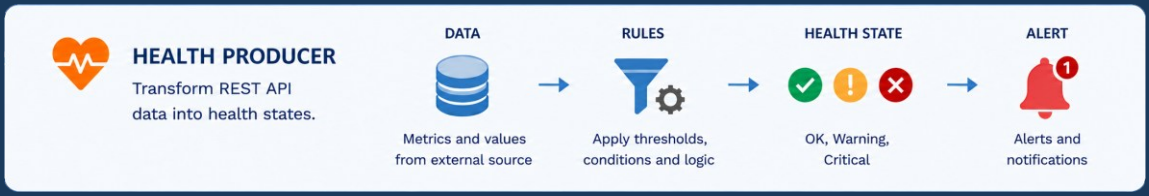
CEEVIEW

REST Monitor Learning Series

Tutorial 1

Creating a Multi-Asset Health Producer

Monitoring Nordic Retail Devices



Version 2.19 · DataIntegration Package

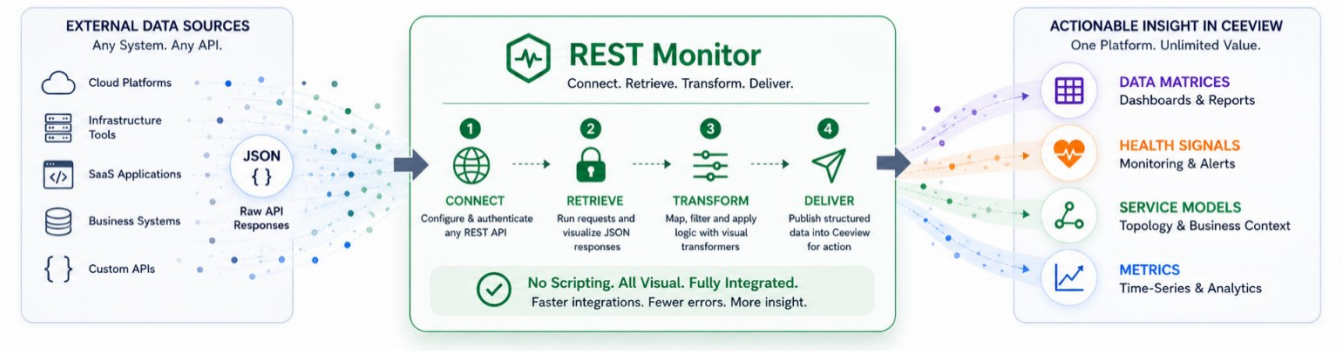
This tutorial is Tutorial 1 in the REST Monitor Learning Series.

In this tutorial, you'll create the Nordic Retail Example REST Monitor and configure a Health Producer that transforms REST API JSON data into native Ceeview Health Assets. Along the way you'll also create a reusable Transformer and Editor function that will be used throughout the tutorials that follow.

1. REST Monitor Overview

The Ceeview REST Monitor is part of the DataIntegration package and provides a flexible framework for integrating REST-based APIs into Ceeview. Any system that exposes data through a REST API and returns JSON can be integrated into Ceeview through the REST Monitor.

At scheduled intervals, REST Monitor polls external REST endpoints and transforms JSON responses into Ceeview observability data, including Health Reasons, Metrics, Service Models, and Data Matrices.



The example used throughout this tutorial focuses on a Ceeview Nordic Retail sandbox API, but the same workflow can be used with any REST API.

REST Monitor Components

Producers

Four producer types are available: Health, Metric, Service, and Data Matrix. Each converts endpoint JSON data into a specific Ceeview data type.

Transformer

A no-code rules engine used to map raw JSON values into Ceeview Health, Metric, Service, or Data Matrix values. In this tutorial, CPU utilization is transformed into Health severity states.

Groovy Script

Provides direct access to the JSON response for advanced calculations, data manipulation, conditional logic, and custom output generation.

Execution Log

Each monitor execution is logged and can be used to verify operation and troubleshoot configuration issues.

2. Monitoring Nordic Retail Devices

In this example, we will use a Ceeview sandbox REST API to monitor eleven retail devices distributed across three Nordic Retail locations: Berger, Skullerud, and Asker.

The sandbox API returns a JSON array containing device information including device name, location, CPU utilization, memory availability, network activity, latency measurements, and device score values. Using REST Monitor, this data is transformed into Ceeview Health Assets. The Health Producer automatically creates an Asset for each returned device. If a matching Asset already exists in the Ceeview environment, it is reused rather than created again. All Assets are then placed into an Asset Group named Nordic Retail Example.

This tutorial introduces several REST Monitor capabilities, including multi-asset creation, array iteration, data transformation, and Editor-based value manipulation. These techniques provide a foundation for building additional Producer types within the same REST Monitor in subsequent tutorials.

Prerequisites

Before starting, ensure the following requirements are met:

- Web service module installed in Ceeview.
- At least one Ceeview Measuring Agent installed, online, and available for selection.
- Network access from the selected Measuring Agent to: saas.ceeview.com

1 Create the REST Monitor

Navigate to Infrastructure → Monitors → REST, then click Create.

2 Configuration Tab

Set the following endpoint details:

Field	Value
Name	Nordic Retail Example
URL	`https://saas.ceeview.com/public/examples/rest.json`
Method	`GET`
Header param	Accept = application/json
Measuring Agent	Select a Ceeview Measuring Agent
Interval	Every 5 minutes

About This Example: The Ceeview sandbox REST API is designed for learning REST Monitor. No authentication or API key is required, allowing you to focus on configuration, data transformation, and producer creation.

With the REST Monitor configured, the next step is to verify that the connection to the Ceeview sandbox REST API is successful. A successful connection confirms that REST Monitor can communicate with the endpoint and retrieve the JSON data required for producer configuration.

