

Automation on Airports using API and AWS

In this twenty-first century, technology has taken a major place in the real world. There are many things that can be done by sitting from a lovable place. We all can see in our daily life how dependent we are nowadays with technology . Today, I am discussing here to find all the informations of major airports of Germany . Many taxis, scooters and bicycle renting companies can get benefited with the informations which I explain in this medium article.

The big advantage of modern technology to a company is optimising everything and increasing the profit. For example, if the scooters company distributed many rented scooters to a big airports where there was maximum chance of raining then the company would have to bear the costs while distributing the scooters to different places. But once the company knows the weather conditions and the number of flights landing in each big cities then they can analyze the right quantity of scooters to be distributed in each places for renters.

Web scrapping:

We use Web scraping to collect the structured web data in an automation fashion. Web scrapping involves downloading a page and extracting the informations from it. This is the one of the important tool to bring the informations from wikipedia. The content of a page may be parsed and reformatted, and its data copied into a spreadsheet or loaded into a database. Web scrapers typically take something out of a page, to make use of it for another purpose somewhere else. For example we collect the informations of all the flights status of different airports of Germany using web scrapping(using BeautifulSoup) and bring out the necessary information which we require.



To use the webscrapping we use API keys which you have to find on your own

```
OWM_key = "your_key"
flight_api_key = "your_api_key"
```

We collect informations of all cities of Germany in a dataframe `df_cities` which have a large airports using BeautifulSoup function.

```
import requests
from bs4 import BeautifulSoup as bs
import pandas as pd
import unicodedata
city_lists=['Berlin', 'Dresden', 'Frankfurt am Main',
'Münster', 'Hamburg', 'Cologne', 'Düsseldorf', 'Munich',
'Nuremberg', 'Leipzig', 'Stuttgart', 'Hannover', 'Bremen',
'Dortmund', 'Karlsruhe']

countries=["DE" for i in range(len(city_lists))]

airports_icao=['EDDB', 'EDDC', 'EDDF', 'EDDG', 'EDDH',
'EDDK', 'EDDL', 'EDDM', 'EDDN', 'EDDP', 'EDDS', 'EDDV',
'EDDW', 'EDLW', 'EDSB']

cities=city_lists
def City_info(soup):
    ret_dict = {}
    ret_dict['city'] = soup.h1.get_text()
```

```

        if soup.select_one('.mergedrow:-soup-contains("Mayor")>.infobox-label') != None:
            i = soup.select_one('.mergedrow:-soup-contains("Mayor")>.infobox-label')
            mayor_name_html = i.find_next_sibling()
            mayor_name = unicodedata.normalize('NFKD', mayor_name_html.get_text())
            ret_dict['mayor'] = mayor_name

        if soup.select_one('.mergedrow:-soup-contains("City")>.infobox-label') != None:
            j = soup.select_one('.mergedrow:-soup-contains("City")>.infobox-label')
            area = j.find_next_sibling('td').get_text()
            ret_dict['city_size'] = unicodedata.normalize('NFKD', area)

        if soup.select_one('.mergedtoprow:-soup-contains("Elevation")>.infobox-data') != None:
            k = soup.select_one('.mergedtoprow:-soup-contains("Elevation")>.infobox-data')
            elevation_html = k.get_text()
            ret_dict['elevation'] = unicodedata.normalize('NFKD', elevation_html)

        if soup.select_one('.mergedtoprow:-soup-contains("Population")') != None:
            l = soup.select_one('.mergedtoprow:-soup-contains("Population")')
            c_pop = l.findNext('td').get_text()
            ret_dict['city_population'] = c_pop

        if soup.select_one('.infobox-label>[title^=Urban]') != None:
            m = soup.select_one('.infobox-label>[title^=Urban]')
            u_pop = m.findNext('td')
            ret_dict['urban_population'] = u_pop.get_text()

        if soup.select_one('.infobox-label>[title^=Metro]') != None:
            n = soup.select_one('.infobox-label>[title^=Metro]')
            m_pop = n.findNext('td')
            ret_dict['metro_population'] = m_pop.get_text()

        if soup.select_one('.latitude') != None:
            o = soup.select_one('.latitude')
            ret_dict['lat'] = o.get_text()

        if soup.select_one('.longitude') != None:
            p = soup.select_one('.longitude')
            ret_dict['long'] = p.get_text()

    return ret_dict

list_of_city_info = []
for city in cities:
    url = 'https://en.wikipedia.org/wiki/{}'.format(city)
    web = requests.get(url, 'html.parser')
    soup = bs(web.content)

