BIG DATA Standardisation - Data Lake Ingestion

One Bank One UniCredit

Data Warehousing & Big Data Summer School
3rd Edition

Oana Vasile & Razvan Stoian

Data & Analytics
Data Sourcing and Transformation

21.06.2017

Welcome to UniCredit

Content

- Big Data Introduction
- ETL Process Overview – IBM DataStage
- Study Case: UBIS Implementation

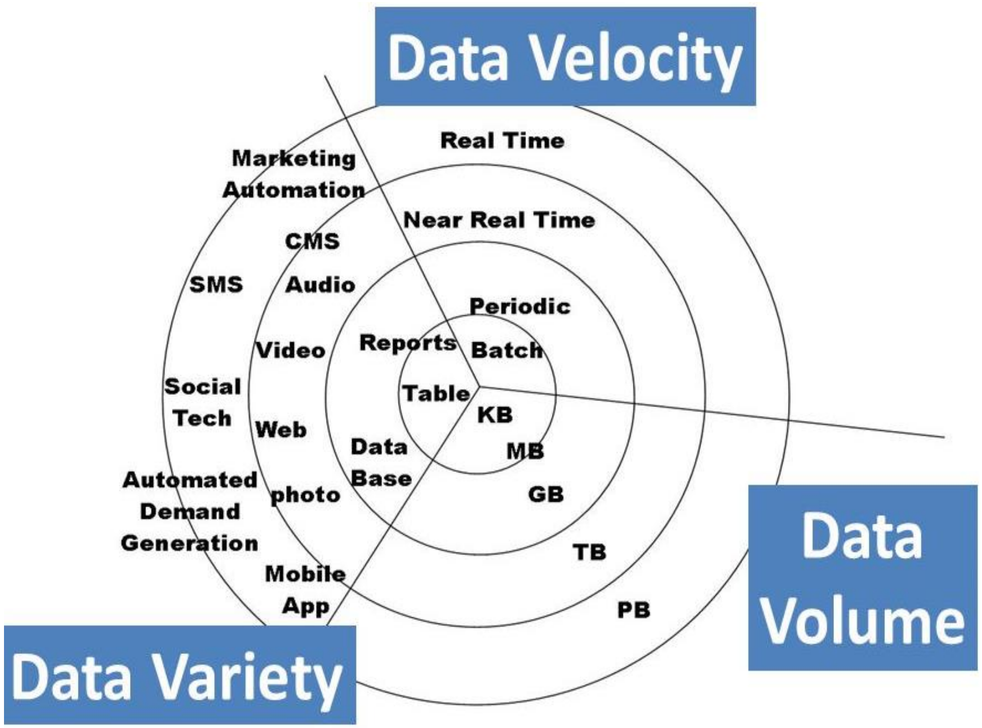
Content

- **Big Data Introduction**
  - Big Data Definition
  - From Data Warehouse to Data Lake
  - HDFS – Reliable storage
  - Cloudera Hadoop Distribution
  - Data files on Hadoop
  - SQL-like querying: HIVE and Impala
- **ETL Process Overview – IBM DataStage**
- **Study Case: UBIS Implementation**

Big Data Definition

Berkeley study: top recurring themes in our thought leaders’ definitions

Big Data Definition: the Vs



From Data Warehouse to Data Lake

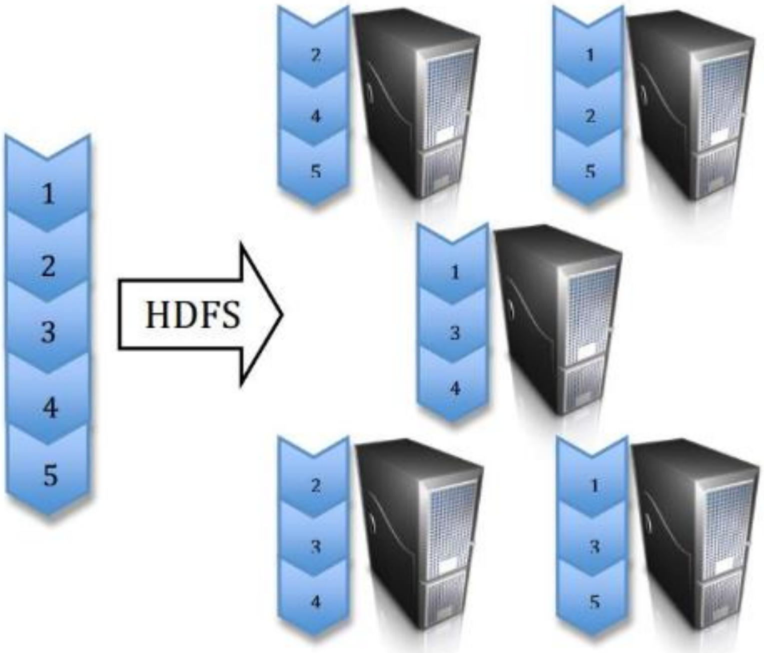
- Historically, analytics and business intelligence workloads are done using a **data warehouse**, a technology that IT departments have tried to make a central data repository.
- **Data warehouses** and databases, by their very nature, are too expensive and too constrained by storage and performance to put all of your data in one place.

- Storage options built on cheap commodity hardware such as the **Hadoop Distributed File System** offered a different approach that was sorely needed as businesses sought to leverage more data and more complex data than ever before.
- **Data Lakes** or **data hubs** storage repositories and processing systems that can ingest data without compromising the data structure -- have become synonymous with modern data architecture and big data management.
- The resulting **data lake** has a major benefit:
  - The lack of a data structure gives data scientists a chance to analyze the data without a predetermined schema and companies can move away from the rigid structure-ingest-analyze process to a more flexible ingest-analyze-understand process.

HDFS – Reliable storage

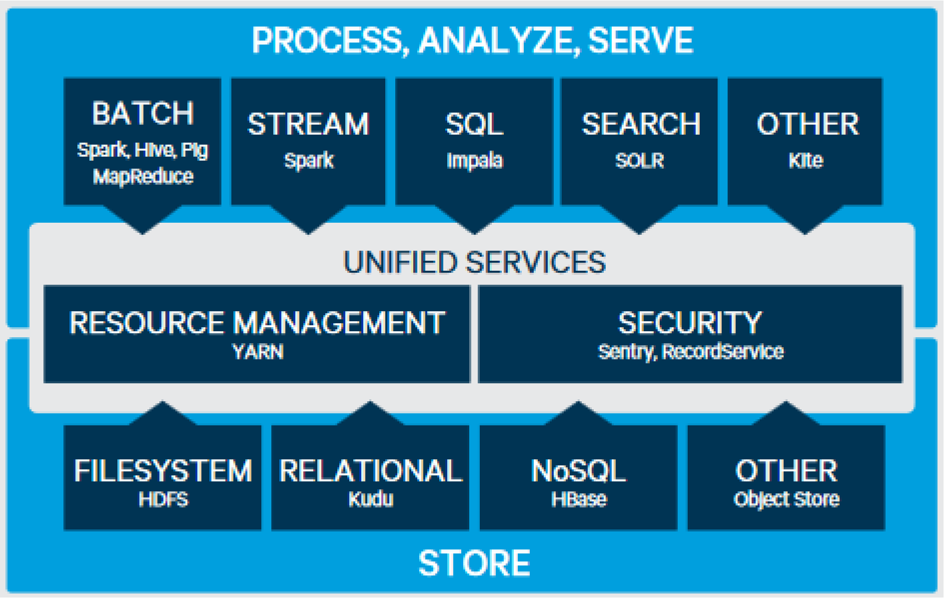
- **Hadoop** includes a fault-tolerant storage system called the **Hadoop Distributed File System**. **HDFS** is able to store huge amounts of information, scale up incrementally and survive the failure of significant parts of the storage infrastructure without losing data.
- Hadoop creates **clusters** of machines and coordinates work among them. Clusters can be built with inexpensive computers. If one fails, Hadoop continues to operate the cluster without losing data or interrupting work, by shifting work to the remaining machines in the cluster.

- HDFS manages storage on the cluster by breaking incoming files into pieces, called **blocks** and storing each of the blocks redundantly across the pool of servers. In the common case, HDFS stores three complete copies of each file by copying each piece to three different servers.



Cloudera Hadoop Distribution

- **Cloudera** distribution including **Apache Hadoop** provides an analytics platform and the latest open source technologies to store, process, discover, model and serve large amounts of data.
- Hadoop is an ecosystem of open source components that fundamentally changes the way enterprises store, process, and analyze data. Unlike traditional systems, Hadoop enables multiple types of analytic workloads to run on the same data, at the same time, at massive scale on industry-standard hardware.
- By integrating Hadoop with more than a dozen other critical open source projects, Cloudera has created a functionally advanced system that helps you perform end-to-end Big Data workflows.



Cloudera Hadoop Distribution

- CDH combines storage and computation into a single, scalable system and delivers the flexibility and economics required to perform operations on big data that are not possible with traditional solutions due to time or cost.

- **Advantages**:
    - Unify storage and computation within a single set of system resources
    - Store data in any format, free from rigid schemas
    - Bring a diverse array of analytic frameworks to a single pool of data—including batch processing, analytic SQL, interactive search, machine learning, stream processing, and a variety of 3rd party applications
    - Process data in parallel and in place with linear scalability
    - Deliver data in real-time to users and applications
    - Integrate with your existing data management and analysis tools

Data files on Hadoop

- Choosing an appropriate file format can have some significant benefits:
  - Faster read times
  - Faster write times
  - Splittable files (so you don't need to read the whole file, just a part of it)
  - Schema evolution support (allowing you to change the fields in a dataset)
  - storage economy (type of file and compression)

HOW TO CHOOSE A DATA FORMAT?

- Choosing a data format is not always black and white, it will depend on several characteristics including:
  - Size and characteristics of the data
  - Project infrastructure
  - Use case scenarios

Data files on Hadoop

Plain text storage (eg, CSV, TSV files)

- Text files are human readable and easily parsable
- Text files are slow to read and write.
- Data size is relatively bulky and not as efficient to query.
- No metadata is stored in the text files so we need to know how the structure of the fields.
- Text files are not splittable after compression
- Limited support for schema evolution: new fields can only be appended at the end of the records and existing fields can never be removed.

JSON (JavaScript Object Notation)

- designed for human-readable data interchange.
- easy to read and write.
- lightweight text-based interchange format.
- language independent.

