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

ОТЧЕТ

по дисциплине «Проектный практикум по разработке ETL-решений»
Вебинар: Бизнес-кейс «Rocket»
Направление подготовки – 38.03.05 «Бизнес-информатика».
профиль подготовки – «Аналитика данных и эффективное управление»

Выполнила:
студентка группы АДЭУ-211
st92
_____

Руководитель:
Кандидат технических наук, доцент
_____

Москва
2025 год

Цель работы: Научиться извлекать данные из внешних источников

Задачи

1. Изучить методы чтения данных из разных источников.
2. Освоить техники обработки и очистки данных.
3. Научиться согласовывать данные из разных источников.
4. Реализовать сохранение обработанных данных.

Ход работы:

1. Развертывание ВМ в VirtualBox

   Запустим виртуальную машину:

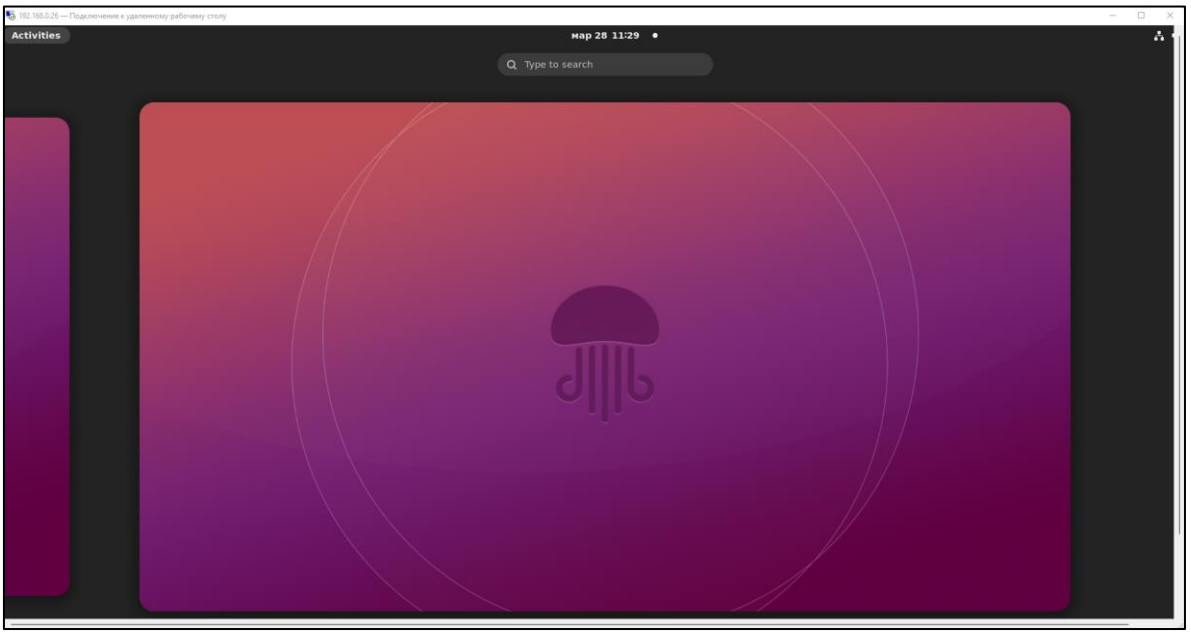

Рисунок 1 – Развернутая ВМ в VirtualBox

2. Скачивание и настройка проекта

Клонируем репозиторий с проектом:

```
dev@dev-vm:~/Downloads$ git clone https://github.com/BosenkoTM/workshop-on-ETL.git
```

Рисунок 2 – Выполнение команды для клонирования репозитория

```
dev@dev-vm:~/Downloads$ ls
 business_case_umbrella_25       nosql-workshop-main
 business_case_umbrella_25.rar   nosql-workshop-main.zip
 business_case_umbrella_25.zip  'Panel Title-data-2025-03-07 06_06_24.csv'
 dba                             progs
 de                              test
 filmdb.movies.csv               try_01..ipynb
 lab_etl                         workshop-on-ETL
 labs_cicd                       workshop-on-ETL-main.zip
```

