Data Lineage

With OpenLineage and Airflow

Ross Turk
Michael Robinson

ASTRONOMER

Agenda

Intro & data lineage strategies	10m
OpenLineage design principles ● Architecture & data model ● Lifecycle of a Job Run	15m
Exercise 1: Setting up Marquez ● Deploying with docker-compose ● Exploring the seed data	20m
Exercise 2: The Lineage API ● Using curl ● Using python	25m
The basics of OpenLineage with Airflow ● Operators & extractors ● Configuration	20m
Exercise 3: Setting up Airflow	20m
Running our pipeline ● Reviewing example DAGs ● Running & verifying the output	30m

Prerequisites

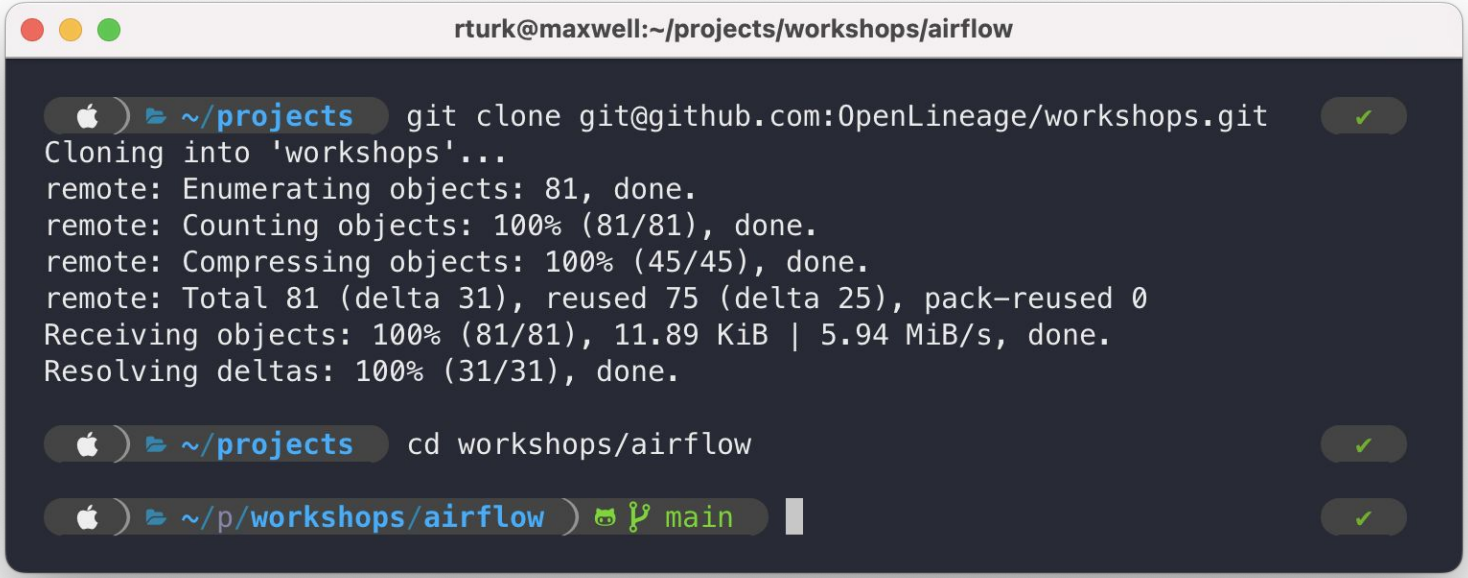
8GB of RAM is recommended for the exercises in this workshop.

1. Install Docker Desktop:
   https://www.docker.com/get-started/

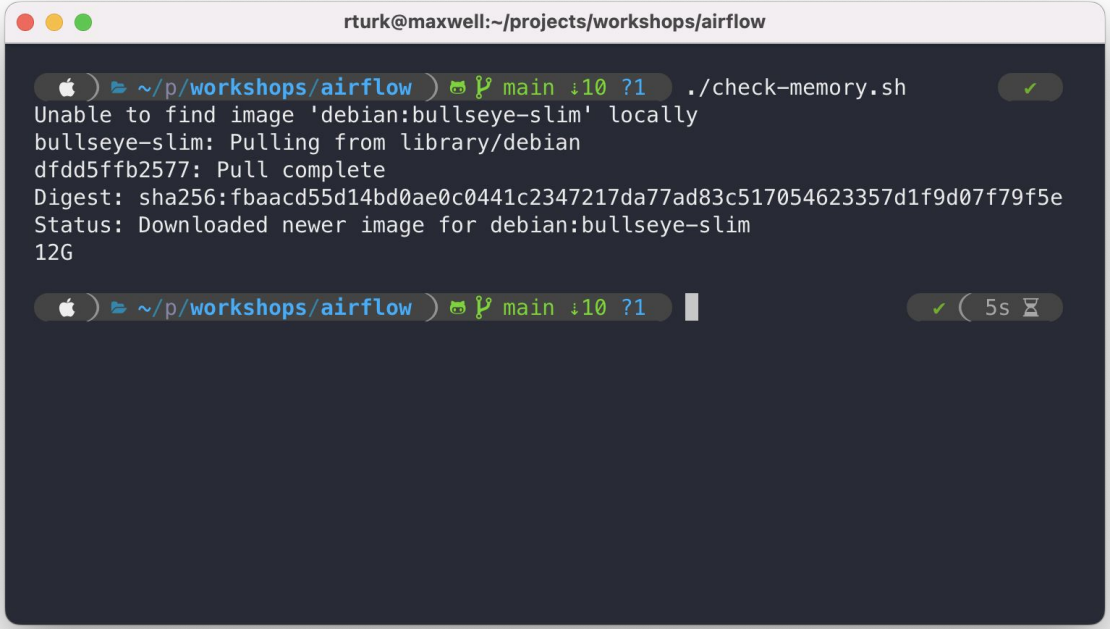
2. Install Git (if it's not already installed):
   https://git-scm.com/downloads