Sequence Files

- Row-based
- More compact than text files
- You can't perform specified key editing, adding, removal: files are append only
- Encapsulated into the Hadoop environment
- Support splitting even when the data inside the file is compressed
- The sequence file reader will read until a sync marker is reached ensuring that a record is read as a whole
- Sequence files do not store metadata, so the only schema evolution option is appending new fields

*) Image and video files have are not treated here and are managed in a particular way (e.g.: leveraging on HIPI ImageBundle framework)

Data files on Hadoop

AVRO

- Row-based
- Direct mapping from/to JSON
- Interoperability: can serialize into Avro/Binary or Avro/Json
- Provides rich data structures
- Map keys can only be strings (could be seen as a limitation)
- Compact binary form
- Extensible schema language
- Untagged data
- Bindings for a wide variety of programming languages
- Dynamic typing
- Provides a remote procedure call
- Supports block compression
- Avro files are splittable
- Best compatibility for evolving data schemas

Columnar File Formats (Parquet)

- Column-oriented
- Efficient in terms of disk I/O and memory

utilization

- Efficiently encoding of nested structures and sparsely populated data.
- Provides extensible support for per-column encodings.
- Provides extensibility of storing multiple types of data in column data.
- Offers better write performance by storing metadata at the end of the file.
- Records in columns are homogeneous so it's easier to apply encoding schemes.
- Parquet supports Avro files via object model converters that map an external object model to Parquet's internal data types

ORC files

- Row-based
- More compact than text files
- You can't perform specified key editing, adding, removal: files are append only
- Encapsulated into the Hadoop environment

- Support splitting even when the data inside the file is compressed
- The sequence file reader will read until a sync marker is reached ensuring that a record is read as a whole
- Sequence files do not store metadata, so the only schema evolution option is appending new fields

SQL-like querying: HIVE and Impala

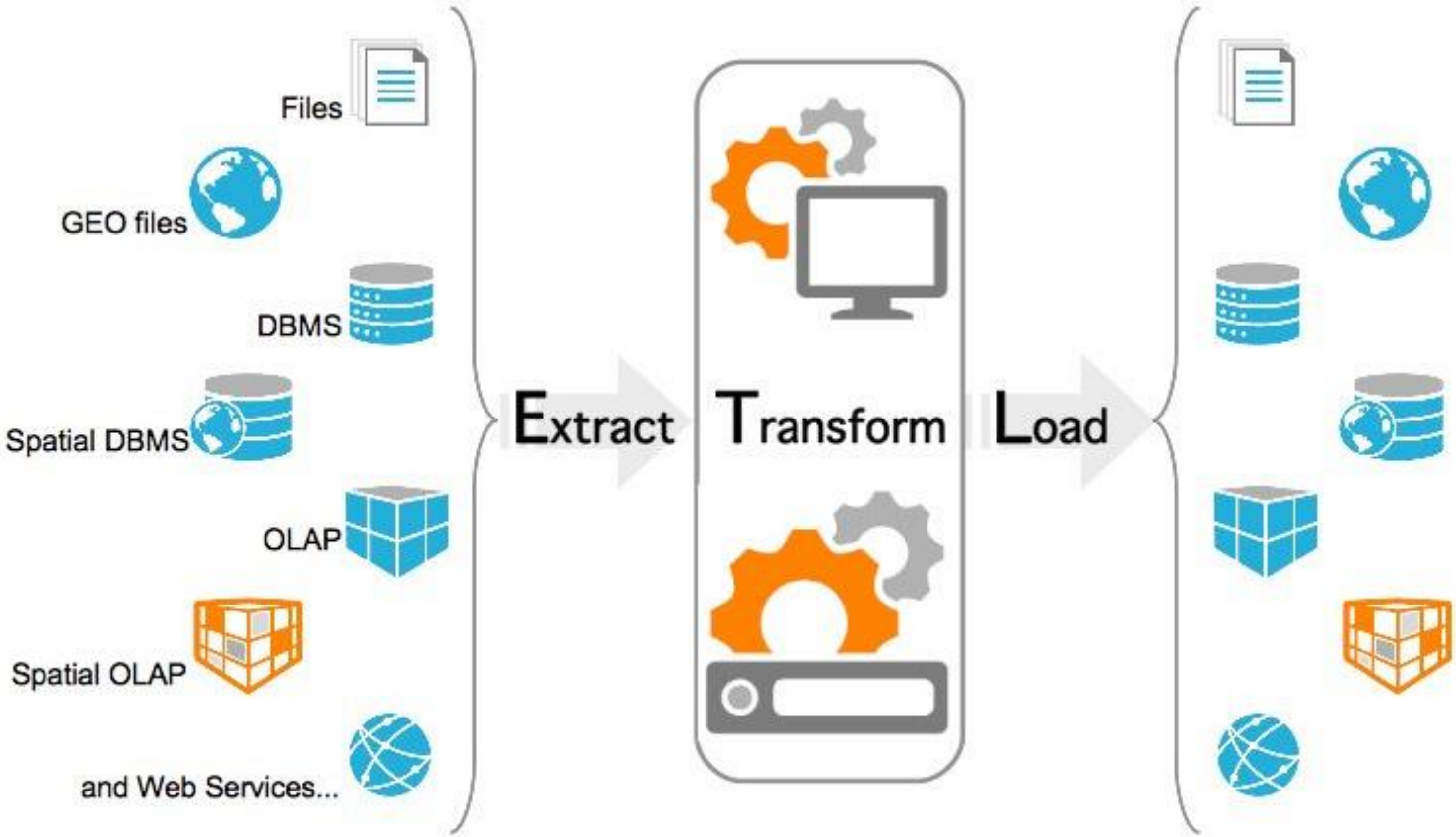
- **Apache Hive**
  - introduced by Facebook to manage and process the large datasets in the distributed storage
  - is an abstraction on Hadoop MapReduce and has its own SQL like language – HiveQL
  - it's a great interface for anyone coming from the relational database world: to use it, you set up structured tables that describe your input and output, issue load commands to ingest your files, and then write your queries as you would in any other relational database
  - **Limitation:** provided a familiar and powerful query mechanism for Hadoop users, but query response times are often unacceptable due to Hive's reliance on MapReduce

- **Cloudera Impala**
  - seeks to improve interactive query response time for Hadoop users
  - extension to Apache Hadoop, providing a very high-performance alternative to the *Hive-on-top-of-MapReduce* model

Content

- Big Data Introduction
- ETL Introduction
    - IBM Datastage
    - Datastage Runtime Architecture on Hadoop
- Study Case: UBIS Implementation

IBM DataStage



DataStage clients

- Designer

- create DataStage jobs

- Director

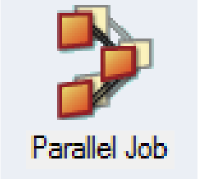
- run and monitor jobs

- Administrator

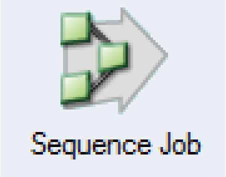
- configure DataStage projects
- administer execution environments

DataStage : Jobs

- **Parallel jobs**
  - executable program
  - development steps:
    - import metadata into the Repository
    - build job in Designer using stages and links
    - compile job in Designer : generates OSH code
    - run and monitor job execution in Director



- **Job Sequences**
  - master controlling job that controls the execution of a set of subordinate jobs
  - passes values to the subordinate job parameters
  - controls the order of execution
  - specifies conditions under which the subordinate jobs get executed



DataStage : Most used stages

- File
  - Sequential file
  - Data Set
  - SAS
- Database
  - ODBC
  - Oracle
  - Teradata
- Processing
  - Sort
  - Remove Duplicates
  - Transformer
  - Aggregator
  - Filter
  - Copy
  - Funnel
  - Join, Lookup, Merge

IBM DataStage on Hadoop

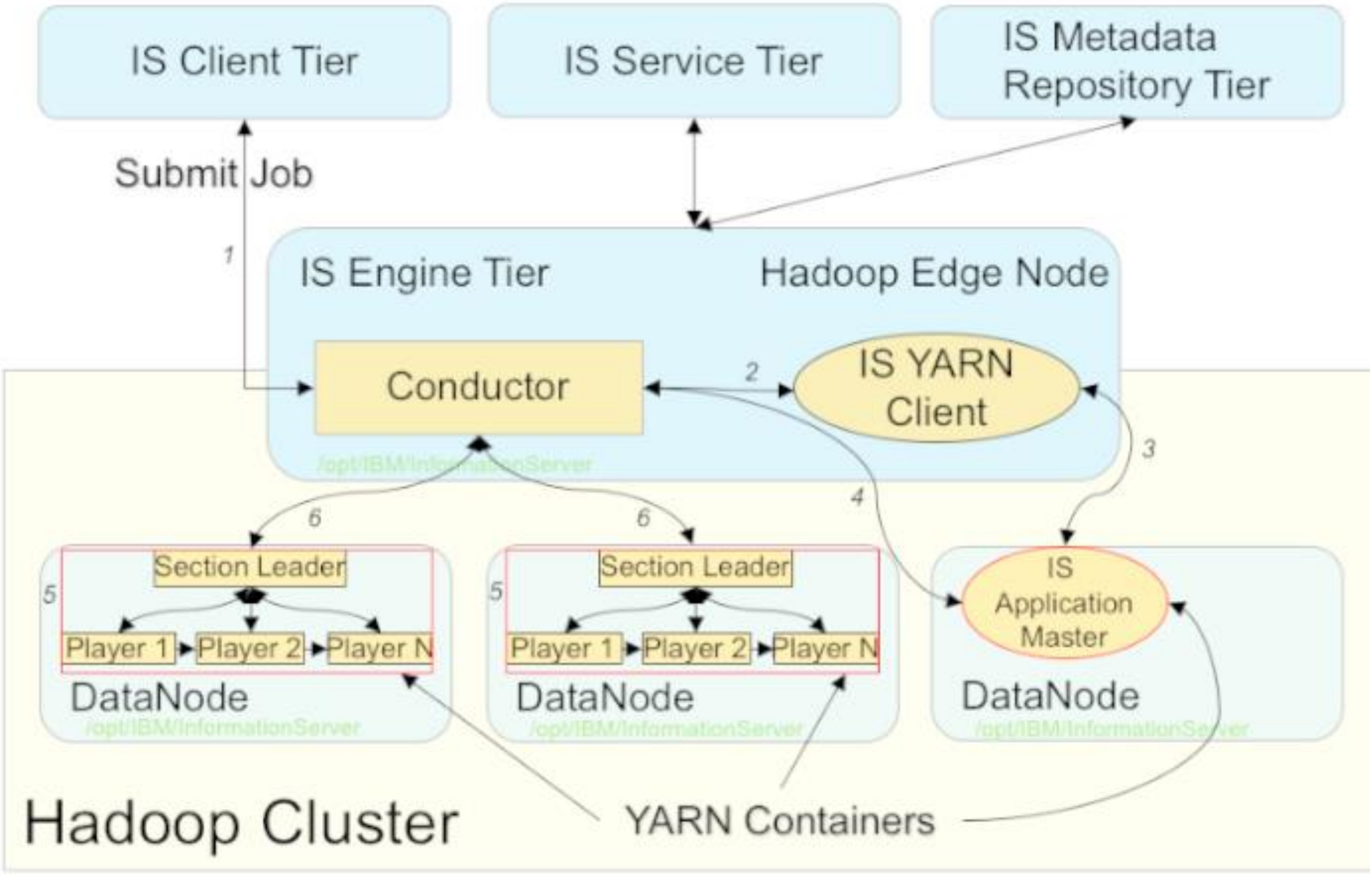
- In release 11.5, Information Server can execute directly inside a Hadoop cluster. This means that all of the data connectivity, transformation, cleansing, enhancement, and data delivery features that thousands of enterprises have relied on for years, can be immediately available to run within the Hadoop platform
- To use the functionality on Hadoop, the engine tier is installed on a Hadoop edge node in a Hadoop cluster. The product is configured to send jobs to the InfoSphere Information Server engine tier in Hadoop so that the jobs will run on the Hadoop cluster.
- The engine tier node communicates with **YARN** to run a job on the compute nodes on a Hadoop cluster.

- Stages for Big Data processing
  - **Big Data File** : enables InfoSphere DataStage to exchange data with Hadoop
  - **File Connector** : write/read AVRO files on Hadoop
  - **HIVE Connector** : access HIVE database

YARN – Datastage Runtime Architecture on Hadoop

Apache Hadoop YARN is the framework for job scheduling and cluster resource management.