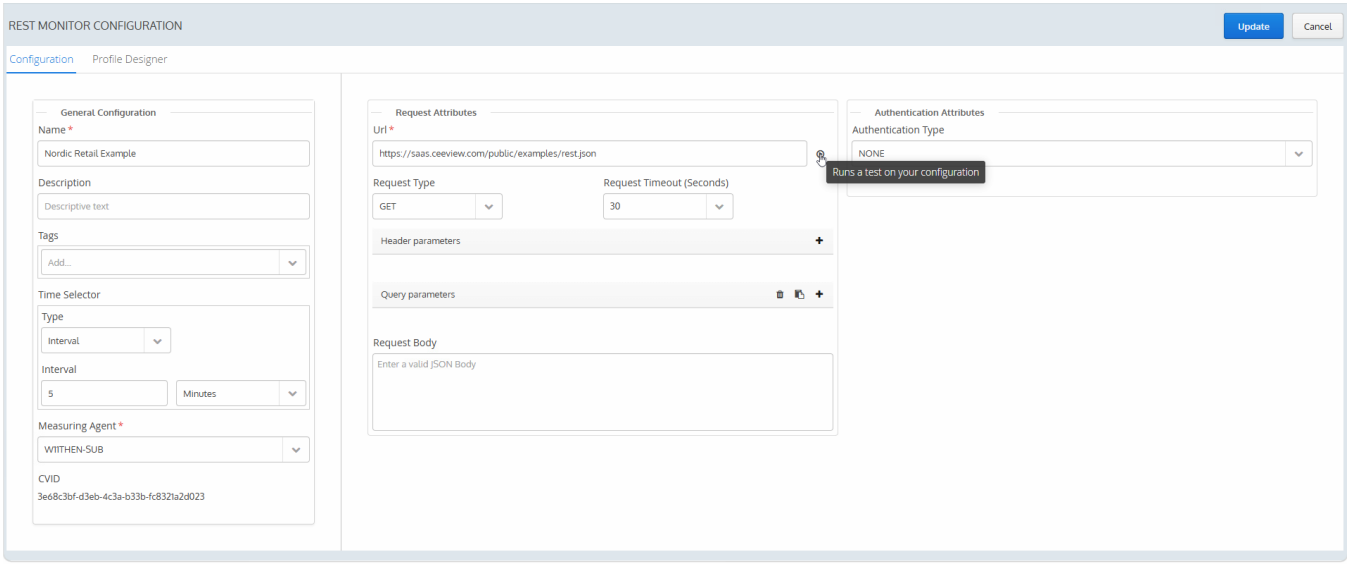


Figure 1 · REST Monitor Configuration tab showing the completed Ceeview sandbox REST API configuration. No query parameters or authentication are required for this tutorial.

After verifying the configuration, click the **Run Connection Test** button to confirm that REST Monitor can successfully communicate with the REST API endpoint.

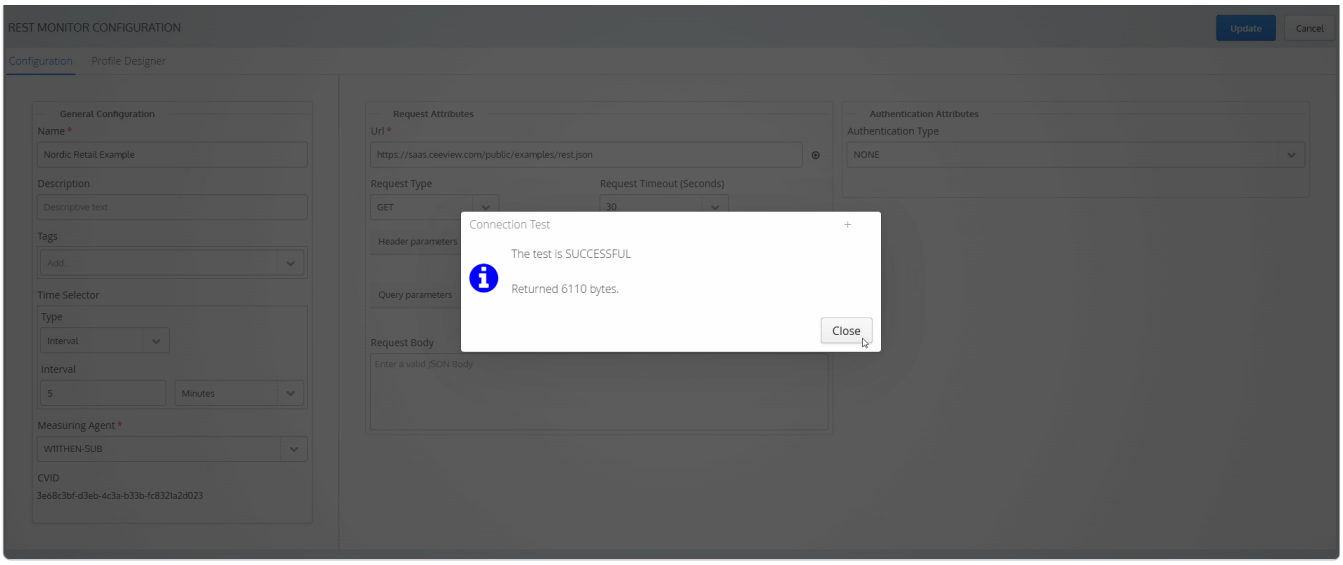


Figure 2 · Successful connection test confirming that REST Monitor can retrieve data from the Ceeview sandbox REST API.

The REST Monitor is now fully configured and connected to the Ceeview sandbox REST API. In the next section, we'll examine the returned JSON data and begin configuring the Health Producer.

3 Understanding the JSON Structure

After successfully testing the connection, open the Profile Designer tab. REST Monitor automatically retrieves and parses the JSON response from the REST API. The JSON Structure panel displays all available fields that can be referenced by Producers, Transformers, and Editor functions.

The Nordic Retail example returns multiple device records. The following table shows key JSON fields returned by the API. Throughout this tutorial, we'll use a subset of these fields to create Health Assets and configure Health monitoring in Ceeview.

JSON Field	Description
`name`	Device name used to create each Asset.
`cpu_total_percentage`	Overall CPU utilization used to determine Asset Health status.
`memory_available_percentage`	Available system memory percentage.
`network_bytes_received_per_sec`	Network receive throughput.
`network_bytes_sent_per_sec`	Network transmit throughput.
`group`	Device location (Berger, Skullerud, or Asker).
`device_score`	Calculated device health score returned by the API.

