Департамент образования и науки города Москвы

Государственное автономное образовательное учреждение высшего образования города Москвы

«Московский городской педагогический университет»

Институт цифровых технологий

**Практическая работа 5**

«Airflow DAG»

Направление подготовки Бизнес-информатика

Направленность (профиль) образовательной программы Аналитика данных и эффективное управление

(очная форма обучения)

Выполнил: Каргин В.А.

Преподаватель: Босенко Т.М.

Москва

2024

Бизнес-кейс “Rocket”

Постановка задачи:

Цель: Постановка задачи: реализовать аналитическое решение для бизнес-кейса “Rocket”.

Цель: Разработать архитектуру решения и реализовать архитектуру данных, процессы ETL для обработки данных

5.1.1. Развернуть ВМ [ubuntu\_mgpu.ova](https://disk.yandex.ru/d/Psofa9xtbgUEOw) в [VirtualBox](https://disk.yandex.ru/d/3fD00plnL_a4Cw).

Выполнено (см. рис. 1)

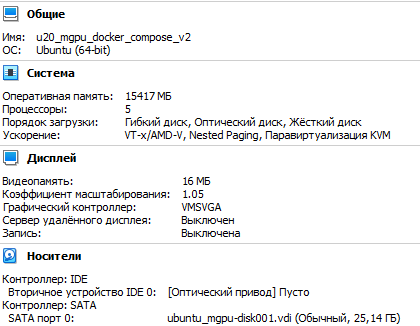


Рисунок 1. Развернутый образ ubuntu в VirtualBox

Проверка доступности библиотеки (см рисунок 2.).