Information Server can communicate with YARN to run a job on the data nodes on a Hadoop cluster in the following way:

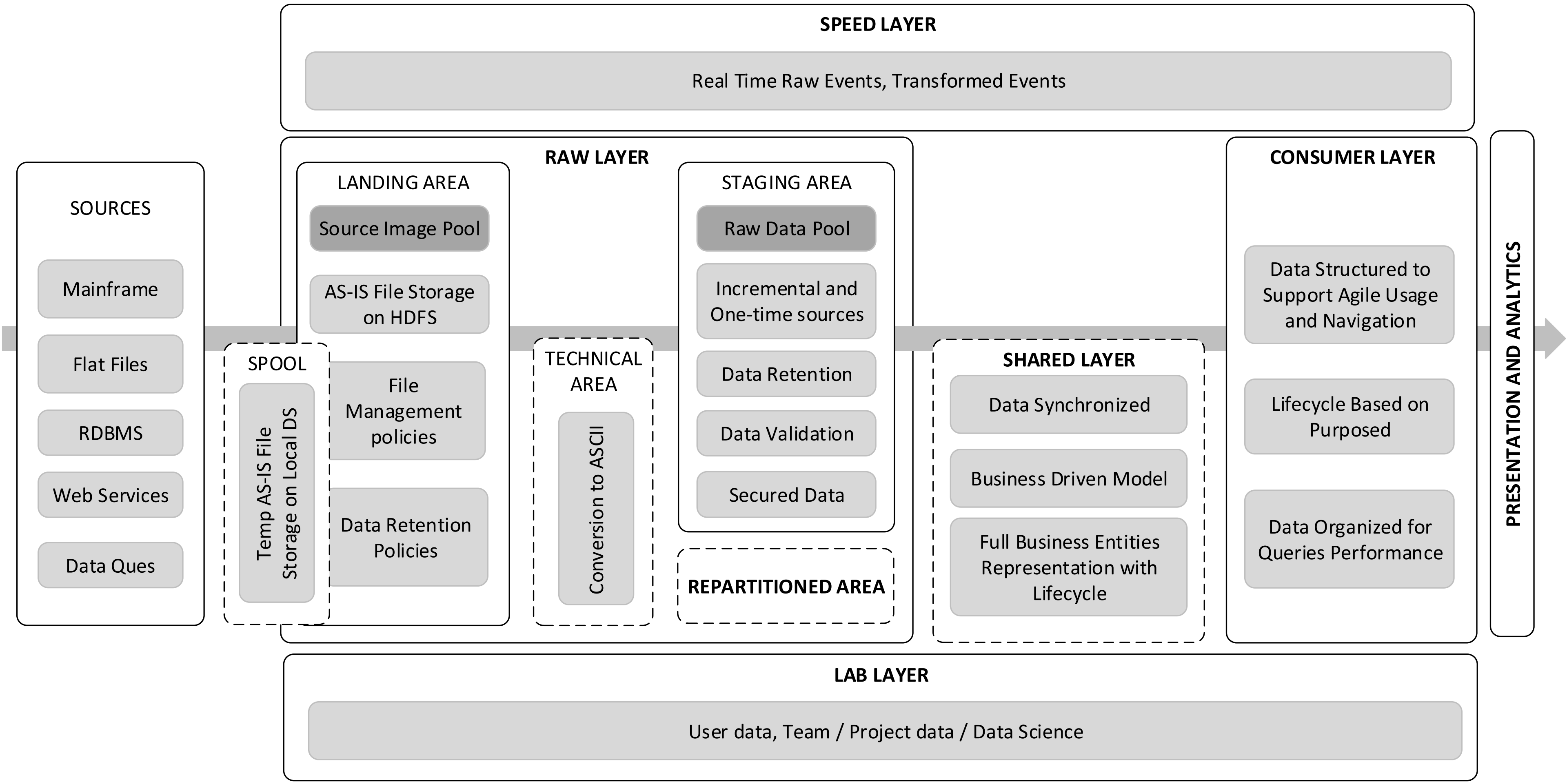


Content

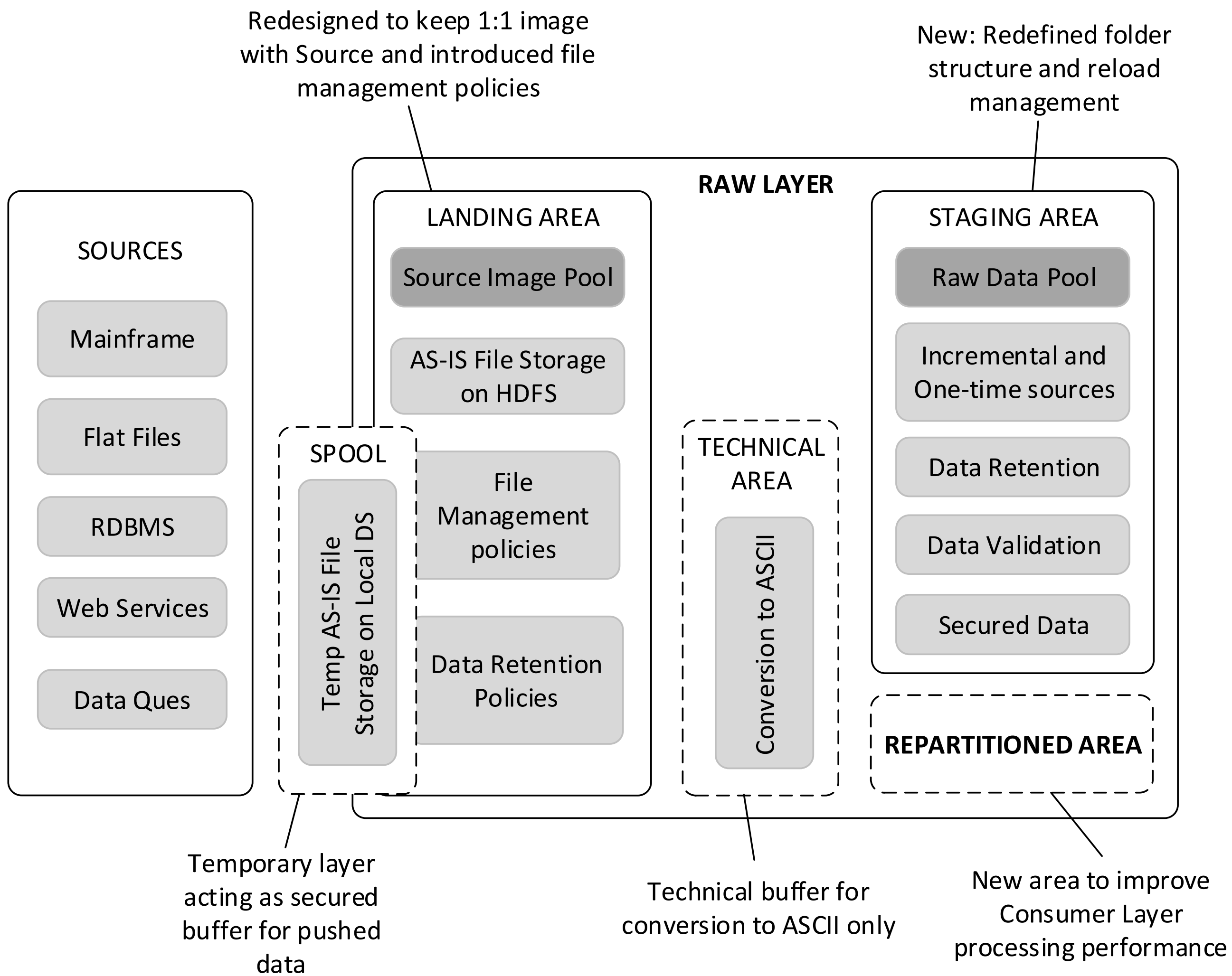
• Study Case: UBIS Implementation

- Data Lake Structure
- Technical Architecture Overview/Framework
- Technical Metadata Repository
- DataStage process
- File compression
- Security : Protegrity and Kerberos
- Process Logging with Splunk

Data Lake Structure

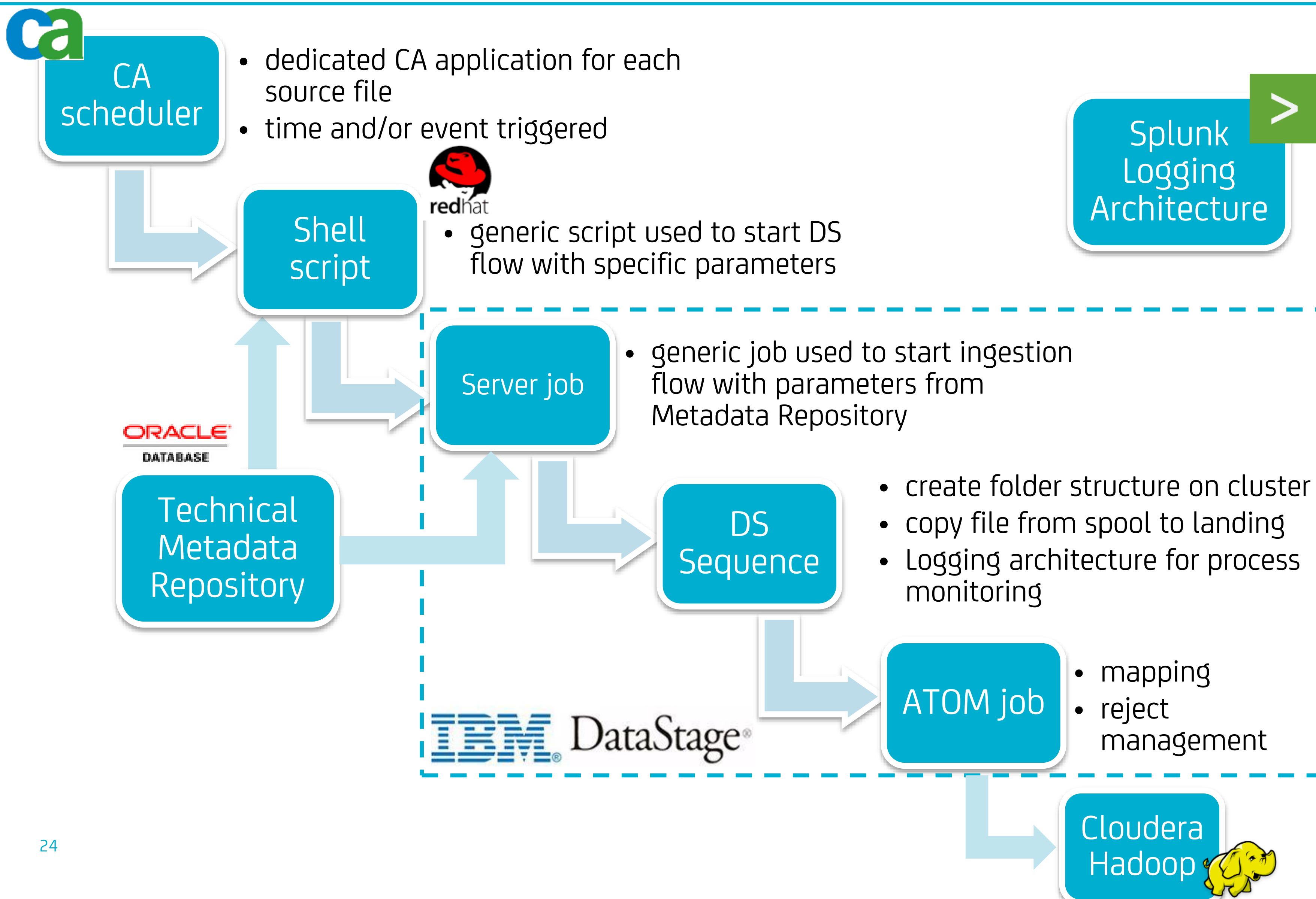


Data Lake Structure: Focus on Batch Ingestion



* Current architecture designed for flat files, Mainframe files., RDBMS is WiP, Data Ques and WebServices have successful POCs

Technical Architecture Overview/Framework



DS Standard Implementation Process for Raw Layer (roles)

1. **Analyst**
   - compiles a standard excel file with:
     - information from analysis of source files
     - mapping definition from source file to raw layer file and data quality check
     - definition of source structure to be used

2. **Developer**
   - get parameters from excel file and insert them into metadata table
   - run setup script to initialize environment
   - develop mapping job (ATOM)
   - run mapping job/sequence by DS client or startup script

Technical Metadata Repository

- An ORACLE database will be used to store the technical metadata – parameters needed to run the loading process

