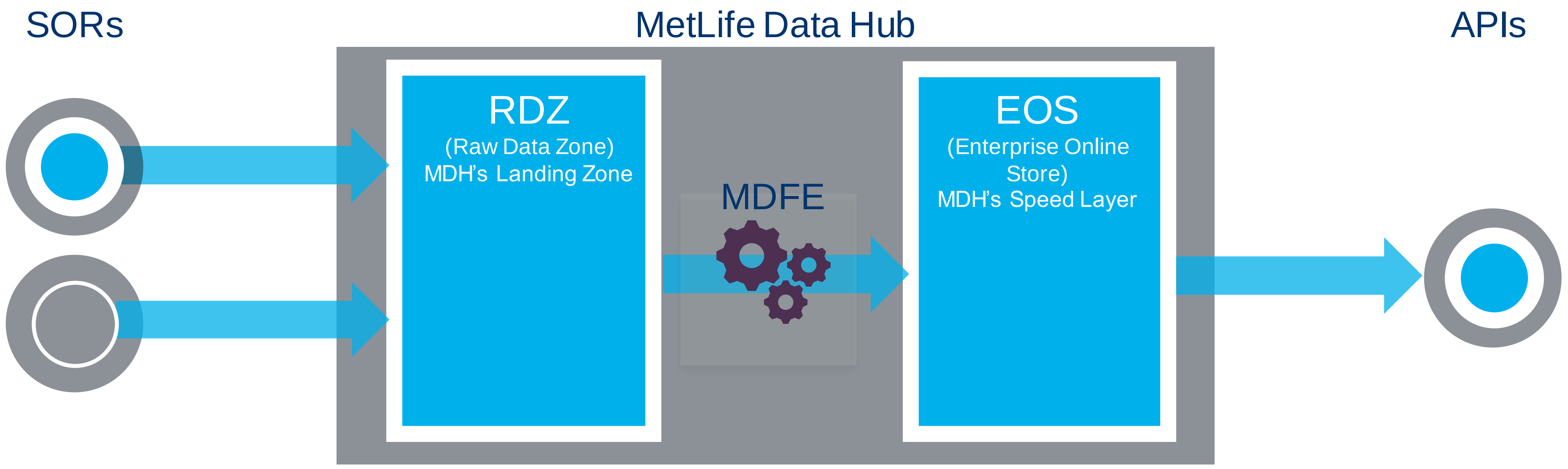
MDFE

MetLife Data Flow Engine

Engine that moves data within MetLife Data Hub



MDFE Overview

MetLife Data Flow Engine is a configuration and metadata driven framework that moves data within the MetLife Data Hub environment. Current version of MDFE is built using custom spark code.

What does MDFE solve for?

Key Constraints

- SORs can communicate data async
- Changes can come in out of sequence
- Latency can change in future
- SOR object hierarchy and EOS object hierarchy can be different

MDFE Solves for

- Data move from RDZ (file, messages) to EOS
- Validation rules
- Transformation to conform to target model
    - Attribute level transformation
    - Relation between objects / entities
    - Aggregate multiple components into an entity
- Relationship management
    - Direct relationships (foreign keys)
    - Inferred – rule based (post processing logic)
    - Orphaned relationships (dirty records)
- Data flow tracking
    - Meta data about source at record / vertex level
    - Process Management
        - Run logs, metrics, rows / vertexes tied to the process

MDFE Coding vs. Config

Common operations through config

Configuration driven

- Attribute level Validations
- Attribute mapping
- Attribute Transformations
- Key generation
- Relationships
- Composite objects

Externalization to feed into data lineage analysis

Vs.

