Implementation of a (Big) Data Management Backbone

Eliya Tiram
Ximena Moure Alassio
Miona Dimic

1. Introduction

Purpose of this project is to implement a Data Management Backbone following the DataOps Agile approach. Data Management backbone represents a data pipeline consisting of multiple zones that function independently but together form a foundation for the data analysis tasks. This pipeline is presented in the picture below.

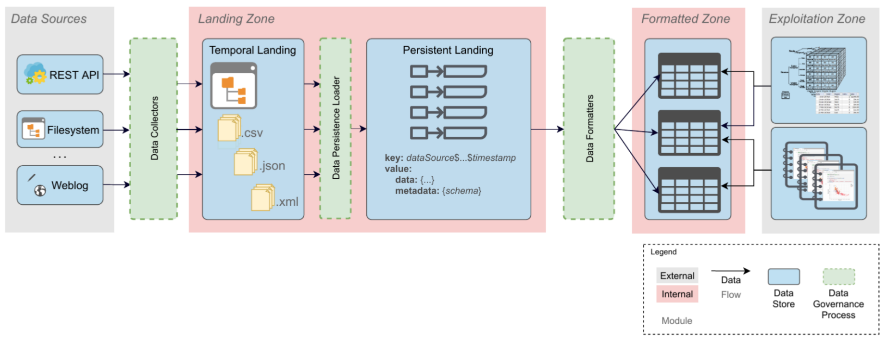


Figure 1.1 - Project high level view

As illustrated in figure 1.1, we distinguish between external and internal zones, together with processes that lead from one zone to another.

Following the pipeline from the start, given the data sources to be used for the purpose of any end-to-end Data Science project, we tend to store that information internally in a zone called *Landing zone*.

Landing zone consists of two subzones - *temporal* and *persistent*. Temporal zone serves as a placeholder for original files, taken as they are, without performing any changes or formatting. It is populated with a certain Data Collector, used to ingest external data to the internal storage. Existence of this zone enables us to backtrack to the start, in case we decide to approach differently, add or remove initial files, etc.

On the other hand, persistent zone represents a fixed and previously defined way to store the data in a manner of right naming convention and file format needed. It is populated with a Data Persistence Loader, usually a script that iterates through all files from the temporal zone, performing the necessary modifications, and storing each in a new storage location called persistent zone.

Data Formators are then used to store persistent data into chosen database. The resulting *formatted zone* should consist of tables corresponding to each source, having the corresponding files merged per each data source. On these we can then perform necessary ETL operations to insure what is needed for integration.
Once sources are integrated, we enter the *Exploitation* zone. This zone represents the basis for Data Analysis Backbone.

In the following sections we explain implementation of data injection into temporal and persistent zone, given the following data sources:

2. Data Sources

To conduct a project, we were provided with three data sources.

1. **Idealista (Barcelona rentals)** consists of json files downloaded from the Search API of https://www.idealista.com/en/. Files are describing daily booking offers posted on the Idealista website, referring to places in a specific area of the city of Barcelona.

2. **Open Data BCN (Territorial distribution of income)** consists of csv files representing the territorial income distribution in the city of Barcelona at the neighbourhood level in years 2007-2017.

3. **Lookup tables -** one for Idealista, and one for Open Data BCN, that map each distinct value for neighbourhood and district to its corresponding Wikidata ID.

Additional data source that was added:

4. **Open Data BCN (Average monthly rent and average rent per surface of the city of Barcelona)** - given per district and neighbourhood, for years 2014 - 2022.

Rationale for choosing this specific data source was the possibility to compare rental prices and distribution of income, considering the size of surface and specific location in the city of Barcelona.

3. Data Structure Design

We have decided to work with **HDFS** together with a **parquet** file format. HDFS is highly scalable and can process large amounts of data in a distributed environment. It also supports fault tolerance to ensure availability. We assume the most of the workload is going to be in the persistence layer. The data sources we work with contain many columns that from an analytical perspective counts as surplus, especially in *idealista* data source. Thus, we understand that a lot of the work in the future pipeline zone, in the formatted zone, would be based on projection.
Parquet files have the ability to partition the data horizontally (row groups) and also vertically within the row groups. Once the data is loaded to the persistence landing it stays there in case we want to retrieve a new column in the formatted zone. For example, a new KPI that needs to be added to existing ones and uses a column that doesn't exist in the formatted zone at the moment.

In addition, parquet files have the ability to **compress** and **sort** the data. Parquet files are compressed using columnar compression techniques, which means that data for each column is compressed separately. This allows for more efficient compression, as data within a column is often similar. Each column is compressed independently, so a different compression algorithm or compression level can be used for each column. This allows for greater flexibility and optimization of compression.

