Работа с данными бизнеса в Airflow

- Автор: Егорова Ольга

Цели и задачи проекта

Сервис Электронных книг предоставляет доступ к контенту разных форматов, включая текст, аудио и не только. Построим пайплайн в Airflow, который будет запускать PySpark-скрипт для обработки данных сервиса и создания витрин, на основе которых можно будет быстрее и проще готовить отчёты.

Описание данных

Таблица `bookmate.audition` содержит данные об активности пользователей и включает столбцы:

- `audition_id` — уникальный идентификатор сессии чтения или прослушивания;
- `puid` — идентификатор пользователя;
- `usage_platform_ru` — название платформы, с помощью которой пользователь взаимодействует с контентом;
- `msk_business_dt_str` — дата и время события (строка, часовой пояс — МСК);
- `app_version` — версия приложения;
- `adult_content_flg` — значение, которое показывает, был ли контент для взрослых (`True` или `False`);
- `hours` — длительность сессии чтения или прослушивания в часах;
- `hours_sessions_long` — длительность длинных сессий в часах;
- `kids_content_flg` — значение, которое показывает, был ли это детский контент (`True` или `False`);
- `main_content_id` — идентификатор основного контента;
- `usage_geo_id` — идентификатор географического местоположения пользователя.

Таблица `bookmate.content` включает столбцы:

- `main_content_id` — идентификатор основного контента;
- `main_author_id` — идентификатор основного автора контента;
- `main_content_type` — тип контента: аудио, текст или другой;
- `main_content_name` — название контента;
- `main_content_duration_hours` — длительность контента в часах;
- `published_topic_title_list` — список жанров или тем контента.

1. ClickHouse

Первое, что нужно сделать - это создать пустую таблицу для записи данных.

Идём в DBeaver и подключаемся к ClickHouse. Для подключения потребуются следующие парамтры:

- `'dbname'` — название базы данных;
- `'host'` — адрес сервера;
- `'password'` — пароль;
- `'port'` — порт;
- `'user'` — пользователь.

С помощью SQL-запроса создаём пустую таблицу для будущего агрегата, назовем её `bookmate_user_aggregate`

```
CREATE TABLE bookmate_user_aggregate
(
    puid String PRIMARY KEY,
    audition_count Int64,
    avg_hours Float32
)
ENGINE = MergeTree()
```

2. Spark-код

Следующим шагом готовим Spark-код и этот скрипт запишем в файл с названием `my_spark_job.py`.

Полный путь к этому файлу будет выглядеть так: `s3a://da-plus-dags/{user}/jobs/my_spark_job.py` , где user=`'**********'`

- DAG при запуске должен передавать в приложение дату выполнения, и по этой дате скрипт должен находить данные в соответствующей папке.

В S3 каждый день появляется файл с данными и он всегда называется одинаково `audition_content.csv`, но сохраняется в разные папки, имя папок имеет формат `data_YYYY_MM_DD`, например `data_2025_01_01`

Когда Airflow запускает наш скрипт, он передать ему параметр — дату выполнения задачи. Наш скрипт "ловит" эту дату и преобразует к виду YYYY_MM_DD: `sys.argv[1].replace('-', '_')`

- Далее Spark должен считать CSV-файл и выполнить агрегацию

```
from pyspark.sql import SparkSession
import pyspark.sql.functions as F
import sys

# Создаём Spark-сессию и при необходимости добавляем конфигурации
spark = SparkSession.builder.appName("myAggregateTest") \
    .config("fs.s3a.endpoint", "storage.yandexcloud.net") \
    .getOrCreate()

# Указываем порт и параметры кластера ClickHouse
jdbcPort = 8443
jdbcHostname = "rc1a-3jouval14nne7aun.mdb.yandexcloud.net"
jdbcDatabase = "playground_da_20250825_4080002560"
jdbcUrl = f"jdbc:clickhouse://{jdbcHostname}:{jdbcPort}/{jdbcDatabase}?ssl=true"

# Получаем аргумент из Airflow
my_date = sys.argv[1].replace('-', '_')

# Считываем исходные данные за нужную дату
df = spark.read.csv(f"s3a://da-plus-dags/script_bookmate/data_{my_date}/audition_content.csv", inferSchema=True, header=True)

# Строим агрегат по пользователям
result_df = df.groupBy("puid").agg(
    F.countDistinct("audition_id").alias("audition_count"),
    F.avg("hours").alias("avg_hours")
)

# Результат агрегации записываем в таблицу ClickHouse с помощью JDBC-подключения
result_df.write.format("jdbc") \
    .option("url", jdbcUrl) \
    .option("user", "**********") \
    .option("password", "******************************") \
    .option("dbtable", "bookmate_user_aggregate") \
    .mode('append') \
    .save()
```

3. Создание DAG

Теперь, когда Spark-код готов, нужно создать DAG, который будет его запускать.

