

Практическая работа №5

Реализация ETL процессов в Apache Airflow

Цель работы

Получение практических навыков по созданию DAG файлов для Apache Airflow и реализации ETL процессов.

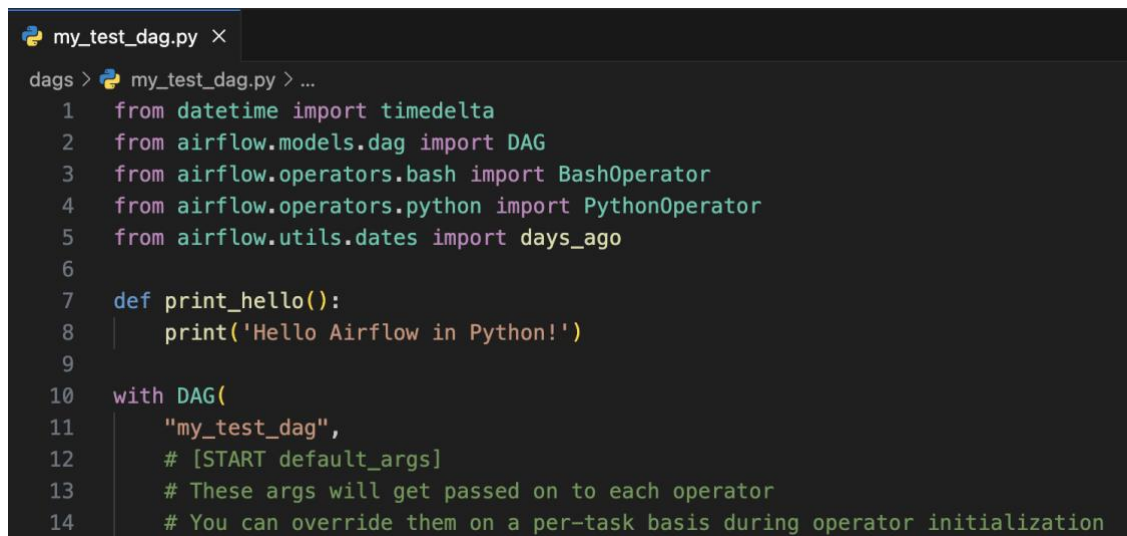
Индивидуальное задание

Создать простой DAG файл, DAG файл для получения, обработки и сохранения данных о погоде с OpenWeather для города Казань, а также DAG файл для получения, обработки и сохранения данных о пользователе VK.

Ход выполнения работы

Упражнение 1 - Создание простого DAG файла

Необходимо создать простой DAG для Apache Airflow, который будет выводить сообщения в bash и python. Создание дага my_test_dag.py в папке dags продемонстрировано на рисунках 1 и 2.



```
my_test_dag.py x
dags > my_test_dag.py > ...
1 from datetime import timedelta
2 from airflow.models.dag import DAG
3 from airflow.operators.bash import BashOperator
4 from airflow.operators.python import PythonOperator
5 from airflow.utils.dates import days_ago
6
7 def print_hello():
8     print('Hello Airflow in Python!')
9
10 with DAG(
11     "my_test_dag",
12     # [START default_args]
13     # These args will get passed on to each operator
14     # You can override them on a per-task basis during operator initialization
```

Рисунок 1 – Создание DAG файла

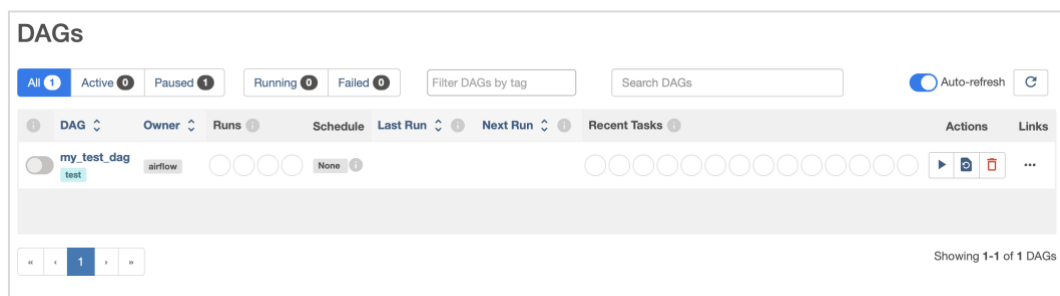


Рисунок 2 – DAG в веб-интерфейсе Airflow

Далее DAG запускается по нажатию на соответствующую кнопку. После запуска можно увидеть текущий статус и время исполнения DAG (рис. 3).

