WBS CODING SCHOOL

**Data Engineering: Pipelines on the Cloud**

2022/ 16/06 Yusuf Arcan Demirel

## {PROJECT_3 : GANS}

Gans is a startup developing an e-scooter-sharing system. It aspires to operate in the most populous cities all around the world. In each city, it will have hundreds of e-scooters parked in the streets and allows users to rent them by the minute.

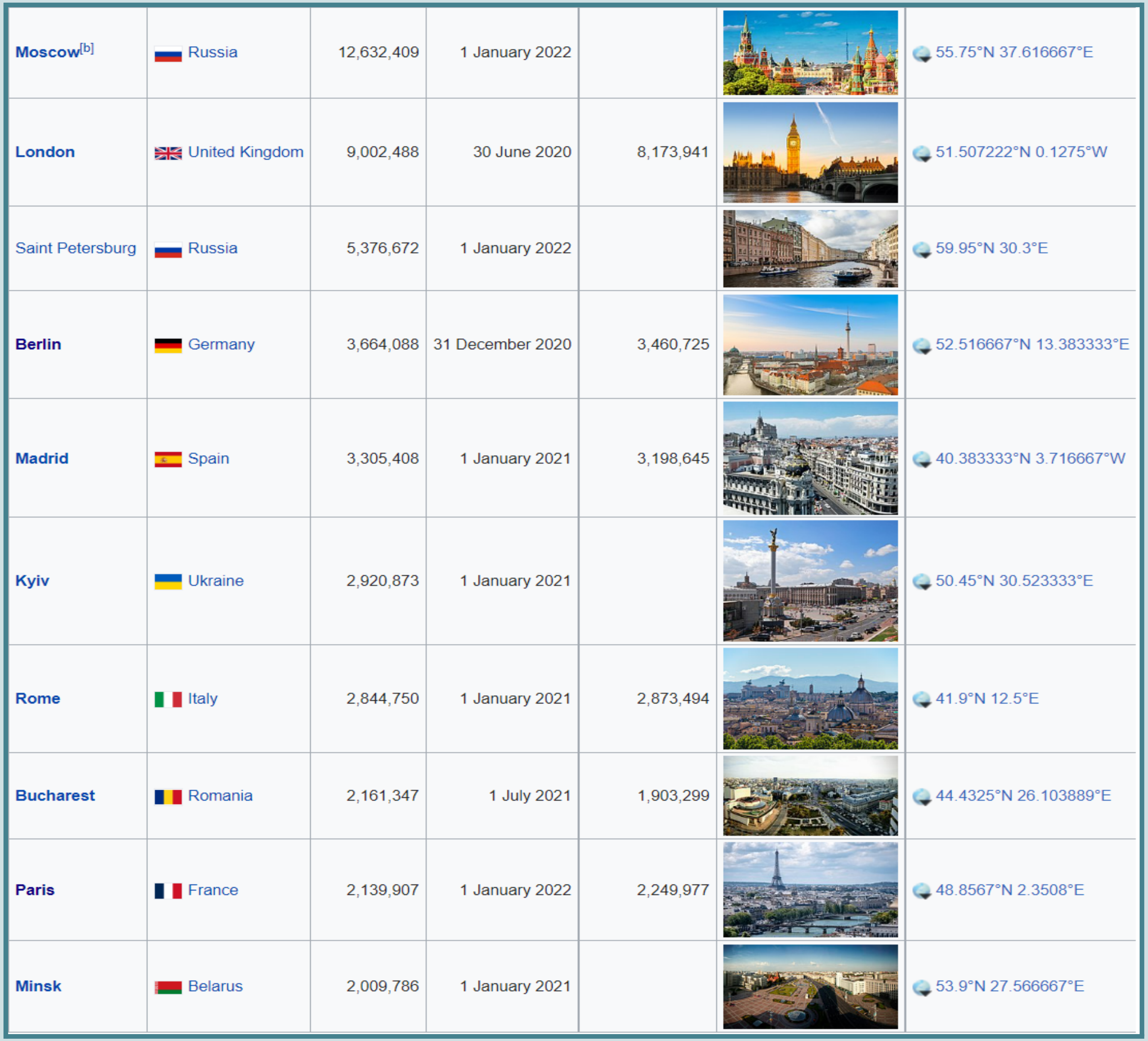
The task will be to collect data from external sources that can potentially help Gans predict e-scooter movement. Since data is needed every day, in real time and accessible by everyone in the company.

