



P R O C E S S M A N U A L

Document Code: PM-MSI

Microsoft Sentinel Integration

Send Auditlog & Events data from your User Portal to your Microsoft Sentinel setup.

 **Admin** By Request

Table of Contents

Introduction	3
Assumptions	3
Prerequisites	3
Integration Tasks – Auditlog Data	5
Breakdown of Tasks.....	5
Task A: Set up Log Analytics Workspace	5
Task B: Create new Azure Logic App.....	8
Task C: Paste in JSON Code.....	10
Task D: Enter Parameters.....	11
Task E: Understand the App Flow	13
Task F: Configure Loop Entries	17
Task G: Test the Integration	20
Integration Tasks – Events Data	22
Breakdown of Tasks.....	22
Task A: Set up Azure Logic App	22
Task B: Enter Parameters.....	24
Task C: Understand the App Flow	24
Task D: Configure Loop Entries	29
Task E: Locate Workspace & Run App	31

Introduction

Microsoft Sentinel offers various ways to consume data from different sources. As Admin By Request provides public REST APIs for pulling Auditlog and Events data (see the documentation [here](#)), it's an easy task to leverage the power of Azure Logic Apps to consume the APIs and forward each new entry to an Azure Log Analytics Workspace for further Sentinel consumption.

We've created an Azure Logic App that requires very few changes before having you up and running with Admin By Request Auditlog and Events data in your Microsoft Sentinel setup. This manual provides a step-by-step guide on how to configure the integration.

Assumptions

The tasks described in this manual assume that the user has access to their Azure Portal, Admin By Request User Portal, and some familiarity with both environments.

Prerequisites


To enable this integration, you must first:

1. Obtain your Admin By Request API Key. This key can be self-generated through your Admin By Request User Portal via **Settings > [OS] Settings > Data > API:**

The screenshot shows the Admin By Request user portal interface. At the top, there's a navigation bar with 'Admin By Request' logo and links for Summary, Auditlog, Requests, Reports, Inventory, Settings (highlighted with a red arrow), Load, Logins, Docs, and Contact. Below this is a section for 'Windows Workstation Global Settings' with a sub-header 'API Access'. The 'API access' toggle is circled in red and set to 'ON'. To its right is a 'Regenerate' button. Below these is a text input field for the 'API Key' containing a long alphanumeric string, with a red arrow pointing to it. At the bottom of the form is a red 'Save' button. On the left sidebar, the 'Data' menu item is highlighted with a red arrow. On the right, there's an 'About API Access' section with explanatory text and a link to documentation.

- !** **IMPORTANT:** Click the **Save** button after Regenerating an API Key, to ensure this is the key used to establish the connection to Azure. A green tick icon will appear next to the **Save** button when the action is complete:



 **NOTE:** The API Key has been blurred out in the above example.

2. Have access to a Log Analytics Workspace to store your audit log entries for Sentinel.

Integration Tasks – Auditlog Data

Breakdown of Tasks

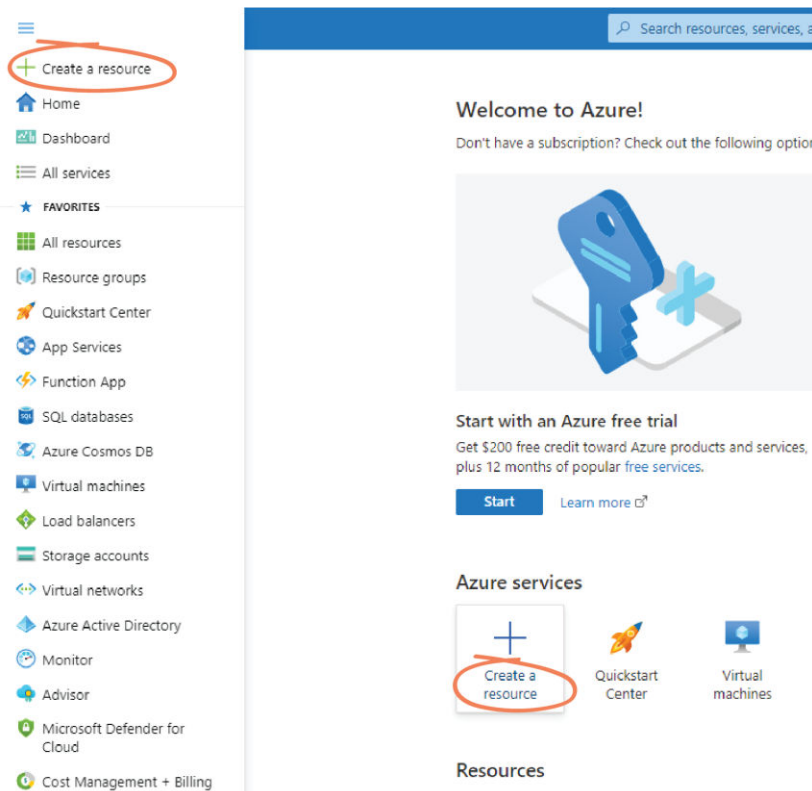
Seven tasks are covered in this section:

1. Task A: Set up Log Analytics Workspace
2. Task B: Create new Azure Logic App
3. Task C: Paste in JSON Code
4. Task D: Enter Parameters
5. Task E: Understand the App Flow
6. Task F: Configure Loop Entries
7. Task G: Test the Integration

Task A: Set up Log Analytics Workspace

A Log Analytics Workspace is the management unit which allows you to store, query, and retain data pulled in from other tools – in this case, Auditlog data pulled from your Admin By Request User Portal. Task A involves setting up this storage unit for use in subsequent tasks.

1. Log in to your Microsoft Azure Portal, and select **Create a resource** from the Home page or the side menu:



2. Use the search box to search for and select **Log Analytics Workspace** from the drop-down menu:

Home >

Create a resource ...

Get Started

Recently created

Categories

- AI + Machine Learning
- Analytics
- Blockchain
- Compute
- Containers

Log Analy

- Log Analytics Workspace
- Log Analyzer
- FortiAnalyzer Centralized Log Analytics
- Azure Log Analytics Agent Health
- HPE OneView for Azure Log Analytics (v1.4.0)

Azure Cosmos DB
Create | Docs | MS Learn

Getting Started?

3. Select **Create**:4. Fill out the Project details, and in the *Instance Details* section, give the Workspace a *Name* and select the appropriate *Region* from the drop-down menu:

Project details
Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.


