

# Prueba Data pipeline

July 13, 2020

## 1 Objetivo

Este trabajo tiene como objetivo el desarrollar un datos utilizando los datos abiertos de la Ciudad de México correspondientes a la ubicación de las unidades del metrobús durante la última hora para obtener un histórico de la posición en la que se encuentra cada unidad que pueda ser consultado mediante un API Rest filtrando por unidad o por alcaldía.

## 2 Requerimientos y reglas de negocio

Presentar un diagrama con el diseño de su solución

Consultar periódicamente la fuente de datos

Obtener la alcaldía correspondiente a cada posición

Almacenar la información en una base de datos

Diseñar e implementar un API que permita consultar la información almacenada, con las siguientes características:

```
<ul>
  <li>Obtener una lista de unidades disponibles.</li>
  <li>Consultar el historial de ubicaciones/fechas de una unidad dado su ID</li>
  <li>Obtener una lista de alcaldías disponibles</li>
  <li>Obtener una lista de unidades que hayan estado dentro de una alcaldía</li>
</ul>
```

### 2.1 Datos de entrada

Los datos obtenidos sobre la ubicación de las unidades del metrobús se encuentran en la página de *Datos Abiertos Ciudad de México* proporcionada por le Gobierno de la ciudad de México y se encuentran en el link [https://datos.cdmx.gob.mx/explore/dataset/prueba\\_fetchdata\\_metrobus/export/](https://datos.cdmx.gob.mx/explore/dataset/prueba_fetchdata_metrobus/export/)

#### 2.1.1 Consideraciones

El sitio proporciona tres formatos de exportar los datos *csv*, *json*, *excel*; para este problema se usara el formato csv. De igual manera, son **207 unidades del metrobús**, la posición de estos se van *actualizando una vez cada hora*.

**Esquema de conjunto de datos** Datos proporcionados por el gobierno de México se describen los 13 atributos que se obtienen de cada unidad del metrobús:

