UNIVERSITAT POLITÈCNICA DE CATALUNYA

FACULTAT D' INFORMÀTICA DE BARCELONA

Official Master in Data Science

Assignment 1

Big Data Management

Míriam Méndez Serrano
Víctor González Monge

Barcelona, 2022/2023 Q2

1. Introduction

In this project, we aim to develop a landing zone with two main components: the temporal zone and the persistent zone. To accomplish this, we will leverage two distributed systems that were covered in our lectures: HDFS and HBase. The project includes three data sources, two of which were provided and one of which we selected. To begin, we will upload the local data using Hadoop Distributed File Systems to the Temporary Landing Zone of our virtual machine. Then, we will use the HDFS located in the virtual machine (virtual box) to upload all the data into the Persistent Landing Zone, utilizing HBase. As part of our development process, we have created two scripts: temporal_zone.py, which enables the upload of local data into HDFS, and persistent_zone.py, which transfers HDFS data into HBase data.

2. Data sources

Two of the data sources for this project were provided on the learnsql2 page, which included a folder named "idealista" containing data in JSON format, another folder named "opendatabcn-income" containing CSV files, and a folder named "lookup_tables" containing the lookup tables for both datasets.

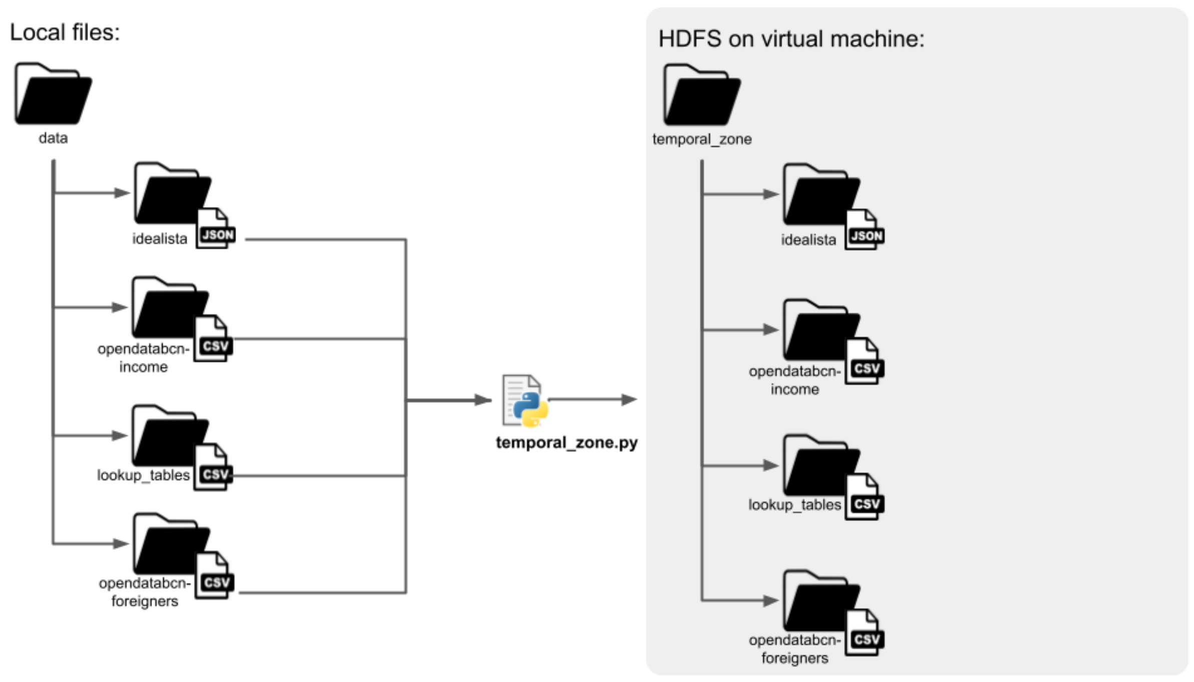
For the third data source, we choose to use the datasets that are available at the following link from 2017 to 2021: Register of inhabitants. Addresses of the city of Barcelona according to the nationality, Spanish or foreign, of the people who inhabit them. This dataset provides addresses of the city of Barcelona according to the nationality (Spanish or foreign) of the people who inhabit them. We obtained this data from the Ajuntament of Barcelona web. Each data file in this dataset was provided in CSV format.

3. Landing Zone

3.1. Temporal Zone

For the Temporal Landing Zone we decided to choose HDFS because it is capable of handling a large number of files, making it scalable for the future, it is resilient to failures so we can avoid losing data and can support a large number of different file formats. All this makes it an excellent choice for the data we have right now and in case there is a need to add more data sources in the future.

First we downloaded all the files from the different sources into our local machine. Then, using python and the module hdfs we created a folder in the HDFS where we can store our data, creating one folder for each of the data sources. After that we uploaded the local files to the HDFS, checking if the file already exists before uploading.



3.2. Persistent Zone

We wanted the Persistent Landing Zone to be able to ingest large datasets of different types, in case that in the future more data sources are added, and that it is capable of handling queries efficiently, so we decided to choose HBase. HBase has good scalability, capable of handling large amounts of data, and taking into account the high probability of having to increase the amount of data sources, this makes it the main reason to choose HBase.

