Azure Log Analytics

- KQL quick commands
- Write custom logs via PowerShell
- Query Azure Firewall Logs
- Quick Start Guide for Azure Workbooks
- Use of variables & dynamic content in Azure Workbooks
- Create log health dashboard using time based data

KQL quick commands

Count entries in table

```
<loganalyticstablename> | summarize count()
```

Get log storage usage

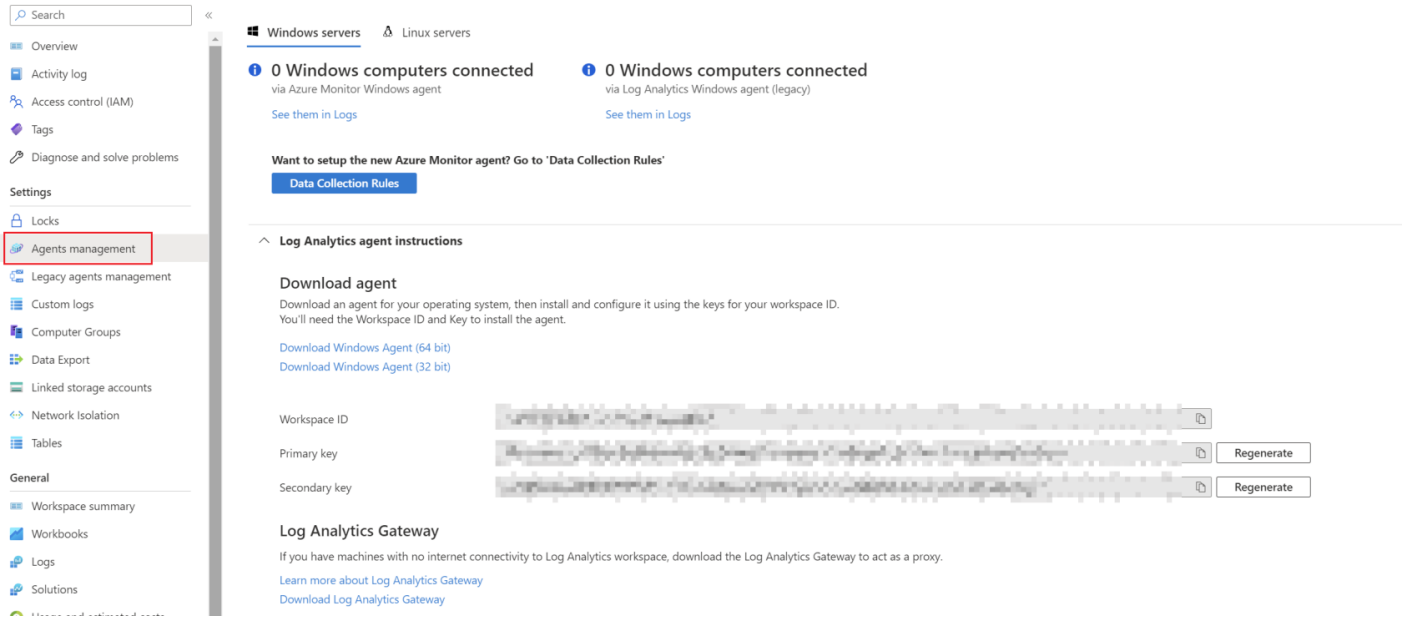
```
Usage | where IsBillable | summarize DataGB = sum(Quantity / 1000)
```

Write custom logs via PowerShell

With Log Analytics you can write custom logs. For this the API can be used to write new logs to the Log Analytics Workspace. The whole API is documented here: https://learn.microsoft.com/en-us/rest/api/loganalytics/create-request

Script template

This script template uses the first three variables. The first two can be found under "Agents management" -> "Log Analytics agent instructions".