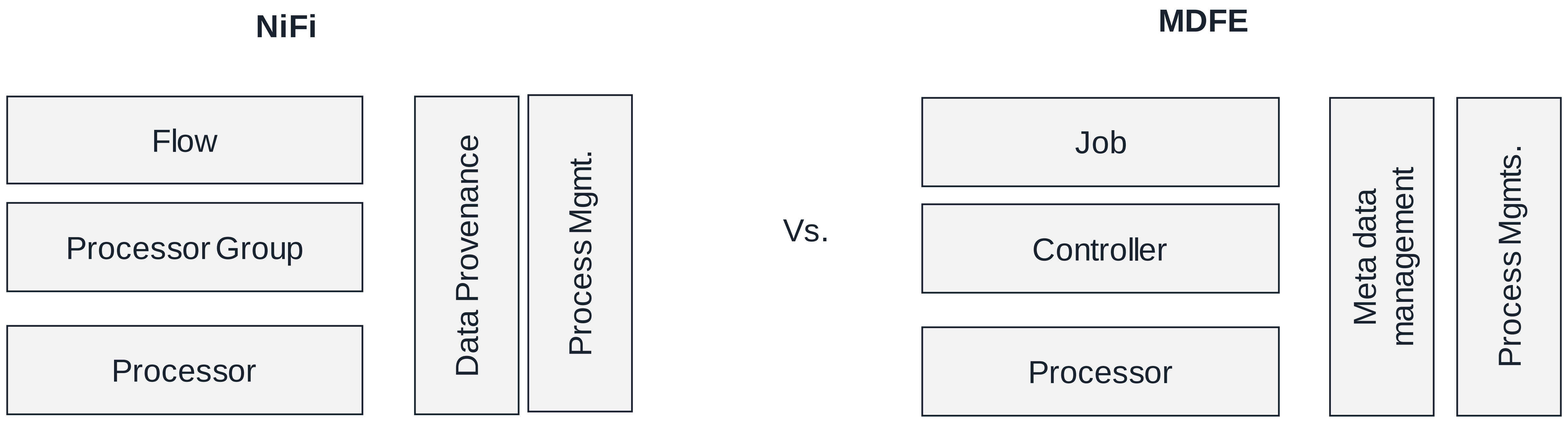
Code Driven

- Complex aggregations
    - Pluggable rules / post processing

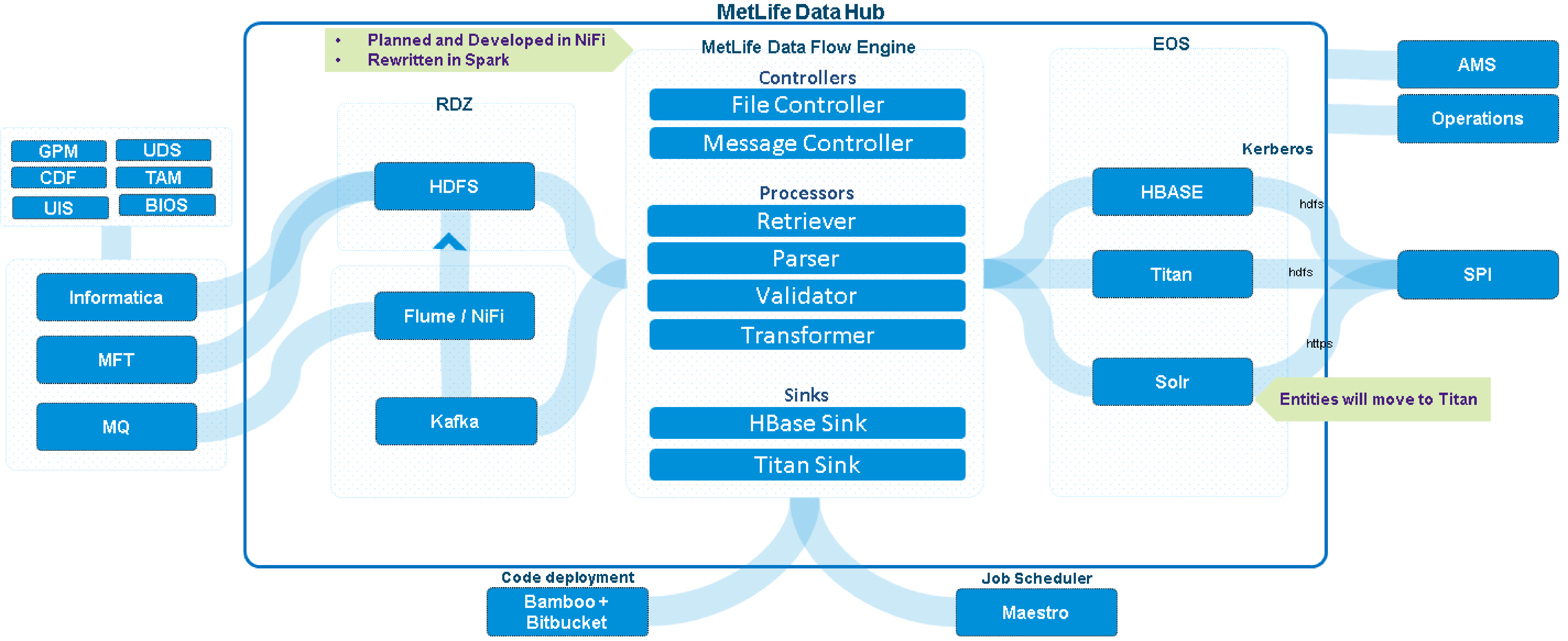
- SOR file level nuances
    - UIS delta file vs. one time differences
    - Generating inferred plan versions for UDS

Note: Graph schema is configured in MDFE, but Titan indexes are created and managed outside MDFE

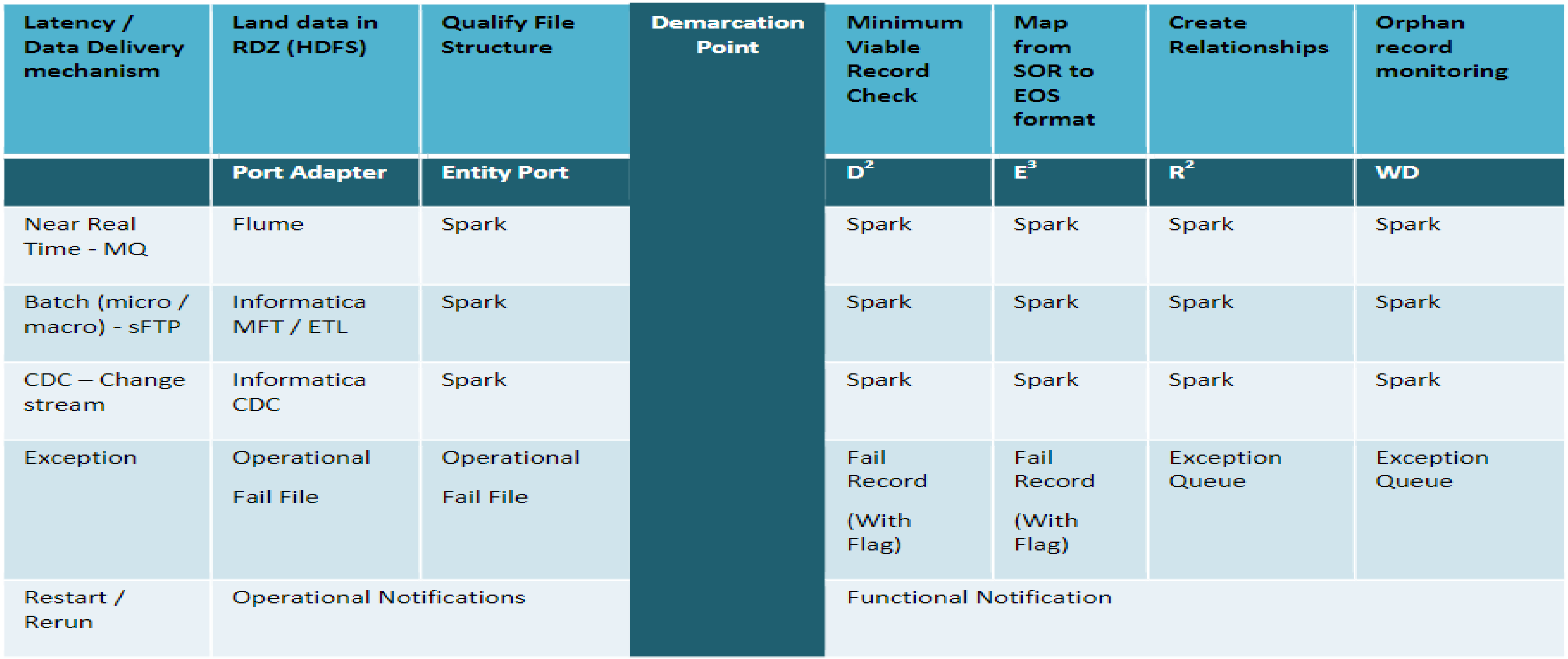
Key MDFE concepts are borrowed from NiFi World …



