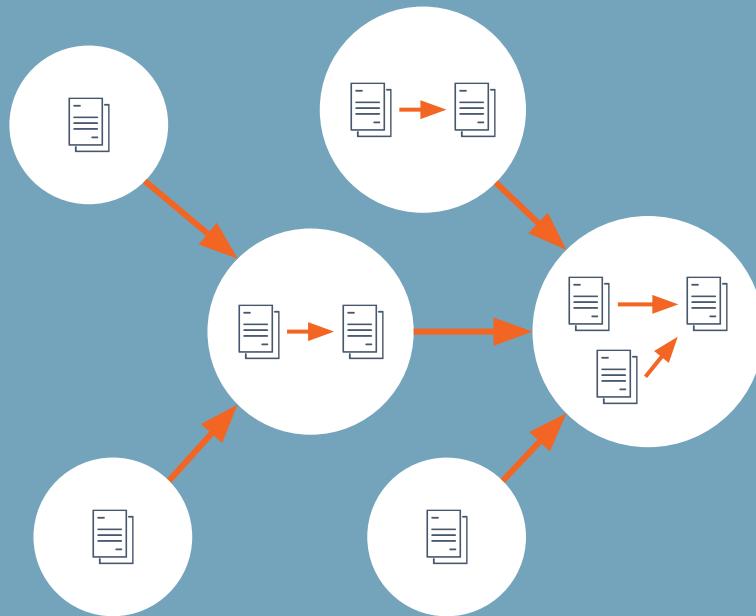
1

Data lineage strategies

What is data lineage?

Data lineage is the set of complex relationships between datasets and jobs in a pipeline.

- Producers & consumers of each dataset
- Inputs and outputs of each job



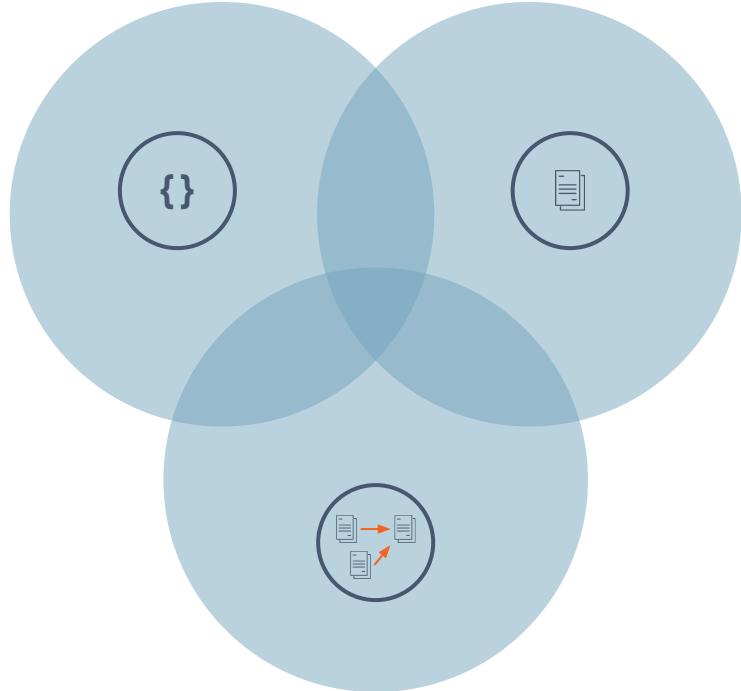
The best time to collect metadata



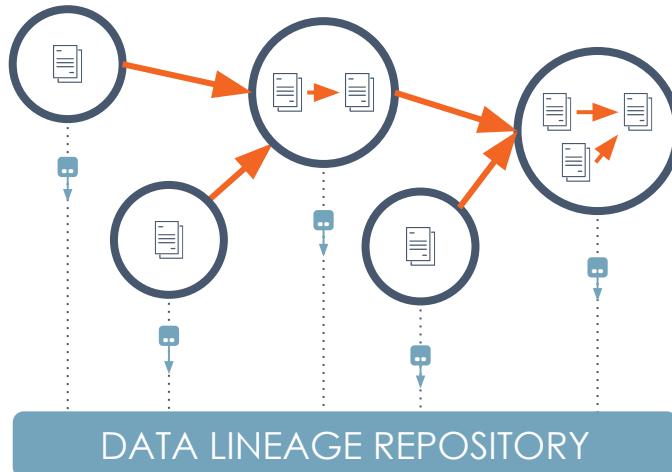
You can try to infer the date and location of an image after the fact...

...or you can capture it when the image is originally created!

Comparing approaches for gathering lineage

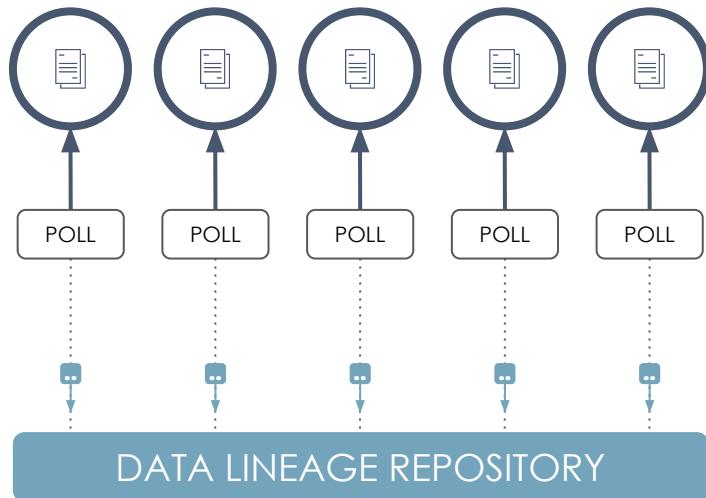


Observe the pipeline



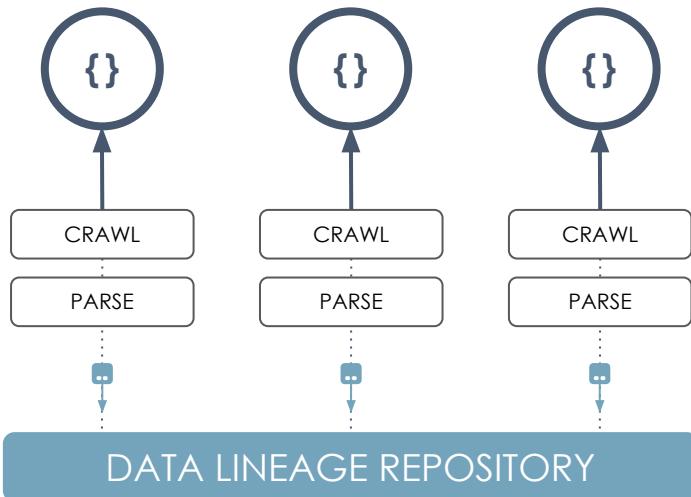
- | Integrate with data orchestration systems
- | As jobs run, observe the way they affect data
- | Report to a lineage metadata repository

