q

**TIBCO Migration Program**  
**Item event PRODUCER streaming**

*March 2024*

|  |  |
| --- | --- |
| Revision: | 1.1 |
| **Date:** | Mar 11, 2024 |
| **Author:** | Suganya Dhandapani(sdhanda2@jcp.com) |

Table of Contents

[1. Introduction 2](#_Toc161408891)

[1.1 Interface Definitions 2](#_Toc161408892)

[1.2 Purpose 2](#_Toc161408893)

[1.3 Document Scope 2](#_Toc161408894)

[1.4 Intended Audience 2](#_Toc161408895)

[1.5 Assumptions & Dependencies 3](#_Toc161408896)

[1.6 Definitions 4](#_Toc161408897)

[2. Current State 6](#_Toc161408898)

[2.1 Poller (EventPublisherAgent) 8](#_Toc161408899)

[2.2 Sparse Event Publication (EntityChangeEvents) 9](#_Toc161408900)

[3. Future State 10](#_Toc161408901)

[3.1 Sequence diagram 13](#_Toc161408902)

[3.2 DESIGN APPROACH 15](#_Toc161408903)

[3.3 TECHNOLOGY Stack 17](#_Toc161408904)

[3.4 ACCEPTANCE CRITERIA/DEFINITION OF DONE 18](#_Toc161408905)

[3.5 EXCEPTION HANDLING & RETRY 18](#_Toc161408906)

[3.6 LOGGING STANDARDS 18](#_Toc161408907)

[3.7 SECURITY 19](#_Toc161408908)

[3.8 PACKAGING & RELEASE 20](#_Toc161408909)

[3.9 REFERENCES 20](#_Toc161408910)

[Appendix A. Fiter Criteria XXI](#_Toc161408911)

[Appendix B. PSEUDO Algorithm and DB Queries XXII](#_Toc161408912)

[Appendix C. ITEM POLLER AND SPARSER – Business works process XXV](#_Toc161408913)

Table of Figures

**No table of figures entries found.**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Rev** | **Date** | **Who** | **Description** |
| 1.0 | 11-Mar-24 | Suganya | Initial creation. |
|  |  |  |  |
|  |  |  |  |

Approvals

|  |  |  |
| --- | --- | --- |
| **Date** | **Approved By** | **Title** |
|  |  |  |

Key Contacts

|  |  |  |
| --- | --- | --- |
| **Name** | **Telephone/Email** | **Role** |
| Phoutty | pkhounborinh@psyncopate.com | Enterprise Architect |
| Ashley | amillette@psyncopate.com | Solutions Architect |
| Suganya | sdhandapani@psyncopate.com | Solutions Architect |
| Venkat | vthandapani@psyncopate.com | Developer |
| Jayasree | jsivaraj@psyncopate.com | Project / Release Manager |
| Rajeev | ralagarsamy@psyncopate.com | Delivery Manager |
| Amrish | akothar1@jcp.com | Sr Technical Project Manager ( JCP ) |
| Milton | mpatri15@jcp.com | Sr Manager ( JCP ) |
| Brian | bpatton@jcp.com | Lead Engineer ( JCP ) |
| Cole Saffle | ccsaffle@jcp.com | Technical Architect ( JCP ) |

# Introduction

As part of an ambitious modernization roadmap JC Penney is implementing a new Enterprise Events Streaming Platform which aims to transform the way Enterprise applications publish and consume relevant information.

A subset of that information is the Item data, which is produced and updated by different enterprise systems, and given the distributed nature of the JC Penney applications, this information needs to be notified and updated allowing the consumer applications to maintain an accurate unified view of the inventory at JC Penney in order to take better business decisions on the best way possible.

Item Producer application is consumer of Item Events in RMS. The synchronization is done through Event Streaming Architecture using Java Spring Boot and Confluent.

## Interface Definitions

The Producer Data Mapping is defined in an excel table in the associated file [Item-Producer-Data-Mapping-v2.xlsx](https://jcp.sharepoint.com/:x:/s/EventStreamingPlatformCoreTeam/EZbSx6MhgMhMk-qCvjF7G_0B8SovkpBsDcM-MWwstvNW8Q?e=98xvm5)

## Purpose

This document describes the components, the data and the interactions involved on publishing the Item events *from the source system Retail Merchandizing System (RMS) to the Kafka topic.*

## Document Scope

This document outlines planned structure, features, and Mapping Document of a project, serving as a comprehensive guide to ensure a cohesive and effective implementation of the proposed design. It defines the project's goals, specifications, and technical details, providing a roadmap for development and collaboration among team members.

## Intended Audience

**Developers** : Those responsible for implementing the design and writing the code.

**Project Managers:** Individuals overseeing the project, ensuring it aligns with business objectives and stays on schedule.

**Stakeholders:** Individuals or groups with a vested interest in the project's success, such as executives, investors, or clients.

**Quality Assurance (QA) Team:** Professionals responsible for testing and validating that the implemented design meets specified requirements.