MDFE was originally created for NiFi … and then refactored and optimized for Spark
(even the naming conventions of the processors follow NiFi conventions)



MDH Data Flow – A high level view

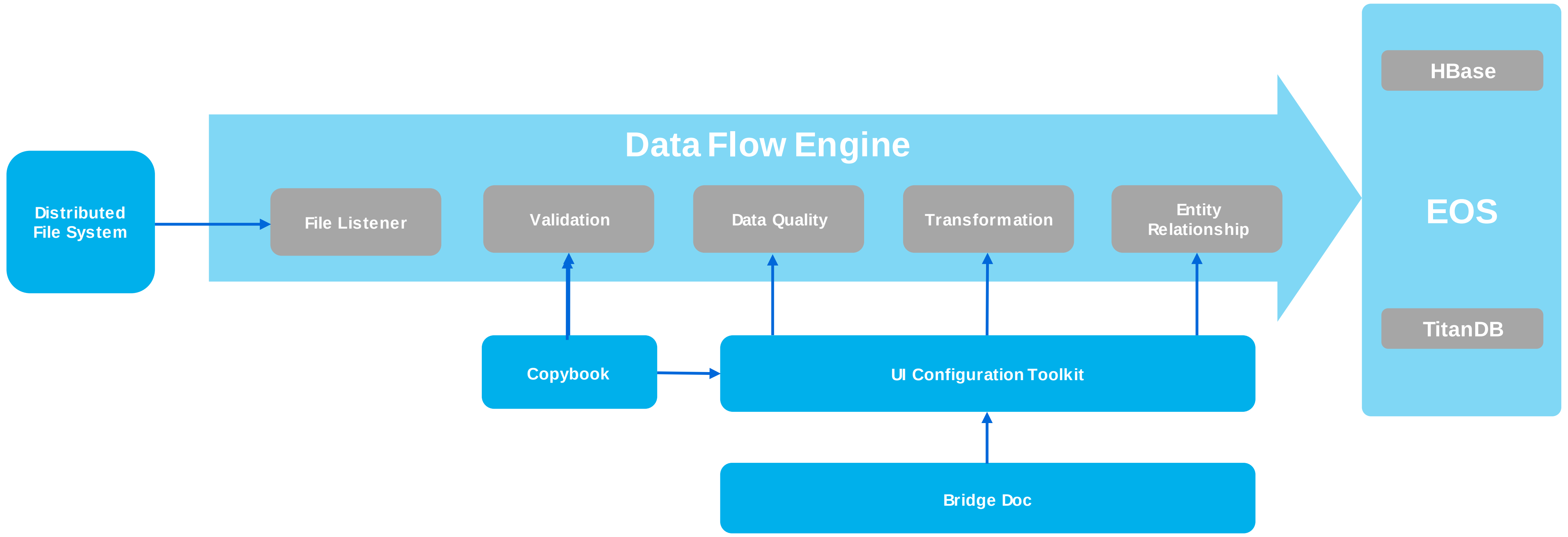


Latency / Data Delivery mechanism	Land data in RDZ (HDFS)	Qualify File Structure	Demarcation Point	Minimum Viable Record Check	Map from SOR to EOS format	Create Relationships	Orphan record monitoring
	Port Adapter	Entity Port		D^2	E^3	R^2	WD
Near Real Time - MQ	Flume	Spark		Spark	Spark	Spark	Spark
Batch (micro / macro) - sFTP	Informatica MFT / ETL	Spark		Spark	Spark	Spark	Spark
CDC – Change stream	Informatica CDC	Spark		Spark	Spark	Spark	Spark
Exception	Operational Fail File	Operational Fail File		Fail Record (With Flag)	Fail Record (With Flag)	Exception Queue	Exception Queue
Restart / Rerun	Operational Notifications			Functional Notification			

Why MDFE?

To make data coming from disparate SOR systems in various formats and latencies available on MDH's speed layer, along with Metadata, Metaprocess and relationships attached to it.

