## Query Space Service – API Design Document

### 📄 Overview

The **Query Space Service** is a strategic microservice within the enterprise data platform, designed to **automate, orchestrate, and persist analytical query execution** in a governed, scalable, and cloud-native architecture. It empowers users to define reusable, validated query definitions; execute them based on **business data readiness signals** or **scheduled intervals**; and manage result lifecycle through **built-in, policy-driven retention**.

A core strength of the service is its **intelligent query sourcing mechanism**, which automatically determines the optimal engine for execution:

* **Databricks** is used for queries that are based on straightforward field selection and filters, easily translatable to SQL for performance-efficient processing near the data.
* **Power BI** is selected when the logic involves **complex DAX**, such as advanced calculations, time intelligence, or semantic layer-specific measures.

This smart routing capability optimizes performance and simplifies query management across varied workloads.

Another cornerstone of the platform is its **Retention-Aware Architecture**. Each query can define a retentionPeriodDays value that controls how long historical results are kept. During each execution cycle, the service enforces this retention policy, automatically deleting older records beyond the configured window — ensuring **storage remains bounded**, **compliant**, and **cost-efficient** without requiring manual intervention.

### **🔹 Key Capabilities Include:**

* ✅ **Logical isolation** of analytical logic via reusable Query Spaces.
* ✅ **Retention-first execution model** to control result storage lifecycle.
* ✅ **Smart query routing** between Databricks and Power BI based on complexity.
* ✅ **Trigger-based automation** using business hierarchy readiness signals.
* ✅ **Scheduled execution** (daily, weekly, monthly) aligned with reporting cycles.
* ✅ **Unified result access** via JSON or CSV API responses with row limits.
* ✅ **Execution metadata APIs** for monitoring status, row counts, and retention state.
* ✅ **Secure, governed usage** through role-based access control (RBAC).

## **. Query Space Creation**

The **Query Space Creation** process enables users to define a logical grouping of related analytical queries. A **Query Space** acts as a reusable container that encapsulates common configuration settings—such as execution policies, retention behavior, readiness triggers, and associated queries—into a single managed object.

This design promotes modularity, improves maintainability, and supports advanced orchestration patterns in enterprise analytics.

### **🔹 Purpose**

Query Spaces provide a framework to:

* Isolate execution context for logically related queries (e.g., grouped by report, subject area, or domain).
* Share common execution controls like frequency, retention, and readiness across multiple queries.
* Reduce operational complexity by managing multiple queries as a single executable unit.

### **🔹 Key Attributes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Type** | **Required** | **Description** |
| name | string | ✅ | Unique name of the Query Space (case-sensitive). Used for retrieval and execution. |
| description | string | ❌ | Optional description of the Query Space’s purpose or context. |
| executionFrequency | enum (Daily, Weekly, Monthly) | ❌ | Defines how often the queries in this space should execute. Optional for readiness-based triggering. |
| enableDataReadinessTrigger | boolean | ❌ | When true, the service monitors specified business nodes and automatically triggers execution when all nodes are marked ready. |
| triggerNodes | string[] | ❌ | Business hierarchy nodes (e.g., ["Investment Bank"]) that must be ready before execution. If omitted, readiness will be inferred from query filters. |
| retentionPeriodDays | integer | ❌ | Specifies how long historical results are retained. Older results are purged after each successful execution. |
| queries | array | ✅ | Collection of query definitions associated with this Query Space. Each must include either daxQuery or selectedFields. |

### **🔹 Query Definition Options**

Each query within a Query Space can be defined using one of two strategies:

1. **Raw DAX Execution (daxQuery)**  
    Users can directly provide a full DAX query. This is suitable for advanced scenarios requiring custom measures, calculations, or pre-authored logic.
2. **Dynamic Field Selection (selectedFields)**  
    Users can define a list of output fields and optional filters. The system will auto-generate a SUMMARIZECOLUMNS-based query using the internal DaxQueryBuilder, injecting necessary date filters and validations.

⚠️ If both daxQuery and selectedFields are provided, the system will prioritize selectedFields.