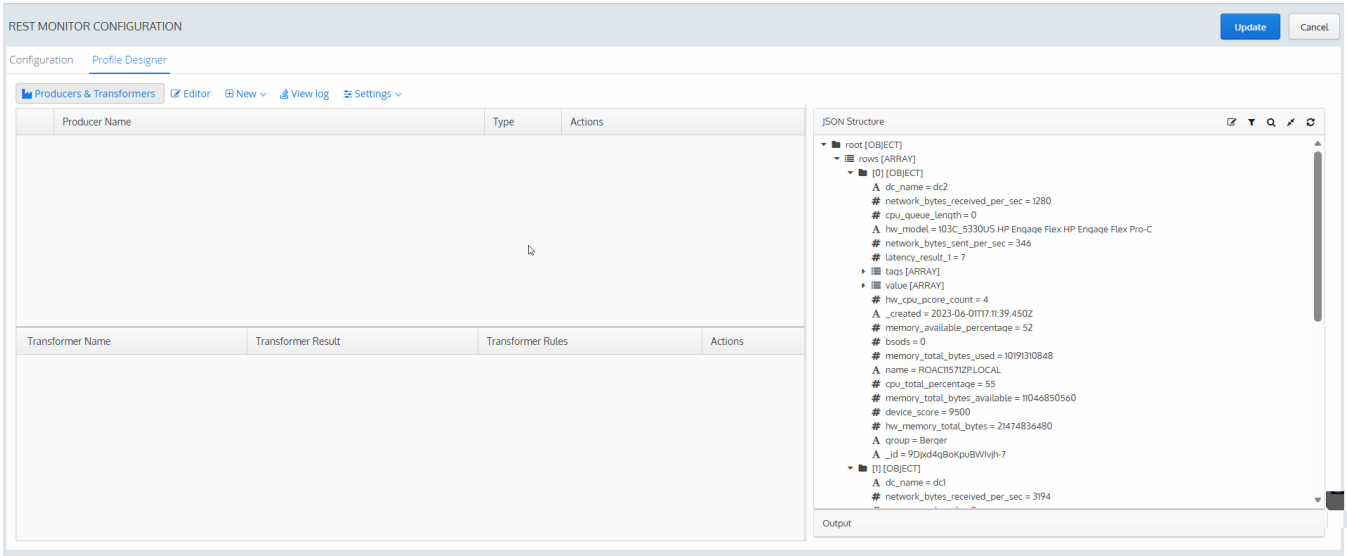


Figure 3 · Profile Designer displaying the parsed JSON structure retrieved from the Ceeview sandbox REST API.

Note: The Ceeview sandbox API used in this tutorial returns a static JSON response. The sample dataset is intentionally fixed so every user completes the tutorial using the same data. Consequently, the generated Health states and Health messages remain the same each time the Producer is executed. When connected to a live REST API, the same configuration automatically reflects changing operational conditions as new JSON data is retrieved.

4 Define a Transformer

Now that we understand the available JSON fields, we'll create a Transformer that converts the returned CPU utilization value into a Ceeview Health severity state.

In the Profile Designer, select New → Transformer.

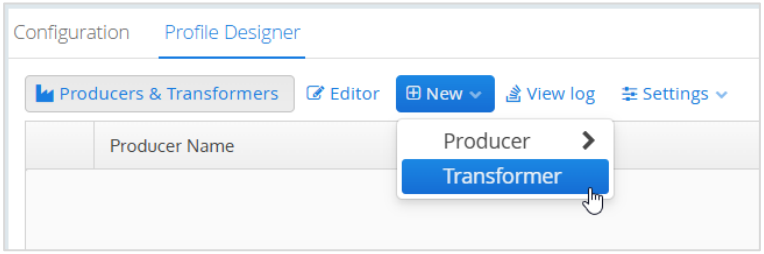


Figure 4 - Select New -> Transformer from the Profile Designer toolbar.

Configure the Transformer

Create a Transformer named cpuSeverity and configure it as shown below.

Condition	Operand	Return Value
GreaterEqual	90	CRITICAL
GreaterEqual	60	MAJOR
GreaterEqual	10	MINOR
Exists	0	NONE

Note: The Exists condition provides the default severity for any value that does not match one of the preceding conditions.

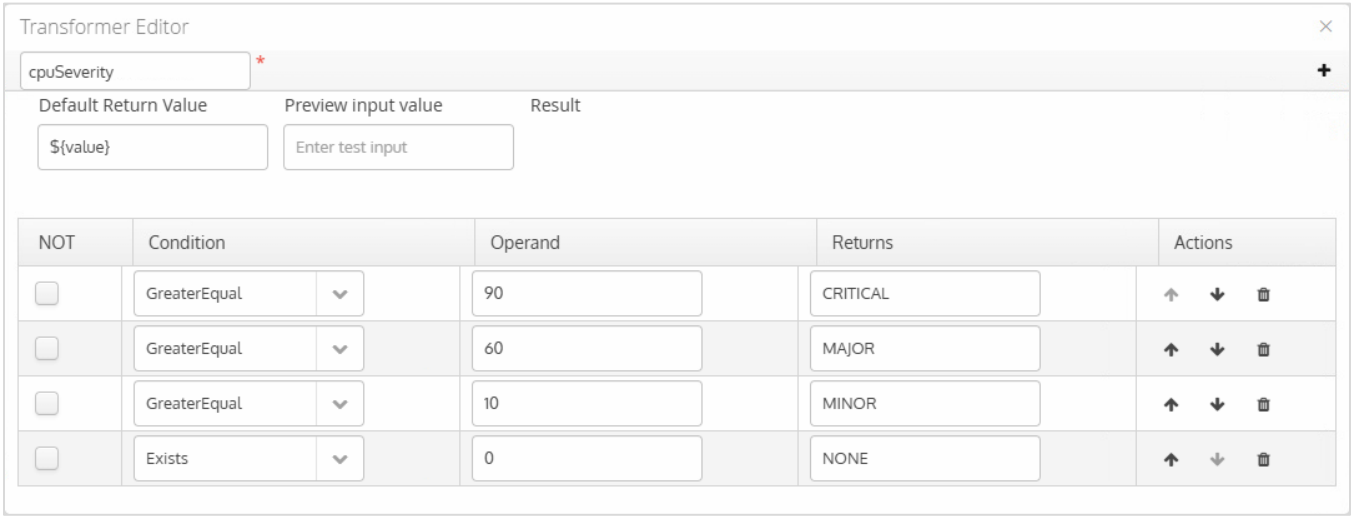


*Figure 5 · Completed **cpuSeverity** Transformer configured to convert CPU utilization values into Health severity levels.*

Note: Close the Transformer Editor using the **X** in the upper-right corner. REST Monitor will prompt you to save the Transformer before returning to the Profile Designer.