Process query / activity logs



- | Integrate with data stores and warehouses
- | Regularly process query logs to trace lineage
- | Report to a lineage metadata repository

Analyze source code



- Integrate with source code repositories
- Look for queries and parse them for lineage
- Report to a lineage metadata repository

It's a patchwork



2

OpenLineage
design principles

OpenLineage

Mission

To define an **open standard** for the collection of lineage metadata from pipelines **as they are running**.

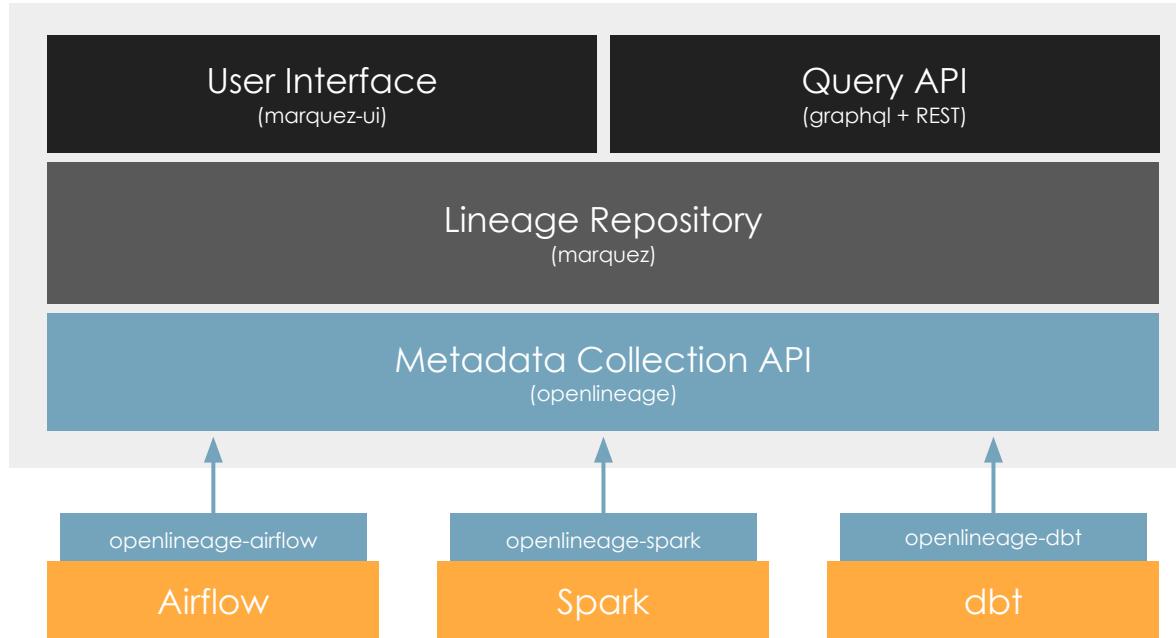


OpenLineage contributors

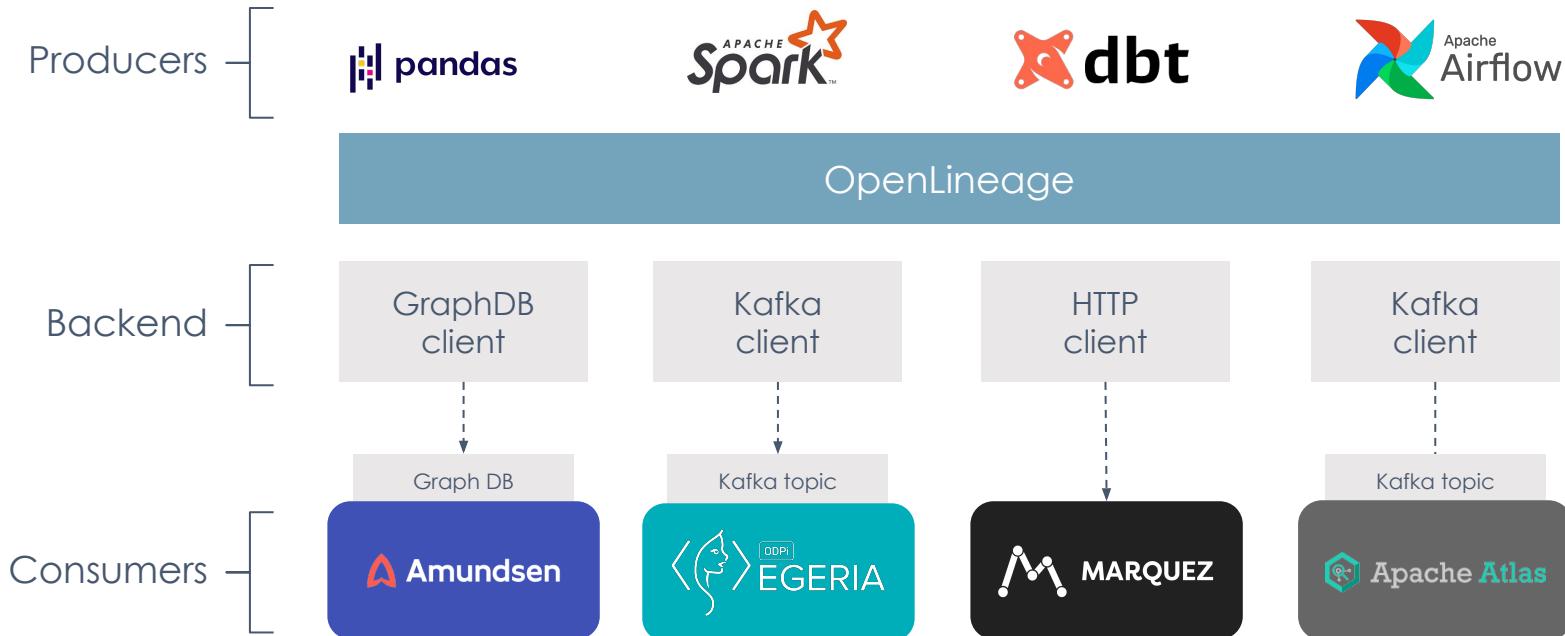


great_expectations

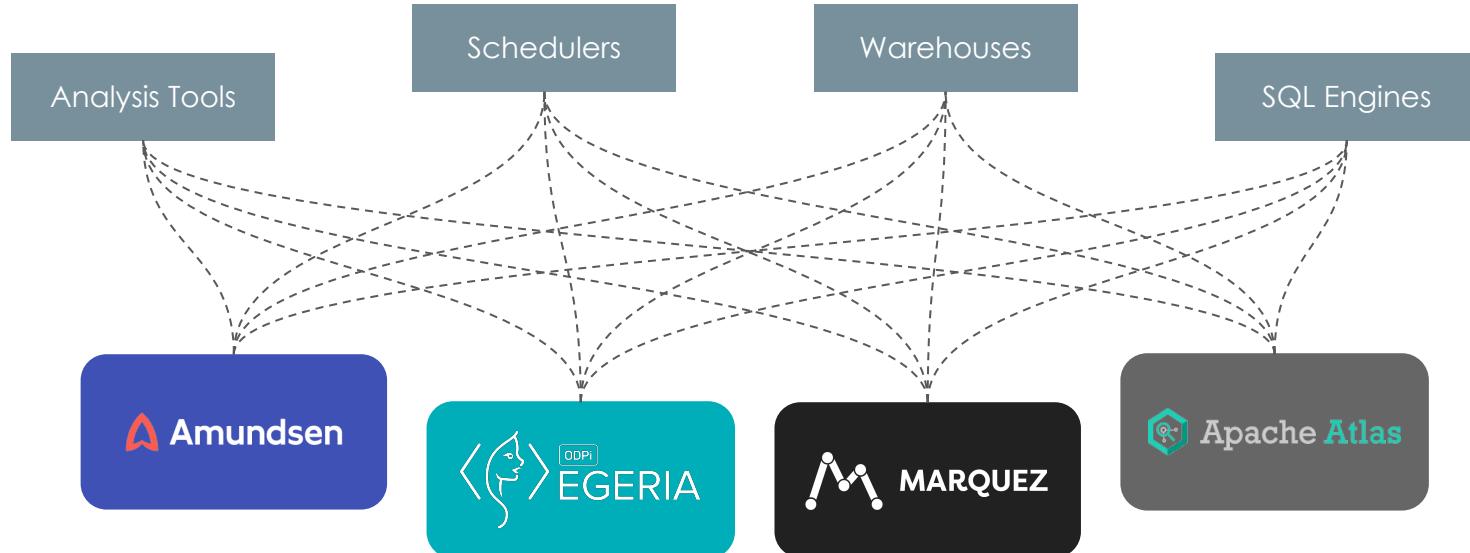
The OpenLineage Stack



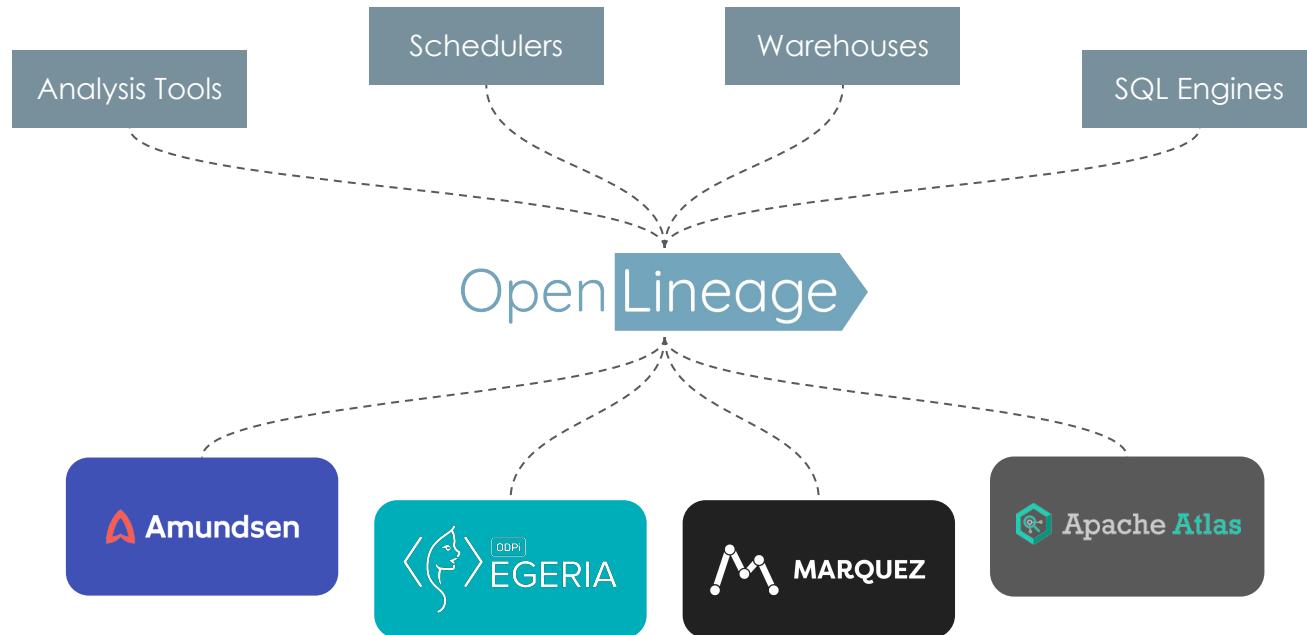
Where OpenLineage potentially fits



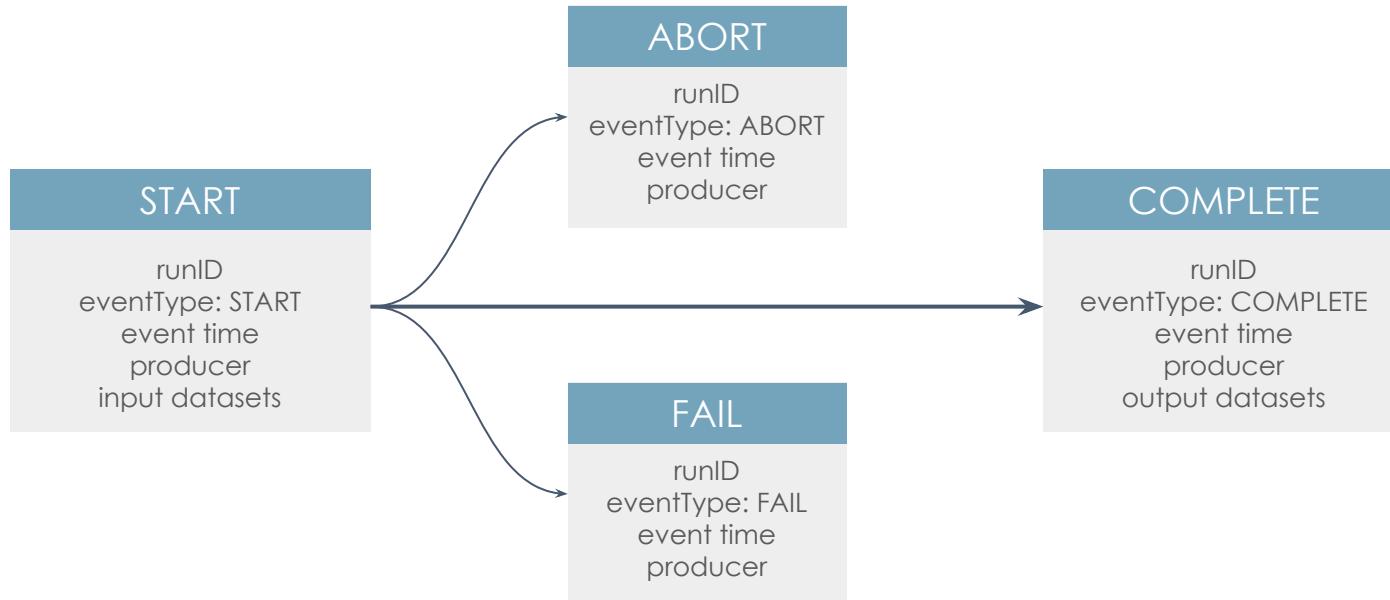
Before OpenLineage