**Solution Architects:** Individuals responsible for the overall system and solution architecture and integration.

## Assumptions & Dependencies

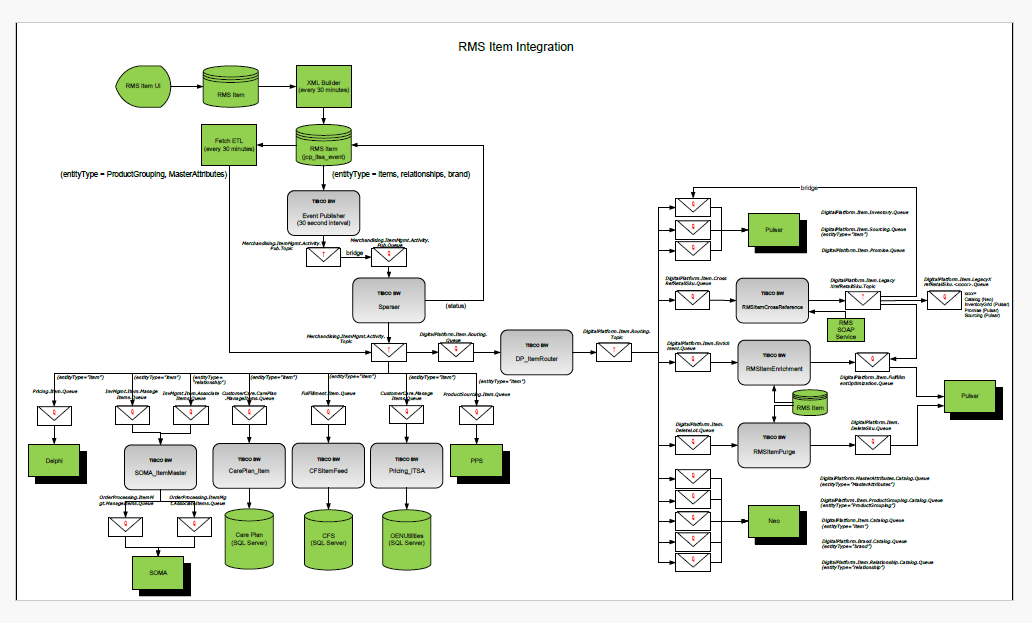
* Producer application will be created using Java 17 and Spring Boot 3.1.5, with subsequent changes and upgrades managed by JCP.
* During development will exclusively focus on conducting unit tests. The integration of the source, acceptance testing, and performance testing tasks will be managed by JCP.
* JCP is responsible for addressing any bugs identified during the User Acceptance Testing (UAT) and performance testing phases.
* Property values such as polling, delay, fetch, timeouts and the likes are set to default values in the lower environments.

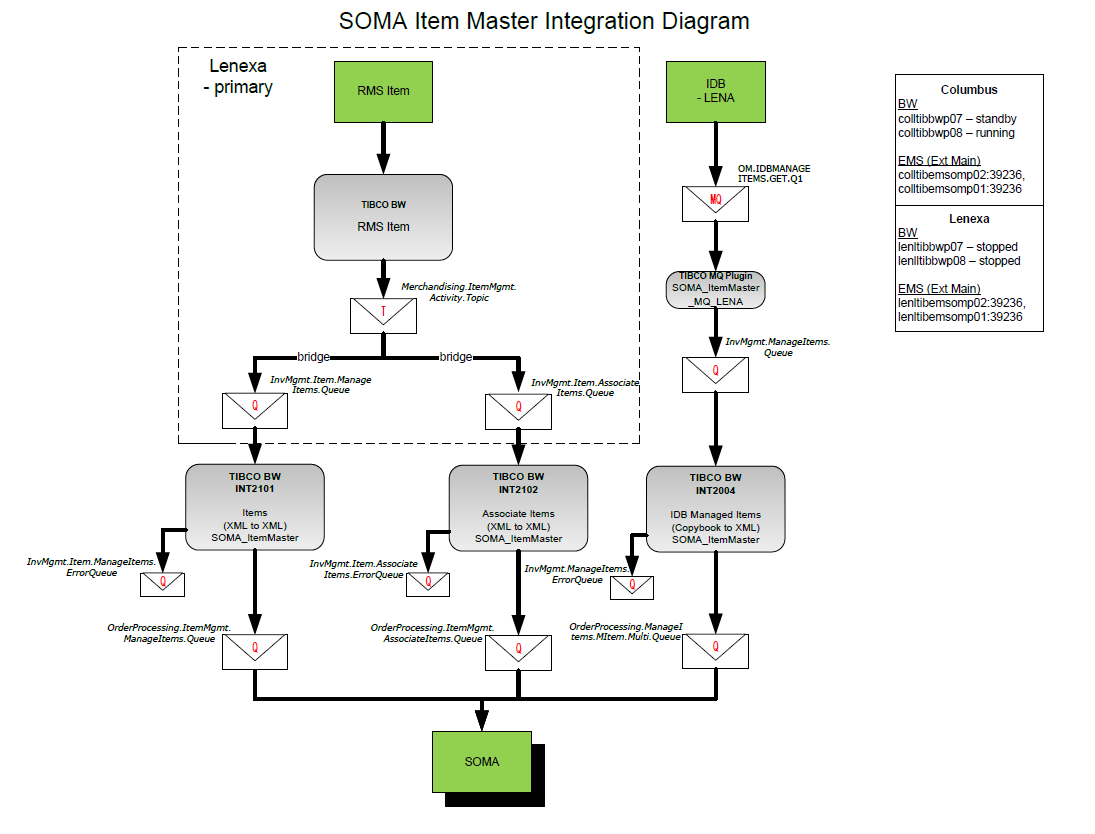
## Definitions

The table below defines acronyms and terms used in this design specification.

