Data lineage and observability with OpenLineage

datakin

Julien Le Dem, CTO and Co-Founder Datakin | Mai 2021

AGENDA

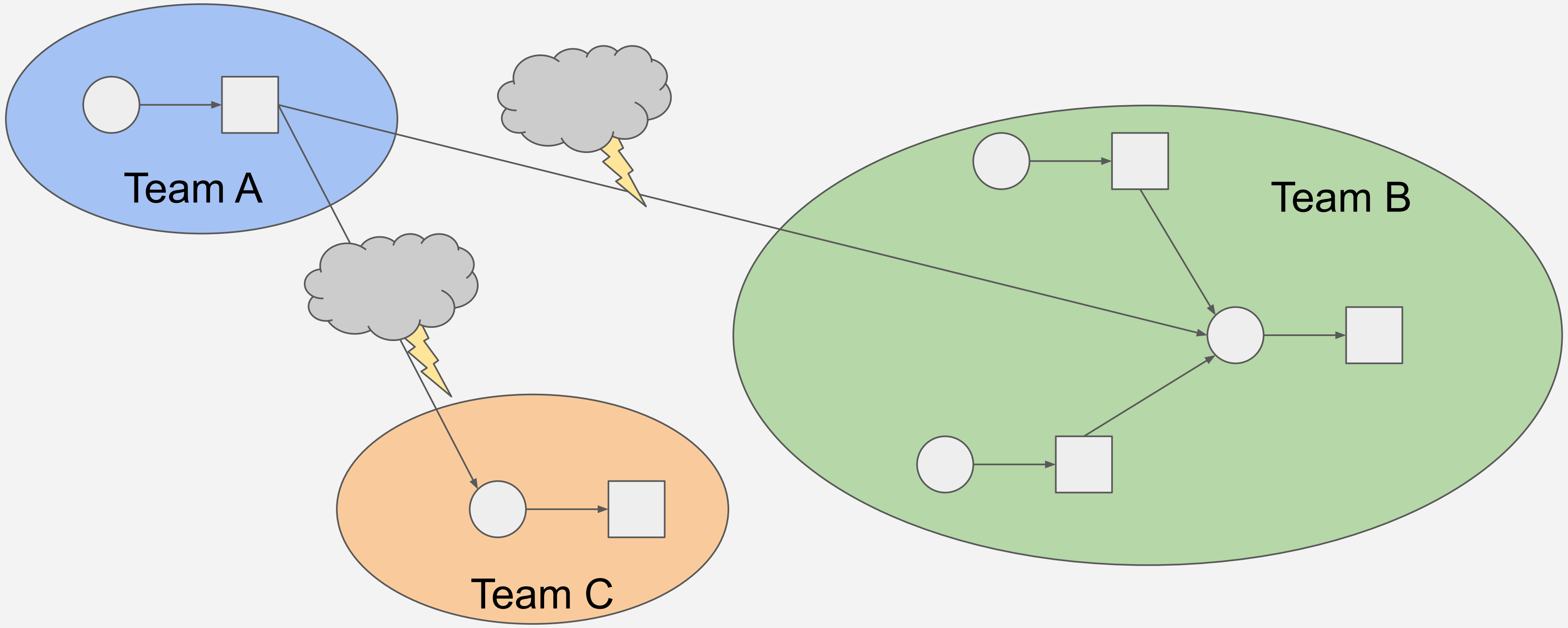
- The need for metadata
- **OpenLineage** - the open standard for lineage collection - and **Marquez**, its reference implementation
- Spark observability with OpenLineage

