Cross-Platform Data Lineage with OpenLineage

Julien Le Dem, Chief Architect
Willy Lulciuc, Software Engineer
Astronomer | June 2023

Agenda

The need for lineage metadata

OpenLineage and Marquez

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

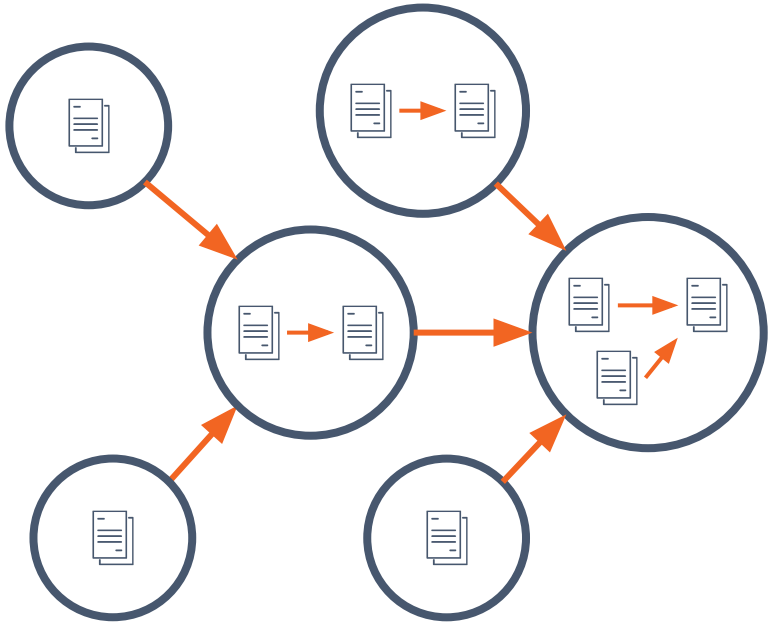
Data observability across your ecosystem

Demo!

