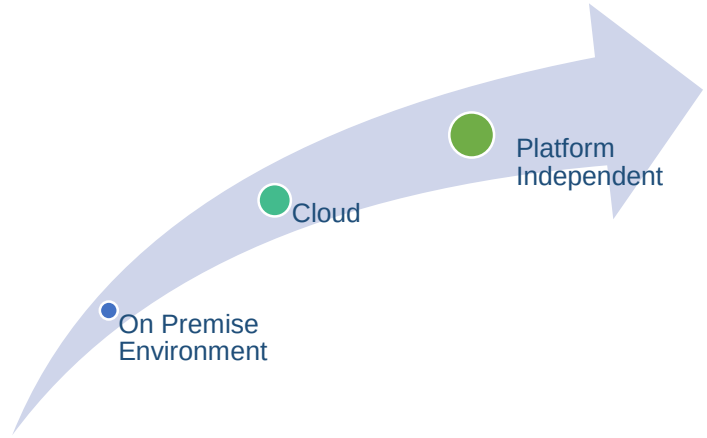
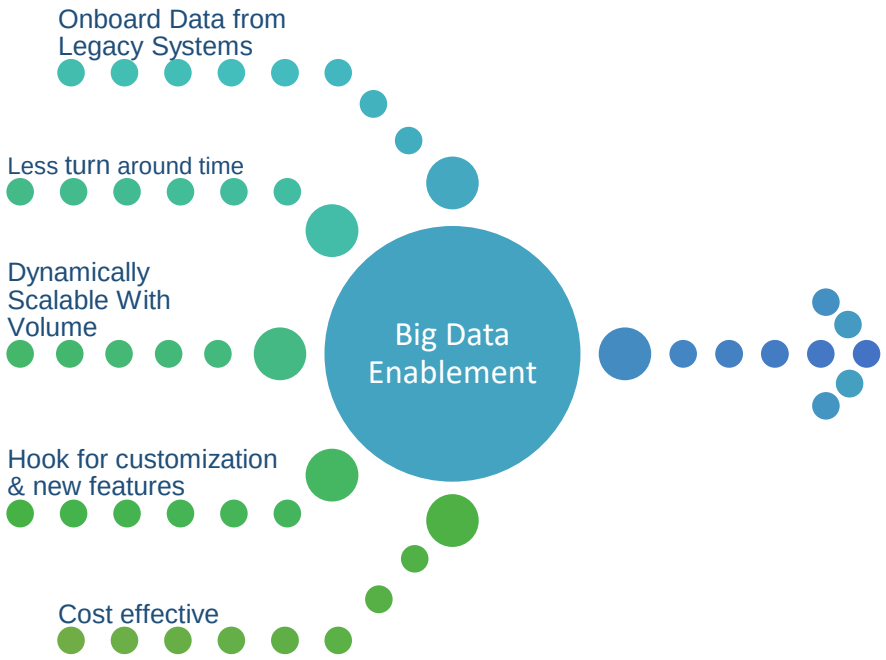
Cloud Enabled Ingestion Framework - Introduction





