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»

КАФЕДРА «Информационная безопасность»

ОТЧЕТ

по лабораторной работе № 5

по курсу «Искусственный интеллект»

на тему: «Python + Airflow, оркестрация данных»

Вариант № 6

Студент ИУ8-13М (Группа) _____ (Подпись, дата) Савватеев А. Э. (И. О. Фамилия)

Преподаватель _____ (Подпись, дата) Зотов М. В. (И. О. Фамилия)

2025 г.

СОДЕРЖАНИЕ

1 ЦЕЛЬ РАБОТЫ

Реализовать систему обновления данных посредством системы-оркестрации Apache Airflow для автоматизации ETL-процессов обработки данных о доставке заказов.

2 ТРЕБОВАНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

- Создан DAG, который осуществляет инициализацию схемы и таблиц внутри метабазы данных Airflow
- Создан DAG, который осуществляет загрузку данных в созданную таблицу из CSV-файла
- Создан DAG, который осуществляет расчет метрик согласно заданию варианта
- После выполнения расчета, в созданной таблице для метрик можно ознакомиться с показателями
- Все DAG-файлы размещены в директории `students_folder/savandr/lab_5/dags/`

3 ПОСТАНОВКА ЗАДАЧИ

3.1 Вариант 6

Условия: Компания занимается доставкой товаров. История изменений статусов заказов хранится в таблице `logistics.deliveries` (`order_id BIGINT`, `status TEXT`, `update_date DATE`).

Задача: Вывести количество заказов, получивших статус `'delivered'`, за каждый день.

3.2 Структура базы данных

Для решения задачи необходимо создать две таблицы в схеме `logistics`:

- `logistics.deliveries` — основная таблица с данными о доставках:
  - `order_id` — уникальный идентификатор заказа (PRIMARY KEY)
  - `status` — статус заказа (delivered, shipped, pending, cancelled, returned)
  - `update_date` — дата обновления статуса
- `logistics.daily_delivered_stats` — таблица с агрегированными метриками:
  - `report_date` — дата отчета (PRIMARY KEY)
  - `delivered_count` — количество доставленных заказов за день

4 ХОД РАБОТЫ

4.1 Инфраструктура

Для выполнения лабораторной работы была развернута инфраструктура Apache Airflow с использованием Docker Compose. Конфигурация включает следующие сервисы:

- **PostgreSQL** — база данных для метаданных Airflow и хранения данных приложения
- **Redis** — брокер сообщений для CeleryExecutor
- **Airflow Webserver** — веб-интерфейс для управления DAG
- **Airflow Scheduler** — планировщик задач
- **Airflow Worker** — исполнитель задач
- **Airflow Triggerer** — обработчик триггеров для отложенных задач

Файл `docker-compose.yml` содержит полную конфигурацию всех сервисов с необходимыми переменными окружения и volume-монтированием директории `dags/`.

4.2 DAG 1: Инициализация схемы (init_schema.py)

Первый DAG создает необходимую структуру базы данных. Он включает две последовательные задачи:

4.2.1 Задача create_schema

Создает схему `logistics` в базе данных PostgreSQL:

```
CREATE SCHEMA IF NOT EXISTS logistics;
```

4.2.2 Задача create_table

Создает две таблицы в схеме `logistics`:

```
CREATE TABLE IF NOT EXISTS logistics.deliveries (
    order_id SERIAL PRIMARY KEY,
    status TEXT NOT NULL,
```

```
    update_date DATE NOT NULL
);

CREATE TABLE IF NOT EXISTS logistics.daily_delivered_stats (
    report_date DATE PRIMARY KEY,
    delivered_count INT NOT NULL
);
```

4.2.3 Параметры DAG

- `dag_id`: `init_schema`
- `schedule_interval`: `@once` — выполняется однократно
- `start_date`: 2025-03-20
- `catchup`: False — не запускать пропущенные выполнения
- `tags`: ['lab5']

Задачи выполняются последовательно: сначала создается схема, затем таблицы.

4.3 DAG 2: Загрузка данных (init_data.py)

Второй DAG загружает тестовые данные из CSV-файла в таблицу `logistics.deliveries`.

