OpenLineage: from Operators to Hooks

OpenLineage LF AI & DATA ASTRONOMER

Maciej Obuchowski

in

Astronomer

Maciej is a software engineer, Airflow and OpenLineage commiter. He loves rock climbing, contributing to open source data projects and playing with cats.

Sessions by Maciej Obuchowski

- OpenLineage: From Operators to Hooks (2024)
- OpenLineage in Airflow: A Comprehensive Guide (2023)
- OpenLineage & Airflow - data lineage has never been easier (2022)

OpenLineage

ASTRONOMER

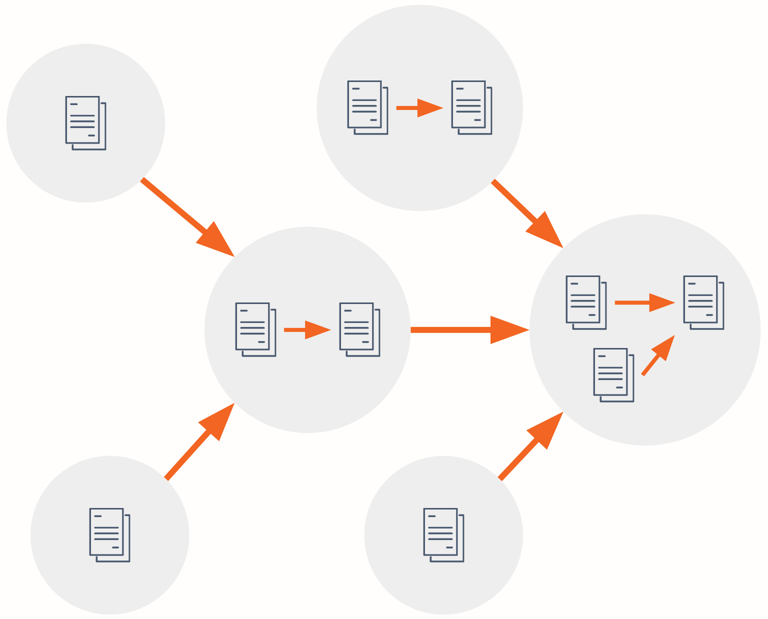
Agenda

- What is OpenLineage
- OpenLineage Airflow Integration
- Getting Lineage From Hooks
- Peek into the future?

What is Data Lineage?

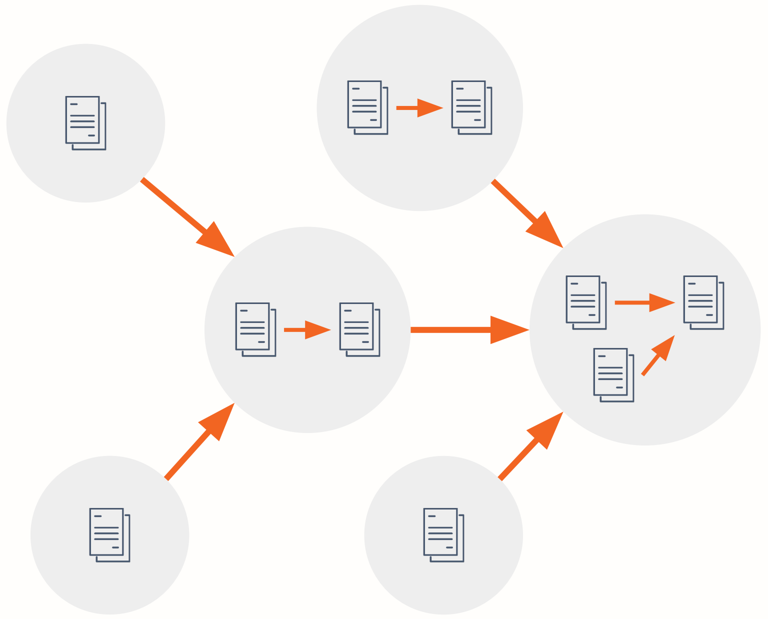
Data lineage is the set of complex relationships between datasets and jobs in data pipelines.

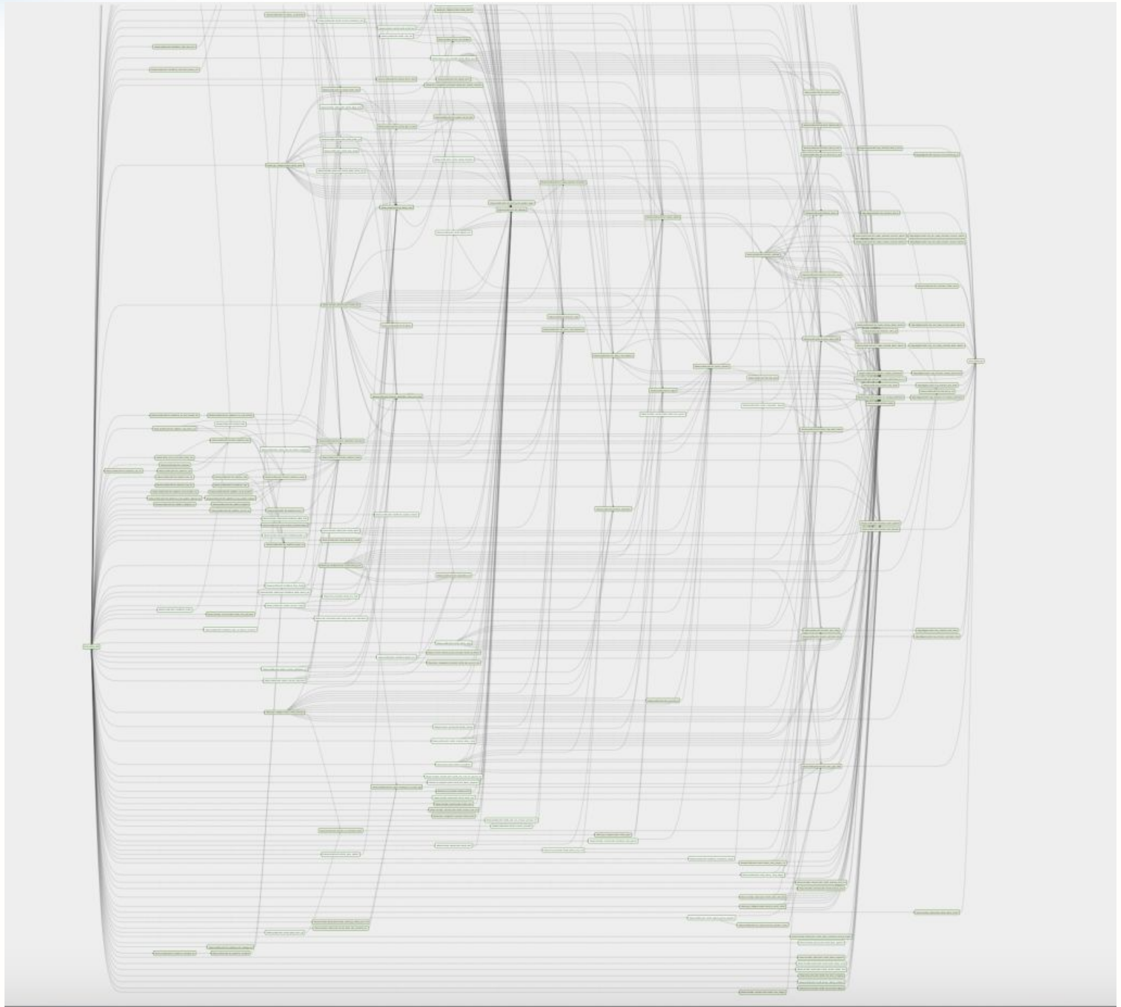
- Producers & consumers of each dataset
- Inputs and outputs of each job



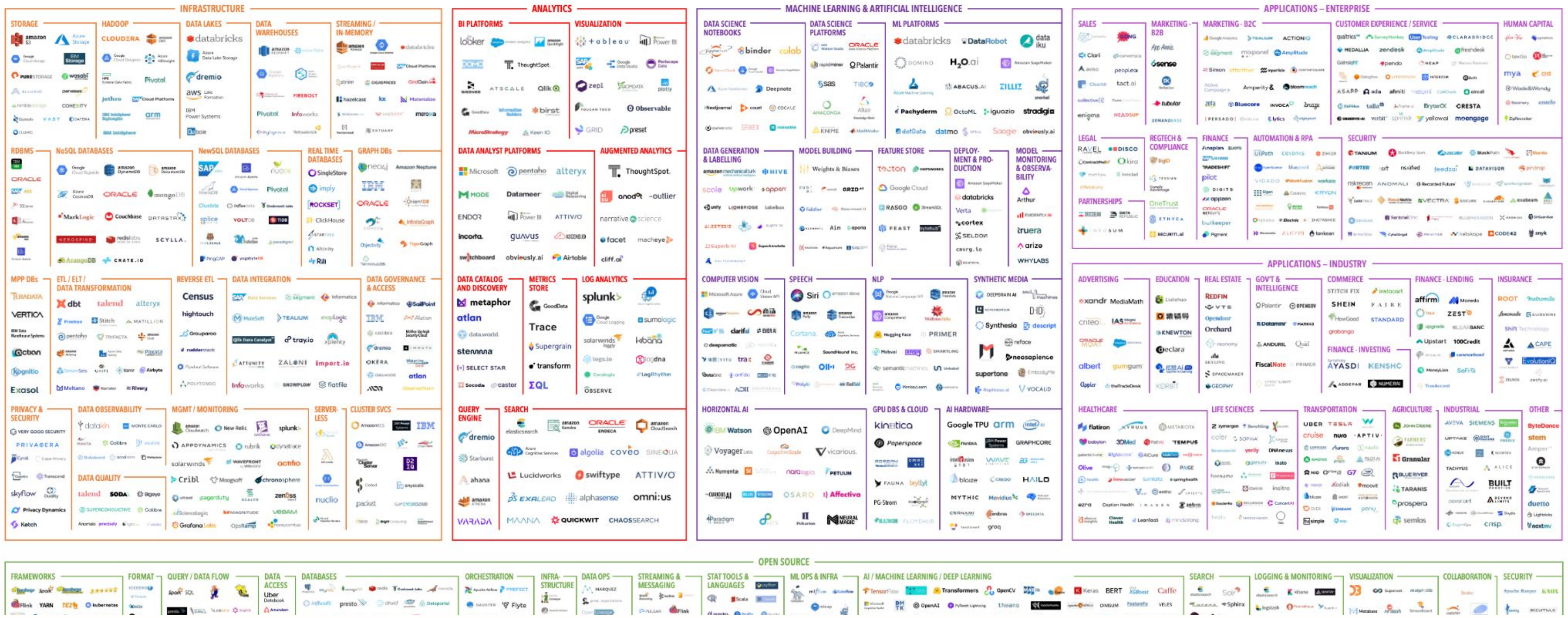
What problems Data Lineage solves

- Holistic view on data flowing through organization
- Security and Compliance
  - A German bank suffered data breach from vendor - but was wholly unaware that PII data was being send there.
  - Prod data send to dev env
- Impact analysis - pipeline failed, which other datasets it affects





Working with data in 2024



OpenLineage

ASTRONOMER

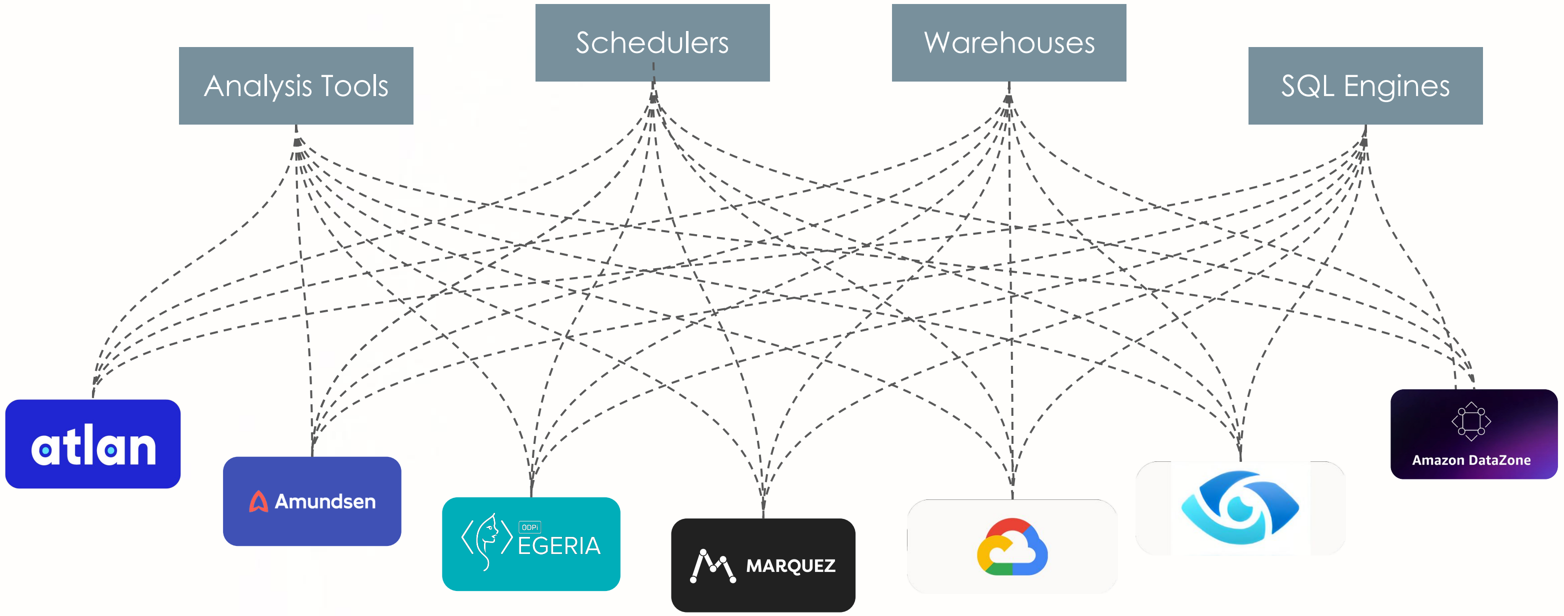
OpenLineage

Mission:

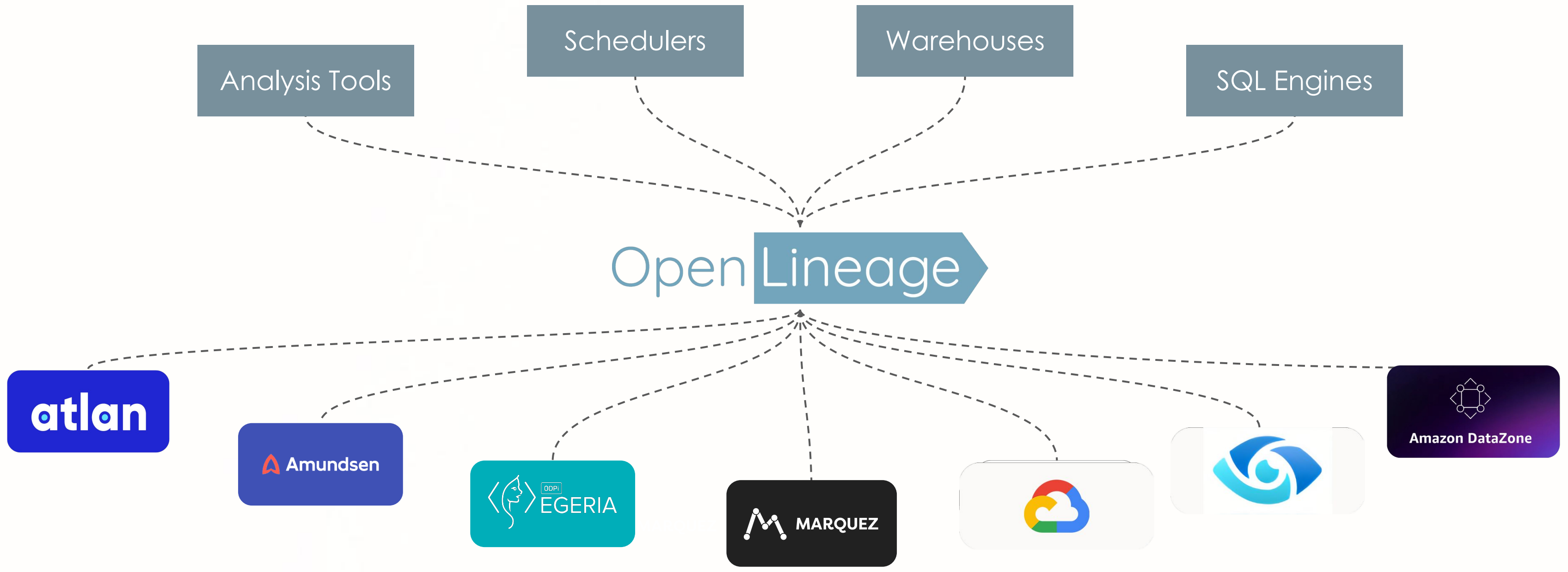
To define an **open standard** for the collection of lineage metadata from pipelines **as they are running**.

LF AI & DATA

The Data World Without OpenLineage

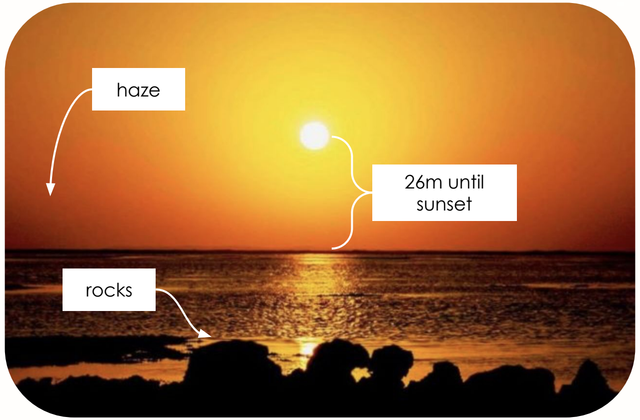


The Data World With OpenLineage

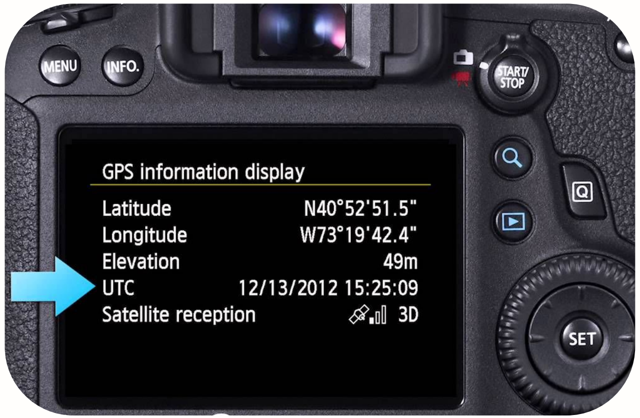


https://openlineage.io/ecosystem

Why runtime?



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



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

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

OpenLineage Integrations

Metadata producers

Apache Airflow

APACHE Spark

dbt

Flink

trino

Google Cloud

ODPi EGERIA

databricks

Keboola

Metadata consumers

Microsoft

Amundsen

ASTRONOMER

ODPi EGERIA

manta

Amazon DataZone

MARQUEZ

atlan

Google Cloud

snowflake

Grai

Collibra

DataHub

OpenLineage Contributors

ASTRONOMER

getindata
Part of Xebia

pandas

DECATHLON

Microsoft

APACHE Spark™

dbt

Amundsen

Parquet

snowflake

ODPi EGERIA

ICEBERG

Apache Airflow

MARQUEZ

matillion

asana

NATURAL INTELLIGENCE

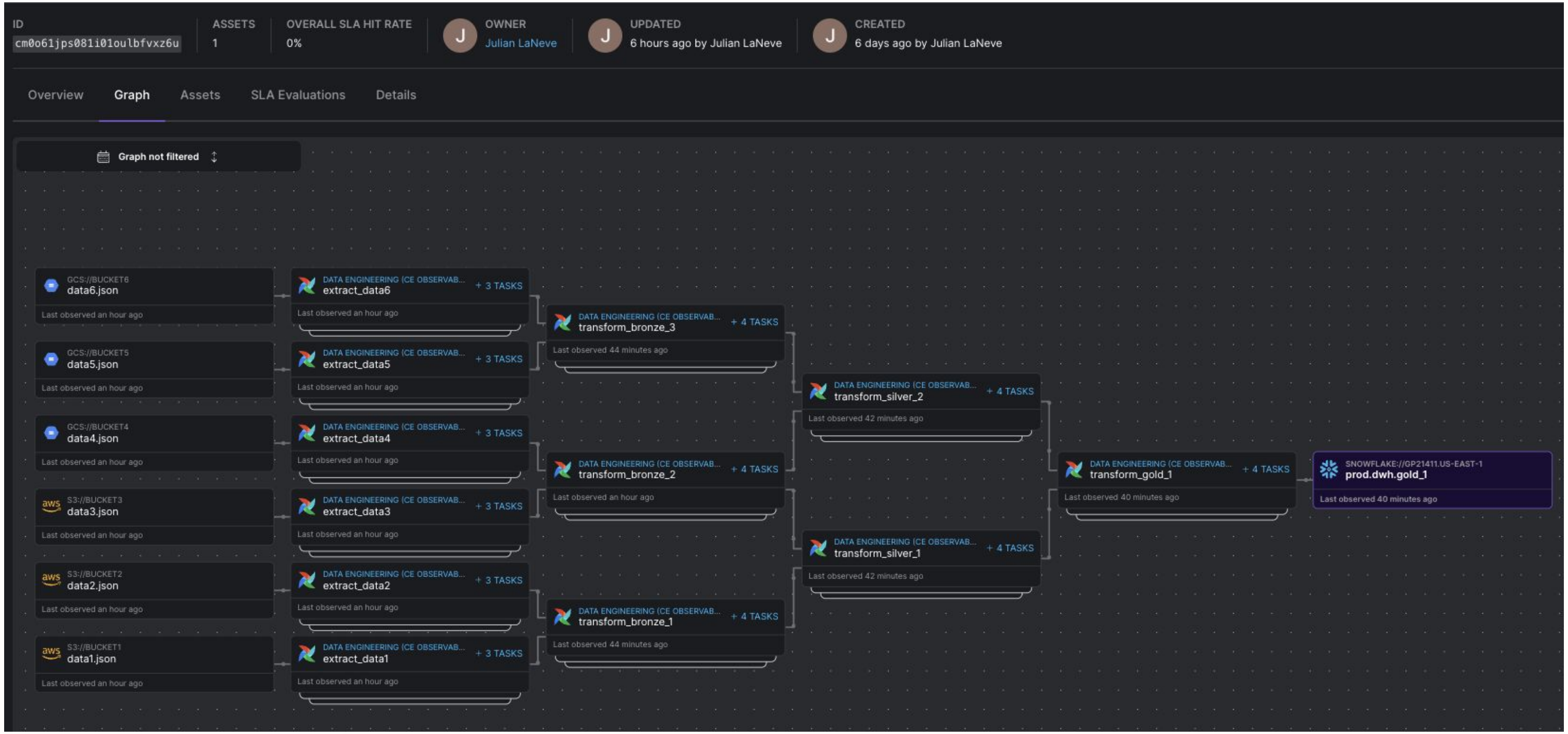
Bloomberg®

Booking.com

OpenLineage

ASTRONOMER

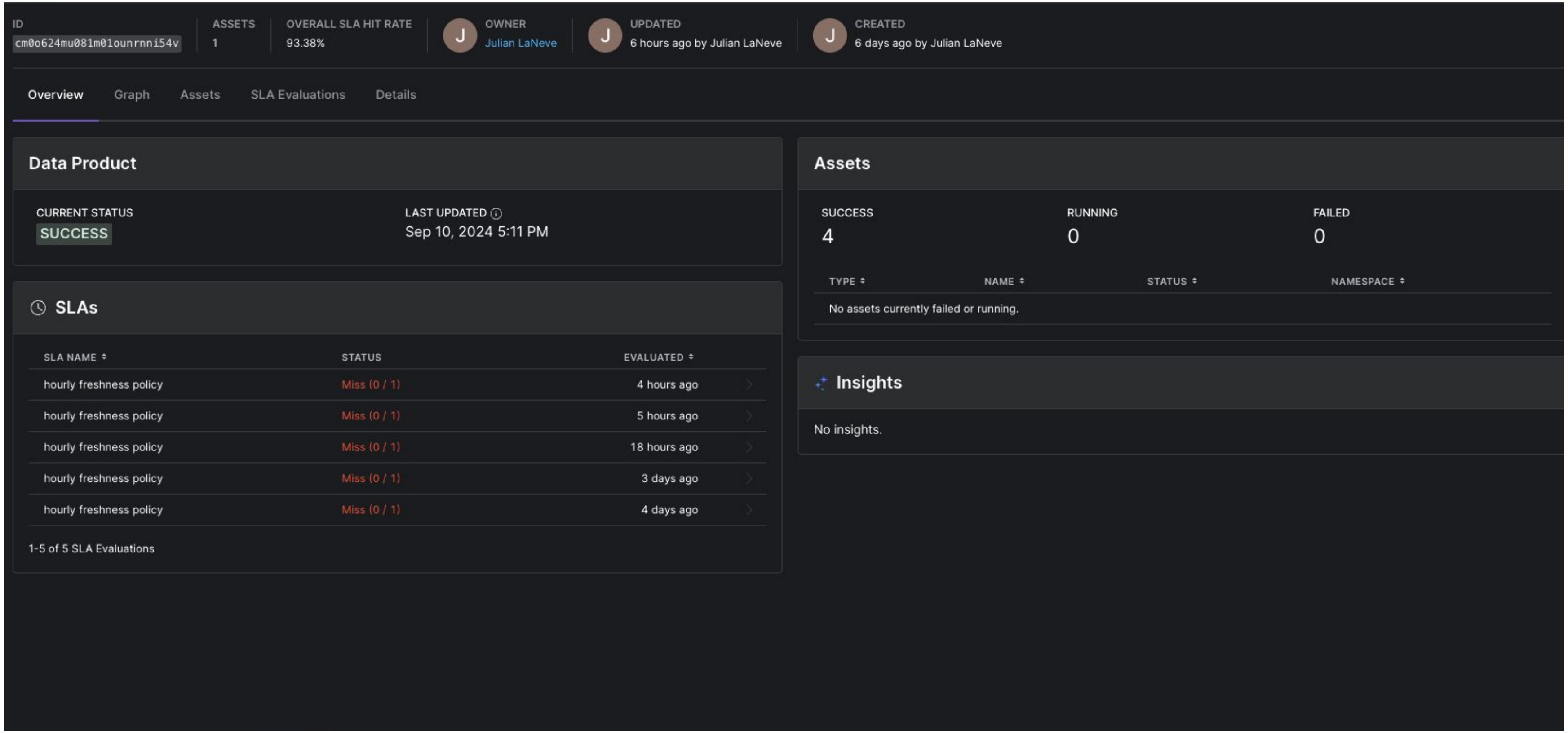
Astro Observe



ID: cm0o61jps081i01oulbfvxz6u
ASSETS: 1
OVERALL SLA HIT RATE: 0%
OWNER: Julian LaNeve
UPDATED: 6 hours ago by Julian LaNeve
CREATED: 6 days ago by Julian LaNeve

