Extend KQL Data with Azure Function as Resource Exporter

In complex cloud environments, effective monitoring and observability increasingly require data from sources beyond native Azure telemetry. While Azure native resources in combination with Azure Log Analytics offer deep visibility into Azure resources, they don’t directly support querying external systems such as Microsoft Graph API, or third-party platforms like ServiceNow, CRMs, etc. To bridge this gap, you can extend the telemetry using Azure Functions that collect, transform, and retrieve external data on demand using HTTP requests and making it accessible through Kusto Query Language (KQL).

> This is a **conceptual architecture** meant to demonstrate how you can enrich your observability stack. It’s not a one-click or fully packaged solution, but rather a pattern that can be tailored to your specific API sources, data schemas, and monitoring goals.

Architecture

At its core, the approach involves using Azure Functions as external data connector — fetching data from APIs, converting it into structured records, and making it available to your existing monitoring data platform. Once the Azure Function is created, it becomes available to query, visualize, and alert on. It then works just like native or custom Azure logs.

This pattern unlocks powerful use cases, such as:

- Correlating Entra ID sign-ins with metadata from ServiceNow or Microsoft Graph API.
- Contextualizing security alerts with vulnerability data from external scanners.
- Tracking compliance-related signals alongside Azure Policy states.

In the sections that follow, it is outlined how this architecture can be implemented, the technologies involved, and important operational considerations for building a reliable, secure, and maintainable integration.

Azure Function

The Azure Function acts as a lightweight API endpoint that retrieves data from external systems, transforms it into structured JSON, and serves it to KQL using an HTTP trigger. This allows real-time enrichment of queries with external context.

Authentication

To securely access external APIs like Microsoft Graph, your Azure Function must authenticate properly. The recommended method is using **Managed Identity**, which allows the function to obtain tokens without storing secrets.

Ensure the function's identity has the necessary Graph API permissions (e.g., `Group.Read.All`) granted via Entra ID and admin consent. Learn more about System Managed Identities here: Use System Managed Ide... | LNC DOCS

Example Code

```
using namespace System.Net

param($Request, $TriggerMetadata)

if($request.Query.evaluationType -eq "entraidgroups" -and $request.Query.groupid){
    # Get Graph API Access Token
    Import-Module Microsoft.Graph.Authentication
    $requiredPermissions = @(
        "Group.Read.All"
    )
    $permissionsList = $requiredPermissions -join ', '
    Connect-MgGraph -Identity -NoWelcome
    $mgGraphRequest = Invoke-MgGraphRequest -Uri "https://graph.microsoft.com/v1.0/" -
OutputType HttpResponseMessage
    $Global:MicrosoftEntraIDAccessToken =
$mgGraphRequest.RequestMessage.Headers.Authorization.Parameter

    $uri =
"https://graph.microsoft.com/v1.0/groups/$($request.Query.groupid)/members"#?`$select=id,displ
ayName"
    $allGroups = Invoke-RestMethod -Method GET -Uri $uri -Header @{Authorization = "Bearer
$Global:MicrosoftEntraIDAccessToken" }
```

```
    $outputjson = ($allGroups.value)
}

Push-OutputBinding -Name Response -Value ([HttpResponseContext]@{
    StatusCode = [HttpStatusCode]::OK
    Body = $outputjson
})
```

KQL Query

Once your Azure Function is returning structured JSON data, you can integrate it into your monitoring workflow using `externaldata()` in KQL:

```
externaldata(id:string, displayName:string) // Extend with fields from your API response
[

'https://<functionappname>.azurewebsites.net/api/<functionname>?code=<functionaccesskey>&evalu
ationType=entraidgroupmembers&groupid=<yourgroupid>'
]
with(format='multijson')
```

Key Considerations

- **Schema Definition**: The columns in `externaldata()` must match the structure of the JSON returned by your Function. Incorrect or missing fields will cause the query to fail or return nulls.
- **Security**: Avoid embedding access keys directly in shared queries. Use managed identity or securely store secrets if possible.
- **Performance**: This pattern pulls data at query time — ideal for on-demand enrichment, but not suitable for high-frequency or large-scale queries. Cache results where needed.
- **Error Handling**: Ensure your Function returns consistent, valid JSON. Add logging and status code checks to troubleshoot failed queries.

With the right setup, this method allows seamless integration of external data into your KQL dashboards and alerts, expanding the scope of what you can monitor and correlate in Azure.

Revision #3
Created 2025-05-31 19:04:41 UTC by Luca Noah Caprez
Updated 2025-06-11 09:18:18 UTC by Luca Noah Caprez