

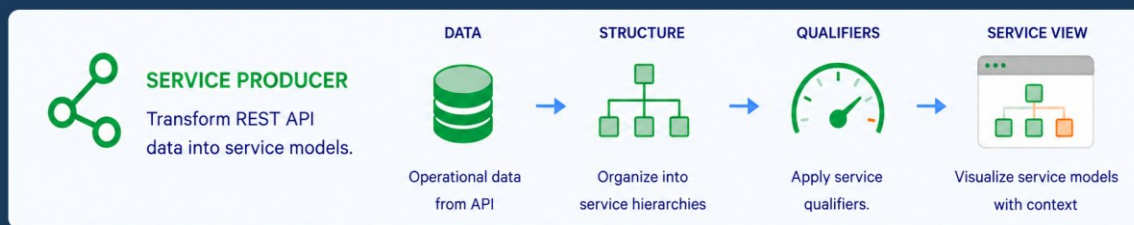
CEEVIEW

REST Monitor Learning Series

Tutorial 3

Creating Operational Service Models with Service Producers

Building Live Ceeview Service Models from REST API Data



Version 2.19 · DataIntegration Package

This tutorial is Tutorial 3 in the REST Monitor Learning Series and assumes Tutorial 1 — Health Producer and Tutorial 2 — Metric Producer have been completed.

In this tutorial, you'll extend the Nordic Retail Example REST Monitor by adding Service Producers that create live Ceeview service models from REST API data.

1. Starting Point from Tutorials 1 and 2

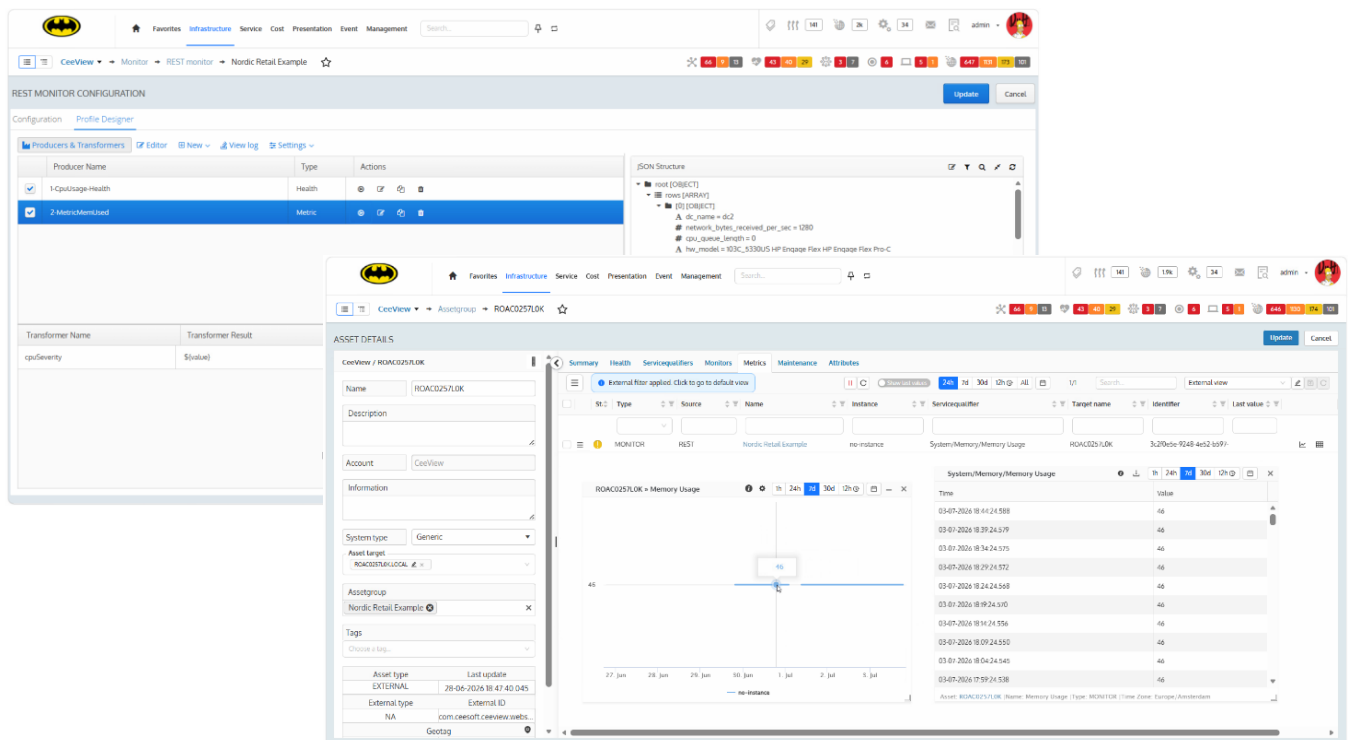
Before beginning this tutorial, your environment should match the completed configuration from **Tutorial 1 — Creating a Multi-Asset Health Producer** and **Tutorial 2 — Creating a Time-Series Metric Producer**.

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 Health Assets
- ✓ Memory Usage Metrics associated with the same Health Assets

Experienced Ceeview users: If you're already familiar with REST Monitor configuration, you may begin with this tutorial directly. Otherwise, refer to Tutorial 1 for the initial REST Monitor setup.

The first screenshot confirms that the Nordic Retail Example REST Monitor now contains both the Health Producer from Tutorial 1 and the Metric Producer from Tutorial 2. The second screenshot confirms that metric history is available for one of the Health Assets created earlier.



What You'll Add: This tutorial adds Service Producers 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, and the new Service Producers will create live Ceeview service models from the same REST API data.

2. From JSON Payloads to Operational Service Models

REST APIs often return more than individual values. They also expose fields that describe how an environment is organized—such as locations, devices, operational resources, and the operational status used to qualify those resources.

