Департамент образования 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 »

Практическая работа 28.03

«Бизнес-кейс «Rocket»»

Выполнила:

Студентка группы АДЭУ-211

Кравцова Алёна Евгеньевна

Руководитель:

Босенко Т.М

Москва

2025

Цель работы: спроектировать и реализовать верхнеуровневую архитектуру Rocket, а также построить архитектуру DAG. Запустить и разобрать бизнес-кейс Rocket.

Задачи:

1. Сделать файл с расширением .sh для копирования полученных изображений в локальную ОС;
2. Спроектировать верхнеуровневую архитектуру аналитического решения задания Бизнес-кейса «Rocket» в draw.io;
3. Спроектировать архитектуру DAG Бизнес-кейса «Rocket» в draw.io;
4. Выполнить индивидуальное задание.

Перед выполнением задач работы необходимо настроить систему. После клонирования репозитория перейдем в папку business_case_rocket_25 (Рис. 1), в которой расположен интересующий dockerfile и приступим к сборке образа (Рис. 2).

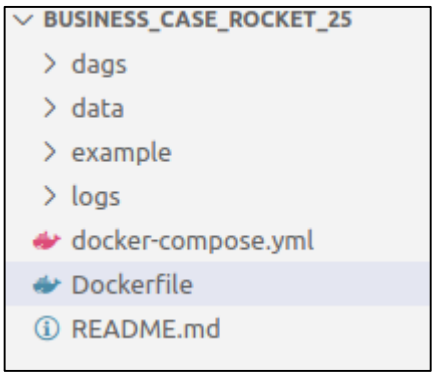

Рис. 1 – Необходимая директория

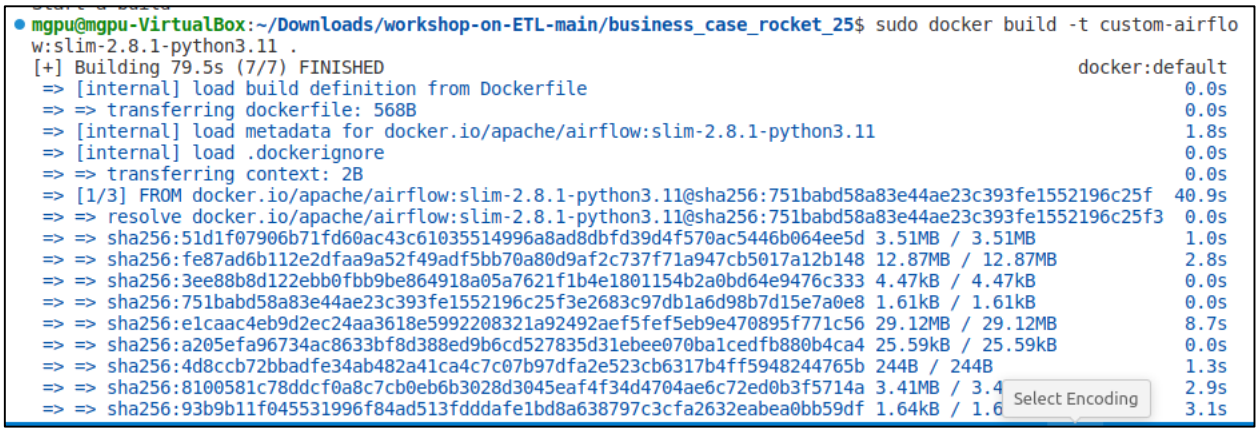

Рис. 2 – Сборка airflow

Далее поднимем все сервисы (Рис. 3).

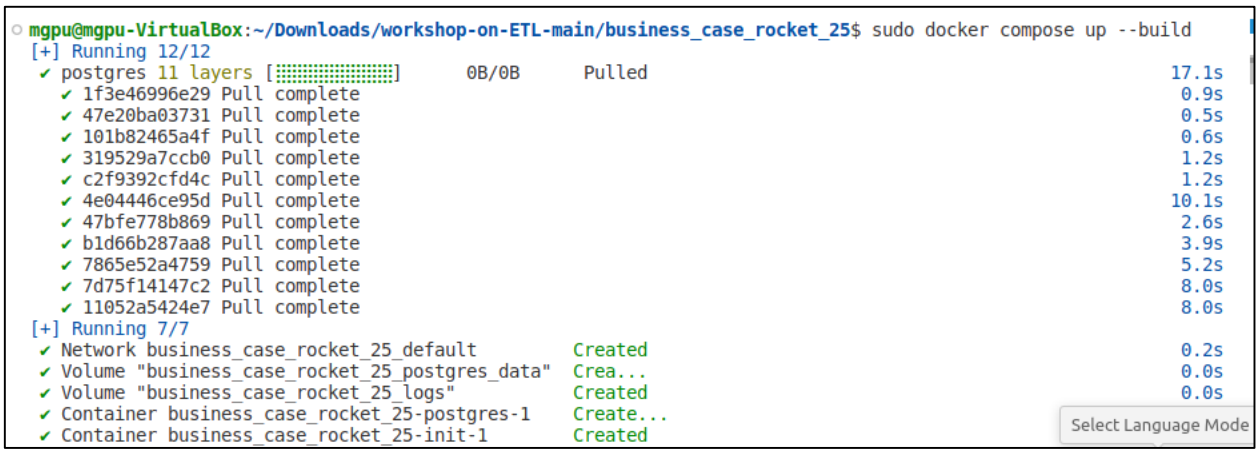

Рис. 3 – Запуск сервисов

Airflow запущен успешно (Рис. 4).