Subscription * ⓘ Azure subscription 1

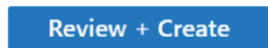
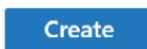
Resource group * ⓘ (New) Sentinel-Test
[Create new](#)

Instance details

Name * ⓘ SentinelLogs ✓

Region * ⓘ Australia Southeast

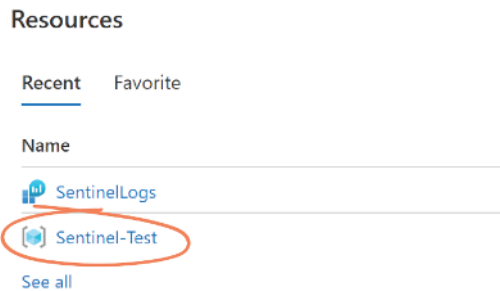
 **NOTE:** In the above screenshot, we have created a new Resource group called *Sentinel-Test* for the purpose of this demonstration.

5. Select the **Review + Create** button at the bottom of the page:6. When validation has passed, select **Create**, and wait for deployment to complete:

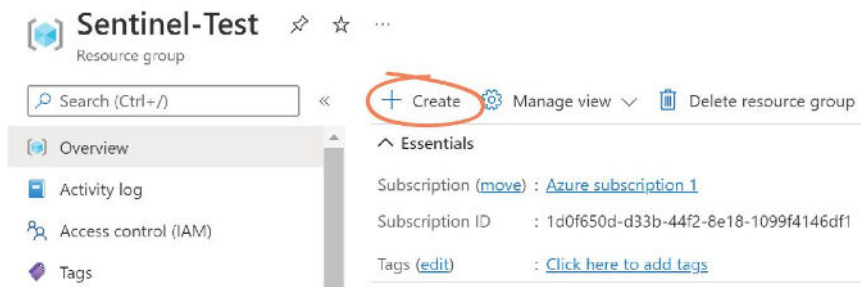
Task B: Create new Azure Logic App

An Azure Logic App is needed to consume the Admin By Request Auditlog API and forward each new entry to the Azure Log Analytics Workspace created in Task A.

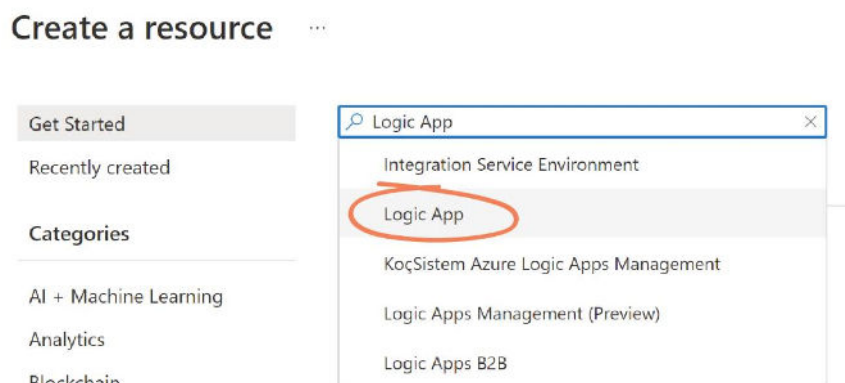
1. Navigate to *Resource groups* and select the Resource Group used in Task A from the *Recent* list under *Resources* – in this example, **Sentinel-Test**:



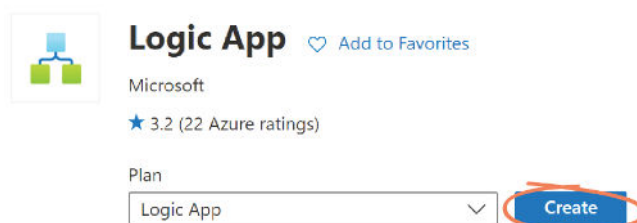
2. Once in Sentinel-Test, select the **Create** button:



3. Use the Search bar to locate and select **Logic App** from the drop-down menu:



4. Click **Create**:



- In the *Plan* section, select your *Plan type*. In this example, we use **Consumption**:

Plan

The plan type you choose dictates how your app scales, what features are enabled, and how it is priced. [Learn more](#)

Plan type *

Standard: Best for enterprise-level, serverless applications, with event-based scaling and networking isolation.

Consumption: Best for entry-level. Pay only as much as your workflow runs.

[Looking for the classic consumption create experience? Click here](#)

- In the *Instance Details* section, enter a *Logic App name* (in this case, **Sentinel-Logic-App**) and select the appropriate *Region*:


Instance Details


Logic App name *

Region *

Enable log analytics * Yes No

- Select the **Review + Create** button, followed by **Create**.
- Once deployment is complete, click **Go to resource**:

 Your deployment is complete

 Deployment name: Microsoft.Web-LogicAppConsumption-Portal-2...
Subscription: [Azure subscription 1](#)
Resource group: [Sentinel-Test](#)

∨ **Deployment details** [\(Download\)](#)

∧ **Next steps**

[Setup log analytics for your app.](#) Recommended

[Go to resource](#)

Task C: Paste in JSON Code

To get the app behaving how we need it to for this integration, you need to replace the default code in the Logic app code view with the JSON code we have written – access it [here](#).

1. In the *Logic Apps Designer* page, select the app you created in Task B from the top menu (in this case, **Sentinel-Logic-App**):

Home > Microsoft.Web-LogicAppConsumption-Portal-2207b2d0-b993 > Sentinel-Logic-App >
Logic Apps Designer ...

2. From the left-hand menu, under *Development Tools*, select **Logic app code view**:

Development Tools

Logic app designer

</> Logic app code view

Versions

API connections

Quick start guides

3. Open the Admin By Request JSON code (found [here](#)) and select and copy all. Navigate back to the *Logic app code view* in Azure, and replace the existing code with the code copied from the JSON file:

```

511     },
512     "contentVersion": "1.0.0.0",
513     "outputs": {},
514     "parameters": {
515       "apiKey": {
516         "defaultValue": "xxxxxx",
517         "type": "String"
518       },
519       "LogName": {
520         "defaultValue": "AdminByRequestLogs",
521         "type": "String"
522       }
523     },
524     "triggers": {
525       "Recurrence": {
526         "evaluatedRecurrence": {
527           "frequency": "Day",
528           "interval": 1,
529           "startTime": "2022-06-22T15:00:00Z"
530         },
531         "recurrence": {
532           "frequency": "Day",
533           "interval": 1,
534           "startTime": "2022-06-22T15:00:00Z"
535         },
536         "type": "Recurrence"
537       }
538     },
539     "parameters": {}
540   }
541 }

```

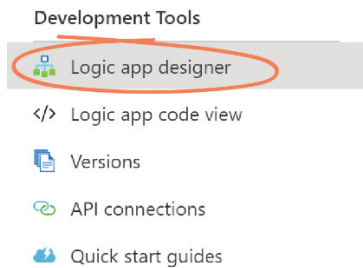
4. Click **Save**:

Save X Discard

Task D: Enter Parameters

The Admin By Request API Key (found in the [Prerequisite](#) section of this document) is used to establish a connection between the Azure Logic App and your Admin By Request User Portal.

