Data lineage and observability with OpenLineage

datakin

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

AGENDA

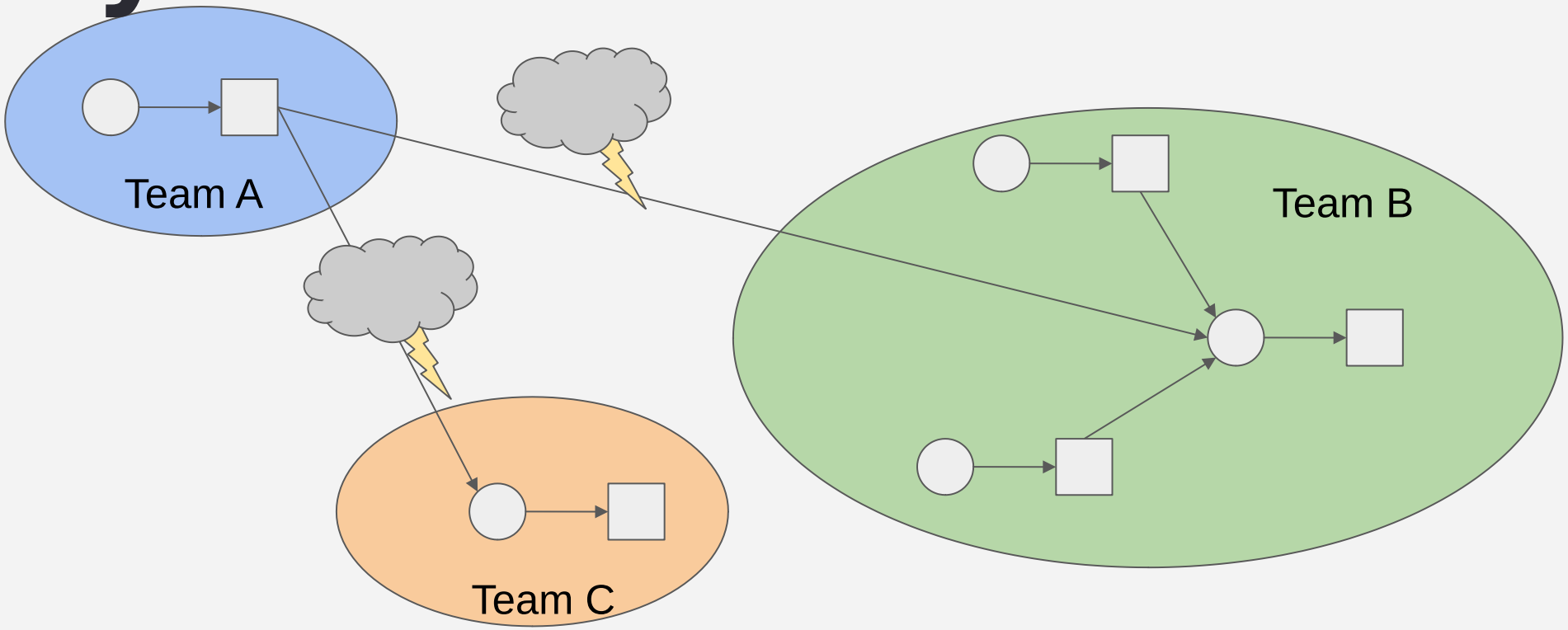
- The need for metadata
- **OpenLineage** - the open standard for lineage collection - and **Marquez**, its reference implementation
- Data observability in practice

