United Kingdom Hydrology Database

Eduardo Fernández Villazón

Udacity Data Engineering Capstone Project

06-Oct-2020

Scope of the Project and Choice of Dataset

The scope of this project is to create a database with hydrological data from historical datasets for different hydrological measures, such as rainfall, water level or water flow. Other climatic variables would welcome but are not considered key to the project.

The final purpose of this database will be to help understand the correlation between these variables using a very precise and frequent measurement of these variables. This potentially could be done at a later stage by training a ML algorithm with the data. This better understanding could help predict flooding in advance.

1. Choice of datasets:

With this objective in mind, I looked for the APIs provided by several countries to see which ones offered datasets with high-resolution measurements and with multiple hydrological variables. I found that the UK government provides with 2 different APIs the main variables that I seek to study. In these APIs, the variables in scope appear with a frequency of 15 mins/measurement for many stations. The two APIs I will be using in the project are:

- https://environment.data.gov.uk/hydrology/
- https://environment.data.gov.uk/flood-monitoring/

At the time of the project both services are considered to be in a beta phase and are provided free of charge under the Open Government Licence v3.0, which requires me to attribute the source of the data. In fact, in some occasion I found that the API was not working as it specified in the documentation.

2. Choice of Tools:

With the objective of building a database with a great amount of hydrological data and for it to be kept up to date, I chose to develop the ETL using Airflow. The Airflow environment has been setup in a Docker container for development and portability purposes. Some ad-hoc code has been developed as well, to allow the container to run on an EC2 machine.

The final dataset will be stored in a data lake, using columnar storage (parquet), which in the future will be used to feed the ML algorithm.

Data Exploration and Assessment

1. Dataset Exploration Hydrology API:

This API provides information from 937 measuring stations. Not all of the stations provide 15-minute measurements. Even though the API documentation describes that it can provide up to 4 variables, in practice this API only provides the hydrological variable "water flow", which is a measure in m^3/s of the amount of water circulating in a river.

The amount of stations (order of 1000) should provide and order of 100000 measures per day analysed in the database, in total (97 measures per day and station for a 15-minute period between measures). The API returns daily measures from up to 2005, so potentially we could have up to the order of 500 million measures for this variable.

Note: we will see, however, that the real amount of measures is much smaller as not all stations collect data every 15 minutes since 2005.

2. Dataset Exploration Flood Monitoring API:

Similarly, this API provides 948 measuring stations for the parameter "rainfall", and 3482 measuring stations for the parameter "water level". Rainfall is the amount of water collected (in mm) in a rain gauge or equivalent. The water level is a relative measure in m of the level of the water in a specific location. For the same timeframe, these stations could provide up to 2300 million measures for both variables with a 15-minute frequency.

Note: we will see, however, that the real amount of measures is much smaller as not all stations collect data every 15 minutes since 2005.

3. Dataset Sizing:

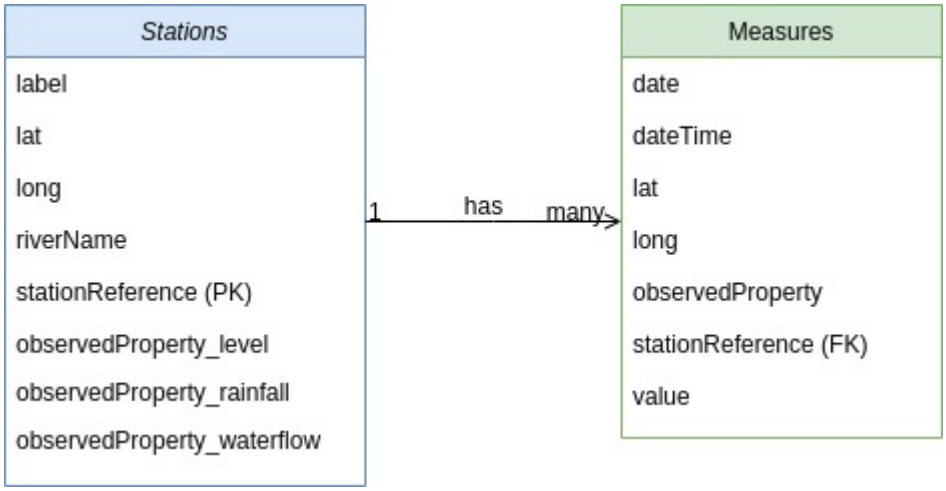
Even though the real amount of measures available will probably be much smaller than the 2800 million estimated earlier, it is clear that we will have no problem in obtaining 1 million lines of data. For this project, we will manage around 1 million lines of data, however, the project will be developed as to be easily scalable if needed.