- From the left-hand menu, under *Development Tools*, select **Logic app designer**:



- From the top menu, select **Parameters**:



- The two parameters required for the integration are:

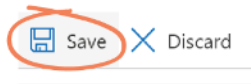
- ApiKey - String*: The API Key obtained from your Admin By Request User Portal (see [Prerequisite](#)).
- LogName - String*: The name you would like for the custom log in your Log Analytics Workspace.

In the *Default Value* field for each of these parameters, replace the placeholder text with the appropriate / desired value:

Two screenshots of the parameter configuration form in the Logic App Designer. The first screenshot shows the 'ApiKey' parameter with a blurred default value. The second screenshot shows the 'LogName' parameter with the default value 'AdminByRequestLogs'.

NOTE: In the above screenshot, the API Key is blurred out, and we have used *AdminByRequestLogs* as the *LogName*.

4. Select the **Save** button in the Logic app designer:

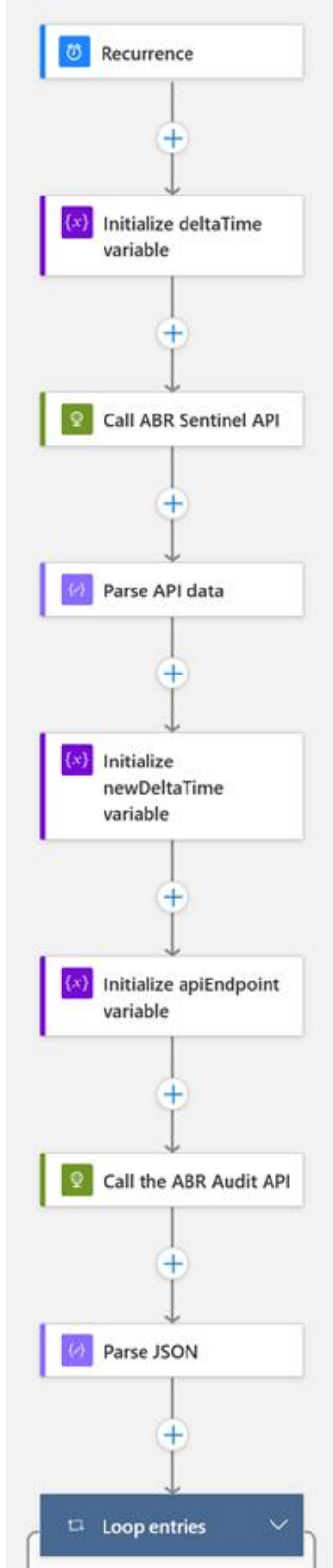


5. Close the Parameter window by selecting the **X** in the top right-hand corner.

Task E: Understand the App Flow

In this Task, we take a look at what's going on 'behind the scenes' - at API calls, variables, and loops involved in the preconfigured app flow.

The app flow has nine segments arranged as follows:



- **Recurrence** – This tells the app when it should run. In our example we've set up a recurring trigger that runs once every day. You can replace this trigger with whatever works best for your setup.

The screenshot shows the 'Recurrence' configuration panel. It includes a title bar with a refresh icon and a three-dot menu. Below are three main sections:

- * Interval**: A text input field containing the value '1'.
- * Frequency**: A dropdown menu currently set to 'Day'.
- Start time**: A date-time input field containing '2022-06-22T15:00:00Z'.

 At the bottom, there is a text input field labeled 'Add new parameter' with a dropdown arrow.

- **Initialize deltaTime variable** – In order to call the Admin By Request Audit API, we need a variable containing the 'from' ticks. Basically, telling the Audit API to 'give me all audit logs since this time'. This is defaulted to the number of ticks representing DateTime.Now.

The screenshot shows the 'Initialize deltaTime variable' configuration panel. It includes a title bar with a variable icon and a three-dot menu. Below are three main sections:

- * Name**: A text input field containing 'deltaTime'.
- * Type**: A dropdown menu currently set to 'Integer'.
- Value**: A text input field containing a function call 'ticks(...)' with a small 'x' icon to its right.

 At the bottom, there is a text input field labeled 'Add new parameter' with a dropdown arrow.

- **Call ABR Sentinel API** – Since Logic Apps don't hold any state, we need some way of storing the last time the Audit API was called for a given API-key. We've created an API endpoint that allows you to do just this. We simply call the SetDeltaTime endpoint with your API Key and the deltaTime variable, and the API returns that value for when the Audit endpoint was last called – and it stores the new value, so that the next time the Logic App runs, it has the correct tick-values to ensure that you don't get any duplicate entries.


The screenshot shows the 'Call ABR Sentinel API' configuration panel. It includes a title bar with a globe icon and a three-dot menu. Below are several sections:

- * Method**: A dropdown menu set to 'POST'.
- * URI**: A text input field containing 'https://sentinel.adminbyrequest.com/Audit/SetDeltaTime'.
- Headers**: A table with two columns: 'Enter key' and 'Enter value'.
- Queries**: A table with two columns: 'Enter key' and 'Enter value'.
- Body**: A text area containing a JSON object:


```
{
  "ApiKey": "@ ApiKey x",
  "Ticks": "{x} deltaTime x"
}
```
- Cookie**: A text input field labeled 'Enter HTTP cookie'.

 At the bottom, there is a text input field labeled 'Add new parameter' with a dropdown arrow.

- **Parse API Data** – The result from the API needs to be parsed in order to use the resulting variables.

>>  Parse API data ...

Parameters Settings Code View Run After ...

*Content


*Schema

```

{
  "properties": {
    "apiEndpoint": {
      "type": "string"
    },
    "success": {
      "type": "boolean"
    },
    "ticks": {
      "type": "integer"
    }
  }
}
    
```

[Use sample payload to generate schema](#)

- **Initialize newDeltaTime and apiEndpoint variable** – With the values from the SetDeltaTime endpoint, we need to store two variables: newDeltaTime and apiEndpoint. These variables hold the tick-value for when the Audit API was last called, as well as the Admin By Request endpoint to call for Audit logs.


>>  Initialize newDeltaTime variable ...

Parameters Settings Code View Run After ...

*Name

*Type

Value

>>  Initialize apiEndpoint variable ...


Parameters Settings Code View Run After ...

*Name

*Type

Value

- **Call the ABR Audit API** – Now it's a matter of calling the Admin By Request Audit endpoint with your API Key, as well as the newDeltaTime variable.


>>  Call the ABR Audit API ...

Parameters Settings Code View Run After ...

* Method

* URI

Headers

apikey	 ApiKey x	<input type="text"/>
Enter key	Enter value	<input type="text"/>

Queries

Enter key	Enter value	<input type="text"/>
-----------	-------------	----------------------

Body

Cookie

- **Parse JSON** – The next step parses the response from the Audit API as JSON using a schema based on the response type from the Audit API (view the [Auditlog API documentation](#) for more information on this).