Зададим параметры для DAG:

- Обязательно укажем уникальный идентификатор (`dag_id`)
- DAG будет запускаться каждый день (здесь используем псевдоним `@daily`)
- Обязательно укажем начало запуска с 1 января 2025 года (`start_date=datetime(2025, 1, 1)`)
- При этом запускать DAG за пропущенные даты не будем, пусть он начнет работать с текущего момента (`catchup=False`)

➕

Добавим проверку входного файла - DAG не должен стартовать, пока в S3 не появится файл с данными за нужную дату. Здесь будем использовать сенсор `S3KeySensor`:

- Обязательно укажем уникальный идентификатор задачи в DAG `task_id='wait_for_audition_content'`
- Укажем имя бакета, в котором нужно проверить наличие файла (в данном случае это `bucket_name='da-plus-dags'`)
- Укажем путь к файлу, который нужно найти. При этом мы помним, что имя папки где хранится файл ежедневно меняется, поэтому задаем динамическое имя папки - `data_{{ds.replace('-', '_')}}`. Тогда путь будет выглядеть так: `bucket_key="script_bookmate/data_{{ds.replace('-', '_')}}/audition_content.csv"`
- Пусть сенсор ищет точное совпадение с bucket_key, то есть нам нужен только указанный файл (`wildcard_match-False`).
- Укажем идентификатор подключения (`aws_conn_id`) к S3 в Airflow, в котором хранятся ключи и данные для доступа к хранилищу.
- Укажем как часто сенсор будет проверять наличие файла. Сделаем каждые 5 минут (`poke_interval`)
- Укажем максимальное время ожидания 1 час. (`timeout`)
- Пусть сенсор остается активным в памяти и проверяет условие через заданный интервал (`mode='poke'`)

➕

Создадим задачу:

Чтобы запускать Spark-приложения на кластере прямо из Airflow, создадим собственный оператор. Для этого используем механизм наследования Python. Мы берём родительский класс `DataprocCreatePysparkJobOperator` и на его основе создаём новый класс `PysparkJobOperator`, который унаследует все поля и методы исходного класса, но будет усовершенствован.

❓ Почему используем именно оператор `DataprocCreatePysparkJobOperator`?
❗ Так как мы используем сервисы Яндекса, то другой оператор, применяемый обычно для похожих задач - `SparkSubmitOperator` в нашем случае не подойдет.

❓ Зачем создавать собственный оператор, а не воспользоваться `DataprocCreatePysparkJobOperator`?
❗ У оператора `DataprocCreatePysparkJobOperator` есть параметр `args` — это список аргументов, которые будут переданы в

PySpark-скрипт. То есть можно было бы в качестве аргумента передать дату и записать `args=["{{ ds }}"]`. НО по умолчанию `args` не шаблонизируется. Это значит, внутрь скрипта передастся строка `{{ ds }}`. То есть вместо ожидаемого `data_2025-09-01` получим `data_{{ ds }}`
ПОЭТОМУ
Мы создаем новый оператор, при этом используем функции оператора `DataprocCreatePysparkJobOperator` и добавим к ним новые, а именно сделаем так, чтобы `args` шаблонизировался. Тогда наш новый оператор будет корректно воспринимать переменные и подставлять их значения.

❓ Как сделать `args` шаблонизируемым?
❗ Поле `template_fields` внутри `DataprocCreatePysparkJobOperator` определяет, какие аргументы могут принимать переменные Airflow. В исходном коде поле `template_fields` класса `DataprocCreatePysparkJobOperator`:

```
template_fields = ("cluster_id", )
```

Это значит, что только `cluster_id` поддерживает шаблоны Airflow вроде `{{ ds }}`.

В нашем случае также нужны шаблоны и в `args`. Поэтому переопределим поле `template_fields` и указываем все поля, в которых должна происходить замена:

```
template_fields = ("cluster_id", "args",)
```

Теперь и `cluster_id`, и `args` поддерживают шаблоны Airflow.

- Обязательно укажем уникальный идентификатор задачи в Airflow `task_id`
- Обязательно укажем уникальное имя задания Dataproc `name`
- В приложение нужно передать дату запуска DAG, чтобы скрипт обработал данные за этот день `args= ["{{ds}}"]`
- Укажем путь к файлу Spark-приложения `main_python_file_uri`
- Укажем на каком кластере запускать задание `cluster_id`

