

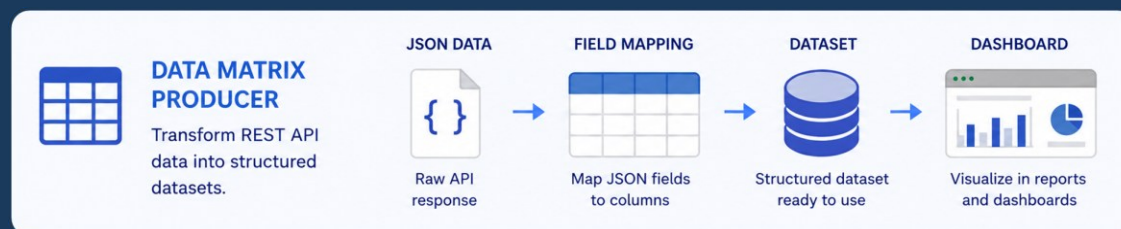
CEEVIEW

REST Monitor Learning Series

Tutorial 4

Creating a Structured Data Matrix Producer

Transforming REST API Responses into Reusable Operational Datasets



Version 2.19 · DataIntegration Package

This tutorial is Tutorial 4 in the REST Monitor Learning Series and assumes the Health, Metric, and Service Producer tutorial have been completed.

In this tutorial, you'll extend the Nordic Retail REST Monitor by adding a Data Matrix Producer that transforms REST API responses into reusable datasets for dashboards, reporting, and analysis.

1. Starting Point from Tutorials 1–3

Before beginning this tutorial, your REST Monitor should match the completed configuration from the previous 3 tutorials:

Tutorial 1 — Creating a Multi-Asset Health Producer

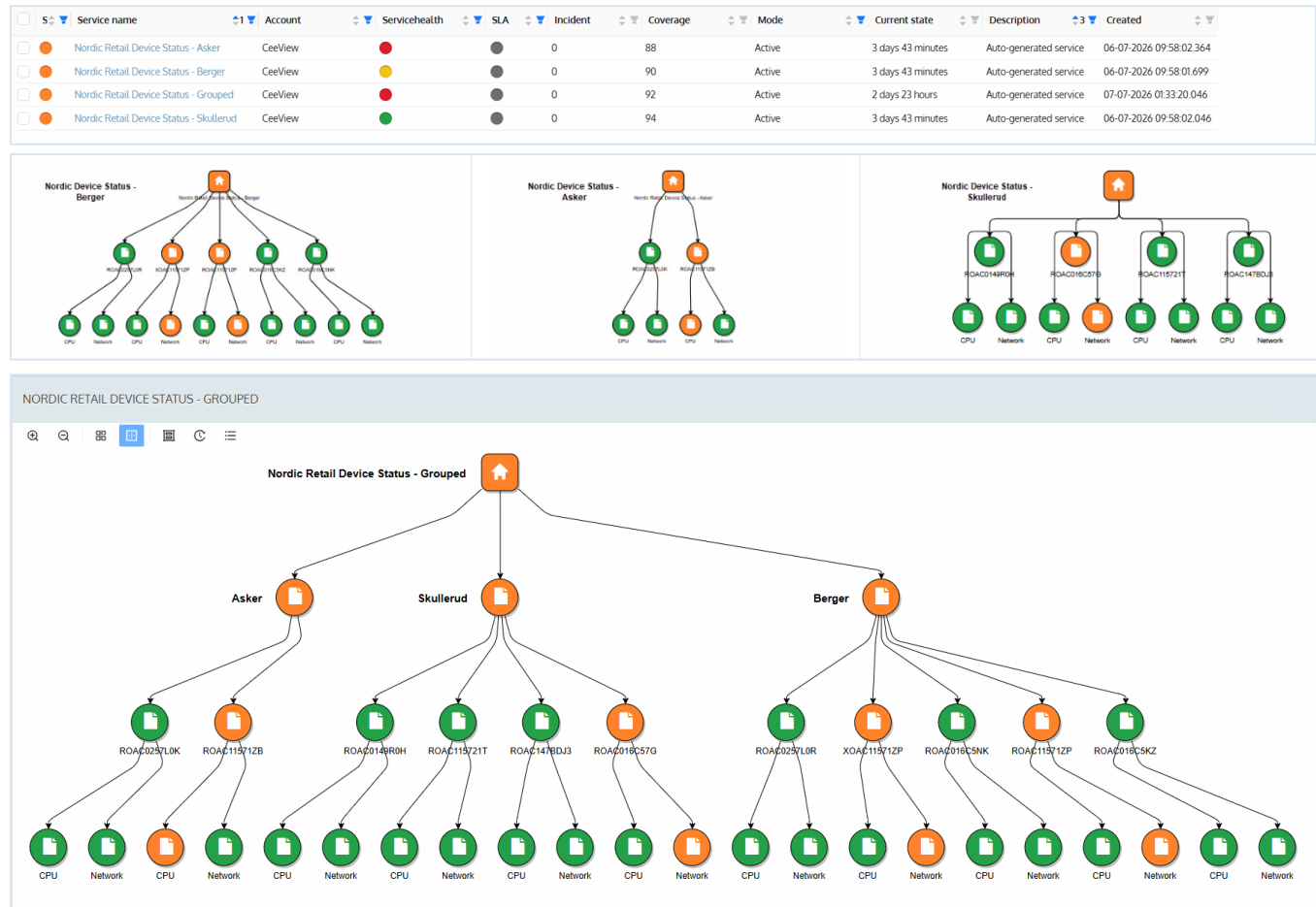
Tutorial 2 — Creating a Time-Series Metric Producer