### **🔹 Filter Support**

Optional filters can be applied at the query level to narrow the result set. Each filter must reference a field in the 'TableName'[ColumnName] format and include one or more values:

json

CopyEdit

"filters": [  
 {  
 "fieldName": "Business Hierarchy['Business Group']",  
 "values": ["Investment Bank"]  
 }  
]

### **🔹 CoB Date Enforcement**

All queries are automatically rewritten to enforce a strict CoB Date filter, even if not explicitly defined by the user. This ensures consistency across results and alignment with scheduled or triggered execution dates.

Injected filter:

dax

CopyEdit

'Cob Date'[Cob Date] = DATE(YYYY, MM, DD)

### **🔹 Sample Payload**

json

CopyEdit

{  
 "querySpace": {  
 "name": "Trend Queries",  
 "description": "Views for Trend Report",  
 "executionFrequency": "Daily",  
 "enableDataReadinessTrigger": true,  
 "retentionPeriodDays": 60,  
 "triggerNodes": ["Investment Bank"],  
 "queries": [  
 {  
 "name": "CR\_CreditDelta\_Country\_Trend\_SNDM",  
 "selectedFields": [  
 "Cob Date[Cob Date]",  
 "Risk Factor[MRC Single Country of Risk]",  
 "Business Hierarchy['Business Group']",  
 "Issuer[Bloomberg Industry Sector]",  
 "[Credit Delta]"  
 ],  
 "filters": [  
 {  
 "fieldName": "Business Hierarchy['Business Group']",  
 "values": ["Investment Bank"]  
 }  
 ]  
 },  
 {  
 "name": "ERS\_IB\_CCS\_CCSR\_CS01\_TopAbs",  
 "daxQuery": "EVALUATE SUMMARIZECOLUMNS('Cob Date'[Cob Date], 'Business Hierarchy'[Sector], \"TopAbs\", [TopAbs])"  
 }  
 ]  
 }  
}

### **🔹 Best Practices**

* Always name queries and query spaces descriptively to support traceability.
* Define retention policies proactively to avoid unnecessary result accumulation.
* Use selectedFields whenever possible to allow validation, flexibility, and source-routing optimization.
* Include CoB-specific filters in visualizations and downstream expectations to match retained query result slices.

🔧 Query and Query Space Management

The service provides dedicated endpoints for managing queries and query spaces after creation. These include update and delete operations for lifecycle and administrative control.

#### 🔸 Update Query Space

PUT /powerbi/api/queryspaces/{querySpaceName}

* Updates metadata and query definitions within a Query Space.
* Partial updates are supported.

#### 🔸 Delete Query Space

DELETE /powerbi/api/queryspaces/{querySpaceName}

* Removes the entire Query Space and all its associated query definitions and result data.
* Requires elevated permissions.

#### 🔸 Update Query

PUT /powerbi/api/queries/{queryId}

* Updates a single query’s definition (e.g., DAX expression, filters, fields).

#### 🔸 Delete Query

DELETE /powerbi/api/queries/{queryId}

* Deletes a single query from its parent Query Space.

⚠️ Deleting queries or spaces also removes historical result data associated with them.

## ✅ Query and Query Space Validation

To ensure correctness and consistency, the service performs layered validation when creating or updating queries and query spaces.

### Validation Rules

1. **Required Field Validation**
   * Ensures all mandatory fields are provided.
   * Validates presence of either daxQuery or selectedFields.
2. **DAX Query Validation**
   * Parses and validates raw DAX syntax (if provided).
   * Verifies correctness using DAX parser.
3. **Attribute Format & Existence Check**
   * Attributes in daxQuery, selectedFields, filters, and triggerNodes must exist.
   * Must follow format: 'TableName'[ColumnName]
4. **CoB Date Filter Enforcement**
   * Queries always execute for the current CoB date.
   * The service rewrites or injects a strict filter clause:
   * 'Cob Date'[Cob Date] = DATE(YYYY, MM, DD)
   * Removes any existing CoB filters to enforce alignment.

### Failure Handling

* Validation failures return 400 Bad Request with descriptive error messages.
* Successful validation stores and activates the query or space.