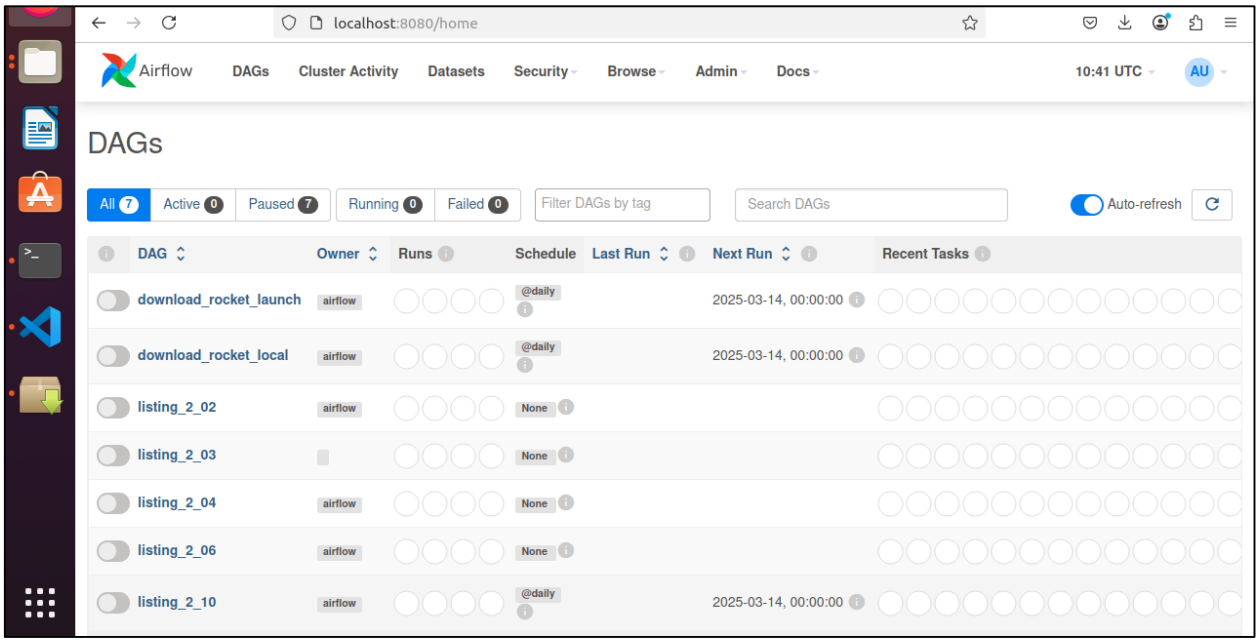

Рис. 4 – Airflow

Проверим работоспособность и запустим DAG, предварительно выдадим все права на папку data (Рис. 5).

```
mgpu@mgpu-VirtualBox:~/Downloads/workshop-on-ETL-main/business_case_rocket_25$ sudo chown -R 50000
:50000 ./data
```

Рис. 5 – Настройка прав доступа

Итак, DAG выполнен успешно (Рис. 6).

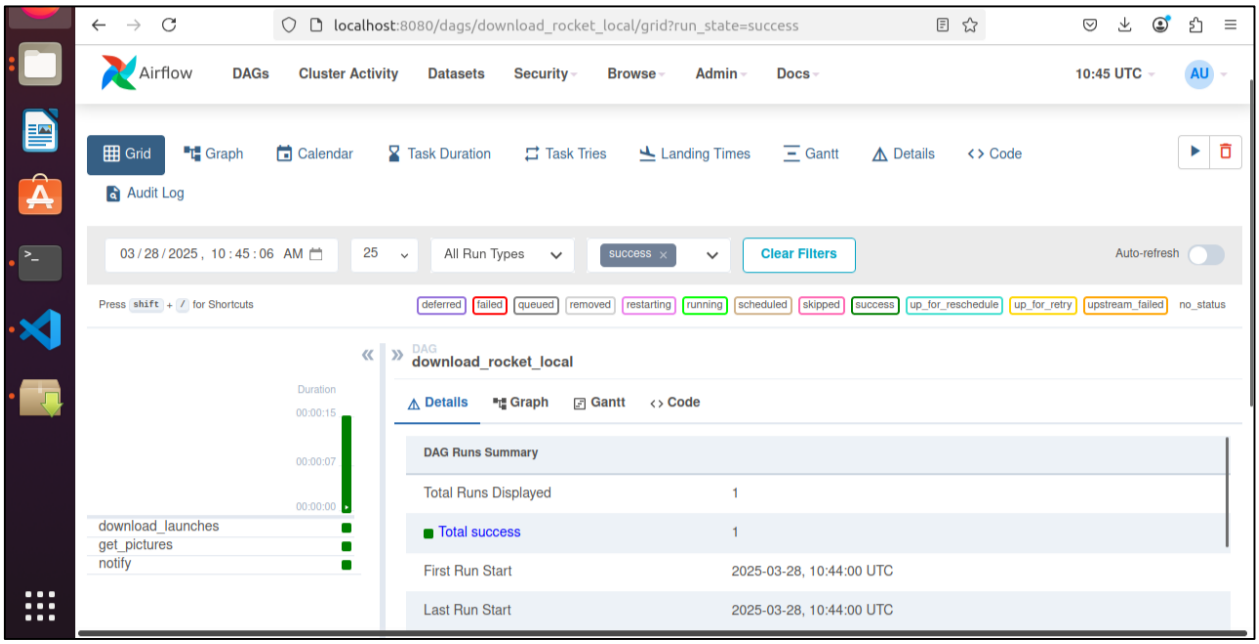

Рис. 6 – Успешный запуск DAG

И в папке data появились изображения (Рис. 7).