Tutorial 3 — Creating Operational Service Models with Service Producers

You should already have:

- ✓ REST Monitor: **Nordic Retail Example**
- ✓ Health Producer: **1-CpuUsage-Health**
- ✓ Metric Producer: **2-MetricMemUsed**
- ✓ Transformer: **cpuSeverity**
- ✓ Editor function: **stripWorkgroup()**
- ✓ Asset Group: **Nordic Retail Example** with eleven Assets
- ✓ Memory Usage Metrics associated with the same Health Assets
- ✓ Service Producers: **3-Svc-Group-CPU, 4-Svc-Group-Network**
- ✓ Service Producers: **5-Service-CPU, 6-Service-Network**
- ✓ Service Models: **Asker, Berger, Skullerud, & Nordic Retail Grouped**

The screenshots below confirm the completed environment from Tutorials 1–3. Health Assets and Metrics continue to be generated, while the Service Producers have created four operational Service Models—three location models and one unified Nordic Retail business service.



What You'll Add: This tutorial adds a **Data Matrix Producer** to the existing Nordic Retail Example REST Monitor. When complete, the Health Producer will continue creating Health Assets, the Metric Producer will continue recording time-series Metrics, the Service Producers will continue maintaining Service Models, and the new Data Matrix Producer will transform the same REST API response into reusable operational datasets for dashboards, reports, and analysis.

2. Making Hidden API Data Usable

Every REST API contains far more operational information than can be visualized directly. Those APIs return data in JSON format—ideal for applications, but difficult to search, report on, or reuse.

The Data Matrix Producer reveals this hidden value by organizing JSON response into structure datasets—turning raw API data into information that can be analyzed, compared and shared throughout Ceeview.

What Kinds of API Data Become Valuable When Structured?

API across technology and business systems expose many types of information. When structured into tables, this data becomes clear, searchable, and operationally useful.

 Asset & Inventory Data Discover and manage the resources you own. • Device / Asset Name • Location • Hardware Model • Serial Number • OS / Firmware • Asset Status ➔ Inventory reports, lifecycle tracking, compliance	 Performance & Health Data Understand how resources are performing. • CPU / Memory / Disk • Network Throughput • Latency / Response Time • Utilization • Availability / Uptime • Error Counts ➔ Capacity reports, performance trends, threshold analysis	 Cloud & Cost Data Gain visibility into cloud spend and usage. • Resource / VM • Cloud Account / Region • Owner / Business Unit • Instance Type • Monthly Cost • Status / Tags ➔ Cost analysis, chargeback/showback, optimization	 Business & Operational Data Bring business context into operational visibility. • Store / Location • Department / Team • Sales / Transactions • Customer / Account • SLA / Contract Details • Dates / Status ➔ Business dashboards, operational reports, trend analysis
 APIs already hold the information—you just need to structure it. From infrastructure to cloud to business systems, the potential for reusable operational datasets is enormous. The Data Matrix Producer unlocks that potential.			

With Data Matrices you can:

 Build dashboards Create reliable data sources for live dashboards and KPIs.	 Generate reports Export structured data for scheduled, on-demand, and ad-hoc reports.	 Explore data Sort, filter, search, and drill into operational records with ease.	 Correlate information Join API data with other operational datasets for deeper insight across the organization.
 The Data Matrix Producer is often the first step in turning raw operational data into information that people can search, report on, compare, correlate, and visualize.			

