

# OpenLineage & Airflow: A Deeper Dive

ASTRONOMER

# Our Speakers



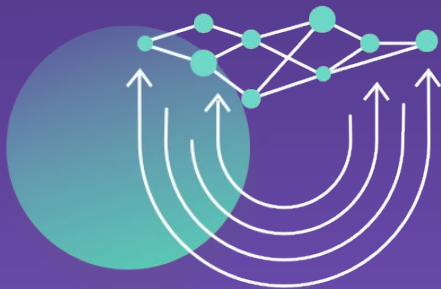
**Ross Turk**  
Senior Director,  
Community



**Michael Collado**  
Staff Software Engineer


# ASTRONOMER

The company building  
**Apache Airflow**



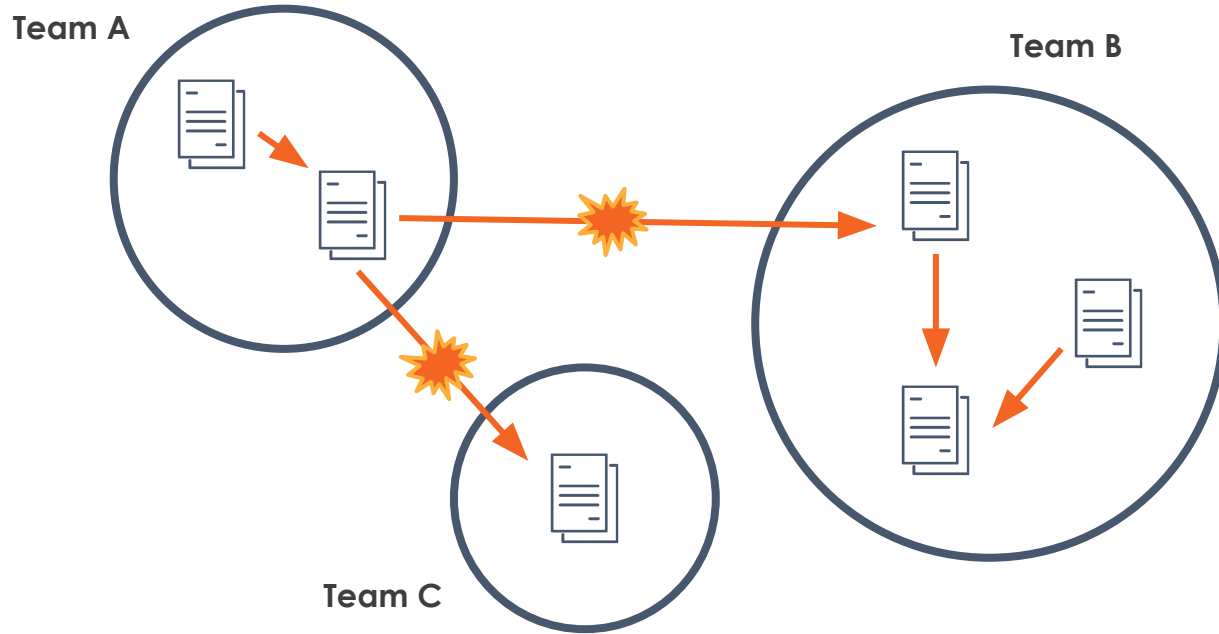
The principal sponsors of  
**OpenLineage**





What is data  
lineage and  
why should we  
all care?