 Parse JSON i ...

* Content

* Schema

```

{
  "properties": {
    "entries": {
      "items": {
        "properties": {
          "application": {
            "properties": {
              "file": {
                "type": [

```

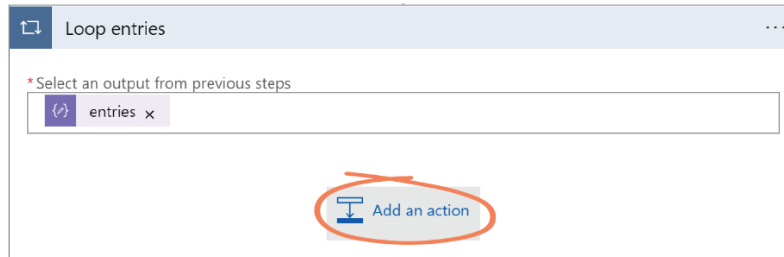
[Use sample payload to generate schema](#)

- **Loop entries** – The final step in the app flow simply loops through every entry from the Audit call. Here you decide what to do with the data. (See Task F, below.)

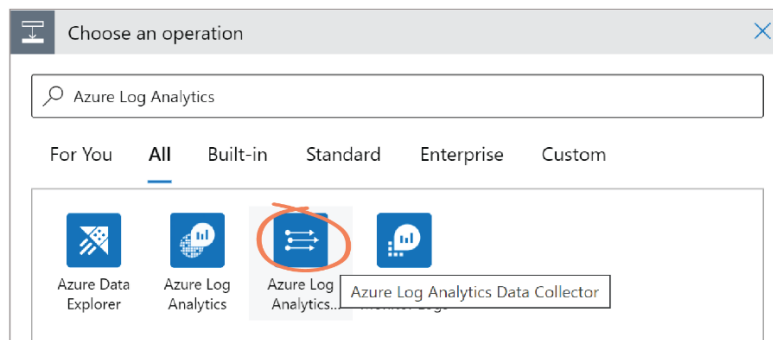
Task F: Configure Loop Entries

In order to send data to your Azure Log Analytics Workspace, you must add an action for each entry in the dataset.

1. Select the *Loop entries* segment of the app flow and click the **Add an Action** button:

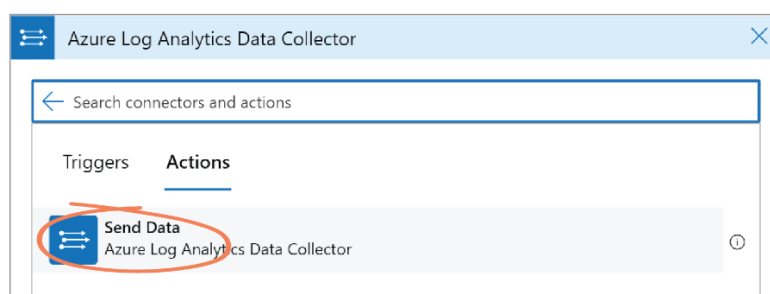


2. In the *Choose an operation* section, use the search box to locate and select **Azure Log Analytics Data Collector**:



IMPORTANT: Be sure to select **Azure Log Analytics Data Collector**, rather than *Azure Log Analytics* (to the left in the screenshot above).

3. Select **Send Data** under the *Actions* tab:



IMPORTANT: After clicking Send Data, you may be prompted to create a connection. If so, follow the steps below (steps 4-7). Otherwise, skip to step 8.

4. If prompted to create a connection, in the *Connection name* field, choose your desired name – in this case, we've used *AzureLogConnector*:

Azure Log Analytics Data Collector

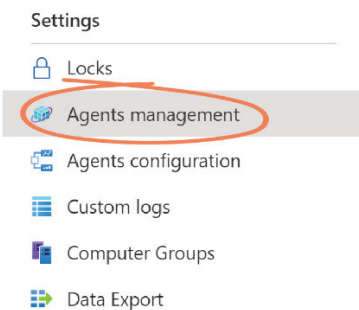
* Connection name: AzureLogConnector

* Workspace ID ⓘ: The unique identifier of the Azure Log Analytics workspace.

* Workspace Key ⓘ: The primary or secondary key of the Azure Log Analytics workspace.

Create

5. To locate the *Workspace ID* and *Workspace Key*, open your Log Analytics Workspace (i.e., *SentinelLog*) in a new tab and select **Agents Management** under *Settings* from the left-hand menu:



6. Copy the *Workspace ID* and *Primary key* values from this page:

Workspace ID ←

Primary key ←

Regenerate

7. Navigate back to the Logic App, paste the keys into their corresponding fields in the *Azure Log Analytics Data Collector* window, and select **Create**:

Azure Log Analytics Data Collector

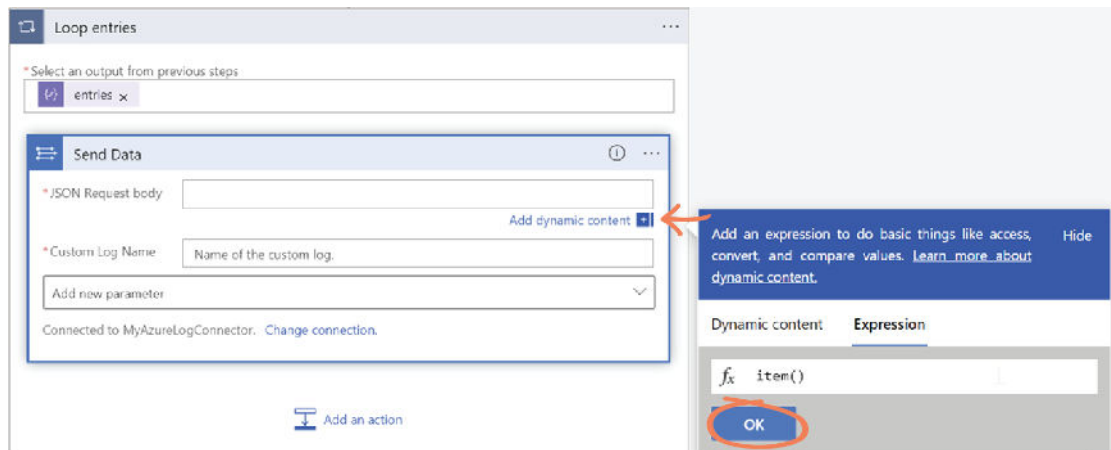
* Connection name: AzureLogConnector

* Workspace ID ⓘ: [Copied Workspace ID]

* Workspace Key ⓘ: [Copied Primary Key]