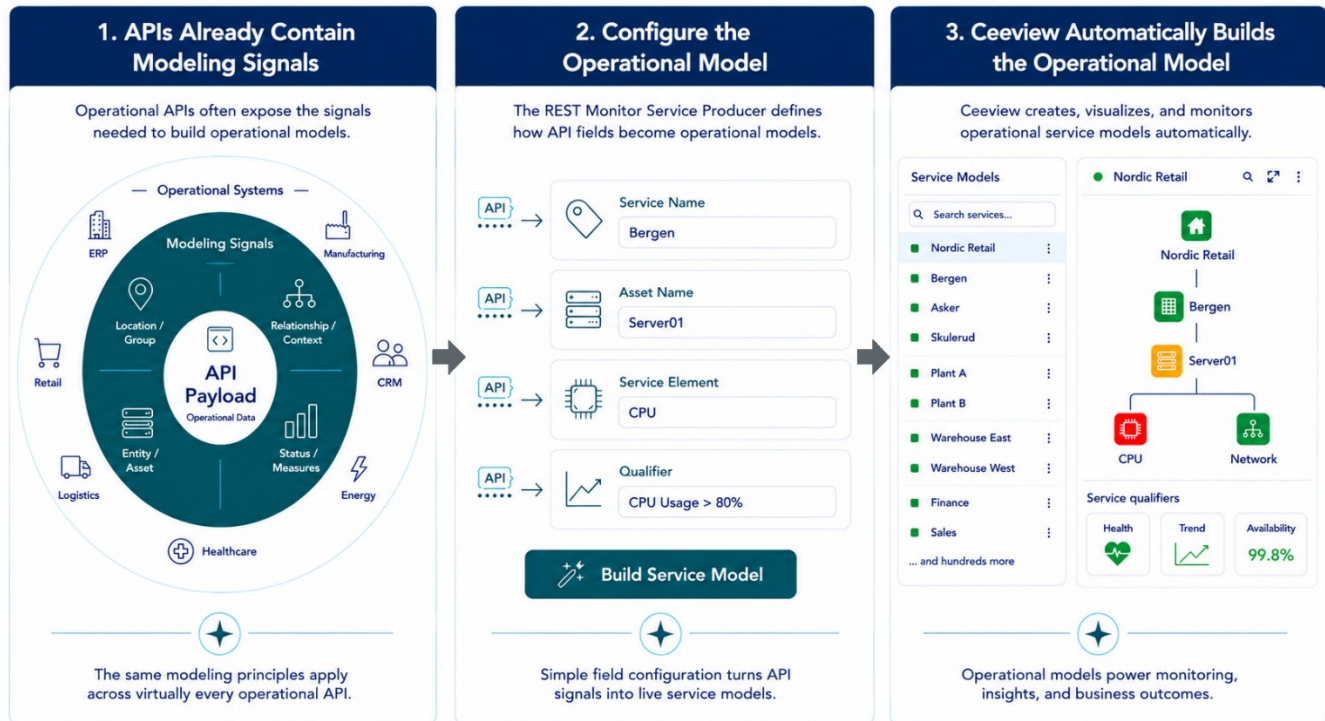


Figure 1 · Service Producers transform API modeling signals into live, monitorable service models in Ceeview.

In this tutorial, you'll use the Nordic Retail REST API to build a service model that organizes locations, server devices, service elements, and service qualifiers:

Nordic Retail → Bergen / Asker / Skulerud → Server devices → CPU / Network → Service Qualifiers

This model becomes visible in Ceeview's Service section and can be monitored, qualified, and used for dashboards, reports, alerts, and operational visibility.

3. Extending the REST Monitor with Service Producers

Now that the REST Monitor contains the Health and Metric Producers from the previous tutorials, this section introduces Service Producers, which automatically transform operational JSON data into native Ceeview Service Models.

The illustration below provides a blueprint for the service models you'll build throughout this tutorial. Starting with a single REST API response, the Service Producers organize devices into operational hierarchies, attach CPU and Network monitoring to each device, and ultimately combine multiple location-based service models into a unified business service.

The Service Producers created throughout this tutorial progressively extend the Nordic Retail operational model:

- **Service Producer 1** — Create location/device Service Models with CPU observability.
- **Service Producer 2** — Extend the location/device Service Models with Network observability.
- **Service Producer 3** — Build a unified Nordic Retail business service that combines the individual location/device Service Models.

Each Service Producer builds upon the previous one, demonstrating how REST Monitor progressively transforms operational API data into rich, hierarchical Ceeview Services.

JSON Payload (one record)

JSON Structure

```

root [OBJECT]
├── rows [ARRAY]
│   └── [0] [OBJECT]
│       ├── group = Berger
│       ├── _id = 9Djxd4qBoKpuBWlvjh-7
│       ├── name = ROAC115721T.LOCAL
│       ├── # cpu_total_percentage = 55
│       ├── # cpu_queue_length = 0
│       ├── # network_bytes_sent_per_sec = 825
│       ├── # latency_result_1 = 1
│       ├── hw_model = 103C_5330US HP Engage Flex HP Engage Flex Pro-C
│       ├── hw_cpu_pcore_count = 4
│       ├── _created = 2023-07-13T10:20:06.761Z
│       ├── memory_available_percentage = 21
│       ├── bsods = 0
│       ├── memory_total_bytes_used = 6563311616
│       ├── memory_total_bytes_available = 1789947904
│       ├── device_score = 10000
│       ├── hw_memory_total_bytes = 8589934592
│       ├── tags [ARRAY]
│       └── value [ARRAY]
│           ├── [1] [OBJECT]
│           ├── [2] [OBJECT]
│           ├── [3] [OBJECT]
│           ├── [4] [OBJECT]
│           └── [5] [OBJECT]

```

Location/Device Service Model (one model per location)

NORDIC RETAIL DEVICE STATUS - SKULLERUD

Unified Business Service Model (all locations combined)

NORDIC RETAIL DEVICE STATUS - GROUPED

JSON Field	Operational Meaning
1 group	Retail location / region (e.g., Berger, Asker, Skullerud). Used to create location services.
2 name	Server device name. Used to create device service elements under each location.
3 cpu_total_percentage	Current CPU utilization percentage. Used as a CPU service qualifier.
4 cpu_queue_length	Current CPU queue length. Used as a CPU queue service qualifier.
5 network_bytes_sent_per_sec	Network throughput (bytes sent per second). Used as a Network service qualifier.
6 latency_result_1	Network latency measurement. Used as a Network latency service qualifier.