# Building a healthy data ecosystem



# Limited metadata = limited context



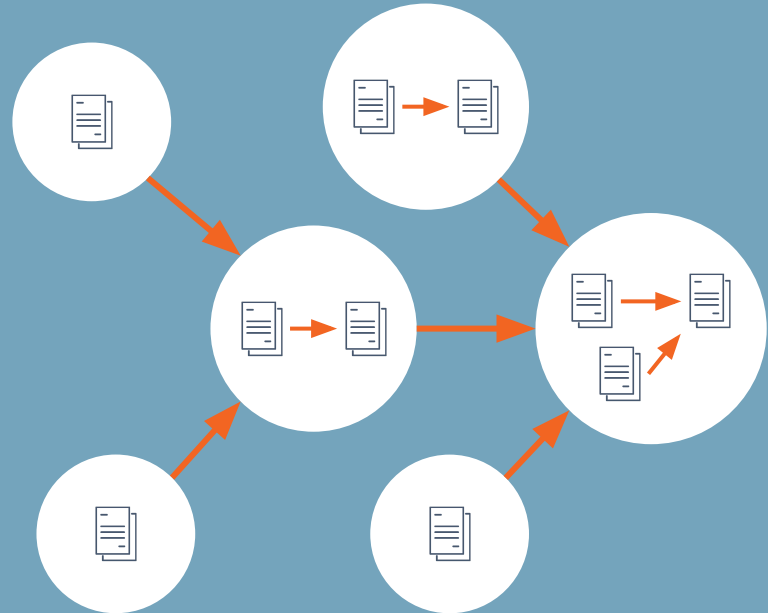
***DATA***


- 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?

# The key = data lineage

Data lineage contains what we need to know to solve our most complicated problems.

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





How can we  
collect and  
study data  
lineage?



# Infer or observe?

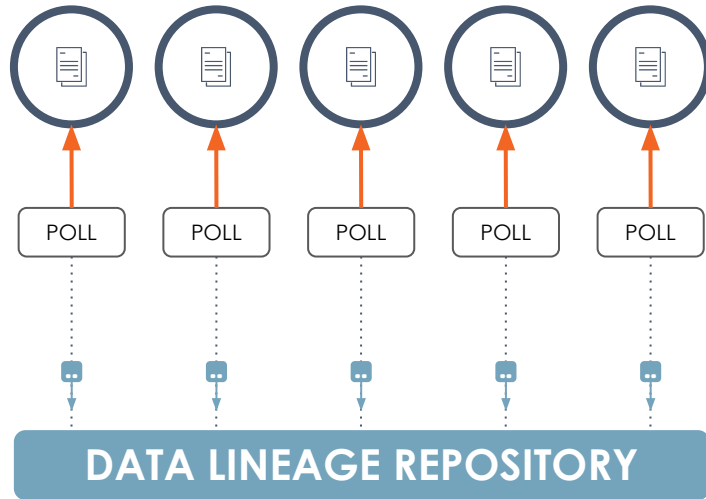


You can try to infer the date and location of an image after the fact...



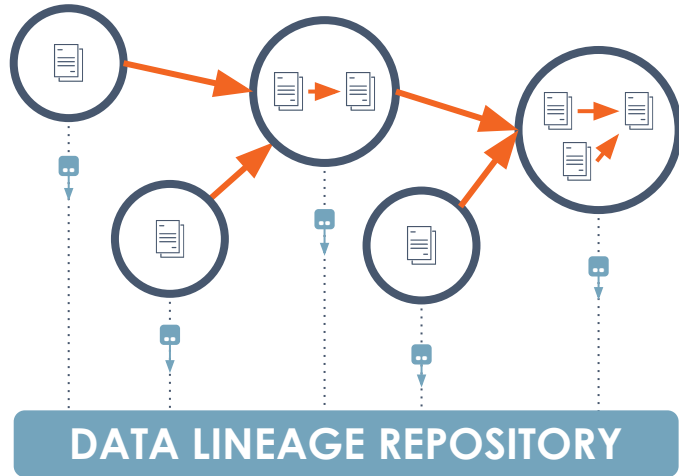
...or you can capture it when the image is originally created!

# Forensic data lineage



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

# Operational data lineage




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

# OMG the possibilities are endless

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





How can we  
study lineage in  
heterogeneous  
pipelines?

