MS Defender to ADX Migration

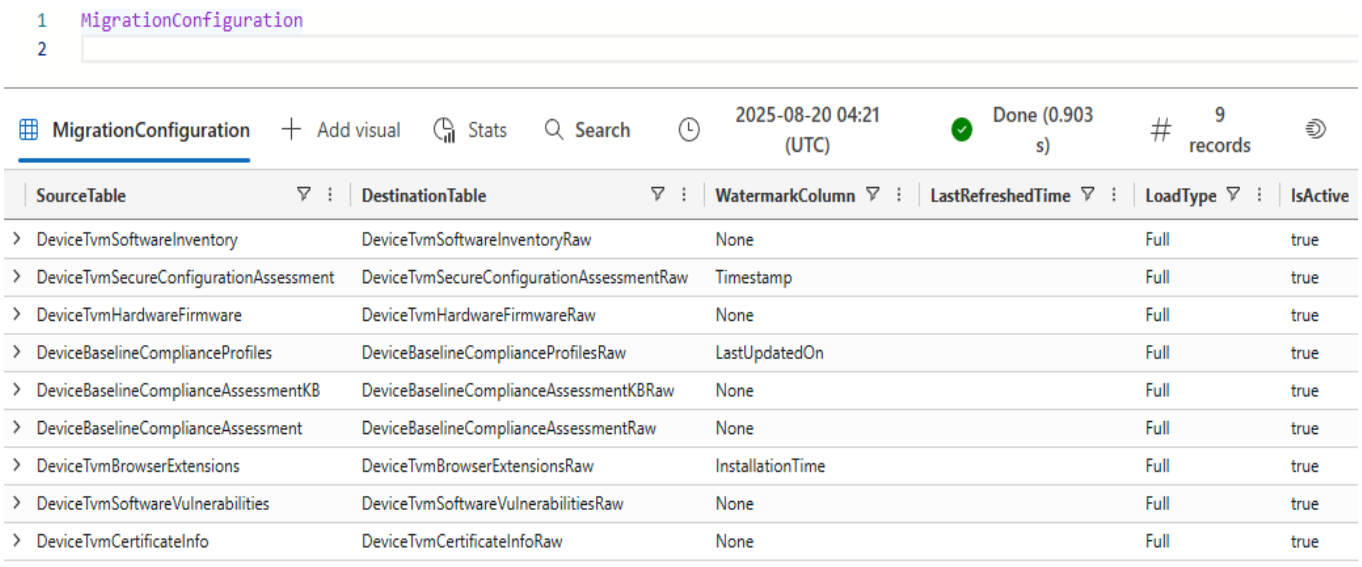
Migration is the process of extracting Microsoft Defender data (via Advanced Hunting API) and loading it into Azure Data Explorer (ADX) tables for analytics, retention, and integration with other security data.

Step-1 Creation of configuration and Audit tables and Other API Tables in Azure Data Explorer Database

a) Migration Configuration Table

This table stores which Defender source tables should be migrated, their destination in ADX, and the state of migration (last refresh time, load type, etc.).

```
.create table MigrationConfiguration (
    SourceTable: string,         // Defender table name (e.g., DeviceNetworkEvents)
    DestinationTable: string,     // ADX table name where data will be ingested
    WatermarkColumn: string,      // Column used for incremental loads (usually timestamp)
    LastRefreshedTime: datetime,  // Last successful ingestion timestamp
    LastRefreshedTimeInString: string, // String version of timestamp
    LoadType: string,           // Full / Incr
    IsActive: bool              // Whether this table should be migrated
)
```



```
MigrationConfiguration
```

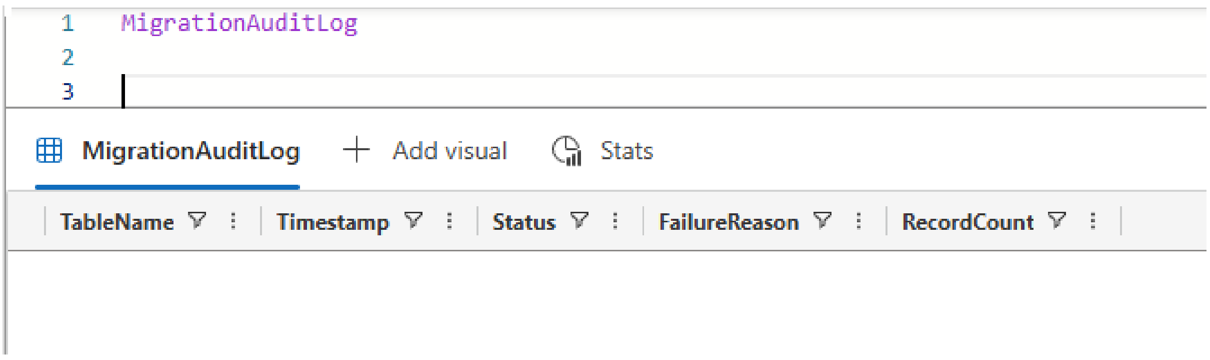
MigrationConfiguration | Add visual | Stats | Search | 2025-08-20 04:21 (UTC) | Done (0.903 s) | 9 records