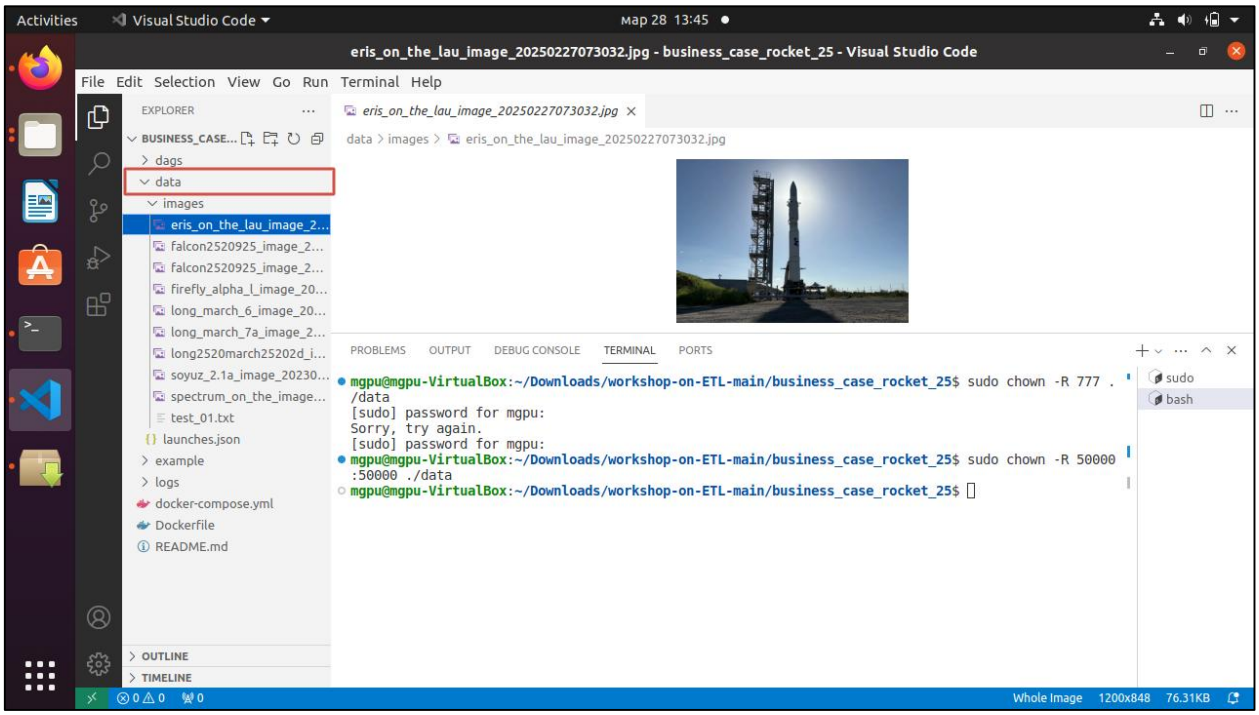

Рис. 7 – Полученные изображения

Задание 1.1. Спроектировать верхнеуровневую архитектуру аналитического решения задания Бизнес-кейса «Rocket» в draw.io. Необходимо использовать:

- Source Layer - слой источников данных.
- Storage Layer - слой хранения данных.
- Business Layer - слой для доступа к данным пользователей.

  На рисунке 8 представлена получившаяся архитектура

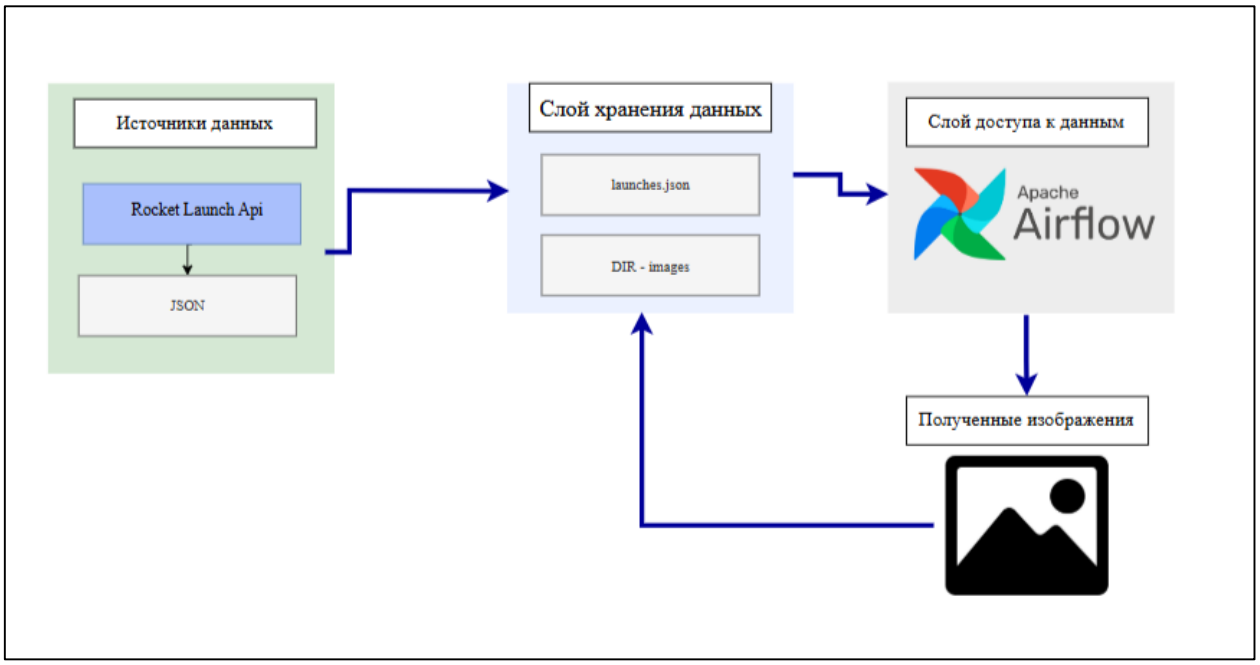

Рис. 8 – Архитектура кейса Rocket

Задание 1.2. Спроектировать архитектуру DAG Бизнес-кейса «Rocket» в draw.io. Необходимо использовать:

- Source Layer - слой источников данных.
- Storage Layer - слой хранения данных.
- Business Layer - слой для доступа к данным пользователей.