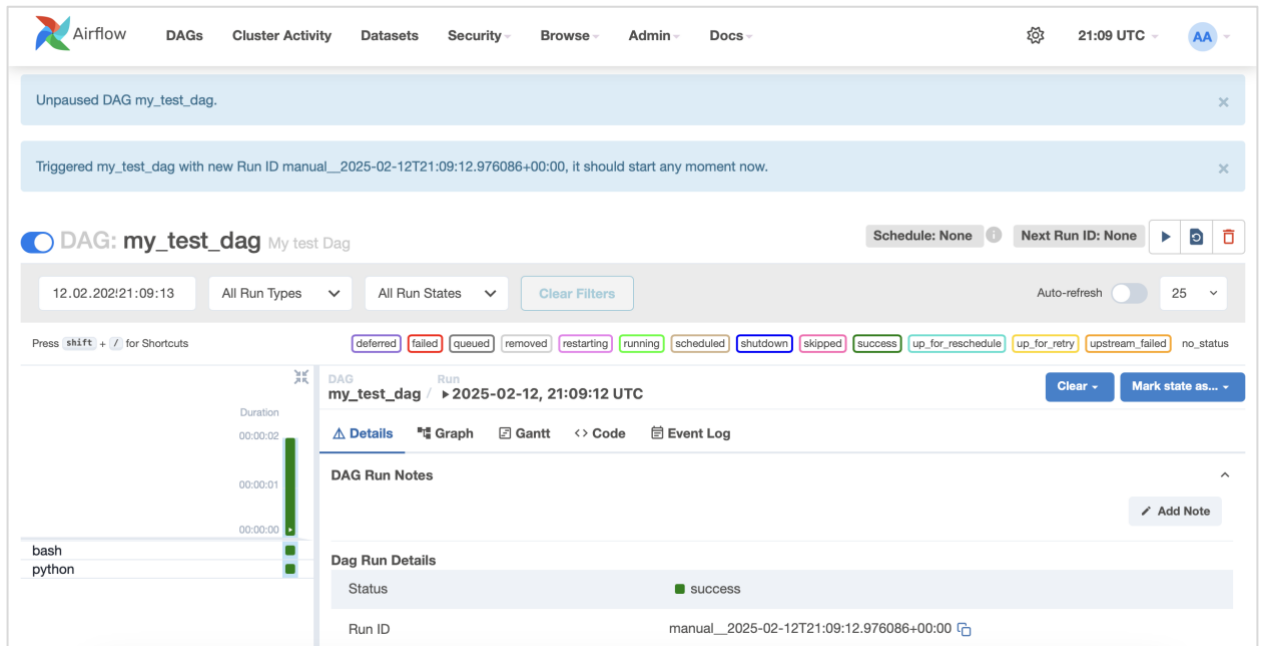


Рисунок 3 – Статус и время исполнения DAG

Далее просматривается подробная информация по каждой, а именно - лог исполнения задачи. В нашем случае у задачи 'bash' в логах выводится «Hello Airflow in Bash!», а у задачи 'python' — «Hello Airflow in Python!» (рис. 4).

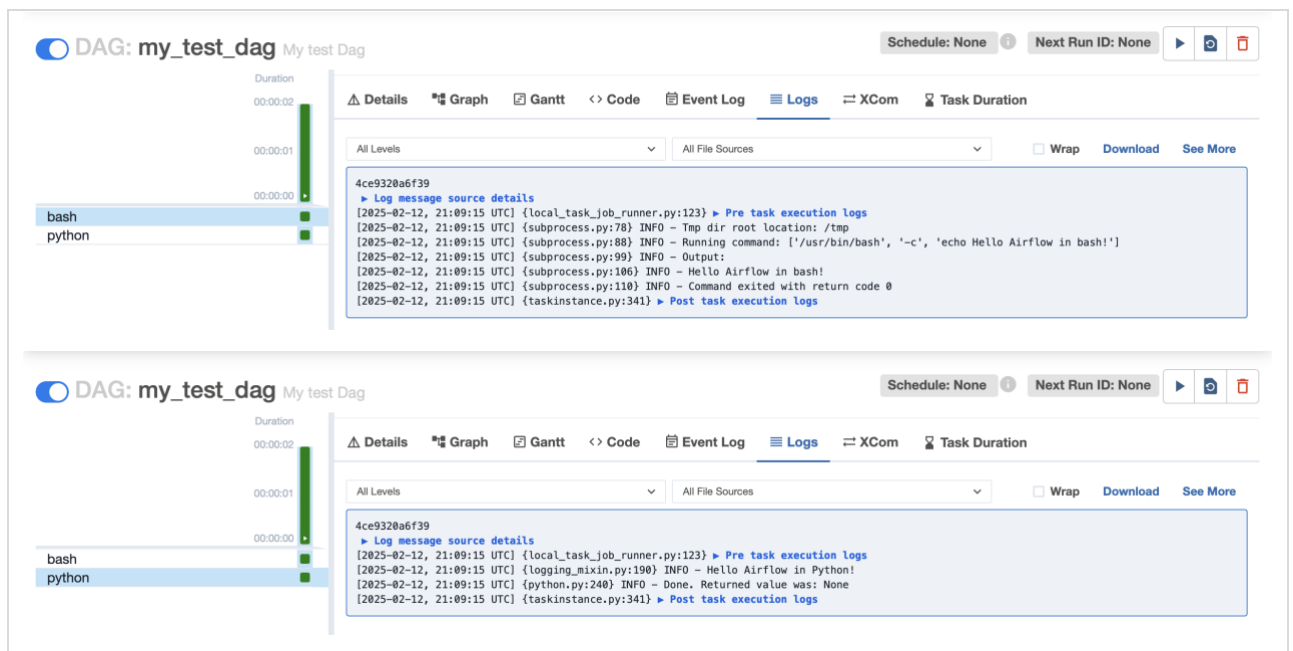


Рисунок 4 – Просмотр логов

Упражнение 2 – Создание DAG файла для получения, обработки и сохранения данных о погоде с OpenWeather.

Необходимо разработать DAG для Apache Airflow, который будет загружать данные о погоде из публичного API, обрабатывать их и сохранять в базу данных.

Для этой задачи будет использоваться публичный API OpenWeatherMap для получения данных о погоде. API предоставляет информацию о текущей погоде, прогнозе и исторических данных.

Пример запроса к API:

```
GET http://api.openweathermap.org/data/2.5/weather?q=London&appid=YOUR_API_KEY
```

Создается DAG с именем `weather_data_pipeline_dag.py`. DAG должен выполняться ежедневно в полночь.

DAG должен состоять из следующих задач:

- Download Data: загрузка данных о погоде из API;
- OpenWeatherMap;
- Process Data: очистка и преобразование загруженных данных;
- Save Data: сохранение обработанных данных в файл `parquet`;
- Export to Postgres: сохранение обработанных данных в БД.

Для получения данных о погоде с OpenWeather приобретается уникальный API Key, который добавляется в качестве переменной в Airflow во вкладке Admin → Variables (рис. 5).

Add Variable	
Key *	OPENWEATHER_API_KEY
Val	[Redacted]
Description	API ключ от OpenWeather

Рисунок 5 – Добавление временного API Key

Далее необходимо подключиться к БД. При разворачивании контейнеров, Postgres был развернут с пользователем Airflow. Создается подключение к PostgreSQL (рис. 6) и создается БД для записи данных (рис. 7).

