

Департамент образования и науки города Москвы
Государственное автономное образовательное учреждение высшего
образования города Москвы
«Московский городской педагогический университет»
Институт цифрового образования
Департамент информатики, управления и технологий

ДИСЦИПЛИНА:

Проектный практикум по разработке ETL-решений

Лабораторная работа 5.2

Разработка алгоритмов для трансформации данных. Airflow DAG

Выполнила: Шведова С.С., группа: АДЭУ-211

Преподаватель: Босенко Т.М.

Москва

2025

Задачи:

5.1.1. Развернуть Конфигурация репозиторий ВМ в VirtualBox.

5.1.2. Клонировать на ПК задание Бизнес-кейс «Rocket» в домашний каталог ВМ.

5.1.3. Запустить контейнер с кейсом, изучить основные элементы DAG в Apache Airflow.

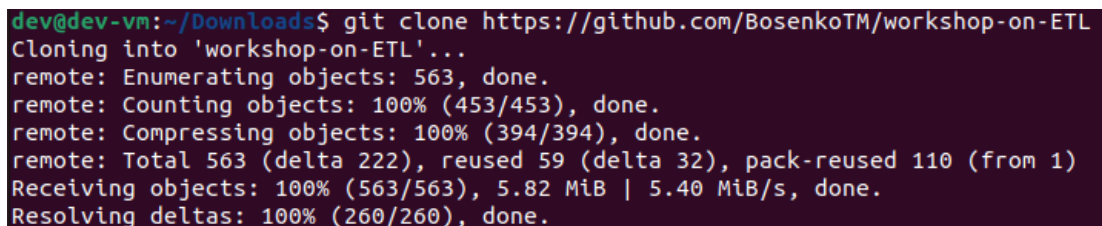
5.1.4. Создать исполняемый файл с расширением .sh, который автоматизирует выгрузку данных из контейнера в основную ОС данных, полученные в результате работы DAG в Apache Airflow.

5.1.5. Спроектировать верхнеуровневую архитектуру аналитического решения задания Бизнес-кейса «Rocket» в draw.io.

5.1.6. Спроектировать архитектуру DAG Бизнес-кейса «Rocket» в draw.io.

5.1.7. Построить диаграмму Ганта работы DAG в Apache Airflow.

Для начала надо клонировать репозиторий (рисунок 1).



```
dev@dev-vm: ~/Downloads$ git clone https://github.com/BosenkoTM/workshop-on-ETL
Cloning into 'workshop-on-ETL'...
remote: Enumerating objects: 563, done.
remote: Counting objects: 100% (453/453), done.
remote: Compressing objects: 100% (394/394), done.
remote: Total 563 (delta 222), reused 59 (delta 32), pack-reused 110 (from 1)
Receiving objects: 100% (563/563), 5.82 MiB | 5.40 MiB/s, done.
Resolving deltas: 100% (260/260), done.
```

Рисунок 1. Клонирование репозитория

На рисунке 2 показан даг, который нужен

```

1 import json
2 import pathlib
3
4 import airflow
5 import requests
6 import requests.exceptions as requests_exceptions
7 from airflow import DAG
8 from airflow.operators.bash import BashOperator
9 from airflow.operators.python import PythonOperator
10
11 dag = DAG(
12     dag_id="listing_2_18",
13     start_date=airflow.utils.dates.days_ago(14),
14     schedule_interval="@daily",
15 )
16
17 download_launches = BashOperator(
18     task_id="download_launches",
19     bash_command="curl -o /tmp/launches.json -L 'https://1l.thespacedevs.com/2.0.0/launch/upcoming'", # noqa: E501
20     dag=dag,
21 )
22
23
24 def _get_pictures():
25     # Ensure directory exists
26     pathlib.Path("/tmp/images").mkdir(parents=True, exist_ok=True)
27
28     # Download all pictures in launches.json
29     with open("/tmp/launches.json") as f:
30         launches = json.load(f)
31         image_urls = [launch["image"] for launch in launches["results"]]
32         for image_url in image_urls:
33             try:
34                 response = requests.get(image_url)
35                 image_filename = image_url.split("/")[-1]
36                 target_file = f"/tmp/images/{image_filename}"
37                 with open(target_file, "wb") as f:
38                     f.write(response.content)
39                 print(f"Downloaded {image_url} to {target_file}")
40             except requests_exceptions.MissingSchema:
41                 print(f"{image_url} appears to be an invalid URL.")
42             except requests_exceptions.ConnectionError:
43                 print(f"Could not connect to {image_url}.")
44
45
46 get_pictures = PythonOperator(
47     task_id="get_pictures", python_callable=_get_pictures, dag=dag
48 )
49
50 notify = BashOperator(
51     task_id="notify",
52     bash_command="echo 'There are now $(ls /tmp/images/ | wc -l) images.'",
53     dag=dag,
54 )
55
56 download_launches >> get_pictures >> notify