vehicle\_id: tipo texto

trip\_start\_date: tipo texto

date\_updated: tipo texto

position\_longitude tipo decimal

trip\_schedule\_relationship: tipo int

position\_speed: tipo int

vehicle\_current\_status: tipo int

trip\_route\_id: tipo texto

position\_odometer: tipo int

vehicle\_label: tipo texto

trip\_id: tipo texto

position\_latitude: tipo decimal

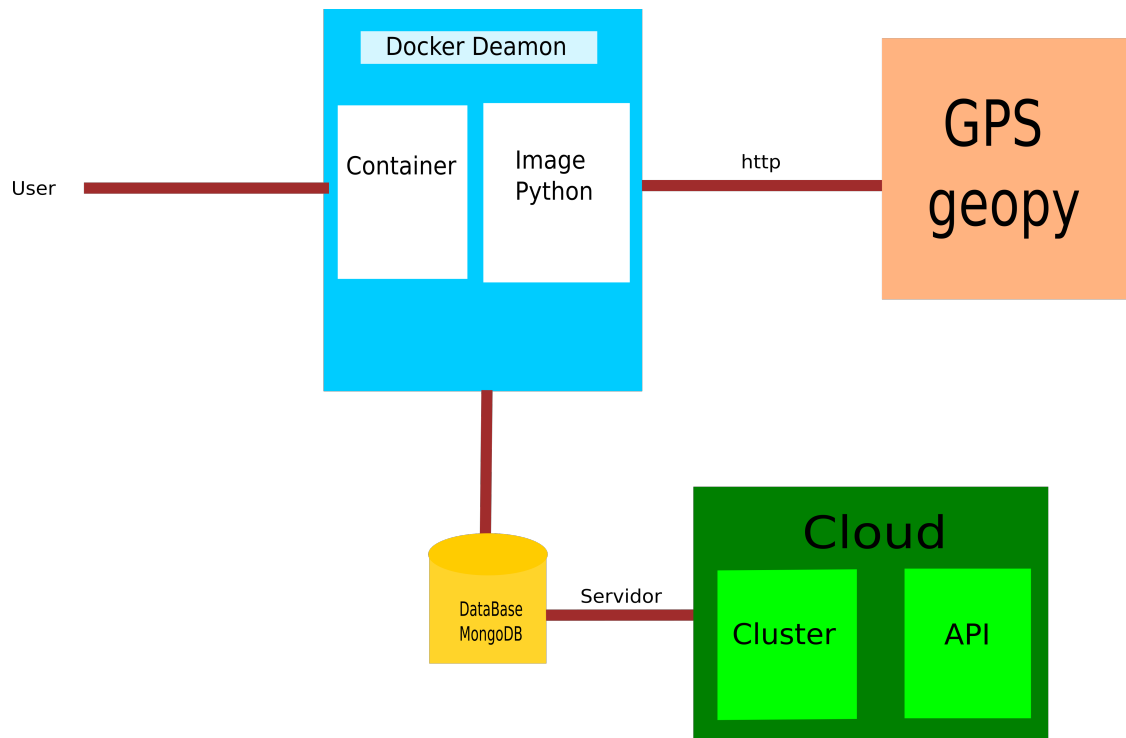
geographic\_point: tipo geo\_2d

Se debe tener en cuenta que no tienen ninguna descripción o interpretación de dichos valores, puede existir una ambigüedad de interpretación en el caso del atributo *vehicle\_current\_status* donde no se describe el significado de cada valor.

**Lenguaje de programación** Se desarrollará el trabajo en el sistema operativo ubuntu en su versión 18.04 con el lenguaje de programación *Python*.

## 2.2 Diagrama de la solución

Se describe en el siguiente diagrama la estructura que se está realizando para cumplir con los requerimientos y reglas de negocio.



## 2.3 Descarga del documento

El siguiente fragmento de código tiene como objetivo descargar el archivo del link previamente mencionado a partir del modulo wget.

```
[ ]: #Verificar la existencia del archivo
import os
namepath = '/home/mb.csv' # se descarga el archivo en la ruta namepath
if os.path.exists(namepath):
    os.remove(namepath) # se actualiza el archivo, es decir, si existe se
    ↪reemplaza por otro

# Descargar el archivo de las unidades de metrobus
import wget
url = 'https://datos.cdmx.gob.mx/explore/dataset/prueba_fetchdata_metrobus/
    ↪download/?format=csv&timezone=America/
    ↪Mexico_City&lang=es&use_labels_for_header=true&csv_separator=%2C' #link del
    ↪archivo
wget.download(url,namepath) #sentencia para descargar el archivo
```

Al concluir su descarga se lee el achivo usando l modulo de pandas para leer la columna que se llama *geographic\_point* que indica la longitud y latitud de cada unidad.

```
[ ]: # Leer el archivo
import pandas as pd
df = pd.read_csv(namepath) # se lee el archivo en u ndataframe
```

```
# indica en geographic_point la columna que tiene las coordenadas para cada
↳ instancia
coords = df["geographic_point"]
```

## 2.4 Obtener la alcaldía correspondiente a cada posición

Para identificar a que alcaldía corresponde cada posición de las unidades de metrobus se hizo uso del módulo *geopy* que está basado en la geolocalización a partir de una consulta mediante el método http, para esto se agregó una columna más al *Dataframe* creado por pandas para almacenarse con *mongodb*.

```
[ ]: # Identificar a partir de la posición a que alcaldía pertenece
from geopy.geocoders import Nominatim # se hace uso de una librería llamada geopy
import certifi
from six.moves import urllib
def uo(args, **kwargs):
    return urllib.request.urlopen(args, cafile=certifi.where(), **kwargs)
geolocator = Nominatim()
geolocator.urlopen = uo # indica por medio de una solicitud http la posición en
↳ el mapa de esas coordenadas dando una serie de datos como país, estado,
↳ ciudad, alcaldía, colonia, calle, código postal

alcaldia = []
for i in coords: # se genera una lista de las alcaldías de cada instancia
    try:
        location = geolocator.reverse(i, language='en')
        li = list(location.address.split(","))
        alcaldia.append(li[len(li)-4]) # se almacena la alcaldía
        #print(k)

    except:
        alcaldia.append("") # en caso contrario no se considera alcaldía
↳ alguna.
df['alcaldia'] = alcaldia
```

## 2.5 Almacenar la información en una base de datos

Se usa el módulo de MongoDB con la plataforma Web de Mongo Atlas usando el servicio de la nube de Amazon AWS.