## [‘METHODS’, ‘TOOLS’, ‘RESOURCES’]

- ➔ Web Scraping
  - ◆ Python Beautifulsoup
    - Wikipedia
- ➔ Data with APIs
  - ◆ Python Requests
    - Open Weather and Rapid API
- ➔ Data Storage
  - ◆ MySQL Local and Python SQLAlchemy
- ➔ Cloud Pipeline
  - ◆ MySQL Cloud and AWS Lambda Functions

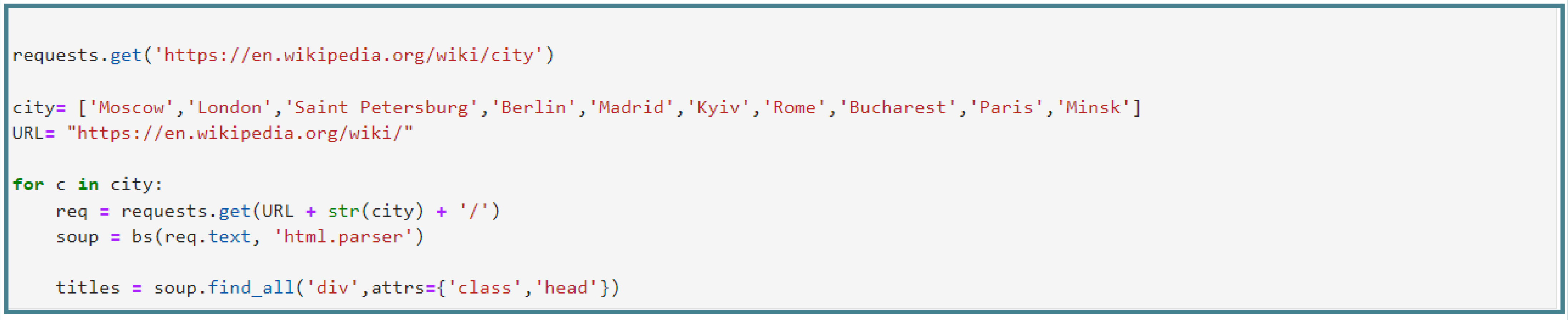
## *#WEB SCRAPING#*

➢ The first requests were started with collecting demographic data for the largest European cities from Wikipedia.



| | | | | | | |
|---|---|---|---|---|---|---|
| Moscow[b] | Russia | 12,632,409 | 1 January 2022 | | | 55.75°N 37.616667°E |
| London | United Kingdom | 9,002,488 | 30 June 2020 | 8,173,941 | | 51.507222°N 0.1275°W |
| Saint Petersburg | Russia | 5,376,672 | 1 January 2022 | | | 59.95°N 30.3°E |
| Berlin | Germany | 3,664,088 | 31 December 2020 | 3,460,725 | | 52.516667°N 13.383333°E |
| Madrid | Spain | 3,305,408 | 1 January 2021 | 3,198,645 | | 40.383333°N 3.716667°W |
| Kyiv | Ukraine | 2,920,873 | 1 January 2021 | | | 50.45°N 30.523333°E |
| Rome | Italy | 2,844,750 | 1 January 2021 | 2,873,494 | | 41.9°N 12.5°E |
| Bucharest | Romania | 2,161,347 | 1 July 2021 | 1,903,299 | | 44.4325°N 26.103889°E |
| Paris | France | 2,139,907 | 1 January 2022 | 2,249,977 | | 48.8567°N 2.3508°E |
| Minsk | Belarus | 2,009,786 | 1 January 2021 | | | 53.9°N 27.566667°E |