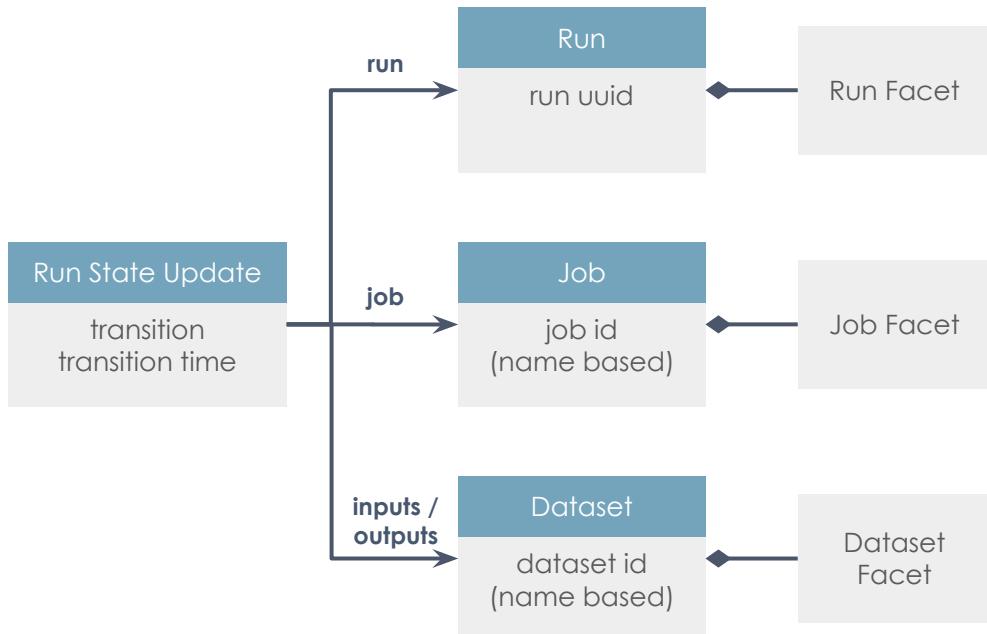
With OpenLineage



Lifecycle of a Job Run



Data model



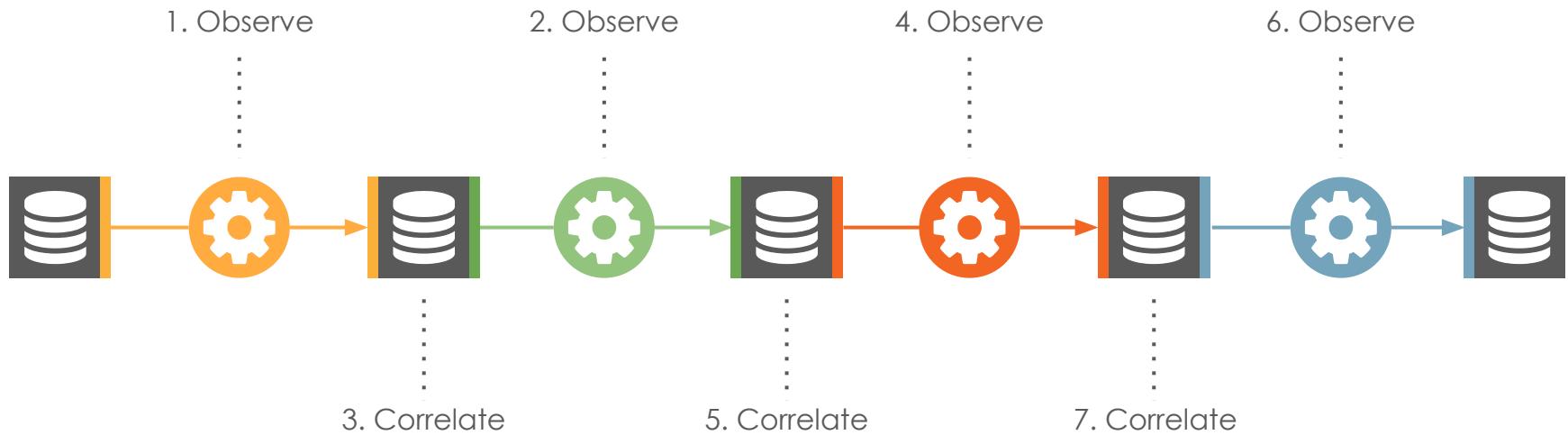
Built around core entities:
Datasets, Jobs, and Runs

Defined as a JSON Schema
spec

Consistent naming for:

- Jobs (*scheduler.job.task*)
- Datasets (*instance.schema.table*)

Lineage is built on correlations



Dataset names are used to stitch together observations of job runs into a lineage graph.

