

****Дипломная работа по курсу ****

Цель: построить пайплайн через Airflow

****Задание: ****

Вам необходимо построить airflow пайплайн выгрузки ежедневных отчётов по количеству поездок на велосипедах в городе Нью-Йорк.

Рекомендации при выполнении работы:

Пайплайн должен состоять из следующих шагов:

1. Отслеживание появления новых файлов в своём бакете на AWS S3. Представим, что пользователь или провайдер данных будет загружать новые исторические данные по поездкам в Ваш бакет;
2. При появлении нового файла запускается оператор импорта данных в созданную таблицу базы данных Clickhouse;
3. Необходимо сформировать таблицы с ежедневными отчётами по следующим критериям:
 - количество поездок в день
 - средняя продолжительность поездок в день
 - распределение поездок пользователей, разбитых по категории «gender»
4. Данные статистики необходимо загрузить на специальный S3 бакет с хранящимися отчётами по загруженным файлам.

0. Корректировка задания

- т.к. новые регистрации в AWS недоступны, исходные файлы переложим в Object Storage YandexCloud, файлы с итоговыми отчетами будем сохранять туда же





- с февраля 2021 г. в кампании изменилась политика учета поездок и клиентов, в связи с этим уже больше года отсутствует колонка “gender”, поэтому делить будем по другому атрибуту – “member_casual”

- в задании требуется настроить пайплайн выгрузки ежедневных отчетов, однако исследование файлов с данными показывает, что каждый содержит информацию за месяц и их загрузка носит хаотический характер (были периоды, когда единоразово загружались данные за 7-9 месяцев, но бывали и периоды, когда загрузка происходила регулярно в начале месяца за предыдущий). Поэтому, в реальной ситуации более логичным было бы воспользоваться возможностями YandexCloud, а именно сервисом Cloud Function и попробовать настроить функцию (импорт из S3 данных в основную таблицу, подсчет и занесение данных в таблицы отчетов, перезапись файлов отчетов), срабатывающую по триггеру – появление нового файла в Object Storage. Но тогда Airflow будет не нужен. Значит, исходные таблицы и отчеты делаем для ежемесячных данных, а потом представим, что у нас ежедневные поступления данных и будем строить пайплайн исходя из этого.

1. Разбираемся в исходных данных. Чтобы разработать пайплайн, нужно понимать, что за данные и как с ними работать.

Citi Bike — это крупнейшая в Америке программа проката велосипедов, включающая 25 000 велосипедов и более 1 500 станций на Манхэттене, Бруклине и пр. Данные публикуются самой компанией. Структура данных описана на сайте <https://ride.citibikenyc.com/system-data>

Файлы с данными в хранилище:

<input type="checkbox"/>	Имя	Размер
<input type="checkbox"/>	 202111-citibike-tripdata.csv	386.31 МБ
<input type="checkbox"/>	 202112-citibike-tripdata.csv	312.88 МБ
<input type="checkbox"/>	 202201-citibike-tripdata.csv	188.31 МБ
<input type="checkbox"/>	 202202-citibike-tripdata.csv	221.43 МБ

Видно, что URL-адреса следуют фиксированному шаблону YYYYmm-citibike-tripdata.csv/

Ссылка на один из файлов: <https://storage.yandexcloud.net/citibike-input/202202-citibike-tripdata.csv>

Скачаем один из файлов и проверим данные вручную.

```
df=pd.read_csv('https://storage.yandexcloud.net/citibike-input/202202-citibike-tripdata.csv')
df.head()
```

	ride_id	rideable_type	started_at	ended_at	start_station_name	start_station_id	end_station_name	end_station_id	start_lat	start_lng	end_lat	end_lng	member_casual
0	6C29577D0141AACD	electric_bike	2022-02-27 16:00:58	2022-02-27 16:10:09	Cliff St & Fulton St	5065.14	Norfolk St & Broome St	5374.01	40.708380	-74.004950	40.717227	-73.988021	member
1	353462A61AFB32C8	classic_bike	2022-02-02 18:09:30	2022-02-02 18:28:55	2 Ave & E 31 St	6197.02	Norfolk St & Broome St	5374.01	40.742909	-73.977061	40.717227	-73.988021	member
2	220081222C7CC84D	classic_bike	2022-02-10 19:41:09	2022-02-10 19:46:03	2 Ave & E 31 St	6197.02	5 Ave & E 29 St	6248.06	40.742909	-73.977061	40.745168	-73.986831	member
3	C66CB5EDE74A7404	classic_bike	2022-02-14 15:52:02	2022-02-14 16:01:58	Bridge St & Water St	4968.03	Henry St & Atlantic Ave	4531.05	40.702969	-73.984691	40.690893	-73.996123	member
4	6B997C1E4DCDD6BF	classic_bike	2022-02-23 13:25:07	2022-02-23 15:36:45	Lafayette St & Jersey St	5561.06	Fulton St & Broadway	5175.08	40.724561	-73.995653	40.711066	-74.009447	member