```

```
list_of_city_info.append(City_info(soup))
df_cities = pd.DataFrame(list_of_city_info)
df_cities.head(5)
```

	city	mayor	city_size	elevation	city_population	urban_population	metro_population	lat	long
0	Berlin	Franziska Giffey (SPD)	891.7 km2 (344.3 sq mi)	34 m (112 ft)	3,769,495	4,473,101	6,144,600	52°31'12"N	13°24'18"E
1	Dresden	NaN	328.8 km2 (127.0 sq mi)	113 m (371 ft)	556,227	790,400[3]	1,343,305[2]	51°03'00"N	13°44'24"E
2	Frankfurt	Peter Feldmann[1] (SPD)	248.31 km2 (95.87 sq mi)	112 m (367 ft)	764,104	2,319,029[3]	5,604,523[2]	50°06'38"N	08°40'56"E
3	Münster	NaN	NaN	60 m (200 ft)	316,403	NaN	NaN	51°57'45"N	07°37'32"E
4	Hamburg	Peter Tschentscher (SPD)	755.22 km2 (291.59 sq mi)	NaN	1,845,229	2,484,800[1]	5,107,429	53°33'00"N	10°00'00"E

We collect weather information using web scrapping with OWM API key:

```
# look for the fields that could be relevant:
# better field descriptions
https://www.weatherbit.io/api/weather-forecast-5-day
all_weather=[]
# datetime, temperature, wind, prob_perc, rain_qty, snow =
[], [], [], [], [], []
# for forecast_api in forecast_apis:

for i in range(len(city_lists)):
    city=city_lists[i]
    country=countries[i]
    #response=responses[i]
    forecast_api = responses[i].json()['list']
    weather_info = []
    for forecast_3h in forecast_api:
        weather_hour = {}
        # datetime utc
        weather_hour['datetime'] = forecast_3h['dt_txt']
        # temperature
        weather_hour['temperature'] = forecast_3h['main']
    ['temp']
        # wind
        weather_hour['wind'] = forecast_3h['wind']['speed']
        # probability precipitation
        try: weather_hour['prob_perc'] =
float(forecast_3h['pop'])
        except: weather_hour['prob_perc'] = 0
        # rain
        try: weather_hour['rain_qty'] =
float(forecast_3h['rain']['3h'])
        except: weather_hour['rain_qty'] = 0
```

```
# wind
try: weather_hour['snow'] =
float(forecast_3h['snow']['3h'])
except: weather_hour['snow'] = 0

weather_hour['municipality_iso_country'] = city +
',' + country
weather_info.append(weather_hour)
all_weather.append(weather_info)

all_weather_data = pd.concat([pd.DataFrame(a) for a in
all_weather])
```

Airports detail of Germany using API keys of Aerodatabox:



google.com

In the following code we are extracting data using API of Aerodatabox of cities with time interval of 9 hours from actual time:

```
import requests
from IPython.display import JSON
responses=[]
from datetime import datetime, timedelta

for i in range(len(airports_icao)):
    to_local_time = datetime.now().strftime('%Y-%m-%dT%H:00')
    from_local_time = (datetime.now() +
timedelta(hours=9)).strftime('%Y-%m-%dT%H:00')
    url =
f"https://aerodatabox.p.rapidapi.com/flights/airports/icao/{airports_icao[i]}/{to_local_time}/{from_local_time}"
```

```

        querystring =
        {"withLeg": "true", "withCancelled": "true", "withCodeshared": "
        true", "withCargo": "true", "withPrivate": "false", "withLocatio
        n": "false"}
        headers = {
            'x-rapidapi-host': "aerodatabox.p.rapidapi.com",
            'x-rapidapi-key': flight_api_key
        }
        responses.append(requests.request("GET", url,
        headers=headers, params=querystring))