Архитектура DAG представлена на рисунке 9.

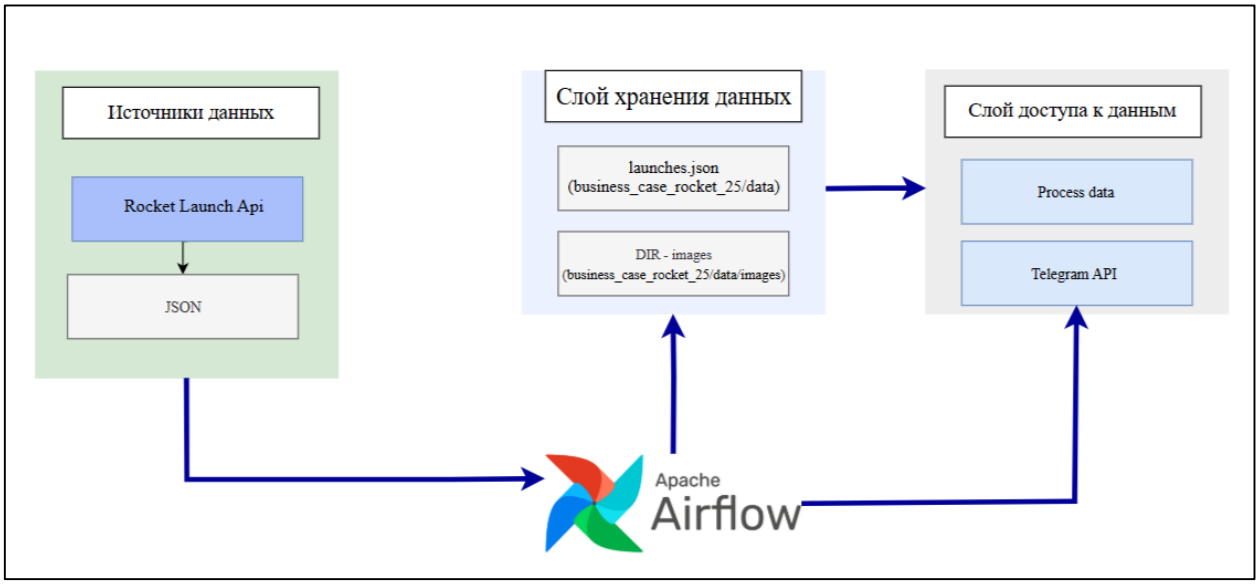

Рис. 9 – Архитектура DAG с учетом задания

Задание 2. Копирование данных в локальную ОС.

Далее создадим файл copy.sh (Рис. 10), который выгружает данные в ОС.