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1233714 entries, 0 to 1233713
Data columns (total 13 columns):
#   Column                Non-Null Count  Dtype
---  ---
0   ride_id                1233714 non-null object
1   rideable_type          1233714 non-null object
2   started_at             1233714 non-null object
3   ended_at               1233714 non-null object
4   start_station_name     1233713 non-null object
5   start_station_id       1233713 non-null object
6   end_station_name       1231390 non-null object
7   end_station_id         1231390 non-null object
8   start_lat              1233714 non-null float64
9   start_lng              1233714 non-null float64
10  end_lat                1232503 non-null float64
11  end_lng                1232503 non-null float64
12  member_casual          1233714 non-null object
dtypes: float64(4), object(9)
memory usage: 122.4+ MB
```

```
[4] df['member_casual'].unique()
```

```
array(['member', 'casual'], dtype=object)
```

```
df['rideable_type'].unique()
```

```
array(['electric_bike', 'classic_bike', 'docked_bike'], dtype=object)
```

Структура данных:

Наименование атрибута	Значение
ride_id	ID поездки
rideable_type	тип байка (принимаемые значения: electric_bike, classic_bike, docked_bike)
started_at	время начала поездки в формате YYYY-mm-dd HH:MM:SS
ended_at	время окончания поездки в формате YYYY-mm-dd HH:MM:SS
start_station_name	название станции начала поездки
start_station_id	ID станции начала поездки
end_station_name	название станции окончания поездки
end_station_id	ID станции окончания поездки
start_lat	широта точки нахождения станции начала поездки
start_lng	долгота точки нахождения станции начала поездки
end_lat	широта точки нахождения станции окончания поездки
end_lng	долгота точки нахождения станции окончания поездки
member_casual	есть годовое членство у клиента (member) или нет (casual)

Т.к. ClickHouse «любит» широкие таблицы «не любит» join, выделять станции и информацию о них в отдельный справочник не будем, а создадим в качестве основного хранилища таблицу, которая будет включать в себя все атрибуты (citibike_all).

В отчетные таблицы сделаем выборку из основной согласно выставленным условиям:

- daily_count_ride – подсчет ежедневного количества поездок;
- daily_average_duration_ride – подсчет среднеспредельной продолжительности поездок;
- daily_diff_members – распределение по member / casual.

2. Создаем базу данных ClickHouse, импортируем в нее исторические данные и считаем по ним отчеты

I вариант – из Docker-образа

Создаем базу данных CitiBike

```
767b6c16faaf :) CREATE DATABASE IF NOT EXISTS CitiBike

CREATE DATABASE IF NOT EXISTS CitiBike

Query id: 896f2e42-000a-48fd-ab50-f19e0570cecb

Ok.

0 rows in set. Elapsed: 0.052 sec.
```

Создаем основную таблицу citibike_all

```
767b6c16faaf :) CREATE TABLE CitiBike.citibike_all (ride_id String, rideable_type String, started_at DateTime, ended_at
DateTime, start_station_name String, start_station_id String, end_station_name String, end_station_id String, start_lat
String, start_lng String, end_lat String, end_lng String, member_casual String) ENGINE=MergeTree ORDER BY started_at

CREATE TABLE CitiBike.citibike_all
(
  `ride_id` String,
  `rideable_ty
  `started_at` DateTime,
  `ended_at` DateTime,
  `start_station_name` String,
  `start_station_id` String,
  `end_station_name` String,
  `end_station_id` String,
  `start_lat` String,
  `start_lng` String,
  `end_lat` String,
  `end_lng` String,
  `member_casual` String
)
ENGINE = MergeTree
ORDER BY started_at

Query id: c67f1ea5-5e1f-4266-89d4-fe0e1547d84b

Ok.