```

We collected all the information in **responses**. We apply **.json()** for each item of responses and collect the informations of flights which are landing in different cities and form a data frame **arrival_cities** as computed below:

```

import requests
from IPython.display import JSON
responses=[]
from datetime import datetime, timedelta

for i in range(len(airports_icao)):

    arrivals_list=[]
    for i in range(len(responses)):
        #for response in responses:
            arrivals_list.append(responses[i].json()['arrivals'])

def get_flight_info(flight_json, icao):
    # terminal
    try: terminal = flight_json['arrival']['terminal']
    except: terminal = None
    # aircraft
    try: aircraft = flight_json['aircraft']['model']
    except: aircraft = None

    return {
        'dep_airport':flight_json['departure']['airport']
    ['name'],
        'sched_arr_loc_time':flight_json['arrival']
    ['scheduledTimeLocal'],
        'terminal':terminal,
        'status':flight_json['status'],
        'aircraft':aircraft,
        'icao_code':icao
        #'icao_code':airport_icoa
    }

import pandas as pd
pds=[]

```

```
for i in range(len(city_lists)):
    pds.append(pd.DataFrame([get_flight_info(flight,
airports_icao[i]) for flight in arrivals_list[i]]))

print([a.shape for a in pds])
arrivals_cities=pd.concat(pds)
```

SQLAlchemy:



SQLAlchemy is the **Python SQL toolkit** and Object Relational Mapper that gives application developers the full power and flexibility of SQL. Python has different libraries to bring necessary informations using APIs. One can clean the data using pandas in python. To produce a database in SQL we create first a Schema in MySQL using the following one line code:

```
CREATE DATABASE project;
```

After creating the database the following code connects our python code to SQL:

```
import pandas as pd

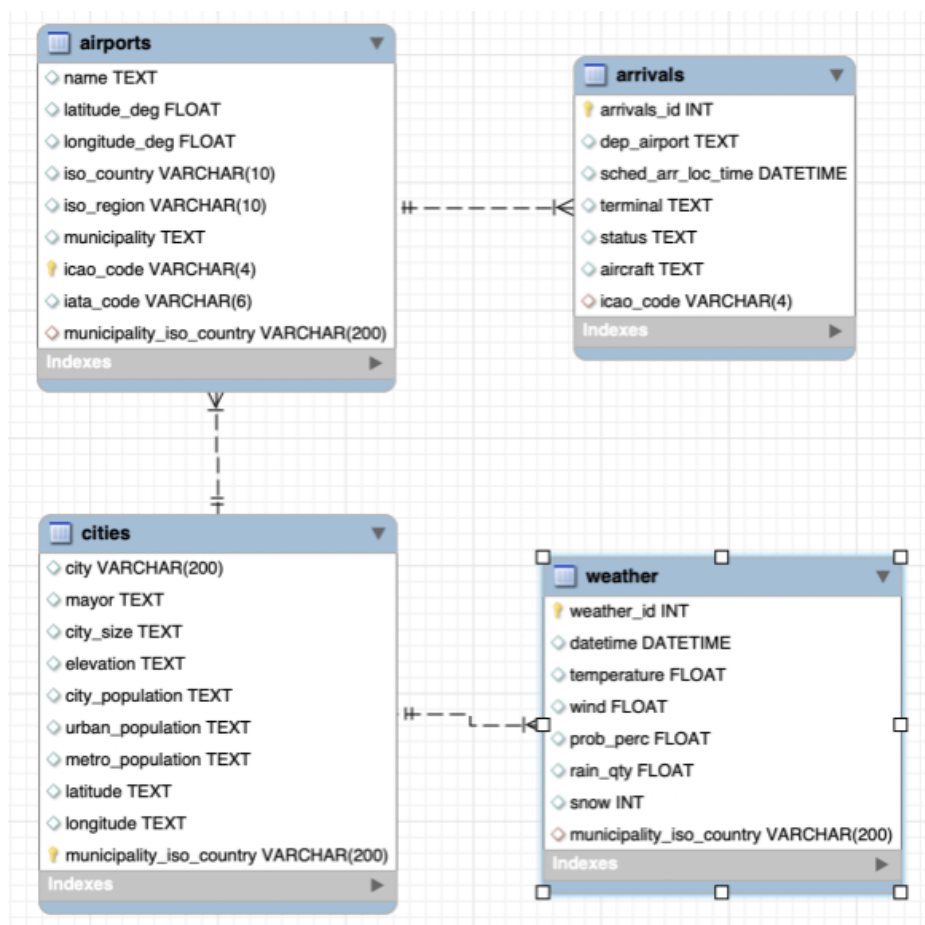
schema= "project"
host="abcd"
user="abcd"
password="password"
port=3306
con = f'mysql+pymysql://{user}:{password}@{host}:{port}/{schema}'
```

