



> Конспект > 4 урок > Сложные пайплайны, часть 2

> Оглавление

> [Оглавление](#)

> [XCom](#)

> [Taskflow API](#)

> [SubDags](#)

[Что нужно знать про SubDag-и, чтобы ими пользоваться](#)

> [TaskGroup](#)

> [Динамическое создание дагов](#)

> [Best Practice](#)

> XCom

XCom (от «cross-communication») – механизм, позволяющий задачам взаимодействовать друг с другом, поскольку по умолчанию задачи полностью изолированы и могут выполняться на совершенно разных машинах.

Каждое сообщение имеет 3 параметра:

- ID дага
- ID задачи
- Ключ

Методы:

- **xcom_push** – передаёт параметры, которые хотим получить в другой задаче
- **xcom_pull** – забирает эти параметры

XCom предназначен для передачи небольших значений, но с Airflow v.2 можем написать собственный backend и сможем передавать сообщения любого размера, используя любое хранилище и любой способ сериализации.

Отправляем XCom двумя способами: явным (`explicit_push_func`) и неявным (`implicit_push_func`)

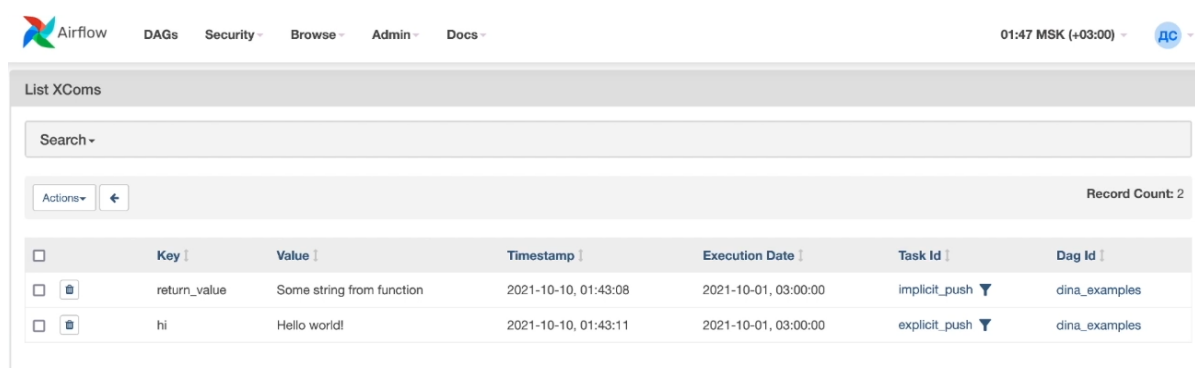
```
def explicit_push_func(**kwargs):
    kwargs['ti'].xcom_push(value='Hello world!', key='hi')

def implicit_push_func():
    return 'Some string from function'

explicit_push = PythonOperator(
    task_id='explicit_push',
    python_callable=explicit_push_func,
    provide_context=True
)

implicit_push = PythonOperator(
    task_id='implicit_push',
    python_callable=implicit_push_func
)
```

Эти строки в интерфейсе Airflow:



	Key	Value	Timestamp	Execution Date	Task Id	Dag Id
<input type="checkbox"/>	return_value	Some string from function	2021-10-10, 01:43:08	2021-10-01, 03:00:00	implicit_push	dina_examples
<input type="checkbox"/>	hi	Hello world!	2021-10-10, 01:43:11	2021-10-01, 03:00:00	explicit_push	dina_examples

Два способа прочитав из XCom-а: через метод `xcom_pull` и через шаблон `jinja`

```
def print_both_func(**kwargs):
    logging.info('-----')
    logging.info(kwargs['ti'].xcom_pull(task_ids='explicit_push', key='hi'))
    logging.info(kwargs['templates_dict']['implicit'])
    logging.info('-----')

print_both = PythonOperator(
    task_id='print_both',
    python_callable=print_both_func,
```

```

templates_dict={'implicit': '{{ ti.xcom_pull(task_ids="implicit_push") }}'},
provide_context=True
)