0 rows in set. Elapsed: 0.125 sec.
```

Создаем 3 таблицы для отчетов

```
767b6c16faaf :) create table CitiBike.daily_count_ride (date_ride Date, count_ride Int32) ENGINE=MergeTree ORDER BY date
_ride

CREATE TABLE CitiBike.daily_count_ride
(
  `date_ride` Date,
  `count_ride` Int32
)
ENGINE = MergeTree
ORDER BY date_ride

Query id: 5877cbaa-6d75-4a4b-8585-4b761556e748

0 rows in set. Elapsed: 0.252 sec.
```

еще 2 аналогично

Импортируем данные из самого первого (по отчетному периоду) файла csv, находящегося в хранилище. В нашем случае это будет <https://storage.yandexcloud.net/citibike-input/202111-citibike-tripdata.csv>

```
767b6c16faaf :) INSERT INTO CitiBike.citibike_all (ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_station_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual) SELECT ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_station_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual FROM s3('https://storage.yandexcloud.net/citibike-input/202111-citibike-tripdata.csv', 'CSVWithNames', 'ride_id String, rideable_type String, started_at DateTime, ended_at DateTime, start_station_name String, start_station_id String, end_station_name String, end_station_id String, start_lat String, start_lng String, end_lat String, end_lng String, member_casual String')

INSERT INTO CitiBike.citibike_all (ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_station_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual) SELECT
    ride_id,
    rideable_type,
    started_at,
    ended_at,
    start_station_name,
    start_station_id,
    end_station_name,
    end_station_id,
    start_lat,
    start_lng,
    end_lat,
    end_lng,
    member_casual
FROM s3('https://storage.yandexcloud.net/citibike-input/202111-citibike-tripdata.csv', 'CSVWithNames', 'ride_id String, rideable_type String, started_at DateTime, ended_at DateTime, start_station_name String, start_station_id String, end_station_name String, end_station_id String, start_lat String, start_lng String, end_lat String, end_lng String, member_casual String')

Query id: dbcd765f-7869-4daf-b983-4756b1c98f6f

Ok.

0 rows in set. Elapsed: 41.411 sec. Processed 2.16 million rows, 523.83 MB (52.14 thousand rows/s., 12.65 MB/s.)
```

Проверим, что получилось

```
767b6c16faaf :) select count(*) from CitiBike.citibike_all

SELECT count(*)
FROM CitiBike.citibike_all

Query id: dc9256e6-f72d-4a35-b3dc-1646b2c0bcd9

count(*)
2159284

1 rows in set. Elapsed: 0.003 sec.
```

Добавим данные еще из двух файлов <https://storage.yandexcloud.net/citibike-input/202201-citibike-tripdata.csv> и <https://storage.yandexcloud.net/citibike-input/202202-citibike-tripdata.csv>. Т.к. названия файлов у нас идентичны, отличаются только месяцем и годом, используем подстановку:

<https://storage.yandexcloud.net/citibike-input/20220{1..2}-citibike-tripdata.csv>

Также этот вариант будет удобен если нужно импортировать одновременно много файлов, а не два как в данном случае.

```
767b6c16faaf :) INSERT INTO CitiBike.citibike_all (ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_station_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual) SELECT ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_station_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual FROM s3('https://storage.yandexcloud.net/citibike-input/20220{1..2}-citibike-tripdata.csv', 'CSVWithNames', 'ride_id String, rideable_type String, started_at DateTime, ended_at DateTime, start_station_name String, start_station_id String, end_station_name String, end_station_id String, start_lat String, start_lng String, end_lat String, end_lng String, member_casual String')

INSERT INTO CitiBike.citibike_all (ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_station_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual) SELECT
    ride_id,
    rideable_type,
    started_at,
    ended_at,
    start_station_name,
    start_station_id,
    end_station_name,
    end_station_id,
    start_lat,
    start_lng,
    end_lat,
    end_lng,
    member_casual
FROM s3('https://storage.yandexcloud.net/citibike-input/20220{1..2}-citibike-tripdata.csv', 'CSVWithNames', 'ride_id String, rideable_type String, started_at DateTime, ended_at DateTime, start_station_name String, start_station_id String, end_station_name String, end_station_id String, start_lat String, start_lng String, end_lat String, end_lng String, member_casual String')

Query id: 83434c0e-6836-40c0-852f-8bb926d664b4

Ok.

