

# Урок 8. Airflow -2

List Pool

# Pools

Search ▾



Actions ▾



Pool ↕

Slots ↕

Running Slots



default\_pool

128

0



wget\_pool

2

0

```
aggregate_db_message_job = BashOperator(  
    task_id="aggregate_db_message_job",  
    execution_timeout=timedelta(hours=3),  
    pool="ep_data_pipeline_db_msg_agg",  
    bash_command=aggregate_db_message_job_cmd,  
    dag=dag,  
)
```

<https://airflow.apache.org/docs/apache-airflow/stable/concepts/pools.html>

# Sensors

```
waiting_for_file = FileSensor(  
    task_id='waiting_for_file',  
    poke_interval=30,  
    timeout=60 * 30  
)
```

- `poke_interval` : When using `poke` mode, this is the time in seconds that the sensor waits before checking the condition again. The default is 30 seconds.
- `exponential_backoff` : When set to `True` , this setting creates exponentially longer wait times between pokes in `poke` mode.
- `timeout` : The maximum amount of time in seconds that the sensor should check the condition for. If the condition has not been met when this time is reached, the task fails.
- `soft_fail` : If set to `True` , the task is marked as skipped if the condition is not met by the `timeout` .

# Connections



Airflow

DAGs

Security ▾

Browse ▾

Admin ▾

Docs ▾

## List Connection

Search ▾



Actions ▾



- Variables
- Configurations
- Connections
- Plugins
- Pools
- XComs



Conn Id ↑↓

Conn Type ↑↓

Description ↑↓

Host ↑↓



dwh\_prod\_s\_airflow

ms\_sql

dwh.prod.lan



dwh\_prod\_spark\_writer

ms\_sql

dwh.prod.lan

<https://airflow.apache.org/docs/apache-airflow/stable/howto/connection.html>

# Connections

Нужно исполнять внутри контейнера с веб сервисом:

```
sudo docker-compose exec airflow-webser airflow connections add ...
```

```
airflow connections add 'my_prod_db' \  
    --conn-uri 'my-conn-type://login:password@host:port/schema?param1=val1&param2=val2'
```

Alternatively you may specify each parameter individually:

```
airflow connections add 'my_prod_db' \  
    --conn-type 'my-conn-type' \  
    --conn-login 'login' \  
    --conn-password 'password' \  
    --conn-host 'host' \  
    --conn-port 'port' \  
    --conn-schema 'schema' \  
    ...
```

# Airflow scheduler

Может иметь несколько инстансов

- запуск дагов (DAG Run) по расписанию,
- мониторинг и обнаружение новых DAGов: постоянный парсинг файлов и синхронизация с базой

Без запущенного планировщика невозможно выполнить DAG.

# Airflow executors

- SequentialExecutor - только последовательный запуск задач
- LocalExecutor - несколько дагов одновременно, но на одной машине
- CeleryExecutor - требует брокера сообщений (Redis)
- DaskExecutor
- KubernetesExecutor - pod инстанс для каждой задачи

# start\_date и execution\_date

**Start Date** это дата начала от которой следует начинать запускать DAG согласно расписания `schedule_interval`.

**Execution Date** это дата выполнения конкретного запуска.

```
from airflow.operators.python import get_current_context
context = get_current_context()
execution_date = context['execution_date']
```



# Catchup

По умолчанию планировщик Airflow поставит в расписание все запуски начиная со `start_date + schedule_interval` и до `end_date` (или текущего времени). Отменить эти запуски можно опцией `catchup=False`

```
@dag(default_args=default_args,  
      schedule_interval='@monthly',  
      start_date=datetime(2020, 1, 1),  
      catchup=False,  
      )
```

# XComs (“cross-communications”)

XCom позволяет задачам обмениваться данными. Хорошо работает только для сериализуемых (JSON/pickle) данных малого размера. Работает автоматически (код слева), либо вручную (код справа)

```
def print_xcom(**kwargs):
    ti = kwargs['ti']
    print('The value is: {}'.format(
        ti.xcom_pull(task_ids='hello_world')
    ))

with DAG(
    dag_id='xcom_dag',
    start_date=datetime(2021, 3, 1),
    schedule_interval='@once',
) as dag:

    cmd = BashOperator(
        task_id='hello_world',
        bash_command='echo "Hello world"',
    )

    printer = PythonOperator(
        task_id='printer',
        python_callable=print_xcom,
    )

    cmd >> printer
```

```
def push_cmd(**kwargs):
    ti = kwargs['ti']
    ti.xcom_push(value='cat /etc/passwd | wc -l', key='passwd_len')

def pull_result(**kwargs):
    ti = kwargs['ti']
    print(ti.xcom_pull(task_ids="push_cmd"))
```

<https://khashtamov.com/ru/apache-airflow-xcom/>

```

@task
def download_titanic_dataset():
    url = 'https://web.stanford.edu/class/archive/cs/cs109/cs109.116
c.csv'
    response = requests.get(url, stream=True)
    response.raise_for_status()
    filepath = os.path.join(os.path.expanduser('~'), 'titanic.csv')
    with open(filepath, 'w', encoding='utf-8') as f:
        for chunk in response.iter_lines():
            f.write('{}\n'.format(chunk.decode('utf-8')))
    return filepath

@task
def get_number_of_lines(file_path):
    lines = 0
    with open(file_path) as f:
        for line in f:
            if line:
                lines += 1
    return lines

file_path = download_titanic_dataset()
number_of_lines = get_number_of_lines(file_path)

check_if_file_exists >> file_path

```

# TaskFlow API

Можно не использовать XCom

Работает только в Airflow 2.0+

<https://khashtamov.com/ru/airflow-taskflow-api/>

check\_file\_existence

download\_titanic\_dataset

get\_number\_of\_lines

# Динамическое создание дагов

1. Все даги из одного файла
2. Генерация скриптов для каждого дага

<https://www.astronomer.io/guides/dynamically-generating-dags>