The third variable "LogType" defines the type of log you are going to send. This can be the same as existing log entries or a completly new one. This type of log defines if a new table or an existing table is created or used to store the logs.

```
$customerId = ""
$sharedKey = ""
$LogType = ""

Function Build-Signature ($customerId, $sharedKey, $date, $contentLength, $method,
```

```
$contentType, $resource){
    $xHeaders = "x-ms-date:" + $date
    $stringToHash = $method + "`n" + $contentLength + "`n" + $contentType + "`n" + $xHeaders +
"`n" + $resource

    $bytesToHash = [Text.Encoding]::UTF8.GetBytes($stringToHash)
    $keyBytes = [Convert]::FromBase64String($sharedKey)

    $sha256 = New-Object System.Security.Cryptography.HMACSHA256
    $sha256.Key = $keyBytes
    $calculatedHash = $sha256.ComputeHash($bytesToHash)
    $encodedHash = [Convert]::ToBase64String($calculatedHash)
    $authorization = 'SharedKey {0}:{1}' -f $customerId,$encodedHash
    return $authorization
}

Function Post-LogAnalyticsData ($customerId, $sharedKey, $body, $logType){
    $method = "POST"
    $contentType = "application/json"
    $resource = "/api/logs"
    $rfc1123date = ([DateTime]::UtcNow).ToString("r")
    $contentLength = $body.Length
    $signature = Build-Signature -customerId $customerId -sharedKey $sharedKey -date
$rfc1123date -contentLength $contentLength -method $method -contentType $contentType -resource
$resource

    $uri = "https://" + $customerId + ".ods.opinsights.azure.com" + $resource + "?api-
version=2016-04-01"

    $headers = @{
        "Authorization" = $signature;
        "Log-Type" = $logType;
        "x-ms-date" = $rfc1123date;
    }

    $response = Invoke-WebRequest -Uri $uri -Method $method -ContentType $contentType -Headers
$headers -Body $body -UseBasicParsing
    return $response.StatusCode
}
```

```
$Properties = [Ordered] @{
    "ComputerName"     = $env:computername
    "User"             = $env:Username
}

$CustomLogs = New-Object -TypeName "PSObject" -Property $Properties | ConvertTo-JSON -Depth 10



#Submit the data to the API endpoint
$params = @{
    CustomerId = $customerId
    SharedKey  = $sharedKey
    Body       = ([System.Text.Encoding]::UTF8.GetBytes($CustomLogs))
    LogType    = $LogType
}
$LogResponse = Post-LogAnalyticsData @params
$LogResponse
```

The variable "LogResponse" is in a successful execution filled with the value 200.

Use Case

A possible use case is for example is to write an Azure Function to provide a custom Rest API to write logs to a function. This has multiple advantages over writing directly to the Log Analytics Workspace. Everything from less code to credential leaks will be provided with a simple REST Call to the Azure Function. More on this in the chapter "Azure Function".