```

Рисунок 2. Необходимый даг

Затем нужно запустить контейнеры, это показано на рисунке 3

```

dev@dev-vm:~/Downloads/workshop-on-ETL/business_case_rocket$ docker compose up -d
WARN[0000] /home/dev/Downloads/workshop-on-ETL/business_case_rocket/docker-compose.yml: the attribute `version`
is obsolete, it will be ignored, please remove it to avoid potential confusion
[+] Running 35/35
  ✓ postgres Pulled                                16.9s
  ✓ init Pulled                                    21.2s
  ✓ webserver Pulled                               21.2s
  ✓ scheduler Pulled                               21.2s
[+] Running 6/6
  ✓ Network business_case_rocket default            Created           0.1s
  ✓ Volume "business_case_rocket_logs"              Created           0.0s
  ✓ Container business_case_rocket-postgres-1       Started           23.1s
  ✓ Container business_case_rocket-init-1           Started           0.9s
  ✓ Container business_case_rocket-webserver-1      Started           1.0s
  ✓ Container business_case_rocket-scheduler-1      Started           1.0s
dev@dev-vm:~/Downloads/workshop-on-ETL/business_case_rocket$

```

Рисунок 3. Запуск контейнеров

После захода в аирфлоу необходимо запустить даг, это показано на рисунке 4.

