Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления» (ИУ)

КАФЕДРА «Информационная безопасность» (ИУ8)

Отчёт

по лабораторной работе № 5-6
по дисциплине «Программирование на Python»

Тема: «Python + Airflow, оркестрация данных»

Выполнил: Бородин Г.
студент группы ИУ8-13М

Проверил: Зотов М.

г. Москва,2025 г.

1. ЦЕЛЬ РАБОТЫ

Реализовать систему обновления данных посредствам системы-оркестрации Airflow

2. ПОСТАНОВКА ЗАДАЧИ

- Создан даг, который осуществляется инициализацию вашей схемы внутри метабазы данных Airflow(таблицы, функции)
- Создан даг, который осуществляет загрузку данных в вашу таблицу
- Создан даг, который осуществляет расчет метрик согласно заданию
- После выполнения расчета, в вашей созданной таблице для метрик, можно ознакомиться с показателями.

Условия: Ваша компания занимается авиаперевозками. Ваши основные данные находятся в таблице logistic.transfers (id BIGINT, from TEXT, to TEXT, departure_date DATE, arriving_date). Вам нужно вывести общее количество отправленных рейсов на каждый день.

3. ХОД РАБОТЫ

ИСПОЛЬЗУЕТСЯ В ДАННОЙ РАБОТЕ: Python; Apache Airflow (оркестрация задач и DAG); Docker и docker-compose (развёртывание Airflow + PostgreSQL + Redis); PostgreSQL (хранение исходных данных и метрик); провайдер Airflow apache-airflow-providers-postgres (PostgresOperator/PostgresHook); GitHub (хранение кода и конфигураций). Для выполнения SQL используется PostgresOperator, для генерации данных применяется PythonOperator и PostgresHook.

МОЙ ХОД РАБОТЫ:

ШАГ 1. Анализ задачи и проектирование пайплайна.

Определены входные данные: таблица logistic.transfers (id, from, to, departure_date, arriving_date). Определён результат: метрика “количество отправленных рейсов по дням” с сохранением в отдельной таблице. Пайплайн разбит на три независимых DAG:

инициализация схемы и таблиц;

загрузка (генерация) данных в transfers;

расчёт ежедневной метрики и запись в таблицу метрик.

ШАГ 2. Создание схемы и таблиц в Postgres (DAG init_schema_airline).

Реализован DAG, который создаёт схему logistic, таблицу исходных данных logistic.transfers и таблицу метрик logistic.daily_flights_count. Для выполнения SQL используется PostgresOperator.

ШАГ 3. Загрузка данных (DAG init_airline_data).

Реализован DAG генерации тестовых рейсов. Перед генерацией таблица logistic.transfers очищается. Далее данные создаются порциями

(chunks) в нескольких параллельных задачах TaskGroup: для каждой порции генерируются аэропорты вылета/прилёта и даты в заданном диапазоне, затем записи вставляются в Postgres через PostgresHook.

ШАГ 4. Расчёт метрик (DAG calculate_daily_flights).

Реализован DAG, который очищает таблицу метрик и заново рассчитывает “количество рейсов на каждый день” запросом GROUP BY departure_date по таблице logistic.transfers, после чего записывает результат в logistic.daily_flights_count.

ШАГ 5. Развёртывание окружения.

Подготовлены конфигурации Docker (Dockerfile, docker-compose, requirements, .env), обеспечивающие запуск Airflow с Postgres и Redis, а также установку нужного провайдера Postgres для операторов и хуков.

4. ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы были изучены основы оркестрации задач в Apache Airflow и принципы построения ETL/ELT-процессов на базе DAG. Были реализованы три DAG: создание схемы и таблиц в Postgres, загрузка (генерация) тестовых данных в таблицу рейсов и расчёт ежедневной метрики количества отправленных рейсов с сохранением результата в отдельной таблице метрик. Также выполнена настройка окружения в Docker (Airflow + Postgres + Redis) и подключение провайдера Postgres для выполнения SQL и работы с БД.