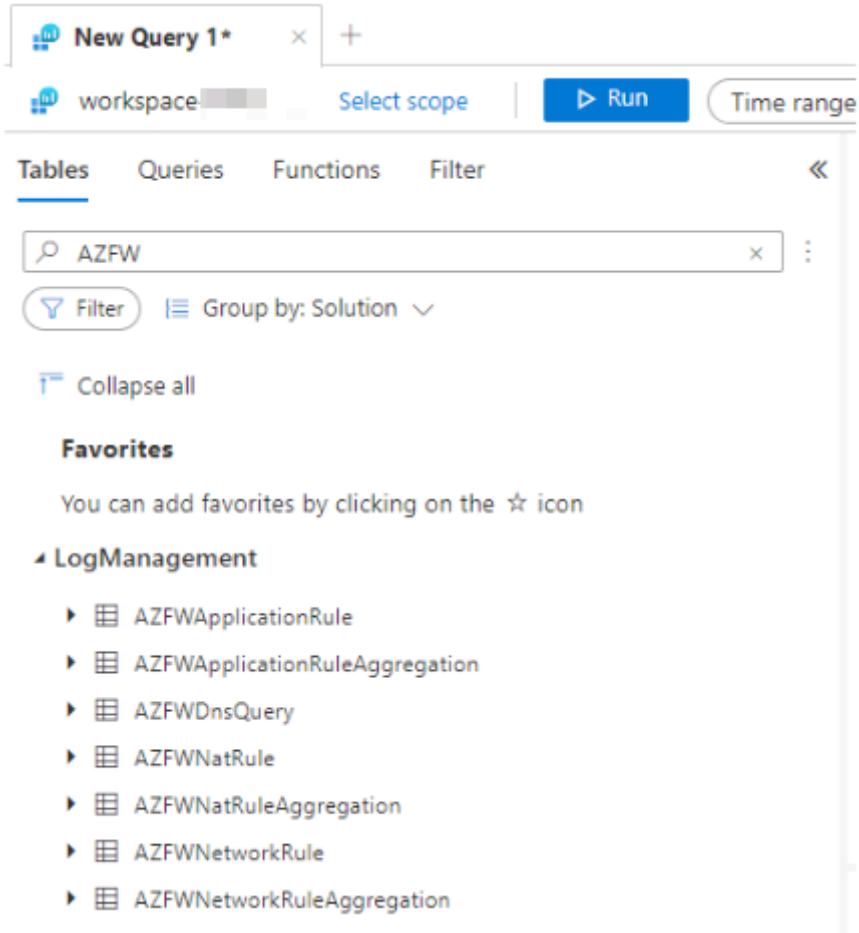
Query Azure Firewall Logs

Azure Firewall Logs can be stored in an Azure Log Analytics Workspace. This workspace then contains all status logs along with permitted and denied connections. So, to find out if a connection is wrongly blocked or to make a specific firewall request, we can use these logs to give us insights.

Find log tables

First of all you have to select the scope on which you want to search for the logs. You can choose the Log Analytics scope with "Select scope".

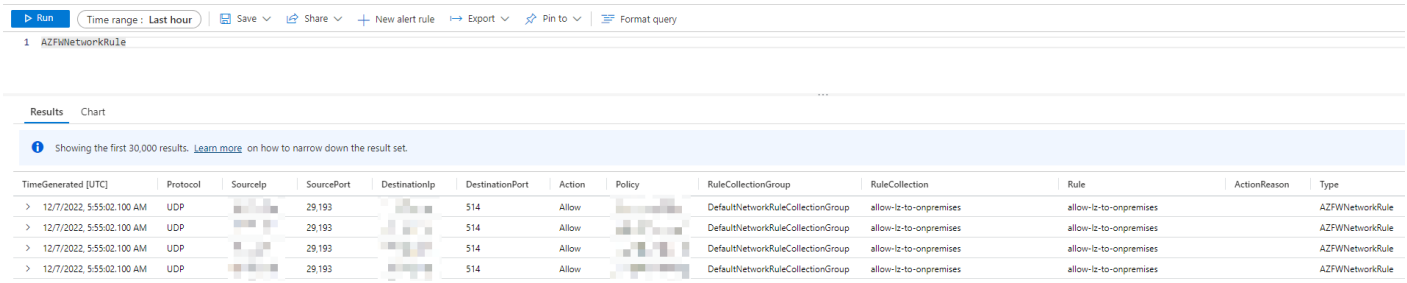
Azure Firewalls save logs to different tables. To find all the different log table you can search in the tables pane for "AZFW". These Tables contain the corresponding log data.



Azure Firewall rule logs are stored within the "AZFWNetworkRule" table.

View whole table content

To view the whole table content you can write the name of the table into the KQL (Kusto Query Language) section. In this case "AZFWNetworkRule" is enough to see all the permitted and denied connections.



Filter logs after IP address

Most of the time we want to filter for specific addresses. These Firwall logs can be queried with the powerful KQL language. This language helps to explore data and discover patterns, identify anomalies and outliers, create statistical modeling, and more. The query uses schema entities that are organized in a hierarchy similar to SQL's: databases, tables, and columns.

Filter for source IPs

```
AZFWNetworkRule
| where SourceIp == "<yoursourceipadress>"
```

Filter for destination IPs

```
AZFWNetworkRule
| where DestinationIp == "<yourdestinationipadress>"
```

Quick Start Guide for Azure Workbooks

Azure Workbooks are a powerful tool for building interactive dashboards and visualizing data at scale. This guide provides you with a structured, practical approach to deploying and managing Azure Workbooks, based on real-world experience and best practices.

Use case

In today's data-driven world, cloud services like Azure, Microsoft 365 and others provide a wealth of data waiting to be analyzed. Azure Log Analytics, powered by Kusto Query Language (KQL), enables users to explore and visualize data effectively. Azure Workbooks are ideal for building KPI dashboards, providing health overviews of systems, analyzing historical trends, monitoring security and compliance, delivering custom reports for stakeholders, and visualizing DevOps or automation metrics. Let's dive into how to leverage KQL-Based Workbooks in Azure Log Analytics to gain valuable insights through compelling graphs and diagrams.

Azure Workbooks Overview

Azure Workbooks are a flexible data visualization platform within the Azure ecosystem. They enable IT professionals to create interactive dashboards using KQL-based graphs and fetch live data from multiple sources. Since Azure Workbooks are no more than standard Azure resources this means that they can be customize with Azure RBAC permissions. Workbooks are highly portable and scalable, leveraging a JSON-based structure for easy export, import, and sharing. Filtering options via parameters allow customization for different audiences.