0 rows in set. Elapsed: 45.391 sec. Processed 2.29 million rows, 555.38 MB (50.37 thousand rows/s., 12.24 MB/s.)
```

Будем считать, что исторические данные для работы подготовлены.

Теперь посчитаем отчеты за прошедшие периоды. Считать будем на основе данных, лежащих в основной таблице citibike_all

```
767b6c16faaf :) insert into CitiBike.daily_count_ride (date_ride, count_ride) select a.started_at::Date, count(a.ride_id) from CitiBike.citibike_all as a group by a.started_at::Date

INSERT INTO CitiBike.daily_count_ride (date_ride, count_ride) SELECT
    CAST(a.started_at, 'Date'),
    count(a.ride_id)
FROM CitiBike.citibike_all AS a
GROUP BY CAST(a.started_at, 'Date')

Query id: a99923e3-e472-4e57-830f-421e0cd03cd2

Ok.

0 rows in set. Elapsed: 0.429 sec. Processed 6.19 million rows, 179.62 MB (14.44 million rows/s., 418.79 MB/s.)

767b6c16faaf :) insert into CitiBike.daily_average_duration_ride (date_ride, average_duration_ride) select a.started_at::Date, Round((AVG(a.ended_at-a.started_at))/60, 0) from CitiBike.citibike_all as a group by a.started_at::Date

INSERT INTO CitiBike.daily_average_duration_ride (date_ride, average_duration_ride) SELECT
    CAST(a.started_at, 'Date'),
    Round(AVG(a.ended_at - a.started_at) / 60, 0)
FROM CitiBike.citibike_all AS a
GROUP BY CAST(a.started_at, 'Date')

Query id: f05dbb2f-81f2-4381-9bc4-3177bc125c77

Ok.

0 rows in set. Elapsed: 0.182 sec. Processed 6.19 million rows, 49.55 MB (33.94 million rows/s., 271.51 MB/s.)

767b6c16faaf :) insert into CitiBike.daily_diff_members (date_ride, member_casual, count_ride, average_duration_ride) select a.started_at::Date, a.member_casual, count(a.ride_id), Round((AVG(a.ended_at-a.started_at))/60, 0) from CitiBike.citibike_all as a group by a.started_at::Date, a.member_casual order by member_casual

INSERT INTO CitiBike.daily_diff_members (date_ride, member_casual, count_ride, average_duration_ride) SELECT
    CAST(a.started_at, 'Date'),
    a.member_casual,
    count(a.ride_id),
    Round(AVG(a.ended_at - a.started_at) / 60, 0)
FROM CitiBike.citibike_all AS a
GROUP BY
    CAST(a.started_at, 'Date'),
    a.member_casual
ORDER BY member_casual ASC

Query id: 4b6863d7-67e7-452a-8b1c-bd6e737e4881

Ok.

0 rows in set. Elapsed: 0.543 sec. Processed 6.19 million rows, 297.30 MB (11.40 million rows/s., 547.31 MB/s.)
```

Посмотрим, что получилось

```
67b6c16faaf :) select * from CitiBike.daily_diff_members limit 10
```

```
SELECT *  
FROM CitiBike.daily_diff_members  
LIMIT 10
```

Query id: e7c7aff6-3994-4752-a59c-a4cf2558f872

date Ride	member_casual	count Ride	average_duration Ride
2021-11-01	casual	18633	23
2021-11-01	member	71022	14
2021-11-02	casual	15221	22
2021-11-02	member	67590	14
2021-11-03	casual	15757	21
2021-11-03	member	71502	14
2021-11-04	casual	16031	22
2021-11-04	member	69727	14
2021-11-05	casual	18955	26
2021-11-05	member	67324	14

10 rows in set. Elapsed: 0.017 sec.

```
67b6c16faaf :) select * from CitiBike.daily_average_duration Ride limit 5
```

```
SELECT *  
FROM CitiBike.daily_average_duration Ride  
LIMIT 5
```

Query id: a266f91e-3477-4c44-99c7-68edc8e44f8c

date Ride	average_duration Ride
2021-11-01	16
2021-11-02	15
2021-11-03	15
2021-11-04	15
2021-11-05	17

5 rows in set. Elapsed: 0.016 sec.