3. Configure Docker Desktop:
   At least **4GB RAM** should be allocated for containers in the Resources section

Checking out the workshop project



Verifying available memory



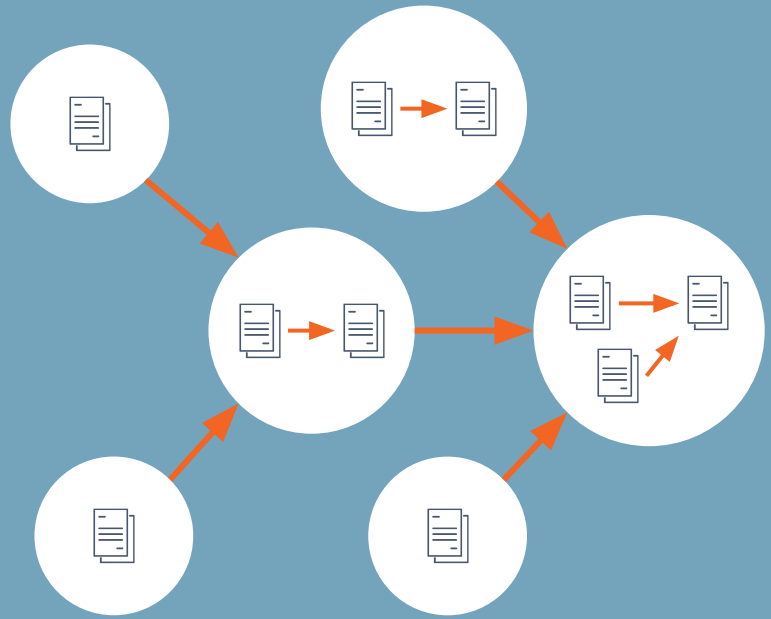
1

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

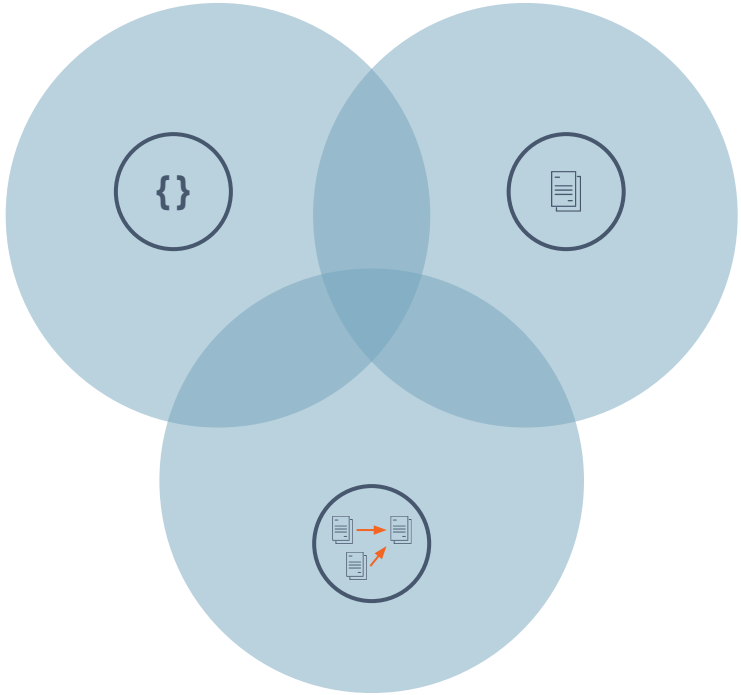


The best time to collect metadata

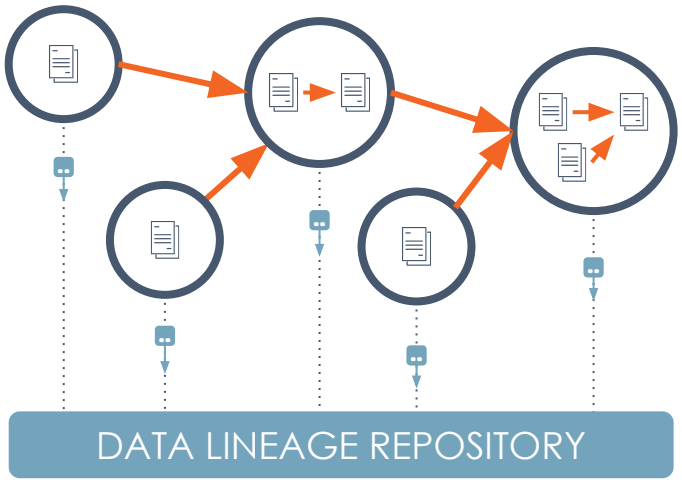
You can try to infer the date and location of an image after the fact…

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

Comparing approaches for gathering lineage

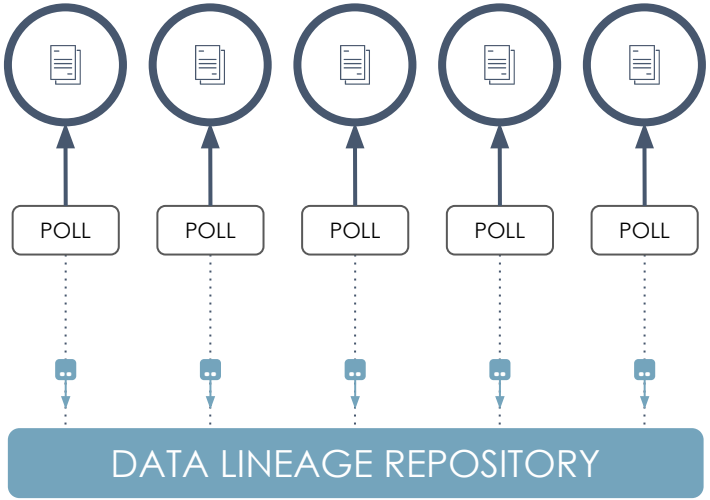


Observe the pipeline



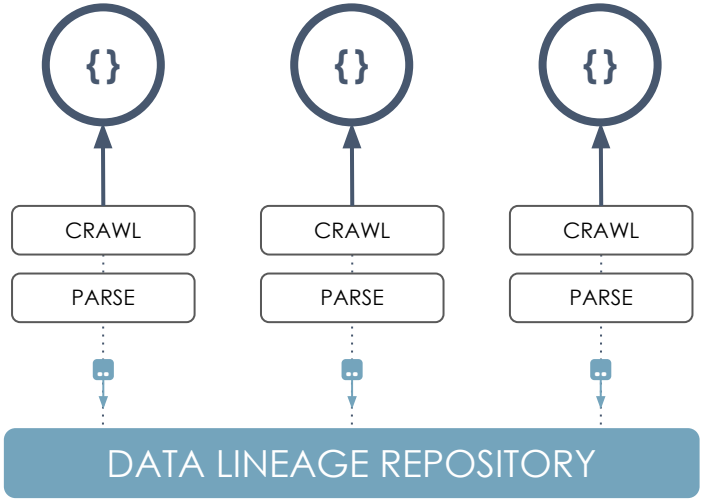
- Integrate with data orchestration systems
- As jobs run, observe the way they affect data
- Report to a lineage metadata repository