5 Create an Editor Function

While Transformers convert returned values into business logic, the REST Monitor Editor provides Groovy functions that manipulate data before it is consumed by Producers. In this section, we'll create an Editor function that removes the .LOCAL suffix from each device name, resulting in cleaner Asset names throughout Ceeview.

Create the Function

Open the Editor from the Profile Designer toolbar and copy the following function into the Editor.

```
//Remove ".LOCAL" from variable
def stripWorkgroup(String name) {
    if (name.contains(".LOCAL")) {
        return name.replace(".LOCAL", "")
    }
    return name
}
```

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

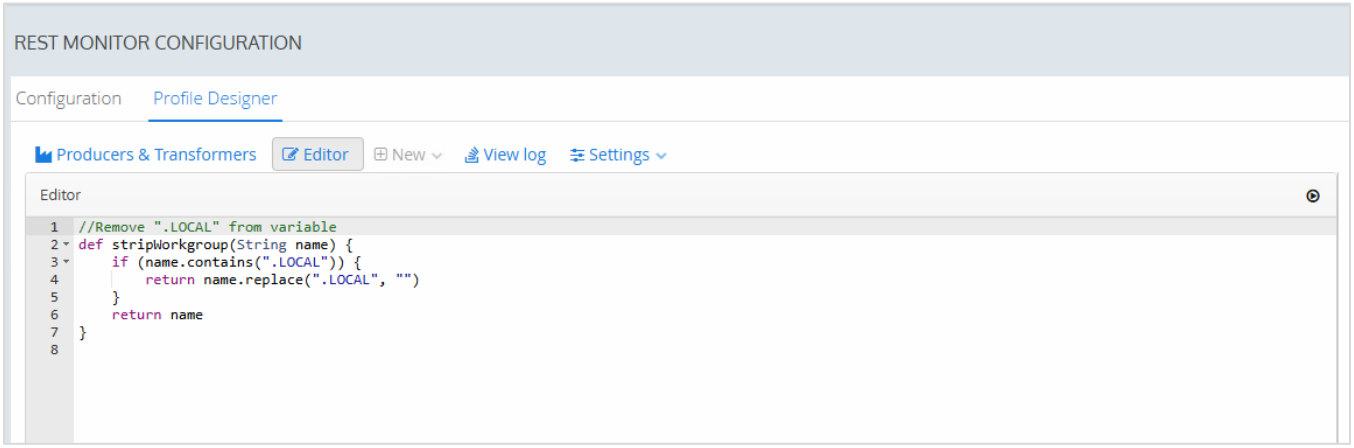


Figure 6 · Editor containing the reusable stripWorkgroup() function used to remove the .LOCAL suffix from device names before creating Assets.

Editor Reference: REST Monitor's Editor supports reusable Groovy functions that can be referenced throughout the REST Monitor configuration. For additional scripting examples, function reference information, and Editor documentation, see the REST Monitor section of the Ceeview Documentation.

The stripWorkgroup() function is now available throughout this REST Monitor and can be referenced by Producers and Transformers. In the next section, we'll reference this function while configuring the Health Producer to create cleaner Asset names.

6 Create a Health Producer

The Health Producer transforms JSON device data into Ceeview Health Assets. In this section, you'll configure the Producer to automatically create one Health Asset for each device returned by the REST API.

In the Profile Designer select New → Producer → Health.

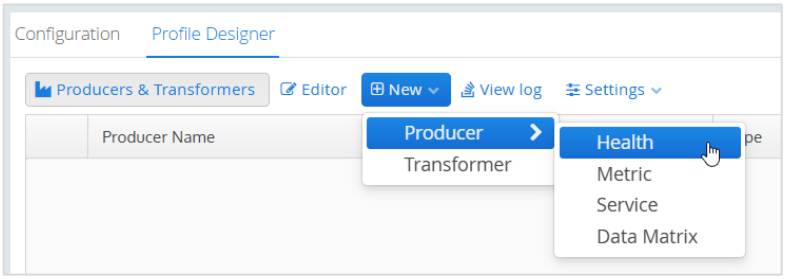


Figure 7 · Selecting New → Producer → Health from the menu.

General Properties

Complete the General Properties tab as shown below. These settings define how the Health Producer creates Health Assets, identifies Health targets, and processes each device returned by the REST API.