Furthermore, parquet files also have the ability to compress complex **nested data** for each level of the hierarchy. As a result, it enables more efficient storage and faster query performance [Parquet Apache]. Parquet files also have the ability to sort the data based on one or more columns. Sorting can be useful for improving query performance by reducing the amount of data that needs to be read from disk [Amazon Blog Experiment - section 5]. When data is sorted in a Parquet file, it can be read more efficiently because records with similar values are grouped together. Sorting is not for free and it's important to carefully consider the columns to sort on and the order in which to sort them to ensure that the benefits of sorting outweigh the additional processing time required [Medium]. In this project we are loading historic data from previous years and each datum is loaded once. However it can be read many times, thus the mechanism of HDFS and parquet files is adequate

4. Data Pipeline -Temporal and Persistent zone

We explain the first two stages of the pipeline: the first is to upload the data to the temporal landing zone from an external machine, given the data that is locally on a machine or as an external source like *opendatabcn* where we use http request. The second stage is to transfer the data to the persistence layer. Figure 3.1 describes the data flow within and between these stages.

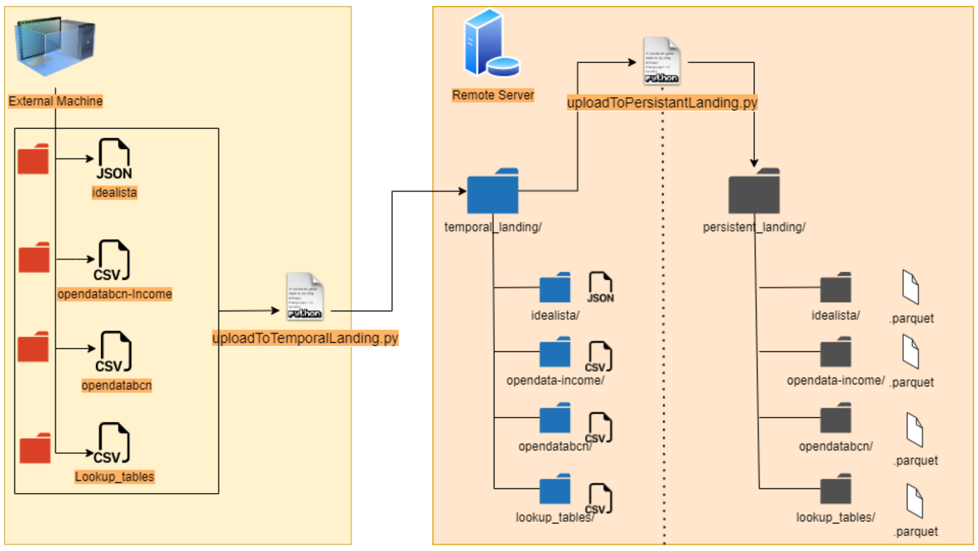


Figure 3.1 - Process diagram of the dataflow

Following process steps ensure creation of both temporal and persistent zone:

4.1. Data Collector

As a starting point, we firstly downloaded the provided datasets (Idealista, OpenData BCN and lookup files) manually into our local machines.
Then, we run a script called `opendatacollector.py` to automatically collect data about the mentioned sources.

As previously mentioned, the data collector should ensure that the **temporal zone** is populated.
It takes the data from the external source and uploads them to the remote server (virtual machine), to the temporal landing using HDFS (Hadoop Distributed File System). The files are being uploaded as they are with no change or manipulation.

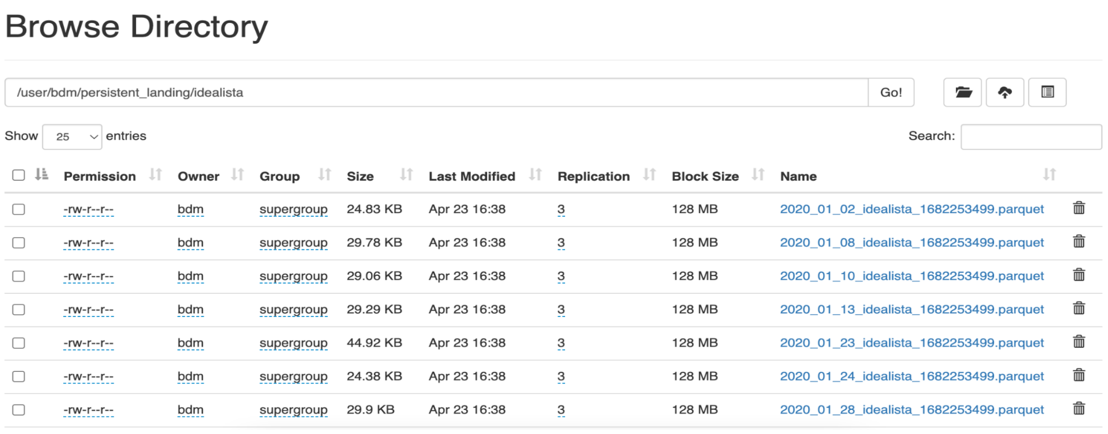
The python script *`uploadToTemporalLanding.py`*, being the data collector, is also taking care that there will be no duplicate data, even though in this architecture it is planned to be run only once. This solution is robust because if we add more data to the existing folders no further actions would be needed. We also have HDFS scalability which ensures that large amounts of data would also be supported.