| **Term** | **Definition** |
| --- | --- |
| Batch | For the purposes of this document, the term “batch” refers to action on multiple records or items rather than single records and items. Batch processing typically occurs periodically (e.g. hourly, daily or weekly, or some other period) and acts on a relatively large collection of records. |
| Integration | Refers to a computer application or program that connects one system or application to another system or application. An integration is often synonymous with “service” or “API.” |
| Real-Time | For the purposes of this document, the term “real-time” refers to an action that takes place immediately when a change of state occurs on a single record or item. For example, if an employee is created in one system, a “real-time” integration would immediately act on that newly created employee, potentially transferring it to another system. |
| CDM | **C**ommon **D**ata **M**odel is a standardized and structured way of organizing and defining data entities and relationships, providing a shared understanding across an organization's data landscape. It establishes a common language for data interoperability, allowing for seamless integration and consistency in data representation. |
| TIBCO | **T**he **I**nformation **B**us **CO**mpany, specializes in integration, analytics, and event processing software. TIBCO provides a range of products and solutions designed to help organizations connect disparate systems, analyze data, and automate business processes in real-time. |
| JCP | **JC** **P**enney, a prominent American department store chain, has a rich history dating back to its founding in 1902. Known for offering a wide range of affordable apparel, accessories, and home goods. |
| API | An **A**pplication **P**rogramming **I**nterface acts as a set of protocols and tools that allows different software applications to communicate with each other enabling the seamless integration of services. |
| EMS | **E**nterprise **M**essaging **S**ervice is a standards-based enterprise messaging platform that brings together different IT assets and communications technologies on a common real-time backbone and manages the real-time flow of information. |
| ESB | **E**nterprise **S**ervice **B**us is a software architecture that facilitates seamless communication and integration among diverse applications within an enterprise. |
| BW | **B**usiness **W**orks TIBCO BusinessWorks is an integration platform provided by TIBCO Software Inc. It is designed to facilitate the development, deployment, and management of integration solutions for connecting various applications and systems within an organization. BusinessWorks is part of the TIBCO Connected Intelligence platform and offers a visual development environment for creating integration processes. |
| RMS | The Oracle **R**etail **M**erchandising **S**ystem (RMS), or Merchandising, is used to execute core merchandising activities, including product management, inventory replenishment, purchasing, vendor management, and financial tracking. |
| SR | A **S**chema **R**egistry is a centralized repository that manages and stores schemas, ensuring consistency and compatibility in the serialization and deserialization of data within a distributed system or data pipeline. |
| SOA | **S**ervice **O**riented **A**rchitecture is a design paradigm that structures software applications as a collection of loosely coupled and interoperable services, promoting modularity and flexibility in system development. |
| ESA | **E**vent **S**treaming **A**rchitecture is a system design approach that leverages real-time, continuous streams of events to enable seamless communication, data processing, and event-driven applications. |
| CDC | **C**hange **D**ata **C**apture is a design pattern that allows to track changes in a database |

# Current State





## Poller (EventPublisherAgent)

1. Poll at 30 second intervals
2. Fetch events from table JCPRMS.jcp\_itsa\_event
3. The Fetch uses the BW concept of “Process in Subsets”. Each subset has a configuration maximum (default 250) rows in each subset. The Fetch iteration is executed until the last subset is fetched. The purpose is to better manage JVM heap space as the XML payload is returned in the JDBC results set.
4. Generate UUID
5. Determine type of event. Either item, relationship, or brand
6. Parse the event. Each event has its own schema
7. Publish to topic JCP-xxx.Merchandising.ItemMgmt.Activity.Pub.Topic
   1. Body mapped from column EVENT\_CONTENT\_TXT directly
   2. JMS Header properties

|  |  |  |
| --- | --- | --- |
| entityType | string | item, relationship, brand, MasterAttributes, ProductGrouping |
| futureOnlyy | boolean | Default to false |
| currentOnly | boolean | Default to true |
| eventID | string | map from column JCP\_ITSA\_EVENT\_SEQ |
| NMR\_SOURCE\_VERSION | string | v2 |
| itemStateCode | string | from itemStatecode |
| entityStatusCode | string | from itemStatusCode, brandStatusCode, or relationshipStatusCode |
| JMS\_TIBCO\_COMPRESS | string |  |

1. There are complex transitions in the process to handle:
   1. Validate complete payload
   2. If status is not known
   3. Max retries not reached (believe this is obsolete and always false)
2. Update JCPRMS.jcp\_itsa\_event. EVENT\_PUBLICATION\_STATUS\_CODE = 'SUCCEEDED'