Key capabilities

- Aggregate data from diverse sources for unified, single-pane-of-glass monitoring.
- Integrate live data from Log Analytics, Microsoft Graph API, Azure Resource Graph, external HTTP endpoints, serverless functions, storage buckets, and third-party APIs.
- Control access with Azure RBAC for secure, role-based visibility.
- Easily scale, export, and import dashboards across environments.
- Customize dashboard views using parameter filtering.

Technology stack

Azure Workbooks are built on JSON templates work like the Azure Resource Manager (ARM) templates. Therefore it is very simple to automate it via REST APIs and the Azure Management API (https://management.azure.com). They are also compatible with Infrastructure as Code (IaC) tools such as Bicep & Terraform.

Data sourcing

Azure Workbooks support a broad range of data sources, making them highly adaptable for complex monitoring and reporting needs. This flexibility enables you to visualize and analyze data from virtually any system within your IT environment.

Built in data sources:

- **Log Analytics Workspace**
- **Microsoft Graph API***
- **Azure Resource Graph**
- **External HTTP Endpoints**

Using the "externaldata" functionality from KQL:

- **Serverless Functions**
- **Storage Buckets**
- **Third-Party APIs**

For scenarios involving complex data aggregation, transformation or exotic permissions on APIs, it is best practice to offload these tasks to Azure Functions or similar serverless solutions. This approach ensures scalability and keeps your dashboards responsive, even when working with large or intricate datasets.

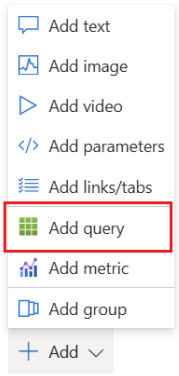
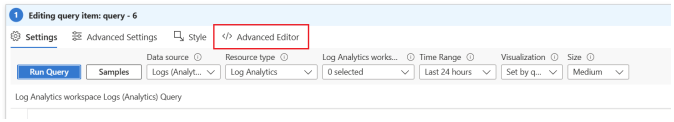
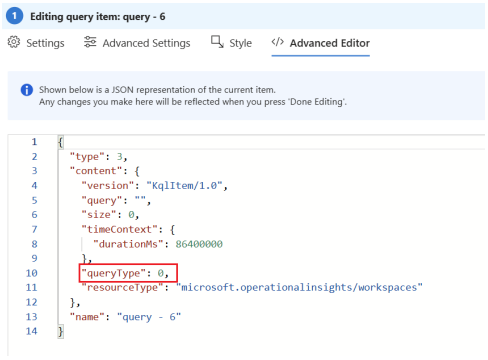
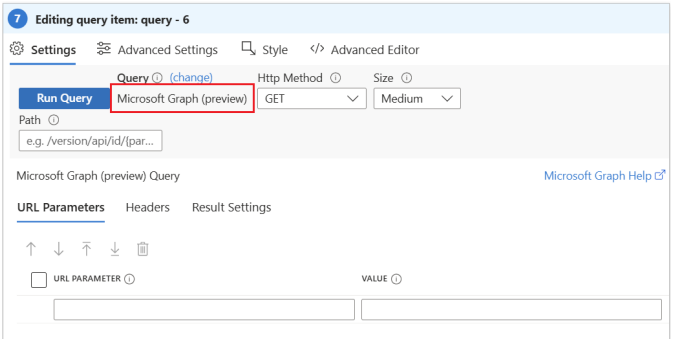
Graph API permission limitations

The integration of Microsoft Graph API is currently limited to these scopes (because of the service principle “Microsoft Azure AppInsightsExtension” under the hood):

- DeviceManagementApps.Read.All
- DeviceManagementConfiguration.Read.All
- DeviceManagementManagedDevices.Read.All
- Policy.Read.All
- Policy.Read.ConditionalAccess
- User.Read
- User.ReadBasic.All
- WindowsUpdates.ReadWrite.All
- profile email openid

Graph API source availability

The data source "Microsoft Graph API" is currently only available using the Advanced Editor with the adjustments of JSON values. Unfortunately it is not selectable in the GUI yet (may change soon).

