HCSC Data Accelerator program

Membership Technical Design Document

Version 0.05

Last Modified on Aug 18, 2018

Document History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Author** | **Version** | **Change Reference** |
| June 6, 2018 | Infosys Team | 0.01 | Shared with HCSC, specifically for section 6.3 TMG Design  Updated Monthly frequency for HPS and ‘On’ & ‘Off’ exchange |
| June 14, 2018 | Infosys Team | 0.02 | Shared with HCSC, specifically for section 3.2 Ingestion patterns and 8.0 Environment and Standards |
| June 28, 2018 | Infosys Team | 0.03 | After incorporating multiple review comments from HCSC, the updated document shared with HCSC. The sections updated are as follows –  Section 2.1 In scope  Section 2.2 Out of Scope  Section 3.4 key Design Decision |
| July 19, 2018 | Infosys Team | 0.04 | Section  Picking latest records  History table in merge  Updates to Recon process  History loads |
| July 26, 2018 | Infosys Team | 0.04 | REH – Point in time history  DBs & Tables for data sources |
| Aug 13, 2018 | Infosys Team | 0.4 | Included Point in time History for HPS and TMG  Included Manufactured History details for HPS and TMG |
| Aug 18, 2018 | Infosys Team | 0.05 | Font, Style, Look & feel changes |

Review and Sign-off

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Reviewer** | **Status** | **Comments** |
|  | HCSC Team | Under review |  |

Table of Contents

[1. Document Information 8](#_Toc522530562)

[1.1 Purpose of Document 8](#_Toc522530563)

[1.2 References 8](#_Toc522530564)

[1.3 Acronyms 9](#_Toc522530565)

[2. Scope & Assumptions 9](#_Toc522530566)

[2.1 In Scope 9](#_Toc522530567)

[2.2 Out of Scope 9](#_Toc522530568)

[3. Technical Approach 10](#_Toc522530569)

[3.1 Logical Architecture 10](#_Toc522530570)

[3.2 Source file formats Ingestion 10](#_Toc522530571)

[3.3 High level Design 11](#_Toc522530572)

[3.3.1 Bluestar 12](#_Toc522530573)

[3.3.2 HPS & RMS 13](#_Toc522530574)

[3.4 Key Design Decision 13](#_Toc522530575)

[3.4.1 Logical model for the gold layer 13](#_Toc522530576)

[3.4.2 Physical model for the gold layer 14](#_Toc522530577)

[4. Data Ingestion patterns 14](#_Toc522530578)

[4.1 Batch loads 14](#_Toc522530579)

[4.2 NRT loads 15](#_Toc522530580)

[5. Data Processing 15](#_Toc522530581)

[5.1 RAW Layer 16](#_Toc522530582)

[5.2 Merge Layer 16](#_Toc522530583)

[5.3 Gold Layer (CEI layer) 16](#_Toc522530584)

[5.4 Audit and logging 17](#_Toc522530585)

[5.4.1 Audit 17](#_Toc522530586)

[5.4.2 Logging 21](#_Toc522530587)

[5.5 End to end scheduling 21](#_Toc522530588)

[6. Detailed Design 21](#_Toc522530589)

[6.1 Bluestar 21](#_Toc522530590)

[6.2 HPS 22](#_Toc522530591)

[6.2.1 Tech Stack 22](#_Toc522530592)

[6.2.1.1 Hive 22](#_Toc522530593)

[6.2.1.2 Spark 23](#_Toc522530594)

[6.2.1.3 Kafka 23](#_Toc522530595)

[6.2.2 Process Overview 23](#_Toc522530596)

[6.2.2.1 Ingestion 23](#_Toc522530597)

[6.2.2.2 Merging 26](#_Toc522530598)

[6.2.2.3 Gold 26](#_Toc522530599)

[6.2.3 Format Standardization 28](#_Toc522530600)

[6.2.4 Full Loads 28](#_Toc522530601)

[6.2.5 Incremental Loads 28](#_Toc522530602)

[6.2.6 Low Level Steps 28](#_Toc522530603)

[6.2.7 HPS Gold PIT 29](#_Toc522530604)

[6.2.8 HPS Manufactured History 30](#_Toc522530605)

[6.3 RMS 33](#_Toc522530606)

[6.3.1 Tech Stack 33](#_Toc522530607)

[6.3.1.2 Hive 33](#_Toc522530608)

[6.3.1.2 Spark 33](#_Toc522530609)

[6.3.1.3 Kafka 34](#_Toc522530610)

[6.3.2 Process Overview 34](#_Toc522530611)

[6.3.2.1 Ingestion 34](#_Toc522530612)

[6.3.2.2 Merging 37](#_Toc522530613)

[6.3.2.3 Gold 38](#_Toc522530614)

[6.3.3 Format Standardization 39](#_Toc522530615)

[6.3.4 Full Loads (Day 0 load) 39](#_Toc522530616)

[6.3.5 Gold PIT 40](#_Toc522530617)

[6.3.6 Manufactured History 43](#_Toc522530618)

[6.3.7 Incremental Loads 45](#_Toc522530619)

[6.3.8 Target Table Updates 46](#_Toc522530620)

[6.3.9 Low level steps 46](#_Toc522530621)

[6.3.10 Business Scenarios 46](#_Toc522530622)

[6.3.11 Table Structure 48](#_Toc522530623)

[6.4 TMG 52](#_Toc522530624)

[6.4.1 Tech Stack 53](#_Toc522530625)

[6.4.1.2 Hive 53](#_Toc522530626)

[6.4.1.2 Spark 53](#_Toc522530627)

[6.4.2 Process Overview 53](#_Toc522530628)

[6.4.2.1 Ingestion 53](#_Toc522530629)

[6.4.2.2 Merging 56](#_Toc522530630)

[6.4.2.3 Reconciliation Process 58](#_Toc522530631)

[6.4.2.4 Gold 59](#_Toc522530632)

[6.4.2.5 TMG Gold PIT 60](#_Toc522530633)

[6.4.2.6 TMG Manufactured History 62](#_Toc522530634)

[6.4.2.7 Day 0 load 66](#_Toc522530635)

[6.4.3 Format Standardization 67](#_Toc522530636)

[6.4.4 Full Loads 67](#_Toc522530637)

[6.4.5 Incremental Loads 67](#_Toc522530638)

[6.4.6 Low level Steps 67](#_Toc522530639)

[6.5 VANTAGE 1.0 70](#_Toc522530640)

[6.5.1 Tech Stack 71](#_Toc522530641)

[6.5.1.1 Hive 71](#_Toc522530642)

[6.5.1.2 Spark 71](#_Toc522530643)

[6.5.2 Process Overview 72](#_Toc522530644)

[6.5.2.1 Ingestion 72](#_Toc522530645)

[6.5.2.2 Merging 74](#_Toc522530646)

[6.5.2.3 Gold 74](#_Toc522530647)

[6.5.3 Format Standardization 75](#_Toc522530648)

[6.5.4 Full Loads 75](#_Toc522530649)

[6.5.5 Incremental Loads 75](#_Toc522530650)

[6.6 VANTAGE 2.0 75](#_Toc522530651)

[6.6.1 Tech Stack 75](#_Toc522530652)

[6.6.1.1 Hive 75](#_Toc522530653)

[6.6.1.2 Spark 75](#_Toc522530654)

[6.6.2 Process Overview 76](#_Toc522530655)

[6.6.2.1 Ingestion 76](#_Toc522530656)

[6.6.2.2 Merging 78](#_Toc522530657)

[6.6.2.3 Gold 79](#_Toc522530658)

[6.6.3 Format Standardization 79](#_Toc522530659)

[6.6.4 Full Loads 79](#_Toc522530660)

[6.6.5 Incremental Loads 79](#_Toc522530661)

[6.7 System80 80](#_Toc522530662)

[6.8 GOLD to REH 80](#_Toc522530663)

[7. Subject Areas 80](#_Toc522530664)

[7.1 Membership Data Model and Standards 81](#_Toc522530665)

[7.1.1 Membership Data Group Classification 81](#_Toc522530666)

[7.1.2 Source Data Application System (Files) And Contents 81](#_Toc522530667)

[7.2 File Types 82](#_Toc522530668)

[8.Enterprise Consumption Views (ECV) 83](#_Toc522530669)

[9. Environment and Standards 84](#_Toc522530670)

[9.1 Incoming Files/Sources 84](#_Toc522530671)

[9.2 Databases/tables 84](#_Toc522530672)

[9.3 Hadoop 85](#_Toc522530673)

[9.4 Yarn Queues 85](#_Toc522530674)

[9.5 Webroles/NPID 85](#_Toc522530675)

[9.6 HDFS Paths 86](#_Toc522530676)

[9.7 App Edge Node Script Paths 87](#_Toc522530677)

[9.8 Zena 87](#_Toc522530678)

[10. Document Approval 89](#_Toc522530679)

[11. Appendix 90](#_Toc522530680)

# Document Information

## Purpose of Document

This design document would describe different data sources in the current scope, the data ingestion, data processing, business rules required for the business transformations and data quality. The document would describe different data frameworks and modules which will be used for or data ingestion and data processing.

The amount of details captured may vary but suffice for the current scope of the project. The document would be updated with next level of technical and environment specific details as the established patterns are extended during project implementation.

## References

|  |  |  |
| --- | --- | --- |
| **Reference Framework Standards** | | |
| **Document Name** | **Description** | **Location/file (link if possible)** |
| Data Lake Gold Detailed Technical Design v5DWG | Design document from HCSC |  |
|  |  |  |

## Acronyms

|  |  |  |
| --- | --- | --- |
| **Acronyms** | | |
| **Name** | **Meaning** | **Comments** |
|  |  |  |
|  |  |  |

# 2. Scope & Assumptions

## 2.1 In Scope

* Ingestion of membership data for the following sources
  + Bluestar
  + HPS
  + TMG
  + RMS
  + Vantage 1.0
  + Vantage 2.0
  + System80
  + MIP
* Bluestar – Building data pipeline from RAW layer to Gold layer
* RMS – XML data ingestion starting from IBM MQ source for RMS
* TMG – Data from HDFS raw layer
* Data ingestion pipeline building for data sources as included in the design document

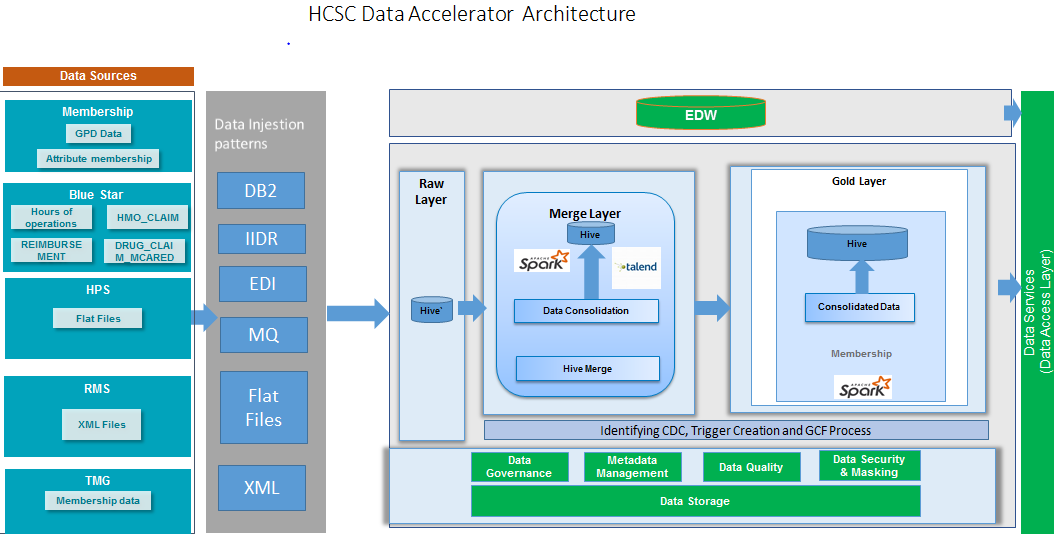
## 2.2 Out of Scope

* Gold Consumption (data delivery) – enterprise consumption views
* Any type of cleansing, verifying, updating and correcting of source data on its loading into any hops including but not limited to Raw, Merge, Gold
* Application of any Standardization Rules in any of data hops (Raw, Merge or Gold layer )

# 3. Technical Approach

## 3.1 Logical Architecture

The below figure describes the enterprise architecture view for membership Gold Data Accelerator project. The components and frameworks displayed in the diagram are explained in detail in later sections.



## 3.2 Source file formats Ingestion

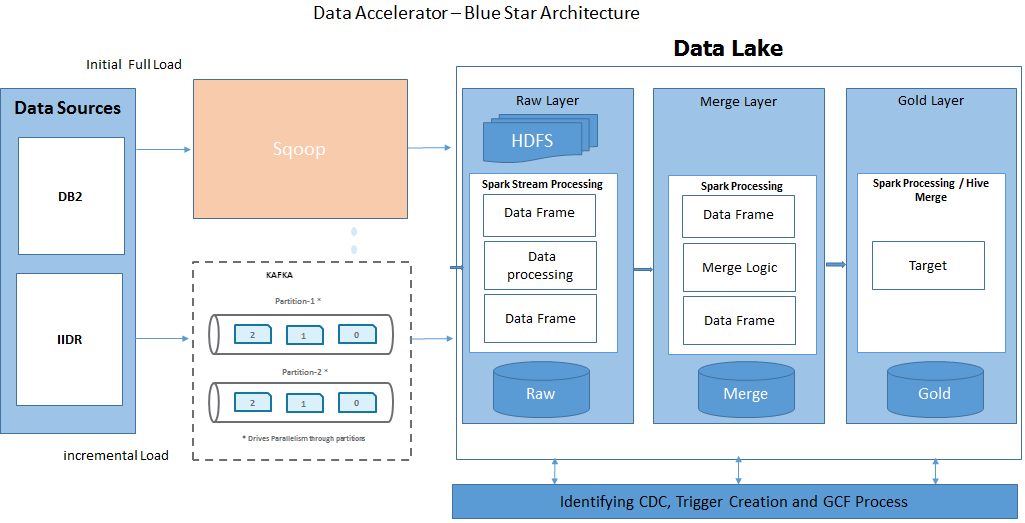
| **Source System** | **Files / Database feed** | **Technology** | **Raw layer** | **Merge layer** | **Gold Layer** | **Comments** |
| --- | --- | --- | --- | --- | --- | --- |
| Bluestar | DB2 database | Sqoop / Hive | Hive ORC , External | Hive ORC , hive merge, Internal | Hive ORC , Hive merge, Internal | Transactional property=true for target table |
| Bluestar | IIDR | Kafka / Spark | Hive ORC format | Hive ORC , hive merge | Hive ORC , Hive merge | Transactional property=true for target table |
| HPS | Flat files (With Header) | Hive / Spark / Kafka | Hive ORC format | Hive ORC , hive merge | Hive ORC , Hive merge | Transactional property=true for target table |
| HPS | Flat files (Without Header) | Hive / Spark / Kafka | Hive ORC format | Hive ORC , hive merge | Hive ORC , Hive merge | Transactional property=true for target table |
| RMS | IBM MQ (XML) | Dstreams / HDFS storage | Hive ORC format | Hive ORC , hive merge | Hive ORC , Hive merge | Transactional property=true for target table |
| TMG | DAF Files | Hive / Spark / Kafka | Hive ORC format | Hive ORC , hive merge | Hive ORC , Hive merge | Transactional property=true for target table |
| TMG | POA Files | Hive / Spark / Kafka | Hive ORC format | Hive ORC , hive merge | Hive ORC , Hive merge | Transactional property=true for target table |
| TMG | MT Medsupp files | Hive / Spark / Kafka | Hive ORC format | Hive ORC , hive merge | Hive ORC , Hive merge | Transactional property=true for target table |
| TMG | HCSC Supplemental Files | Hive / Spark | JSON ( File converted to JSON from Multi-record file ) | Hive ORC , hive merge | Hive ORC , Hive merge | Transactional property=true for target table |
| VMS 1.0 | EBCDIC to ASCII File | Hive / Spark / Kafka | Hive ORC format | Hive ORC , hive merge | Hive ORC , Hive merge | Transactional property=true for target table |
| VMS 2.0 | Flat File with pipe delimited | Hive / Spark / Kafka | Hive ORC format | Hive ORC , hive merge | Hive ORC , Hive merge | Transactional property=true for target table |
| System80 |  |  |  |  |  |  |

## 

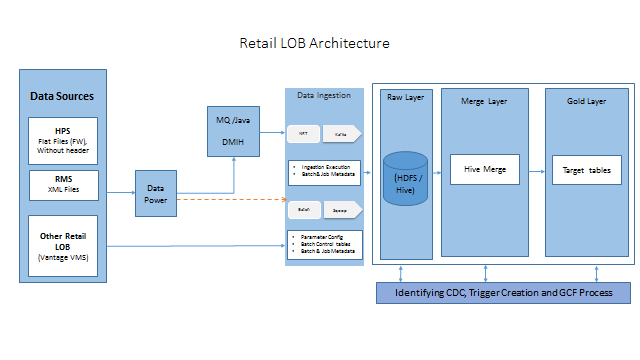
## 3.3 High level Design

The below figure displays a High level design for the various data sources in the data accelerator project.

### 3.3.1 Bluestar



### 3.3.2 HPS & RMS



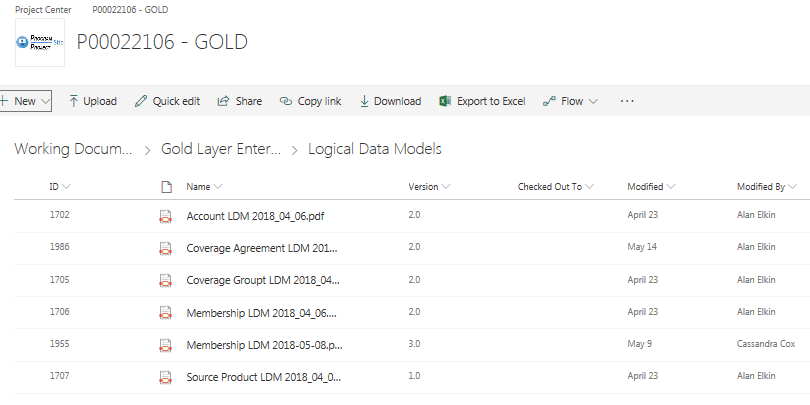
## 3.4 Key Design Decision

### 3.4.1 Logical model for the gold layer

The logical model is explained in the document included here for different target groups

[*https://myfyi.sharepoint.com/:f:/r/teams/projects/P00022106/Working%20Documents/Gold%20Layer%20Enterprise%20Data%20Modeling%20Artifacts/Logical%20Data%20Models?csf=1&e=tUbTq7*](https://myfyi.sharepoint.com/:f:/r/teams/projects/P00022106/Working%20Documents/Gold%20Layer%20Enterprise%20Data%20Modeling%20Artifacts/Logical%20Data%20Models?csf=1&e=tUbTq7)

Screen capture from SharePoint as below -



### 3.4.2 Physical model for the gold layer

Sharepoint link for all the physical models [Teamleads will fill the link ]

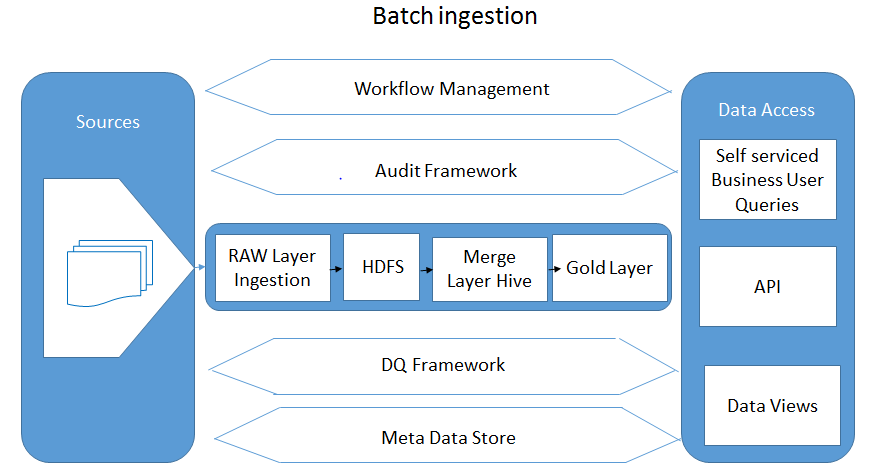
Please upload all Hive DDL scripts in sharepoint source wise and put sharepoint link here

# 4. Data Ingestion patterns

This process explains the data ingestion into raw layer i.e. HDFS storage and further into Hive managed tables.

## 4.1 Batch loads

Full File Loads will load all of the “latest and greatest” records by key regardless of the file date of the ingestion file. The below figure illustrates architecture for batch load processing.



Incremental File Processing will load all of the “latest and greatest” records based on the file date of the ingestion file where the file date is the same as the CURRENT DATE.

File Patterns:

Flat files (Fixed length)

Flat files (delimited)

XML

Source data would be moved to HDFS and loaded into Hive external tables for further processing. The data is loaded into different partitions in the RAW layer.