## Sparse Event Publication (EntityChangeEvents)

1. Receive from queue JCP-xxx.Merchandising.ItemMgmt.Activity.Pub.Queue

Bridged from topic JCP-xxx.Merchandising.ItemMgmt.Activity.Pub.Topic

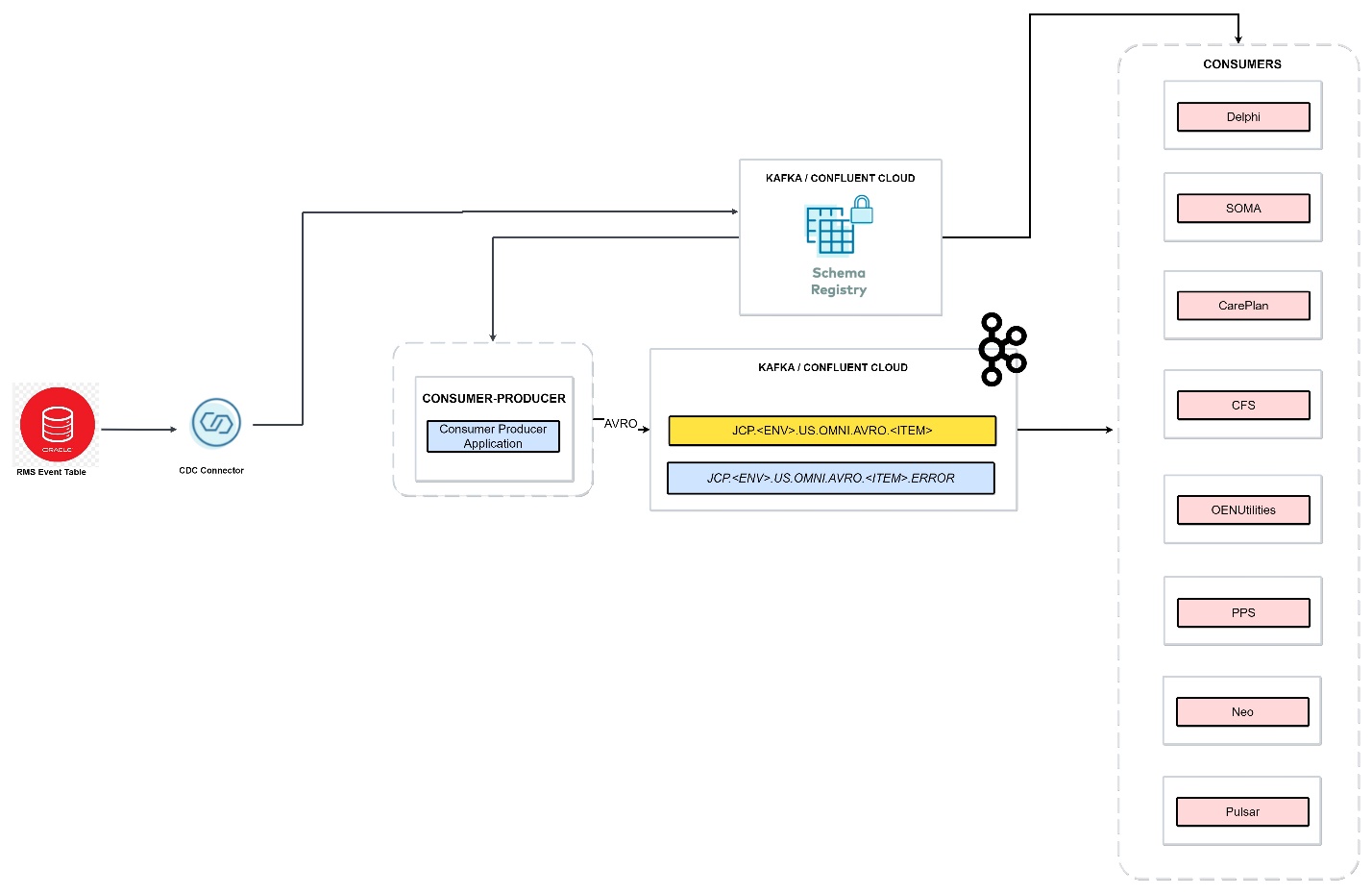
1. Route message to dynamic sub process. Routing based on JMS Header attribute entityType from the Poller process.
2. Separate sub processes for:
   1. Item
   2. Brand
   3. Relationship
   4. Group Relationship
   5. Pair Relationship
3. Each sub process does similar processing:
   1. Parse XML
   2. Generate UUID
   3. Map item attributes – it is fuzzy on the original intent
   4. Remove namespaces to confirm to XSD – it is fuzzy on the original intent
   5. Publish to topic JCP-xxx.Merchandising.ItemMgmt.Activity.Topic
      1. JMS Header properties. The purpose of the properties is to the consuming topic to queue bridges can filter on specific message types that are needed.

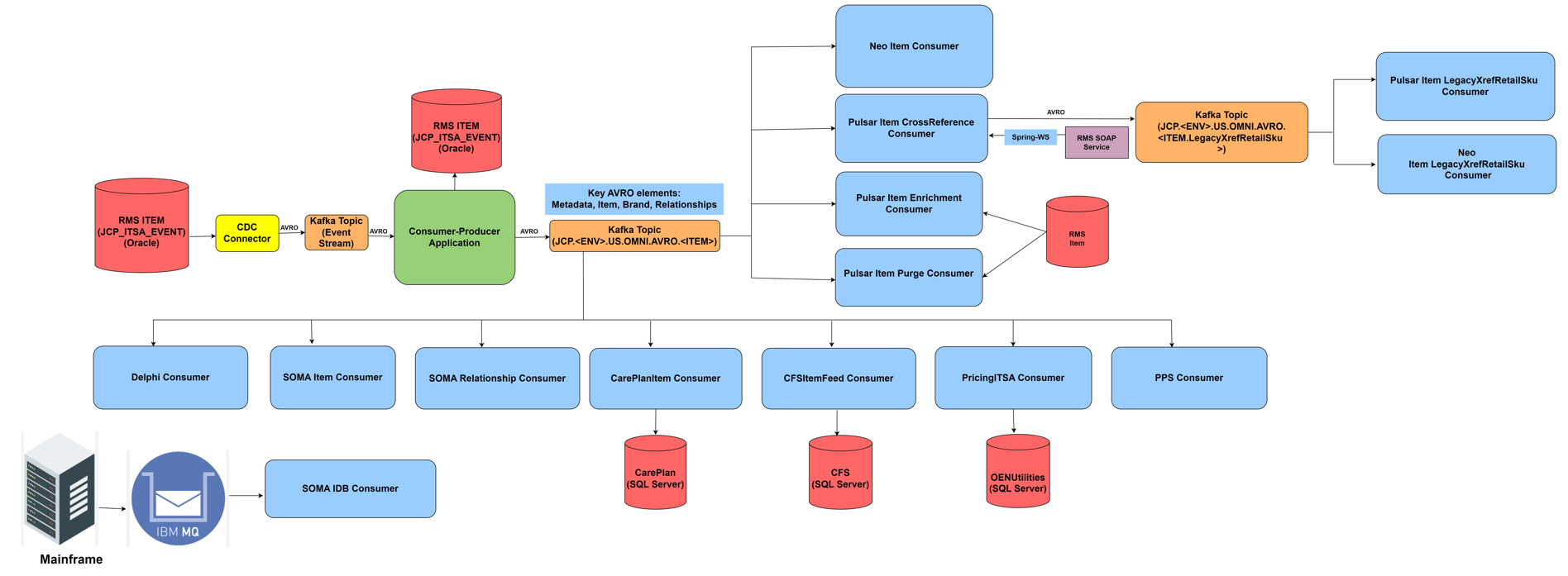
|  |  |  |
| --- | --- | --- |
| entityType | string | item, relationship, brand, MasterAttributes, ProductGrouping |
| entityID | string |  |
| brandModifierType | string | new, change, delete |
| itemModifierType | string | new, change, delete |
| JMS\_TIBCO\_COMPRESS | boolean | true, false |
| groupRelationshipModifierType | string | new, change, delete |
| pairRelationshipModifierType | string | new, change, delete |
| relationshipType | string | KTC, SUBN, SUBP, TRNS, ASME |
| isItemDefModifierSet | boolean | true, false |
| isItemMerchHierModifierSet | boolean | true, false |
| isItemChannelModifierSet | boolean | true, false |
| isItemBrandModifierSet | boolean | true, false |
| isItemProdHierModifierSet | boolean | true, false |
| isItemAliasModifierSet | boolean | true, false |
| isItemAttrTypModifierSet | boolean | true, false |
| isItemOptionModifierSet | boolean | true, false |
| isItemCostModifierSet | boolean | true, false |
| isItemSeasonModifierSet | boolean | true, false |
| isItemCyclDtModifierSet | boolean | true, false |
| isItemVendorModifierSet | boolean | true, false |
| isItemRelationshipModifierSet | boolean | true, false |
| isItemSKUModifierSet | boolean | true, false |
| futureOnly | boolean | true, false |
| currentOnly | boolean | true, false |
| eventID | string |  |
| NMR\_SOURCE\_VERSION | string |  |
| itemStateCode | string |  |
| entityStatusCode | string |  |