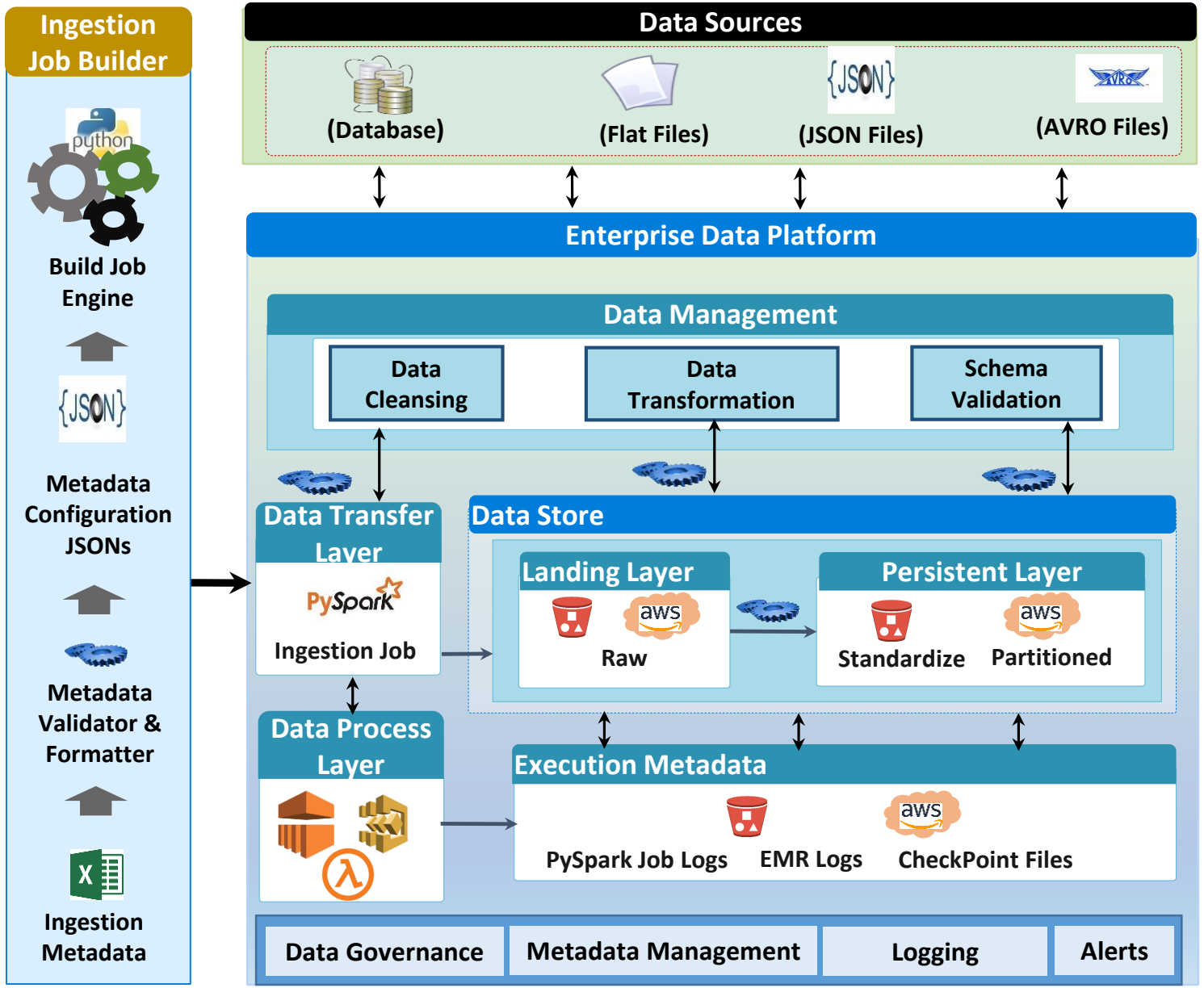
OPPORTUNITY

- *As industry is maturing overall w.r.t. Big Data Technologies, most of the customers are planning to shift towards cloud platform. Keeping that in mind, there should be a solution which should be platform independent – On-Premise, Cloud (AWS, Azure, Google), Distribution agnostic keeping the following artifacts tightly coupled:*
  - *Portability across different platforms*
  - *Optimized deployment time*
  - *Optimized Cost w.r.t. development and customization*
  - *Scalability*

SOLUTION / APPROACH

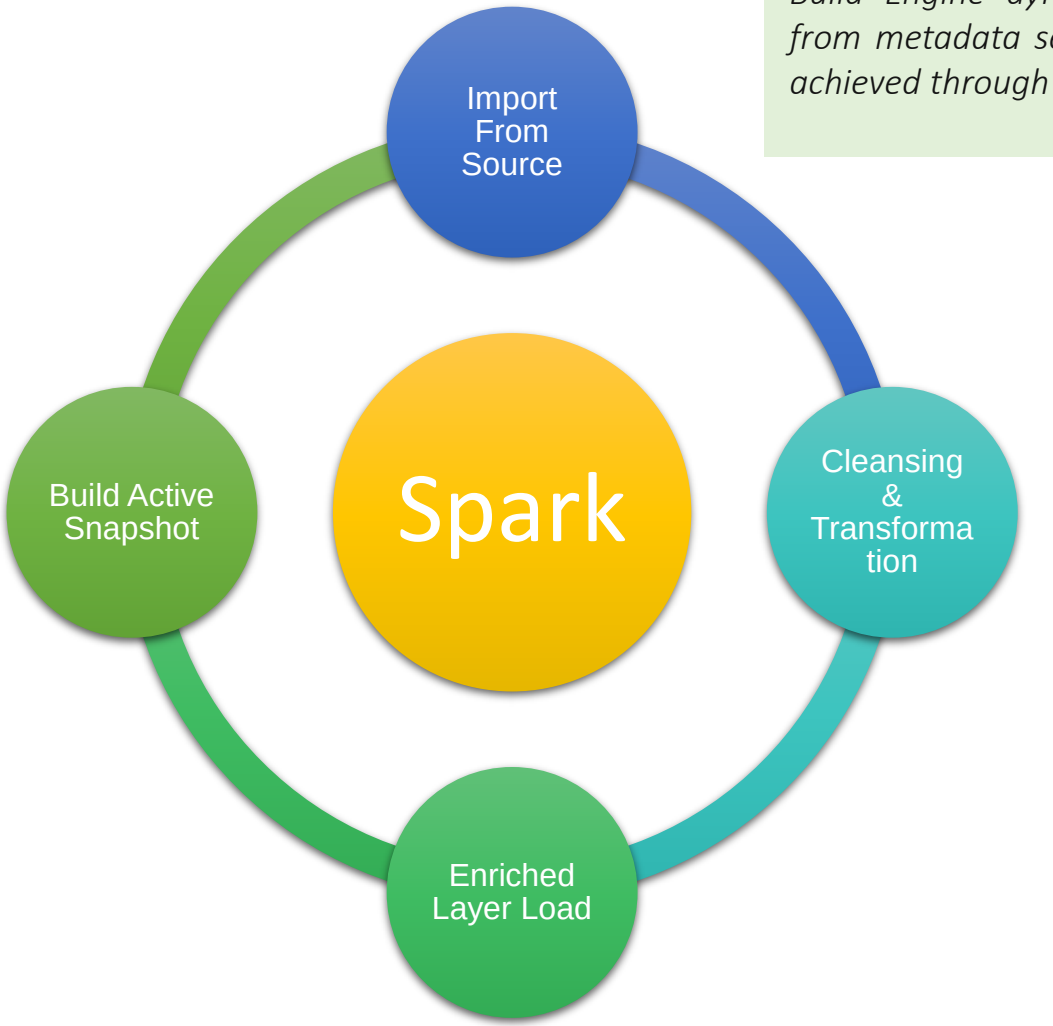
1. *Pure Spark based solution can be ported to any platform*
2. *Platform based API oriented solution to Read and Store Data*
3. *Hook for quickly onboard customer specific Cleansing and Transformation rules*
4. *Code Build Engine dynamically built the Spark jobs from Metadata keeping restart ability in mind from failure point*
5. *Metadata based Execution Engine dynamically orchestrates between several feeds within a source (Serial, Parallel, Predecessor dependencies)*
6. *Framework will be easily deployable across Client environments as Build Engine is Python based solution*

