Sharing Your First Messages Using XCOMs

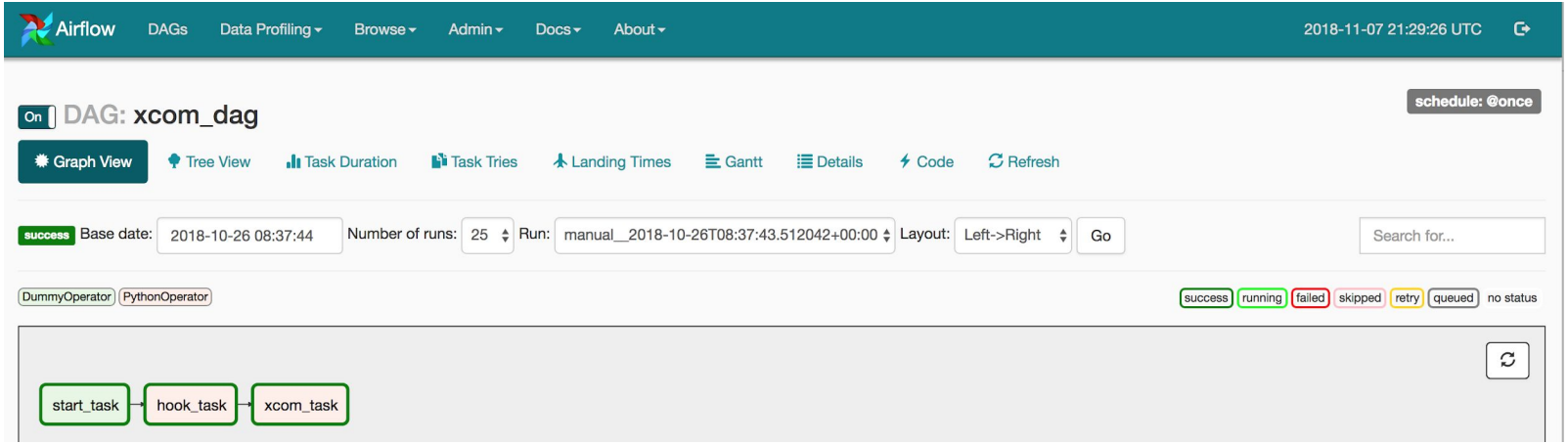
Time to practice

Instructions

- `cd ~`
- `cp ~/airflow_files/xcom_dag.py ~/airflow/dags`
- If your scheduler / webserver is running you can restart them (It's not mandatory but it's jut to save time since the schedule and webserver will be immediately aware of the new DAG).
- `vim ~/airflow/dags/xcom_dag.py` to take a look at the code

Instructions

- Trigger the DAG “xcom_dag” from Airflow UI and once it has finished executing you should see the following from the “Graph View”:



Instructions

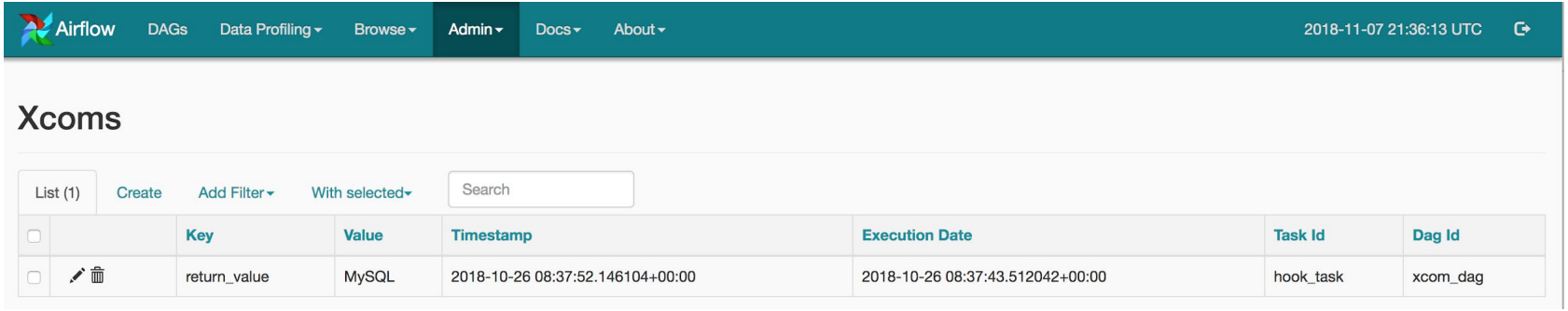
- Now click on the “xcom_task” node and “View Log”. You should see the following:

```
[2018-10-26 08:37:54,744] {models.py:1569} INFO - Executing <Task(PythonOperator): xcom_task> on 2018-10-26T08:37:43.512042+00:00
[2018-10-26 08:37:54,744] {base_task_runner.py:124} INFO - Running: ['bash', '-c', u'airflow run xcom_dag xcom_task 2018-10-26T08:
[2018-10-26 08:37:55,254] {base_task_runner.py:107} INFO - Job 31: Subtask xcom_task [2018-10-26 08:37:55,252] {settings.py:174} I
[2018-10-26 08:37:55,451] {base_task_runner.py:107} INFO - Job 31: Subtask xcom_task [2018-10-26 08:37:55,441] {__init__.py:51} IN
[2018-10-26 08:37:55,738] {base_task_runner.py:107} INFO - Job 31: Subtask xcom_task [2018-10-26 08:37:55,733] {models.py:258} INF
[2018-10-26 08:37:55,849] {base_task_runner.py:107} INFO - Job 31: Subtask xcom_task [2018-10-26 08:37:55,847] {cli.py:492} INFO -
[2018-10-26 08:37:55,991] {logging_mixin.py:95} INFO - source fetch from XCOM: MySQL
```