On your Azure Workbook click "Add" and select "Add query".	
Change to "Advanced Editor" tab.	
Adjust the JSON property "queryType" **from 0 (default) to 14**. This is the id of the Microsoft Graph API query type. Hit "Done Editing" below.	
Change to the editor mode with "Edit". Now you can look under "Settings" -> "Query" that it changed to "Microsoft Graph API" query. Now you are ready to build your visualizations.	

Create dashboards yourself

Workbooks give you the flexibility to go beyond prebuilt templates and design dashboards tailored to your own environment. By combining data sources, parameters, and custom visualizations, you can build views that answer exactly the questions your team cares about—whether it’s operational monitoring, cost tracking, or security insights. This section shows you how to start from scratch and

gradually shape a workbook into a reusable dashboard.

1. **Creating an Azure Workbook:** Navigate to the Azure portal and search for "Workbooks" Then create a new workbook or continue editing existing Workbooks. You can also choose from existing templates to accelerate your analysis.
2. **Setting Up Azure Log Analytics (optional):** Ensure you have an active Azure subscription and access to your desired data inside Azure Log Analytics workspaces. For custom logs you can create a custom Log Analytics workspace from the Azure portal and connect your data sources, such as Azure Monitor, Application Insights, or custom logs.
3. **Writing KQL Queries:** In the workbook, use Kusto Query Language (KQL) to craft queries that retrieve data from your Log Analytics workspace. KQL allows you to filter, aggregate, and transform data efficiently.
4. **Visualization with Graphs:** After querying the data, enhance your insights by creating visualizations with graphs. Choose from various graph types such as line charts, bar charts, pie charts, or area charts, depending on the data and the story you want to convey.
5. **Diagram Visualization:** Apart from traditional graphs, KQL-Based Workbooks enable you to create dynamic and interactive diagrams. Utilize diagrams like topology maps, network diagrams, or flowcharts to depict the relationships and dependencies within your data.
6. **Interactive Controls:** Enhance user experience by adding interactive controls like drop-down menus, time range selectors, or variable inputs. These controls allow users to customize and explore data on their terms.
7. **Sharing and Collaboration:** Share your workbook with relevant stakeholders, empowering them to view and interact with the data-driven insights. Collaborate with teams by granting them access to workbooks or exporting them for offline review.
8. **Automate with Scheduled Queries:** Automate data analysis by scheduling queries to run at regular intervals. This ensures that your workbooks stay up-to-date with the latest data without manual intervention.

Tips from the Field

Version Control System for Workbook Templates

- **Always export to JSON**: Don’t rely on the portal as your single source of truth. Export your workbook and keep it in Git (GitHub/Azure DevOps) so changes are tracked.
- **Keep changes small**: Workbook JSON can be huge; commit only what’s necessary so diffs stay readable.
- **Review like code**: Use pull requests to avoid accidental overwrites when multiple people edit the same workbook.
- **Automate deployments**: Bicep/ARM templates and a pipeline (e.g., GitHub Actions) make it easy to roll out workbooks to test and prod without manual clicks.

Performance Optimizations

- **Filter first, visualize later**: In KQL, apply `where` filters early to cut down the dataset before summarizing or charting.
- **Default to shorter time ranges**: Don’t make “Last 30 days” your default – that’s a recipe for slow queries. Stick to 24h or 7d for interactive dashboards.
- **Aggregate smartly**: Summarize data before rendering charts. Plotting millions of rows isn’t useful and often just hits limits.
- **Watch out for hidden limits**: Workbooks and Log Analytics both truncate results – if you see “partial data,” simplify the query.
- **Test outside the workbook**: Run your query in Log Analytics first. If it’s slow there, it won’t magically be fast in a dashboard.
- **Cache where possible**: For static or near-static data (like subscription metadata), use parameters or external data sources instead of re-querying every refresh.

RBAC Permissions Work

One of the most common real-world blockers in adopting Azure Workbooks is that **your developer permissions are not the same as your viewer’s**. A workbook that looks perfect to you may show empty visuals or errors to someone else.

