

# №3

## Запускаем Airflow в Docker

Итак, нужно взять скрипт `rostelecom_processing`, который извлекает данные из таблицы №1, обрабатывает и помещает в таблицу №2 (Обозначения условные). Напишем Airflow DAG. Для этого для начала нужно поднять Airflow локально. Пропишем конфигурацию в `yaml` файле:

```
version: '3'
services:
  clickhouse-server-test:
    image: bitnami/clickhouse:latest
    container_name: clickhouse-server_2
    ports:
      - "8123:8123"
      - "9000:9000"
    environment:
      - CLICKHOUSE_ADMIN_PASSWORD=pivanet
    ulimits:
      nofile:
        soft: 262144
        hard: 262144

  postgres:
    image: postgres:13
    environment:
      - POSTGRES_USER=airflow
      - POSTGRES_PASSWORD=airflow
      - POSTGRES_DB=airflow
    volumes:
      - postgres_db:/var/lib/postgresql/data

  airflow-webserver:
```

```

image: apache/airflow:2.2.3
restart: always
depends_on:
  - postgres
environment:
  - AIRFLOW__CORE__EXECUTOR=LocalExecutor
  - AIRFLOW__CORE__SQL_ALCHEMY_CONN=postgresql+psycopg2://a:
  - AIRFLOW__CORE__FERNET_KEY=xDbjnEidiZ8ixmKgR-FpfcMqwg13Y
  - AIRFLOW__CORE__LOAD_EXAMPLES=False
ports:
  - "8080:8080"
volumes:
  - ./dags:/opt/airflow/dags
  - ./logs:/opt/airflow/logs
  - ./plugins:/opt/airflow/plugins
command: ["webserver"]

```

#### airflow-scheduler:

```

image: apache/airflow:2.2.3
restart: always
depends_on:
  - airflow-webserver
  - postgres
environment:
  - AIRFLOW__CORE__EXECUTOR=LocalExecutor
  - AIRFLOW__CORE__SQL_ALCHEMY_CONN=postgresql+psycopg2://a:
  - AIRFLOW__CORE__FERNET_KEY=xDbjnEidiZ8ixmKgR-FpfcMqwg13Y
volumes:
  - ./dags:/opt/airflow/dags
  - ./logs:/opt/airflow/logs
  - ./plugins:/opt/airflow/plugins
command: ["scheduler"]

```

#### volumes:

```
postgres_db:
```

Запускаем контейнеры.

```
docker-compose up -d
```

<input type="checkbox"/>			<a href="#">rotest</a>	Running (4/4)
<input type="checkbox"/>			<a href="#">airflow-scheduler-1</a> 5da54c086471	<a href="#">apache/airflow:2.2.3</a> Running
<input type="checkbox"/>			<a href="#">airflow-webserver-1</a> 736a0d93751d	Running <a href="#">8080:8080</a>
<input type="checkbox"/>			<a href="#">clickhouse-server_2</a> 93ce69228c72	Running <a href="#">8123:8123</a> <a href="#">Show all ports (2)</a>
<input type="checkbox"/>			<a href="#">postgres-1</a> 024b15fcdce3	Running <a href="#">postgres:13</a>

Контейнеры успешно запустились. Теперь перейдем в веб-интерфейс Airflow

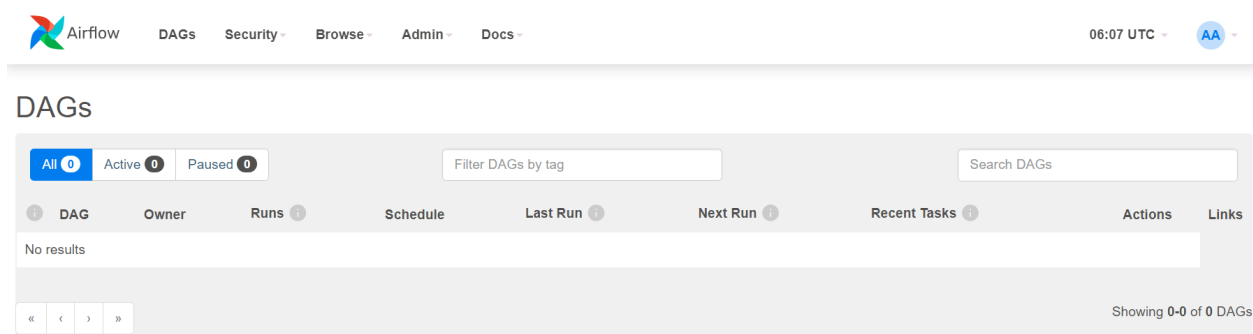
## Пишем тестовый DAG на обработку данных

Чтобы убедиться, что Airflow нормально взаимодействует с ClickHouse, напишем тестовый DAG.