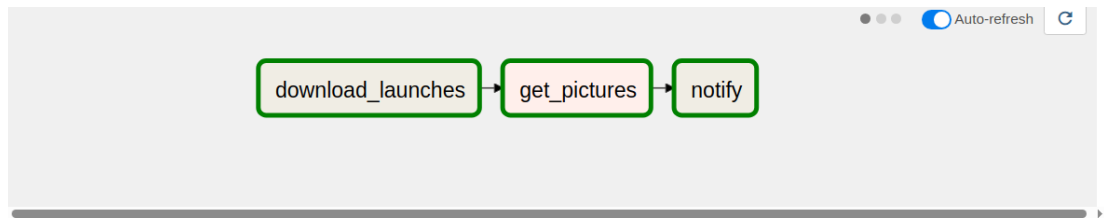


Рисунок 4. Запуск дага

На рисунке 5 показана диаграмма Ганта дага listing_2_10

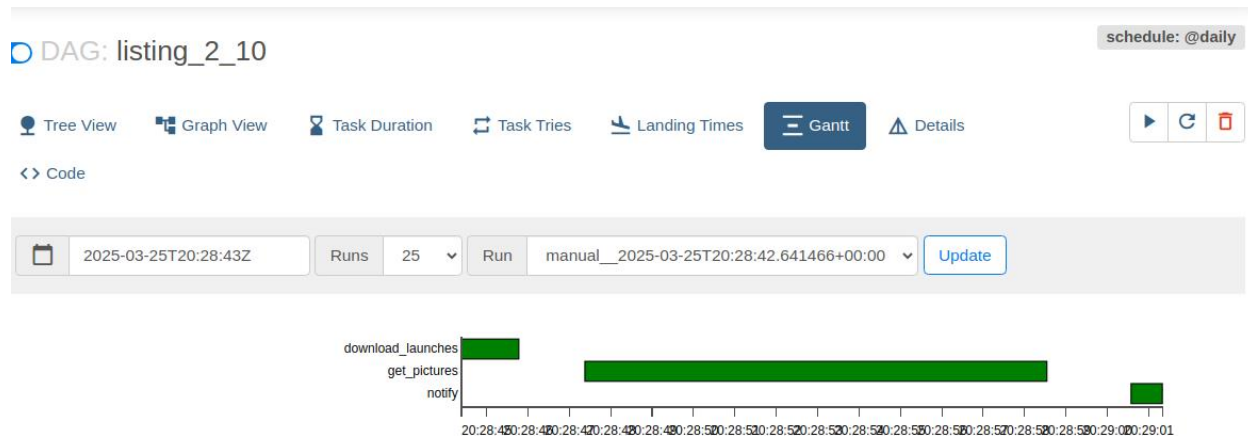


Рисунок 5. Диаграмма Ганта

На рисунке 6 показан sh файл для автоматической выгрузки данных из контейнера в основную ОС данных, полученные в результате работы DAG в Apache Airflow.

```
GNU nano 6.2 . /export_data.sh
#!/bin/bash

# Название контейнера Airflow
CONTAINER_NAME="business_case_rocket-scheduler-1"

# Путь к каталогу в контейнере, из которого нужно выгрузить данные
CONTAINER_PATH="/tmp/launches.json"
# Путь к каталогу в основной ОС, в который нужно выгрузить данные
HOST_PATH="/home/dev/Downloads/data"

# Создание каталога в основной ОС, если он не существует
mkdir -p "$HOST_PATH"

# Копирование данных из контейнера в основную ОС
docker cp "$CONTAINER_NAME": "$CONTAINER_PATH" "$HOST_PATH"

echo "Данные успешно выгружены из контейнера в основную ОС."
```

Рисунок 6. Sh файл

На рисунке 7 показано выполнение этого файла, которое успешно получилось.

```
dev@dev-vm:~/Downloads$ ./export_data.sh
Successfully copied 24.1kB to /home/dev/Downloads/data
Данные успешно выгружены из контейнера в основную ОС.
```

Рисунок 7. Выполнение Sh файла

На рисунке 8 показано, что файл скопировался в нужный каталог.

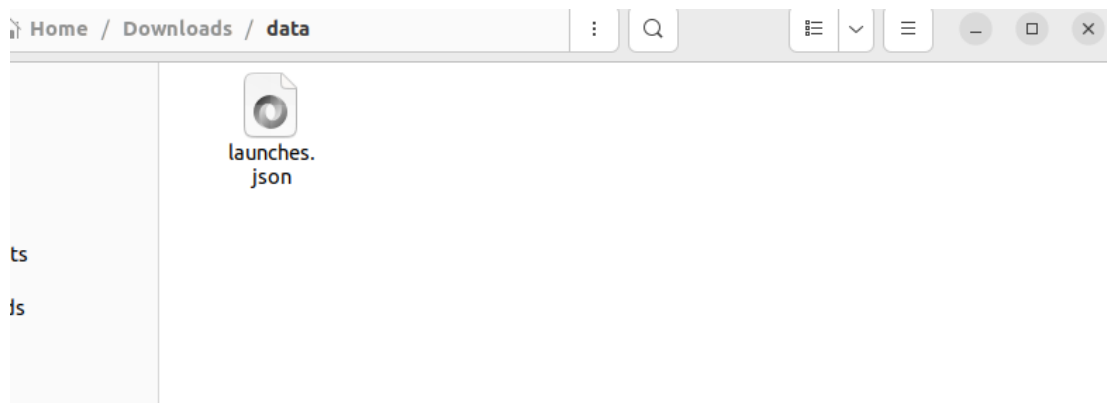


Рисунок 8. Файл скопировался

На рисунке 9 продемонстрирована верхнеуровневая архитектура.