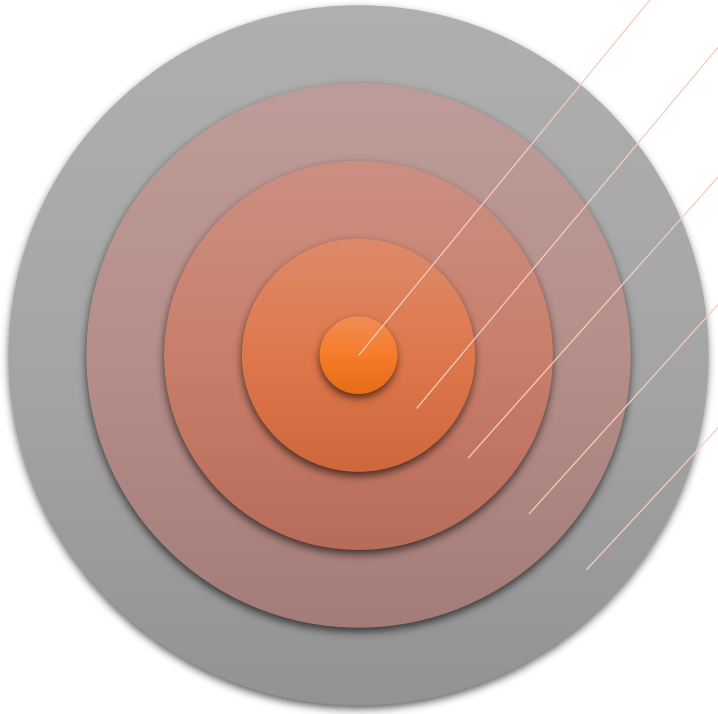
Framework Conceptual Architecture



Design Principles

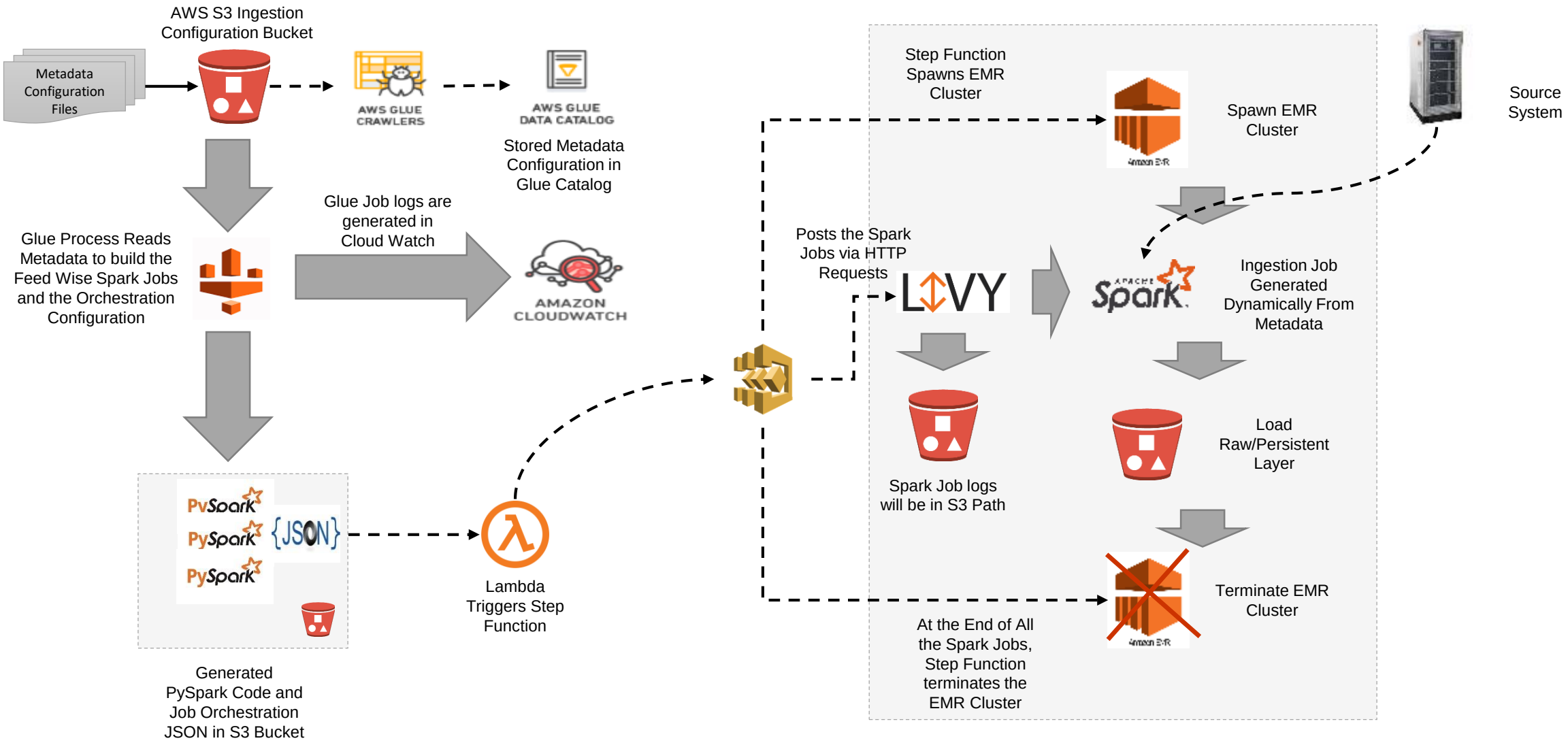
Build Engine dynamically builds the Spark Jobs from metadata setup. All the data flow steps are achieved through PySpark.