Naming conventions

	Formulae	Examples
Datasets	host + database + table bucket + path host + port + path project + dataset + table	postgres://db.foo.com/metrics.salesorders s3://sales-metrics/orders.csv hdfs://stg.foo.com:salesorders.csv bigquery:metrics.sales.orders
Jobs	namespace + name namespace + project + name	staging.load_orders_from_csv prod.orders_etl.count_orders
Runs	Client-provided UUID	1c0386aa-0979-41e3-9861-3a330623effa

Extending the model with Facets

Facets are atomic pieces of metadata attached to core entities.

Self-documenting

Facets can be given unique,
memorable names

Familiar

Facets are defined using JSON
schema objects

Flexible

Facets can be attached to any
core entity: Job, Dataset & Run

Scalable

Prefixes on names are used to
establish discrete namespaces

Facet examples

Dataset:

- Stats
- Schema
- Version
- Quality metrics

Job:

- Source code
- Dependencies
- Source control
- Query plan

Run:

- Scheduled time
- Batch ID
- Query profile
- Params

3

Setting up
Marquez

Checking out the Marquez project

```
rturk@maxwell:~/projects/workshops/airflow/e1-marquez/marquez

apple ~ p workshops/a/e1-marquez main git clone git@github.com:Marquez
Project/marquez.git
Cloning into 'marquez'...
remote: Enumerating objects: 21854, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 21854 (delta 2), reused 6 (delta 0), pack-reused 21844
Receiving objects: 100% (21854/21854), 20.87 MiB | 26.12 MiB/s, done.
Resolving deltas: 100% (12855/12855), done.

apple ~ p workshops/a/e1-marquez main cd marquez ✓
apple ~ p workshops/a/e1/marquez main ✓
```

Starting up Marquez

```
./docker/up.sh -s

marquez_utils
marquez_db-init
41bd218c1b8bd488b56312780142ed9c099eef49c1f6c8b6a1ca7864470d7b12
marquez-volume-helper
Creating marquez-db ... done
Creating marquez-api ... done
Creating marquez-web ... done
Creating seed-marquez-with-metadata ... done
Attaching to marquez-db, marquez-api, marquez-web, seed-marquez-with-metadata
marquez-db      | The files belonging to this database system will be owned by u
ser "postgres".
marquez-db      | This user must also own the server process.
marquez-db      |
marquez-db      | The database cluster will be initialized with locale "en_US.ut
f8".
marquez-api     | wait-for-it.sh: waiting 15 seconds for db:5432
marquez-db      | The default database encoding has accordingly been set to "UTF
```

About the Marquez start script

```
docker/up.sh --seed
```

Load the database with seed data

After starting Marquez, simulate a series of lineage events for a fictional food delivery service pipeline. Good for exploring the Marquez UI + the OpenLineage data model and API.

```
docker/up.sh --detach
```

Run in detached mode

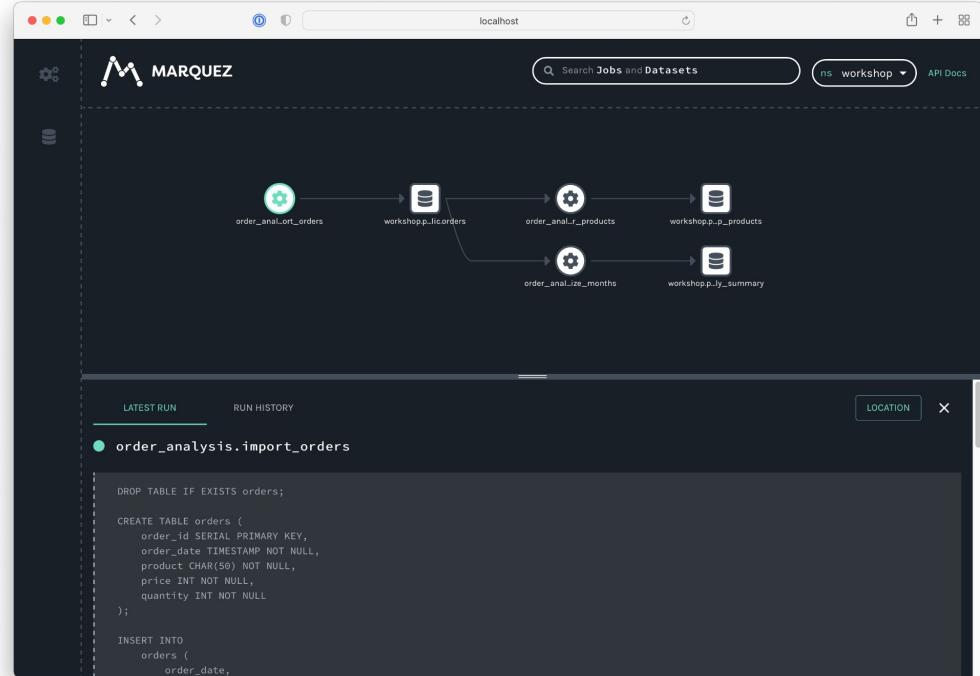
This will cause everything to run in the background (cool!) but also it won't show logs (aww!)

```
docker/up.sh --build
```

Build from source

Build everything, instead of grabbing the latest images from Docker Hub. For development.

