DIPLOMA WORK

Maksim Kravchenko

Contents

[Requirements: 2](#_Toc100972829)

[Limitations: 2](#_Toc100972830)

[1. Solution description. 3](#_Toc100972831)

[1.1. Architecture 3](#_Toc100972832)

[1.2. Environment Setup 6](#_Toc100972833)

[1.3. Additional python packages 10](#_Toc100972834)

[1.4. Lambda function 11](#_Toc100972835)

[1.5. my\_etl.py 11](#_Toc100972836)

[1.6. Output reports 12](#_Toc100972837)

[1.7. Working example 15](#_Toc100972838)

## Requirements:

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

## Limitations:

1. To simplify the solution the user require to store files in .csv format
2. To simplify the configuration the EC2 instance was configured to accept inbound traffic from all destinations and ports. Allow Inbound: 0.0.0.0/0

# Solution description.

# Architecture

The Solution Architecture is represented on the Figure 1. Architecture



Figure 1. Architecture

The components are:

1. AWS S3

The AWS S3 component will be used to create two S3 bucket

|  |  |
| --- | --- |
| Bucket name | Description |
| netology-input | The Bucket is used to store statistics files in csv format from the <https://s3.amazonaws.com/tripdata/index.html>  The Bucket will be configured to trigger the AWS Lambda function on the event of uploading the file to the bucket |
| netology-output | The Bucket is used to store the final reports in .csv format. |

1. AWS Lambda function

The AWS lambda function will be used to trigger the Airflow DAG my\_etl\_v1. The Lambda function is triggered by S3 bucket netology-input. The event type is POST. The result is to send http request to trigger Airflow DAG via stable v1 API. The HTTP request is:

URL = http://18.117.115.131:8080/api/v1/dags/my\_etl\_v1/dagRuns

Authorization: Basic YWRtaW46YWRtaW4='

'Content-Type: application/json'

Body: {"conf": {"file":"Key"}}

Where Key – is the file name which was uploaded to S3 bucket.

*Note: The using of Lambda function will give the benefits that the data pipeline becomes almost real-time. Also it is possible to upload several files at the same time as each of them will trigger the Airflow independently.*

1. AWS EC2.

The EC2 instance will be used to run the following components of the solution:

|  |  |
| --- | --- |
| Component running on EC2 instance | Description |
| Clickhouse | Clickhouse database server 18.16.1  Clickhouse client version 18.16.1  The clickhouse will be used to store the data of the csv files. |
| Airflow | Airflow 2.2.5.  The Airflow will be used as orchestrator to run the DAG. |

1. Clickhouse Database.

The Yandex column database. Will be running on top of the EC2 instance.

The database is initialized with the:

Database: netology

Table: rides

1. Airflow

The opensource Orchestrator. Will be used to run the DAG

1. DAG

My\_etl.py is the DAG which will:

1. Be triggered by Lambda function on the event of object upload into netology-input bucket.
2. Perform the following procedures:
   1. \_\_load csv file to dataframe from S3 netology\_input Bucket\_\_
   2. \_\_get the YYYY-MM of rides in csv file\_\_
   3. \_\_load the data to the Database\_\_
   4. \_\_SQL quary Number of Rides/day\_\_
   5. \_\_Store the result into DataFrame\_\_
   6. \_\_Write DataFrame into "Number\_Riders\_per\_Day\_Date.CSV" file into S3 netology\_output Bucket\_\_
   7. \_\_SQL quary Average time of Rides/day\_\_
   8. \_\_Store the result into DataFrame\_\_
   9. \_\_Write DataFrame into "Average\_Ride\_Time\_per\_Day\_Date.CSV" file into S3 netology\_output Bucket\_\_
   10. \_\_SQL quary Rides/gender/day\_\_
   11. \_\_Store the result into DataFrame\_\_
   12. \_\_Write DataFrame into "Ride\_per\_gender\_per\_Day\_Date.CSV" file into S3 netology\_output Bucket\_\_

# Environment Setup

The following steps require to be performed to setup the environment.