- ➢ Demographic data contents were pulled out with Python Beautifulsoup.



```
requests.get('https://en.wikipedia.org/wiki/city')

city= ['Moscow','London','Saint Petersburg','Berlin','Madrid','Kyiv','Rome','Bucharest','Paris','Minsk']
URL= "https://en.wikipedia.org/wiki/"

for c in city:
    req = requests.get(URL + str(city) + '/')
    soup = bs(req.text, 'html.parser')

    titles = soup.find_all('div',attrs={'class','head'})
```

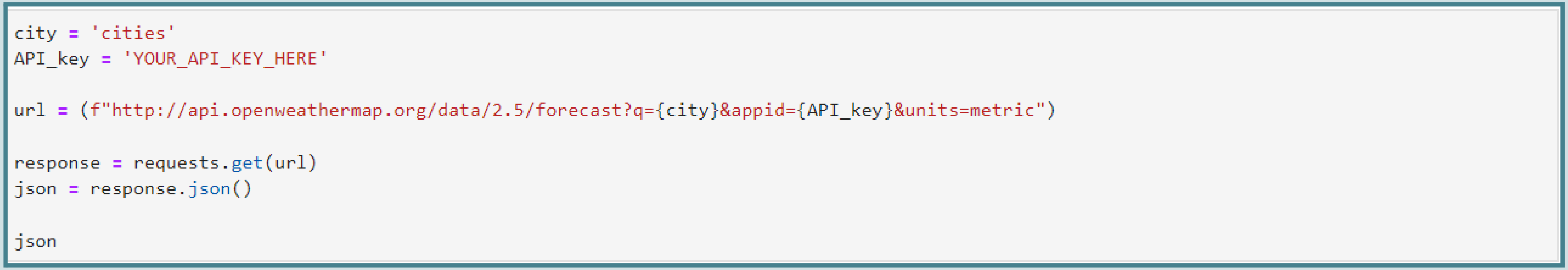
- ➢ After web scraping, all data were collected under the desired titles (latitude, longitude etc.), cleaned and saved as a ‘csv’ file.

## *#DATA with APIs#*

- ➢ Two different API sources were used for weather and flight information.

### **Weather- OpenWeatherMap**

- ➢ Registered OpenWeatherMap and then accessed the free API Key.
- ➢ 3-hour and 5-day forecast APIs were used as weather conditions.



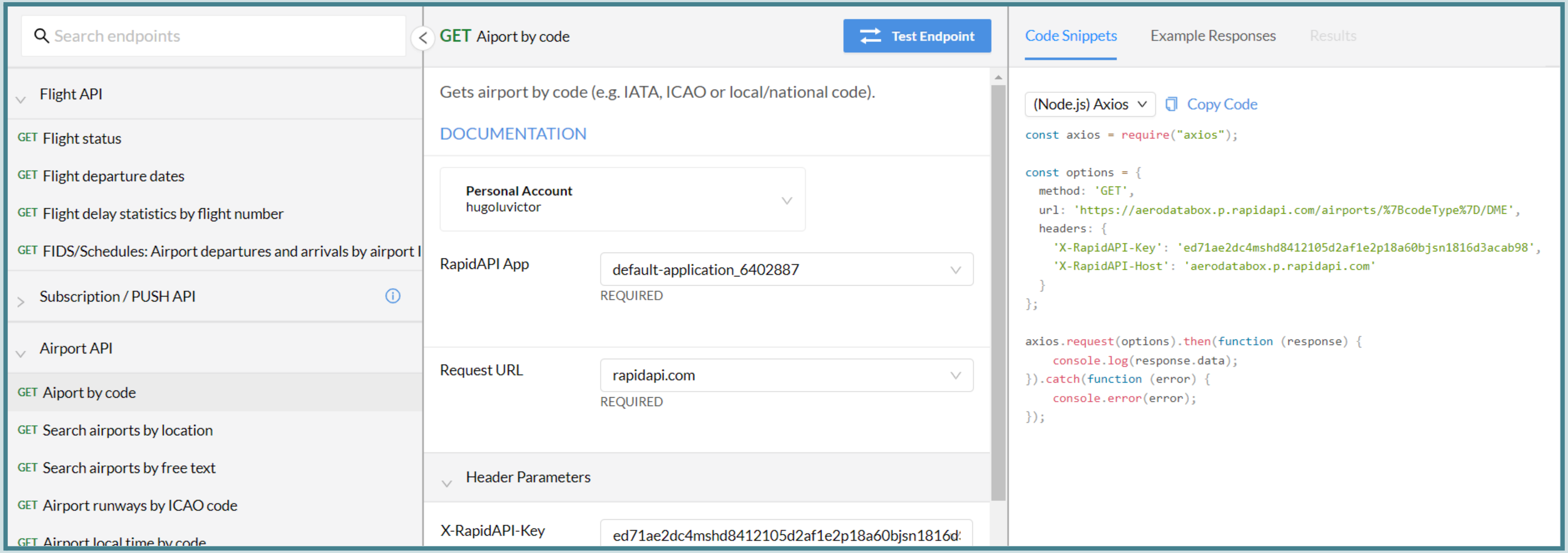
```
city = 'cities'
API_key = 'YOUR_API_KEY_HERE'

url = (f"http://api.openweathermap.org/data/2.5/forecast?q={city}&appid={API_key}&units=metric")

response = requests.get(url)
json = response.json()

json
```