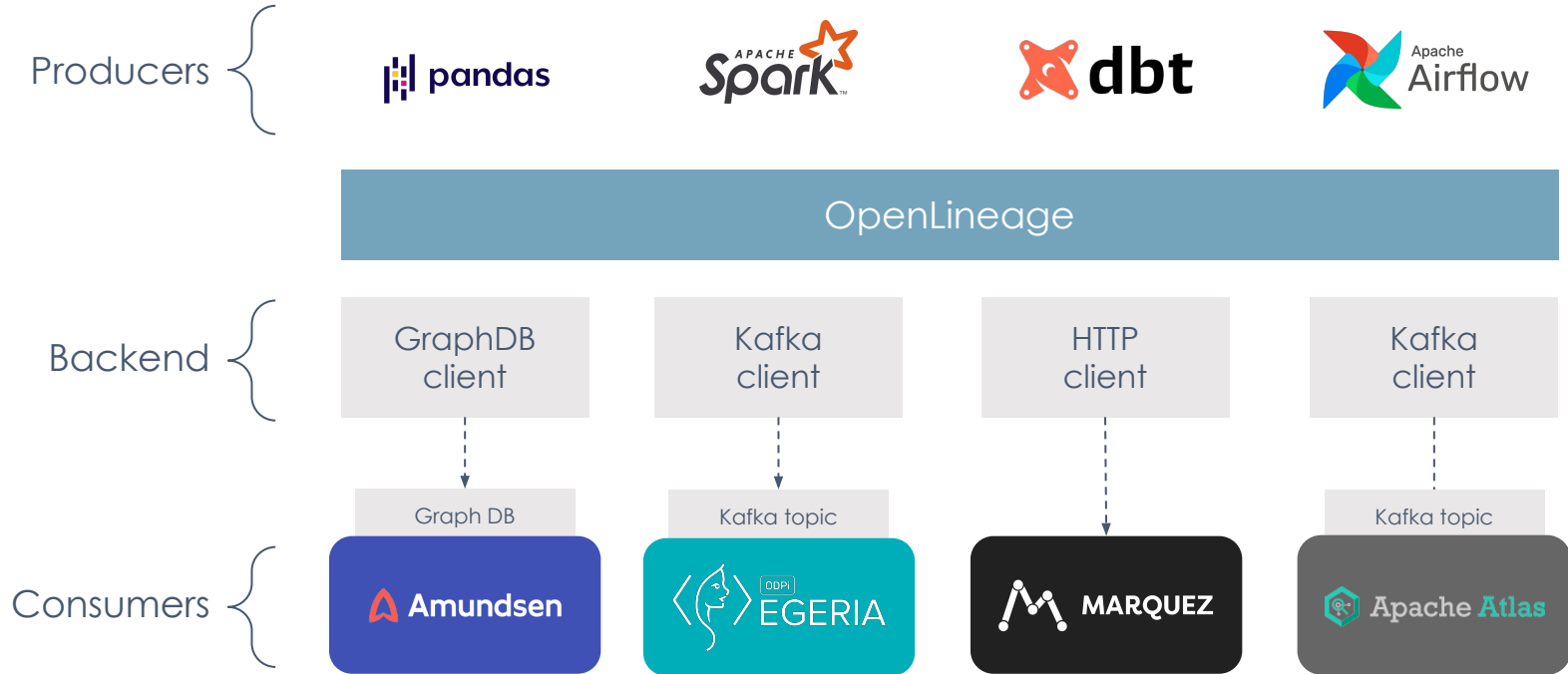
# OpenLineage

## Mission

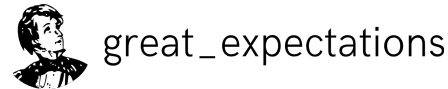
To define an **open standard** for the collection of lineage metadata from pipelines as they are running.