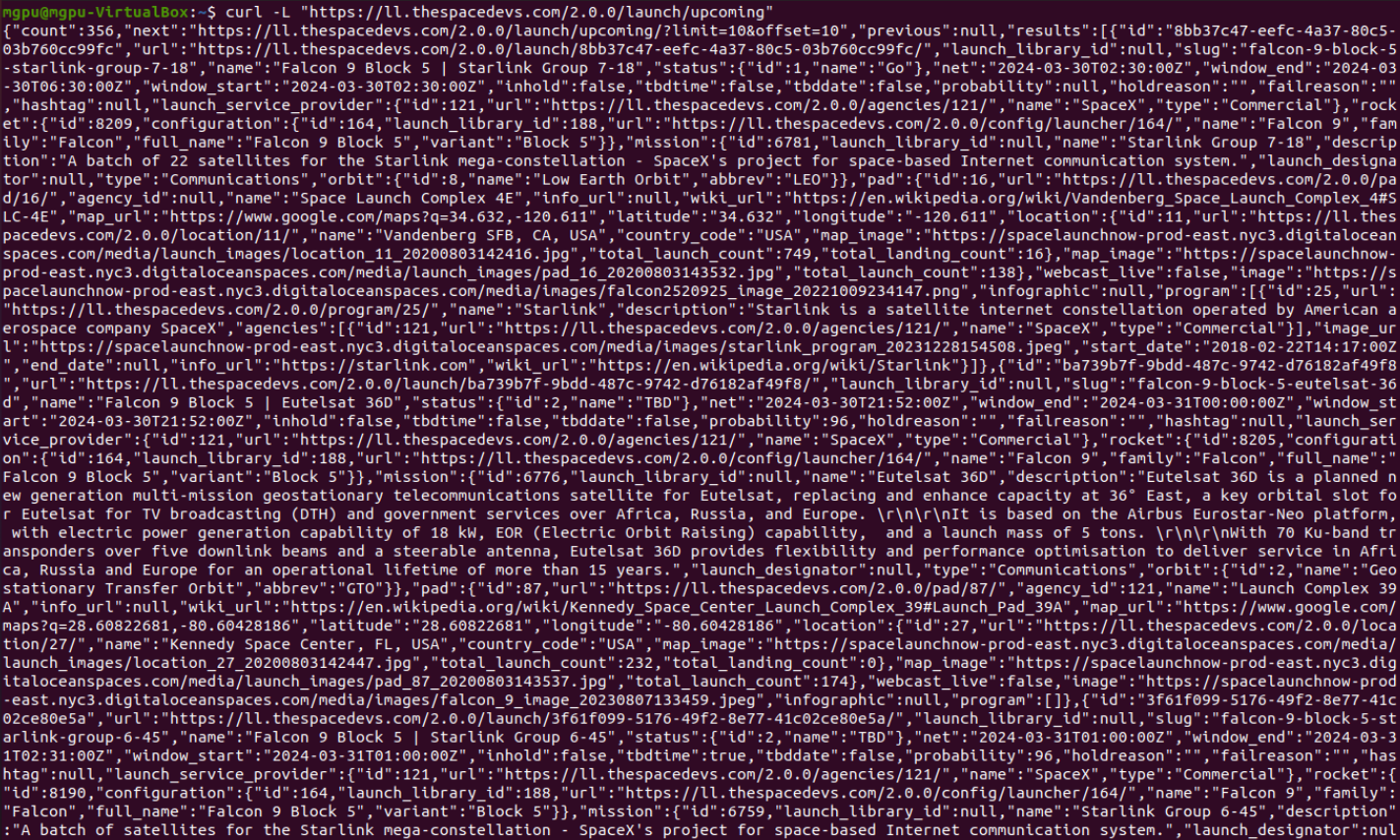


Рисунок 2. Проверка библиотеки

Скачивание файла (рис.3).

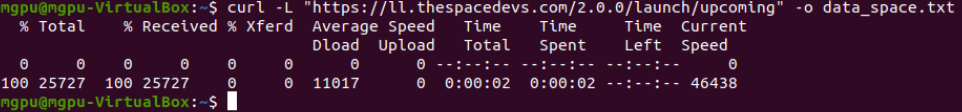


Рисунок 3. Скачивание файла data.space.txt

Содержимое скачанного файла на рис 4. 

Рисунок 4. Содержимое файла data.space.txt

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

Выполнено (см. рис 5).

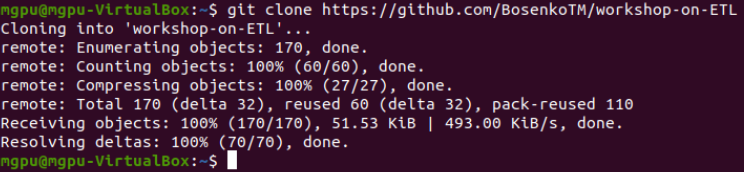


Рисунок 5. Клонирование репозитория

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

Переходим в каталог с бизнес-кейсом (см. рисунок 6).

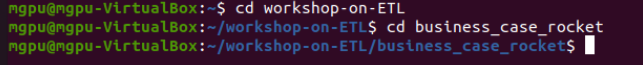


Рисунок 6. Переход в каталог бизнес-кейса

Процесс запуска Bash для загрузки ответа URL-адреса с помощью Curl. BashOperator (используется для запуска сценария Bash) (см. рисунок 7).

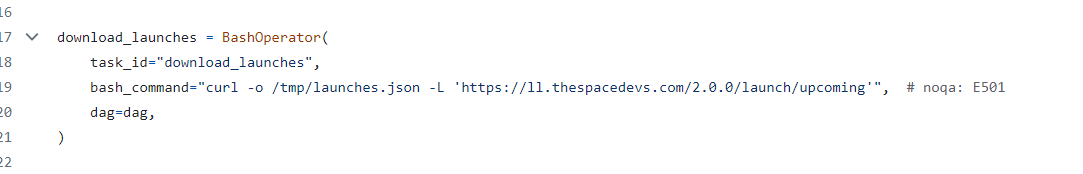


Рисунок 7. Оператор Bash

Функция Python проанализирует ответ и загрузит все изображения ракет. Вызов функции Python в DAG с помощью PythonOperator (см. рисунок 8).