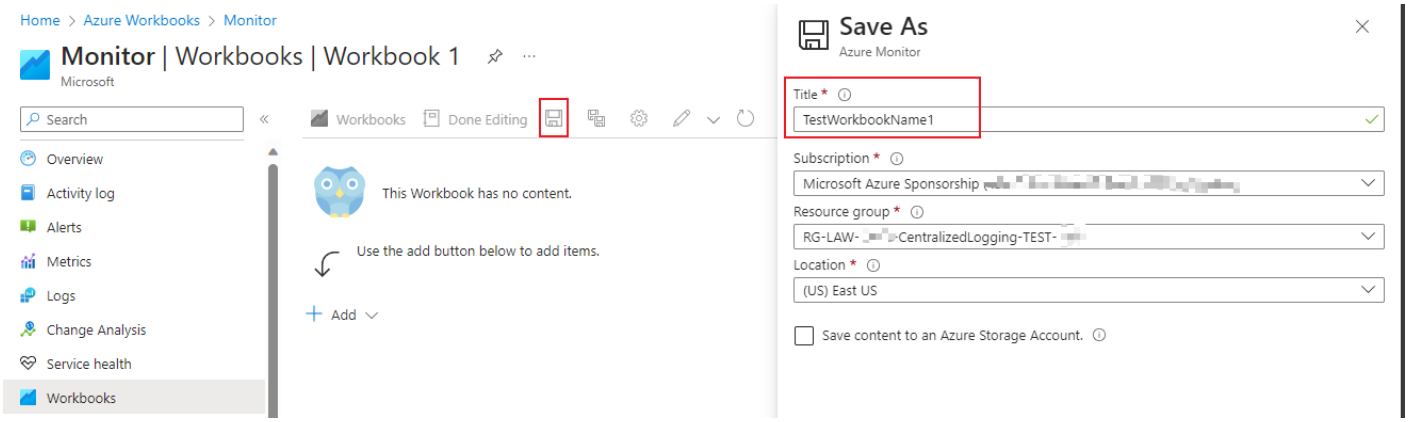
- **Log Analytics:** Viewers need at least *Log Analytics Reader* to run queries. Without these permissions the charts show no data.
- **Azure Resource Graph:** Results are trimmed to the subscriptions and resources the viewer can access. They may see less than you.
- **Microsoft Graph:** Your account may have tenant-wide Graph API rights, but viewers usually have scoped access (e.g. Intune Scope Tags or Entra ID Administrative Units. Grant the required permissions or offload it to an Azure Function with a Managed Identity for example.

Use of variables & dynamic content in Azure Workbooks

Within large Azure workbooks (dashboards), we want to be dynamic and allow users to specify their own parameters. This enables a more precise evaluation based on e.g. time period, resources and much more.

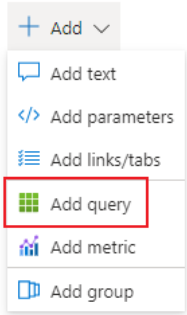
Create Azure Workbook

Search in Azure Portal for "Azure Workbooks" and create a new Workbook. It is recommended to save the workbook beforehand. For this step click on the save icon and set the name you want. Then choose the subscription and resource group parameters. in the end you have to specify a location.

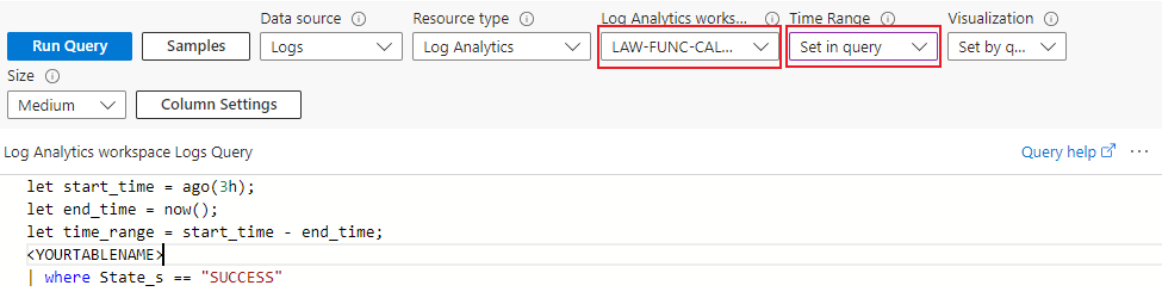


Create first data visualization

Inside the newly created we can start the first query. Click "Add" and choose "Add query".



Inside this query you have to specify the workspace as this is the source for the queries (your log analytics workspace or storage account of choice). Set the parameter "Time Range" to the value "Set in query" thus we will set this dynamically using Workbook parameters.

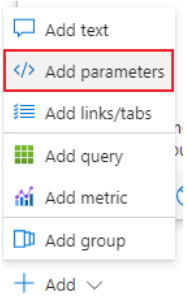


Write your log query in KQL so that the output meets your expectations. AI Chatbots can be a big help for writing queries.

