Этот DAG предназначен для взаимодействия с погодными API и отправки полученных данных в чат Telegram. Он извлекает данные о погоде из сервисов Yandex и OpenWeather, обрабатывает их, записывает результат в БД MySQL и может использоваться для отправки этих данных в Telegram (отправку в telegram запустить не удалось).

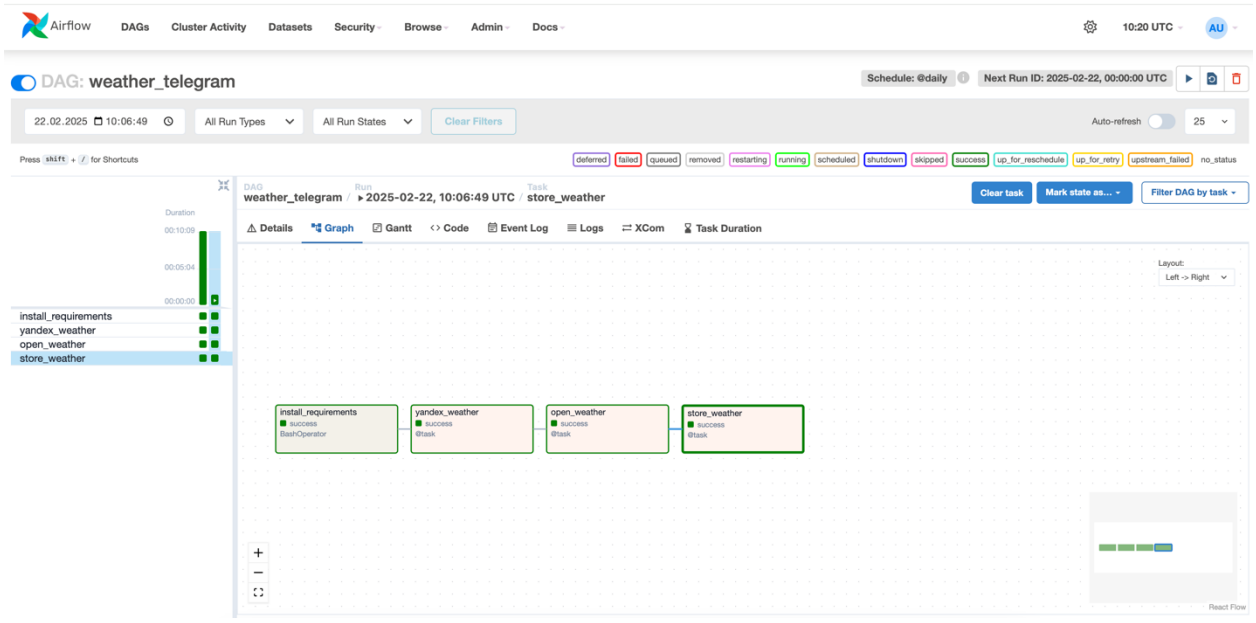
Вот подробное описание DAG:

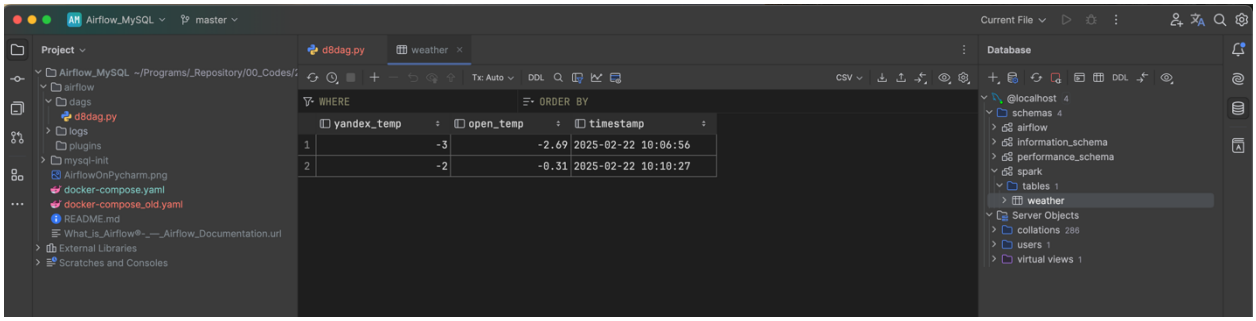
```
# docker exec -it -u root airflow-webserver bash && apt-get update && apt-get
upgrade
# pip install apache-airflow-providers-telegram python-telegram-bot

import datetime
import os
import requests
import pendulum
import pandas as pd
from airflow.decorators import dag, task
from airflow.hooks.base import BaseHook
from airflow.models import Variable
from sqlalchemy import create_engine
from airflow.operators.bash import BashOperator

# Отключаем прокси, если они мешают
os.environ["no_proxy"] = "*"

YANDEX_API_KEY = '33f45b91-bcd4-46e4-adc2-33cfdbbdd88e'
OPENWEATHER_API_KEY = '2cd78e55c423fc81cebc1487134a6300'
TELEGRAM_TOKEN = '7248934509:AAE4KtbM5wDjmrU-xFH8bMAwBwDZw-C_cMA'
CHAT_ID = '783160683'

# Дефолтные аргументы DAG
default_args = {
    "owner": "airflow",
    "retries": 3,
}


@dag(
    dag_id="weather_telegram",
    schedule="@daily",
    start_date=pendulum.datetime(2023, 1, 1, tz="UTC"),
    catchup=False,
    dagrun_timeout=datetime.timedelta(minutes=60),
    default_args=default_args,
    tags=["weather", "telegram"],
)
def weather_etl():
    # Шаг 1: Установка необходимых зависимостей
    install_requirements = BashOperator(
        task_id="install_requirements",
        bash_command=(
            "pip install apache-airflow-providers-telegram python-telegram-
bot "
            "requests pandas sqlalchemy "
            "apache-airflow-providers-mysql"
        ),
    )

    @task(task_id="yandex_weather")
    def get_yandex_weather(**kwargs):
        ti = kwargs["ti"]
        url =
"https://api.weather.yandex.ru/v2/informers/?lat=55.75396&lon=37.620393"
        headers = {"X-Yandex-API-Key": YANDEX_API_KEY}
```

```
        response = requests.get(url, headers=headers)
        if response.status_code != 200:
            raise ValueError(f"Ошибка API Яндекса: {response.status_code},
{response.text}")

        temp = response.json().get("fact", {}).get("temp")
        if temp is None:
            raise ValueError("Не удалось получить температуру из ответа API
Яндекса")

        ti.xcom_push(key="weather", value=temp)
        return temp

    @task(task_id="open_weather")
    def get_open_weather(**kwargs):
        ti = kwargs["ti"]
        url =
f"https://api.openweathermap.org/data/2.5/weather?lat=55.75&lon=37.61&appid={
OPENWEATHER_API_KEY}"

        response = requests.get(url)
        if response.status_code != 200:
            raise ValueError(f"Ошибка API OpenWeather:
{response.status_code}, {response.text}")

        temp_k = response.json().get("main", {}).get("temp")
        if temp_k is None:
            raise ValueError("Не удалось получить температуру из ответа
OpenWeather")

        temp_c = round(temp_k - 273.15, 2)
        ti.xcom_push(key="open_weather", value=temp_c)
        return temp_c

    @task(task_id="store_weather")
    def store_weather(**kwargs):
        ti = kwargs["ti"]

        yandex_temp = ti.xcom_pull(task_ids="yandex_weather", key="weather")
        open_temp = ti.xcom_pull(task_ids="open_weather", key="open_weather")
        timestamp = datetime.datetime.now()

        # Настройки подключения к MySQL
        MYSQL_USER = "airflow"
        MYSQL_PASSWORD = "airflow"
        MYSQL_HOST = "mysql-db"
        MYSQL_DB = "spark"

        # Подключение к MySQL через SQLAlchemy с таймаутом
        con =
create_engine(f"mysql+pymysql://{MYSQL_USER}:{MYSQL_PASSWORD}@{MYSQL_HOST}:33
06/{MYSQL_DB}",
                              connect_args={'connect_timeout': 10})

        # db_conn = BaseHook.get_connection("mysql_conn")  # Используем
Airflow Connection
        # engine = create_engine(
        #
f"mysql://{db_conn.login}:{db_conn.password}@{db_conn.host}:{db_conn.port}/{d
b_conn.schema}")

        data = [[yandex_temp, open_temp, timestamp]]
        df = pd.DataFrame(data, columns=["yandex_temp", "open_temp",
"timestamp"])
```

```
        df.to_sql("weather", con=con, if_exists="append", index=False)

    # from airflow.providers.telegram.operators.telegram import
TelegramOperator
    #
    # send_message_telegram_task = TelegramOperator(
    #     task_id="send_message_telegram",
    #     telegram_conn_id="telegram_conn",
    #     token=TELEGRAM_TOKEN,
    #     chat_id=CHAT_ID,
    #     text="""
    #         Weather in Moscow:
    #         Yandex: {{ ti.xcom_pull(task_ids='yandex_weather',
key='weather') }} degrees
    #         OpenWeather: {{ ti.xcom_pull(task_ids='open_weather',
key='open_weather') }} degrees
    #     """,
    # )

    # Определяем последовательность задач
    install_requirements >> get_yandex_weather() >> get_open_weather() >>
store_weather()
    # >> send_message_telegram_task()


dag = weather_etl()
```

Результат выполнения:





Общее описание:

Этот DAG включает следующие задачи:

1. **Получение погоды из Yandex**: Извлекает данные о температуре из API Yandex Weather, используя ключ API Yandex и координаты (широту и долготу) конкретного местоположения.

2. **Получение погоды из OpenWeather**: Извлекает данные о температуре из API OpenWeather, переводит температуру из Кельвина в Цельсия и сохраняет результат в XCom.

3. **Отправка погоды в Telegram**: Отправляет собранную информацию о погоде (из Yandex и OpenWeather) в указанный чат Telegram через бота.

Ключевые компоненты:

• **API ключ Yandex**: Используется для получения данных о погоде из Yandex.

• **API ключ OpenWeather**: Используется для получения данных о погоде из OpenWeather.

• **Токен Telegram и Chat ID**: Используются для отправки данных о погоде в чат Telegram.

Основные шаги:

1. **Задача погоды Yandex (get_yandex_wether)**:

• Эта функция вызывает API Yandex Weather с заранее заданными координатами (широта и долгота).

• Функция извлекает текущую температуру из данных Yandex и сохраняет её в XCom.

2. **Задача погоды OpenWeather (get_open_wether)**:

• Эта функция вызывает API OpenWeather с фиксированными координатами, затем обрабатывает температуру, преобразуя её из Кельвинов в Цельсии.

• Результат сохраняется в XCom для дальнейшего использования.

3. **Задача отчетности о погоде (get_wether)**:

• Эта функция извлекает данные о погоде из XCom для обоих источников (Yandex и OpenWeather).

• Форматирует данные и может использоваться для вывода информации или отправки данных в другие системы, например, в Telegram.