datakin

The need for Metadata

datakin

Building a healthy data ecosystem

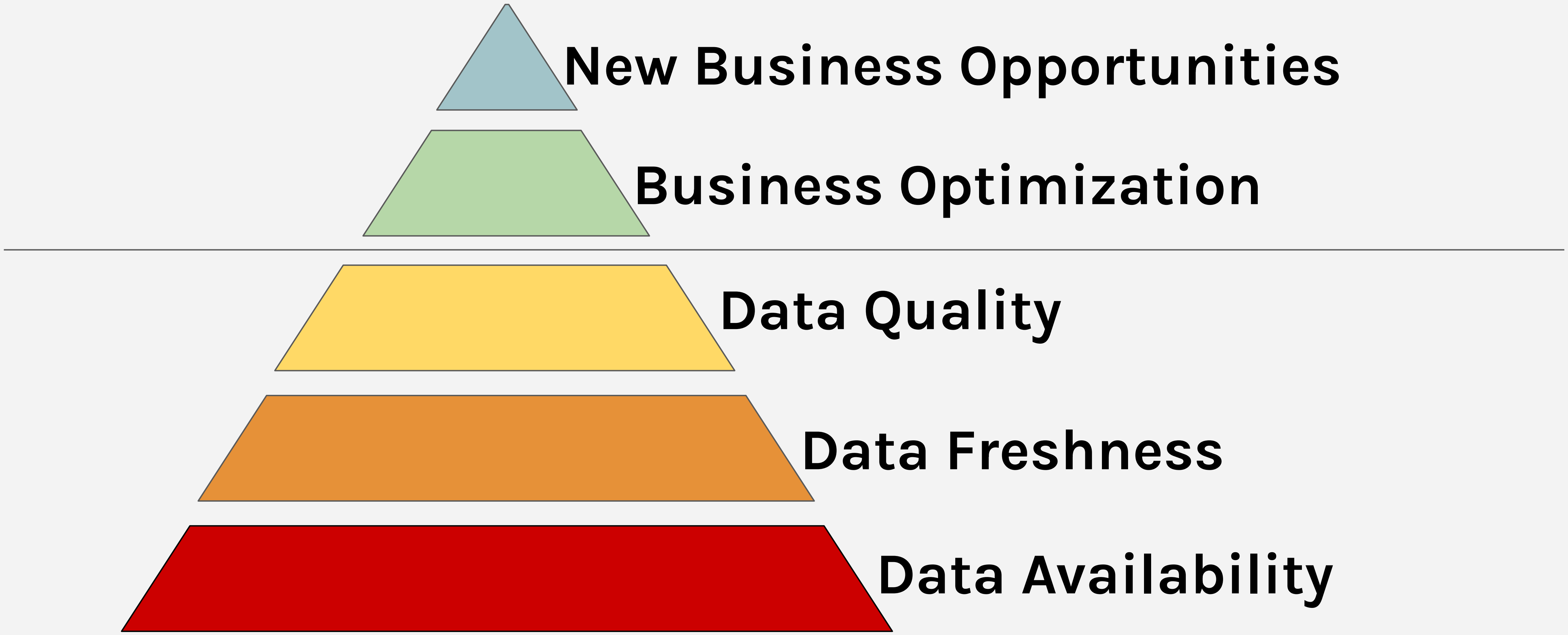


Today: Limited context

DATA

- What is the data source?
- What is the schema?
- Who is the owner?
- How often is it updated?
- Where is it coming from?
- Who is using the data?
- What has changed?

~~Maslow's~~ Data hierarchy of needs



OpenLineage

datakin

OpenLineage contributors

Creators and contributors from major open source projects involved

pandas

MARQUEZ

Data Hub

APACHE Spark

dbt

Amundsen

Parquet

Apache Airflow

ODPi EGERIA

ICEBERG

PREFECT

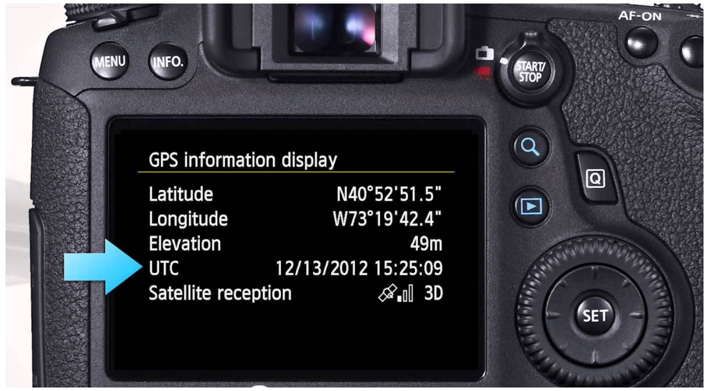
great_expectations

datakin

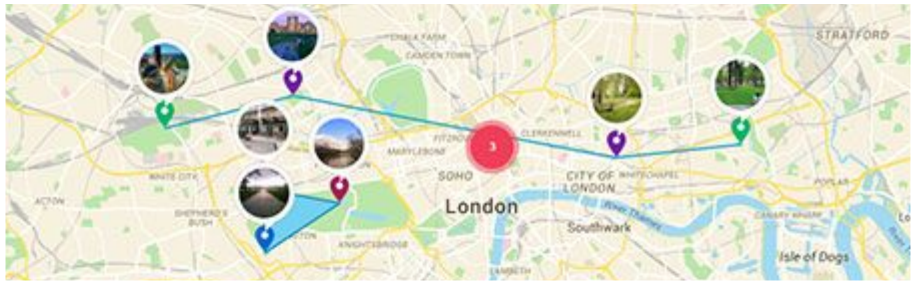
Purpose:

Define an Open standard for metadata and lineage collection by instrumenting data pipelines as they are running.