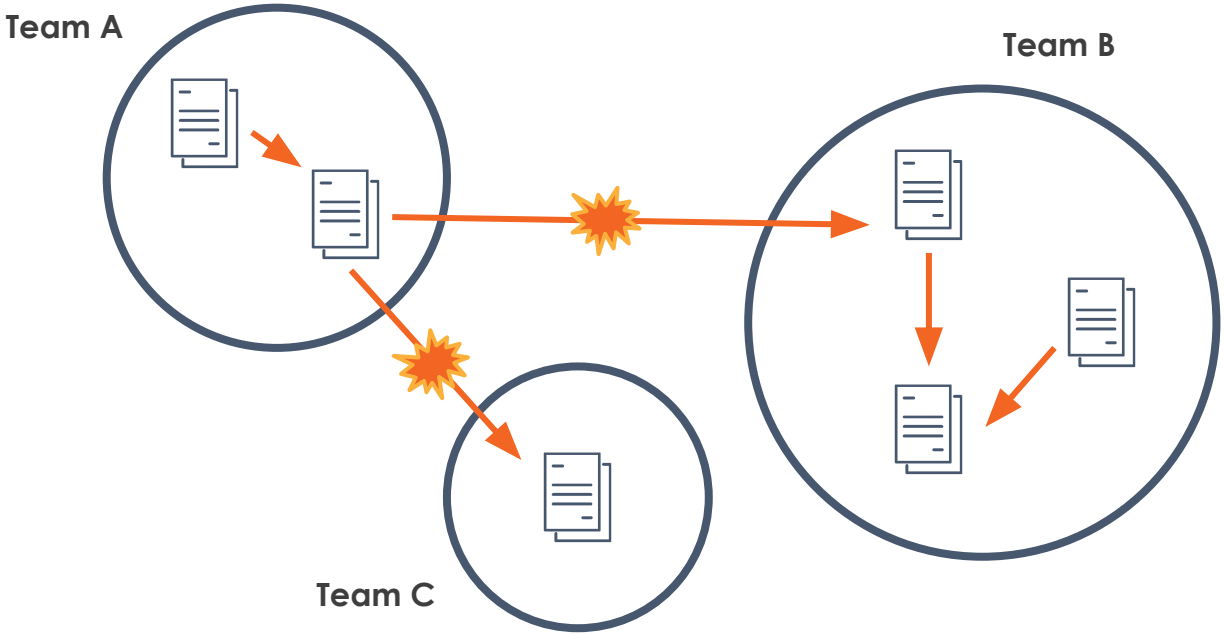
The key = data lineage

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



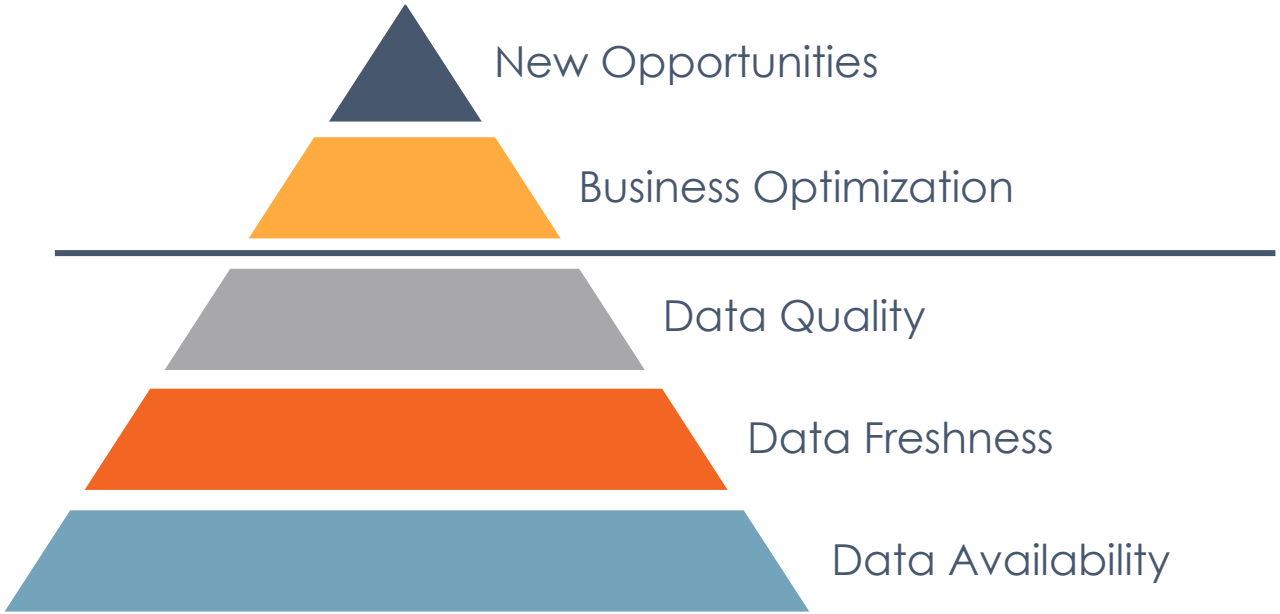
Building a healthy data ecosystem



Limited metadata = limited context

- 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



OpenLineage

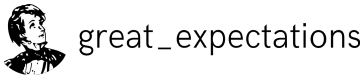
Mission

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

OpenLineage contributors

OpenLineage Integrations

Metadata producers



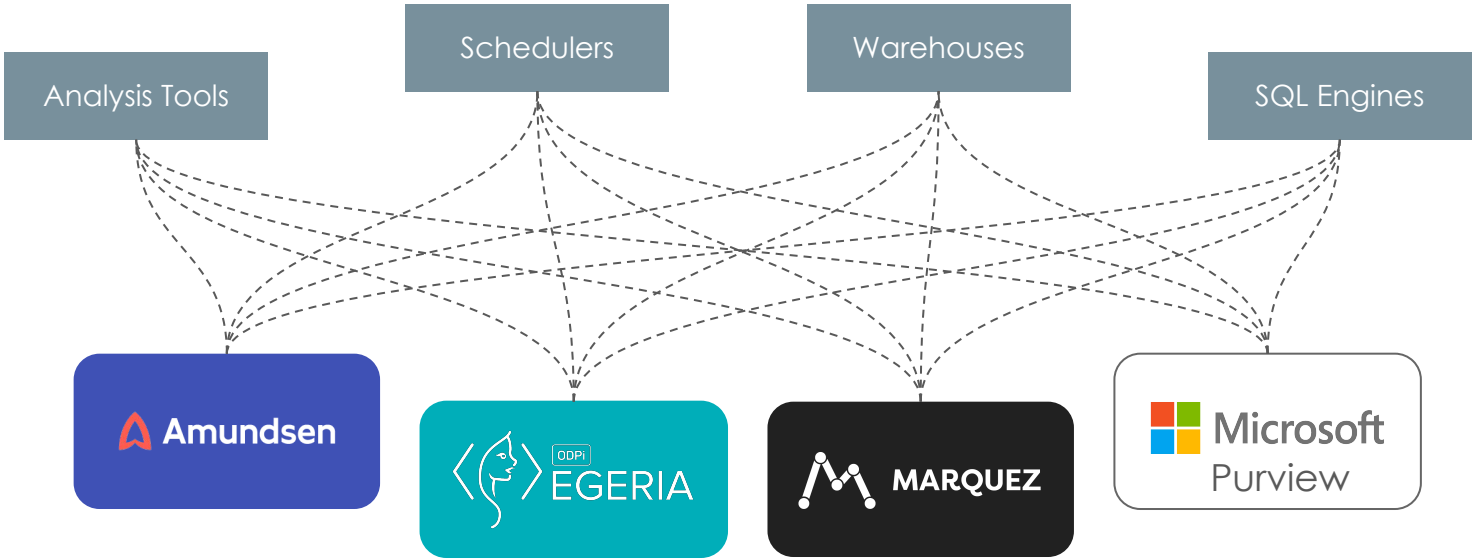
Metadata consumers

The best time to collect metadata

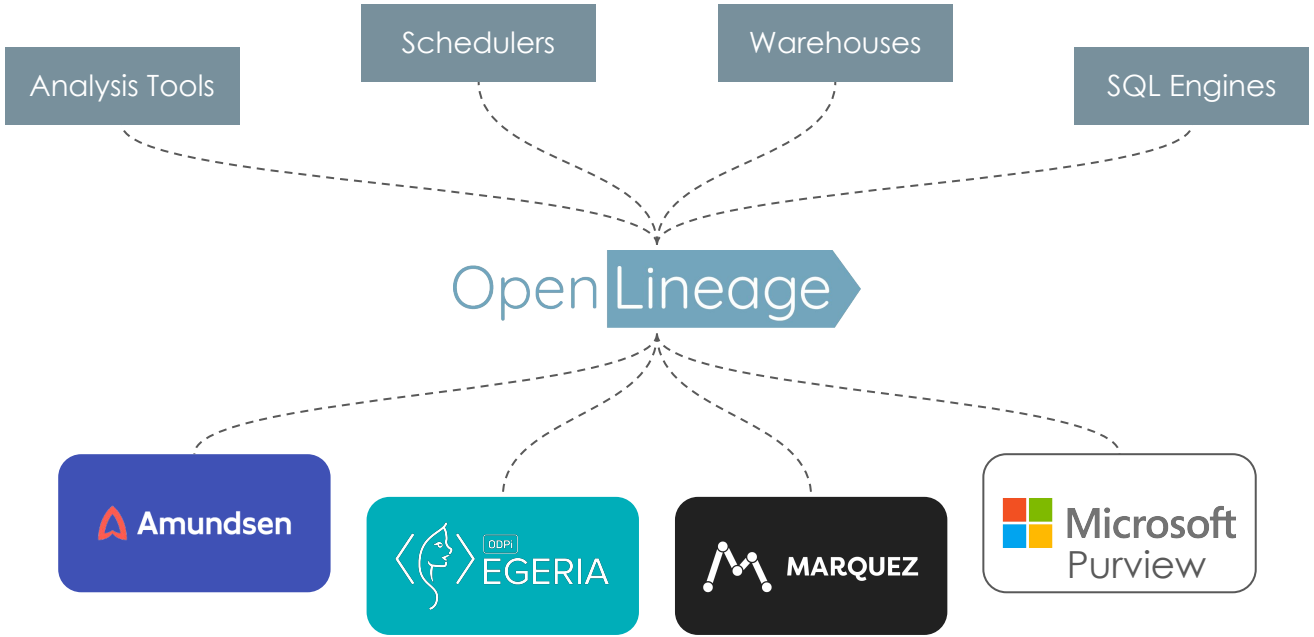
You can try to infer the date and location of an image after the fact…

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

Before OpenLineage



With OpenLineage



The snowball effect

How OpenLineage events work

Lineage is reported as a series of asynchronous run events.