## 4.2 NRT loads

NRT (Near real time) data will be loaded through Kafka. First identify / capture the CDC Keys (source table primary keys) which are changed and write in Kafka to spin off no of processes.  We are identifying the CDC keys that are changed from the source tables by seeing the metadata (primary Keys) and writing KEYS into Kafka. Hive tables are partitioned by Date Fields

When the data is coming at regular interval, it is streamed into the RAW layer and then converted to loadable format. For example, RMS, we are streaming XML messages onto HDFS and then converting into Json format before loading into Hive tables.

# 5. Data Processing

## 5.1 RAW Layer

The HCSC datalake raw layer comprises of different source system data stored in Hadoop file system optimized for further processing. The ingestion process appending various source data as received from multiple source systems in different frequencies (minutes, daily, weekly, monthly) keeping mostly source facing structure. The incoming data stored in raw layer in specific partitions, using compression and in optimized row columnar format (ORC). All the raw layer data organized as external tables. The data is loaded into Hive tables using ORC format. The data ingested at raw layer can be retrieved using Hive query language. Since at this raw layer, data is only appended without updating or deleting, this layer can serve as a lineage of data over period of time.

Primary file types ingested:

* Flat files (Fixed length)
* Flat files (delimited)
* Multi record file
* XML

## 5.2 Merge Layer

The HCSC datalake merge layer contains current snapshot of the data. The merge layer table structures are mostly maintaining one to one mapping with a source file structure available at the raw layer. Since this merge layer consists of current snapshot of data, it primarily using upsert processes, for example, if a record already exists in this layer, the new incoming record will update this matched record otherwise it will insert. Merge layer contains only the latest record. Hive merge process has been utilized to accomplish the current snapshot. For an incremental load process, only the “latest and greatest” records are exported into a Hive table using the Hive RANK function by enforcing order by descending on change data capture timestamp. This rank function facilitates to identify “current record”. The data is stored in hive tables in this merge layer as internal ORC formatted tables.

## 5.3 Gold Layer (CEI layer)

The HCSC datalake gold layer contains consolidated data from one or many tables available from merge layer. These tables’ data will be updated based on matching record set. The target tables are same for all the data sources but could accommodate different grains. All the gold layer tables are optimized row columnar formatted, internally managed by hive. The gold layer’s table consolidated data created by joining the multiple hive tables as laid down in the mapping document. This internal Hive ORC table contains the results of a HQL join query and will cater further information for the Enterprise Data Model for all current records.

When a full file load is run for the first time, gold layer table will be empty but all full file current records will be merged into this “empty” table.

When an incremental file load is run, the CDC records will be merged here based on the business key. This key plays an important role for keeping historical data. Expectation is some kind of date will be part of this business key where there is a need to maintain history.

This data is not meant for consumption.

We can query this data using Hive query language from hive tables.

The run book for Bluestar describing the steps, shell script names and commands to execute is attached for easy reference.



## 5.4 Audit and logging

### 5.4.1 Audit

As a part of Bluestar Audit Processing, following tables supposed to be loaded -

* BLUESTAR\_WORK.PROC\_EXEC\_GOLD: Job status is captured once Kafka Producer/Consumer are completed.
* BLUESTAR\_WORK.PROC\_EXEC\_KEYS\_GOLD: Source table Primary keys and Target Table Primary keys are loaded into this Table.

In case of Kafka Producer run: BLUESTAR\_WORK.PROC\_EXEC\_KEYS\_GOLD extracts data from corresponding source CDC Tables. In case of multiple target table separated by ‘|’ in source parameter file, the corresponding target keys from all the corresponding source CDC tables will be loaded.

In case of Kafka Consumer run: BLUESTAR\_WORK.PROC\_EXEC\_KEYS\_GOLD Extracts data from corresponding Target table CDC\_KEYS table.

We achieve this above mentioned functionality with following two shell scripts:

* gold\_bstar\_audit\_producer\_wrapper.sh
* gold\_bstar\_audit\_consumer\_wrapper.sh

Script Components:



gold\_bstar\_audit\_producer\_wrapper.sh

This script is called by Zena scheduler after the Kafka Producer job is completed.

Usage:

sh gold\_bstar\_audit\_producer\_wrapper.sh <commonParameter\_file> <SourceTableName>

Example: sh gold\_bstar\_audit\_producer\_wrapper.sh /datalakebin/prod/gold/integration/params/membership/bluestar/paramFiles.prm vend

Step 1: By setting following Values from the parameter File

This Script will Load Data into BLUESTAR\_WORK.PROC\_EXEC\_GOLD : by using the child Script gold\_bstar\_audit\_load.sh

|  |  |
| --- | --- |
| **Columns** | **Source Value** |
| portfolio | From Parameter File |
| subject\_area | From Parameter File |
| create\_ts | Current Timestamp |
| src\_sys | From Parameter File |
| src\_tbl | From Parameter File |
| trg\_tbl | From Parameter File |
| exec\_func | STEP 1: KAFKA PRODUCER |
| exec\_sts | Completed |
| script\_nm | gold\_bstar\_audit\_producer\_wrapper |
| kafka\_topicname | From Parameter File |
| kafka\_offset | From Parameter File |
| zenajob\_nm | ZENA |
| exec\_dt | Current date |

Step 2: This Script will Load Data into BLUESTAR\_WORK.PROC\_EXEC\_KEYS\_GOLD: by using the child Script gold\_bstar\_audit\_writekeys.sh

| **Columns** | **CDC Source Table** |
| --- | --- |
| portfolio | From Parameter File |
| subject\_area | From Parameter File |
| crte\_ts | Unixtime (UNIX\_TIMESTAMP()) |
| src\_sys | From Parameter File |
| src\_tbl | From Parameter File |
| trg\_tbl | From Parameter File |
| exec\_func | STEP 1: Kafka Producer |
| exec\_sts | Completed |
| script\_nm | gold\_bstar\_audit\_producer\_wrapper |
| zenajob\_nm | Zena |
| src\_upd\_ts | Source CDC table : Column values from Source table parameter File |
| src\_sys\_natr\_key | Source CDC table : Column values from Source table parameter File |
| trg\_sys\_natr\_key | Source CDC table : Column values from Source table parameter File |
| kafka\_topicname | From Parameter File |
| kafka\_offset | From Parameter File |
| src\_db\_actn\_cd | Source CDC table : cdc\_actn\_cd |
| exec\_dt | Current date |

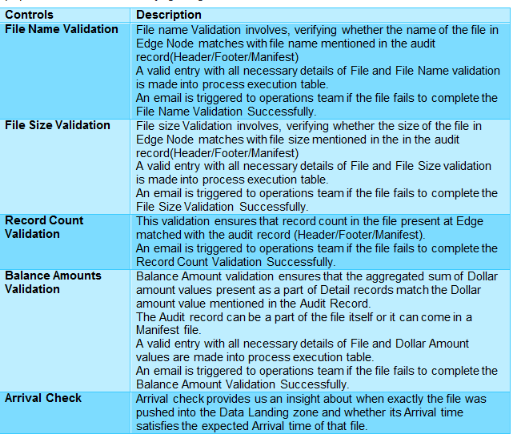
Please note that if you run a count of records in these two tables in “Test” environment, you need to utilize

* set hive.execution.engine=tez;
* set tez.queue.name=ETL;

Standard Audit framework is being utilized for membership ingestion to apply DISCO controls.

Audit framework reusable components will be used to validate input files before ingesting into datalake. Sharepoint link follows -

<https://myfyi.sharepoint.com/teams/projects/P00021363/Working%20Documents/Forms/AllItems.aspx?RootFolder=%2Fteams%2Fprojects%2FP00021363%2FWorking%20Documents%2FAudit%20Framework%2FTraining&FolderCTID=0x0120002BFAFC282ACEAF4D9691A3A6151B8E61&View=%7B0FE689B7%2D3DF5%2D4911%2D8248%2DED56457F293D%7D>



Audit Script Name: /datalakebin/prod/audit/ingestion/src/scripts/audit\_main.sh

No of Input parameters: 8

List of Parameters and sample values:

proc\_nm=TIS\_Datalake\_GPD\_S26\_File\_Ingestion\_Process

file\_pattern=HNM\_DAILY\_CONTACT\_LOG\*csv

hdfs\_file\_path=/prod/work/TMG/raw/

file\_name=HNM\_DAILY\_CONTACT\_LOG\_20170803\_040231.csv

schm\_file\_path=/datalake/Prod/Params/

schm\_file\_name=HPS\_DW\_REC\_CNT\_Schema.txt

edgenode\_path=/datalake/Prod/Work/Incoming/

mnf\_filename=NA

### 5.4.2 Logging

An audit framework is defined by creating two tables (proc\_exec\_gold & proc\_exec\_keys). The tables should be populated after the execution of each job. proc\_exec\_keys load keys (src\_sys\_natr\_key, trg\_sys\_natr\_key) into the table (Use CDC tables to populate the keys).

Proc Tables: These Tables should be loaded after the execution of each Job. As per the current Process we do capture Audit only for Kafka Write Job, Kafka Read Job and for GCF Load which uses Talend Jobs. We are not capturing audit for temp table and for Final Gold Layer table. We use a common/generic Job which we will call after execution of each Job to write into Proc tables.

we don’t have proc tables to write in for Merge (on both curr and target)

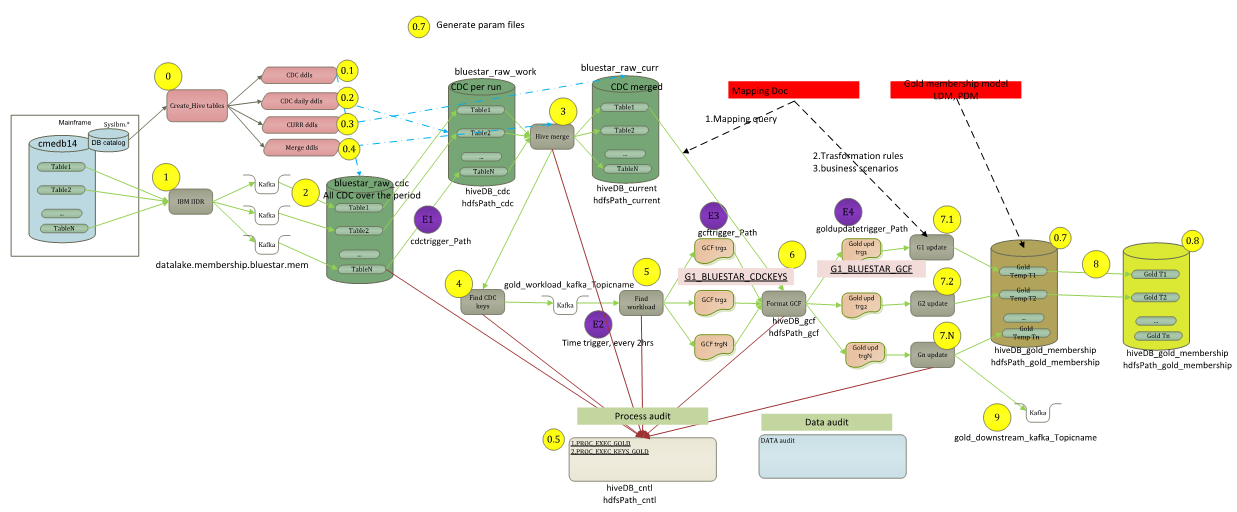
Proc\_exec\_keys – Loads keys (src\_sys\_natr\_key, trg\_sys\_natr\_key) into the table (Use CDC tables to populate the keys) - will be appended each time. So if there 1 million source load keys - this table will insert 1 million records We have to use purge Process to clean up the table over a period of time.

## 5.5 End to end scheduling

The scheduling of the jobs is done through zena scheduler. Currently Zena scheduling process is done for Bluestar.

# 6. Detailed Design

## 6.1 Bluestar

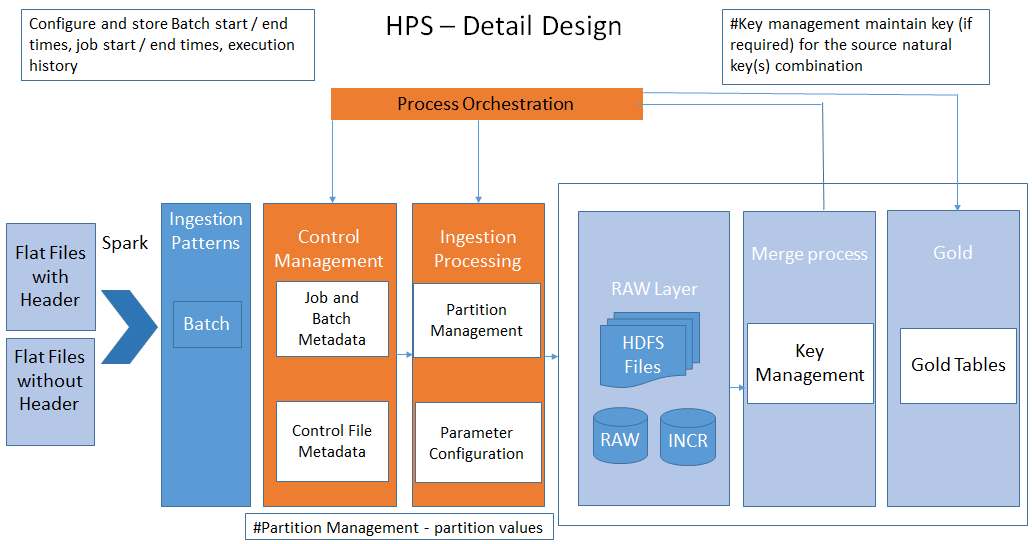


|  |  |  |
| --- | --- | --- |
| **Sl #** | **DB Name** | **DB Purpose** |
| 1 | BLUESTAR\_RAW | This database will contain bluestar source tables |
| 2 | BLUESTAR\_CDC | This database will contain bluestar CDC tables |
| 3 | BLUESTAR\_CURRENT | This database will contain bluestar current tables |
| 4 | BLUESTAR\_WORK | This database will contain bluestar specific CDC key tables , Audit tables |
| 5 | GOLD\_MEMBERSHIP\_WORK | This database will contain membership related temp tables |
| 6 | GOLD\_MEMBERSHIP\_CDC | This database will contain membership related GCF tables |
| 7 | GOLD\_MEMBERSHIP | This database will contain membership related gold target tables |

## 

## 6.2 HPS

The detailed design for HPS is illustrated in the below diagram.



## 6.2.1 Tech Stack

### 6.2.1.1 Hive

Hive is used to store data from Flat files in the RAW layer as well as merge & Gold layers. We are using external tables in RAW layer and internal tables in merge & gold layers.

### 6.2.1.2 Spark

Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams.

### 6.2.1.3 Kafka

Kafka is generally used for two broad classes of applications:

* Building real-time streaming data pipelines that reliably get data between systems or applications
* Building real-time streaming applications that transform or react to the streams of data

## 6.2.2 Process Overview

### 6.2.2.1 Ingestion

Source system data will be ingested to HDFS. For example, for a given primary key from the source table, an ingestion file may have multiple rows with the same key for a given snapshot.

There are two types of file formats used for ingestion. They are as given below.

* Flat Files with header (delimiter, fixed length)
* Flat Files without header (Shall provide meta information about Schema)

While writing into Hive tables, fixed length files would be converted into ORC format.

|  |  |  |
| --- | --- | --- |
| **Sl #** | **DB Name** | **DB Purpose** |
| 1 | HPS\_RAW | This database will contain HPS source tables |
| 2 | HPS\_INCR | This database will contain HPS CDC tables |
| 3 | HPS\_MERGE | This database will contain HPS current tables |
| 4 | HPS\_WORK | This database will contain HPS specific CDC key tables |
| 5 | GOLD\_MEMBERSHIP\_WORK | This database will contain membership related temp tables |
| 6 | GOLD\_MEMBERSHIP\_CDC | This database will contain membership related GCF tables |
| 7 | GOLD\_MEMBERSHIP | This database will contain membership related gold target tables |
| 8 | GOLd\_MEMBERSHIP\_HISTORY | This database will contain point in time historical data |

Frequency of the files:

* Daily: The loading process is defined in the next sections. Implementation steps are also given below.
* Monthly: The loading process for monthly files is same as loading process for daily files except the job schedule and change in parameter files if any. Monthly file is also having same type of data as daily file. If new records are present in the file, we have to insert them and for existing records, we have to update the records while loading into merge layer.

**Modes:**

There are two modes of operating the health care systems.

1. On Exchange: Enrollment to the market place
2. Off Exchange: Not enrolled through market place but got enrolled in some other way

We have to check whether we have this ‘On’ and ‘Off’ mode as a column / flag in the data. If we have this field, then we can de-activate the record if the member is in ‘Off’ mode.

There are some important tables in HPS as given below.

|  |  |
| --- | --- |
| **Entity** | **Entity Description** |
| Case\_Master | Contains basic and high level info about Cases |
| Employee | Contains data elements for each insured member |
| Dependent | Data elements for data associated with each member |
| Covg\_History | Location for all historical references to changes in coverage |
| Case\_Broker | Key Information from Case Master and Broker tables |
| Broker | Identifies Agent Broker ID’s |
| Benefit\_Details | Benefit detail data broken out by carrier |
| UW\_Census | Demographic Data |

Other tables can be found in the shared location. Directory Structure details are available in the following table:

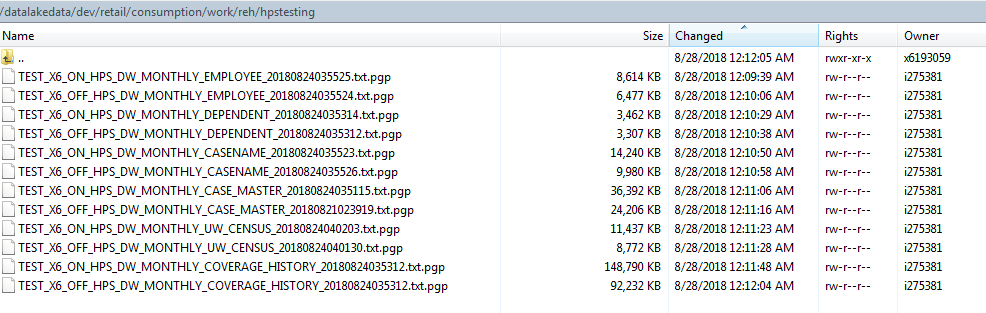
|  |  |  |  |
| --- | --- | --- | --- |
| **Sl** | **Element Description** | **Value** | **Comments** |
| 1 | Scripts Path | /datalakedata/prod/gold/integration/work/membership/hps/src/ | linux path |
| 2 | Paramfiles Path | /datalakedata/prod/gold/integration/params/membership/hps | linux path |
| 3 | Logpath | /datalakedata/dev/gold/integration/logs/membership/hps | linux path |
| 4 | Work path | /datalakedata/prod/gold/ingestion/work/ingestVms | NA |
| 5 | WorkIn path | /datalakedata/prod/gold/ingestion/work/ingestVms/Incoming | NA |
| 6 | Workoutpath | /datalakedata/prod/gold/ingestion/work/ingestVms/Outgoing | NA |
| 7 | HDFSworkpath | /dev/incoming/raw/Membership/HPS | hdfs path |
| 8 | AuditHDFSworkpath | /prod/work/raw/Membership/VMS/ACTIVE\_AUDIT | NA |
| 9 | GoldconsTriggerfilepath | /datalake/prod/gold/integration/work/Vms/trigger/ | NA |

Other details are as follows:

/datalakedata/prod/gold/integration/xferin/membership/hps/cdctrigger  
/datalakedata/prod/gold/integration/hql/membership/hps  
/datalakedata/dev/gold/integration/logs/membership/hps  
/datalakedata/prod/gold/integration/params/membership/hps  
/datalakedata/prod/gold/integration/xferin/membership/hps/gcftrigger  
/datalakedata/prod/gold/integration/xferin/membership/hps/goldupdatetrigger  
/datalakedata/prod/gold/integration/work/membership/hps/src/

Few HPS test datafiles being made available in DevAzure and Test datalake for facilitating “raw” to “gold” testing

* **DevAzure:** dazrslenap0001.app.dev.hcscint.net
* **Test Box:** twauslaenapp04.app.test.hcscint.net



|  |  |  |  |
| --- | --- | --- | --- |
| **FileType** | **ON/OFF** | **FileName** | **Record #** |
| EMPLOYEE | ON exchange | TEST\_X6\_ON\_HPS\_DW\_MONTHLY\_EMPLOYEE\_20180824035525.txt.pgp | 12801 |
| EMPLOYEE | OFF exchange | TEST\_X6\_OFF\_HPS\_DW\_MONTHLY\_EMPLOYEE\_20180824035524.txt.pgp | 9626 |
| DEPENDNET | ON exchange | TEST\_X6\_ON\_HPS\_DW\_MONTHLY\_DEPENDENT\_20180824035314.txt.pgp | 8731 |
| DEPENDNET | OFF exchange | TEST\_X6\_OFF\_HPS\_DW\_MONTHLY\_DEPENDENT\_20180824035312.txt.pgp | 8340 |
| CASENAME | ON exchange | TEST\_X6\_ON\_HPS\_DW\_MONTHLY\_CASENAME\_20180824035523.txt.pgp | 24630 |
| CASENAME | OFF exchange | TEST\_X6\_OFF\_HPS\_DW\_MONTHLY\_CASENAME\_20180824035526.txt.pgp | 17261 |
| CASE\_MASTER | ON exchange | TEST\_X6\_ON\_HPS\_DW\_MONTHLY\_CASE\_MASTER\_20180824035115.txt.pgp | 24630 |
| CASE\_MASTER | OFF exchange | TEST\_X6\_OFF\_HPS\_DW\_MONTHLY\_CASE\_MASTER\_20180821023919.txt.pgp | 16382 |
| UW\_CENSUS | ON exchange | TEST\_X6\_ON\_HPS\_DW\_MONTHLY\_UW\_CENSUS\_20180824040203.txt.pgp | 33366 |
| UW\_CENSUS | OFF exchange | TEST\_X6\_OFF\_HPS\_DW\_MONTHLY\_UW\_CENSUS\_20180824040130.txt.pgp | 25590 |
| COVERAGE\_HISTORY | ON exchange | TEST\_X6\_ON\_HPS\_DW\_MONTHLY\_COVERAGE\_HISTORY\_20180824035312.txt.pgp | 934726 |
| COVERAGE\_HISTORY | OFF exchange | TEST\_X6\_OFF\_HPS\_DW\_MONTHLY\_COVERAGE\_HISTORY\_20180824035312.txt.pgp | 579419 |

