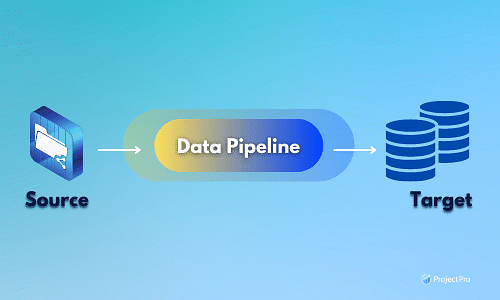
**MY DATA ENGINEERING PIPELINE TASK WITH PYTHON**



It all started after I was employed as a junior data engineer for Gans (a fictional company set up just for learning), an e-scooter-sharing start-up company. It aspires to operate in most populous cities all round the world. In each city, the company will have hundreds of e-scooters parked in the streets and allow users to rent them by the minute. Gans has seen that its operational success depends on having its scooters parked where users need them. The company wants to anticipate as many scooter movements as possible. Predictive modelling is certainly on the roadmap; however the first step is to collect more data, transform it and store it appropriately. This is where I come in:

My 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, the challenge is going to be to assemble and automate a data pipeline in the cloud. I decided to start the project from Germany because Gans is based there and facing tough e-scooter competitors already.

**WEB SCRAPING**

In the first phase of the project, data was scraped from the Internet and collected from APIs. Demographic data were collected for some regional strategic cities in Germany. This was done by web scraping the Wikipedia page (see Figure: 1) using a Python library called Beautiful Soup: a Python library for parsing HTML and XML documents. It creates a parse tree for parsed pages that can be used to extract data from HTML and XML. The find\_all method was deployed in Python script to locate the tag in the HTML where my interested table was located, then a code was written to extract the table of my interest.



Figure 1: Some demographic data in Germany

The extracted data was saved in a Dataframe. The data was later processed for storage. The entire code can be seen on my Github…….



Figure 2: Beautiful Soup

**API DATA COLLECTION**

Data about Airport locations, flights in and out of the Airports and the Weather of the cities of Gans interest were collected from APIs. Since Gan is interested in building predictive models of their e-Scooter movement, these data sets should be collected and automated to reflect today's weather conditions and airport flight activities and predict tomorrow’s weather conditions and flight activities in and around our interested cities. This is seen in the code below:

today = datetime.date.today()

tomorrow = today + datetime.timedelta(days=1)

**WEATHER API — OpenWeatherMap**

Weather data was collected using OpenWeatherMap web API. OpenWeatherMap provides global weather data via API, including current weather data, forecasts, nowcasts and historical weather data for any geographical location. An API key was given to me after 8 hours of signing up for an account with them which I used to exploit their data. I made an API call for their 4 days Hourly Forecast by City and Country name. The request code is seen in Figure:3 below.