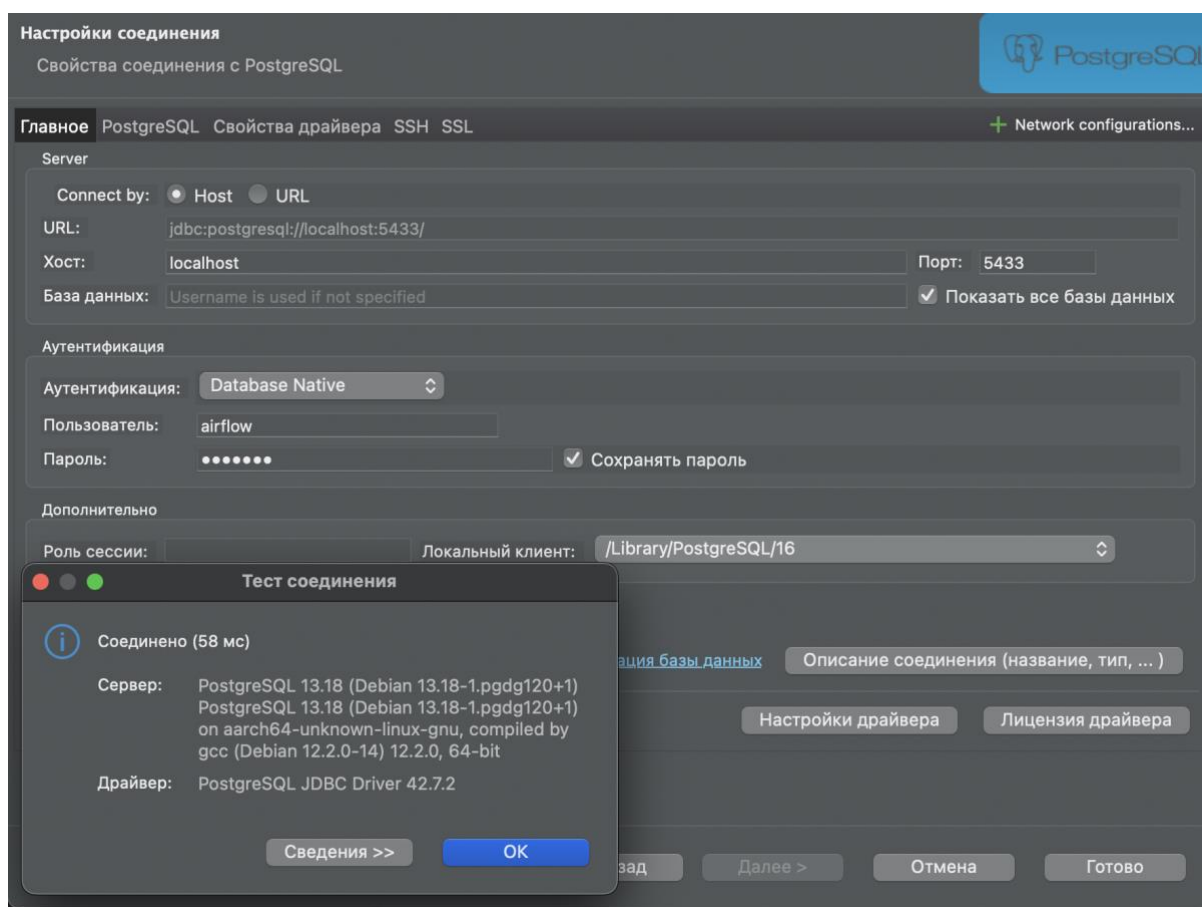


Рисунок 6 – Подключение к БД

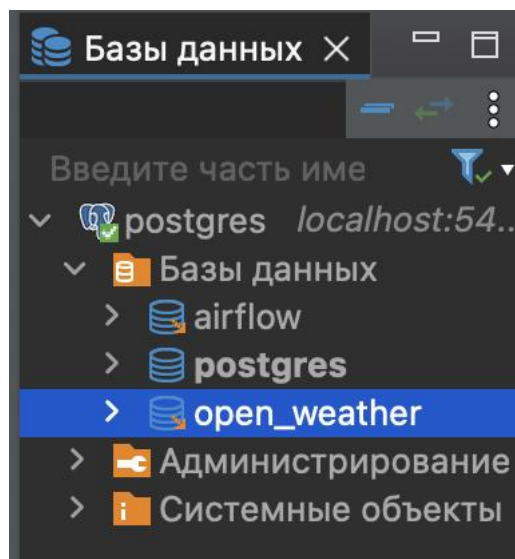


Рисунок 7 – Создание БД open_weather

Для записи данных в БД средствами Airflow, необходимо создать соответствующее подключение к локальной БД (рис. 8).

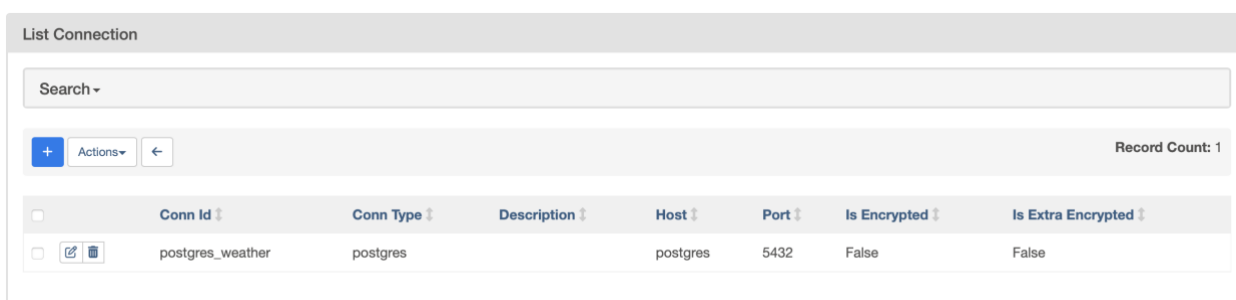


Рисунок 8 – Создание подключения

Далее производится создание DAG-файла `weather_data_pipeline_dag.py` (рис. 9). Этот DAG для Airflow реализует пайплайн для сбора, обработки и сохранения данных о погоде из API OpenWeatherMap. Он выполняет так называемый классический ETL процесс (Extract - извлечение, Transform - преобразование, Load – загрузка/сохранение).

Разработанный DAG будет обрабатывать информацию о погоде такого замечательного города как Казань. Выбор пал именно на нее, так как на прошлой неделе автору данной работы удалось посетить этот город.