Each event passes a unique client-generated run ID to:

- identify the run
- correlate events

Typical event series:

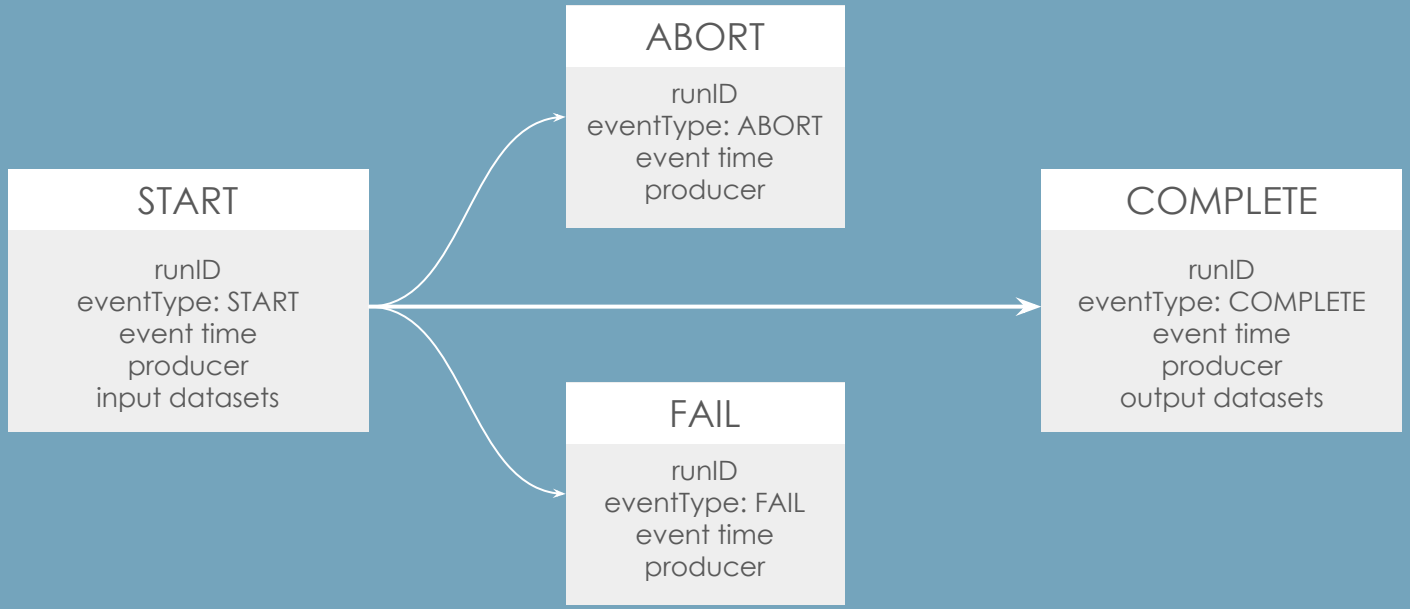
Send start event

- source code version
- run parameters

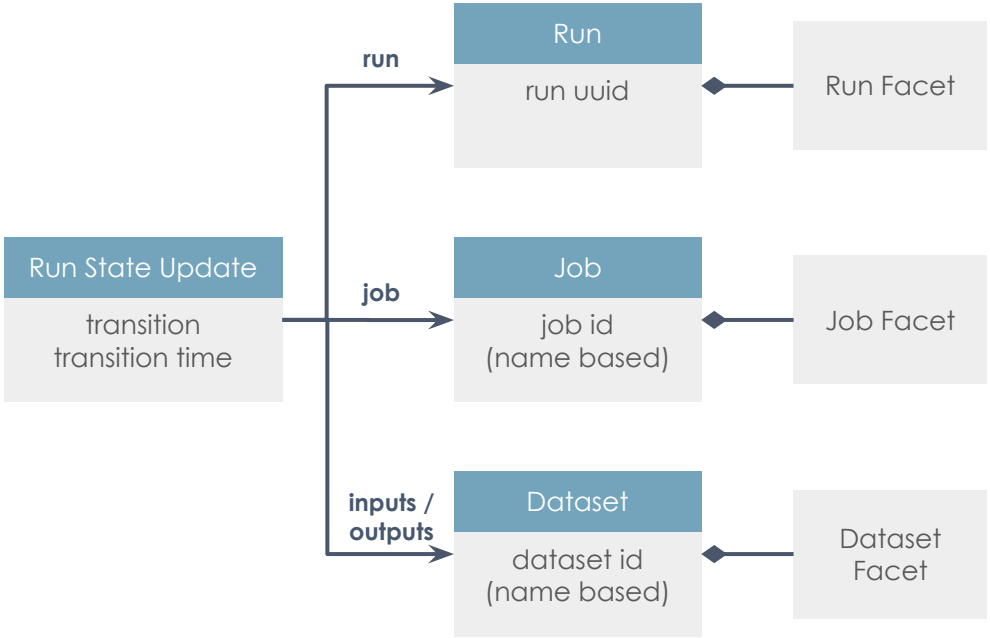
Send complete event

- input dataset
- output dataset version
- output schema

Lifecycle of a job run



Data model


Run
run uuid
run
Run Facet
Run State Update
transition
transition time
Job
job id
(name based)
job
Job Facet
Dataset
dataset id
(name based)
inputs /
outputs
Dataset
Facet