5. ПРИЛОЖЕНИЕ

Реализованный код располагается по следующей ссылке: https://github.com/zezOtik/bmstu--iu8--python/tree/feature/Borodin/Laba5

Листинг1 – DAG calculate_daily_flights: очистка таблицы метрик logistic.daily_flights_count и расчёт количества рейсов по дням (агрегация COUNT(*) GROUP BY departure_date) с записью результата в таблицу метрик.

```
from datetime import datetime
from airflow import DAG
from airflow.providers.postgres.operators.postgres import PostgresOperator

with DAG(
    'calculate_daily_flights',
    start_date=datetime(2023, 1, 1),
    schedule_interval='@daily',
    catchup=False,
    description='Расчёт ежедневного количества отправленных рейсов'
) as dag:

    # Очищаем таблицу с метриками перед новым расчётом
    truncate_metrics = PostgresOperator(
        task_id='truncate_metrics',
        postgres_conn_id='airflow_db',
        sql='TRUNCATE TABLE logistic.daily_flights_count;'
    )

    # Выполняем расчёт количества рейсов по дням
    calculate_metrics = PostgresOperator(
        task_id='calculate_daily_counts',
        postgres_conn_id='airflow_db',
```

```
    sql='''
    -- Заполняем таблицу метрик количеством рейсов по дням
    INSERT INTO logistic.daily_flights_count (flight_date, flight_count)
    SELECT
        departure_date AS flight_date,  -- Дата отправления как ключ метрики
        COUNT(*) AS flight_count        -- Количество рейсов в этот день
    FROM logistic.transfers
    GROUP BY departure_date             -- Группируем по дате отправления
    ORDER BY flight_date;               -- Сортируем по дате
    '''
  )

  # Определяем порядок выполнения задач
  truncate_metrics >> calculate_metrics
```

Листинг 2 – DAG init_airline_data: генерация тестовых данных о рейсах и загрузка их в таблицу logistic.transfers. Данные создаются порциями в параллельных задачах (TaskGroup), вставка выполняется через PostgresHook.

```
from datetime import datetime, timedelta
import random
from airflow import DAG
from airflow.providers.postgres.hooks.postgres import PostgresHook
from airflow.operators.python import PythonOperator
from airflow.utils.task_group import TaskGroup
import math

# Параметры DAG по умолчанию
DEFAULT_ARGS = {
    'owner': 'airflow',
    'depends_on_past': False,
```

```
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

# Список аэропортов мира (коды IATA)
AIRPORTS = [
    'SVO', 'JFK', 'LHR', 'CDG', 'FRA', 'NRT', 'PEK', 'SYD', 'DXB', 'SIN',
    'AMS', 'HKG', 'LAX', 'ORD', 'MAD', 'FCO', 'IST', 'GRU', 'YYZ', 'BOM'
]

def generate_transfers_chunk(**context):
    """
    Генерирует порцию данных о рейсах.

    Параметры из контекста:
    - chunk_number: номер текущей порции
    - total_flights: общее количество рейсов для генерации
    - min_date/max_date: диапазон дат для генерации
    """
    task_id = context['task_instance'].task_id
    chunk_number = int(task_id.split('_')[-1])

    params = context['params']
    total_flights = params['total_flights']
    num_chunks = params['num_chunks']
    min_date_str = context['templates_dict']['min_date']
    max_date_str = context['templates_dict']['max_date']

    # Рассчитываем границы для текущей порции данных
```

```
    chunk_size = math.ceil(total_flights / num_chunks)
    start_idx = chunk_number * chunk_size + 1
    end_idx = min((chunk_number + 1) * chunk_size, total_flights)

    # Парсим даты
    min_date = datetime.strptime(min_date_str, '%Y-%m-%d')
    max_date = datetime.strptime(max_date_str, '%Y-%m-%d')
    max_allowed_date = datetime(2025, 12, 31)
    max_date = min(max_date, max_allowed_date)

    # Подключаемся к базе данных
    pg_hook = PostgresHook(postgres_conn_id='airflow_db')
    conn = pg_hook.get_conn()
    cursor = conn.cursor()

    inserted_count = 0
    # Генерируем рейсы для текущей порции
    for flight_id in range(start_idx, end_idx + 1):
        # Выбираем случайные аэропорты (отправления и назначения)
        origin = random.choice(AIRPORTS)
        destination = random.choice(AIRPORTS)
        # Убеждаемся, что аэропорты отправления и прибытия разные
        while destination == origin:
            destination = random.choice(AIRPORTS)

        # Генерируем случайную дату отправления в заданном диапазоне
        days_range = (max_date - min_date).days
        departure_date = min_date + timedelta(days=random.randint(0,
days_range))
```

```
        # Генерируем длительность перелёта (от 1 до 48 часов)
        flight_duration = timedelta(hours=random.randint(1, 48))
        arrival_datetime = departure_date + flight_duration
        arrival_date = arrival_datetime.date()

        # Вставляем данные в таблицу
        cursor.execute("""
            INSERT INTO logistic.transfers (id, "from", "to", departure_date,
arriving_date)
            VALUES (%s, %s, %s, %s, %s)
            ON CONFLICT (id) DO NOTHING
        """, (flight_id, origin, destination, departure_date, arrival_date))

        inserted_count += 1

    # Фиксируем изменения и закрываем соединение
    conn.commit()
    cursor.close()
    conn.close()

    # Сохраняем количество вставленных записей для логирования
    context['ti'].xcom_push(key=f'chunk_{chunk_number}_count',
value=inserted_count)
    return f"Вставлено {inserted_count} рейсов в порции {chunk_number}"

with DAG(
    dag_id='init_airline_data',
    default_args=DEFAULT_ARGS,
```

```
    description='Генерация тестовых данных о рейсах авиакомпании',
    schedule_interval='@once',
    start_date=datetime(2023, 1, 1),
    catchup=False,
    tags=['авиакомпания', 'генерация-данных'],
    params={
        'total_flights': 10000,  # Общее количество рейсов для генерации
        'num_chunks': 10         # Количество параллельных задач
    }
) as dag:

    # Очищаем таблицу перед вставкой новых данных
    clear_table = PythonOperator(
        task_id='clear_transfers_table',
        python_callable=lambda: PostgresHook(postgres_conn_id='airflow_db')
            .run("TRUNCATE TABLE logistic.transfers RESTART IDENTITY
CASCADE;")
    )

    # Генерируем данные параллельно в нескольких задачах
    with TaskGroup("generate_flight_chunks") as generate_chunks:
        for i in range(dag.params['num_chunks']):
            PythonOperator(
                task_id=f'generate_chunk_{i}',
                python_callable=generate_transfers_chunk,
                templates_dict={
                    'min_date': '{{ macros.datetime(2020, 1, 1).strftime("%Y-%m-%d")
}}',
                    'max_date': '{{ ds }}'
```

```
            }
        )

    # Определяем порядок выполнения задач
    clear_table >> generate_chunks
```