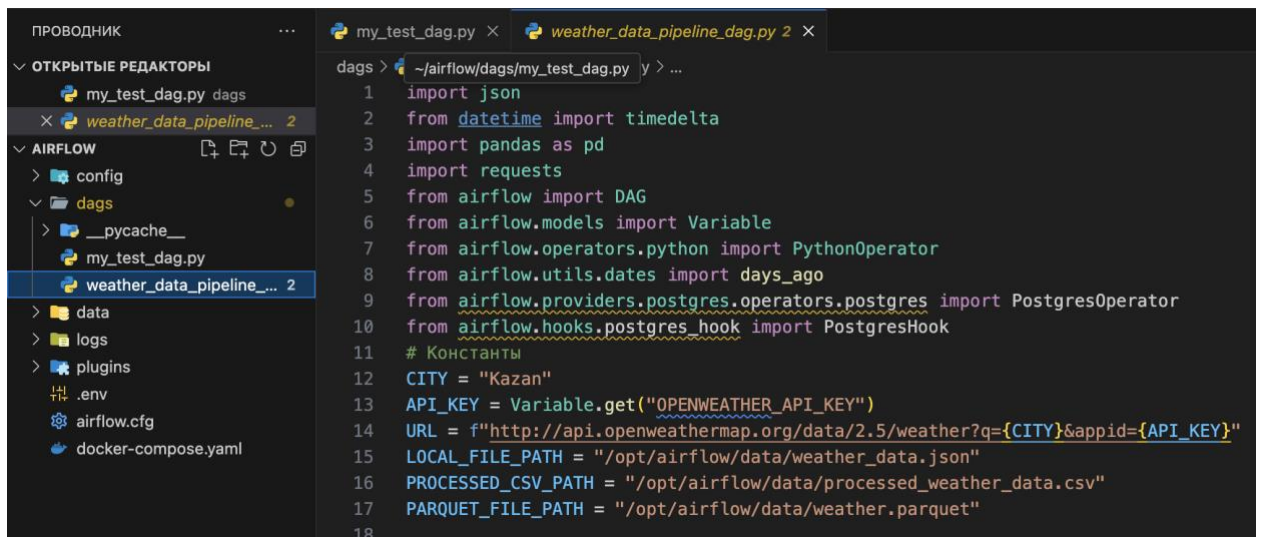


Рисунок 9 – Создание DAG файла

При обновлении списка DAG файлов в веб-интерфейсе Airflow возникла ошибка «Нет модуля airflow.providers.postgres.operators» (рис. 10).

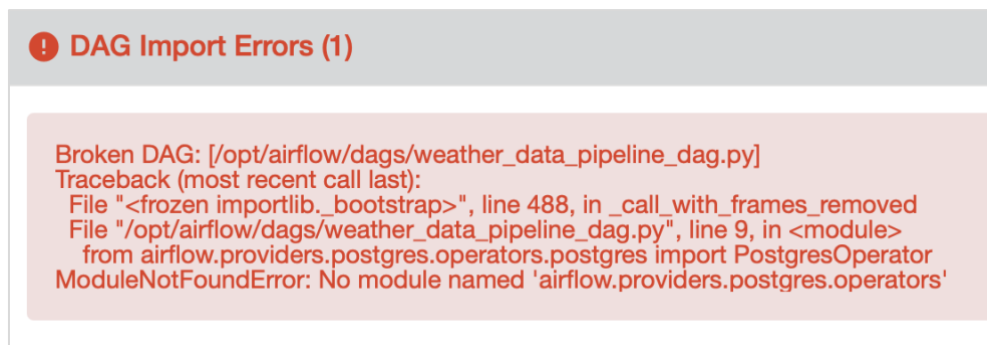


Рисунок 10 – Ошибка импорта модулей

Возможные источники ошибки были исследованы и было выявлено, что ошибка появлялась из-за опечаток в файлах docker-compose.yaml и weather_data_pipeline_dag.py. После исправления опечаток, ошибка исчезла (рис. 11).

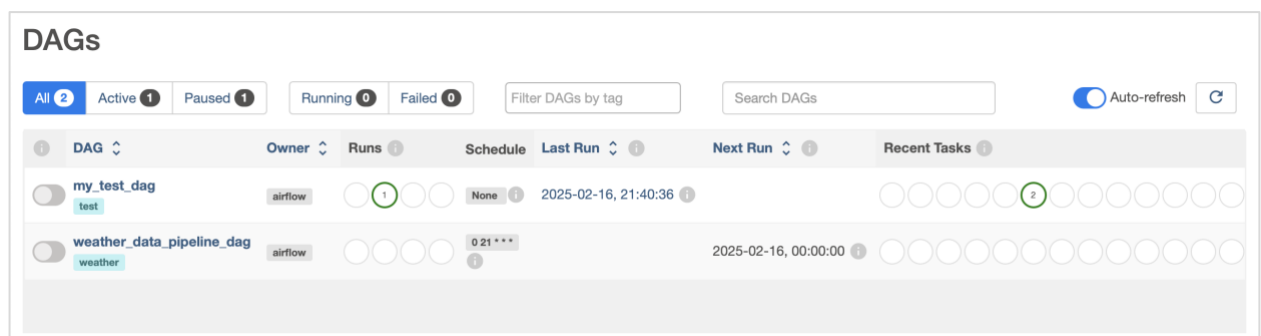


Рисунок 11 – DAG файлы

Для запуска DAG необходимо выбрать нужный даг из списка и нажать кнопку Trigger DAG.

В результате запуска DAG мы получили темно зеленый статус на всех задачах, что говорит об успешной работе DAG.