1. Relationship has a special process where it will call 2 sub processes. Group Relationship and Pair Relationship.
   1. Group – skipped if the Relationship is not Group
   2. Pair – skipped if the Relationship is not Pair

So in the end, there is only 1 message published for a Relationship. And the sub processes are mutually exclusive and only published from one of the processes.

# Future State

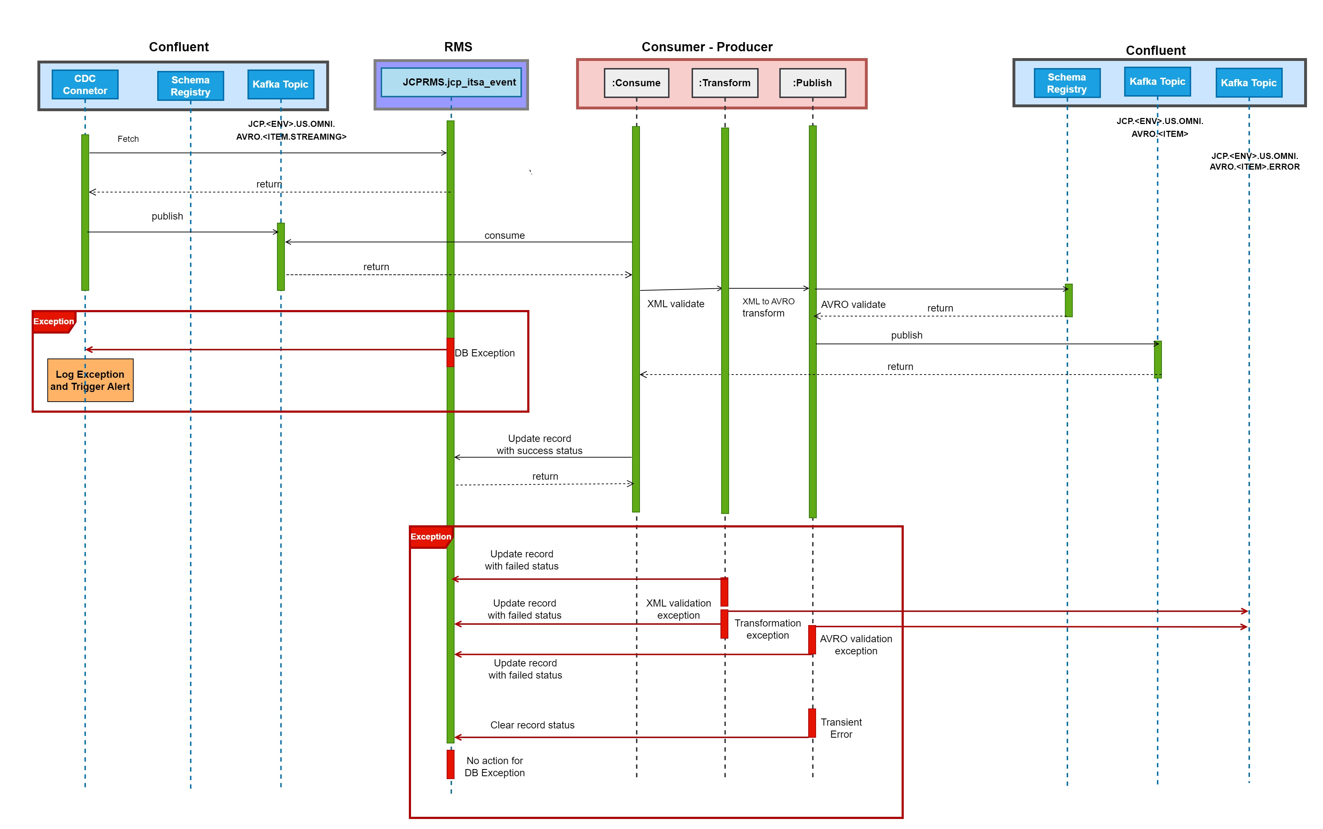




In the future state of our system, there is a strategic shift from TIBCO to Confluent Kafka as the primary source of truth for data, which represents a shift from batch mode to real-time stream processing. Data is ingested from source RMS and on prem CDC confluent connector is used to stream the records to Kafka topic.