Our first look at the Marquez UI



4

Interacting with the Lineage API

Starting a job run

```
rturk@mastro:~/projects/workshops/airflow/e2-lineage-api
(~/p/workshops/a/e2-lineage-api) main bat -p json/startjob.json
{
  "eventType": "START",
  "eventTime": "2020-12-28T19:52:00.001+10:00",
  "run": {
    "runId": "d46e465b-d358-4d32-83d4-df660ff614dd"
  },
  "job": {
    "namespace": "my-namespace",
    "name": "my-job"
  },
  "inputs": [
    {
      "namespace": "my-namespace",
      "name": "my-input"
    }
  ],
  "producer": "https://github.com/OpenLineage/OpenLineage/blob/v1-0-0/client"
}

(~/p/workshops/a/e2-lineage-api) main curl -X POST http://localhost:5000/api/v1/lineage \
-H 'Content-Type: application/json' \
-d @json/startjob.json

(~/p/workshops/a/e2-lineage-api) main
```

Completing a job run

```
rturk@mastro:~/projects/workshops/airflow/e2-lineage-api
(~/p/workshops/a/e2-lineage-api) ✘ main bat -p json/completejob.json
{
  "eventType": "COMPLETE",
  "eventTime": "2020-12-28T20:52:00.001+10:00",
  "run": {
    "runId": "d46e465b-d358-4d32-83d4-df660ff614dd"
  },
  "job": {
    "namespace": "my-namespace",
    "name": "my-job"
  },
  "outputs": [
    {
      "namespace": "my-namespace",
      "name": "my-output"
    }
  ],
  "producer": "https://github.com/OpenLineage/OpenLineage/blob/v1-0-0/client"
}

(~/p/workshops/a/e2-lineage-api) ✘ main curl -X POST http://localhost:5000/api/v1/lineage \
-H 'Content-Type: application/json' \
-d @json/completejob.json

(~/p/workshops/a/e2-lineage-api) ✘ main
```

Example: viewing a job run

The screenshot shows the Marquez UI interface. At the top, there is a navigation bar with icons for settings, search, and user profile, followed by the text "localhost". Below the navigation bar is the Marquez logo and a search bar labeled "Search Jobs and Datasets". A dropdown menu shows "my-namespace" and "API Docs". On the left side, there are two icons: a gear and a database. In the center, there is a diagram of a data pipeline. It starts with a database icon labeled "my-input", followed by a green circular icon with a gear labeled "my-job", and ends with a database icon labeled "my-output". Arrows connect the three components. Below the diagram, there is a timeline with two tabs: "LATEST RUN" and "RUN HISTORY". The "LATEST RUN" tab is selected, showing a single row for "my-job". The table has columns: ID, State, Created At, Started At, Ended At, and Duration. The data for the latest run is:

ID	State	Created At	Started At	Ended At	Duration
d46e465b-d358-4d32-83d4-df660ff614dd	COMPLETED	Dec 28, 2020 04:52am	Dec 28, 2020 04:52am	Dec 28, 2020 05:52am	0m 0s

On the right side of the table, there is a "LOCATION" button with an "X" icon.

```
#!/usr/bin/env python3

from openlineage.client.run import RunEvent, RunState, Run, Job, Dataset
from openlineage.client.client import OpenLineageClient
from datetime import datetime
from uuid import uuid4

# Initialize the OpenLineage client
client = OpenLineageClient.from_environment()

# Specify the producer of this lineage metadata
producer = "https://github.com/OpenLineage/workshops"

# Create some basic Dataset objects for our fictional pipeline
online_orders = Dataset(namespace="workshop", name="online_orders")
mail_orders = Dataset(namespace="workshop", name="mail_orders")
orders = Dataset(namespace="workshop", name="orders")

# Create a Run object with a unique ID
run = Run(str(uuid4()))

# Create a Job object
job = Job(namespace="workshop", name="process_orders")

# Emit a START run event
client.emit(
    RunEvent(
        RunState.START,
        datetime.now().isoformat(),
        run, job, producer
    )
)

#
# This is where our application would do the actual work :)
#

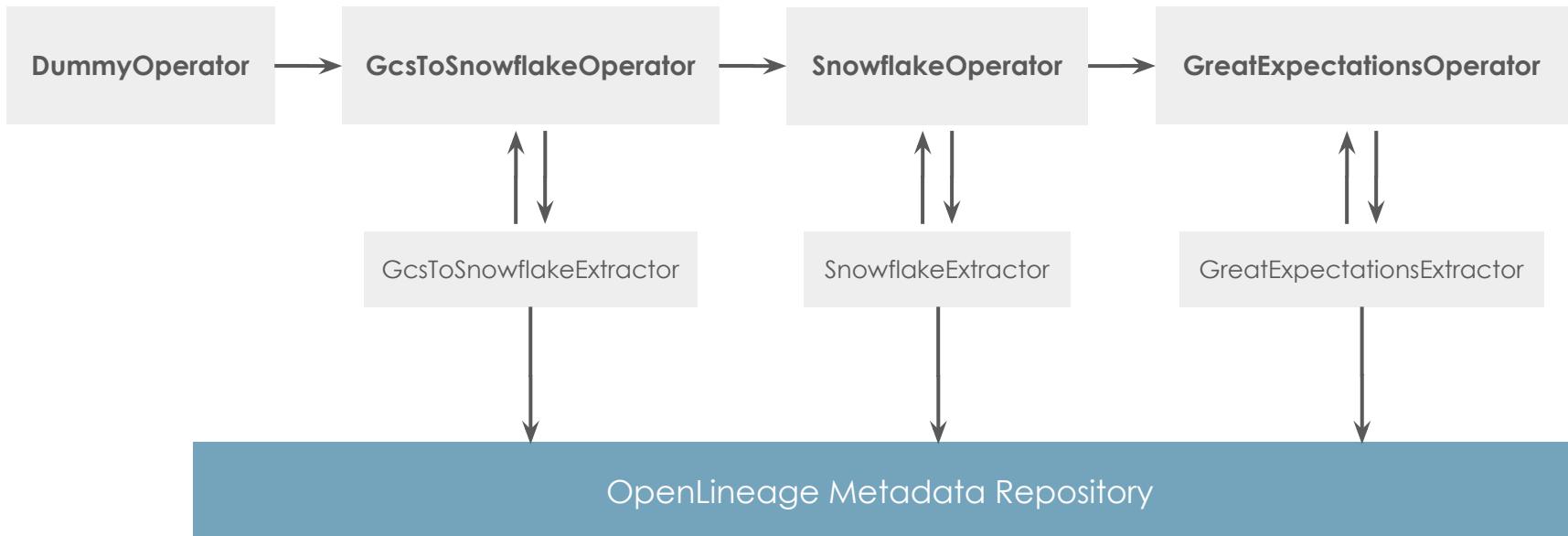
# Emit a COMPLETE run event
client.emit(
    RunEvent(
        RunState.COMPLETE,
        datetime.now().isoformat(),
        run, job, producer,
        inputs=[online_orders, mail_orders],
        outputs=[orders],
    )
)
```