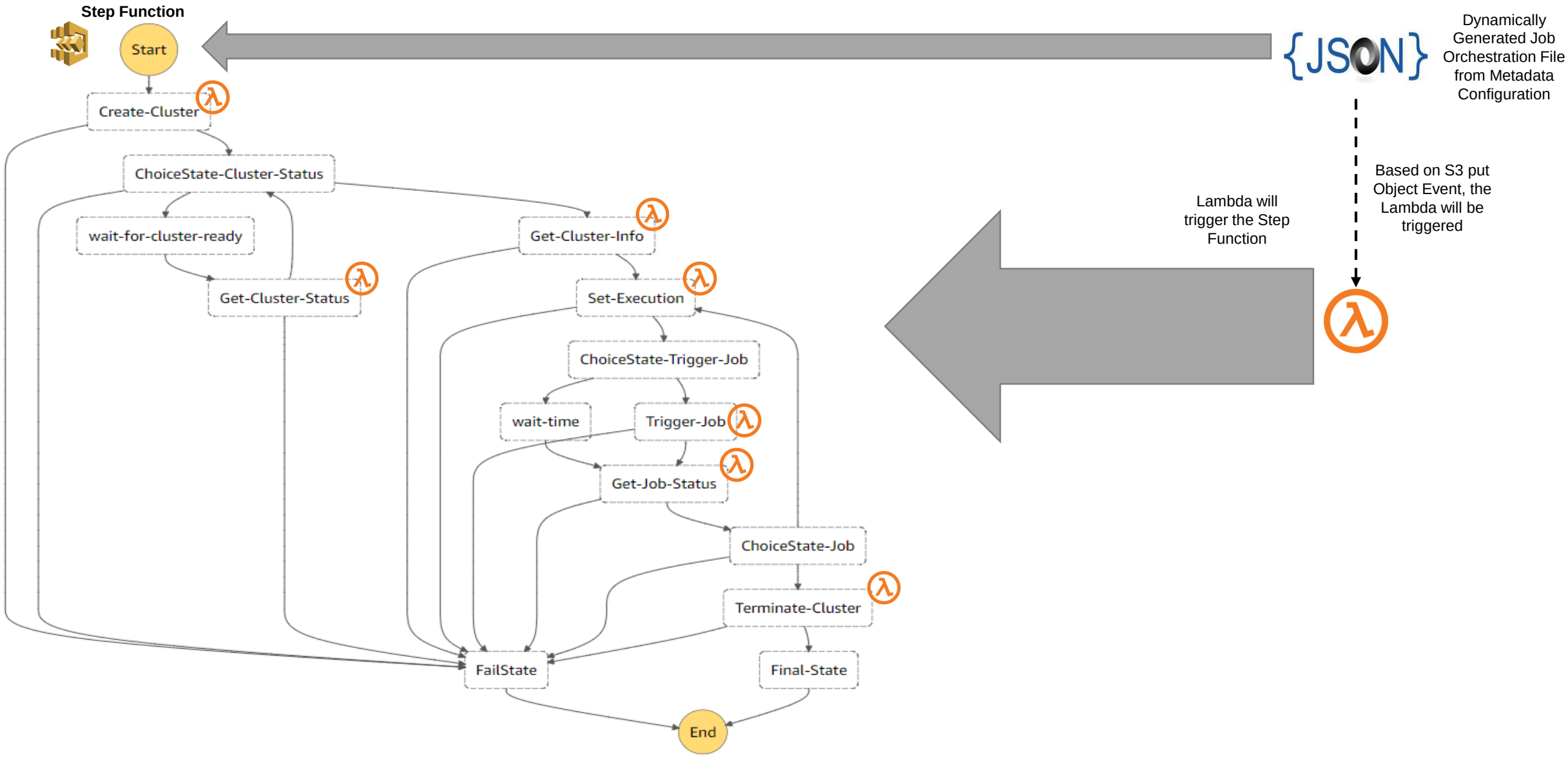


- Platform Independent
- Metadata Driven
- Loosely Coupled Components
- Metaprogrammed
- Dynamic Code Build Engine and Execution Engine