Рисунок 3 – Просмотр содержимого каталога

Перейдем в директорию с бизнес-кейсом "Rocket":

```
dev@dev-vm:~/Downloads$ cd workshop-on-ETL/business_case_rocket_25
dev@dev-vm:~/Downloads/workshop-on-ETL/business_case_rocket_25$
```

Рисунок 4 – Выполнение команды для перехода в необходимую директорию

3. Настройка и сборка контейнеров

Перейдем в каталог проекта, где расположен Dockerfile и построим Docker образ: Используем команду для сборки Docker образа, указав тэг для образа:

```
dev@dev-vm:~/Downloads/workshop-on-ETL/business_case_rocket_25$ sudo docke
r build -t custom-airflow:slim-2.8.1-python3.11 .
[sudo] password for dev:
[+] Building 0.9s (1/2)                                          docker:default
 => [internal] load build definition from Dockerfile                       0.0s
 => => transferring dockerfile: 568B                                       0.0s
 => [internal] load metadata for docker.io/apache/airflow:slim-2.8.        0.9s
```

Рисунок 5 – Выполнение команды для сборки Docker образа

Запустим контейнер с использованием Docker Compose:

```
dev@dev-vm:~/Downloads/workshop-on-ETL/business_case_rocket_25$ sudo docke
r compose up --build
[+] Running 4/4
 ✔ Container business_case_rocket_25-postgres-1   Created                  0.0s
 ✔ Container business_case_rocket_25-init-1       Created                  0.0s
 ✔ Container business_case_rocket_25-scheduler-1  Created                  0.0s
 ✔ Container business_case_rocket_25-webserver-1  Created                  0.0s
Attaching to init-1, postgres-1, scheduler-1, webserver-1
```

Рисунок 6 – Запуск контейнера

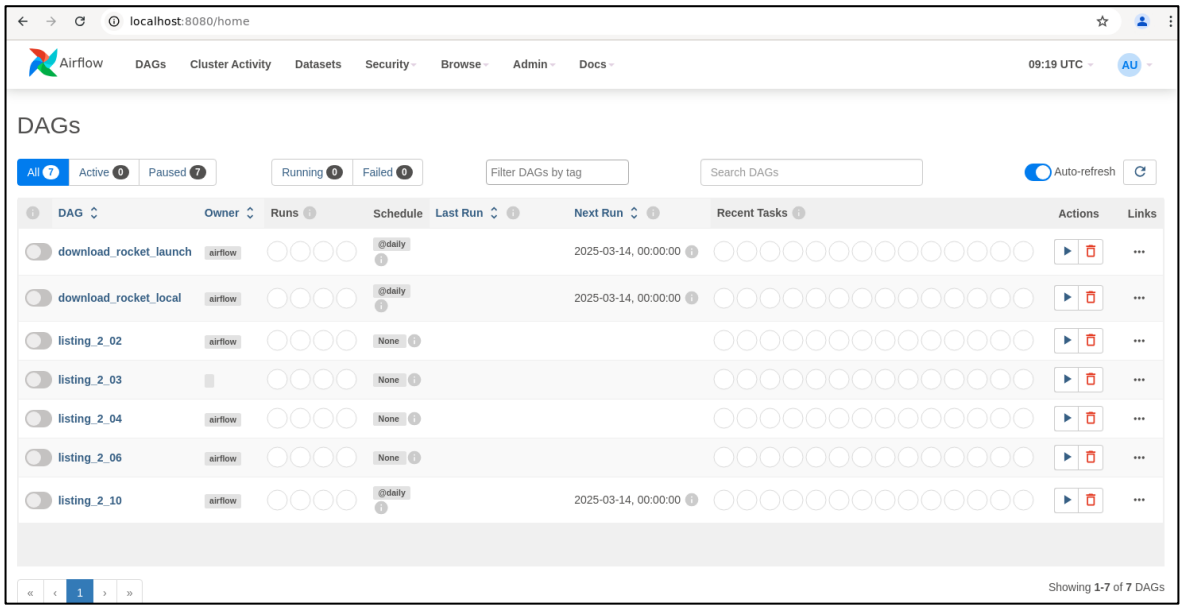

Рисунок 7 – Успешное соединение

Общее задание.