```
67b6c16faaf :) select * from CitiBike.daily_count Ride limit 5
```

```
SELECT *  
FROM CitiBike.daily_count Ride  
LIMIT 5
```

Query id: c7f06b40-3744-44c6-8980-652f7e4ecf05

date Ride	count Ride
2021-11-01	89655
2021-11-02	82811
2021-11-03	87259
2021-11-04	85758
2021-11-05	86279

5 rows in set. Elapsed: 0.013 sec.

```
67b6c16faaf :)
```

Сохраним файлы отчетов в хранилище

```
767b6c16faaf :) INSERT INTO FUNCTION s3('https://storage.yandexcloud.net/citibike-daily-report/daily_diff_members.csv',
'CSV', 'date_ride Date, member_casual String, count_ride Int32, average_duration_ride Float32') SELECT date_ride, member
_casual, count_ride, average_duration_ride FROM CitiBike.daily_diff_members

INSERT INTO FUNCTION s3('https://storage.yandexcloud.net/citibike-daily-report/daily_diff_members.csv', 'CSV', 'date_ride
Date, member_casual String, count_ride Int32, average_duration_ride Float32') SELECT
    date_ride,
    member_casual,
    count_ride,
    average_duration_ride
FROM CitiBike.daily_diff_members

Query id: 74f23fe0-1300-48ee-b792-81b8672dd7e6

Ok.

0 rows in set. Elapsed: 0.156 sec.

767b6c16faaf :) INSERT INTO FUNCTION s3('https://storage.yandexcloud.net/citibike-daily-report/daily_count_ride.csv', 'C
SV', 'date_ride Date, count_ride Int32') SELECT date_ride, count_ride FROM CitiBike.daily_count_ride

INSERT INTO FUNCTION s3('https://storage.yandexcloud.net/citibike-daily-report/daily_count_ride.csv', 'CSV', 'date_ride
Date, count_ride Int32') SELECT
    date_ride,
    count_ride
FROM CitiBike.daily_count_ride

Query id: 1cb0cf32-e3a9-4ae2-ab7c-b13ecf9bf32c

Ok.

0 rows in set. Elapsed: 0.174 sec.

767b6c16faaf :) INSERT INTO FUNCTION s3('https://storage.yandexcloud.net/citibike-daily-report/daily_average_duration_ri
de.csv', 'CSV', 'date_ride Date, average_duration_ride Float32') SELECT date_ride, average_duration_ride FROM CitiBike.d
aily_average_duration_ride

INSERT INTO FUNCTION s3('https://storage.yandexcloud.net/citibike-daily-report/daily_average_duration_ride.csv', 'CSV',
'date_ride Date, average_duration_ride Float32') SELECT
    date_ride,
    average_duration_ride
FROM CitiBike.daily_average_duration_ride










Query id: 2ede52f2-6886-41b5-b047-a5aef1b0d375

Ok.