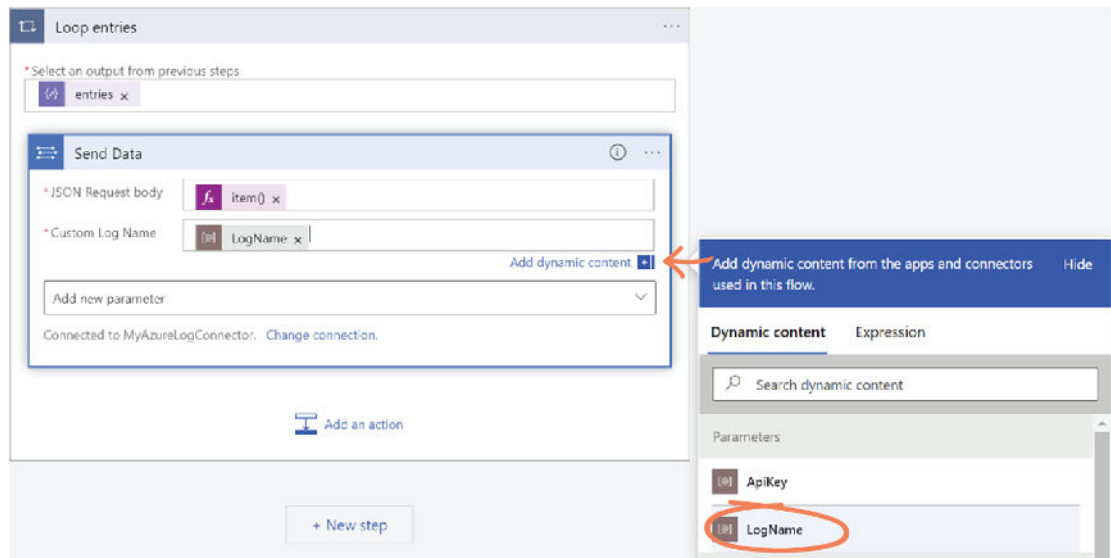
Create

8. In the *JSON Request body* field, select **Add dynamic content**, and in the *Expression* tab, enter *item()* and click **OK**:



NOTE: This selects the current item in the JSON loop and adds it as the request body.

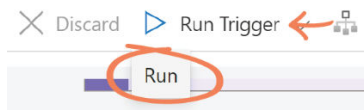
9. In the *Custom Log Name* field, select **Add dynamic content**, and in the *Dynamic content* tab, locate and select the **LogName** parameter:



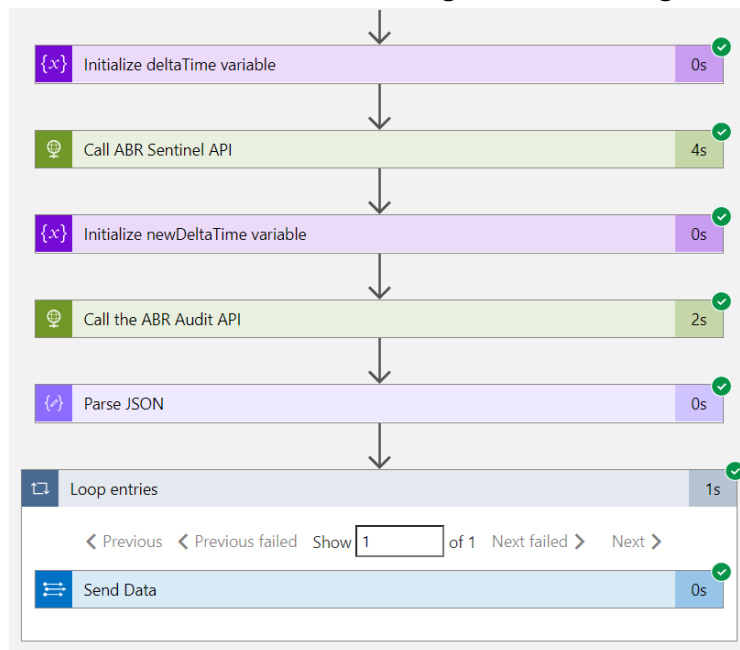
10. Select the **Save** button to save your app.

Task G: Test the Integration

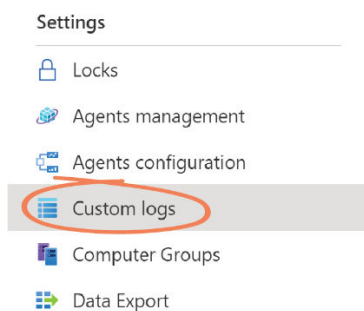
1. Select **Run Trigger > Run** from the top menu:



- NOTE:** You may need to wait a few minutes for the flow to complete. When successful, it should look something like the following:



2. Navigate to your Log Analytics Workspace (i.e., *SentinelLog*), and select **Custom logs** under *Settings* from the right-hand menu:

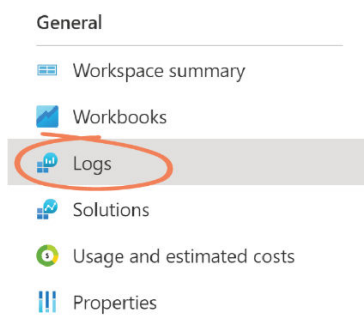


3. Highlight and copy the *Name* of the Custom log listed – in this case, *AdminByRequestLogs_CL*:

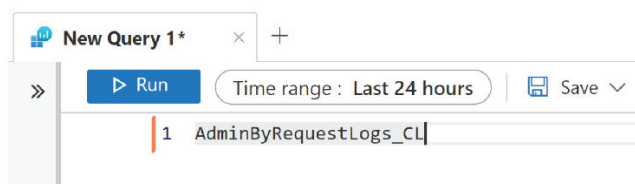
Showing 1 results

Name ↑↓	Type ↑↓
AdminByRequestLogs_CL	Ingestion API

- Select **Logs** under *General* from the left-hand menu:



- Close the *Queries* window that pops up. Paste the copied Custom log in the *New Query* field and select **Run**:




- New entries will begin to display in your Log Analytics Workspace as they are pushed through. Click the drop-down arrow to display details for each entry:

Query editor

Results Chart

TimeGenerated [UTC]	installs_s	uninstalls_s	elevatedApplications_s	scanResults_s
6/29/2022, 4:18:43.318 AM	[]	[]	[{"name": "Windows Command Processor", "path": "C:\\..."}]	[]
TenantId	53731173-cde7-49a8-8802-cc04b4099e90			
SourceSystem	RestAPI			
TimeGenerated [UTC]	2022-06-29T04:18:43.318Z			
installs_s	[]			
uninstalls_s	[]			
	[{ "name": "Windows Command Processor", "path": "C:\\Windows\\System32", "file": "cmd.exe", "version": "10.0.19041.1 (WinBuild.160101.0800)", "vendor": "Microsoft Corporation", "sha256": "B99D61D874728EDC0918CA0EB10EAB93D381E7367E377406E65963366C874450", "scanResult": "Clean", "scanResultCode": 0,			

 **NOTE:** It may take several minutes for log entries to show up in the Log Analytics Workflow.

- Click **Save** to save the query for later use.
- With the Azure Log Analytics Workspace set up, you can now point your Sentinel setup to use this workspace as a data source.

