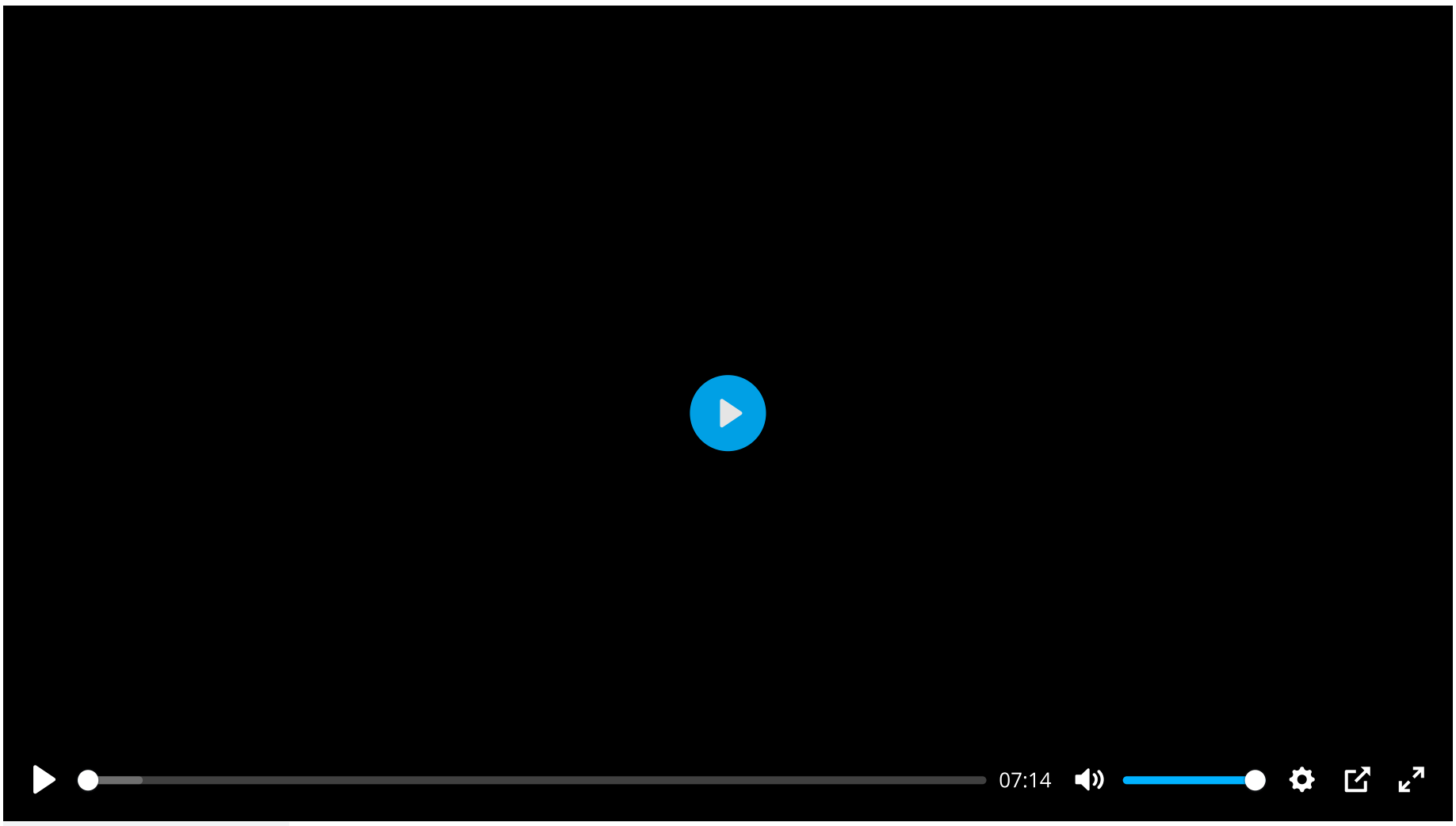
8.3 Создание и работа с DAG



Презентация к уроку 8.3.11

Текстовая расшифровка видео:

ПОЛЕЗНОЕ ПРО DAG

План:

- Провайдеры, крючки, соединения;
- PostgresHook;
- Connections;
- Providers;
- TaskGroup;
- Механизм Xcom.