**Hive RAW Table:**

The ingestion files will be imported into a Hive Database (HPS\_SOURCE\_SYSTEM\_NAME). As an example for HPS Membership, the Hive Database is HPS\_RAW. For a Full Load, all of the records are exported into a RAW table. For eg. we are having one table called HPS\_RAW. Case\_Master table. We will have Hive external tables with YYYYMM partitioned and ORC serialization. Please refer below for each Flat file related tables –

| Source FF name | Corresponding Hive table  Database : **HPS\_RAW** | Hive table properties | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| Partitioned | Bucketed (#) | External | Format | Transactional |
| CASE\_MASTER | HPS\_CASE\_MASTER | Yes- | 0 | Yes | ORC (snappy) | False |
| CASENAME | HPS\_CASE\_NAME | Yes- | 0 | Yes | ORC (snappy) | False |
| EMPLOYEE | HPS\_EMPLOYEE | Yes- | 0 | Yes | ORC (snappy) | False |
| DEPENDENT | HPS\_DEPENDENT | Yes- | 0 | Yes | ORC (snappy) | False |
| COVERAGE\_HISTORY | HPS\_COVERAGE\_HISTORY | Yes- | 0 | Yes | ORC (snappy) | False |
| UW\_CENSUS | HPS\_UW\_CENSUS | Yes- | 0 | Yes | ORC (snappy) | False |

**Hive cdc Table:**

All the incremental files would be placed HDFS. The data would be further written into Kafka topic. This data also merge to curr table. If the RAW table name in HPS\_RAW is hps\_Case\_Master, then we create a table called HPS\_merge.Hps\_Case\_Master\_cdc table for loading this data. The earlier data location will be altered to new location.

| Incremental FF name | Corresponding Hive table  Database : **HPS\_INCR** | **Hive table properties** | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| Partitioned | Bucketed (#) | External | Format | Transactional |
| CASE\_MASTER | Hps\_CASE\_MASTER | No | No | No | ORC | False |
| CASENAME | Hps\_CASENAME | No | No | No | ORC | False |
| EMPLOYEE | Hps\_EMPLOYEE | No | No | No | ORC | False |
| DEPENDENT | Hps\_DEPENDENT | No | No | No | ORC | False |
| COVERAGE\_HISTORY | Hps\_COVERAGE\_HISTORY | No | No | No | ORC | False |
| UW\_CENSUS | HPS\_UW\_CENSUS | No | No | No | ORC | False |

### 6.2.2.2 Merging

We will merge cdc tables available in HPS\_INCR database with curr table HPS\_MERGE database. Now the curr table contains the *latest and greatest* records for any record

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Raw table name | Corresponding Hive table: HPS\_MERGE | **Hive table properties** | | | | |
| Partitioned | Bucketed (#) | External | Format Compression | Transactional |
| HPS\_CASE\_MASTER | HPS\_CASE\_MASTER | NO | 8 | No | ORC Snappy | True |
| HPS\_CASENAME | HPS\_CASENAME | NO | 8 | No | ORC Snappy | True |
| HPS\_EMPLOYEE | HPS\_EMPLOYEE | No | 8 | No | ORC Snappy | True |
| HPS\_DEPENDENT | HPS\_DEPENDENT | No | 8 | No | ORC Snappy | True |
| HPS\_COVERAGE\_HISTORY | HPS\_COVERAGE\_HISTORY | No | 8 | No | ORC Snappy | True |
| UW\_CENSUS | HPS\_UW\_CENSUS | No | 8 | No | ORC Snappy | True |

### 6.2.2.3 Gold

We will apply required quality checks and then loading into the final Gold table (SOURCE\_SYSTEM\_NAME\_Gold).

In the eg. we have described, all the data from HPS\_Merge. Case\_Master table would be loaded into HPS\_Gold. Case\_Master table. This is a 1:1 mapping table where all the data from merged table would be loaded into Gold.

This also gives flexibility to apply Data Quality & Transformation rules if any in future.

| Target table name | Relevant Source Tables | Corresponding Hive table | **Hive table properties** | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Partitioned | Bucketed (#) | External | Format | Transactional |
| MBR\_MBRSHIP | EMPLOYEE  DEPENDENT  CASE\_NAME  CASE\_MASTER  COVERAGE\_HISTORY  EDI\_ENROLL\_XREF  UW\_CENSUS |  | NO | 5 | No | ORC Snappy | True |
| MBR\_MBRSHP\_EVNT | COVERAGE\_HISTORY  EDI\_ENROLL\_XREF  EMPLOYEE  DEPENDENT |  | No | 10 | No | ORC Snappy | True |
| MBR\_DETLS | CASE\_NAME  EMPLOYEE  DEPENDENT  CASE\_MASTER  UW\_CENSUS |  | no- | 5 | No | ORC Snappy | True |
| MBRSHP\_SBSDY | CASE\_NAME  EMPLOYEE  DEPENDENT  COVERAGE\_HISTORY |  | no- | 10 | No | ORC Snappy | True |
| MBRSHP\_PROV\_SEL | CASE\_NAME  COVERAGE\_HISTORY  EMPLOYEE  DEPENDENT"  CASE\_MASTER  EDI\_ENROLL\_XREF |  | no- | 5 | No | ORC Snappy | True |
| COVERAGE\_HISTORY | CASE\_NAME  COVERAGE\_HISTORY  CASE\_MASTER  EMPLOYEE  DEPENDENT |  | no- | 10 | No | ORC Snappy | True |
| MBR\_ADDR | CASE\_MASTER  CASE\_NAME  EMPLOYEE  DEPENDENT  CASE\_MASTER |  | no- | 5 | No | ORC Snappy | True |
| MBR\_MCARE | CASE\_NAME  EMPLOYEE  DEPENDENT  CASE\_MASTER  COVERAGE\_HISTORY  EDI\_ENROLL\_XREF |  | no- | 10 | No | ORC Snappy | True |

## 6.2.3 Format Standardization

The source files are Flat Files with header and Flat files without header. Format standardization is not required in this case.

## 6.2.4 Full Loads

Full loads are not part of current HPS design and hence not described here.

## 6.2.5 Incremental Loads

Incremental File Processing will be done on a daily basis and based on the file date of the ingestion file where the file date is the same as the CURRENT DATE. All the previous files would be moved to an archive location.

The procedure for incremental loads is explained in detail in the **6.1.2 Process Overview**.

## 6.2.6 Low Level Steps

The implementation steps for Daily / Monthly frequency are as given below:

Step 1: Files are present in HDFS. Load Raw data into HPS\_RAW. Hps\_Case\_Master in orc format

Step 2: HPS\_INCR.HPS\_Case\_Master, HPS\_MERGE.HPS\_Case\_Master are created by using shell scripts and parameter files and create cdckeys table manually for each target table.

Step 3: Prepare and execute script to load data CDC.

Step 4: Create param files for each source table in the respective Linux path.

Step 5: Alter file location based on current timestamp to get the latest file data.

Step 6: Prepare and execute shell scripts for creating HPS\_INCR.HPS\_Case\_Master, HPS\_MERGE.HPS\_Case\_Master are created and generate merge statements

Step 7: CDC tables contain changed rows / new rows in each individual table (every time overwrite happens in CDC and merged into merge table)

Step 8: Kafka code to create CDC keys data that contains all the 4 tables data of changed rows. Kafka writes into individual target CDC keys table.

Step 9: Create GCF table by inner joining CDC Keys & Curr tables.

Step 9: Temp table will be created by applying transformations on GCF table.

Step 10: Use hqls to merge data into Gold layer.

There are 3 partitions in RAW, 1) load Type (F (Full), I (Incremental)), Date and Time. For eg. The files will be stored in ./F/Date/Time folder.

Bucketing: on CIM no & other primary keys of sources tables in Merge layer (8 buckets)

Bucketing: on CIM no & other primary keys of sources tables in Gold layer (8 buckets)

## 6.2.7 HPS Gold PIT

In addition - there will be PIT (point in-time history) loads which will keep history. PIT will hold data in different database as explained below taking data from HPS\_GOLD database/ The process will append every day’s data from temp tables to PIT history tables All these PIT history tables will be added with one column “REC\_EFF\_DT” which will hold the current timestamp of ETL process.

****

**Following tables are candidate for PIT –**

1. MBR\_ADDR
2. MBR\_DETLS
3. MBR\_MBRSHP
4. MBR\_MBRSHP\_EVNT
5. MBR\_MCARE
6. MBR\_SBSDY
7. MBRSHP\_COVRG\_ELE
8. MBRSHP\_PROV\_SEL

There will be 8 history tables corresponding to all these above 8 Gold tables and will be available in the database GOLD\_PIT. The history table’s names are as follows –

1. MBR\_ADDR
2. MBR\_DETLS
3. MBR\_MBRSHP
4. MBR\_MBRSHP\_EVNT
5. MBR\_MCARE
6. MBR\_SBSDY
7. MBRSHP\_COVRG\_ELE
8. MBRSHP\_PROV\_SEL

The history loads are introduced late in the data lake. When we are getting the updated data, the **point in time history table** keeps the history of the data along with latest record. Gold tables only contain all the latest snapshots.

The features of point in time history:

1. Data is present in Temp table.
2. Data is loaded into Gold table as per the current process
3. Take data from Temp table and insert into the Gold PIT table. It is appending data into this table unlike Upsert we are doing in Gold table.

Create one extra column Record Effective Date which is nothing but a system time in this table.

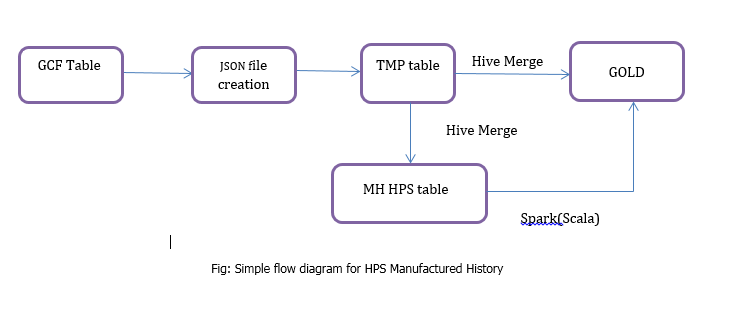
## 6.2.8 HPS Manufactured History

For any input table which has Date column value as N/A or NULL from source, populate those Date fields with Current Date and insert the record in the table.

In future, for the same Date column, if we get value from source, update that column with the value from Source (SCD Type 2) and insert as new record with Effective Date as current Load date.

Below tables are considered for this scenario and Manufactured History record will be created for the same for the columns for which the Mapping has NULL values or marked as N/A.

1. MBR\_ADDR
2. MBR\_DETLS
3. MBR\_MBRSHP
4. MBR\_MBRSHP\_EVNT
5. MBR\_MCARE
6. MBRSHP\_COVRG\_ELE
7. MBRSHP\_PROV\_SEL



**MH - Manufacturing History**

1) GOLD\_MEMBERSHIP\_WORK.MH\_HPS\_<table\_name> is target facing persistent table (without any complex datatypes like struct)

2) We need to create GOLD\_MEMBERSHIP\_WORK.MH\_HPS\_<table\_name> - table which is going to hold only enterprise business key eff\_dt, end\_dt and other respective columns for that particular table

(ex target\_sys\_natr\_key, eff\_dt\_end\_dt, c1,c2,c3....)

3) Retrieve (pull) data from temp table which is getting populated in the main gold table creation phase after gcf table and push it into MH\_<<table\_name>>. When inserting data into MH\_<<table\_name>>, eff\_dt and end\_dt will get populated with etl timestamps.

4) From CDCkeys table we will know what are all the keys got changed and all those keys are present in the MH\_<<table\_name>> table

5) Search using Scala through Enterprise BK along with eff\_dt and retrieve the latest record out of it.

6) MH table will have enterprise business key, other columns available in struct and any otehr columns if mentioned in mapping document which needs to be loaded with some values

7) Eff Dt will be loaded with current processing timestamp

8) End data will be loaded through high date (12/31/9999)

9) Gold target table needs to be rebuild each time. If it has 3 records earlier and another record arrives, the struct will have 4 records

10) Currently there are no "I" or "U" flags available in source data but data ingestion process actually identifying the upsert candidates. Please populate CDC\_ACTN\_CD accordingly. This will be applicable general starting from Merge layer. For new records it needs to be "I" and for existing records it needs to be "U"

**MH Table**

BK1 01/01/2018 12/31/9999 c1, c2, c3

BK1 02/01/2018 12/31/9999 c1, c2, c3 🡺 New Row

**Gold Table with Struct data structure**

BK1 01/01/2018 12/31/9999 c1, c2, c3

Search: Search by BK1 [Enterprise business key].

Hive merge is limited to BK1.

Scala code is going to build next array <01/01/2018 12/31/9999 c1, c2, c3>

We are going to search BK1 and new record and see whether we get any match in the struct. As, there is no match we have to do insert, which is update scenario. So we update old record and append new record through scala.

BK1 01/01/2018 12/31/9999 c1, c2, c3

BK1 02/01/2018 12/31/9999 c1, c2, c3 => This is New Struct

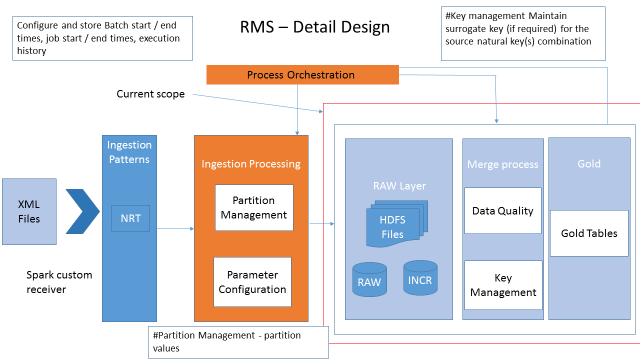
01/01/2018 02/01/2018 c1, c2, c3

Search this date for update

Search this date for insert

## 6.3 RMS

The detailed design for RMS is illustrated in the below diagram.



## 6.3.1 Tech Stack

### 6.3.1.2 Hive

Hive is used to store data from XML files in the RAW layer as well as merge & Gold layers. We are using external tables in RAW layer and internal tables in merge & gold layers.

### 6.3.1.2 Spark

Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams. Spark streaming is used for ingesting the streaming data from the source. It is a kind of NRT (Near Real Time) data that is streamed into the application.

### 6.3.1.3 Kafka

Kafka is generally used for two broad classes of applications:

* Building real-time streaming data pipelines that reliably get data between systems or applications
* Building real-time streaming applications that transform or react to the streams of data

Kafka is used to ingest NRT data for consuming in the form of topics.

## 6.3.2 Process Overview

### 6.3.2.1 Ingestion

Source system data will be ingested to HDFS. XML is semi-structured. Since the structure of XML is variable by design, we cannot have defined mapping. Thus, to process the XML in Hadoop, you need to know the tags required to extract the data.

The type of file format used for ingestion is XML files. This is the only source file format we have currently for loading.

Ingestion Patterns:

Only XML files is the source for RMS and all the initial and incremental data would be coming through XML files.

The database structure shall be as follows:

|  |  |  |
| --- | --- | --- |
| **Sl #** | **DB Name** | **DB Purpose** |
| 1 | RMS\_RAW | This database will contain RMS source tables |
| 2 | RMS\_INCR | This database will contain RMS CDC tables |
| 3 | RMS\_MERGE | This database will contain RMS current tables |
| ~~4~~ | RMS\_WORK | This database will contain RMS specific CDC key tables |
| 5 | GOLD\_MEMBERSHIP\_WORK | This database will contain membership related temp tables |
| 6 | GOLD\_MEMBERSHIP\_CDC | This database will contain membership related GCF tables |
| 7 | GOLD\_MEMBERSHIP | This database will contain membership related gold target tables |
| 8 | GOLD\_MEMBERSHIP\_HISTORY | This database will contain point in time historical data |

There sample file in RMS is as given below.



Currently the data is having one complex type. All the cXML's files should be in consistency with dmih.xsd provided by the client.

~~We should take data from Data Power Message Broker and every 15 min, data needs to be processed.~~

~~Use custom receivers and subscribe to MQ channel using MQ libraries. Spark streaming application creates spark context to consume custom receivers. Then Dstreams are serialized into RAW table. Data would be loaded in Json format and subsequently in a single Hive table.~~

~~Assumptions:~~

1. ~~Since Spark can read XML source data from existing MQ, need not write a Kafka producer to write the data into Kafka Topic. (eliminating data redundancy)~~
2. ~~Assuming the MQ message contains only one transaction XML message (start tag:~~ *~~<MembershipTrx>~~* ~~end tag:~~ *~~</MembershipTrx>)~~* ~~which will avoid the xml parsing issues.~~
3. ~~Target layer must contain the latest XML data and RAW layer contains all the XML data that we get from MQ.~~

~~Ingestion Process:~~

~~If there is any historical data, need to process them and push it into Raw and Target layers.~~

1. ~~Write a custom receiver to stream the data from MQ to Spark Dstreams (if there are no pre build libraries).~~
2. ~~For each RDD in Dstream write the XML messages to RAW layer~~
3. ~~For each RDD in Dstream contains individual xml messages. Write a xml parser program to convert the individual message to Json string as value and pull a unique identifier as key (like~~ *~~SourceSystemName, CorporateEntityCode, SubscriberIdNumber~~*~~)~~

~~Create a DF using these key and value~~

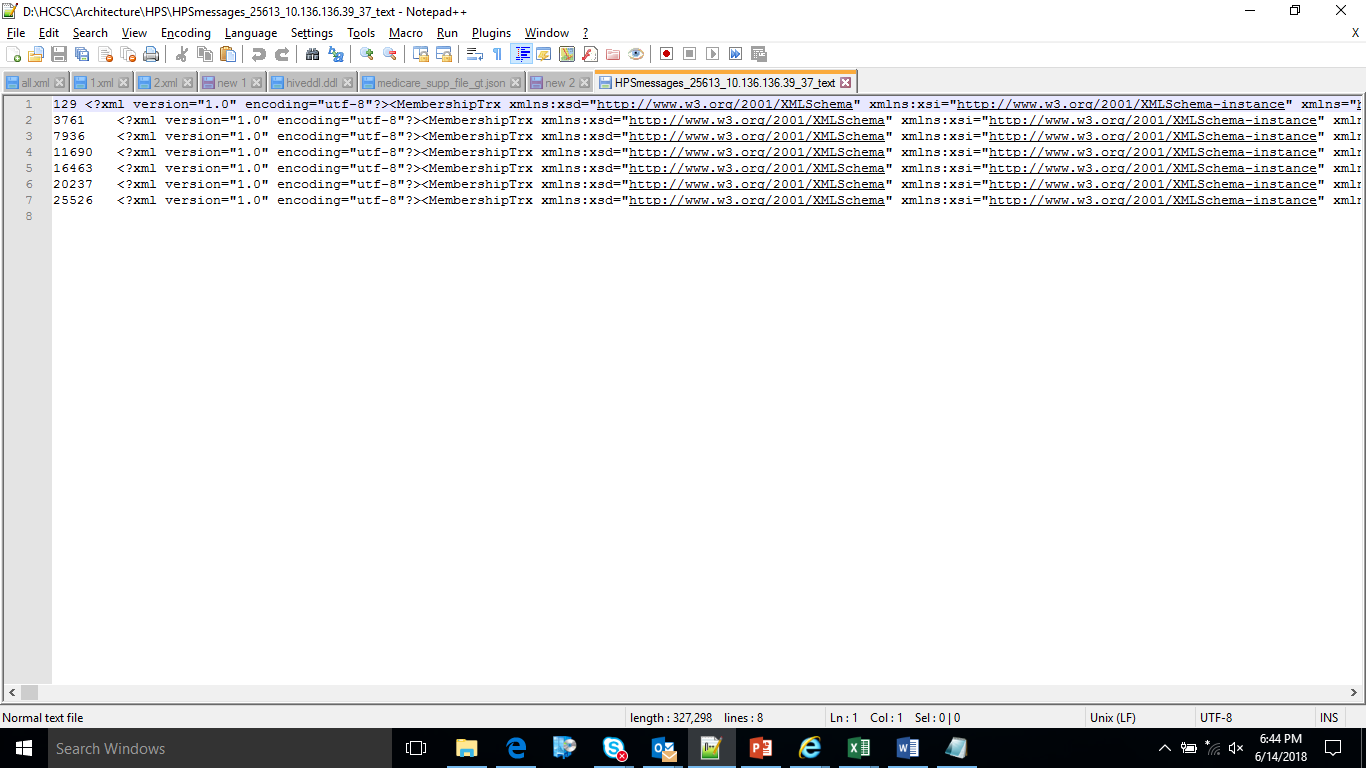
~~DF contains 2 fields one is key: String and value: String, embed key and value as single JSON string. As below~~