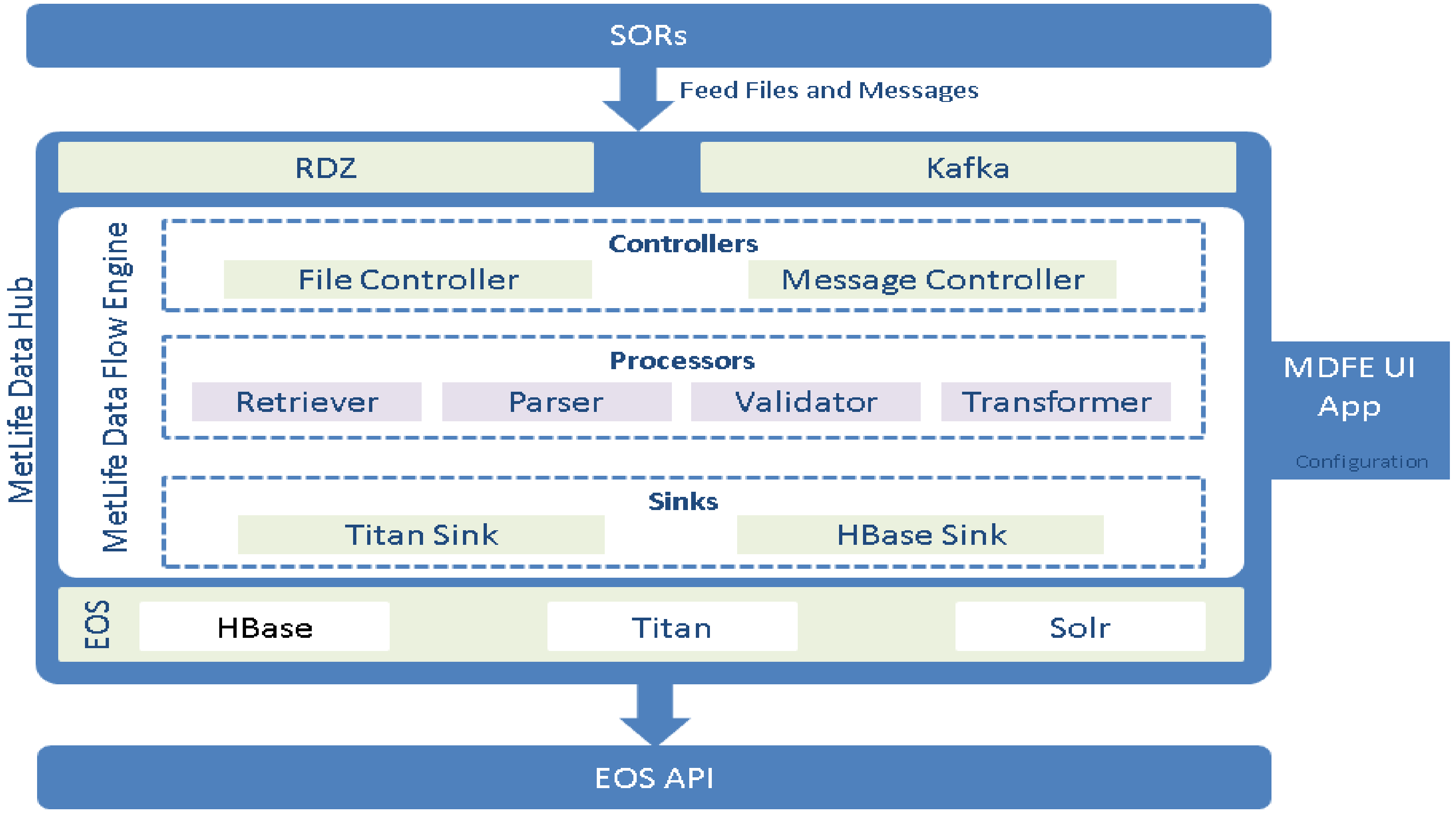
Latencies - Near real time: <5 min, Micro batch: 5-60 min, Macro batch: >60 min and < 24hrs



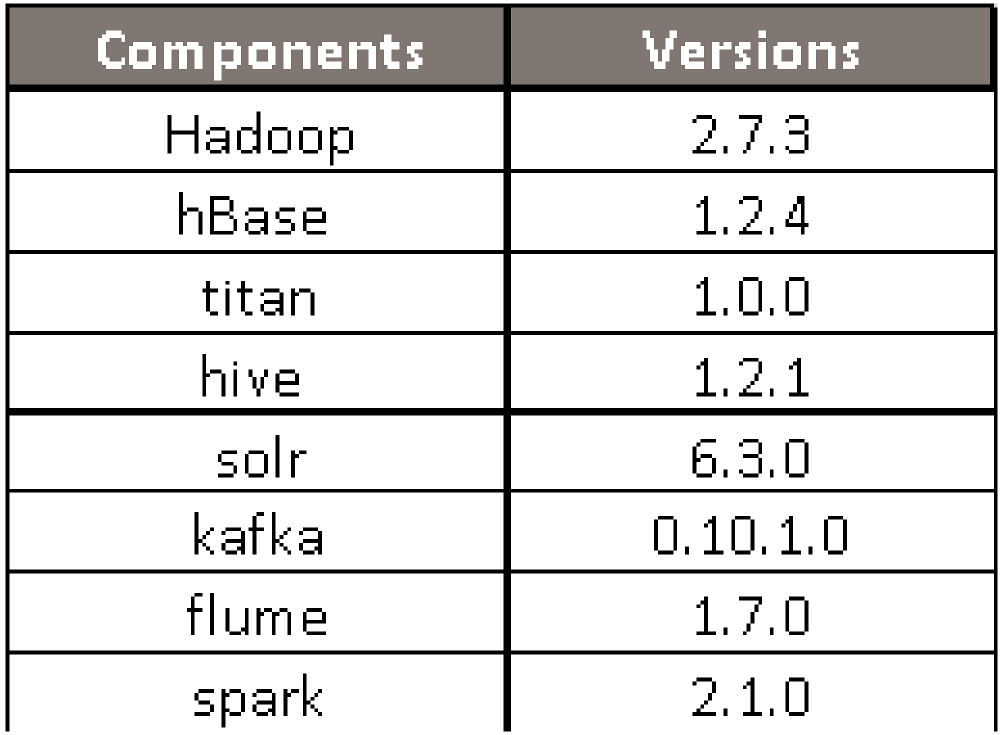
What does it do ?

