Implementing Event-Based DAGs with Airflow
My Background

- Data Engineering consultant
- Field Engineer at Astronomer
- Lead Developer Advocate at Astronomer
- All of these roles have been (at least partially) about helping people adopt Airflow
Agenda

- The “What?” and “Why?” of event-based triggering
- Methods for implementation
  - Operators
  - Sensors
  - Deferrable Operators
  - The API
- Looking forward
What is “event-based triggering”?

Time-based scheduling

Event-based triggering
Of course! Airflow is not “fancy CRON” - it’s a fully functional orchestrator.

Its job is to manage the running of your tasks - and not all tasks need to run on a schedule.

All solutions presented today are based on fully supported Airflow features.
Why Event-Based Triggering?

Running DAGs on an **ad-hoc basis** can be helpful for many applications.

At Astronomer, we’ve seen use cases like:

- **Your website has a form page for potential customers to fill out.** After the form is submitted, you have a DAG that processes the data. You want the data ASAP, and customers don’t fill out forms on schedules.

- **Your company’s data ecosystem includes many AWS services, and Airflow for orchestration.** When a particular AWS state is reached, you run a lambda function which triggers your DAG.

- **Your team uses Airflow for ML orchestration, and one DAG generates reports based on completed models.** Model training time varies based on the data, so the reporting DAG can’t always run at the same time.
So How Can I Make My DAGs Event-Based?
TriggerDagRunOperator
TriggerDagRun:

For when the trigger event comes from another DAG in the same environment

How to Implement

```python
41 trigger_dependent_dag = TriggerDagRunOperator(
42     task_id="trigger_dependent_dag",
43     trigger_dag_id="dependent-dag",
44     wait_for_completion=True
45 )
```

Relevant Use Cases

- Cross-DAG dependencies
  - Reporting DAG should only run after data ML training DAG has completed
  - A task depends on the results of another task, but for a different execution date
Pros

- Easy to implement
- `Wait_for_completion` param gives you options for complex DAG dependencies

Cons

- Both controller and triggered DAGs must be in the same Airflow environment
Sensors
Sensors:
For when you’re not quite sure of the right time

How to Implement

```python
wait_for_dw_transformations = DbtCloudJobRunSensor(
    task_id="wait_for_dw_transformations",
    run_id=transform_salesforce_dw.output,
    poke_interval=600,
    timeout=3600,
)
```

Relevant Use Cases

- Process data only after it has arrived in S3, GCS, etc.
- Run your DAG after an external service has completed, e.g. Azure Data Factory, SageMaker, dbt Cloud job run
- When you know generally when something should run, but want to wait for the exact right time
Sensors

Pro:
- Effectively just another operator in your DAG
- Highly use-case specific

Con:
- Once the sensor’s event is received, it won’t run again for that DAG run
- Long-running sensors can incur high resource costs
- A sensor might not exist for your particular use case
Deferrable Operators
Deferrable Operators:

For when sensors are ideal but the waiting is expensive

How to Implement

```python
async_sensor = DateTimeSensorAsync(
    task_id="async_task",
    target_time=">{{ macros.datetime.utcnow() + macros.timedelta(minutes=20) }}"
)
```

Relevant Use Cases

- Whenever you would use a sensor, but want to save on compute
Deferrable Operators

Pros

- Major compute savings over traditional sensors; helps with both scalability and cost
- Only updates needed to DAGs are import paths

Cons

- Must have a deferrable operator/sensor written, unless you want to write your own
- Must have a triggerer running
- Not the best for truly ad-hoc
Airflow API
The API:

For when the trigger event is truly random

How to Implement

Relevant Use Cases

- Trigger a DAG when someone fills in a website form
- Trigger a DAG when an analyst runs a query
- Trigger a DAG when an external service completes
Pros

- Trigger any time, from anywhere. The best way to implement truly ad-hoc triggering
- Fully stable REST API with Airflow 2

Cons

- Request only triggers the DAG, it doesn’t wait for it to complete or retrieve a status (although other requests can)
- Requires configuring API authentication before using; default is to deny all requests
Looking Forward
Future Triggerer Features

Note

Currently Triggers are only used up to their first event, as they are only used for resuming deferred tasks (which happens on the first event fired). However, we plan to allow DAGs to be launched from triggers in future, which is where multi-event triggers will be more useful.
Goal:

Enable the triggering of DAGs based on dataset updates
Tell Airflow What You Think!

Take the annual Airflow User Survey
(open until June 3rd)

Thank You!