Big Data Management

Project (P1): Landing Zone

Master in Data Science

April 4th, 2022

Team-Locals11-P1-H

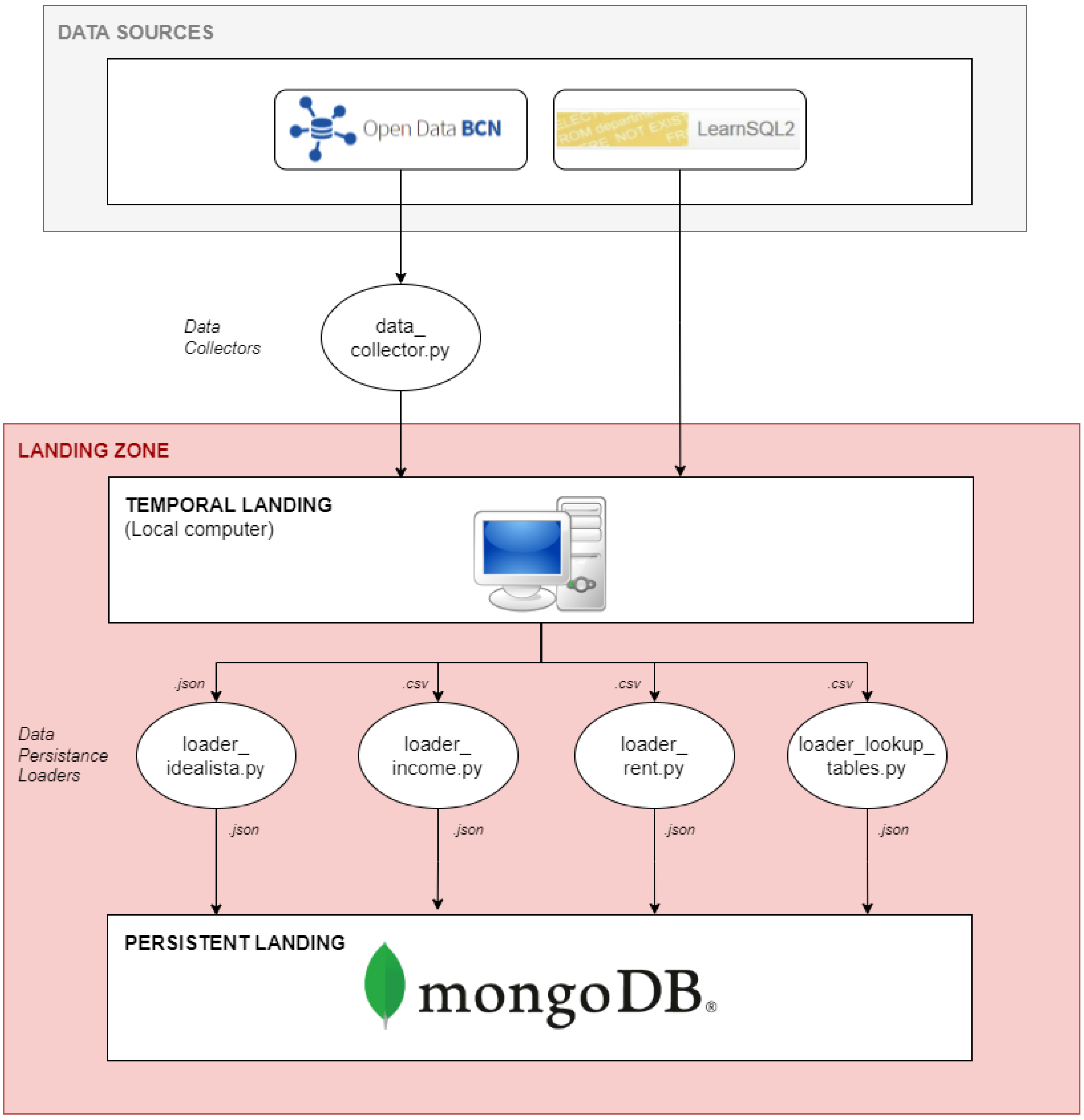
Meritxell Arbiol Lopez..................... meritxell.arbiol@estudiantat.upc.edu

Andrea Iglesias Munilla.................. andrea.iglesias@estudiantat.upc.edu

Introduction

In this project we have implemented the first part of the Data Management Backbone process: having the third data source chosen by us, we have created a Data collector with python so that automatically the script collects all the dataset from the chosen data source (the chosen one is from OpenData BCN) and download in in the Temporary Landing (in our local computer). Then, through specific loaders for each dataset (the one chosen by us, and those provided to us for the project), we ingest the data into the Persistent Landing.