## Plataforma de Atlas Mongodb

Base de datos de las unidades del metrobús almacenada en Atlas Mongodb

```
[ ]: # se almacena en Mongodb
from pymongo import MongoClient
client = MongoClient("mongodb+srv://toto:toto@clustertest.pyelj.mongodb.net/
↳toto?retryWrites=true&w=majority") # dirección de mi base de datos
db = client['toto']
collection = db['toto']
df.reset_index(inplace=True)
data_dict = df.to_dict("records")
collection.insert_many(data_dict)
```

## 2.6 Consultar periódicamente la fuente de datos

Para realizar la obtención periódica de los datos se usa un bucle infinito donde se realizará el proceso cada 3600 segundos o 1 hora, ya que la lista de las unidades se van actualizando cada hora, para tener un historial de estos.

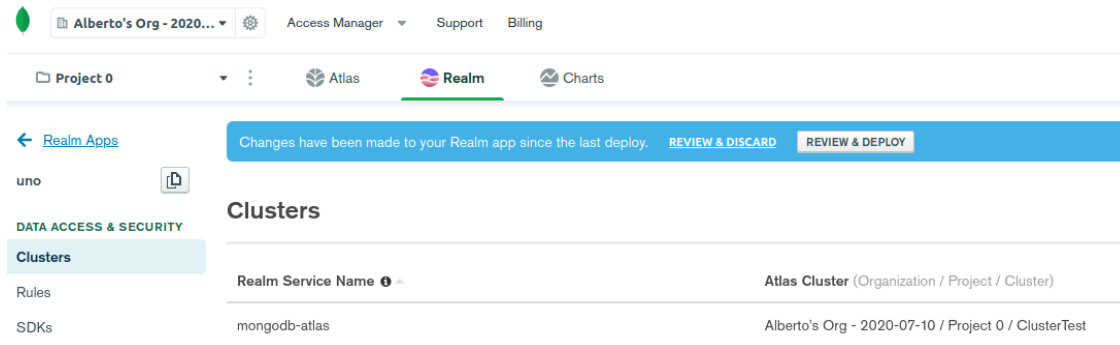
## 2.7 Diseñar e implementar un API que permita consultar la información almacenada

Gracias a la plataforma de MongoDB tiene la opción de hacer la API apoyada con GraphQL dónde se debe realizar la conexión de la base de datos con una aplicación, posteriormente se enlaza esta con el cluster y la base que se subió de las unidades del metrobús.