Followed by a Java/SpringBoot Producer application, which consumes the event from Source CDC Kafka topic and filters, transforms them to a standardized Avro format before publishing them onto a target Kafka topic. Subsequent applications will consume from this target Kafka topic within the cluster, with the ability to perform transformations such as field selection and transaction/item filtering. These transformative operations will be executed by additional Java/SpringBoot applications, acting as a consumer.

## Sequence diagram



**CDC Connector**

1. CDC connector is configured to stream the records from Item event table to the kafka source topic with single partition to preserve the order of processing the items.

2. Any exception occurred will be logged and alert will be triggered

**Consume**

1. Consume the message from Kafka source topic and apply the filter condition to proceed further. Refer Appendix A. Filter Criteria.

**Transform**

1. Determine type of event. Either item, relationship, or brand

2. Validate and unmarshall XML using the respective event type schema

3. Check for payload, If Payload is Incomplete, add INFO log, Remove the Event and exit else Proceed

4. Check for Status, If Status is not known add INFO log and exit else Proceed

5. Use the mapping document and map the XML pojo to AVRO object

6. Refer Appendix B. PSEUDO Algorithm and DB Queries for more details.

**Publish**

Publish AVRO to kafka Item topic and Persist success. (Refer Appendix B)

**Kafka Key:**

Kafka Key plays crucial role in Item Producer design to ensure that -

1. Updates for an Item would be processed in sequence by a consumer as they are in the same partition.
2. For Relationships, the relationshipNumber item will place it in the same partition as corresponding Item updates for the same, and in sequence.

Select Kafka Key based on the event type:

Item - Item/ItemDefinition/ItemNumber

Brand - Brand/BrandNumber

Relationship (Kit)-Relationships/ItemGroupRelationships/ItemGroupRelationship/RelationshipNumber

Relationship (TRNS) - Relationships/ItemPairRelationships/ItemPairRelationship/Item/RelationshipDefinition/RelationshipNumber

**Kafka header:**

requestId - > UUID

**Kafka message:** ItemMaster AVRO Object

**Exception Handling:**

1. Exception during Validating and Unmarshalling the XML content are as captured as ValidationException and UnmarshalException, add ERROR logs and Remove item(Refer Appendix B) and will not be retried.

2. Exception during mapping or transformation are captured as Transformation exception, add ERROR logs, Send message to Kafka Error topic and RemoveItem (Refer Appendix B) and will not be retried

3. Exception during Publish to kafka(Transient error) are captured as KafkaException, will be retried internally by kafka using delivery timeout. After deliver timeout, if message is not producer kafka, it will be considered as failure. Add ERROR log and message should not be acknowledged, so that it can be reprocessed from source topic. Database Persist Success should be bypassed.

4. Exception during DB Persist success and Failure are captured as DBException, add ERROR log and will be retried for connectivity errors using the kafka consumer framework using blocking retry strategy to preserve the sequence of processing the messages. Key properties for retry are backoff-max-failure and backoff-interval.

**Key Design Decisions:**

1. Removal of futureEffective in the xsd schema (not mapped to Avro)
2. Removal of messageHeader in the xsd schema (not mapped to Avro)
3. Removal of pageDetail in the xsd schema (not mapped to Avro)
4. In lieu of JMS header attributes, MetaData element added to Avro schema to contain the same.
5. jcp\_itsa\_event is not monitored for failures currently and no action were taken by the application, will be the same in future state as well.
6. Combined 2 TIBCO BW process (POLLER and SPARSER) to one Producer in future state
7. XML NameSpace conversions in TIBCO BW were ignored as they are not appliable in future state.
8. futureOnly and currentOnly filter conditions are removed in future state.

## DESIGN APPROACH

* Bloking retry mechanism will exclusively be implemented for handling connectivity errors.
* Validation procedures entail fetching the most recent schema from the Schema registry.
* During runtime, Secret Keys and credentials are dynamically retrieved from Kubernetes Secrets.

**Key Producer Configurations**

* session.timeout.ms: 45000
  + Increase based on load test and performance test (default 10000) - start with 45000 recommended. This allows for high availability.
* acks: all
  + The number of acknowledgments the producer requires the leader to have received before considering a request complete. This controls the durability of records that are sent.
    - acks=all This means the leader will wait for the full set of in-sync replicas to acknowledge the record. This guarantees that the record will not be lost as long as at least one in-sync replica remains alive. This is the strongest available guarantee. This is equivalent to the acks=-1 setting.
    - Note that enabling idempotence requires this config value to be ‘all’. If conflicting configurations are set and idempotence is not explicitly enabled, idempotence is disabled.
* enable-idempotence: true
  + When set to ‘true’, the producer will ensure that exactly one copy of each message is written in the stream. If ‘false’, producer retries due to broker failures, etc., may write duplicates of the retried message in the stream. Note that enabling idempotence requires max.in.flight.requests.per.connection to be less than or equal to 5 (with message ordering preserved for any allowable value), retries to be greater than 0, and acks must be ‘all’.
  + Idempotence is enabled by default if no conflicting configurations are set. If conflicting configurations are set and idempotence is not explicitly enabled, idempotence is disabled. If idempotence is explicitly enabled and conflicting configurations are set, a ConfigException is thrown.
* deliver-timeout-ms: 120000 (default)
  + An upper bound on the time to report success or failure after a call to send() returns. This limits the total time that a record will be delayed prior to sending, the time to await acknowledgement from the broker (if expected), and the time allowed for retriable send failures. The producer may report failure to send a record earlier than this config if either an unrecoverable error is encountered, the retries have been exhausted, or the record is added to a batch which reached an earlier delivery expiration deadline. The value of this config should be greater than or equal to the sum of request.timeout.ms and linger.ms.
* max-in-flight-requests-per-connection: 5
  + The maximum number of unacknowledged requests the client will send on a single connection before blocking. Note that if this configuration is set to be greater than 1 and enable.idempotence is set to false, there is a risk of message reordering after a failed send due to retries (i.e., if retries are enabled); if retries are disabled or if enable.idempotence is set to true, ordering will be preserved. Additionally, enabling idempotence requires the value of this configuration to be less than or equal to 5. If conflicting configurations are set and idempotence is not explicitly enabled, idempotence is disabled.
* retry-backoff-ms: 100 (default)
  + The amount of time to wait before attempting to retry a failed request to a given topic partition. This avoids repeatedly sending requests in a tight loop under some failure scenarios.