```

Результат:

```

*** Reading local file: /var/log/airflow/dina_examples/print_both/2021-10-08T00:00:00+00:00/3.log
[2021-10-09 22:45:25,762] {taskinstance.py:903} INFO - Dependencies all met for <TaskInstance: dina_examples.print_both 2021-10-08T00:00:00+00:00>
[2021-10-09 22:45:25,800] {taskinstance.py:903} INFO - Dependencies all met for <TaskInstance: dina_examples.print_both 2021-10-08T00:00:00+00:00>
[2021-10-09 22:45:25,800] {taskinstance.py:1095} INFO -
-----
[2021-10-09 22:45:25,801] {taskinstance.py:1096} INFO - Starting attempt 3 of 3
[2021-10-09 22:45:25,801] {taskinstance.py:1097} INFO -
-----
[2021-10-09 22:45:25,814] {taskinstance.py:1115} INFO - Executing <Task(PythonOperator): print_both> on 2021-10-08T00:00:00+00:00
[2021-10-09 22:45:25,821] {standard_task_runner.py:52} INFO - Started process 616492 to run task
[2021-10-09 22:45:25,831] {standard_task_runner.py:76} INFO - Running: ['airflow', 'tasks', 'run', 'dina_examples', 'print_both', '2021-10-08T00:00:00+00:00']
[2021-10-09 22:45:25,835] {standard_task_runner.py:77} INFO - Job 782: Subtask print_both
[2021-10-09 22:45:25,921] {logging_mixin.py:109} INFO - Running <TaskInstance: dina_examples.print_both 2021-10-08T00:00:00+00:00>
[2021-10-09 22:45:26,072] {taskinstance.py:1254} INFO - Exporting the following env vars:
AIRFLOW_CTX_DAG_OWNER=Karpov
AIRFLOW_CTX_DAG_ID=dina_examples
AIRFLOW_CTX_TASK_ID=print_both
AIRFLOW_CTX_EXECUTION_DATE=2021-10-08T00:00:00+00:00
AIRFLOW_CTX_DAG_RUN_ID=scheduled__2021-10-08T00:00:00+00:00
[2021-10-09 22:45:26,074] {dina_examples.py:110} INFO - -----
[2021-10-09 22:45:26,095] {dina_examples.py:111} INFO - Hello world!
[2021-10-09 22:45:26,096] {dina_examples.py:112} INFO - Some string from function
[2021-10-09 22:45:26,096] {dina_examples.py:113} INFO - -----
[2021-10-09 22:45:26,096] {python.py:151} INFO - Done. Returned value was: None
[2021-10-09 22:45:26,112] {taskinstance.py:1219} INFO - Marking task as SUCCESS. dag_id=dina_examples, task_id=print_both,
[2021-10-09 22:45:26,164] {local_task_job.py:151} INFO - Task exited with return code 0
[2021-10-09 22:45:26,197] {local_task_job.py:261} INFO - 0 downstream tasks scheduled from follow-on schedule check

