# **Power BI Refresh Strategy Evaluation:**

## **Overview**

To address user-facing query errors and performance degradation during Power BI dataset refreshes, two alternative architectural solutions are proposed. Each approach aims to ensure uninterrupted access to data while maintaining model freshness and minimizing resource usage.

## **Solution 1: Dual Model Architecture with Thin Client Routing**

### **Description**

Solution 1 employs a **dual-model strategy** where two identical semantic models—ARC Risk Model - Node1 and ARC Risk Model - Node2—operate in tandem within the same Power BI Premium workspace. One model serves users (**Reader**), while the other undergoes data refresh operations (**Writer**). A third component, the **Thin Client Model** (ARC Risk Model), acts as a routing layer using a live connection to dynamically connect users to the active Reader model.

This approach isolates users from the refresh process entirely, delivering uninterrupted Import-mode performance with clean DAX logic and high operational stability.

### **Architecture Components**

|  |  |
| --- | --- |
| **Component** | **Description** |
| ARC Risk Model | Thin client model that connects users to the current Reader model via live connection |
| ARC Risk Model - Node1/2 | Full semantic models, identical in structure and logic. Alternate roles between Reader and Writer |

### **Operational Flow**

1. **Initialization**:
   1. Both Node1 and Node2 are deployed with the same schema and business logic.
   2. The Thin Client (ARC Risk Model) is configured to point initially to Node1.
2. **Refresh Cycle**:
   1. The refresh service sets Node1 as the current Reader and Node2 as the Writer.
   2. New data feeds are processed and loaded into Node2 using processData followed by calculate.
   3. Once the refresh completes, and optional data validation passes, the Thin Client is updated to point to Node2.
   4. The roles are flipped: Node2 becomes the Reader, and Node1 is prepared as the new Writer.
3. **Repeat Process**:
   1. This cycle continues throughout the day as new feeds arrive, maintaining consistent user access.

### **Benefits**

|  |  |
| --- | --- |
| Benefit | Description |
| **Uninterrupted User Access** | Users always query the Reader model, which is never under refresh. |
| **High Query Performance** | Queries leverage fully loaded Import tables for optimal speed. |
| **Clean DAX Logic** | No need for conditional logic in measures; queries are routed externally. |
| **Error Isolation** | Refresh failures do not affect the user-facing model. |
| **Supports Pre-Validation** | Data in the Writer model can be verified before going live. |
| **Shadow Replica Compatible** | Fully compatible with Premium refresh replicas for optimized memory usage. |
| **Scales Well** | Ideal for enterprise-scale models with frequent or complex refresh needs. |

### **Considerations**

* Requires automation to manage role switching and Thin Client updates.
* Higher memory usage since both full models are loaded in memory.
* Schema and logic changes must be deployed consistently across both nodes.
* Cache will reset when flipping to a newly refreshed model.

## **Solution 2: Single Composite Model with Dynamic Query Routing via Aggregation Awareness**

### **Description**

Solution 2 leverages Power BI's native **composite model** capabilities to dynamically route queries between Import-mode and DirectQuery-mode storage, depending on the state of the model during refresh. The single semantic model (ARC Risk Model) consists of:

* An **aggregation table** running in **Import mode**, optimized for fast in-memory analytics.
* A **detail table** running in **DirectQuery mode**, connected to Databricks for real-time data access.

Power BI's **aggregation awareness** automatically chooses between Import and DirectQuery tables based on query structure. This solution takes advantage of that behavior and introduces a mechanism to override it dynamically.

A Power BI table named REFRESH\_CONTROL contains a single Boolean column REFRESH\_ACTIVE. This table is managed by the external refresh service:

* When REFRESH\_ACTIVE = FALSE, the model runs normally. Queries use the high-performance Import-mode aggregation table.
* When REFRESH\_ACTIVE = TRUE, base measures in the model apply a dummy, non-impactful filter to an attribute from the DirectQuery table. This forces Power BI to bypass Import tables and execute queries directly against the Databricks detail table.

This approach ensures that users can continue querying the model during refresh without receiving stale or incomplete data.

### **Architecture Components**

* **ARC Risk Model**: Single model using both Import and DirectQuery storage modes.
* **REFRESH\_CONTROL**: Power BI table with a REFRESH\_ACTIVE flag.
* **Refresh Service**: Maintains the flag state during feed/event processing.