In a later stage, when I use this data for the final use intended, I will reuse the project and obtain the maximum amount of data available.

Data Model Definition

By simple observation of the preliminary results of the queries we realise the following:

1. A single station may measure one or more hydrological properties. Information on the stations will only need to be recorded once, and will therefore constitute the dimension table. The stations will be recovered from both APIs and merged into a single database.
2. The measures of said properties will constitute a fact table which will record the individual measurement as an event in time and space. The reason for this is that it would seem reasonable to consider that the variables may be intercorrelated not only in time, but also in space. Each measurement is associated to only one property.



1. *The stations dimension table:*

This table will be formed with the stations from both APIs, and give specific information of each station. For each station we will record which parameters it measures (one hot-encoded columns), the name of the station as “label”, its latitude and longitude, the name of the river which it measures (if any), and the station reference, which will be the Primary Key for this table.

Once clean, the stations table will be used as a reference by the rest of the operators to call the API.

2. *The measures fact table:*

In this table we will record the value of the measure, under “value” and its type under “observedProperty”. The moment the measure is taken is recorded in the “dateTime” column, and a “date” column has been included, which will be useful for partitioning the data. We will include the geographical coordinates of the station, for computational purposes, as well as the station reference which will be the foreign key allowing us to link with the dimension table.

The “stationReference” and “observedProperty” columns will also be used for data partitioning and the primary key will be composed of 3 columns: “stationReference”, “observedProperty” and “dateTime”. They are the minimum way identify a single measure.

The choice of “stationReference”, “observedProperty” and “date” columns is found to be optimal from the computational perspective as the number of measures per station, per property and date will always be the same: 97.

Note: the partitioning will only take place at the final stage of the project, when it is stored in parquet format for future use.

Data Cleaning

In order to clean the data, we will have to perform a series of steps. Both the fact table and the dimension table are generated from data from the 2 different APIs. The APIs are called with the parameter included in the request. This means that in the dimension table, we could potentially find that a single measurement station appears several times for different observed properties. To solve this issue, we will use a staging table where we will copy all the data and where the cleaning will be performed. The cleaning consists mainly of 3 steps:

1. We will drop the null values for the station reference, which will be our unique identifier.
2. We will one-hot encode the observed property parameter, allowing to aggregate the stations that measure more than one parameter.
3. We will drop the duplicate values for the station reference.

Then we will insert this data into the final table with a Primary Key constraint for the “stationReference” column, which will act as a quality check for the data.

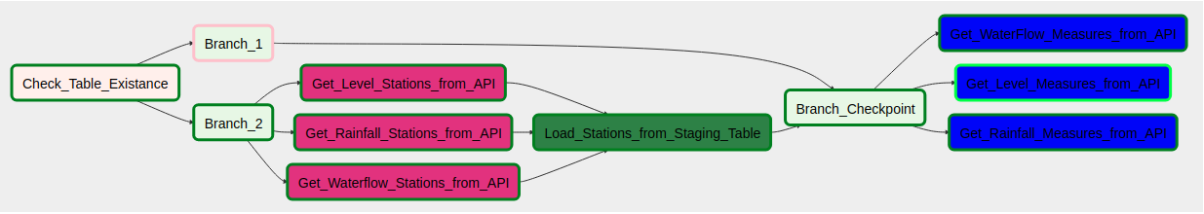
The fact table is filled by calling the APIs with the already clean stations from the station table, and they will be called once per day (dag scheduled daily) for each property. To clean the measures table, we will drop the null for “value”. There can be no more nulls in other columns as they are values used to call the API or values coming from the stations table. For some observed properties we have to add the date column extracting the value from the dateTime attribute, for consistency with the data model.