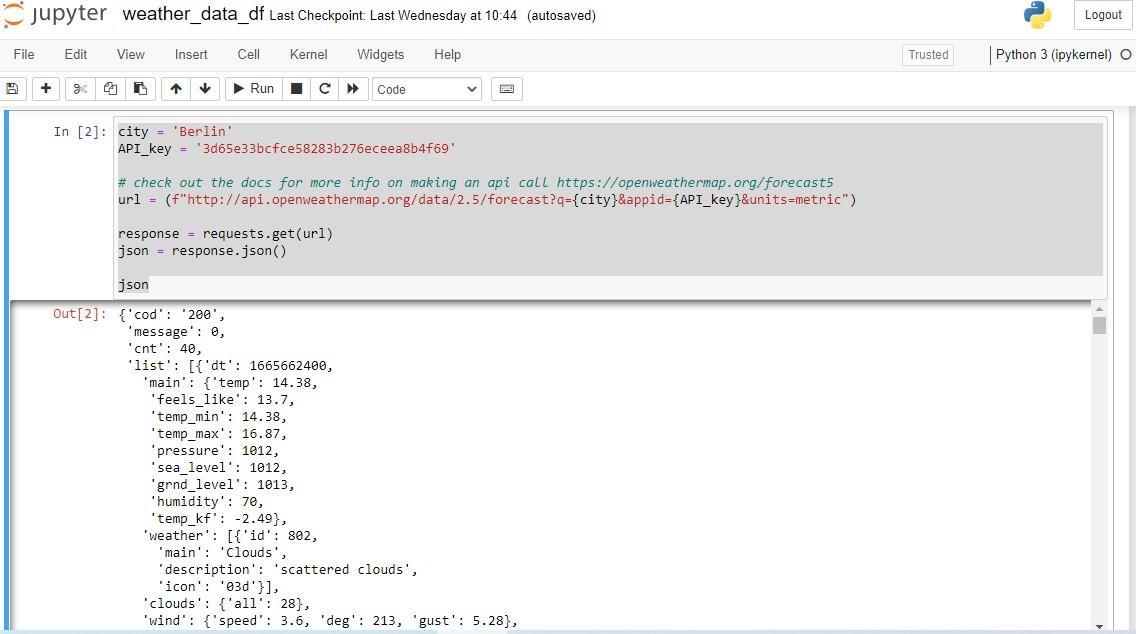


Figure 3: The request code used to collect Weather API data

The response I got from the request was saved in JSON format, this can however be changed to HTML or XML format by adding mode parameters. The response data in the JSON file was iterated and only the interested data were selected. The selected data after iteration were stored in the dictionary and converted to Dataframe for processing and storage. See my GitHub how the data were stored in Dictionary and DataFrame .

**FLIGHT API - AeroDataBox**

Airport locations, Flights arrival and departure to these Airport locations data were collected using AeroDataBox API. Access was given to this API data after I signed up for an account and subscribed for AeroDataBox API basic plan. I searched for Airport departures and arrivals data in the search section. See below, the search criteria created. I also searched for Airport locations in the Endpoints (see Figure: 4) and the codes from these searches were copied and pasted on the Jupyter Notebook where a request was made to the AeroDataBox API URL and the corresponding responses stored in JSON files. Python codes were used to iterate through the JSON files and the interested data sets were extracted and saved in DataFrames. See my Githubon the code used to extract this API and how they were saved in DataFrame.

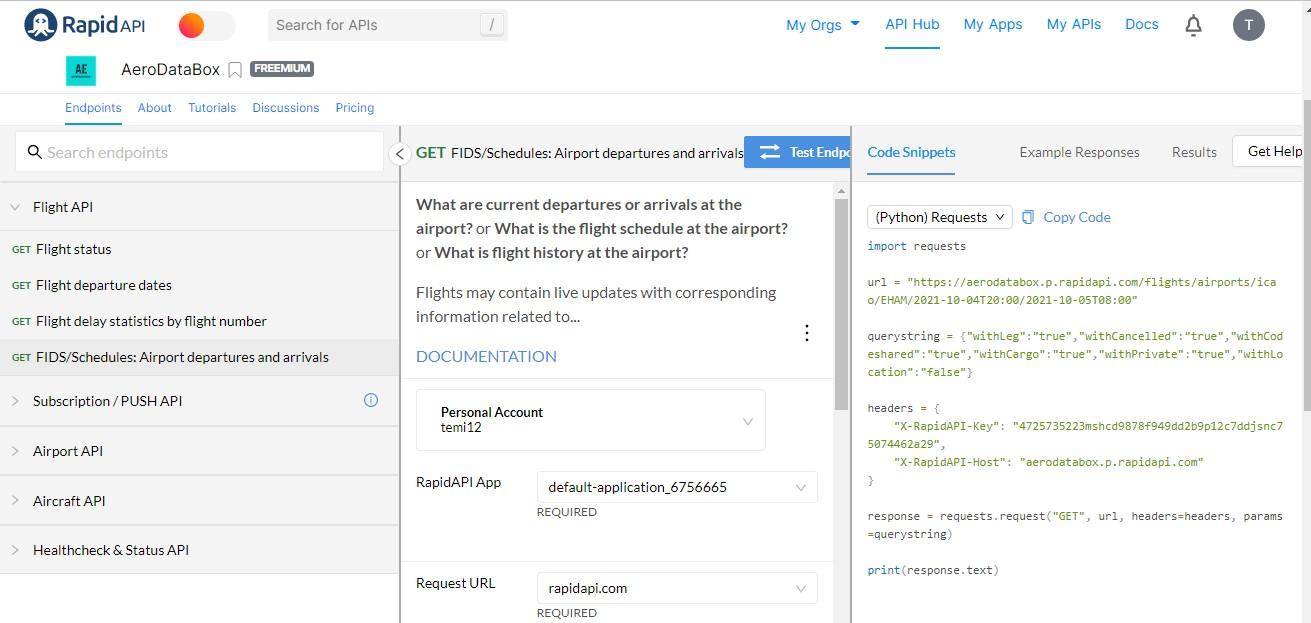


Figure 4: AeroDataBox API Endpoints

**DATA STORING**

The second phase of this project was to store the extracted data sets from web scraping and APIs. These data were stored in a MySQL database and I used MySQL Workbench (a graphical tool for working with MySQL servers and databases) to create and virtualize the database (see Figure: 5). The database and its tables were created by running a written script on MySQL Workbench and connected using primary and foreign keys. The script for creating the database can be seen on my Github.