In the next section, you'll configure a Data Matrix Producer that maps JSON fields into columns, creates a reusable operational dataset, and stores it in Ceeview for dashboards, reports, search, and further analysis.

3. Extending the REST Monitor with Data Matrices

Now that the REST Monitor contains the Health, Metric, and Service Producers from the previous tutorials, this section adds a Data Matrix Producer that transforms operational JSON data into a reusable Ceeview dataset.

Rather than creating Assets, Metrics, or Service Models, the Data Matrix Producer maps JSON fields, applies transformation logic, and organizes the results into a structured table that can be searched, reported on, and reused throughout Ceeview.

The blueprint below illustrates how operational JSON data is transformed into a reusable Ceeview dataset.

Throughout this tutorial, the Data Matrix Producer:

- maps JSON fields into reusable columns
- applies editor functions where needed
- creates a structured Data Matrix
- stores the dataset in Ceeview Data Matrix Storage

JSON Payload (one record)

JSON Structure

```

root [OBJECT]
└─ rows [ARRAY]
  └─ [0] [OBJECT]
    1 A name = ROAC115721T.LOCAL
    2 # device_score = 9500
    3 A group = Berger
    # cpu_total_percentage = 55
    # cpu_queue_length = 0
    # network_bytes_sent_per_sec = 825
    # latency_result_1 = 1
    # network_bytes_received_per_sec = 1280
    # network_bytes_sent_per_sec = 346
    # hw_model = 103C_5330US HP Enqueue Flex HP Enqueue Flex Pro-C
    # hw_cpu_pcore_count = 4
    # hw_memory_total_bytes = 21474836480
    [1] [OBJECT]
    [2] [OBJECT]
    [3] [OBJECT]
    [4] [OBJECT]
    [5] [OBJECT]

```

Resulting Data Matrix (11 rows by 4 columns)

DataMatrixStorage Data: HEZPFOPGYC 'deviceScore'

Size: 11 rows by 4 columns

Time	Name	Group_name	Device_score
1783324432344	ROAC115712P	Berger	95
1783324432344	XOAC115712P	Berger	100
1783324432344	ROAC016C5KZ	Berger	100
1783324432344	ROAC0257L0R	Berger	100
1783324432344	ROAC016C5NK	Berger	100
1783324432344	ROAC016C57G	Berger	100
1783324432344	ROAC115721T	Skullerud	100
1783324432344	ROAC0149R0H	Skullerud	100
1783324432344	ROAC147BDJ3	Skullerud	100
1783324432344	ROAC115712B	Asker	100
1783324432344	ROAC0257L0K	Asker	100

Editor Function: scoreConvert()

```

1 //Remove ".LOCAL" from variable
2 - def stripworkgroup(String name) {
3   if (name.contains(".LOCAL")) {
4     return name.replace(".LOCAL", "")
5   }
6   return name
7 }
8
9
10 //Divide score reported by 100
11 def scoreConvert(score) {
12   return (score.toBigDecimal() / 100).toInteger() // ensure Integer is returned
13 }

```

About scoreConvert()

The scoreConvert() function divides the raw device_score value (0–10,000 scale) by 100 and converts the result to an integer, producing a simplified 0–100 device score.

This function is referenced in the Data Matrix Producer to store clean, consistent scores.

Field Mapping Summary

#	JSON Field	Used For
1	name	Device name (cleaned of .LOCAL)
2	device_score	Device score (0–10,000) converted to 0–100 using scoreConvert()
3	group	Location / region for grouping (e.g., Berger, Asker, Skullerud)

The Data Matrix Producer maps the JSON fields above into a structured operational dataset with 4 columns: Time, Name, Group_name, and Device_score.

The remainder of this chapter walks through the complete transformation—from individual JSON fields to a reusable operational dataset stored in Ceeview.

Step 1: Add the scoreConvert() Function

The blueprint on the previous page introduced the **scoreConvert()** function used by the Data Matrix Producer. In this step, you'll add the reusable function to the REST Monitor Editor so it's available when configuring the Data Matrix Builder.

Create the Function

Open the **Editor** from the Profile Designer toolbar and add the following function beneath the existing **stripWorkgroup()** function:

```
//Divide score reported by 100
def scoreConvert(score) {
    return (score.toBigDecimal() / 100).toInteger() // ensure Integer is returned
}
```

After adding the function, the completed Editor should appear as shown below.