0 rows in set. Elapsed: 0.172 sec.
```

Все три файла отчета появились в хранилище в своем бакете

console.cloud.yandex.ru/folders/b1g1t4frli2efslp8742/storage/buckets/citibike-daily-report

cloud-tj1807	default	Object Storage / Бакеты / citibike-daily-report
 citibike-daily-report Бакет - 11.29 КБ	<input type="text" value="Имя объекта"/>	
 Объекты	<input type="checkbox"/> Имя	Размер
 Веб-сайт	<input type="checkbox"/>  daily_average_duration_ride.csv	1.88 КБ
 Жизненный цикл	<input type="checkbox"/>  daily_count_ride.csv	2.22 КБ
 CORS	<input type="checkbox"/>  daily_diff_members.csv	7.19 КБ
		Класс хранилища
		Последнее изменение
		
		Стандартное
		02 июня 2022 16:20
		...
		Стандартное
		02 июня 2022 16:19
		...
		Стандартное
		02 июня 2022 16:09
		...

II вариант – в облаке

Создадим кластер ClickHouse в облаке

← → ↺ console.cloud.yandex.ru/folders/b1g1t4fri2efslp8742/managed-clickhouse/cluster/c9qvmgukfejdso9ptssc/view

cloud-tj1807 default Managed Service for ClickHouse / Кластеры / clickhouse882 ☆

clickhouse882
Alive

Обзор

Шарды

Группы шардов

Хосты

Словари

Машинное обучение

Схемы формата данных

SQL

DataLens

Операции

Резервные копии

Мониторинг

Логи

Топология

Документация

Начать работу с ClickHouse

Управление пользователями БД

SQL-запросы в консоли управления

Подключиться к БД в кластере ClickHouse

Шардирование

Обзор

Общее

Идентификатор.....c9qvmgukfejdso9ptssc

Дата создания.....27 мая 2022, в 08:47

Окружение.....PRESTABLE

Облачная сеть.....default

Версия.....22.3

Описание.....test 2

Состояние кластера **Alive**

Все хосты работают нормально, все запущенные операции были успешно выполнены.

Ресурсы

CLICKHOUSE shard1

Класс хоста
b2.nano (2 vCPU, 5% vCPU rate, 2 ГБ RAM)

Хранилище
10 ГБ network-hdd ?

База данных

Управление пользователями через SQL.....включен

Управление базами данных через SQL.....включен

Дополнительные настройки

Начало резервного копирования (UTC).....22:00

Техническое обслуживание (UTC).....Произвольное время

Прделаем те же самые шаги по созданию базы данных, основной и отчетных таблиц и их наполнению (записывать файлы с отчетами в бакет повторно не будем). Все это можно проделать как с помощью окна ввода SQL запросов в Managed Service for ClickHouse

The screenshot shows the ClickHouse Managed Service console. On the left is a sidebar with navigation options like Overview, Schemas, Groups, Hosts, Dictionaries, Machine Learning, Schemas, DataLens, Operations, Backup, Monitoring, Logs, Topology, Documentation, and a link to start work with ClickHouse. The main panel is titled 'SQL' and shows a query: `SELECT count(*) FROM CitiBike.daily_count_ride`. Below the query, there are buttons: 'Выполнить' (Execute), 'Остановить' (Stop), 'Очистить' (Clear), and a green 'COMPLETED' status. To the right of the query, there is a table with two columns: '#', 'date_ride', and 'count_ride'. The table contains 20 rows of data, showing dates from 2021-02-01 to 2021-02-21 and corresponding counts.

#	date_ride	count_ride
0	"2021-02-01"	249
1	"2021-02-03"	6488
2	"2021-02-04"	18792
3	"2021-02-05"	24378
4	"2021-02-06"	31236
5	"2021-02-07"	8308
6	"2021-02-08"	18534
7	"2021-02-09"	22482
8	"2021-02-10"	28232
9	"2021-02-11"	22979
10	"2021-02-12"	26009
11	"2021-02-13"	23084
12	"2021-02-14"	23441
13	"2021-02-15"	23404
14	"2021-02-16"	31048
15	"2021-02-17"	30902
16	"2021-02-18"	7630
17	"2021-02-19"	12061
18	"2021-02-20"	24992
19	"2021-02-21"	26475

Так и подключившись из графических IDE – в моем случае это DBeaver

The screenshot shows the DBeaver 21.2.5 interface. The left sidebar shows the 'CitiBike' database structure with tables like 'daily_average_duration_ride', 'daily_count_ride', and 'daily_diff_members'. The main panel shows the 'citibike_all' table structure. The table has 13 columns: 'ride_id' (String), 'rideable_type' (String), 'started_at' (DateTime), 'ended_at' (DateTime), 'start_station_name' (String), 'start_station_id' (String), 'end_station_name' (String), 'end_station_id' (String), 'start_lat' (String), 'start_lng' (String), 'end_lat' (String), 'end_lng' (String), and 'member_casual' (String). The table is named 'citibike_all' and is of type 'TABLE' in the 'CitiBike' schema. The bottom status bar shows '13 элементов' (13 elements) and 'MSK ru'.