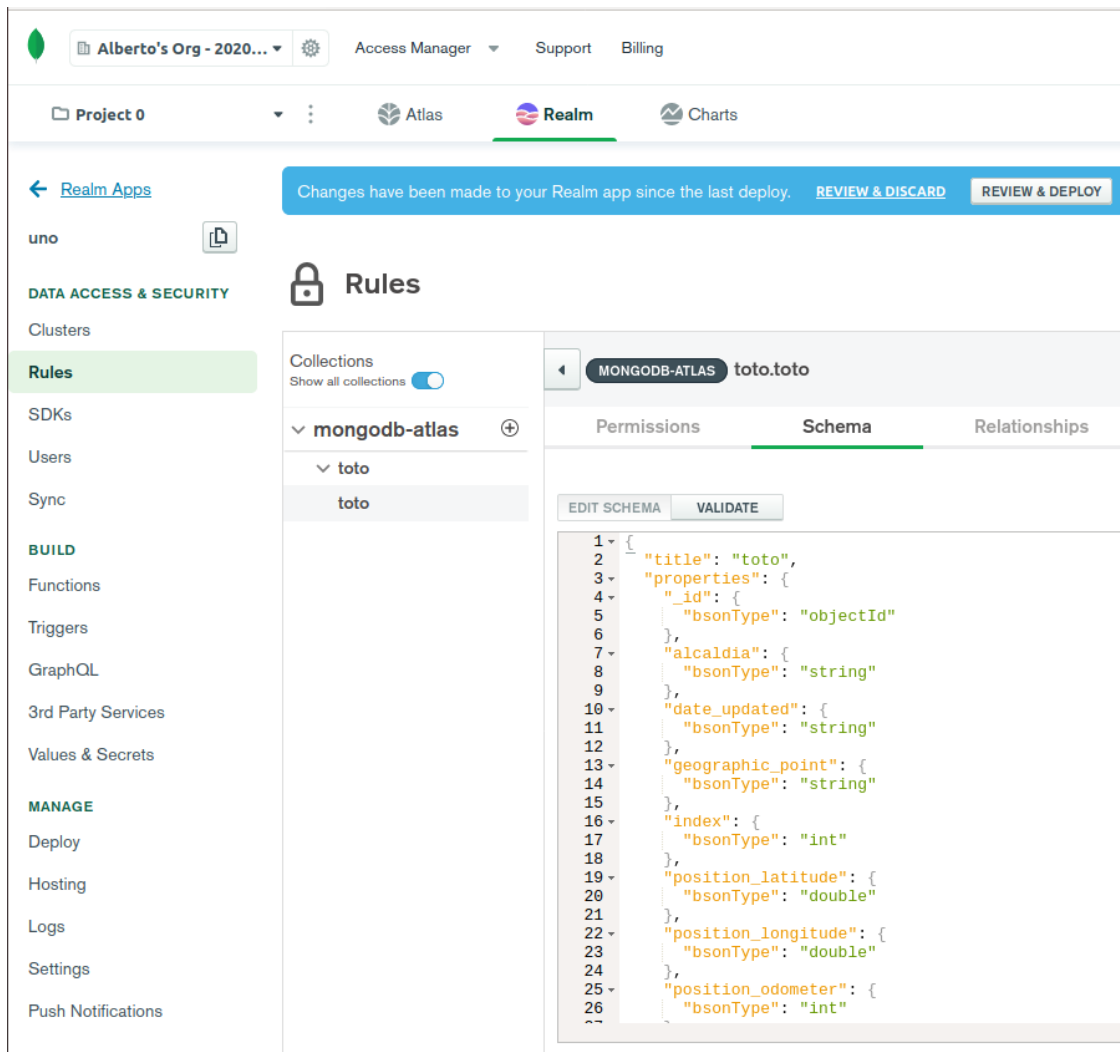
### Aplicación para la API

The screenshot shows the MongoDB Atlas GraphQL interface for a project named 'uno'. The left sidebar contains navigation links for 'DATA ACCESS & SECURITY' (Clusters, Rules, SDKs, Users, Sync), 'BUILD' (Functions, Triggers, GraphQL, 3rd Party Services, Values & Secrets), and 'MANAGE' (Deploy, Hosting, Logs, Settings, Push Notifications). The main content area displays a progress bar for 'You're almost done setting up your application...' with three steps: 'Link an Atlas Cluster' (ClusterTest Linked), 'Create a Schema' (Completed), and 'Apply Rules' (Completed). Below this, a 'Usage This Month' section shows metrics for Requests (0.00 M), Data Transfer (0.00 GB), Compute Runtime (0.00 Hours), and Sync Runtime (0.00 Hours).