Framework Design - AWS



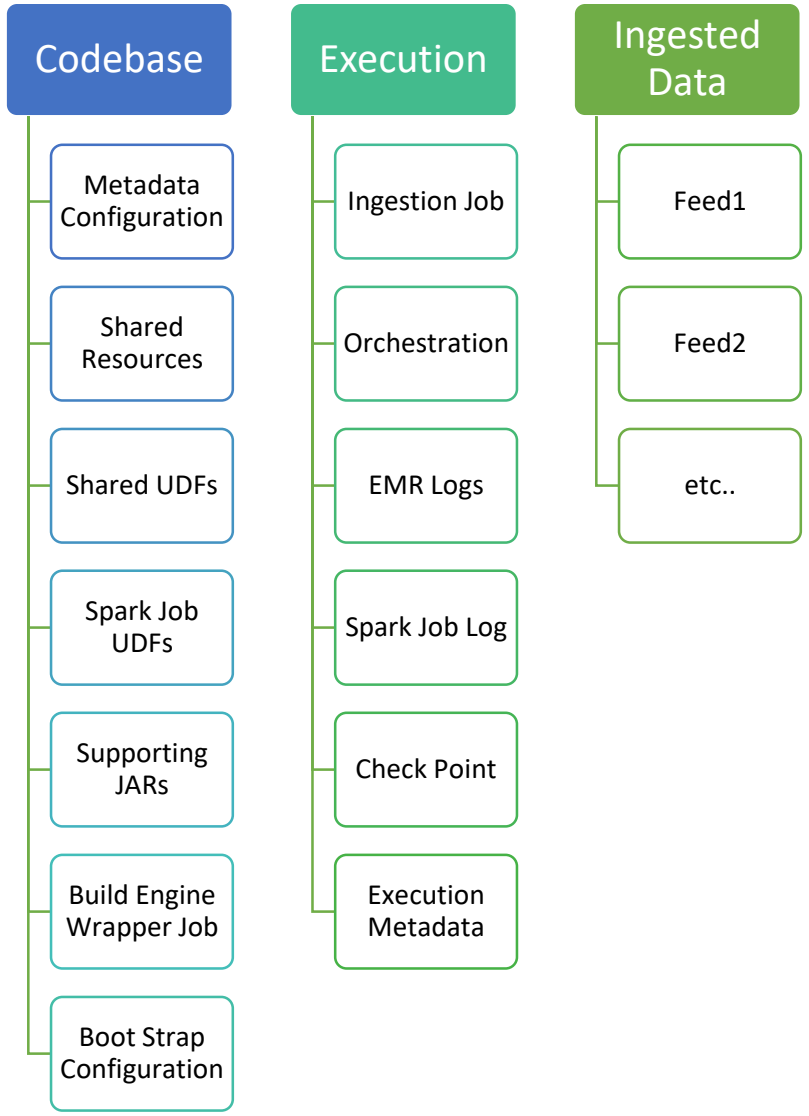
Orchestration Design - AWS



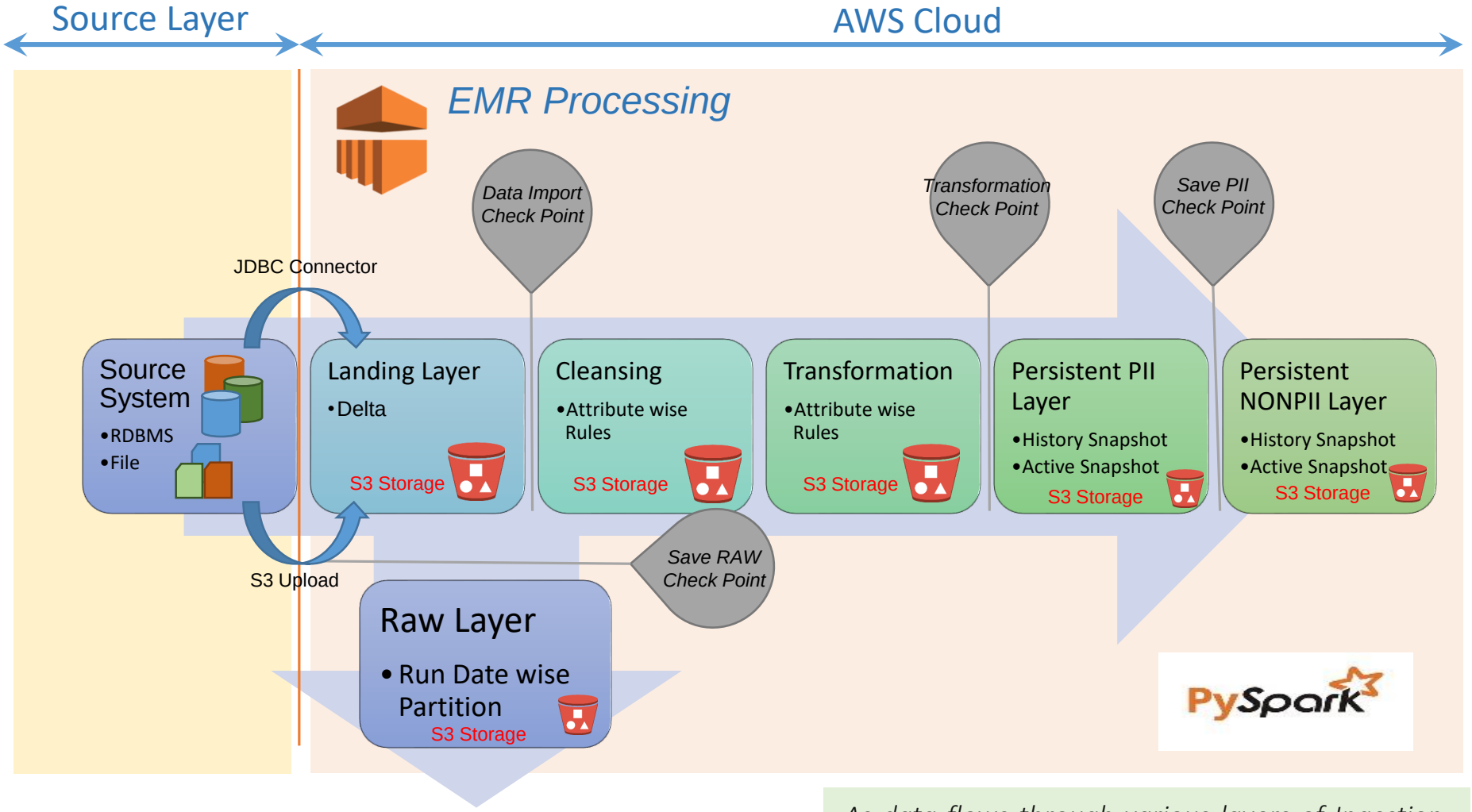
Directory Structure

Codebase Folder	Purpose
Metadata Configuration	*Metadata configuration for Source, Feed, Feed Attributes*
Shared Resources	*Common configuration setup files*
Shared UDFs	*UDFs to Dynamically generate the PySpark code per Metadata Configuration*
Spark JOB UDFs	*UDFs for Cleansing and Transformation rules*
Supporting JARs	*JAR files required for dynamically generated PySpark Job*
Build Engine Wrapper	*Wrapper script which uses the Shared UDFs to dynamically generate the PySpark Ingestion Job*
Boot Strap Configuration	*Boot Strap configuration script to spawn EMR Cluster*

Execution Folder	Purpose
Ingestion Job	*Generated PySpark Job script*
Orchestration	*Generated Feed wise execution orchestration configuration if ingestion process is executed at source level*
EMR Logs	*EMR Cluster logs for each time a cluster is spinning up*
Spark Job Logs	*Feed wise PySpark job logs*



Data Flow – Feed Level



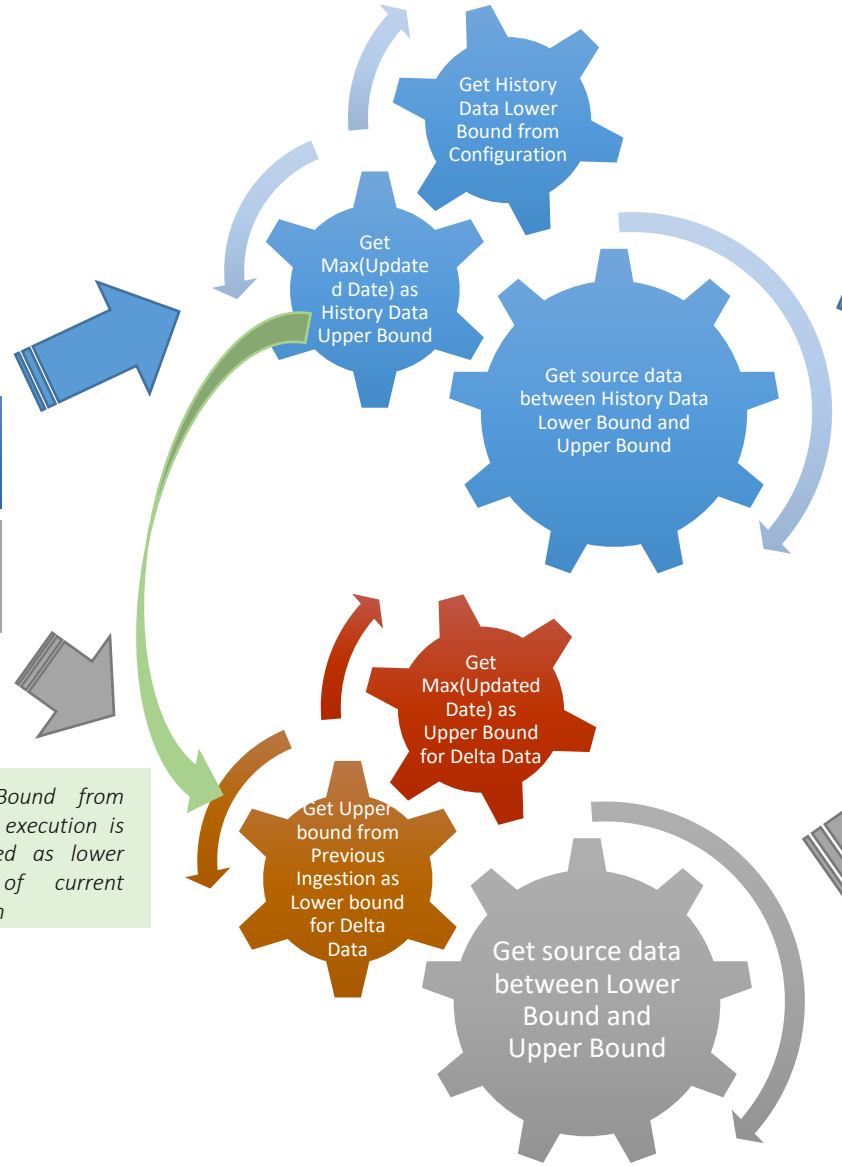
As data flows through various layers of Ingestion, checkpoints are defined and data is saved intermittently for Restart ability and recovery from failure point.