datakin

The need for Metadata

datakin

Building a healthy data ecosystem



datakin

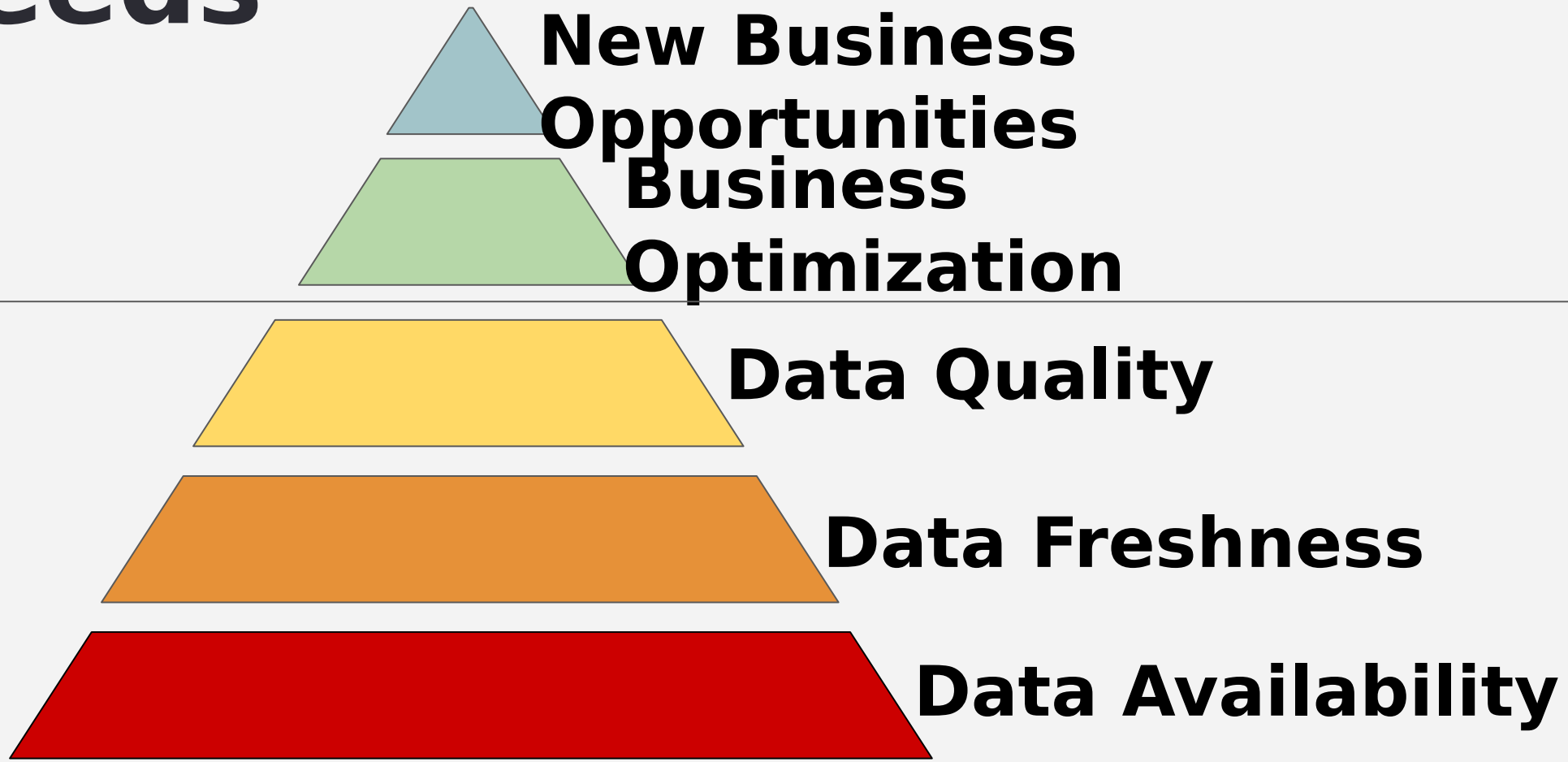
Today: Limited context

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?

datakin

~~Maslow's~~ Data hierarchy of needs



datakin

OpenLineage

datakin

OpenLineage contributors

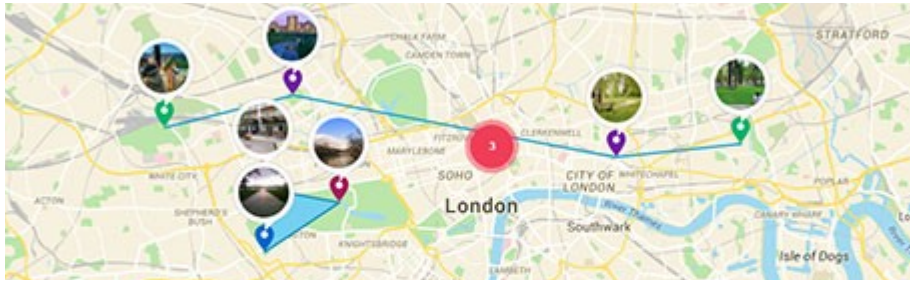
Creators and contributors from major open source projects involved

datakin

Purpose:

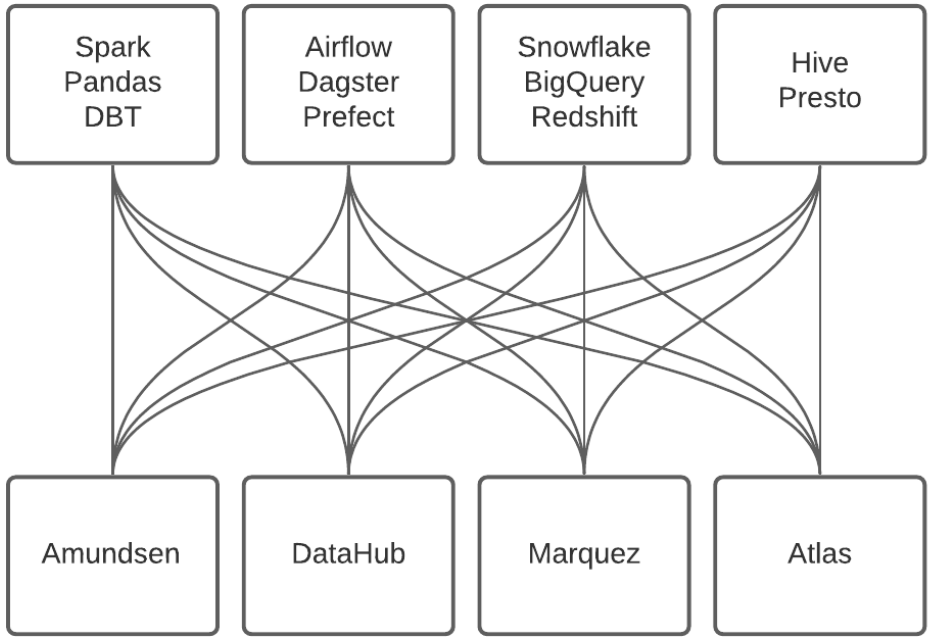
Define an Open standard for metadata and lineage collection by instrumenting data pipelines as they are running.