MDFE Ingesting data into EOS, a simplified view

MDFE Code Structure



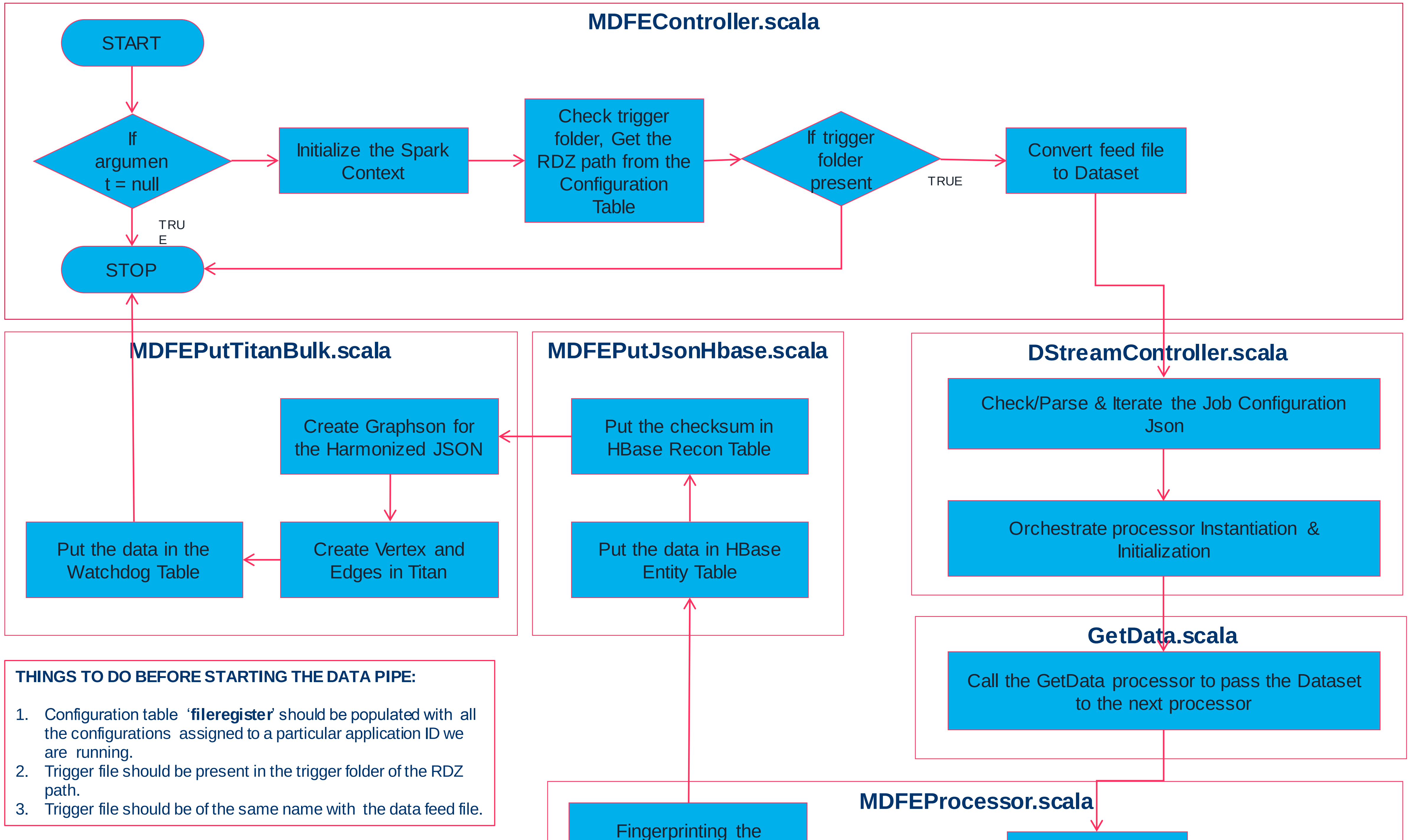
MDFE Component Structure



Components	Versions
Hadoop	2.7.3
hBase	1.2.4
titan	1.0.0
hive	1.2.1
solr	6.3.0
kafka	0.10.1.0
flume	1.7.0
spark	2.1.0

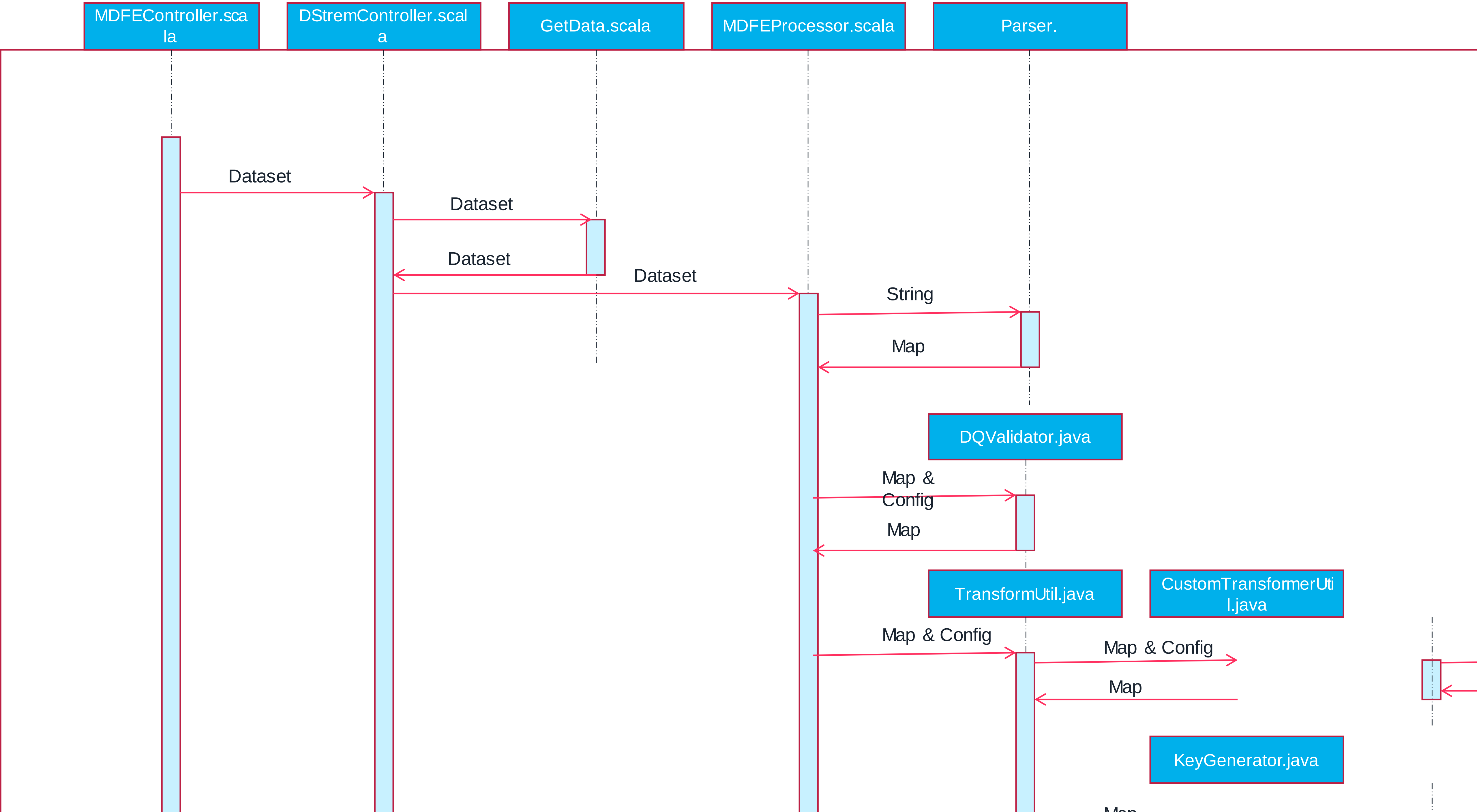
Tech Stack

MDFE is a spark based engine, which runs on IBM Big Insights Platform.