CDC Mechanism – Feed Level

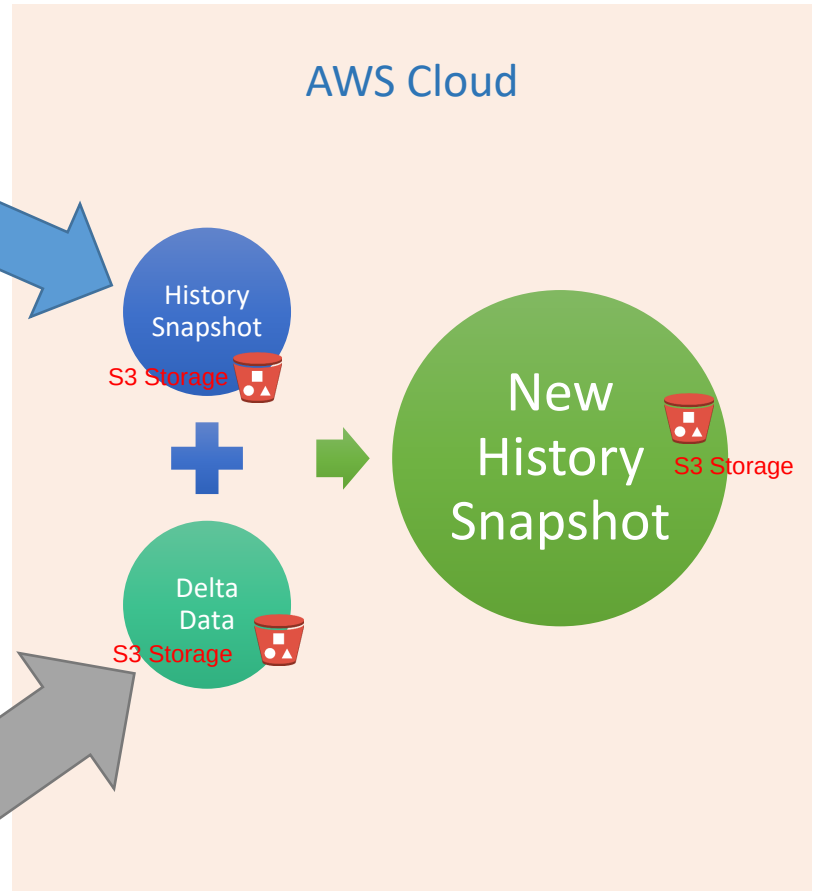
Source Feed has an audit column to identify the CDC data. Consider Updated Date as audit column

