Data Lake Storage and Framework

Table of Contents

1. Overview

Any enterprise will have many operational systems which are critical for managing the day to day operations of business. When it comes to analysis the data and taking effective business decisions there are many challenges faced by enterprises, some of the challenges have been listed below

Data Complexity - ETL Mesh

- Fragmented data flow using traditional ETL tools requires more effort
- Same data being represented into multiple reports
- Report gets data from multiple data sources – Significant ETL effort

Scalability & Compute Limitation

- Legacy Enterprise Data Warehouses have severe capacity and compute scalability issues

Data Silos and limitation of current information system architecture

- Enterprise data is scattered
- Difficult to run predictive analytics to derive value hidden in data - Disparate systems
- No solution exists to process unstructured dataset or Real-time data feeds
- Self Service to data requires significant reoccurring ETL and modeling effort

In order to drive value and new services from data, enterprises have to change the way the data is handled. The Enterprise Data Lake / Enterprise Data Hub (EDH) using Big Data ecosystem is the future state solution for enterprises to ensure that the business users get the data in time to take effective business decisions and drive profitable growth.

2. DataLake/Hub

Data lake is a unified solution to store all data, for as long as desired or required, in its original fidelity; integrated with existing infrastructure and tools.

It provides flexibility to run a variety of enterprise workloads—including batch processing, interactive SQL, enterprise search, and advanced analytics.

It provides robust security, governance, data protection, and management that enterprises require. Leveraging the "Source-Once and Re-Use" approach, drives efficiency, reduces data silos reduces latency and time to value, massively improves analytics and discovery, greatly reduces costs

This document will provide an overview of data architecture approach specifically for Big Data and will also provide details related to various data operations, patterns, solutions, tools/techniques and best practices to ensure a standard way of implementing the big data solutions.

Big Data is not just about velocity, volume, veracity and variety. It is about how you identify the right information from data that is growing exponentially, and use it to add business value. Hadoop provides a low cost, but dependable solution to tackle data management problems. However, don't expect great results by just getting your Big Data project underway. It requires a focused, analytical, use-case driven approach that organizations need to be seriously committed to. That's when you can transform your big data into what it is meant to give you – smart information that directly translates into efficiencies, returns and growth!

To achieve business efficiencies, returns and growth, data architecture in is a critical piece, as it sets data standards for all its data systems as a vision or a model of the eventual interactions between those data systems. Data Lake describes how data is processed, stored, and utilized in an information system. It provides criteria for data processing operations so as to make it possible to design data flows and also control the flow of data in the system.

3. Data Architecture

A well-defined Data Architecture provides the ability to meet data volume, latency, quality, volatility, variety, auditability, traceability, security and integration requirements. It serves as a blueprint and guide for current and future data projects.

In particular, data architecture describes:

- How data is persistently stored?
- How components and processes reference and manipulate this data?
- How external/legacy systems access the data?
- Interfaces to data managed by external/legacy systems
- Implementation of common data operation

Consistent, reliable, scalable and reusable. These are all hallmarks of a data architecture approach that supports a growing enterprise. Having a solid data architecture plan in place will allow you to leverage and realize the full value of your data. In the following sections we will explore the way to build a consistent, reliable, scalable and reusable Enterprise Data Lake/Hub.

4. Layered Data Architecture

In any data lake/hub, a data life-cycle aims at converting raw data to a more transformed/aggregated or de-normalized structure. To achieve so, it involves complex processing steps, security, user access/authentication/authorization, fail-over recovery and data load to variety of downstream systems. To follow the layer architecture below are the following considerations.

4.1 Considerations

- File system
- Logical structure

- Physical structure
- File formats
- Compression

4.2 File System

- HDFS/GCS/S3/ADLS

4.3 Logical Structure

Logically we split the data within our platform into different layers:

4.3.1 Raw

a. Contains raw data “as is” without any business logic.

b. The only permissible actions are:

   1. Place files in correct directory
   2. Ensure file name and SHA 256 checksum uniqueness.
   3. Files may be compressed and multiple small files may be placed inside a container directory.

   1 saurav 200

4.3.2 Cleansed

1. Raw files are broken down into records and fields. No type checking will take place. Everything will be kept as STRING data type except complex data type.
2. Some check needs to be performed like dedup and primary key check.
3. Semi structured data like XML,JSON and TSV,CSV will be converted into Avro file format with right compression codec.
4. UUID and timestamp are added to each record for traceability.
5. Different ingestion mechanisms (e.g. Sqoop, Airflow, Informatica, Gateway) bypass the Raw layer, so it is critical to treat the Cleansed layer as raw, immutable data.
6. **History is maintained in this layer.’**
7. **Cleaning of Data was done - Not null check, File name checks, Business checks (e.g. Country code cannot 123), Null value replacement**

4.3.3 Published (Not created by us, it is created by Consumers)

1. Data is stored in the form required for specific use cases.
2. There may be intermediate layers within the published area, e.g. for unions and joining.
3. There is no single enterprise wide data model. All objects within this layer are use case specific (but follow agreed business definitions and information architecture standards).