Purpose:
EXIF for data pipelines

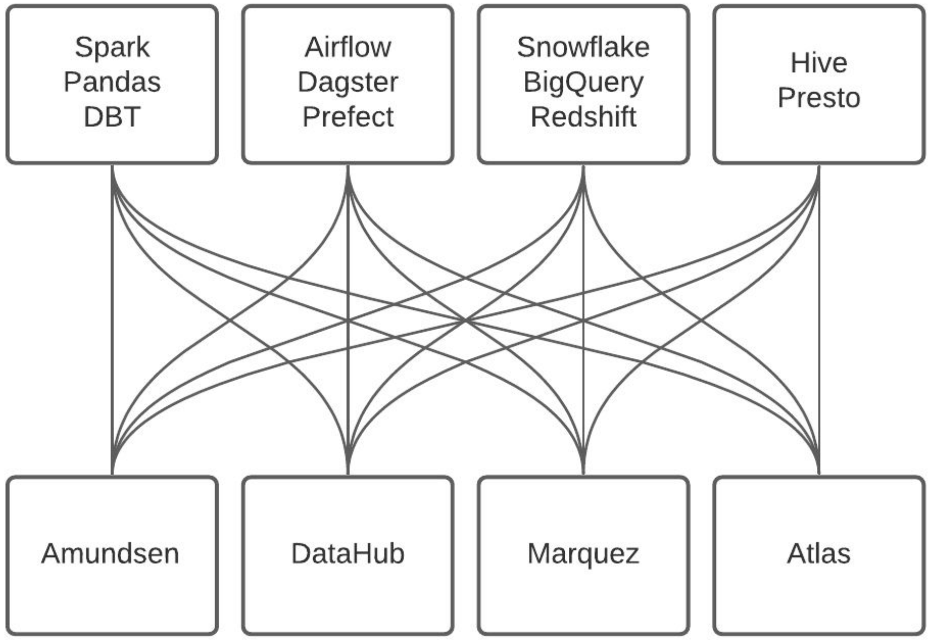

GPS information display

Latitude	N40°52'51.5"
Longitude	W73°19'42.4"
Elevation	49m
UTC	12/13/2012 15:25:09
Satellite reception	3D

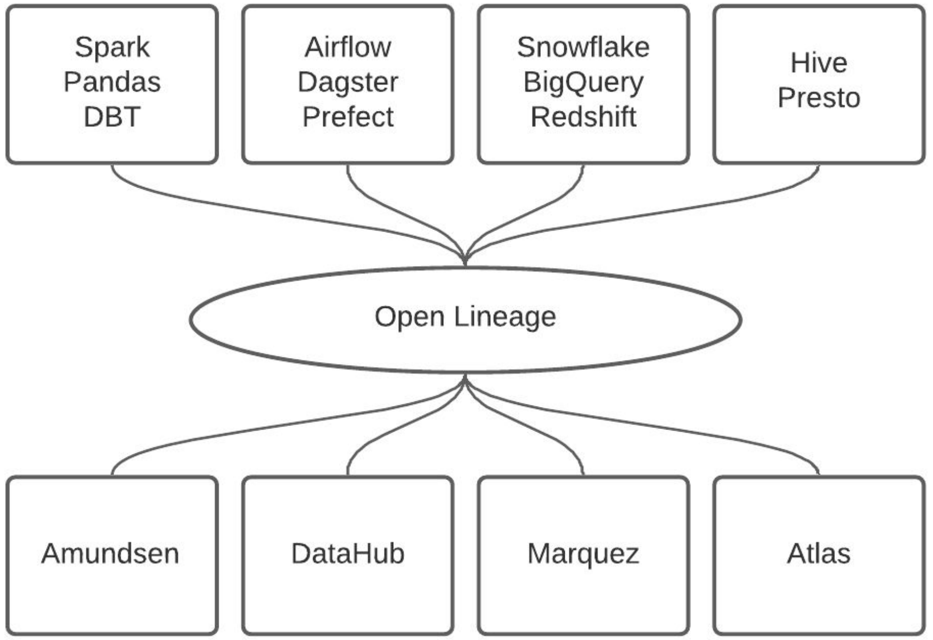


Problem

Before:



With Open Lineage



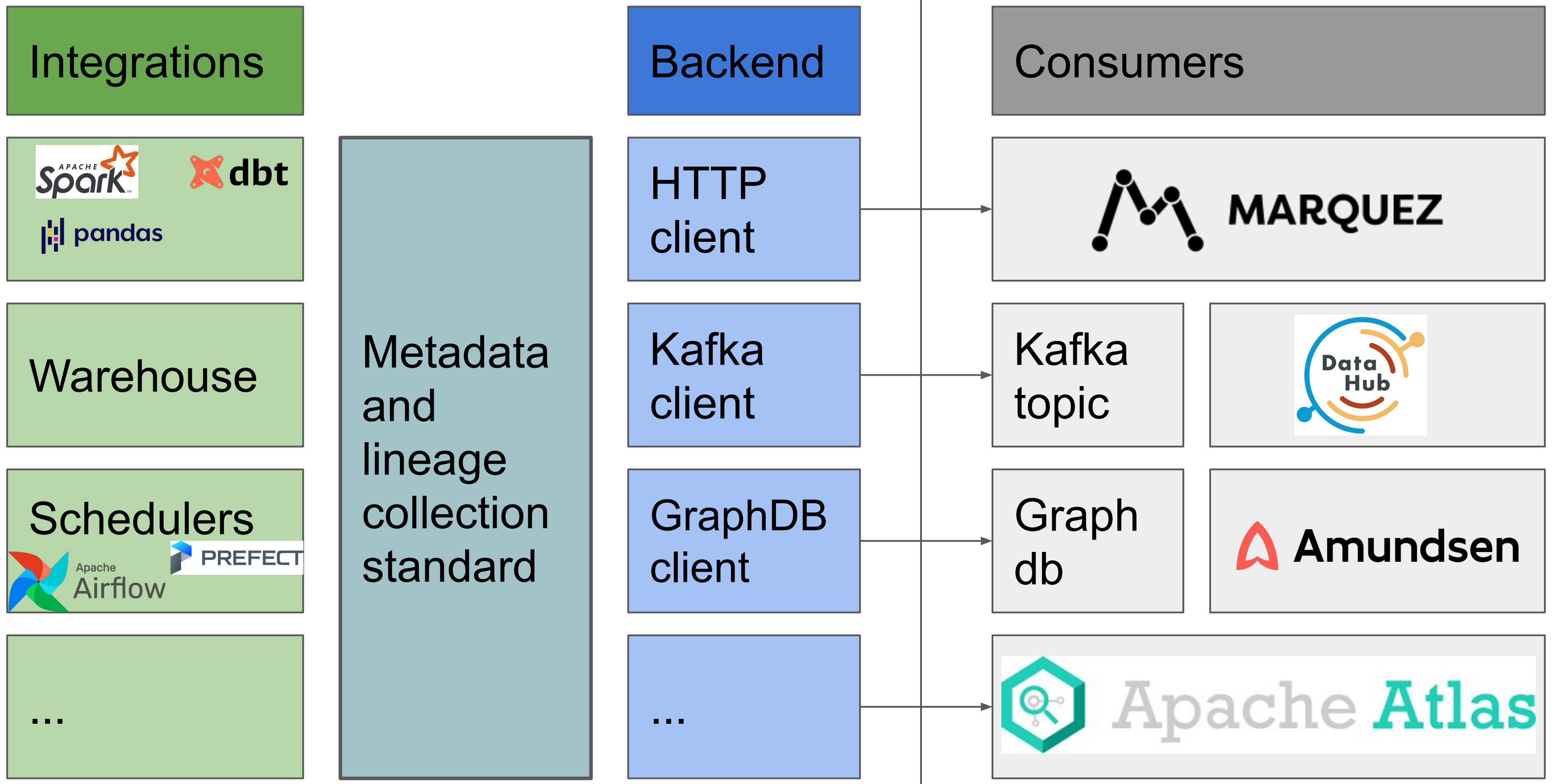
- Duplication of effort: Each project has to instrument all jobs
- Integrations are external and can break with new versions