Overview
Graph
Assets
SLA Evaluations
Details
Graph not filtered
GCS://BUCKET6 data6.json Last observed an hour ago
GCS://BUCKET5 data5.json Last observed an hour ago
GCS://BUCKET4 data4.json Last observed an hour ago
S3://BUCKET3 data3.json Last observed an hour ago
S3://BUCKET2 data2.json Last observed an hour ago
S3://BUCKET1 data1.json Last observed an hour ago
DATA ENGINEERING (CE OBSERVAB... extract_data6 + 3 TASKS Last observed an hour ago
DATA ENGINEERING (CE OBSERVAB... extract_data5 + 3 TASKS Last observed an hour ago
DATA ENGINEERING (CE OBSERVAB... extract_data4 + 3 TASKS Last observed an hour ago
DATA ENGINEERING (CE OBSERVAB... extract_data3 + 3 TASKS Last observed an hour ago
DATA ENGINEERING (CE OBSERVAB... extract_data2 + 3 TASKS Last observed an hour ago
DATA ENGINEERING (CE OBSERVAB... extract_data1 + 3 TASKS Last observed an hour ago
DATA ENGINEERING (CE OBSERVAB... transform_bronze_3 + 4 TASKS Last observed 44 minutes ago
DATA ENGINEERING (CE OBSERVAB... transform_bronze_2 + 4 TASKS Last observed an hour ago
DATA ENGINEERING (CE OBSERVAB... transform_bronze_1 + 4 TASKS Last observed 44 minutes ago
DATA ENGINEERING (CE OBSERVAB... transform_silver_2 + 4 TASKS Last observed 42 minutes ago
DATA ENGINEERING (CE OBSERVAB... transform_silver_1 + 4 TASKS Last observed 42 minutes ago
DATA ENGINEERING (CE OBSERVAB... transform_gold_1 + 4 TASKS Last observed 40 minutes ago
SNOWFLAKE://GP21411.US-EAST-1 prod.dwh.gold_1 Last observed 40 minutes ago

Astro Observe



ID cm0o624mu081m01ounrnni54v | ASSETS 1 | OVERALL SLA HIT RATE 93.38% | OWNER Julian LaNeve | UPDATED 6 hours ago by Julian LaNeve | CREATED 6 days ago by Julian LaNeve

Overview | Graph | Assets | SLA Evaluations | Details

Data Product

CURRENT STATUS
SUCCESS

LAST UPDATED
Sep 10, 2024 5:11 PM

SLAs

SLA NAME	STATUS	EVALUATED
hourly freshness policy	Miss (0 / 1)	4 hours ago
hourly freshness policy	Miss (0 / 1)	5 hours ago
hourly freshness policy	Miss (0 / 1)	18 hours ago
hourly freshness policy	Miss (0 / 1)	3 days ago
hourly freshness policy	Miss (0 / 1)	4 days ago

1-5 of 5 SLA Evaluations

Assets

SUCCESS	RUNNING	FAILED
4	0	0

TYPE	NAME	STATUS	NAMESPACE

No assets currently failed or running.

Insights

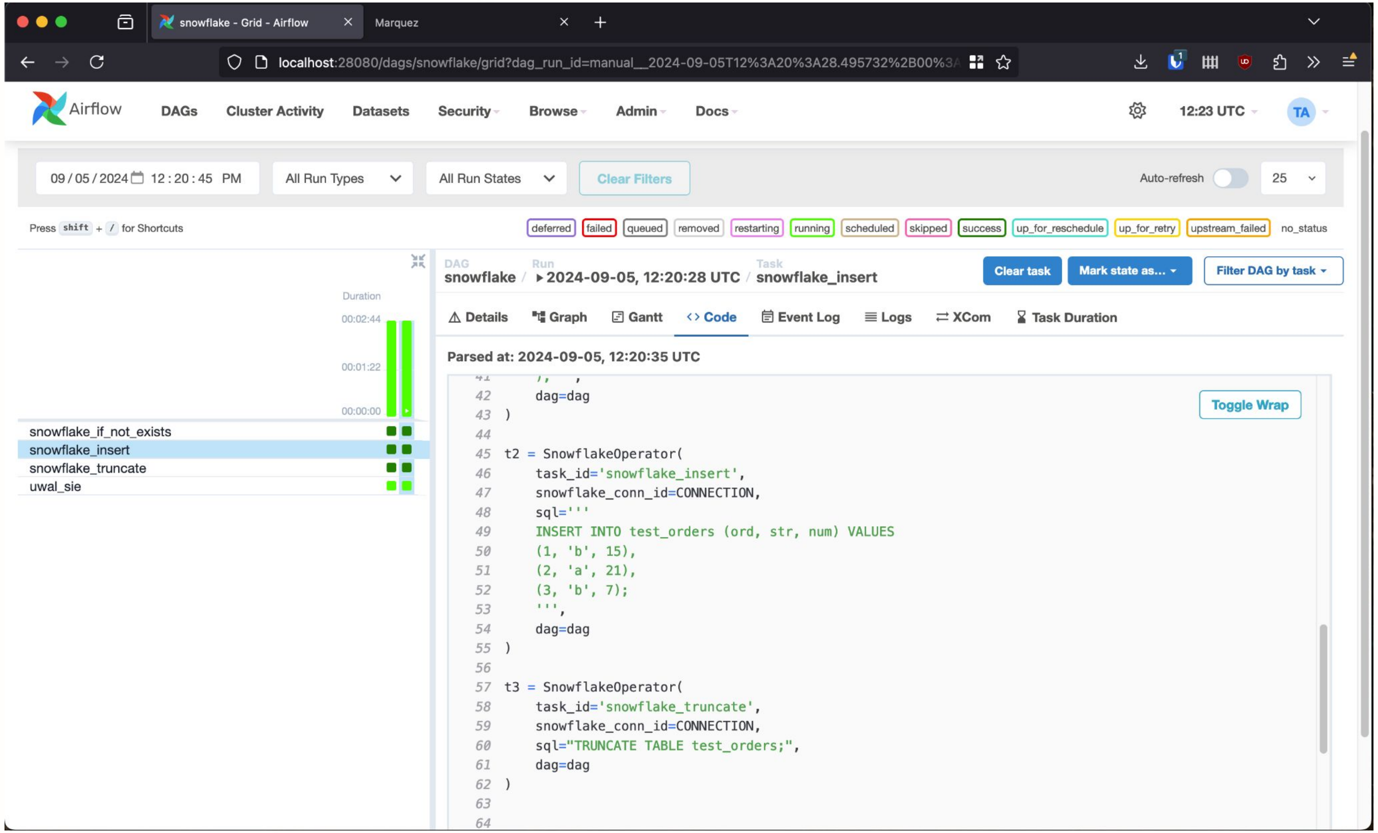
No insights.

OpenLineage

ASTRONOMER

OpenLineage Airflow Integration

What does it do



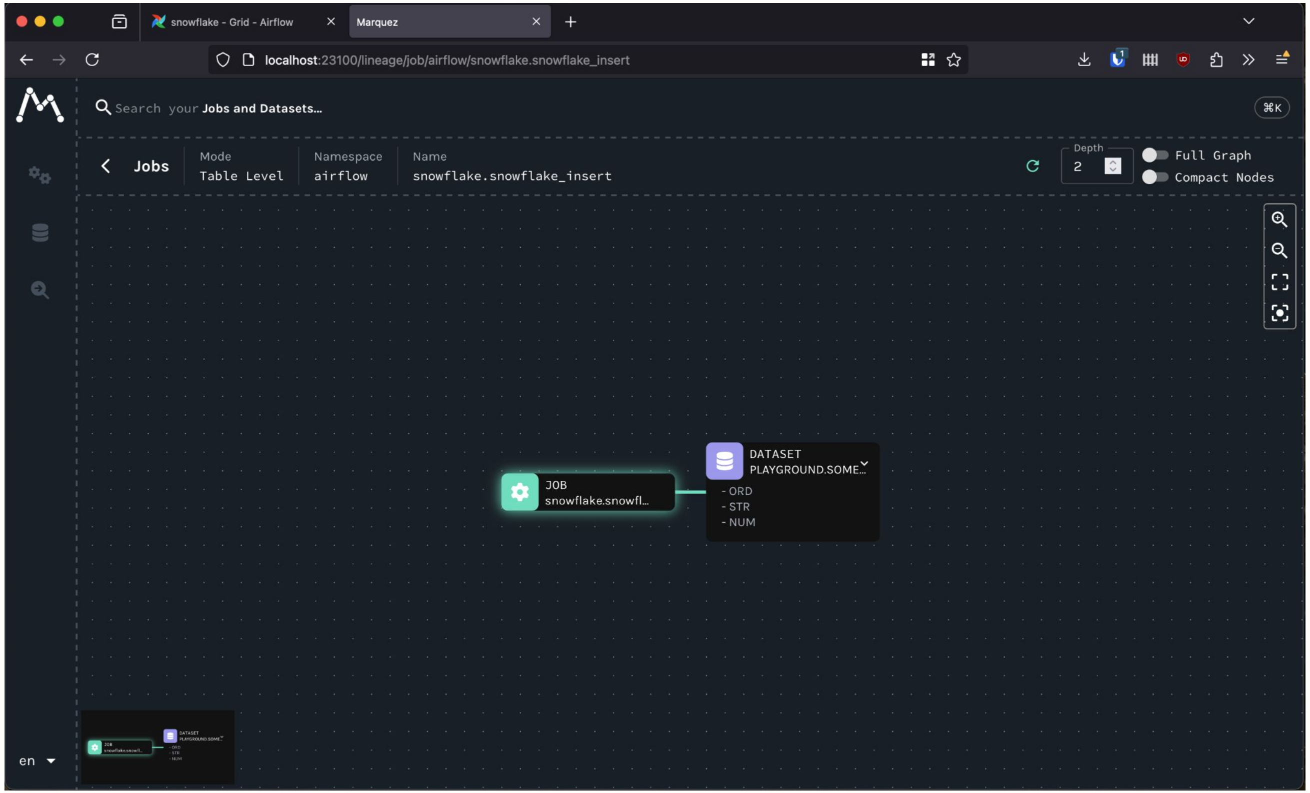
Parsed at: 2024-09-05, 12:20:35 UTC

```
    ),
    dag=dag
)

t2 = SnowflakeOperator(
    task_id='snowflake_insert',
    snowflake_conn_id=CONNECTION,
    sql='''
    INSERT INTO test_orders (ord, str, num) VALUES
    (1, 'b', 15),
    (2, 'a', 21),
    (3, 'b', 7);
    ''',
    dag=dag
)

t3 = SnowflakeOperator(
    task_id='snowflake_truncate',
    snowflake_conn_id=CONNECTION,
    sql="TRUNCATE TABLE test_orders;",
    dag=dag
)

```

OpenLineage

ASTRONOMER

What does it do



snowflake - Grid - Airflow

Marquez

localhost:23100/lineage/job/airflow/snowflake.snowflake_insert

Search your Jobs and Datasets...

Jobs

Mode: Table Level

Namespace: airflow

Name: snowflake.snowflake_insert

Depth: 2

Full Graph

Compact Nodes

JOB
snowflake.snowfl...