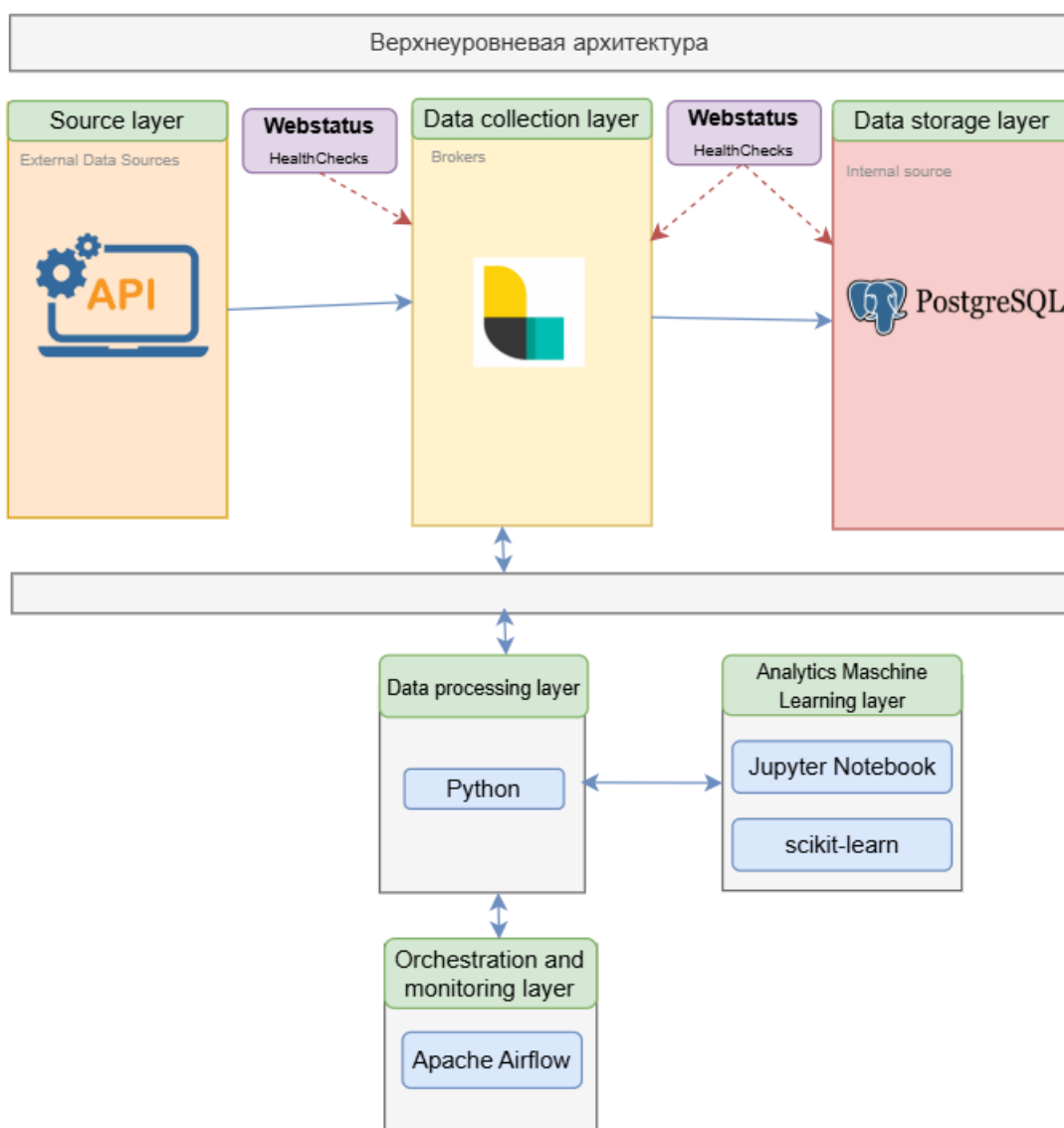


Рисунок 9. Верхнеуровневая архитектура.

1. **Слой источника данных (API Rocket):** Использование API для сбора данных — это распространенный подход.

2. **Слой сбора данных (Logstash):** Logstash — это мощный инструмент для сбора, обработки и отправки данных. Он хорошо интегрируется с различными источниками данных и может обрабатывать данные в реальном времени.

3. **Хранение данных (PostgreSQL):** PostgreSQL — это надежная реляционная база данных, которая поддерживает сложные запросы и транзакции. Она подходит для хранения структурированных данных.

4. **Обработка данных (Python):** Python обеспечивает обработку данных и является простым инструментом с множеством библиотек.

5. **Аналитика и машинное обучение (Jupyter Notebook и scikit-learn):** Jupyter Notebook предоставляет удобную среду для анализа данных и разработки моделей машинного обучения. Scikit-learn — это мощная библиотека для машинного обучения, которая поддерживает множество алгоритмов.

6. **Оркестрация и мониторинг (Apache Airflow):** Apache Airflow — это отличный инструмент для оркестрации рабочих процессов. Он позволяет планировать и управлять задачами, а также отслеживать их выполнение.

Эта архитектура позволяет эффективно собирать, хранить, обрабатывать и анализировать данные, обеспечивая гибкость и масштабируемость.

На рисунке 10 показана архитектура dag.