Листинг 3 – DAG init_schema_airline: создание схемы logistic, таблицы исходных данных logistic.transfers и таблицы метрик logistic.daily_flights_count средствами PostgresOperator.

```
from datetime import datetime
from airflow import DAG
from airflow.providers.postgres.operators.postgres import PostgresOperator

with DAG(
    'init_schema_airline',
    start_date=datetime(2023, 1, 1),
    schedule_interval='@once',
    catchup=False,
    description='Создание схемы и таблиц для данных о авиаперевозках'
) as dag:

    # Создаём схему и основную таблицу рейсов
    create_transfers_table = PostgresOperator(
        task_id='create_transfers_table',
        postgres_conn_id='airflow_db',
        sql="'
        -- Создаём схему для логистических данных
        CREATE SCHEMA IF NOT EXISTS logistic;

        -- Создаём таблицу рейсов
```

```
    -- Важно: поле "from" является зарезервированным словом SQL, поэтому берётся в кавычки
    CREATE TABLE IF NOT EXISTS logistic.transfers (
        id BIGINT PRIMARY KEY,
        "from" TEXT NOT NULL,    -- Аэропорт отправления
        "to" TEXT NOT NULL,      -- Аэропорт прибытия
        departure_date DATE NOT NULL,  -- Дата отправления
        arriving_date DATE NOT NULL    -- Дата прибытия
    );
    '''
)

# Создаём таблицу для хранения метрик
create_metrics_table = PostgresOperator(
    task_id='create_metrics_table',
    postgres_conn_id='airflow_db',
    sql='''
    -- Таблица для хранения ежедневной статистики по рейсам
    CREATE TABLE IF NOT EXISTS logistic.daily_flights_count (
        flight_date DATE PRIMARY KEY,  -- Дата рейса
        flight_count INTEGER NOT NULL  -- Количество рейсов
    );
    '''
)

# Определяем порядок выполнения задач
create_transfers_table >> create_metrics_table
```