4.3.4 LOB/Canonical

1. Data is stored in this layer as per different business matrix dependent on LOB.e.g Finance, Ops, etc etc

2. Data is stored as columnar data storage in parquet file format.

3. This layer is specifically for data presentation in reporting.

4.3.5 Star schema / Modelled- Created on Data Warehouse-e.g Big Query (Optional - needed when u want to create a dashboard)

1. Data is stored in the form required for specific reporting use cases.
2. There may be intermediate layers within the published area, e.g. for unions and joining.

4.3.6 Rejected/ Error

1. Errors may occur when data is being transformed (i.e. moved between layers), e.g. when data does not conform to the specified schema in the Cleansed or Modelled layers. Such data will be stored in a dedicated area.
2. It stores failed records while performing checks between RAW to cleansed.

   LOB – Line of business e.g. Finance, HR, Commercial

4.4 Directory Structure

1. HDFS physical layout is shown below, which reflects the logical separation into Raw, Cleansed and Published layers.
2. <subject> and <interface version> subdirectories are optional in all layers.

Layer	HDFS Path	Notes	Partition Scheme
Raw History	/<env>/raw/<application>/history/<table_name>/<partition_scheme>/<files>	Raw data in full snapshot format	<YYYY-M
Raw	/<env>/raw/<application>/data/<table_name>/<partition_scheme>/<files>	Raw incremental data	Based loading scheme.
Raw Real-Time	/<env>/raw/<application>/rt_data/<table_name>/<partition_scheme>/<files>	Raw data for real time.	<YYYY-M HH:MI>
Cleansed	/<env>/cleansed/<application>/data/<table_name>/<partition_scheme>/<files>	cleansed data	<YYYY-M
Published	/<env>/published/<LOB>/<application>/data/<subject>/<partition_scheme>/<files>	Published data (batch views)	
Published-Archived	/<env>/published/<LOB>/<application>-archive/data/<subject>/<partition_scheme>/<files>	Archived batch views	
Rejected	/<env>/raw/<application>/reject/<table_name>/<partition_scheme>/<files>	Data which cannot be transformed from Raw to Cleansed layer	<YYYY-M
Rejected	/<env>/cleansed/<application>/reject/<table_name>/<partition_scheme>/<files>	Data which cannot be transformed from cleansed to Published layer	<YYYY-M

3. <partition_scheme> is based on tools we are using while doing ETL. In case of informatica or SQL transformation, we must use partition by a single string <yyyy-mm-dd>. But in case of spark or NoSQL transformations, we should go with a multi-depth partition with The Year, Months, and Days all as their own e.g. /<yyyy>/<mm>/<dd>

4.5 Partition Scheme

A sample use case / table has been created on data lake to test query performance on different partitioning schemes. Below are the findings.

Table Structure:

tripId: Incremental - 1,2,3,4,5…

date: Date field is dynamic starting form date 2015-01-01 to date 2018-12-31

tripId	source	destination	Date
1	Bangalore	Denmark	2018-05-04

Data Preparation:

	Single String	Nested Partitions
Insert Time (Hive Map Reduce) 1,00,000 Rows; 3.5 MB	960 Sec (16 Min)	1015 sec (17 Min)
Insert Time (Hive Map Reduce) 10,00,000 Rows; 35 MB	1488 Sec (25 Min)	2344sec (39 Min)
No of Partition	860	860

Data Query Analysis: Data query analysis on Single string partition

Single String (yyyy-mm-dd)	Query Time in Seconds (1,00,000 Rows)	Query Time in Seconds (10,00,000 Rows)
SELECT count(*) FROM mblr.trips_1 WHERE date_part IN ('2015-01-01', '2015-02-03', '2016-01-01');	19	19
SELECT count(*) FROM mblr.trips_1 WHERE date_part IN ('2016-05-10');	-	15
SELECT count(*),date_part FROM mblr.trips_1 WHERE date_part IN ('2015-01-01', '2015-02-03', '2016-01-01') group by date_part;	21	17
SELECT count(*) FROM mblr.trips_1 WHERE date_part LIKE '2016%';	9	6
SELECT count(*) FROM mblr.trips_1 WHERE date_part LIKE '2016-06%';	4	3