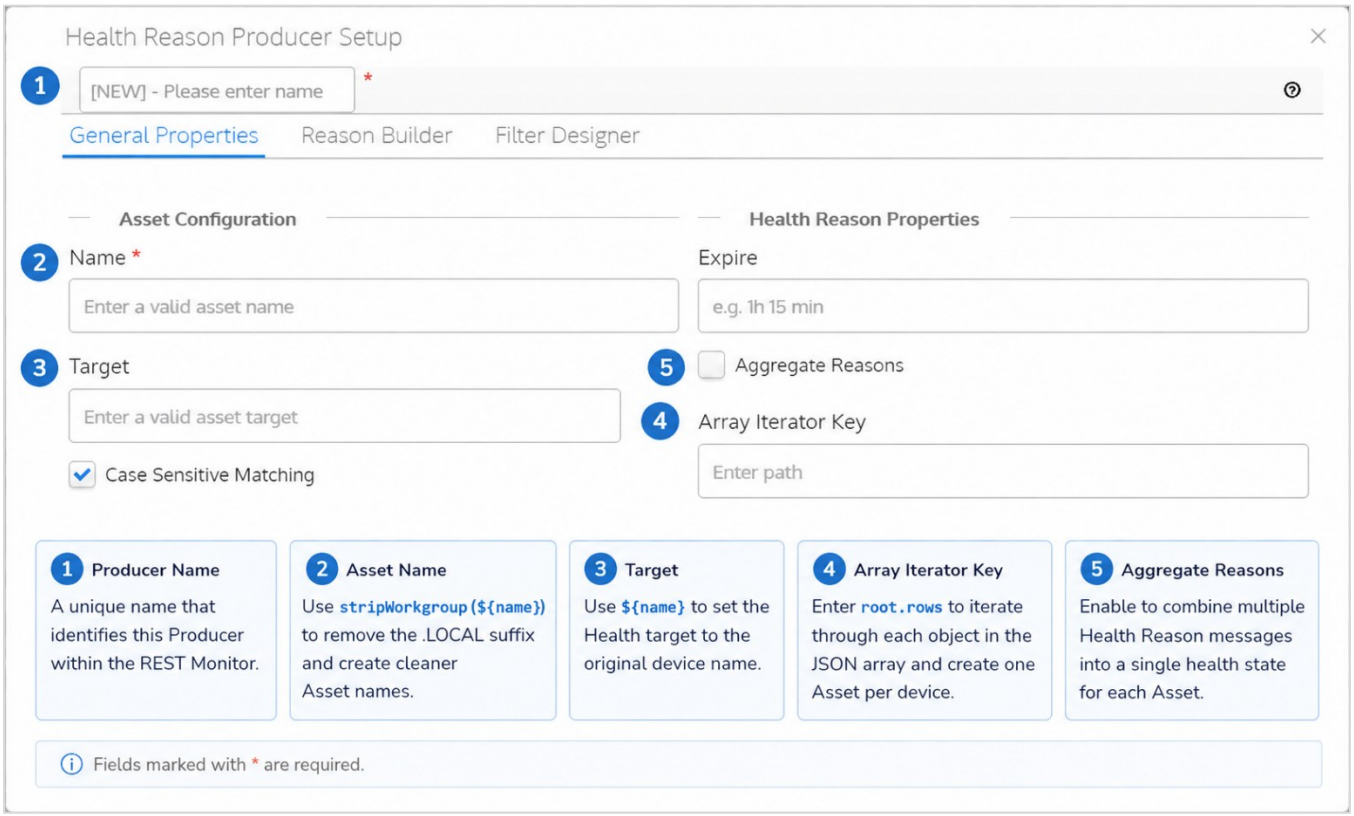


Figure 8 · General Properties configuration for creating one Health Asset for each device returned by the Nordic Retail example REST API.

Use the following values when configuring the General Properties tab. The table complements the annotated illustration above by providing the specific values entered for each setting.

#	Configuration	Tutorial Value	Notes
1	Producer Name	1-CpuUsage-Health	Name displayed within the REST Monitor.
2	Name	stripWorkgroup(${name})	See Note Below.
3	Target	${name}	See Note Below.
4	Array Iterator Key	root.rows	Creates one Asset per JSON device.
5	Aggregate Reasons	Enabled	Combines multiple Health Reasons.

Using JSON Expressions and Editor Functions: REST Monitor references JSON values using expressions enclosed in ${...}. Selecting the JSON **name** field automatically generates the expression ${name}. This tutorial uses that expression in two ways:

- **Target:** ${name} uses the original device name as the Health target.
- **Asset Name:** stripWorkgroup(${name}) passes the same JSON value to the Editor function created in the previous section, removing the .LOCAL suffix before creating each Asset.

Configure the Reason Builder

With the General Properties complete, the next step is configuring the Reason Builder, which defines the Health information generated for each Asset.

Field	Purpose
Severity	Health state
Description	Health message

Configure Severity

1. Drag the **cpu_total_percentage** field from the JSON panel and drop it onto the Reason Builder dropzone.

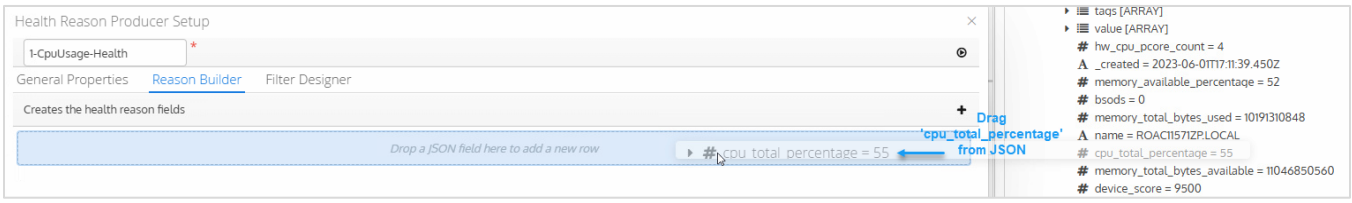


Figure 9 · Dragging the cpu_total_percentage field from the JSON Structure panel into the Reason Builder dropzone.

2. A new Reason Builder row is automatically created and populated. Select **severity** from the field type dropdown.

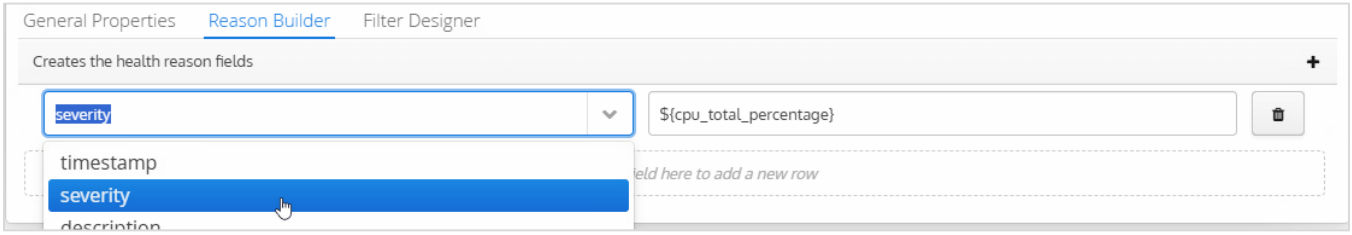


Figure 10 · The drop action automatically creates a Reason Builder row. Select severity as the health reason field type.