## **📡 3. Data Readiness Monitoring**

**Data Readiness Monitoring** is a core automation capability of the Query Space Service that ensures analytical queries are only executed **when all upstream data dependencies are complete and validated**. This feature helps avoid premature or invalid query executions, which may lead to incomplete or inaccurate results.

The service integrates with the **Power BI ARC Model**, which tracks readiness at the level of business hierarchy nodes such as "Investment Bank", "Personal Banking", or other domain-specific entities. Each query or Query Space can subscribe to these nodes to monitor their readiness status for a given **CoB (Close of Business)** date.

### **🔹 Key Concepts**

|  |  |
| --- | --- |
| **Concept** | **Description** |
| **Trigger Nodes** | Business hierarchy nodes whose readiness signals drive the query execution (e.g., "Investment Bank"). |
| **CoB Date** | The logical execution date for which readiness is evaluated and data is retrieved. |
| **Readiness Event** | A system-generated signal indicating that all required data is available and validated for a specific node and CoB date. |
| **Automatic Execution** | When all required trigger nodes are ready, the Query Space is automatically executed for the matching CoB date. |

### **🔹 How It Works**

1. Each Query Space may define one or more triggerNodes that it depends on.
2. The service continuously monitors readiness status for these nodes using a shared readiness registry provided by the ARC platform.
3. When all triggerNodes are marked as **Ready** for a specific CoB Date, the Query Space is automatically submitted for execution.
4. If any node is **not ready**, execution is deferred.
5. The system re-evaluates readiness periodically or in response to readiness updates published via eventing.

### **🔹 Readiness Monitoring Endpoint**

The service exposes an endpoint to retrieve the real-time readiness status of all configured Query Spaces and their trigger nodes:

GET /powerbi/api/readiness/status

#### **🔸 Sample Response**

json

CopyEdit

[  
 {  
 "querySpace": "Trend Queries",  
 "coBDate": "2025-07-02T00:00:00",  
 "querySpaceStatus": "Pending",  
 "readinessStatus": "Pending",  
 "queryId": 56,  
 "queryName": "CR\_CreditDelta\_Country\_Trend\_SNDM",  
 "queryStatus": "Pending",  
 "totalNodes": 1,  
 "readyNodes": 0,  
 "pendingNodes": 1  
 }  
]

#### **🔸 Response Fields**

|  |  |
| --- | --- |
| **Field** | **Description** |
| querySpace | Name of the Query Space being monitored. |
| coBDate | Date for which readiness is being evaluated. |
| readinessStatus | Overall readiness status (Ready / Pending) for the Query Space. |
| queryName / queryId | Identifier of the individual query in the Query Space. |
| queryStatus | Execution status of the query (Pending, Completed, Running, Failed). |
| readyNodes | Number of trigger nodes currently marked as Ready. |
| pendingNodes | Number of nodes still waiting for readiness confirmation. |

### **🔹 Implicit Trigger Inference**

If no triggerNodes are explicitly defined in the Query Space payload, the service attempts to **infer trigger nodes** automatically from the filter definitions or fields used within the queries. For example, a filter like:

{  
 "fieldName": "Business Hierarchy['Business Group']",  
 "values": ["Investment Bank"]  
}

…will automatically register "Investment Bank" as a readiness dependency.

This provides flexibility for users who want lightweight configurations without duplicating node logic.

### **🔹 Benefits of Readiness Monitoring**

* ✅ **Ensures data integrity** by only running queries when upstream data is complete.
* ✅ **Prevents incomplete or invalid snapshots** caused by early execution.
* ✅ **Automates execution flow** without requiring manual intervention.
* ✅ **Enables robust dependency-driven orchestration** across microservices.

### **🔹 Best Practices**

* Explicitly define triggerNodes when multiple data domains are involved to avoid unintended inferences.
* Monitor readiness status regularly during data processing windows to troubleshoot delays.
* Combine readiness-based execution with retention to enable reliable, time-bound trend analysis.

**4. Data Retention**

The **Data Retention** feature is a foundational component of the Query Space Service. It ensures that historical query results are preserved only for a **defined period**, supporting regulatory requirements, analytical consistency, and system performance. By automatically purging older results after each successful execution, the service helps prevent uncontrolled storage growth and enforces **governed data lifecycle management**.