Data Query Analysis: Data query analysis on Nested partition

Nested Partitions (yyyy/mm/dd)	Query Time in Seconds (1,00,000 Rows)	Query Time in Seconds (10,00,000 Rows)
SELECT count(*) FROM mblr.trips_2 WHERE (YEAR=2015 AND MONTH=01 AND DAY=01) OR (YEAR=2015 AND MONTH=02 AND DAY=03) OR (YEAR=2016 AND MONTH=01 AND DAY=01);	2	2
SELECT count(*) FROM mblr.trips_2 WHERE (YEAR=2016 AND MONTH=05 AND DAY=10);	-	1
SELECT count(*),YEAR,MONTH,DAY FROM mblr.trips_2 WHERE (YEAR=2015 AND MONTH=01 AND DAY=01) OR (YEAR=2015 AND MONTH=02 AND DAY=03) OR (YEAR=2016 AND MONTH=01 AND DAY=01) group by YEAR,MONTH,DAY;	22	16
SELECT count(*) FROM mblr.trips_2 WHERE YEAR=2016;	6	1
SELECT count(*) FROM mblr.trips_2 WHERE YEAR=2015 and MONTH=06;	2	1

4.6 File Formats

1. HDFS does not enforce any particular file format or schema when storing data.
2. HDFS has known limitations in dealing with very large number of small files. Hence “container files” to store these together within a single file (without losing fidelity). So a different approach might be required for “large” vs “small” files.
3. Ability to split a file as input to multiple tasks is very important.
4. Choices for consideration:
    a. Raw (files are stored “as is”)
    b. File System based data structures (containers), e.g. SequenceFiles
    c. Serialization format, Avro will be used.
    d. Columnar format, Parquet will be used.

4.7 Compression

1. Compression has two important benefits:
    a. Reduce disk and network I/O (at the expense of CPU) – important factor in a distributed system

    b. Reduce the amount of storage required (at the expense of CPU and some extra latency)

2. There are a number of compression algorithm choices.
3. Ability to split a file as input to multiple tasks is very important.
4. Speed vs size tradeoffs.
    a. Snappy, LZO/LZOP, Gzip, bzip2
    b. Google Snappy has wide industry support and offers good trade off between speed and compression factor. Unless used with a container format (e.g. SequenceFiles or Avro) it is *not* splittable. For compressing large raw files, LZOP can be considered; but there will be cases where compression does not make sense, e.g. files such as video which are already compressed.

4.8 File Formats and Compression

Layer	**File Format and Compression**	**Notes**
Raw (small files)	Raw+ Snappy (block compressed)	Small vs large to be defined
Raw (large files)	Raw + No compression Raw + Snappy	Some files (e.g. video) may already be compressed
Cleansed	Avro + Snappy	No type checking, Every fields will be of STRING data type
Published	Parquet	Depends on the exact use case
Star Modelled	Variety of file formats, e.g. Avro, Parquet	Depends on the exact use case
Rejected	SequenceFiles + Snappy (block compressed) Raw	Depends on the format of the incoming rejected data

4.9 Data Flow between layers

4.10 Schema

Everything should be schema driven starting from data ingestion to data transformation. We should keep our avsc schema for each layer in below folder structure.

Layer	HDFS Path	Notes
Raw	/conf/avro/<LOB>/<SOURCE_NAME>/raw/incremental/1.0/2015-03-01/shipping.avsc	Raw incremental data schema
Cleansed	/conf/avro/<LOB>/<SOURCE_NAME>/cleansed/full/2015-03-01/ shipping.avsc	Full structured snapshot of source data schema in case of full load.
Cleansed	/conf/avro/<LOB>//<SOURCE_NAME>/cleansed/incremental/1.0/2015-03-01/shipping.avsc	Cleansed incremental data
Published	/conf/avro/<LOB>/<SOURCE_NAME>/published/processing/2015-03-01/shipping.avsc	Intermediate layer for joining and processing
Published	/conf/avro/<LOB>/<SOURCE_NAME>/published/archive/public/2015-02-28/shipping.avsc	Archived version of final output of fact data
Rejected	/conf/avro/<LOB>/<SOURCE_NAME>/rejected/raw/2015-03-01/shipping.avsc	Raw data which could not be transformed from Raw to Cleansed layer
Rejected	/conf/avro/<LOB>/<SOURCE_NAME>/rejected/cleansed/2015-03-01/shipping.avsc	cleansed data which could not be transformed from Cleansed to Published layer

4.11 Retention Policy

Layer	Retention Period	Notes
Raw	Truncate/ Load	
Cleansed	Forever	
Published		Data will be moved from published to published archive after X days, X -> decided by business

Published Archive	Forever	
Rejected		

5. Error Handling

1. Ingestion Jobs

    a. Avro schema is applied at the time of loading data. In case of schema mismatch, the entire job will fail and will need to be re-run after schema error is corrected.

    b. Job will fail if database credentials are not valid and the job will need to be re-run.

2. Raw to Cleansed

    a. Custom parsers will be written to apply Avro schemas on top of the Raw data.(Not needed)

    b. Error handler will move any records which do not conform to the schema to the right folder in the Rejected area.

    c. We may use existing converter to convert data e.g informatica BDM transformation.