Integration Tasks – Events Data

Breakdown of Tasks

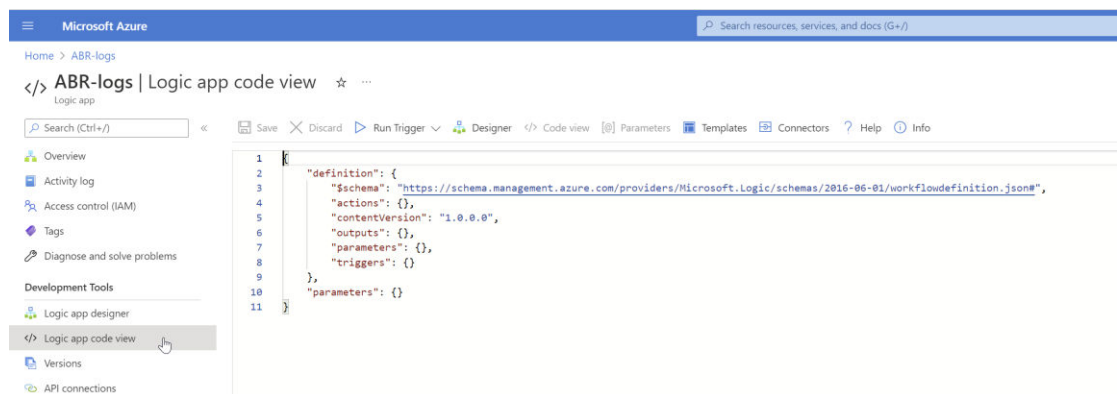
Seven tasks are covered in this section:

1. Task A: Set up Azure Logic App
2. Task B: Enter Parameters
3. Task C: Understand the App Flow
4. Task D: Configure Loop Entries
5. Task E: Locate Workspace & Run App

Task A: Set up Azure Logic App

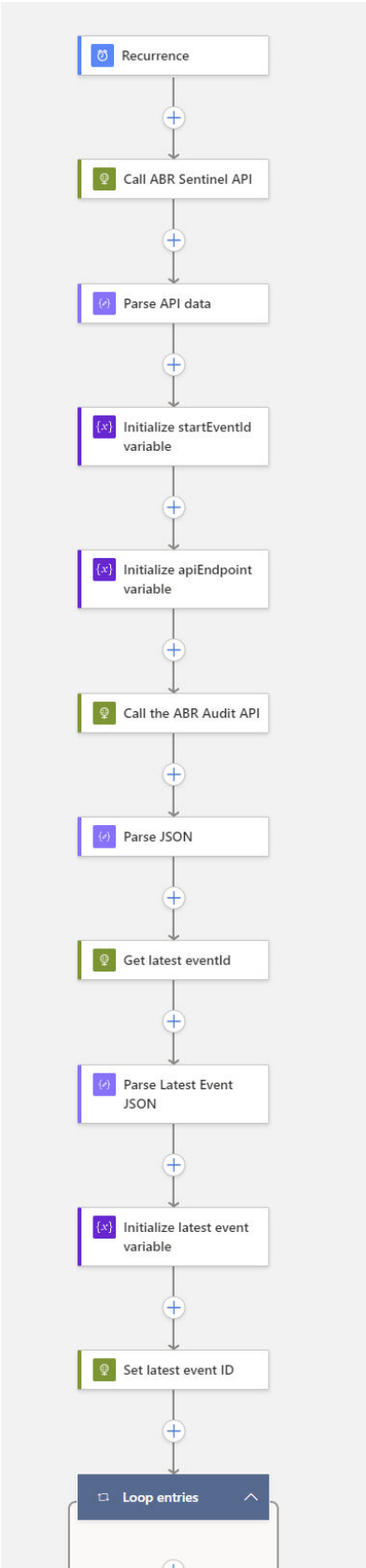
1. In order to get started, first set up an Azure Logic App within Azure Portal. You can choose to either setup the app from scratch – or you can choose to use the Admin By Request template. If you choose to use the Admin By Request template, jump into the Code View of your Logic App and paste in the following application code:

[Obtain Code](#)



```
1
2
3  "definition": {
4    "schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-06-01/workflowdefinition.json#",
5    "actions": {},
6    "contentVersion": "1.0.0.0",
7    "outputs": {},
8    "parameters": {},
9    "triggers": {}
10  },
11  "parameters": {}
```

2. Now that we have the structure for the Logic App, navigate to the Logic App Designer to view the different steps:



Task B: Enter Parameters

- The app has set up two Parameters. Replace these with your own API-key and the name you'd like for the custom log in your Log Analytics Workspace:

✕

Name *

Type *

Default Value

Actual Value

✕

Name *

Type *

Default Value

Actual Value

Task C: Understand the App Flow

- The flow of the app is simple – let's go through the different bits:

- Recurrence** tells the app when it should run. In our example we've set up a recurring trigger that runs once every day. You can replace this trigger with whatever works best for your setup:

🕒 Recurrence
⋮

* Interval

* Frequency

Start time


- In order to call the Admin By Request Events API, we'll need to fetch the EventID to serve as the starting point for the query. Furthermore, we need to determine which API endpoint to use. This is done by calling the Admin By Request Sentinel API and storing this information:


>>  Call ABR Sentinel API ...

Parameters Settings Code View Testing ...

* Method


* URI

Headers 

Queries 

Body

Cookie



>>  Parse API data ...

Parameters Settings Code View Run After ...

* Content

* Schema

```
{
  "properties": {
    "eventId": {
      "type": "integer"
    },
    "publicApiUrl": {
      "type": "string"
    },
    "success": {
      "type": "boolean"
    }
  }
}
```

Use sample payload to generate schema

>>  Initialize startEventId variable ...

Parameters Settings Code View Run After ...

* Name

* Type

Value

>>  Initialize apiEndpoint variable ...

Parameters	Settings	Code View	Run After	...
* Name	<input type="text" value="apiEndpoint"/>			
* Type	<input type="text" value="String"/>			
Value	<input type="text" value="{x} publicApiUrl x"/>			

- Once we have the startEventId and the apiEndpoint parameters in place, we can use these to call the Admin By Request Events API to fetch the events that have been created since the query last ran:

>>  Call the ABR Audit API ...