Another important aspect of HBase is that it is a key-value native database. This means that HBase organizes and retrieves data based on the row key, which acts as a unique identifier for each row in the table. The value of each row is stored in column families, which contain one or more columns, and each column can have multiple versions.

The key-value native design of HBase is important for several reasons. First, it enables efficient storage and retrieval of large amounts of data, as the data is stored in a way that is optimized for high-speed lookups and scans. Second, it allows for flexible schema design, as the structure of the data can be easily adapted and modified by adding or removing columns, column families, or changing the row key design. This makes HBase well-suited for use cases where the schema of the data is not known in advance, or where the schema is likely to evolve over time.

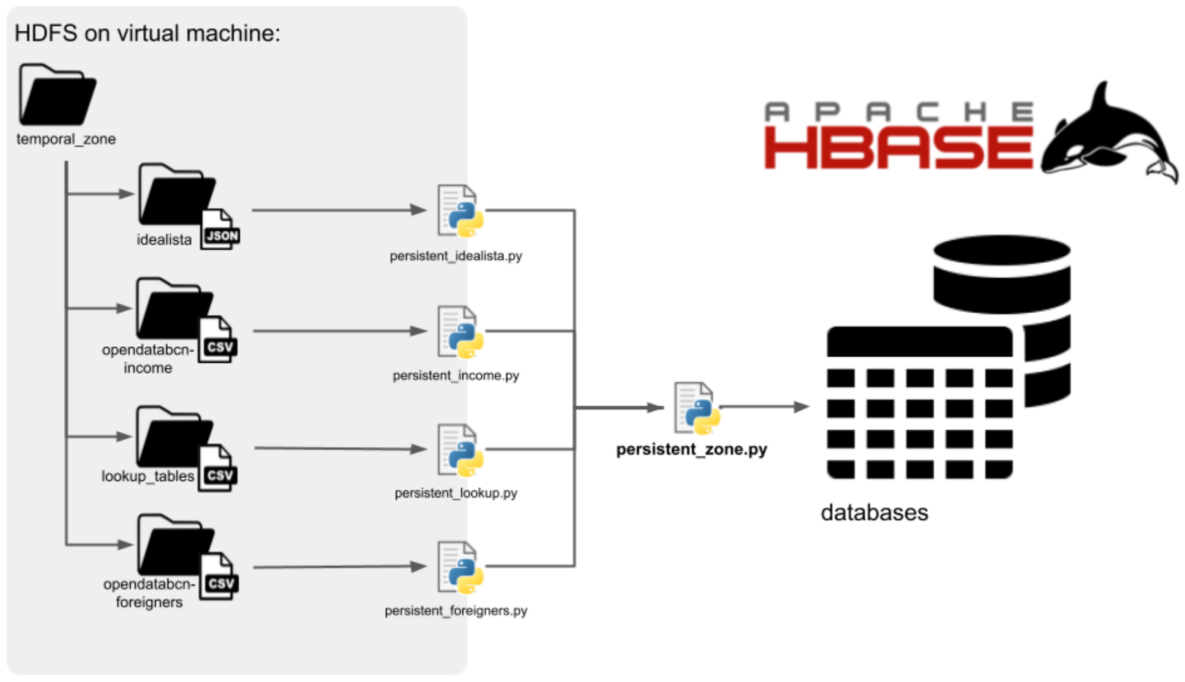
In addition, scans, for instance, can benefit from the B+ tree and the clustered index to only read data of interest and to avoid wasting resources on non-relevant data for the current query. That's why it is so important to decide what or how the row keys are going to be defined.

We assumed that the file name through the different datasources is different, so we decided to choose the **file name as a row-key** to simplify data retrieval and enable faster queries and **the content of the file as the value**. Due to the amount of data that the system right now deals with we decided to load all the data into one table, not affecting the query time by much. By this way we can make it easier to access and retrieve data associated with that source. This can be particularly useful in scenarios where you need to query data from a specific data source, or when you need to join data from different sources based on their identifier.

Compared to Parquet in HDFS, HBase provides a more flexible schema design and a key-value storage model, which can be more suitable for handling complex data structures and evolving data schemas. While Parquet is a columnar file format that can provide faster data access for analytical queries, it may not be as efficient for transactional workloads that require frequent updates and random access to data.

Compared to MongoDB, HBase provides better data consistency and integrity, thanks to its support for atomic operations and row-level locking. MongoDB lacks integrity constraints and may not provide the same level of transactional guarantees as HBase. Additionally, HBase's distributed architecture and ability to scale horizontally make it more suitable for handling large datasets with high levels of concurrent access.

The implementation of the persistent zone looks like the figure below. We implemented a data persistence Loaders per source, as it was asked in the statements, and then each of these was runned in persistent_zone.py. All the loaders, firstly are connected to HDFS and to the HBase databases table (if it isn't exist, will be created). Once all is successfully connected, HDFS sends all the information to HBase which stores the name of the file as the key and its content as the value.


HDFS on virtual machine:
temporal_zone
idealista
JSON
opendatabcn-income
CSV
lookup_tables
CSV
opendatabcn-foreigners
CSV
persistent_idealista.py
persistent_income.py
persistent_lookup.py
persistent_foreigners.py
persistent_zone.py
APACHE HBASE
databases