Листинг 4 – Dockerfile: сборка кастомного образа Airflow, установка системных зависимостей и Python-зависимостей (включая провайдер Postgres) для выполнения DAG, использующих PostgresOperator/PostgresHook.

```
FROM apache/airflow:2.10.5

# Установка системных зависимостей (только если они необходимы для компиляции)
USER root
RUN apt-get update && apt-get install -y --no-install-recommends \
    gcc \
    && rm -rf /var/lib/apt/lists/*

# Возврат к пользователю airflow
USER airflow

# Копирование и установка Python-зависимостей
COPY requirements.txt .
RUN pip install --no-cache-dir "apache-airflow==${AIRFLOW_VERSION}" -r requirements.txt
```

Листинг 5 – docker-compose.yaml: конфигурация окружения Airflow (webserver/scheduler/worker/triggerer) с CeleryExecutor, Redis и Postgres, подключение volume для DAG/логов и базовая инициализация Airflow.

```
# Licensed to the Apache Software Foundation (ASF) under one
# or more contributor license agreements.  See the NOTICE file
# distributed with this work for additional information
# regarding copyright ownership.  The ASF licenses this file
# to you under the Apache License, Version 2.0 (the
# "License"); you may not use this file except in compliance
# with the License.  You may obtain a copy of the License at
#
#   http://www.apache.org/licenses/LICENSE-2.0
#
```

```
# Unless required by applicable law or agreed to in writing,
# software distributed under the License is distributed on an
# "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
# KIND, either express or implied.  See the License for the
# specific language governing permissions and limitations
# under the License.
#

# Basic Airflow cluster configuration for CeleryExecutor with Redis and PostgreSQL.
#
# WARNING: This configuration is for local development. Do not use it in a production deployment.
#
# This configuration supports basic configuration using environment variables or an .env file
# The following variables are supported:
#
# AIRFLOW_IMAGE_NAME           - Docker image name used to run Airflow.
#                                Default: apache/airflow:2.10.5
# AIRFLOW_UID                  - User ID in Airflow containers
#                                Default: 50000
# AIRFLOW_PROJ_DIR             - Base path to which all the files will be volumed.
#                                Default: .
# Those configurations are useful mostly in case of standalone testing/running Airflow in test/try-out mode
#
```

```
# _AIRFLOW_WWW_USER_USERNAME   - Username for the administrator account (if requested).
#                              Default: airflow
# _AIRFLOW_WWW_USER_PASSWORD   - Password for the administrator account (if requested).
#                              Default: airflow
# _PIP_ADDITIONAL_REQUIREMENTS - Additional PIP requirements to add when starting all containers.
#                              Use this option ONLY for quick checks. Installing requirements at container
#                              startup is done EVERY TIME the service is started.
#                              A better way is to build a custom image or extend the official image
#                              as described in https://airflow.apache.org/docs/docker-stack/build.html.
#                              Default: ''
#
# Feel free to modify this file to suit your needs.
---
x-airflow-common:
  &airflow-common
  # In order to add custom dependencies or upgrade provider packages you can use your extended image.
  # Comment the image line, place your Dockerfile in the directory where you placed the docker-compose.yaml
  # and uncomment the "build" line below, Then run `docker-compose build` to build the images.
  # image: ${AIRFLOW_IMAGE_NAME:-apache/airflow:2.10.5}
  build: .
```

```
  environment:
    &airflow-common-env
    AIRFLOW__CORE__EXECUTOR: CeleryExecutor
    AIRFLOW__DATABASE__SQL_ALCHEMY_CONN:
postgresql+psycopg2://airflow:airflow@postgres/airflow
    AIRFLOW__CELERY__RESULT_BACKEND:
db+postgresql://airflow:airflow@postgres/airflow
    AIRFLOW__CELERY__BROKER_URL: redis://:@redis:6379/0
    AIRFLOW__CORE__FERNET_KEY: ''