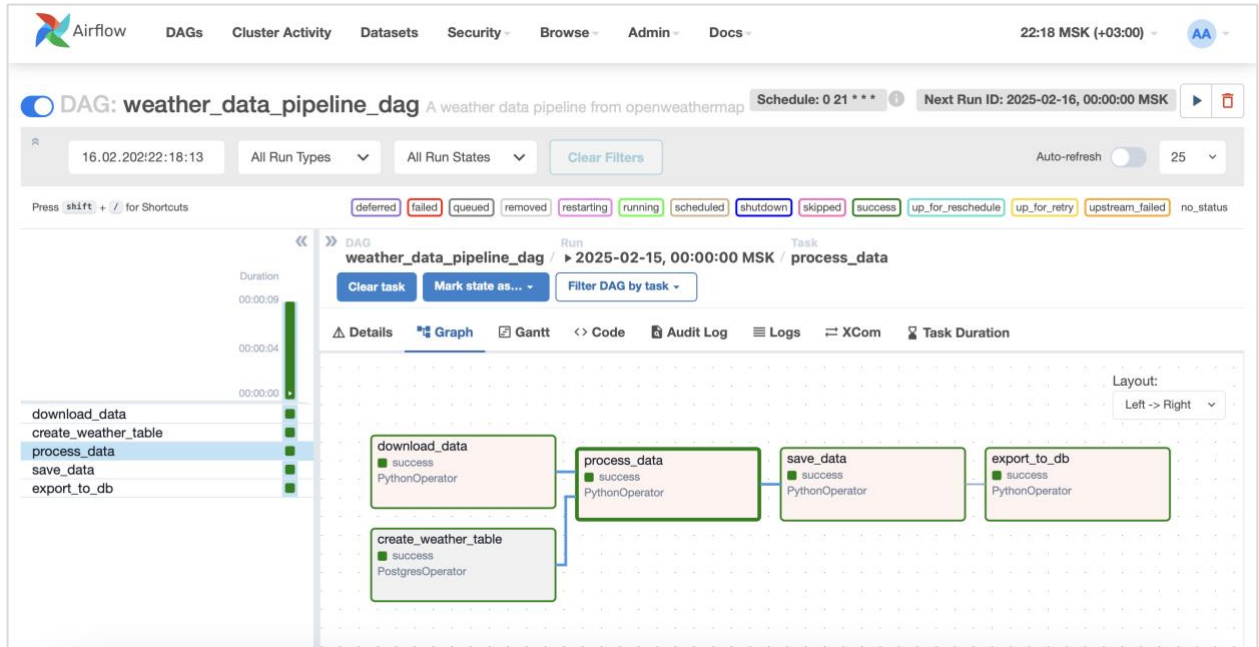


Рисунок 12 – Запуск DAG

Также в результате запуска дага были созданы 3 файла, которые находятся в директории data (рис. 13).

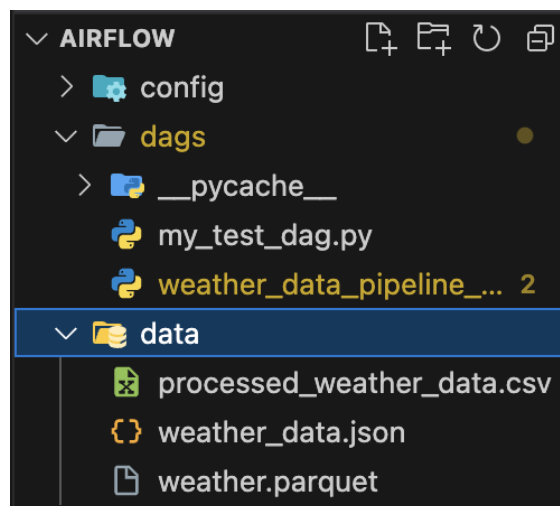


Рисунок 13 – Директория data

Обработанные данные были записаны в БД. Для проверки открывается DBeaver и делается запрос к БД weather_data (рис. 14).

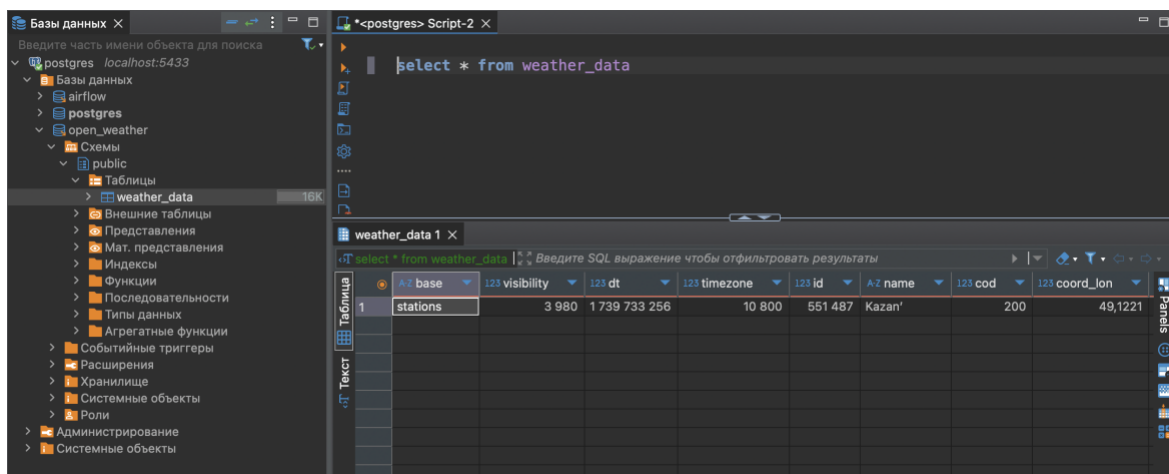


Рисунок 14 – Записи в БД

Далее реализовывается ETL процесс для выгрузки данных из открытого VK API, который будет сохранять в parquet, csv, json файлы, а также в БД Postgres.

Для начала изучается документация к API [1]. Затем приобретается личный токен, после чего в разделе API на сайте документации проверяется его действительность (рис. 15).