DATASET
PLAYGROUND.SOME...
- ORD
- STR
- NUM

en

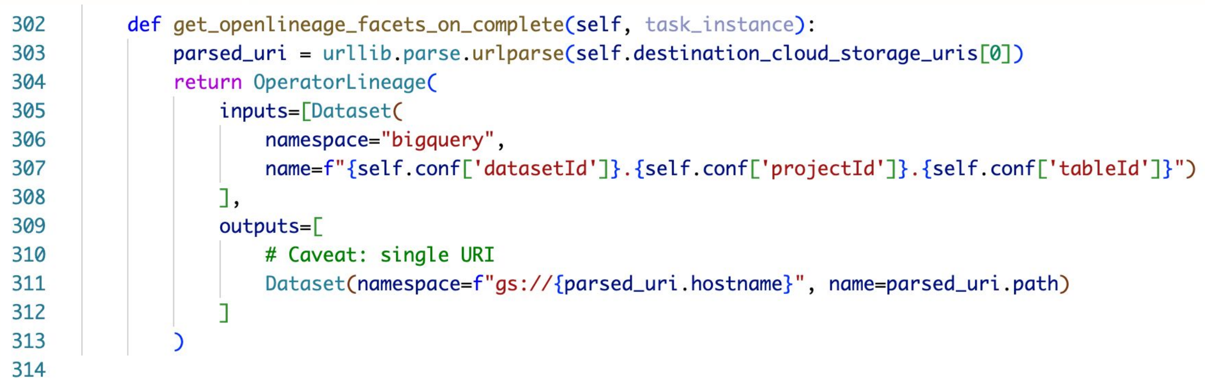
OpenLineage

ASTRONOMER

2.7+ AIP-53 Implementation

- OpenLineage is part of Airflow since 2.7+ introduced in AIP-53
- Part of implementation happens in Operators
- START, COMPLETE, FAIL states are exposed via different get_openlineage_facets_* methods returning OperatorLineage class

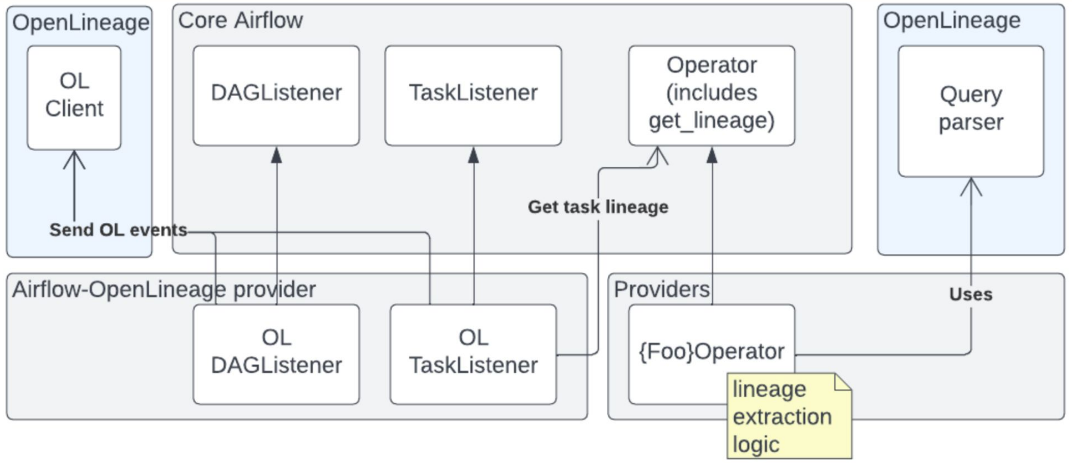
Ex. on BigQueryToGCSOperator



```python
    def get_openlineage_facets_on_complete(self, task_instance):
        parsed_uri = urllib.parse.urlparse(self.destination_cloud_storage_uris[0])
        return OperatorLineage(
            inputs=[Dataset(
                namespace="bigquery",
                name=f"{self.conf['datasetId']}.{self.conf['projectId']}.{self.conf['tableId']}")
            ],
            outputs=[
                # Caveat: single URI
                Dataset(namespace=f"gs://{parsed_uri.hostname}", name=parsed_uri.path)
            ]
        )

```

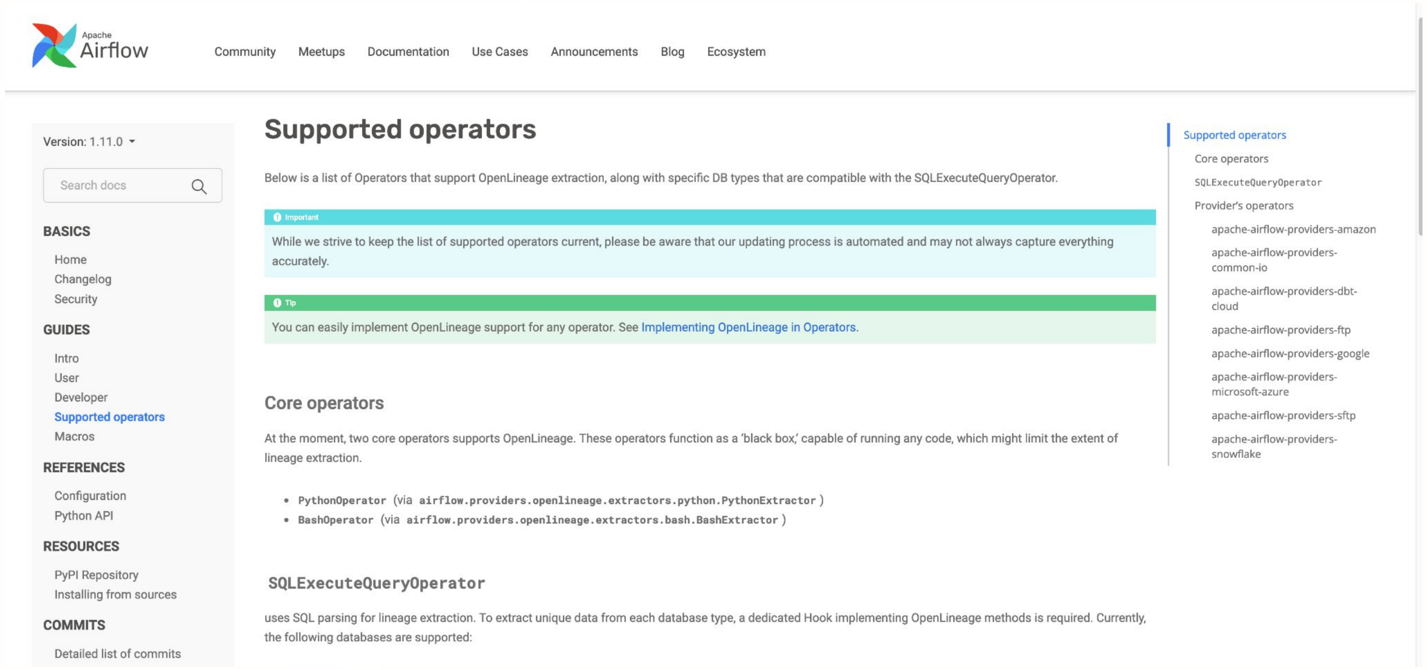
How does it work

- OpenLineage Provider has listener that gets called by Airflow by Listener API
    - On worker, before task runs, and after task completes or fails
- Checks if Operator has implemented lineage methods, calls them
- Enrich data returned from Operator with common Airflow facets
- More on that in last year's presentation https://youtu.be/SZBVgREqets



Supported Operators

https://airflow.apache.org/docs/apache-airflow-providers-openlineage/stable/supported_classes.html
Caveat - does not make sense for OpenLineage to cover all Operators



Supported operators

Below is a list of Operators that support OpenLineage extraction, along with specific DB types that are compatible with the SQLExecuteQueryOperator.

Important

While we strive to keep the list of supported operators current, please be aware that our updating process is automated and may not always capture everything accurately.

Tip

You can easily implement OpenLineage support for any operator. See Implementing OpenLineage in Operators.

Core operators

At the moment, two core operators supports OpenLineage. These operators function as a 'black box,' capable of running any code, which might limit the extent of lineage extraction.

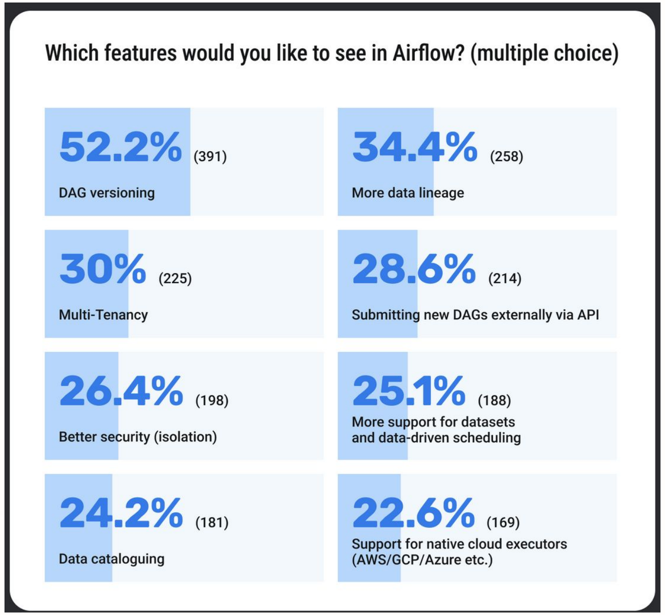
- `PythonOperator` (via `airflow.providers.openlineage.extractors.python.PythonExtractor`)
- `BashOperator` (via `airflow.providers.openlineage.extractors.bash.BashExtractor`)

`SQLExecuteQueryOperator`

uses SQL parsing for lineage extraction. To extract unique data from each database type, a dedicated Hook implementing OpenLineage methods is required. Currently, the following databases are supported:

Is this enough?

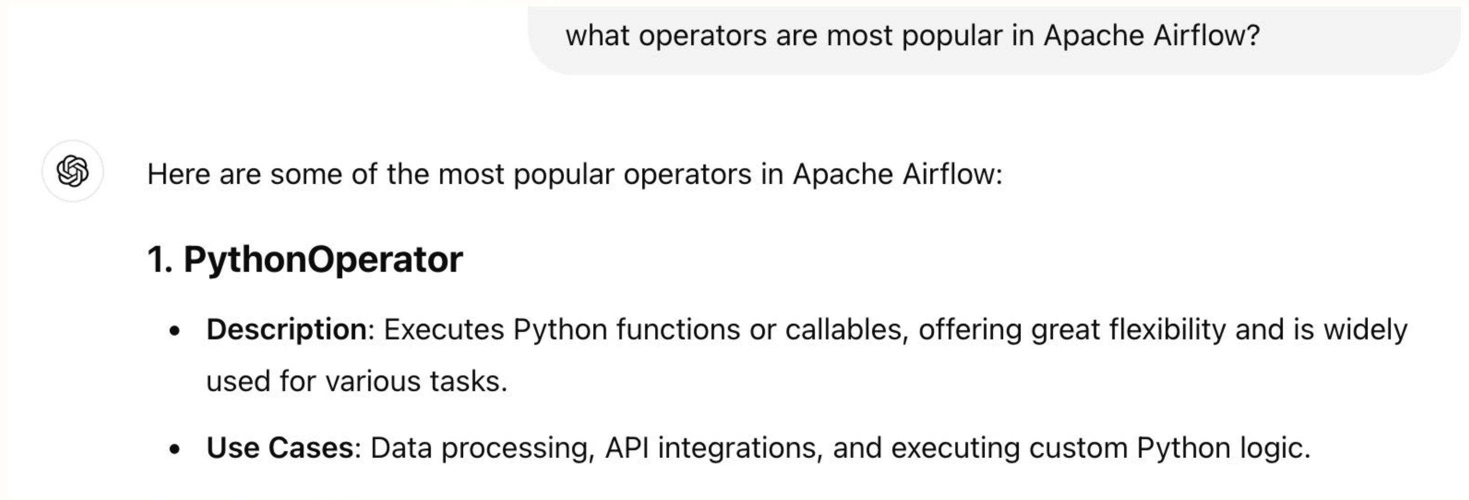
Airflow Survey 2023 results



Which features would you like to see in Airflow? (multiple choice)

Percentage	Count	Feature
52.2%	(391)	DAG versioning
34.4%	(258)	More data lineage
30%	(225)	Multi-Tenancy
28.6%	(214)	Submitting new DAGs externally via API
26.4%	(198)	Better security (isolation)
25.1%	(188)	More support for datasets and data-driven scheduling
24.2%	(181)	Data cataloguing
22.6%	(169)	Support for native cloud executors (AWS/GCP/Azure etc.)

How people use Airflow?



what operators are most popular in Apache Airflow?

Here are some of the most popular operators in Apache Airflow:

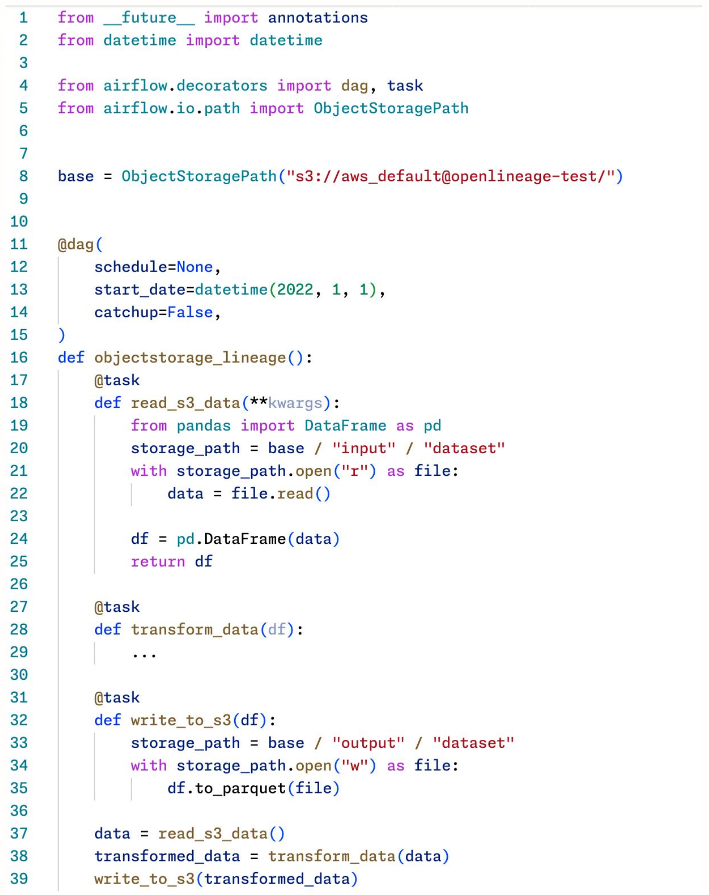
1. PythonOperator

- **Description**: Executes Python functions or callables, offering great flexibility and is widely used for various tasks.
- **Use Cases**: Data processing, API integrations, and executing custom Python logic.

