

TABLE OF CONTENTS

**Artifacts .....2**

**Generating – Parquet data files..... 2**

**Creation scripts..... 2**

        Records table ..... 2

        Sensors table..... 2

**Queries..... 3**

        General report ..... 3

        Report for license plate YUYRB78292 ..... 3

**Reasoning .....5**

**Schema ..... 5**

**Query ..... 7**

**Cost estimation .....8**

**Alternative approaches .....10**



# Amazon Athena

## ARTIFACTS

### GENERATING – PARQUET DATA FILES

I built script which convert the data from CSV / JSON(GZIP) to Parquet (GZIP) format (I also created README.md which describes the usage), In addition attached converted files (Directory - `parquet_converted_files_20200705-203559`):

```
Python3 ./converter --src ./converter/data --dest ./ --bucket <bucket_name >
```

Will generate the following directory structure as follow (In S3 and locally):



### CREATION SCRIPTS

#### RECORDS TABLE

```
CREATE EXTERNAL TABLE IF NOT EXISTS records (
  `LicensePlate` string,
  `Sensor` int,
  `Time` string
)
STORED AS PARQUET
LOCATION 's3://idc-ex2/records/'
tblproperties ("parquet.compress"="gzip")
```

#### SENSORS TABLE

```
CREATE EXTERNAL TABLE IF NOT EXISTS sensors (
  `Sensor` int,
  `Pricing` double
)
STORED AS PARQUET
LOCATION 's3://idc-ex2/sensors/'
tblproperties ("parquet.compress"="gzip")
```

## QUERIES

### GENERAL REPORT

```
SELECT LicensePlate as "License plate",
SUM(Pricing) as "Total Cost",
DATE_FORMAT(date_parse(Time,'%Y-%m-%d %h:%i:%s'),'%m') AS "Month",
DATE_FORMAT(date_parse(Time,'%Y-%m-%d %h:%i:%s'),'%Y') AS "Year",
count(*) as "Number of tolls"
FROM records
JOIN sensors ON records.Sensor=sensors.Sensor
GROUP BY LicensePlate,
DATE_FORMAT(date_parse(Time,'%Y-%m-%d %h:%i:%s'),'%Y'),
DATE_FORMAT(date_parse(Time,'%Y-%m-%d %h:%i:%s'),'%m')
```

Output:

	License plate ▾	Total Cost ▾	Month ▾	Year ▾	Number of tolls ▾
1	IMRDW90207	8.01	03	2020	3
2	CKEOV87991	24.36	03	2020	8
3	JAVDJ35703	18.42	03	2020	7
4	IGGYB37828	10.52	03	2020	4
5	PQZHY80267	20.4	03	2020	8
6	HVKAN35109	18.369999999999997	03	2020	7
7	MZGXT62646	5.52	03	2020	3
8	JVUHE15224	27.349999999999994	03	2020	10
9	ZCPWV57130	24.099999999999998	03	2020	10
10	QJMSA29989	17.659999999999997	03	2020	6

### REPORT FOR LICENSE PLATE YUYRB78292

```
WITH report AS (
SELECT LicensePlate as "License plate",
SUM(Pricing) as "Total Cost",
DATE_FORMAT(date_parse(Time,'%Y-%m-%d %h:%i:%s'),'%m') AS "Month",
DATE_FORMAT(date_parse(Time,'%Y-%m-%d %h:%i:%s'),'%Y') AS "Year",
count(*) as "Number of tolls"
FROM records
JOIN sensors ON records.Sensor=sensors.Sensor
GROUP BY LicensePlate,
DATE_FORMAT(date_parse(Time,'%Y-%m-%d %h:%i:%s'),'%Y'),
DATE_FORMAT(date_parse(Time,'%Y-%m-%d %h:%i:%s'),'%m')
)

select *
FROM report
WHERE "License plate"='YUYRB78292';
```

## Output:

## Results



	▲ License plate ▼	Total Cost ▼	Month ▼	Year ▼	Number of tolls ▼
1	YUYRB78292	29.090000000000003	03	2020	11
2	YUYRB78292	13.76	04	2020	4

## REASONING

### SCHEMA

First I chose to split the data to 2 different directories in order to query only the data I need in specific query, Therefore I split the data to 2 directories as follow:

1. Records – Directory which contains all cars records.
2. Sensors – Sensors pricing table.

When choosing the schema I considered the following objectives:

1. Cost:
  - a. S3 storage cost.
  - b. Athena query cost.
2. Write:
  - a. S3 – execution time.
  - b. Athena – No option for writing data.
3. Read:
  - a. S3 – execution time.
  - b. Athena – execution time.