### Conexión con la base de datos del cluster



## Esquema de la base de datos para la api GraphQL



### 2.7.1 Obtener una lista de unidades disponibles

Usando la opción de GraphQL de la plataforma y al no existir una descripción de los valores se considera el atributo *vehicle\_id* el de la unidad y el valor **1** como disponible, siendo esto realizado mediante la consulta:

```
[ ]: query {
  todos (query:{vehicle_current_status:1}) {
    vehicle_id
  }
}
```



**Salida:**

## 2.7.2 Consultar el historial de las ubicaciones/fechas de una unidad dado su ID

Se emplea el siguiente query considerando un id aleatorio con oes el *valor 28*, desplegando la alcaldía, fecha y posición geográfica en coordenadas:

```
[ ]: query {
  todos (query:{vehicle_id:28}) {
    vehicle_id
    alcaldia
    date_updated
    geographic_point
  }
}
```



The screenshot shows the GraphQL IDE interface. On the left, a query is defined with variables for vehicle\_id and vehicle\_current\_status. The query is executed, and the JSON response is displayed on the right. The response contains a list of 'totos' objects, each with fields for 'alcaldia', 'date\_updated', 'geographic\_point', and 'vehicle\_id'.

```

20 #
21 # Keyboard shortcuts:
22 #
23 # Prettify Query: Shift-Ctrl-P (or press the prettify button above)
24 #
25 # Merge Query: Shift-Ctrl-M (or press the merge button above)
26 #
27 # Run Query: Ctrl-Enter (or press the play button above)
28 #
29 # Auto Complete: Ctrl-Space (or just start typing)
30 #
31
32
33
34 query {
35   totos (query:{vehicle_id:28}) {
36     vehicle_id
37     alcaldia
38     date_updated
39     geographic_point
40   }
41 }
42
QUERY VARIABLES

```

```

{
  "data": {
    "totos": [
      {
        "alcaldia": " Azcapotzalco",
        "date_updated": "2020-07-10 18:00:02",
        "geographic_point": "19.496099472,-99.1545028687",
        "vehicle_id": 28
      },
      {
        "alcaldia": " Azcapotzalco",
        "date_updated": "2020-07-10 18:00:02",
        "geographic_point": "19.496099472,-99.1545028687",
        "vehicle_id": 28
      },
      {
        "alcaldia": " Azcapotzalco",
        "date_updated": "2020-07-10 21:00:07",
        "geographic_point": "19.496099472,-99.1545028687",
        "vehicle_id": 28
      },
      {
        "alcaldia": " Azcapotzalco",
        "date_updated": "2020-07-12 14:00:05",

```

Salida:

### 2.7.3 Obtener una lista de alcaldías disponibles

Considerando el valor 1 como disponible sólo se tiene que desplegar el valor de las alcaldías:

```
[ ]: query {
  totos (query:{vehicle_current_status:1}) {
    alcaldia
  }
}
```

The screenshot shows the GraphQL IDE interface. On the left, a query is defined with variables for vehicle\_id and vehicle\_current\_status. The query is executed, and the JSON response is displayed on the right. The response contains a list of 'totos' objects, each with fields for 'alcaldia', 'date\_updated', 'geographic\_point', and 'vehicle\_id'.

```

17 # Nothing is appearing in the documentation explorer, you may need
18 # to resolve some warnings or errors before using GraphQL. A sample query
19 # for the Toto type has been generated for you below.
20 #
21 # Keyboard shortcuts:
22 #
23 # Prettify Query: Shift-Ctrl-P (or press the prettify button above)
24 #
25 # Merge Query: Shift-Ctrl-M (or press the merge button above)
26 #
27 # Run Query: Ctrl-Enter (or press the play button above)
28 #
29 # Auto Complete: Ctrl-Space (or just start typing)
30 #
31
32
33
34 query {
35   totos (query:{vehicle_current_status:1}) {
36     alcaldia
37   }
38 }
39
QUERY VARIABLES

```

```

{
  "data": {
    "totos": [
      {
        "alcaldia": " Cuauhtémoc"
      },
      {
        "alcaldia": " Gustavo A. Madero"
      },
      {
        "alcaldia": " Gustavo A. Madero"
      },
      {
        "alcaldia": " Venustiano Carranza"
      },
      {
        "alcaldia": " Azcapotzalco"
      },
      {
        "alcaldia": " Tlalpan"
      },
      {
        "alcaldia": " Gustavo A. Madero"
      }
    ]
  }
}
```