3. Apply the **cpuSeverity** Transformer to the generated value field.

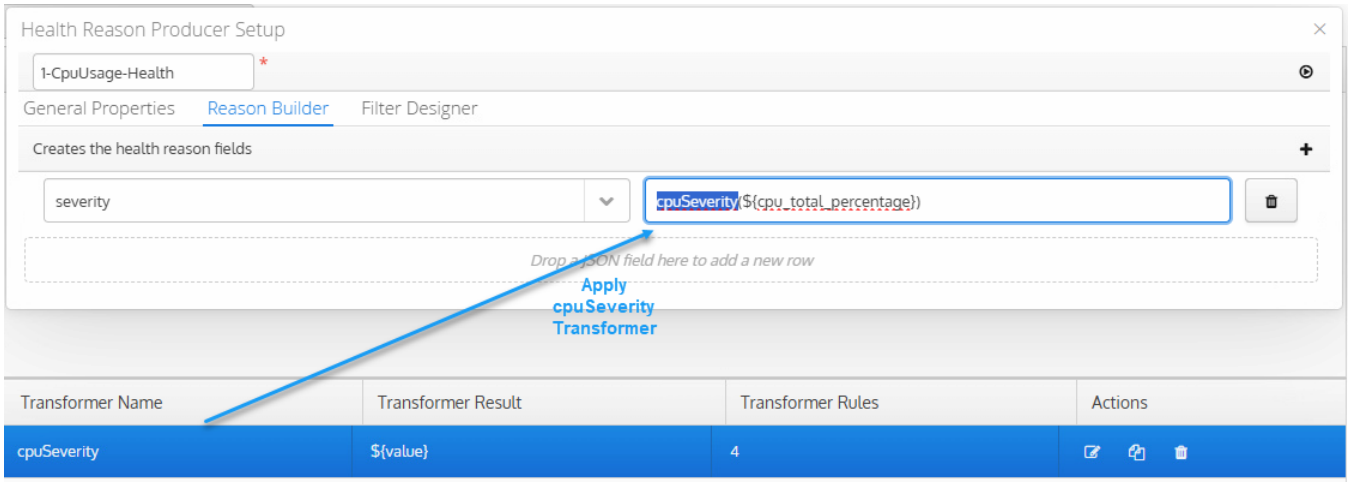


Figure 11 · Applying the cpuSeverity Transformer to convert CPU usage values into health severity states.

Create Dynamic Health Messages

Create a dynamic Health Reason description by combining static text with the CPU utilization value returned by the REST API.

```
Current CPU usage is now: ${cpu_total_percentage}
```

1. Drag the **cpu_total_percentage** field from the JSON Structure panel into the Reason Builder dropzone to create a second row.

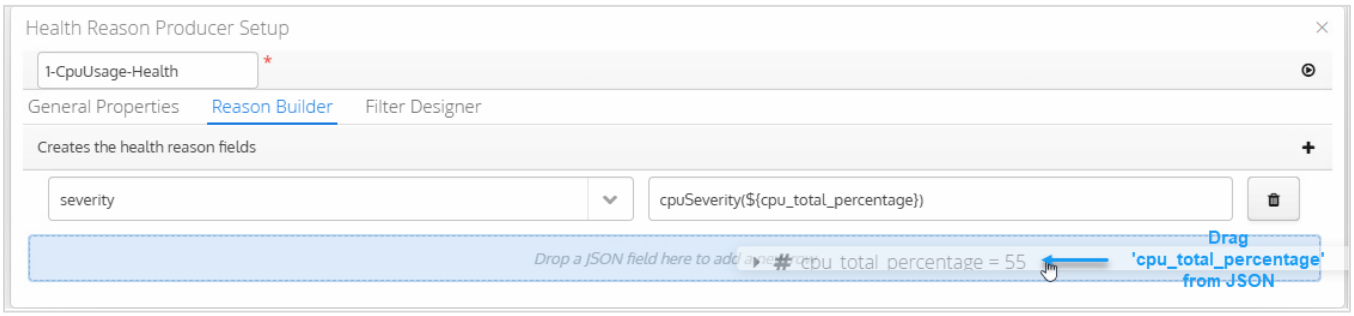


Figure 12 · Dragging the cpu_total_percentage field into the Reason Builder to create a description field.

2. Select **description** from the field type dropdown.

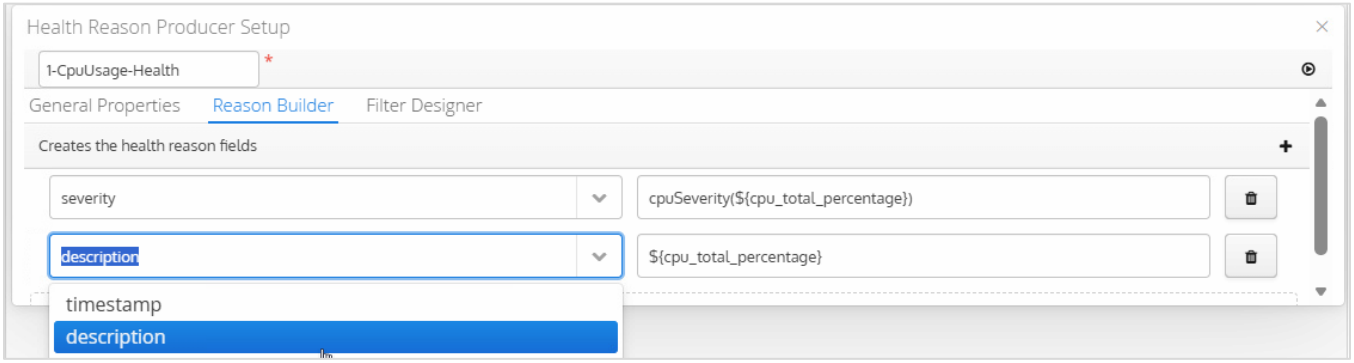


Figure 13 · Configuring a dynamic description that displays the current CPU usage returned by the API.

3. Extend the generated JSON expression with descriptive text to create a dynamic health message.

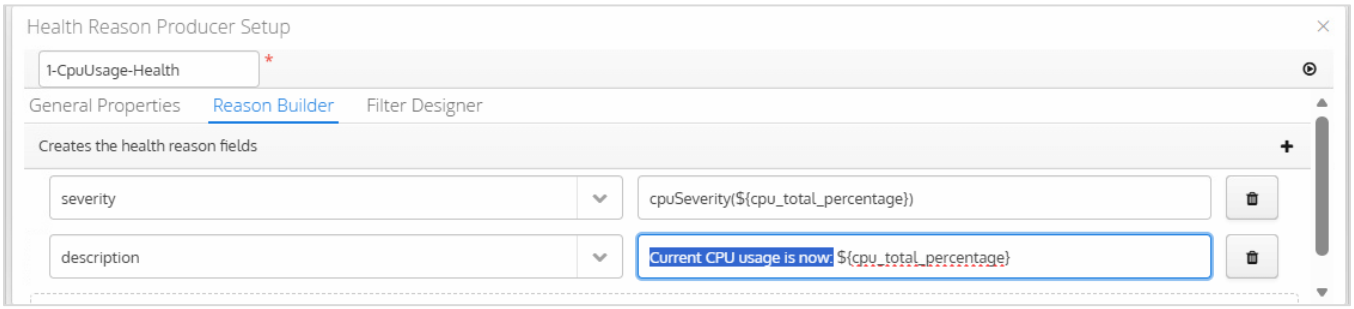


Figure 14 · Extending the generated JSON expression to create a dynamic health message displaying the current CPU usage.