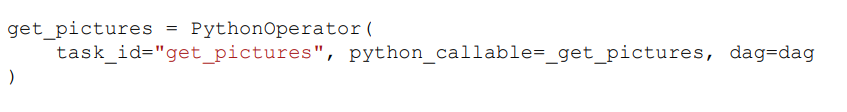


Рисунок 8. Python оператор

Запуск контейнера (см. рис. 9)

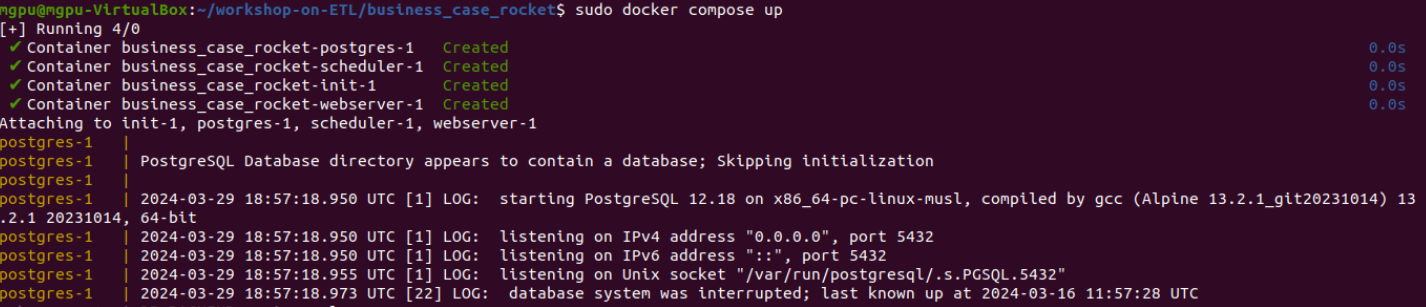


Рисунок 9. Запуск контейнера

Проверка запуска контейнера (см. рис. 10).