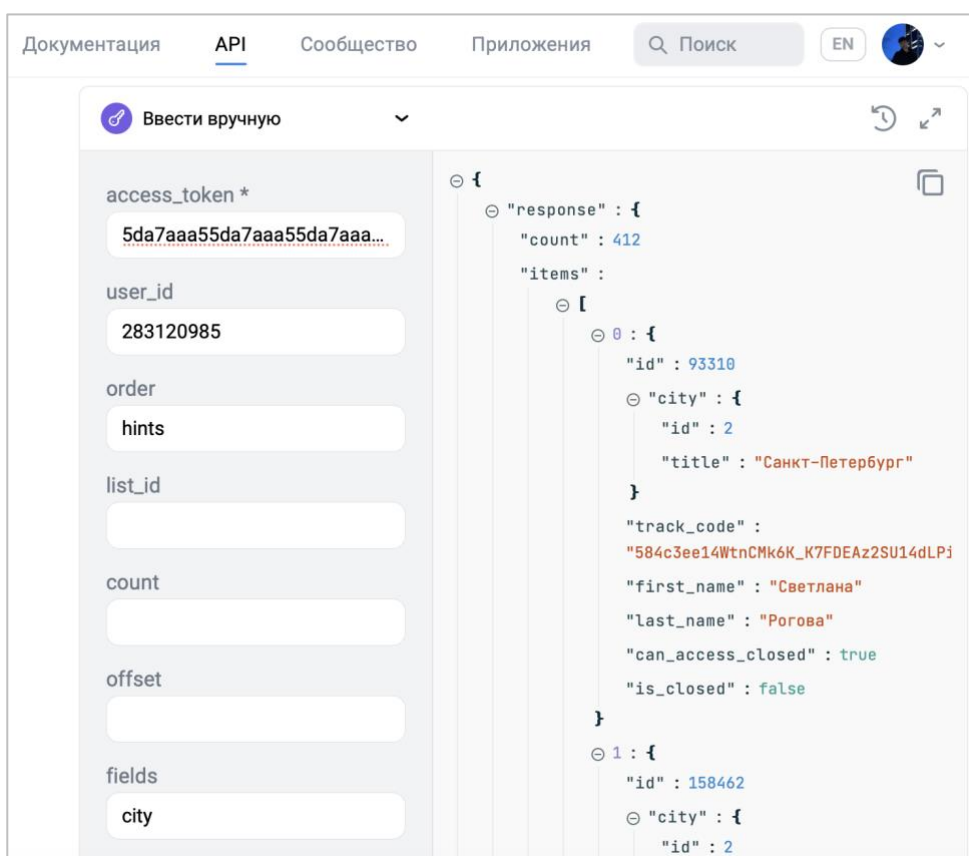


Рисунок 15 – Проверка токена

В веб-интерфейсе Airflow создаются переменные для хранения токена и ID пользователя, информацию о котором необходимо извлечь (рис. 16 (а. б)).

a)

The screenshot shows the 'Edit Variable' interface. The 'Key' field is labeled 'Key *' and contains the text 'VK_ACCESS_TOKEN'. The 'Val' field is labeled 'Val' and contains a series of asterisks '*****'. The 'Description' field is labeled 'Description' and contains the text 'Токен от ВК'. At the bottom of the form, there is a blue 'Save' button with a document icon and a white 'Back' button with a left arrow icon.

б)

The screenshot shows the 'Add Variable' interface. The 'Key' field is labeled 'Key *' and contains the text 'VK_USER_ID'. The 'Val' field is labeled 'Val' and contains the number '283120985'. The 'Description' field is labeled 'Description' and contains the text 'Мой VK ID'. At the bottom of the form, there is a blue 'Save' button with a document icon and a white 'Back' button with a left arrow icon.

Рисунок 16 – (а) создание переменной с токеном;

(б) создание переменной с ID пользователя

Создается БД в DBeaver, куда будут выгружаться данные (рис. 17).

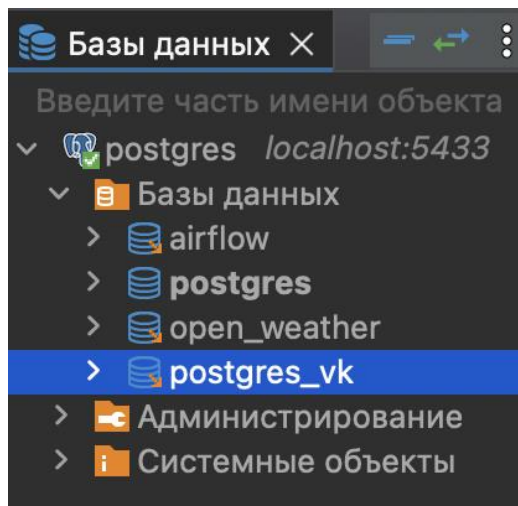


Рисунок 17 – Создание БД

Создается подключение к БД в панели Admin -> Connection (рис. 18).

List Connection							
Search ▾							
<div> <div>+</div> <div>Actions ▾</div> <div>←</div> </div> <div>Record Count: 2</div>							
<input type="checkbox"/>	Conn Id	Conn Type	Description	Host	Port	Is Encrypted	Is Extra Encrypted
<input type="checkbox"/>	postgres_vk	postgres	Подключение к локальной БД postgres_vk	postgres	5432	False	False
<input type="checkbox"/>	postgres_weather	postgres	Подключение к локальной БД	postgres	5432	False	False

Рисунок 18 – Создание подключения

Следующим шагом создается DAG файл (рис. 19), где прописывается вся логика ETL процесса. Код файла представлен в приложении А.