After cleaning the data a Primary Key constraint is imposed on the ensemble ("stationReference", "observedProperty", "dateTime"), which means that there can only be one measure from a certain station, for a certain property at a given moment in time. This serves as a second quality check on the data.

In both tables, the columns are sorted in the dataframes before being loaded to the SQL table to ensure the columns always appear in the same order.

ETL DAG

The design for the DAG that Airflow runs for the ETL pipeline is as in the graph shown below:



Alternative Scenarios:

1. *Data increased by 100x:*

This would be no problem for the ETL to handle as a full DAG run for a day takes only a few minutes and would therefore go up to a couple of hours to process (1 container). This might or might not be acceptable depending on the final use of the machine learning application that would be developed. If this application were to be time critical, execution time could be reduced by deploying more workers, or allocating more resources to the container. Eventually several containers could be deployed in parallel, however, the container developed for this application would not support this characteristic directly and would require some adaptation.

2. *The pipelines would be run on a daily basis by 7 am everyday:*

The current pipeline is scheduled to run on a daily basis, so this would not be an issue at all. We would just need to specify this time in the schedule in Airflow.

3. *The database would need to be accessed by 100+ people:*

The final dataset is stored in parquet format, so it should support hundreds of people for read only access to the database, given that the file is hosted on a platform that supports this volume of access. Write access to the database is discourage, and data cleaning should be refined at the pipeline level.

Defending Decisions:

1. *Choice of tools & technologies:*

I think Airflow was the right decision to future proof the project for larger volumes of data and because of its scheduling function which allows us to easily add only the new data for that time period. On the downside, it added a full layer of complexity for development and debugging and required the deployment of a Docker container to redress these complexities. Adding the Docker container added in itself another layer of complexity as I had to learn how to use Docker for this purpose.

The project code has been developed to store the data on the AWS S3 platform, but this could be easily changed. Also, because the application has been developed in a container, it can be easily deployed in any machine, or potentially deployed in parallel with other copies of the container, with some small adaptations.

To handle the dataframes I decided to use pandas, as it is an industry standard and provided all the necessary functionality.

2. Data Model:

As of today, it seems that the data model will be able to solve the problems it was set out to solve, however, we will not know this for sure until I start with the machine learning part of the project at a later stage. It might be interesting to include other weather or climatic variables in the future but I think that even then the data model would be able to easily accommodate it.

Annex: Project Instructions:

To run this project there are two options: to run the project in the local machine or to run them in an EC2 AWS machine. In both cases, the final result will be stored in an S3 bucket (uncomment in the code), which requires AWS S3 credentials to be stored in an Airflow connection.

1. *Local machine:*

i. Pull the GitHub repository: https://github.com/EduardoFernandezVillazon/Docker_Airflow_Setup_UK_HDE.git
ii. While on the project directory run the commad: ***sudo docker-compose -f docker-compose-LocalExecutor.yml up*** to set up the Airflow Webserver container and the Postgres container.
iii. Enter the Airflow UI in port 8080 and add the connection called "aws_credentials" that give read and write access to S3.
iv. Unpause the DAG and run until completion.

2. *EC2 Machine:*

i. Run AWS configure and enter root credentials.
ii. Pull the GitHub repository: https://github.com/EduardoFernandezVillazon/UK_Hydrology_Data_Engineering.git
iii. Run the scripts create_cluster.py and quick_SSH.py. The first one will launch an EC2 cluster and the second one will generate the bash command to SSH into it.
iv. SSH into the cluster and wait until the launch script is completed (check the launch logs).
v. Run cd .. twice then run ***sudo docker exec -it docker_airflow_setup_uk_hde_webserver_1 /entrypoint.sh bash -c "./docker_automated_script.sh"*** to start the airflow webserver.
vi. You will be prompted to introduce your S3 credentials for the "aws_credentials" connection.
vii. The DAG will start running automatically. You can use delete_cluster.py to clean up the resources when the tasks are completed.