**Key Consumer Configurations**

* The listener is set to poll for new messages in real time, fetching a maximum of 500 events per iteration. The minimum it will pull would be 1 byte. A Blocking strategy is employed for processing these events.
  + fetch.min.bytes=1 (default 1)
    - The minimum amount of data the server should return for a fetch request. If insufficient data is available, the request will wait for that much data to accumulate before answering the request.
  + Batch Size: 500
    - The maximum number of records returned in a single call to poll(). Note, that max.poll.records does not impact the underlying fetching behavior. The consumer will cache the records from each fetch request and returns them incrementally from each poll.
  + Max Polling interval: 300000 ms
    - The maximum delay between invocations of poll() when using consumer group management. This place an upper bound on the amount of time that the consumer can be idle before fetching more records. If poll() is not called before expiration of this timeout, then the consumer is considered failed and the group will rebalance in order to reassign the partitions to another member. For consumers using a non-null group.instance.id which reach this timeout, partitions will not be immediately reassigned. Instead, the consumer will stop sending heartbeats and partitions will be reassigned after expiration of session.timeout.ms. This mirrors the behavior of a static consumer which has shutdown.
  + session.timeout.ms:
    - Increase based on load test and performance test (default 10000) - start with 45000 recommended. This allows for high availability.
* backoff-interval:
  + The amount of time to wait before attempting to retry a failed request to a given topic partition. This avoids repeatedly sending requests in a tight loop under some failure scenarios.
* backoff-max-failure:
  + maximum number of retry attempts to reprocess the particular message in the kafka topic.

## TECHNOLOGY Stack

|  |  |
| --- | --- |
| Technology | Version |
| Java | 17 |
| Spring boot | 3.1.5 |

## ACCEPTANCE CRITERIA/DEFINITION OF DONE

* Develop Kafka Consumer-Producer application in Spring Boot 3.1.5 and JDK 17 to consume Item events from source kafka topic using CDC connector. will also provide error handling.
  + Producer application will contain a message transformer module applying transformations to the data to render it in the format of the defined AVRO schema.
  + Producer application will validate transactions against the appropriate AVRO schema and write all messages to the Item Kafka Topic.
  + Producer application will be configured for error handling.
* Provide documentation and conduct a code walkthrough to familiarize the RMS team with the functionality and structure of the developed applications.

## EXCEPTION HANDLING & RETRY

|  |  |
| --- | --- |
| The following topic are used for exception handling JCP.<ENV>.US.OMNI.AVRO.<ITEM>.ERROR | Messages are sent to this topic when they are considered as data issue during validation and transformation. Also whenever retry is not applicable/needed. |

**Error Topic Monitoring**

Monitoring will be set up for the ERROR and exceptions will be reported via tickets. Best practices for monitoring in this topic is in the process of being developed.

## LOGGING STANDARDS

SLF4j framework is used for logging, the following standards will be enforced while logging messages ,

* Unique id will be determined for each transaction.
* All the messages that enter and exit the component must be captured.
* Appropriate log categories and levels must be captured.

## SECURITY

|  |  |
| --- | --- |
| **Input Validation** | Schema registry is used to store the schemas, applications validation the inputs must get the use the schema registry to validate. |
| **Secure Connection** | All the connections are secured by SSL   * Confluent ( broker , schema registry )   + TLS 1.2 via SASL/SSL |
| **Logging** | Never log any sensitive information  ERROR/FATAL   * Log the stack track of errors here to help troubleshoot and resolve any fetal errors. Always include correlation ids and timestamps.   TRACE   * Ensure traceability is there are correlation id (transactions ids) sent via headers on all messages. At a minimum log these correlations ids.   DEBUG   * Allow class debug options to be enabled during development if needed. Change the log level to debug if the logs are verbose.   INFO   * These will be used at a minimum for simple lightweight correlation tracking. |
| **Secure Properties** | Sensitive properties like secrets and password keys will be set as an environment variables. It is recommended that we pull the configuration from the Key Vault or Secrets manager. Using CICD Pipelines are also acceptable to inject secrets and passwords. The applications should not contain any secrets or passwords. |

## PACKAGING & RELEASE

|  |  |
| --- | --- |
| **Repository** | https://stash.jcpenney.com/projects/ICOE/repos/<TBD> |
| **Build** | Gradle |
| **Deployment** | Jenkins , AWS & Python Scripting - TBD |
| **Environments** | Confluent – Prod & Non-Prod ( Dev & Stage )  AWS – Dev , Stage & Prod |
| **Release process** | WIP in defining the release process. Dev – on event of successful check in  Stage – PR approval process  Prod – CAB , PR approval process |

## REFERENCES

| **Title** | **Revision** | **Description** |
| --- | --- | --- |
| [Document title] | [rev] | [Description] |