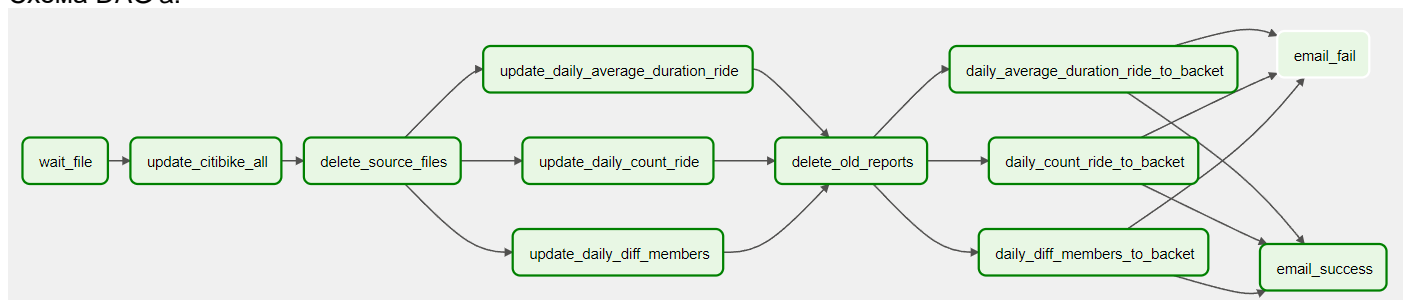
Колонки	Название	#	Тип Данных	Длина	Масштаб	Not Null	Авто генерация	Auto увеличение
ABC	ride_id	1	String			[v]	[]	[]
ABC	rideable_type	2	String			[v]	[]	[]
ABC	started_at	3	DateTime	19		[v]	[]	[]
ABC	ended_at	4	DateTime	19		[v]	[]	[]
ABC	start_station_name	5	String			[v]	[]	[]
ABC	start_station_id	6	String			[v]	[]	[]
ABC	end_station_name	7	String			[v]	[]	[]
ABC	end_station_id	8	String			[v]	[]	[]
ABC	start_lat	9	String			[v]	[]	[]
ABC	start_lng	10	String			[v]	[]	[]
ABC	end_lat	11	String			[v]	[]	[]
ABC	end_lng	12	String			[v]	[]	[]
ABC	member_casual	13	String			[v]	[]	[]

3. Теперь нужно настроить регулярную обработку попавших в хранилище файлов, т.е. повторить все, что было сделано применительно к новым данным.

Наш пайплайн будет состоять из следующих шагов:

1. Проверяем наличие новых файлов (wait_file)
2. Импортируем данные из нового файла в основную таблицу БД (update_citibike_all)
3. Во избежание повторной обработки файлов-источников, удалим их сразу после загрузки в основную таблицу (delete_source_files)
4. Считаем новые данные и добавляем в таблицы отчетов (update_daily_...)
5. Удаляем старые файлы отчетов из хранилища (delete_old_reports)
6. Перезаписываем отчеты в хранилище под теми же именами – это сделано для того, чтобы в дальнейшем связанный с файлами отчетов дашборд обновлялся автоматически, не требуя каждый раз заменить имя ежедневно обновляемого файла (daily_..._to_bucket)
7. Отправляем оповещение на email (email_success и email_fail)

Схема DAG'a:



3.0. а) Для начала создадим подключения Airflow к бакетам ObjectStorage (storage-citibike-daily-report, storage-citibike-input) и БД (clickhouse_CitiBike)

Airflow interface showing connections:

Conn Id	Conn Type	Description	Host	Port	Is Encrypted	Is Extra Encrypted
clickhouse_CitiBike	ts		localhost	8123	False	False
storage-citibike-daily-report	s3		https://storage.yandexcloud.net/citibike-daily-report		False	False
storage-citibike-input	s3		https://storage.yandexcloud.net/citibike-input		False	False

б) Исходя из того, что по условиям задания выгрузка ежедневная и отчеты ежедневные, будем считать, что файлы в своем названии вместо год-месяц содержат год-месяц-день, т.е. вида <https://storage.yandexcloud.net/citibike-input/20220125-citibike-tripdata.csv> - данные за 25 января 2022 г.

с) Работа с датами: т.к. загрузка данных за определенный период осуществляется в следующем периоде (фактически: в этом месяце за прошлый, по условию задания: сегодня за вчерашний день), импорт данных мы будем осуществлять, если найдем файл с датой за вчера. Предположим, что данные за 24.01.2022 будут загружены в произвольное время 25.01.2022. DAG выполняется для заданного интервала как только тайм-слот этого интервала будет пройден, т.е. при schedule_interval='@daily' DAG для интервала 25.01.2022 будет выполнен в 00:00 26.01.2022. Значение переменной data_interval_start в таком случае будет 2022-01-25.

Для подстановки в имена файлов будем использовать {{ yesterday_ds_nodash }} – дата за день до даты исполнения в формате YYYYMMDD, как раз такой, как и нужен (данная переменная в Airflow 2.3.2 считается устаревшей, но сохранена для обратной совместимости)

Для фильтров в SQL запросах воспользуемся конструкцией
 {{ (data_interval_start – timedelta(days=1)) }}::Timestamp::Date

д) Для работы с ClickHouse используем плагин Airflow ClickHouse Plugin (<https://pydigger.com/pypi/airflow-clickhouse-plugin>), который содержит `ClickHouseOperator`, `ClickHouseHook` и `ClickHouseSqlSensor`

е) Импортируем нужные библиотеки