**EC2**

Instance type: t2.xlarge

CPU: 4

RAM: 16Gb

Disk: 12 GB

OS: Ubuntu 20.04.3 LTS

Allow inbound: 0.0.0.0/0

**Clickhouse Installation:**

echo "deb http://repo.yandex.ru/clickhouse/deb/stable/ main/" | sudo tee /etc/apt/sources.list.d/clickhouse.list

sudo apt-get update

sudo apt-get install -y clickhouse-server clickhouse-client

//Enable connection from outside. Required only for the testing of python

sudo nano /etc/clickhouse-server/config.xml

-------------------------------------

<listen\_host>0.0.0.0</listen\_host>

-------------------------------------

sudo systemctl enable clickhouse-server.service

sudo service clickhouse-server start

sudo service clickhouse-server status

**Airflow Installation:**

sudo apt-get install pip

sudo pip install boto3

sudo SLUGIFY\_USES\_TEXT\_UNIDECODE=yes pip install -U apache-airflow

sudo pip install apache-airflow[]

sudo -H pip install testresources

sudo -H pip install six==1.10.0

sudo su

echo ‘export PATH=/usr/local/bin:$PATH’ >> /root/.bash\_profile

source /root/.bash\_profile

exit

airflow db init

ls -la /home/ubuntu/airflow/

sudo mkdir /home/ubuntu/airflow/dags

sudo chmod 777 /home/ubuntu/airflow/dags

airflow users create --role Admin --username admin --email admin --firstname admin --lastname admin --password admin

//Configure Airflow service file to be able to start service automatically during the system start-up

sudo touch /etc/systemd/system/airflow-webserver.service

sudo nano /etc/systemd/system/airflow-webserver.service

---------------------

[Unit]

Description=Airflow webserver daemon

After=network.target postgresql.service mysql.service

Wants=postgresql.service mysql.service[Service]

EnvironmentFile=/etc/environment

User=ubuntu

Group=ubuntu

Type=simple

ExecStart= /usr/local/bin/airflow webserver

Restart=on-failure

RestartSec=5s

PrivateTmp=true[Install]

WantedBy=multi-user.target

---------------------

sudo systemctl daemon-reload

sudo systemctl enable airflow-webserver.service

//Configure Airflow schedule-service file to be able to start service automatically during the system start-up

sudo touch /etc/systemd/system/airflow-schedluer.service

sudo nano /etc/systemd/system/airflow-schedluer.service

---------------------

[Unit]

Description=Airflow scheduler daemon

After=network.target postgresql.service mysql.service

Wants=postgresql.service mysql.service

[Service]

EnvironmentFile=/etc/environment

User=ubuntu

Group=ubuntu

Type=simple

ExecStart=/usr/local/bin/airflow scheduler

Restart=always

RestartSec=5s

[Install]

WantedBy=multi-user.target

----------------------

sudo systemctl daemon-reload

sudo systemctl enable airflow-schedluer.service

//Configure Airflow basic Authentication for the lambda

nano vi airflow.cfg

-----------------------------------------

auth\_backend = airflow.api.auth.backend.basic\_auth

----------------------------------------

/////////////////////////////////////

//restart

/////////////////////////////////////

sudo systemctl stop airflow-schedluer.service

sudo systemctl stop airflow-webserver.service

sudo systemctl start airflow-schedluer.service

sudo systemctl start airflow-webserver.service

**DAG setup:**

Place the script into /home/ubuntu/airflow/dags/ folder

ubuntu@ip-172-31-39-151:~/airflow/dags$ ls -la

drwxrwxrwx 3 root root 4096 Apr 15 14:03 .

drwxrwxr-x 4 ubuntu ubuntu 4096 Apr 15 16:25 ..

drwxr-xr-x 2 ubuntu ubuntu 4096 Apr 15 14:03 \_\_pycache\_\_

-rwxrwxrwx 1 root root 7437 Apr 15 14:03 my\_etl.py

//restart the schedule service to upload the DAG (or wait 5 mins)

sudo systemctl stop airflow-schedluer.service

sudo systemctl start airflow-schedluer.service

**Clickhouse setup:**

//Setup the initial Database and Table for the clickhouse where all information will be stored

//Please note this is was performed one time using Jupyter Nodebook.