Table 2: References

1. Fiter Criteria

*[Start any information to be included in an appendix here.]*

Filter the events and process further if any one of the below listed conditions are met.

a) Event that require publication to enterprise topic, if not a future effective event but not to be backfill to PDB

upper (EVENT\_PUBLICATION\_STATUS\_CODE) not in ('SUCCEEDED', 'FAILED-2')

and EVENT\_BACKFILL\_STATUS\_CODE is null

and (FUTURE\_EFF\_IND is null or FUTURE\_EFF\_IND = 'N'))

b) Event that may or may not been backfill to PDB but require publication to enterprise topic

upper (EVENT\_PUBLICATION\_STATUS\_CODE) not in ('SUCCEEDED', 'FAILED-2')

and EVENT\_BACKFILL\_STATUS\_CODE not in ('BFINPROGRESS'))

c) Event that cannot be backfill (i.e., max failed times reached) but still require publication to enterprise topic, if not a future effective event

upper (EVENT\_PUBLICATION\_STATUS\_CODE) not in ('SUCCEEDED', 'FAILED-2')

and EVENT\_BACKFILL\_STATUS\_CODE not in ('BFSUCCEEDED')

and (FUTURE\_EFF\_IND is null or FUTURE\_EFF\_IND = 'N'))

d) Event that may have not been backfill but require publication to enterprise topic

upper (EVENT\_PUBLICATION\_STATUS\_CODE) not in ('SUCCEEDED', 'FAILED-2')

and EVENT\_BACKFILL\_STATUS\_CODE not in ('BFFAILED-2')

e) Event that cannot be published to enterprise topic but require to be backfill to PDB

upper (EVENT\_PUBLICATION\_STATUS\_CODE) in ('SUCCEEDED', 'FAILED-2')

and EVENT\_BACKFILL\_STATUS\_CODE not in (‘BFFAILED-2', 'BFINPROGRESS', 'BFSUCCEEDED'))

1. PSEUDO Algorithm and DB Queries

*[Start any information to be included in an appendix here.]*

**1. Constants**

* brandRootElement = "commonBrandRoot"
* itemRootElement = "commonItemResponseRoot"
* relationshipRootElement = "relationshipResponse"
* brandEntityType = "brand"
* itemEntityType = "item"
* relationshipEntityType = "relationship"
* changeModifier = "change"
* failedToPublishStatus = "FAILED"
* readyToPublishStatus = "READYTOPUBLISH"
* succeededStatus = "SUCCEEDED"

**2. Determine Event Type**

If(Index-of(upper-case(tokenize(trim(Event/Record/EVENT\_CONTENT\_TXT),[1]), upper-case(itemRootElement))!=-1)then itemEntityType

Else If(Index-of(upper-case(tokenize(trim(Event/Record/EVENT\_CONTENT\_TXT),[1]), upper-case(relationshipRootElement))!=-1)then relationshipEntityType

Else If(Index-of(upper-case(tokenize(trim(Event/Record/EVENT\_CONTENT\_TXT),[1]), upper-case(brandRootElement))!=-1)then brandEntityType

3. **Check for Payload**

isPayloadIncomplete(){

if ((EventType == itemEntityType and lower-case(pfx:commonItemResponseRoot/pfx:itsaItems/pfx:item[1]/pfx:itemDefinition/@modifier)= changeModifier and count(pfx:commonItemResponseRoot/pfx:itsaItems/pfx:item[1]/pfx:itemDefinition/pfx:itemDescription)=0 ) or

(EventType = relationshipEntityType and count(pfx6:relationshipResponse/pfx6:relationships/pfx6:itemGroupRelationships/ns5:itemGroupRelationship[1]/ns4:relationshipDefinition) = 0 and count(pfx6:relationshipResponse/pfx6:relationships/pfx6:itemPairRelationships/ns6:itemPairRelationship[1]/ns6:item/ns4:relationshipDefinition) = 0)) return true; else return false;}

4. **Remove Event**

update jcp\_itsa\_event

set EVENT\_PUBLICATION\_STATUS\_CODE = ‘**FAILED-2’**,

EVENT\_BACKFILL\_STATUS\_CODE = ‘**BFFAILED-2’**,

CREATE\_ID = 'PUBLISHER\_AGENT',

CREATE\_DATETIME = CURRENT\_TIMESTAMP ,

SPARSER\_EVENT\_PUBLICATION\_NM = 'FAILED',

SPARSER\_DATETIME = SYSDATE

where JCP\_ITSA\_EVENT\_SEQ = **Event/Record/JCP\_ITSA\_EVENT\_SEQ**

5. **Check for Status**

isStatusNotKnown(){

if ((starts-with(upper-case (Event/Record/EVENT\_PUBLICATION\_STATUS\_CODE),upper-case(failedToPublishStatus))or compare(upper-case(Event/Record/EVENT\_PUBLICATION\_STATUS\_CODE),upper-case(readyToPublishStatus)) = 0 or compare(upper-case(Event/Record/EVENT\_PUBLICATION\_STATUS\_CODE),upper-case(succeededStatus)) = 0))then return false; return true;}

6. **Persist Success**

update jcp\_itsa\_event

set EVENT\_PUBLICATION\_STATUS\_CODE = **'SUCCEEDED'**,

CREATE\_ID= 'PUBLISHER\_AGENT',

CREATE\_DATETIME = SYSTIMESTAMP,

SPARSER\_EVENT\_PUBLICATION\_NM = **'SUCCEEDED'**,

SPARSER\_DATETIME = SYSTIMESTAMP

where JCP\_ITSA\_EVENT\_SEQ = **Event/Record/JCP\_ITSA\_EVENT\_SEQ**

1. ITEM POLLER AND SPARSER – Business works process

