**Epic Title**

Enhancing DCM System with Key Business Element (KBE) Catalog Details

**Objective**

To enhance the Data Catalog Management (DCM) system by integrating Key Business Element (KBE) catalog details sourced from the Metaportal system. This integration will enable Line of Businesses (LOBs) to access critical business elements within the DCM, supporting downstream utilization by the DQ4QD system.

**Description**

The DCM system currently manages metadata but lacks detailed KBE catalog information required by LOBs. By integrating KBE details from the Metaportal system and other system, the DCM will provide a centralized, enriched catalog that supports LOB-specific requirements and ensures compatibility with the DQ4QD system for data quality processes. This Epic focuses on seamless data integration, user interface updates, and robust data validation to maintain consistency and reliability.

**Acceptance Criteria**

1. **KBE Data Integration**:
   * KBE catalog details are successfully extracted from the Metaportal system and other systems.
   * Define the Data mappings between Metaportal KBE fields and DCM schema. Integration supports incremental updates to reflect changes in Metaportal data.
2. **DCM System Enhancements**:
   * DCM database schema is updated to accommodate KBE attributes (e.g., element name, description, LOB owner, data type).
   * API endpoints are developed or updated to handle KBE data ingestion and retrieval.
   * Data validation rules ensure KBE data integrity (e.g., mandatory fields, format consistency).
3. **User Interface Updates**:
   * DCM UI includes a dedicated KBE catalog view displaying key attributes (e.g., element name, LOB, status).
   * Search and filter functionalities allow users to query KBE details by LOB, element type, or other relevant criteria.
   * UI is responsive and accessible per organizational standards.
4. **DQ4QD Compatibility**:
   * KBE data in DCM is structured to meet DQ4QD system input requirements.
   * A data export mechanism (e.g., API or file-based) is implemented for DQ4QD consumption.
   * Documentation is provided detailing KBE data structure and access methods for DQ4QD integration.
5. **Testing and Validation**:
   * Unit tests cover new API endpoints, data validation, and UI components.
   * End-to-end tests verify data flow from Metaportal to DCM to DQ4QD.
   * UAT confirms LOB satisfaction with KBE accessibility and usability.
6. **Documentation and Training**:
   * Technical documentation details integration architecture, API usage, and data schema.
   * User guides are updated to include KBE catalog navigation and usage.
   * Training sessions are conducted for LOB stakeholders.

**Design Details**

**Architecture**

* **Data Integration Layer**:
  + Use ETL processes to extract KBE data from Metaportal via API or database queries.
  + Transform data to align with DCM schema using a mapping configuration (e.g., JSON or YAML).
  + Load transformed data into DCM using batch processing for efficiency.
* **DCM System**:
  + Extend DCM database with new tables or fields for KBE attributes (e.g., kbe\_catalog table with columns: element\_id, name, description, lob\_owner, data\_type, status).
  + Develop RESTful APIs for KBE data CRUD operations, secured with OAuth 2.0.
  + Implement validation logic to enforce data quality (e.g., regex for formats, mandatory field checks).
* **User Interface**:
  + Integrate search/filter controls using Elastic Search.
  + Ensure UI communicates with backend APIs via asynchronous calls (e.g., Axios).
* **DQ4QD Integration**:
  + Provide a dedicated API endpoint or SFTP export for KBE data in JSON/CSV format.
  + Ensure data includes metadata (e.g., timestamps, source system) for traceability.

**Assumptions**

* Metaportal system provides accessible APIs or database views for KBE data.
* LOBs have defined key KBE attributes (e.g., name, type, owner).
* Infrastructure supports additional storage and compute for KBE records.

**Risks and Mitigations**

* **Risk**: Metaportal data inconsistencies.
  + **Mitigation**: Implement robust validation and logging during ETL.
* **Risk**: Performance degradation with large datasets.
  + **Mitigation**: Optimize queries with indexing and caching.
* **Risk**: LOB resistance to new UI.
  + **Mitigation**: Involve LOBs in UI design via prototypes and feedback sessions.

Continuation of existing EPICs for adding ingestion data, SLA data, Lineage data and data related to retention.

**Epic: Streamlined Vault System Integration with AITs and Automated Reporting Dashboard**

**Objective**

To enhance the vault system process by integrating it with different AITs, implementing a user-friendly UI dashboard using the Streamlit Python library, and automating end-of-day summary notifications for teams regarding secret rotations and password updates for various system IDs (e.g., Oracle, SQL Server, Teradata, and service\_id). This will improve security, visibility, and operational efficiency across systems.

