Marc Fortó Cornella
Íñigo Pikabea

April 7, 2024

BDM Report P1 - Landing Zone

1 Introduction

This project consists of the first part of a two-part project aimed at creating a Data Management Backbone. In this initial phase, the main objectives are:

- Identify data sources
- Implement data collectors
- Decide on the structure and deploy the temporal landing zone
- Implement the Data Persistence Loaders
- Decide on the structure and deploy the Persistent Landing Zone

We were asked to use three mandatory source datasets that were stored locally, plus one extra dataset of our choice that must require the implementation of a Data Collector (i.e., be external). For this extra dataset, we decided to use the OpenData BCN API and downloaded the list of accidents handled by the local police in the city of Barcelona (accidents-gu-bcn) in .csv format.
We decided to divide both data governance processes into two Python scripts ***data_collector.py*** and ***data_loader.py***, which can be executed separately depending on the argument added to the ***python3 main.py 'argument'*** command. All the code can be found in our **Github repository** [1]. The instructions for executing the code can also be found there.
The technologies selected for each part of the Landing Zone and the general structure of the first part of the Data Management Backbone are represented in the following diagram (see Figure 1).

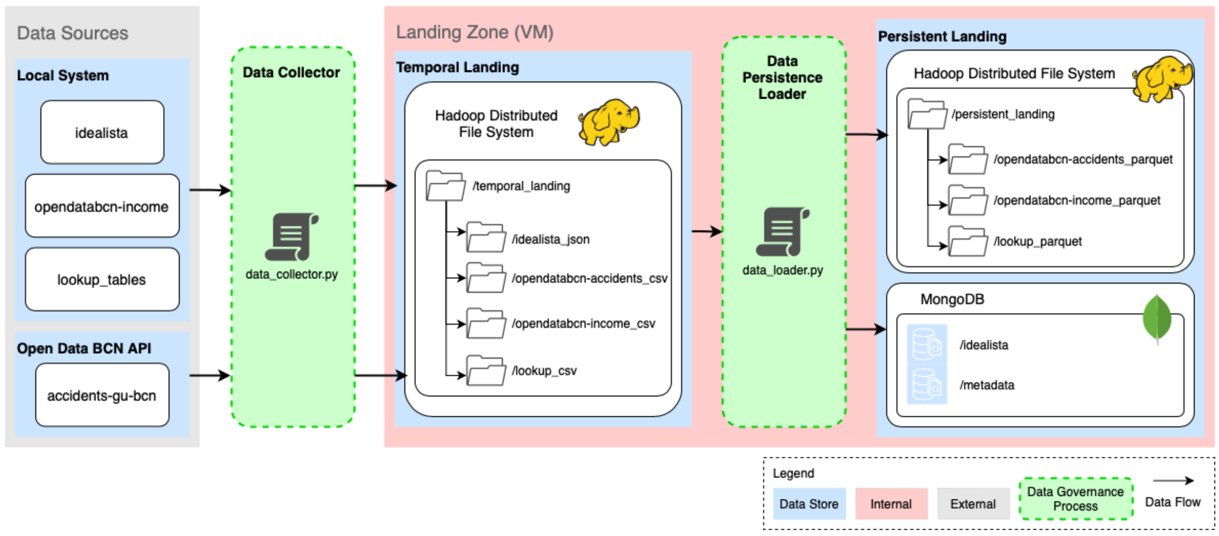

Figure 1: Landing Zone Diagram

In the following sections, we will justify our choice of modeling approach and technology and discuss the pros and cons of our decisions.

2 Temporal Landing Zone

The temporal landing 'stores temporarily the files' (though not necessarily deleted), until they are processed.

2.1 Architecture

For this part, we decided to load the source data into a Hadoop Distributed File System (HDFS). The main reasons that drove us to use HDFS and not another option are the following:

- **Distributed Storing:** HDFS supports all different types of files without any modifications. In contrast, technologies such as HBase or MongoDB both require some sort of data formatting. In HBase, data needs to be converted into key-value pairs, and with MongoDB, data needs to be converted into JSON files.
- **Scalability:** HDFS has efficient management of files and is optimized for very large files. In contrast, HBase requires more setup and configuration, and MongoDB may not be as efficient as it stores binary data within documents.
- **Tolerance to Failures:** HDFS provides fault tolerance by replicating data across multiple nodes, ensuring that data is not lost even in the event of node failures.
- **Append Data:** HDFS efficiently appends data to the end of files. If we are receiving daily new data, we could append this data to the same file if we divide our files by years.
- **Querying:** HDFS will always query in a complete scan fashion, as files are only queried to be sent to the persistent zone, thus reducing the need to optimize the read component of the storage.