### **Operational Flow**

1. **Normal Operation (REFRESH\_ACTIVE = FALSE)**:
   1. Power BI uses aggregation awareness to route most user queries to the in-memory Import-mode table.
   2. This ensures fast performance and efficient use of Premium resources.
2. **During Refresh (REFRESH\_ACTIVE = TRUE)**:
   1. The refresh service sets the control flag REFRESH\_ACTIVE to TRUE before starting any data refresh process.
   2. All base measures in the model are written to check this flag.
   3. When the flag is TRUE, a dummy filter (e.g., DetailTable[ID] <> -1) is applied within the DAX measure logic.
   4. This causes Power BI’s storage engine to bypass the Import table and execute the query using the DirectQuery detail table in Databricks.
   5. This ensures users receive up-to-date results from Databricks, even if Import tables are mid-refresh or incomplete.
3. **After Refresh Completion**:
   1. The refresh service sets REFRESH\_ACTIVE back to FALSE once the data loading and processing operations are complete.
   2. Queries automatically resume running against the optimized Import-mode aggregation table without requiring model switching or reprocessing.

### **Benefits**

|  |  |
| --- | --- |
| Benefit | Description |
| **Single model deployment** | Simplifies lifecycle management—only one semantic model to maintain and version. |
| **Continued availability during refresh** | Users can query the model even when refresh is in progress. Queries are redirected to Databricks. |
| **Efficient use of composite model design** | Leverages Power BI’s aggregation awareness for optimal performance and flexibility. |
| **Lower memory footprint** | No duplication of semantic models—only one model occupies Premium memory. |
| **No external switching logic required** | The query path is automatically managed by DAX logic and the REFRESH\_CONTROL table. |
| **Seamless user experience** | Transitions between Import and DirectQuery modes are invisible to end users. |
| **Ideal for near real-time scenarios** | Works well with event-driven or feed-based refresh pipelines. |

* Lower memory footprint
* No need for model switching or thin client updates
* Seamless fallback to real-time data during refresh

## **Side-by-Side Comparison**

|  |  |  |
| --- | --- | --- |
| Feature / Criteria | Solution 1: Dual-Model + Thin Client | Solution 2: Single Model + REFRESH\_CONTROL |
| **User Query Isolation During Refresh** | ✅ Fully isolated | ⚠️ Partial (uses DirectQuery during refresh) |
| **Query Performance During Refresh** | ✅ Always fast (Import) | ❌ Slower (DirectQuery fallback) |
| **Model Complexity** | Moderate (3 datasets) | High (DAX conditional logic required) |
| **Deployment Overhead** | ❌ Two full models to deploy | ✅ One model only |
| **Automation Requirements** | Requires slot switching & routing | Simple flag update by refresh service |
| **Memory Usage** | ❌ High (both models loaded) | ✅ Low (one model) |
| **DAX Maintenance Burden** | ✅ Clean logic | ❌ Measures must check REFRESH\_ACTIVE |
| **Refresh Validation Control** | ✅ Can validate Writer model | ⚠️ Not natively supported |
| **Shadow Replica Compatibility** | ✅ Fully supported | ✅ Supported (but less beneficial) |
| **Cold Cache Impact** | ⚠️ Cache resets after model switch | ✅ Cache remains intact |
| **Ideal Use Case** | High availability, frequent refresh | Simpler refresh needs, tighter resources |

## **Summary Recommendations**

|  |  |
| --- | --- |
| Priority | Recommended Solution |
| Uninterrupted Import performance | Solution 1 |
| Simple deployment | Solution 2 |
| Frequent event-driven refreshes | Solution 1 |
| Memory-constrained environments | Solution 2 |
| Need for pre-refresh validation | Solution 1 |

Hi Team,

As part of our efforts to eliminate user-facing errors and performance issues during Power BI model refreshes, I’ve evaluated two architectural solutions and documented them in detail. Please find the attached document, which outlines both approaches along with a side-by-side comparison.

Below is a quick summary for your convenience:

### **✅ Solution 1 – Dual-Model with Thin Client Routing**

* Uses two full semantic models (Node1, Node2) plus a thin client (ARC Risk Model) that routes user queries to the current active model.
* While one model is refreshed (Writer), the other remains available to users (Reader).
* Fully isolates users from refresh operations, maintaining uninterrupted Import-mode performance.

**Pros:**

* No user errors during refresh
* High performance (always using Import)
* Supports pre-validation before go-live

**Considerations:**

* Requires more memory (two full models loaded)
* Needs automation for switching and routing

### **✅ Solution 2 – Single Composite Model with REFRESH\_CONTROL**

* A single model with both Import and DirectQuery tables.
* Queries are dynamically routed to DirectQuery during refresh by setting a control flag (REFRESH\_ACTIVE) and injecting a dummy filter in DAX.
* Once refresh completes, queries resume using Import tables.

**Pros:**

* Lower memory usage
* Simpler deployment (only one model)
* Works well for event-driven, near real-time data flows

**Considerations:**

* Slightly slower performance during refresh
* DAX measures require extra logic to check the refresh flag

### **📊 Summary Recommendation**

* For **maximum performance and refresh isolation**, Solution 1 is ideal.
* For **simpler deployment and lower resource usage**, Solution 2 is a strong alternative.

Please review the attached document for full details. I look forward to discussing your thoughts in our next sync.

Best regards,  
 **[Your Name]**