The remainder of this chapter demonstrates how each part of this blueprint is created using the REST Monitor Service Producer.

3.1 Service Producer 1 – Create Location/Device Service Models with CPU Observability

Step 1: Create Service Producer

In the Profile Designer select New → Producer → Service.

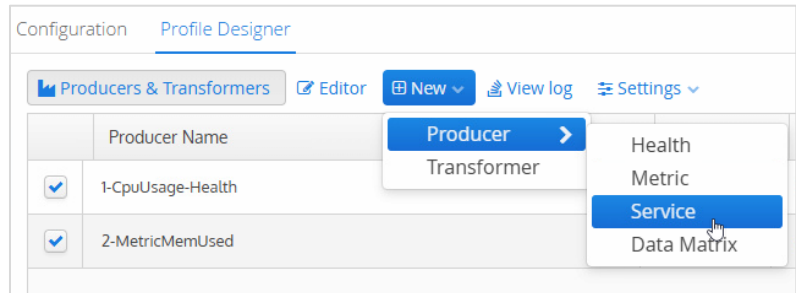


Figure 2 · Selecting New → Producer → Service from the menu.

Step 2: Configure General Properties

Complete the General Properties tab as shown below. The highlighted settings define how the Service Producer creates the service model structure, associates service elements with the Health Assets created in Tutorial 1, and prepares the model for service qualification. Fields that behave the same as in previous tutorials should be configured using the existing values.

Service Producer Setup

1

3-Service-Group-CPU

General Properties

Service Qualifier Builder

Filter Designer

2

Service Name *

Nordic Retail Device Status - \${group}

3

Service Element Name *

stripWorkgroup(\${name})/CPU

4

Service Element: Aggregate Rule

1

5

☐ Enforce Model

Asset Name *

stripWorkgroup(\${name})

Asset Target

\${name}

☒ Case Sensitive Matching

6

Builder Properties

Service Qualifier Fields

unit

name

algorithmClassName

threshold

algorithmProperties

value

Array Iterator Key

root.rows

1

Producer Name

A descriptive name displayed within the REST Monitor. For this tutorial, use "3-Service-Group-CPU".

2

Service Name

Name assigned to the service model and parent node in the service model. The example creates a grouped service name using "Nordic Retail Device Status - \${group}".

3

Service Element Name

Creates the service element path for each device. In this example, the path creates the CPU service element under each device using "stripWorkgroup(\${name})/CPU".

4

Aggregate Rule

Defines how many Service Qualifiers must evaluate as true. For this Service Producer, "1" requires one qualifier. A percentage (for example, 50%) may also be used.

5

Enforce Model

When enabled, REST Monitor enforces the service model structure defined by this producer. Leave this unchecked in this tutorial.

6

Service Qualifier Fields

Defines which fields are available in the Service Qualifier Builder. These fields support qualifier configuration such as name, value, threshold, unit, algorithmClassName, and algorithmProperties.

7

Understanding Expressions

Service Producer expressions combine JSON Variables with Editor Functions to transform REST API fields into meaningful operational Service Models.

JSON Variable

\${group}

Reads a value directly from the REST API JSON payload.

+

Editor Function

stripWorkgroup(\${name})

Transforms (formats) a JSON value before it is used.

Examples used in this tutorial

Service Name

Nordic Retail Device Status - \${group}

↓

Nordic Retail Device Status - Berger

Service Element Name

stripWorkgroup(\${name})/CPU

↓

ROAC115721T/CPU

①

Fields marked with * are required.

①

\$(field) references a value from the REST API JSON payload. function(\$(field)) calls an Editor Function using the JSON value as input.

Figure 3 · Service Producer General Properties. Beyond configuring the producer, this page explains how JSON variables, expressions, and Service Qualifiers combine to transform REST API data into operational Service Models.

Use the following values when configuring the General Properties tab. The table below summarizes each setting, the value used throughout this tutorial, and the purpose of each configuration.

#	Configuration	Tutorial Value	Purpose
1	Producer Name	3-Service-Group-CPU	Display name shown within the REST Monitor.
2	Service Name	Nordic Retail Device Status - \${group}	Creates the parent service node for each location (for example, Nordic Retail Device Status – Bergen).

Drag and edit

Drag group from the JSON Structure into Service Name, then edit the expression.

1 Drag 'group' from JSON into Service Name

Service Model Configuration

Service Name *

Enter a valid service name: A group = Berger

Builder Properties

Service Qualifier Fields

name X value X

Drag
'group'
from
JSON

```
# memory_total_bytes_available = 11
# device_score = 9500
# hw_memory_total_bytes = 2147483
A group = Berger
A id = 9Djxd4qBoKpuBWihj-7
▶ {} [OBJECT]
```

2 Edit the expression in Service Name

Service Model Configuration

Service Name *

Nordic Retail Device Status - \${group}

3	Service Element Name	stripWorkgroup(\${name})/CPU	Creates a CPU service element beneath each device using the cleaned device name.
---	----------------------	------------------------------	--

Drag and edit

Drag name from the JSON Structure into Service Element Name, then apply the Editor function and /CPU path.

1 Drag from JSON

Service Element Name *

Enter a valid service element name: A name = ROAC1157112PLOCAL

Array Iterator Key

Enter path

Drag
'name'
from JSON

```
# bsods = 0
# memory_total_bytes_used = 10191310848
A name = ROAC1157112PLOCAL
# cpu_total_percentage = 55
# memory_total_bytes_available = 11046850560
```

2 Edit the expression

Service Element Name *

stripWorkgroup(\${name})/CPU

4	Service Element Aggregate Rule	1	Applies the default aggregation rule used to evaluate the service element.
5	Enforce Model	Unchecked	Leave disabled in this tutorial.
6	Service Qualifier Fields	unit, name, algorithmClassName, threshold, algorithmProperties, value	Makes these API fields available to the Service Qualifier Builder.

	The remaining General Properties fields are configured the same as in Tutorials 1 and 2 and continue on the next page.		
	Asset Name	stripWorkgroup(\${name})	Uses the same naming logic as the Health Producer so service elements map to the correct Health Assets.
	Asset Target	\${name}	Matches each generated service element to the corresponding Health Asset created in Tutorial 1.
	Case Sensitive Matching	Enabled	Compares asset names using case-sensitive matching.
	Array Iterator Key	root.rows	Iterates through each device returned by the REST API to create the service model.