```
import airflow.utils.dates
from datetime import datetime, timedelta
from textwrap import dedent

from airflow import DAG

from airflow_clickhouse_plugin.operators.clickhouse_operator import ClickHouseOperator
from airflow_clickhouse_plugin.hooks.clickhouse_hook import ClickHouseHook
from airflow_clickhouse_plugin.sensors.clickhouse_sql_sensor import ClickHouseSqlSensor

from airflow.utils.trigger_rule import TriggerRule
from airflow.operators.email import EmailOperator
```

ф) Создаем экземпляр объекта DAG

```
default_args = {
    'owner': 'airflow',
    'start_date': datetime(2022, 5, 31),
    'email': ['example@mail.ru'],
    'retries': 3,
    'retry_delay': timedelta(minutes=5),
}

dag = DAG(
    dag_id='clickhouse ETL',
    default_args=default_args,
    description='dag_for_diplom',
    schedule_interval='@daily',
    tags=['***'],
)
```

3.1. Проверяем наличие новых файлов

Наиболее подходящим вариантом для реализации этого шага был бы `airflow.sensors.s3_key_sensor`, однако к версии Airflow 2.3.2 он переехал в `airflow.providers.amazon.aws.sensors.s3.S3KeySensor`, который поддерживает только URL-адрес в стиле `s3://`, поэтому ничего не получилось (Invalid bucket name "https://storage.yandexcloud.net/citibike-input": Bucket name must be an ARN matching the regex `^arn:(aws).*:(s3|s3-object-lambda):[a-z\\-0-9]*:[0-9]{12}:accesspoint[/:][a-zA-Z0-9\\-\\.]{1,63}$|^arn:(aws).*:s3-outposts:[a-z\\-0-9]+:[0-9]{12}:outpost[/:][a-zA-Z0-9\\-\\.]{1,63}/[:]accesspoint[/:][a-zA-Z0-9\\-\\.]{1,63}$`).

Других более-менее подходящих сенсоров не наблюдается, поэтому воспользуемся `ClickHouseSqlSensor`: пропишем в нем подсчет количества поездок в файле с искомой датой, условием успеха поставим количество поездок `>0` (если файла нет, то и считать будет нечего, выполнение нашего оператора завершится неуспехом и дальнейшая цепочка не запустится).

```
# сенсор, который будет проверять наличие нового файла
wait_file = ClickHouseSqlSensor(
    task_id='wait_file',
    database='CitiBike',
    sql="SELECT count(ride_id) \
        FROM s3('https://storage.yandexcloud.net/citibike-input/{{ yesterday_ds_nodash }}-citibike-tripdata.csv',\
    clickhouse_conn_id='storage-citibike-input',
    success=lambda cnt: cnt > 0,
    dag=dag
)
```

3.2. Импортируем данные из нового файла в основную таблицу БД (`update_citibike_all`) воспользовавшись `ClickHouseOperator`.

```
# обновляем основную таблицу
update_citibike_all=ClickHouseOperator(
    task_id='update_citibike_all',
    database='CitiBike',
    sql=
        ...
        INSERT INTO CitiBike.citibike_all (
            ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id,
            end_station_name, end_station_id, start_lat, start_lng, end_lat, end_lng,
            member_casual)
        SELECT ride_id, rideable_type, started_at, ended_at, start_station_name,
            start_station_id, end_station_name, end_station_id, start_lat, start_lng, end_lat,
            end_lng, member_casual
        FROM s3('https://storage.yandexcloud.net/citibike-input/{{ yesterday_ds_nodash }}\-citibike-tripdata.csv',
            'CSVWithNames',
            'ride_id String, rideable_type String, started_at DateTime, ended_at DateTime,
            start_station_name String, start_station_id String, end_station_name String,
            end_station_id String, start_lat String, start_lng String, end_lat String,
            end_lng String, member_casual String')
        ...
    clickhouse_conn_id='clickhouse_CitiBike',
    dag=dag,
)
```

3.3. и далее - прописываем оставшиеся шаги