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

Определим параметры для файла export_airflow_data.sh:

```
# Параметры
CONTAINER_NAME="business_case_rocket_25-webserver-1"
AIRFLOW_DATA_DIR="/opt/airflow/data"
HOST_EXPORT_DIR="$HOME/airflow_data_export"
TIMESTAMP=$(date +"%Y%m%d_%H%M%S")
```

Рисунок 8 – Определение параметров

Определим команду для создания директории для логов:

```
mkdir -p "$HOST_EXPORT_DIR/$TIMESTAMP"
```

Рисунок 9 – Команда для создания директории для логов в родительской директории

Определим команды для копирования файлов:

```
# 1. Копируем JSON файлы
echo "Exporting JSON files..."
docker cp "$CONTAINER_NAME:$AIRFLOW_DATA_DIR/launches.json" "$HOST_EXPORT_DIR/$TIMESTAMP/launch

# 2. Копируем изображения
echo "Exporting images..."
docker cp "$CONTAINER_NAME:$AIRFLOW_DATA_DIR/images/" "$HOST_EXPORT_DIR/$TIMESTAMP/images/"
```

Рисунок 10 – Часть кода с копированием и созданием файлов

Выдадим права на исполнение файла:

```
dev@dev-vm:~/Downloads/workshop-on-ETL/business_case_rocket_25$ chmod +x export_airflow_data.sh
dev@dev-vm:~/Downloads/workshop-on-ETL/business_case_rocket_25$
```

Рисунок 11 – Выполнение команды для выдачи прав на исполнение файла

```
dev@dev-vm:~/Downloads/workshop-on-ETL/business_case_rocket_25$ ./export_airflow_data.sh
Exporting JSON files...
Successfully copied 27.6kB to /home/dev/airflow_data_export/20250328_124537/launches.json
Exporting images...
Successfully copied 1.64MB to /home/dev/airflow_data_export/20250328_124537/images/
-----------------------------------------
Export completed successfully!
Exported data location: /home/dev/airflow_data_export/20250328_124537
Contents:
total 32K
drwxrwxr-x 2 dev docker 4,0K мар 28 10:05 images
-rw-rw-r-- 1 dev docker  26K мар 28 10:05 launches.json
```

Рисунок 12 – Выполнение команды для исполнения файла

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.

Спроектируем верхнеуровневую архитектуру аналитического решения задания Бизнес-кейса «Rocket»:

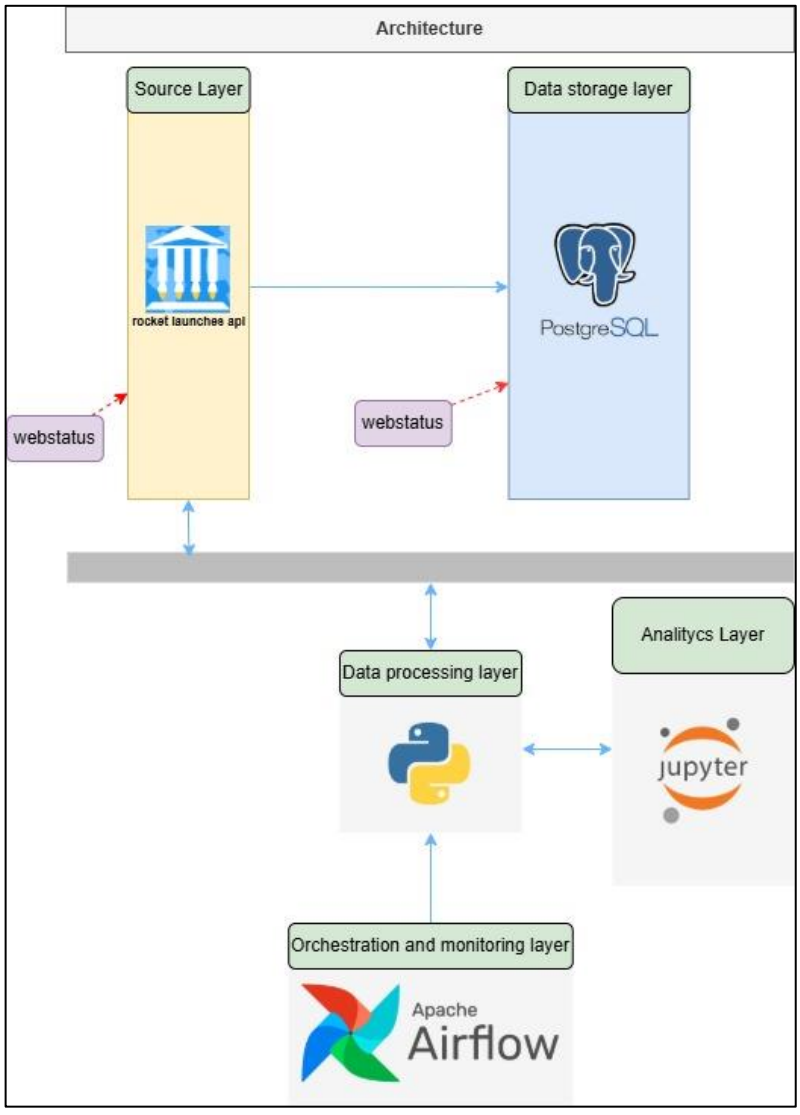

Рисунок 13 – Верхнеуровневая архитектура бизнес-кейса

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.