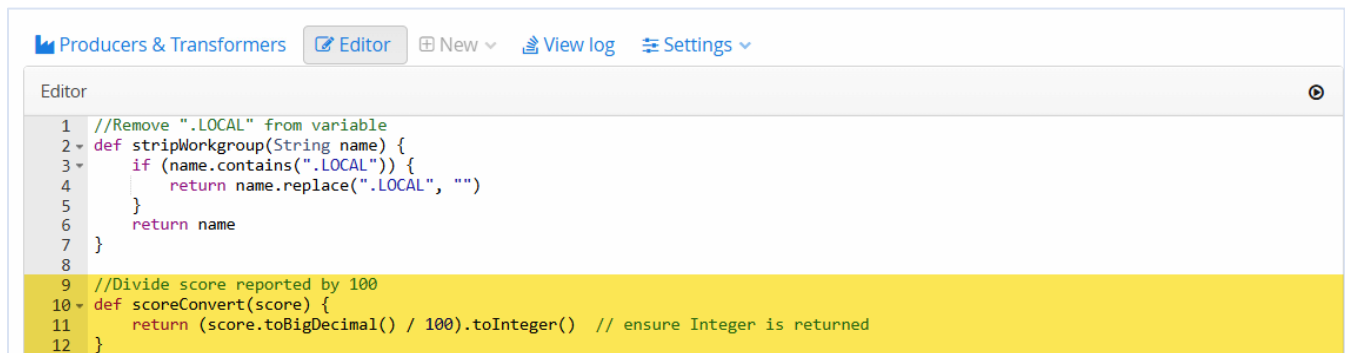


Figure 1 · Editor containing the reusable **stripWorkgroup()** and **scoreConvert()** functions used throughout this REST Monitor configuration.

Click **Update** to save the new Editor function before continuing.

With the **scoreConvert()** function in place, you're ready to configure the Data Matrix Builder and map the JSON fields introduced in the blueprint.

Step 2: Create Data Matrix Producer

In the Profile Designer select **New** → **Producer** → **Data Matrix**.

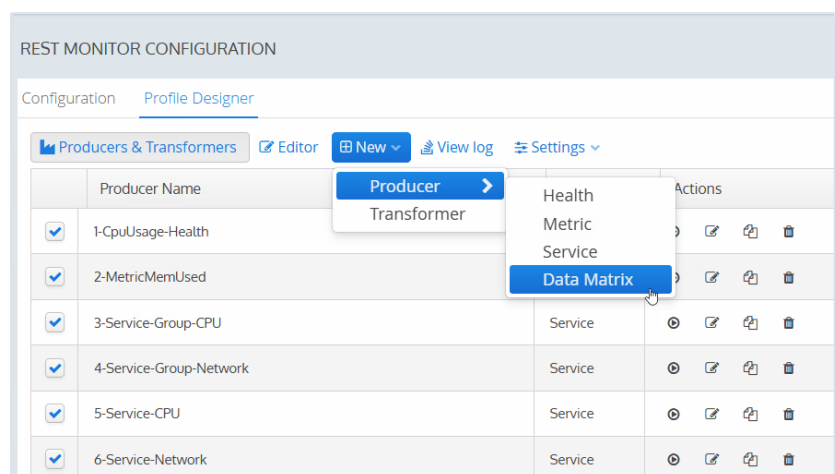


Figure 2 · Selecting **New** → **Producer** → **Data Matrix** from the menu

Step 3: Configure General Properties

Complete the **General Properties** tab as shown below. These settings define where the Data Matrix will be stored and how the producer iterates through the REST API response to create a reusable operational dataset.

DataMatrix Producer Setup

1 7-Device-Score-Matrix *

General Properties DataMatrix Builder Filter Designer

2 Path * Nordic_Retail_Device_Score

3 Array Iterator Key root.rows

Description Structured device inventory and score dataset

☒ Replace Content

1 **Producer Name**
A descriptive name displayed within the REST Monitor. Any meaningful name may be used.

2 **Path**
Specifies where the Data Matrix is stored in **Ceeview Data Matrix Storage**. The producer updates this dataset each time it runs, making it available throughout Ceeview for dashboards, reports, searches, and analytics.

3 **Array Iterator Key**
Enter **root.rows** to iterate through each device returned by the REST API while building the Data Matrix.

i Fields marked with * are required.