TaskFlow API

```python
from airflow.decorators import dag, task
from airflow.utils.dates import days_ago
from airflow.providers.amazon.aws.hooks.s3 import S3Hook

@dag(schedule_interval=None, start_date=days_ago(1), catchup=False)
def s3_data_processing_dag():

    @task
    def read_data_from_s3(bucket_name: str, key: str) -> str:
        s3_hook = S3Hook(aws_conn_id='my_aws_conn')
        data = s3_hook.read_key(key=key, bucket_name=bucket_name)
        return data

    @task
    def process_data(data: str) -> str:
        processed_data = data.upper()
        return processed_data

    @task
    def write_data_to_s3(bucket_name: str, key: str, data: str):
        s3_hook = S3Hook(aws_conn_id='my_aws_conn')
        s3_hook.load_string(string_data=data, key=key, bucket_name=bucket_name, replace=True)

    raw_data = read_data_from_s3(bucket_name='my_bucket', key='input_data.txt')
    processed_data = process_data(raw_data)
    write_data_to_s3(bucket_name='my_bucket', key='output_data.txt', data=processed_data)
```

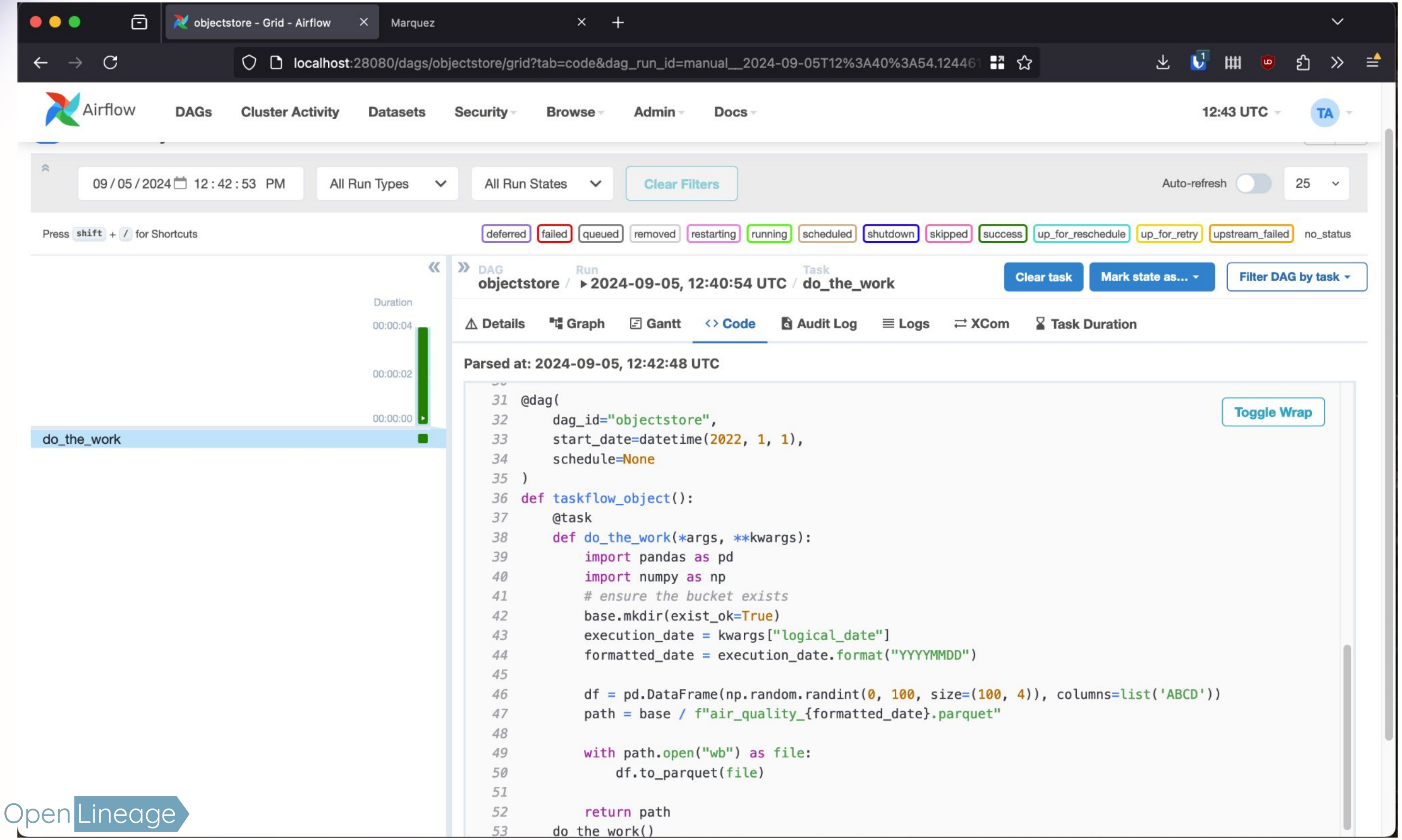
Object Storage



```python
from __future__ import annotations
from datetime import datetime

from airflow.decorators import dag, task
from airflow.io.path import ObjectStoragePath


base = ObjectStoragePath("s3://aws_default@openlineage-test/")


@dag(
    schedule=None,
    start_date=datetime(2022, 1, 1),
    catchup=False,
)
def objectstorage_lineage():
    @task
    def read_s3_data(**kwargs):
        from pandas import DataFrame as pd
        storage_path = base / "input" / "dataset"
        with storage_path.open("r") as file:
            data = file.read()

        df = pd.DataFrame(data)
        return df

    @task
    def transform_data(df):
        ...

    @task
    def write_to_s3(df):
        storage_path = base / "output" / "dataset"
        with storage_path.open("w") as file:
            df.to_parquet(file)

    data = read_s3_data()
    transformed_data = transform_data(data)
    write_to_s3(transformed_data)
```

Let’s look at another DAG

Airflow 2.9.3


objectstore - Grid - Airflow
Marquez
localhost:28080/dags/objectstore/grid?tab=code&dag_run_id=manual__2024-09-05T12%3A40%3A54.12446
Airflow
DAGs
Cluster Activity
Datasets
Security
Browse
Admin
Docs
12:43 UTC
TA
09 / 05 / 2024 12 : 42 : 53 PM
All Run Types
All Run States
Clear Filters
Auto-refresh
25
Press shift + / for Shortcuts
deferred failed queued removed restarting running scheduled shutdown skipped success up_for_reschedule up_for_retry upstream_failed no_status
Duration
00:00:04
00:00:02
00:00:00
do_the_work
DAG objectstore
Run 2024-09-05, 12:40:54 UTC
Task do_the_work
Clear task
Mark state as...
Filter DAG by task
Details
Graph
Gantt
Code
Audit Log
Logs
XCom
Task Duration
Toggle Wrap

Parsed at: 2024-09-05, 12:42:48 UTC

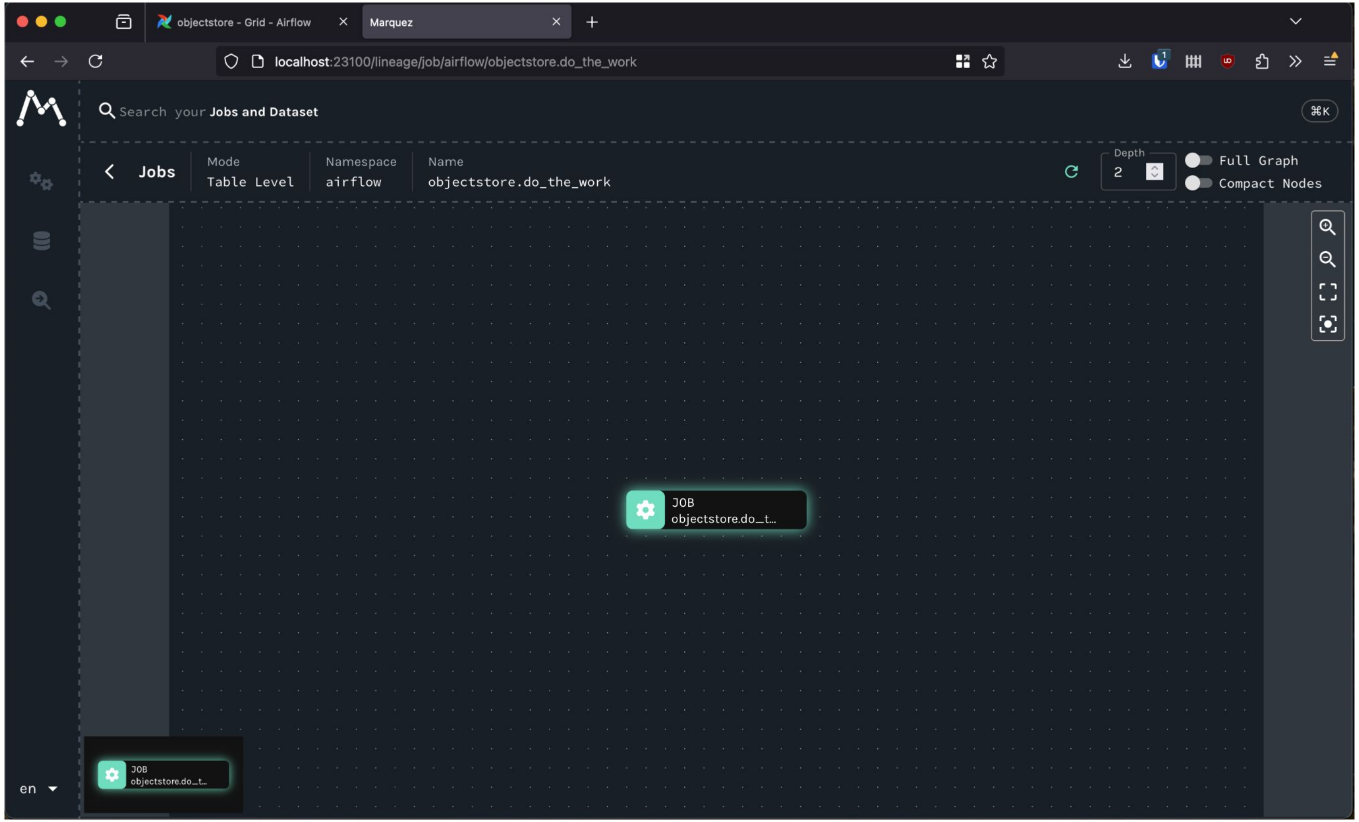
```python
@dag(
    dag_id="objectstore",
    start_date=datetime(2022, 1, 1),
    schedule=None
)
def taskflow_object():
    @task
    def do_the_work(*args, **kwargs):
        import pandas as pd
        import numpy as np
        # ensure the bucket exists
        base.mkdir(exist_ok=True)
        execution_date = kwargs["logical_date"]
        formatted_date = execution_date.format("YYYYMMDD")

        df = pd.DataFrame(np.random.randint(0, 100, size=(100, 4)), columns=list('ABCD'))
        path = base / f"air_quality_{formatted_date}.parquet"

        with path.open("wb") as file:
            df.to_parquet(file)

        return path
    do_the_work()
```

OpenLineage

ASTRONOMER

Let's look at another DAG

Airflow 2.9.3



objectstore - Grid - Airflow

Marquez

localhost:23100/lineage/job/airflow/objectstore.do_the_work

Search your Jobs and Dataset

Jobs

Mode: Table Level

Namespace: airflow

Name: objectstore.do_the_work

Depth: 2

Full Graph

Compact Nodes

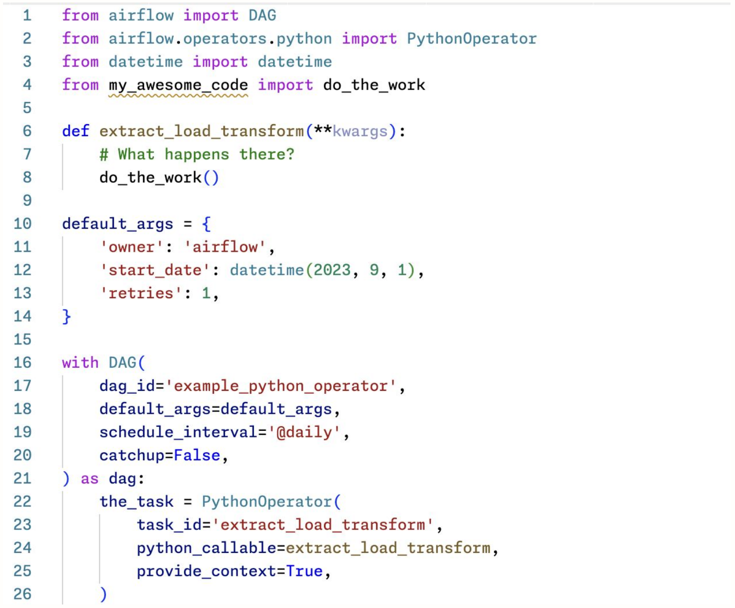
JOB objectstore.do_t...