Salida

### 2.7.4 Obtener una lista de unidades que hayan estado dentro de una alcaldía

Para esta consulta se obtiene los id's de cada unidad seleccionando pro ejemplo la alcaldía *Tlalpan*

```
[ ]: query {
  totos (query:{alcaldia:" Tlalpan"}) {
    vehicle_id
  }
}
```

```
}  
}
```



The screenshot shows the GraphQL IDE interface. On the left, a query is defined: `query {  
 todos(query:{alcaldia:" Tlalpan"}) {  
 vehicle_id  
 }  
}`. The right pane displays the JSON response: `{  
 "data": {  
 "todos": [  
 {  
 "vehicle_id": 531  
 },  
 {  
 "vehicle_id": 1254  
 },  
 {  
 "vehicle_id": 1257  
 },  
 {  
 "vehicle_id": 533  
 },  
 {  
 "vehicle_id": 177  
 },  
 {  
 "vehicle_id": 1254  
 },  
 {  
 "vehicle_id": 531  
 }  
 ]  
 }  
}`. The bottom status bar indicates 'QUERY VARIABLES'.

Salida

## 2.8 Docker

Se usa el siguiente **Dockerfile** para instalar las dependencias de python necesarias para funcionar el archivo *script.py* que se indico al inicio dle documento

```
FROM python:2 WORKDIR /usr/src/app MAINTAINER Alberto alberto.maldo1312@gmail.com  
RUN pip install wget RUN pip install pandas RUN pip install geopy RUN pip install certifi RUN  
pip install six RUN pip install pymongo RUN pip install dnspython COPY . . CMD [ "python",  
"/script.py" ]
```

Posteriormente, se usa el siguiente comando para construir el contenedor:

```
[ ]: docker build -t m-python .
```

Finalmente se corre el contenedor con el archivo en uso para correr en ciclos de una hora el archivo *script.pi*

```
[ ]: docker run -it --rm --name my-running-app m-python
```

## 2.9 Puntos extras

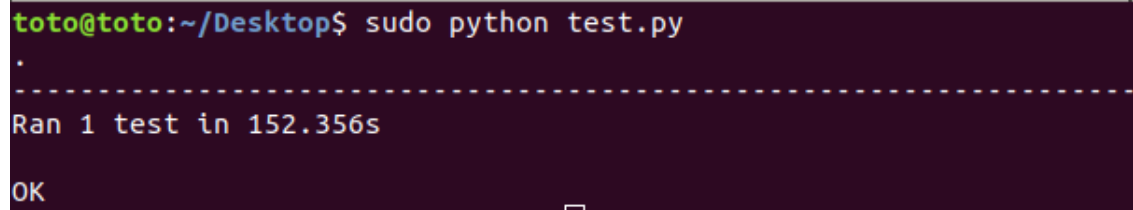
La API esta basada a partir de la paltforma de Mongodb que se conecta a GraphQL donde se puede genera la API. De igual manera para pruebas unitarias es considerano una iteración de dicho proceso.

### 2.9.1 Prueba unitaria

Usando el módldo unittest que se encarga de identificar diferentes scripts basados en PYthon para hacer pruebas unitarias.

```
[ ]: import unittest
import ex2
class TestMyModule(unittest.TestCase):
    def test_sum(self):
        self.assertEqual(ex2.prueba(), True)
if __name__ == "__main__":
    unittest.main()
```

El resultado de dicha prueba es de 152.356s y de una respuesta positiva.

A terminal window with a dark purple background. The prompt is 'toto@toto:~/Desktop\$'. The command 'sudo python test.py' has been executed. The output shows a single dot '.' on the first line, followed by a dashed line '-----' on the second line. The third line shows 'Ran 1 test in 152.356s'. The fourth line shows 'OK'.

```
toto@toto:~/Desktop$ sudo python test.py
.
-----
Ran 1 test in 152.356s
OK
```