- ➢ Responses were received in 5 days and 8 different times of the day, that's why the json list was iterated each time, so only the data of emphasis were selected.

## Flights- AeroDataBox

- ➢ AeroDataBox data is only accessible via RapidAPI and only 200 queries per month are free.

- ➢ All airports were listed with ICAO codes and saved as csv files.


Search endpoints
Flight API
GET Flight status
GET Flight departure dates
GET Flight delay statistics by flight number
GET FIDS/Schedules: Airport departures and arrivals by airport I
Subscription / PUSH API
Airport API
GET Aiport by code
GET Search airports by location
GET Search airports by free text
GET Airport runways by ICAO code
GET Airport local time by code
Test Endpoint
Code Snippets
Example Responses
Results
(Node.js) Axios
Copy Code
Header Parameters

GET Aiport by code

Gets airport by code (e.g. IATA, ICAO or local/national code).

DOCUMENTATION

Personal Account
hugoluvictor

RapidAPI App: default-application_6402887
REQUIRED

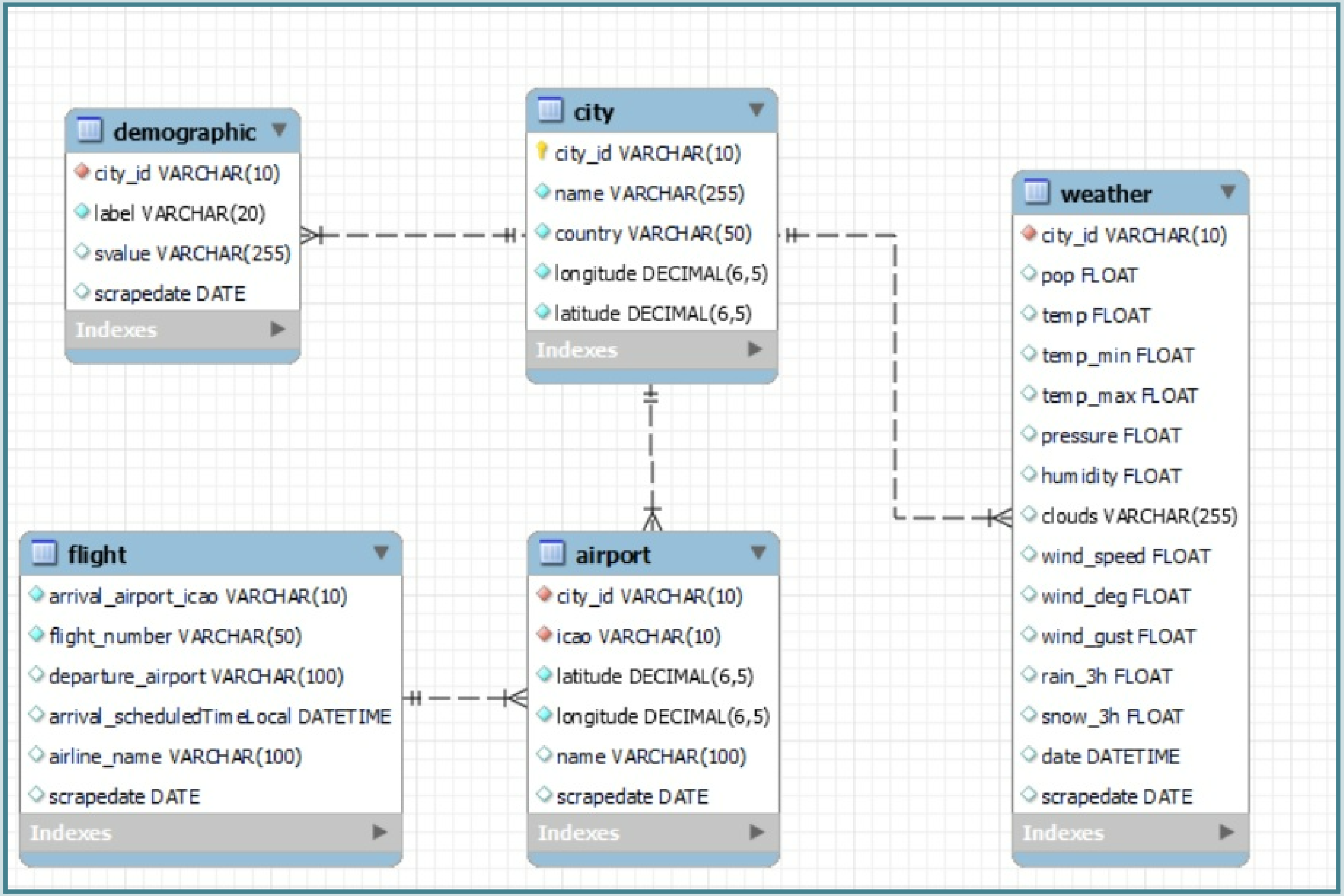
Request URL: rapidapi.com
REQUIRED

X-RapidAPI-Key: ed71ae2dc4mshd8412105d2af1e2p18a60bjsn1816d:

```
const axios = require("axios");

const options = {
  method: 'GET',
  url: 'https://aerodatabox.p.rapidapi.com/airports/%7BcodeType%7D/DME',
  headers: {
    'X-RapidAPI-Key': 'ed71ae2dc4mshd8412105d2af1e2p18a60bjsn1816d3acab98',
    'X-RapidAPI-Host': 'aerodatabox.p.rapidapi.com'
  }
};

axios.request(options).then(function (response) {
    console.log(response.data);
}).catch(function (error) {
    console.error(error);
});
```

## *#DATA STORAGE#*

- A local database was created in MySQL Workbench to transfer all the obtained data.



**demographic**
city_id VARCHAR(10)
label VARCHAR(20)
svalue VARCHAR(255)
scrapedate DATE
Indexes

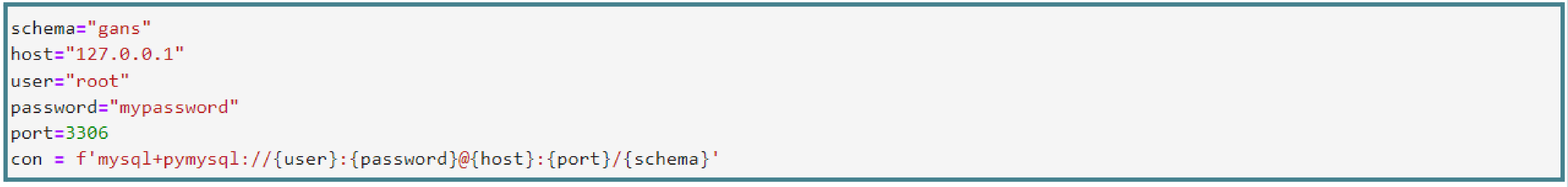
**city**
city_id VARCHAR(10)
name VARCHAR(255)
country VARCHAR(50)
longitude DECIMAL(6,5)
latitude DECIMAL(6,5)
Indexes