OpenLineage

ASTRONOMER

The problem: arbitrary code

- People love writing their own code!
- Non-code operators: we generally know what they do (*)
- Code operators: can be anything!
- Is there anything we can do?



```python
from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime
from my_awesome_code import do_the_work

def extract_load_transform(**kwargs):
    # What happens there?
    do_the_work()

default_args = {
    'owner': 'airflow',
    'start_date': datetime(2023, 9, 1),
    'retries': 1,
}

with DAG(
    dag_id='example_python_operator',
    default_args=default_args,
    schedule_interval='@daily',
    catchup=False,
) as dag:
    the_task = PythonOperator(
        task_id='extract_load_transform',
        python_callable=extract_load_transform,
        provide_context=True,
    )
```

Instrument the hooks

What are hooks?

- Hooks - a mechanism for communicating with external systems
- Operations on hooks closely resemble those exposed by external systems
- Ex. copy file between object storage buckets, execute SQL on some database, or even create a Kubernetes job

```python
def copy(
    self,
    source_bucket: str,
    source_object: str,
    destination_bucket: str | None = None,
    destination_object: str | None = None,
) -> None:
    """
    Copy an object from a bucket to another, with renaming if requested.

    destination_bucket or destination_object can be omitted, in which case
    source bucket/object is used, but not both.

    :param source_bucket: The bucket of the object to copy from.
    :param source_object: The object to copy.
    :param destination_bucket: The destination of the object to copied to.
        Can be omitted; then the same bucket is used.
    :param destination_object: The (renamed) path of the object if given.
        Can be omitted; then the same name is used.
    """
    destination_bucket = destination_bucket or source_bucket
    destination_object = destination_object or source_object

    if source_bucket == destination_bucket and source_object == destination_object:
        raise ValueError(
            f"Either source/destination bucket or source/destination object must be different, "
            f"not both the same: bucket={source_bucket}, object={source_object}"
        )
    if not source_bucket or not source_object:
        raise ValueError("source_bucket and source_object cannot be empty.")

    client = self.get_conn()
    source_bucket = client.bucket(source_bucket)
    source_object = source_bucket.blob(source_object)  # type: ignore[attr-defined]
    destination_bucket = client.bucket(destination_bucket)
    destination_object = source_bucket.copy_blob(  # type: ignore[attr-defined]
        blob=source_object, destination_bucket=destination_bucket, new_name=destination_object
    )

    self.log.info(
        "Object %s in bucket %s copied to object %s in bucket %s",
        source_object.name,  # type: ignore[attr-defined]
        source_bucket.name,  # type: ignore[attr-defined]
        destination_object.name,  # type: ignore[union-attr]
        destination_bucket.name,  # type: ignore[union-attr]
    )
```

How to instrument hooks?

- We can't copy the "pull" approach we have for Operators
- One method vs many methods
- We can "push" it from the instrumented method
- Where to?

```python
def copy(
    self,
    source_bucket: str,
    source_object: str,
    destination_bucket: str | None = None,
    destination_object: str | None = None,
) -> None:
    """
    Copy an object from a bucket to another, with renaming if requested.

    destination_bucket or destination_object can be omitted, in which case
    source bucket/object is used, but not both.

    :param source_bucket: The bucket of the object to copy from.
    :param source_object: The object to copy.
    :param destination_bucket: The destination of the object to copied to.
        Can be omitted; then the same bucket is used.
    :param destination_object: The (renamed) path of the object if given.
        Can be omitted; then the same name is used.
    """
    destination_bucket = destination_bucket or source_bucket
    destination_object = destination_object or source_object

    if source_bucket == destination_bucket and source_object == destination_object:
        raise ValueError(
            f"Either source/destination bucket or source/destination object must be different, "
            f"not both the same: bucket={source_bucket}, object={source_object}"
        )
    if not source_bucket or not source_object:
        raise ValueError("source_bucket and source_object cannot be empty.")

    client = self.get_conn()
    source_bucket = client.bucket(source_bucket)
    source_object = source_bucket.blob(source_object)  # type: ignore[attr-defined]
    destination_bucket = client.bucket(destination_bucket)
    destination_object = source_bucket.copy_blob(  # type: ignore[attr-defined]
        blob=source_object, destination_bucket=destination_bucket, new_name=destination_object
    )

    self.log.info(
        "Object %s in bucket %s copied to object %s in bucket %s",
        source_object.name,  # type: ignore[attr-defined]
        source_bucket.name,  # type: ignore[attr-defined]
        destination_object.name,  # type: ignore[union-attr]
        destination_bucket.name,  # type: ignore[union-attr]
    )
```

What does the instrumentation look like?

- Added HookLineageCollector on Worker with .add_input_dataset and .add_output_dataset methods
- Those methods accept dataset_kwargs that are used to construct the compliant dataset later
- Methods in hooks that modify datasets call those methods to register dataset changes
- Data is then deduplicated - we don’t want to see hundreds of writes to same dataset

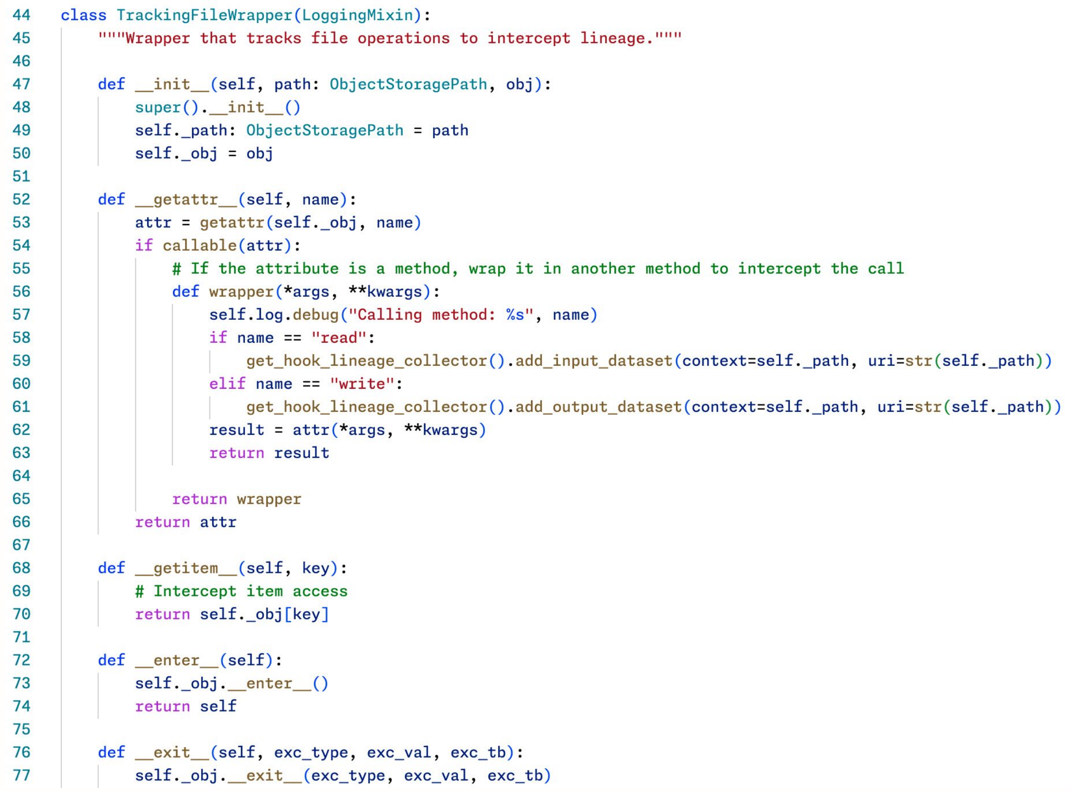
```python
@unify_bucket_name_and_key
@provide_bucket_name
def load_file(
    self,
    filename: Path | str,
    key: str,
    bucket_name: str | None = None,
    replace: bool = False,
    encrypt: bool = False,
    gzip: bool = False,
    acl_policy: str | None = None,
) -> None:
    filename = str(filename)
    if not replace and self.check_for_key(key, bucket_name):
        raise ValueError(f"The key {key} already exists.")

    extra_args = self.extra_args
    if encrypt:
        extra_args["ServerSideEncryption"] = "AES256"
    if gzip:
        with open(filename, "rb") as f_in:
            filename_gz = f"{f_in.name}.gz"
            with gz.open(filename_gz, "wb") as f_out:
                shutil.copyfileobj(f_in, f_out)
                filename = filename_gz
    if acl_policy:
        extra_args["ACL"] = acl_policy

    client = self.get_conn()
    client.upload_file(filename, bucket_name, key, ExtraArgs=extra_args, Config=self.transfer_conf
    get_hook_lineage_collector().add_input_dataset(
        context=self, scheme="file", dataset_kwargs={"path": filename}
    )
    get_hook_lineage_collector().add_output_dataset(
        context=self, scheme="s3", dataset_kwargs={"bucket": bucket_name, "key": key}
    )
```

OpenLineage

ASTRONOMER

What does the instrumentation look like?

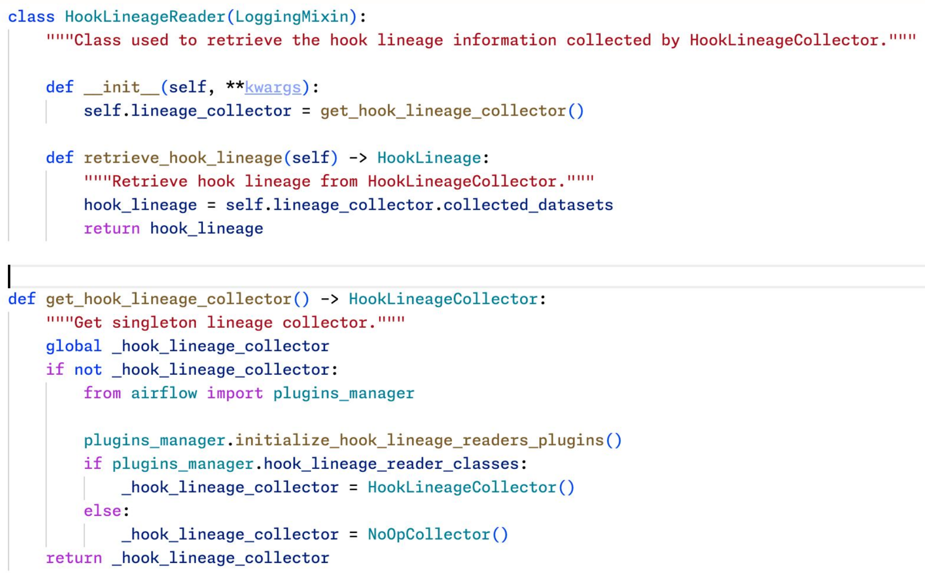
- Object Storage: track reads and writes across file-like objects
- No additional instrumentation needed for different types of paths



```python
class TrackingFileWrapper(LoggingMixin):
    """Wrapper that tracks file operations to intercept lineage."""

    def __init__(self, path: ObjectStoragePath, obj):
        super().__init__()
        self._path: ObjectStoragePath = path
        self._obj = obj

    def __getattr__(self, name):
        attr = getattr(self._obj, name)
        if callable(attr):
            # If the attribute is a method, wrap it in another method to intercept the call
            def wrapper(*args, **kwargs):
                self.log.debug("Calling method: %s", name)
                if name == "read":
                    get_hook_lineage_collector().add_input_dataset(context=self._path, uri=str(self._path))
                elif name == "write":
                    get_hook_lineage_collector().add_output_dataset(context=self._path, uri=str(self._path))
                result = attr(*args, **kwargs)
                return result

            return wrapper
        return attr

    def __getitem__(self, key):
        # Intercept item access
        return self._obj[key]

    def __enter__(self):
        self._obj.__enter__()
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        self._obj.__exit__(exc_type, exc_val, exc_tb)
```

Works only if you use it

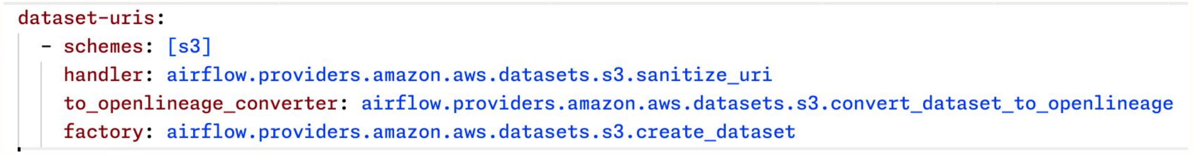
- Not OpenLineage specific - you can write your own plugin, register HookLineageReader and use the gathered hook lineage for your own purposes
- If there are no HookLineageReaders, the .add_input_datasets and .add_output_datasets just send data to /dev/null



```python
class HookLineageReader(LoggingMixin):
    """Class used to retrieve the hook lineage information collected by HookLineageCollector."""

    def __init__(self, **kwargs):
        self.lineage_collector = get_hook_lineage_collector()

    def retrieve_hook_lineage(self) -> HookLineage:
        """Retrieve hook lineage from HookLineageCollector."""
        hook_lineage = self.lineage_collector.collected_datasets
        return hook_lineage


def get_hook_lineage_collector() -> HookLineageCollector:
    """Get singleton lineage collector."""
    global _hook_lineage_collector
    if not _hook_lineage_collector:
        from airflow import plugins_manager

        plugins_manager.initialize_hook_lineage_readers_plugins()
        if plugins_manager.hook_lineage_reader_classes:
            _hook_lineage_collector = HookLineageCollector()
        else:
            _hook_lineage_collector = NoOpCollector()
    return _hook_lineage_collector
```