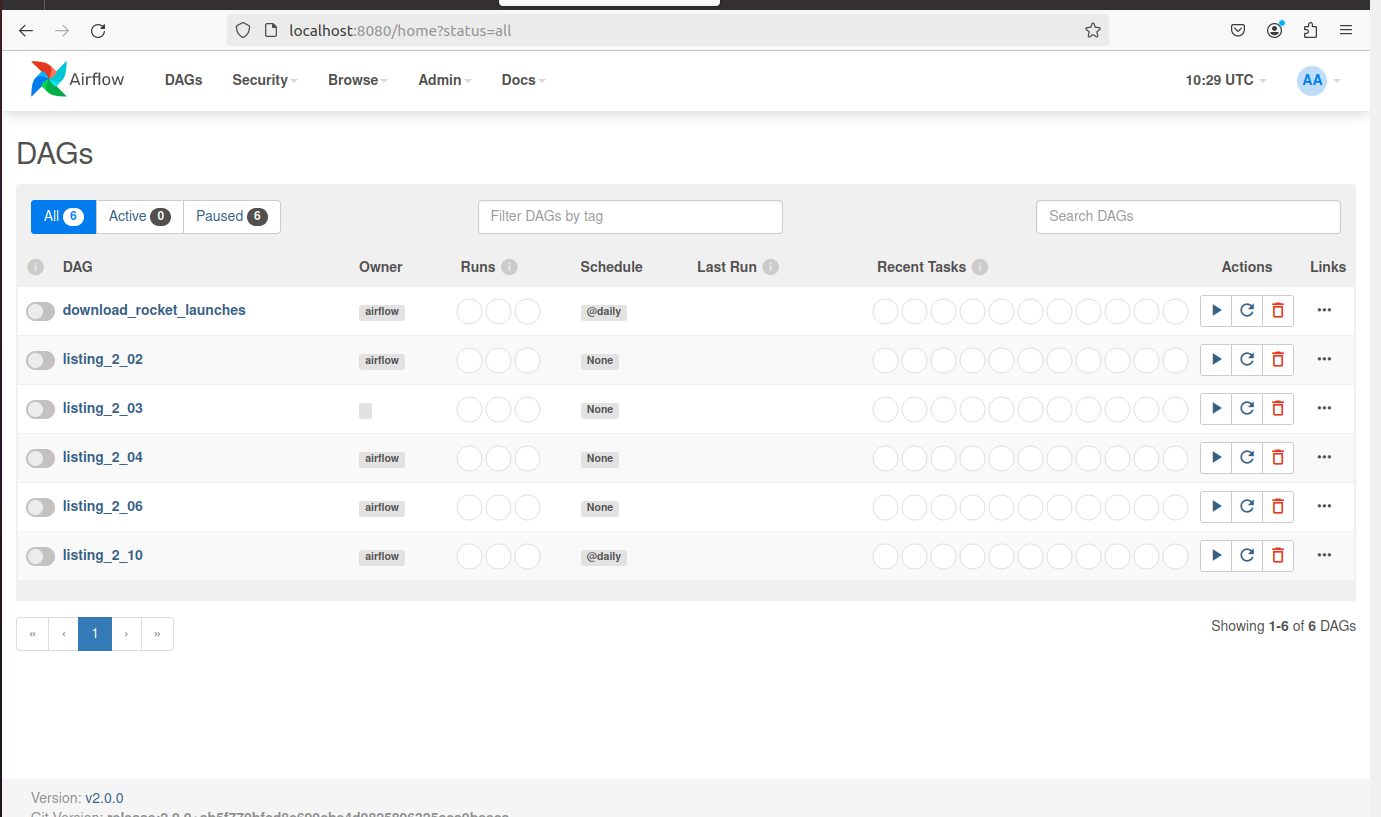


Рисунок 10. Работающий Airflow

Изучим детально разделы DAG download\_rocket\_launches:

1. Дерево задач (см. рис. 11),

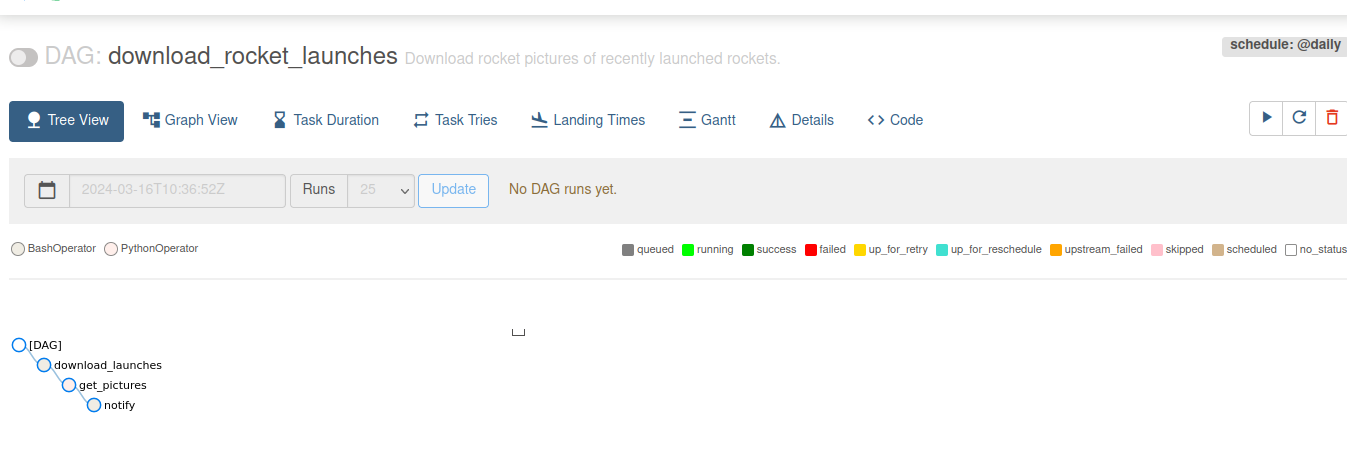


Рисунок 11. Дерево задач

1. Граф (см. рис. 12),

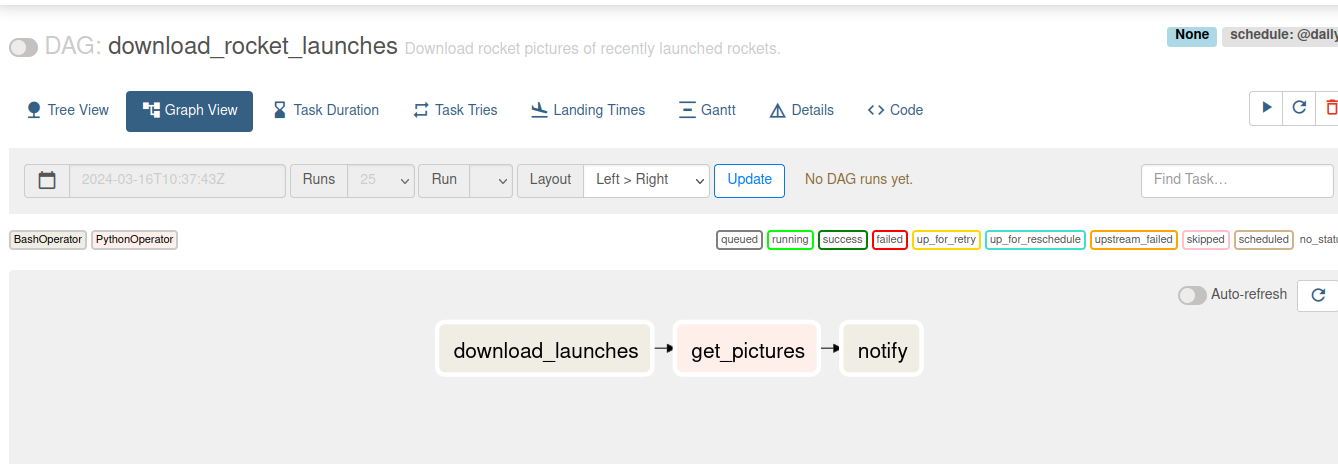


Рисунок 12. Граф

1. Диаграмма Ганта (см. рис. 13).

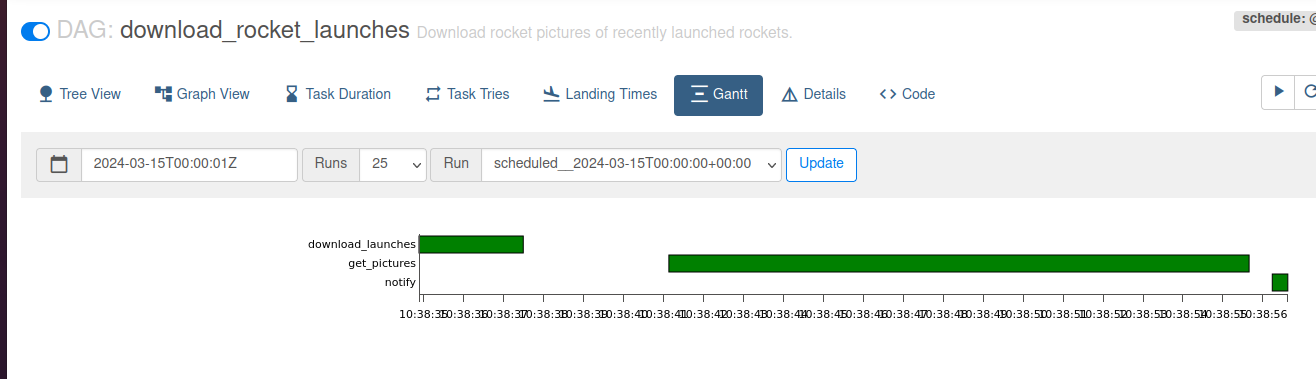


Рисунок 13. Диаграмма Ганта

Также помимо этих разделов существуют и другие, такие как, длительность задач, попытки задач, детали, код.