**weather**
city_id VARCHAR(10)
pop FLOAT
temp FLOAT
temp_min FLOAT
temp_max FLOAT
pressure FLOAT
humidity FLOAT
clouds VARCHAR(255)
wind_speed FLOAT
wind_deg FLOAT
wind_gust FLOAT
rain_3h FLOAT
snow_3h FLOAT
date DATETIME
scrapedate DATE
Indexes

**flight**
arrival_airport_icao VARCHAR(10)
flight_number VARCHAR(50)
departure_airport VARCHAR(100)
arrival_scheduledTimeLocal DATETIME
airline_name VARCHAR(100)
scrapedate DATE
Indexes

**airport**
city_id VARCHAR(10)
icao VARCHAR(10)
latitude DECIMAL(6,5)
longitude DECIMAL(6,5)
name VARCHAR(100)
scrapedate DATE
Indexes

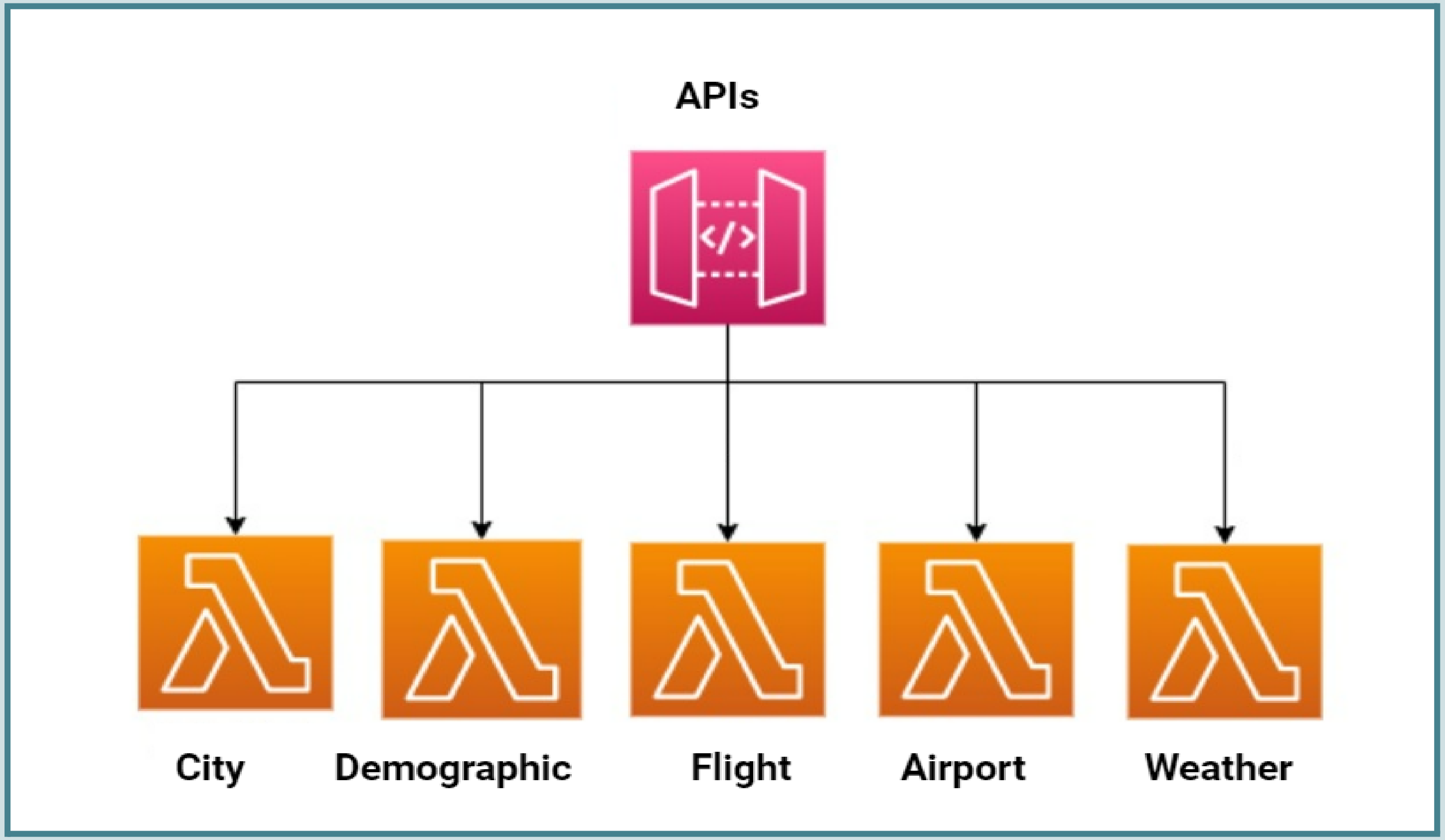
- Through SQLAlchemy, all query results were transferred to the MySQL local Database that we created before.