~~{key: “HPS|IL1|000921415975”, “MemberTx”: {“TransactionHeader”:{ “CtfTypeId:2261”, “SourceLastUpdateTS”: 20180422233010630793 ……. ……….}}}~~

1. ~~a> if target table is transaction enabled and orc, write the dataframe to temp hive table and using hive merge command update the hive table (conditions: latest Transaction id or latest source message created timestamp)~~

~~b> if target table is partitioned on specified distinct keys, identify the updated partitions and do a left outer join with the DF and hive target table and insert overwrite the target table partitions.~~

1. ~~Final target table is containing two fields key and value. Create a hive view on top of this table using Json SerDe~~



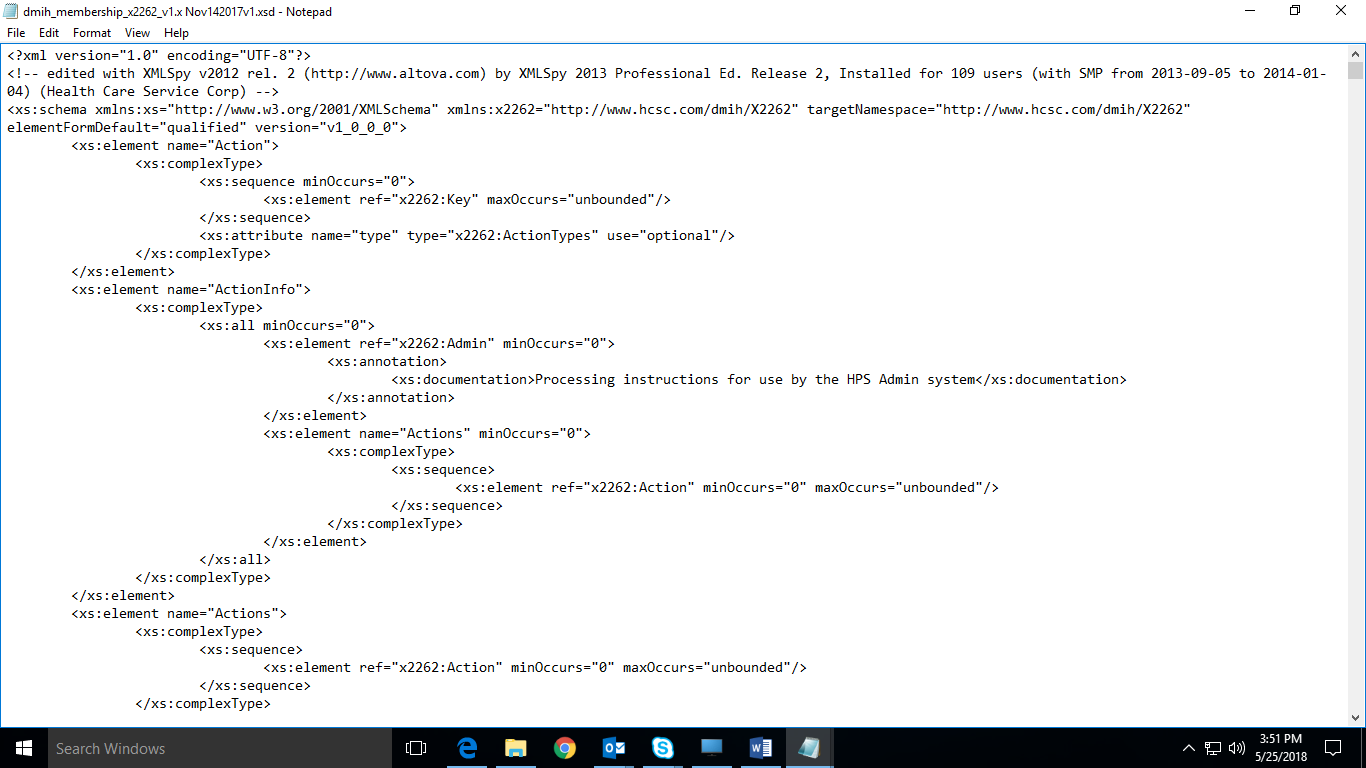
The XML files contains Transaction header and various elements under each header. The TransactionHeader is the parent Header level data and contains several sub-elements in it. Each XML file contains one TransactionHeader data and this can be loaded into a separate table as a header or master record. The rest of the elements would be loaded into separate tables as given in the Table structure document attached here. Please note that we would be getting multiple xml files to load the data. The job should pick each of the files and loaded into the tables.

The key information for each table is also included in the Table structure document.



For eg. RAW table can be *RMS\_RAW.*Member. When we get new xml files at an interval of 15 min, we will append data to the RAW table and store the data.

The XSD document is as given in the below screenshot.



### 6.3.2.2 Merging

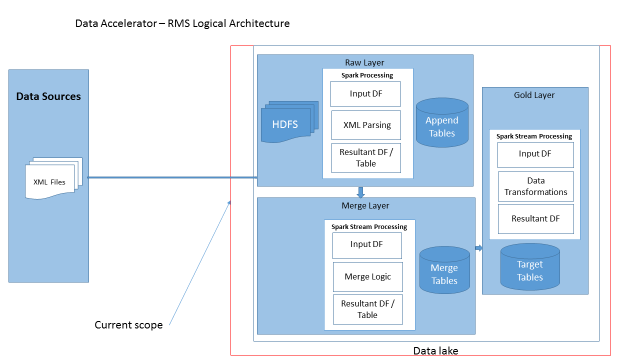
When we are merging for the first time, all the data from the RAW table is read into the merging table. At this stage, merged table is a snapshot of RAW table and contains current data at any time.

For eg. merged table can be named as RMS\_MERGE.Member and it is one table.

When we are loading incremental data, the data would be appended to RAW layer as mentioned above and keeping the original XML file in HDFS location. All the incremental data would be loaded into RMS\_Incr.Member table in the form Json data and subsequently into Hive tables.

While merging we will merge only latest records into merge table comparing the data in incremental data load table using hive merge. Target table merge will happen in ORC format.

In this eg. we will compare records from RMS\_Incr.Member table and RMS\_CURR*.*Member and upsert the data.



### 6.3.2.3 Gold

The final target tables for RMS also same as Bluestar & HPS data sources. We need to update the same target tables with 1:1 mapping without any transformations into the Gold table.

For eg. the table name would be GOLD\_MEMBERSHIP.MBR\_MBRSHP table.

GOLD\_MEMBERSHIP\_WORK will contain membership related temp tables. GOLD\_MEMBERSHIP\_CDC will contain membership related GCF tables

## 6.3.3 Format Standardization

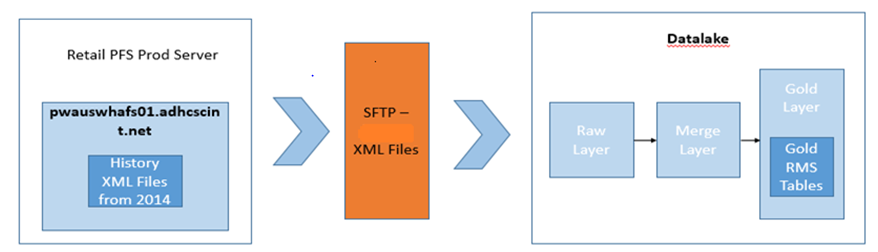
The source files are XML Files with multiple levels nested within the elements / nodes.

## 6.3.4 Full Loads (Day 0 load)

Full loads are when we are loading the XML files first time as part of current RMS design. The process includes loading data into RAW tables. Then creating a snapshot of RAW table in Merge table. The requirement is to transfer the data to Gold layer without implementing any business rules.

History Data will be received starting 2014 as Feed XML or Multiple XML files for all 12 tables. These files will be SFTP from Retail CFS PROD Server - **pwauswhafs01.adhcscint.net** to datalake**.** This is the server we are using to SFTP Vantage coverages to GOLD using below SFTP.

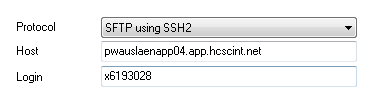
**High Level Flow Diagram – Full Load:**



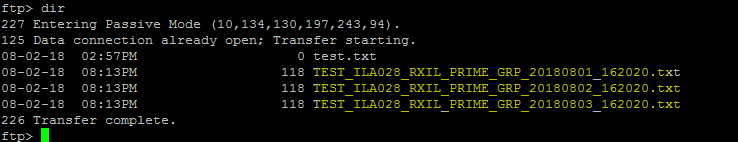
We have observed that the FTP ports are already open between Retail CFS PROD server (pwauswhafs01.adhcscint.net) to Datalake.

Retail CFS PROD server will hold onetime xml files or multiple XML files – it will avoid the effort by ESP Team to work on opening ftp port from datalake server

SFTP protocol details below -



Small test successfully transfers files to Retail CFS server from datalake and vice versa using SFTP. We just need to create an automated script.



## 6.3.5 Gold PIT

In addition - there will be PIT (point in-time history) loads which will keep history. PIT will hold data in different database as explained below taking data from Gold tables. The PIT process will append every day’s data from temp tables to PIT history tables and run in batch mode once a night at 11:00 PM CST (assuming there are no dependency at this 11:00 PM CST) . All these PIT history tables will be added with one column “REC\_EFF\_DT” which will hold the current timestamp of ETL process.

There will be a new table GOLD\_MEMBERSHIP\_WORK.“TEMP\_PIT\_RMS\_<TableName>”. The purpose of this table will be to hold entire day’s data coming at different times. This table will be populated using Hive merge so that if same record is coming multiple times in a day – it should not insert but only updates. The GOLD\_MEMBERSHIP\_WORK.”TEMP\_PIT\_RMS” will be getting data from temp table’s.

****

**Following tables are candidate for PIT –**

1. MBR\_DETLS
2. MBR\_ADDR
3. MBR\_MBRSHP\_EVNT
4. MBR\_MCARE
5. MBR\_MBRSHP
6. MBRSHP\_COVRG\_ELE
7. MBRSHP\_SBSDY
8. MBRSHP\_PROV\_SEL
9. MBRSHP\_PRDCR\_ASGNMT ( *earlier it was MBRSHP\_AGNT\_ASGMNT – not created yet*)

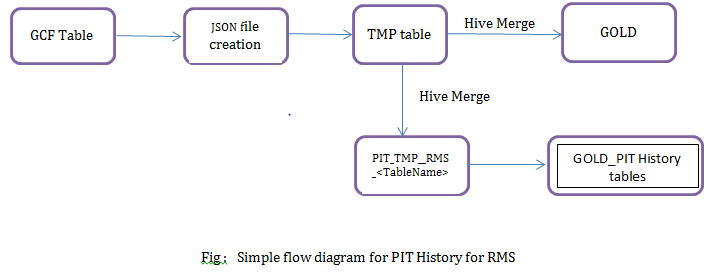
There will be 9 history tables corresponding to all these above 9 Gold tables and will be available in the database GOLD\_MEMBERSHIP\_HISTORY. The history table’s names are as follows –

1. MBR\_DETLS
2. MBR\_ADDR
3. MBR\_MBRSHP\_EVNT
4. MBR\_MCARE
5. MBR\_MBRSHP
6. MBRSHP\_COVRG\_ELE
7. MBRSHP\_SBSDY
8. MBRSHP\_PROV\_SEL
9. MBRSHP\_PRDCR\_ASGNMT (earlier it was MBRSHP\_AGNT\_ASGMNT – not created yet)

The history loads are introduced late in the data lake. When we are getting the updated data, the **point in time history table** keeps the history of the data along with latest record. Gold tables only contain all the latest snapshots.

The features of point in time history:

* Data is present in Temp table.
* Data is loaded into Gold table as per the current process of hive merging
* Take data from Temp table and insert into the GOLD\_MEMBERSHIP\_WORK.“TEMP\_PIT\_RMS\_<TableName> table for the entire day using Hive merge process so that if same member’s data is coming twice or more times in a day, it will only consider the latest changes for that specific member.
* Around 11;00 PM CST, once the entire day’s data captured, it can append into corresponding History table into GOLD\_PIT database. Note that history table will be of same structure as in Gold with one extra column “REC\_EFF\_DATE” which is ETL processing timestamp
* Once the entire data appended in GOLD\_PIT database, the daywide temp tables needs to be purged so that it can start holding next day’s data



Initial Load:

|  |  |  |
| --- | --- | --- |
| Member ID | Mem Name | Key |
| 1021 | ABCD | 145 |

Gold Table:

|  |  |  |
| --- | --- | --- |
| Member ID | Mem Name | Key |
| 1021 | ABCD | 145 |

History table: Assuming the record added on 01/21/2016 -

|  |  |  |  |
| --- | --- | --- | --- |
| Member ID | Mem Name | Column value | Rec\_Eff\_Date |
| 1021 | ABCD | 145 | 01/21/2016 |

Incremental Load:

|  |  |  |
| --- | --- | --- |
| Member ID | Mem Name | Column value |
| 1021 | ABCD | 219 |

After getting updated record through incremental load:

Gold table:

|  |  |  |
| --- | --- | --- |
| Member ID | Mem Name | Column value |
| 1021 | ABCD | 219 |

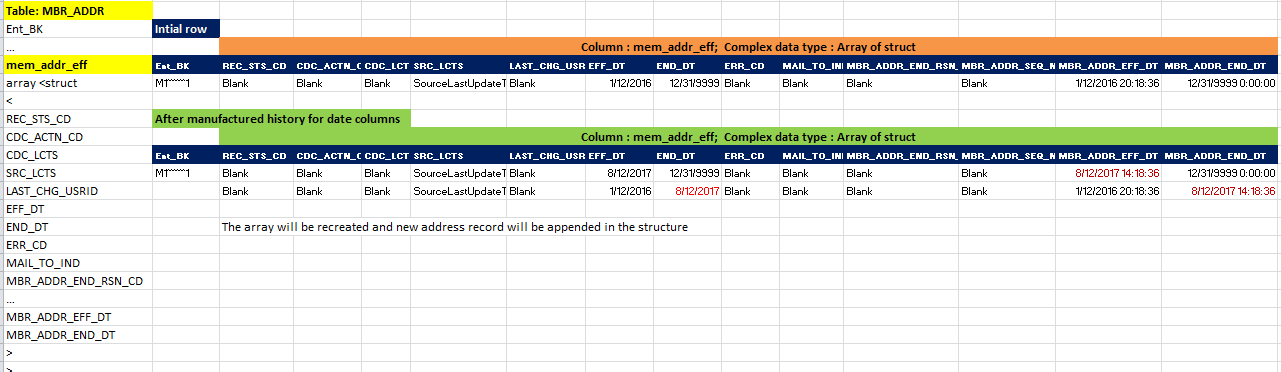
History table: Assuming a new record arrived on 12/04/2017.

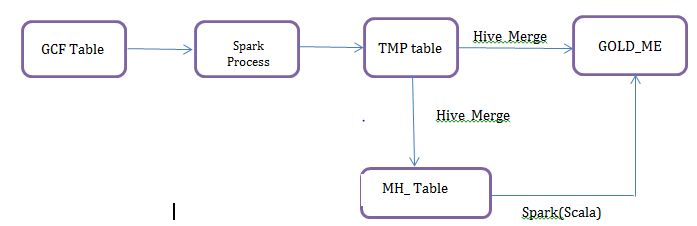
|  |  |  |  |
| --- | --- | --- | --- |
| Member ID | Mem Name | Column value | Rec\_Eff\_Date |
| 1021 | ABCD | 219 | 12/04/2017 |
| 1021 | ABCD | 145 | 01/21/2016 |

In case of multiple loads into temp table in a day, we can append data to a Hist\_temp table and finally push data into PIT History table.

## 6.3.6 Manufactured History

Manufactured History data is loaded when we don’t have data in some fields which are important for business. Initially we get some data for a particular member and stored in the struct. When we get additional information for the struct in the same record, we will append those values in the struct. In the below example, initial row contains data before adding manufactured data. The second table contains one more record in the struct with EFF\_DT as 8/12/2017.





**MH - Manufacturing History**

1) GOLD\_MEMBERSHIP\_WORK\_HIST.MH\_RMS\_<table\_name> is target facing persistent table (without any complex datatypes like struct)

2) We need to create GOLD\_MEMBERSHIP\_WORK.MH\_RMS\_<table\_name> - table which is going to hold only enterprise business key eff\_dt, end\_dt and other respective columns for that particular table

(ex target\_sys\_natr\_key, eff\_dt\_end\_dt, c1,c2,c3....)

3) We retrieve (pull) data from temp table which is getting populated in the main gold table creation phase after gcf table and push it into MH\_<<table\_name>>. When inserting data into MH\_<<table\_name>> we will populate the null eff\_dt and end\_dt with etl timestamps

4) From CDCkeys table we will know what are all the keys got changed and all those keys are present in the MH\_<<table\_name>> table

5) We are doing search using Scala through Enterprise BK along with eff\_dt and retrieve the latest record out of it.

6) MH table will have enterprise business key, other columns available in struct and any otehr columns if mentioned in mapping document which needs to be loaded with some values

7) Eff Dt will be loaded with current processing timestamp

8) End data will be loaded through high date (12/31/9999)

9) Gold target table needs to be rebuild each time. If it has 3 records earlier and another record arrives, the struct will have 4 records

10) Currently there are no "I" or "U" flags available in source data but data ingestion process actually identifying the upsert candidates. Please populate CDC\_ACTN\_CD accordingly. This will be applicable general starting from Merge layer. For new records it needs to be "I" and for existing records it needs to be "U"

**MH Table**

BK1 01/01/2018 12/31/9999 c1, c2, c3

BK1 02/01/2018 12/31/9999 c1, c2, c3 🡺 New Row

**Gold Table with Struct data structure**

BK1 01/01/2018 12/31/9999 c1, c2, c3

Search: Search by BK1 [Enterprise business key].

Hive merge is limited to BK1.

Scala code is going to build next array <01/01/2018 12/31/9999 c1, c2, c3>

We are going to search BK1 and new record and see whether we get any match in the struct. As, there is no match we have to do insert, which is update scenario. So we update old record and append new record through scala.

BK1 01/01/2018 12/31/9999 c1, c2, c3

BK1 02/01/2018 12/31/9999 c1, c2, c3 => This is New Struct

01/01/2018 02/01/2018 c1, c2, c3

Search this date for update

Search this date for insert

## 6.3.7 Incremental Loads

The frequency of xml files is 15 minutes and the data shall be loaded from all these files. All the xml files are serialized to sequential files and have a key value pair. The data from XML files will be streamed into the database.

1. Reading binary file from HDFS as ingested by existing process

2. Converting the binary XML file into Text file using HDFS command

3. Parsing the XML file for specific elements which are needed for RMS feed as identified in the mapping documents and appending into 12 tables in the RAW layer

4. Then follow the process of moving data from raw layer to final Gold layer

**Updates:**

we always get full xml file even there is a single attribute change. So, we need to identify the changes in XML files and load them accordingly.

Updates:

* Load the initial file in Json format.
* Complete the process till Gold layer.
* Append second Json file data into RAW layer and load it in Incr layer in Json format.
* Compare incremental data with Current table data
* The changes from both the files identified and upsert into Current table

The procedure for incremental loads is explained in detail in the **6.3.2 Process Overview**.

## 6.3.8 Target Table Updates

We have to note that the target table is same for all the data sources. All the data from Bluestar, HPS & RMS loaded into the same table at different times. Hence, the target table will have latest data from the latest data source updated.

## 6.3.9 Low level steps

The implementation steps for Daily / Monthly frequency are as given below:

Step 1: Files are present in HDFS in Binary Format.

Step 2: Convert Binary format into Text format and Load Raw data into RMS\_RAW.Member

Step 3: RMS\_Incr.Member, RMS\_MERGE.Member, MembershipTrx \_cdckeys are created by using shell scripts and parameter files.

Step 4: Prepare and execute script to load data into RMS\_Incr.

Step 5: Create param files for each source table in the respective Linux path.

Step 6: Alter file location based on current timestamp to get the latest file data.

Step 7: Prepare and execute shell scripts for loading RMS\_INCR.Member and generate merge statements

Step 8: CDC tables contain changed rows / new rows in each individual table (every time overwrite happens in CDC and merged into Curr table)

~~Step 9: Kafka code to create CDC keys data that contains all the participating tables data of changed rows. Kafka writes into individual target CDC keys table.~~

~~Step 10: Create GCF table by inner joining CDC Keys & Curr tables.~~

~~Step 11: Temp table will be created by applying transformations on GCF table.~~

Step 12: Use hqls to merge data into Gold layer.

## 6.3.10 Business Scenarios

Business Scenarios:

* Data is changing at only one level of the file
* Data is changing at multiple levels of the file
* Data elements are missing at child level
* New data is added at different levels
* Data changed between {} and data changed between []

Add Dependent

We compared BeforeAddDependent Json and AfterAddDependentJson Json files and found the following trend.