4.3.1 Задача truncate_table

Очищает таблицу перед загрузкой новых данных для обеспечения идемпотентности:

```
TRUNCATE TABLE logistics.deliveries;
```

4.3.2 Задача insert_data

Читает данные из CSV-файла `/opt/airflow/dags/deliveries.csv` и вставляет их в таблицу:

```
with open(CSV_FILE_PATH, 'r') as file:
    reader = csv.reader(file)
    next(reader)
```

```
    for row in reader:
        status = row[0]
        update_date = row[1]
        rows_to_insert.append({
            'status': status,
            'update_date': update_date
        })

# п я я п п п п  п п п п я я
INSERT INTO logistics.deliveries (status, update_date)
VALUES (:status, :update_date)
```

4.3.3 Структура CSV-файла

Файл `deliveries.csv` содержит 2002 записи со следующей структурой:

- `status` — статус заказа (delivered, shipped, pending, cancelled, returned)
- `update_date` — дата обновления статуса в формате YYYY-MM-DD

Пример данных:

```
status,update_date
returned,2025-10-18
shipped,2025-09-08
delivered,2025-10-30
delivered,2025-10-24
```

4.3.4 Параметры DAG

- `dag_id`: `init_data`
- `schedule_interval`: `@once`
- `start_date`: 2025-03-20
- `catchup`: False

4.4 DAG 3: Расчет метрик (collect_data.py)

Третий DAG выполняет инкрементальный расчет метрик — подсчитывает количество доставленных заказов за каждый день.

4.4.1 Задача collect_data

Использует технику UPSERT (INSERT ... ON CONFLICT) для инкрементального обновления статистики:

```
INSERT INTO logistics.daily_delivered_stats(
    report_date,
    delivered_count
)
SELECT :run_date, COUNT(order_id)
FROM logistics.deliveries
WHERE status = 'delivered'
  AND update_date = :run_date
ON CONFLICT (report_date)
DO UPDATE SET delivered_count = EXCLUDED.delivered_count
```

Этот подход обеспечивает:

- Идемпотентность — повторное выполнение не создаст дубликаты
- Инкрементальность — обрабатывается только дата текущего запуска
- Эффективность — обновление происходит за один запрос

4.4.2 Параметры DAG

- `dag_id`: `collect_data`
- `schedule_interval`: `@daily` — выполняется ежедневно
- `start_date`: 2025-03-20
- `catchup`: True — выполнить все пропущенные запуски
- `op_kwargs`: `{{"run_date": "{{ ds }}"}}` — передача даты выполнения

Параметр `catchup=True` позволяет DAG обработать все даты с `start_date` до текущей даты, что важно для заполнения исторических данных.

4.4.3 Логирование

Задача выводит информацию о процессе обработки:

- Дату обработки
- Количество доставленных заказов за эту дату

4.5 Общая архитектура решения

Три DAG образуют полноценный ETL-pipeline:

1. **Extract (init_data)** — извлечение данных из CSV-файла
2. **Transform (collect_data)** — агрегация данных по дням с фильтрацией по статусу
3. **Load (init_data, collect_data)** — загрузка результатов в таблицы базы данных

Последовательность выполнения:

1. Запуск `init_schema` — создание структуры БД (однократно)
2. Запуск `init_data` — загрузка исходных данных (однократно)
3. Запуск `collect_data` — ежедневный расчет метрик (регулярно)

4.6 Особенности реализации

4.6.1 Использование SQLAlchemy

Все DAG используют SQLAlchemy для взаимодействия с PostgreSQL:

- Создание подключения через `create_engine`
- Использование параметризованных запросов через `text()` для защиты от SQL-инъекций
- Явное управление соединениями через context manager

4.6.2 Конфигурация ARGS

Все DAG используют единый набор параметров:

```
ARGS = {
    "owner": "bmstu",
    "email": ['wzomzot@hop.ru', '1@mail.ru'],
    "email_on_failure": True,
    "email_on_retry": False,
    "start_date": datetime(2025, 3, 20),
    "pool": "default_pool",
    "queue": "default"
}
```

Это обеспечивает единообразие и упрощает поддержку.

4.6.3 Идемпотентность

Все операции спроектированы как идемпотентные:

- `CREATE TABLE IF NOT EXISTS`
- `CREATE SCHEMA IF NOT EXISTS`
- `TRUNCATE` перед загрузкой
- `INSERT ... ON CONFLICT` для метрик

5 РЕЗУЛЬТАТЫ РАБОТЫ

5.1 Запуск DAG

После развертывания инфраструктуры и размещения DAG-файлов в директории `dags/`, все три DAG стали доступны в веб-интерфейсе Airflow.

5.1.1 Выполнение init_schema

DAG успешно создал схему `logistics` и две таблицы:

- `logistics.deliveries`
- `logistics.daily_delivered_stats`

5.1.2 Выполнение init_data

DAG загрузил 2001 запись (без учета заголовка) из CSV-файла в таблицу `logistics.deliveries`.

5.1.3 Выполнение collect_data

DAG обработал все даты с 2025-03-20 по текущую дату благодаря параметру `catchup=True`. Для каждой даты была рассчитана статистика количества доставленных заказов.