Purpose:
EXIF for data pipelines

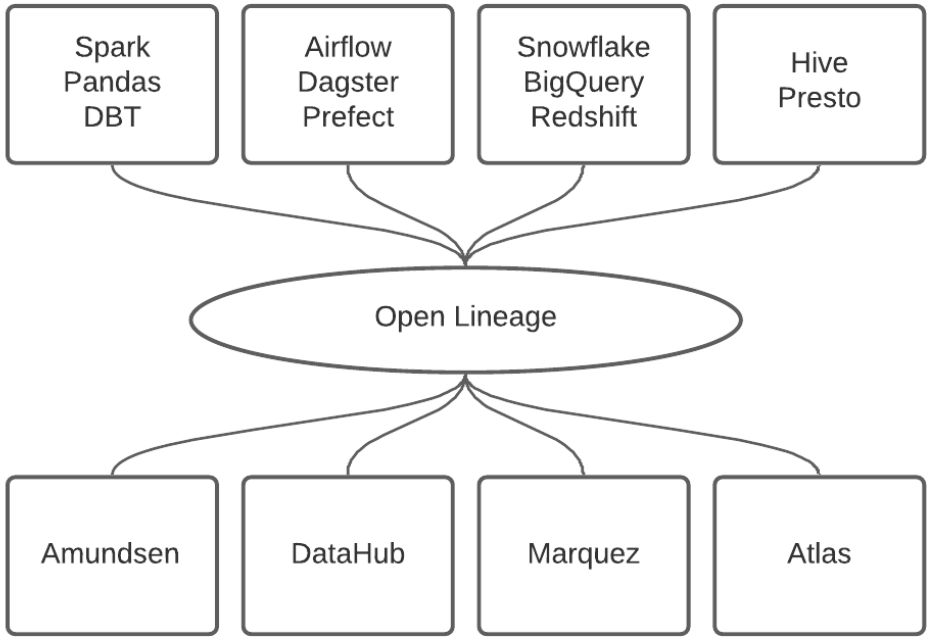


Problem

Before:



With Open Lineage



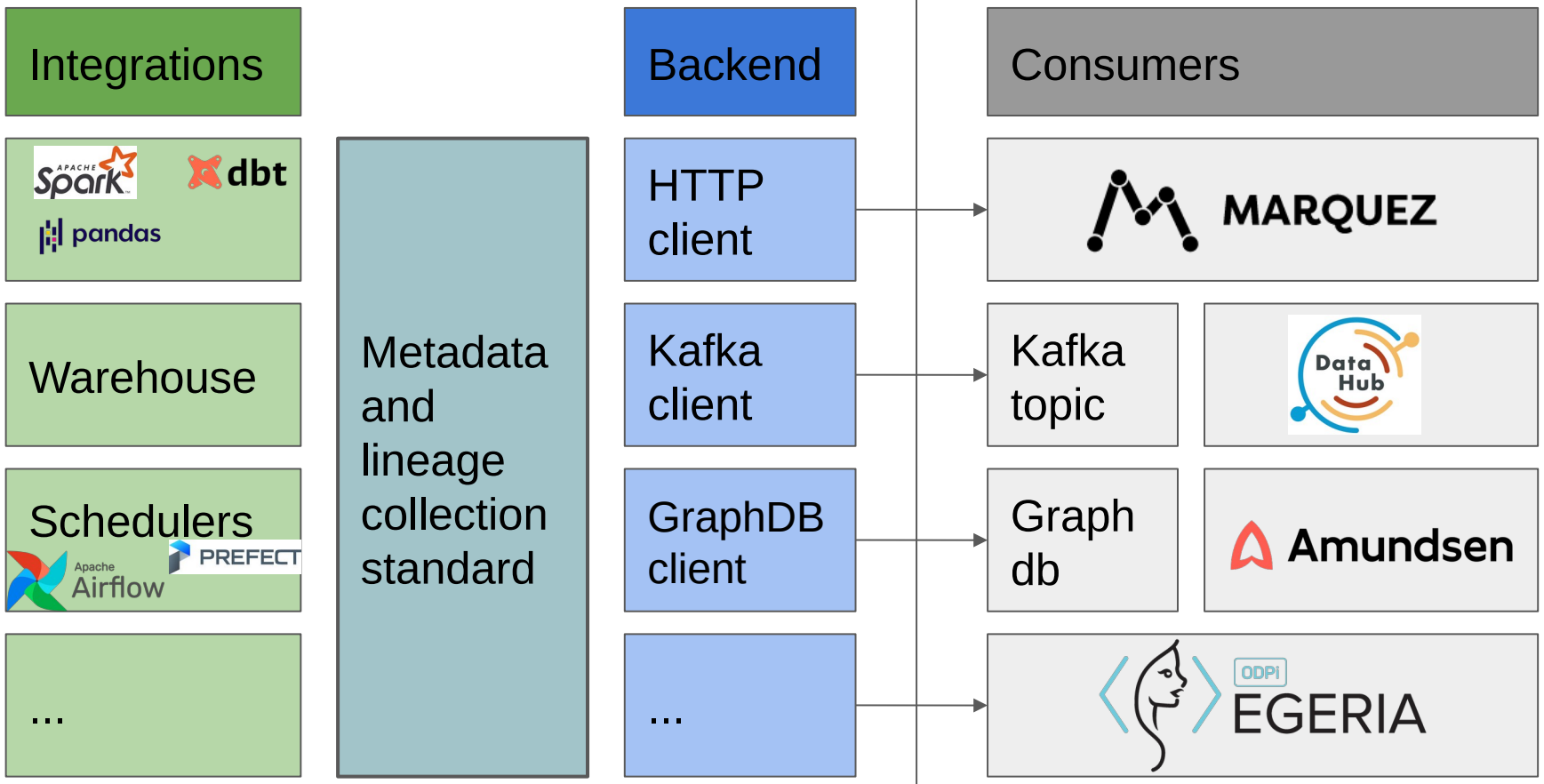
- Duplication of effort: Each project has to instrument all jobs
- Integrations are external and can break with new versions