AddDependent is having 9 levels of data.

Sub-elements of the file are

* Client
* Members
* MemberContacts
* MembershipEnrollments
* MembershipProviders
* InterestedParties
* BrokerInfo
* EnrollemntInfo

These 8 sub-elements are common across all the files even though there is change in data.

One sample file is attached that is showing the difference in data.



*Plan Change*

Loaded Plan change is having the following changes from before file and after file.

Content mismatch

Wrong Message id

*VoluntaryWithdrawl*

VoluntaryWithdrawl is having the following changes from before file and after file.

End Date is different

Maintenance Type Code is changed

Message ID is wrong

*BrokerChange*

BrokerChange is having the following changes from before file and after file.

Changes in Transaction Header elements are found

Message ID, Create Date, MaintReasonCode, Source System Name, Source Update TS are changing.

*InitialEnrollment*

Before Json file is missing to compare the data.

*ProviderChange*

ProviderChange is having the following changes from before file and after file.

Message ID, Create Date, Source Update TS are changed.

*ActiveRenewal*

ActiveRenewal is having the following changes from before file and after file.

Message ID, Create Date, Source Update TS are changed in Transaction Header.

Street Address Line1, Text, RenewalTypeCode, Amount, Application Source Code, @Field, EappNumber are changing.

*PassiveRenewal*

PassiveRenewal is having the following changes from before file and after file.

Message ID, Create Date, Source Update TS are changed in Transaction Header.

ExchangeMbrPremiumAmt, MaintReasonCode values are changed.

*PTD\_Maintenance*

PTD\_Maintenance is having the following changes from before file and after file.

Message ID, Create Date, Source Update TS are changed in Transaction Header.

Text, Amount values are changed.

## 6.3.11 Table Structure

Table structure can be defined on the basis of keys defined for the XML elements.

**Assumption:** Keys are defined at Level 1 by Data modelers, then we can have 12 tables which are to be loaded with the relevant data. They are as follows:

* Transaction Header
* Member
* Member Event
* Membership Provider
* Member Details
* Membership Enrollment
* Medicare
* Enrollment Info
* Client
* Broker
* Address
* Coverage Agreement

Because all the files contain these 12 elements even though data is changing, we can define 12 tables and the respective data is loaded into the tables as one row as defined by Json format.

Write a xml parser program to convert the individual message to Json string as value and pull a unique identifier as key (like *SourceSystemName, CorporateEntityCode, SubscriberIdNumber*)

Create a DF using these key and value

DF contains 2 fields one is key: String and value: String, embed key and value as single JSON string. As below

{key: “HPS|IL1|000921415975”, “MemberTx”: {“TransactionHeader”: {“CtfTypeId:2261”, “SourceLastUpdateTS”: 20180422233010630793 ……. ……….}}}

Rest of the process is as explained in section 6.3.2.1.

**Logic for Finding CDC:**

* RMS\_Merge. MembershipTrx \_CDC, RMS\_Merge. Member are created.
* Prepare and execute script to load CDC data.
* Create param files for each source table in the respective Linux path.
* Alter file location based on current timestamp to get the latest file data.
* Prepare and execute shell scripts for creating RMS\_INCR.Member table and RMS\_Merge.Member table and generate merge statements
* Incr tables contains changed rows / new rows in each individual table (every time overwrite happens in CDC and merged into Merge table)
* ~~Kafka code to create CDCkeys data that contains all the tables data of changed rows. Kafka writes into individual target CDC keys table.~~
* ~~Create GCF table by inner joining CDCKeys & Curr tables.~~
* Load input XML file into RMS\_RAW.Member in append mode and Load the same data in RMS\_INCR.Member with truncate and Load mode.
* Using Hive Merge, insert/update the changes from INCR table into MERGE layer table.
* Move the updates from MERGE table to Gold Layer using Hive Merge statements.

**RMS Raw Table:**

| Source FF name | Corresponding Hive table  Database : **RMS\_RAW** | Hive table properties | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| Partitioned | Bucketed (#) | External | Format | Transactional |
| TRANSACTION\_HEADER | TRANSACTION\_HEADER | Yes- | 0 | Yes | ORC (snappy) | False |
| MEMBER | MEMBER | Yes- | 0 | Yes | ORC (snappy) | False |
| MEMBER\_EVENT | MEMBER\_EVENT | Yes- | 0 | Yes | ORC (snappy) | False |
| MEMBERSHIP\_PROVIDER | MEMBERSHIP\_PROVIDER | Yes- | 0 | Yes | ORC (snappy) | False |
| MEMBER\_DETAILS | MEMBER\_DETAILS | Yes- | 0 | Yes | ORC (snappy) | False |
| MEMBERSHIP\_ENROLLMENT | MEMBERSHIP\_ENROLLMENT | Yes- | 0 | Yes | ORC (snappy) | False |
| MEDICARE | MEDICARE | Yes- | 0 | Yes | ORC (snappy) | False |
| ENROLLMENT\_INFO | ENROLLMENT\_INFO | Yes- | 0 | Yes | ORC (snappy) | False |
| CLIENT | CLIENT | Yes- | 0 | Yes | ORC (snappy) | False |
| BROKER | BROKER | Yes- | 0 | Yes | ORC (snappy) | False |
| ADDRESS | ADDRESS | Yes- | 0 | Yes | ORC (snappy) | False |
| CONVERAGE\_ENROLLMENT | CONVERAGE\_ENROLLMENT | Yes- | 0 | Yes | ORC (snappy) | False |

**Hive CDC Table:**

| Source FF name | Corresponding Hive table  Database : **RMS\_INCR** | Hive table properties | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| Partitioned | Bucketed (#) | External | Format | Transactional |
| TRANSACTION\_HEADER | TRANSACTION\_HEADER | Yes- | 0 | Yes | ORC (snappy) | False |
| MEMBER | MEMBER | Yes- | 0 | Yes | ORC (snappy) | False |
| MEMBER\_EVENT | MEMBER\_EVENT | Yes- | 0 | Yes | ORC (snappy) | False |
| MEMBERSHIP\_PROVIDER | MEMBERSHIP\_PROVIDER | Yes- | 0 | Yes | ORC (snappy) | False |
| MEMBER\_DETAILS | MEMBER\_DETAILS | Yes- | 0 | Yes | ORC (snappy) | False |
| MEMBERSHIP\_ENROLLMENT | MEMBERSHIP\_ENROLLMENT | Yes- | 0 | Yes | ORC (snappy) | False |
| MEDICARE | MEDICARE | Yes- | 0 | Yes | ORC (snappy) | False |
| ENROLLMENT\_INFO | ENROLLMENT\_INFO | Yes- | 0 | Yes | ORC (snappy) | False |
| CLIENT | CLIENT | Yes- | 0 | Yes | ORC (snappy) | False |
| BROKER | BROKER | Yes- | 0 | Yes | ORC (snappy) | False |
| ADDRESS | ADDRESS | Yes- | 0 | Yes | ORC (snappy) | False |
| CONVERAGE\_ENROLLMENT | CONVERAGE\_ENROLLMENT | Yes- | 0 | Yes | ORC (snappy) | False |

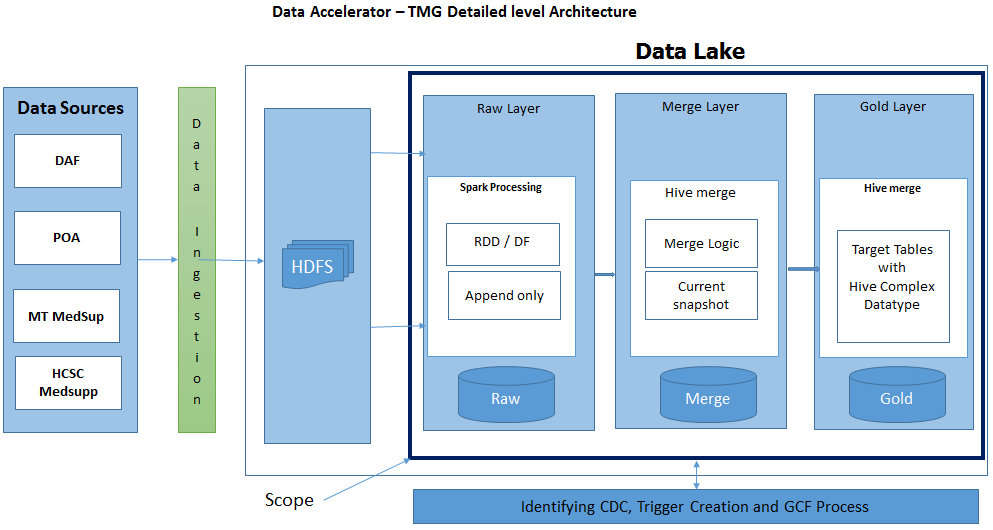
**Hive Merge Table:**

We will merge CDC tables available in RMS\_INCR database with curr table RMS\_MERGE database. Now the curr table contains the latest and greatest records for any record.

| Source FF name | Corresponding Hive table  Database : **RMS\_MERGE** | Hive table properties | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| Partitioned | Bucketed (#) | External | Format | Transactional |
| TRANSACTION\_HEADER | TRANSACTION\_HEADER | Yes- | 0 | Yes | ORC (snappy) | False |
| MEMBER | MEMBER | Yes- | 0 | Yes | ORC (snappy) | False |
| MEMBER\_EVENT | MEMBER\_EVENT | Yes- | 0 | Yes | ORC (snappy) | False |
| MEMBERSHIP\_PROVIDER | MEMBERSHIP\_PROVIDER | Yes- | 0 | Yes | ORC (snappy) | False |
| MEMBER\_DETAILS | MEMBER\_DETAILS | Yes- | 0 | Yes | ORC (snappy) | False |
| MEMBERSHIP\_ENROLLMENT | MEMBERSHIP\_ENROLLMENT | Yes- | 0 | Yes | ORC (snappy) | False |
| MEDICARE | MEDICARE | Yes- | 0 | Yes | ORC (snappy) | False |
| ENROLLMENT\_INFO | ENROLLMENT\_INFO | Yes- | 0 | Yes | ORC (snappy) | False |
| CLIENT | CLIENT | Yes- | 0 | Yes | ORC (snappy) | False |
| BROKER | BROKER | Yes- | 0 | Yes | ORC (snappy) | False |
| ADDRESS | ADDRESS | Yes- | 0 | Yes | ORC (snappy) | False |
| CONVERAGE\_ENROLLMENT | CONVERAGE\_ENROLLMENT | Yes- | 0 | Yes | ORC (snappy) | False |

## 6.4 TMG

The detailed design for TMG is illustrated in the below diagram.



### 6.4.1 Tech Stack

### 6.4.1.2 Hive

Hive is used to store data from flat files in the RAW layer as well as merge & Gold layers. We are using external tables in RAW layer and internal tables in merge & gold layers.

### 6.4.1.2 Spark

Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams.

### 6.4.2 Process Overview

### 6.4.2.1 Ingestion

Source system data files are available in HDFS Incoming RAW. We need to ingest those files data into consumable RAW layer (Hive ORC) and subsequently to merge and Gold layers. There are 4 types of source files used for ingestion. They are DAF, POA, MT MedSup and HCSC Supplemental file.

* DAF
* POA
* MT MedSup
* HCSC Supplemental file

The file frequency and type of load is explained in the following table.

|  |  |  |  |
| --- | --- | --- | --- |
| Type of File | Incremental / Full | Frequency | No of source files |
| DAF | I | Daily | 6 |
|  | F | Weekly / Monthly | 6 |
| POA | I | Daily | 3 |
| MT Medsup | I | Daily | 1 |
|  | F | Daily | 1 |
| HCSC Supplemental file | F | Monthly | 3 |

**HCSC Supplemental monthly file details**

Every member must contain one Subscriber\_Record and at least one event record. In the event that a Subscriber record does not pull at least one event record the member will fall to the internal error report to be reviewed.

The member’s will be listed in ascending order by Member ID. The event lines should present in alphabetical order with multiple event lines to produce in chronological date order.

Record order should be as follows:

Subscriber\_Member

CRCO (if applicable) – If Multiple for Subscriber/Group, sort by Effective Date in ascending order

DABL (if applicable) – If Multiple for Subscriber/Group, sort by Effective Date in ascending order

DETH (if applicable) - If Multiple for Subscriber/Group, sort by Effective Date in ascending order

EGHP (if applicable) – If Multiple for Subscriber/Group, sort by Effective Date in ascending order

ESEO (if applicable) – If Multiple for Subscriber/Group, sort by Effective Date in ascending order

ESRD (if applicable) – If Multiple for Subscriber/Group, sort by Effective Date in ascending order

HICN (if applicable) – If Multiple for Subscriber/Group, sort by Effective Date in ascending order

HSPC (if applicable) – If Multiple for Subscriber/Group, sort by Effective Date in ascending order

INST (if applicable) – If Multiple for Subscriber/Group, sort by Effective Date in ascending order

LATE (if applicable) – If Multiple for Subscriber/Group, sort by Effective Date in ascending order

LICS (if applicable) – If Multiple for Subscriber/Group, sort by Effective Date in ascending order

MDCD (if applicable) – If Multiple for Subscriber/Group, sort by Effective Date in ascending order

MSP (if applicable) – If Multiple for Subscriber/Group, sort by Effective Date in ascending order

PBP (if applicable) – If Multiple for Subscriber/Group, sort by Effective Date in ascending order

PCO (if applicable) – If Multiple for Subscriber/Group, sort by Effective Date in ascending order

PREM (if applicable) – If Multiple for Subscriber/Group, sort by Effective Date in ascending order

The file running timings are as given below.

**MAPD** – the second Friday of the month at 6am

**PDP** – the second Thursday of the month at 6am

**MMAI** – the second Thursday of the month at 4am.

The HDFS paths are detailed for each type of files.

|  |  |
| --- | --- |
| **Type of file** | **Path** |
| DAF | /prod/incoming/raw/Membership/TMG/DAF |
| POA | /prod/incoming/raw/Membership/TMG/POA |
| MT Medsup | /prod/incoming/raw/Membership/TMG/MedSupp/MT/ |
| HCSC Supplemental Files | /prod/incoming/raw/Membership/TMG/daf\_supp |

While writing into Hive tables, files would be converted into ORC format. The HCSC Supplemental file is placed in the shared location.

****

|  |  |  |
| --- | --- | --- |
| **Sl #** | **DB Name** | **DB Purpose** |
| 1 | TMG\_RAW | This database will contain TMG source tables |
| 2 | TMG\_INCR | This database will contain TMG CDC tables |
| 3 | TMG\_MERGE | This database will contain TMG current tables |
| 4 | TMG\_WORK | This database will contain TMG specific CDC key tables |
| 5 | GOLD\_MEMBERSHIP\_WORK | This database will contain membership related temp tables |
| 6 | GOLD\_MEMBERSHIP\_CDC | This database will contain membership related GCF tables |
| 7 | GOLD\_MEMBERSHIP | This database will contain membership related gold target tables |
| 8 | GOLd\_MEMBERSHIP\_HISTORY | This database will contain point in time historical data |

**Hive RAW Table:**

The ingestion files will be imported into a Hive Database (SOURCE\_SYSTEM\_NAME\_RAW). As an example for TMG Membership, the Hive Database is TMG\_RAW. For a Full Load, all of the records are imported into a RAW table. For eg. we are having one table called TMG\_RAW. dy\_actvy\_file\_mbrshp\_medicare. We will have Hive external tables with MMYYYY partitioned and text serialization (except HCSC Supp type). Please refer below for each Flat file related tables.

Assuming that all type of files will be placed in HDFS RAW layer without any sub-directories. Based on file pattern / type pattern, copy / move the files into respective partitioned directories with ORC serialization except HSCS Supp files. (<Table>/<dt=MMYYYY>/files\*).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Source FF name | Corresponding Hive table  Database : **TMG\_raw** | Hive table properties | | | | |
| Partitioned |  | External | Format | Transactional |
| Medicare | dy\_actvy\_file\_mbrshp\_medicare | Yes - |  | Yes | ORC | False |
| Medicaid | dy\_actvy\_file\_mbrshp\_medicaid | Yes- |  | Yes | ORC | False |
| POA | pwr\_of\_atty\_hist | Yes- |  | Yes | ORC | False |
| MT Medsupp | tmg\_mt\_med\_supp | Yes- |  | Yes | ORC | False |

For HCSC Supp type of files:

Since these are multi record files (semi-structured), not creating Hive tables on top of this data. Instead, copy them into (<Table>/<dt=MMYYYY>/files\*).

**Hive Current Table:**

All the incremental files would be placed on HDFS. Spark application can read data from RAW tables based on type and partition as a data frame and insert / overwrite into current table. Current table is created before starting this process. Capture CDC keys by comparing the data from Current table with Merge table. When we load the data for first time, merged table contains all the latest and greatest data. If the RAW table is TMG\_RAW. dy\_actvy\_file\_mbrshp\_medicare, then we create a table called TMG\_Incr. dy\_actvy\_file\_mbrshp\_medicare table for loading this data. Every time, this Curr table will be refreshed.

**HCSC Supplemental file:**

The loading of supp files into Curr table is as follows:

Curr / Merge

Spark / Hive merge

HDFS

Gold

HDFS: supp/dt=201801/\*.txt

RAW Files

RAW

Find CDC keys using left outer joining CURR table (driving table) and merge table and store in a temp table.

| Incremental FF name | Corresponding Hive table  Database : **TMG\_Incr** | **Hive table properties** | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| Partitioned | Bucketed (#) | External | Format | Transactional |
| Medicare | dy\_actvy\_file\_mbrshp\_medicare | No | No | No | TEXT | False |
| Medicaid | dy\_actvy\_file\_mbrshp\_medicaid | No | No | No | TEXT | False |
| POA | pwr\_of\_atty\_hist | No | No | No | TEXT | False |
| MT Medsupp | tmg\_mt\_med\_supp | No | No | No | TEXT | False |
| HCSC Suppl | tmg\_medicare\_supp | No | No | No | TEXT | False |

### 6.4.2.2 Merging

The generated CDC keys in the previous step are stored in a temp table. Based on these keys, generate a hive merge script and update in merge table.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Raw table name | Corresponding Hive table  Database TMG\_Merge | **Hive table properties** | | | | |
| Partitioned | Bucketed (#) | External | Format Compression | Transactional |
| Medicare | dy\_actvy\_file\_mbrshp\_medicare | Yes - | Yes | No | ORC Snappy | True |
| Medicaid | dy\_actvy\_file\_mbrshp\_medicaid | Yes- | Yes | No | ORC Snappy | True |
| POA | pwr\_of\_atty\_hist | Yes- | Yes | No | ORC Snappy | True |
| MT Medsupp | tmg\_mt\_med\_supp | Yes- | Yes | No | ORC Snappy | True |
| HCSC Suppl | tmg\_medicare\_supp | Yes- | Yes | No | ORC Snappy | True |

**Supp file:**

The generated CDC keys in the previous step are stored in a temp table. Based on these keys, generate necessary hive merge script and update in merge table.

* Keys used in CDC process to identity delta records
* DAF Keys: To be filled in after getting confirmation from Ben / Kiran
* POA Keys: To be filled in after getting confirmation from Ben / Kiran

**Picking the latest record from raw to merge layer:**

When incoming raw record's column doesn't have timestamp which might be part of business key. We need to group by the business key and in case count () > 1, assign sequence numbers for multiple records with same business key and pick the nth record. Assuming oldest record has assigned value 1 and last record has value n (which is latest).