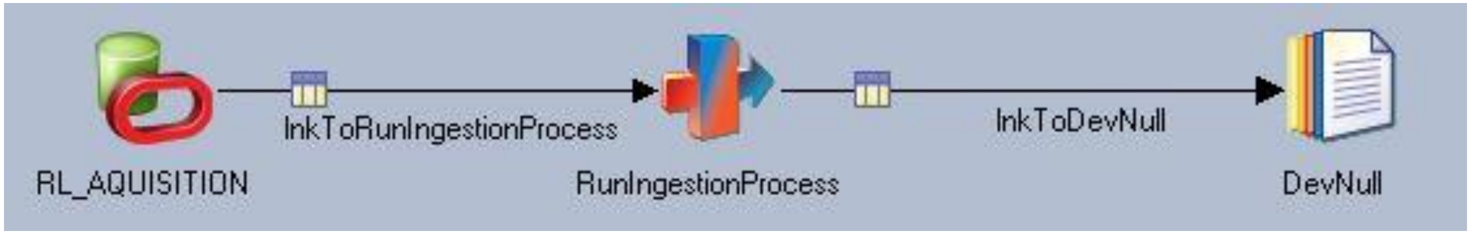
- The metadata model includes:
    - **CBD_RL_ACQUISITION** – operational table with information for Raw Layer Acquisition process
        - contains Raw Layer file register and information needed to manage the file from Spool to STG
        - used by Server job, JS_RunIngestionProc, to retrieve input parameters and start the Ingestion Sequence
    - **CBD_RL_ACQUISITION_H** – history table
        - keep track of changes on parameter values
        - used to rerun processes with older configurations

    - CBD_DS_CFG_APT – Datastage APT configuration lookup table
    - CBD_DS_CFG_APT_YARN – Datstage APT YARN configuration lookup table

        They contain the possible values for APT files to be used running Datastage sequence and jobs – available for tuning operations

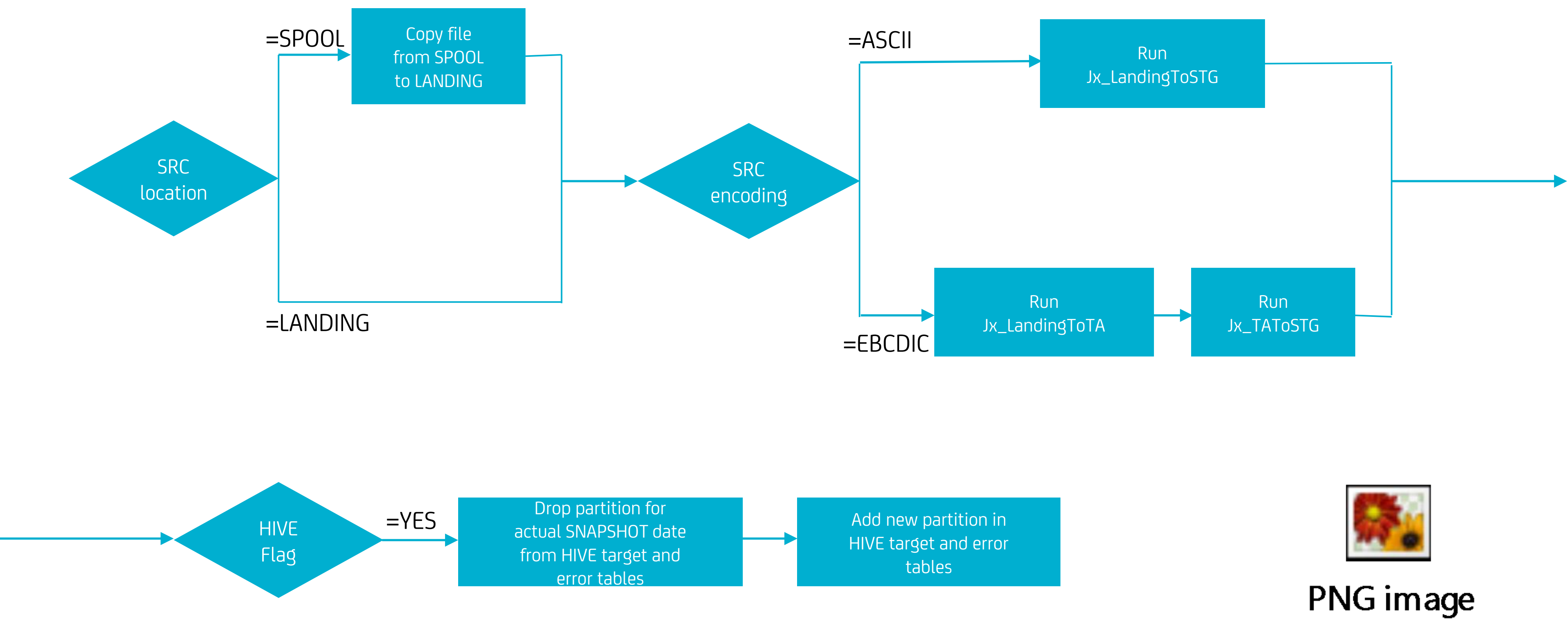
Technical Metadata → Server job

- gets information from the Metadata Repository
- starts the run of the Sequence that controls the loading process

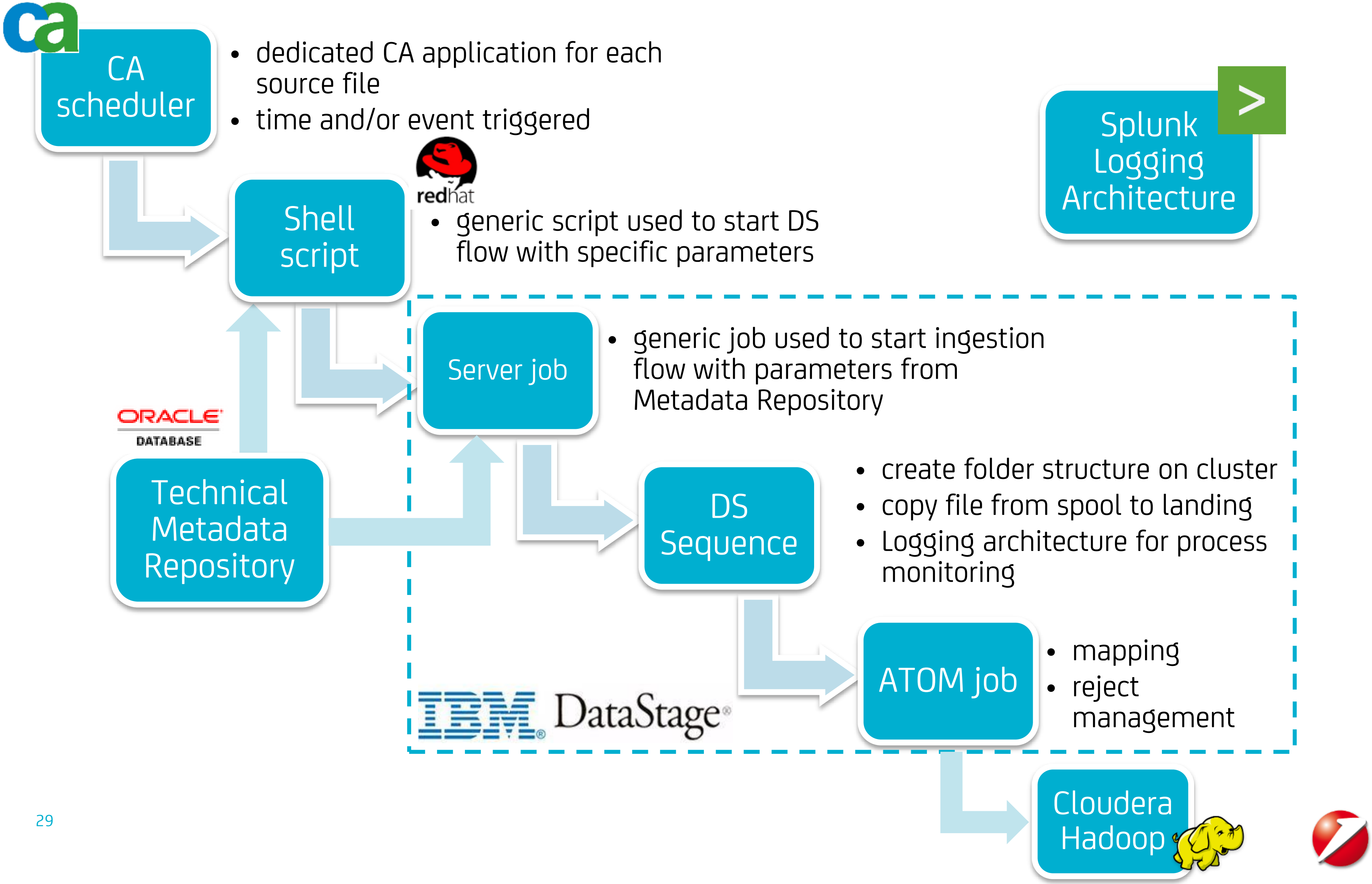


- Input parameters to be provided at runtime:
  - RL_FILE_NAME
  - RL_ORGANIZATION
  - SNAPSHOT_DATE
  - SPOOL_FILENAME - optional
  - VALIDITY_DATE - optional

Server Job → DS Sequence

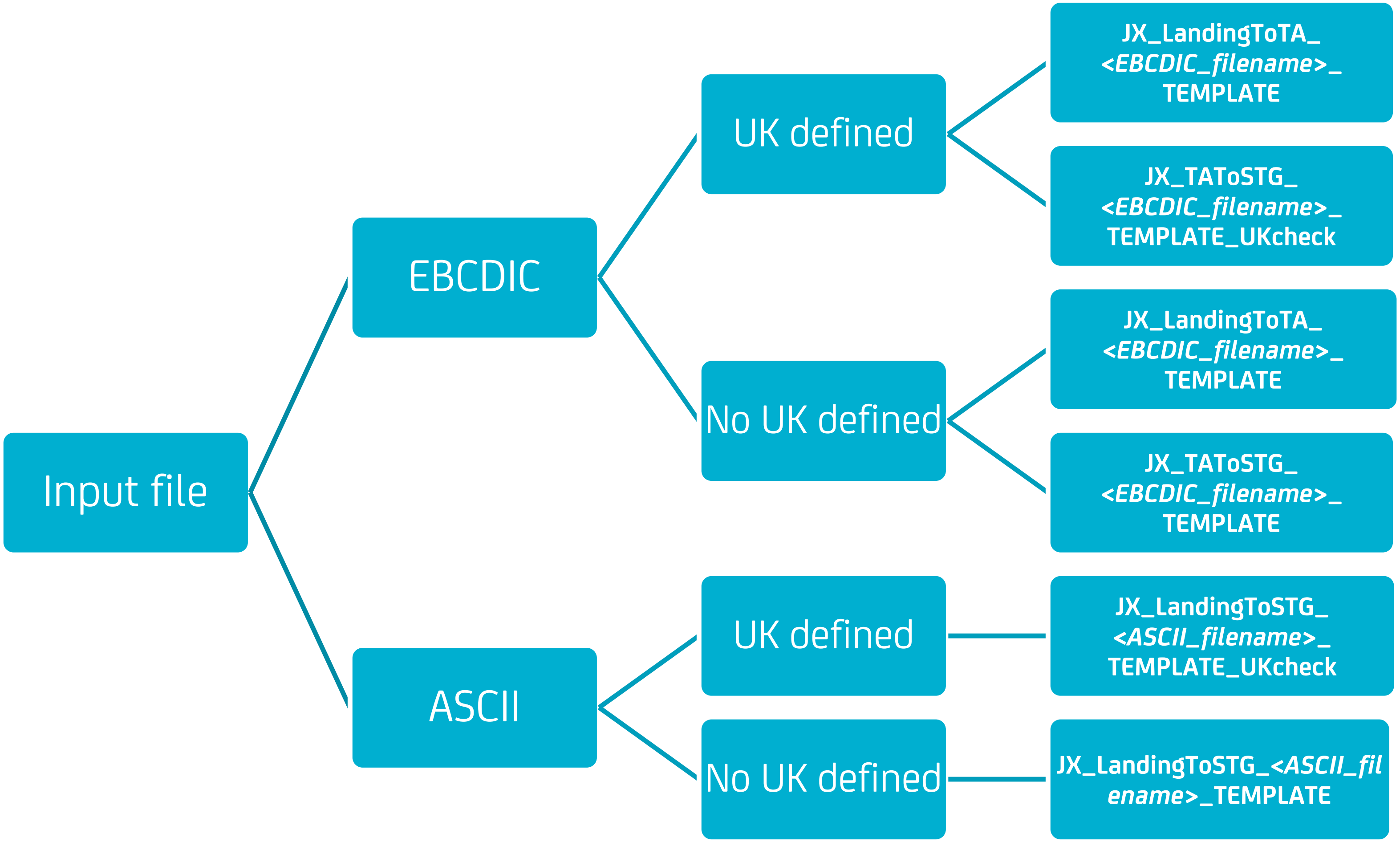


RECAP: Technical Architecture Overview/Framework (NEXT: ATOM JOB)