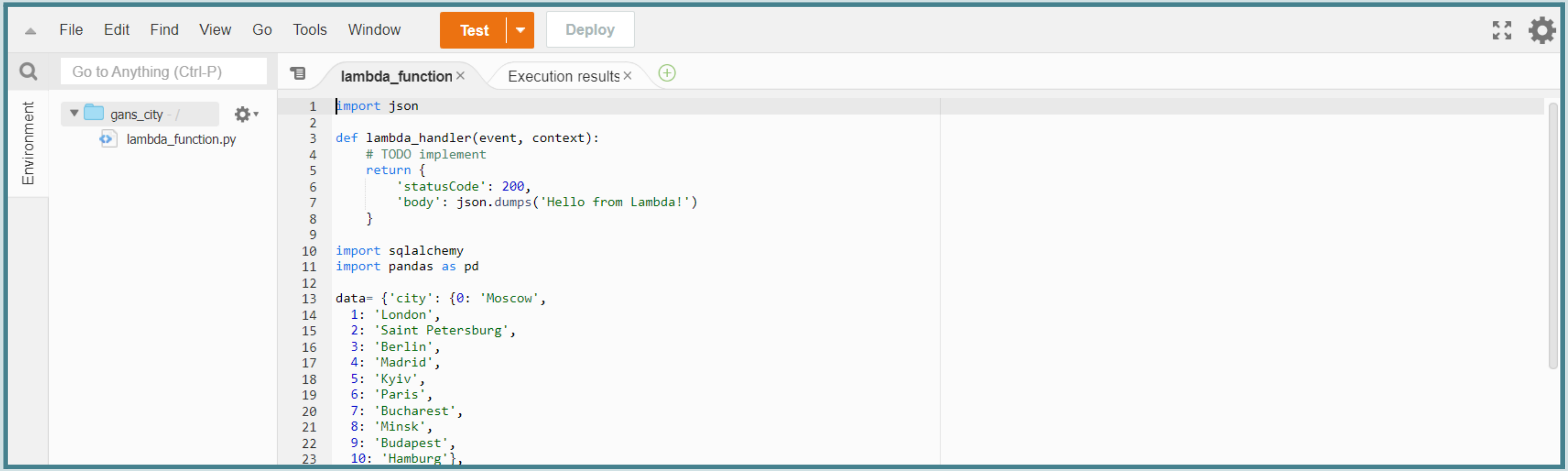
```
schema="gans"
host="127.0.0.1"
user="root"
password="mypassword"
port=3306
con = f'mysql+pymysql://{user}:{password}@{host}:{port}/{schema}'
```

### *#CLOUD PIPELINE#*

- Amazon AWS account was opened as a Cloud Database and connection with the local database was established.