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

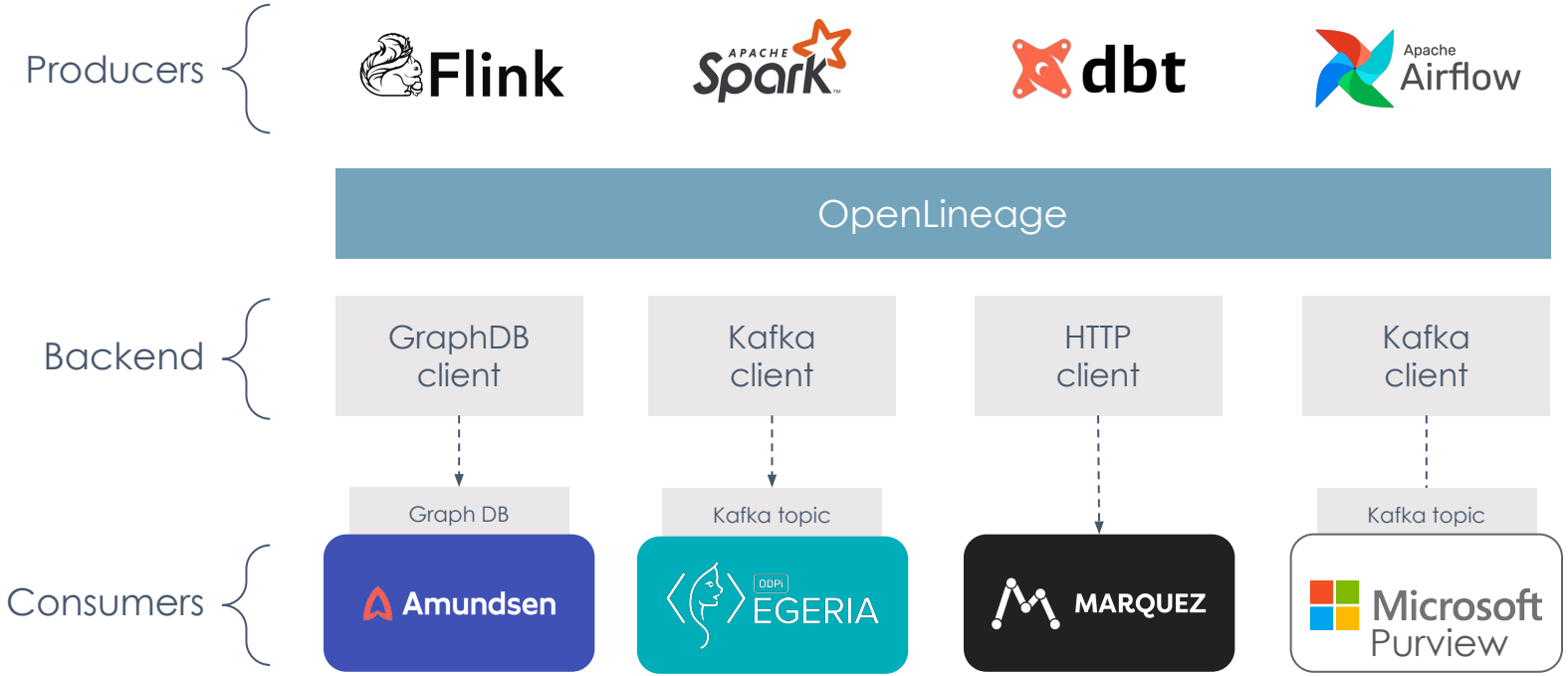
Job:

- Source code
- Dependencies
- Source control
- Query plan

Run:

- Scheduled time
- Batch ID
- Query profile
- Params

Where OpenLineage potentially fits

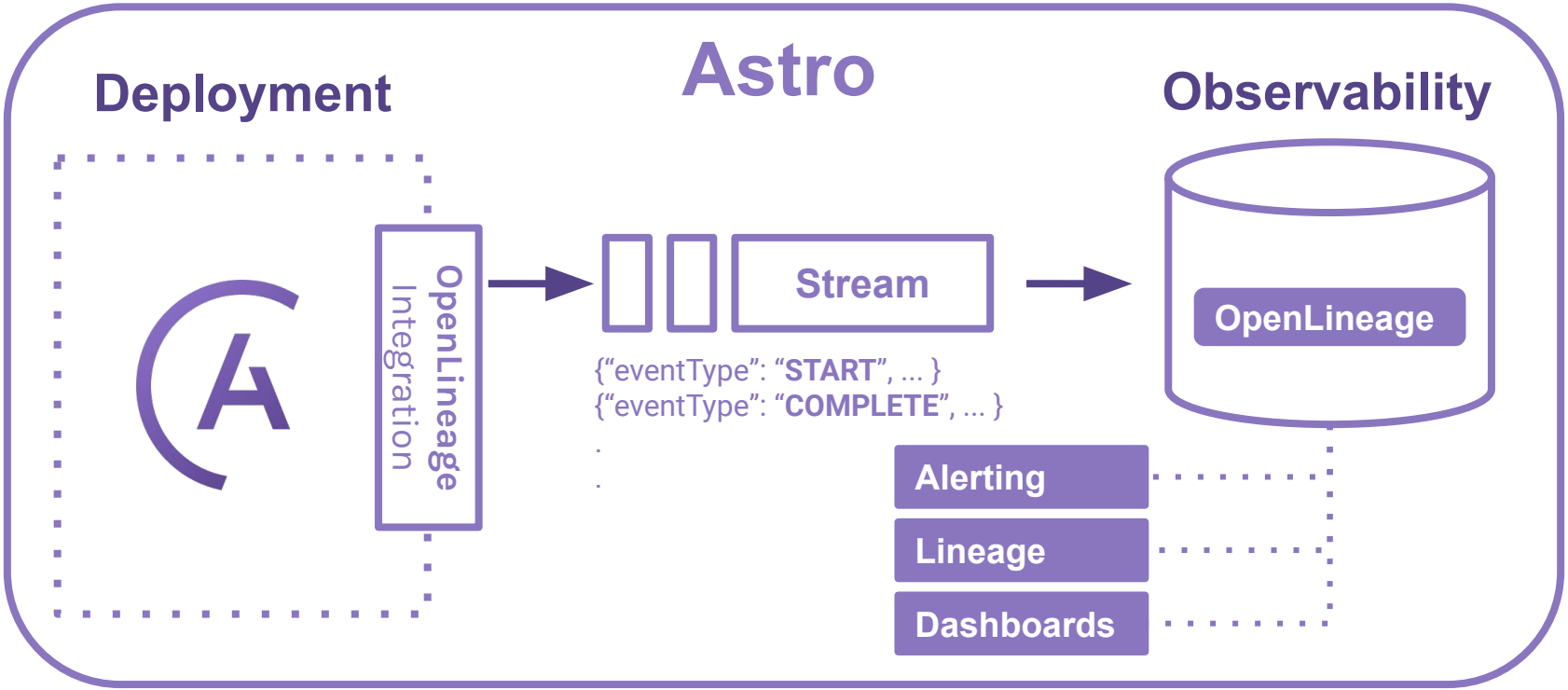


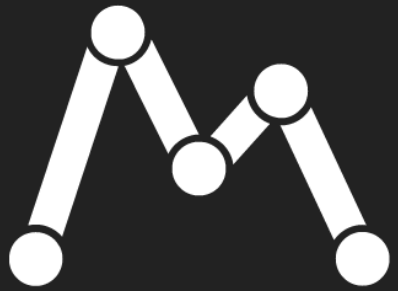
Data Observability with **OpenLineage** in Practice

