Data Sharing in your DAG

It’s very common to share data between tasks. The Airflow’s XCOM is the mechanism that lets Tasks talk to each other.

Prerequisites

The DAG `extract_stars.py` under the folder `dags` should look like that:

```
from airflow import DAG
from airflow.operators.bash import BashOperator
from airflow.providers.http.operators.http import SimpleHttpOperator

from datetime import datetime

with DAG('extract_stars', schedule_interval='@daily', start_date=datetime(2022, 1, 1), cat
chup=False) as dag:

    get_date = BashOperator(
        task_id="get_date",
        bash_command="echo {{ data_interval_start }}"
    )

    query_github_stats = SimpleHttpOperator(
        task_id="query_github_stats",
        endpoint="{{ var.value.endpoint }}",
        method="GET",
        http_conn_id="github_api",
        log_response=True
    )
```

Process the Github Stars

First, add the following Python function `_print_stargazers` just before the DAG definition of `extract_stars.py`

```
import json

def _print_stargazers(github_stats: str, date: str):
  github_stats_json = json.loads(github_stats)
```

```
  airflow_stars = github_stats_json.get("stargazers_count")
  print(f"As of {date}, Apache Airflow has {airflow_stars} stars on Github!")

with DAG('extract_stars', schedule_interval='@daily', start_date=datetime(2022, 1, 1), cat
chup=False) as dag:
...
```

The function parses the JSON data that `query_github_stats` downloads and prints the number of Github stars on the standard output for a given date.

> ⚠️ Don't forget to import json

Create a new task `print_stargazers` with the PythonOperator and the following parameters:

- task_id: `print_stars`
- python_callable: `print_stargazers`
- op_args: `[]`

Go to `registry.astronomer.io` find the PythonOperator and try to implement the task.

The solution is just below

Solution

```
from airflow import DAG
from airflow.operators.bash import BashOperator
from airflow.providers.http.operators.http import SimpleHttpOperator
from airflow.operators.python import PythonOperator

import json
from datetime import datetime

def _print_stargazers(github_stats: str, date: str):
  github_stats_json = json.loads(github_stats)
  airflow_stars = github_stats_json.get("stargazers_count")
  print(f"As of {date}, Apache Airflow has {airflow_stars} stars on Github!")

with DAG('extract_stars', schedule_interval='@daily', start_date=datetime(2022, 1, 1), cat
chup=False) as dag:
```

```
    get_date = BashOperator(
        task_id="get_date",
        bash_command="echo {{ data_interval_start }}"
    )

    query_github_stats = SimpleHttpOperator(
        task_id="query_github_stats",
        endpoint="{{ var.value.endpoint }}",
        method="GET",
        http_conn_id="github_api",
        log_response=True
    )

    print_stargazers = PythonOperator(
        task_id="print_stars",
        python_callable=_print_stargazers,
        op_args=[]
    )
```

Sharing data with XCOMs

Now you have the `print_stargazers` task, you still need to provide the parameters `github_stats` and `date` that respectively come from `query_github_stats` and get_date.

First, go to `localhost:8080`, click `Admin`, `XCOMs`.

You should see something like that:

⚠️ If you don’t have those XCOMs (the value may differ), you must run your DAG first.

The date comes from the `get_date` task

The JSON data comes from the `query_github_stats` task

Both have the key `return_value` since their Operators have automatically pushed them.

By default, the BashOperator pushes an XCOM with the last line printed on the standard output

The SimpleHttpOperator pushed an XCOM with the data at the HTTP endpoint

Back to your code, you have to pull those two XCOMs in the task `print_stargazers`

In the `op_args` argument between the square brackets, add the following values:

```
print_stargazers = PythonOperator(
        task_id="print_stars",
        python_callable=_print_stargazers,
        op_args=["{{ ti.xcom_pull(task_ids='query_github_stats') }}",
                 "{{ ti.xcom_pull(task_ids='get_date') }}"]
    )
```

Notice that we mix XCOMs with templating.

That allows us to pull the XCOMs we need and pass them to the `_print_stargazers` function.