- Effort of integration is shared
- Integration can be pushed in each project: no need to play catch up

datakin

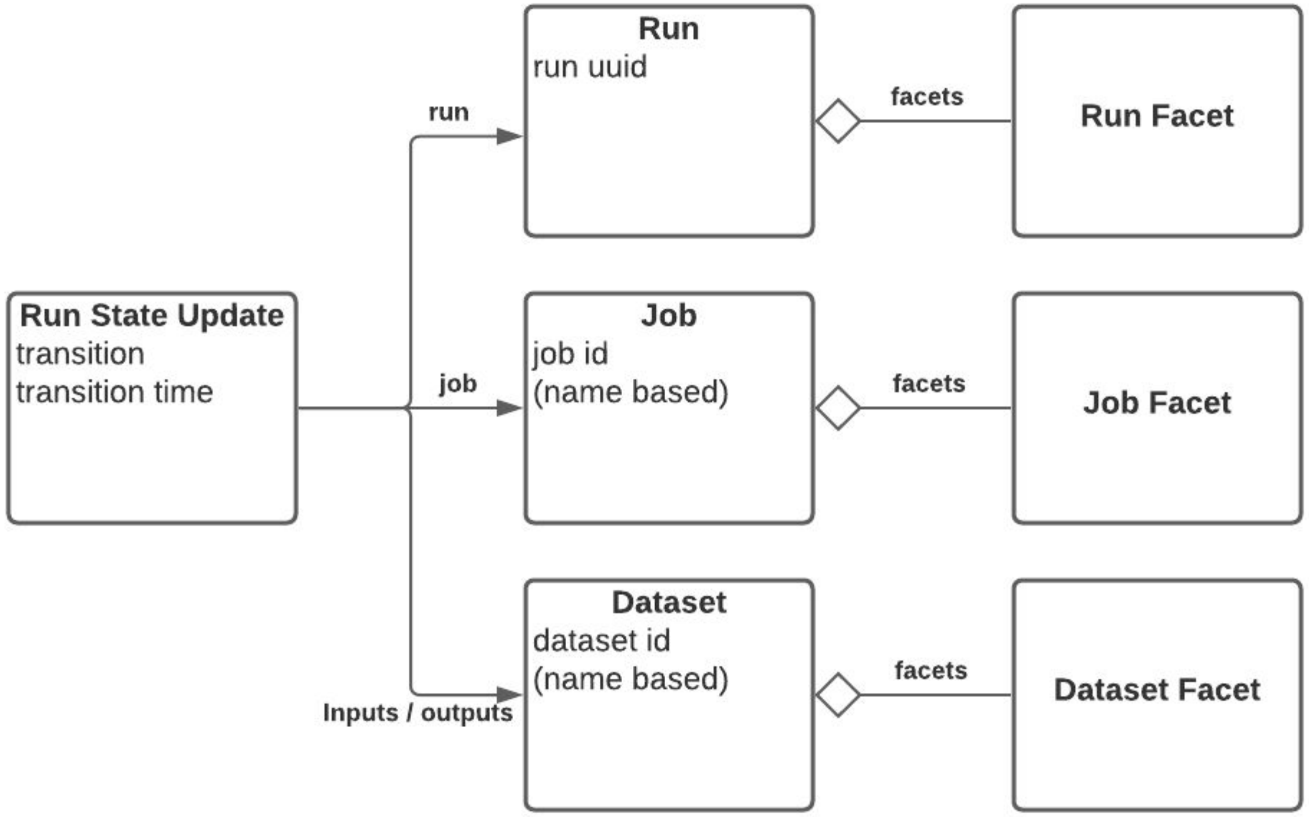
Open Lineage scope

Not in scope



Core Model:

- JSONSchema spec
- Consistent naming:
  Jobs:
    scheduler.job.task
  Datasets:
    instance.schema.table