**Description**

The vault system manages the secure storage and rotation of credentials (secrets and passwords) for multiple systems, including Oracle, SQL Server, Teradata, and service IDs. This epic focuses on enabling seamless integration with other AITs to streamline credential updates, creating a Streamlit-based UI dashboard for real-time monitoring and management of the vaulting process, and automating daily summary reports to notify teams of pending secret rotations or password updates. The dashboard will provide visibility into the status of system IDs, rotation schedules, and compliance, while notifications ensure timely action by respective teams.

**Acceptance Criteria**

1. **Integration with AITs**:
   * The vault system successfully integrates with different AITs to automate credential updates across Oracle, SQL Server, Teradata, and service\_id systems.
   * Automated workflows trigger secret rotation and password updates without manual intervention.
   * Integration supports error handling and logging for failed operations.
2. **Streamlit UI Dashboard**:
   * A Streamlit-based dashboard is implemented, accessible to authorized AIT owner/users.
   * The dashboard displays real-time status of system IDs, including last rotation date, next scheduled rotation, and any pending actions.
   * The dashboard includes filters for specific systems (e.g., Oracle, SQL Server) and search functionality by service\_id.
   * The dashboard supports role-based access control to restrict sensitive information.
   * The UI is responsive and user-friendly, with clear visualizations (e.g., tables, charts) for rotation status and compliance metrics.
3. **Automated Notifications**:
   * Daily summary reports are generated and sent to respective teams by end of day (configurable time, e.g., 6 PM).
   * Reports include details of system IDs requiring secret rotation or password updates.
   * Notifications are delivered via email
   * Notifications include actionable links or instructions to address pending actions.
4. **Security and Compliance**:
   * All credential updates comply with organizational security policies (e.g., password complexity, rotation frequency).
   * Audit logs are maintained for all vaulting activities, including rotation events and access to the dashboard.
   * The system ensures no sensitive credentials are exposed in the dashboard or notifications.
5. **Performance and Scalability**:
   * The Streamlit dashboard loads within 5 seconds for typical queries.
   * The system supports scalability for additional AIT integrations in the future.

**Design Details**

**System Architecture**

* **Vault System**: Leverages HashiCorp Vault for secure credential storage and rotation. APIs will be used to integrate with AITs for automated updates.
* **AIT Integration**: Custom Python artifact to connect the vault system with AITs. Each AIT will have a dedicated connector module for modularity.
* **Streamlit Dashboard**:
  + Built using the Streamlit Python library for rapid UI development.
  + Backend connects to the vault system via APIs to fetch real-time data.
  + Visualizations include tables for system ID status and charts for rotation trends.
  + Authentication layer (e.g., OAuth or SSO) to secure dashboard access.
* **Notification System**:
  + Austosys scheduler to trigger daily report generation.
  + Integration with email servers (e.g., SMTP)
  + Reports generated in HTML/PDF format for clarity and portability.

**Workflow**

1. **Credential Rotation**:
   * AITs trigger secret rotation or password updates via vault system APIs.
   * The vault system updates credentials and logs the activity.
2. **Dashboard Updates**:
   * Streamlit dashboard polls the vault system periodically (e.g., every frequency minutes) to display updated statuses.
   * Users can filter and search for specific system IDs or view compliance metrics.
3. **Notifications**:
   * At the configured time, the scheduler generates a report summarizing pending actions.
   * Reports are sent to team-specific channels or email addresses.
4. **Audit and Monitoring**:
   * All actions (rotations, dashboard access, notification delivery) are logged for compliance.
   * Alerts are triggered for failed rotations or integration issues.
   * on dashboard usage and notification workflows.

**Risks and Mitigation**

* **Risk**: AIT integration failures due to incompatible APIs.
  + **Mitigation**: Develop modular connectors with fallback mechanisms and thorough testing.
* **Risk**: Dashboard performance issues with large datasets.
  + **Mitigation**: Optimize queries and implement caching.
* **Risk**: Notification delivery failures.
  + **Mitigation**: Use reliable messaging platforms and include retry logic.

**Epic: Tech Debt (Ozone File System Compatibility, Private Banking Data Integration and History data enablement)**

**Objective**

The objective of this epic is to address technical debt in the DCM system by enhancing its compatibility with the Ozone file system while enabling seamless integration with the ITC Hadoop team’s Ozone file system and integrating support for private banking data alongside global banking data. The solution ensures that private banking and global banking data remain isolated, preventing co-mingling in searches and Historical data enablement for asset updates.