### **🔹 Purpose**

Data retention allows the platform to:

* Retain historical results for trend analysis, audit trails, or SLA reporting.
* Automatically clean up outdated results based on a defined **retention window**.
* Ensure the system remains performant and cost-efficient over time.

### **🔹 How It Works**

Each query within a Query Space can optionally define a retentionPeriodDays parameter, indicating how many days of historical results should be retained. After each successful query execution, the service checks this setting and **purges all older result records** that exceed the retention window.

This behavior is **isolated per query** and ensures that:

* New results are always retained.
* Older results are safely deleted in the same transactional execution flow.
* Retention logic runs **automatically** without requiring manual intervention.

### **🔹 Retention Enforcement Flow**

1. A query executes successfully for a given CoB Date.
2. The system calculates the **cutoff date** = CoB Date - retentionPeriodDays.
3. All previously stored results for that query **older than the cutoff** are deleted.
4. The latest result is saved and made available through the Results API.

⚠️ Retention is only applied **after successful execution**. Failed or incomplete runs do not trigger cleanup.

### **🔹 Configuration Example**

"retentionPeriodDays": 60

This configuration ensures only the latest **60 CoB dates** of results are retained. Anything older is deleted after each new successful execution.

### **🔹 Internal Implementation**

* The retention logic is executed **in-line during post-processing**, following query success.
* Retention operates **per query**, even within multi-query Query Spaces.
* Cleanup is **idempotent** and will retry once on failure, with errors logged for observability.

### **🔹 Integration with Result APIs**

Data retrieval endpoints always reflect retention:

* Older results will no longer be available via GET /results.
* Attempting to retrieve purged results will return 204 No Content.

### **🔹 Benefits**

* ✅ **Automated data lifecycle management** aligned with governance policies.
* ✅ **Improved storage efficiency** across high-frequency execution workloads.
* ✅ **Guaranteed consistency** of retained time slices for trend reporting.
* ✅ **Reduced operational overhead** by eliminating the need for manual clean-up scripts.

### **🔹 Best Practices**

* Always define a retentionPeriodDays value for recurring queries.
* Align retention settings with downstream data consumers (e.g., reports showing last 30 or 60 days).
* Use tighter retention windows for test, staging, or exploratory queries.
* Monitor retention behavior via execution metadata to ensure expected cleanup.

## **Result Retrieval**

The **Result Retrieval** capability enables consumers of the Query Space Service to **programmatically access previously executed query results**. These results are stored and made available in a structured format, enabling seamless integration with downstream systems, reporting tools, automated pipelines, and audit workflows.

The retrieval mechanism is optimized for **performance**, **scalability**, and **consistency**, with support for output in both JSON and CSV formats.

### **🔹 Purpose**

* Provide structured access to historical query results by Query Space and CoB Date.
* Enable external systems and reporting tools to consume result data via API.
* Support export, comparison, and post-processing use cases.
* Ensure retrieved results reflect applied filters, CoB alignment, and retention policies.

### **🔹 Endpoint**

GET /powerbi/api/queryspaces/results/{querySpaceName}?cobDate=YYYY-MM-DD&format=json|csv

### **🔹 Query Parameters**

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Required** | **Description** |
| querySpaceName | ✅ | The name of the Query Space whose results should be retrieved. Case-sensitive. |
| cobDate | ✅ | The specific CoB Date (Close of Business) for which results are requested. Format: YYYY-MM-DD. |
| format | ❌ | Optional. The desired output format. Supported values: json (default), csv. |

### **🔹 Output Formats**

#### **✅ JSON Format**

Structured tabular data including metadata:

json

CopyEdit

{  
 "results": [  
 {  
 "tables": [  
 {  
 "name": "Result",  
 "columns": [  
 { "name": "Business Group", "dataType": "string" },  
 { "name": "CoB Date", "dataType": "datetime" },  
 { "name": "Credit Delta", "dataType": "double" }  
 ],  
 "rows": [  
 ["Investment Bank", "2025-07-01T00:00:00", -491575928539.063],  
 ["Personal Banking", "2025-07-01T00:00:00", -21618347299.507]  
 ]  
 }  
 ]  
 }  
 ]  
}

* Designed for APIs and automation tools to parse directly.
* Includes data types for all columns.