Next, configure the Service Qualifier Builder to define how each generated CPU service element will be evaluated.

Step 3: Configure Service Qualifiers

In the Service Qualifier Builder tab, define the qualifier rows that will be attached to each generated CPU service element. In this tutorial, two qualifiers are created: **Cpu Usage** and **Cpu Queue Length**.

Each row defines the value to evaluate, the threshold to compare against, the unit of measure, the algorithm used, and any required algorithm properties.

Service Producer Setup

3-Service-Group-CPU

General PropertiesService Qualifier BuilderFilter Designer

Service Qualifier Configuration

1 Name	2 Value	3 Threshold	4 Unit	5 Algorithmclassname	6 Algorithmproperties
Cpu Usage	\${cpu_total_percentage}	80	PERCENT	StaticThresholdAlgorithm	operator,LT
Cpu Queue Length	\${cpu_queue_length}	10	VALUE	StaticThresholdAlgorithm	Algorithmproperties

Drop a JSON field here to add a new row

1 Name

Label shown on the generated service element for each qualifier. In this tutorial, use "Cpu Usage" and "Cpu Queue Length".

2 Value

The JSON value or expression to evaluate. Use `${cpu_total_percentage}` for CPU usage and `${cpu_queue_length}` for CPU queue length.

3 Threshold

The comparison value used by the selected algorithm. Enter **80** for Cpu Usage and **10** for Cpu Queue Length.

4 Unit

Selects the unit used to interpret the qualifier value. The unit list contains many options. In this tutorial, use **PERCENT** for Cpu Usage and **VALUE** for Cpu Queue Length.

5 AlgorithmClassName

Selects the evaluation method. Available options include **StaticThresholdAlgorithm**, **RangeAlgorithm**, **DynamicAlgorithm**, **PercentileAlgorithm**, **DeltaAlgorithm**, and **AvailableAlgorithm**. This tutorial uses **StaticThresholdAlgorithm**.

6 AlgorithmProperties

Supplies additional settings required by some algorithms. For **StaticThresholdAlgorithm**, **AlgorithmProperties** can define comparison behavior. In this tutorial, use **"operator,LT"** as shown.

AlgorithmProperties

Example

operator,LT

StaticThresholdAlgorithm Operators

Code	Meaning
EQ	Equal
NE	Not Equal
GT	Greater Than
GE	Greater Than or Equal
LT	Less Than
LE	Less Than or Equal

These operator codes are used in the **AlgorithmProperties** field when the selected **AlgorithmClassName** is **StaticThresholdAlgorithm**.

Figure 4 · Service Qualifier Builder configuration for CPU service elements.

Step 4: Test the Service Producer Configuration

Click Run for the Service Producer to execute the configuration using the current JSON response. This confirms that the Service Producer can generate the expected service model structure, CPU service elements, and service qualifiers.

Review the Output panel to confirm that service elements and service qualifiers are generated before saving the Producer.

The screenshot displays the Ceeview REST Monitor interface. On the left, a table lists three producers: '1-CpuUsage-Health' (Health), '2-MetricMemUsed' (Metric), and '3-Service-Group-CPU' (Service). The '3-Service-Group-CPU' producer is selected, and its 'Test Producer' button is highlighted with a blue arrow. Below this table is another table for 'Transformer Name', 'Transformer Result', 'Transformer Rules', and 'Actions', with one row for 'cpuSeverity'. On the right, the 'JSON Structure' panel shows a root object with an array of rows. The 'Output' panel shows the execution details for the '3-Service-Group-CPU' producer, including the service model, service element, and service qualifiers for two different CPU metrics.

Figure 5 · Executing the Service Producer generates service elements and service qualifiers from the Nordic Retail REST API.

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

Step 5: Update the REST Monitor

Click Update to save the REST Monitor configuration. The Service Producer is now part of the Nordic Retail Example REST Monitor and will execute each time the monitor runs.

The screenshot shows the Ceeview REST Monitor interface with the 'Update' button highlighted in the top right corner. Below the button, the 'JSON Structure' panel is visible, showing a root object with an array of rows, and a sub-array for the first row.

Figure 6 · Click Update to save the REST Monitor configuration and activate the new Service Producer.

Step 6: Verify the Generated Service Models

After the REST Monitor is updated, Ceeview creates service models from the Service Producer configuration. In this tutorial, the API group field creates one service model per Nordic Retail location: **Bergen**, **Asker**, and **Skullerud**.

Each service model contains the devices returned by the API, with a CPU service element and CPU service qualifiers attached beneath each device.

Open the Service section

From the top navigation, select Service, then open the Service view. This displays the service models available in Ceeview.

If your environment contains many service models, use the search field to filter the list. Enter **nordic** to locate the service models created from the Nordic Retail REST API.

The list shows the three location-based service models created from the API group field: **Bergen**, **Asker**, and **Skullerud**. Their health indicators reflect the CPU service qualifier calculations configured in the previous step.

Service name	Account	SLA	Incident	Coverage	Mode	Current state	Description	Created
Nordic Retail Device Status - Asker	CeeView		0	75	Active	5 hours 23 minutes	Auto-generated service	06-07-2026 09:58:02.364
Nordic Retail Device Status - Bergen	CeeView		0	100	Active	5 hours 23 minutes	Auto-generated service	06-07-2026 09:58:01.699
Nordic Retail Device Status - Skullerud	CeeView		0	100	Active	5 hours 23 minutes	Auto-generated service	06-07-2026 09:58:02.046

Review the Generated Service Model

Select **Nordic Retail Device Status – Bergen** from the filtered service model list to open its detail view.

The Service Details page provides an operational overview of the generated service model. Because the Service Producer creates a native Ceeview Service, the model can immediately leverage platform capabilities such as Service Level Agreements (SLAs), operating periods, maintenance schedules, team ownership, documentation, state overrides, incident management, and service history.