DS Sequence → Atom (ASCII vs EBCDIC templates)

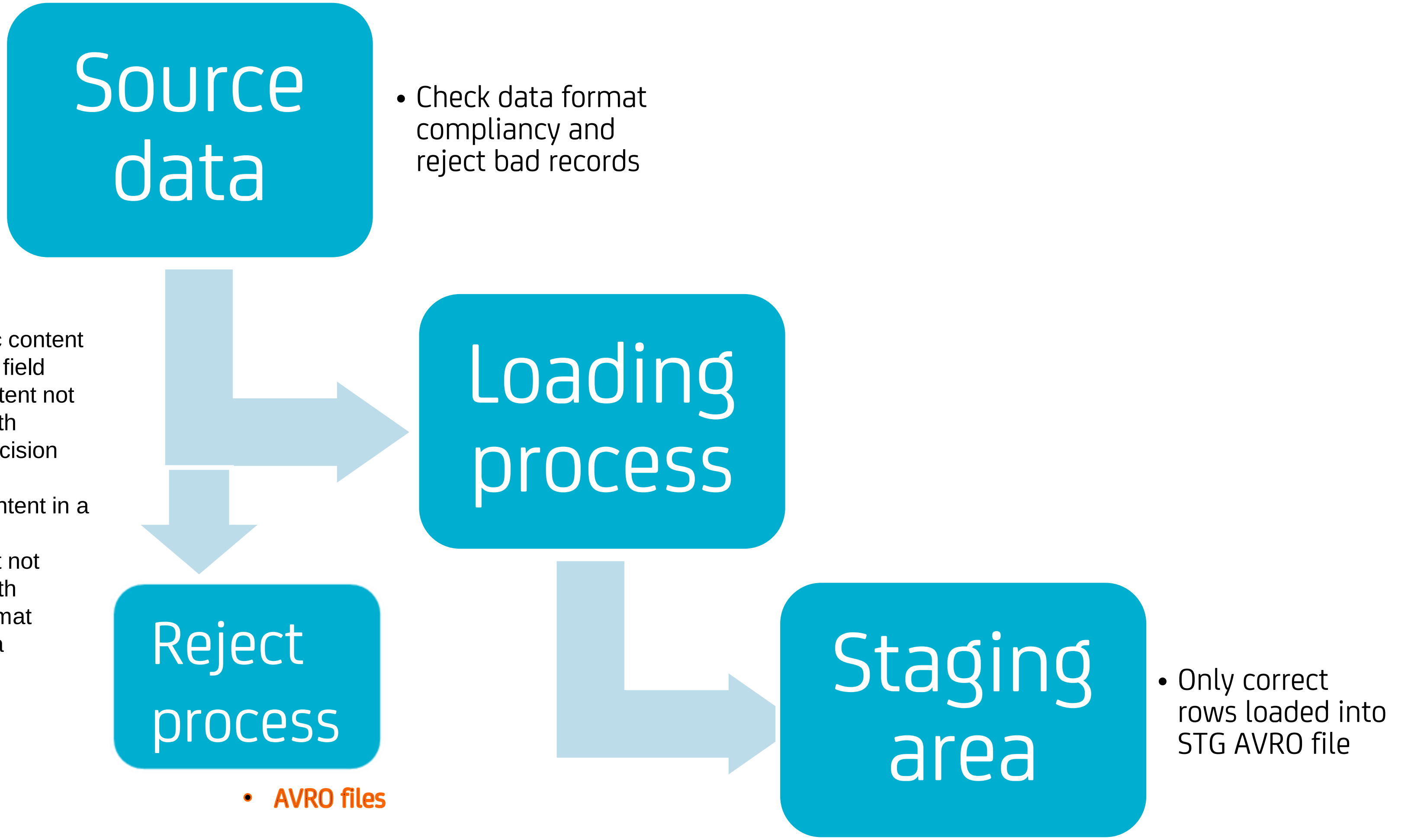
- different prototypes depending on encoding of source files and DQ checks



Atom: Syntactic DQ checks on input data

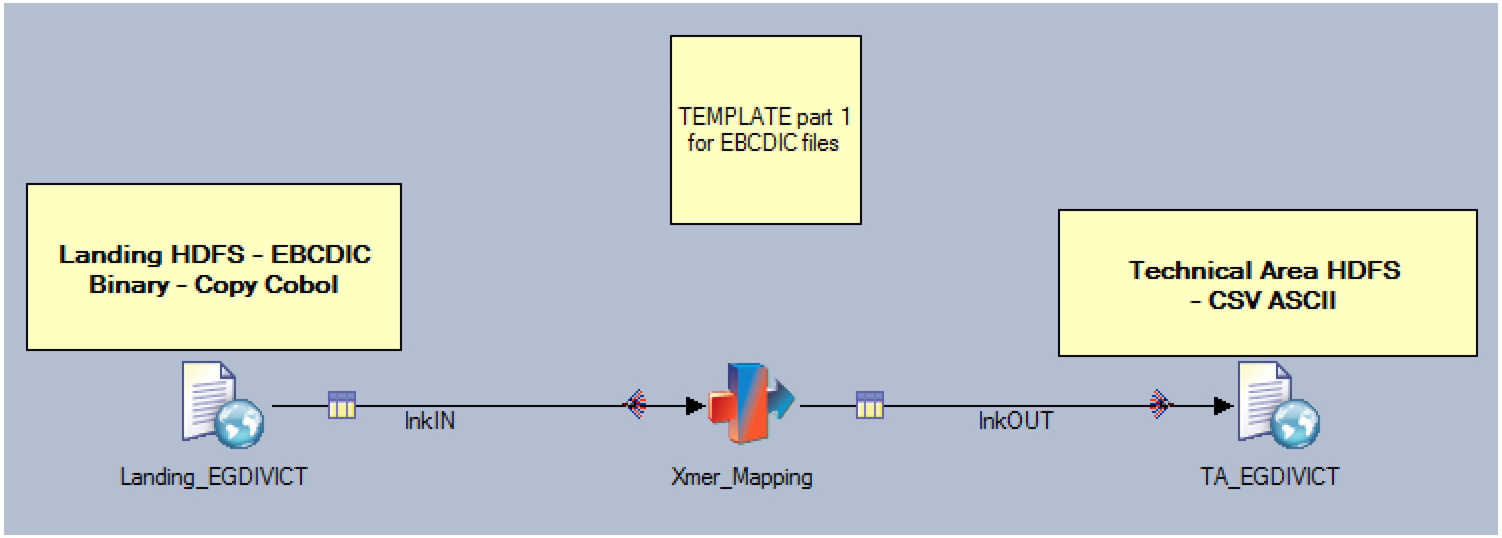
DQ CHECKS

- Non numeric content in a numeric field
- Decimal content not compliant with declared precision and scale
- Non date content in a date field
- Date content not compliant with declared format
- Violation of a declared UK



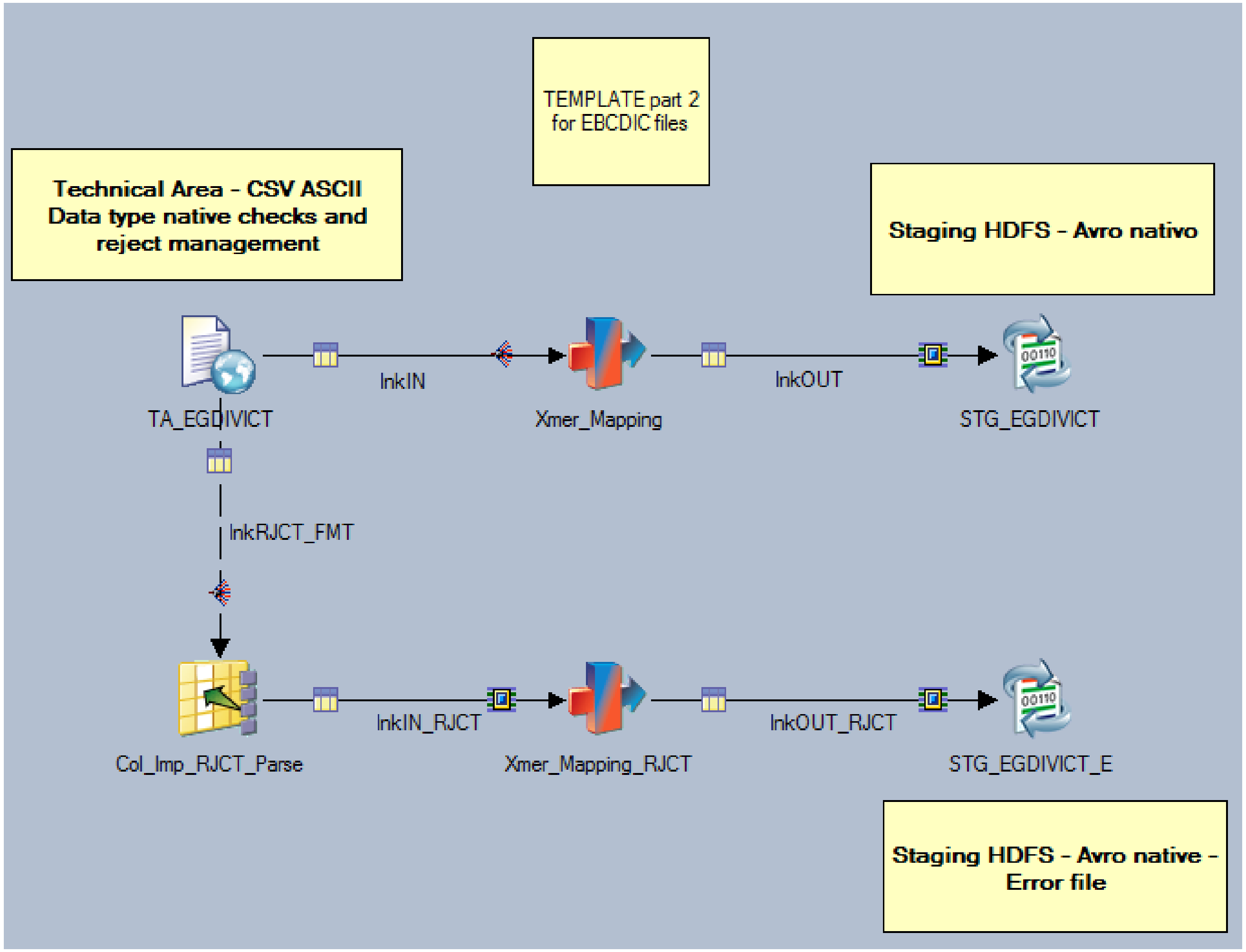
ATOM for EBCDIC file
Landing to TA

- For each EBCDIC file, two steps needed:
  - conversion from EBCDIC to ASCII and store file temporary in Technical Area
  - perform DQ checks and write target AVRO file in Staging area