- 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

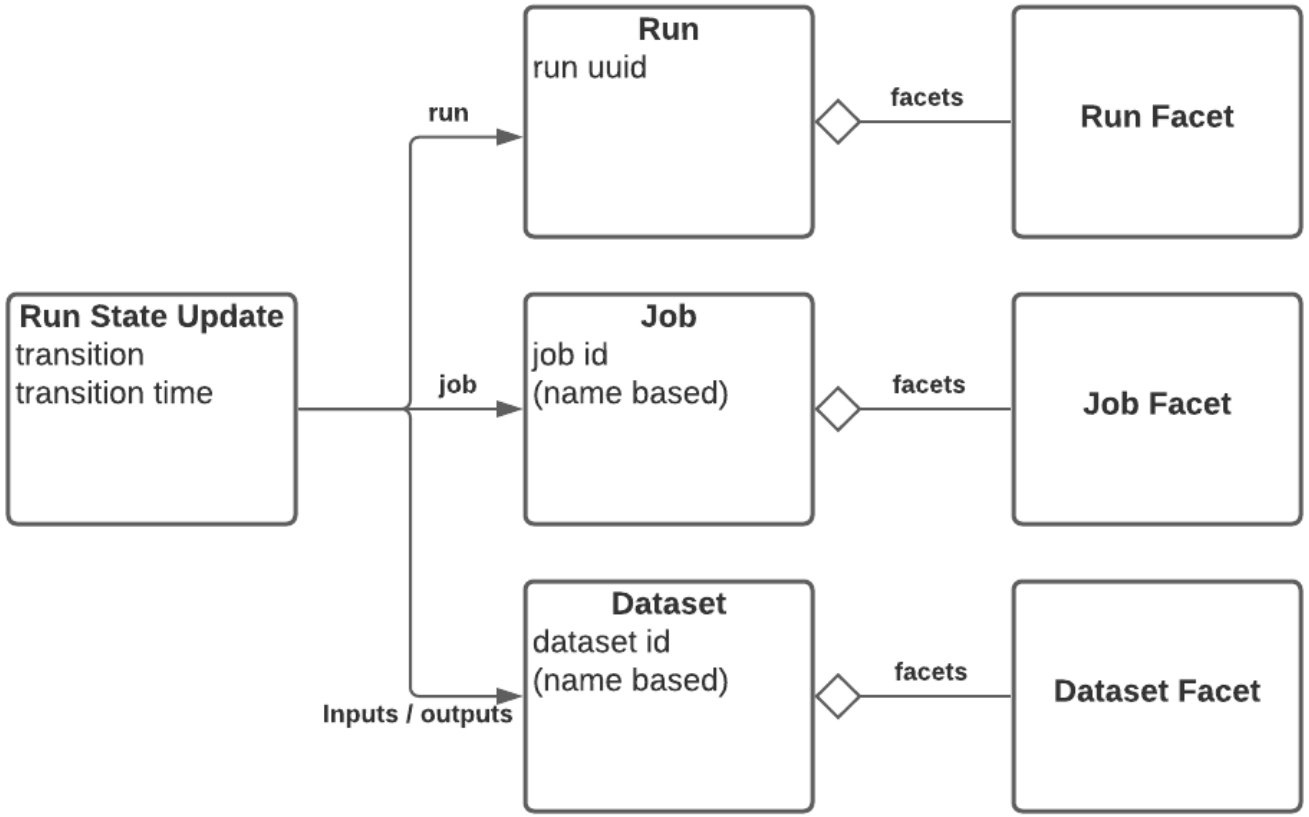


datakin