The Summary cards provide an at-a-glance view of the service, while the Service View panel displays the generated topology for the Bergen location. In the next step, you'll open this topology for a closer inspection of the generated service elements and service qualifiers.

Service Details

Summary | Documents | Operating period | Maintenance | State override | Servicetory | Revision

Name: Nordic Retail Device Status - Bergen
Description: Auto-generated service
Account: CeeView
Team: NEW TEAM
Information:
Tags:

Service is Shared: ☐

Time settings:
Time zone: Europe/Berlin
Service Time: 06-07-2026 15:31:53.408

Advanced:
CVID: 10577748-2e6d-4973-ba3e-3a33b4b94607

Service Metrics:
5h 25m (Service is SLA)
0 (Service level agreement)
5 (Service asset)
0 (Assetgroup)
0 (Incidents)

Service history:
Service breaches: 0
Service downtime: 0.00%
1h 2.4% 7d 30d 12h 1d

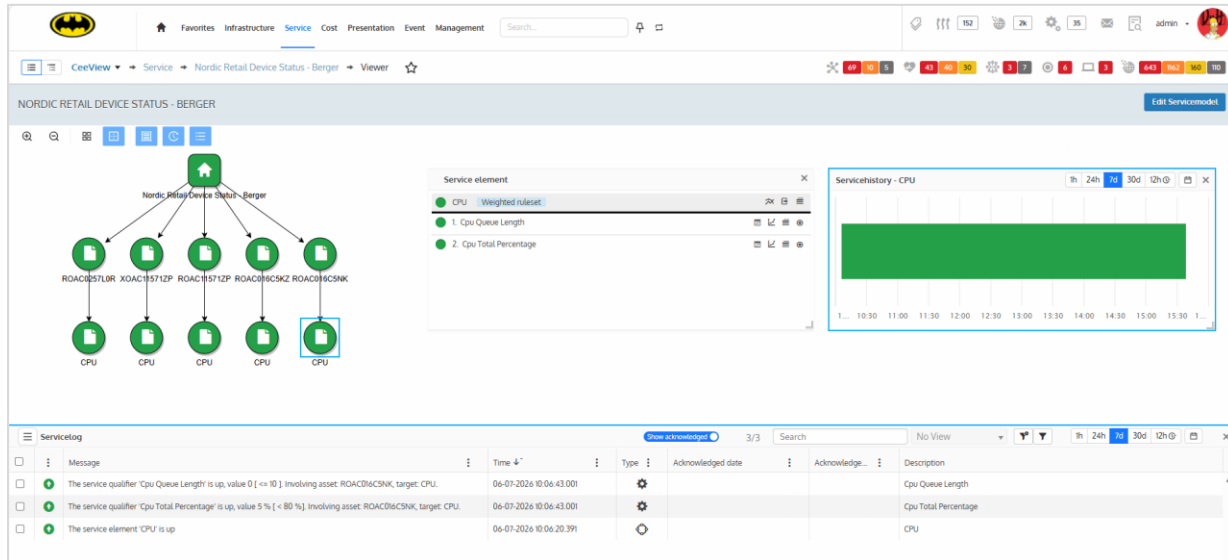
Service view:
Topology diagram showing a central node connected to multiple CPU nodes.

Rootcause:
0/0
There are no root cause for the time period.

Explore the Generated Service Topology

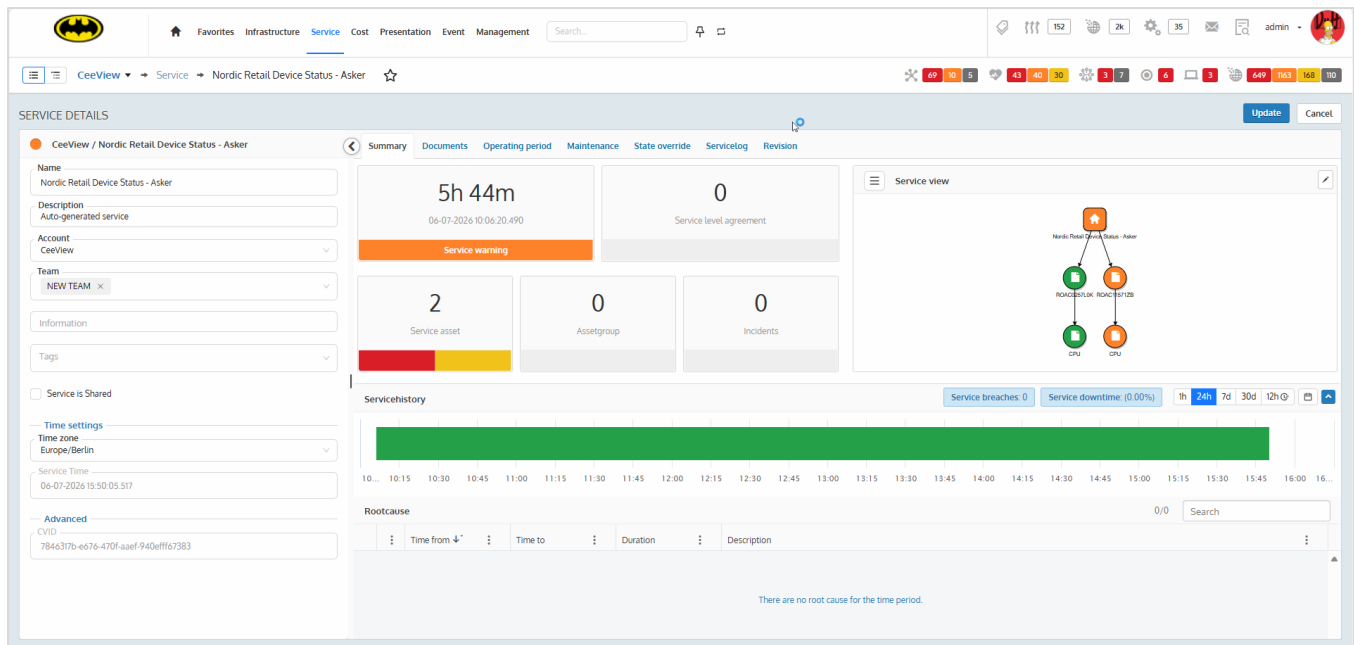
Select the **Service View** panel to open an enlarged interactive view of the generated Service.