# Where OpenLineage potentially fits




# OpenLineage contributors





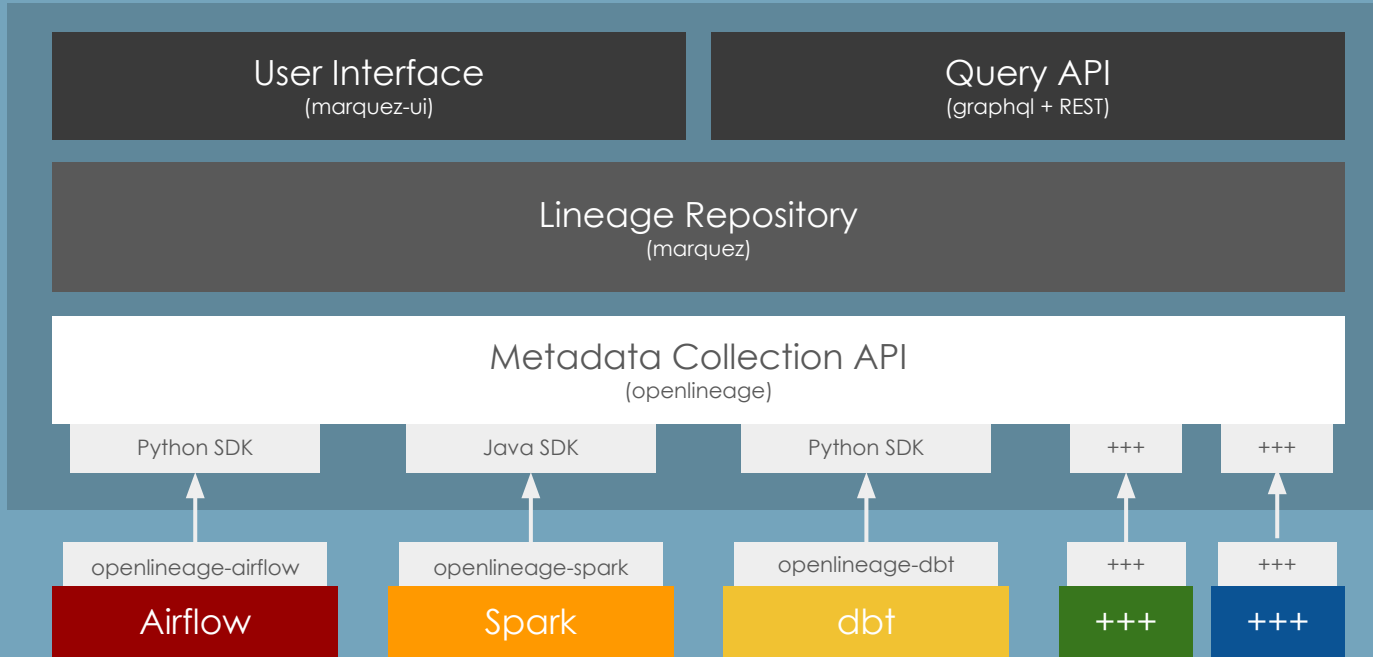
# The snowball effect



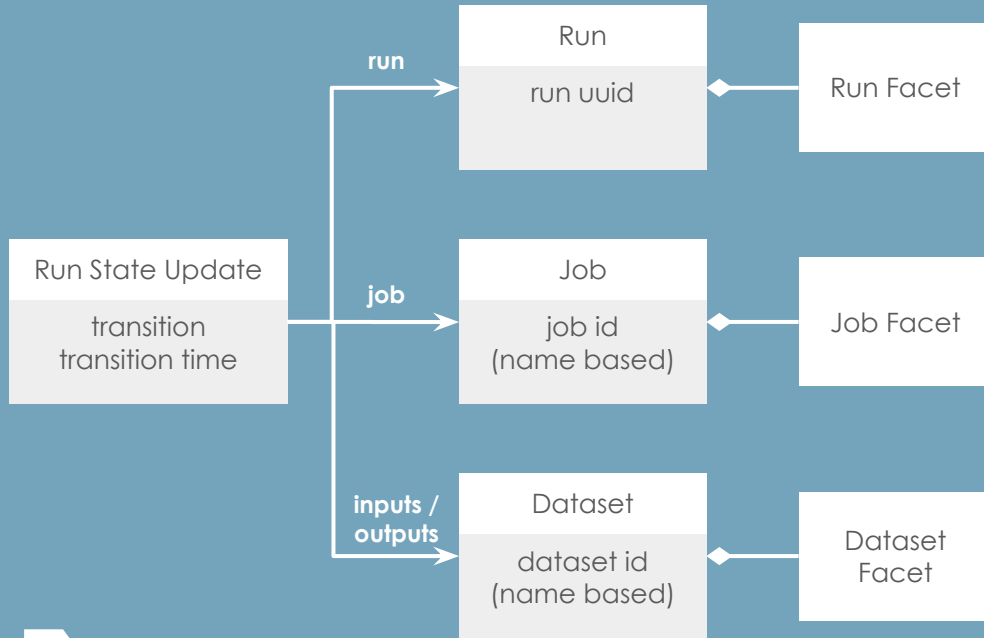


How does  
OpenLineage  
work?