#### **✅ CSV Format**

csv

CopyEdit

Business Group,CoB Date,Credit Delta  
Investment Bank,2025-07-01T00:00:00,-491575928539.063  
Personal Banking,2025-07-01T00:00:00,-21618347299.507

* Ideal for Excel, Power BI import, or file-based consumers.
* Uses UTF-8 encoding, with comma as delimiter.

### **🔹 Behavior and Constraints**

|  |  |
| --- | --- |
| **Behavior** | **Description** |
| CoB Date Binding | The retrieval is always for a **specific execution date** (CoB Date). No bulk result export is supported. |
| Retention-Aware | If the result for a given CoB Date has been purged due to the configured retentionPeriodDays, the API returns **204 No Content**. |
| Format Limitation | Maximum response size: **1 million rows** per result. If exceeded, query optimization is required. |
| Execution Source | Retrieved results reflect the output of the **executed query**, whether sourced from Power BI or Databricks. |
| Read-Only | Retrieval endpoints are strictly **read-only**. No query execution or modification occurs during this operation. |

### **🔹 Use Cases**

* Automating downstream ingestion of query results into data lakes, warehouses, or dashboards.
* Enabling on-demand download for users who require snapshots for external analysis.
* Validating CoB-specific query outputs as part of reconciliation or quality checks.
* Supporting third-party tools that consume CSV/JSON via RESTful services.

### **🔹 Error Handling**

|  |  |
| --- | --- |
| **Code** | **Scenario** |
| 204 No Content | No result exists for the provided CoB Date (likely purged or never executed). |
| 404 Not Found | The specified Query Space does not exist. |
| 400 Bad Request | Missing or invalid query parameters. |
| 500 Internal Server Error | Unexpected system error during result retrieval. |

### **Execution Summary**

The **Execution Summary** provides a consolidated view of query execution activity within a Query Space. It enables teams to monitor execution history, track data readiness alignment, validate outcomes, and support audit or reconciliation processes.

This summary serves as an essential operational and analytical tool, offering visibility into the performance and behavior of both scheduled and readiness-triggered executions.

### **🔹 Purpose**

* Track when and how each query was executed.
* View the current status of queries and their readiness dependencies.
* Analyze execution timelines, durations, row counts, and outcomes.
* Support observability, alerting, and issue investigation workflows.

### **🔹 Endpoint**

swift

CopyEdit

GET /powerbi/api/queryspaces/execution/summary/{querySpaceName}

### **🔹 Response Schema**

The endpoint returns a list of execution entries for each query in the specified Query Space:

json

CopyEdit

[  
 {  
 "queryId": 72,  
 "queryName": "ERS\_IB\_CCS\_CCSR\_CS01\_TopAbs",  
 "querySpace": "Trend Queries",  
 "coBDate": "2025-07-01T00:00:00",  
 "queryStatus": "Completed",  
 "executionStart": "2025-07-01T02:40:10.144Z",  
 "executionEnd": "2025-07-01T02:40:12.653Z",  
 "rowCount": 89,  
 "outputFormat": "JSON",  
 "executionDurationMs": 2509,  
 "executionSource": "PowerBI",  
 "readyNodes": ["Investment Bank"],  
 "allNodesReady": true  
 }  
]

### **🔹 Key Fields**

|  |  |
| --- | --- |
| **Field** | **Description** |
| queryId / queryName | Unique identifier and name of the executed query. |
| querySpace | Name of the parent Query Space. |
| coBDate | The business date for which the query was executed. |
| queryStatus | Current execution status: Pending, Running, Completed, or Failed. |
| executionStart / executionEnd | Timestamps indicating when execution began and ended. |
| executionDurationMs | Duration of execution in milliseconds. |
| rowCount | Number of rows returned by the execution. |
| outputFormat | The result format: JSON or CSV. |
| executionSource | Indicates the backend engine used: PowerBI or Databricks. |
| readyNodes | List of trigger nodes that were ready at the time of execution. |
| allNodesReady | Boolean indicating if all required trigger nodes were ready. |

### **🔹 Use Cases**

* Validate whether queries executed correctly and returned expected results.
* Investigate execution delays or failures by analyzing duration and timestamps.
* Confirm alignment between execution timing and data readiness.
* Support external dashboards or operational reports using execution statistics.
* Enhance audit trails with detailed logs of when and how queries were run.