Итак, при запуске DAG задачи начнут выполнятся, результат работы видно на графе (см. рис. 14)

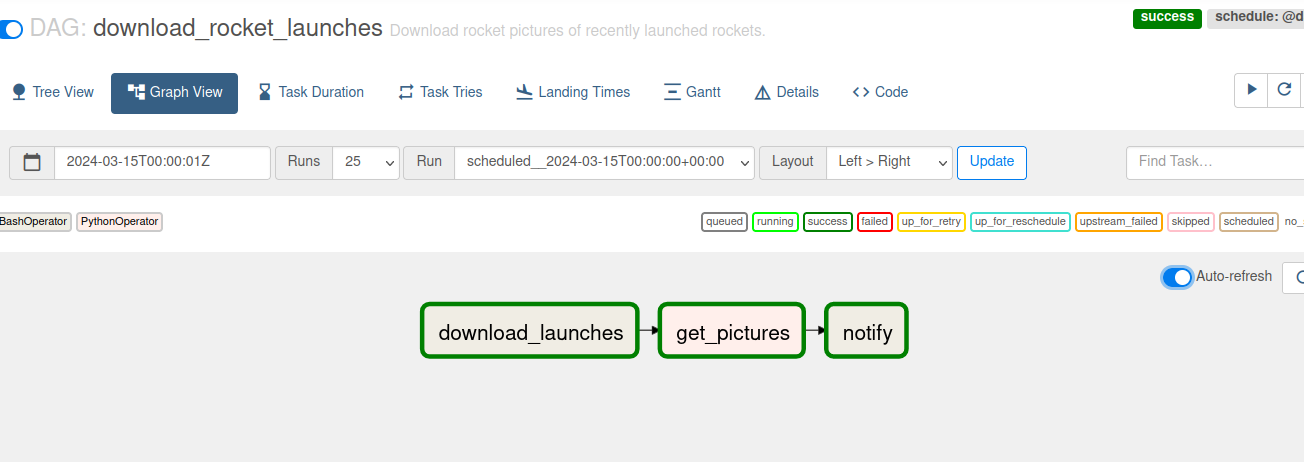


Рисунок 14. Граф отработанного DAG

Просмотр логов в файловой системе оболочки (см. рис. 15)

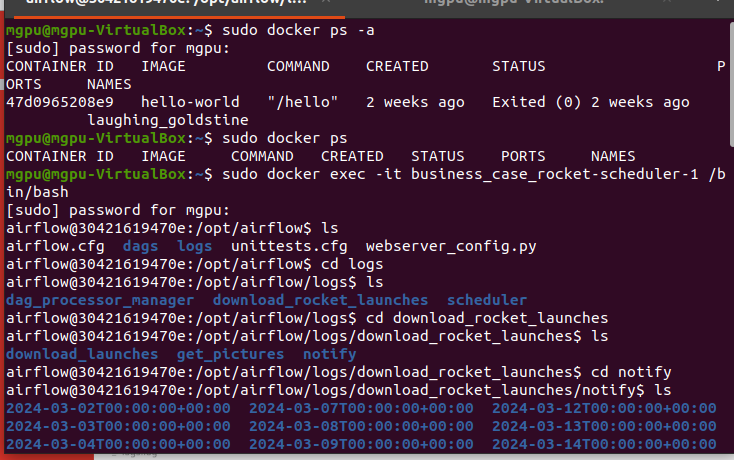


Рисунок 15. Просмотр лагов по задаче notify

Выгрузка полученного результата работы DAG в основной каталог ОС (см. рис. 16, рис. 17, рис. 18, рис. 19).