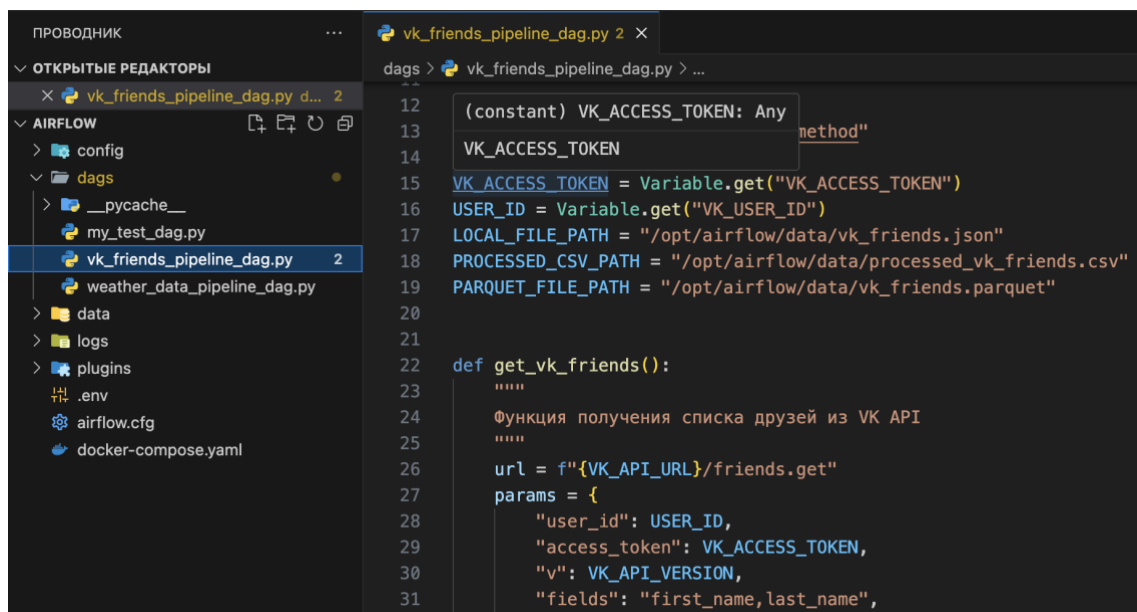


Рисунок 19 – DAG файл

DAG успешно отображился в веб-интерфейсе Airflow (рис. 20).

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks
my_test_dag test	airflow	1	None	2025-02-16, 21:40:36		2
vk_friends_pipeline_dag vk	airflow	1	0 21 ***		2025-02-16, 00:00:00	
weather_data_pipeline_dag weather	airflow	1	0 21 ***	2025-02-16, 22:17:16	2025-02-16, 00:00:00	5

Рисунок 20 – DAG в веб-интерфейсе Airflow

Далее DAG запускается и проверяется его работа (рис. 21).

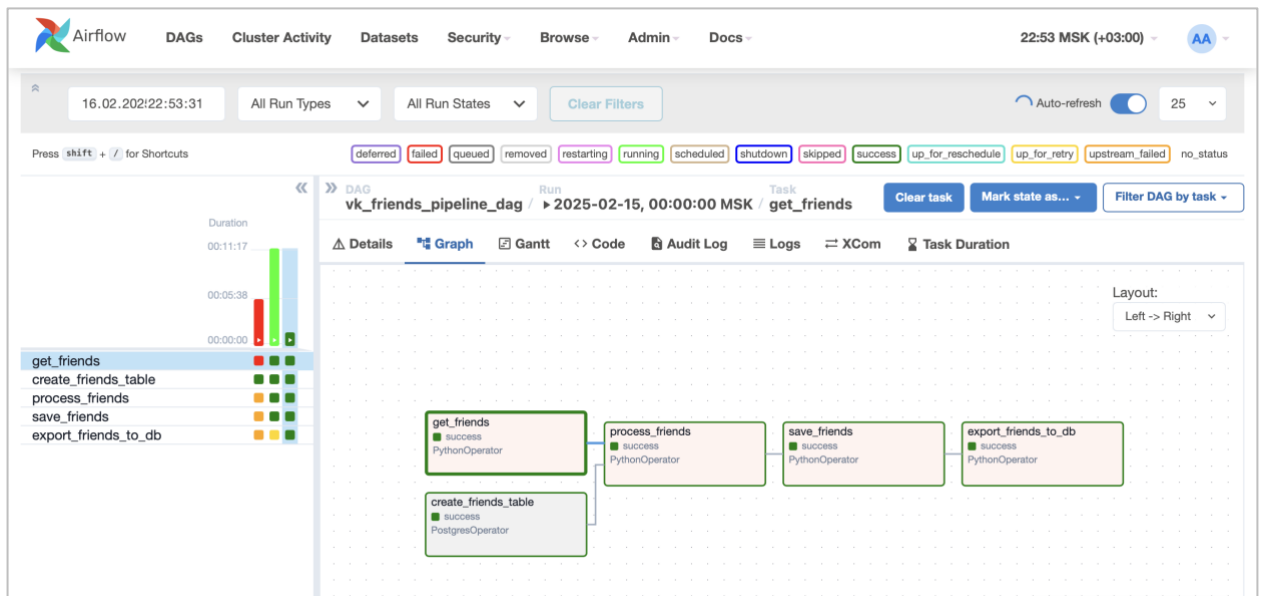


Рисунок 21 – Запуск DAG

При первых запусках DAG возникали ошибки при выполнении задачи. Проблема была в недействительности токена VK API. По этой причине в отчете присутствует этап проверки действительности токена.

Результат выполнения задачи и представлен на рисунке 22 (а, б).

а)

	user_id	first_name	last_name
395	767 298 823	Sarvar	Sodikov
396	768 660 236	Lyolya	Myaf
397	772 132 512	Tokhir	Sharipov
398	783 140 494	Oksana	Drugakova
399	786 833 273	Madiyor	Lazizov
400	806 929 650	Daniil	Khranovsky
401	808 171 323	Ali	Abdullaev
402	815 737 605	Ilyukha	Skornyakov
403	817 586 095	Shaxjan	Djuraevv
404	823 653 160	Kopitsentr	Lensoveta
405	834 649 443	Fozilzhon	Kodirov
406	846 486 007	Rafail	Davar
407	852 852 050	Sherali	Rashidov
408	875 462 383	Rustam	Sultonov
409	879 226 467	Guvanch	Chashemov
410	880 175 376	Abdulla	Sultanov
411	887 791 716	Kodirova	Dildora
412	948 475 718	Timur	Makhkamov