**Example Query used for CDC keys:**

SELECT

DY\_ACTVY\_FILE\_MBRSHP\_MEDICARE.SRC\_DB\_ACTN\_CD AS SRC\_CDC\_ACTN\_CD,

DY\_ACTVY\_FILE\_MBRSHP\_MEDICARE.cdc\_upd\_ts AS SRC\_CDC\_LCTS,

DY\_ACTVY\_FILE\_MBRSHP\_MEDICARE.BA\_CVG\_RT\_LCTS,

DY\_ACTVY\_FILE\_MBRSHP\_MEDICARE.LAST\_CHG\_USRID AS SRC\_LAST\_CHG\_USRID,

DY\_ACTVY\_FILE\_MBRSHP\_MEDICARE.BA\_CVG\_RT\_EFF\_DT,

DY\_ACTVY\_FILE\_MBRSHP\_MEDICARE.BA\_CVG\_RT\_END\_DT,

DY\_ACTVY\_FILE\_MBRSHP\_MEDICARE.ERR\_CD,

DY\_ACTVY\_FILE\_MBRSHP\_MEDICARE.ACCT\_NBR ,

DY\_ACTVY\_FILE\_MBRSHP\_MEDICARE.CORP\_ENT\_CD ,

DY\_ACTVY\_FILE\_MBRSHP\_MEDICARE.BEN\_AGMT\_NBR,

DY\_ACTVY\_FILE\_MBRSHP\_MEDICARE.CORP\_TIER\_NBR,

DY\_ACTVY\_FILE\_MBRSHP\_MEDICARE.TIER\_AGRGT\_SEQ\_NBR AS SRC\_TIER\_AGRGT\_SEQ\_NBR,

DY\_ACTVY\_FILE\_MBRSHP\_MEDICARE.RATE\_STRUC\_TYP\_CD,

DY\_ACTVY\_FILE\_MBRSHP\_MEDICARE.BA\_CVG\_RTE\_SEQ\_NBR,

TIER\_AGRGT.TIER\_ABBR\_CD,

DY\_ACTVY\_FILE\_MBRSHP\_MEDICARE.ACA\_FEE\_AMT,

DY\_ACTVY\_FILE\_MBRSHP\_MEDICARE.ADMIN\_CHRG\_CVG\_PCT,

DY\_ACTVY\_FILE\_MBRSHP\_MEDICARE.AGE\_BRKT\_METH\_CD,

SRC\_DB\_ACTN\_CD,

cdc\_upd\_ts,

BA\_CVG\_RT\_LCTS,

LAST\_CHG\_USRID,

BA\_CVG\_RT\_EFF\_DT,

BA\_CVG\_RT\_END\_DT,

ERR\_CD,

ACCT\_NBR ,

CORP\_ENT\_CD ,

BEN\_AGMT\_NBR,

CORP\_TIER\_NBR,

TIER\_AGRGT\_SEQ\_NBR,

RATE\_STRUC\_TYP\_CD,

BA\_CVG\_RTE\_SEQ\_NBR,

ACA\_FEE\_AMT,

ADMIN\_CHRG\_CVG\_PCT,

AGE\_BRKT\_METH\_CD,

CHG\_RSN\_CD,

CAN\_CD,

FR\_AGE\_NBR,

GNDR\_CD,

MCARE\_CRVE\_OUT\_IND,

PREV\_CAN\_CD,

PREV\_END\_DT,

RATE\_AMT,

RT\_CHG\_CVG\_PCT,

RT\_SZ\_CD,

RT\_TYP\_CD,

THRU\_AGE\_NBR,

TOT\_ADMIN\_CHRG\_PCT,

TOT\_RT\_CHG\_PCT,

rank() over (partition BY CORP\_TIER\_NBR,corp\_ent\_cd,TIER\_AGRGT\_SEQ\_NBR order by BA\_CVG\_RT\_EFF\_DT desc) as rnk

FROM TMG\_gold.dy\_actvy\_file\_mbrshp\_medicare\_curr

) DY\_ACTVY\_FILE\_MBRSHP\_MEDICARE

on DY\_ACTVY\_FILE\_MBRSHP\_MEDICARE.corp\_ent\_cd        =  TIER\_AGRGT.corp\_ent\_cd

and DY\_ACTVY\_FILE\_MBRSHP\_MEDICARE.corp\_tier\_nbr      =  TIER\_AGRGT.corp\_tier\_nbr

and DY\_ACTVY\_FILE\_MBRSHP\_MEDICARE.tier\_agrgt\_seq\_nbr =  TIER\_AGRGT.tier\_agrgt\_seq\_nbr

where DY\_ACTVY\_FILE\_MBRSHP\_MEDICARE.rnk='1';

### 6.4.2.3 Reconciliation Process

Reconciliation process is comparing the data from monthly full file with the data already loaded through incremental files in the data lake. Reconciliation file with monthly frequency has more accurate data.

The data loaded through daily loads into RAW layer is reconciled against the data loaded with monthly data and check whether all the data is loaded into the layer. Also, reconciliation process takes care of data that is properly moved to the next layer that is merge. When an incremental data load is happening on a daily basis after monthly load, we can check the accuracy of the incremental data with monthly recon files which are more accurate.

Similarly, the reconciliation process checks whether all the data is properly merged into Gold layer and required fields are updated.

**Process to mark a record as “Deleted” in Gold table –**

There may be a chance that some dropped member is not present into full reconciliation file comparing to the final gold tables dataset. “Dropped Member” means depending upon the situation of a subscriber one record may be dropped compared to the gold table dataset. In that case we need to update that specific record as “deleted” in the Gold layer table. Follow the below steps to search and update

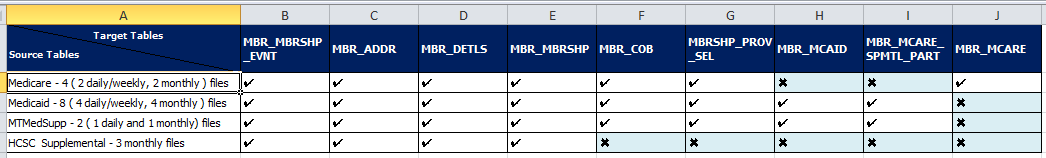
1. Once the new Full reconciliation file received, identify all the records **having distinct enterprise\_business\_key columns** and store into an array
2. Pull up all the distinct **enterprise\_business\_key columns** from corresponding gold table so that you can compare with the previously stored set (lookup)
3. Once an **enterprise\_business\_key columns** is identified which is present in Gold table dataset but not in reconciliation file, then that record needs to be marked as deleted in the gold table
4. Column level information needed from mapping document which column need to be updated in that gold table as “deleted”

### 6.4.2.4 Gold

We will apply required quality checks and then loading into the final Gold table GOLD\_MEMBERSHIP using hive merge. There are no business rules defined at this moment.

As we have described, all the data from TMG\_Merge. dy\_actvy\_file\_mbrshp\_medicare table would be loaded into necessary target tables (GOLD\_MEMBERSHIP.TargetTable) as per mapping document. Hive merge will be used to update the data in target tables.

Following matrix illustrates the assignment of source files to different target tables.



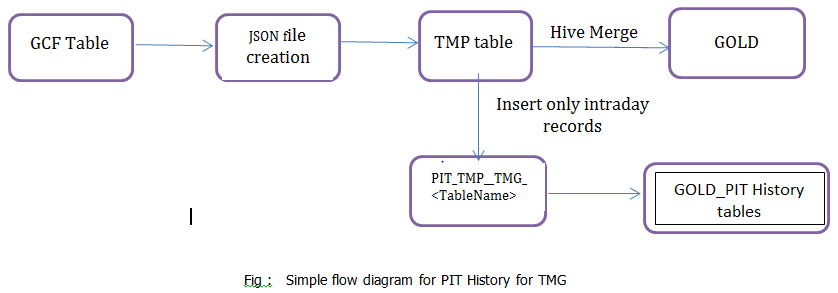
| **Target table name** | **Relevant Source Tables** | **Corresponding Hive table** | **Hive table properties** | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Partitioned** | **Bucketed (#)** | **External** | **Format** | **Transactional** |
| MBR\_MBRSHP\_EVNT | Medicare |  | Yes6 | Yes | No | ORC Snappy | TRUE |
| Medicaid |
| MTMedSupp |
| HCSC Supp |
| MBR\_ADDR | Medicare |  | Yes | Yes | No | ORC Snappy | TRUE |
| Medicaid |
| MTMedSupp |
| HCSC Supp |
| MBR\_DETLS | Medicare |  | Yes | Yes | No | ORC Snappy | TRUE |
| Medicaid |
| MTMedSupp |
| HCSC Supp |
| MBR\_MBRSHP | Medicare |  | Yes | Yes | No | ORC Snappy | TRUE |
| Medicaid |
| MTMedSupp |
| HCSC Supp |
| MBR\_COB | Medicare |  | Yes | Yes | No | ORC Snappy | TRUE |
| Medicaid |
| MTMedSupp |
| MBRSHP\_PROV\_SEL | Medicare |  | Yes | Yes | No | ORC Snappy | TRUE |
| Medicaid |
| MTMedSupp |
| MBR\_MCAID | Medicaid |  | Yes | Yes | No | ORC Snappy | TRUE |
| MTMedSupp |
| MBR\_MCARE\_SPMTL\_PART | Medicaid |  | Yes | Yes | No | ORC Snappy | TRUE |
| MTMedSupp |
| MBR\_MCARE | Medicare |  | Yes | Yes | No | ORC Snappy | TRUE |

### 6.4.2.5 TMG Gold PIT

In addition - there will be PIT (point in-time history) loads which will keep history. PIT will hold data in different database as explained below taking data from TMG\_GOLD database/ The process will append every day’s data from temp tables to PIT history tables All these PIT history tables will be added with one column “REC\_EFF\_DT” which will hold the current timestamp of ETL process. There may be Terminated/Suspended members, may be applicable only for few cases where the subscriber is active. Consider lth following scenario for member DOB. Please refer to embedded excel.

**For PIT strategy and day 0 approaches – refer to the following PPT**

****



**Following tables are candidate for PIT –**

1. MBR\_ADDR
2. MBR\_COB
3. MBR\_DETLS
4. MBR\_MBRSHP
5. MBR\_MBRSHP\_EVNT
6. MBR\_MCAID
7. MBR\_MCARE
8. MBRSHP\_MCARE\_SPMTL\_PRT
9. MBRSHP\_PROV\_SEL
10. COVRG\_AGRMT\_DTLS
11. MBRSHP\_PRDCR\_ASGNMT

There will be 9 history tables corresponding to all these above 9 Gold tables and will be available in the database GOLD\_MEMBERSHIP\_HISTORY. The PIT history table’s name are as follows –

1. MBR\_ADDR
2. MBR\_COB
3. MBR\_DETLS
4. MBR\_MBRSHP
5. MBR\_MBRSHP\_EVNT
6. MBR\_MCAID
7. MBR\_MCARE
8. MBRSHP\_MCARE\_SPMTL\_PRT
9. MBRSHP\_PROV\_SEL
10. COVRG\_AGRMT\_DTLS
11. MBRSHP\_PRDCR\_ASGNMT

The history loads are introduced late in the data lake. When we are getting the updated data, the **point in time history table** keeps the history of the data along with latest record. Gold tables only contain all the latest snapshots.

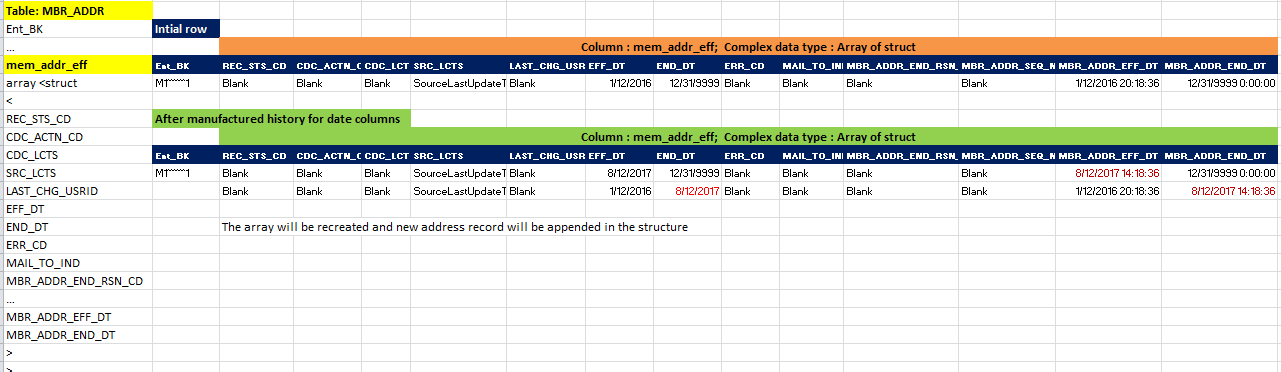
The features of point in time history:

1. Data is present in Temp table.
2. Data is loaded into Gold table as per the current process
3. Take data from Temp table and insert into the Gold PIT table. It is appending data into this table unlike upsert we are doing in Gold table.
4. Create one extra column Record Effective Date ( REC-EFF\_DT) which is nothing but a system time in this table.

### 6.4.2.6 TMG Manufactured History

All gold tables are having enterprise\_business\_key which contains the concatenated values of the primary key columns of the table. For example, MBR\_ADDR will have concatenated key’s as below (may be changed over time) –

ACCT\_NBR^CORP\_ENT\_CD^COVRG\_AGRMT\_ID^SRC\_PROD\_COVRG\_EFF\_DT^PROD\_TYP\_CD^SRC\_PROD\_NBR^MEM\_ADDR\_EFF

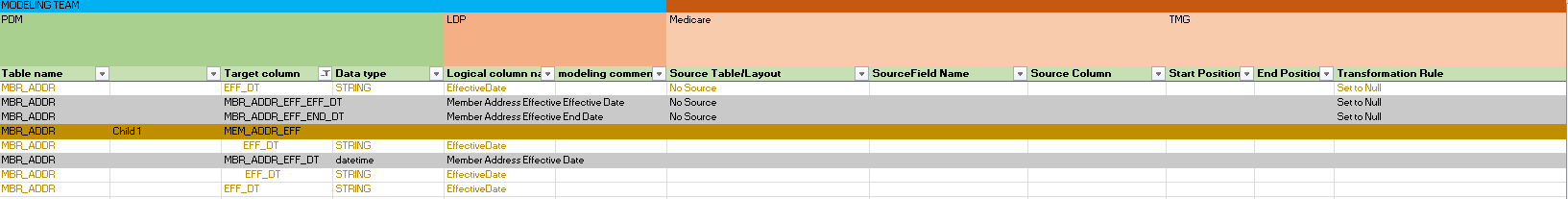


**Step-1:** Identifying for which target tables, history tables need to be generated.

For a given target table we need to figure out business effective dates for which there is no source provided in the mapping document or not at all mapped in the mapping document.

For example:

Below is the mbr\_addr target table having no source columns in the Medicare source. As it is “set to null” we need manufacture the data.



**Step-2:** How to insert the data into history table

For a given business key there may be multiple records in a single file, we will group by the records under a specific business keys and assign sequence number to get the latest record. We pick the latest record for a given business key - assigning rank over partition by business key, ordering record sequence number in descending order.

Step-3: Please find the below sample query for latest and greatest records into the gold conformed format table.

SELECT

MA.ACCT\_NBR

MA.CORP\_ENT\_CD

MA.SUB\_ID

MA.MAILG\_CTY\_NM,

MA.PERM\_CNTY\_NM,

MA.sub\_email\_addr\_txt,

MA.MAILG\_PHN\_NBR ,

MA.MAILG\_STR\_ADDR\_1\_TXT,

MA.MAILG\_STR\_ADDR\_2\_TXT,

MA.MAILG\_STR\_ADDR\_3\_TXT

MA.ST\_CD,

MA.MAILG\_ZIP\_CD,

MA.SAFE\_ADDR\_IND,

MA.MAILG\_ST\_CD,

FROM

(SELECT

MAD.ACCT\_NBR

MAD.CORP\_ENT\_CD

MAD.SUB\_ID

MAD.MAILG\_CTY\_NM,

MAD.PERM\_CNTY\_NM,

MAD.sub\_email\_addr\_txt,

MAD.MAILG\_PHN\_NBR,

MAD.MAILG\_STR\_ADDR\_1\_TXT,

MAD.MAILG\_STR\_ADDR\_2\_TXT,

MAD.MAILG\_STR\_ADDR\_3\_TXT

MAD.ST\_CD,

MAD.MAILG\_ZIP\_CD,

MAD.SAFE\_ADDR\_IND,

MAD.MAILG\_ST\_CD,

rank () over (partition BY MAD.ACCT\_NBR, MAD.CORP\_ENT\_CD, MAD.SUB\_ID order by MAD.dy\_actvy\_file\_curr\_mbrshp\_id desc)

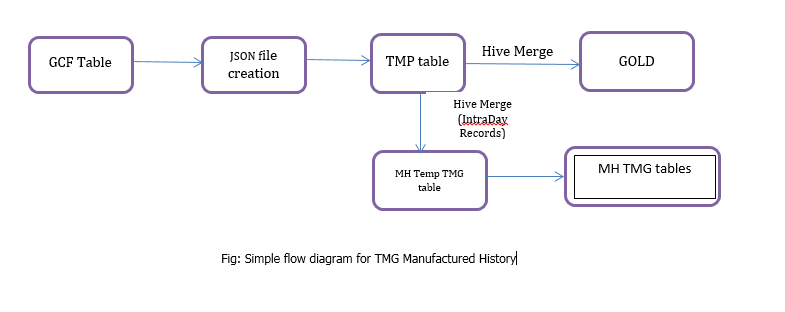
as rank FROM tmg\_merge.dy\_actvy\_file\_mbrshp\_medicare MAD) MA INNER JOIN tmg\_merge.mbr\_mcaid\_tmg\_cdckeys MCEK on MCEK.enterprise\_business\_key = CONCAT\_WS('^', trim(ACCT\_NBR),trim(CORP\_ENT\_CD),trim(COVRG\_AGRMT\_ID),trim(SRC\_PROD\_COVRG\_EFF\_DT),trim(PROD\_TYP\_CD),trim(SRC\_PROD\_NBR),trim(MEM\_ADDR\_EFF));

For any input table which has Date column value as N/A or NULL from source, populate those Date fields with Current Date and insert the record in the table.

In future, for the same Date column, if we get value from source, update that column with the value from Source (SCD Type 2) and insert as new record with Effective Date as current Load date.

Below TMG target tables are considered for this scenario and Manufactured History record will be created for the same for the columns for which the Mapping has NULL values or marked as N/A.

1. MBR\_ADDR
2. MBR\_MBRSHP
3. MBR\_DETLS



**MH - Manufacturing History Steps**

1) GOLD\_MEMBERSHIP\_WORK.MH\_TMG\_<table\_name> is target facing persistent table (**without any complex datatypes like struct**)

2) We need to create table GOLD\_MEMBERSHIP\_WORK.MH\_TMG\_<table\_name> - table which is going to hold only enterprise business key eff\_dt, end\_dt and other respective columns which are present in the struct table

(ex enterprise\_business\_key, eff\_dt\_end\_dt, c1,c2,c3....)

3) For each intra-day loads, data from work table will be appended GOLD\_MEMBERSHIP\_WORK.MH\_TMG\_<table\_name> table.

4) We retrieve (pull) data from Manufactured History temp table which is getting populated in the main gold table creation phase after gcf table and push it into MH\_<<table\_name>>. When inserting data into GOLD\_MEMBERSHIP\_WORK.MH\_TMG\_<table\_name>, eff\_dt and end\_dt will get populated with etl timestamps.

5) From CDCkeys table we will know what are all the keys got changed and all those keys are present in the MH\_<<table\_name>> table

We are doing search using Scala through Enterprise BK along with eff\_dt and retrieve the latest record out of it.

6) MH table will have enterprise business key, other columns available in struct and any otehr columns if mentioned in mapping document which needs to be loaded with some values

7) Eff Dt will be loaded with current processing timestamp

8) End data will be loaded through high date (12/31/9999)

9) Gold target table needs to be rebuild each time. If it has 3 records earlier and another record arrives, the struct will have 4 records

10) Currently there are no "I" or "U" flags available in source data but data ingestion process actually identifying the upsert candidates. Please populate CDC\_ACTN\_CD accordingly. This will be applicable general starting from Merge layer. For new records it needs to be "I" and for existing records it needs to be "U"

**MH Table**

BK1 01/01/2018 12/31/9999 c1, c2, c3

BK1 02/01/2018 12/31/9999 c1, c2, c3 🡺 New Row

**Gold Table with Struct data structure**

BK1 01/01/2018 12/31/9999 c1, c2, c3

Search: Search by BK1 [Enterprise business key].

Hive merge is limited to BK1.

Scala code is going to build next array <01/01/2018 12/31/9999 c1, c2, c3>

We are going to search BK1 and new record and see whether we get any match in the struct. As, there is no match we have to do insert. If there is a match, then its update scenario and we update old record and append new record through scala.

BK1 01/01/2018 12/31/9999 c1, c2, c3

BK1 02/01/2018 12/31/9999 c1, c2, c3 => This is New Struct

BK1 01/01/2018 02/01/2018 c1, c2, c3

Search this date for update. Update today’s date for end\_dt for this record.

Insert a new record with today’s date for eff\_dt and High Date(12/31/9999) as end\_dt

### 6.4.2.7 Day 0 load

Requirements for keeping data in **Day 0 load** varies by different types of file. As per our requirements -

* IL Medicaid-2014 Onwards
* NM Medicaid-2013 Onwards
* TX Medicaid -2015 Onwards
* MT Medicaid-2016 Onwards
* Medicare data-Available Data Starting 2012 Onwards
* MT Medsupp-2016 Onwards
* HCSC Supplemental Medicare Events-2017 Onwards
* POA-2016 Onwards

Since all these files are *supposed* to be same in structure as the current files in Medicare, Medicaid, MT Medsupp and HCSC Supplemental files – please follow the same process as laid down in **6.4.2.1.**

## 6.4.3 Format Standardization

The source files are Flat Files with pipe delimiter. Format standardization is not required in this case.

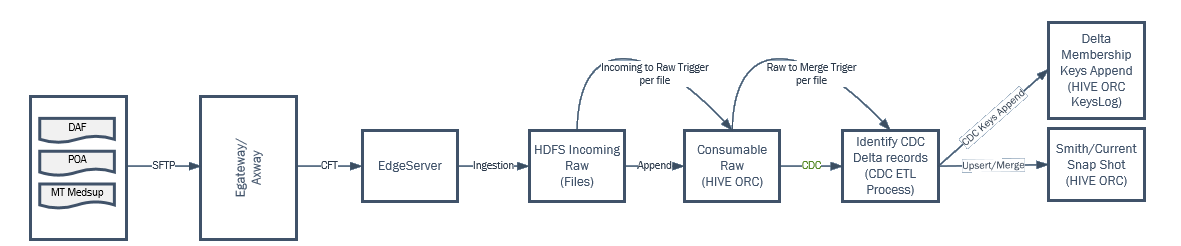
## 6.4.4 Full Loads

For all types of files full loads are to be processed as per the frequency given in section 6.3.2.1. As the files are available in Incoming HDFS RAW, data has to be loaded into consumable RAW (Hive ORC). Identify CDC delta records. The keys that are used in CDC process to identify delta records are provided in 6.3.2.2 section.