Once the connection is created between python and SQL Schema, we just need to hit the following code:

```
all_weather_data.to_sql('weather', if_exists='append',
con=con, index=False)
arrivals_cities.to_sql('arrivals', if_exists='append',
con=con, index=False)
df_cities.to_sql('cities', if_exists='append', con=con,
index=False)
```

in Python which sends all the information in our Schema. If you need more informations to be included then you extract informations using API keys and push it in your schema.

I have sent all the informations in my schema known as 'project' and produced the following ERR diagram:



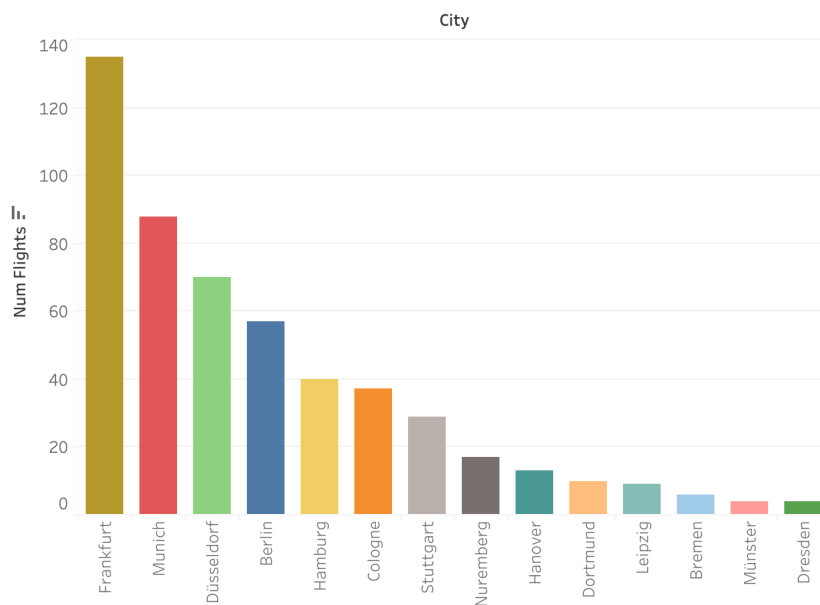
SQL-Tableau:

The data base 'project' consists of information in the time between 15:20–00:20 of August 09, 2022(For you change the time accordingly) from which I constructed a table which explains number of flights landing with other details:

	date	city	city_population	num_flights	time_hour	temperature	rain_qty
0	2022-08-09	Berlin	3,769,495	8	15	21.87	0.0
1	2022-08-09	Berlin	3,769,495	12	16	26.85	0.0
2	2022-08-09	Berlin	3,769,495	14	17	28.47	0.0
3	2022-08-09	Berlin	3,769,495	16	18	17.64	0.0
4	2022-08-09	Berlin	3,769,495	7	19	19.88	0.0
5	2022-08-09	Berlin	3,769,495	10	20	29.47	0.0
6	2022-08-09	Berlin	3,769,495	11	21	21.87	0.0
7	2022-08-09	Berlin	3,769,495	21	22	18.74	0.0
8	2022-08-09	Bremen	566,573	1	16	25.00	0.0
9	2022-08-09	Bremen	566,573	1	17	24.84	0.0

From above table we construct a graph which consists of all flight landing at aforementioned time.

number of flights landing in different cities



AWS(Amazon Web Service):