AIP-60 compliant datasets

- Operators don't always work on datasets they know - GCSToS3Operator
- Object Storage does not explicitly know what the file is, it operates on abstract files
- AIP-60 Dataset is an URI, OpenLineage dataset is name+namespace
- Providers know the dataset abstractions they own
- Providers can provide good factory method for AIP-60 URI, and how to translate it to OpenLineage dataset

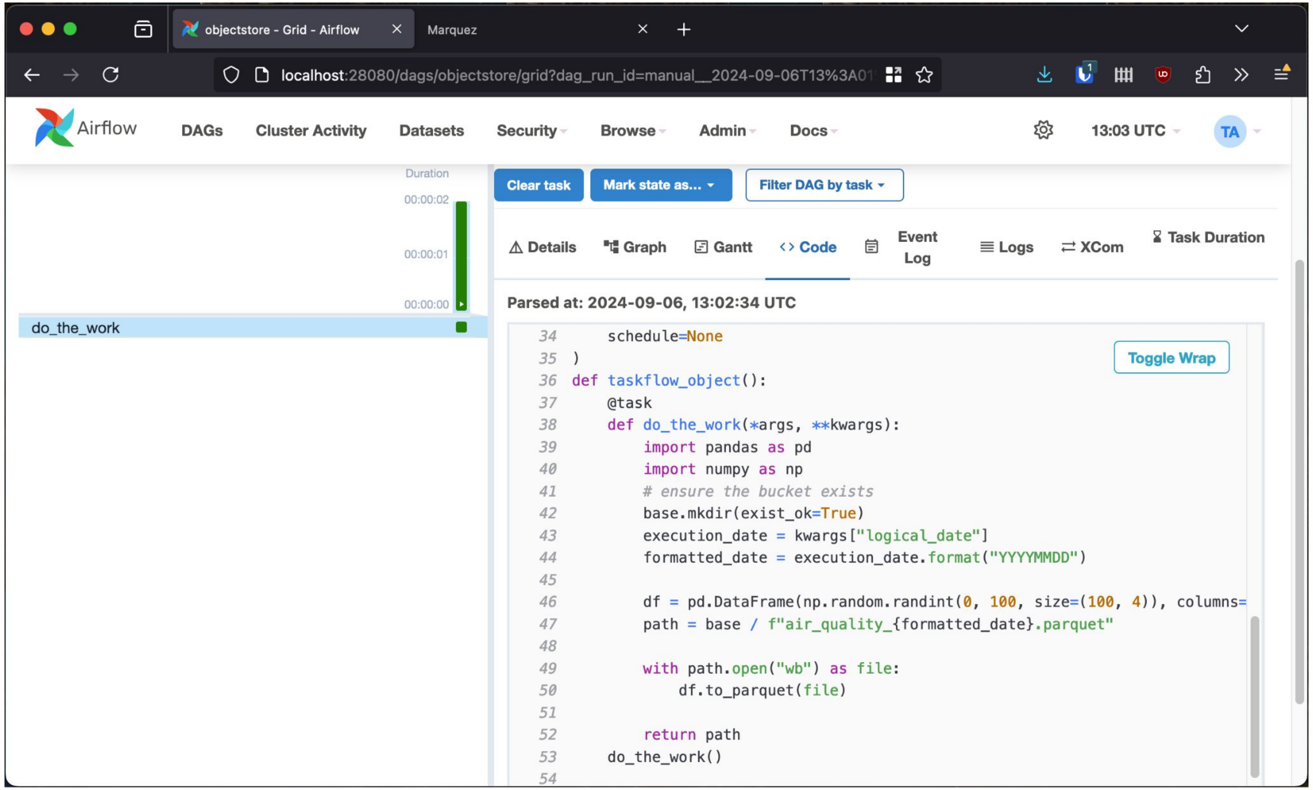


```yaml
dataset-uris:
  - schemes: [s3]
    handler: airflow.providers.amazon.aws.datasets.s3.sanitize_uri
    to_openlineage_converter: airflow.providers.amazon.aws.datasets.s3.convert_dataset_to_openlineage
    factory: airflow.providers.amazon.aws.datasets.s3.create_dataset
```

OpenLineage

ASTRONOMER

Same Object Storage DAG in Airflow 2.10



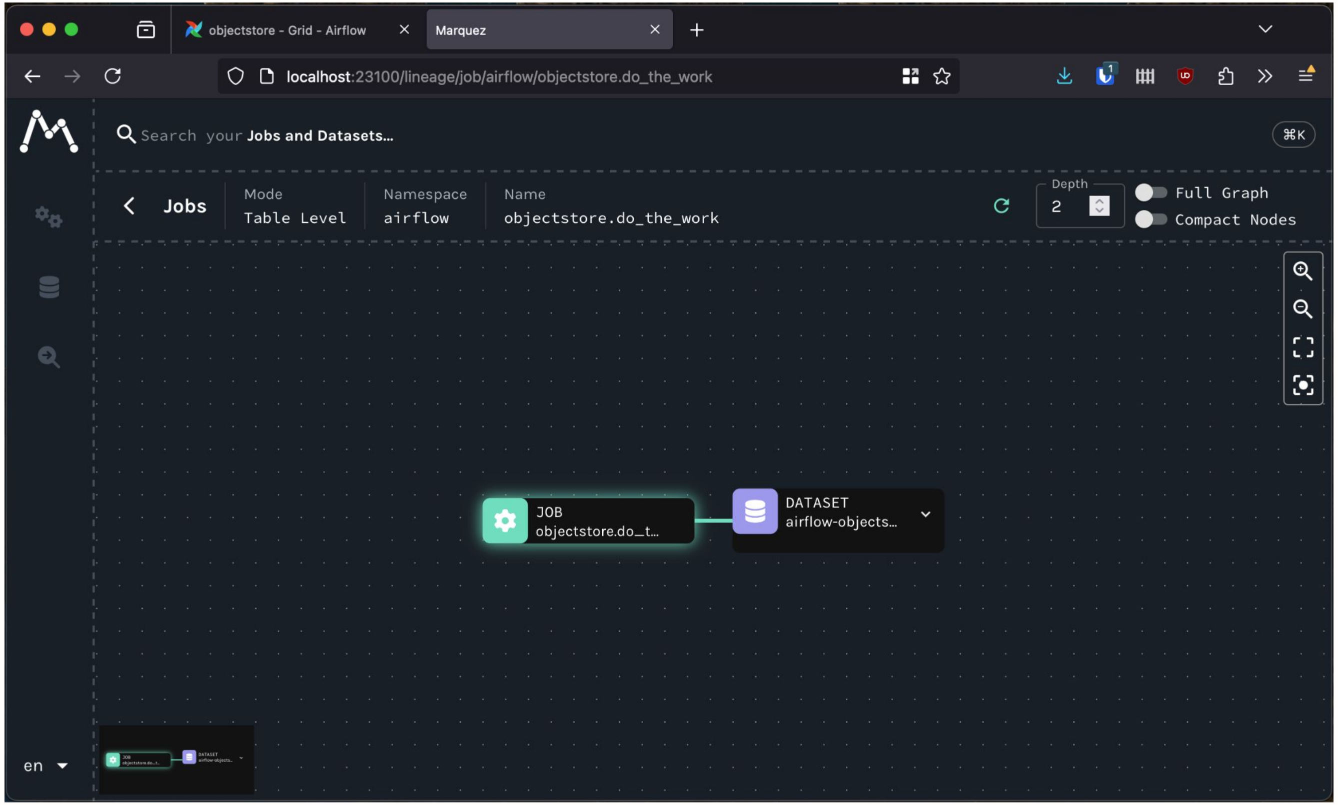
Parsed at: 2024-09-06, 13:02:34 UTC

```
    schedule=None
)
def taskflow_object():
    @task
    def do_the_work(*args, **kwargs):
        import pandas as pd
        import numpy as np
        # ensure the bucket exists
        base.mkdir(exist_ok=True)
        execution_date = kwargs["logical_date"]
        formatted_date = execution_date.format("YYYYMMDD")

        df = pd.DataFrame(np.random.randint(0, 100, size=(100, 4)), columns=
        path = base / f"air_quality_{formatted_date}.parquet"

        with path.open("wb") as file:
            df.to_parquet(file)

        return path
    do_the_work()
```

OpenLineage

ASTRONOMER

Same Object Storage DAG in Airflow 2.10


objectstore - Grid - Airflow
Marquez
localhost:23100/lineage/job/airflow/objectstore.do_the_work
Search your Jobs and Datasets...
⌘K
Jobs
Full Graph
Compact Nodes
JOB
objectstore.do_t...
DATASET
airflow-objects...
en

Mode: Table Level
Namespace: airflow
Name: objectstore.do_the_work
Depth: 2

Lineage For Python Tasks

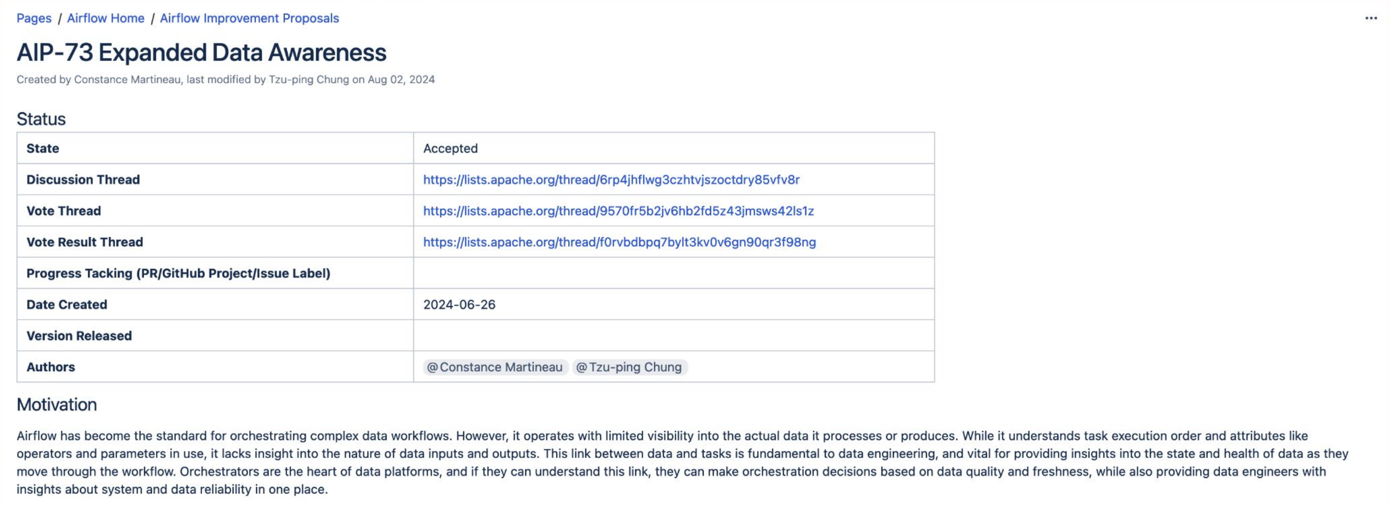
- It does not matter how you author your Python tasks, it captures hook lineage whether you use PythonOperator, TaskFlow or Custom Operators
- However, it has to be the same process as worker - python-based KubernetesOperator job won't work

Peek into the future

As in, what’s happening with OL in Airflow 3?