Parameters	Settings	Code View	Run After	...
* Method	<input type="text" value="GET"/>			
* URI	<input type="text" value="{x} apiEndpoint x /events?startId={x} startEventId x &take=10000"/>			
Headers	apikey	<input type="text" value="@ ApiKey x"/>	✕ 📄	
	Enter key	Enter value		
Queries	Enter key	Enter value	📄	
	Enter key	Enter value		
Body	<input type="text" value="Enter request content"/>			
Cookie	<input type="text" value="Enter HTTP cookie"/>			
<input type="text" value="Add new parameter"/>				

- We can then parse the result of this API call to be used when looping through the data later on.

>>  Parse JSON ...

Parameters Settings Code View Run After ...


* Content  Body x

* Schema

```
{  
  "items": {  
    "properties": {  
      "additionalData": {},  
      "alertAccount": {},  
      "application": {  
        "properties": {  
          "file": {},  
          "name": {},  
          "path": {}  
        }  
      }  
    }  
  }  
}
```


[Use sample payload to generate schema](#)

- Because Logic Apps can't hold state, we'll need to save the latest EventID to the Admin By Request Sentinel API in order to have the starting point for the next time the Logic App runs.



>>  Get latest eventId ...

Parameters Settings Code View Run After ...


* Method GET v

* URI  apiEndpoint x /events?last=1

Headers

apikey	 ApiKey x	X 
Enter key	Enter value	

Queries

Enter key	Enter value	
-----------	-------------	---

Body

Enter request content

Cookie

Enter HTTP cookie

Add new parameter v

>>  Parse Latest Event JSON

Parameters Settings Code View Run After ...


* Content  Body x

```
{
  "items": {
    "properties": {
      "additionalData": {},
      "alertAccount": {},
      "application": {
        "properties": {
          "file": {},
          "name": {},
          "start": {}
        }
      }
    }
  }
}
```

Use sample payload to generate schema



>>  Initialize latest event variable

Parameters Settings Code View Run After ...

* Name latestEventId
* Type Integer
Value  body(...) x

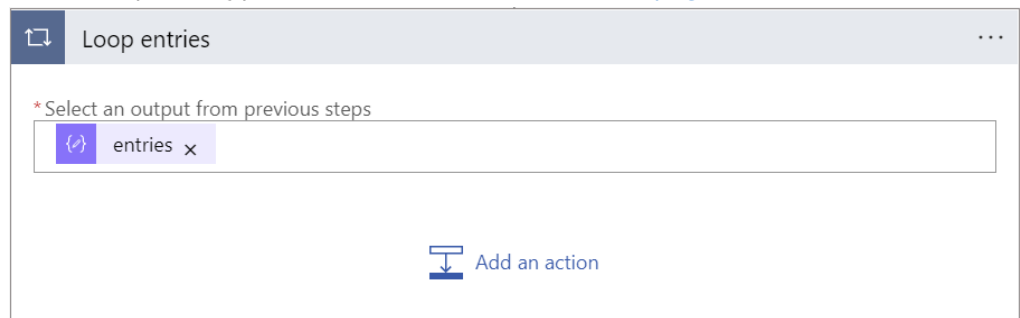
>>  Set latest event ID

Parameters Settings Code View Run After ...

* Method POST
* URI https://sentinel.adminbyrequest.com/Events/SetEventStartId
Headers Enter key Enter value
Queries Enter key Enter value
Body {
 "ApiKey":  ApiKey x ,
 "EventStartId":  latestEventId x
}
Cookie Enter HTTP cookie

Add new parameter

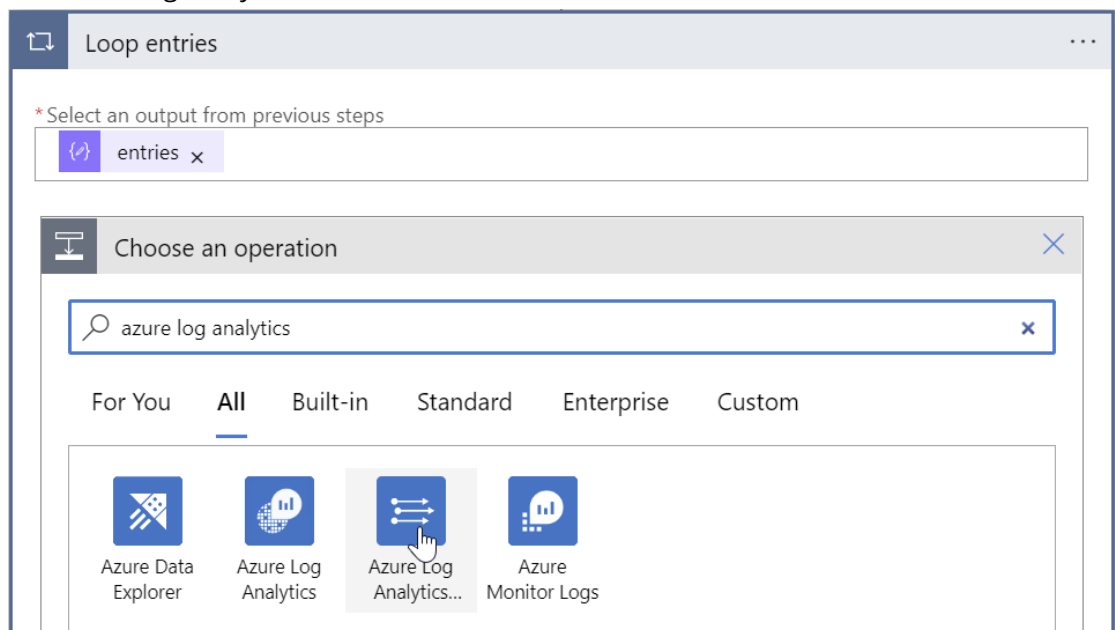
- Next up we parse the response from the Event API as JSON using a schema based on the response type from the Events API (view [this page](#) for more information).