Powered by **OpenLineage**

Data Reliability	**Data Recovery**	**Data Audit**
• Data freshness • Data quality • Impact Analysis	• Reduce mean time to resolution • Backfills	• Data cost • Data compliance

Astronomer Data Observability Powered by **OpenLineage**

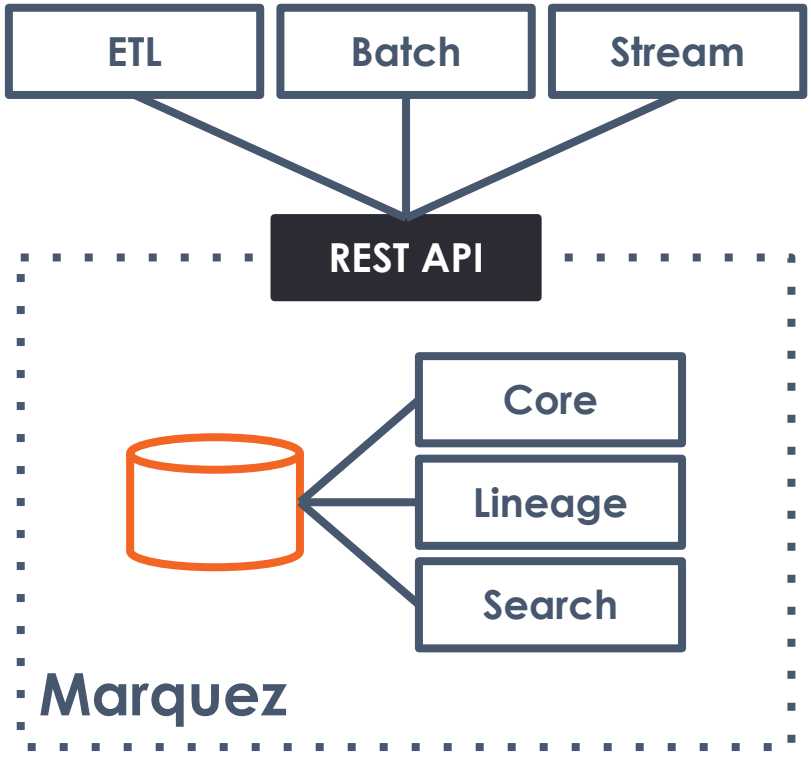




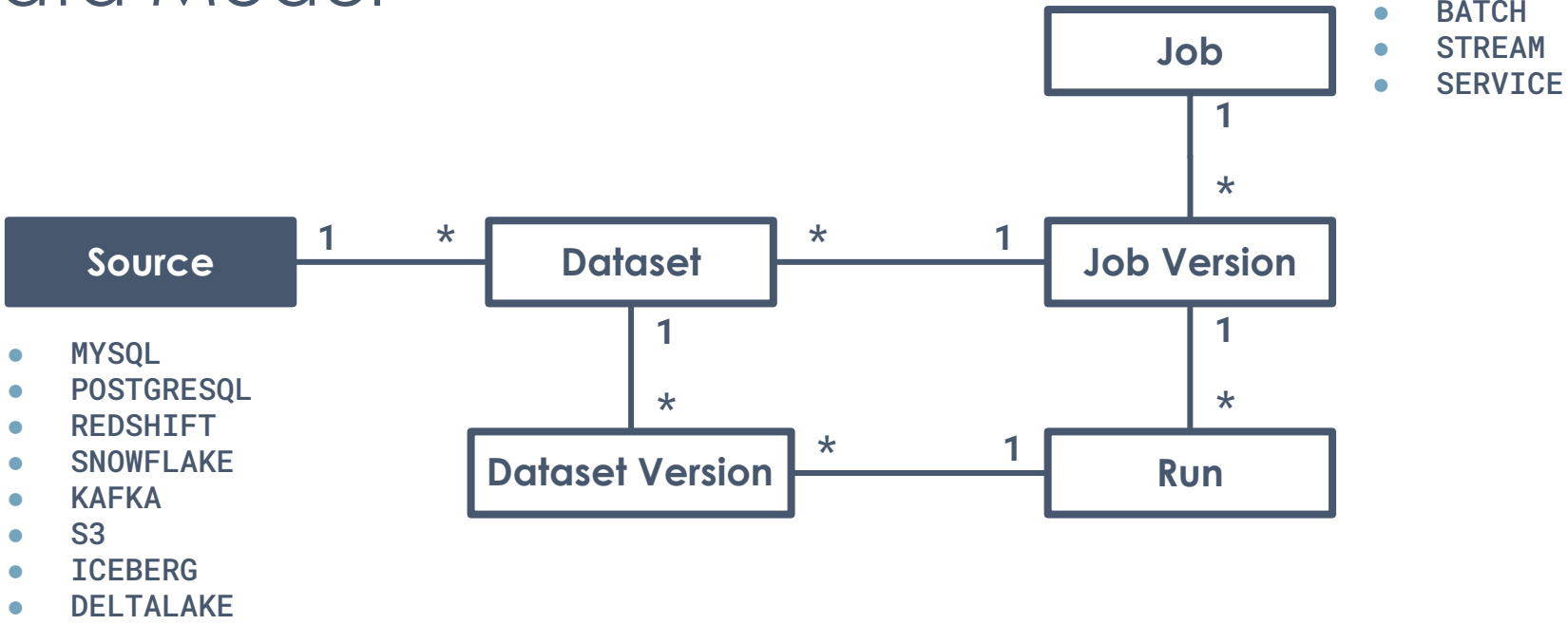
MARQUEZ

Metadata Service for **Operational Lineage**

- **Open source**, LFAI + Data incubating stage
- **Reference** implementation of OpenLineage
- **Centralized** metadata management
- **Features**
  - Simple operation and design
  - Easily collect OpenLineage events
  - Static **+** run-level lineage
  - Metadata versioning