~~Datasets~~ Assets

- Despite all those changes, not everything can be detected.
- OpenLineage integration will be able to take advantage of that and expose annotated lineage



Pages / Airflow Home / Airflow Improvement Proposals

AIP-73 Expanded Data Awareness

Created by Constance Martineau, last modified by Tzu-ping Chung on Aug 02, 2024

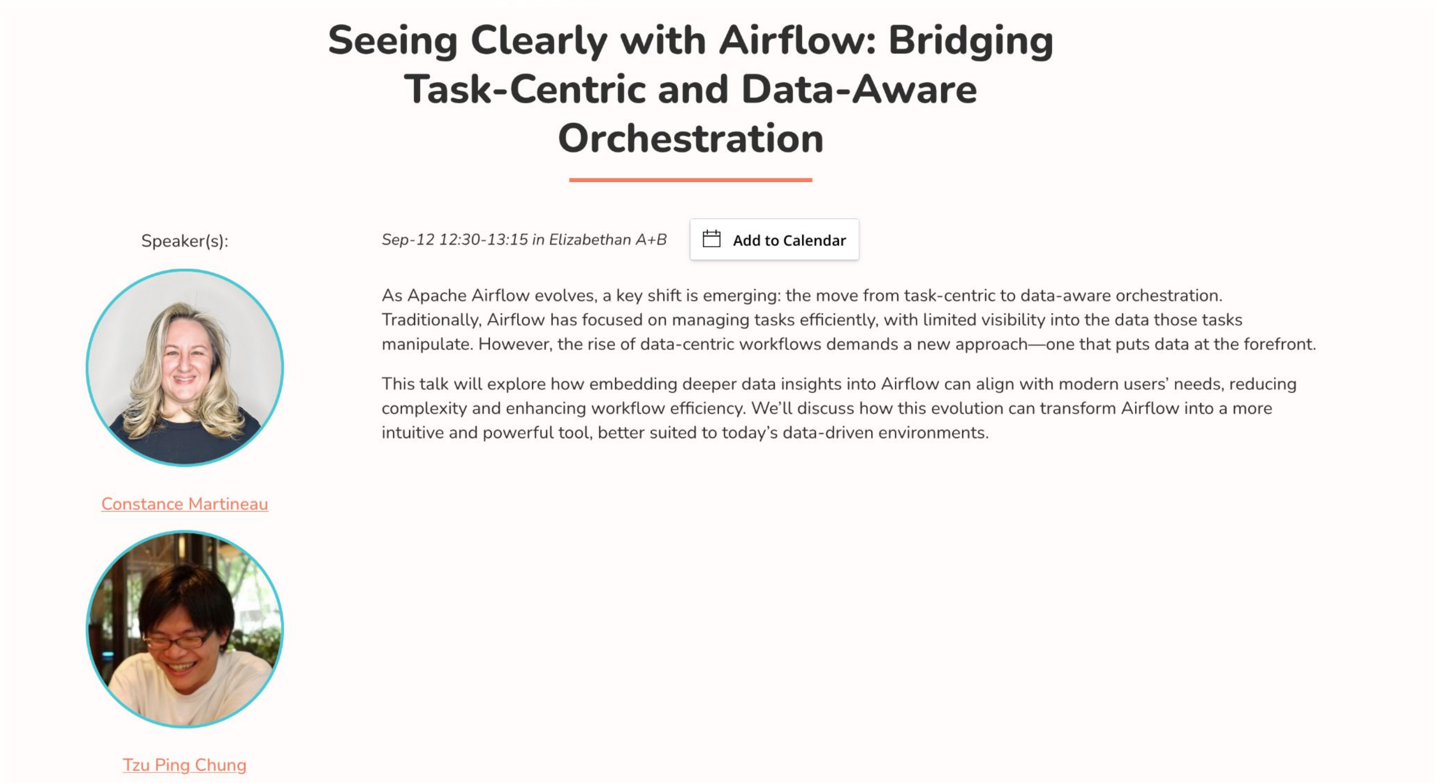
Status

State	Accepted
Discussion Thread	https://lists.apache.org/thread/6rp4jhflwg3czhtvjszoctdry85vfv8r
Vote Thread	https://lists.apache.org/thread/9570fr5b2jv6hb2fd5z43jmsws42ls1z
Vote Result Thread	https://lists.apache.org/thread/f0rvbdbpq7bylt3kv0v6gn90qr3f98ng
Progress Tacking (PR/GitHub Project/Issue Label)	
Date Created	2024-06-26
Version Released	
Authors	@Constance Martineau @Tzu-ping Chung

Motivation

Airflow has become the standard for orchestrating complex data workflows. However, it operates with limited visibility into the actual data it processes or produces. While it understands task execution order and attributes like operators and parameters in use, it lacks insight into the nature of data inputs and outputs. This link between data and tasks is fundamental to data engineering, and vital for providing insights into the state and health of data as they move through the workflow. Orchestrators are the heart of data platforms, and if they can understand this link, they can make orchestration decisions based on data quality and freshness, while also providing data engineers with insights about system and data reliability in one place.

~~Datasets~~ Assets



Seeing Clearly with Airflow: Bridging Task-Centric and Data-Aware Orchestration

Speaker(s):

Constance Martineau

Tzu Ping Chung

Sep-12 12:30-13:15 in Elizabethan A+B

As Apache Airflow evolves, a key shift is emerging: the move from task-centric to data-aware orchestration. Traditionally, Airflow has focused on managing tasks efficiently, with limited visibility into the data those tasks manipulate. However, the rise of data-centric workflows demands a new approach—one that puts data at the forefront.

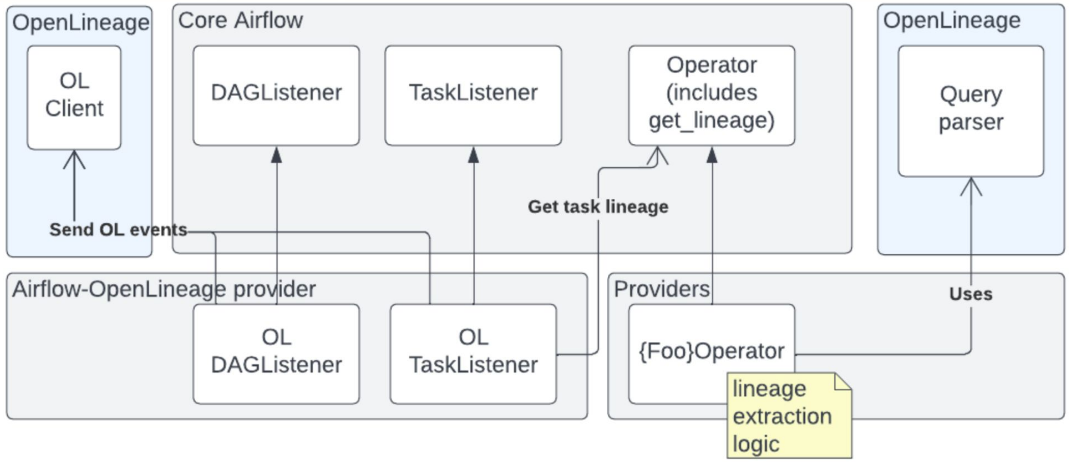
This talk will explore how embedding deeper data insights into Airflow can align with modern users' needs, reducing complexity and enhancing workflow efficiency. We'll discuss how this evolution can transform Airflow into a more intuitive and powerful tool, better suited to today's data-driven environments.

Synergy

- Building on Airflow - using Connections, Hooks, Assets, Object Storage gives you much more than using Airflow as a “dumb” scheduler

AIP-72 Task Execution Interface

- Decoupling access to database from Worker requires us to rework approach for integration
- Below diagram won't be accurate: worker after completion will send metadata using AIP-72 interface, and OL will process it asynchronously



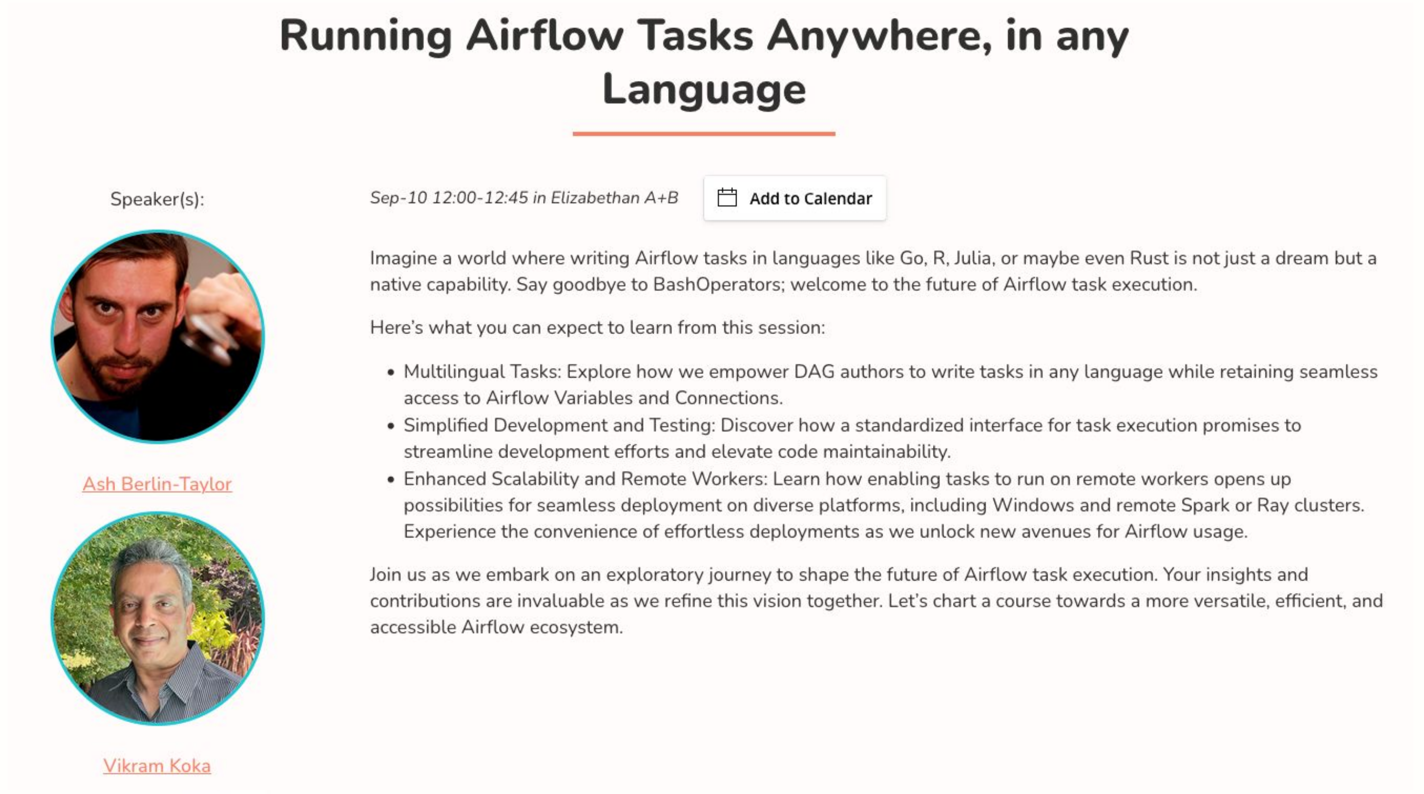
OpenLineage

ASTRONOMER

AIP-72 Task Execution Interface

- Gains:
  - Async event emission: does not block a worker slot
  - Isolated execution: won’t affect worker if some bug happen
  - Possible to enrich the data more due to asynchronous execution
- However, it’s possible that contributing OpenLineage Operator implementations will get more complex

What AIP-72 enables



Running Airflow Tasks Anywhere, in any Language

Speaker(s):

Ash Berlin-Taylor

Vikram Koka

Sep-10 12:00-12:45 in Elizabethan A+B Add to Calendar

Imagine a world where writing Airflow tasks in languages like Go, R, Julia, or maybe even Rust is not just a dream but a native capability. Say goodbye to BashOperators; welcome to the future of Airflow task execution.

Here's what you can expect to learn from this session:

- Multilingual Tasks: Explore how we empower DAG authors to write tasks in any language while retaining seamless access to Airflow Variables and Connections.
- Simplified Development and Testing: Discover how a standardized interface for task execution promises to streamline development efforts and elevate code maintainability.
- Enhanced Scalability and Remote Workers: Learn how enabling tasks to run on remote workers opens up possibilities for seamless deployment on diverse platforms, including Windows and remote Spark or Ray clusters. Experience the convenience of effortless deployments as we unlock new avenues for Airflow usage.

Join us as we embark on an exploratory journey to shape the future of Airflow task execution. Your insights and contributions are invaluable as we refine this vision together. Let's chart a course towards a more versatile, efficient, and accessible Airflow ecosystem.

Related event: OpenLineage Meetup

OpenLineage

Get more details and the signup link at **https://openlineage.io/blog**

Where

Astronomer
8 California St.

When

Thursday, 9/12
6-9 pm

Join us for in-depth talks & discussion over dinner!

Agenda

- **Unlocking Data Products with OpenLineage at Astronomer**: Julian LaNeve and Jason Ma, Astronomer.
- **OpenLineage: From Operators to Hooks** by Maciej Obuchowski, Astronomer+GetInData/Xebia.
- **Activating Operational Metadata with Airflow, Atlan and Openlineage** by Kacper Muda, GetInData/Xebia.
- **Hamilton, a Scaffold for all Your Python Platform Concerns (and a New OpenLineage Producer)** by Stefan Krawczyk
- **Lightning Talk on New Marquez Features and the Marquez Project Roadmap** by Willy Lulciuc, Marquez Lead, and Peter Hicks, Marquez Committer.

ASTRONOMER

Thank you!
Any questions?

Bonus slides

Consistent API introduction

- Adding new APIs is easy, right?
    - Add new method to Airflow Core
    - Make providers use it
    - Release it all
    - Then it breaks on older versions of Airflow

Consistent API introduction

- Adding new APIs is easy, right?
  - Add new method to Airflow Core
  - Make providers use it
  - Release it all
  - Then it breaks on older versions of Airflow
- Make provider check Airflow version, and perform an action only if version matches the requirement
  - Requires potentially a lot of duplicated work across providers

Consistent API introduction

- Adding new APIs is easy, right?
    - Add new method to Airflow Core
    - Make providers use it
    - Release it all
    - Then it breaks on older versions of Airflow
- Make provider check Airflow version, and perform an action only if version matches the requirement
    - Requires potentially a lot of duplicated work across providers
- Airflow's solution: common.compat provider
    - Introduce a method within compat provider
    - Method should perform version check, and have fallback for older version