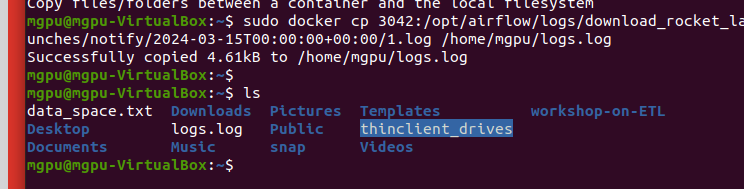


Рисунок 16. Выгрузка лога задачи notify

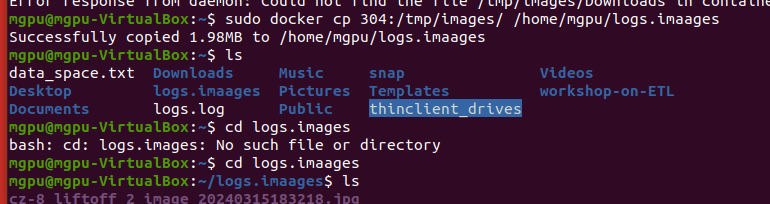


Рисунок 17. Выгрузка лога задачи get\_pictures

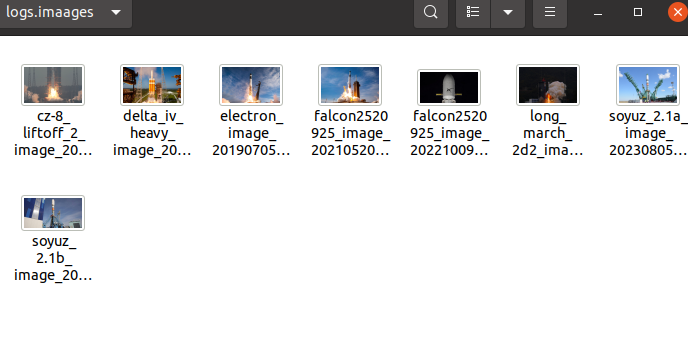


Рисунок 18. Скачанный лог с изображениями

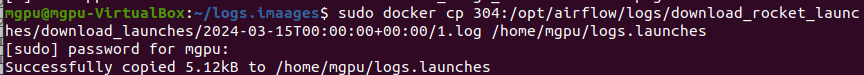


Рисунок 19. Выгрузка лога задачи download\_launches

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.

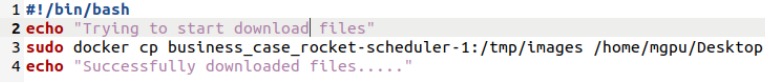


Рисунок 20. Исходный код исполняемого файла

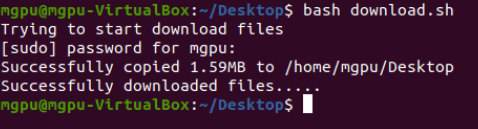


Рисунок 21. Результат работы исполняемого файла

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

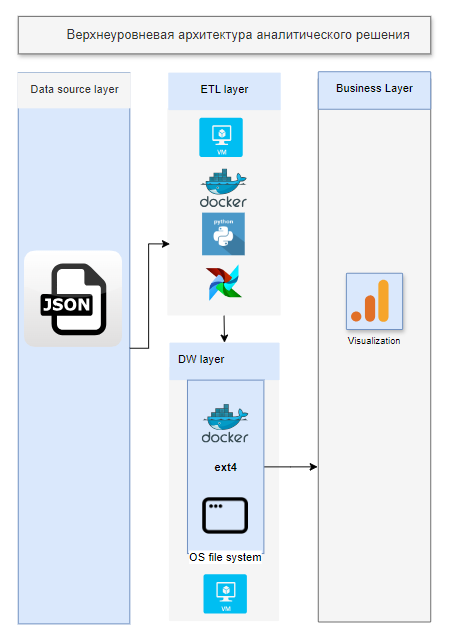


Рисунок 22. Верхнеуровневая архитектура аналитического решения.

