Lineage

Note

Lineage support is very experimental and subject to change.

Airflow can help track origins of data, what happens to it and where it moves over time. This can aid having audit trails and data governance, but also debugging of data flows.

Airflow tracks data by means of inlets and outlets of the tasks. Let's work from an example and see how it works.

```
from airflow.operators.bash import BashOperator
from airflow.operators.dummy import DummyOperator
from airflow.lineage import AUTO
from airflow.lineage.entities import File
from airflow.models import DAG
from airflow.utils.dates import days_ago
from datetime import timedelta

FILE_CATEGORIES = ["CAT1", "CAT2", "CAT3"]

args = {
    'owner': 'airflow',
    'start_date': days_ago(2)
}

dag = DAG(
    dag_id='example_lineage', default_args=args,
    schedule_interval='0 0 * * *',
    dagrun_timeout=timedelta(minutes=60))

f_final = File(url="/tmp/final")
run_this_last = DummyOperator(task_id='run_this_last', dag=dag,
    inlets=AUTO,
    outlets=f_final)

f_in = File(url="/tmp/whole_directory/")
outlets = []
for file in FILE_CATEGORIES:
    f_out = File(url="/tmp/{}/{{{{ execution_date }}}}".format(file))
    outlets.append(f_out)

run_this = BashOperator(
    task_id='run_me_first', bash_command='echo 1', dag=dag,
    inlets=f_in,
    outlets=outlets
)
run_this.set_downstream(run_this_last)
```

Inlets can be a (list of) upstream task ids or statically defined as an attr annotated object as is, for example, the `File` object. Outlets can only be attr annotated object. Both are rendered at run time. However the outlets of a task in case they are inlets to another task will not be re-rendered for the downstream task.

Note

Operators can add inlets and outlets automatically if the operator supports it.

In the example DAG task `run_this` (task_id=``run_me_first``) is a BashOperator that takes 3 inlets: `CAT1`, `CAT2`, `CAT3`, that are generated from a list. Note that `execution_date` is a templated field and will be rendered when the task is running.

Note

Behind the scenes Airflow prepares the lineage metadata as part of the `pre_execute` method of a task. When the task has finished execution `post_execute` is called and lineage metadata is pushed into XCOM. Thus if you are creating your own operators that override this method make sure to decorate your method with `prepare_lineage` and `apply_lineage` respectively.

Shorthand notation

Shorthand notation is available as well, this works almost equal to unix command line pipes, inputs and outputs. Note that operator precedence still applies. Also the `|` operator will only work when the left hand side either has outlets defined (e.g. by using `add_outlets(..)` or has out of the box support of lineage `operator.supports_lineage == True`.

```
f_in > run_this | (run_this_last > outlets)
```

Previous Next

Was this entry helpful?

☆ ☆ ☆ ☆ ☆