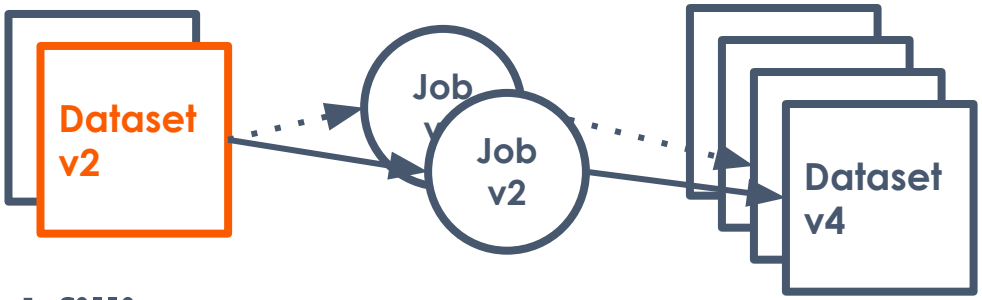


Precise and **Highly Dimensional** Data Model



Design Benefits

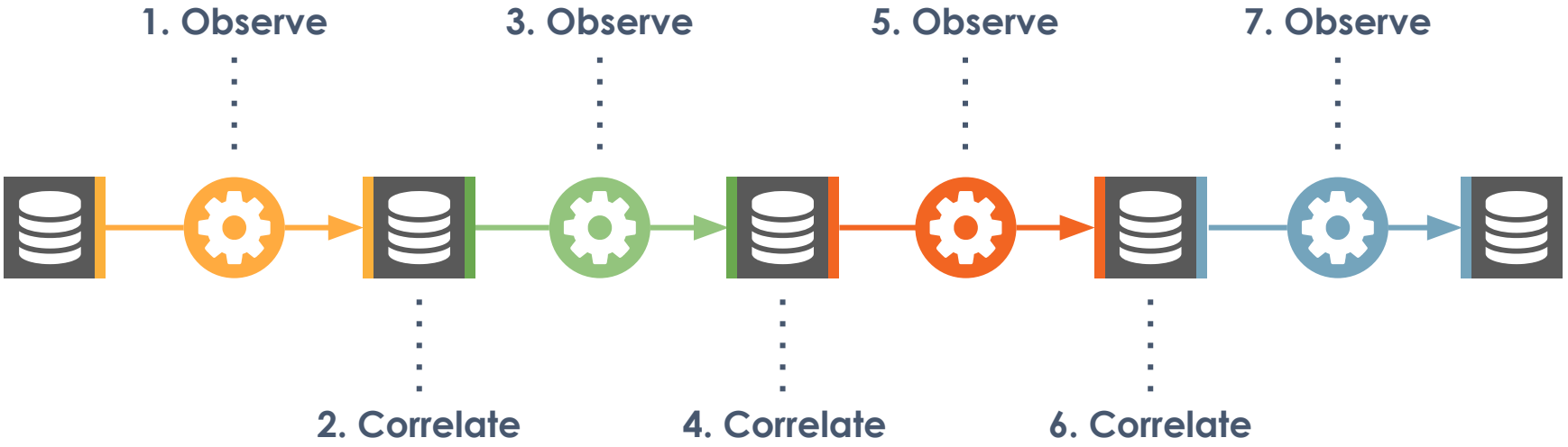
- **Debugging**
  - What **job version(s)** produced and consumed **dataset version X**?



- **Backfilling**
  - Full / incremental processing

Data Observability **Across** Your **Ecosystem**!

Enforcement of Job and Dataset Ownership



Dataset names link **job runs**, and this relationship is used to *materialize* the **lineage graph**.

START JobRun

```
$ curl -X POST http://localhost:5000/api/v1/lineage \
  -H 'Content-Type: application/json' \
  -d '{
        "eventType": "START",
        "eventTime": "2023-06-28T19:49:24.201361Z",
        "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/...",
        "schemaURL": "https://openlineage.io/spec/2-0-0/..."
      }'
```

Observe

Correlate

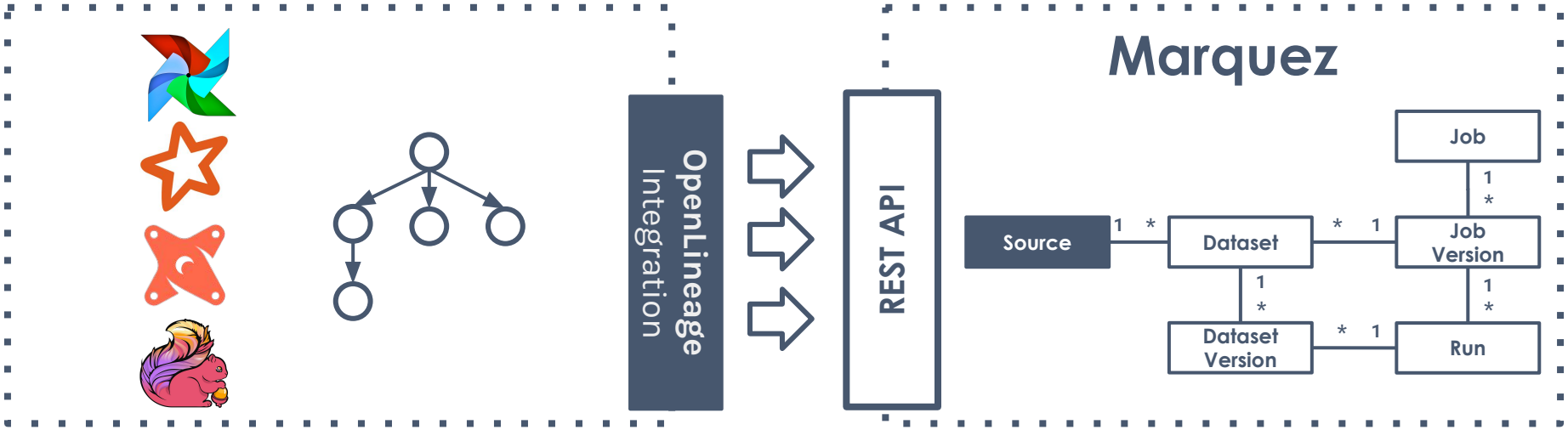
COMPLETE JobRun

```
$ curl -X POST http://localhost:5000/api/v1/lineage \
  -H 'Content-Type: application/json' \
  -d '{
        "eventType": "COMPLETE",
        "eventTime": "2023-06-28T20:15:24.201361Z",
        "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/...",
        "schemaURL": "https://openlineage.io/spec/2-0-0/..."
      }'
```