client = ClickClient(host=IP, settings={'use\_numpy': True})

client.execute(f"CREATE DATABASE IF NOT EXISTS {Database}")

client.execute("CREATE TABLE IF NOT EXISTS netology.rides("

"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 Float32, start\_lng Float32, end\_lat Float32, end\_lng Float32, member\_casual String) ENGINE = MergeTree() ORDER BY started\_at")

# Additional python packages

The following additional python packages will be installed on the target EC2 instance.

|  |  |  |
| --- | --- | --- |
| **Package** | **Command** | **Description** |
| boto3 | pip install boto3 | AWS client to work with AWS services.  Services used in the solution.   * S3 * EC2 |
| clickhouse-driver | pip install clickhouse-driver[numpy] | Clickhouse driver to work with clickhouse from python. Numpy Module is required to upload and download the data to and from clickhouse database to DataFrame |
| six | pip3 install six==1.15.0 | To parse URLs. Required to be version 1.15.0 or higher |

# Lambda function



The main part of the Lambda function is taken from the Amazon blueprint “s3-get-object-python”. The add-on part was to generate the request

# my\_etl.py

The my\_etl.py consists of 2 parts:

1. Python Procedures:

|  |  |
| --- | --- |
| def read\_file(Bucket,Key): | Reading the file from the AWS S3 Bucket and store it to the DataFrame. Also the Year and Month of the Data in the csv file.  The input parameters:  - Bucket: Bucket name {String},  - Key: File to read {String}  The return:  - DataFrame,  - Year{int},  - Month{int} |
| def write\_file(DataFrame,Bucket,Key): | Write DataFrame to the AWS S3 Bucket.  The input parameters:  - DataFrame: Dataframe  - Bucket: Bucket name {String},  - Key: File to read {String}  The return: None |
| def validate\_dataframe(Dataframe): | Validate the DataFrame for the gender column. If doesn't exist add the column 'gender' = 0 <int>.  The input parameters:  - Dataframe: Dataframe  The return:  - Dataframe |
| def insert\_df\_to\_CH(DataFrame, Database, Table, IP) | Insert DataFrame to ClickHouse.  The input parameters:  - DataFrame: Dataframe  - Database: Database name {String}  - Table: Table name {String}  The return: None |
| def sql\_request(query,IP): | Query request to the clickhouse Database. Store the result in the DataFrame  The input parameters:  - query: query to the Database {String}  - IP: IP address of the Clickhouse server {String}  The return:  - output\_df: Dataframe |
| def main(\*\*context): | The main Procedure which will be called by DAG.  The input parameters:  - context: dictionary of parameters passed through the JSON by HTTP request. In the current realization only one parameter passed file name of the uploaded to S3 object.  The return: None |

1. DAG description:

# Output reports

The output reports are stored as csv files to the netology-output bucket.

|  |  |
| --- | --- |
| Report name | Description |
| [<YYYY>\_<MM>\_report\_average\_ride\_time.csv](https://s3.console.aws.amazon.com/s3/object/netology-output?region=us-east-2&prefix=2021_10_report_average_ride_time.csv) | The report contains the average ride time per day for the month |
| [<YYYY>\_<MM>\_[report\_number\_of\_rides.csv](https://s3.console.aws.amazon.com/s3/object/netology-output?region=us-east-2&prefix=2021_10_report_number_of_rides.csv)](https://s3.console.aws.amazon.com/s3/object/netology-output?region=us-east-2&prefix=2021_10_report_average_ride_time.csv) | The report contains the Total number of rides per day for the month |
| [<YYYY>\_<MM>\_[report\_rides\_gender.csv](https://s3.console.aws.amazon.com/s3/object/netology-output?region=us-east-2&prefix=2021_10_report_rides_gender.csv)](https://s3.console.aws.amazon.com/s3/object/netology-output?region=us-east-2&prefix=2021_10_report_average_ride_time.csv) | The report contains the number of rides per day per gender |