For Full load, we will have more than one Trigger File in the location : /datalakedata/prod/gold/integration/params/tmg/ConsRawTrigger/\*

In Raw hive table files are appended and merged hive table upsert mechanism and logging keys of CDC delta to membership key table.

The data flow diagram is as given below fir existing process:



## 6.4.5 Incremental Loads

Incremental File Processing will be done on a daily basis and based on the file date of the ingestion file where the file date is the same as the CURRENT DATE. All the previous files would be moved to an archive location.

The procedure for incremental loads is explained in detail in the **6.3.2 Process Overview**.

## 6.4.6 Low level Steps

The following steps are required to load Medicaid, Medicare & MT Medsup across data pipeline. Update the parameter files & paths with the respective details and tables names while loading different tables.

Step 1

Read the data from source file and create external Hive table on HDFS in orc format.

Step 2

Creating Current layer table

sh -x /datalakedata/prod/gold/integration/work/membership/tmg/src/tmg\_create\_currentDB\_tables.sh /datalakedata/prod/gold/integration/params/membership/tmg/paramFiles.prm

Step 3

Creating CDC  table and Merge Statements

sh -x /datalakedata/prod/gold/integration/work/membership/tmg/src/tmg\_create\_CDCTables\_and\_hiveMergeStmts.sh /datalakedata/prod/gold/integration/params/membership/tmg/paramFiles.prm

Step 4

Executing merge statements and Alter cdc location.

sh -x /datalakedata/prod/gold/integration/work/membership/tmg/src/bluestar\_hiveMerge\_CDCchanges\_withCurrentDB.sh /datalakedata/prod/gold/integration/params/membership/tmg/paramFiles.prm /datalakedata/prod/gold/integration/params/membership/tmg/MBR\_ADDR.prm /datalakedata/dev/gold/integration/xferin/membership/tmg/cdctrigger/processed/MBR\_ADDR-tmg\_cdcfile\_20180517\_initial.trg

Step 5

Kafka write job :

sh -x  /datalakedata/prod/gold/integration/src/scripts/membership/tmg/tmg\_common\_find\_CDCKeyChanges\_writeToKafkaTopic/bluestar\_common\_find\_CDCKeyChanges\_writeToKafkaTopic/tmg\_common\_find\_CDCKeyChanges\_writeToKafkaTopic\_run.sh  --context\_param paramfile="/datalakedata/prod/gold/integration/params/membership/tmg/paramFiles.prm" --context\_param datalake\_tablename="/datalakedata/prod/gold/integration/params/membership/tmg/MBR\_ADDR.prm" /datalakedata/prod/gold/integration/xferin/membership/tmg/cdctrigger/processed/MBR\_ADDR-tmg\_cdcfile\_20180517\_initial.trg

Step 6

Kafka Read job:

sh -x /datalakedata/dev/gold/integration/work/membership/tmg/src/tmg\_common\_find\_workload\_readFromKafkaTopic\_0.1/tmg\_common\_find\_workload\_readFromKafkaTopic/tmg\_common\_find\_workload\_readFromKafkaTopic\_run.sh --context\_param paramfile="/datalakedata/prod/gold/integration/params/membership/tmg/paramFiles.prm" --context\_param datalake\_tablename="/datalakedata/prod/gold/integration/params/membership/tmg/MBR\_ADDR.prm"

Step 7

GCF Load:

sh -x /datalakedata/prod/gold/integration/src/scripts/membership/enterprise/common\_tmg\_format\_GCF\_0.1/common\_tmg\_format\_GCF/common\_bluestar\_format\_GCF\_run.sh --context\_param paramfile="/datalakedata/prod/gold/integration/params/membership/tmg/paramFiles.prm" --context\_param datalake\_tablename="/datalakedata/prod/gold/integration/params/membership/tmg/MBR\_ADDR.prm" /datalakedata/prod/gold/integration/xferin/membership/tmg/gcftrigger/processed/MBR\_ADDR-2018-05-17\_041905.trg

Step 8

Temp table will be created by applying necessary derivations on GCF table.

Step 9

Use hqls to merge data into Gold layer.

Assuming that all type of files will be placed in HDFS RAW layer without any sub-directories. Based on file pattern / type pattern, copy / move the files into respective partitioned directories with ORC serialization except HSCS Supp files. (<Table>/<dt=MMYYYY>/files\*).

**Steps for loading HCSC Supp type of files:**

Step 1

Since these are multi record files (semi-structured), not creating Hive tables on top of this data. Instead, copy them into (<Table>/<dt=MMYYYY>/files\*).

Step 2

Use spark (scala) code to create Json format tables

The code for supp file will be in this manner:

*val rdd= sc.wholeTextFile(“~/supp/dt=mmyyyy/\*”)*

RDD is a tuple (combination) of filename and list of lines. (RDD=(file1.txt, list (line1, line2…)))

*Rdd\_lines=rdd.map(e=> e.\_2)*

Create a broadcast variable for type (CRCO, DABL etc.) along with its schema. Use these broadcast variables in functions to create a Json.

Write a function which takes list of lines and returns a Json string.

Def fun(list(lines)): string=

{

// make a Json for each supp file chunk using the broadcast variables.

}

//use function

Rddlist.map (lines => fun(lines)).flatMap

Curr table is in Json format. Merge table is in ORC format.

Eg. {CRCO: [{C1:V1,C2:V2}], DABL: [{C1:V1,C2:V2}]}

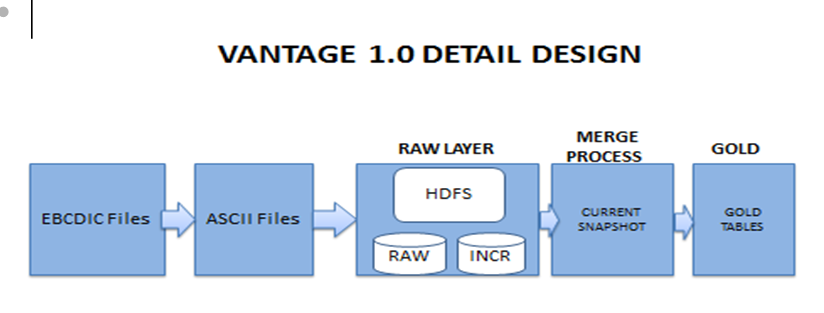
Step 3

Load data into CDC tables in the same way as explained in Step 2.

Step 4

Follow the same steps as for the Medicare files (step 3 onwards) to complete loading and merging process to merge layer and gold layer.

## 6.5 VANTAGE 1.0

The DFD for VANTAGE 1.0 is illustrated in the below diagram

## 6.5.1 Tech Stack

### 6.5.1.1 Hive

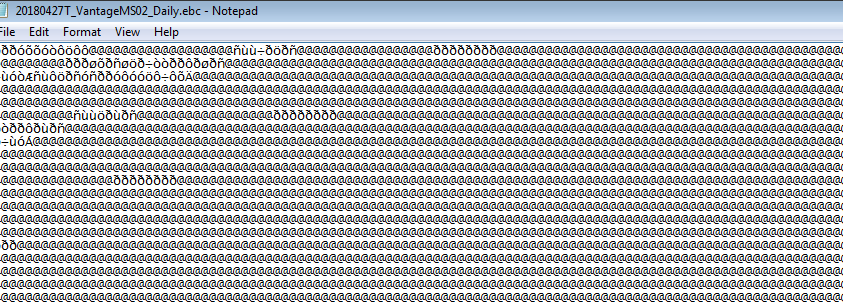
* Hive is used to store data from ASCII files in the RAW layer as well as merge & Gold layers. We are using external tables in RAW layer and internal tables in merge & gold layers.

### 6.5.1.2 Spark

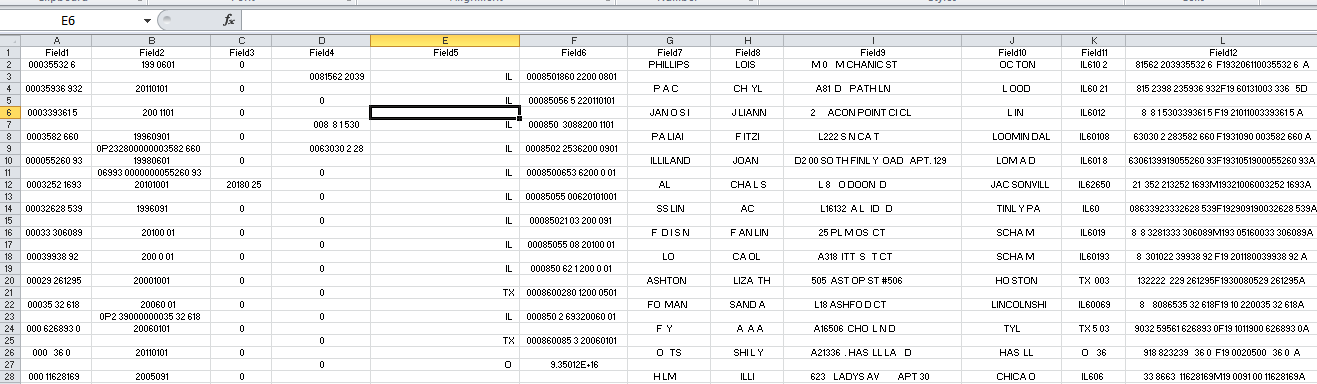
Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams.

**EBCDIC to ASCII**

Raw data comes in EBCDIC format and should be converted into ASCII because EBCDIC is unreadable whereas ASCII is readable. It is converted through JAVA program. Sample picture of EBCDIC format is below:



**Sample picture of ASCII format is below**:



Sometimes raw data comes in EBCDIC with packed decimals and it is also converted into ASCII.The JAVA program is written in such a format which converts any type of EBCDIC file.

## 6.5.2 Process Overview

### 6.5.2.1 Ingestion

Source system data will be ingested to HDFS. The Vantage files initially land on the edge server as EBCIDIC file and those files are ingested into the HDFS locations below…

/prod/incoming/raw/Membership/Vantage/MedSupp/EBCIDIC/MS02/\*/\*/\*/(‘\*\_VantageMS02\_Daily.txt’ or \*\_VantageMS02\_Weekly.txt’)

/prod/incoming/raw/Membership/Vantage/MedSupp/EBCIDIC/MS05/\*/\*/\*/(‘\*\_VantageMS05\_Daily.txt’ or \*\_VantageMS05\_Weekly.txt’)

Then there is another conversion process that will automatically trigger directly after that will convert the files to ASCII and then ingest them into the following HDFS dirs..

/prod/incoming/raw/Membership/Vantage/MedSupp/ASCII/MS02/\*/\*/\*/ (‘\*\_VantageMS02\_Weekly\_Ascii.txt’ or ‘\*\_VantageMS02\_Daily\_Ascii.txt’)

/prod/incoming/raw/Membership/Vantage/MedSupp/ASCII/MS05/\*/\*/\*/ (‘\*\_VantageMS05\_Weekly\_Ascii.txt’ or ‘\*\_VantageMS05\_Daily\_Ascii.txt’)

The daily files are loaded into the ‘\*\_curr\_\*’ tables only. But once a weekly file (which contains all history) is received, then all data in the ‘\*\_curr\_\*’ tables are deleted and the data from the weekly file is loaded into the ‘\*\_curr\_\*’ table and also inserted into the ‘\*weekly\*’ table.

The pig scripts are located in /datalake/Prod/Scripts/PIG/GPD folder in test and prod. The data in dev may not be in sync.

The file is a mainframe file in EBCIDIC format and is not readable. It is converted into ASCII in order to get it in a readable format and loaded it into hive.

**Hive RAW Table:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Source FF name | Corresponding Hive table | Hive table properties | | | | |
| Partitioned | Bucketed (#) | External | Format | Transactional |
| gpd1006\_vms\_ms02\_daily\_work\_load | tmg\_raw\_mbrshp.dmds\_ms02\_curr\_mbrshp | Yes - | 5 | Yes | ORC (snappy) | FALSE |
|
| gpd1006\_vms\_ms05\_daily\_work\_load | tmg\_raw\_mbrshp.dmds\_ms05\_curr\_covrg | Yes- | 10 | Yes | ORC (snappy) | FALSE |
|

The ASCII files are then loaded into hive into the following tables.

* tmg\_raw\_mbrshp.dmds\_ms02\_curr\_mbrshp
* tmg\_raw\_mbrshp.dmds\_ms02\_hist\_mbrshp
* tmg\_raw\_mbrshp.dmds\_ms05\_curr\_covrg
* tmg\_raw\_mbrshp .dmds\_ms05\_hist\_covrg

**Hive Current Table:**

All the incremental files would be placed HDFS. The data would be further written into Kafka topic. This data also appended to merge table.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Incremental FF name | Corresponding Hive table | Hive table properties | | | | |
| Partitioned | Bucketed (#) | External | Format | Transactional |
| gpd1006\_vms\_ms02\_daily\_work\_load | tmg\_raw\_mbrshp.dmds\_ms02\_curr\_mbrshp | Yes - | No | No | TEXT | FALSE |
|
| gpd1006\_vms\_ms05\_daily\_work\_load | tmg\_raw\_mbrshp.dmds\_ms05\_curr\_covrg | Yes- | No | No | TEXT | FALSE |
|

### 6.5.2.2 Merging

First time, by comparing the records from RAW table and Incremental table, we will filter the records for merging and stored into the merging table All the *latest and greatest* records would be merged and stored in this table.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Raw table name | Corresponding Hive table | Hive table properties | | | | |
| Partitioned | Bucketed (#) | Internal | Format | Transactional |
| gpd1006\_vms\_ms02\_daily\_work\_load | tmg\_raw\_mbrshp.dmds\_ms02\_curr\_mbrshp | Yes - | No | No | ORC SNAPPY | TRUE |
|
| gpd1006\_vms\_ms05\_daily\_work\_load | tmg\_raw\_mbrshp.dmds\_ms05\_curr\_covrg | Yes- | No | No | ORC SNAPPY | TRUE |
|

### 6.5.2.3 Gold

We will apply required quality checks and then loading into the final Gold table. Tables are yet to be given by Mapper.

| Target table name | Relevant Source Tables | **Hive table properties** | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| Partitioned | Bucketed (#) | External | Format | Transactional |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

**MID XREF Document location**

From Arch perspective, it has been depicted in the diagram.

<https://myfyi.sharepoint.com/:u:/r/teams/Architecture/Architecture%20Documents/CertifiedLayer.vsd?d=wdcee7fffb4814628af4bebeed4c2a809&csf=1>  (High Level Membership tab).

MID assignment happens in IBM MDM component. That's the core capability of IBM MDM.  We are ingesting MID XREF from DMIH into Data Lake.  For those use cases where there is a need to tie all the memberships across the LOBs, that can join with this table and identify as the same person.  That's the basic notion of any MDM design.

## 6.5.3 Format Standardization

The source files are EBCIDIC and converted into ASCII. Format standardization is not required in this case.

## 6.5.4 Full Loads

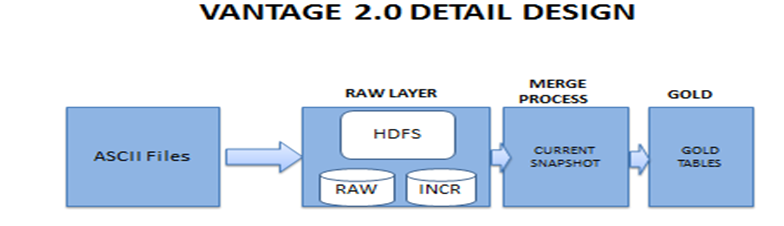
Full loads are part of current Vantage 1.0 design.

## 6.5.5 Incremental Loads

Incremental File Processing will be done on a daily basis and based on the file date of the ingestion file where the file date is the same as the CURRENT DATE. All the previous files would be moved to an archive location.

## 6.6 VANTAGE 2.0

The DFD for VANTAGE 2.0 is illustrated in the below diagram.



The Vantage files initially land on the edge server as ASCII files and those files are ingested into the HDFS locations.

## 6.6.1 Tech Stack

### 6.6.1.1 Hive

* Hive is used to store data from ASCII files in the RAW layer as well as merge & Gold layers. We are using external tables in RAW layer and internal tables in merge & gold layers.

### 6.6.1.2 Spark

Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of data streams.

## 6.6.2 Process Overview

## 6.6.2.1 Ingestion

Source system data will be ingested to HDFS. Directory Structure details are available in the following table:

| **SL No** | **Element Description** | **Value** | **Comments** |
| --- | --- | --- | --- |
| 1 | ScriptsPath | /datalakebin/prod/gold/ingestion/src/scripts/ingestVms/ | Scripts will be available in this path |
| 2 | ParamFilePath | /datalakebin/prod/gold/ingestion/params/ingestVms/ | Param files will be available in this path |
| 3 | LogPath | /datalakedata/prod/gold/ingestion/logs/ingestVms/ |  |
| 4 | WorkPath | /datalakedata/prod/gold/ingestion/work/ingestVms/ |  |
| 5 | WorkInPath | /datalakedata/prod/gold/ingestion/work/ingestVms/Incoming/ |  |
| 6 | WorkOutPath | /datalakedata/prod/gold/ingestion/work/ingestVms/Outgoing/ |  |
| 7 | HDFSWorkPath | /prod/work/raw/Membership/VMS/ |  |
| 8 | AuditHDFSWorkPath | /prod/work/raw/Membership/VMS/ACTIVE\_AUDIT/ |  |
| 9 | GoldConsTriggerFilePath | /datalakedata/prod/gold/integration/work/Vms/trigger/ |  |
|  |  |  |  |
|  |  |  |  |

Red fonts are referring to environment. It may be either Dev or Test or Prod.

Sample data is available on

https://myfyi.sharepoint.com/teams/projects/P00022106/Working%20Documents/Forms/AllItems.aspx?id=%2Fteams%2Fprojects%2FP00022106%2FWorking%20Documents%2FIngestion%2FMembership%2FVantage%202%2E0%2FRequirements%2FReference%2FSample%20VMS2%2E0%20Data-As%20of%20Aug%2Ezip&parent=%2Fteams%2Fprojects%2FP00022106%2FWorking%20Documents%2FIngestion%2FMembership%2FVantage%202%2E0%2FRequirements%2FReference

**HDFS Folder details**

/prod/incoming/raw/Membership/VMS

These ASCII files are ingested into the HDFS locations. Full files come under weekly whereas incremental files come under daily. All the VMS Member(MS02) and Coverage(MS05) data files will be ingested under following HDFS paths: \* HDFS Directory Path: /<HDFSENV>/incoming/raw/Membership/VMS \* HDFSENV: dev, test, prod. HDFS partitions will use the ingestion date and example is below:

PARTITIONED BY (

`inc\_ful\_flag` string,

`file\_date` string )

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ''

LINES TERMINATED BY '\n'

STORED AS INPUTFORMAT

'org.apache.hadoop.mapred.TextInputFormat'

OUTPUTFORMAT

'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'

LOCATION

'hdfs://TSTHDPHA/test/incoming/raw/Membership/VMS/VMS2\_MS05'

All VMS Member(MS02) and Coverage(MS05) Data files can be access from following Hive DB and Table HiveDB\_Name = VMS\_RAW

HiveTable\_Name = VM2\_MS02, VMS2\_MS05.

**Hive RAW Table:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Source FF name | Corresponding Hive table | Hive table properties | | | | |
| Partitioned   |  |  | | --- | --- | | Daily |  | | Bucketed (#) | External | Format | Transactional |
| vms2\_ms02 | vms2\_ms02 | Yes | 5 | Yes | ORC (snappy) | FALSE |
|
| vms2\_ms05 | vms2\_ms05 | Yes | 10 | Yes | ORC (snappy) | FALSE |
|

The ASCII files are then loaded into hive into the following tables.

**Hive Current Table:**

All the incremental files would be placed HDFS. This data also appended to merge table.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Incremental FF name | Corresponding Hive table | Hive table properties | | | | |
| Partitioned | Bucketed (#) | External | Format | Transactional |
| MS02\_VMS2\_INCR\_<YYMMDDHHMMSS>.FTP | vms2\_ms02 | Yes - | No | No | TEXT | FALSE |
|
| MS05\_VMS2\_INCR\_<YYMMDDHHMMSS>.FTP | vms2\_ms05 | Yes- | No | No | TEXT | FALSE |
|

## 6.6.2.2 Merging

First time, by comparing the records from RAW table and Incremental table, we will filter the records for merging and stored into the merging table All the *latest and greatest* records would be merged and stored in this table.