The following file formats supported by Athena:

Data Format ☒ Apache Web Logs

☐ CSV

☐ TSV

☐ Text File with Custom Delimiters

☐ JSON

☐ Parquet

☐ ORC

Regex

I chose to focus on 2 file format which considered to be optimized to BIG-data handling:

- Parquet.
- Avro.

Comparing the objectives of the 2 files formats:

1. Cost – As we can see in the following artifacts of the same data using GZIP, We will get smaller size when using Parquet/Avro (The screenshot is Parquet(GZIP)):

▶ data	Today at 15:13	4.1 MB
▶ parquet_co...705-163214	Today at 16:32	3.8 MB






Therefore we will achieve:

- a. S3 – lower cost using Parquet/Avro storing/updating the data.
- b. Athena – lower cost using Parquet/Avro due to fact that it charged on size of the data which scans (compressed or not).

- c.
2. Write – In any case for updating the data:
    - a. Updating existing file – We will need to download the file -> update it -> upload it, In those case we will achieve faster execution when the size of the data is reduced as possible due to internet speed , therefore we will achieve better performance using Avro/Parquet as I showed before the size of those formats is smaller than json.
    - b. Creating new data – In this scenario we will only need to upload the new file, So again we will achieve better performance as we reduce the file size, so we will choose Avro/Parquet.

\*\*\* Avro known for the speed of its writing, However its not our concern currently.
  3. Read – Parquet known for its reading performance compare to all other file formats, the task requires mostly reading a lot of data when the data scale up, Therefor for achieving better reading we will choose Parquet.

#### BIG DATA FORMATS COMPARISON

	Avro	Parquet
Schema Evolution Support		
Compression		
Splitability		
Most Compatible Platforms	Kafka, Druid	Impala, Arrow Drill, Spark
Row or Column	Row	Column
Read or Write	Write	Read

<https://www.datanami.com/2018/05/16/big-data-file-formats-demystified/>

In conclusion, I will choose parquet file format for performing this monthly task.

## QUERY

```
-- Choosing LicensePlate as first column
SELECT LicensePlate as "License plate",
-- Total cost calculation based on grouping by License Plate, Month, Year with same values
SUM(Pricing) as "Total Cost",
-- Month column
DATE_FORMAT(date_parse(Time,'%Y-%m-%d %h:%i:%s'), '%m') AS "Month",
-- Year column
DATE_FORMAT(date_parse(Time,'%Y-%m-%d %h:%i:%s'), '%Y') AS "Year",
-- Number of tolls based on grouping by License Plate, Month, Year with same values
count(*) as "Number of tolls"
FROM records
-- Joining sensors price to each record for allowing calculation
JOIN sensors ON records.Sensor=sensors.Sensor
-- Group rules as specified above
GROUP BY LicensePlate,
DATE_FORMAT(date_parse(Time,'%Y-%m-%d %h:%i:%s'), '%Y'),
DATE_FORMAT(date_parse(Time,'%Y-%m-%d %h:%i:%s'), '%m')
```

In line comment – explaining each expression in the Query.

## COST ESTIMATION

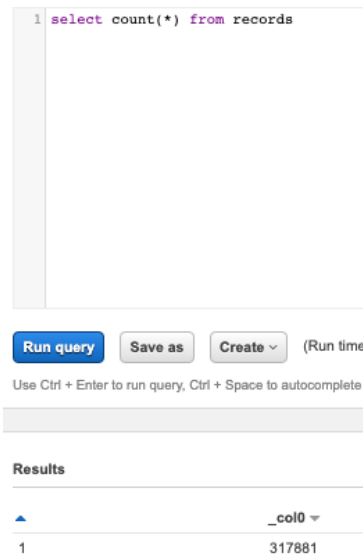
Given the following:

- Sensors = 15,000
- Records every day = 50,000
- Days in month = 30

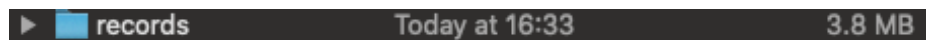
### Size estimation:

Calculating record size by given data:

Number of records: **317881**



When the data size is:  $3.8MB = 3.8 \times 2^{20} MB$

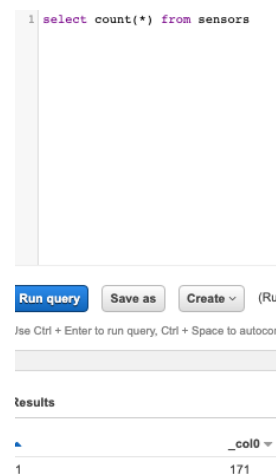


Therefore estimated size of each record is:

$$\text{Record size} = \frac{3.8 MB}{317881} = \frac{3.8 \times 2^{20}}{317881} = 12.5348 \text{ Bytes}$$

Calculating sensor size by given data:

Number of sensors: **171**



When the data size is: **3KB**





Therefore estimated size of each record is:

$$\text{Record size} = \frac{3 \text{ KB}}{171} = \frac{3 \times 2^{10}}{171} = 17.9649 \text{ Bytes}$$

**Cost calculation:**

$$\text{records per day} = 50,000 \times 15,000 = 750,000,000$$

$$\text{records per month} = 2.25 \times 10^{10}$$

Assuming we have 1 month of data, Data size which will be scanned is:

$$\begin{aligned} \text{Size scanned} &= \text{records per month} \times \text{record size} + \text{sensors amount} \times \\ \text{sensor size} &= 2.25 \times 10^{10} \times 12.5348 \text{ Bytes} + 15,000 \times 17.9649 \text{ Bytes} = \\ 2.82 \times 10^{11} \text{ Bytes} &= 262.632 \text{ GB} = 262.632 \times 2^{30} \text{ Bytes} \end{aligned}$$

### Athena

Assuming the following pricing:

**Price per query**

Region: US East (Ohio) ↕

\$5.00 per TB of data scanned

Athena charged by size data which scanned, Therefore for specific month the cost will be:

$$\text{Athena monthly query} = \frac{262.632 \times 2^{30} \text{ Bytes}}{1 \text{ TB}} \times 5\$ = \frac{262.632 \times 2^{30} \text{ Bytes}}{2^{40} \text{ Bytes}} = 1.2823\$$$

### S3

Assume the following S3-Standard pricing:

**S3 Standard** - General purpose storage for any type of data, typically used for frequently accessed data

First 50 TB / Month	\$0.023 per GB
Next 450 TB / Month	\$0.022 per GB
Over 500 TB / Month	\$0.021 per GB

$$\begin{aligned} \text{S3 price} &= \frac{262.632 \times 2^{30} \text{ Bytes}}{1 \text{ GB}} \times 0.023\$ = \frac{262.632 \times 2^{30} \text{ Bytes}}{2^{30} \text{ Bytes}} \times 0.023\$ \\ &= 6.0405\$ \end{aligned}$$

In total assuming every month we are deleting old data it will cost 7.322836\$ per month.

## ALTERNATIVE APPROACHES

Preparing cluster of instances which store the data or single instance which manage the data, For example creating RDS instance with MySQL DB, Storing and updating the data in the instance.

### Using Athena

#### Advantages

- Server less.
- Much lower price – EBS/EFS used in RDS is much more expensive than S3 storage.
- Support of wide range of data formats – out of the box.
- Fast queries – without the need to ETL (Extract transform load data).
- Good UI.
- IAM – can be managed in AWS consoles.
- Ease to create.

#### Dis-advantages

- Can't write back to S3 (INSERT SQL command).
- Not all DLLs functions supported.
- Work only on external tables.
- Size limitation in Amount of tables for example.

### Using RDS

#### Advantages

- Able to make everything in your own way and configuration:
  - Functions you created.
  - Security enforcement.
- Able to insert data to existing tables without replacing many records.
- Could be faster due to more accessible storage (Like storage in RAM).

#### Dis-advantages

- Much more expensive – EBS/EFS is much more expensive than most S3 storage classes.
- Require specific steps to create (Not out of the box solution).