    AIRFLOW__CORE__DAGS_ARE_PAUSED_AT_CREATION: 'true'
    AIRFLOW__CORE__LOAD_EXAMPLES: 'true'
    AIRFLOW__API__AUTH_BACKENDS:
'airflow.api.auth.backend.basic_auth,airflow.api.auth.backend.session'
    # yamllint disable rule:line-length
    # Use simple http server on scheduler for health checks
    # See https://airflow.apache.org/docs/apache-airflow/stable/administration-
and-deployment/logging-monitoring/check-health.html#scheduler-health-check-
server
    # yamllint enable rule:line-length
    AIRFLOW__SCHEDULER__ENABLE_HEALTH_CHECK: 'true'
    # WARNING: Use _PIP_ADDITIONAL_REQUIREMENTS option ONLY
for a quick checks
    # for other purpose (development, test and especially production usage)
build/extend Airflow image.
    _PIP_ADDITIONAL_REQUIREMENTS:
${_PIP_ADDITIONAL_REQUIREMENTS:-}
    # The following line can be used to set a custom config file, stored in the local
config folder
```

```
    # If you want to use it, outcomment it and replace airflow.cfg with the name of your config file
    # AIRFLOW_CONFIG: '/opt/airflow/config/airflow.cfg'
  volumes:
    - ${AIRFLOW_PROJ_DIR:-.}/dags:/opt/airflow/dags
    - ${AIRFLOW_PROJ_DIR:-.}/logs:/opt/airflow/logs
    - ${AIRFLOW_PROJ_DIR:-.}/config:/opt/airflow/config
    - ${AIRFLOW_PROJ_DIR:-.}/plugins:/opt/airflow/plugins
  user: "${AIRFLOW_UID:-50000}:0"
  depends_on:
    &airflow-common-depends-on
    redis:
      condition: service_healthy
    postgres:
      condition: service_healthy

services:
  postgres:
    image: postgres:13
    environment:
      POSTGRES_USER: airflow
      POSTGRES_PASSWORD: airflow
      POSTGRES_DB: airflow
    volumes:
      - postgres-db-volume:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD", "pg_isready", "-U", "airflow"]
      interval: 10s
      retries: 5
```

```
    start_period: 5s
  restart: always

redis:
  # Redis is limited to 7.2-bookworm due to licencing change
  # https://redis.io/blog/redis-adopts-dual-source-available-licensing/
  image: redis:7.2-bookworm
  expose:
    - 6379
  healthcheck:
    test: ["CMD", "redis-cli", "ping"]
    interval: 10s
    timeout: 30s
    retries: 50
    start_period: 30s
  restart: always

airflow-webserver:
  <<: *airflow-common
  command: webserver
  ports:
    - "8080:8080"
  healthcheck:
    test: ["CMD", "curl", "--fail", "http://localhost:8080/health"]
    interval: 30s
    timeout: 10s
    retries: 5
    start_period: 30s
  restart: always
```

```
    depends_on:
      <<: *airflow-common-depends-on
      airflow-init:
        condition: service_completed_successfully

  airflow-scheduler:
    <<: *airflow-common
    command: scheduler
    healthcheck:
      test: ["CMD", "curl", "--fail", "http://localhost:8974/health"]
      interval: 30s
      timeout: 10s
      retries: 5
      start_period: 30s
    restart: always
    depends_on:
      <<: *airflow-common-depends-on
      airflow-init:
        condition: service_completed_successfully

  airflow-worker:
    <<: *airflow-common
    command: celery worker
    healthcheck:
      # yamllint disable rule:line-length
      test:
        - "CMD-SHELL"
        - 'celery --app airflow.providers.celery.executors.celery_executor.app
inspect ping -d "celery@$${HOSTNAME}" || celery --app
```

```
airflow.executors.celery_executor.app inspect ping -d
"celery@$${HOSTNAME}"'
      interval: 30s
      timeout: 10s
      retries: 5
      start_period: 30s
    environment:
      <<: *airflow-common-env
      # Required to handle warm shutdown of the celery workers properly
      # See https://airflow.apache.org/docs/docker-stack/entrypoint.html#signal-
propagation
      DUMB_INIT_SETSID: "0"
    restart: always
    depends_on:
      <<: *airflow-common-depends-on
      airflow-init:
        condition: service_completed_successfully

  airflow-triggerer:
    <<: *airflow-common
    command: triggerer
    healthcheck:
      test: ["CMD-SHELL", 'airflow jobs check --job-type TriggererJob --
hostname "$${HOSTNAME}"']
      interval: 30s
      timeout: 10s
      retries: 5
      start_period: 30s
    restart: always
```

```
    depends_on:
      <<: *airflow-common-depends-on