In the image below, we have a detailed diagram in which we can see more clearly each of the steps we have followed:



Identify Data Sources

The datasets we have worked with are:

- Idealista dataset → (JSONs) provided to us
- Income dataset → (CSVs) provided to us
- Lookup tables → (CSVs) provided to us
- Rental dataset → (CSVs) new dataset

This last dataset is the one that has been chosen by us, we have found it in OpenData BCN. In it we have information of Average monthly rent (€ / month) and average rent per surface (€ / m2 per month) of the city of Barcelona. The URL address to which this data source is located is:

https://opendata-ajuntament.barcelona.cat/data/en/dataset/est-mercat-immobiliari-lloguer-mitja-mensual. The format of this dataset is CSV.

Implement Data Collectors

First, the files we need to use for the project are found in the Big Data Manager course in learnsql2. Therefore, our first source is **learnsql2** from where we manually download the data folder containing the necessary data in the idealista, *lookup_tables* and opendatabcn-income subfolders. To carry out the data ingestion process, we assume that the entire folder has been downloaded and it is located in the same project/directory where the main program is launched.

Regarding the third data source that we have chosen, we have created a program that extracts all the CSV files found on this source directly from **open data BCN**. The program, *data_collector.py*, uses the request and BeautifulSoup of bs4 libraries so that from a URL, it finds all the CSV files that are available there, and downloads them into our folder called data, in the subfolder opendatabcn-rent.

Temporal Landing

After we have used the data collector, we will have on our computer all the files that we need in our project located in the *data* folder. Therefore, our temporal landing will be found on our **local machine**, more specifically in the *data* folder with its respective subfolders and files.

Data Persistence Loaders

We have created a different Load program for each source to send the data from our computer into MongoDB. This way it is possible to add more sources and create more Loaders programs in an easy way. At the same time it is also easy to load only some of the sources and not all of them by commenting the unwanted loader lines in the main program.

The names of each of the programs created to do the Load of each data source are:

- *loader_idealista.py*
- *loader_income.py*
- *loader_lookup_tables.py*
- *loader_rent.py*

One thing to keep in mind in this step is that before ingesting anything with any of the implemented loaders, we decided to clear the MongoDB collections each time before ingesting data to avoid having duplicate data.

Idealista

The loader used for the idealista dataset is called *loader_idealista.py*. First of all, we create the connection between mongoDB and our local computer. Before sending the data to the collection, we parse the name of each file and extract the extraction date to insert that date in each json and not lose that information, which is important for removing duplicates in the future and keeping the latest update. Finally the json files are sent to mongoDB and inserted into the “idealista” collection using the insert_many function.

Income

The loader used for the income dataset is called *loader_income.py*. In this program we change the file format creating a JSON of each record of the CSV file, we create the connection between mongoDB and our local computer, and then we do the insertion. This data is inserted in the collection called "opendatabcn_income".

Lookup-table

The loader used for the lookup-table is called *loader_lookup_tables.py*. In this program we change the file format from CSV to JSON, we create the connection between mongoDB of our VM and our local computer, and we do the insertion. The collection where it is inserted is called "lookup_tables".

Rent

The loader used for the rent dataset is called *loader_rent.py*. Once the data collector is done, we have the CSV files in our Temporal Landing. To ingest them to our Persistent Landing (to MongoDB) we have the loader module: we change the file format from CSV to JSON, create the connection between the MongoDB of our VM and our local computer, and we do the insertion. We ingest it in a new collection called "opendatabcn_rent".

Persistent Landing Zone

We have chosen for the persistent landing to ingest the different datasets MongoDB. We assume that during the ingestion, the mongod server of the virtual machine is started.

One of the main reasons why we have chosen MongoDB is that most of the datasets we have are in JSON format since the number of files we have from Idealista is much larger than those from OpenBCN (rent and income datasets), and MongoDB stores in JSON.

Another point in favor is that MongoDB is more robust than HBase, so it can be more flexible to possible changes. This is important because we have to take into account that our data collector will be collecting new data that will be ingested in the data source (OpenData BCN) periodically, and it could be that one day these data present small modifications such as differences in the metadata or modifications in the number of columns, so it is important that if we want it to be an automatic process it has to be flexible to possible changes.

Also, another thing that is always good to keep in mind is that the available set of commands in MongoDB shell is much richer than of HBase, because it provides more query mechanisms and alternatives. This is important to consider if we know what queries we will need to make in the future.

One thing that works against us is that using MongoDB, we store the metadata for each record ingested. So when we store large amounts of data it is more optimal to do it with HBase, but in our case the level of data stored is very low.