**Description**

The DCM system requires updates to align with the Ozone file system for improved scalability and data management, in collaboration with the ITC Hadoop team. Additionally, the system must be enhanced to support private banking data, ensuring strict isolation from global banking data to maintain privacy and compliance. This involves refactoring existing code, implementing new data segregation mechanisms, and ensuring compatibility with Ozone’s distributed architecture. And enabling history data capture in case of any asset update, so user will be able to trace how the asset update has happened over time.

**Acceptance Criteria**

1. **Ozone File System Compatibility**:
   * DCM system code is updated to support the Ozone file system.
   * Connect with ITC Hadoop team’s regarding Ozone environment, and how their setup needs to accommodate ozone changes
   * Data stored in Ozone is accessible and retrievable by the DCM system without errors.
   * Performance benchmarks confirm no degradation in data processing speed compared to the current system.
2. **Private Banking Data Integration**:
   * DCM system supports private banking data alongside global banking data.
   * Private banking data is stored and processed in a dedicated, isolated namespace or schema.
   * Global banking and private banking data are not visible to each other in search queries or data retrieval operations.
3. **History data capture for asset update:**
   * DCM system will start support historic data capture for any future updates.
   * Version history needs to be maintained for all the asset and new updates will be have latest version.
4. **Data Isolation and Privacy**:
   * Access controls are implemented to ensure private banking data is only accessible to authorized users/roles.
   * Search functionality enforces strict segregation, preventing private banking data from appearing in global banking searches and vice versa.
   * Audit logs capture all access attempts to private banking data for compliance purposes.
5. **Collaboration and Deployment**:
   * Discussion and validation of changes performed by the ITC Hadoop team for Ozone compatibility.
   * Deployment process includes rollback mechanisms in case of integration failures.
   * Documentation is updated to reflect new system architecture, data flows, and access controls.

**Design Details**

**1. Ozone File System Integration**

* **Objective**: Refactor DCM system to leverage and enable Ozone’s distributed file system for scalability and compatibility.
* **Approach**:
  + Update DCM’s data access layer to use Ozone’s APIs for file read/write operations.
  + Collaborate with ITC Hadoop team to align on Ozone configuration (e.g., bucket layouts, access policies).
  + Implement retry mechanisms for handling transient failures in Ozone’s distributed environment.

**2. Private Banking Data Integration**

* **Objective**: Enable DCM to handle private banking data with strict isolation from global banking data.
* **Approach**:
  + Create separate data schemas or namespaces for private and global banking data within the DCM system.
  + Implement restrict data access based on user roles.
  + Modify search and query logic to filter results based on data type (private vs. global) and user permissions.
* **Components**:
  + Schema Design: Use separate database tables/collections or Ozone buckets for private and global banking data.
  + Search Engine: Update search indexing to tag data by type (private/global) and enforce isolation in query results.

**3. History data capture for asset update:**

* **Objective -** Enable the DCM system to capture and maintain historical data for all asset updates, ensuring version history is preserved and the latest version is clearly identified.
* **Approach**
  + Implement a versioning mechanism within the DCM system to track and store historical data for asset updates.
  + Ensure each update is assigned a new version number, with metadata to identify the latest version.
  + Provide query capabilities to access historical versions and audit trails for assets.
* **Components**
  + Create a history table/collection (e.g., asset\_history) to store snapshots of asset data for each update, including fields for asset\_id, version\_number, updated\_at, updated\_by, and asset\_data.
  + Use a version\_number field (auto-incremented integer or timestamp-based) to track versions, with a flag (e.g., is\_latest) to mark the current version.
  + Modify the DCM update process to archive the current asset state in the history table/collection before applying updates.
  + Implement a trigger or middleware to assign a new version number and update the is\_latest flag for each asset modification.

**4 . Data Isolation and Privacy**

* **Objective**: Ensure private and global banking data remain segregated in storage and search.
* **Approach**:
  + Use cryptographic hashing or tokenization for sensitive private banking data to enhance security.
  + Implement query filters to prevent cross-data visibility (e.g., private banking data appearing in global banking searches).
  + Log all data access events to an audit trail for compliance with banking regulations.
* **Components**:
  + Data Encryption: Apply encryption at rest for private banking data using industry-standard algorithms (e.g., AES-256).
  + Query Filters: Modify search APIs to include data-type checks before returning results.
  + Audit Logging: Integrate with a centralized logging system to record access attempts.

