**Design and Workflow Document: Power BI Refresh History Integration with Kibana**

**1. Project Overview**

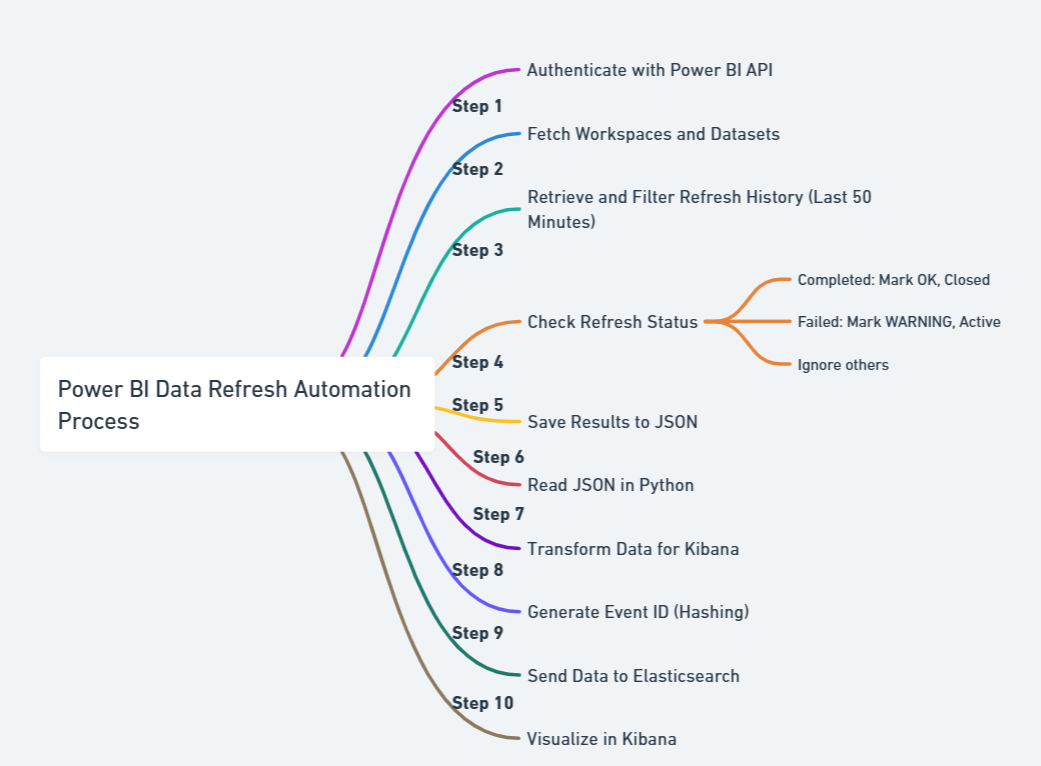
This project automates the extraction, transformation, and indexing of Power BI dataset refresh history. It leverages Power BI APIs to gather refresh data, processes it using Python and PowerShell, and sends the results to a Kibana dashboard for visualization.

**2. Components Involved**

* **PowerShell**: Handles interaction with Power BI APIs to extract refresh history.
* **Python**: Manages workflow orchestration, JSON data processing, and data indexing into Elasticsearch (for Kibana).
* **Elasticsearch/Kibana**: Provides a platform for storing and visualizing refresh data as actionable insights.

**3. Workflow Diagram**

Refer to the attached flowchart for a clear visualization of the entire process.



**4. Process Flow**

**Step 1: Start**

* Initiate the workflow using the Python script.
* Define configuration values (e.g., tenant ID, client ID, client secret) in a separate configuration file (**power\_bi\_config.py**) for security and modularity.

**Step 2: Extract Data from Power BI**

* Run a PowerShell script using the **subprocess** module in Python.
* The PowerShell script:
  1. Authenticates with Power BI using client credentials.
  2. Retrieves the list of Power BI workspaces.
  3. For each workspace, fetches the datasets and their refresh history.
  4. Filters refresh events completed in the last 50 minutes and with specific statuses (**Completed, Failed, and Cancelled**).
  5. Saves the filtered data to a JSON file.

**Step 3: Read and Process JSON Data**

* Python reads the generated JSON file.
* For each entry in the JSON:
  + Extracts critical fields like **Workspace Name, Report Name, Refresh Status**, etc.
  + Logs the details for debugging or verification.

**Step 4: Generate Event Metadata**

* For each dataset entry:
  1. Assigns event metadata such as level (**OK** for **Completed**, **WARNING** for **Failed**), status (**Closed** or **Active**), and a descriptive title.
  2. Creates a unique **EventMatchID** by hashing key fields (**EventSource, EventNode, EventResource, and EventInstance**).

**Step 5: Index Data into Elasticsearch**

* Converts processed data into a payload compatible with Elasticsearch.
* Sends the payload to an index in Elasticsearch (**common-events-\***) with a timestamped index name.
* Handles errors during indexing and logs the status.

**Step 6: Visualize in Kibana**

* The indexed data is available in Kibana for monitoring and analysis. Users can view refresh statuses, identify failures, and drill into detailed metrics.

**Step 7: End**

* Workflow completes successfully. If errors occur, they are logged for troubleshooting.

**5. Tools and Libraries**

* **Power BI API**: Fetches workspace, dataset, and refresh history data.
* **PowerShell**: Automates data extraction and filtering.
* **Python Modules**:
  + **subprocess**: Runs the PowerShell script.
  + **json**: Reads and processes JSON data.
  + **hashlib**: Generates unique hashes for events.
  + **elasticsearch**: Interfaces with Elasticsearch to index data.
* **Elasticsearch/Kibana**: Visualizes indexed data in an intuitive dashboard.

**6. Security Considerations**

* **Configuration File**: Sensitive data like tenant ID, client ID, and secrets are stored securely in Azure vault and securely fetched during release power\_bi\_config.py.
* **Authentication Tokens**: Used temporarily to ensure secure communication with Power BI APIs.

**7. Error Handling**

* **PowerShell Script Errors**: Logs any failures during execution and continues processing other datasets.
* **JSON File Read/Write Errors**: Handles missing or corrupt JSON files gracefully.
* **Elasticsearch Indexing Failures**: Logs detailed error messages if indexing fails.

**8. Key Highlights**

* Filters refresh data intelligently based on time and status.
* Enhances observability by sending data to Kibana in a structured format.
* Provides real-time insights into Power BI dataset refresh performance.