# Working example

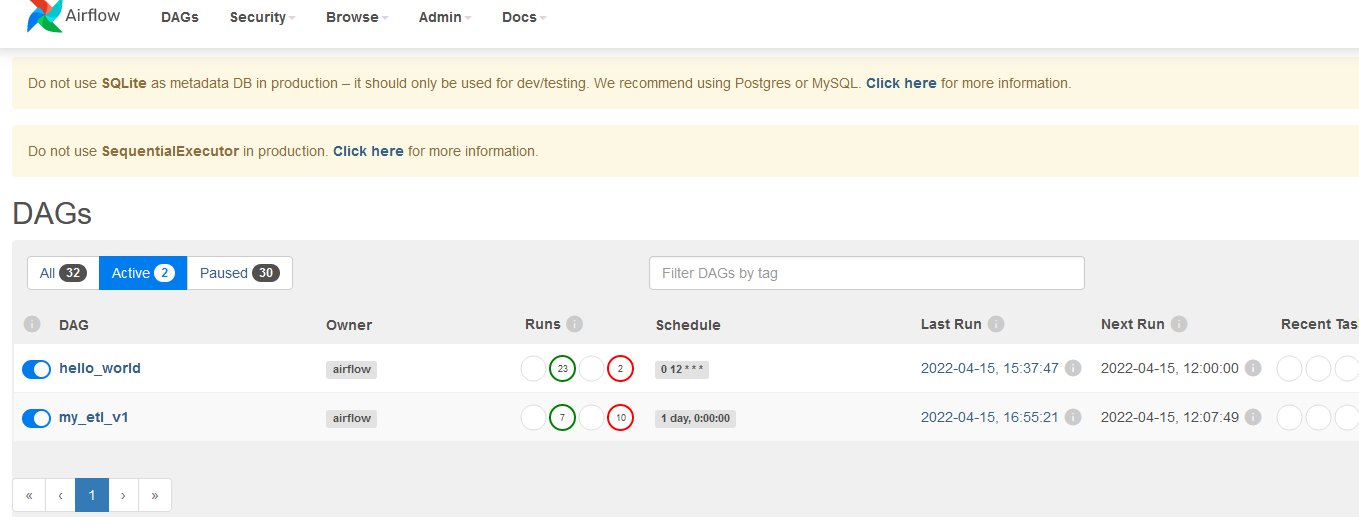
Step 1:

|  |  |
| --- | --- |
| No object on the netology-input | No object on the netology-output |

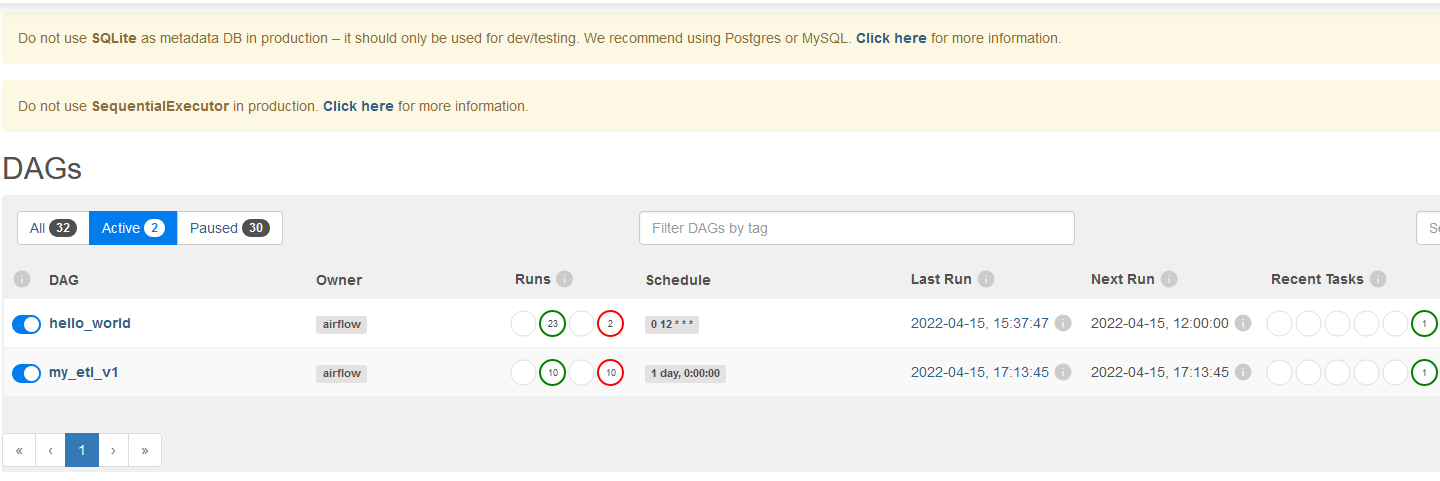
Step 2: Drop 3 files to the netology-input.

