UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

# Big Data Management

## Project 1
## Landing Zone

Darryl Abraham

Riccardo Paciello

darryl.abraham@estudiantat.upc.edu
riccardo.paciello@estudiantat.upc.edu

April 7, 2024

# Contents

# 1 Introduction

DataOps is a methodology of data handling and processing that takes principles from DevOps, focusing more on the data itself and its ever changing nature. At the core of DataOps lies three main functional components, the data management backbone, the data analysis backbone, and data governance. The data management backbone is the first step in the data life cycle. The data management backbone has two main goals, firstly ingesting and storing the data which includes data integration (semantic and syntactic homogenisation of data) and cleaning data (removing duplicates, etc.), and secondly exposing a centralised repository of cleaned and profiled data ready for use by the data analysis backbone. To fulfill this goal, the data management backbone utilises the notion of zones, the landing zone, the formatted zone, and the exploitation zone, as shown diagrammatically in Fig 1. These zones describe at what stage the data is in the pipeline, automated by tools in between them, namely, the data collectors, data persistence loaders, and the data formatters. The first part of the Big Data Management project consists of developing the landing zone (highlighted in yellow in Fig 1), comprised of the data collectors which collect the data and place them in the temporal landing zone. After which the data is moved to the persistent landing zone by the data persistence loaders.

This report follows our development of the tools, the data collectors and the data persistence loaders, and the use of a distributed storage system, to mimic a big data pipeline for a chosen objective. To do so we adhere to big data principles and constraints set in the project definition. We first introduce each tool used, and at the end of the report discuss the advantages and disadvantages.

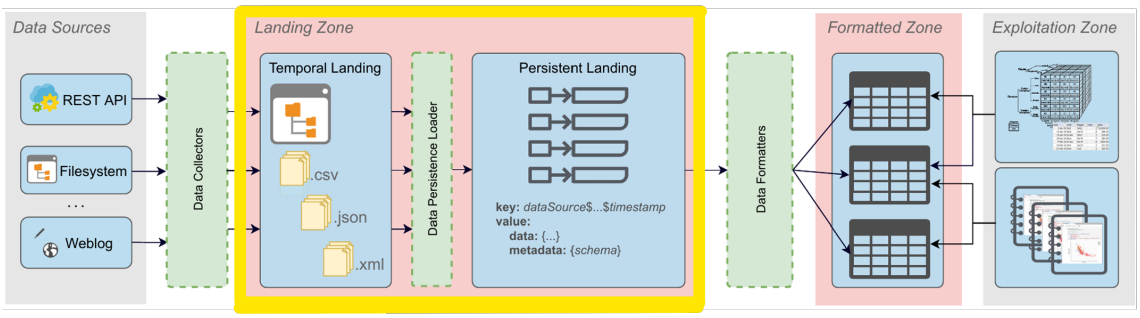


Figure 1: The data management backbone

# 2 Objective

The chosen objective of the data management backbone is to provide researchers with high quality data to perform analytic tasks about the relationship between rent and income across different neighbourhoods of Barcelona. Additionally, the third dataset chosen for our project is from OpenDataBCN, and provides demographic data about immigrant populations, such as their gender and age range. Using this data, in conjunction with the provided data we can analyse the effects and causes of changes in rental price and income in relation to immigrant population demographics in different neighbourhoods.

We define use of the data to be for analyses and modelling to better understand these factors.

# 3 The Landing Zone Pipeline

Prior to describing in detail the methods and tools implemented, we shall first holistically describe the data flow from and to each subzone. Our solution uses Hadoop Distributed File System (HDFS) as the data stores for each intermediate step, and Python code to create the data collectors and the data persistence loaders. The data collectors and data persistence loaders interact with HDFS through the 'hdfs' python library. See the diagram of the pipeline in Fig 3.