The expanded topology provides a focused view of the hierarchy created by the Service Producer, making it easier to inspect the generated service elements and their current health. Selecting a **CPU** service element displays the configured service qualifiers, their current evaluation status, historical trends, and related service events.



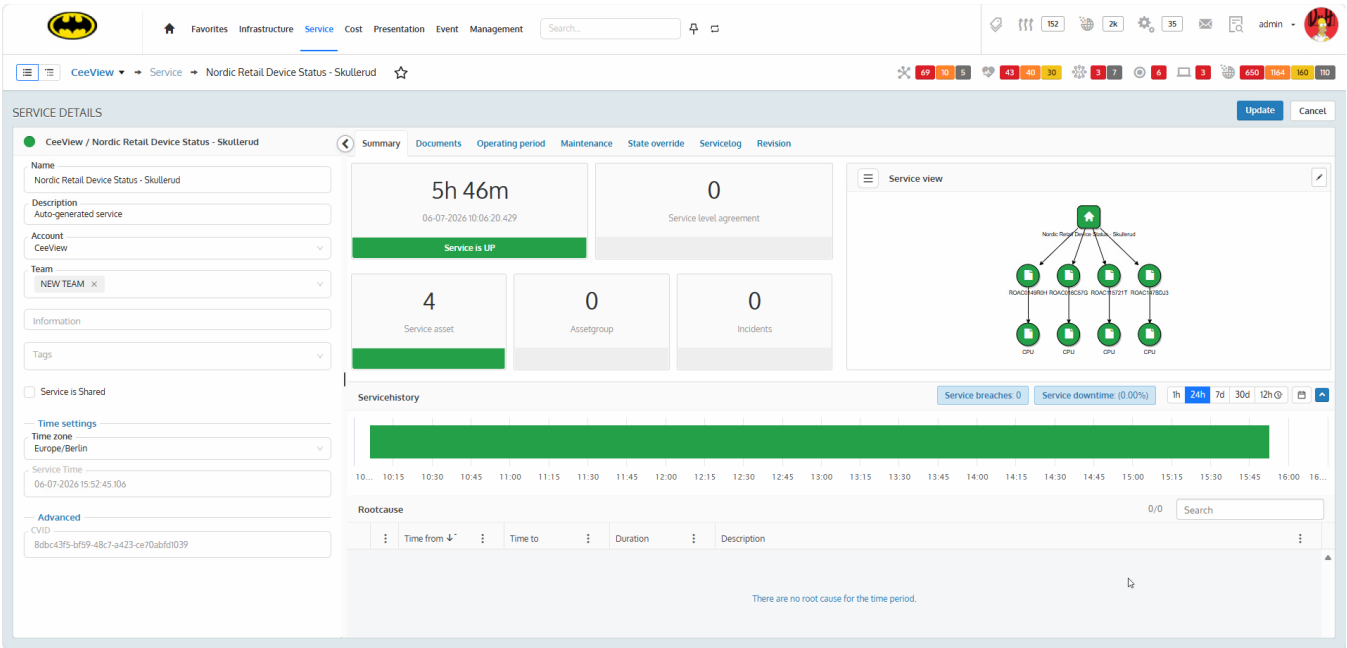
Verify the Asker Service Model

The Asker service model should display the same generated structure as Bergen, including the parent service, devices, CPU service elements, and service qualifiers.



Verify the Skullerud Service Model

Likewise, the **Skullerud** service model should contain the same generated hierarchy and CPU service qualifiers, confirming that the Service Producer created a consistent service model for every location returned by the REST API.



At this point, the REST Monitor is fully operational. Each execution interval retrieves the latest JSON data, maintains the generated Health Assets, records new CPU Metrics, refreshes the service models, and reevaluates the service qualifiers. Together, these producers continuously synchronize the Nordic Retail operational model with the latest REST API data.

Step 7: Confirm the Updated REST Monitor

Return to the Nordic Retail Example REST Monitor. The Profile Designer now contains the original Health Producer, the Metric Producer, and the new CPU-focused Service Producer.

The REST Monitor will execute these producers each time it runs, updating Health Assets, time-series Metrics, and the generated CPU service models from the same REST API response.

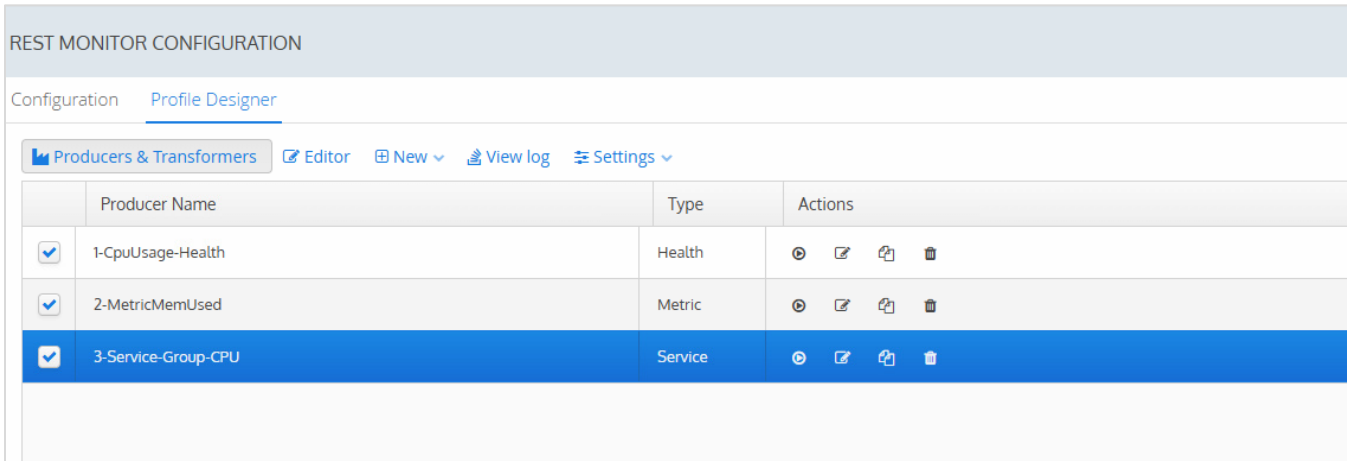


Figure 8 · Nordic Retail Example REST Monitor with Health, Metric, and Location/Device/CPU Service Producers configured.