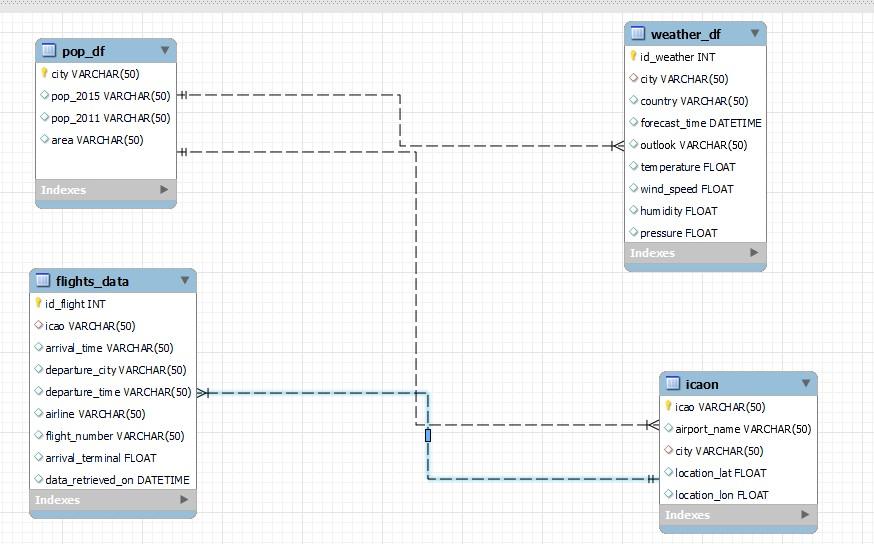


Figure 5: The EER Diagram for my project database containing 4 tables connected through primary and foreign keys

It was time to transfer the data from data frames in Jupyter Notebook to my newly created database in MySQL. In achieving this, I used a Python library called SQLAlchemy (a Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL). In the connection code (see Figure 6), schema is the name of the database I am connecting to; the host is the location of the database server, the user and password are the login details to the database server and the port refers to the optional database port. The mysql+pymysql refers to the type of the database and the DBAPI I am using. Con variable created and passed to\_sql() function, the data frame is sent directly to my created database in MySQL.



Fig 6: The code used in connecting pop\_df to my created database in MySQL

My flight data, flight location date (icaon), weather data and population data were all sent to my database in MySQL using this connection code, stored there and ready to be query at any time, see Figure:7. The database was at this time running on my pc’s local server and each time I want to update the dynamic data (flights and flight locations data) , I will have to run the python scripts again manually on Jupyter Notebook, but this can be eliminated by taking my created database to AWS Crowd.