Within the HDFS, the temporary landing zone can be found under */temporal_landing*, and inside this folder, we have one sub-folder for each data source. Data arrives at this directory in two ways. Figure 1 shows this design visually. On the one hand, we have data collected from the local system of our computer. On the other hand we have data collected from the Open Data BCN API. However, any API that fetches data can insert data in the HDFS in a similar way, as should be possible with any Big Data Management system.

2.2 Assumptions

We make the assumption that the data is large. The data is currently limited to a few megabytes, but in the case of big data, it can easily exceed several gigabytes. Big data is the reason for using HDFS, as the more data in the database, the more uniform the distribution will be.

The second assumption takes into account the way the data would be collected in an actual implementation. Now, the data is retrieved from the local system. In a more realistic scenario, the data would be gathered from the cloud (internet or database provided by the source) and loaded immediately into HDFS, without any local storage.

We've designed the temporary landing zone with the understanding that the data within it won't require frequent querying and therefore doesn't need to be distributed evenly across HDFS. While HDFS assigns locations randomly, the varying file sizes mean that some machines will end up with more data than others.

Given that this zone is designated as temporary and no modifications to the data are anticipated, files are ingested in their original formats.

3 Persistent Landing Zone

The Persistent Landing Zone ensures long-term data storage, providing a stable and secure environment where data is not just stored but also organized for optimized access and analysis.

3.1 Architecture

The Persistent Landing Zone is designed to efficiently store and manage data for long-term access and analysis. For structured data originating from CSV files, we utilize Apache Parquet format within the Hadoop Distributed File System, while for semi-structured or schema-less JSON data, we leverage MongoDB. This bifurcated approach allows us to optimize storage and querying capabilities based on the nature of each data type.

The decision to bifurcate the Persistent Landing Zone into two distinct parts, utilizing HDFS for Apache Parquet files and MongoDB for JSON documents, is strategically motivated by the principle of efficiency and optimization. This separation allows us to tailor our storage solutions to the intrinsic characteristics of the data, ensuring that each data type is stored in the most suitable environment for its nature. Below, I'll elaborate on this strategic choice:

Parquet and Hadoop for CSV:

- **Optimized Storage**: Parquet, a columnar storage format, significantly reduces the storage space required for our structured data. Its efficient compression and encoding schemes are particularly effective for the repetitive nature of many dataset fields, minimizing the storage footprint.

- **Enhanced Read Performance**: The columnar format of Parquet allows for faster retrieval of specific columns during analysis, which is advantageous for big data analytics operations that often only need a subset of the data, thereby reducing I/O operations.
- **Scalability**: Leveraging HDFS for storing Parquet files ensures that our data storage solution is inherently scalable. HDFS can handle vast amounts of data distributed across many nodes, ensuring data availability and fault tolerance.

MongoDB for JSON:

- **Flexibility**: MongoDB stores data in BSON format, which is similar to JSON. This makes it ideal for storing our semi-structured or schema-less data, as it can easily accommodate variations in data structure without the need for significant transformations.
- **Query Performance**: MongoDB provides powerful indexing capabilities that improve query performance. Indexes can be created on any attribute, making data retrieval operations faster and more efficient.
- **Scalability and Availability**: MongoDB's distributed architecture allows for data sharding and replication, enhancing data availability and allowing the database to scale horizontally as the dataset grows.

3.2 Assumptions

As previously discussed, we operate under the assumption that we will be handling a significant volume of data. Consequently, we anticipate the deployment of substantial distributed computing resources, specifically designed to handle such data volumes.

In terms of querying, our system is expected to perform specific queries targeting particular columns within our datasets, rather than retrieving entire records. This selective querying rationale significantly influenced our preference for the Apache Parquet format over alternatives like Avro. Parquet's columnar storage format is inherently more efficient for such use cases, as it allows for the reading of only the necessary columns, thereby reducing I/O operations and speeding up query execution.

References

[1] Github Repository. Landing_Zone_BDM. `https://github.com/inigopm/Landing_Zone_BDM`.