History Data
- Updated Date = Day 0

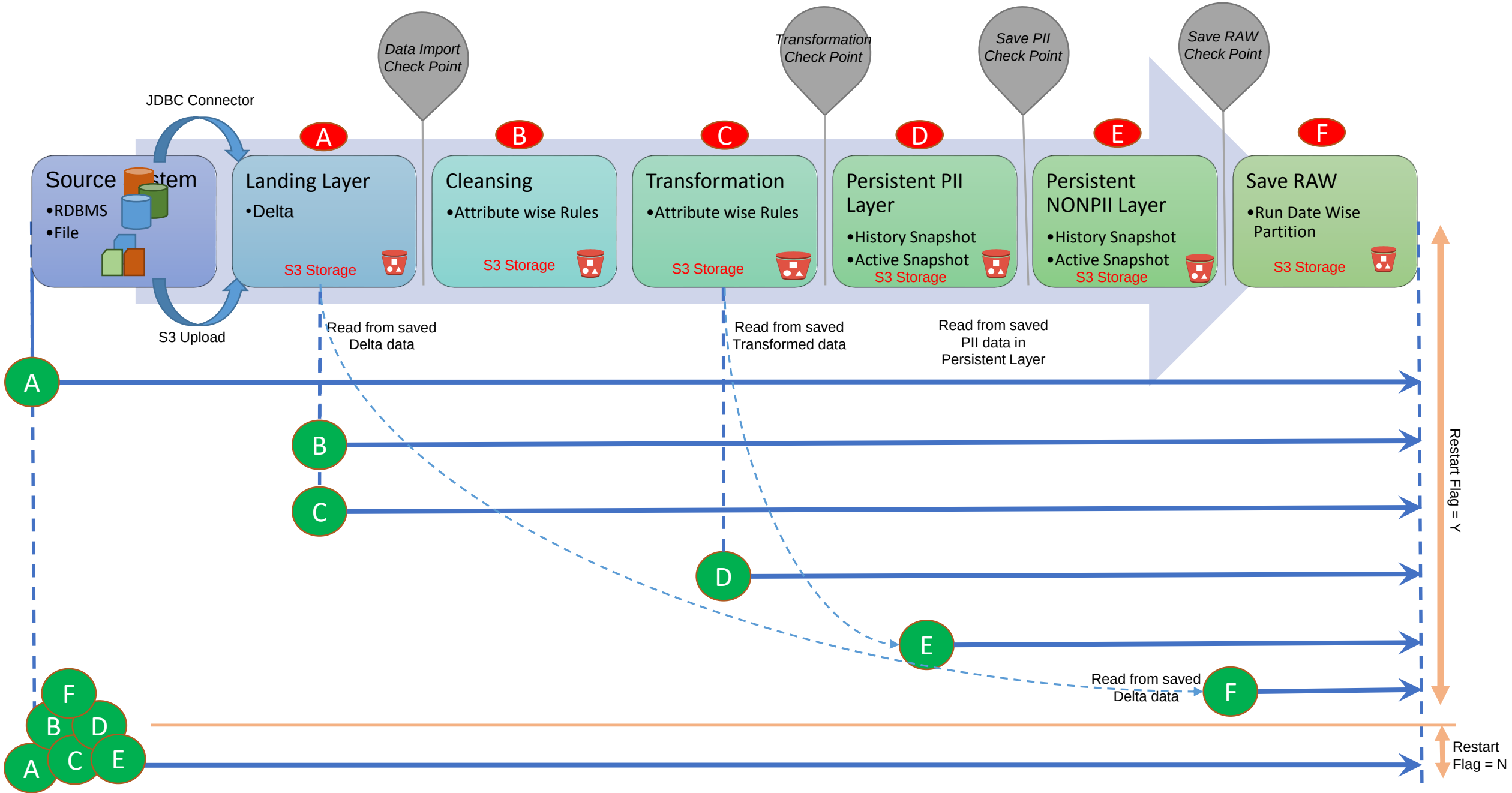
Delta Data
- Updated Date = Day 1



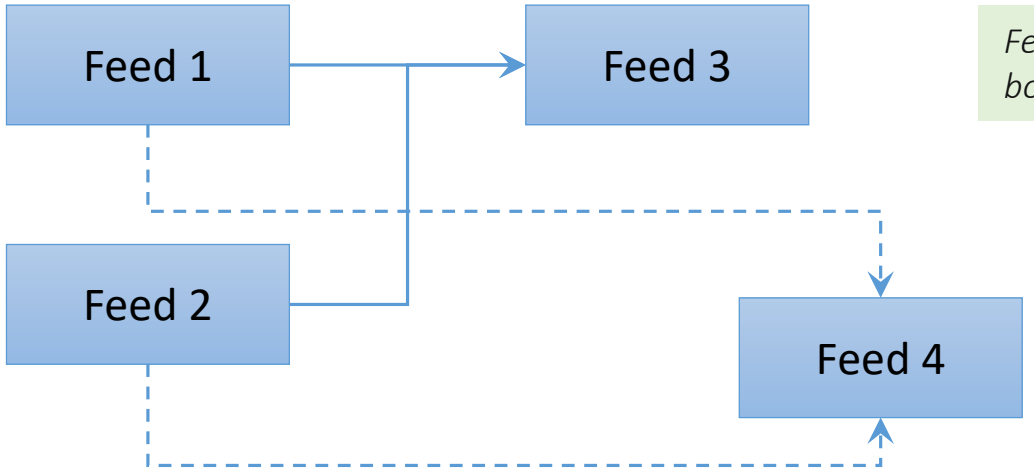
Upper Bound from previous execution is considered as lower bound of current execution



Restart ability – Feed Level



Restart ability – Source Level



Feed3 has predecessor on Feed1 and Feed2. Once both of them are successful, then Feed3 will start

Feed4 has predecessor on either Feed1 and Feed2. Once either of them is successful, then Feed4 will start

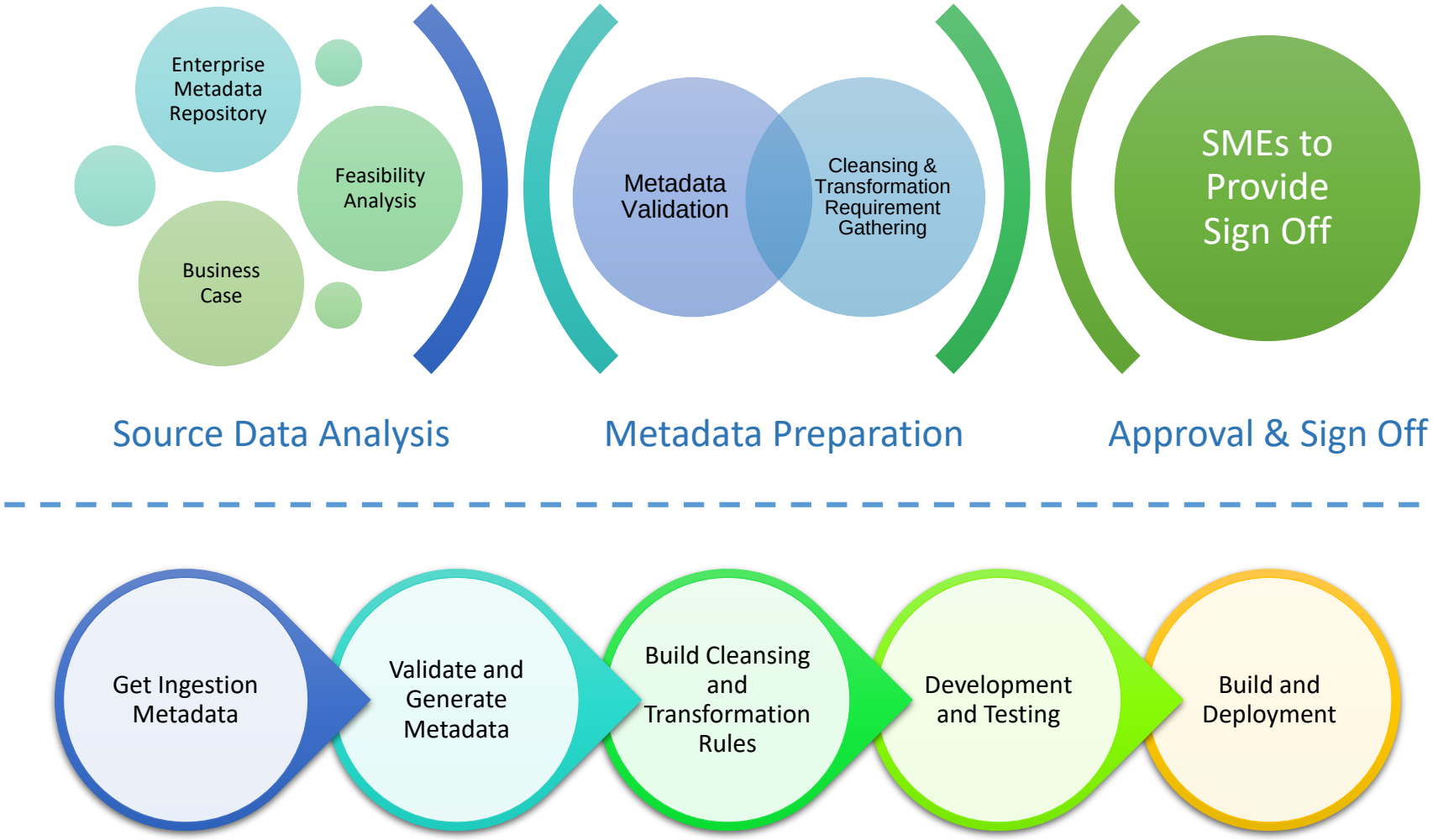
Failure Scenario	Status	On Restart (Restart Flag = Y)
Feed1	*Success*	*Process will only build PySpark job for Feed2 from failure point and Feed3 from beginning*
Feed2	*Failed*	
Feed3	*Skipped*	
Feed4	*Success*	