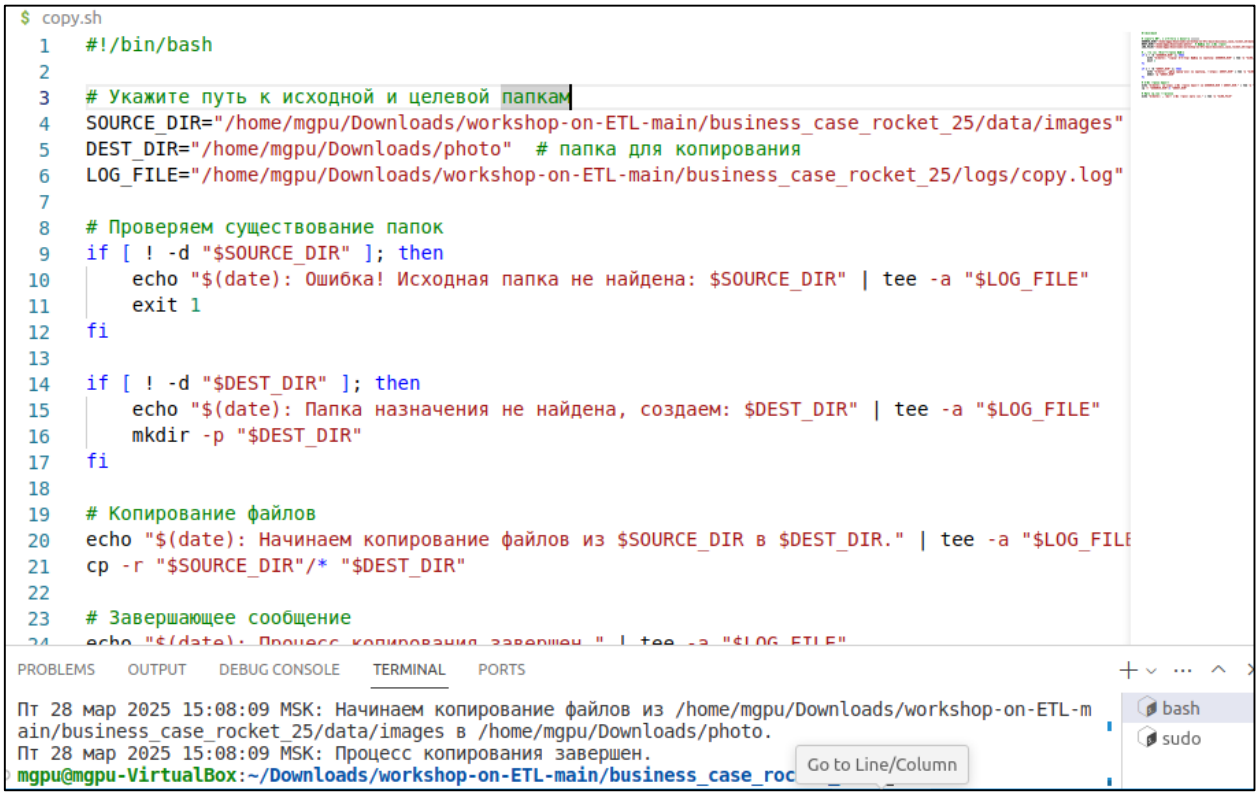

Рис. 10 – copy.sh

После запуска данные скопировались в указанную папку Downloads/photo (Рис. 11).

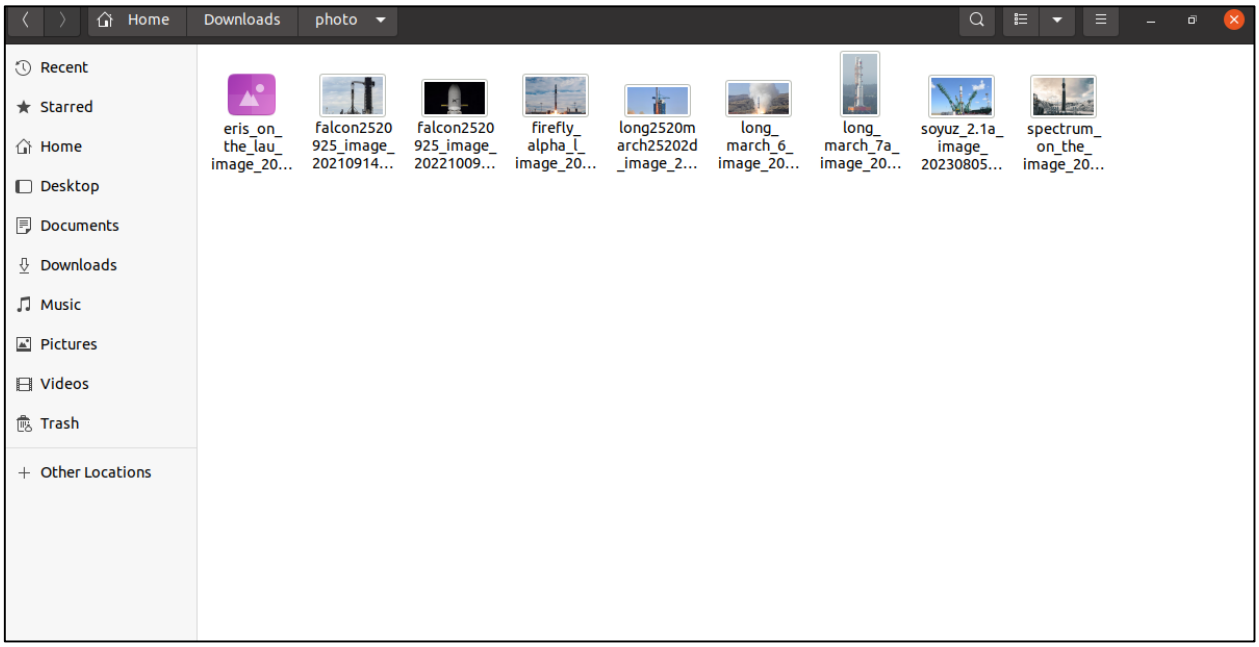

Рис. 11 – Результат копирования файлов

Индивидуальное задание. Вариант 6.

Настроить Airflow для автоматической обработки данных по запуску ракет	Реализовать систему уведомлений при сбоях загрузки изображений	Провести анализ на количество и частоту запусков ракет за последний месяц

1. Настроить Airflow для автоматической обработки данных по запуску ракет.

Для автоматической обработки данных по запуску ракет необходимо в коде DAG прописать параметр schedule, например, ежедневно (Рис. 12).

```
schedule_interval="@daily",
```

Рис. 12 – Установка расписания запуска

При данном параметре DAG будет запускаться автоматически каждый день с 00:00 текущего дня до 00:00 следующего дня (Рис. 13).