Execute the Producer

Click the Run button in the upper-right corner of the Health Producer window to execute the configuration using the current JSON response.

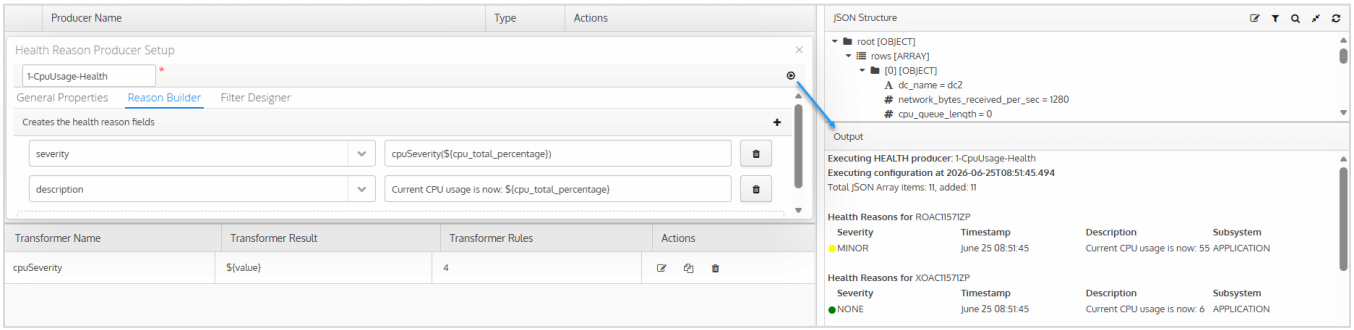


Figure 15 · Producer execution output showing generated Health Reasons for multiple Nordic Retail devices.

> **Note:** After completing the Reason Builder configuration, close the Health Producer window using the **X** in the upper-right corner. REST Monitor will prompt you to save the Producer before returning to the Profile Designer.

Update the REST Monitor

Click **Update** to save the REST Monitor configuration and activate the Health Producer.

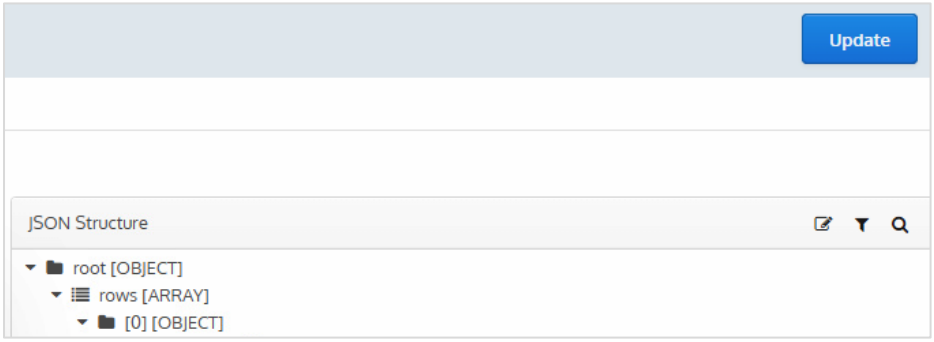


Figure 16 · Click Update to save and activate the monitor configuration.

Verify the Generated Health Assets

Navigate to Infrastructure → Asset Groups to view the Health Assets created by the REST Monitor.

If your Ceeview environment contains many Asset Groups, enter Nordic Retail in the search field to quickly locate the newly created Nordic Retail Example Asset Group.

After execution, the Health Producer has automatically:

✓ Created the Nordic Retail Example Asset Group

✓ Created eleven Health Assets

✓ Assigned Health states based on CPU utilization

✓ Generated descriptive Health messages

✓ Made the Assets immediately available throughout Ceeview

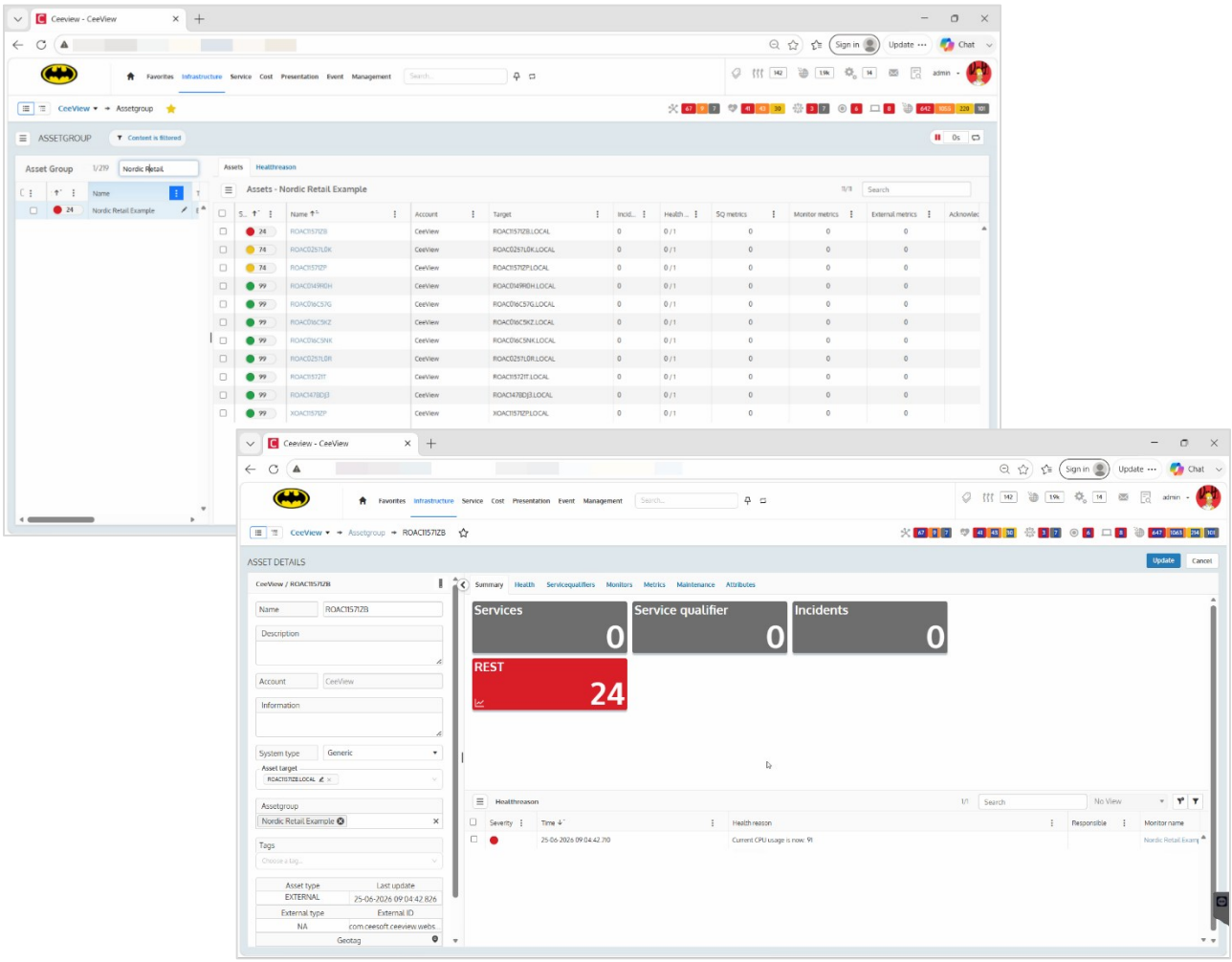
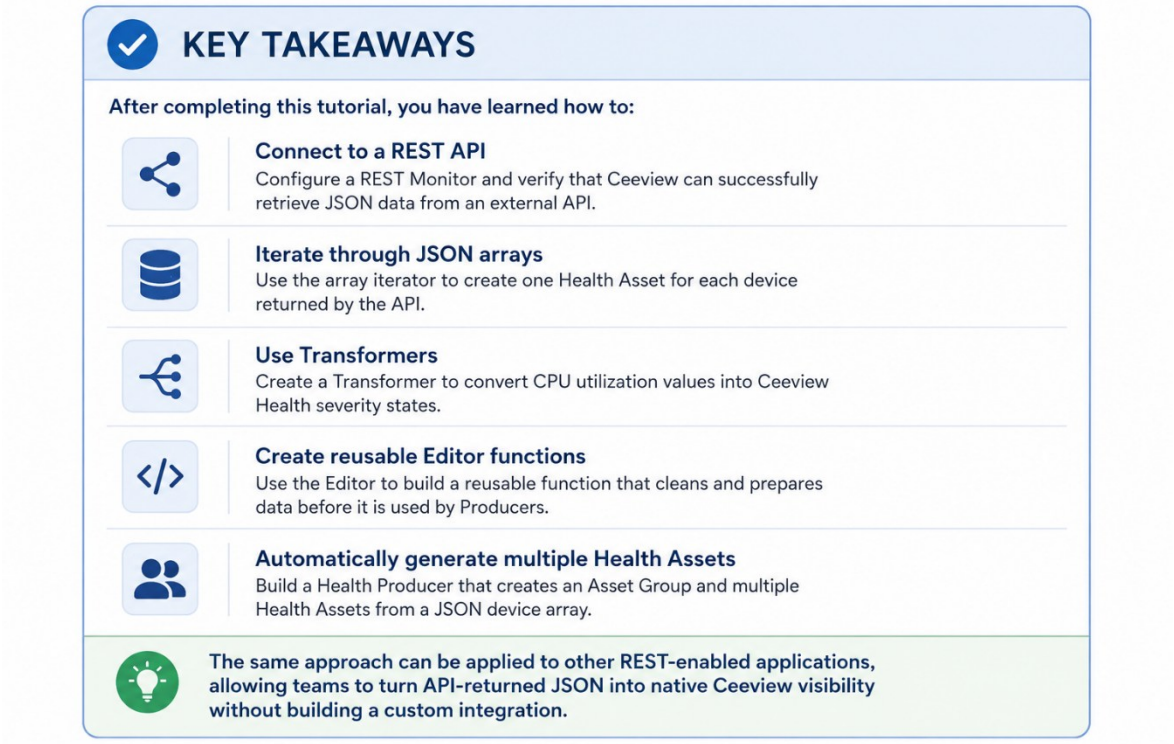


Figure 17 · The generated Nordic Retail Example Asset Group (top) containing eleven Health Assets, and an individual Health Asset (bottom) showing the calculated Health state and generated Health Reason.

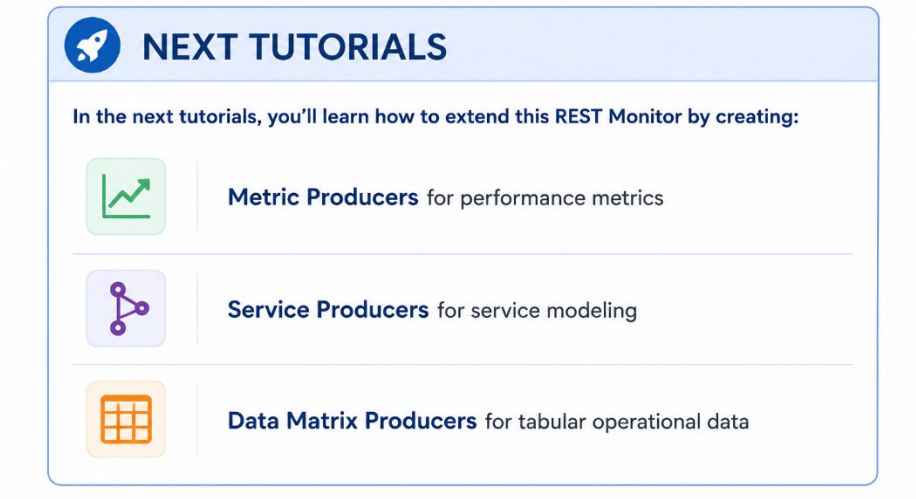
At this point, the REST Monitor is fully operational. Each time the configured interval executes, the Health Producer retrieves the latest JSON data, updates the Health Assets, recalculates Health states, and refreshes the associated Health Reasons automatically.

Continue the REST Monitor Learning Series

Congratulations—you've successfully built a REST Monitor that automatically creates Health Assets from a JSON device array. The concepts introduced here form the **foundation** for every tutorial that follows.



The REST Monitor created in this tutorial becomes the foundation for the remainder of the learning series. Rather than creating additional REST Monitors, you'll continue extending this same monitor by adding additional Producer types.



Together, these Producer types demonstrate one of REST Monitor's greatest strengths: a single REST API can simultaneously create multiple native Ceeview object types—including Health Assets, Metrics, Services, and Data Matrices—providing comprehensive operational visibility from a single JSON source.