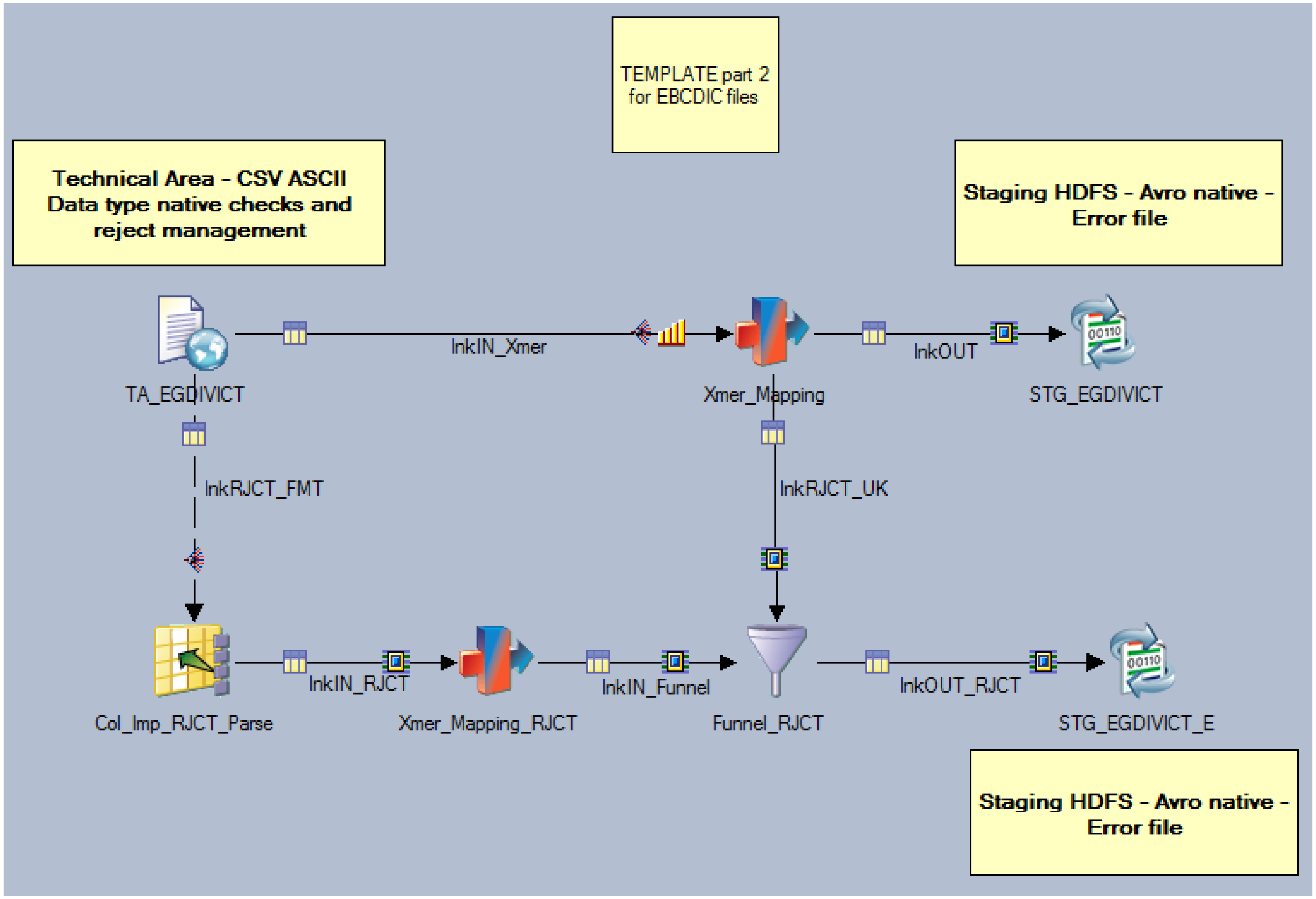
- **Landing file** : specific table definition loaded from CopyCOBOL -> *EGDIVICT.CopyCOBOL*
- **TA file** : table definition with all fields defined as VarChar -> *EGDIVICT.AllVarChar*

ATOM for EBCDIC file
TA to STG only with syntactic DQ checks



- **TA file**: table definition with expected data types, to perform syntactic DQ checks –>*EGDIVICT.DQ*
- **STG file**: table definition with DS AVRO correspondent data types –> *EGDIVICT.DSAvro*
- **Reject file**: table definition with all fields string and additional ErrorMessage column to store rejection reason –> *EGDIVICT.AllVarChar*

ATOM for EBCDIC file
TA to STG with additional UK violation check



- *Xmer_Mapping*

Sort input data by UK

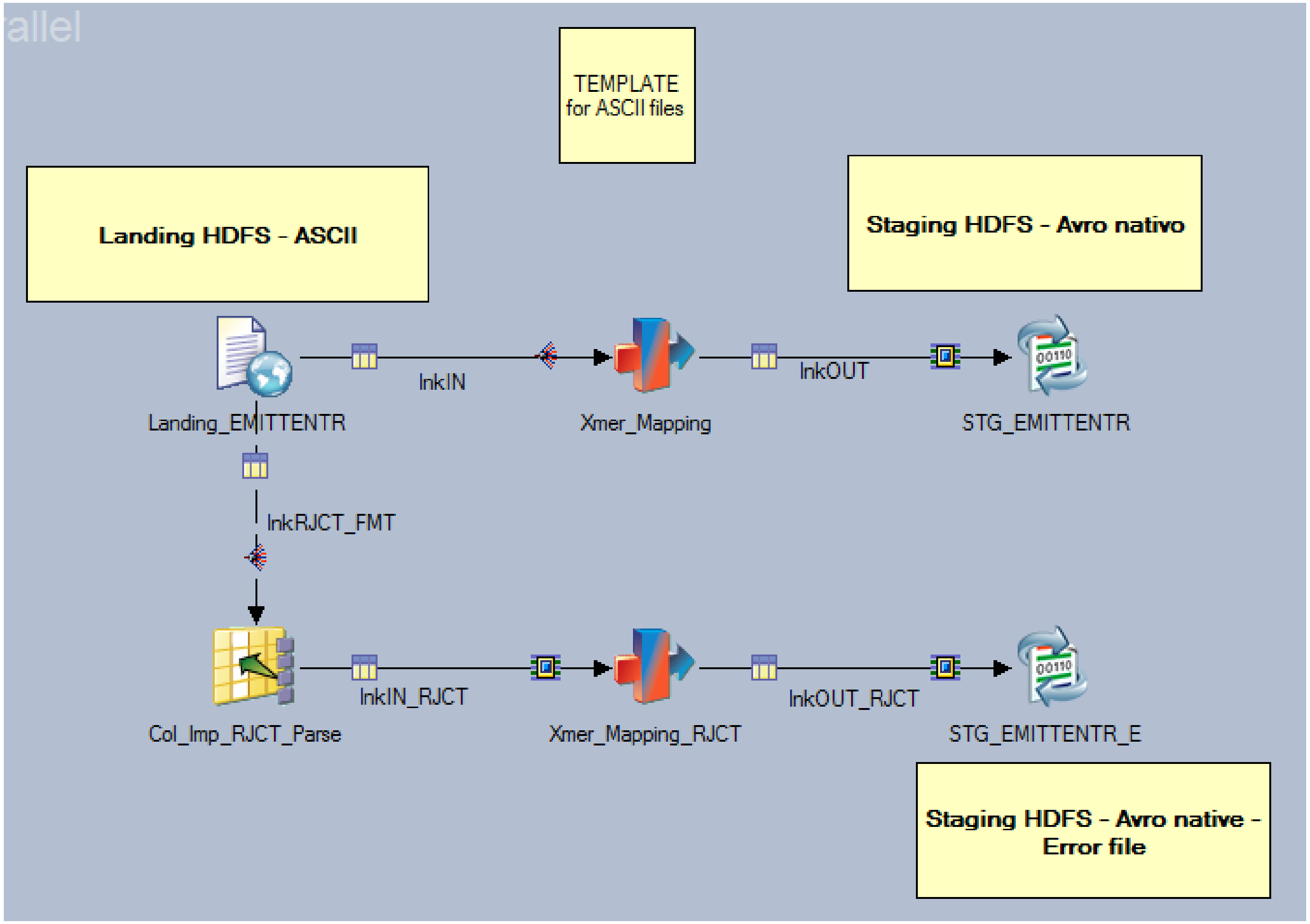
Implements Loop Condition to count number of records with the same value of the UK