- The important line is where you can read “source fetch from XCOM: MySQL”. As expected, the task “xcom_task” pulled the message pushed by the task “hook_task”. Those two tasks have communicated together through XCOMs.

Instructions

- You can actually visualise the XCOMs saved into the metadata database by going to “Admin” and “XComs”. You should have the following screen:



Instructions

		Key	Value	Timestamp	Execution Date	Task Id	Dag Id
☐	✎ 🗑	return_value	MySQL	2018-10-26 08:37:52.146104+00:00	2018-10-26 08:37:43.512042+00:00	hook_task	xcom_dag

- Notice the key “return value”. In the code, we didn’t set any key to our message and we actually didn’t even use the method `xcom_push()` .Why? Because by returning a value from the `python_callable`function, the value is automatically pushed into a XCOM. That’s why we get “return_value” as key.
- Value is simply the returned value.
- Timestamp is when the XCOM has been pushed.
- Execution Date is here, equals to the execution date of the DAGRun.
- Task Id is equal to the task id of the task having pushed the XCom. Here “hook_task”.
- Finally, the Dag Id refers to the DAG having created the XCom.

Instructions

- What if rather than returning only one value, we decide to return all the values?

```
def get_activated_sources():
        request = "SELECT * FROM course.source"
        pg_hook = PostgresHook(postgre_conn_id="postgre_sql", schema="airflow_mdb")
        connection = pg_hook.get_conn()
        cursor = connection.cursor()
        cursor.execute(request)
        sources = cursor.fetchall()
        #for source in sources:
        #       if source[1]:
        #               return source[0]
        return sources
```

Instructions

- Trigger again the DAG “xcom_dag” and wait for it to finish.
- Now go back to “Admin” and “XComs” and you should have the following XCOM:

☐	✎ 🗑	return_value	('MySQL', True), ('S3', False), ('Mongo', False), ('PostgreSQL', False)	2018-10-26 09:09:08.491328+00:00	2018-10-26 09:07:56.706125+00:00	hook_task	xcom_dag

- As you can see, all records have been returned.
- Again, I strongly discourage you to return everything as we just did. XCOMs are suitable for small values to share not for large sets of data.

Instructions

- Finally, what if we want to use `xcom_push()` function?

```
def get_activated_sources(**kwargs):
        ti = kwargs['ti']
        request = "SELECT * FROM course.source"
        pg_hook = PostgresHook(postgre_conn_id="postgre_sql", schema="airflow_mdb")
        connection = pg_hook.get_conn()
        cursor = connection.cursor()
        cursor.execute(request)
        sources = cursor.fetchall()
        for source in sources:
                if source[1]:
                        ti.xcom_push(key='activated_source', value=source[0])
                        return None

def source_to_use(**kwargs):
        ti = kwargs['ti']
        source = ti.xcom_pull(key='activated_source', task_ids='hook_task')
        print("source fetch from XCOM: {}".format(source))

with DAG('xcom_dag',
        default_args=default_args,
        schedule_interval='@once') as dag:

        start_task = DummyOperator(task_id='start_task')
        hook_task = PythonOperator(task_id='hook_task', python_callable=get_activated_sources, provide_context=True)
        xcom_task = PythonOperator(task_id='xcom_task', python_callable=source_to_use, provide_context=True)
```

Instructions

- Trigger the DAG and once it has finished executing you should see the following from the log view of the task “xcom_task”:

```
[2018-10-26 09:28:58,608] {logging_mixin.py:95} INFO - source fetch from XCOM: MySQL
```

- And if you go to “Admin” and “XComs” you should have the following created XCom:

☐	✎ 🗑	activated_source	MySQL	2018-10-26 09:28:55.530778+00:00	2018-10-26 09:28:46.393198+00:00	hook_task	xcom_dag

- Notice the key value is now equal to “activated_source” which is the key we set from the xcom_push() function. As expected, the “xcom_task” pulled out the right message according to the key from the push done by the “hook_task”.

Important Notes

- In both operators in which we used the functions xcom_push() and xcom_pull(), the parameter `provide_context` has been set to `True`. When `provide_context` is set to True, Airflow will pass a set of keyword arguments that can be used in your function. Those keyword arguments are passed through `**kwargs` inside the function (More on **kwargs here). xcom_push() and xcom_pull() are called from a TaskInstance object. By using 'ti' key from **kwargs, we get the TaskInstance object representing the task running the python_callable function needed to pull or push a XCOM.
- Don't forget to clean XCOMs from the "XComs" view of Airflow UI.