Dependencies

Since `print_stargazers` needs both tasks `get_date` and `query_github_stats` to run, you must define the dependencies accordingly.

At the end of the DAG, define the following dependencies

```
get_date >> query_github_stats >> print_stargazers
```

Save the file and check on the Airflow UI that you don’t have any errors.

Final code

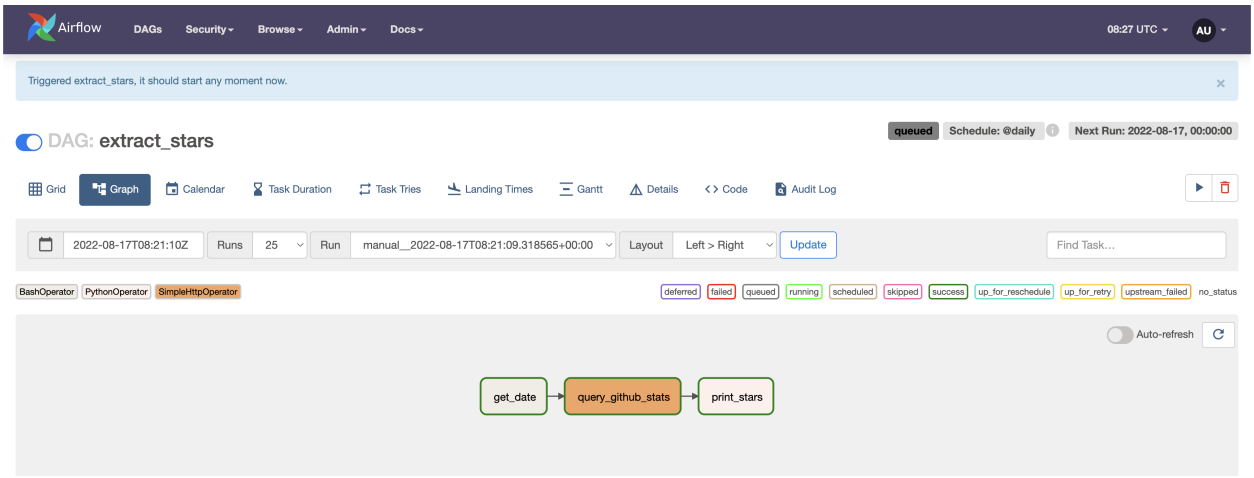
The final version of the `extract_stars` DAG:

```
from airflow import DAG
from airflow.operators.bash import BashOperator
from airflow.providers.http.operators.http import SimpleHttpOperator
from airflow.operators.python import PythonOperator

import json
from datetime import datetime

def _print_stargazers(github_stats: str, date: str):
  github_stats_json = json.loads(github_stats)
  airflow_stars = github_stats_json.get("stargazers_count")
  print(f"As of {date}, Apache Airflow has {airflow_stars} stars on Github!")

with DAG('extract_stars', schedule_interval='@daily', start_date=datetime(2022, 1, 1), cat
chup=False) as dag:

    get_date = BashOperator(
        task_id="get_date",
        bash_command="echo {{ data_interval_start }}"
    )

    query_github_stats = SimpleHttpOperator(
        task_id="query_github_stats",
        endpoint="{{ var.value.endpoint }}",
        method="GET",
        http_conn_id="github_api",
        log_response=True
    )

    print_stargazers = PythonOperator(
        task_id="print_stars",
        python_callable=_print_stargazers,
        op_args=["{{ ti.xcom_pull(task_ids='query_github_stats') }}", "{{ ti.xcom_pull(tas
k_ids='get_date') }}"]
    )

    get_date >> query_github_stats >> print_stargazers
```

Run your DAG!

On the Airflow UI, trigger the DAG and go the `Graph View`

You should end up with the following tasks:



Click on `print_stars` and `Log`

You should see something like that:

Well done! You have successfully built your first DAG on Airflow! 😎

Additional resources

Airflow's XCOMs: https://airflow.apache.org/docs/apache-airflow/stable/concepts/xcoms.html