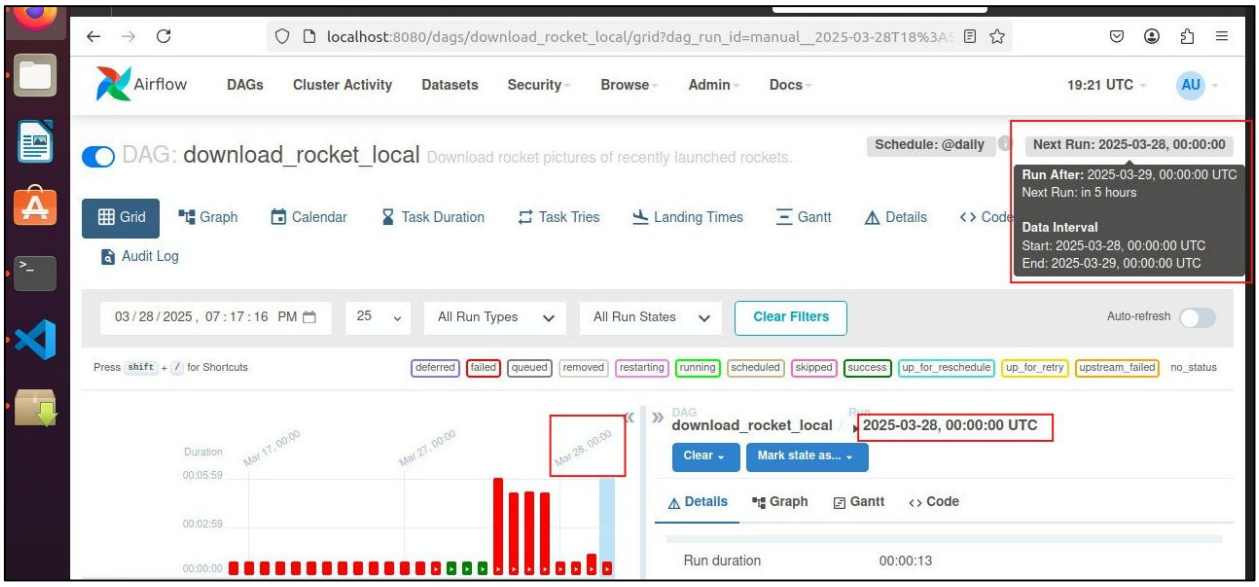

Рис. 13 – DAG запущенный автоматически

2. Реализовать систему уведомлений при сбоях загрузки изображений.

Для отправки уведомлений будет выбран Telegram бот за счет простоты внедрения и повышения удобства.

Сперва необходимо создать тг бота и начать с ним общение, чтобы получить TOKEN_ID и CHAT_ID. TOKEN_ID узнается после создания бота (Рис. 14).

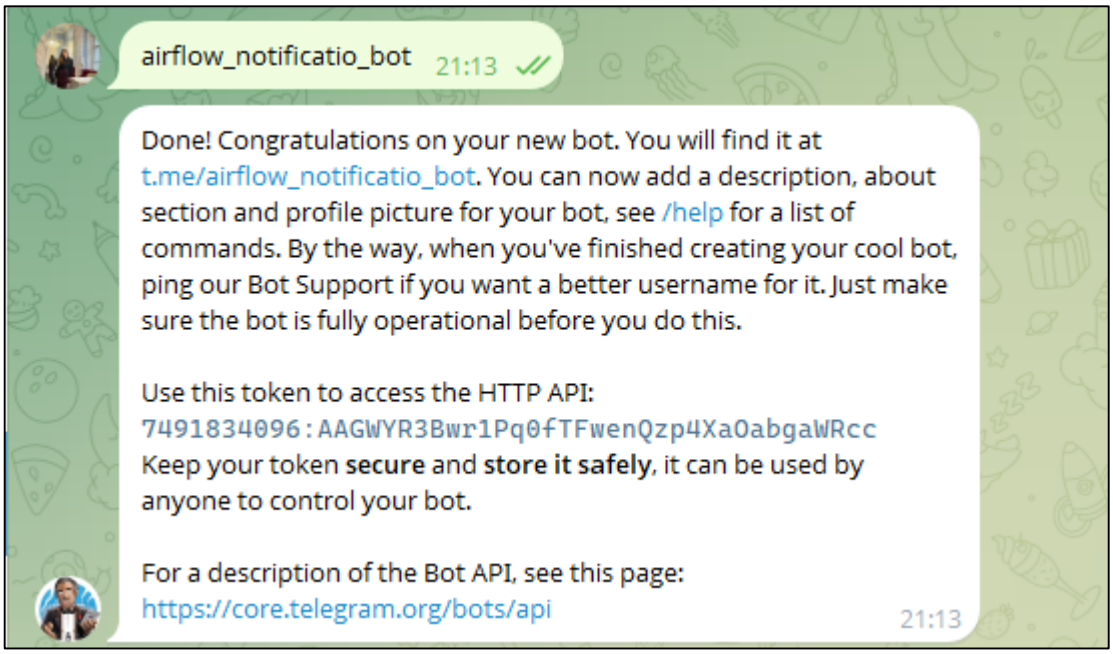

Рис. 14 – TOKEN_ID

Далее введем в браузере api запрос, в ответе которого расположен CHAT_ID (Рис. 15).

Рис. 15 – CHAT_ID

Далее добавим в даг следующий код:

```
import requests

TELEGRAM_BOT_TOKEN = " 7491834096:AAGWYR3Bwr1Pq0fTFwenQzp4XaOabgaWRcc "  #
Токен бота
```

```
TELEGRAM_CHAT_ID = " 1482158775"  # chat_id

    def send_telegram_alert(context):
        """Функция отправки сообщений в Telegram при ошибке DAG."""
        message = f"❌ Ошибка в DAG {context['task_instance'].dag_id}\nTask:
    {context['task_instance'].task_id}\nError: {context['exception']}"
        url = f"https://api.telegram.org/bot{TELEGRAM_BOT_TOKEN}/sendMessage"
        params = {"chat_id": TELEGRAM_CHAT_ID, "text": message}
        try:
            response = requests.get(url, params=params)
            response.raise_for_status()
        except requests.exceptions.RequestException as e:
            print(f"Ошибка отправки в Telegram: {e}")
```

Далее специально допустим ошибку в Dag'е для проверки уведомления, например, в наименовании переменной (Рис. 16).

```
40  def _get_pictures():
41      images_dir = "/opt/airflow/data/images"
42      pathlib.Path(images_dire).mkdir(parents=True, exist_ok=True)
43
44      with open("/opt/airflow/data/launches.json") as f:
45          launches = json.load(f)
46          image_urls = [launch["image"] for launch in launches["results"]]
47          for image_url in image_urls:
48              try:
49                  response = requests.get(image_url)
```

Рис. 16 – Допущение ошибки в наименовании переменной

Далее перезапустим контейнеры и запустим даг. Даг отработал с ошибкой (Рис. 17).