# The OpenLineage Stack



# Data model



Built around core entities:  
Datasets, Jobs, and Runs

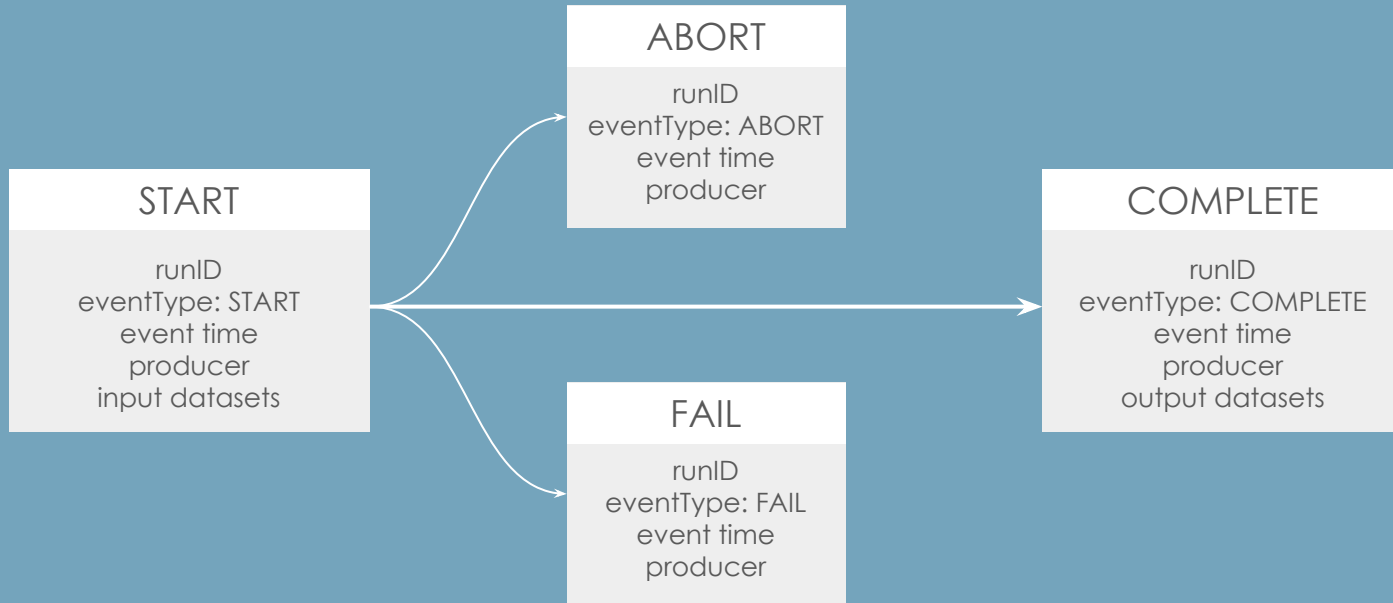
Defined as a JSON  
Schema spec

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

# 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

# Lifecycle of a job run



# 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:

- Quality metrics
- Size metrics
- Schema
- Version

## Job:

- Source code
- Dependencies
- Source control
- Query plan

## Run:

- Scheduled time
- Batch ID
- Query profile
- Params





What does an  
OpenLineage  
event look like?