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

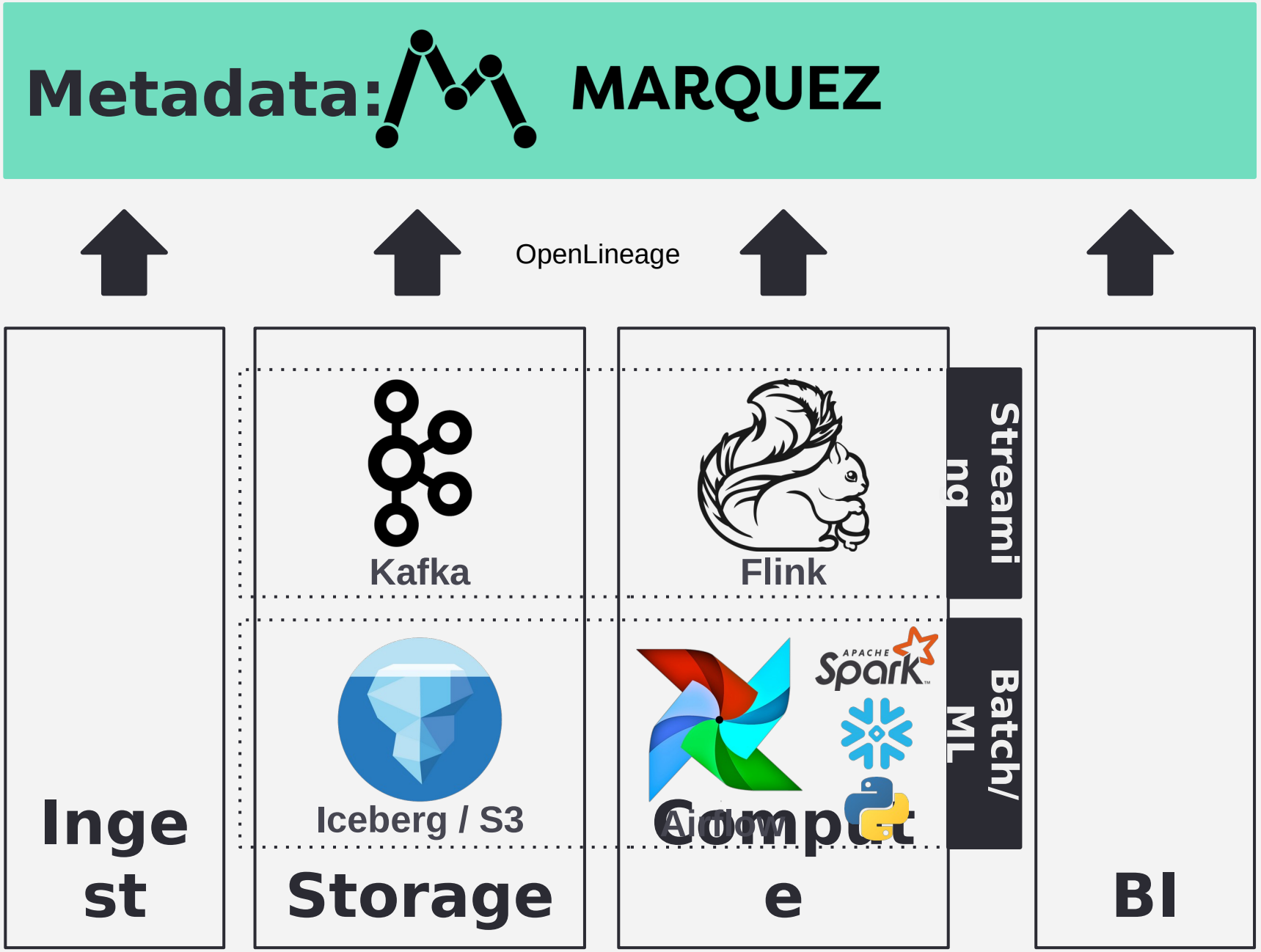
Job:
- Source code
- Dependencies
- Source control
- Query plan