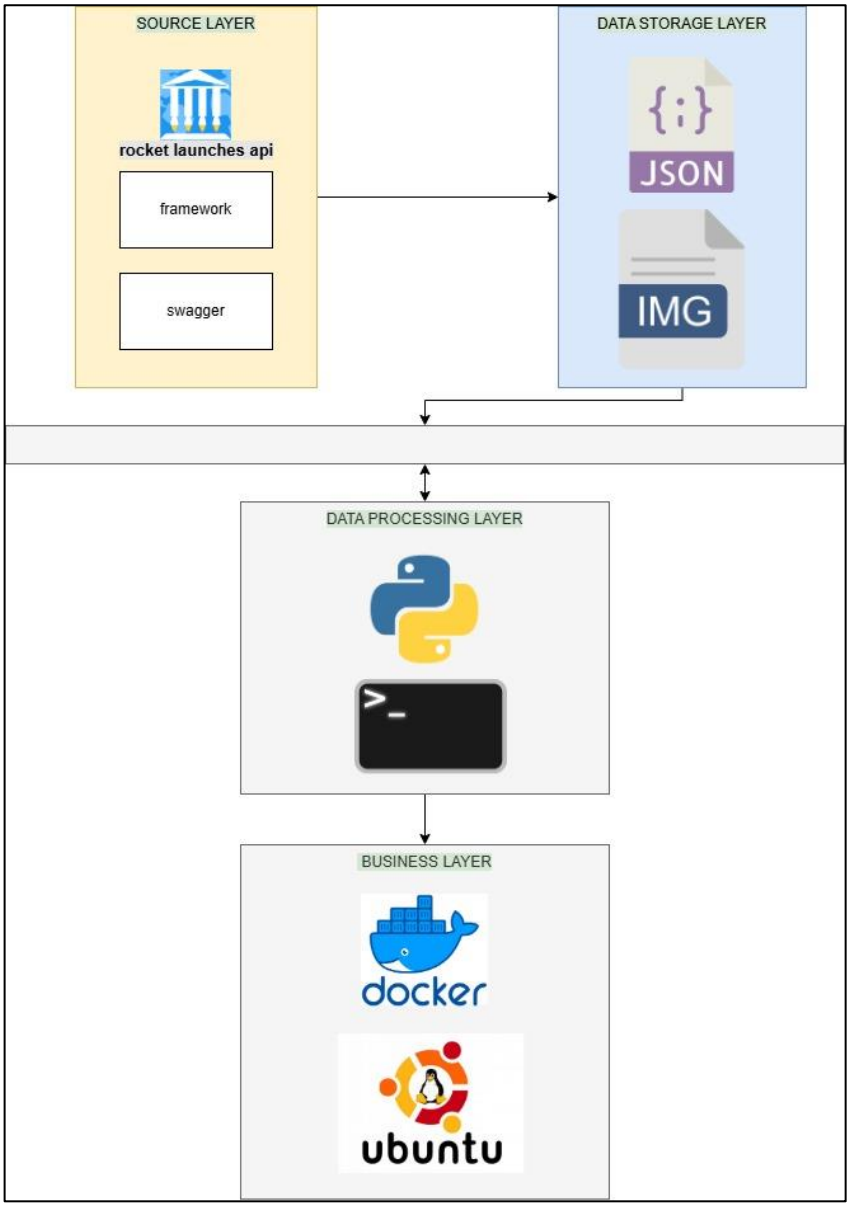

Рисунок 14 – Верхнеуровневая архитектура DAG

1.3 Построить диаграмму Ганта работы DAG в Apache Airflow.

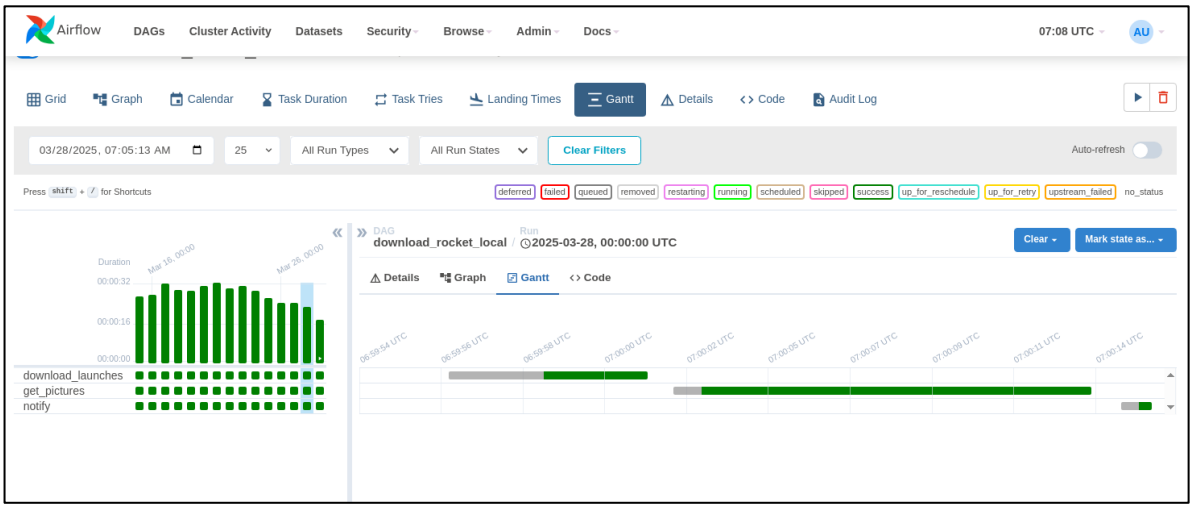

Рисунок 15 – Диаграмма ганта в Apache Airflow

Индивидуальное задание

1. Изучить архитектуру системы Airflow для обработки данных с использованием DAG

Архитектура включает следующие компоненты:

Компоненты DAG:

- download_launches: BashOperator для загрузки JSON-данных о предстоящих запусках ракет через API
- get_pictures: PythonOperator для обработки JSON и загрузки изображений ракет
- notify: BashOperator для уведомления о количестве загруженных изображений

Инфраструктура:

- PostgreSQL: Хранит метаданные Airflow (DAGs, задачи, статусы выполнения)
- Webserver: Предоставляет веб-интерфейс для мониторинга
- Scheduler: Планирует и координирует выполнение задач
- Executor (LocalExecutor): Выполняет задачи локально в контейнере

Особенности реализации:

- Используется монтирование томов для хранения данных (./data) и DAG-файлов (./dags)

- Настроено автоматическое создание администратора при инициализации
- Используется кастомный образ Airflow с дополнительными зависимостями