3. Cleansed to Canonical

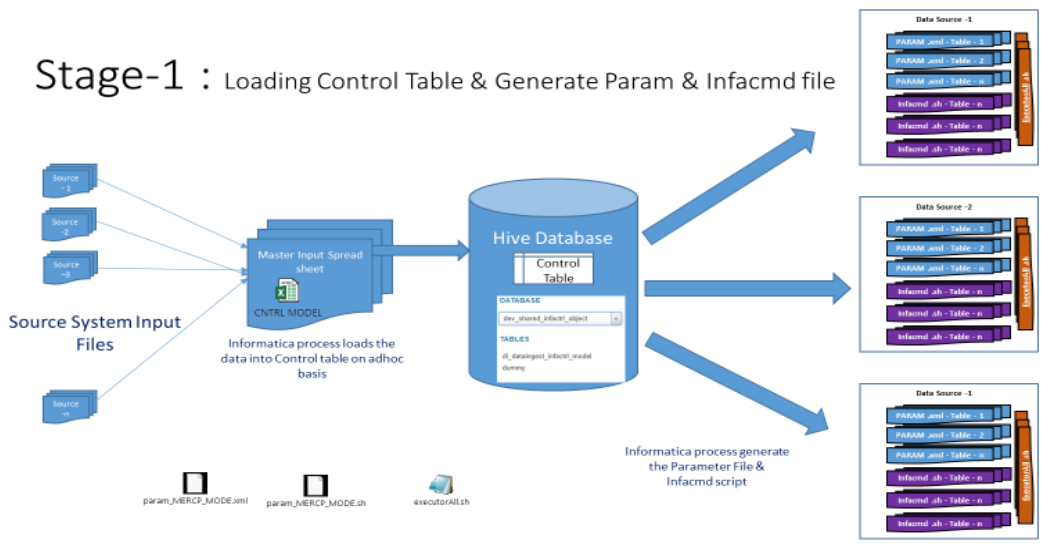
    a. Custom parsers will be written to apply Parquet schemas on top of the Cleansed data.

    b. Error handler will move any records which do not conform to the schema to the right folder in the Rejected area.

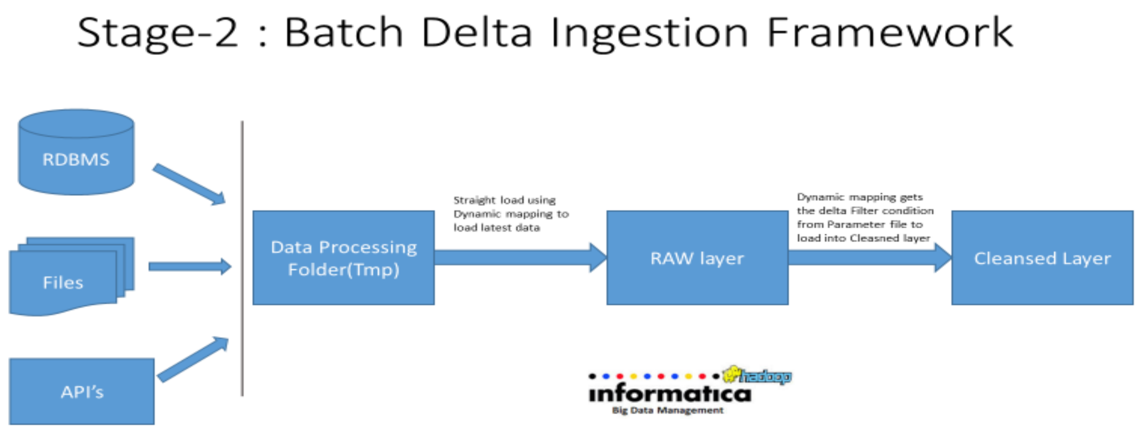
6. Data Ingestion Mechanism (Current architecture)

There are multiple different tools used for data retrieval from source system. Informatica, OGG, gateway are some of them. These tool store data till tmp/RAW. After that informatica ingest data till cleanse layer. Ingestion can be segregated into 2 stages :

a) **Data Preparation :** In this stage, metadata/control data needs to inserted into control tables. CSV fle is being used to store control data. This file needs to be inserted using informatica to respective tables.



b) Data Ingestion :


Stage-2 : Batch Delta Ingestion Framework
RDBMS
Files
API's
Data Processing Folder(Tmp)
Straight load using Dynamic mapping to load latest data
RAW layer
Dynamic mapping gets the delta Filter condition from Parameter file to load into Cleasned layer
Cleansed Layer
hadoop
informatica
Big Data Management

References :

https://community.hortonworks.com/questions/29031/best-pratices-for-hive-partitioning-especially-by.html