```

> Taskflow API

Основная идея Taskflow API – каждый таск взаимодействует друг с другом напрямую. Т.е. результат работы одного таска является входными параметрами для другого таска и т.д.

```

@dag(default_args=DEFAULT_ARGS,
      schedule_interval='@daily',
      tags=['karpov'])
def dina_taskflow():

    @task
    def list_of_nums():
        return [1, 2, 3, 4, 6]

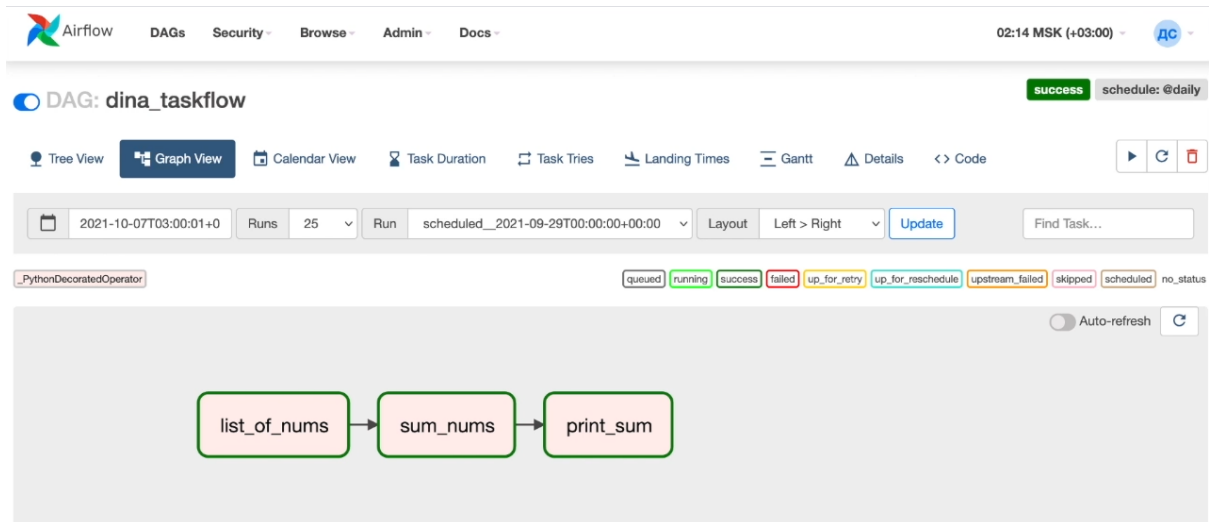
    @task
    def sum_nums(nums: list):
        return sum(nums)

    @task
    def print_sum(total: int):
        logging.info(str(total))

```

```
print_sum(sum_nums(list_of_nums()))  
  
dina_taskflow_dag = dina_taskflow()
```

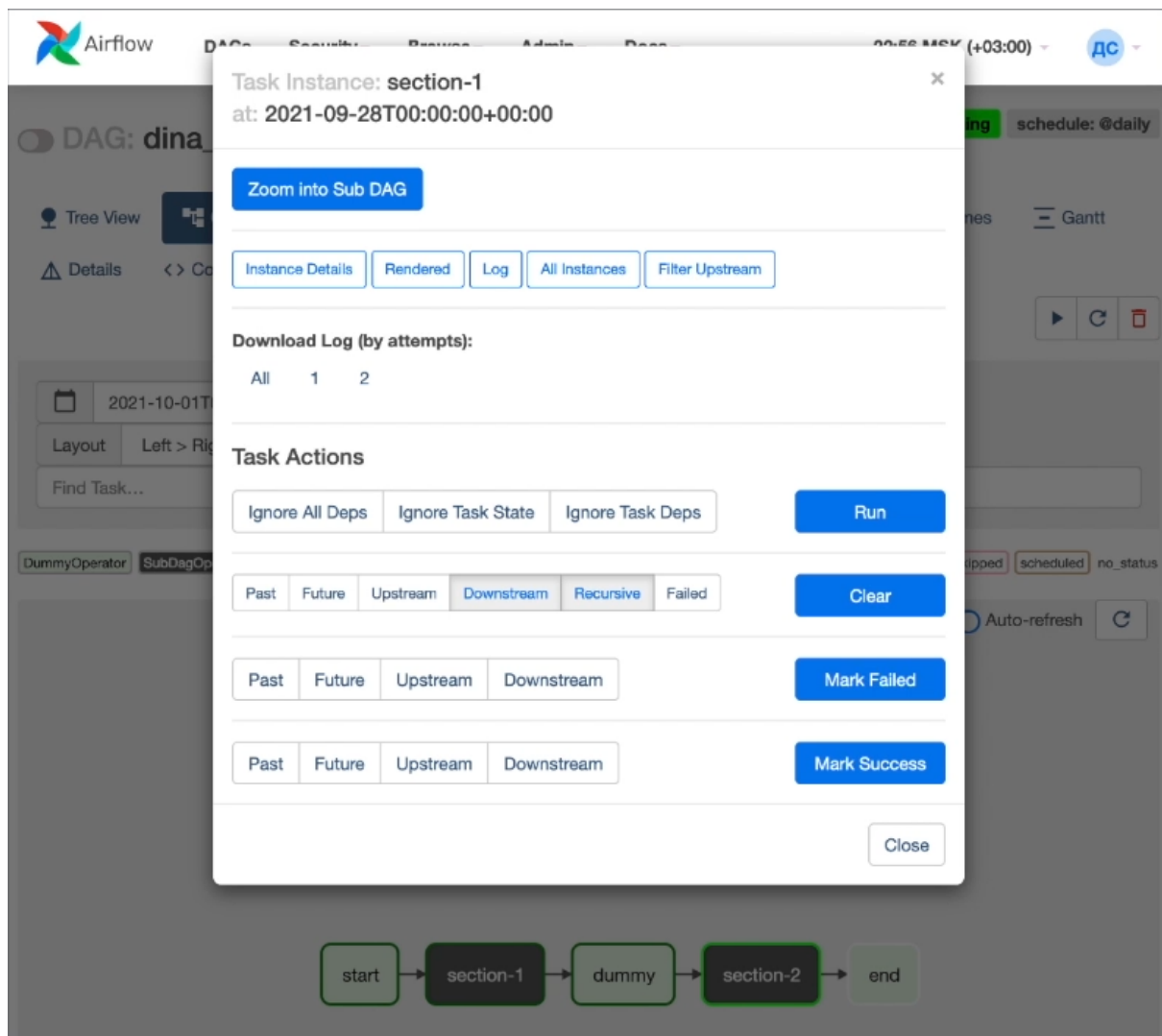
Как это выглядит в интерфейсе Airflow:



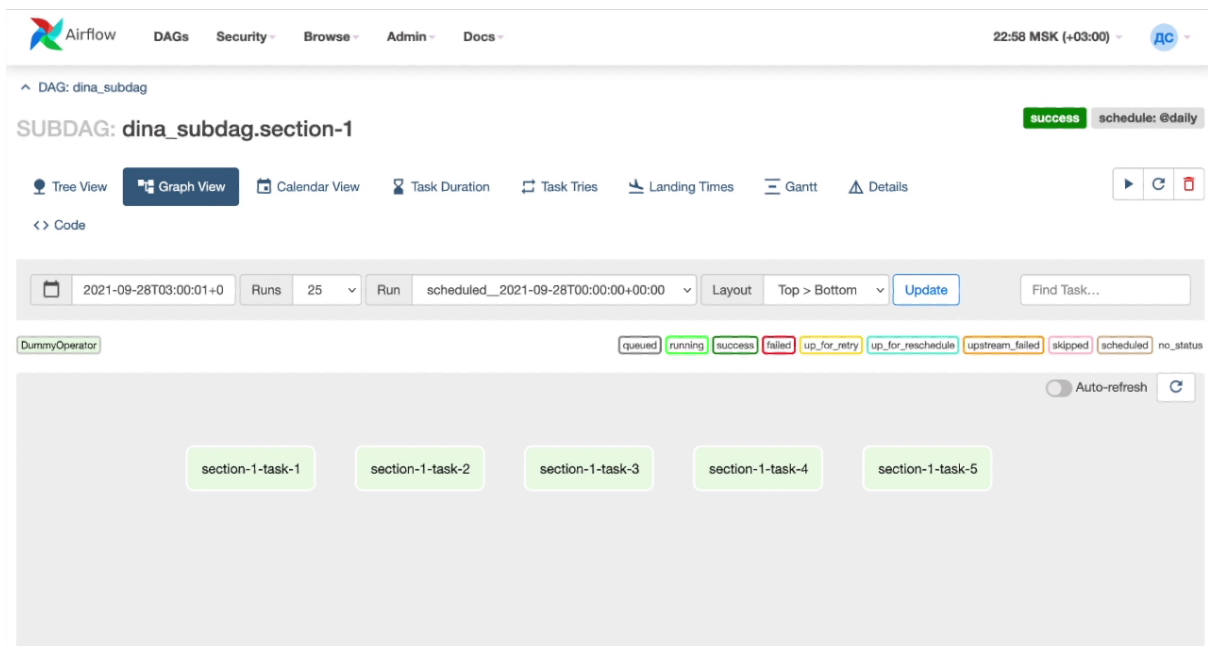
> SubDags

SubDags – дочерние даги. У SubDags две цели:

- переиспользование кода
- визуальная группировка



Обратим внимание на даг с дальнего плана, состоящий из 5 задач. Задачи section-1 и section-2 – это задачи типа **SubDagOperator**. При клике на них во всплывающем окне нажимаем на кнопку «Zoom into Sub DAG» и попадаем в отдельный даг, состоящий из 5 независимых друг от друга задач.



Функция, которая возвращает SubDag с задачами:

```
def subdag(parent_dag_name, child_dag_name, args):
    dag_subdag = DAG(
        dag_id=f'{parent_dag_name}.{child_dag_name}',
        default_args=args,
        start_date=days_ago(2),
        schedule_interval="@daily",
    )

    for i in range(5):
        DummyOperator(
            task_id=f'{child_dag_name}-task-{i + 1}',
            default_args=args,
            dag=dag_subdag,
        )

    return dag_subdag
```

Использование этого SubDag-a:

```
with DAG(DAG_NAME,
        schedule_interval='@daily',
        default_args=DEFAULT_ARGS,
        max_active_runs=1,
        tags=['karpov']) as dag:

    start = DummyOperator(task_id='start')
    dummy = DummyOperator(task_id='dummy')
    end = DummyOperator(task_id='end')
```

```

section_1 = SubDagOperator(
    task_id='section-1',
    subdag=subdag(DAG_NAME, 'section-1', DEFAULT_ARGS),
)

section_2 = SubDagOperator(
    task_id='section-2',
    subdag=subdag(DAG_NAME, 'section-2', DEFAULT_ARGS),
)

start >> section_1 >> dummy >> section_2 >> end

```

Что нужно знать про SubDag-и, чтобы ими пользоваться

- Расписание у дага и сабдага должны совпадать
- Название сабдагов: parent.child
- Состояние сабдага и таска SubDagOperator независимы
- По возможности избегайте использование сабдагов

> TaskGroup

Цели:

- Группировка задач
- Переиспользование кода

```

with DAG('dina_taskgroup',
    schedule_interval='@daily',
    default_args=DEFAULT_ARGS,
    max_active_runs=1,
    tags=['karpov']
) as dag:

    start = DummyOperator(task_id='start')

    with TaskGroup(group_id='group1') as tg1:
        for i in range(5):
            DummyOperator(task_id=f'task{i+1}')

    dummy = DummyOperator(task_id='dummy')

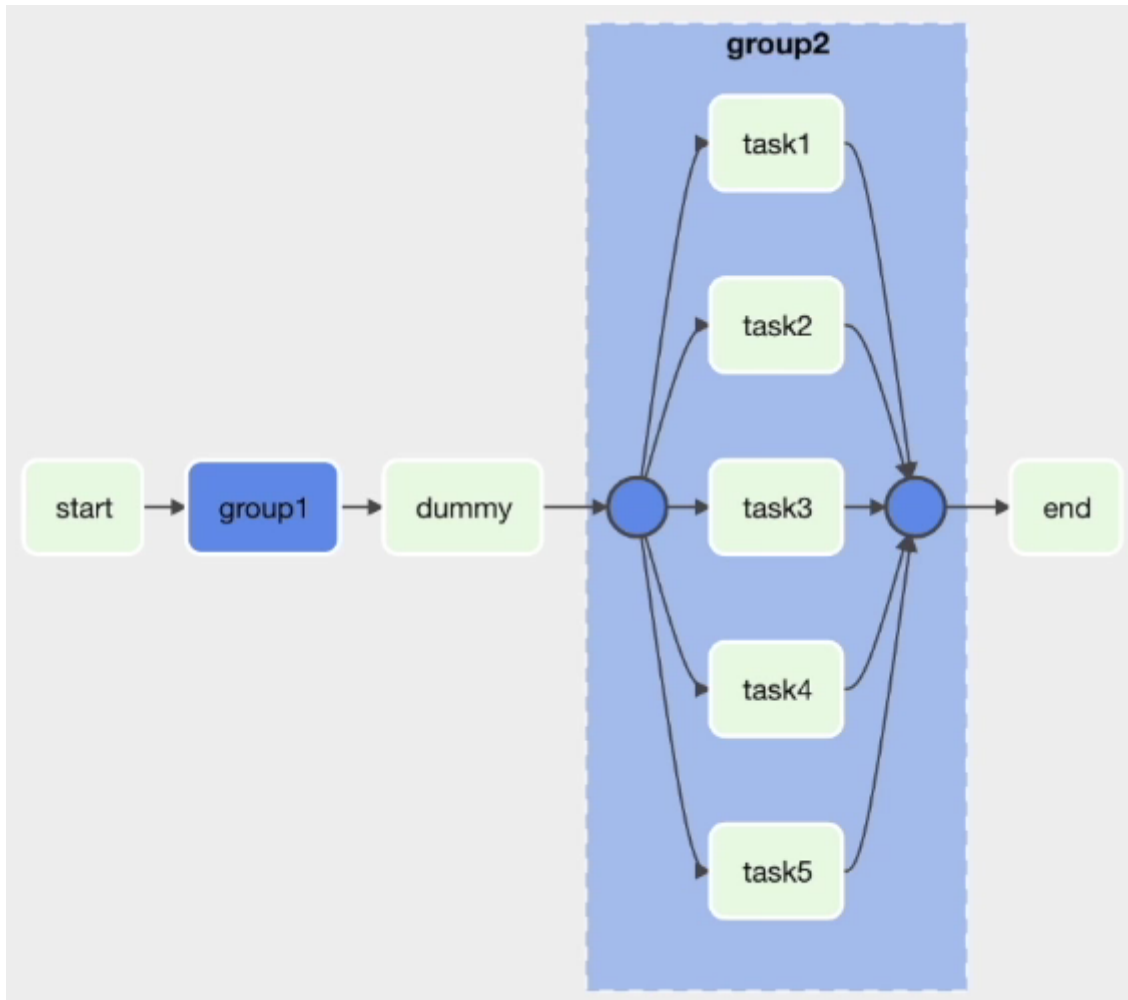
    with TaskGroup(group_id='group_2') as tg2:
        for i in range(5):
            DummyOperator(task_id=f'task{i+1}')

    end = DummyOperator(task_id='end')

```

```
start >> tg1 >> dummy >> tg2 >> end
```

Как это выглядит в интерфейсе Airflow:



> Динамическое создание дагов

- Скрипты должны находиться в **DAG_FOLDER** (внутри DAG_FOLDER можно организовать хранение по папкам)
- **dag в globals()** (переменная дага должна быть объявлена в глобальной области видимости)

Варианты для динамического создания дага:

- **Статическая генерация нескольких одинаковых дагов**: используется конструктор дага – функция, в которую передаём параметры, после

выполнения генерируется даг с нужным количеством задач. Одному скрипту соответствует любое количество дагов.

- **Генерация дага из глобальных переменных/соединений:** используется конструктор дага – функция, в которой используются глобальные переменные, которые есть в Airflow.
- **Генерация дага на основе json/yaml-файла:** удобно для работы с БД. Например, для каждой таблицы БД создаётся свой словарь json с параметрами, далее автопилот (код, который генерирует даги из json) смотрит на эти параметры и выстраивает по ним даг.

> Best Practice

- **Сохраняйте идемпотентность:** результат работы одного задания при перезапуске не должен изменяться
- **Не храните пароли в коде:** используйте connection-ы
- **Не храните файлы локально:** используйте промежуточные хранилища, HDFS, S3 и т.д.
- **Убирайте лишний код верхнего уровня:** всю логику переносите в код задания
- **Не используйте переменные Airflow:** загружайте переменные из jinja, загружайте переменные внутри задания, используйте переменные окружения