5.2 Анализ результатов

После выполнения всех DAG в таблице `logistics.daily_delivered_stats` содержится агрегированная информация о количестве доставок по дням. Данные можно проанализировать следующим SQL-запросом:

```
SELECT report_date, delivered_count
FROM logistics.daily_delivered_stats
ORDER BY report_date;
```

Это позволяет:

- Отслеживать динамику доставок
- Выявлять пиковые дни

- Планировать ресурсы службы доставки
- Анализировать эффективность логистики

5.3 Мониторинг

Веб-интерфейс Airflow предоставляет полную информацию о выполнении DAG:

- Статус выполнения каждой задачи
- Логи выполнения
- Длительность выполнения
- Граф зависимостей задач
- История запусков

6 ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы была успешно реализована система оркестрации данных с использованием Apache Airflow. Разработаны три DAG, которые обеспечивают полный цикл обработки данных:

1. Инициализация структуры базы данных
2. Загрузка исходных данных из CSV-файла
3. Регулярный расчет бизнес-метрик

Основные достижения:

- Созданы идемпотентные и надежные DAG
- Реализован инкрементальный подход к обработке данных
- Обеспечена масштабируемость решения
- Настроен мониторинг и логирование
- Применены best practices работы с Airflow

Разработанная система может быть легко расширена для обработки дополнительных метрик, интеграции с другими источниками данных и добавления новых этапов обработки.

Все требования лабораторной работы выполнены. Код размещен в директории `students_folder/savandr/lab_5/dags/`.

ПРИЛОЖЕНИЕ А

Листинг А.1 – DAG инициализации схемы базы данных

```
from airflow import DAG
from datetime import datetime

from airflow.operators.python import PythonOperator

ARGS = {
    "owner": "bmstu",
    "email": ['wzomzot@hop.ru', '1@mail.ru'],
    "email_on_failure": True,
    "email_on_retry": False,
    "start_date": datetime(2025, 3, 20),  # важный атрибут
    "pool": "default_pool",
    "queue": "default"
}


def create_table():
    import pandas as pd
    from sqlalchemy import create_engine, text

    # Подключение к PostgreSQL
    engine =
       create_engine("postgresql+psycopg2://airflow:airflow@postgres/a

    # SQL-запрос для создания таблицы
    create_table_query = """
    CREATE TABLE IF NOT EXISTS logistics.deliveries (
        order_id SERIAL PRIMARY KEY,
        status TEXT NOT NULL,
        update_date DATE NOT NULL
    );

    CREATE TABLE IF NOT EXISTS logistics.daily_delivered_stats (
        report_date DATE PRIMARY KEY,
        delivered_count INT NOT NULL
    );
    """
```

```
37
38      # Выполняем запрос
39      with engine.connect() as conn:
40          conn.execute(text(create_table_query))
41
42      print("Таблица 'deliveries' успешно создана!")
43
44  def create_schema():
45      import pandas as pd
46      from sqlalchemy import create_engine, text
47
48      # Подключение к PostgreSQL
49      engine =
           create_engine("postgresql+psycopg2://airflow:airflow@postgres/a
50
51      # SQL-запрос для создания таблицы
52      create_schema_query = """
53      CREATE SCHEMA IF NOT EXISTS logistics;
54      """
55
56      # Выполняем запрос
57      with engine.connect() as conn:
58          conn.execute(text(create_schema_query))
59
60      print("Схема 'logistics' успешно создана!")
61
62
63  with DAG(dag_id='init_schema',  # важный атрибут
64           default_args=ARGS,
65           schedule_interval='@once',
66           max_active_runs=1,
67           start_date=datetime(2025, 3, 20),
68           catchup=False,
69           tags=['lab5']) as dag:
70      t_create_schema = PythonOperator(
71          task_id='create_schema',
72          dag=dag,
73          python_callable=create_schema
74      )
75
76      t_create_table = PythonOperator(
```

```
77          task_id='create_table',
78          dag=dag,
79          python_callable=create_table
80      )
81
82      run = t_create_schema >> t_create_table
```