### **🔹 Status Values**

|  |  |
| --- | --- |
| **Status** | **Meaning** |
| Pending | Query is registered but not yet executed. |
| Running | Query is currently being processed. |
| Completed | Query executed successfully and result is available. |
| Failed | Execution failed due to an error in processing or data readiness. |

## **📊 7. Power BI Integration**

The **Query Space Service** integrates seamlessly with the **ARC Strategic Power BI Semantic Model**, providing **direct, real-time access to query execution results**. For each Query Space created, a corresponding table is dynamically added to the Power BI model, exposing the query results in a **single, unified, and optimized view**.

These tables use **DirectQuery mode**, ensuring that users always see the latest query results stored in the retention layer without data duplication.

### **🔹 Integration Workflow**

1. **Dynamic Table Creation**
   1. When a Query Space is created, a new table is generated in the Power BI semantic model.
   2. The table name matches the Query Space name and includes all **combined output attributes** from the defined queries.
   3. The schema dynamically evolves as queries in the space are added, updated, or removed.
2. **DirectQuery on Retention Storage**
   1. The Power BI table is configured to run in **DirectQuery mode**, connecting directly to the retention storage (e.g., Delta Lake, Databricks SQL Warehouse).
   2. Every time a user queries the table, Power BI sends a query to the retention storage and retrieves the latest execution results.
3. **Data Flow: From Execution to Reporting**
   1. When a query completes execution, results are written to the retention storage.
   2. The Power BI table reflects this new data instantly, without requiring dataset refreshes or additional ETL processes.

### **🔹 Retention Storage as the Data Source**

The retention storage layer is exposed to Power BI as a **SQL-compatible query endpoint**, typically leveraging:

* Databricks SQL Warehouse
* Azure SQL Database
* Synapse Dedicated SQL Pools
* Delta Lake (with JDBC/ODBC or Databricks SQL Endpoint)

DirectQuery allows Power BI to run live queries on the pre-aggregated, retained data, ensuring fast query times and up-to-date results.

### **🔹 Key Architectural Benefits**

|  |  |
| --- | --- |
| **Capability** | **Description** |
| **Automatic Table Creation** | Power BI tables are created dynamically from the Query Space definition, no manual intervention needed. |
| **Unified Schema** | All output columns from queries are combined into a single Power BI table per Query Space. |
| **Real-Time Access** | DirectQuery mode ensures users always see the latest query execution results. |
| **Optimized Performance** | Queries executed in the retention layer are pre-aggregated for fast analytical reads. |
| **Seamless Integration** | Eliminates the need for separate ETL processes between query execution and reporting. |

### **🔹 Example Use Case**

|  |  |
| --- | --- |
| **Scenario** | **Example** |
| Query Space created | Market\_Risk\_Queries |
| Power BI table created | Market\_Risk\_Queries |
| Queries run daily | Results written to retention storage with a 60-day retention period |
| Power BI report created | DirectQuery retrieves results from retention storage each time the user opens the report |
| Data availability | Users instantly access the latest CoB execution results without waiting for refreshes |

### **🔹 Performance Considerations**

* The retention storage is optimized for **analytical workloads**, using partitioning (e.g., by CoB Date) and indexing where applicable.
* Power BI’s **DirectQuery caching and query folding** optimize performance on large data sets.
* Aggregated results minimize data volume compared to raw transactional queries.

## **Query Space Execution**

The **Query Space Service** provides a dynamic and automated execution framework, capable of running queries based on business data readiness, scheduled intervals, or manual triggers. Execution is optimized through intelligent engine selection, routing queries to either **Databricks** for SQL-translatable logic or **Power BI** for complex DAX expressions.