|  |  |
| --- | --- |
|  |  |

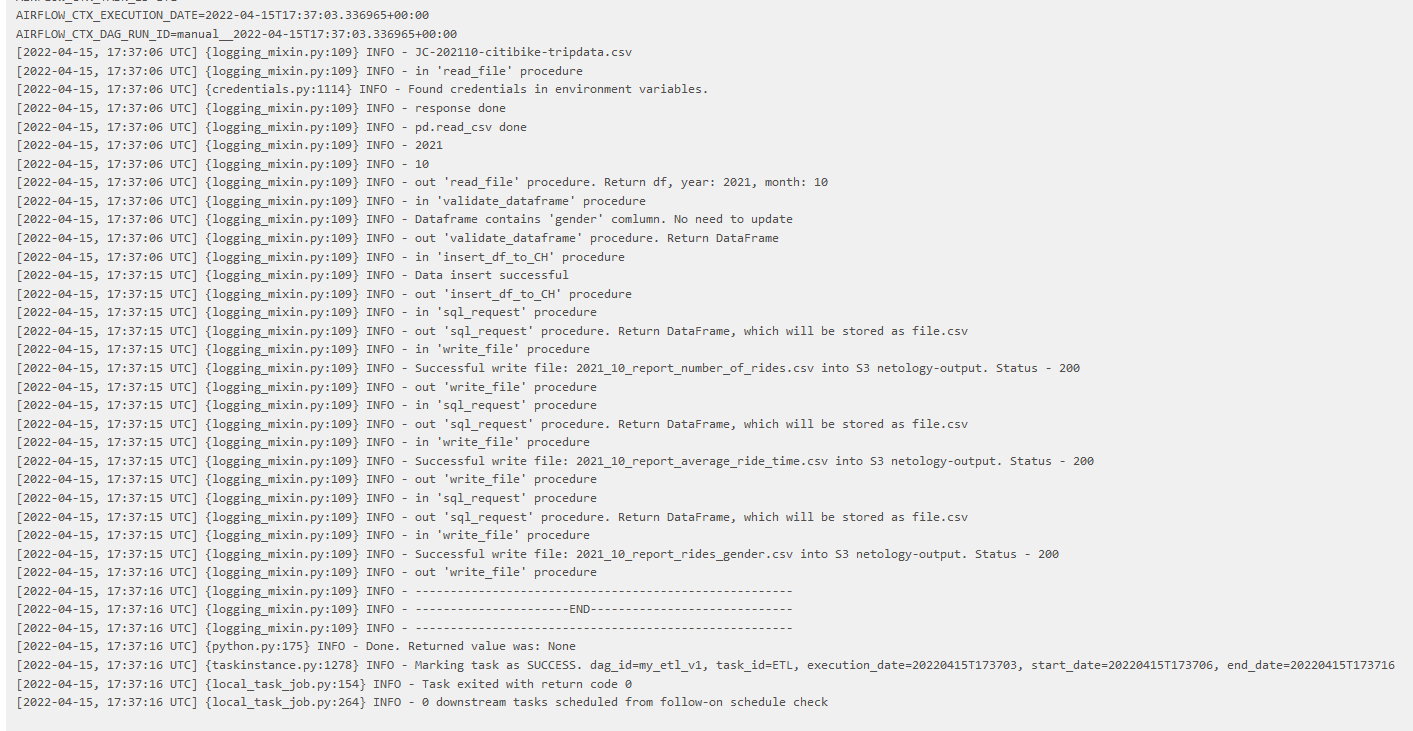
Step 3: Airflow status before



Step 4: Airflow status after



Step 5: Airflow log



Step 5: netology-output bucket