Figure 3 · Data Matrix Producer General Properties configuration. These settings define the Data Matrix location and how the producer processes each device returned by the Nordic Retail REST API.

Next, configure the **Data Matrix Builder** to define the columns and transformation logic used to build the Data Matrix.

Step 4: Build the Data Matrix

The Data Matrix Builder defines the columns that make up the operational dataset. Each row maps a JSON field into a Data Matrix column, with optional editor functions applied where data transformation is required.

Complete the following steps to build the Data Matrix columns.

1. Drag **name** from the JSON Structure into the first Data Matrix row.

General Properties | **DataMatrix Builder** | Filter Designer

DataMatrix Column Configuration

Name	Type	Value
	String	stripWorkgroup(\${name})

Drag 'name' from JSON

```

# hw cpu pcore count = 4
# _created = 2023-06-01T17:11:39.450Z
# memory available percentage = 52
# bsods = 0
# memory total bytes used = 10191310848
# name = ROAC1571ZP.LOCAL
# cpu total percentage = 55
  
```

2. Apply **stripWorkgroup(\${name})** to remove the **.LOCAL** suffix from the device name.

General Properties | **DataMatrix Builder** | Filter Designer

DataMatrix Column Configuration

Name	Type	Value
name	String	stripWorkgroup(\${name})

Apply 'stripWorkgroup()' function

3. Drag **group** into the next Data Matrix row.

General Properties | **DataMatrix Builder** | Filter Designer

DataMatrix Column Configuration

Name	Type	Value
name	String	stripWorkgroup(\${name})
group	String	\${group}

Drag 'group' from JSON

```

# memory available percentage = 52
# bsods = 0
# memory total bytes used = 10191310848
# name = ROAC1571ZP.LOCAL
# cpu total percentage = 55
# memory_total_bytes_available = 11046850560
# device score = 9500
# hw_memory_total_bytes = 21474836480
# group = Berger
# _id = 9Djxd4qBokpuBWhvjh-7
  
```

4. Drag **device_score** into the next Data Matrix row.

General Properties | **DataMatrix Builder** | Filter Designer

DataMatrix Column Configuration

Name	Type	Value
name	String	stripWorkgroup(\${name})
group	String	\${group}
device_score	Number	scoreConvert(\${device_score})

Drag 'device_score' from JSON

```

# taos [ARRAY]
# value [ARRAY]
# hw cpu pcore count = 4
# _created = 2023-06-01T17:11:39.450Z
# memory available percentage = 52
# bsods = 0
# memory total bytes used = 10191310848
# name = ROAC1571ZP.LOCAL
# cpu total percentage = 55
# memory_total_bytes_available = 11046850560
# device score = 9500
# hw_memory_total_bytes = 21474836480
  
```

5. Apply **scoreConvert(\${device_score})** to normalize the device score before storing it in the Data Matrix.

DataMatrix Column Configuration

Name	Type	Value
name	String	stripWorkgroup(\${name})
group_name	String	\${group}
device_score	Number	scoreConvert(\${device_score})

Apply 'scoreConvert()' function

Step 5: Execute the DataMatrix Producer

Click **Run** in the upper-right corner of the Data Matrix Producer to execute the configuration using the current JSON response.

Review the Output panel to verify that the Data Matrix is created successfully before saving the Producer.

7-Device-Score-Matrix

General PropertiesDataMatrix BuilderFilter Designer

DataMatrix Column Configuration

Name	Type	Value
name	String	stripWorkgroup(\$(name))
group_name	String	\$(group)
device_score	Number	scoreConvert(\$(device_score))

Drop a JSON field here to add a new row

JSON Structure

Output

Executing DATAMATRIX producer: 7-Device-Score-Matrix
Executing configuration at 2026-07-13T23:17:31.858
Total JSON Array items: 11, added: 11

DataMatrix: Nordic_Retail_Device_Score
Rows: 11, Columns: 3

Name	Group_name	Device_score
ROAC11571ZP	Berger	95
XOAC11571ZP	Berger	100
ROAC016C5KZ	Berger	100
ROAC0257LOR	Berger	100
ROAC016C5NK	Berger	100
ROAC016C57G	Skullerud	100
ROAC11572IT	Skullerud	100
ROAC0149ROH	Skullerud	100
ROAC147BDJ3	Skullerud	100
ROAC11571ZB	Asker	100
ROAC0257LOR	Asker	100

Execution completed successfully in 19 ms.