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

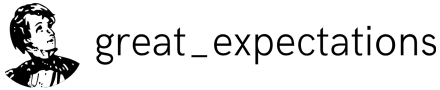
OpenLineage design principles

OpenLineage

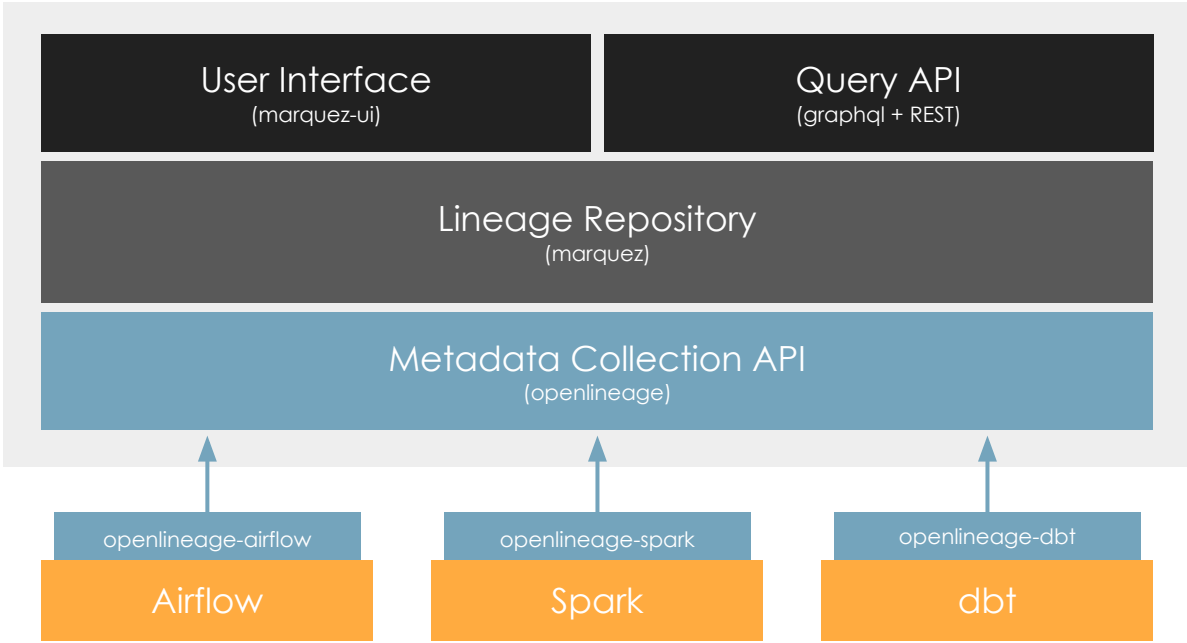
Mission

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

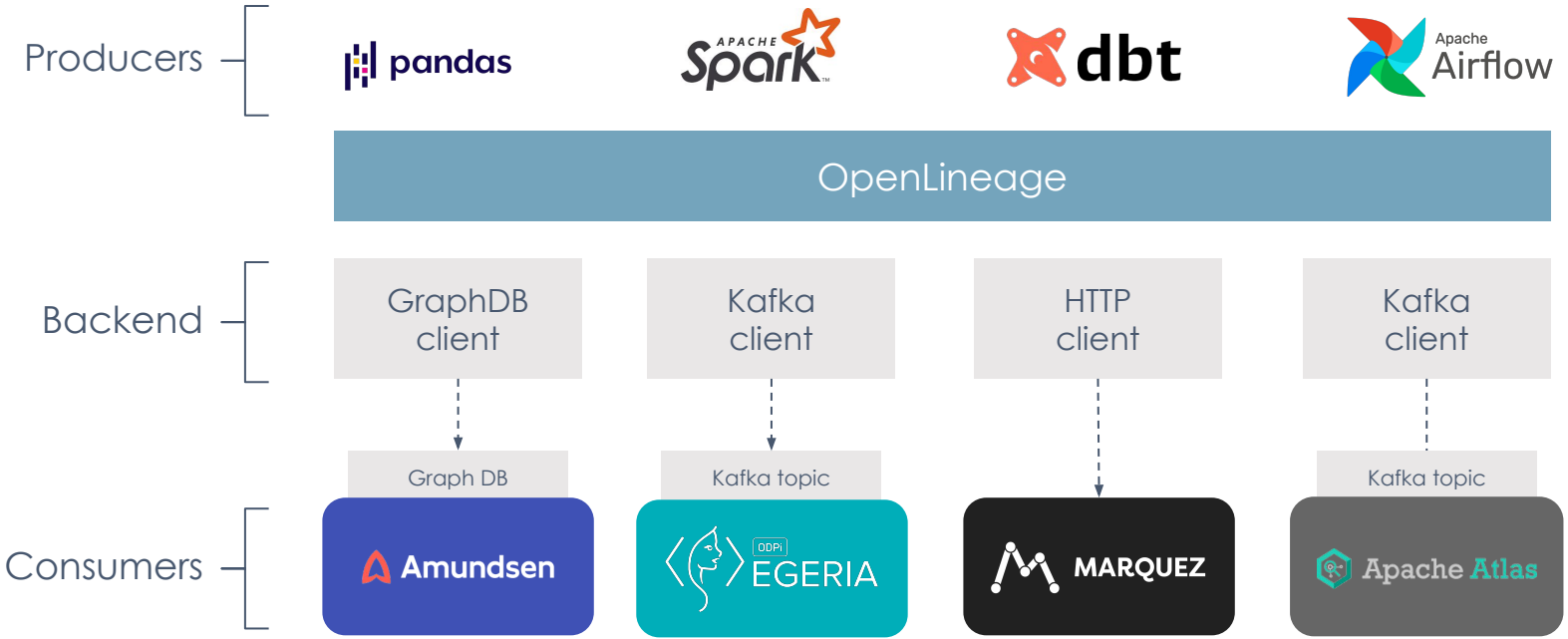