Провайдеры, крючки, соединения

Крючки – это функционал, появившийся в Airflow2.0, который позволяет сделать подключение к внешнему сервису, обернув его в какую-либо функцию (например, в Python operator).

У крючков есть дополнительный функционал:

- PostgresHook позволяет работать с такими данными, как Pandas, и делать над ними какие-либо преобразования.

Крючок – это прослойка между внешним API и оператором.

Хуки документация

PostgresHook

PostgresHook умеет преобразовывать DataFrame, запускать что-либо:

- get_pandas_df;
- run;
- bulk_dump;
- bulk_load.

Connections

Connections – это подключения к внешним сервисам.

Нами уже были прописаны Connections к Postgres и к HTTP. С этими connections можно работать в коде. Так, например, можно получить какие-то параметры connections и использовать их в дальнейшей работе:

```
from airflow.hooks.base_hook import BaseHook

host = BaseHook.get_connection("postgres_default").host
pass = BaseHook.get_connection("postgres_default").password
```

Providers

Providers – это огромное количество операторов, которые позволяют работать со внешними сервисами, учитывая особенности этих сервисов.

Providers необходимы в использовании, так как это инструменты, соединяющие внешние сервисы с другими внешними сервисами.

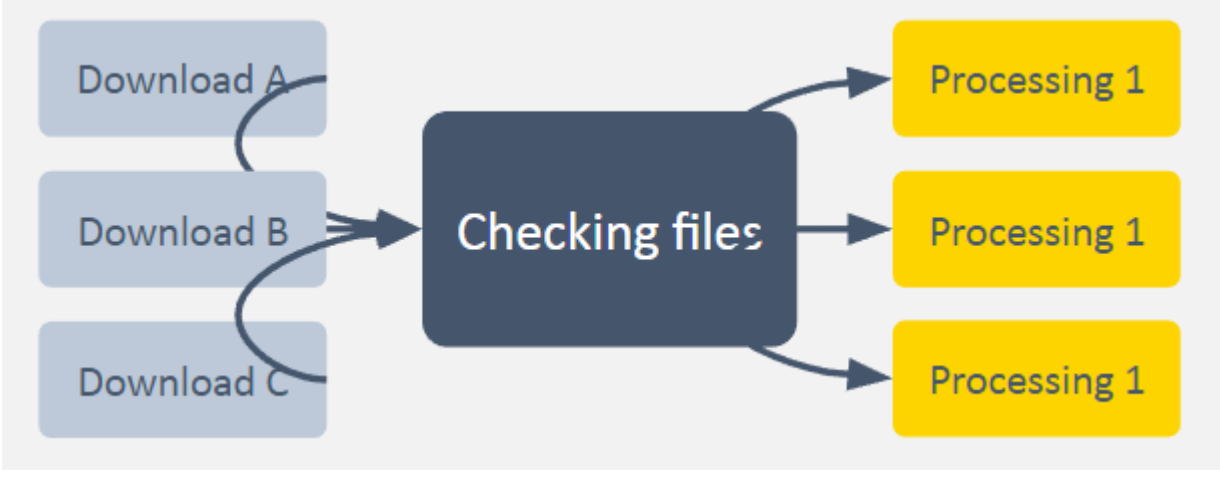
Провайдеров очень много. Это:

- Базы данных;
- Сервисы телеграмм;
- Slack;
- Интернет;
- Сокеты и т.д.

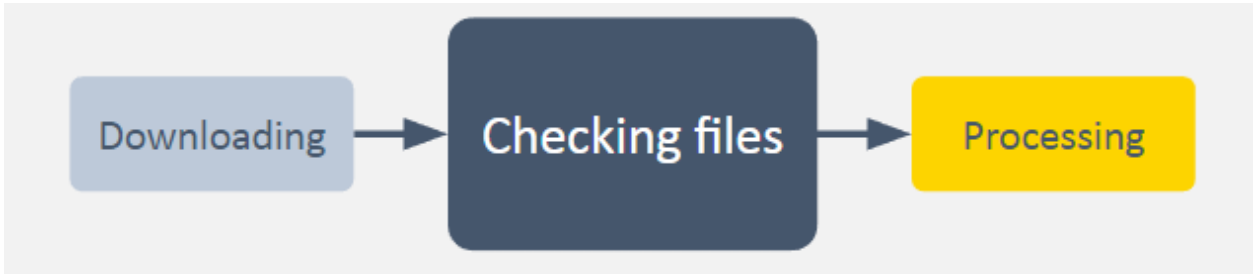
Провайдеры (документация)

