**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.