Note: After verifying the Output panel, close the Data Matrix Producer window using the X in the upper-right corner. REST Monitor prompts you to save the Producer before returning to the Profile Designer.

Step 6: Update the REST Monitor

Click Update to save the REST Monitor configuration and activate the new Data Matrix Producer. The producer will execute automatically each time the REST Monitor runs.

Update

JSON Structure

root [OBJECT]

rows [ARRAY]

[0] [OBJECT]

Figure 4 · Saving the REST Monitor activates the new Data Matrix Producer.

Step 7: Verify the Data Matrix

The Data Matrix has now been created and stored in Ceeview. In this step, you'll verify the resulting dataset by viewing it in **Data Matrix Storage**.

1. From the Ceeview main menu, open **Management** → **Integration** → **Sidekick Utilities**.
2. Select **Data Sources** from the left navigation.
3. Open the **DataMatrix Storage** tab.
4. Locate the **Nordic_Retail_Device_Score** entry.
5. Select the entry and click **View Table**.

Time	Name	Group_name	Device_score
1783966354902	ROAC11571ZP	Berger	95
1783966354902	XOAC11571ZP	Berger	100
1783966354902	ROAC016C5KZ	Berger	100
1783966354902	ROAC0257LOR	Berger	100
1783966354902	ROAC016C5NK	Berger	100
1783966354902	ROAC016C57G	Skullerud	100
1783966354902	ROAC115721T	Skullerud	100
1783966354902	ROAC0149ROH	Skullerud	100
1783966354902	ROAC147BDJ3	Skullerud	100
1783966354902	ROAC11571ZB	Asker	100
1783966354902	ROAC0257L0K	Asker	100

Figure 5 · Viewing the completed **Nordic_Retail_Device_Score** Data Matrix in Ceeview Data Matrix Storage.

The completed Data Matrix contains one row for each device returned by the REST API. The dataset includes the normalized device name, location, and converted device score that were configured throughout this tutorial.

The Data Matrix is now available within **Data Matrix Storage**. From here, it can be promoted to **All DataSources** where it becomes available for dashboards, reports, searches, and other Ceeview capabilities. That process is covered separately from this tutorial.

Step 8: Verify the REST Monitor Configuration

After updating the configuration, the Nordic Retail Example REST Monitor now includes the new **Data Matrix Producer** alongside the previously created Health, Metric, and Service Producers. Together, these producers transform the same REST API response into multiple native Ceeview objects.

REST MONITOR CONFIGURATION

Configuration Profile Designer

Producers & Transformers

Editor

New ▾

View log

Settings ▾

	Producer Name	Type	Actions
☒	1-CpuUsage-Health	Health	
☒	2-MetricMemUsed	Metric	
☒	3-Service-Group-CPU	Service	
☒	4-Service-Group-Network	Service	
☒	5-Service-CPU	Service	
☒	6-Service-Network	Service	
☒	7-Device-Score-Matrix	Datamatrix	

Figure 6 · Completed REST Monitor containing Health, Metric, Service, and Data Matrix Producers.

Complete the REST Monitor Learning Series

Congratulations! You've now completed the REST Monitor Learning Series. Starting from a single REST API, you've progressively built native Ceeview Health Assets, Metrics, Service Models, and reusable Data Matrices that work together to deliver rich operational visibility.

✓ KEY TAKEAWAYS

After completing this tutorial, you now know how to:



Build reusable operational datasets

Transform REST API responses into structured Data Matrices that can be searched, reported on, and reused throughout Ceeview.



Normalize API data

Apply Editor functions to standardize values before storing them in the Data Matrix.



Create structured tables from JSON

Map JSON fields into reusable columns without writing code.



Store operational data in Ceeview

Generate Data Matrices that become persistent datasets within Ceeview Data Matrix Storage.



Prepare data for analytics

Build datasets ready for dashboards, reports, filtering, searches, and future operational analysis.



One REST API can now produce Health Assets, Metrics, Service Models, and Data Matrices—creating multiple operational perspectives from the same continuously monitored data source.



LEARNING SERIES PROGRESS

Build the same REST Monitor step by step.



Tutorial 1

Health Producer

Completed



Tutorial 2

Metric Producer

Completed



Tutorial 3

Service Producer

Completed



Tutorial 4

Data Matrix Producer

Completed

Across these four tutorials, you've progressively extended the same REST Monitor to build multiple operational views from a single continuously monitored REST API—without duplicating data collection or writing code.

A single REST API isn't just a source of data—it becomes a foundation for operational visibility.