# Example: starting a job run

```
~/projects/astro ) git main bat -p 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"
}

~/projects/astro ) git main curl -X POST http://nuckles.rtrk.us:5000/api/v1/lineage \
-H 'Content-Type: application/json' \
-d @startjob.json

~/projects/astro ) git main
```

# Example: completing a job run

```
Apple > ~/projects/astro git main ?1 bat -p 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"
}

Apple > ~/projects/astro git main ?1 curl -X POST http://nuckles.rtrk.us:5000/api/v1/lineage \
-H 'Content-Type: application/json' \
-d @completejob.json


Apple > ~/projects/astro git main ?1 ✓
```



Example:  
viewing a  
job run

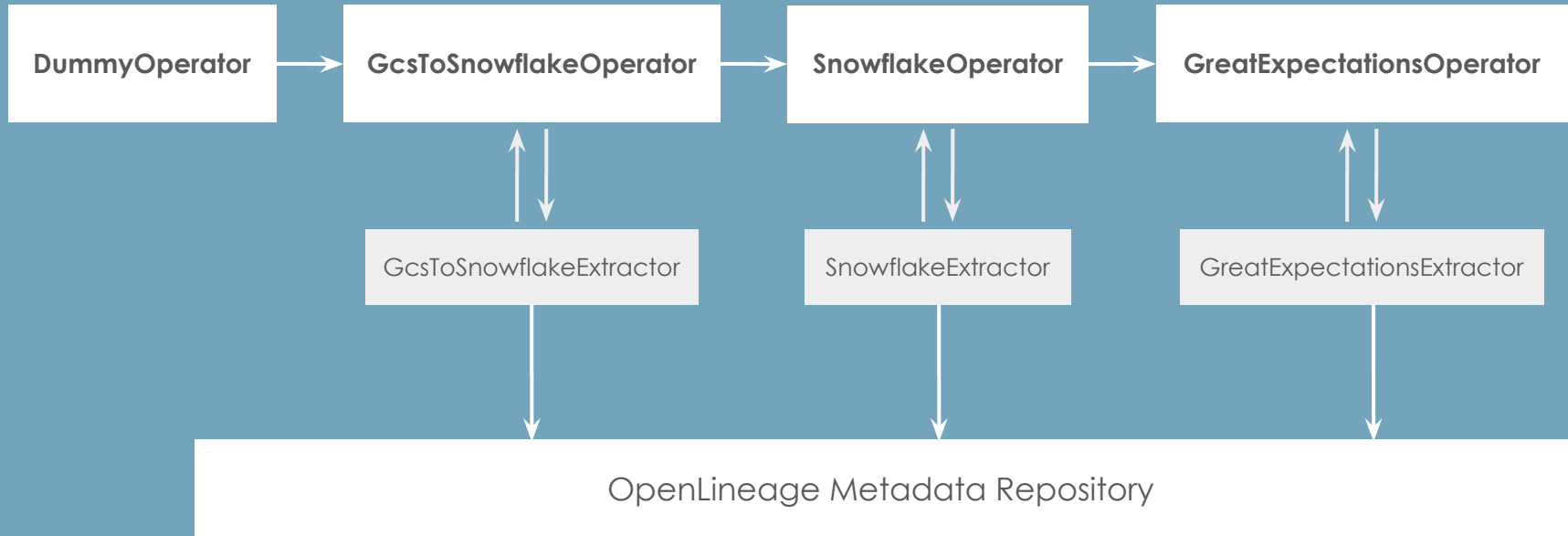
The screenshot shows the Datakin web interface in a browser window. The URL bar displays `dkhq.datakin.com`. The interface includes a sidebar with navigation icons, a top navigation bar with the Datakin logo and a search bar containing `my-output`, and a main content area. The main content area displays a workflow diagram with two components: `Base` (containing `my-input`) and `my-namespace` (containing `my-job` and `my-output`). An orange arrow connects `my-input` to `my-job`. Below the diagram is a tabbed interface with tabs for `Info`, `Inputs/Outputs`, `Quality`, and `Compare`. The `Inputs/Outputs` tab is active, showing a table for `my-namespace / my-output`.

Attribute	Type	Description
a	VARCHAR	
b	VARCHAR	



How does  
Airflow work with  
OpenLineage?

# Lineage is collected with Extractors



# Enabling the integration

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

```
bat -p .env  
OPENLINEAGE_EXTRACTOR_MyOperator=full.path.to.MyExtractor
```

```
~/projects/astro
```



# 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



Code example

Thanks :)