3.2 Service Producer 2 – Extend Location/Device Service Models with Network Observability

The first Service Producer created three location/device Service models that monitor **CPU** utilization across the Nordic Retail environment.

This second Service Producer extends those existing Service models by adding **Network** service elements and service qualifiers beneath each device. Rather than creating new Service models, this producer enhances the existing hierarchy using the same REST API response.

This demonstrates how additional operational resources can be layered onto existing Service models without rebuilding them, allowing operational visibility to evolve incrementally as monitoring requirements grow.

Step 1: Create Service Producer

In the Profile Designer select New → Producer → Service.

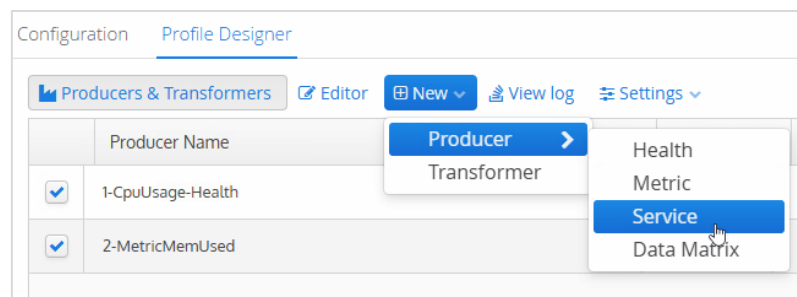


Figure 2 · Selecting New → Producer → Service from the menu.

Step 2: Configure General Properties

Most General Properties remain identical to **Service Producer 1**. Update the highlighted settings below to create **Network** service elements while retaining the existing location/device Service model structure.

The screenshot shows the 'Service Producer Setup' dialog box with the 'General Properties' tab selected. The 'Service Model Configuration' section contains three fields: 'Service Name' with the value 'Nordic Retail Device Status - \${group}', 'Service Element Name' with the value 'stripWorkgroup(\${name})/Network', and 'Service Element Aggregate Rule' with the value '50%'. The 'Builder Properties' section contains 'Service Qualifier Fields' with a list of fields: 'name', 'value', 'unit', 'threshold', 'algorithmClassName', and 'algorithmProperties'. The 'Array Iterator Key' field has the value 'root.rows'.

Step 3: Configure Service Qualifiers

In the Service Qualifier Builder tab, define the qualifier rows that will be attached to each generated Network service element. Configure two Network service qualifiers that complement the existing CPU qualifiers created in Service Producer 1.

Each row defines the value to evaluate, the threshold to compare against, the unit of measure, the algorithm used, and any required algorithm properties.

The screenshot shows the 'Service Qualifier Builder' tab for '4-Service-Group-Network'. It displays a table with two rows of qualifiers:

Name	Value	Threshold	Unit	AlgorithmClassname	AlgorithmProperties
Bytes sent per sec	#{network_bytes_sent_per_sec}	Threshold	BYTE_PER_SECOND	AvailableAlgorithm	AlgorithmProperties
Latency	#{latency_result_1}	5	VALUE	StaticThresholdAlgo	operator,LT

Below the table is a dashed box with the text: 'Drop a JSON field here to add a new row'.

On the right, the 'JSON Structure' panel shows a snippet of the JSON response:

```

{
  "rows": [
    {
      "dc_name": "dc2",
      "network_bytes_received_per_sec": 1280,
      "cpu_queue_length": 0,
      "hw_model": "103C_5330US HP Enclave Flex HP Enclave",
      "network_bytes_sent_per_sec": 346,
      "latency_result_1": 7
    }
  ],
  "tags": [
    {
      "value": [
        {
          "hw_cpu_pcore_count": 4,
          "_created": "2023-06-01T17:11:39.450Z"
        }
      ]
    }
  ]
}

```

Step 4: Test the Service Producer Configuration

Click Run for the Service Producer to execute the configuration using the current JSON response. This confirms that the Service Producer can generate the expected service model structure, Network service elements, and service qualifiers.

Review the Output panel to confirm that service elements and service qualifiers are generated before saving the Producer.

The screenshot shows the 'Profile Designer' tab. In the 'Producers & Transformers' table, the '4-Service-Group-Network' producer is selected, and the 'Test Producer' button is highlighted with a blue arrow.

The 'Output' panel on the right shows the results of the test:

```

Executing SERVICE producer: 4-Service-Group-Network
Executing configuration at 2024-07-07T00:40:35.395
Total JSON Array items: 11, added: 11

ServiceModel: Nordic Retail Device Status - Berger
ServiceElement: ROACH1571ZP/Network
ServiceQualifiers: 2
Name      Value      Unit      Threshold  AlgorithmClassName  AlgorithmProperties Asset
Bytes sent per sec 346      BYTE_PER_SECOND  5          AvailableAlgorithm   StaticThresholdAlgorithm(operator=LT) ROACH1571ZP
Latency          7        VALUE          5          StaticThresholdAlgo  operator=LT          ROACH1571ZP

ServiceElement: XOACH1571ZP/Network
ServiceQualifiers: 2
Name      Value      Unit      Threshold  AlgorithmClassName  AlgorithmProperties Asset
Bytes sent per sec 2342     BYTE_PER_SECOND  5          AvailableAlgorithm   StaticThresholdAlgorithm(operator=LT) XOACH1571ZP
Latency          7        VALUE          5          StaticThresholdAlgo  operator=LT          XOACH1571ZP

ServiceElement: ROACH016C5KZ/Network
ServiceQualifiers: 2