Рис. 17 – Результат отработки DAG с ошибкой

После чего пришло уведомление от тг-бота (Рис. 18).

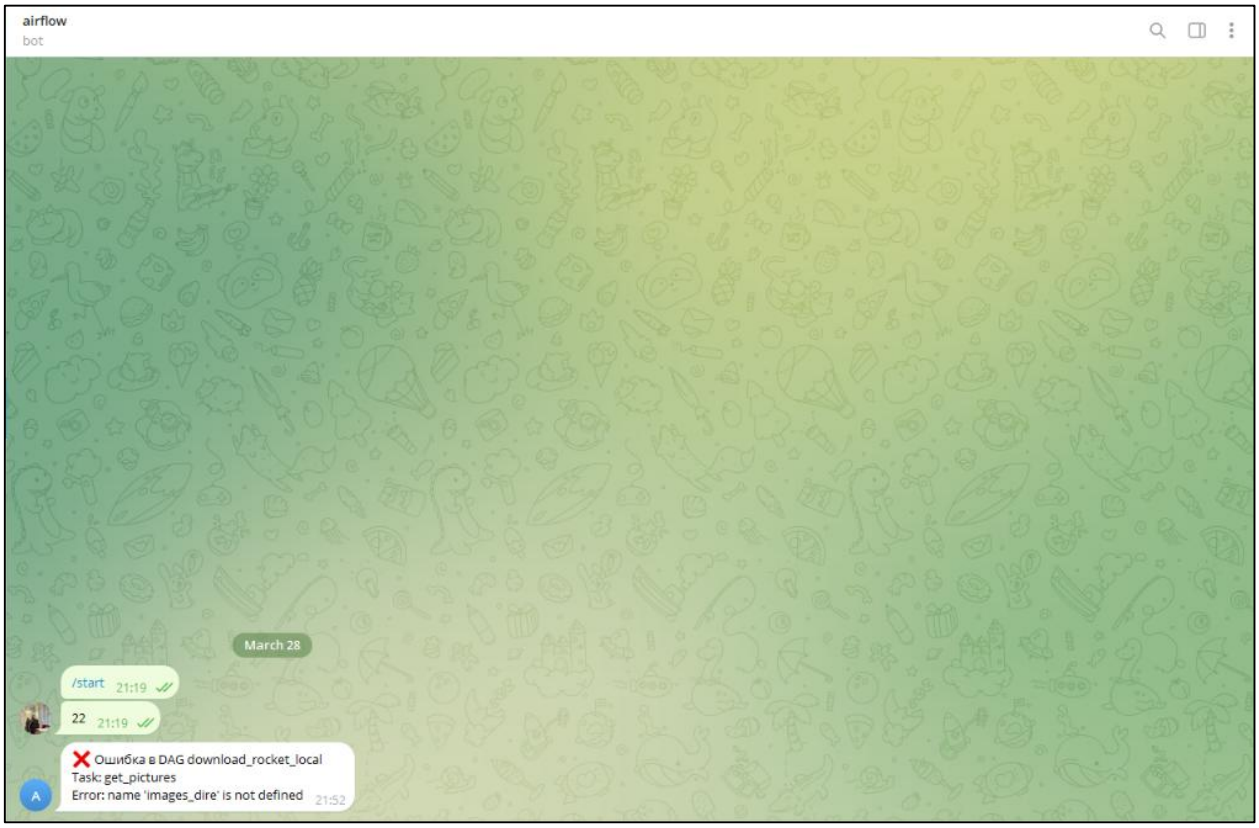

Рис. 18 – Уведомление об ошибке от тг-бота

3. Провести анализ на количество и частоту запусков ракет за последний месяц

Загрузим launches.json в google colab для дальнейшего анализа (Рис. 19).