MDFEController.scala
START
If argumen t = null
TRU E
STOP
Initialize the Spark Context
Check trigger folder, Get the RDZ path from the Configuration Table
If trigger folder present
TRUE
Convert feed file to Dataset
MDFEPutTitanBulk.scala
Create Graphson for the Harmonized JSON
Put the data in the Watchdog Table
Create Vertex and Edges in Titan
MDFEPutJsonHbase.scala
Put the checksum in HBase Recon Table
Put the data in HBase Entity Table
DStreamController.scala
Check/Parse & Iterate the Job Configuration Json
Orchestrate processor Instantiation & Initialization
GetData.scala
Call the GetData processor to pass the Dataset to the next processor
MDFEProcessor.scala
Fingerprinting the

THINGS TO DO BEFORE STARTING THE DATA PIPE:

1. Configuration table '**fileregister**' should be populated with all the configurations assigned to a particular application ID we are running.
2. Trigger file should be present in the trigger folder of the RDZ path.
3. Trigger file should be of the same name with the data feed file.


MDFEController.scala
DStremController.scala
GetData.scala
MDFEProcessor.scala
Parser.
Dataset
Dataset
Dataset
Dataset
String
Map
DQValidator.java
Map & Config
Map
TransformUtil.java
CustomTransformerUtil.java
Map & Config
Map & Config
Map
KeyGenerator.java

Plan	Code	Build	Test	Release	Deploy	Run
JIRA	BitBucket	Bamboo	Bamboo	UDeploy	UDeploy	Maestro

1

Build & Package : Bamboo

- Build Jar
- Create tar file containing Jar and script files
- Create a release version
- URL:

2

Repository : Bitbucket

Development Branch: depends on the relase
Baselined Version : ***
URL:

3

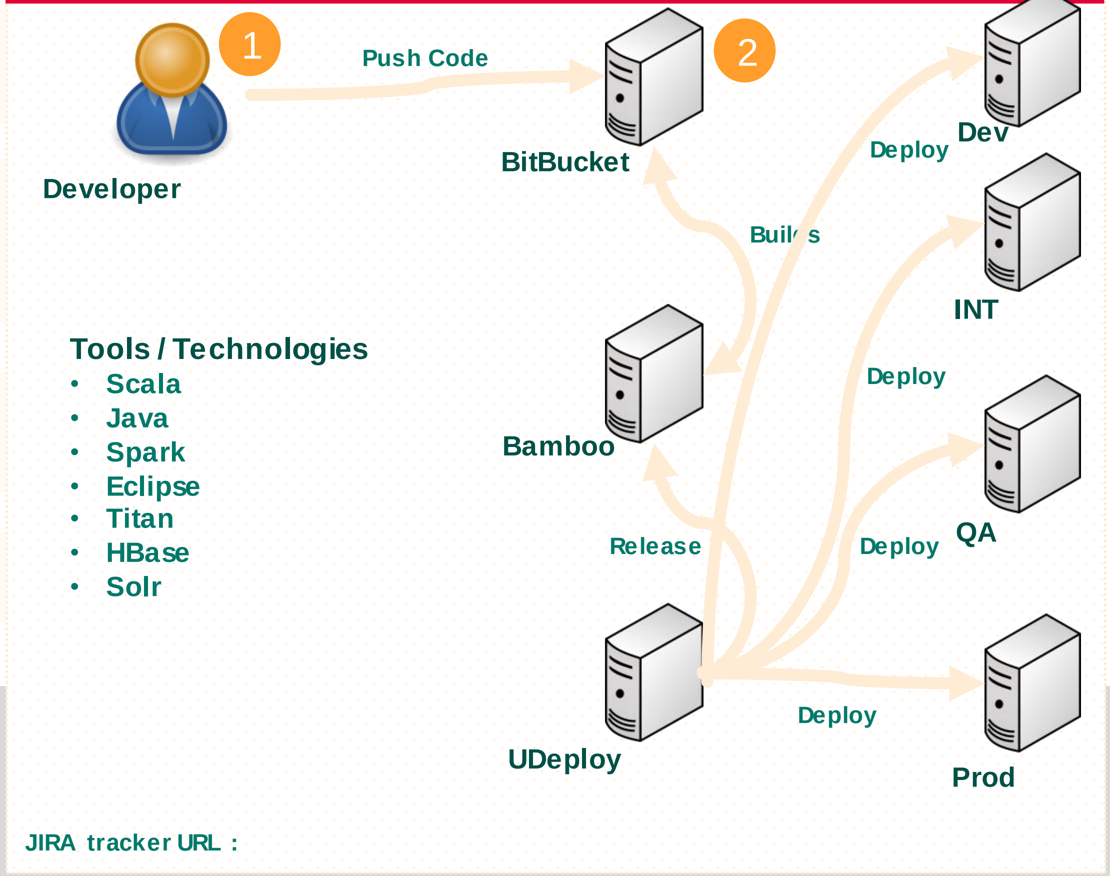
Deploy : Udeploy [Not in Prod]

- Create a Snapshot of the baselined version - Eg: 10856-MDFE-Rel2018.1.0-<relDate>
- Open tickets and get necessary approvals to run Prod deployment procedure

4

Run: Maestro

DevOps Flow



Tools / Technologies

- Scala
- Java
- Spark
- Eclipse
- Titan
- HBase
- Solr

JIRA tracker URL :

How do we build and Deploy

etLife Data Flow Engine is a custom built engine that moves data within the MetLife Data Hub environment.