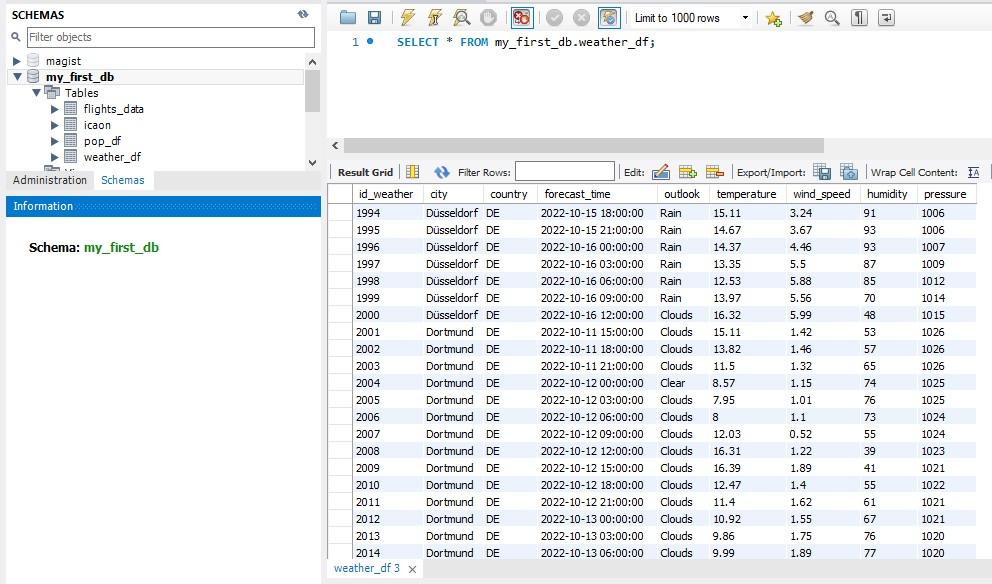


Fig 7: Output of a query from my database in MySQL local server

**TRANSFERING THE PIPELINE TO AWS CLOUD**

The last phase of the project was to automate my database in the cloud using Amazon Web Services (AWS) as a service provider. The first step was to open an AWS basic account. After that I created a cloud database using RDS Instance (see Figure 8). I also created a new database in MySQL Workbench and connect it to my created cloud RDS instance with a password and the endpoint of the RDS Instance.

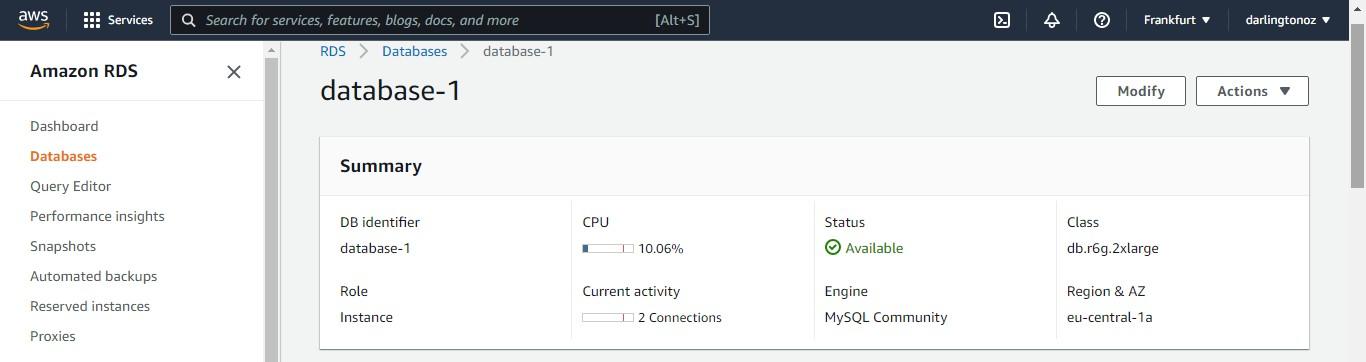


Fig 8: My created database in AWS

In the second step, I used an AWS service called Lambda function to transfer my data frames to a cloud database, See Figure: 9. AWS Lambda is a serverless compute service that runs your code in response to events and automatically manages the underlying compute resources for you. These events may include changes in state or an update. My Python code used earlier in transferring my data frames to my local server database in MySQL was used again but this time written as a function in a lambda\_handler() within the function and whenever the function is run the code within lambda\_handler() is triggered. I added a layer to the AWS Lambda for my code to get access to Python libraries that I used in my function code. I then created a new Role in the AWS Identity and Access Management (IAM). This will help me securely control who gets access and permission to my created Lambda function.

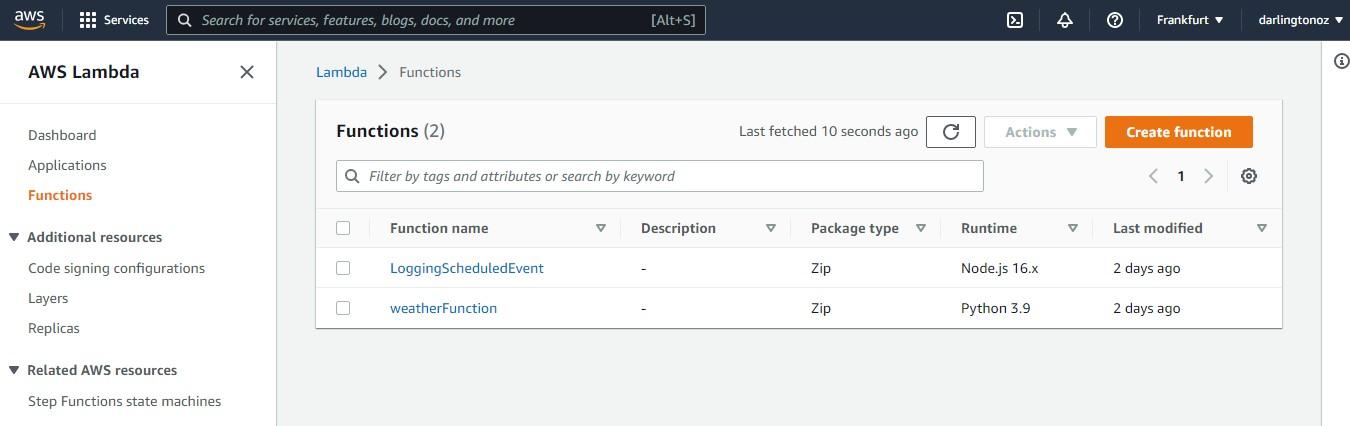


Fig 9: Lambda Function (weatherfunction)