Protocol:

- Asynchronous events:

  Unique run id for identifying a run and correlate events

- Configurable backend:
  - Kafka
  - Http

Examples:

- Run Start event
  - source code version
  - run parameters

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

Facets

- **Extensible:**

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

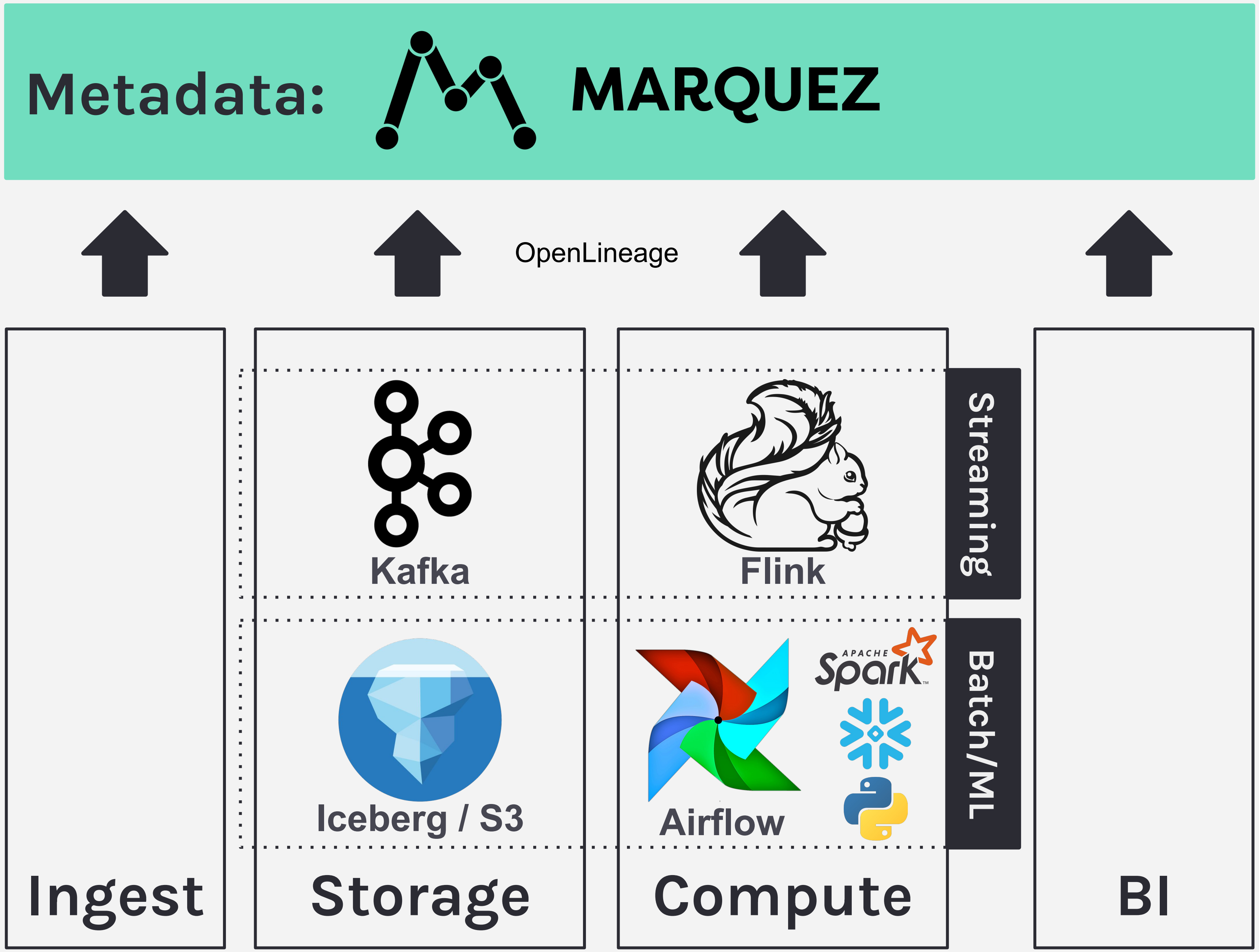
Job:
- Source code
- Dependencies
- params
- Source control
- Query plan
- Query profile