Чтобы залогиниться нужно создать юзера. Сделаем стандартного юзера admin

```
docker exec -it rotest-airflow-webserver-1 bash
```

```
airflow users create --role Admin --username admin --email adm:
```

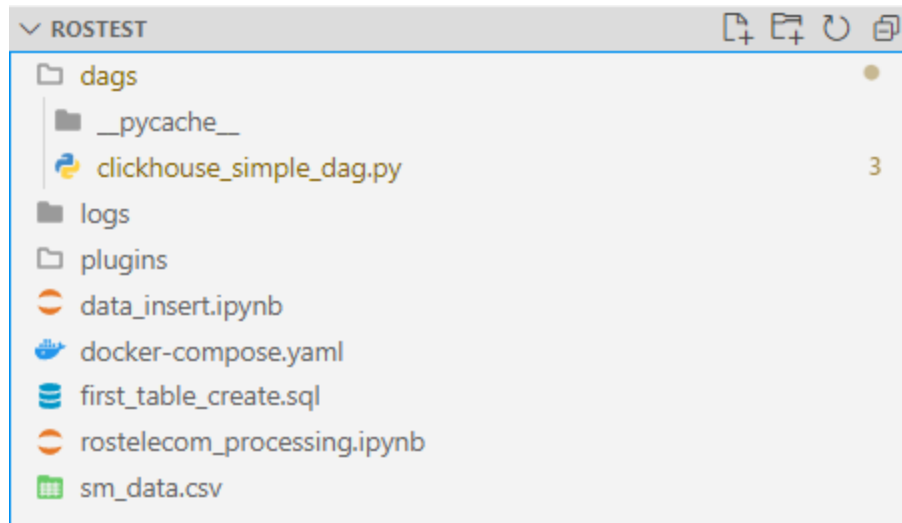


Переходим в Admin → Connections

Создаем новое соединение. Важно указать правильный host, чтобы Airflow обращался к Clickhouse по имени контейнера.

Напишем тестовый DAG, чтобы проверить, что Airflow может взаимодействовать с БД. Поместим DAG в папку dags проекта. После этого нужно установить необходимые библиотеки для работы:

```
pip install apache-airflow==2.2.3
```



Код DAG на выборку данных из таблицы:

```
from datetime import datetime, timedelta
from airflow import DAG
from airflow.providers.http.operators.http import SimpleHttpOperator
from airflow.operators.dummy import DummyOperator

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'email': ['your.email@example.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
    'start_date': datetime(2024, 1, 1),
}

dag = DAG(
    'clickhouse_simple_dag',
```

```

        default_args=default_args,
        description='A simple DAG to query ClickHouse',
        schedule_interval=timedelta(days=1),
        catchup=False,
    )

    start = DummyOperator(
        task_id='start',
        dag=dag,
    )

    query_clickhouse = SimpleHttpOperator(
        task_id='query_clickhouse',
        http_conn_id='clickhouse_default', # ID вашего соединения с ClickHouse
        endpoint='', # Здесь может быть ваш endpoint, если требуется
        method='POST',
        data="SELECT * FROM db_test.logss LIMIT 10;", # Пример SQL
        headers={"Content-Type": "application/sql"},
        response_check=lambda response: True if response.status_code == 200 else False,
        dag=dag,
    )

    end = DummyOperator(
        task_id='end',
        dag=dag,
    )

    start >> query_clickhouse >> end

```

Airflow видит DAG:



DAGs

Security

Bro

## DAGs

All **1** Active **1** Paused **0**



DAG

Owner



clickhouse\_simple\_dag

airflow



1



Airflow

DAGs

Security

Browse

Admin

Docs

DAG: clickhouse\_simple\_dag A simple DAG to query ClickHouse

success

Schedule: 1 day, 0:00:00



<> Code



2024-05-27T07:08:58Z

Runs

25

Run

manual\_\_2024-05-27T07:08:57.882972+00:00

Layout

Left > Right

Update

DummyOperator

SimpleHttpOperator

queued

running

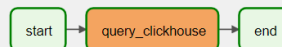
success

failed

up\_for\_retry

up\_for\_reschedule

upstream\_failed



DAG выполнен успешно:

Логи:

```
[2024-05-27, 07:08:59 UTC] {base.py:79} INFO - Using connection
[2024-05-27, 07:08:59 UTC] {http.py:140} INFO - Sending 'POST' t
[2024-05-27, 07:08:59 UTC] {taskinstance.py:1277} INFO - Marking
```

```
[2024-05-27, 07:09:00 UTC] {local_task_job.py:154} INFO - Task <
[2024-05-27, 07:09:00 UTC] {local_task_job.py:264} INFO - 1 dow
```

## Пишем актуальный DAG на обработку данных

Логика будет разделена на два файла. Один (dag\_script.py) содержит процесс обработки, второй (dag\_script\_run\_v1.py) логику DAG.