The last step I took in this phase was to Automate my database in the cloud so it gets updated to my pre-set time without me manually running the function code. To achieve this, I used an AWS service called AWS EventBridge (a serverless event bus service that you can use to connect your applications with data from a variety of sources). It enables real-time access to changes in data in AWS services. All that needs to be done is to set the timing of the EventBridge and at the set time the EventHandler triggers my Lambda function and in response the function code will run and the database will be updated. In this Project, I set the EventBridge to trigger every day at 6am. From the Amazon CloudWatch Logs, I was able to monitor, observe and access log files from the automated data pipeline, see Figure:10.

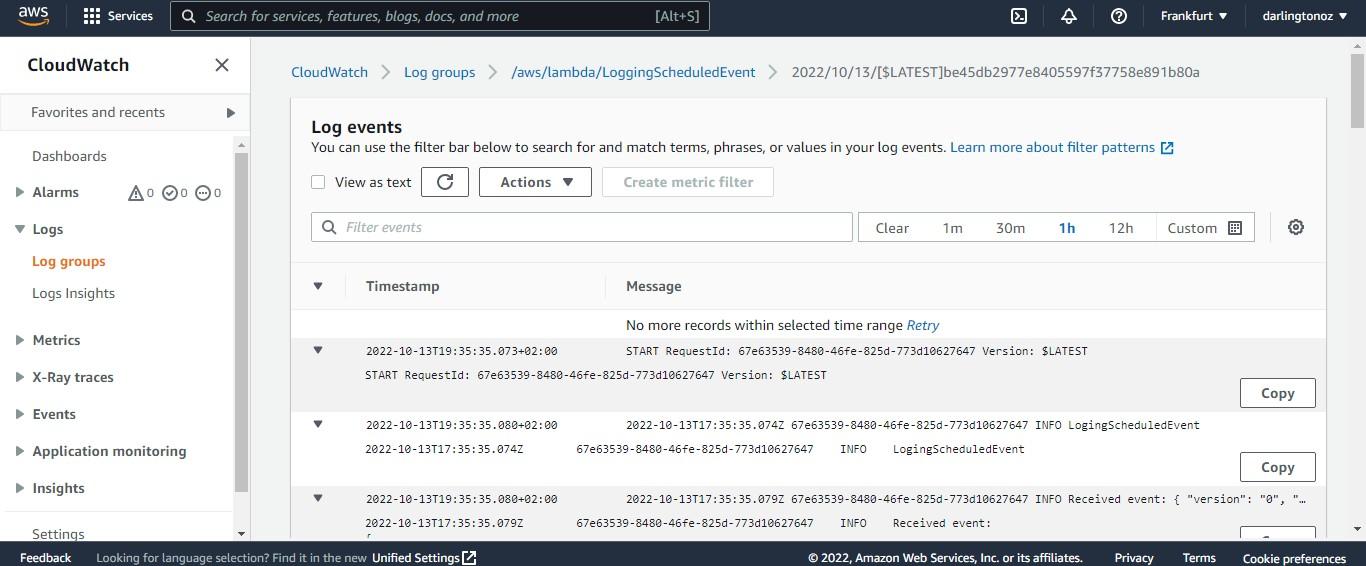


Fig 10: Log events of the automated data pipeline

**PROJECT SUMMARY**

I have been able to gather needed data for Gans to kick start their data predictive modelling Journey in Germany. These data sets are saved in database in MySQL and in AWS Cloud and I have been able to automate this process in AWS Cloud. The function code I set up in AWS Lambda is running in the Cloud every day at 6 am collecting data from the Internet and saving them to my database. The data in the database will continue to grow bigger and the Data Scientist will have to start analysing the data for Gans to get some valuable insights about the data. Mission accomplished! It was an insightful journey for me, I was able to learn more than just basic Data Engineering skills. I look forward to the next task from Gans.

Here is a quick recap on how i got it done:

* Wikipedia sites got scraped for information about population and cities in Germany
* APIs sites used in collecting flight and weather data
* database created and setup in the cloud
* code transformed to function and put in the cloud
* Function code schedule be run whenever needed