SourceTable	DestinationTable	WatermarkColumn	LastRefreshedTime	LoadType	IsActive
DeviceTvmSoftwareInventory	DeviceTvmSoftwareInventoryRaw	None		Full	true
DeviceTvmSecureConfigurationAssessment	DeviceTvmSecureConfigurationAssessmentRaw	Timestamp		Full	true
DeviceTvmHardwareFirmware	DeviceTvmHardwareFirmwareRaw	None		Full	true
DeviceBaselineComplianceProfiles	DeviceBaselineComplianceProfilesRaw	LastUpdatedOn		Full	true
DeviceBaselineComplianceAssessmentKB	DeviceBaselineComplianceAssessmentKBRaw	None		Full	true
DeviceBaselineComplianceAssessment	DeviceBaselineComplianceAssessmentRaw	None		Full	true
DeviceTvmBrowserExtensions	DeviceTvmBrowserExtensionsRaw	InstallationTime		Full	true
DeviceTvmSoftwareVulnerabilities	DeviceTvmSoftwareVulnerabilitiesRaw	None		Full	true
DeviceTvmCertificateInfo	DeviceTvmCertificateInfoRaw	None		Full	true

b) Migration Audit Log Table

This table stores the **status of each ingestion run** for monitoring & troubleshooting.

```
.create table MigrationAuditLog (
    TableName: string,      // Destination table name
    Timestamp: datetime,     // When the migration ran
    Status: string,         // Success / Failed
    FailureReason: string,  // Error message if failed
    RecordCount: int        // Number of records ingested
)
```



```
MigrationAuditLog
```

MigrationAuditLog | + Add visual | Stats

TableName	Timestamp	Status	FailureReason	RecordCount

c) Destination (API) Tables

We are loading data from Defender tables into following instance of table

We have created RAW tables with Records column as dynamic datatype and and created an json ingestion mapping name : DynamicJsonMap as following

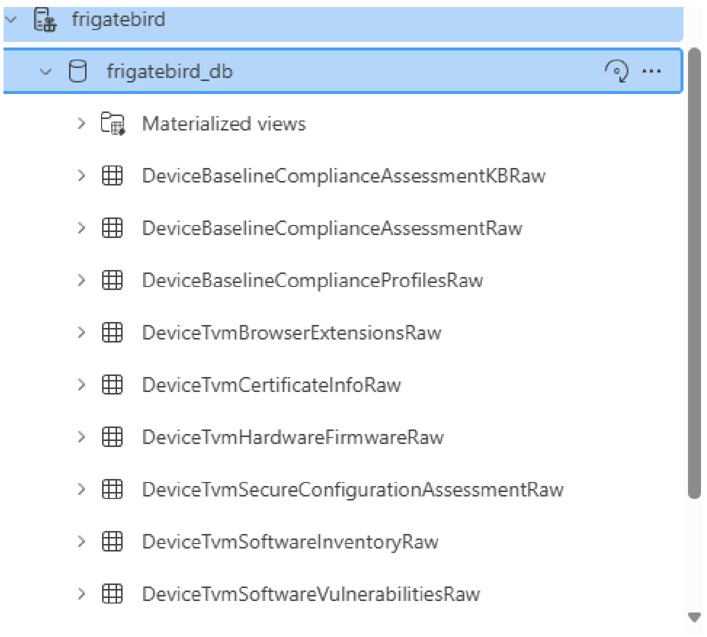
e.g.

```
.create table DeviceTvmSoftwareInventoryRaw (Records: dynamic)

.create-or-alter table DeviceTvmSoftwareInventoryRaw ingestion json mapping
'DynamicJsonMap'

[
 {"column":"Records","Properties":{"path":"$"},"datatype":"dynamic"}
]
```

Following is destination Raw Tables in ADX database where we are storing data from MS Defender Tables as shown in the following POC screen shot


frigatebird
frigatebird_db
Materialized views
DeviceBaselineComplianceAssessmentKBRaw
DeviceBaselineComplianceAssessmentRaw
DeviceBaselineComplianceProfilesRaw
DeviceTvmBrowserExtensionsRaw
DeviceTvmCertificateInfoRaw
DeviceTvmHardwareFirmwareRaw
DeviceTvmSecureConfigurationAssessmentRaw
DeviceTvmSoftwareInventoryRaw
DeviceTvmSoftwareVulnerabilitiesRaw

Step-2 : Access to ADX for service principal

To grant Azure Data Explorer (ADX) access to a service principal, you need to assign the service principal appropriate security roles. These roles determine what actions the service principal can perform within the ADX cluster and its databases.