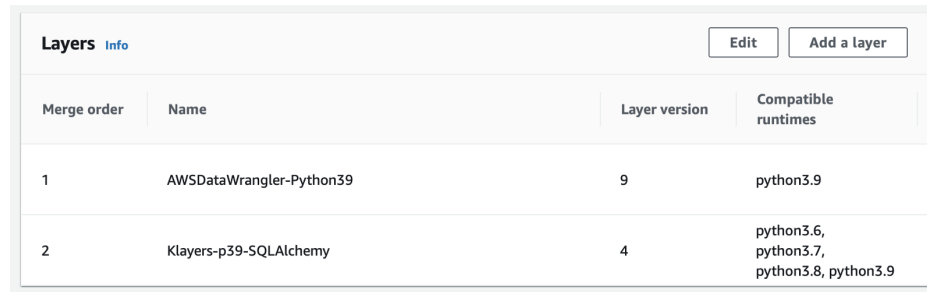
AWS is a cloud service from Amazon, which provides services including computing, storage, networking, database, analytics etc.. It is also used to form the building blocks to create and deploy any type of applications in the cloud. Building blocks are actually designed to work with each other, and result in applications that are sophisticated and highly scalable.

Update all the python code into AWS interface:

The beauty of this article is to update all the python code into AWS Lambda function to generate all the information automatically according to our choices. I generate all my code to find details of airports and the planes landing for next nine hours. With this output I generate two tables: one consists number of flights arriving in the airport along with weather informations and another with the similar informations but hourly. We can easily get the time of big demands of the scooters from hourly detail. We follow the following instructions to obtain our results:

1. Set the Lambda function on AWS:

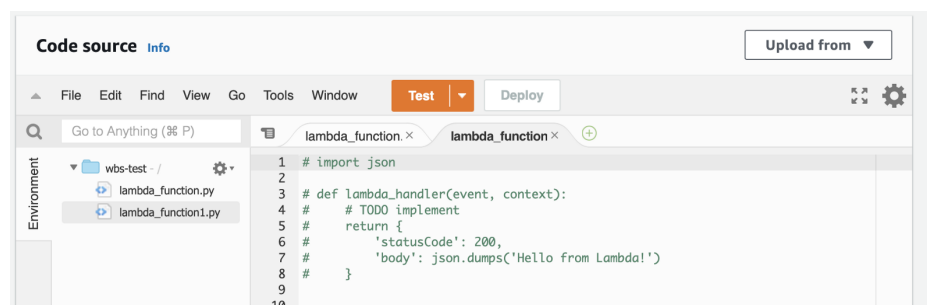
Once you have created an account in AWS you have to set a Lambda function. The Lambda function you have created does not consist of some of the python libraries, so you have to add these libraries on Add layer which you can find below on Lambda function.



Layers Info			
Merge order	Name	Layer version	Compatible runtimes
1	AWSDDataWrangler-Python39	9	python3.9
2	Klayers-p39-SQLAlchemy	4	python3.6, python3.7, python3.8, python3.9

2. Generate the python code:

Once you have added the python libraries you just need to set your code to `lambda_function.py` as shown below:



```

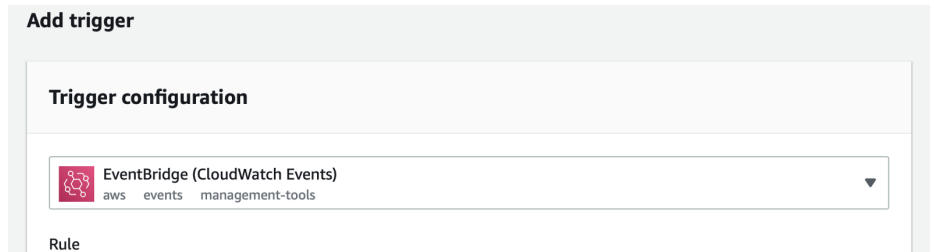
1 # import json
2
3 # def lambda_handler(event, context):
4 #     # TODO implement
5 #     return {
6 #         'statusCode': 200,
7 #         'body': json.dumps('Hello from Lambda!')
8 #     }
9
10

```

Apply **Deploy** button to save your code and then press **Test** button to produce the result which you require. One can set here the time accordingly to get the results when ever required.

3. Add Trigger:

In the Lambda function, we create an event trigger



using **CloudWatch Events** to produce the results regularly to set the plan for car/scooter companies to optimize the costs regularly. For example, I modify my code to collect all informations in the time between 0:00 to 23:59 every next day.

Contact:

Github: <https://github.com/aslam7861>

Email: aliaslam9439@gmail.com