Run:
- Schedule time
- Batch id

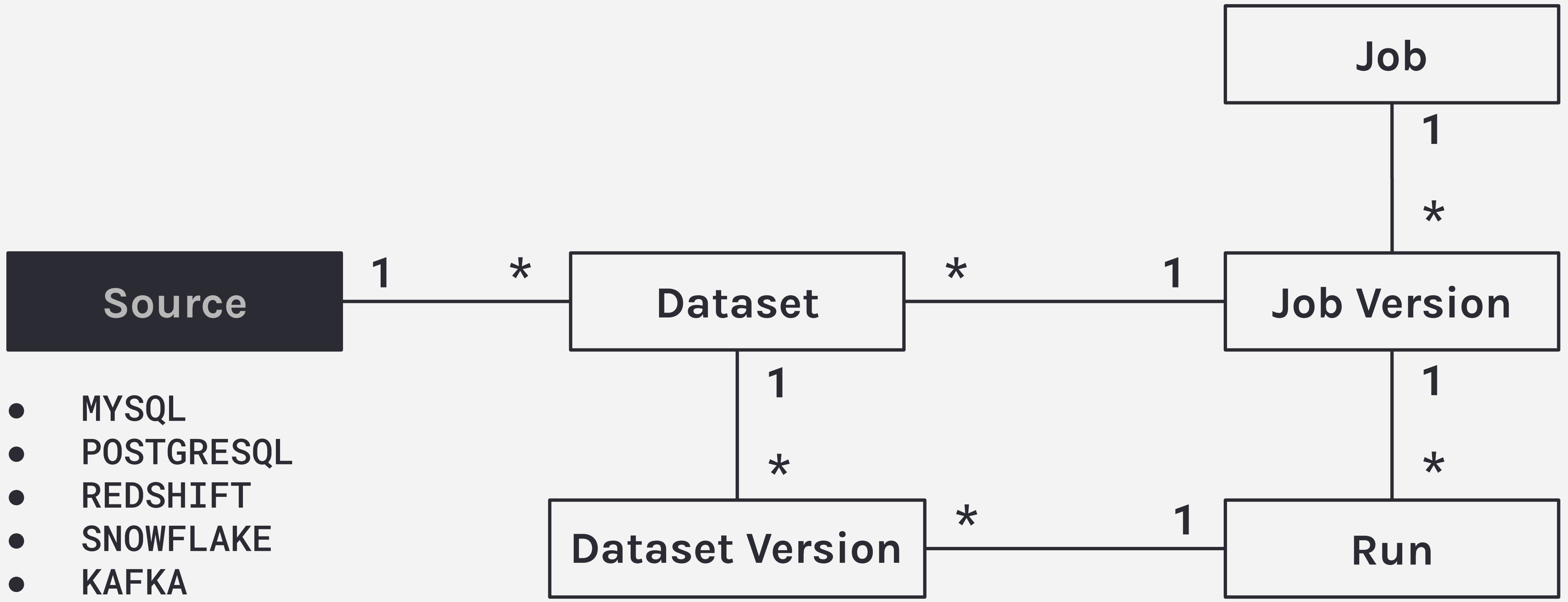
MARQUEZ

- **Data Platform** built around **Marquez**

- **Integrations**
  - Ingest
  - Storage
  - Compute



Marquez: Data model

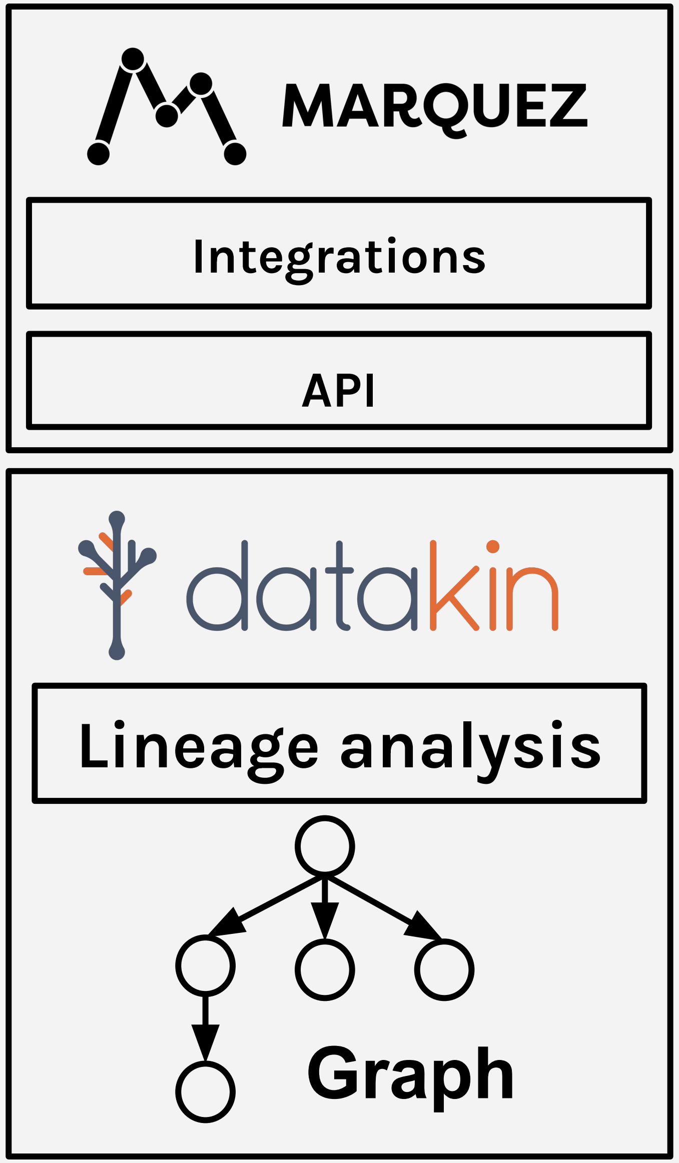


- BATCH
- STREAM
- SERVICE

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

MARQUEZ

Datakin leverages Marquez metadata



- **Open Lineage and Marquez standardize metadata collection**
  - Job runs
  - Parameters
  - Version
  - Inputs / outputs
- **Datakin enables**
  - Understanding operational dependencies
  - Impact analysis
  - Troubleshooting: What has changed since the last time it worked?