**5. Testing and Validation**

* **Unit Tests**: Cover all new and modified code paths, focusing on Ozone integration and data isolation logic.
* **Integration Tests**: Validate end-to-end data flows between DCM and Ozone, including private/global banking data segregation.
* **Security Tests**: Perform penetration testing to ensure private banking data cannot be accessed by unauthorized users.
* **Performance Tests**: Benchmark system performance under mixed workloads to ensure no degradation.

**6. Deployment and Rollback**

* **Deployment Plan**:
  + Deploy changes in a staging environment for ITC Hadoop team validation.
  + Use blue-green deployment to minimize downtime during production rollout.
  + Monitor system health post-deployment using existing monitoring tools.
* **Rollback Plan**:
  + Maintain backups of current DCM system configuration and data.
  + Implement automated rollback scripts to revert to the previous system state if critical issues are detected.

**Dependencies**

* **ITC Hadoop Team**: For Ozone configuration details and validation support.
* **Compliance Team**: To ensure private banking data handling meets regulatory requirements.

# Epic: Integrating Kafka Metadata Details into DCM System

## Epic Title

Integrating Kafka Metadata Details into DCM System for Enhanced Cluster Management

## Objective

To enable seamless integration of Kafka metadata from EEH Tier 0 and Tier 1 clusters into the DCM system, capturing critical details such as topic names, partition details, schema details, consumer groups, and producer/consumer application details. This epic aims to create a robust system for the DCM team to automate metadata collection, facilitate schema and topic updates, and provide a standardized template for the EEH team to input producer and consumer details, ensuring accurate and efficient cluster management.

## Description

The DCM team will collaborate with the EEH Kafka team to gather comprehensive metadata for Kafka topics in both Tier 0 and Tier 1 clusters. This includes topic names, partition details, schema details, consumer groups, and upstream/downstream application details. A script will be developed to automate the capture of technical metadata (topic names, partitions, schemas, and consumer groups) from the Kafka clusters. Additionally, the DCM team will design a system to manage schema updates and other topic modifications within the DCM platform. To streamline the process, a template will be created for the EEH team to provide producer and consumer details, including team information, which will be integrated into the DCM system. This initiative will enhance visibility, traceability, and management of Kafka-based data pipelines.

## Acceptance Criteria

1. **Metadata Collection**:
   * A script is developed and tested to successfully extract topic names, partition details, schema details, and consumer groups from EEH Tier 0 and Tier 1 Kafka clusters.
   * The script runs without errors and populates the DCM system with accurate metadata.
2. **System for Updates**:
   * A system is implemented within the DCM platform to allow seamless updates to Kafka topic schemas and other metadata.
   * The system supports version control for schema changes and logs updates for auditability.
3. **Template for EEH Team**:
   * A standardized template is created for the EEH team to input producer and consumer details, including upstream/downstream application details and team information.
   * The template is user-friendly, validated for completeness, and integrated into the DCM system.
4. **Integration and Validation**:
   * All metadata (technical and producer/consumer details) is successfully integrated into the DCM system and accessible via a unified interface.
   * Validation tests confirm that the integrated data is accurate and consistent with the Kafka cluster configurations.
5. **Documentation and Training**:
   * Comprehensive documentation is provided for the script, update system, and template usage.
   * Training is conducted for the EEH and DCM teams to ensure effective use of the new system and template.

## Design Details

1. **Metadata Extraction Script**:
   * **Functionality**: Connects to EEH Tier 0 and Tier 1 clusters, retrieves topic metadata (name, partitions, schema), and consumer group details using Kafka Admin APIs.
   * **Output**: Stores extracted metadata in a structured format (e.g., JSON) compatible with the DCM database.
   * **Error Handling**: Includes logging and retry mechanisms for robust operation.
2. **DCM System Enhancements**:
   * **Database**: Extend the DCM database schema to store Kafka metadata (e.g., tables for topics, partitions, schemas, consumer groups).
   * **Version Control**: Implement schema versioning to track changes and ensure backward compatibility.
3. **EEH Template**:
   * **Format**: Create a standard DCM template for capturing producer/consumer details, including application names, team contacts, and upstream/downstream dependencies.
   * **Validation**: Include field validation to ensure mandatory fields (e.g., application name, team contact) are provided.
4. **User Interface**:
   * Develop a DCM dashboard module to display Kafka metadata, including topic details, consumer groups, and producer/consumer information.
   * Provide search and filter capabilities for easy navigation.
5. **Testing and Deployment**:
   * **Unit Tests**: Validate script functionality and API endpoints.
   * **Integration Tests**: Ensure seamless data flow from Kafka clusters to the DCM system.