Review Process

Typical reviews include

- High level and detailed design document reviews and manual and automated code reviews and security reviews (veracode scans)
- Architecture reviews to confirm that the development confirms to the architecture

Current status

- MetLife and Cognizant Architecture and Development teams have completed few rounds of review
- Knowledgent has completed design and code reviews

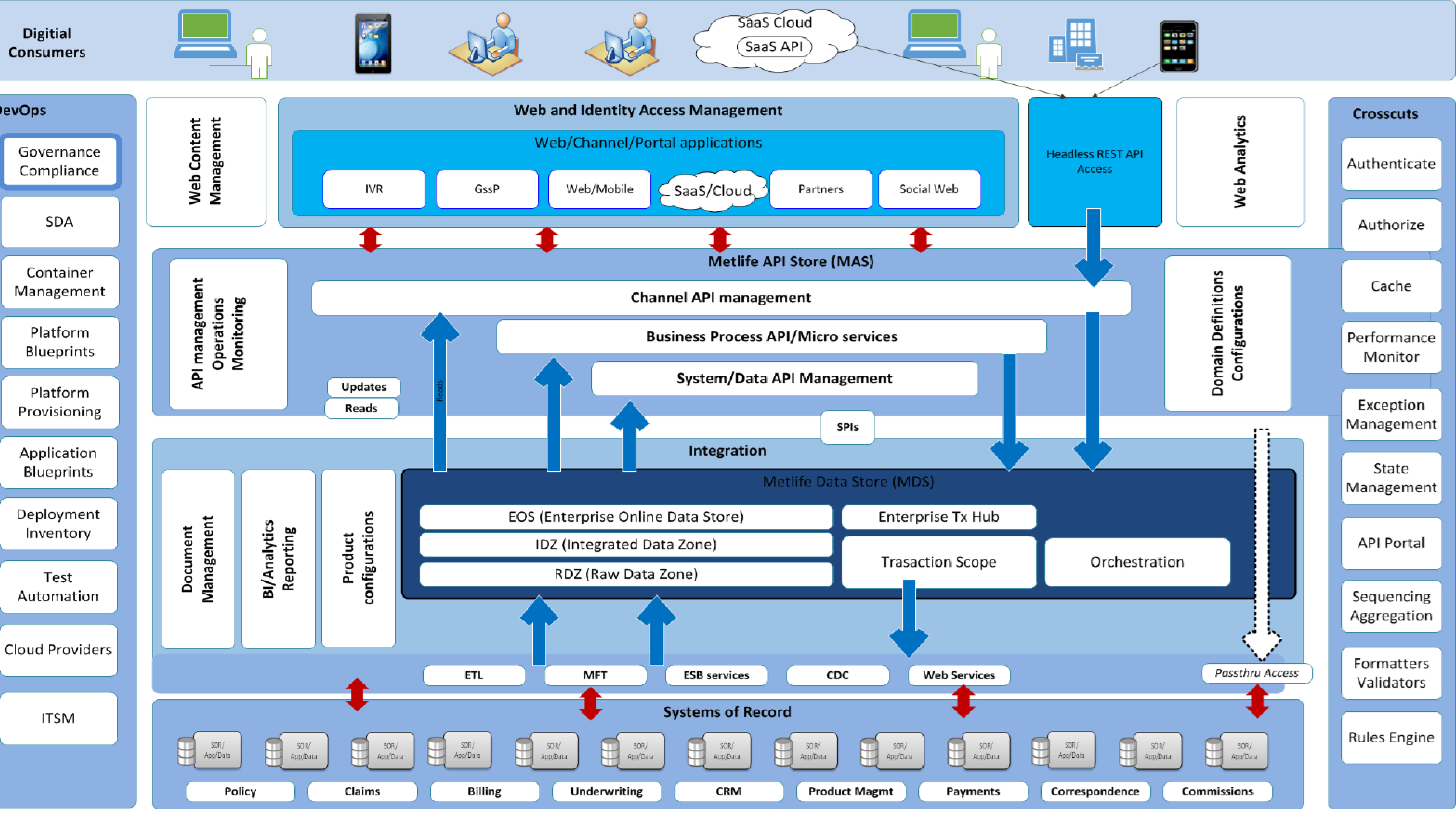
Design Document Reviews

Sharepoint links ->

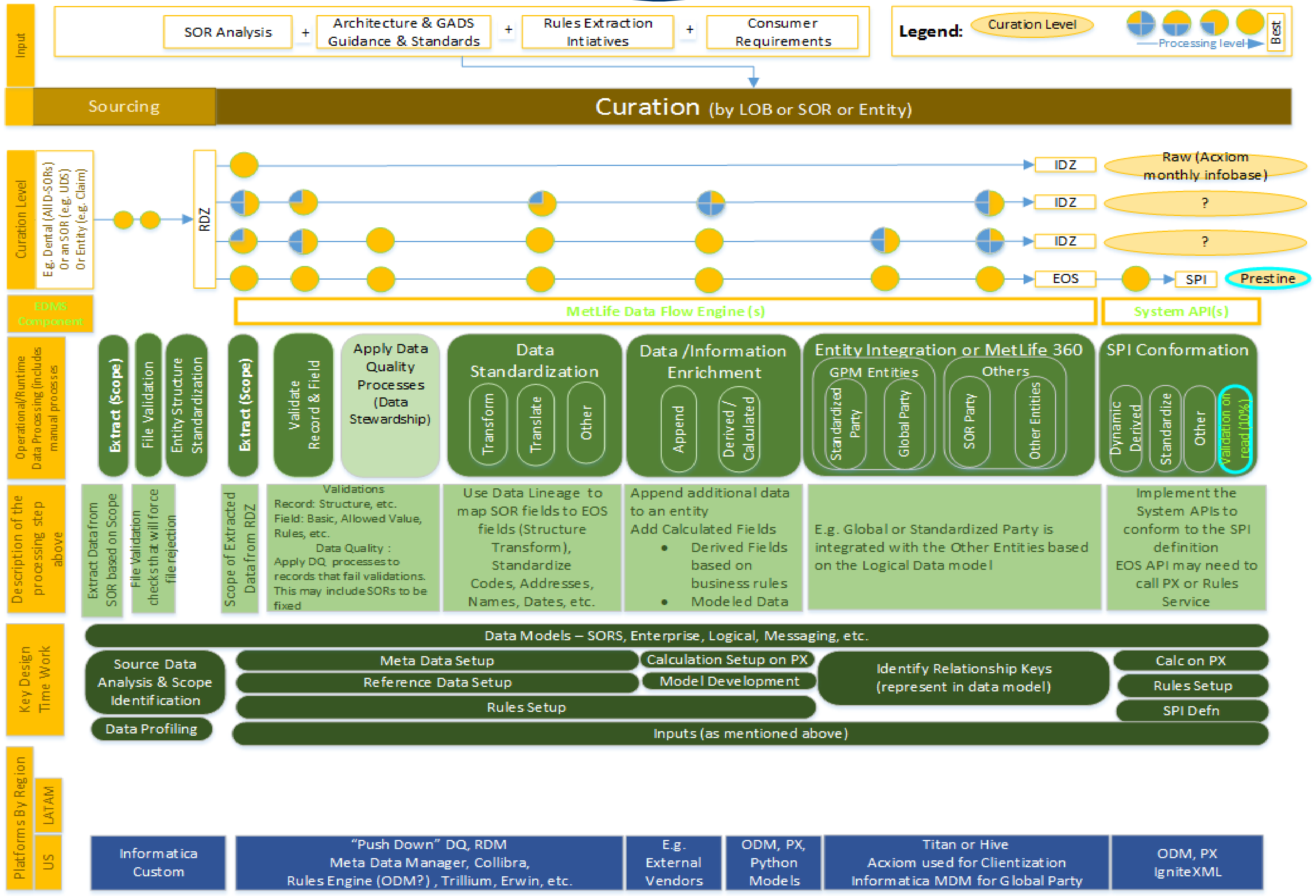
Code Reviews

Bitbucket location ->

Appendix


Digitial Consumers
SaaS Cloud
SaaS API
evOps
Governance Compliance
SDA
Container Management
Platform Blueprints
Platform Provisioning
Application Blueprints
Deployment Inventory
Test Automation
Cloud Providers
ITSM
Web Content Management
Web and Identity Access Management
Web/Channel/Portal applications
IVR
GssP
Web/Mobile
SaaS/Cloud
Partners
Social Web
Headless REST API Access
Web Analytics
Crosscuts
Authenticate
Authorize
Cache
Performance Monitor
Exception Management
State Management
API Portal
Sequencing Aggregation
Formatters Validators
Rules Engine
Metlife API Store (MAS)
API management Operations Monitoring
Channel API management
Business Process API/Micro services
System/Data API Management
Domain Definitions Configurations
Updates
Reads
Reads
SPIs
Integration
Document Management
BI/Analytics Reporting
Product configurations
Metlife Data Store (MDS)
EOS (Enterprise Online Data Store)
IDZ (Integrated Data Zone)
RDZ (Raw Data Zone)
Enterprise Tx Hub
Trasaction Scope
Orchestration
ETL
MFT
ESB services
CDC
Web Services
Passthru Access
Systems of Record
SOR/App/Data
Policy
Claims
Billing
Underwriting
CRM
Product Magmt
Payments
Correspondence
Commissions

Data Curation


Input
SOR Analysis
+
Architecture & GADS Guidance & Standards
+
Rules Extraction Intiatives
+
Consumer Requirements
Legend:
Curation Level
Processing level
Best
Sourcing
Curation (by LOB or SOR or Entity)
Curation Level
Eg. Dental (All ID-SORs) Or an SOR (e.g. UDS) Or Entity (e.g. Claim)