Когда делаем новый DAG нужно не забыть перекинуть выполняемый скрипт в папку /dags и указать имя файла через нужный путь, который доступен Аирфлоу:

```
/opt/airflow/dags/dag_script.py
```

```
docker exec -it rostest-airflow-webserver-1 pip install clickho
```

(Установка драйвера в контейнер)

```
docker exec -it rostest-airflow-scheduler-1 pip install clickho
```

Далее мы формируем файл dag\_script\_run\_v1.py, который будет выполнять dag\_script.py в DAG. dag\_script.py представляет из себя скрипт обработки, который берет данные из таблицы №1, преобразует и вставляет в таблицу №2 (Обозначения условные)

### DAG скрипт:

```
from datetime import datetime, timedelta
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from airflow.operators.dummy import DummyOperator
import json
import pandas as pd
```



```

from clickhouse_driver import Client
from pandas import json_normalize
import re
from dag_script import *

# Функция для запуска Python скрипта
def run_python_script():
    exec(open('/opt/airflow/dags/dag_script.py').read())

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'email': ['example@email.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
    'start_date': datetime(2024, 1, 1),
}

dag = DAG(
    'clickhouse_script_execution',
    default_args=default_args,
    description='Run a Python script to interact with ClickHouse',
    schedule_interval=timedelta(days=1),
    catchup=False,
)

start = DummyOperator(
    task_id='start',
    dag=dag,
)

run_script = PythonOperator(
    task_id='run_dag_script',
    python_callable=run_python_script,

```

```

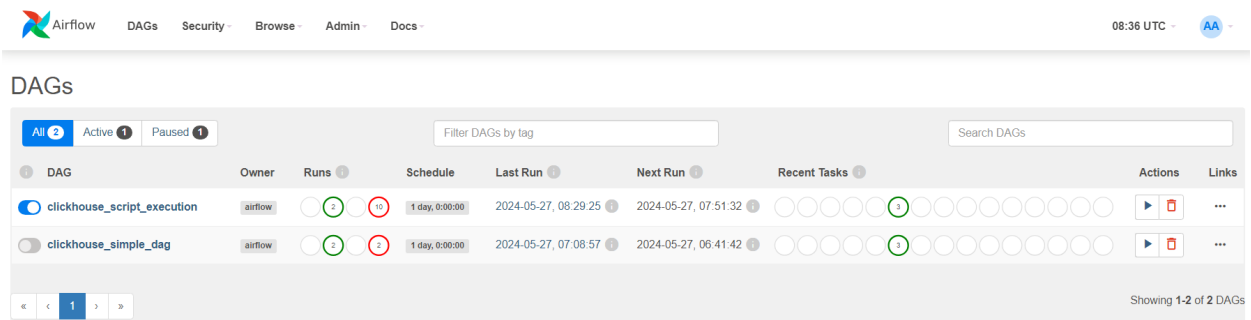
        dag=dag,
    )

    end = DummyOperator(
        task_id='end',
        dag=dag,
    )

    start >> run_script >> end

```

Airflow видит наш новый DAG (clickhouse\_script\_execution):



The screenshot shows the Airflow web interface. At the top, there's a navigation bar with links for DAGs, Security, Browse, Admin, and Docs. The main heading is "DAGs". Below it, there are tabs for "All" (2), "Active" (1), and "Paused" (1). A search bar and a filter box are also present. The table below lists the DAGs:

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
<input checked="" type="checkbox"/> clickhouse_script_execution	airflow	2 (1 green, 1 red)	1 day, 0:00:00	2024-05-27, 08:29:25	2024-05-27, 07:51:32	1 green, 1 red, 10 white	[Play] [Stop] [More]	
<input type="checkbox"/> clickhouse_simple_dag	airflow	2 (2 green)	1 day, 0:00:00	2024-05-27, 07:08:57	2024-05-27, 06:41:42	2 green, 5 white	[Play] [Stop] [More]	

At the bottom, there's a pagination bar showing "Showing 1-2 of 2 DAGs".

Таблица до выполнения DAG:

[illegible]

Запустим DAG:

The screenshot displays the Dagster web interface for a DAG named 'clickhouse\_script\_execution'. The top navigation bar includes links for Airflow, DAGs, Security, Browse, Admin, and Docs. The main header shows the DAG name and a description: 'Run a Python script to interact with ClickHouse'. The status is 'success', and the schedule is '1 day, 0:00:00'. The next run is scheduled for '2024-05-27, 07:51:32'.


Below the header, there are tabs for 'Tree', 'Graph', 'Calendar', 'Task Duration', 'Task Tries', 'Landing Times', 'Gantt', 'Details', and '<> Code'. The 'Graph' tab is selected, showing a simple linear DAG with three nodes: 'start', 'run\_dag\_script', and 'end'.

The execution history table shows a single run with the ID 'manual\_\_2024-05-27T08:29:25.370880+00:00'. The status is 'success'. The table has columns for 'Runs', 'Status', 'Run ID', 'Layout', and 'Update'. The 'Layout' dropdown is set to 'Left > Right'.

At the bottom, there are buttons for 'Trigger DAG' and 'Trigger DAG w/ config'. The 'Auto-refresh' checkbox is checked.

Логи:

Таблица после работы скрипта:

Обновить Save Cancel  Экспорт данных ... 200 3

## Итоги:

- ClickHouse развернут в Docker контейнере
- Создана БД, таблица
- В таблицу занесены данные из тестового набора
- Сделан скрипт для обработки тестовых данных и создания таблицы.
- Создана таблица для дальнейшей работы с данными специалистами по BI
- Развернут Airflow в Docker контейнере
- Написан DAG, который берет данные из первой таблицы, обрабатывает и заносит во вторую таблицу