OpenLineage contributors



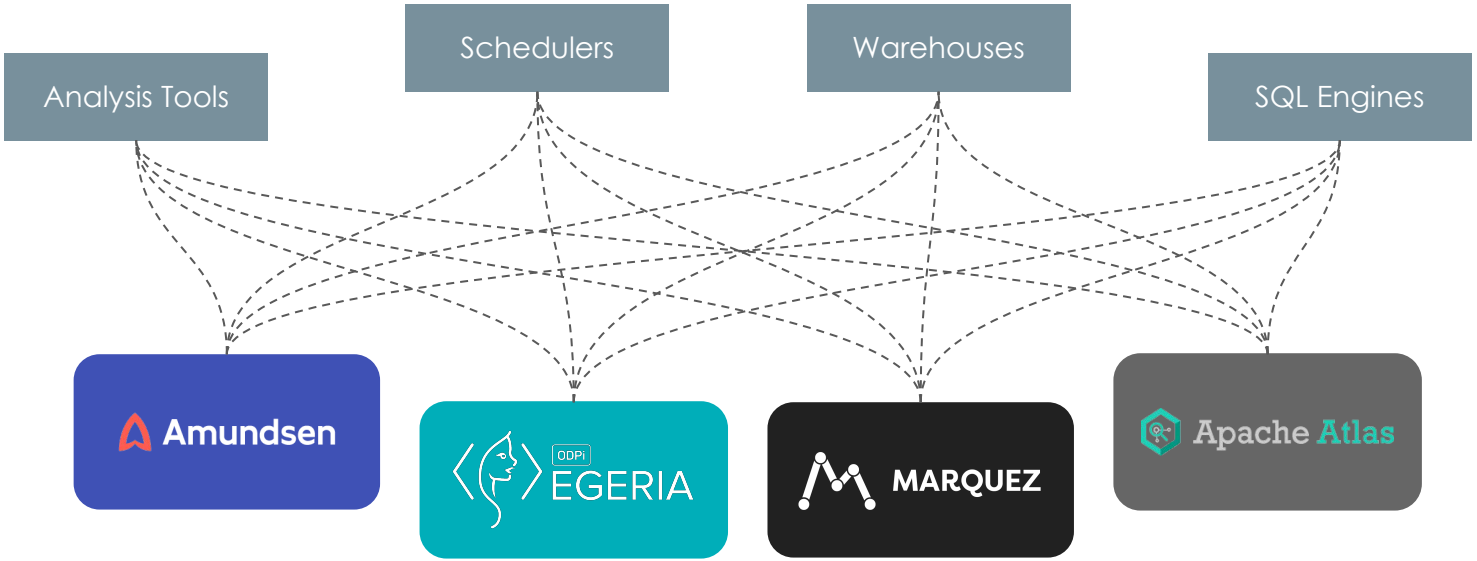
The OpenLineage Stack



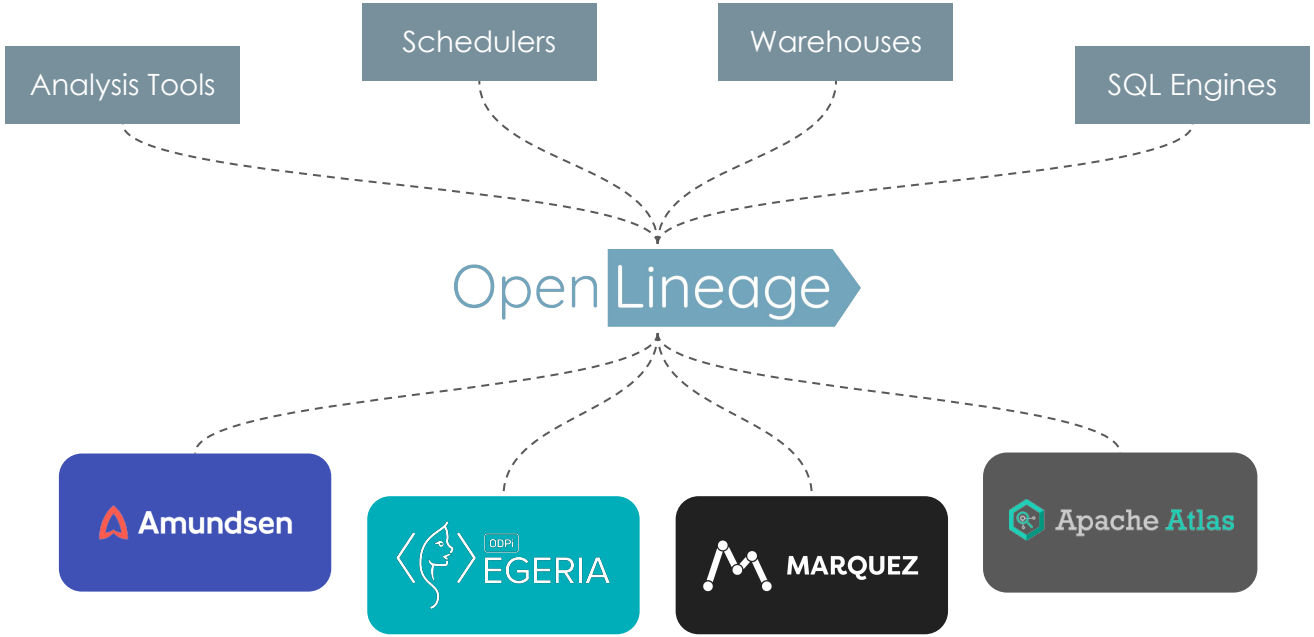
Where OpenLineage potentially fits



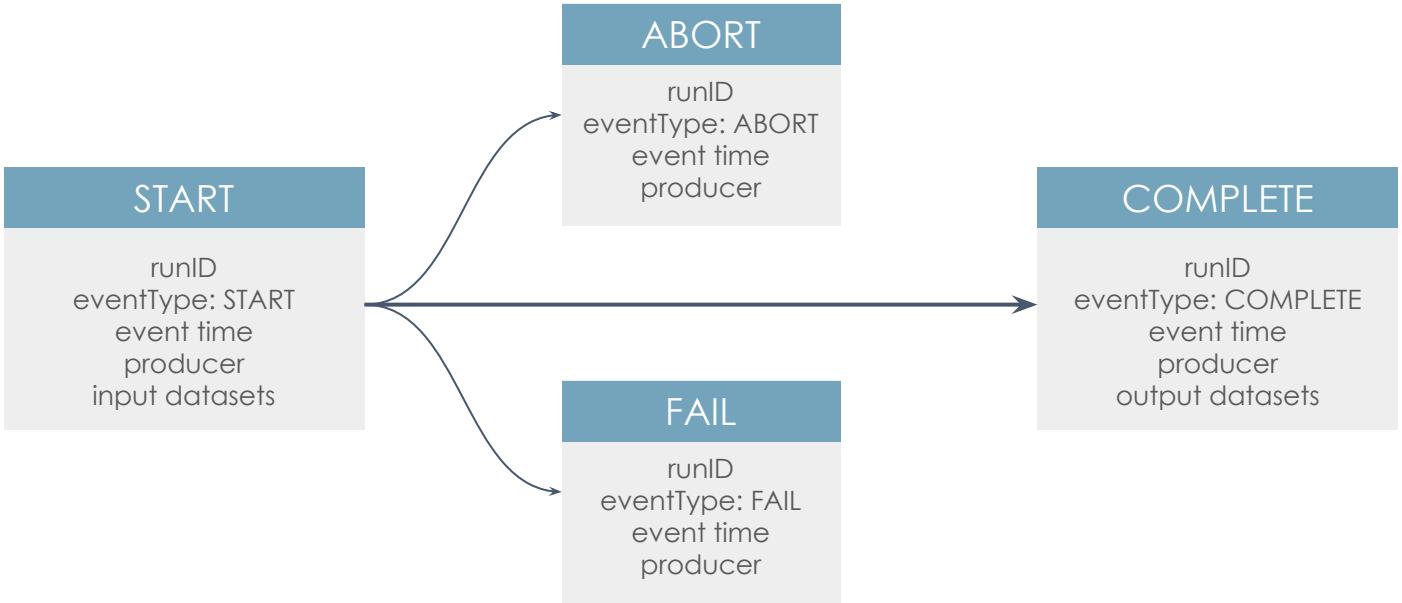
Before OpenLineage



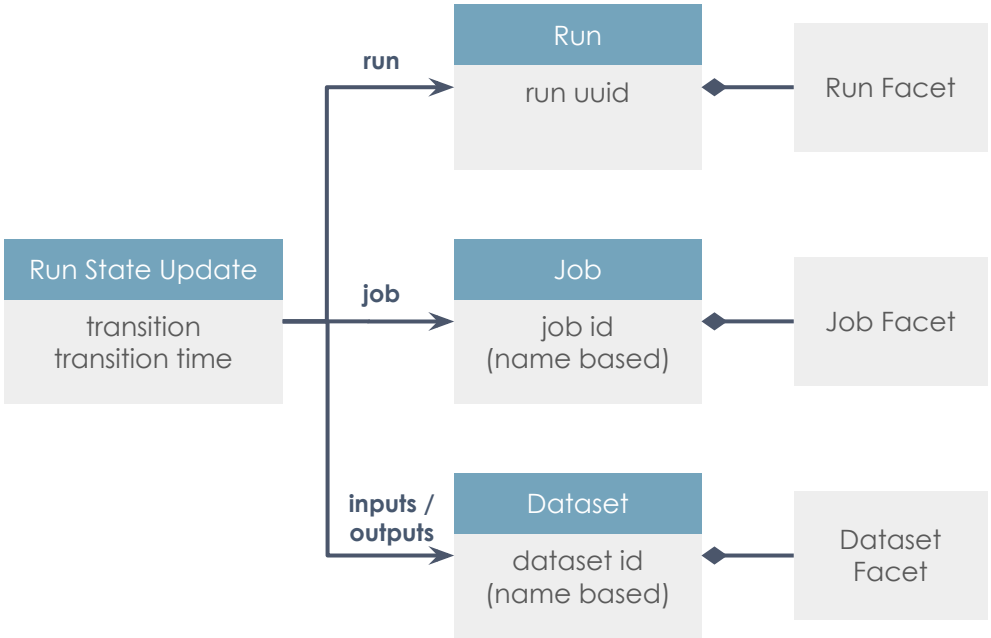