Uses constraint to filter out on lnkRJCT_UK, lines with duplicated UK value

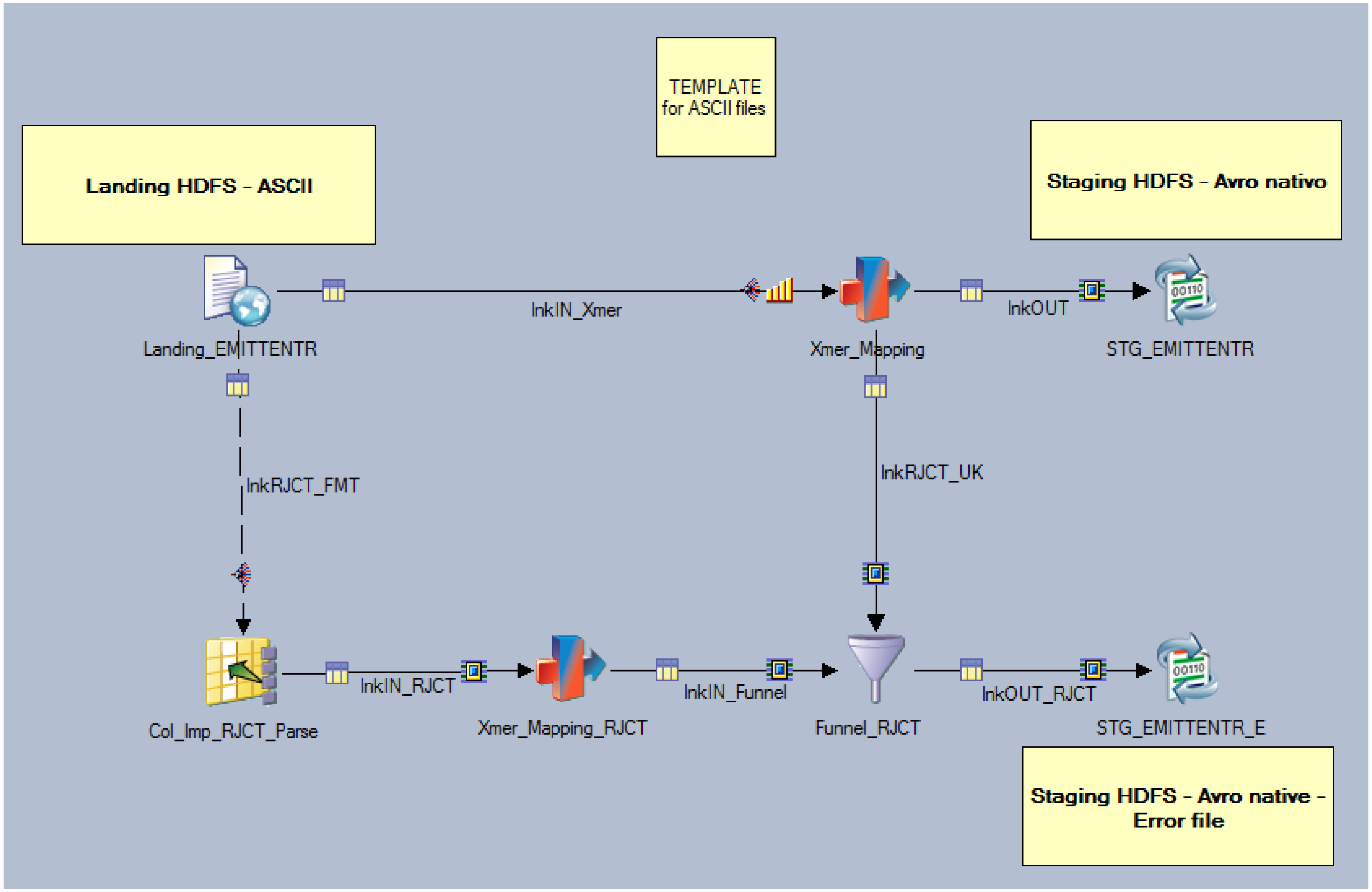
- *Funnel_RJCT*

Union between rejects from data type check and UK check

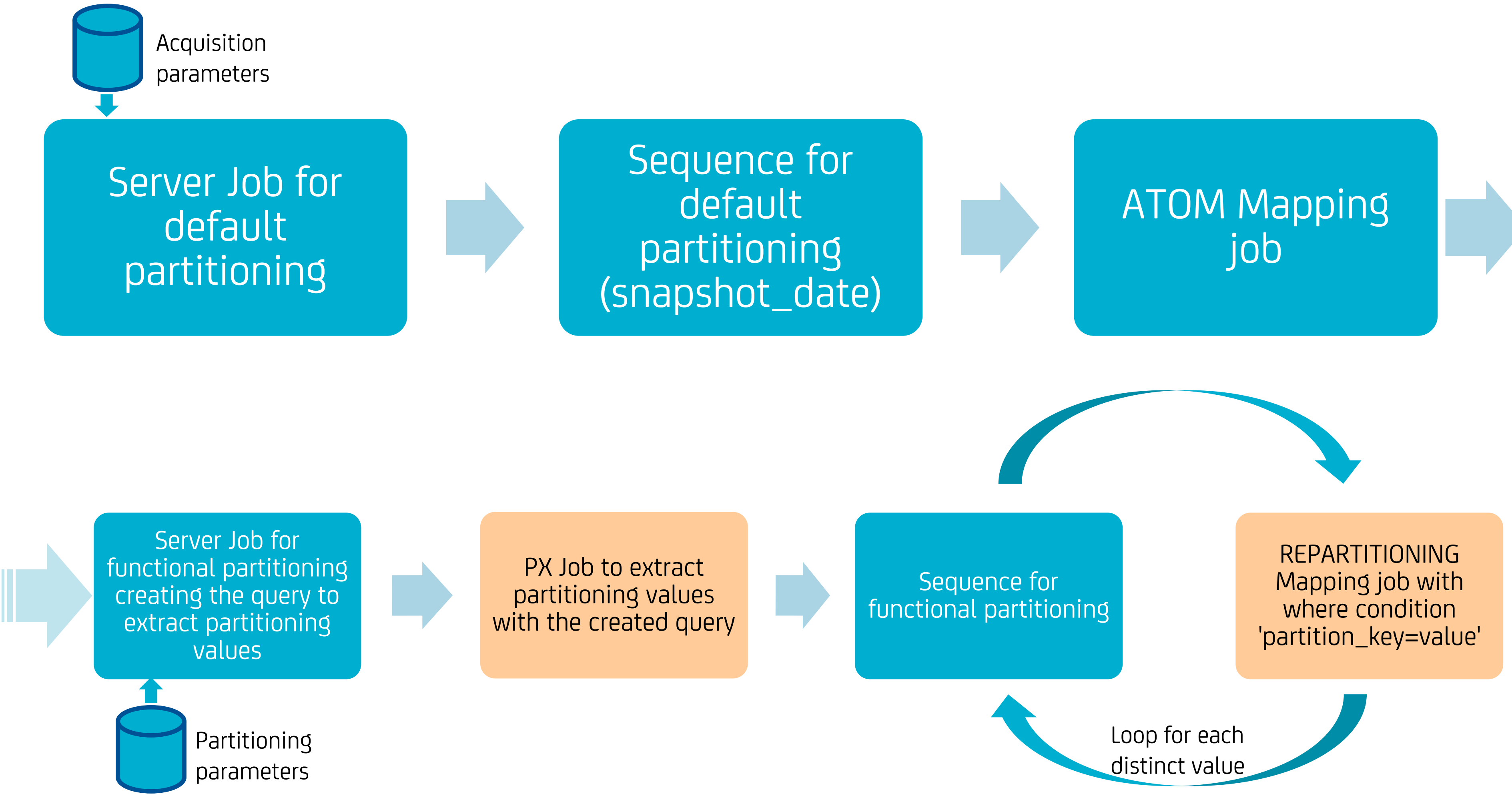
ATOM for ASCII files
Landing to STG only with syntactic DQ checks



ATOM for ASCII files
Landing to STG with additional UK violation check



Repartitioning process



RECAP: Technical Architecture Overview/Framework

CA scheduler

- dedicated CA application for each source file
- time and/or event triggered

Shell script

- generic script used to start DS flow with specific parameters

Splunk Logging Architecture

Server job

- generic job used to start ingestion flow with parameters from Metadata Repository

Technical Metadata Repository

DS Sequence

- create folder structure on cluster
- copy file from spool to landing
- Logging architecture for process monitoring

ATOM job

- mapping
- reject management

IBM DataStage

Cloudera Hadoop

Target structure : AVRO files

- **AVRO schema**

Two **AVRO** schemas are needed: ADF0BDA0_ATTGAR.avsc – for target AVRO file (real data types)
ADF0BDA0_ATTGAR**_e**.avsc – for reject AVRO file (all columns defined as string and one additional column for **DSErrorMessage**)


```
{
"type": "record",
"name": "ATTGAR",
"fields": [
{"name":"CORA_BANCA","type":["long","null"]}
,{"name":"CORA_NDG","type":["string","null"]}
,{"name":"CORA_PROG_GAR","type":["long","null"]}
,{"name":"CORA_ATTRIBUTO_GAR","type":["long","null"]}
,{"name":"CORA_VALORE_ATT_GAR","type":["string","null"]}
,{"name":"CORA_DATA_ELAB","type":["long","null"]}]
}
```

```
{
"type": "record",
"name": "ATTGAR_e",
"fields": [
{"name":"CORA_BANCA","type":["string","null"]}
,{"name":"CORA_NDG","type":["string","null"]}
,{"name":"CORA_PROG_GAR","type":["string","null"]}
,{"name":"CORA_ATTRIBUTO_GAR","type":["string","null"]}
,{"name":"CORA_VALORE_ATT_GAR","type":["string","null"]}
,{"name":"CORA_DATA_ELAB","type":["string","null"]}
,{"name":"DSErrorMessage","type":["string","null"]}]
}
```

Target structure : AVRO file

- Check data inside AVRO file

hadoop jar /opt/cloudera/parcels/CDH/lib/avro/avro-tools.jar tojson *<filepath>/<filename>*

```
tucbd798@hdptemu03 /hdp_spool/tucbd798 #hadoop jar /opt/cloudera/parcels/CDH/lib/avro/avro-tools.jar tojson
 /data/TEST/STG/UCI-UC0/UGI-ADF/ADF0BDA0.ATTGAR.M/snapshot_date=20170331/000000_0 |more
{"CORA_BANCA":{"long":1},"CORA_NDG":{"string":"0000000000000020"},"CORA_PROG_GAR":{"long":3},"CORA_ATTRIBUT
O_GAR":{"long":1059},"CORA_VALORE_ATT_GAR":{"string":"UCCB"},"CORA_DATA_ELAB":{"long":20170331}}
{"CORA_BANCA":{"long":1},"CORA_NDG":{"string":"0000000000000020"},"CORA_PROG_GAR":{"long":3},"CORA_ATTRIBUT
O_GAR":{"long":1064},"CORA_VALORE_ATT_GAR":{"string":"03226"},"CORA_DATA_ELAB":{"long":20170331}}
{"CORA_BANCA":{"long":1},"CORA_NDG":{"string":"0000000000000020"},"CORA_PROG_GAR":{"long":3},"CORA_ATTRIBUT
O_GAR":{"long":1093},"CORA_VALORE_ATT_GAR":{"string":"0000000230017"},"CORA_DATA_ELAB":{"long":20170331}}
{"CORA_BANCA":{"long":1},"CORA_NDG":{"string":"0000000000000020"},"CORA_PROG_GAR":{"long":3},"CORA_ATTRIBUT
O_GAR":{"long":1094},"CORA_VALORE_ATT_GAR":{"string":"20160826"},"CORA_DATA_ELAB":{"long":20170331}}
{"CORA_BANCA":{"long":1},"CORA_NDG":{"string":"0000000000000020"},"CORA_PROG_GAR":{"long":3},"CORA_ATTRIBUT
O_GAR":{"long":1098},"CORA_VALORE_ATT_GAR":{"string":"S"},"CORA_DATA_ELAB":{"long":20170331}}
{"CORA_BANCA":{"long":1},"CORA_NDG":{"string":"0000000000000020"},"CORA_PROG_GAR":{"long":3},"CORA_ATTRIBUT
O_GAR":{"long":1118},"CORA_VALORE_ATT_GAR":{"string":"05"},"CORA_DATA_ELAB":{"long":20170331}}
{"CORA_BANCA":{"long":1},"CORA_NDG":{"string":"0000000000000020"},"CORA_PROG_GAR":{"long":33},"CORA_ATTRIBU
TO_GAR":{"long":1064},"CORA_VALORE_ATT_GAR":{"string":"02008"},"CORA_DATA_ELAB":{"long":20170331}}
{"CORA_BANCA":{"long":1},"CORA_NDG":{"string":"0000000000000020"},"CORA_PROG_GAR":{"long":34},"CORA_ATTRIBU
TO_GAR":{"long":1064},"CORA_VALORE_ATT_GAR":{"string":"02008"},"CORA_DATA_ELAB":{"long":20170331}}
{"CORA_BANCA":{"long":1},"CORA_NDG":{"string":"0000000000000020"},"CORA_PROG_GAR":{"long":34},"CORA_ATTRIBU
TO_GAR":{"long":1093},"CORA_VALORE_ATT_GAR":{"string":"0000000426344"},"CORA_DATA_ELAB":{"long":20170331}}
{"CORA_BANCA":{"long":1},"CORA_NDG":{"string":"0000000000000020"},"CORA_PROG_GAR":{"long":34},"CORA_ATTRIBU
TO_GAR":{"long":1094},"CORA_VALORE_ATT_GAR":{"string":"20160826"},"CORA_DATA_ELAB":{"long":20170331}}
```

Target structure : HIVE table

- connect to HIVE : **beeline –u** *<connection string>*

```
0: jdbc:hive2://hdptemu:10000/> show create table adf0bda0_attgar;
INFO  : Compiling command(queryId=hive_20170619142525_6d138017-9bed-40f7-aa16-8ad779b08ca5): show create ta
INFO  : Semantic Analysis Completed
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:createtab_stmt, type:string, comment:f
INFO  : Completed compiling command(queryId=hive_20170619142525_6d138017-9bed-40f7-aa16-8ad779b08ca5); Time
INFO  : Executing command(queryId=hive_20170619142525_6d138017-9bed-40f7-aa16-8ad779b08ca5): show create ta
INFO  : Starting task [Stage-0:DDL] in serial mode
INFO  : Completed executing command(queryId=hive_20170619142525_6d138017-9bed-40f7-aa16-8ad779b08ca5); Time
INFO  : OK
+---------------------------------------------------------------------------------------------------------+-
|                                              createtab_stmt                                              |
+---------------------------------------------------------------------------------------------------------+-
| CREATE EXTERNAL TABLE `adf0bda0_attgar`(                                                                 |
|   `cora_banca` bigint COMMENT '',                                                                        |
|   `cora_ndg` string COMMENT '',                                                                          |
|   `cora_prog_gar` bigint COMMENT '',                                                                     |
|   `cora_attributo_gar` bigint COMMENT '',                                                                |
|   `cora_valore_att_gar` string COMMENT '',                                                               |
|   `cora_data_elab` bigint COMMENT '')                                                                    |
| PARTITIONED BY (                                                                                         |
|   `snapshot_date` string)                                                                                |
| ROW FORMAT SERDE                                                                                         |
|   'org.apache.hadoop.hive.serde2.avro.AvroSerDe'                                                         |
| STORED AS INPUTFORMAT                                                                                    |
|   'org.apache.hadoop.hive.ql.io.avro.AvroContainerInputFormat'                                           |
| OUTPUTFORMAT                                                                                             |
|   'org.apache.hadoop.hive.ql.io.avro.AvroContainerOutputFormat'                                          |
| LOCATION                                                                                                 |
|   'hdfs://hdfs-munich-test/data/TEST/STG/UCI-UC0/UGI-ADF/ADF0BDA0.ATTGAR.M'                              |
| TBLPROPERTIES (                                                                                          |
|   'avro.schema.url'='hdfs://hdfs-munich-test//data/TEST/STG/UCI-UC0/UGI-ADF/ADF0BDA0.ATTGAR.M.avsc',     |
|   'transient_lastDdlTime'='1493818049')                                                                  |
+---------------------------------------------------------------------------------------------------------+-
```