Hive Merge helps in updating the data from curr tables data to weekly file.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Raw table name | Corresponding Hive table | Hive table properties | | | | |
| Partitioned | Bucketed (#) | Internal | Format | Transactional |
| vms2\_ms02 | vms2\_ms02 | Yes - | No | No | ORC SNAPPY | TRUE |
|
| vms2\_ms05 | vms2\_ms05 | Yes- | No | No | ORC SNAPPY | TRUE |
|

## 6.6.2.3 Gold

We will apply required quality checks and then loading into the final Gold table. Tables are yet to be finalized by Mapper.

| Target table name | Relevant Source Tables | **Hive table properties** | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| Partitioned | Bucketed (#) | External | Format | Transactional |
| MBR\_ADDR | vms2\_ms02  vms2\_ms05 | Yes - | 5 | No | ORC Snappy | True |
| MBR\_DETLS | vms2\_ms02  vms2\_ms05 | Yes- | 10 | No | ORC Snappy | True |
| MBR\_MBRSHP\_EVNT | vms2\_ms02  vms2\_ms05 | Yes- | 5 | No | ORC Snappy | True |
| MBR\_MCARE | vms2\_ms02  vms2\_ms05 | Yes- | 10 | No | ORC Snappy | True |
| MBR\_MCARE\_SPMTL\_PRT | vms2\_ms02  vms2\_ms05 | Yes- | 5 | No | ORC Snappy | True |

This also gives flexibility to apply Data Quality & Transformation rules if any in future.

## 6.6.3 Format Standardization

The source files are ASCII and loaded into hive .

## 6.6.4 Full Loads

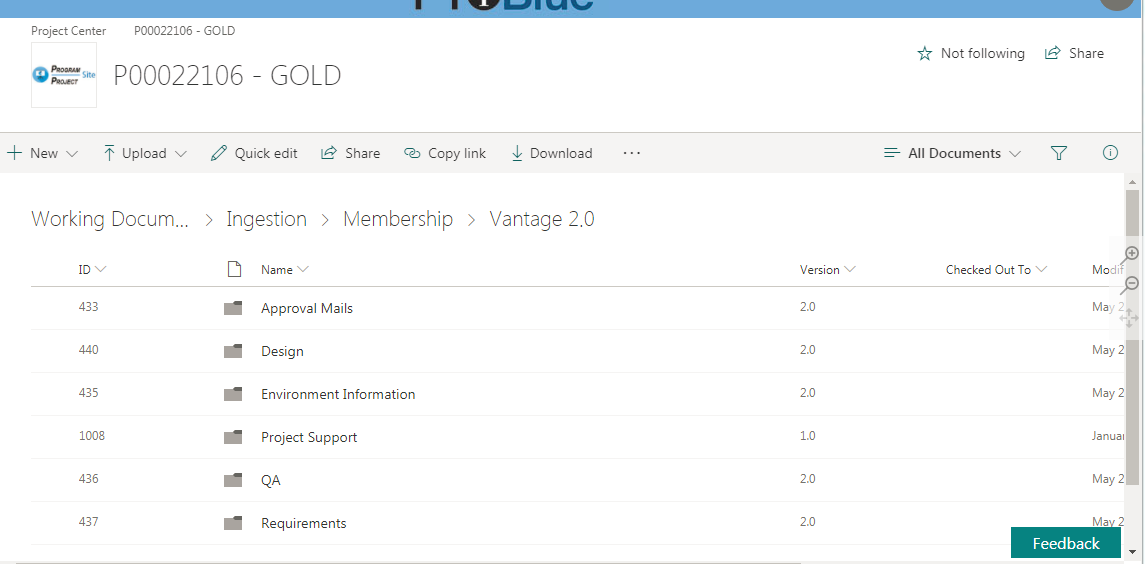
Full loads are part of current Vantage 2.0 design.

## 6.6.5 Incremental Loads

Incremental File Processing will be done on a daily basis and based on the file date of the ingestion file where the file date is the same as the CURRENT DATE.

**Share point Details:**

<https://myfyi.sharepoint.com/teams/projects/P00022106/Working%20Documents/Forms/AllItems.aspx>



## 6.7 System80

## 6.8 GOLD to REH

# 7. Subject Areas

There are 3 subject areas considered for Data Accelerator project.

* + Membership
* Account (Employer)
* Source Product (Plans)
* Member
* Coverage Group (Benefits)
* Coverage Agreement (benefits Agreement)
  + Provider
  + Claims
  + Clinical (EMR) -> Electronic Medical records)

Currently we are detailing about Membership domain which consists of 14 data sources. Each data source is grouped into one of the line of business as mentioned in 2.1.1.

## 7.1 Membership Data Model and Standards

### 7.1.1 Membership Data Group Classification

Current Membership data is classified into 3 lines of business

* Groups
* Government Programs
* Retail

### 7.1.2 Source Data Application System (Files) And Contents

For Group, HCSC uses Bluestar Application top store all Current Membership system files. All member data is stores in 53 tables. Also, there are additional files which contain additional information related to membership which are student Health and ingestion for additional tables required for salesforce, FSD 20/20, ACA, ECR, Stop Loss, HAS, Commissions PCPM and MEH (Membership Eligibility Hub) etc.

* For Government program, HCSC uses TMG data files.
* For Retail, TMG data files are used as well as RMS. There are total 44 files- On and Off Exchange for HPS but mapping refers only to 9 fixed-width files
* For all LOB, HCSC uses MID (MID View), BluePCS (XML) and EPP (EPP File layout).



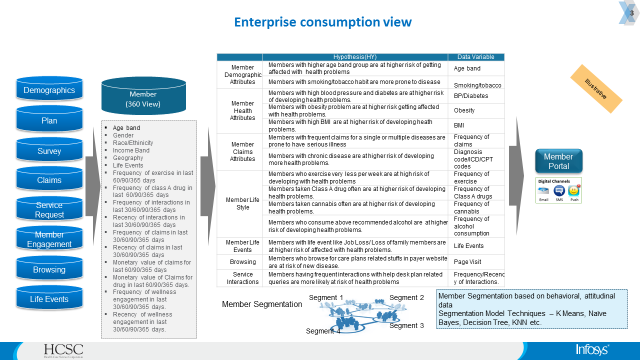
## 7.2 File Types

There are different types of file formats to be loaded into the system. Here are some of the file formats used in the project.

* Flat files with headers
* Flat files without headers
* XML files
* MQ
* Cobol Copybooks
* Json

# 8.Enterprise Consumption Views (ECV)

We can create enterprise consumption views on the Gold data. A member 360 view contains different plans, claims, member engagement etc. We can view based on Age, Gender. Frequency of exercise etc. We can segement member data based on behavioral data and attitudinal data.



# 9. Environment and Standards

## 9.1 Incoming Files/Sources

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Source | Feed Type | File Name | File Format | Frequency | Incr vs Full |
| DB2 | Membership – Bluestar | No source files since getting data into RAW database. /datalakedata/prod/gold/integration/src/scripts/membership/bluestar/  bluestar\_create\_currentDB\_tables.sh | Hive | One time | FUll |
| IIDR | Membership - Bluestar | No source files since getting data into RAW database. But reference purpose, given is the name of shell script file to run common\_bluestar\_format\_GCF\_run.sh | Hive | Daily | Incremental |
| Flat Files (WH) | Membership - HPS | HPSYYMMDDHHMM.txt, HPSmessages\_25613\_10.136.136.39\_37\_text | Flat File | Daily | Incremental |
| Flat Files (WTH) | Membership - HPS | HPSYYMMDDHHMM.pgp | Flat File |  | Incremental |
| IBM MQ (XML) | Membership - RMS | HBEN392\_VACYYYYMMDDHHMMSS.txt | MQ | NRT (5 min) | Streaming |
| DAF-Medicare | Membership - TMG | MBS\_MMA\_Daily\_Subscriber\_Activity\_File\_20171211\_191451.txt.pgp.txt | Flat File | Daily / Weekly / Monthly | Full / Incremental |
| DAF-Medicaid | Membership - TMG | NICE\_MU\_TYP\_REF\_YYYY\_MM\_DD.csv | Flat File | Daily / Weekly / Monthly | Full/ Incremental |
| POA | Membership - TMG | NICE\_QUE\_ID\_REF\_YYYY\_MM\_DD.csv | Flat File | Daily | Incremental |
| MT Medsupp | Membership - TMG | NICE\_JOB\_CAT\_REF\_YYYY\_MM\_DD.csv | Flat File | Daily | Full/ Incremental |
| HCSC-Supp | Membership - TMG | NICE\_GT\_LT\_EQ\_REF\_YYYY\_MM\_DD.csv | Flat File | Monthly | Full |

## 

## 9.2 Databases/tables

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Source | Database type | Database Layer | Database Name | Table Name | Table Type | PartitionsYN | File Format |
| DB2 / IIDR | Hive | Incoming | Bstar\_RAW | Mem | External | Yes | RDBMS |
| DB2 / IIDR | Hive | Merge | Bstar\_Curr | Mem | Internal | Yes | RDBMS |
| DB2 / IIDR | Hive | Gold | Bstar\_Gold | Mem | Internal | No | RDBMS |
| Flat Files (WH / WTH) | Hive | Incoming | HPS\_RAW | Case\_Master | External | Yes | FLAT |
| Flat Files (WH / WTH) | Hive | Merge | HPS\_Merge | Case\_Master | Internal | Yes | FLAT |
| Flat Files (WH / WTH) | Hive | Gold | HPS\_Gold | Case\_Master | Internal | Yes | FLAT |
| IBM MQ (XML) | Hive | Incoming | RMS\_RAW | Mem\_Detl | External | Yes | MQ |
| IBM MQ (XML) | Hive | Gold | RMS\_Gold | Mem\_Detl | Internal | Yes | MQ |

## 9.3 Hadoop

|  |  |  |  |
| --- | --- | --- | --- |
| Environment | Hostname | IP Address |  |
| Dev | dazrslenap0001.app.dev.hcscint.net | 10.233.37.144 |  |
| Test | twauslaenapp03.app.test.hcscint.net | 10.134.136.78 |  |
| Prod | pwauslaenapp03.app.hcscint.net | **10.136.136.37** |  |

## 

## 9.4 Yarn Queues

This ingestion process does not need any queue as there are no MR/Tez jobs.

## 9.5 Webroles/NPID

|  |  |  |  |
| --- | --- | --- | --- |
| Source | ETL Web role | NPID | Environments |
| DSO | HDOF hdp ETL dev Azure | hdcorp01 | Dev Azure |
| DSO | FWID\_Jenkins\_Users | hdcorp01 | Dev Azure |
| DSO | HDOF hdp ETL audt frwk\_DEV | hdcorp01 | Dev Azure |
| DSO | Jira\_Users | hdcorp01 | Dev Azure |
| DSO | GitHub\_User\_Access | hdcorp01 | Dev Azure |
| DSO | Hadoop\_user\_dev | hdcorp01 | Dev Azure |

## 

## 9.6 HDFS Paths

|  |  |  |
| --- | --- | --- |
| Source | Feed Type | Incoming HDFS Paths |
| DB2 | Membership – Bluestar | /datalakedata/prod/gold/integration/src/scripts/membership/bluestar |
| ITDR | Membership - Bluestar | /datalakedata/prod/gold/integration/src/scripts/membership/bluestar |
| Flat Files (WH) | Membership - HPS | To be Updated |
| Flat Files (WTH) | Membership - HPS | To be Updated |
| IBM MQ (XML) | Membership - RMS | To be Updated |
|  | Membership - TMG | /prod/incoming/raw/Membership/TMG/DAF |
|  | Membership - TMG | /prod/incoming/raw/Membership/TMG/DAF |
|  | Membership - TMG | /prod/incoming/raw/Membership/TMG/POA |
|  | Membership - TMG | /prod/incoming/raw/Membership/TMG/MedSupp/MT/ |
|  | Membership - TMG | /prod/incoming/raw/Membership/TMG/daf\_supp |
| VMS | Membership - Vantage | To be Updated |

## 9.7 App Edge Node Script Paths

Generally it will follow the following path for keeping script files

datalakebin/<**environment**>/gold/ingestion/src/scripts/ingest/<**SourceName**>/

<**environment**> will vary depending on lower / higher environments and <**SourceName**> will vary depending on source.

|  |  |  |
| --- | --- | --- |
| Source | Type | Edge Node Path |
| Bluestar | DB Files | /datalakebin/dev/gold/ingestion/src/scripts/ingest/bluestar/ |
| HPS | Flat File | /datalakebin/dev/gold/ingestion/src/scripts/ingest/hps/<Necessary Subfolders> |
| TMG-DAF-Medicare | Flat File | /datalakebin/dev/gold/ingestion/src/scripts/ingest/tmg/daf-medicare/ |
| TMG-DAF-Medicaid | Flat File | /datalakebin/dev/gold/ingestion/src/scripts/ingest/tmg/daf-medicaid/ |
| TMG-POA | Flat File | /datalakebin/dev/gold/ingestion/src/scripts/ingest/tmg/poa/ |
| TMG-MT Medsupp | Flat File | /datalakebin/dev/gold/ingestion/src/scripts/ingest/tmg/mt-medsupp/ |
| TMG- HCSC-Supp | Flat File (Multi-record) | /datalakebin/dev/gold/ingestion/src/scripts/ingest/hcsc-supp/ |
| System80 | Flat file | /datalakebin/dev/gold/ingestion/src/scripts/ingest/system80/ |
| VMS | ASCII File – 1.0 | /datalakebin/dev/gold/ingestion/src/scripts/ingest/vm1.0/ |
| VMS | Flat File – pipe delimitted | /datalakebin/dev/gold/ingestion/src/scripts/ingest/vm2.0/ |
| RMS | XML Files | /datalakebin/dev/gold/ingestion/src/scripts/ingest/rms/ |
|  |  |  |

## 

## 9.8 Zena

All the Zena process should be Directory Monitor event based triggers and process events should be developed in the below location of Zena Dev environment.

Approach is to create one master resource to control all the number of jobs from current to gold across all membership systems. Please create one resource per target table when we have more than one system loading the same table. Steps -

Create a resource with target table name and quantity as 1. We have to add this table level resource to each CURRENT (MERGE) to gold Zena process.

* Create one master resource to add to all ZENA jobs which load from current to gold tables.
* Create one resource per target table with quantity one and this resource should be added to each of the zena jobs (bstar , tmg , restail , rms)

TIS Datalake\DA

**Zena components naming convention:** Currently we are using only for Bluestar ingestion.

|  |  |  |  |
| --- | --- | --- | --- |
| Source | Feed Type | Process Name | Event Name |
| Bluestar | Shell script | 01\_datalake\_gold\_bluestar\_create\_currentDB\_tables\_process | Create current tables |
| Bluestar | Shell Script | 02\_datalake\_gold\_bluestar\_create\_CDCTables\_and\_hiveMergeStmts\_process | Create CDC tables and merge statements |
| Bluestar | Shell Script, Talend Job. | 1\_2\_3\_datalake\_gold\_bluestar\_hiveMerge\_ writeToKafka\_process | Executing Merge and writing Keys to Kafka |
| Bluestar | Talend Job | 02\_datalake\_gold\_bluestar\_create\_CDCTables\_and\_hiveMergeStmts\_process | Reading Kafka |
| Bluestar | Talend Job | 02\_datalake\_gold\_bluestar\_create\_CDCTables\_and\_hiveMergeStmts\_process | Creating GCF (Gold Conformed Format). |
| Bluestar | Talend Job | 1\_2\_3\_datalake\_gold\_bluestar\_hiveMerge\_ writeToKafka\_process | Creating Enterprise Layer |

Zena Login: **hdcrp01 login for Corporate projects**

GitHub Url: <https://ghe.fyiblue.com/HCSC-Pilot/datalake_gold>

Branch: [ETDL-373591-GOLD](https://ghe.fyiblue.com/HCSC-Pilot/datalake_gold/tree/ETDL-373591-GOLD)

Jenkins Url: <http://10.234.3.241/job/TIS-DATALAKE/job/datalake_gold/job/gold_ingestBluestar_deploy/>

We will be using datalake\_gold git repo for all the gold domains and below are the details of saving source code to Gut repo.

  All the source code will be saved under the below folder structure on GitHub.

- /datalakebin/prod/gold/integration/src/scala/[membership](https://ghe.fyiblue.com/HCSC-Pilot/datalake_gold/tree/ETDL-373591-GOLD/datalakebin/gold/integration/lib/membership)/bluestar/<source code>

 All the Build application jars will be saved under the below folder of edge servers and Git repo as well - /datalakebin/prod/gold/integration/bin/[membership](https://ghe.fyiblue.com/HCSC-Pilot/datalake_gold/tree/ETDL-373591-GOLD/datalakebin/gold/integration/lib/membership)/bluestar/<Applicatonname>.jar

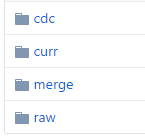
* All the shell script file : /[datalakebin](https://ghe.fyiblue.com/HCSC-Pilot/datalake_gold/tree/ETDL-373591-GOLD/datalakebin)/[gold](https://ghe.fyiblue.com/HCSC-Pilot/datalake_gold/tree/ETDL-373591-GOLD/datalakebin/gold)/[integration](https://ghe.fyiblue.com/HCSC-Pilot/datalake_gold/tree/ETDL-373591-GOLD/datalakebin/gold/integration)/[src](https://ghe.fyiblue.com/HCSC-Pilot/datalake_gold/tree/ETDL-373591-GOLD/datalakebin/gold/integration/src)/[scripts](https://ghe.fyiblue.com/HCSC-Pilot/datalake_gold/tree/ETDL-373591-GOLD/datalakebin/gold/integration/src/scripts)/[membership](https://ghe.fyiblue.com/HCSC-Pilot/datalake_gold/tree/ETDL-373591-GOLD/datalakebin/gold/integration/src/scripts/membership)/bluestar/<shell script>

                Examples :

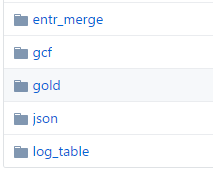
 /datalakebin/prod/gold/integration/bin/[membership](https://ghe.fyiblue.com/HCSC-Pilot/datalake_gold/tree/ETDL-373591-GOLD/datalakebin/gold/integration/bin/membership)/bluestar/

cid:image003.png@01D4125B.8ED2F1A0

* All the jar required for scala code : /[datalakebin](https://ghe.fyiblue.com/HCSC-Pilot/datalake_gold/tree/ETDL-373591-GOLD/datalakebin)/[gold](https://ghe.fyiblue.com/HCSC-Pilot/datalake_gold/tree/ETDL-373591-GOLD/datalakebin/gold)/[integration](https://ghe.fyiblue.com/HCSC-Pilot/datalake_gold/tree/ETDL-373591-GOLD/datalakebin/gold/integration)/[lib](https://ghe.fyiblue.com/HCSC-Pilot/datalake_gold/tree/ETDL-373591-GOLD/datalakebin/gold/integration/lib)/[membership](https://ghe.fyiblue.com/HCSC-Pilot/datalake_gold/tree/ETDL-373591-GOLD/datalakebin/gold/integration/lib/membership)/bluestar/
* Hql: /[datalakedata](https://ghe.fyiblue.com/HCSC-Pilot/datalake_gold/tree/ETDL-373591-GOLD/datalakedata)/[gold](https://ghe.fyiblue.com/HCSC-Pilot/datalake_gold/tree/ETDL-373591-GOLD/datalakedata/gold)/[integration](https://ghe.fyiblue.com/HCSC-Pilot/datalake_gold/tree/ETDL-373591-GOLD/datalakedata/gold/integration)/[hql](https://ghe.fyiblue.com/HCSC-Pilot/datalake_gold/tree/ETDL-373591-GOLD/datalakedata/gold/integration/hql)/[membership](https://ghe.fyiblue.com/HCSC-Pilot/datalake_gold/tree/ETDL-373591-GOLD/datalakedata/gold/integration/hql/membership)/bluestar/runtime/



/[datalakedata](https://ghe.fyiblue.com/HCSC-Pilot/datalake_gold/tree/ETDL-373591-GOLD/datalakedata)/[gold](https://ghe.fyiblue.com/HCSC-Pilot/datalake_gold/tree/ETDL-373591-GOLD/datalakedata/gold)/[integration](https://ghe.fyiblue.com/HCSC-Pilot/datalake_gold/tree/ETDL-373591-GOLD/datalakedata/gold/integration)/[hql](https://ghe.fyiblue.com/HCSC-Pilot/datalake_gold/tree/ETDL-373591-GOLD/datalakedata/gold/integration/hql)/[membership](https://ghe.fyiblue.com/HCSC-Pilot/datalake_gold/tree/ETDL-373591-GOLD/datalakedata/gold/integration/hql/membership)/bluestar/static/



* All the parameter file : /[datalakebin](https://ghe.fyiblue.com/HCSC-Pilot/datalake_gold/tree/ETDL-373591-GOLD/datalakebin)/[gold](https://ghe.fyiblue.com/HCSC-Pilot/datalake_gold/tree/ETDL-373591-GOLD/datalakebin/gold)/[integration](https://ghe.fyiblue.com/HCSC-Pilot/datalake_gold/tree/ETDL-373591-GOLD/datalakebin/gold/integration)/[params](https://ghe.fyiblue.com/HCSC-Pilot/datalake_gold/tree/ETDL-373591-GOLD/datalakebin/gold/integration/params)/[membership](https://ghe.fyiblue.com/HCSC-Pilot/datalake_gold/tree/ETDL-373591-GOLD/datalakebin/gold/integration/params/membership)/bluestar/

# 10. Document Approval

The following individual approvals are required at the completion of this document. Approval by signature here or by email is acceptable.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Title** | **Date** | **Signature** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 11. Appendix