5.1.6. Спроектировать архитектуру DAG Бизнес-кейса «Rocket».



Рисунок 23. Архитектура DAG Бизнес-кейса «Rocket»

5.1.7. Построить диаграмму Ганта работы DAG в Apache Airflow.

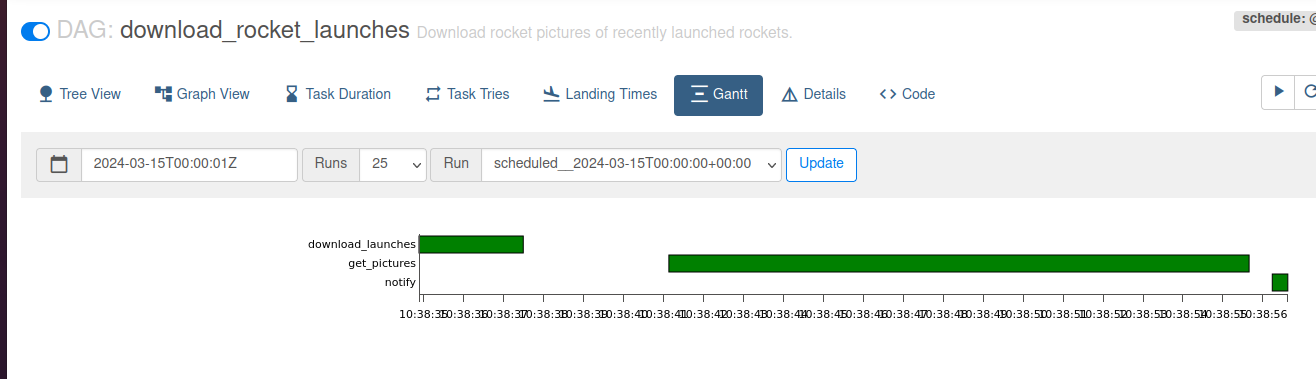


Рисунок 24. Диаграмма Ганта

Исходный код DAGs

import json

import pathlib

import airflow

import requests

import requests.exceptions as requests\_exceptions

from airflow import DAG

from airflow.operators.bash import BashOperator

from airflow.operators.python import PythonOperator

dag = DAG(

dag\_id="listing\_2\_10",

start\_date=airflow.utils.dates.days\_ago(14),

schedule\_interval="@daily",

)

download\_launches = BashOperator(

task\_id="download\_launches",

bash\_command="curl -o /tmp/launches.json -L 'https://ll.thespacedevs.com/2.0.0/launch/upcoming'", # noqa: E501

dag=dag,

)

def \_get\_pictures():

# Ensure directory exists

pathlib.Path("/tmp/images").mkdir(parents=True, exist\_ok=True)

# Download all pictures in launches.json

with open("/tmp/launches.json") as f:

launches = json.load(f)

image\_urls = [launch["image"] for launch in launches["results"]]

for image\_url in image\_urls:

try:

response = requests.get(image\_url)

image\_filename = image\_url.split("/")[-1]

target\_file = f"/tmp/images/{image\_filename}"

with open(target\_file, "wb") as f:

f.write(response.content)

print(f"Downloaded {image\_url} to {target\_file}")

except requests\_exceptions.MissingSchema:

print(f"{image\_url} appears to be an invalid URL.")

except requests\_exceptions.ConnectionError:

print(f"Could not connect to {image\_url}.")

get\_pictures = PythonOperator(

task\_id="get\_pictures", python\_callable=\_get\_pictures, dag=dag

)

notify = BashOperator(

task\_id="notify",

bash\_command='echo "There are now $(ls /tmp/images/ | wc -l) images."',

dag=dag,

)

download\_launches >> get\_pictures >> notify