Target structure : HIVE table

```
INFO  : Executing command(queryId=hive_20170619143030_beb54c85-5f85-4762-8f45-6c377839b5c8): select * from adf
0bda0_attgar where snapshot_date=20170331 limit 10
INFO  : Completed executing command(queryId=hive_20170619143030_beb54c85-5f85-4762-8f45-6c377839b5c8); Time ta
ken: 0.001 seconds
INFO  : OK
+------------------------------+---------------------------+---------------------------------+------------------
------------------+---------------------------------------+----------------------------------+-----------------
---------------+--+
| adf0bda0_attgar.cora_banca  | adf0bda0_attgar.cora_ndg  | adf0bda0_attgar.cora_prog_gar  | adf0bda0_attgar.c
ora_attributo_gar  | adf0bda0_attgar.cora_valore_att_gar  | adf0bda0_attgar.cora_data_elab  | adf0bda0_attgar.
snapshot_date  |
+------------------------------+---------------------------+---------------------------------+------------------
------------------+---------------------------------------+----------------------------------+-----------------
---------------+--+
| 1                            | 0000000000000020          | 3                               | 1059
                  | UCCB                                  | 20170331                         | 20170331
               |
| 1                            | 0000000000000020          | 3                               | 1064
                  | 03226                                 | 20170331                         | 20170331
               |
| 1                            | 0000000000000020          | 3                               | 1093
                  | 0000000230017                         | 20170331                         | 20170331
               |
| 1                            | 0000000000000020          | 3                               | 1094
                  | 20160826                              | 20170331                         | 20170331
               |
| 1                            | 0000000000000020          | 3                               | 1098
                  | S                                     | 20170331                         | 20170331
               |
| 1                            | 0000000000000020          | 3                               | 1118
                  | 05                                    | 20170331                         | 20170331
               |
| 1                            | 0000000000000020          | 33                              | 1064
                  | 02008                                 | 20170331                         | 20170331
```

File compression

- **File compression brings two major benefits:**
    1. it reduces the space needed to store files
    2. it speeds up data transfer across the network or to or from disk.

- **Compression formats:** gzip, bzip2, LZO, **Snappy**

- **Reasons to compress:**
    - Data is mostly stored and not frequently processed. It is usual DWH scenario
    - Compression factor is very high and thereof we save a lot of I/O.
    - Decompression is very fast (like Snappy) and thereof we have a some gain with little price
    - Data already arrived compressed
- **Reasons not to compress**
    - Compressed data is not splittable. Have to be noted that many modern format are built with block level compression to enable splitting and other partial processing of the files.
    - Data is created in the cluster and compression takes significant time. Have to be noted that compression usually much more CPU intensive then decompression.
    - Data has little redundancy and compression gives little gain.

Security : Protegrity

- As organizations leverage Big Data to analyze ever larger quantities of data, the challenge of effectively protecting sensitive data while maintaining usability becomes increasingly difficult. **Protegrity**, the leading innovator of advanced data security solutions, offers the most comprehensive package of Hadoop security available to protect assets and meet regulatory compliance while preserving the performance and analytics vital to Big Data platforms.

- **Protegrity**'s well-established file and field level data encryption and tokenization technology can be employed within Big Data environments, creating a seamless network of data-centric security far stronger than access controls alone.

- **Protegrity Big Data Protector** protects any sensitive file stored in the Hadoop Distributed File System (HDFS) and any sensitive data item stored within a file. Vaultless Tokenization is a form of data protection that converts sensitive data into fake data. The real data can be retrieved only by authorized users.

Security : Kerberos

- **Kerberos** is a system for authenticating access to distributed services. In Hadoop, a user or a process may *delegate* the authority to another process, which can then talk to the desired service with the delegated authority. These delegation rights are both limited
  - in scope : the principal delegates authority on a service-by-service basis
  - and in time : it guarantees that if the secret used to act as a delegate, the *token*, is stolen, there is only a finite time for which it can be used.

- A **principal** is an identity in the system: a person or a thing like the Hadoop namenode which has been given an identity.
- In Hadoop, a different principal is usually created for each service and machine in the cluster, such as hdfs/node1, hdfs/node2, ... etc. These principals would then be used for all HDFS daemons running on node1, node2, etc.

- Kerberos is considered "the best there is" in terms of securing distributed systems. Its use of tickets is designed to limit the load on the KDC(**Key Distribution Center**), as it is only interacted with when a principal requests a ticket, rather than having to validate every single request.

Kerberos configuration when connecting to the cluster from command prompt

- In order to access the cluster, the user needs an active KEYTAB file

  - check the KEYTAB : **klist** command

```
[c312899@hdptemu03 ~]$ sudo su - tucbd798
tucbd798@hdptemu03 /hdp_spool/tucbd798 #klist
Ticket cache: FILE:/tmp/krb5cc_498235159
Default principal: tucbd798@SD01.UNICREDITGROUP.EU

Valid starting     Expires            Service principal
02/22/17 10:29:59  02/22/17 20:29:59  krbtgt/SD01.UNICREDITGROUP.EU@SD01.UNICREDITGROUP.EU
        renew until 03/01/17 10:29:59
tucbd798@hdptemu03 /hdp_spool/tucbd798 #
```

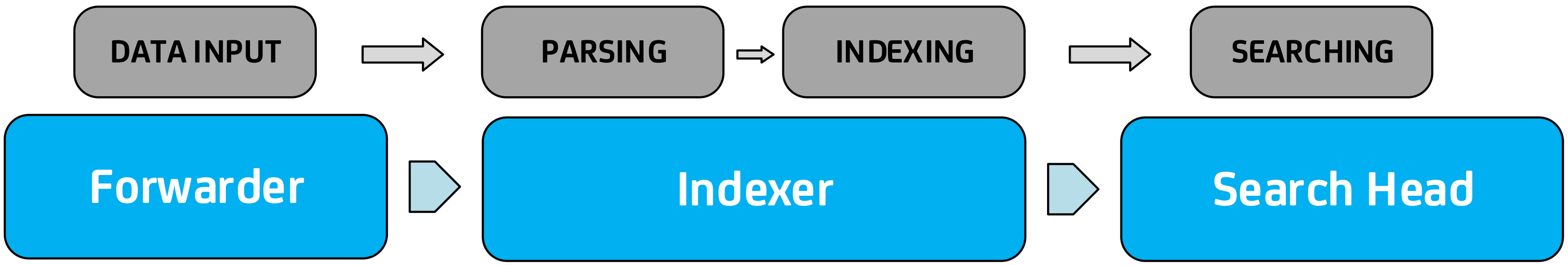
  - to generate new KEYTAB file run following command:

    kinit -kt *<user>***.keytab** *<user>*

Process Logging with Splunk >

- What is Splunk?
  - "Google for Logfiles": It stores all logs and provides very fast search capabilities roughly in the same way Google does for the internet.
  - powerful tool for sifting through vast amounts of data and performing statistical operations on what is relevant in a specific context.

- How Does it Work?

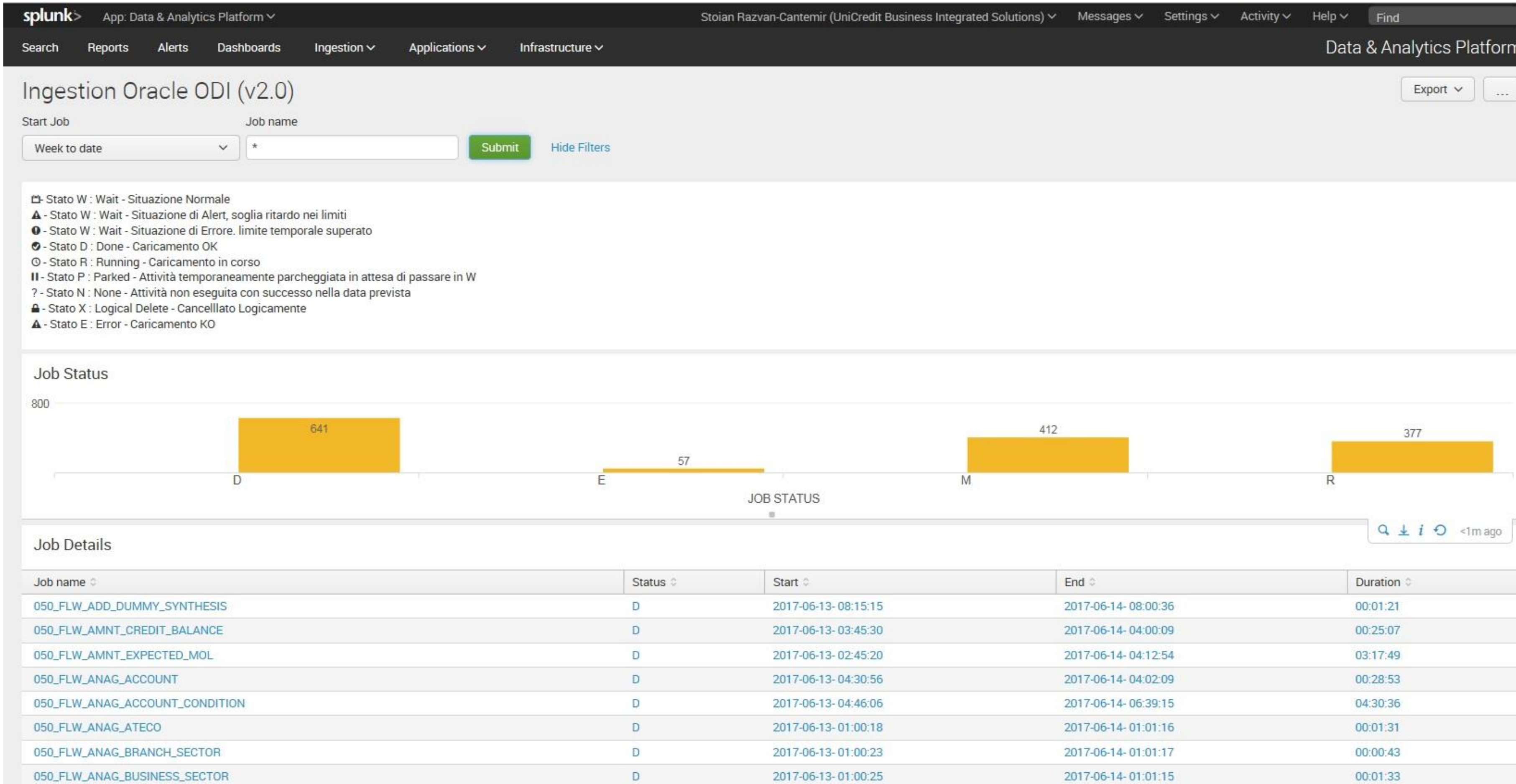


Forwarder
- collects data
- forwards to indexer in real-time

Indexer
- processes data in real-time
- stores & indexes the data on disk

Search Head
- means of interactions with Splunk
- allows searching, analysis and visualization

Splunk > (implementation example)



Q&A

OanaAndreea.Vasile@unicredit.eu
Razvan-Cantemir.Stoian@unicredit.eu

APPENDIX : Abbreviation

Abbreviation	Description
DS	Datastage
DQ	Data Quality
UK	Unique Key
JX	Custom abbreviation used for Parallel DS Job
TA	Technical Area
STG	Staging
AVSC	AVRO Schema
RL	Raw Layer
SRC	Source
RJCT	Reject