Using the Python client

5

OpenLineage &
Airflow

Lineage is collected with Extractors



Enabling the integration (before 2.3)

Airflow
2.1+

```
apple ~> ~/projects/astro bat -p .env
AIRFLOW__LINEAGE__BACKEND=openlineage.lineage_backend.OpenLineageBackend
✓
apple ~> ~/projects/astro
```

Airflow
1.10+

```
apple ~> ~/p/astro git ⌂ main !1 git diff
diff --git a/mydag.py b/mydag.py
index 44d86a5..bc97743 100644
--- a/mydag.py
+++ b/mydag.py
@@ -1,3 +1,3 @@
 mydag.py

-from airflow import DAG
+from openlineage.airflow import DAG
apple ~> ~/p/astro git ⌂ main !1
```

Available Extractors

BigQueryOperator

PostgresOperator

SnowflakeOperator

GreatExpectationsOperator

PythonOperator

BashOperator

Registering new Extractors

```
MacBook-Pro:~ projects$ bat -p .env  
OPENLINEAGE_EXTRACTOR_MyExtractor=full.path.to.MyExtractor
```

```
MacBook-Pro:~ projects$
```

How to build an Extractor

```
from openlineage.airflow.extractors.base import (
    BaseExtractor,
    TaskMetadata
)
from openlineage.client.run import Dataset
from typing import List

class MyExtractor(BaseExtractor):
    @classmethod
    def get_operator_classnames(cls) -> List[str]:
        return ['MyOperator', 'MyOtherOperator']

    def extract(self) -> TaskMetadata:
        return TaskMetadata(
            name=f"{self.operator.dag_id}.{self.operator.task_id}",
            inputs=[Dataset(...)],
            outputs=[Dataset(...)]
        )
```

Extend the
OpenLineage
BaseExtractor

Extract lineage
metadata for
these operators

Record **these**
Dataset objects
as inputs/outputs

6

Setting up Airflow with OpenLineage

```

git diff 7a255ea3e1b08130dc18efed7d26267e34ca5e977 docker-compose.yaml

diff --git a/airflow/e3-airflow/docker-compose.yaml b/airflow/e3-airflow/docker-compose.yaml
index f50f4c2..b2c3e1a 100644
--- a/airflow/e3-airflow/docker-compose.yaml
+++ b/airflow/e3-airflow/docker-compose.yaml
@@ -44,8 +44,8 @@ x-airflow-common:
    # In order to add custom dependencies or upgrade provider packages you can use your extended image.
    # Comment the image line, place your Dockerfile in the directory where you placed the docker-compose.yaml
    # and uncomment the "build" line below. Then run docker-compose build to build the images.
- image: ${AIRFLOW_IMAGE_NAME:-apache/airflow:2.3.0}
- # build: .
+ # image: ${AIRFLOW_IMAGE_NAME:-apache/airflow:2.3.0}
+ build: .
environment:
  airflow-common-env
  AIRFLOW_CORE_EXECUTOR: CeleryExecutor
@@ -56,9 +56,12 @@ x-airflow-common:
  AIRFLOW_CELERY_BROKER_URL: redis://:@redis:6379/0
  AIRFLOW_CORE_FERNET_KEY: ''
  AIRFLOW_CORE_DAGS_ARE_PAUSED_AT_CREATION: 'true'
- AIRFLOW_CORE_LOAD_EXAMPLES: 'true'
+ AIRFLOW_CORE_LOAD_EXAMPLES: 'false'
  AIRFLOW_API_AUTH_BACKENDS: 'airflow.api.auth.backend.basic_auth'
  _PIP_ADDITIONAL_REQUIREMENTS: ${_PIP_ADDITIONAL_REQUIREMENTS:-}
  OPENLINEAGE_URL: 'http://marquez-api:5000'
+ OPENLINEAGE_NAMESPACE: 'workshop'
+ AIRFLOW_CONN_WORKSHOP: 'postgresql://workshop:workshop@workshop-db/workshop'
volumes:
  - ./dags:/opt/airflow/dags
  - ./logs:/opt/airflow/logs
@@ -86,6 +89,15 @@ services:
  retries: 5
  restart: always

+ workshop-db:
+   image: postgres:13
+   environment:
+     POSTGRES_USER: workshop
+     POSTGRES_PASSWORD: workshop
+     POSTGRES_DB: workshop
+   ports:
+     - 5432:5432
+
redis:
  image: redis:latest
  expose:
@@ -268,5 +280,10 @@ services:
  airflow-init:
    condition: service_completed_successfully

+networks:
+ default:
+   external: true
+   name: marquez_default
+
volumes:
  postgres-db-volume:
(END)

```

Build our own Airflow docker image with OpenLineage installed

Disable examples

Set OpenLineage endpoint to Marquez and provide namespace

Configure Airflow connection for our workshop database

Deploy our workshop database :)

Deploy in the same network as Marquez, for convenience

docker-compose.yml

Dockerfile

```
rturk@mastro:~/projects/workshops/airflow/e3-airflow
FROM apache/airflow:2.3.0
RUN pip install --no-cache-dir openlineage-airflow
RUN pip install --no-cache-dir openlineage-sql

rturk@mastro:~/projects/workshops/airflow/e3-airflow
```

Initializing Airflow

```
docker-compose up airflow-init

Apple ) ~ /p/workshops/airflow ) ⚡ main cd e3-airflow
Apple ) ~ /p/workshops/a/e3-airflow ) ⚡ main docker-compose up airflow-init
[+] Running 20/20
:: redis Pulled
:: b2c9187c1872 Pull complete          8.6s
:: 001678ad0f6a Pull complete          3.9s
:: b6402ea36002 Pull complete          4.3s
:: 9d24e4746f83 Pull complete          5.2s
:: 931c944d57b3 Pull complete          5.3s
:: 931c944d57b3 Pull complete          5.3s
:: postgres Pulled                     10.2s
:: dfdd5ffb2577 Pull complete          2.5s
:: 504b51ce876f Pull complete          2.7s
:: 6d167d70ff7f Pull complete          2.7s
:: de307e992d6e Pull complete          2.9s
:: f5268e872817 Pull complete          3.2s
:: 974a8608df9f Pull complete          3.3s
:: 010114fdc649 Pull complete          3.4s
:: d8bb5c9771a3 Pull complete          3.5s
:: cc6c4daadd98 Pull complete          7.0s
:: 88a57f481a28 Pull complete          7.0s
:: b864b30dbdde Pull complete          7.0s
:: 90b12271a51c Pull complete          7.1s
:: 05a512ca1027 Pull complete          7.1s
[+] Building 1 4 - [0/0] FINISHED
```