      airflow-init:
        condition: service_completed_successfully

  airflow-init:
    <<: *airflow-common
    entrypoint: /bin/bash
    # yamllint disable rule:line-length
    command:
      - -c
      - |
        if [[ -z "${AIRFLOW_UID}" ]]; then
          echo
          echo -e "\033[1;33mWARNING!!!: AIRFLOW_UID not set!\e[0m"
          echo "If you are on Linux, you SHOULD follow the instructions below to set "
          echo "AIRFLOW_UID environment variable, otherwise files will be owned by root."
          echo "For other operating systems you can get rid of the warning with manually created .env file:"
          echo "    See: https://airflow.apache.org/docs/apache-airflow/stable/howto/docker-compose/index.html#setting-the-right-airflow-user"
          echo
        fi
        one_meg=1048576
        mem_available=$$(($$(getconf _PHYS_PAGES) * $$(getconf PAGE_SIZE) / one_meg))
        cpus_available=$$(grep -cE 'cpu[0-9]+' /proc/stat)
```

```
    disk_available=$$(df / | tail -1 | awk '{print $$4}')
    warning_resources="false"
    if (( mem_available < 4000 )) ; then
      echo
      echo -e "\033[1;33mWARNING!!!: Not enough memory available for Docker.\e[0m"
      echo "At least 4GB of memory required. You have $$(numfmt --to iec $$((mem_available * one_meg)))"
      echo
      warning_resources="true"
    fi
    if (( cpus_available < 2 )); then
      echo
      echo -e "\033[1;33mWARNING!!!: Not enough CPUS available for Docker.\e[0m"
      echo "At least 2 CPUs recommended. You have $${cpus_available}"
      echo
      warning_resources="true"
    fi
    if (( disk_available < one_meg * 10 )); then
      echo
      echo -e "\033[1;33mWARNING!!!: Not enough Disk space available for Docker.\e[0m"
      echo "At least 10 GBs recommended. You have $$(numfmt --to iec $$((disk_available * 1024 )))"
      echo
      warning_resources="true"
    fi
    if [[ $${warning_resources} == "true" ]]; then
```

```
        echo
        echo -e "\033[1;33mWARNING!!!: You have not enough resources to run Airflow (see above)!\e[0m"
        echo "Please follow the instructions to increase amount of resources available:"
        echo "   https://airflow.apache.org/docs/apache-airflow/stable/howto/docker-compose/index.html#before-you-begin"
        echo
      fi
      mkdir -p /sources/logs /sources/dags /sources/plugins
      chown -R "${AIRFLOW_UID}:0" /sources/{logs,dags,plugins}
      exec /entrypoint airflow version
  # yamllint enable rule:line-length
  environment:
    <<: *airflow-common-env
    _AIRFLOW_DB_MIGRATE: 'true'
    _AIRFLOW_WWW_USER_CREATE: 'true'
    _AIRFLOW_WWW_USER_USERNAME: ${_AIRFLOW_WWW_USER_USERNAME:-airflow}
    _AIRFLOW_WWW_USER_PASSWORD: ${_AIRFLOW_WWW_USER_PASSWORD:-airflow}
    _PIP_ADDITIONAL_REQUIREMENTS: ''
  user: "0:0"
  volumes:
    - ${AIRFLOW_PROJ_DIR:-.}:/sources

airflow-cli:
  <<: *airflow-common
  profiles:
```

```
    - debug
  environment:
    <<: *airflow-common-env
    CONNECTION_CHECK_MAX_COUNT: "0"
  # Workaround for entrypoint issue. See:
https://github.com/apache/airflow/issues/16252
  command:
    - bash
    - -c
    - airflow


 # You can enable flower by adding "--profile flower" option e.g. docker-
compose --profile flower up
 # or by explicitly targeted on the command line e.g. docker-compose up flower.
 # See: https://docs.docker.com/compose/profiles/
 flower:
  <<: *airflow-common
  command: celery flower
  profiles:
    - flower
  ports:
    - "5555:5555"
  healthcheck:
    test: ["CMD", "curl", "--fail", "http://localhost:5555/"]
    interval: 30s
    timeout: 10s
    retries: 5
    start_period: 30s
  restart: always
```

```
    depends_on:
      <<: *airflow-common-depends-on
      airflow-init:
        condition: service_completed_successfully


volumes:
  postgres-db-volume:
```

Листинг 6 – requirements.txt: список Python-зависимостей проекта (провайдер Airflow для Postgres).

```
apache-airflow-providers-postgres==5.9.0
```

Листинг 7 – .env: переменные окружения для docker-compose (идентификатор пользователя AIRFLOW_UID для корректных прав на файлы в volume).

```
AIRFLOW_UID=1000
```