Spark observability with OpenLineage

datakin

Spark java agent

spark.driver.extraJavaOptions:

```
-javaagent:marquez-spark-0.13.1.jar={argument}
```

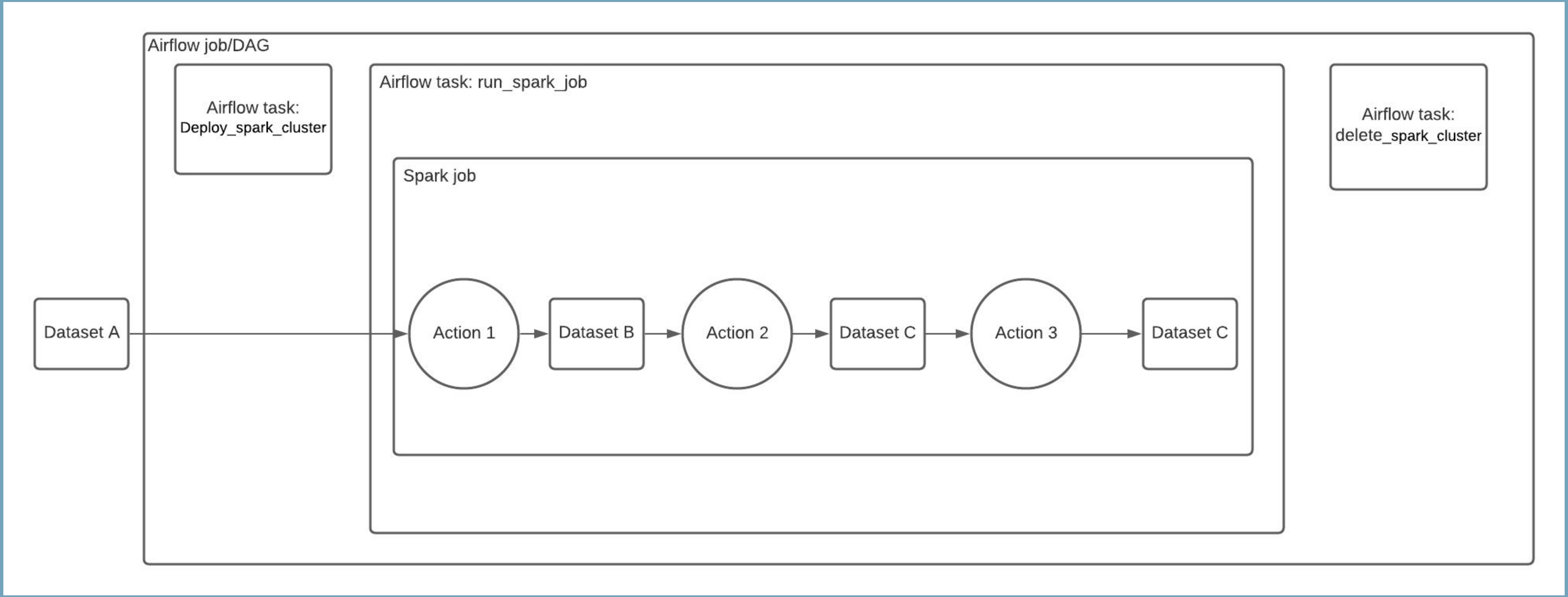
Metadata collected

Lineage: inputs/outputs

Data volume: row count/byte size

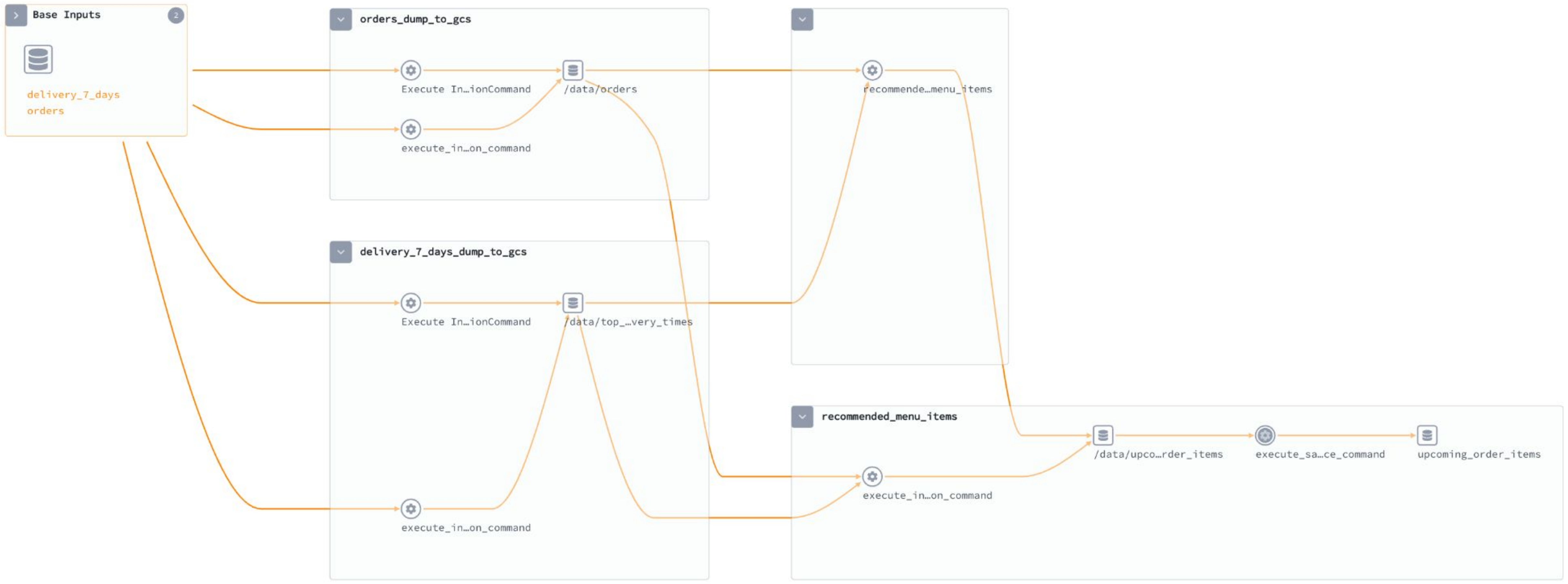
Logical plan

Lineage model



Airflow job/DAG
Airflow task: Deploy_spark_cluster
Airflow task: run_spark_job
Spark job
Dataset A
Action 1
Dataset B
Action 2
Dataset C
Action 3
Dataset C