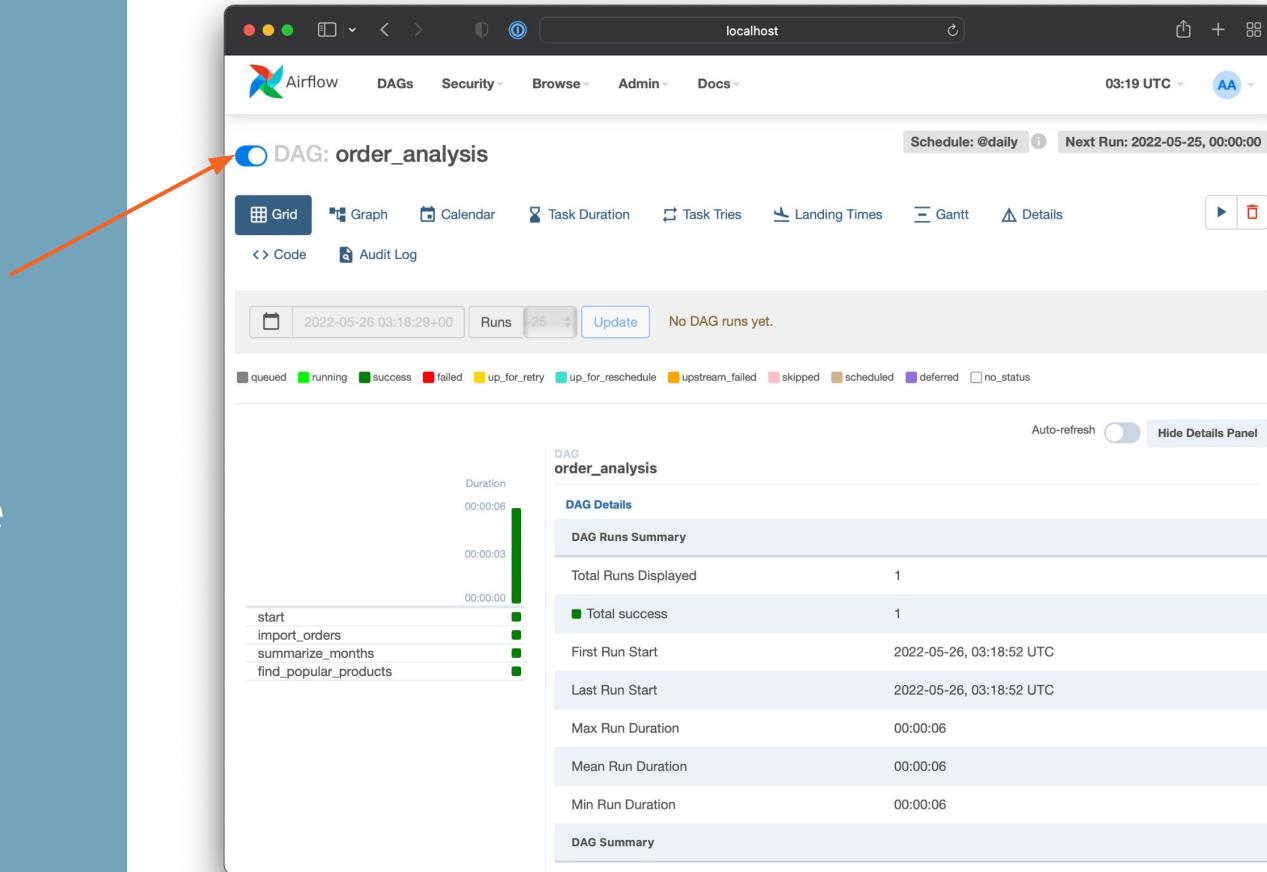
Starting up Airflow

```
apple ~ ~/workshops/a/e3-airflow main docker-compose up
[+] Building 0.9s (16/16) FINISHED
=> [e3-airflow_airflow-webserver internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 32B 0.0s
=> [e3-airflow_flower internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 32B 0.0s
=> [e3-airflow_airflow-scheduler internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 32B 0.0s
=> [e3-airflow_airflow-triggerer internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 32B 0.0s
=> [e3-airflow_airflow-worker internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 32B 0.0s
=> [e3-airflow_airflow-webserver internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [e3-airflow_flower internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [e3-airflow_airflow-scheduler internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [e3-airflow_airflow-triggerer internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [e3-airflow_airflow-worker internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [e3-airflow_airflow-webserver internal] load metadata for docker.io/apache/airflow:2.3.0 0.5s
=> [e3-airflow_airflow-scheduler 1/4] FROM docker.io/apache/airflow:2.3.0@sha256:8195064e6b5a2 0.0s
```

7

Running our
example pipeline

Triggering the pipeline



The screenshot shows the Airflow web interface for the DAG: order_analysis. At the top, there is a navigation bar with links for Airflow, DAGs, Security, Browse, Admin, and Docs. The current DAG is highlighted. The status bar indicates the time as 03:19 UTC and the next run as 2022-05-25, 00:00:00. Below the navigation, there are several tabs: Grid, Graph, Calendar, Task Duration, Task Tries, Landing Times, Gantt, Details, Code, and Audit Log. The Grid tab is selected. A large orange arrow points from the left towards the 'Run DAG' button. The main content area shows a timeline for the DAG runs. A single run is listed with a start time of 2022-05-26 03:18:29+00 and a duration of 00:00:06. The tasks listed are start, import_orders, summarize_months, and find_popular_products, all of which are marked as success. To the right, there is a 'DAG Details' panel with sections for DAG Runs Summary, DAG Summary, and a DAG Run Details table. The table shows one total run displayed, with one success, starting at 2022-05-26, 03:18:52 UTC.

Total Runs Displayed	1
Total success	1
First Run Start	2022-05-26, 03:18:52 UTC
Last Run Start	2022-05-26, 03:18:52 UTC
Max Run Duration	00:00:06
Mean Run Duration	00:00:06
Min Run Duration	00:00:06

Thanks :)