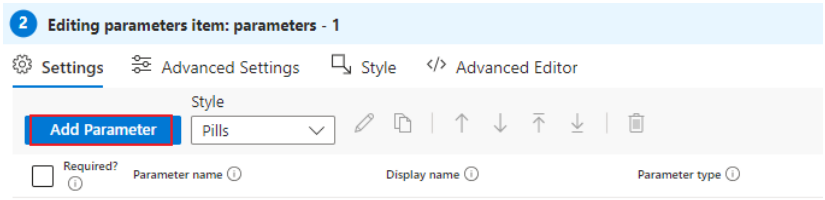
Add dynamic parameters

To display charts and tables based on the parameters we set, we can use the so-called "Parameter" functionality of Azure Workbooks.

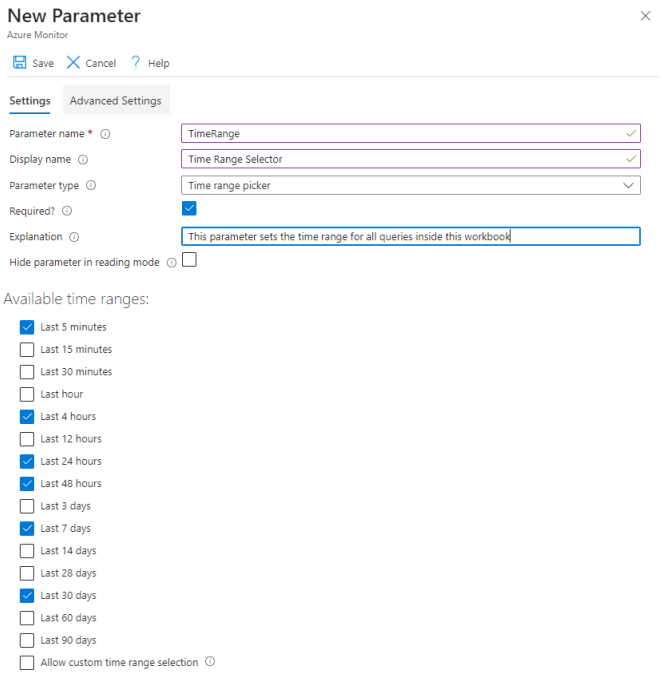
First we have to create a parameter set. Click "Add" and choose "Add parameters":



Click "Add Parameter":



Fill out the form based on your needs:



Use of dynamic parameters

After we have created a dynamic parameter, we can use the values inside our KQL queries. The use is different based on the selected data type of the parameter.

Using the "Previews" pane on the editing page of the parameter we can select values and see their according value.

Previews

When editing, your parameter will look like this:



When not editing, your parameter will look like this:



When replaced in a query or text item, {TimeRange} will become:

`{TimeRange}` ==> `Last 4 hours`

`{TimeRange:label}` ==> `Last 4 hours`

`{TimeRange:value}` ==> `> ago(4h)`

`{TimeRange:query}` ==> `> ago(4h)`

`{TimeRange:start}` ==> `12/19/2023 6:05 AM`

`{TimeRange:end}` ==> `12/19/2023 10:05 AM`

`{TimeRange:startISO}` ==> `2023-12-19T05:05:22.014Z`

`{TimeRange:endISO}` ==> `2023-12-19T09:05:22.014Z`

`{TimeRange:startUnix}` ==> `1702962322`

`{TimeRange:endUnix}` ==> `1702976722`

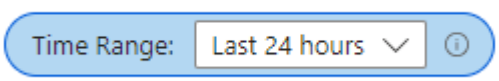
`{TimeRange:grain}` ==> `5m`

`{TimeRange:seconds}` ==> `14400`

So we can work with these variables inside our queries and update our query from before. This enables us to display the events based on the selected value of the parameter:

```
let start_time = {TimeRange:value};
let end_time = now();
let time_range = start_time - end_time;
<YOURTABLENAME>
| where State_s == "SUCCESS"
```

This will replace {TimeRange:value} with the value "ago(24h)" when the selection is the following:



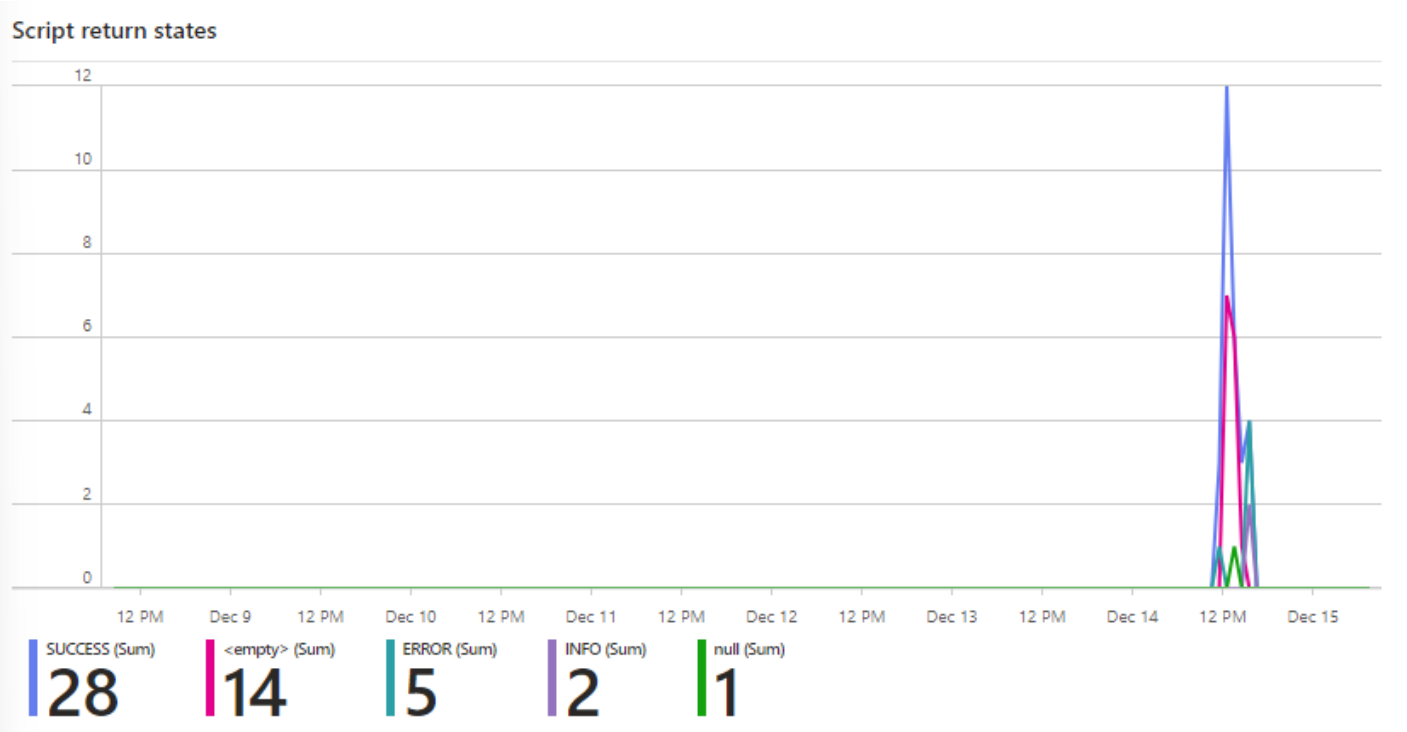
Create log health dashboard using time based data

This article is about how to create a dashboard using log data. We want to analyze relevant time-sensitive data in a dynamic and visually meaningful way.

Use case

An example application is when an application or script feeds data from various endpoints or microservices via a central logging solution and this data is then to be visualized. You can find more information about a simple and scalable solution for central log management here: [Centralize log collect... | LNC DOCS (lucanoahcaprez.ch)]

This example graph displays values of a PowerShell script that runs at the log in of a user. The script then has statuses, which are then written to an Azure Log Analytics Workspace. This diagram can now be used to see at what time which status was created.



Behind this view there is the following KQL query:

```
let start_time = {TimeRange:start};
let end_time = now();
let time_range = start_time - end_time;
<yourlogtable>
| where TimeGenerated >= start_time and TimeGenerated <= end_time
| make-series count() default=0 on TimeGenerated from start_time to end_time step 1h by
State_s
| render timechart
```

This query uses the value of the variable "{TimeRange:start}" for dynamic change based on the selection. More on this dynamic strategy is described here: Use of variables & dyn... | LNC DOCS (lucanoahcaprez.ch)