Guided Tour to DAG Authoring
Jed Cunningham
Staff Software Engineer @ astronomer.io

Committer / PMC
(After) Party Under the Stars

Wednesday, September 20th
6:30pm-10:00pm

The Sheraton Centre
123 Queen St W
(7 min walk)

RSVP Now
Let’s flow together

Workshop

Get Airflow Certified

Thursday, September 21st
12:00 pm in Trinity 4

Marc Lamberti
Head of Customer Education
at Astronomer
Agenda

- **Deferrable Tasks** (Airflow 2.2)
- **Dynamic Task Mapping** (Airflow 2.3)
- **Dynamic DAGs**
- **Datasets** (Airflow 2.4)
- **Setup / Teardown** (Airflow 2.7)
- **Params**
Deferrable Tasks
Deferrable Tasks

→ Airflow 2.2

→ Operator/Sensor that can run async
  Don’t take up a worker slot
Worker

Starts up → Defers

Finish up

Triggerer

Waiting...
Deferrable KPO

```python
KubernetesPodOperator(
    task_id="kpo",
    ...,  
    deferrable=True,
)
```
Deferrable KPO

```
$ kubectl get pods -l airflow-worker -w

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>def-kpo-8czrhyrp</td>
<td>Running</td>
</tr>
<tr>
<td>def-kpo-8czrhyrp</td>
<td>Completed</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>def-kpo-69gnklwz</td>
<td>Running</td>
</tr>
<tr>
<td>def-kpo-69gnklwz</td>
<td>Completed</td>
</tr>
</tbody>
</table>
```
Tons of support now!

Providers with deferrable support:

- AWS
- Google
- Azure
- DBT
- K8s
- More!

[operators] default_deferrable
def execute():
    self.defer(
        trigger=SomeTrigger(),
        method_name="execute_complete",
    )

def execute_complete():
    return
class SomeTrigger(BaseTrigger):

    def serialize(self):
        return ("path.to.SomeTrigger", {})

    async def run(self):
        yield TriggerEvent()
Dynamic Task Mapping
Dynamic Task Mapping

→ Airflow 2.3

→ “For loop” for your tasks
   Based on output from a previous task, or static list

→ “Reduce” tasks
   Task that operates on all results of a mapped task
Not like this:

```python
for file in {s3_bucket}:
    BashOperator({file})
```
Like this:

```python
list_filenames = S3ListOperator(...)

SomeOperator
  .partial(task_id="hello_summit")
  .expand(thing=list_filenames.output)
```
Like this:
Like this:
Reduce:

```python
doubled = times_two.expand(x=[1, 2])
sum_(doubled)
```
Reduce:

times_two [2]
- success
- @task

sum_
- success
- @task
Dynamic DAGs
Auto Registration

```python
for thing in list_of_things:
    with DAG(f"generated_dag_{thing}", ...) as dag:
        ...
        globals()[dag_id]
```
Dynamic DAGs

```python
for thing in list_of_things:
    with DAG(f"generated_dag_{thing}", ...):
        ...
```
Magic Loop in 2.4?

```python
desired_id = get_parsing_context().dag_id

for thing in list_of_things:
    dag_id = f"generated_dag_{thing}"
    if desired_id and desired_id != dag_id:
        continue

...
Dynamic DAGs

→ Positives:
  ● Easy code reuse

→ Negatives:
  ● Debugging complexity
  ● Scaling
Datasets
Datasets

→ Airflow 2.4

→ Data aware scheduling
  Schedule DAG runs based on tasking updating data
Producers: Task A
Datasets: Dataset X
Consumers: DAG A
MyOperator(
  outlets=[
    Dataset("s3://some-bucket/file.csv")
  ],
  ..., 
)
with DAG(
    schedule=Dataset("s3://some-bucket/file.csv")
),
    ..., 
): 
    ...

Next Run

On s3://some-bucket/file.csv
Dataset

dataset_simple_producer

s3://some-bucket/file.csv

datastore

dataset_simple_consumer
Producers

Task A
Task B

Datasets

Dataset X
Dataset Y

Consumers

DAG A
Pending datasets:
another-dataset
s3://some-bucket/file.csv

Click to see more details.

0 of 2 datasets updated
Datasets needed to trigger the next run for

**dataset_simple_consumer2**

1 of 2 datasets updated

<table>
<thead>
<tr>
<th>Dataset URI</th>
<th>Latest Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>another-dataset</td>
<td></td>
</tr>
<tr>
<td>s3://some-bucket/file.csv</td>
<td>2023-09-19, 02:49:41</td>
</tr>
</tbody>
</table>

Close
What can a Dataset be?
Setup / Teardown
Setup / Teardown

→ Airflow 2.7

→ “Bookend” tasks (support tasks)
  
  Cleared automatically
  
  Teardown runs if setup ran
  
  Teardown not considered for DAG run state
Setup / Teardown

create_cluster >> run_query >> delete_cluster

create_cluster >> run_query

run_query >> delete_cluster.as_teardown(setups=create_cluster)
Setup / Teardown

create_cluster
  @task

run_query
  @task

delete_cluster
  @task
Teardown always runs
DAG Run State

<table>
<thead>
<tr>
<th>Task</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>create_cluster</td>
<td>00:00:05</td>
</tr>
<tr>
<td>run_query</td>
<td></td>
</tr>
<tr>
<td>delete_cluster</td>
<td></td>
</tr>
</tbody>
</table>
Clearing

Clear and Retry

**Task:** run_query

Run: manual__2023-09-18T15:17:08.961246+00:00

Include:
- Past
- Future
- **Upstream**
- Downstream
- Recursive
- Only Failed

Affected Tasks: 3

<table>
<thead>
<tr>
<th>TASK NAME</th>
<th>MAP INDEX</th>
<th>RUN ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>create_cluster</td>
<td>-1</td>
<td>manual__2023-09-18T15:17:08.961246+00:00</td>
</tr>
<tr>
<td>run_query</td>
<td>-1</td>
<td>manual__2023-09-18T15:17:08.961246+00:00</td>
</tr>
<tr>
<td>delete_cluster</td>
<td>-1</td>
<td>manual__2023-09-18T15:17:08.961246+00:00</td>
</tr>
</tbody>
</table>
@setup
def create_cluster():
    ...

@teardown
def delete_cluster():
    ...

with create_cluster() >> delete_cluster():
    run_query()
with TaskGroup("my_group") as tg:
    s1 = s1()
    s1 >> w1() >> t1().as_teardown(setups=s1)

tg >> w2()
Params
Params

→ Provide input to runs

→ JSON Schema
with DAG(
    "some_dag",
    params={
        "rounds": Param(5, type="integer", minimum=3),
    },
):

DAG conf Parameters

**rounds**: 5

Generated Configuration JSON

[Trigger] [Cancel]
def summit(params):
    print(f"Doing {params['rounds']} rounds!")

PythonOperator(task_id="summit", python_callable=summit)
BashOperator(
    task_id="hello",
    bash_command='echo "Doing $ROUNDS rounds!"',
    env={"ROUNDS": "{{ params.rounds }}"},
)

⚠️
"branch": Param(
    default="main",
    type="string",
    title="Git Branch",
    description="The branch to deploy",
),
"environment": Param(
    enum=['dev', 'stage', 'prod'],
    default='stage'
),
Get Involved!

→ Over 2600 contributors

→ All contributions are valuable

→ Join #development-first-pr-support on Slack