Run:
- Schedule time
- Batch id
- Query profile
- Parameters

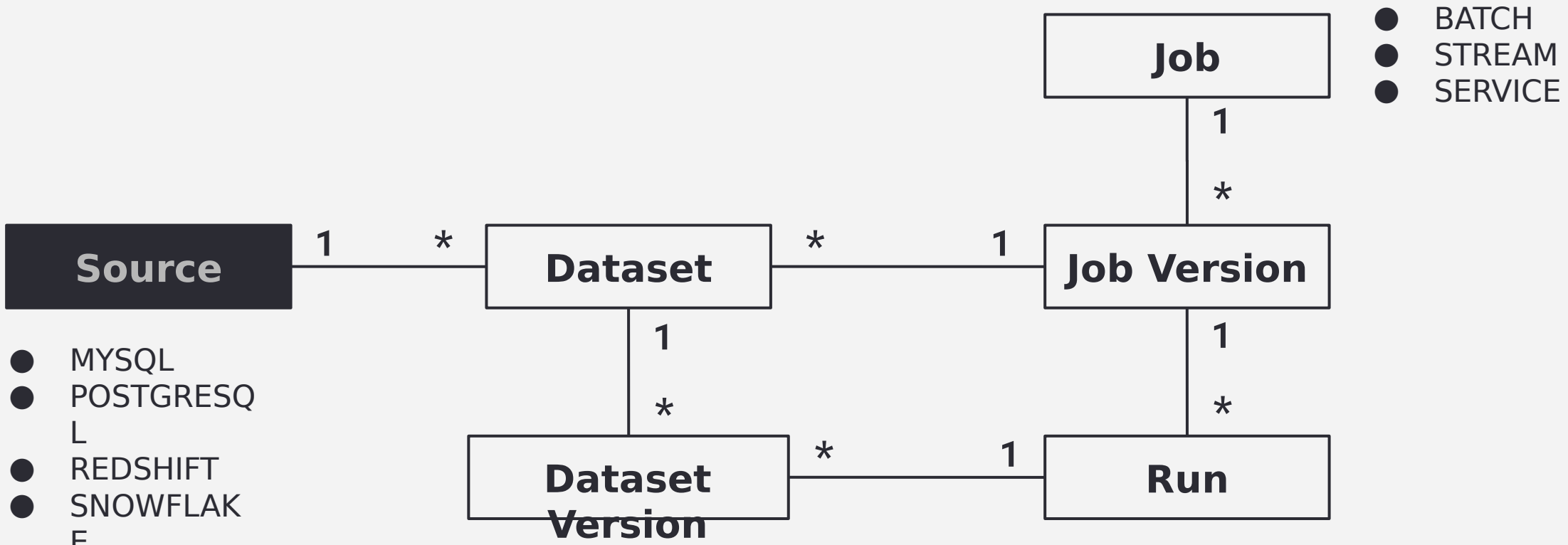
datakin

MARQUEZ

- **Data Platform** built around **Marquez**
- **Integrations**
  - Ingest
  - Storage
  - Compute



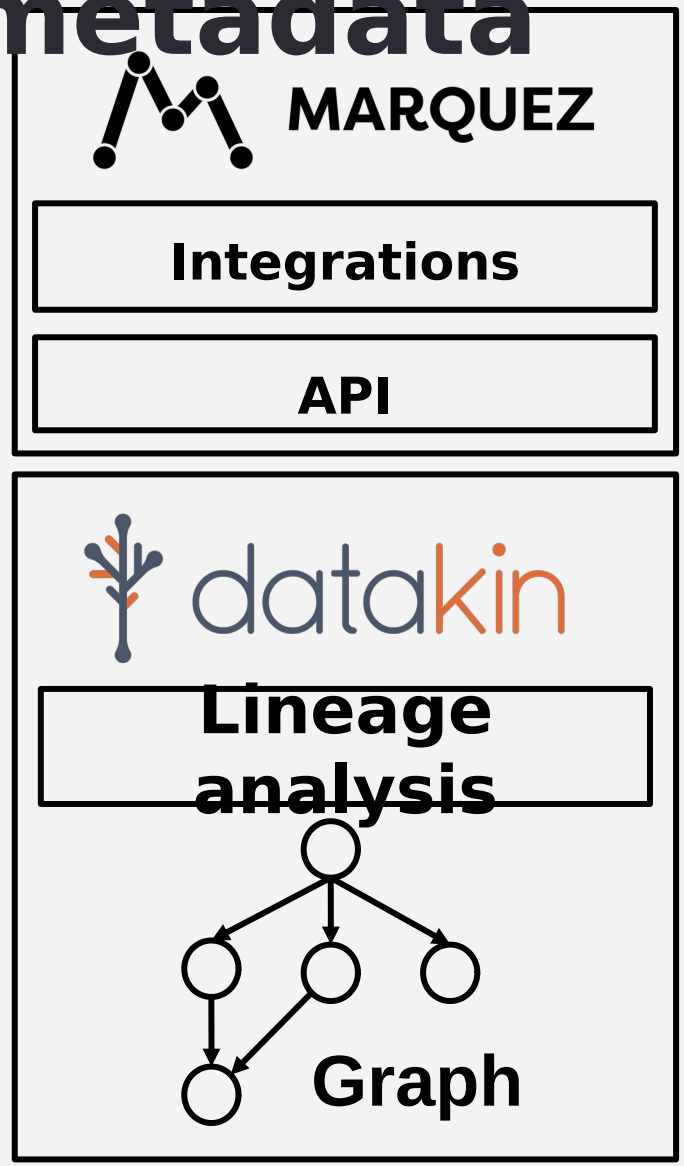
Marquez: Data model



- 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?

datakin

Data observability in practice

datakin

Integrations

Covered:

Apache Airflow

snowflake

Google BigQuery

APACHE Spark™

amazon REDSHIFT

Beta:

great_expectations

dbt

Airflow integration

pip3 install marquez-airflow

MARQUEZ_BACKEND=HTTP

MARQUEZ_URL=http://marquez.example.com

MARQUEZ_NAMESPACE=my_namespace

Spark java agent

spark.driver.extraJavaOptions:

-javaagent:marquez-spark-{version}.jar**=**

http://marquez.example.com/api/v1/namespaces/my_namespace/jobs/{job_name}/runs/{uuid}

Metadata collected

Lineage: inputs/outputs

Data volume: row count/byte size

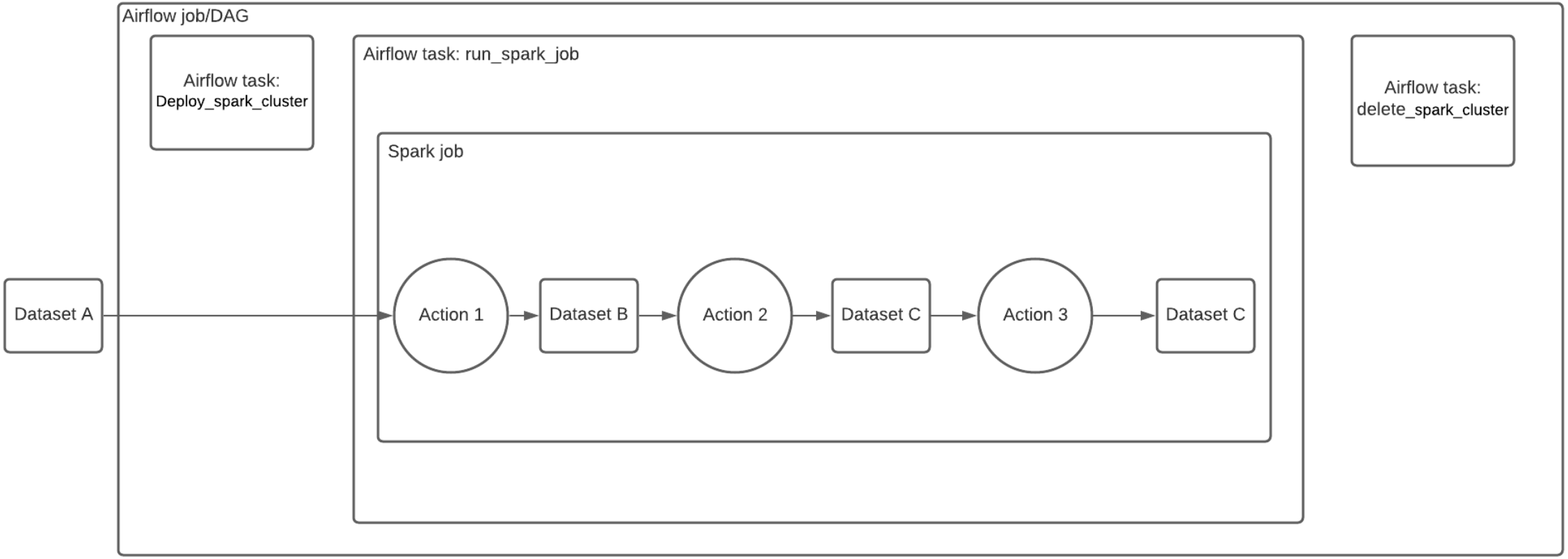
Logical plan

source code location / version

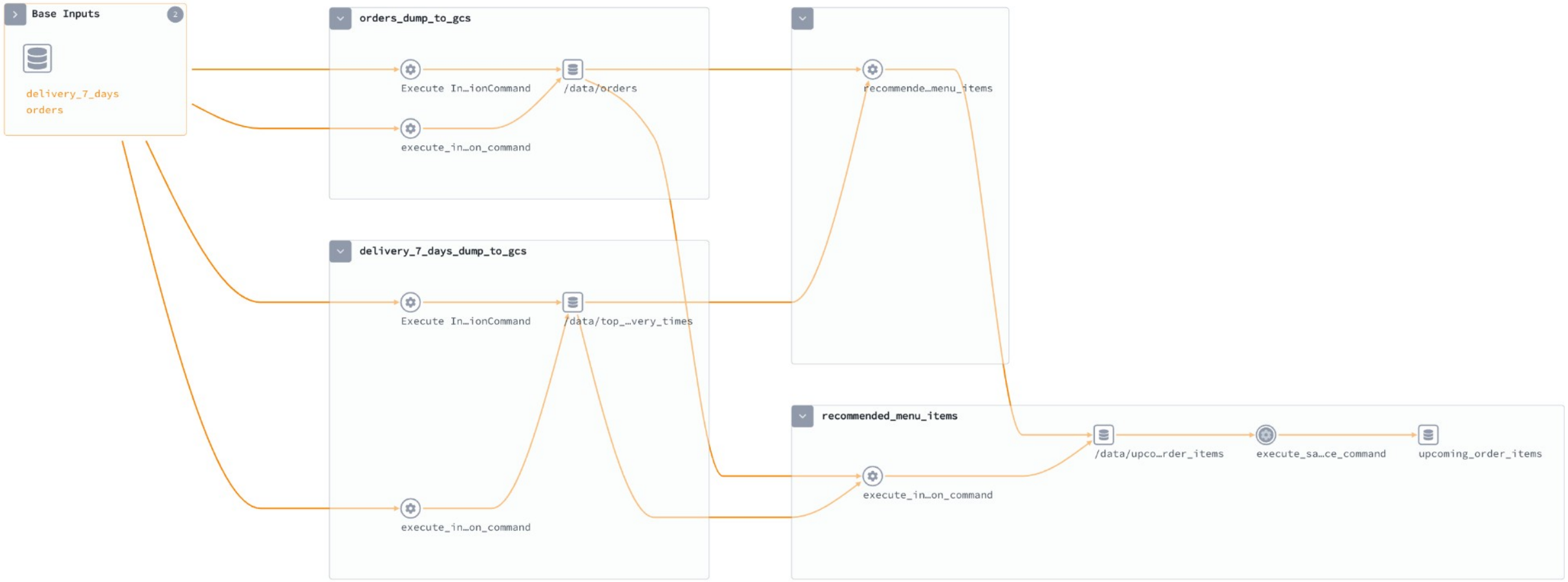
schema

processing time

Lineage model



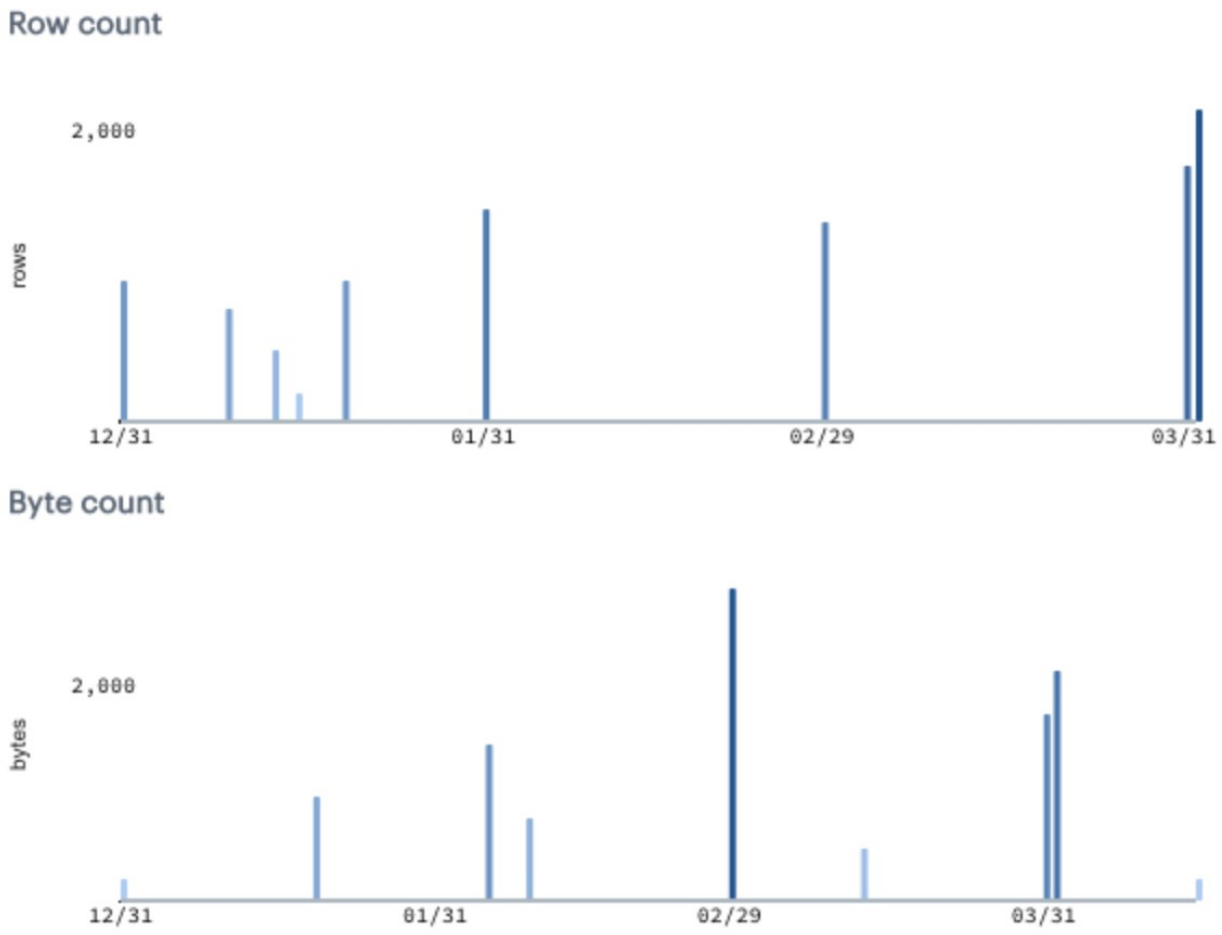
Lineage Example across jobs



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

datakin