4.2. Data Persistence Loader

Once the temporal zone has been created and populated, we proceed with Data Persistence Loader - `uploadToPersistantLanding.py`. This loader takes the files from the HDFS temporal landing directory, iterating through all existing subdirectories corresponding to all data sources added. Looping over the subdirectories, it converts any JSON or CSV file into Parquet format.

In order to perform conversion correctly, it was necessary to specify the schema upfront.
This ensures efficient data storage and retrieval, maintaining data accuracy and consistency, and preventing errors when querying the data.

Once conversion is complete, the loader writes the resulting Parquet files to the corresponding subdirectories in the new HDFS directory.
To each file we add a corresponding timestamp, representing the date of indigestion into the persistent zone.
As an example, in figure 4.2.1, we show content of one of the persistent subdirectories representing the files corresponding to *idealista* data source.


Browse Directory

/user/bdm/persistent_landing/idealista

Show 25 entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	bdm	supergroup	24.83 KB	Apr 23 16:38	3	128 MB	2020_01_02_idealista_1682253499.parquet
-rw-r--r--	bdm	supergroup	29.78 KB	Apr 23 16:38	3	128 MB	2020_01_08_idealista_1682253499.parquet
-rw-r--r--	bdm	supergroup	29.06 KB	Apr 23 16:38	3	128 MB	2020_01_10_idealista_1682253499.parquet
-rw-r--r--	bdm	supergroup	29.29 KB	Apr 23 16:38	3	128 MB	2020_01_13_idealista_1682253499.parquet
-rw-r--r--	bdm	supergroup	44.92 KB	Apr 23 16:38	3	128 MB	2020_01_23_idealista_1682253499.parquet
-rw-r--r--	bdm	supergroup	24.38 KB	Apr 23 16:38	3	128 MB	2020_01_24_idealista_1682253499.parquet
-rw-r--r--	bdm	supergroup	29.9 KB	Apr 23 16:38	3	128 MB	2020_01_28_idealista_1682253499.parquet

Figure 4.2.1 - persistent subdirectory - *idealista* data souce

5. Automatisation

Although each of the processes leading from one zone to another are functionally independent, only once they are being executed consequently, they form a functional DataOps pipeline.
Thus, we decided to create `run.sh` - a shell script file used to execute a set of commands in a predefined sequence. We assume that the mandatory data is already manually downloaded and stored locally.

In order to run the script, it is necessary to set the environment variables in the .env file correctly. It is important to have the environment variables set not only for running just the run.sh script but also for running all the scripts separately.
The following is an example of the .env file:

```
MACHINE_IP=10.4.41.48
MACHINE_PORT=9870
USER_NAME=bdm
PASSWORD=bdm
LOCAL_DATA_PATH=/Users/Desktop/data
LOCAL_PATH_DIRECTORY=/Users/Desktop/data/opendata-rent/
OPEN_DATA_URL=https://opendata-ajuntament.barcelona.cat/data/en/dataset/est
-mercat-immobiliari-lloguer-mitja-mensual
TEMPORAL_LANDING='/user/bdm/temporal_landing'
PERSISTENT_LANDING='/user/bdm/persistent_landing'
```

The first 4 variables are used to connect to HDFS in the virtual machine.
`LOCAL_DATA_PATH` is the local path to where all the data is going to be stored before uploading it to HDFS. In this case, it is going to be stored in a folder called data.
`LOCAL_PATH_DIRECTORY` is the local machine path, where the data from the third dataset is going to be stored. We assume that all the data (the mandatory datasets and the third chosen by us) is going to be stored in the same folder, in this case the one called data. The

folder inside data (opendata-rent) is created when collecting the data so it doesn't have to be created manually, but it has to be specified here.
`OPEN_DATA_URL` is the url from where we are getting our data for the third dataset.
`TEMPORAL_LANDING` is the path to where the local data is going to be stored in HDFS.
`PERSISTENT_LANDING` is the path to the persistent landing in HDFS.

Once the environment variables have been set the run.sh script can be run. It first collects the data for the third dataset. Then, it uploads all the data inside the local folder specified in the .env file to HDFS to the temporal landing and finally it moves all the files from the temporal to the persistent landing.

6. Conclusions

DataOps pipeline represents a powerful methodology that combines principles of agile development, allowing us to extract, transform and analyse data in matter of functionally independent steps, that together form a complex system.
Each step is responsible for specific functionality that moves the data from one state to another. Thus, backtracking to each of those processes, allows us to easily modify each of those functionalities and observe how they affect the data in the next step.

Once each of the sub-processes have been tested, accomplishing the desired transformations from one zone to another, we were able to construct the script that will automatically perform all of those transformations in one run, executing all subprocesses in a predefined order.
For the purposes of this project, specifically focusing on the Data Management backbone, in the next steps we plan to continue with the construction of formatted and exploitation zones.