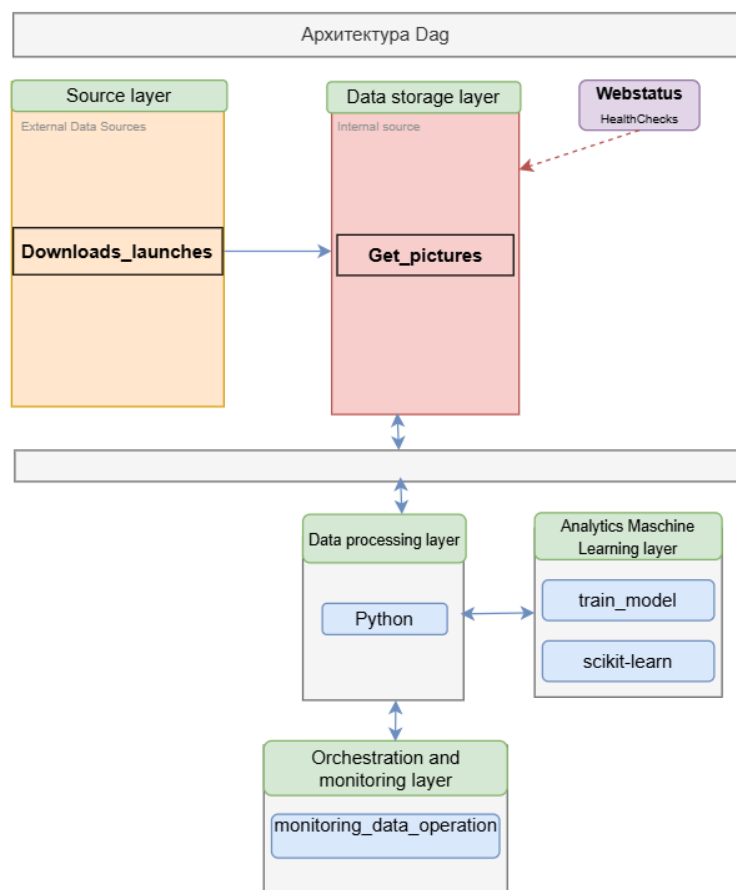


Рисунок 10. Архитектура dag

1. Слой источника данных (Downloads_launches)

Этот компонент отвечает за сбор данных. Он может представлять собой систему, которая загружает данные о запусках. Данные извлекаются и подготавливаются для передачи в следующий компонент.

2. Хранение данных (get_pictures)

Этот компонент, вероятно, отвечает за хранение изображений. После извлечения данных из Downloads_launches, они сохраняются в get_pictures.

3. Обработка данных (Python)

Python используется для обработки данных, полученных из get_pictures. Это может включать в себя очистку, преобразование и подготовку данных для анализа. С помощью библиотек, таких как Pandas или NumPy, вы можете извлекать данные из get_pictures, обрабатывать их и подготавливать для анализа.

4. Аналитика и машинное обучение (train_model и scikit-learn)

Этот компонент отвечает за обучение моделей машинного обучения с использованием scikit-learn. Он может включать в себя как обучение, так и оценку моделей. После обработки данных используется scikit-learn для создания и обучения моделей.

5. Оркестрация и мониторинг (monitoring_data_operation)

Этот компонент отвечает за управление и мониторинг всего рабочего процесса. Он может включать в себя инструменты, такие как Apache Airflow, для автоматизации задач и отслеживания их выполнения. Создается DAG, который описывает последовательность задач, начиная с Downloads_launches, затем переходя к get_pictures, обработке данных с помощью Python, обучению модели и, наконец, мониторингу выполнения всех этих задач.

Выводы

5.1.1. Была развернута Конфигурация репозиторий ВМ в VirtualBox.

5.1.2. Был клонирован на ПК задание Бизнес-кейс «Rocket» в домашний каталог ВМ.

5.1.3. Был запущен контейнер с кейсом.

5.1.4. Был создан исполняемый файл с расширением .sh, который автоматизирует выгрузку данных из контейнера в основную ОС данных, полученные в результате работы DAG в Apache Airflow.

5.1.5. Была спроектирована верхнеуровневая архитектуру аналитического решения задания Бизнес-кейса «Rocket» в draw.io.

5.1.6. Была спроектирована архитектуру DAG Бизнес-кейса «Rocket» в draw.io.

5.1.7. Была построена диаграмма Ганта работы DAG в Apache Airflow.