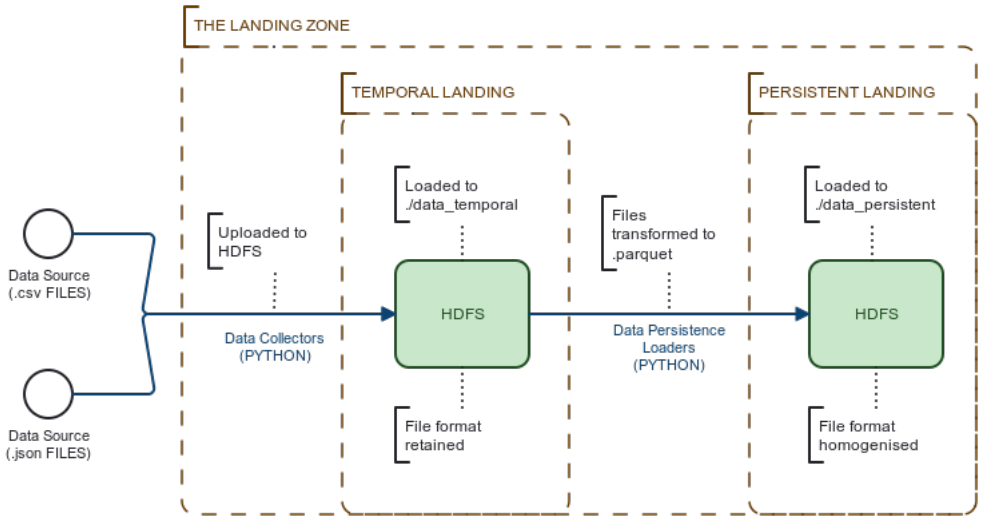


Figure 2: The landing zone

The pipeline follows the ingestion of data from different sources. Using a python script, the files are uploaded to HDFS as they are. The data collector can be called for two types of files, JSON and CSV. Once the files are located in HDFS, they are homogenised to .parquet in a separate "persistent" directory. In the following sections we shall describe in more detail each step in this process.

# 4 Hadoop Distributed File System

For the storage of our data we have chosen to use Hadoop Distributed File System (HDFS). HDFS is a core component of the Apache Hadoop ecosystem. It is the most commonly used open-source distributed file system today. The system operates on a master-slave architecture, consisting of a NameNode and DataNodes. The NameNode stores metada, orchestrating file operations and ensuring consistency, while DataNodes manage data blocks and participate in replication providing fault tolerance. HDFS stores large files by breaking them into manageable blocks (128 in our case), with 3 replicas distributed across DataNodes for parallel processing and efficient retrieval.

For this project we are provided with a Virtual Machine that runs an instance of HDFS. HDFS will be used as the storage system for both subzones, temporal and persistent landing. To interact with HDFS we use python, connecting to the NameNode using the 'hdfs' library. This connection with the InsecureClient function allows us to upload, delete, and manipulate data within the system. The code was developed locally, however is built in such a way that it can run on any machine, with the necessary scripts.

# 5 Parquet

Parquet is a columnar storage format designed for efficient storage and processing of large datasets within the Apache Hadoop ecosystem. Unlike traditional row-based storage formats, Parquet organizes data by columns, storing values from the same column together. This columnar layout enables better compression and encoding techniques. For the persistent zone, we homogenise the data files into Parquet. Our implementation not only provides the ability to transform JSON and CSV files into Parquet, but also offers 4 different compression methods, 'snappy', 'gzip', 'brotli', 'lz4', and 'zstd'. Each have their own advantages, such as snappy being the best balance of speed and size, while gzip for example focuses on size. Additionally, Parquet stores more metadata about columns such as minimum, maximum, and null values.

# 6 Data Collectors

The data collectors are implemented as a python class in the file temporal_landing.py. The class consists of two methods, upload_file and upload_folder. They are built to upload any type of file to HDFS, in a custom directory. The method to upload a folder can be called from the terminal (find all commands available below or in ReadMe.md file). To do so the class has access to the files through a given path and with the 'hdfs' library in python connects to HDFS using the InsecureClient function. Provided with the URL of the HDFS namenode, and the user the data collectors are able to connect and stream upload the files to a desired directory. Uploading of the given data (Idealista and OpenDataBCN-Income) and the chosen datasets (OpenDataBCN-Demographics) takes approximately 10 seconds.

# 7 Data Persistence Loaders

The data persistence loaders are also built in python, found in persistent_landing.py. The class consists of methods json_to_parquet_hdfs, csv_to_parquet_hdfs, a parent function to_parquet_hdfs that combines them depending on the file extension, and a grandfather method named persist that runs the methods below as needed when given a directory path in HDFS. The persist method will then create a copy of the given directory with the root directory name changed to root_persistent and transform all the files within it into Parquet, see Fig 3 for a simple diagram.

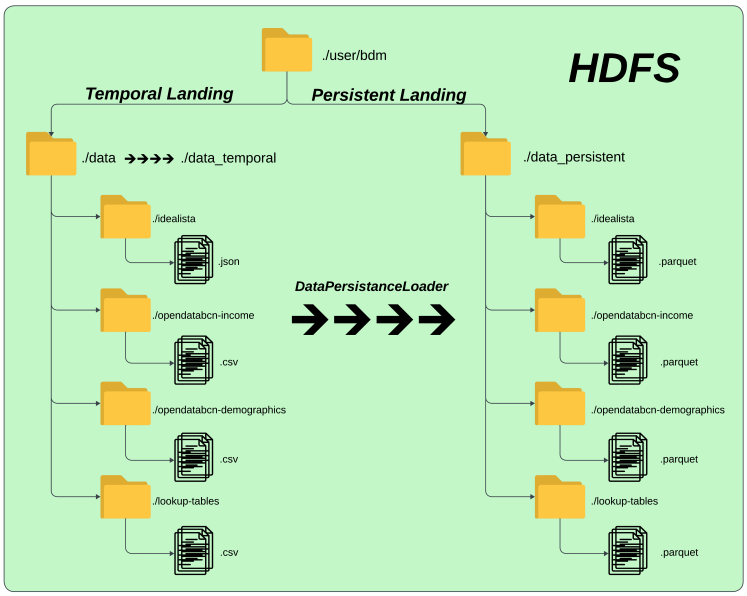


Figure 3: The data persistence loader

# 8 Landing Zone

To run the whole landing zone pipeline, the master script landing_zone.py is used. This script combines the methods and classes from temporal_landing.py and persistent_landing.py. From the terminal simply run any of these commands:

- Run the whole pipeline:
  - *python landing_zone.py url user execute local_dir hdfs_target_dir compression_type drop_temporal_dir*
- Run partially:
  - UPLOAD FOLDER: *python landing_zone.py url user upload local_dir hdfs_target_dir*
  - PERSIST FOLDER: *python landing_zone.py url user persist hdfs_target_dir compression_type drop_temporal_dir*
  - DELETE FOLDER: *python landing_zone.py url user drop hdfs_target_dir*

The arguments are:

- url: the URL of the HDFS namenode (http://10.4.41.45:9870)
- user: the user to connect to HDFS (bdm)
- local_dir: the local directory where the data is stored (./data)
- hdfs_target_dir: the HDFS directory where the data will be stored (temporal landing) (./data)
- compression_type: the Parquet compression type to use when persisting the data Compression types: 'n' (no compression), 'snappy' (optimal for speed and size), 'gzip' (size optimal), 'brotli', 'lz4', 'zstd'
- drop_temporal_dir: whether to drop the temporal landing directory after persisting the data Options: 't' (true), 'f' (false)

Please find more detailed information about running the pipeline on Github.

# 9 Discussion

We believe that our solution is sound, efficient, and robust to change. The use of HDFS allows us to guarantee scalability, fault tolerance, cost effectiveness, high-throughput data access, and can be integrated with Hadoop ecosystem. Additionally, through concurrency control a file can be accessed concurrently, handled by the NameNode. This is a huge advantage in big data management as many concurrent accesses are crucial to an efficient data management pipeline. The disadvantages associated with HDFS, such as its complexity, were avoided as an existing instance was used on a virtual machine with minimal set up. Another avoided disadvantage are security, not a part of the scope of the project, and small data handling, as we are mimicking a big data pipeline. We also believed that it was the right choice for both the temporal landing zone and the persistent landing zone. Using document or key value stores such as MongoDB or Hbase for the temporal landing zone would mean to persist the data already in the temporal landing. The decision to continue using HDFS for the persistent landing zone is due to the fact that its use only for temporal landing would not be efficient. If HDFS is used, it should be used for the entire landing zone, as uploading files temporarily to simply move them elsewhere for the persistent landing zone would be highly inefficient, as the temporal landing zone is just a temporary buffer for the data. Thus, our choice of HDFS for the temporal landing determined its use in the persistent landing, to maintain a resource efficient pipeline [1] [2].

The decision to choose Parquet came from our comparative analysis to Avro and SequenceFile. SequenceFile lacks the capabilities of the other two big data formats in schema evolution, limited support for complex data types, suboptimal performance for analytical queries, and poses integration challenges. The choice between Parquet and Avro is heavily dependent on the objective of the pipeline. Commonly, Parquet is better for analytical tasks as is our case due to its columnar storage which provides for better projections and querying using predicate pushdown. Although Parquet stores extra metadata, such as the column statistics, we believed in our objective to provide the highest quality data, these would provide great intermediate quality checks. This additional data stored is also eliminated due to the ability for Parquet to compress more efficiently than Avro, due to its columnar storage enabling better compression ratios. This can reduce costs in terms of storage and transfer efficiency compared to Avro. Furthermore, the seamless integration of Parquet with ApacheSpark also can later provide highly optimized execution of queries and transformations, necessary in the next steps of the data management backbone. Parquet is more highly integrated in the Hadoop ecosystem maintaining a flexible pipeline. Avro is better for streamed row wise operations such as data interchange between systems, however our objective determines that analyses and modelling would be conducted without using the whole dataset [3] [4] [5].

Our solution takes approximately 10 seconds to upload all the data, and 30 seconds to persist (no compression). Overall execution time: approximately 40 seconds. Hence, with the ultimate objective of providing the highest quality data in the most efficient manner for analytics and research tasks we believe that our solution is optimal.

Code found on Github at https://github.com/darryl-abraham/BDM_Project1/

# References

[1] A. Abelló, "Lecture: Hadoop distributed file system," 2024.

[2] R. Joshii, "What is hdfs? architecture, features, benefits, and examples," Sep 2023.

[3] A. Abelló and S. Nadal, "Big data management lecture notes," Feb 2024.

[4] Á. Alonso Isla *et al.*, "Hdfs file formats: Study and performance comparison," 2018.

[5] A. Prakash, "Parquet vs. avro: A detailed comparison of big data file formats," Aug 2023.