TaskGroup

В некоторых ситуациях удобно сгруппировать таски по какому-то логическому смыслу, например:

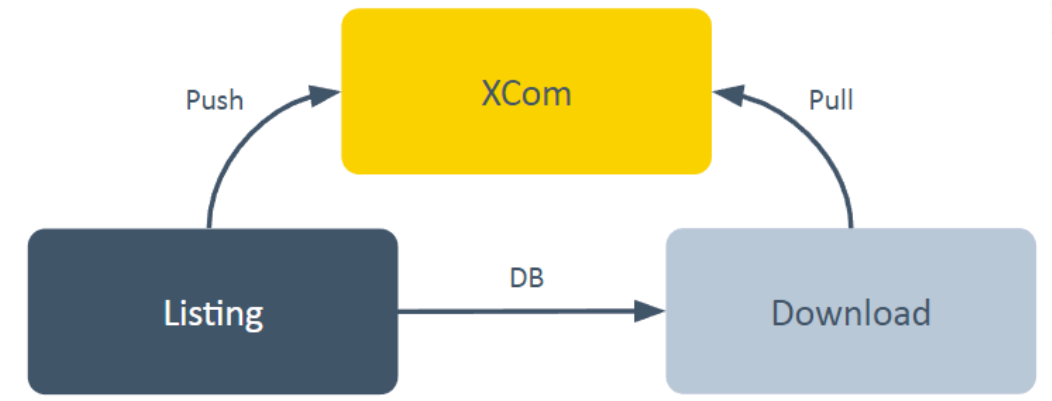


Когда таски выполняют примерно однотипные задачи, проще сгруппировать их следующим образом:



Механизм XCom

Механизм Xcom нужен для того, чтобы передавать данные между тасками. Таски могут выполняться в разных адресных пространствах, на разных машинах, поэтому нужен механизм, который будет передавать информацию между ними.

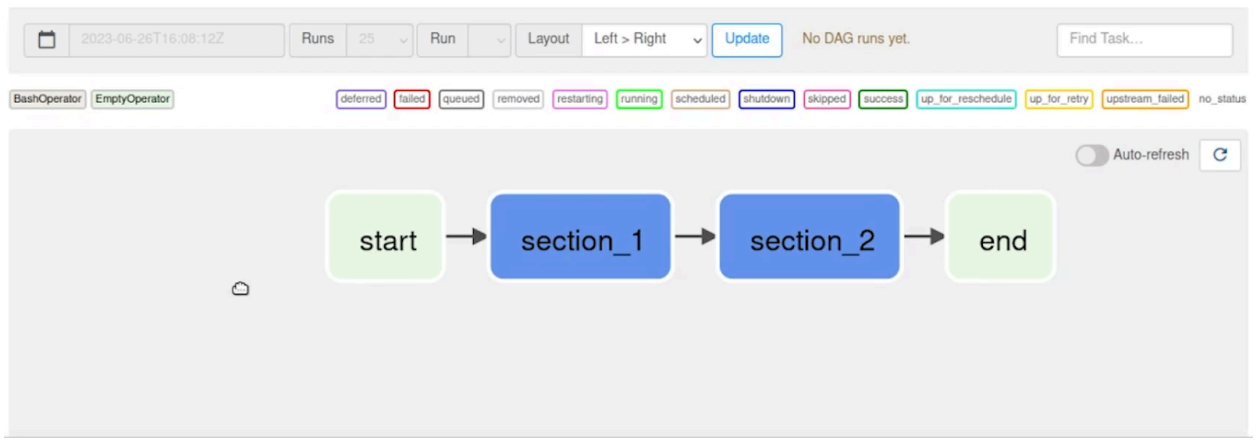


XCom’ы хранятся фактически в metastore, их размер ограничен той базой, которую вы выбрали. Например:

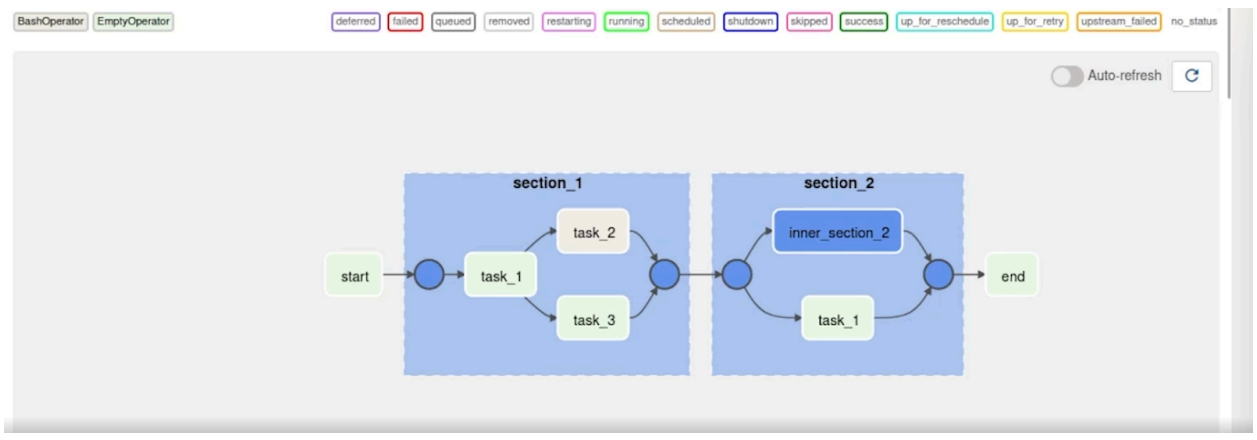
- Для My SQL – 64 Кбайт;
- Для Postgres – 1 Гб;
- Для SQLite – 2 Гб.

Рассмотрим таск-группы:

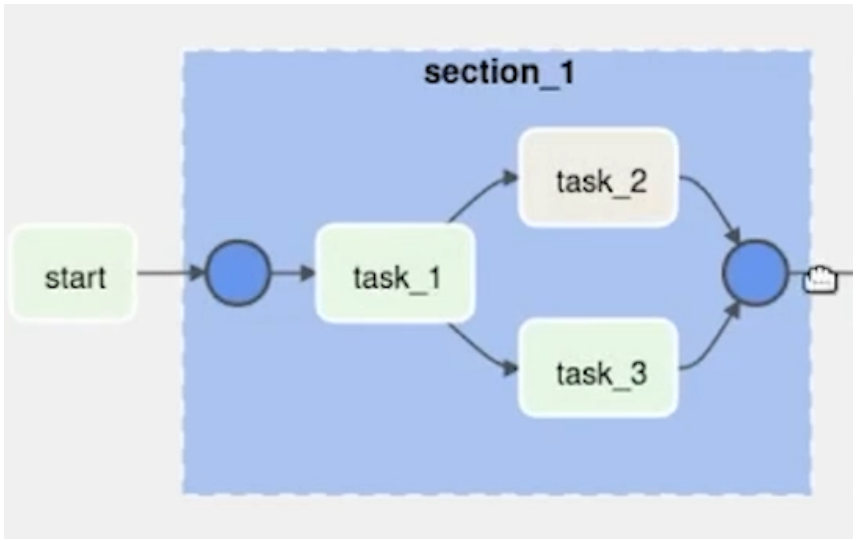
У нас есть Example Task Group в примерах DAG:



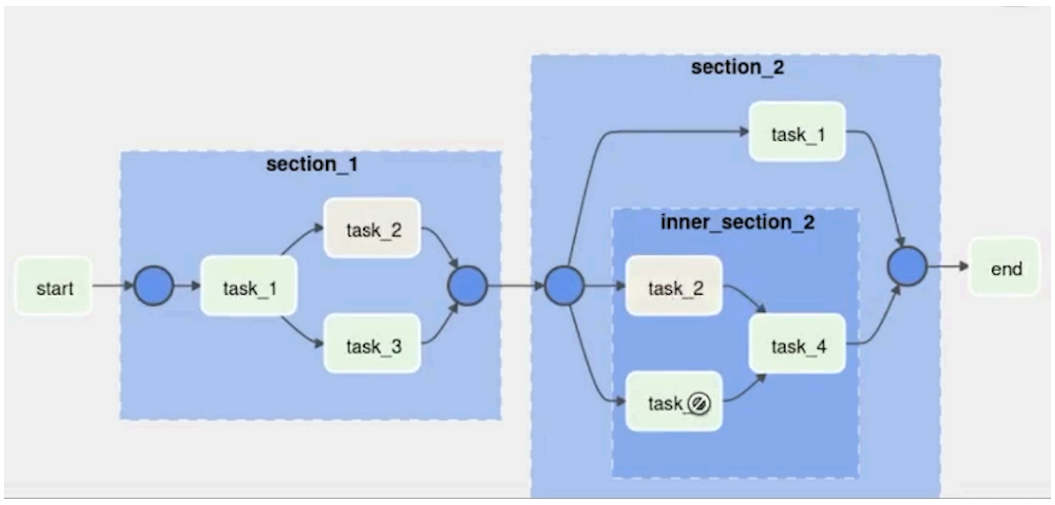
Здесь мы можем посмотреть реализацию группировки тасков:



Например, в данной группе содержится три таска, где после первого следует второй и третий:



А в этой группе вложена подгруппа тасков:



Посмотрим, как это реализовано в коде:

У нас есть оператор контекста «WithTaskGroup». Мы именуем Task for section, указываем таски, затем их зависимости. Таски, находящиеся на одном уровне, указываются в квадратных скобках:

```
# [START howto_task_group_section_1]
with TaskGroup("section_1", tooltip="Tasks for section_1") as section_1:
    task_1 = EmptyOperator(task_id="task_1")
    task_2 = BashOperator(task_id="task_2", bash_command="echo 1")
    task_3 = EmptyOperator(task_id="task_3")

    task_1 >> [task_2, task_3]
# [END howto_task_group_section_1]
```

Оператор «BitShift» указывает их последовательность.

Далее, для второй тест-группы прописывается оператор контекста, таски вне, а также таски, вложенные в подгруппу:

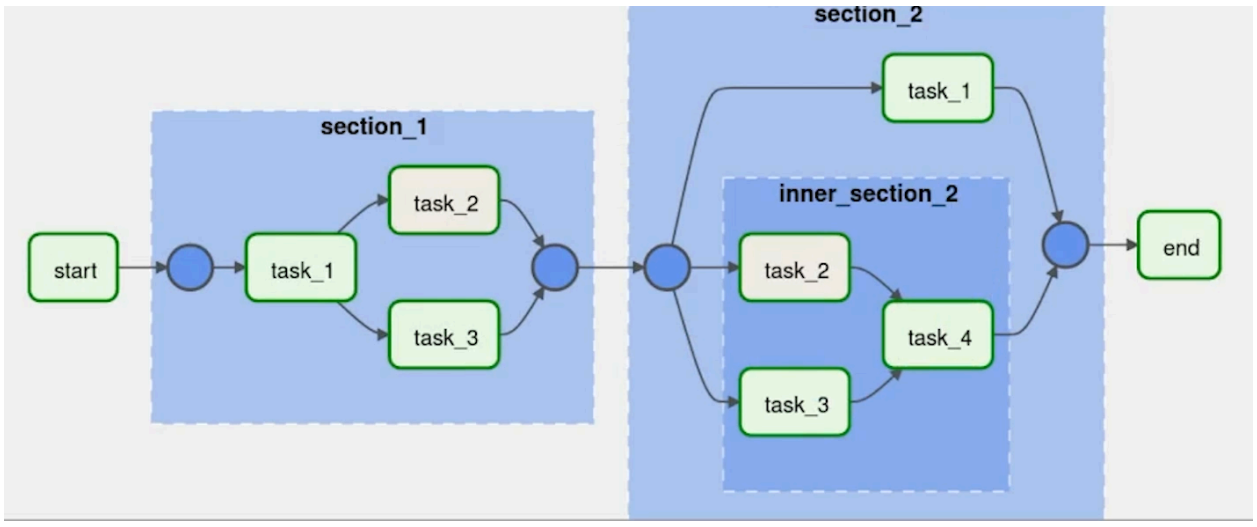
```
# [START howto_task_group_section_2]
with TaskGroup("section_2", tooltip="Tasks for section_2") as section_2:
    task_1 = EmptyOperator(task_id="task_1")
```

В данной части находится EmptyOperator. Также мы указываем связность между собой для секции: первая секция, вторая и конец:

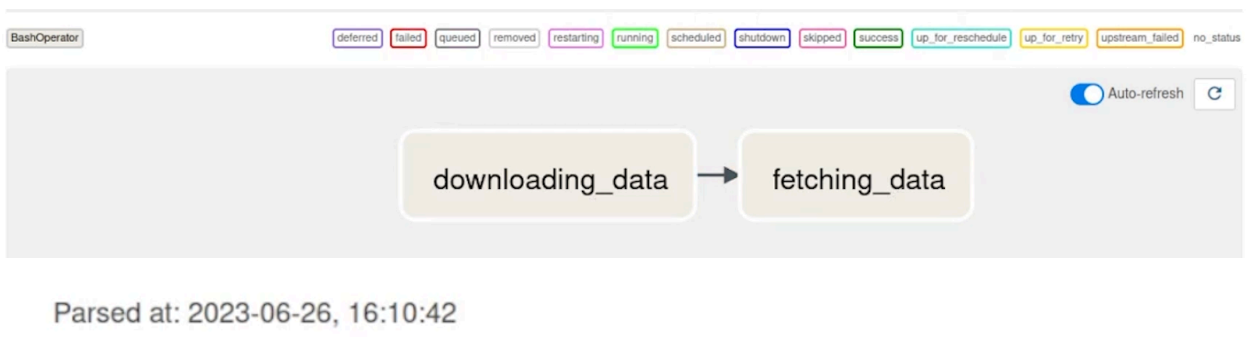
```
        # [START howto_task_group_inner_section_2]
        with TaskGroup("inner_section_2", tooltip="Tasks for inner_section2") as inner_section_2:
            task_2 = BashOperator(task_id="task_2", bash_command="echo 1")
            task_3 = EmptyOperator(task_id="task_3")
            task_4 = EmptyOperator(task_id="task_4")

            [task_2, task_3] >> task_4
        # [END howto_task_group_inner_section_2]

    # [END howto_task_group_section_2]

    end = EmptyOperator(task_id="end")

    start >> section_1 >> section_2 >> end
# [END howto_task_group]
```

Пробуем запустить:

Все таски выполнились последовательно и успешно:



Рассмотрим XCom. Механизм XCom будет рассматриваться на примере скачивания данных и их получения из XCom. Смотрим в коде:

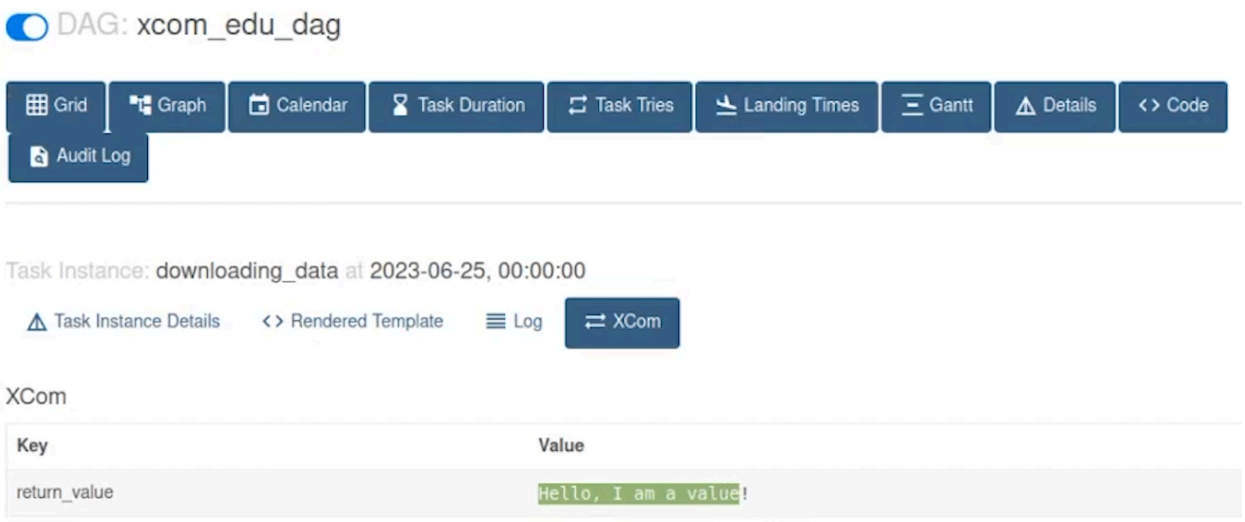



```python
from airflow import DAG
from datetime import timedelta
from airflow.utils.dates import days_ago
from airflow.operators.bash import BashOperator

dag = DAG('xcom_edu_dag',schedule_interval=timedelta(days=1), start_date=days_ago(1))
downloading_data = BashOperator(
    task_id='downloading_data',
    bash_command='echo "Hello, I am a value!"',
    do_xcom_push=True, dag=dag)
fetching_data = BashOperator(task_id='fetching_data',
bash_command="echo 'XCom fetched: {{ ti.xcom_pull(task_ids=[\'downloading_data\'])}}'",
do_xcom_push=False, dag=dag)

downloading_data >> fetching_data
```