Листинг А.2 – DAG загрузки данных из CSV

```
1  import csv
2
3  from airflow import DAG
4  from datetime import datetime
5
6  from airflow.operators.python import PythonOperator
7
8  ARGS = {
9      "owner": "bmstu",
10     "email": ['wzomzot@hop.ru', '1@mail.ru'],
11     "email_on_failure": True,
12     "email_on_retry": False,
13     "start_date": datetime(2025, 3, 20),  # важный атрибут
14     "pool": "default_pool",
15     "queue": "default"
16 }
17
18 CSV_FILE_PATH = '/opt/airflow/dags/deliveries.csv'
19
20 def truncate_table():
21     from sqlalchemy import create_engine, text
22
23     # Подключение к PostgreSQL
24     engine =
          create_engine("postgresql+psycopg2://airflow:airflow@postgres/a
25
26     # SQL-запрос для создания таблицы
27     truncate_table_query = """
28         TRUNCATE TABLE logistics.deliveries;
29     """
30
31     # Выполняем запрос
32     with engine.connect() as conn:
33         conn.execute(text(truncate_table_query))
```

```
34
35      print("Таблица 'deliveries' очищена!")
36
37
38  def init_deliveries():
39      from sqlalchemy import create_engine, text
40
41      # Подключение к PostgreSQL
42      engine =
           create_engine("postgresql+psycopg2://airflow:airflow@postgres/a
43
44      rows_to_insert = []
45      with open(CSV_FILE_PATH, 'r') as file:
46          reader = csv.reader(file)
47          next(reader)  # Пропускаем заголовок
48
49          for row in reader:
50              status = row[0]
51              update_date = row[1]
52
53              rows_to_insert.append({'status': status,
                   'update_date': update_date})
54
55      # Выполняем запрос
56      insert_data_query = """
57          INSERT INTO logistics.deliveries (status, update_date)
               VALUES
58          (:status, :update_date)
59      """
60      with engine.connect() as conn:
61          conn.execute(text(insert_data_query), rows_to_insert)
62
63
64  print("Таблица 'deliveries' наполнена заказами!")
65
66
67  with DAG(dag_id='init_data',  # важный атрибут
68           default_args=ARGS,
69           schedule_interval='@once',
70           max_active_runs=1,
71           start_date=datetime(2025, 3, 20),
```

```
72          catchup=False,
73          tags=['lab5']) as dag:
74      t_truncate_table = PythonOperator(
75          task_id='truncate_table',
76          dag=dag,
77          python_callable=truncate_table
78      )
79
80      t_insert_data = PythonOperator(
81          task_id='insert_data',
82          dag=dag,
83          python_callable=init_deliveries
84      )
85
86      run = t_truncate_table >> t_insert_data
```

Листинг А.3 – DAG расчета метрик доставки

```
1  # /dags/collect_data.py
2  from airflow import DAG
3  from datetime import datetime
4
5  from airflow.operators.python import PythonOperator
6
7  ARGS = {
8      "owner": "bmstu",
9      "email": ['wzomzot@hop.ru', '1@mail.ru'],
10     "email_on_failure": True,
11     "email_on_retry": False,
12     "start_date": datetime(2025, 3, 20),  # важный атрибут
13     "pool": "default_pool",
14     "queue": "default"
15 }
16
17
18 def calculate_and_save_metrics(run_date: str):
19     from sqlalchemy import create_engine, text
20
21     print(f"Обработка метрик за дату: {run_date}")
22
23     # Подключение к PostgreSQL
24     engine =
          create_engine("postgresql+psycopg2://airflow:airflow@postgres/a
```

```
25
26      # SQL-запрос для инкрементального обновления с
            использованием UPSERT
27      upsert_data_query = """
28          INSERT INTO logistics.daily_delivered_stats(report_date,
                delivered_count)
29          SELECT :run_date, COUNT(order_id)
30          FROM logistics.deliveries
31          WHERE status = 'delivered'
32          AND update_date = :run_date
33          ON CONFLICT (report_date)
34          DO UPDATE SET delivered_count = EXCLUDED.delivered_count
35      """
36
37      # Выполняем запрос
38      with engine.connect() as conn:
39          result = conn.execute(text(upsert_data_query),
                {"run_date": run_date})
40
41      # Получаем количество обработанных записей для логирования
42      check_count_query = """
43          SELECT delivered_count FROM
                logistics.daily_delivered_stats
44          WHERE report_date = :run_date
45      """
46
47      with engine.connect() as conn:
48          count_result = conn.execute(text(check_count_query),
                {"run_date": run_date}).fetchone()
49          delivered_count = count_result[0] if count_result else 0
50
51      print(f"Метрики за {run_date} успешно обновлены. Доставлено
            заказов: {delivered_count}")
52
53
54  with DAG(dag_id='collect_data',  # важный атрибут
55           default_args=ARGS,
56           schedule_interval='@daily',
57           max_active_runs=1,
58           start_date=datetime(2025, 3, 20),
59           catchup=True,
```

```
60          tags=['lab5']) as dag:
61      t_process_metrics = PythonOperator(
62          task_id='collect_data',
63          dag=dag,
64          python_callable=calculate_and_save_metrics,
65          op_kwargs={"run_date": "{{ ds }}"}
66      )
67
68      run = t_process_metrics
```