Upstream Data Engineer Home Task

1. Download the [Upstream Vehicle Messages](https://drive.google.com/file/d/1UB3BcnbrQeif24Qv1v7qAWw0E_FSGlio/view?usp=share_link) tar file ([For Mac M1 (linux/arm64)](https://drive.google.com/file/d/1VYzYsjtQZW1GYgvUAfG2u9lK6uAivEgM/view?usp=share_link) )
2. Run the command **docker load -i upstream\_interview.tar** from the downloaded file path.
3. Run the command **docker images** and validate the **upstream-interview** image appears in the list
4. Start the Upstream server by running the command: **docker run -p 9900:9900 upstream-interview**
5. Server is up successfully if you see the log: “Server Started”

**API Documentation**

# GET /upstream/vehicle\_messages?amount=10000

$ curl <http://localhost:9900/upstream/vehicle_messages?amount=10>000

Returns a collection of recently broadcasted vehicle messages, according to the amount parameter passed.

[{

"vin": "1C4NJDBB0GD610265",

"manufacturer": "Jeep",

"year": 2016,

"model": "Compass",

"latitude": 41.52,

"longitude": -88.33,

"timestamp": 1673953205002,

"velocity": 120,

"frontLeftDoorState": "UNLOCKED",

"wipersState": false,

"gearPosition": "3",

"driverSeatbeltState": "LOCKED"

},

{

"vin": "SCFBB03B07GC02942",

...

]

General Notes:

* Please write a concise code, easy to understand, and validated using the software development best practices
* Please commit your code to github repository and share it with netanelm@upstream.auto

Exercise:You will build a local DataLake with three folders:

1. Bronze - raw data fetched from the above API
2. Silver - cleaned and standardized data
3. Gold - generate reports based on the silver dataset

Tasks:

**BRONZE**

1. Read 10K messages and write them as parquet files.
   1. Partitioned the data by date and hour

**SILVER**

Based on the BRONZE parquet files:

1. Identify a manufacturer with trailing space and remove it
2. Filter rows with null vin
3. Standardize gear positions to be integers only

**GOLD**

Based on the SILVER parquet files:

1. Generate a “vin\_last\_state” report that contains the following columns:
   * Vin - the vehicle id
   * Last\_reported\_timestamp - the most recent timestamp, for which we saw this vehicle in the data
   * front\_left\_door\_state - last reported, non-null value
   * Wipers\_state - last reported, non-null value
2. Generate a report, of the top 10 “fastest” vehicles, per hour:
   * Fastest vehicle will be determined by the highest velocity a vehicle has reported
   * Report will contain vin, date-hour, and the top velocity of the vehicle ordered from highest to lowest

Bonus:

* + Write a function that runs a batch SQL detection and produces a report of violating messages
  + It will run on the Bronze data set
  + The API will get as arguments:
    1. List of column names to run the detection on
    2. List SQL Injection detection Regex
  + Outputs a report with the original violating message and the violating column name
  + Example run: sqlInjectionReport([‘vin’], [‘"('(''|[^'])\*')|(;)|(\b(ALTER|CREATE|DELETE|DROP|EXEC(UTE){0,1}|INSERT( +INTO){0,1}|MERGE|SELECT|UPDATE|UNION( +ALL){0,1})\b)"’]

\* Regex string matches can be expensive to compute at scale, how can we make this detection less expensive CPU wise when running on a big batch of data?   
No need to implement a solution, just food for thought