With OpenLineage



Lifecycle of a Job Run



Data model



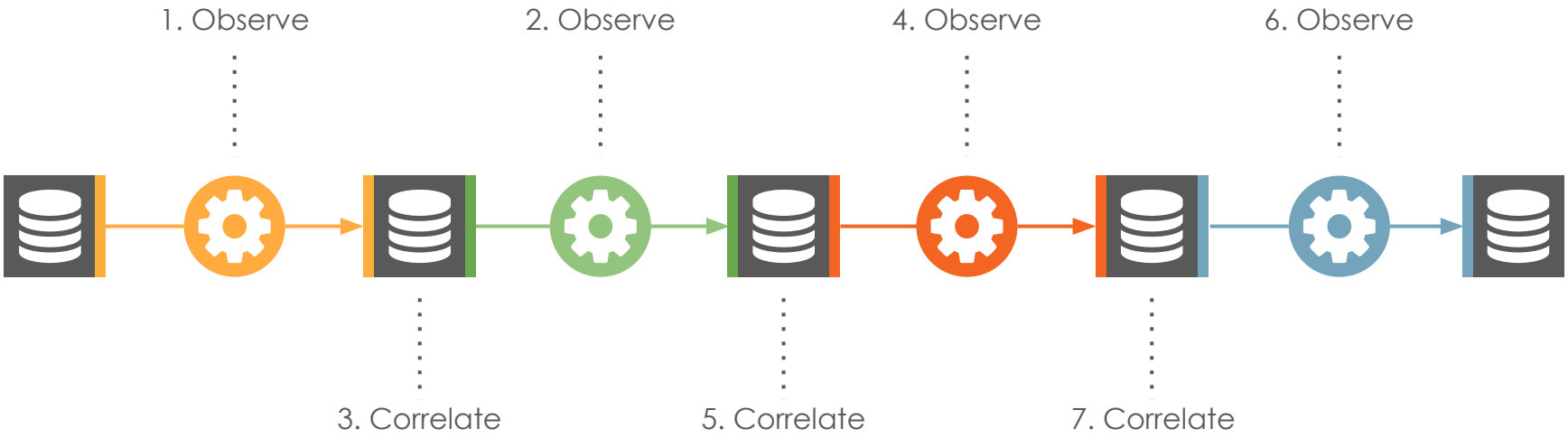
Built around core entities: Datasets, Jobs, and Runs

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

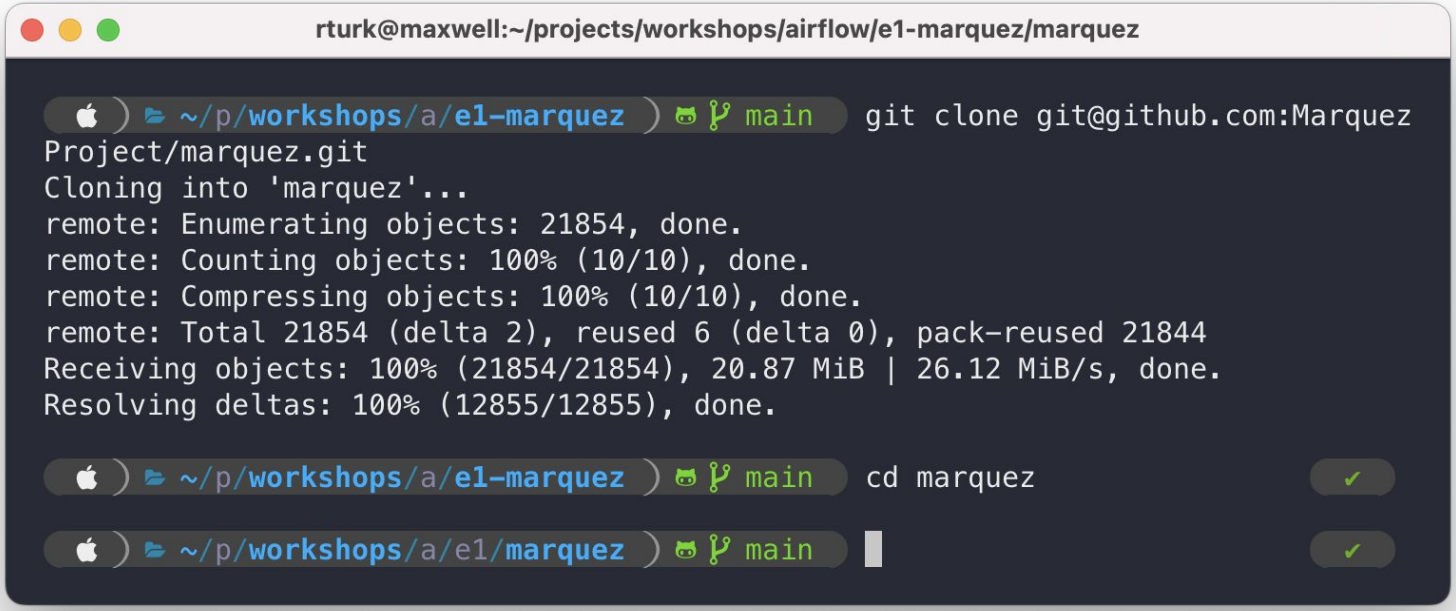
Run:

- Scheduled time
- Batch ID
- Query profile
- Params

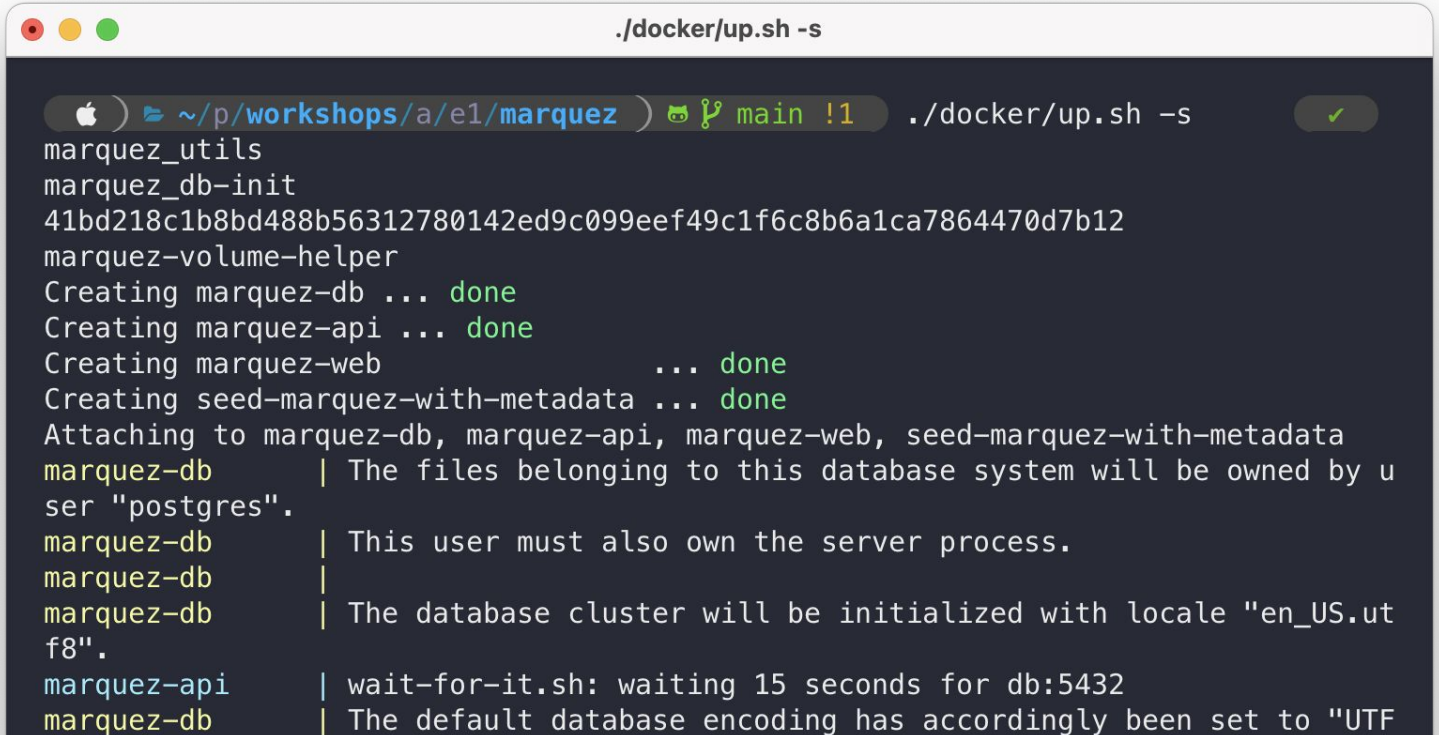
3

Setting up Marquez

Checking out the Marquez project



Starting up Marquez



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

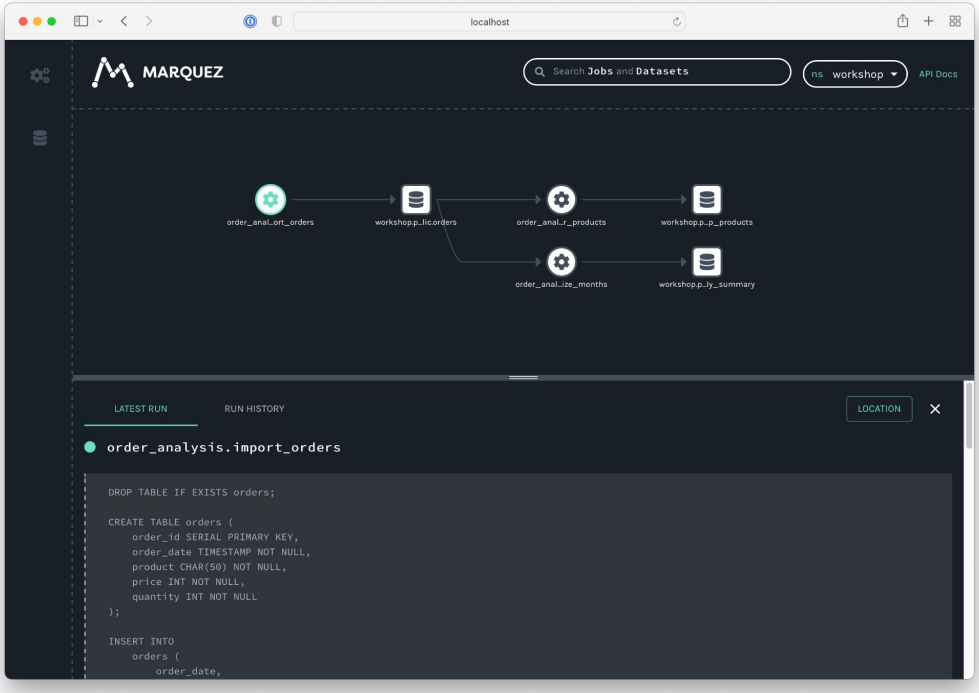
This will cause everything to run in the background (cool!) but also it won't show logs (aww!)

```
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

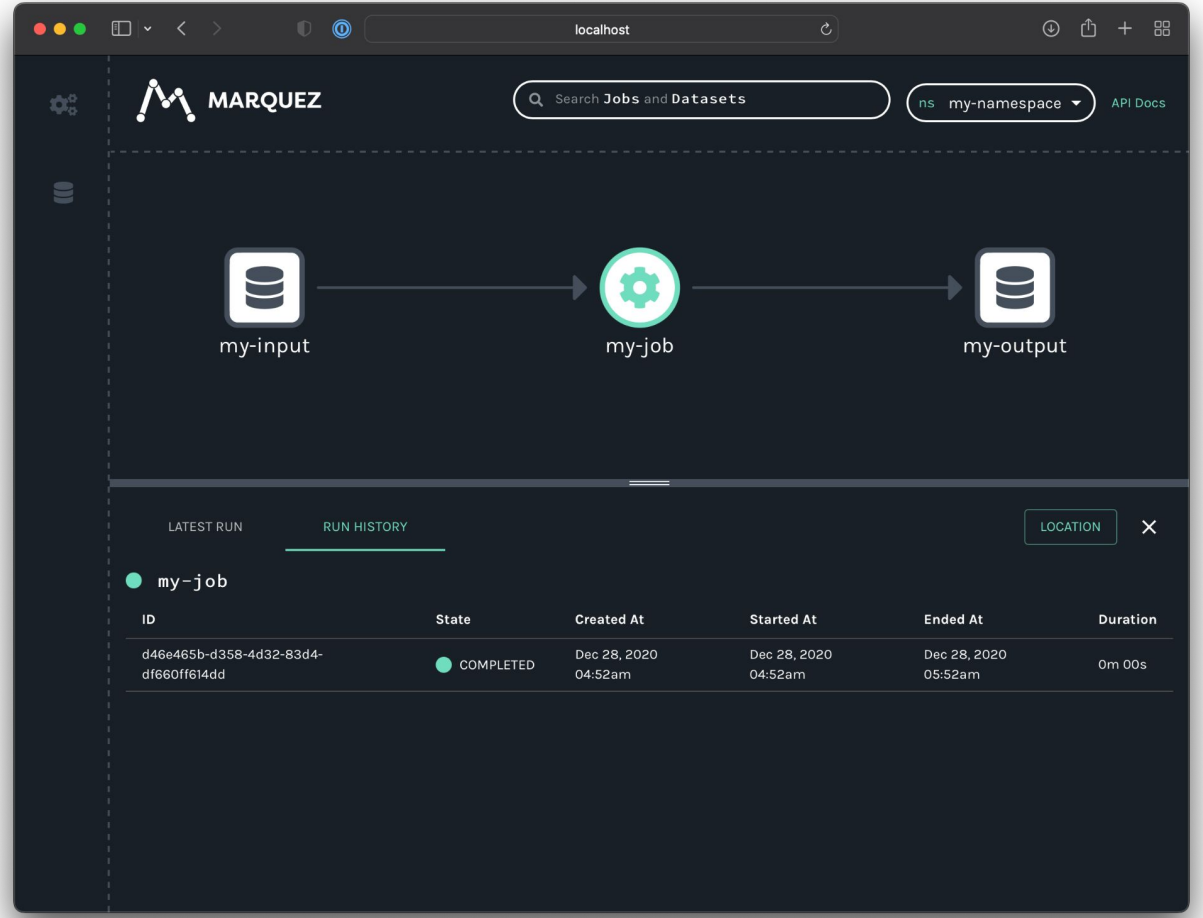
```
~/p/workshops/a/e2-lineage-api main bat -p json/startjob.json
{
  "eventType": "START",
  "eventTime": "2020-12-28T19:52:00.001+10:00",
  "run": {
    "runId": "d46e465b-d358-4d32-83d4-df660ff614dd"
  },
  "job": {
    "namespace": "my-namespace",
    "name": "my-job"
  },
  "inputs": [{
    "namespace": "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