Observe

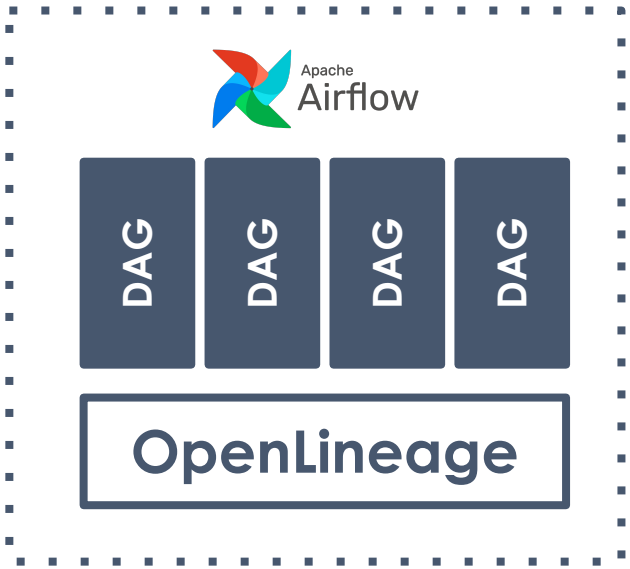
Correlate

Collecting Lineage Metadata in a Nutshell



In Airflow

Airflow Support for OpenLineage



- **Metadata**
  - Task lifecycle
  - Task parameters
  - Task runs linked to **versioned** code
  - Task inputs / outputs
- **Built-in**
  - SQL parser
  - Link to code builder (**GitHub**)
  - Metadata extractors

OpenLineage in Airflow

$AIRFLOW_HOME/plugins/openlineage/adapter.py

adapter.py

```
OpenLineageAdapter.start_task()    # OL.START
OpenLineageAdapter.complete_task() # OL.COMPLETE
OpenLineageAdapter.fail_task()     # OL.FAIL
```

OpenLineage **Config.**

```
$ pip install openlineage-airflow
$ export OPENLINEAGE_URL=http://<host>:<port>
```

OR

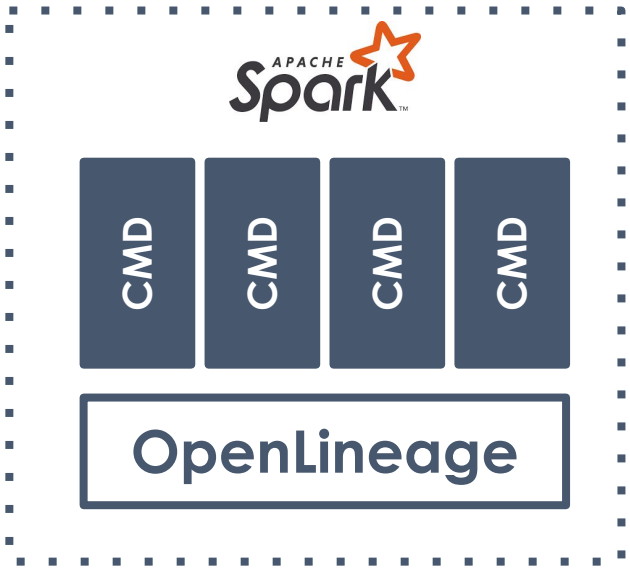
airflow.cfg

```
[openlineage]

disabled   = # OL disabled/enabled
extractors = # OL custom extractos
transport  = # OL backend
```