2. Настроить мониторинг ошибок в процессе скачивания изображений

```
dags > download_rocket_local.py
25  def _get_pictures():
26      pathlib.Path(images_dir).mkdir(parents=True, exist_ok=True)
29
30      # Загружаем все картинки из launches.json
31      with open("/opt/airflow/data/launches.json") as f:
32          launches = json.load(f)
33          image_urls = [launch["image"] for launch in launches["results"]]
34          for image_url in image_urls:
35              try:
36                  response = requests.get(image_url)
37                  image_filename = image_url.split("/")[-1]
38                  target_file = f"{images_dir}/{image_filename}"
39                  with open(target_file, "wb") as f:
40                      f.write(response.content)
41                  print(f"Downloaded {image_url} to {target_file}")
42              except requests_exceptions.MissingSchema:
43                  print(f"{image_url} appears to be an invalid URL.")
44              except requests_exceptions.ConnectionError:
45                  print(f"Could not connect to {image_url}.")
46
```

Рисунок 16 – Часть кода с мониторингом ошибок

3. Создать документ с анализом эффективности загрузки данных о стартах ракет

   Создадим исполняемый файл для выгрузки лога по выполненному DAG

```
$ export_dag_details.sh
1   #!/bin/bash
2
3   # Настройки
4   DAG_ID="download_rocket_local"
5   CONTAINER_NAME="business_case_rocket_25-webserver-1"  # или "airflow-scheduler"
6   OUTPUT_FILE="./data/dag_details_${DAG_ID}.json"
7
8   # Проверяем, что контейнер работает
9   if ! docker ps | grep -q $CONTAINER_NAME; then
10      echo "Ошибка: Контейнер $CONTAINER_NAME не запущен!"
11      exit 1
12  fi
13
14  # Выполняем команду в контейнере для экспорта DAG
15  docker exec -it $CONTAINER_NAME \
16      bash -c "airflow dags list-runs --dag-id $DAG_ID --output json > /opt/airflow/$OUTPUT_FILE"
17
18  # Проверяем результат
19  if [ $? -eq 0 ]; then
20      echo "Детали DAG $DAG_ID успешно сохранены в $OUTPUT_FILE"
21  else
22      echo "Ошибка при экспорте DAG $DAG_ID"
23      exit 1
24  fi
```

Рисунок 17 – Часть кода файла для выгрузки деталей DAG

```
dev@dev-vm:~/Downloads/workshop-on-ETL/business_case_rocket_25$ ./export_dag_details.sh
/home/airflow/.local/lib/python3.11/site-packages/airflow/utils/dot_renderer.py:28 UserWarning: Could n
ot import graphviz. Rendering graph to the graphical format will not be possible.
Детали DAG download_rocket_local успешно сохранены в ./data/dag_details_download_rocket_local.json
```

Рисунок 18 – Исполнение файла

Создадим код на python для анализа данных:

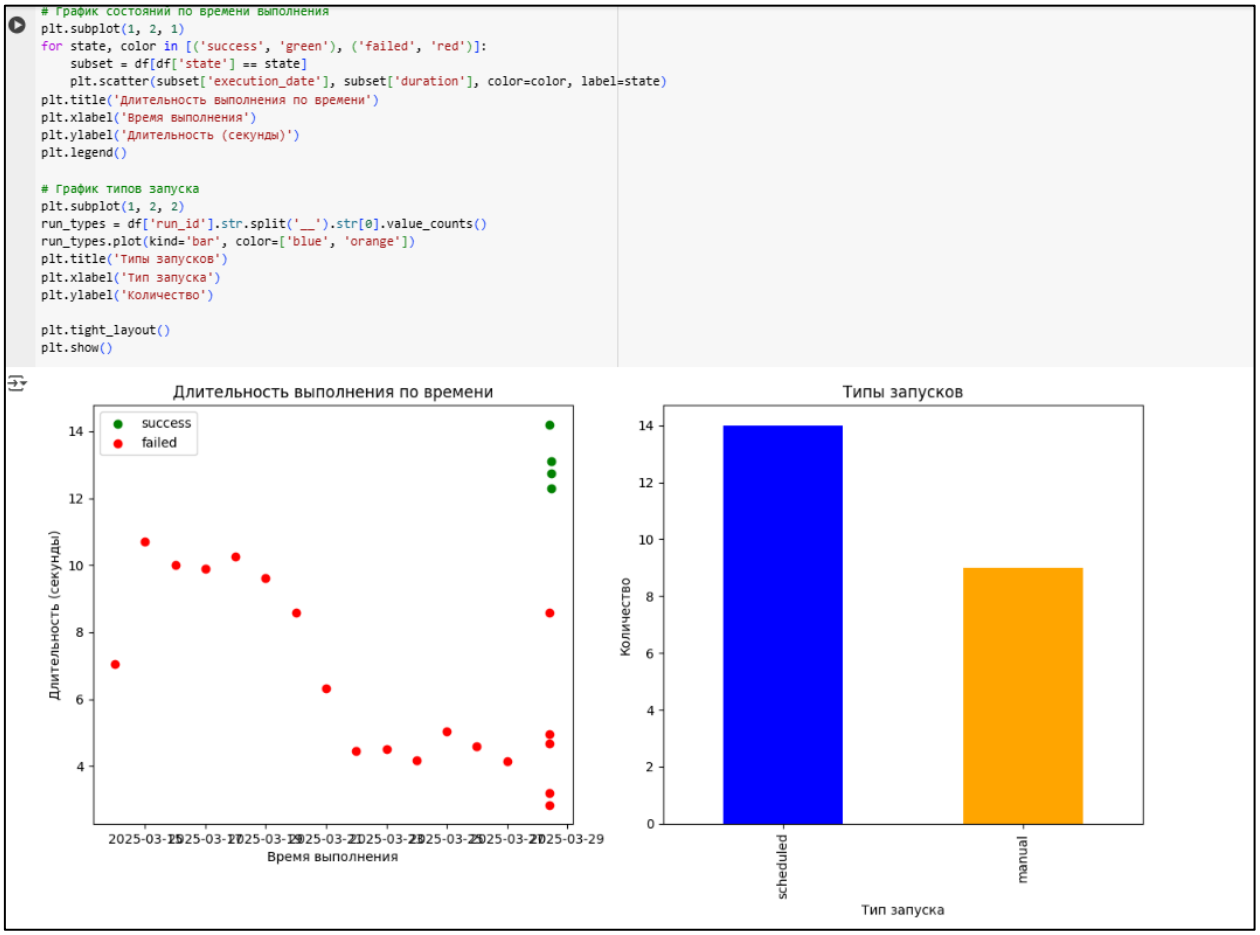

Рисунок 19 – Анализ исполнения DAG по времени

Из первого графика видно, сколько по времени выполнялись успешные и неуспешные DAG в разрезе дат

Из второго графика видно, сколько DAG были выполнены в разрезе их типа запуска (по расписанию или вручную)

```
print("Общая информация:")
print(f"Всего выполнений: {len(df)}")
print(f"Успешных выполнений: {len(df[df['state'] == 'success'])}")
print(f"Неудачных выполнений: {len(df[df['state'] == 'failed'])}")
print(f"Процент успешных выполнений: {len(df[df['state'] == 'success']) / len(df) * 100:.1f}%")

print("\nСтатистика по длительности выполнения (секунды):")
print(df['duration'].describe())
```

```
Общая информация:
Всего выполнений: 23
Успешных выполнений: 4
Неудачных выполнений: 19
Процент успешных выполнений: 17.4%

Статистика по длительности выполнения (секунды):
count    23.000000
mean      7.651366
std       3.530083
min       2.849195
25%       4.544053
50%       7.045725
75%      10.125150
max      14.200761
Name: duration, dtype: float64
```

Рисунок 20 – Анализ показателей

Из данной статистики можно извлечь необходимую информацию для анализа эффективности выполнения DAG. Например, за выполнение 23 DAG зафиксированное максимальное время равно 14.2 секундам, а минимальное 2.8 секундам.

Выводы:

В ходе работы нам удалось создать исполняемые файлы, которые автоматизируют выгрузку информации о DAG из контейнера. Использование контейнера для получения, обработки, трансформации и выгрузки данных экономит время и финансы для бизнеса. Контейнер автоматизирует процессы, выполняет необходимые задачи в короткий срок. Выгруженный файл с деталями по DAG помогает при анализе эффективности загрузки данных. Проанализировав эффективность можно убедиться, что автоматизация процессов для компании очень важна.