б)

	id	track_code	first_name	last_name	can_access_closed	is_closed
1	93310	57e124e9TaYUj6jBF1T3p2uo9upJGxLIOb8ERE46SvP_j8v	Svetlana	Rogova	true	false
2	158462	259954bfcvMH8bMzYukXniVI0b_qIHgtn299eP1WkVCwz	Alexey	Axenov	false	true
3	499540	308c33bdu6rjyPrpLEUK64rMMPvr9A0QZM_ML3FBX-QSN9hRYh7WwbmuzLkJRfsuCPfYgSdegQWpqjB	Maria	Arkharova	true	false
4	1049231	2bf8b7c7KXZ0deXK1-YjUeVsUemeabXYXVBBQOxybbfdqHHvVTj8HS5H0piC5C	Natalya	Grigoryeva	true	false
5	1176353	6a42b16aY5rYGBAVmrUYAujvSJR23nsUS85wDC92k_N	Elena	Yakovleva	true	false
6	1507178	fe0ca6beFNJ25CpYkg4b4veAw_Erg6wA-JgXjLZB8sK9kCCULx5uXyASGFmHTg9uQ-Hmjed7MDiwUwVg	Yulya	Kirillova	true	false
7	1547413	e8d998254eZ3gM0aNPL61BqWsz20A3AnMH2M6DEA1L	Anna	Streltsova	true	false
8	1661150	f10a36fau52CIE1VQeQNjxP3HePIXDUvVKHK-DqSUHGJmDh1KnW9o-5K1ESsA-HJRZY4nPICde9O-JFhA	Ivan	Vetrov	false	true

Рисунок 22 – (а) данные из таблицы в БД;

(б) данные в файлах

Лабораторная работа выполнена успешно и в полном объеме.

Вывод

В ходе выполнения работы я получил практические навыки по созданию DAG файлов в Apache Airflow. Были разработаны три DAG файла: простой DAG с выводом сообщений, DAG для получения, обработки и сохранения данных о погоде из OpenWeather API и DAG для работы с данными из VK API.

В процессе работы настроил подключение к PostgreSQL, создал необходимые базы данных и установил соединения в интерфейсе Airflow.

Возникшие ошибки при импорте модулей и работе с недействительным токеном VK API были успешно выявлены и устранены. В итоге все DAG файлы корректно отобразились, запустились и выполнили свои задачи, а данные были сохранены в файлы и базу данных.

Список использованных источников

1. Документация к открытому VK API, URL: <https://dev.vk.com/ru/reference> (дата обращения: 16.02.2025).

Приложения

Приложение А

Листинг 1 – Код DAG файла vk_friends_pipeline_dag.py

```
import json
from datetime import timedelta
import pandas as pd
import requests
from airflow import DAG
from airflow.models import Variable
from airflow.operators.python import PythonOperator
from airflow.utils.dates import days_ago
from airflow.providers.postgres.operators.postgres import PostgresOperator
from airflow.hooks.postgres_hook import PostgresHook

# Константы
VK_API_URL = "https://api.vk.com/method"
VK_API_VERSION = "5.131"
VK_ACCESS_TOKEN = Variable.get("VK_ACCESS_TOKEN")
USER_ID = Variable.get("VK_USER_ID")
LOCAL_FILE_PATH = "/opt/airflow/data/vk_friends.json"
PROCESSED_CSV_PATH = "/opt/airflow/data/processed_vk_friends.csv"
PARQUET_FILE_PATH = "/opt/airflow/data/vk_friends.parquet"

def get_vk_friends():
    """
    Функция получения списка друзей из VK API
    """
    url = f"{VK_API_URL}/friends.get"
    params = {
        "user_id": USER_ID,
        "access_token": VK_ACCESS_TOKEN,
        "v": VK_API_VERSION,
        "fields": "first_name,last_name",
    }
    response = requests.get(url, params=params)
    if response.status_code == 200:
        friends_data = response.json()
        if "response" in friends_data:
            with open(LOCAL_FILE_PATH, "w") as file:
                json.dump(friends_data["response"]["items"],
file)
        else:
```

```

        raise Exception(f"Error fetching data from VK API:
{friends_data}")
    else:
        raise Exception(f"Error fetching data from VK API:
{response.status_code}")

def process_friends_data():
    """
    Функция обработки данных друзей
    """
    with open(LOCAL_FILE_PATH, "r") as file:
        friends_data = json.load(file)

    df = pd.DataFrame(friends_data)
    df.to_csv(PROCESSED_CSV_PATH, index=False)

def save_friends_data():
    """
    Функция сохранения данных в parquet-файл
    """
    processed_df = pd.read_csv(PROCESSED_CSV_PATH)
    processed_df.to_parquet(PARQUET_FILE_PATH)

def export_friends_to_postgres():
    """
    Функция сохранения данных друзей в Postgres БД
    """

    df = pd.read_csv(PROCESSED_CSV_PATH)

    hook = PostgresHook(postgres_conn_id="postgres_vk")
    conn = hook.get_conn()
    cursor = conn.cursor()

    for index, row in df.iterrows():
        cursor.execute(
            f"""
            INSERT INTO vk_friends
            (user_id, first_name, last_name)
            VALUES (
                {row["id"]},
                '{row["first_name"]}',
                '{row["last_name"]}'
            );
            """
        )

    conn.commit()
    cursor.close()
    conn.close()

```



```

default_args = {
    "depends_on_past": False,
    "email_on_failure": False,
    "email_on_retry": False,
    "retries": 1,
    "retry_delay": timedelta(minutes=2),
    "start_date": days_ago(1),
}

with DAG(
    "vk_friends_pipeline_dag",
    default_args=default_args,
    description="A VK friends data pipeline",
    schedule_interval="0 21 * * *", # Запуск в полночь по MSK
    (UTC+3)
    catchup=False,
    tags=["vk"],
) as dag:
    get_friends_task = PythonOperator(
        task_id="get_friends",
        python_callable=get_vk_friends,
    )

    process_friends_task = PythonOperator(
        task_id="process_friends",
        python_callable=process_friends_data,
    )

    save_friends_task = PythonOperator(
        task_id="save_friends",
        python_callable=save_friends_data,
    )

    create_friends_table = PostgresOperator(
        task_id="create_friends_table",
        postgres_conn_id="postgres_vk",
        sql="""
        CREATE TABLE IF NOT EXISTS vk_friends (
            user_id INT PRIMARY KEY,
            first_name VARCHAR(255),
            last_name VARCHAR(255)
        );
        """,
    )

    export_friends_to_db = PythonOperator(
        task_id="export_friends_to_db",
        python_callable=export_friends_to_postgres,
    )

```

```
[get_friends_task, create_friends_table] >>  
process_friends_task >> save_friends_task >>  
export_friends_to_db
```