Скачивание файлов делает XCom push. Мы видим **XCom_push=true**, это означает следующее: вывод данного оператора помещается в XCom, в данном случае, «Hello, I am a value!».

Fetching data использует XCom_push=false, но при этом делает XCom_pull. Он извлекает по Task ID какие-то данные из XCom. Task ID такой же, как у предыдущего таска.

Запустим и проверим, что находится в XCom. Для этого нажимаем на «Task», затем на «XCom»:



В логах будет написано «XComFetched» и то, какой это XCom. В нашем случае, это «Hello, I am a value!»:

```
*** Found local files:
***   * /opt/airflow/logs/dag_id=xcom_edu_dag/run_id=scheduled__2023-06-25T00:00:00+00:00/task_id=fetching_data/attempt=2.log
[2023-06-26, 16:11:27 UTC] {taskinstance.py:1103} INFO - Dependencies all met for dep_context=non-requeueable deps ti=<TaskInstance: xcom_edu_dag.fetching_dat
[2023-06-26, 16:11:27 UTC] {taskinstance.py:1103} INFO - Dependencies all met for dep_context=requeueable deps ti=<TaskInstance: xcom_edu_dag.fetching_data sc
[2023-06-26, 16:11:27 UTC] {taskinstance.py:1308} INFO - Starting attempt 2 of 2
[2023-06-26, 16:11:27 UTC] {taskinstance.py:1327} INFO - Executing <Task(BashOperator): fetching_data> on 2023-06-25 00:00:00+00:00
[2023-06-26, 16:11:27 UTC] {standard_task_runner.py:57} INFO - Started process 3575 to run task
[2023-06-26, 16:11:27 UTC] {standard_task_runner.py:84} INFO - Running: ['***', 'tasks', 'run', 'xcom_edu_dag', 'fetching_data', 'scheduled__2023-06-25T00:00:
[2023-06-26, 16:11:27 UTC] {standard_task_runner.py:85} INFO - Job 40: Subtask fetching_data
[2023-06-26, 16:11:27 UTC] {task_command.py:410} INFO - Running <TaskInstance: xcom_edu_dag.fetching_data scheduled__2023-06-25T00:00:00+00:00 [running]> on h
[2023-06-26, 16:11:27 UTC] {taskinstance.py:1547} INFO - Exporting env vars: AIRFLOW_CTX_DAG_OWNER='***' AIRFLOW_CTX_DAG_ID='xcom_edu_dag' AIRFLOW_CTX_TASK_ID
[2023-06-26, 16:11:27 UTC] {subprocess.py:63} INFO - Tmp dir root location: /tmp
[2023-06-26, 16:11:27 UTC] {subprocess.py:75} INFO - Running command: ['/bin/bash', '-c', "echo 'XCom fetched: ['Hello, I am a value!']'"]
[2023-06-26, 16:11:27 UTC] {subprocess.py:86} INFO - Output:
[2023-06-26, 16:11:27 UTC] {subprocess.py:93} INFO - XCom fetched: ['Hello, I am a value!']
[2023-06-26, 16:11:27 UTC] {subprocess.py:97} INFO - Command exited with return code 0
[2023-06-26, 16:11:27 UTC] {taskinstance.py:1350} INFO - Marking task as SUCCESS. dag_id=xcom_edu_dag, task_id=fetching_data, execution_date=20230625T000000,
[2023-06-26, 16:11:27 UTC] {local_task_job_runner.py:225} INFO - Task exited with return code 0
[2023-06-26, 16:11:27 UTC] {taskinstance.py:2653} INFO - 0 downstream tasks scheduled from follow-on schedule check
```

У нас есть первый запущенный пайплайн в инстансе Airflow и понимание того, как создавать DAG, как добавлять подключения, как писать код и т.д.

Как вам урок? 😠 😐 🔥

Изучил, далее ›