We can provide viewers ,ingestors and users roles in the ADX DB.

.add database DBName viewers ('aadapp=clientId ;tenantId')

.add database DBName ingestors ('aadapp=clientId ;tenantId')

.add database DBName users ('aadapp=clientId ;tenantId')

We can add any users as following

.add database DBName viewers ('aaduser=xxxxx@domain.com') ' xxxxx@domain.com'

Step-3: Providing access to service principal for defender API

Granting service principal access to Microsoft Defender APIs

To automate interactions with Microsoft Defender APIs (e.g., Microsoft Defender for Endpoint, Microsoft Defender for Cloud Apps, or Microsoft Defender XDR), you need to grant a Service Principal in Azure Active Directory (Azure AD) the necessary permissions. This involves registering an application, assigning the appropriate API permissions, and granting admin consent.

Here's a step-by-step guide:

1.Register an application in Azure AD

- Sign in to the Azure portal.
- Navigate to Azure Active Directory > App registrations > New registration.
- Choose a descriptive name for your application (e.g., "DefenderAPIIntegration") and select Register.

2. Grant API permissions

- On your newly created application's page, select API permissions > Add a permission.
- Select the relevant Defender API based on your needs (e.g., WindowsDefenderATP for Microsoft Defender for Endpoint, Microsoft Cloud App Security for Microsoft Defender for Cloud Apps, or Microsoft Threat Protection for Microsoft Defender XDR).
- Choose Application permissions as the permission type (as you are likely running a background service or daemon without a signed-in user).
- Select the specific permissions required by your application. Examples include:
  - **Alert.Read.All: Read all alerts.**
  - **Machine.Read.All: Read all machine information.**
  - **Software.Read.All: Read all software information.**
  - **User.Read.All: Read user information.**
  - **Vulnerability.Read.All: Read all vulnerability information.**
  - **Incident.Read.All: Read Incident related information**
  - **CustomDetections.ReadWrite.All: allows an application to read and write custom detection rules on behalf of the signed-in user**
  - **AdvancedHunting.Read.All: allows applications to access advanced hunting data in Microsoft Defender for Endpoint**
- Click Add permissions.

3. Grant admin consent

- After adding the necessary permissions, select Grant admin consent for to grant consent for your organization.
- Confirm the action by clicking Yes.

4. Obtain application credentials

- On your application's overview page, locate and copy the Application (client) ID and the Directory (tenant) ID.
- Go to Certificates & secrets and create a New client secret.
- Provide a description for the secret and specify an expiry period.
- Important: Copy the generated client secret value immediately, as it will only be displayed once.

5. Using the credentials in your application

- In your application, you will use the Application (client) ID, Directory (tenant) ID, and the client secret to authenticate with Azure AD and obtain an access token.
- This access token can then be used in the Authorization header (as "Bearer {token}") when making requests to the Defender APIs.

By following these steps, you can successfully provide your service principal with the necessary access to interact with Microsoft Defender APIs. You can find more specific examples for different Defender APIs and permission scopes in the official Microsoft documentation.

Step-4: Step-by-Step Flow of the Script

1. **Prerequisites / Imports**
   - Load Python libraries (azure-kusto-data, azure-kusto-ingest, requests, etc.).
   - Define API limits, page sizes, and globals for throttling.

---

2. **API Throttling Control (api_limiter)**
   - Keeps track of Defender API calls.
   - Ensures no more than API_LIMIT (45/min) requests are sent.
   - If limit reached → script sleeps until reset window (60 sec).

---

3. **Defender API Wrapper (defender_api_post)**
    - Wraps the POST call to Defender's Advanced Hunting API.
    - Handles:
        - 429 Too Many Requests → retries after Retry-After header, reduces PAGE_SIZE.
        - 500+ Server Errors → waits 15 sec and retries.
        - Other errors → raises exceptions.

---

4. **Token Acquisition (GetADXAndDefenderTokenWithIngestURI)**
    - Requests AAD token for Defender API access.
    - Requests ADX token for Kusto cluster ingestion.
    - Returns (aad_token, adx_token, adx_ingest_uri).
    - Tokens auto-refresh every ~1 hour inside the ingestion loop.

---

5. **Main Migration Function (ingest_defender_data)**

(a) Connect to ADX

- Creates KustoClient (for queries/config) and QueuedIngestClient (for ingestion).