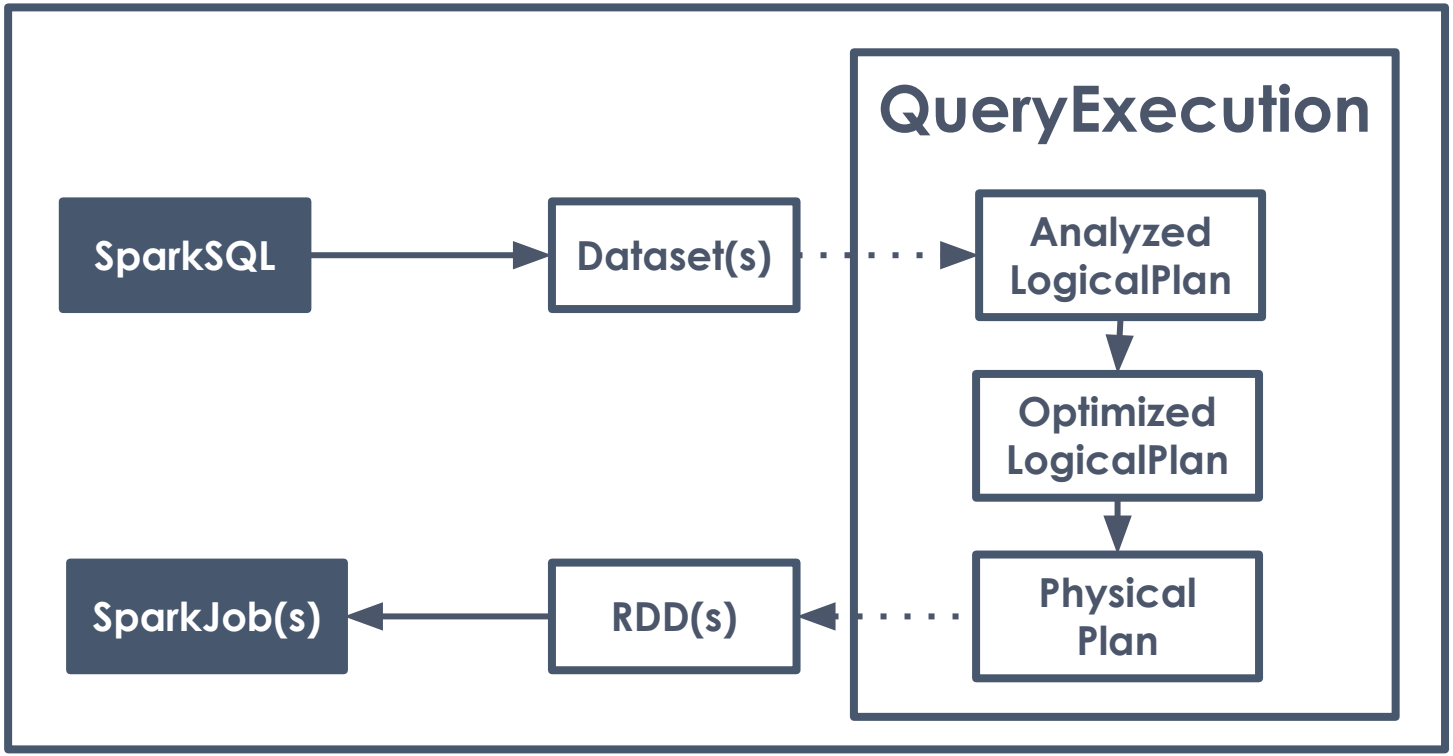
In Spark

Spark Support for OpenLineage



- **Uses** QueryExecutionListener
  - Command lifecycle
  - Analyze **`LogicalPlan`** with **`QueryPlanVisitor`** (inputs/outputs)
- **Built-in**
  - Column-level lineage
  - Data quality metrics

Spark QueryExecution



OpenLineage in Spark

OpenLineageSparkListener.java

```
public class OpenLineageSparkListener
  extends org.apache.spark.scheduler.SparkListener {
 ...
}
```

Metadata

- Serialized LogicalPlan
- Environment properties
- Datasets
  - Schema
  - Source
  - Metrics (**Bytes**/**Row counts**)

OpenLineage **Config.**

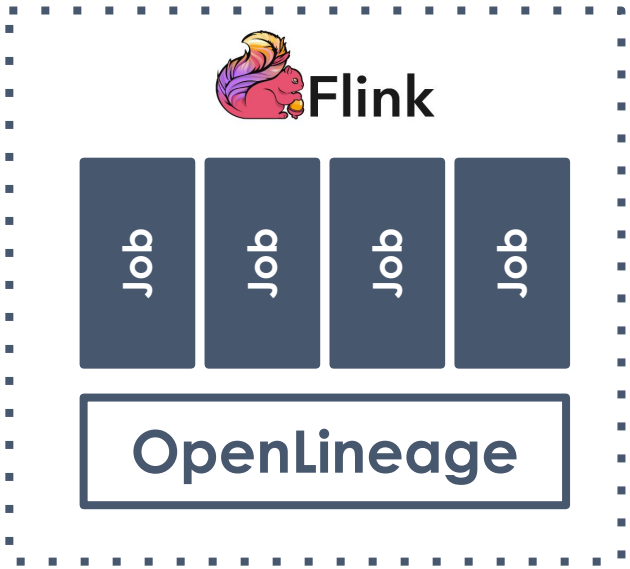
Shell

```
$ pyspark  \
    --conf spark.jars.packages=io.openlineage:openlineage-spark:0.28.+ \
    --conf spark.extraListeners=io.openlineage.spark.agent.OpenLineageSparkListener \
    --conf spark.openlineage.transport.type=http  \
    --conf spark.openlineage.transport.url=http://<host>:<port>

io.openlineage#openlineage-spark added as a dependency
:: resolving dependencies
  found io.openlineage#openlineage-spark;0.28.0 in central
...
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.4.1
      /_/
>>>
```

In Flink

Flink Support for OpenLineage



- **Uses** JobListener
  - Job status changes
  - Job checkpoint metrics
  - **Application Mode** support only
- **Metadata**
  - Job
  - Sources + Sinks as datasets (inputs/outputs)

OpenLineage in Flink

OpenLineageFlinkJobListener.java

```
public class OpenLineageFlinkJobListener
  implements org.apache.flink.core.execution.JobListener {
 onJobSubmitted() # OL.START
 onJobExecuted()  # OL.COMPLETE
}
```

OpenLineage **Config.**

MyFlinkJob.java

```
StreamExecutionEnvironment env = setUp()
JobListener myFlinkJobListener =
  new OpenLineageFlinkJobListener(env)
env.registerJobListener(myFlinkJobListener);
```

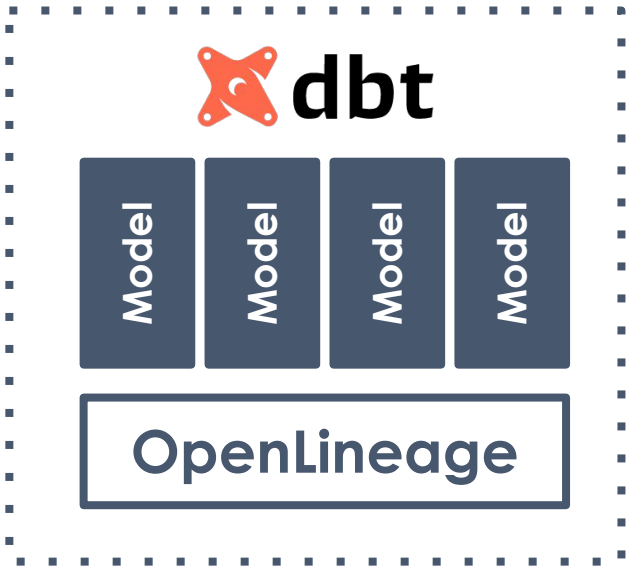
AND

openlineage.yml

```
transport:
  type = http
  url  = http://<host>:<port>
```

In dbt

dbt Support for OpenLineage



- **Uses** `dbt-ol`
  - Wraps `dbt` commands
  - Emits OpenLineage events on run completion
- **Metadata**
  - Sources
  - Models as datasets (dependencies)
    - Schema

OpenLineage **Config.**

```
$ pip install openlineage-dbt
$ export OPENLINEAGE_URL=http://<host>:<port>
```

THEN

Shell

```
$ dbt-ol run

Completed successfully

Done. PASS=2 WARN=0 ERROR=0 SKIP=0 TOTAL=2
Emitted 4 openlineage events
```

Demo!

Coming soon

- Native integration in Airflow (AIP-53)
- Implementation of static lineage
- Flink integration improvements in collaboration with Flink maintainers

  support for streaming datasets!
- Improvements to Databricks specific Spark support.

Join the conversation

github.com/openlineage

bit.ly/OpenLineageSlack

@openlineage

github.com/marquezproject

bit.ly/MarquezSlack

@marquezproject

Thanks :)