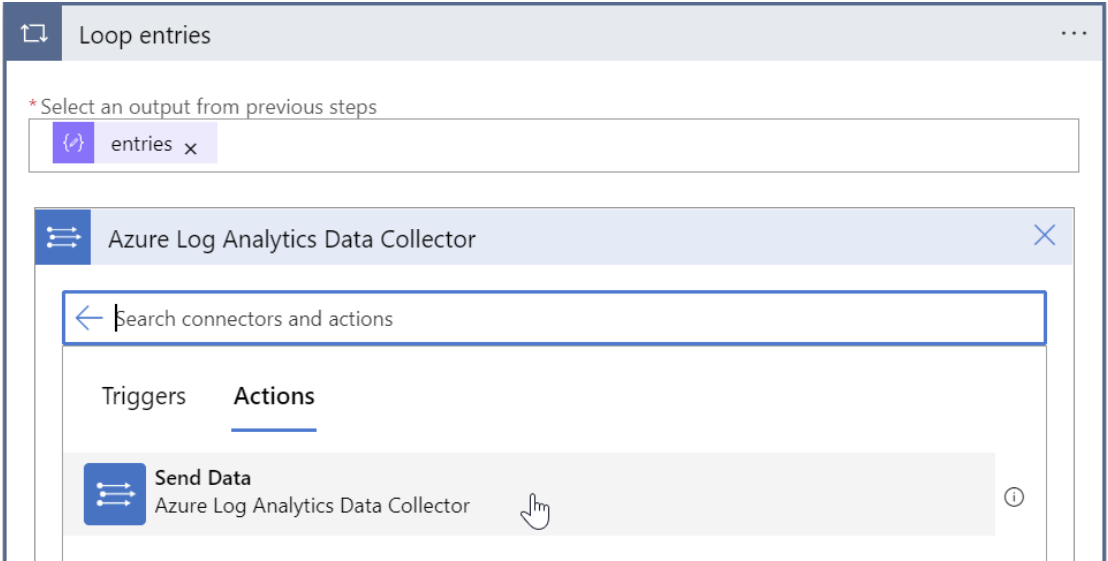


- The last step simply loops through every entry from the Events call. Here you decide what to do with the data.

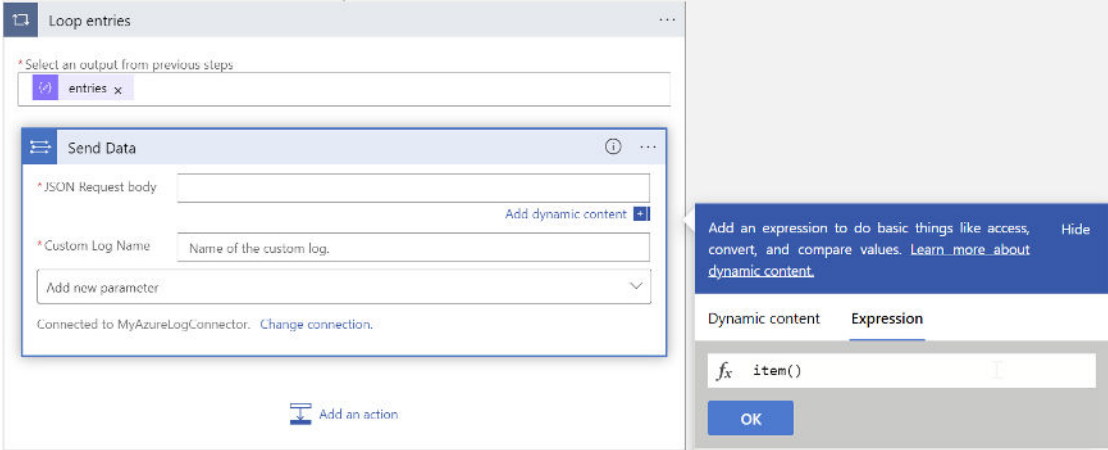
Task D: Configure Loop Entries

1. In order to send these to your Azure Log Analytics Workspace, do the following. Click on "Add an action" within the Loop entries action and search for "Azure Log Analytics". Click the Azure Log Analytics result.

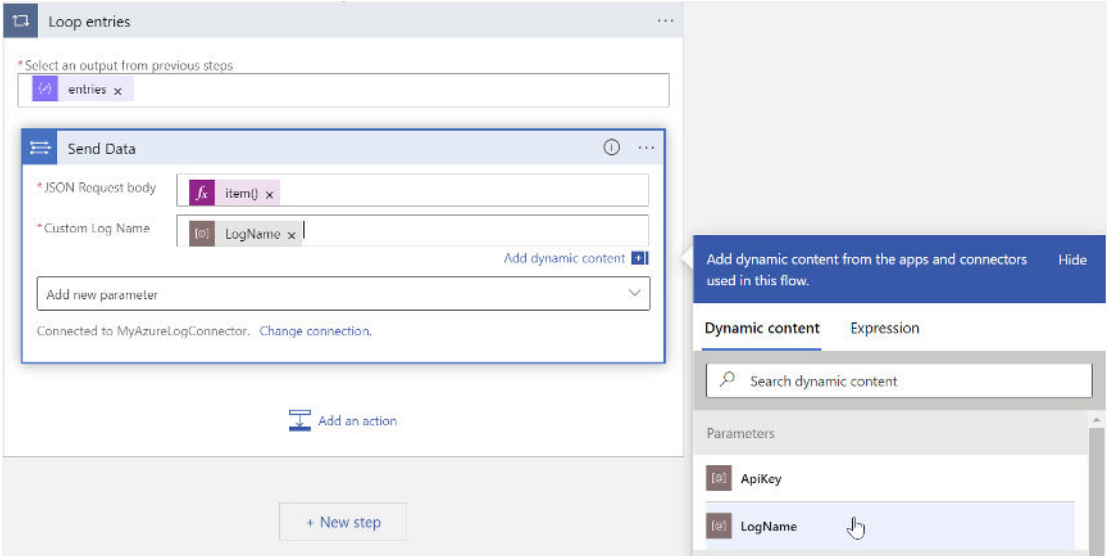




2. Select the 'Send Data' action



3. In the "JSON Request body", select "Expression" in the dynamic content selector – and enter a value of "item()". This will select the current item in the JSON-loop and add it as the request body.



- For the Custom Log Name, select the “LogName” parameter. This will tell the Logic App to send the audit entries to a custom log with the name supplied in the parameter.
- The above assumes that you already have an ApiConnection for your Azure Log Analytics Workspace set up. If this is not the case, you’ll be met with the option to create a new connection:

- Simply fill out the three fields and click “Create”.

Task E: Locate Workspace & Run App

- To find the Workspace ID and the Workspace Key, navigate to the Log Analytics workspace and select “Agents management”:

- Copy the workspace ID and Primary Key and paste these into the connection properties in your Logic App.
- Now, save your app – and you’re done. If you run the app now, you’ll see your new entries showing up in your Log Analytics Workspace:

New Query 1* +

SentinelLogs Select scope Run Time range: Last 24 hours Save Share + New alert rule Export Pin to Format query

Tables Queries Functions

Search Filter Group by: Solution Collapse all

Favorites
You can add favorites by clicking on the star icon

- LogManagement
 - Usage
- Custom Logs

Results Chart

TimeGenerated (UTC)	startTime_s (UTC)	startTimeUTC_1 (UTC)	endTime_1 (UTC)	endTimeUTC_1 (UTC)	installs_s	uninstalls_s	elevatedApplications_s
6/27/2022, 8:47:46.576 A...					0	0	0
TenantId	52616b7e-d679-4d1a-908f-e5e39f7e2f14						
SourceSystem	RestAPI						
TimeGenerated (UTC)	2022-06-27T08:47:46.576Z						
installs_s	0						
uninstalls_s	0						