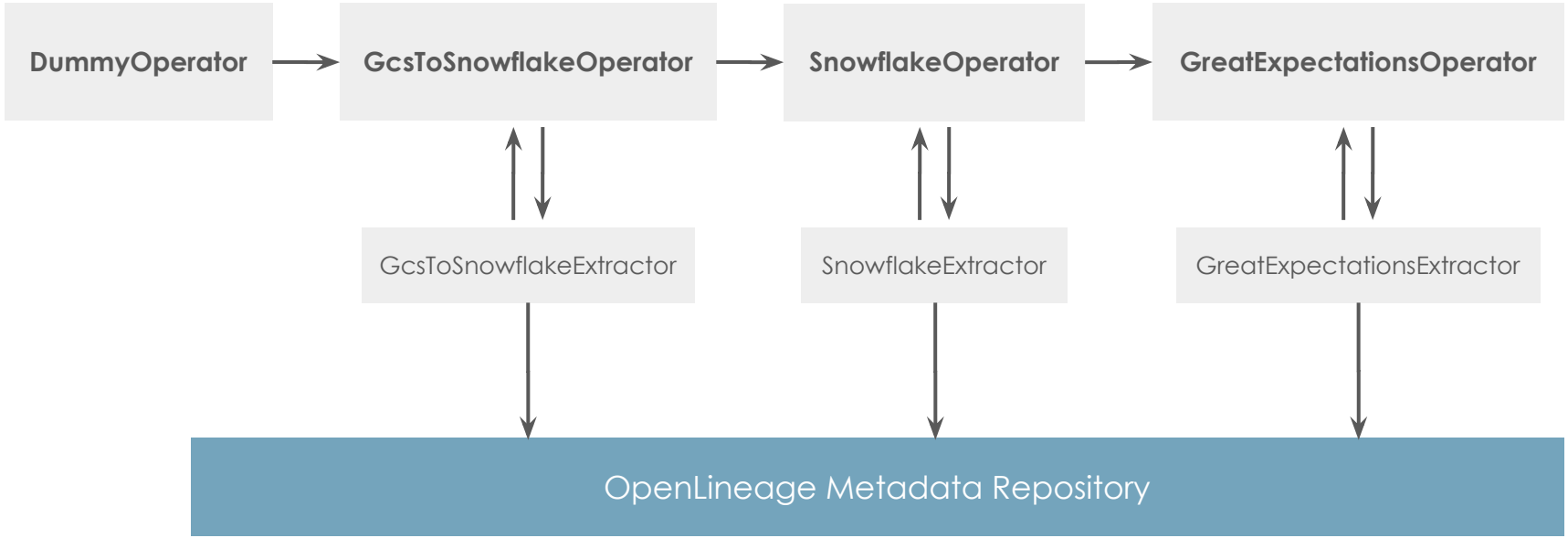
```python
#!/usr/bin/env python3

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+

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

~/projects/astro
```

Airflow 1.10+

```
~/p/astro  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

~/p/astro  main !1
```

Available Extractors

- BigQueryOperator
- PostgresOperator
- SnowflakeOperator
- GreatExpectationsOperator
- PythonOperator
- BashOperator

Registering new Extractors

```
~/projects/astro bat -p .env
OPENLINEAGE_EXTRACTOR_MyOperator=full.path.to.MyExtractor

~/projects/astro
```