RDZ
IDZ
Raw (Acxiom monthly infobase)
IDZ
?
IDZ
?
EOS
SPI
Prestine
EDMS Component
MetLife Data Flow Engine (s)
System API(s)
Operational/Runtime Data Processing (includes manual processes
Extract (Scope)
File Validation
Entity Structure Standardization
Extract (Scope)
Validate Record & Field
Apply Data Quality Processes (Data Stewardship)
Data Standardization
Transform
Translate
Other
Data /Information Enrichment
Append
Derived / Calculated
Entity Integration or MetLife 360
GPM Entities
Standardized Party
Global Party
Others
SOR Party
Other Entities
SPI Conformation
Dynamic Derived
Standardize
Other
Validation on read (10%)
Description of the processing step above
Key Design Time Work
Data Models – SORS, Enterprise, Logical, Messaging, etc.
Source Data Analysis & Scope Identification
Data Profiling
Meta Data Setup
Reference Data Setup
Rules Setup
Calculation Setup on PX
Model Development
Identify Relationship Keys (represent in data model)
Calc on PX
Rules Setup
SPI Defn
Inputs (as mentioned above)
Platforms By Region
LATAM
US

Extract Data from SOR based on Scope

File Validation checks that will force file rejection

Scope of Extracted Data from RDZ

Validations
Record: Structure, etc.
Field: Basic, Allowed Value, Rules, etc.
Data Quality :
Apply DQ processes to records that fail validations. This may include SORs to be fixed

Use Data Lineage to map SOR fields to EOS fields (Structure Transform), Standardize Codes, Addresses, Names, Dates, etc.

Append additional data to an entity
Add Calculated Fields
- Derived Fields based on business rules
- Modeled Data

E.g. Global or Standardized Party is integrated with the Other Entities based on the Logical Data model

Implement the System APIs to conform to the SPI definition
EOS API may need to call PX or Rules Service

Informatica Custom

"Push Down" DQ, RDM
Meta Data Manager, Collibra,
Rules Engine (ODM?) , Trillium, Erwin, etc.

E.g. External Vendors

ODM, PX, Python Models

Titan or Hive
Acxiom used for Clientization
Informatica MDM for Global Party

ODM, PX
IgniteXML