After execution completes, results are persisted in the retention layer and automatically exposed in the ARC Strategic Power BI model. Once all queries in a Query Space have successfully executed, the service emits a **"Query Space Ready" event**, signaling downstream systems and users that the data is available for consumption.

### **🔹 Execution Modes**

|  |  |
| --- | --- |
| **Mode** | **Description** |
| **Manual Execution** | Triggered via REST API calls by users or automation systems. Suitable for ad hoc runs and testing. |
| **Automatic Readiness-Based Execution** | Automatically triggered when all required trigger nodes in the ARC model are marked ready for a given CoB date. |
| **Scheduled Execution** | Runs at intervals defined by the executionFrequency parameter (Daily, Weekly, Monthly), independent of readiness status. |

### **🔹 Smart Query Execution Routing**

Each query in a Query Space is evaluated to determine the most efficient execution engine:

|  |  |
| --- | --- |
| **Engine** | **Use Case** |
| **Databricks** | For queries built using selectedFields and simple filters that can be directly translated into SQL. Executes close to the data source for optimal performance. |
| **Power BI** | For queries using daxQuery or involving complex DAX expressions, time intelligence, semantic calculations, or custom measures. |

This approach balances performance and flexibility, ensuring queries execute where they run most efficiently.

### **🔹 Execution Lifecycle**

1. **Resolve Execution Context:**  
    Determine the CoB Date, evaluate readiness (for readiness-triggered queries), and resolve the applicable queries.
2. **Run Queries:**  
    Execute all queries in the Query Space for the specified CoB Date, in parallel where possible.
3. **Persist Results:**  
    Save results to the retention storage, respecting the retention policy. Clean up historical data older than retentionPeriodDays.
4. **Make Results Available:**  
    Expose the latest results through the ARC Strategic Power BI model using DirectQuery mode.
5. **Notify Consumers:**  
    Publish a standardized **"Query Space Ready" event** to Azure Event Hub, signaling that the Query Space results are now available.

### **🔹 Event Notification: Query Space Ready**

Upon successful completion of all queries, the system raises an event in Azure Event Hub:

json

CopyEdit

{  
 "eventName": "ARC\_PBI\_TREND\_QUERY\_SPACE\_READY",  
 "eventSource": "ARC\_PBI",  
 "sourceEnv": "PROD",  
 "eventMetaData": [  
 {  
 "businessDate": "2023-04-18",  
 "datasetName": "IBTrendQueries",  
 "sourceUpdateTime": "2023-04-18T10:08:12.091Z"  
 }  
 ]  
}

|  |  |
| --- | --- |
| **Field** | **Description** |
| eventName | Standardized event identifier. |
| eventSource | Logical source system (ARC\_PBI). |
| sourceEnv | Environment (PROD, UAT, etc.). |
| businessDate | CoB date of the execution. |
| datasetName | Name of the Query Space (matches the Power BI table). |
| sourceUpdateTime | Timestamp when the Query Space became ready. |

### **🔹 Execution Endpoints**

|  |  |
| --- | --- |
| **Endpoint** | **Description** |
| POST /powerbi/api/queryspaces/execute/{querySpaceName} | Executes all queries in the specified Query Space for the current CoB Date. |
| POST /powerbi/api/queries/execute/{queryId} | Executes a single query. Useful for targeted runs and testing. |

### **🔹 Execution Status Lifecycle**

|  |  |
| --- | --- |
| **Status** | **Description** |
| Pending | Execution has not started for the current CoB Date. |
| Running | Query execution is in progress. |
| Completed | Query execution succeeded and results are available. |
| Failed | Execution failed. Details logged for troubleshooting. |

Execution status is tracked and exposed via the **Execution Summary** and **Readiness Monitoring** APIs.

### **🔹 Best Practices**

* Use readiness-based execution when upstream data dependencies exist.
* Schedule queries for regular reporting where data readiness is not a gating factor.
* Rely on the ready event to trigger downstream processes, ensuring they consume the latest available data.
* Monitor execution times and failures through the Execution Summary endpoint.
* Review and optimize queries for correct engine routing (Databricks vs Power BI).