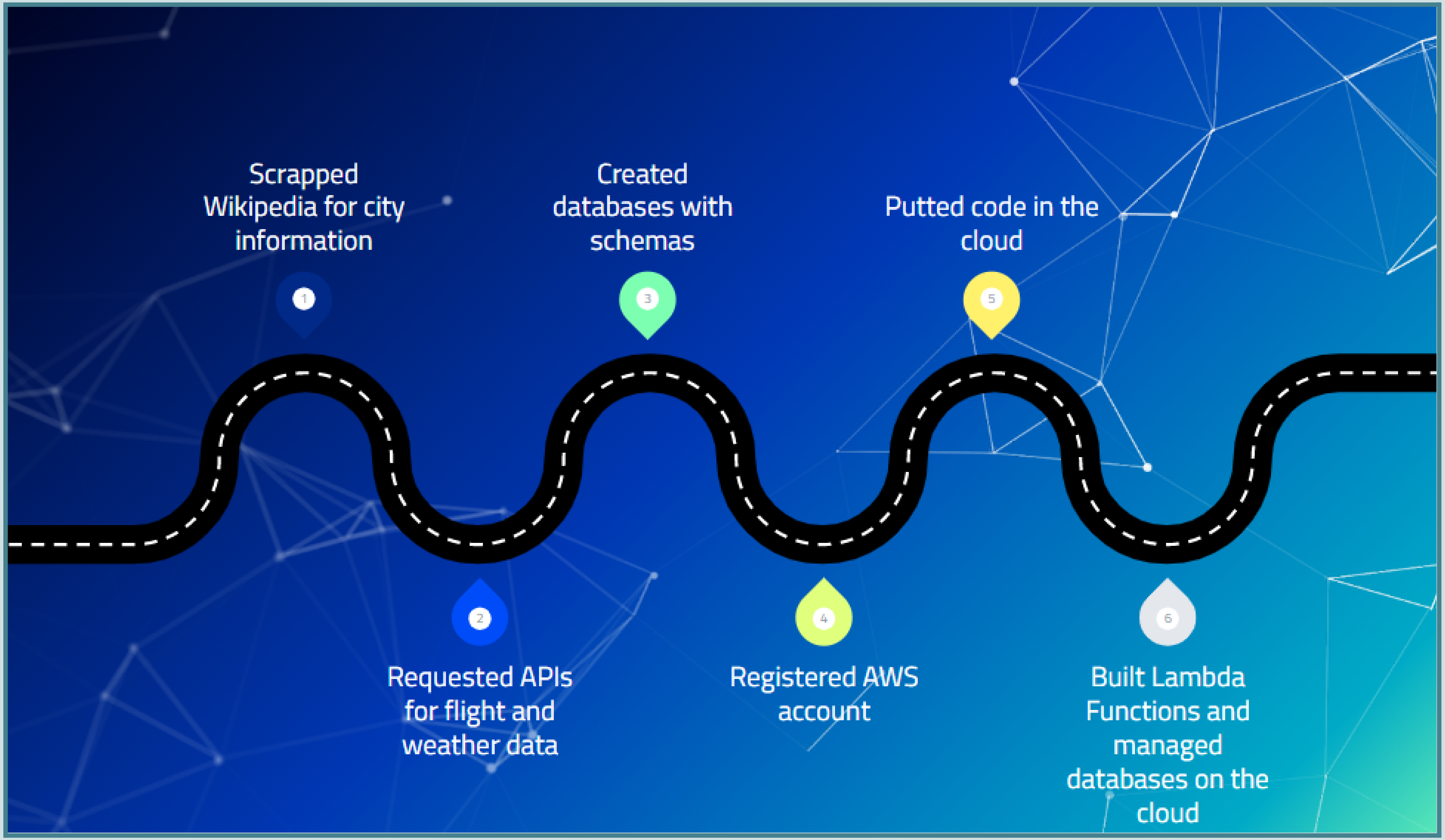
➢ The database, which was migrated to AWS, started to be managed completely over the cloud through Lambda functions.



```python
import json

def lambda_handler(event, context):
    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }

import sqlalchemy
import pandas as pd

data= {'city': {0: 'Moscow',
  1: 'London',
  2: 'Saint Petersburg',
  3: 'Berlin',
  4: 'Madrid',
  5: 'Kyiv',
  6: 'Paris',
  7: 'Bucharest',
  8: 'Minsk',
  9: 'Budapest',
  10: 'Hamburg'},
```

➢ Thanks to the AWS Event Bridge service, Lambda functions can be run in the desired time period with an automatic schedule by adding triggers. However, trigger is not used in this project due to API query limits.

#PROJECT RESULTS#



1. Scrapped Wikipedia for city information

2. Requested APIs for flight and weather data

3. Created databases with schemas

4. Registered AWS account

5. Putted code in the cloud

6. Built Lambda Functions and managed databases on the cloud