Failure Scenario	Status	On Restart (Restart Flag = Y)
Feed1	*Success*	*Process will only build PySpark job for Feed3 from failure point and execute Feed3 ingestion*
Feed2	*Success*	
Feed3	*Failed*	
Feed4	*Success*	

Failure Scenario	Status	On Restart (Restart Flag = Y)
Feed1	*Success*	*Process will only build PySpark job for Feed4 from failure point and execute Feed4 ingestion*
Feed2	*Success*	
Feed3	*Success*	
Feed4	*Failed*	

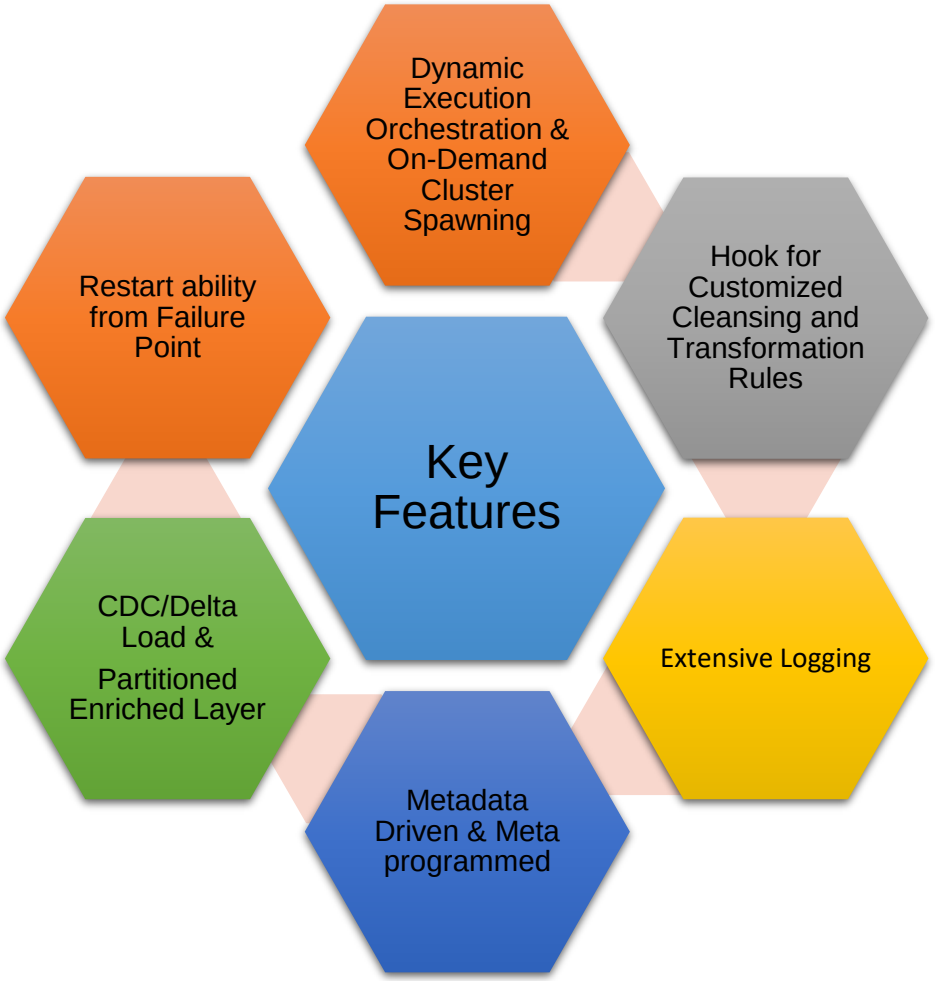
Failure Scenario	Status	On Restart (Restart Flag = Y)
Feed1	*Success*	*Process wont find any failure, so it will exit.*
Feed2	*Success*	
Feed3	*Success*	
Feed4	*Success*	

Development Life Cycle



Current Features

S#	Available Features
1.	*Supports RDBMS – Oracle, SQL Server, TEXT file – Fixed width & delimited*
2.	*Supports data ingestion to Raw & Enriched layer*
3.	*Supports PII and NON PII Ingestion*
4.	*Feed wise Logging mechanism*
5.	*Support data archiving for files on HDFS after processing into Raw & Enriched*
6.	*Support for execution at source or feed level*
7.	*Support for hive table partitioning in the Raw & Enriched layer*
8.	*Dynamically build's the PySpark Code and Spawn EMR Cluster, Executes all the ingestion Source/Feed Level. Once all the ingestion is completed in S3, terminates the cluster.*
9.	*Supports Cleansing and Transformations as configured in Metadata*
10.	*Builds the Job Orchestration configuration dynamically from Metadata configuration and executes the ingestion for different feeds in that fashion* *Within a Source, inter-dependency between feeds (Parallel/Serial) can be configured*
11.	*Supports RAW and Enriched Layer storage in TEXT and Parquet format*
12.	*Supports CDC/Delta Load, Full Load*
13.	*Restart ability within a Source*
14.	*Restart ability within a Feed Ingestion from Failure Point*
15.	*RDBMS Ingestion - Build History and Current Snapshot in Enriched Layer*
16.	*Metadata Generator – Generate JSON metadata configuration files from pre-defined excel spread sheet*



Upcoming Features - Planned

S#	Upcoming Features
1.	*Ingestion of JSON, AVRO File Ingestion*
2.	*Auditing mechanism*
3.	*Support ORC format storage for RAW and Enriched Layer*
4.	*Azure enablement*
5.	*UI Development*