➕

И последним указываем зависимости

```
    #Импортируем модуль для работы с датой и временем
    from datetime import datetime
    # Импортируем класс DAG
    from airflow import DAG
    #Импортируем сенсор
    from airflow.sensors.s3_key_sensor import S3KeySensor
    #Импортируем оператор
    from airflow.providers.yandex.operators.dataproc import DataprocCreatePysparkJobOperator

    #создаем новый класс на основе родительского DataprocCreatePysparkJobOperator
    class PysparkJobOperator(DataprocCreatePysparkJobOperator):
        #переопределяем поле
        template_fields = ("cluster_id", "args",)

    DAG_ID = "audition_content_analysis"

    with DAG(
        # Указываем необходимые параметры:
        #задаем уникальный идентификатор DAG
        dag_id=DAG_ID,
        #задаем расписание запусков -  каждый день
        schedule_interval="@daily",
        # задаем дату старта с 1 января 2025 года
        start_date=datetime(2025, 1, 1),
        #можно указать теги графа
        tags=["audition_content", "aggregate"],
        #режим генерации пропущенных интервалов отключаем
        catchup=False
    ) as dag:
        # Создаем экземпляр сенсора, который будет ждать появления входного файла в S3
        wait_for_input = S3KeySensor(
            #задаем уникальный идентификатор задачи в DAG
            task_id='wait_for_audition_content',
            #проверка каждые 5 минут, то есть 300 сек.
            poke_interval=300,
            #задаем время ожидания 1 час, то есть 3600 сек.
            timeout=3600,
            #указываем имя бакета, в котором нужно проверить наличие файла
            bucket_name='da-plus-dags',
            #указываем путь к файлу, который нужно найти
            bucket_key= "script_bookmate/data_{{ds.replace('-', '_')}}/audition_content.csv",
            #указываем режим работы сенсора - активен все время
            mode='poke',
            # указываем идентификатор подключения к S3 в Airflow
            aws_conn_id='s3',
            # ищем точное совпадение с bucket_key
            wildcard_match=False
        )

        # Запускаем PySpark-задание на кластере Dataproc
        user = '***********'

        run_pyspark = PysparkJobOperator(
            # уникальное имя для задания Dataproc
            name="create_aggregate_and_load",
```

```
        # идентификатор задачи в Airflow
        task_id="run_pyspark_audition_content",
        #идентификатор кластера Dataproc
        cluster_id="c9q4134h5vi546h1e148",
        #аргументы для PySpark-скрипта
        args= ["{{ds}}"],
        #указываем путь к PySpark-скрипту
        main_python_file_uri=f"s3a://da-plus-dags/{user}/jobs/my_spark_job.py"
    )

    # Зависимости
    wait_for_input >> run_pyspark
```

5. Результат

Когда DAG вместе с задачами отработает без ошибок, убедимся, что данные успешно загрузились в ClickHouse.

Для этого выполним в DBeaver запрос:

```
SELECT *
FROM bookmate_user_aggregate
LIMIT 100;
```

Мы должны увидеть агрегированные данные, подготовленные в Spark-приложении.

Результат выполнения запроса - несколько строк из таблицы с данными:

```
puid                                |audition_count|avg_hours   |
------------------------------------+--------------+------------+
68296d6c-f9d6-11ef-be00-c2c9fa6fd3d5|             3|  0.10666667|
68298856-f9d6-11ef-be00-c2c9fa6fd3d5|             2|        1.39|
68298950-f9d6-11ef-be00-c2c9fa6fd3d5|            16| 0.102326386|
6829cfdc-f9d6-11ef-be00-c2c9fa6fd3d5|             1|  0.00055556|
682a015a-f9d6-11ef-be00-c2c9fa6fd3d5|             5|   2.3176112|
682a0984-f9d6-11ef-be00-c2c9fa6fd3d5|             2|       0.195|
682a8ada-f9d6-11ef-be00-c2c9fa6fd3d5|             1|        0.79|
682a9c8c-f9d6-11ef-be00-c2c9fa6fd3d5|             2|         0.1|
682b62b6-f9d6-11ef-be00-c2c9fa6fd3d5|             2|   1.3256944|
682bafe6-f9d6-11ef-be00-c2c9fa6fd3d5|             2|  0.27569446|
682c0ac2-f9d6-11ef-be00-c2c9fa6fd3d5|             1|        0.05|
682c72a0-f9d6-11ef-be00-c2c9fa6fd3d5|             2|        0.17|
682ddf64-f9d6-11ef-be00-c2c9fa6fd3d5|             7| 0.111428574|
```

Резюмируем:

1. Например сейчас 1 сентября 2025 год 12 часов дня. Мы только что загрузили DAG (без догоняющих запусков !). DAG не запустится за сегодняшнюю дату автоматически.
2. Первый запуск произойдет завтра 2 сентября 2025 года в 00:00. Сенсор будет искать файл каждые 5 минут на протяжении 1 часа.
3. Если до 01:00 сенсор НЕ найдет файл, то DAG упадет с таймаутом. И следующая попытка поиска файла будет 3 сентября 2025 года в 00:00.
4. Если до 01:00 сенсор найдет файл, то цикл проверок прерывается и сенсор возвращает TRUE. Airflow переходит к выполнению следующей задачи - запуску PySpark-задания. В результате мы получим агрегированные данные записанные в ClickHouse.
5. Использование переменной `{{ ds }}` позволяет автоматически подставлять даты в пути к файлам. Благодаря этому пайплайн не нужно вручную переписывать под каждую дату.