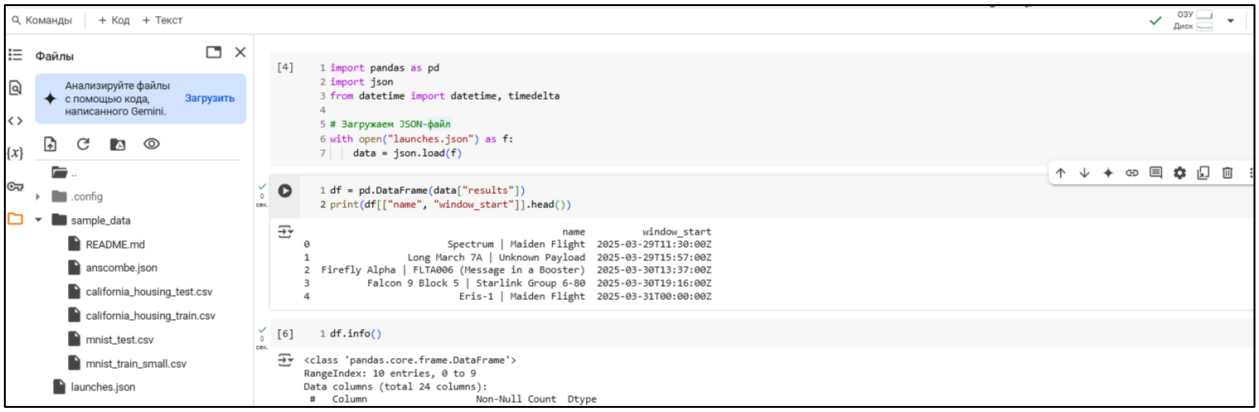

Рис. 19 – Загрузка json файла

Просмотрим типы данных. Для выполнения задания понадобится параметр даты, который имеет тип object. Необходимо преобразовать в тип datetime (Рис. 20).

```
[6]  1 df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 24 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       10 non-null     object
 1   url                      10 non-null     object
 2   launch_library_id        0 non-null      object
 3   slug                     10 non-null     object
 4   name                     10 non-null     object
 5   status                   10 non-null     object
 6   net                      10 non-null     object
 7   window_end               10 non-null     object
 8   window_start             10 non-null     object
 9   inhold                   10 non-null     bool
 10  tbdtime                  10 non-null     bool
 11  tbddate                  10 non-null     bool
 12  probability              0 non-null      object
 13  holdreason               10 non-null     object
 14  failreason               10 non-null     object
 15  hashtag                  0 non-null      object
 16  launch_service_provider  10 non-null     object
 17  rocket                   10 non-null     object
 18  mission                  10 non-null     object
 19  pad                      10 non-null     object
 20  webcast_live             10 non-null     bool
 21  image                    10 non-null     object
 22  infographic              0 non-null      object
 23  program                  10 non-null     object
dtypes: bool(4), object(20)
memory usage: 1.7+ KB

[7]  1 df["window_start"] = pd.to_datetime(df["window_start"])
```

Рис. 20 – Преобразование в тип datetime

Далее просмотрим данные, которые есть в файле (Рис. 21).

```
[33]  1 print("Минимальная дата в данных:", df["window_start"].min())
      2 print("Максимальная дата в данных:", df["window_start"].max())
      3 print("Сегодняшняя дата (UTC):", today)
      4 print("Дата 30 дней назад (UTC):", last_month)
      5

Минимальная дата в данных: 2025-03-29 11:30:00+00:00
Максимальная дата в данных: 2025-04-04 22:42:00+00:00
Сегодняшняя дата (UTC): 2025-03-29 05:11:41.462977+00:00
Дата 30 дней назад (UTC): 2025-02-27 05:11:41.462977+00:00
```

Рис. 21 – Минимальная и максимальная дата

Таким образом, данных за последние 30 дней нет, поэтому проведем анализ данных с 29 марта по 4 апреля. Построим bar chart по сгруппированным количествам запусков по дате (Рис. 22).

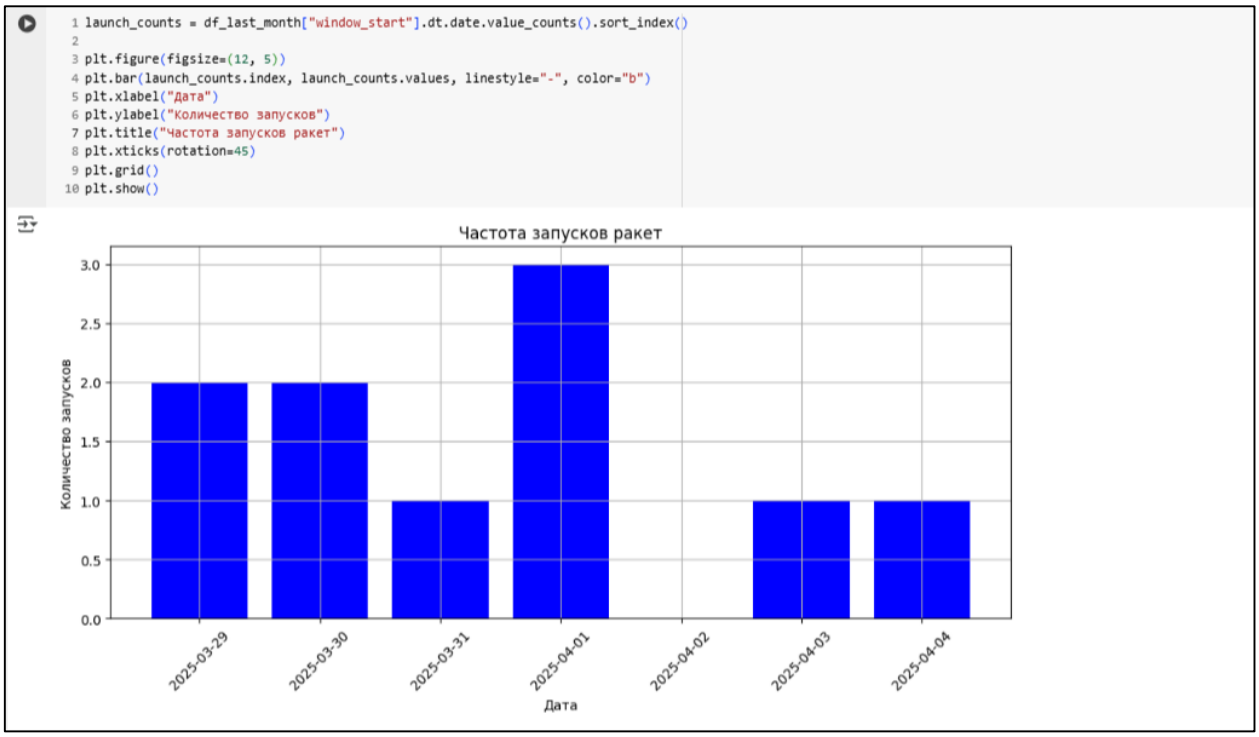

Рис. 22 – Частота запусков ракет

Заключение: таким образом была закреплена работа с airflow, а именно проработка DAG, его запуск и выгрузка данных. Был получен опыт подключения уведомлений через тг-бота. Все задачи были выполнены.