Airflow task: delete_spark_cluster

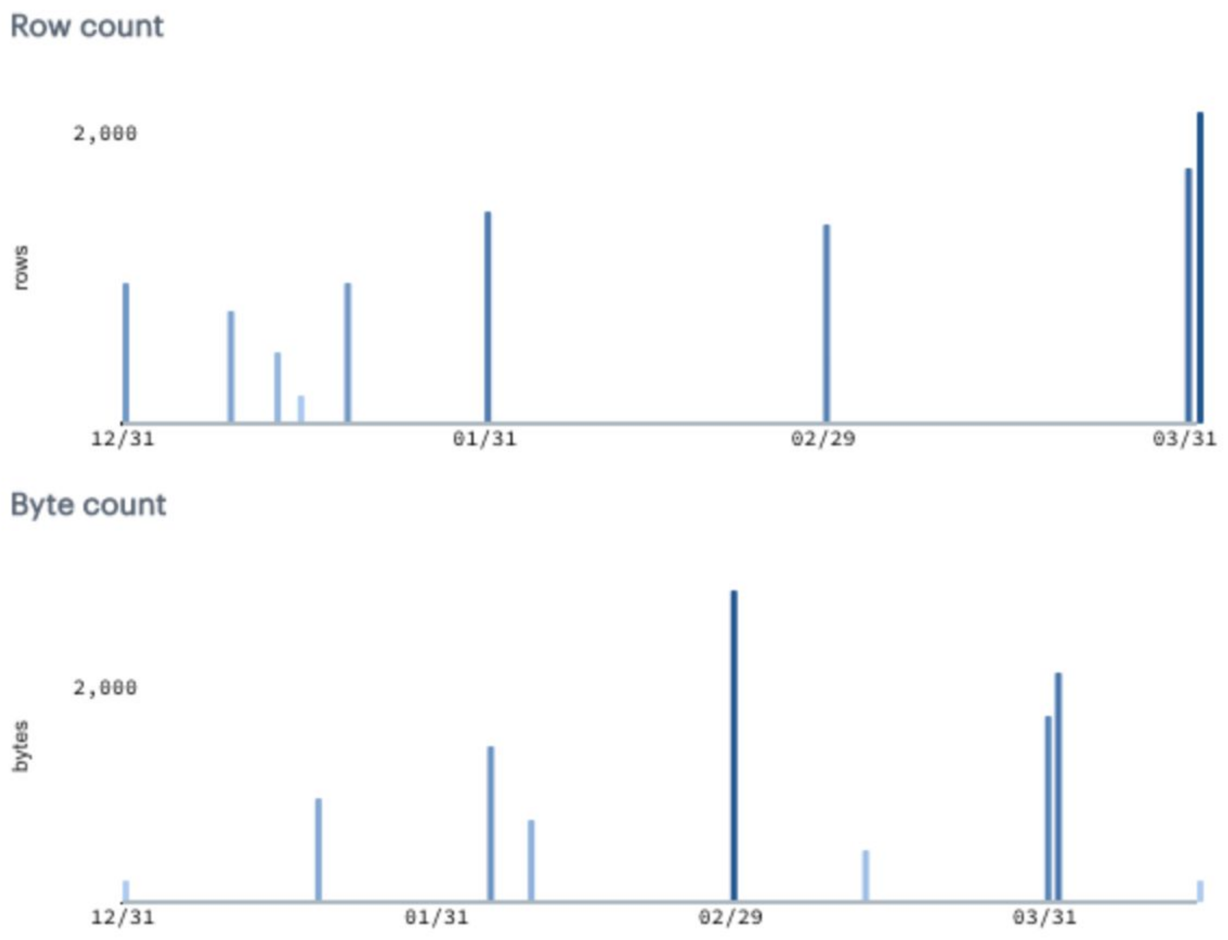
Lineage Example across jobs



datakin

Example of OpenLineage metadata usage:

Data volume evolution



Join the conversation

OpenLineage:

- **Github:** github.com/OpenLineage ★
- **Slack:** OpenLineage.slack.com
- **Twitter:** @OpenLineage
- **Email:** groups.google.com/g/openlineage

Marquez:

- **Github:** github.com/MarquezProject/marquez ★
- **Slack:** MarquezProject.slack.com
- **Twitter:** @MarquezProject