How to build an Extractor

```python
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

Build our own Airflow docker image with OpenLineage installed

Disable examples

Set OpenLineage endpoint to Marquez and provide namespace

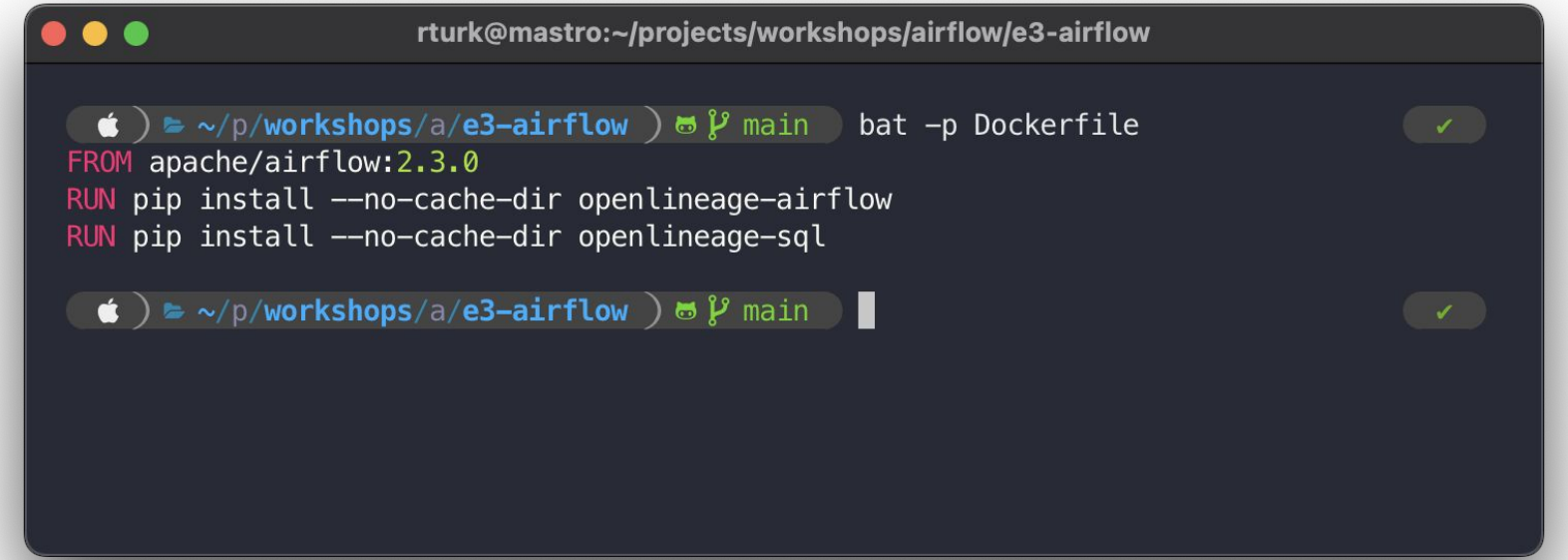
Configure Airflow connection for our workshop database

Deploy our workshop database :)

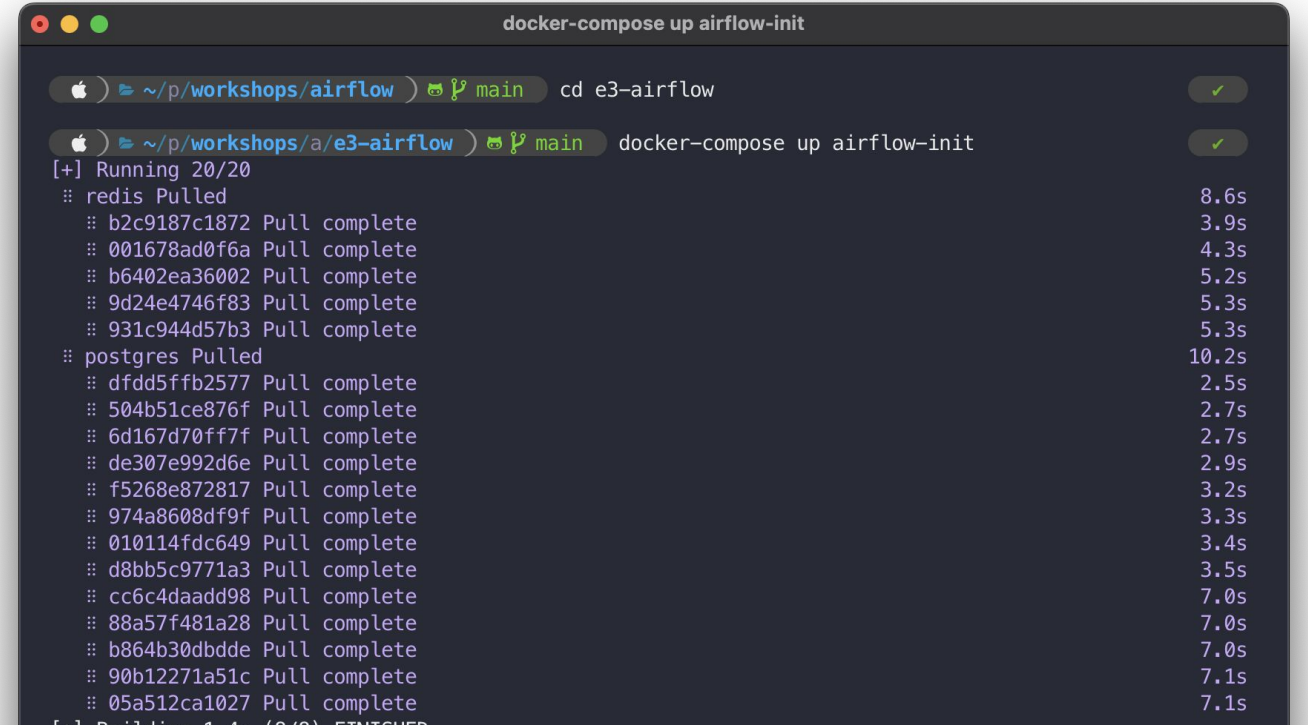
Deploy in the same network as Marquez, for convenience

docker-compose.yml

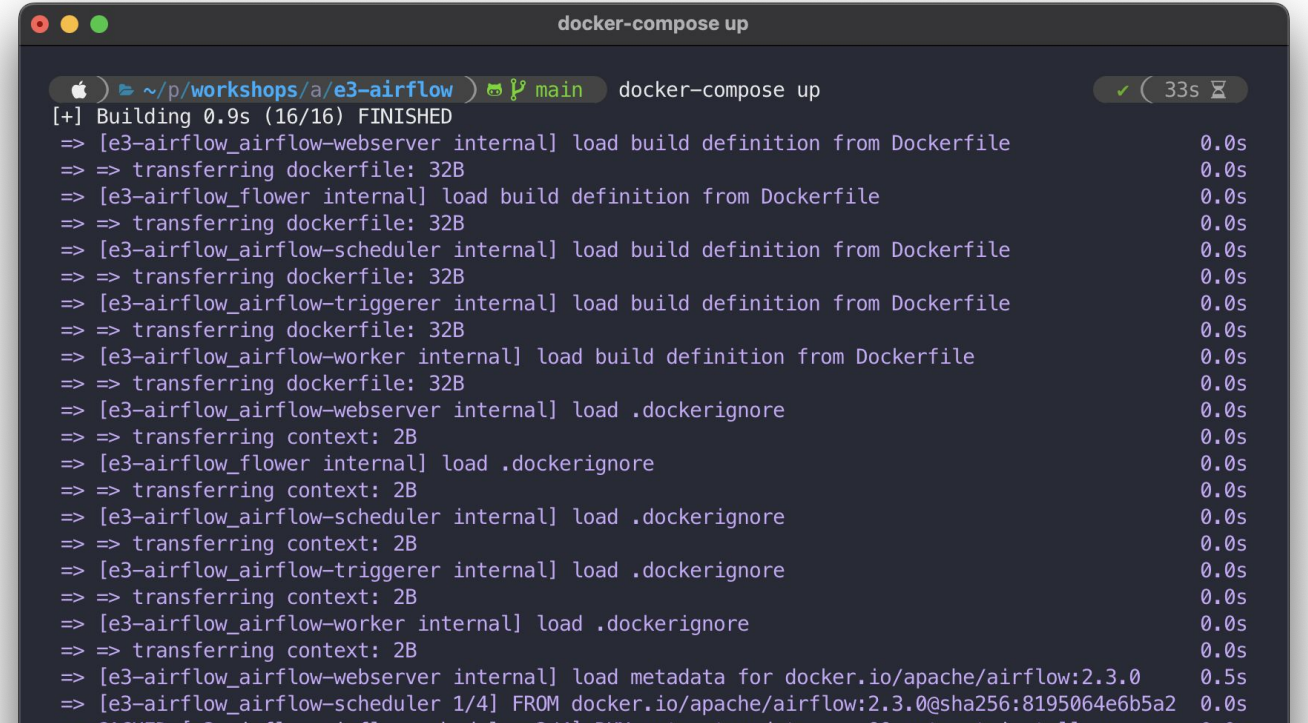
Dockerfile


rturk@mastro:~/projects/workshops/airflow/e3-airflow
~/p/workshops/a/e3-airflow main bat -p Dockerfile
FROM apache/airflow:2.3.0
RUN pip install --no-cache-dir openlineage-airflow
RUN pip install --no-cache-dir openlineage-sql
~/p/workshops/a/e3-airflow main

Initializing Airflow



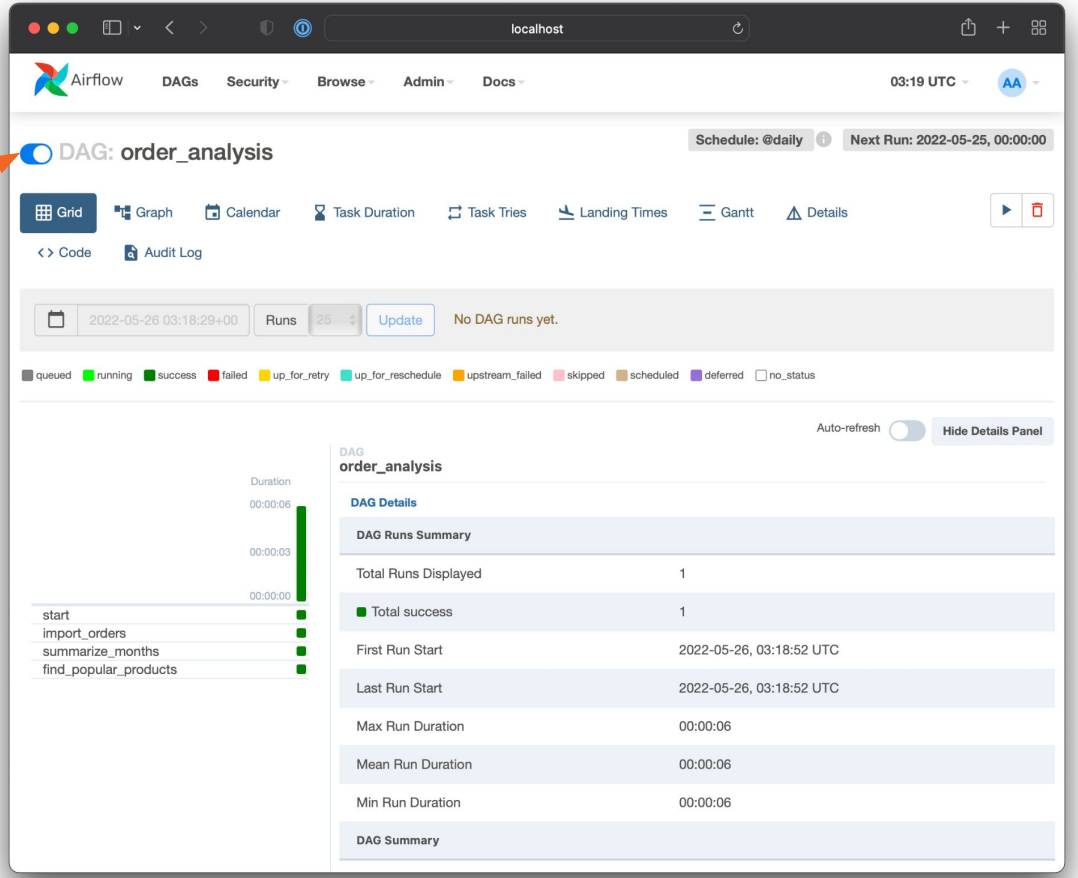
Starting up Airflow



7

Running our example pipeline

Triggering the pipeline



Thanks :)