(b) Get Migration Configuration

- Reads VwLatestMigrationConfiguration view from ADX.
- For each row:
    - SourceTable → Defender API table name.
    - DestinationTable → ADX target table.
    - WatermarkColumn → timestamp column used for incremental loads.
    - LastRefreshedTime → last sync time.
    - LoadType → "Full" or "Incr".

(c) Watermark Setup

- If table already ingested once → use incremental mode.
- If new → start from default 1900-01-01.

(d) Pagination Loop

- Fetches Defender data in batches using PAGE_SIZE.
- Uses row_number() in KQL to page through results.
- Keeps looping until no more data.

(e) Ingest Batch to ADX

- Converts API records to JSON string.
- Streams them to ADX using QueuedIngestClient with mapping "DynamicJsonMap".

(f) Update Migration Configuration

- Writes back the latest LastRefreshedTime (max timestamp from batch).
- Keeps migration state up-to-date.

(g) Token Refresh

- If tokens are older than ~3500 seconds (≈1h), regenerate them.

---

6. **Audit Logging**
   - Logs each migration attempt (success/failure) into MigrationAuditLog.
   - Captures:
     - Table name
     - Timestamp
     - Status (Success / Failed)
     - Error reason
     - Record count ingested

End-to-End Workflow

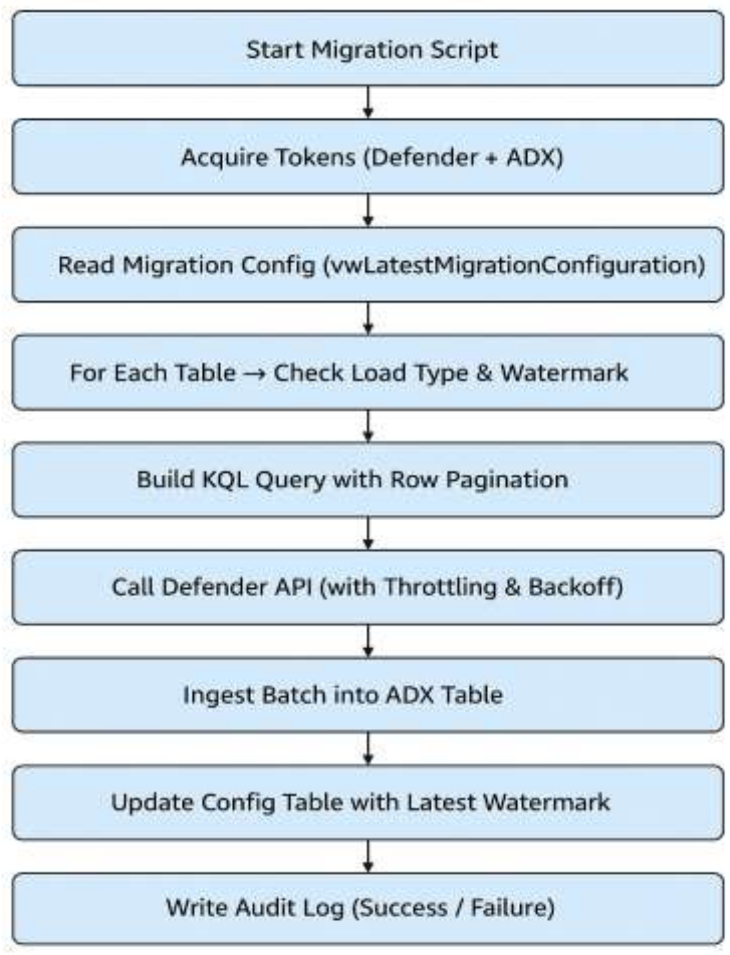
1. Script connects to ADX & Defender.

2. Reads migration config to know which tables need migration.

3. For each table:

- Calculates the watermark (incremental cutoff).
- Pulls data from Defender API in pages (with throttling & backoff).
- Ingests into destination KQL table.
- Updates config table with latest timestamp.
- Writes audit entry.

Flow diagram (step by step) for Migration Pipeline


Start Migration Script
Acquire Tokens (Defender + ADX)
Read Migration Config (vwLatestMigrationConfiguration)
For Each Table → Check Load Type & Watermark
Build KQL Query with Row Pagination
Call Defender API (with Throttling & Backoff)
Ingest Batch into ADX Table
Update Config Table with Latest Watermark
Write Audit Log (Success / Failure)

Execution of Scripts

Step-1 : ADX Database Script

Need to put value for appreg_appid and appreg_tenantid in the script for role of users and ingestors in the above script

Step-2: Insert Script for Configuration Table

Python Script: Migration Script

--------------------Prerequisite to install following Python Libraries-------------------------

```
!pip install azure-kusto-data

!pip install azure-kusto-ingest
```

Before running the python script need to update   clientId, clientSecret and tenantId in the scrip