```

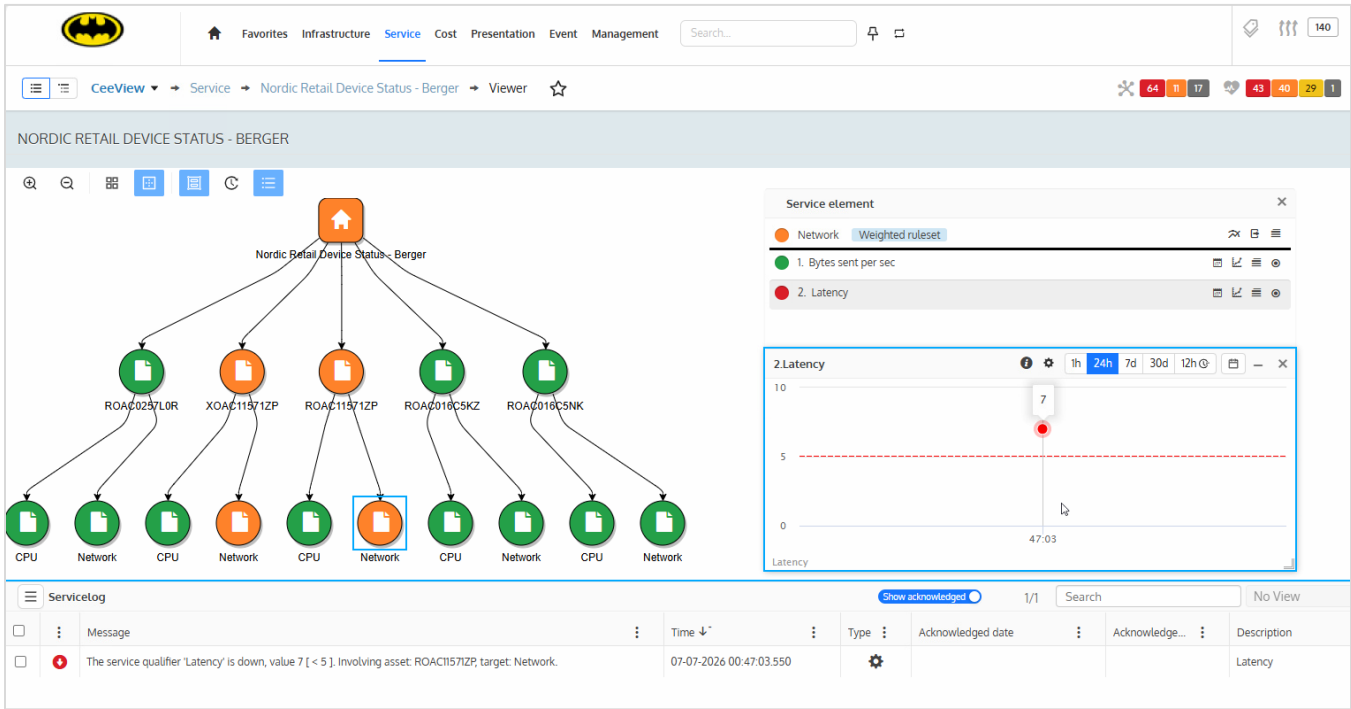
Step 5: Update the REST Monitor

Click Update to save the REST Monitor configuration. The Service Producer is now part of the Nordic Retail Example REST Monitor and will execute each time the monitor runs.

Step 6: Verify the Extended Service Models

After the REST Monitor is updated, Ceeview creates service models from the Service Producer configuration. In this tutorial, the API group field creates one service model per Nordic Retail location: **Bergen**, **Asker**, and **Skullerud**.

Each location/device Service model now contains both **CPU** and **Network** service elements beneath every device. Together, these service elements provide complementary operational visibility while remaining part of the same generated Service model.



Step 7: Confirm the Updated REST Monitor

Return to the Nordic Retail Example REST Monitor. The Profile Designer now contains the original Health Producer, the Metric Producer, and the new Network-focused Service Producer. The REST Monitor now contains two Service Producers that work together to generate and maintain the same location/device Service models with both CPU and Network observability.

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	

3.3 Service Producer 3 – Build a Unified Nordic Retail business Service

Unlike the previous Service Producers, this section creates a new business-level Service hierarchy. Two additional Service Producers are configured—one for CPU observability and one for Network observability—to generate a single unified Service model representing the entire Nordic Retail environment.

Because CPU and Network service elements require independent Service Producers, two additional Producers are configured using the same techniques introduced earlier in this tutorial. Only the Service Name and Service Element Name expressions change to create a unified business hierarchy instead of independent location-based models.

Step 1: Create Service Producer

In the Profile Designer select New → Producer → Service.

Step 2: Configure General Properties

Configure two additional Service Producers using the same approach developed in Sections 3.1 and 3.2.

Only the following settings change:

- Producer Name
- Service Name
- Service Element Name

All remaining General Properties are configured identically to the previous Service Producers.

5-Service-CPU

General Properties | Service Qualifier Builder | Filter Design

Service Model Configuration

Service Name *

Nordic Retail Device Status - Grouped

Service Element Name *

\$(group)/stripWorkgroup(\$(name))/CPU

Service Element Aggregate Rule

1

6-Service-Network

General Properties | Service Qualifier Builder | Filter Design

Service Model Configuration

Service Name *

Nordic Retail Device Status - Grouped

Service Element Name *

\$(group)/stripWorkgroup(\$(name))/Network

Service Element Aggregate Rule

50%

Step 3: Configure Service Qualifiers

Configure the CPU and Network Service Qualifiers using the same settings from Service Producers 1 and 2.

5-Service-CPU

General Properties [Service Qualifier Builder](#) Filter Designer

Service Qualifier Configuration

Name	Value	Threshold	Unit	Algorithmclassname	Algorithmproperties
Cpu Usage	<code>\$(cpu_total_percentage)</code>	80	PERCENT	StaticThresholdAlgo	operator,LT
Cpu Queue Length	<code>\$(cpu_queue_length)</code>	10	VALUE	StaticThresholdAlgo	Algorithmproperties

Drop a JSON field here to add a new row

6-Service-Network

General Properties [Service Qualifier Builder](#) Filter Designer

Service Qualifier Configuration

Name	Value	Threshold	Unit	Algorithmclassname	Algorithmproperties
Bytes sent per sec	<code>\$(network_bytes_sent_p)</code>	Threshold	BYTE_PER_SECOND	AvailableAlgorithm	Algorithmproperties
Latency	<code>\$(latency_result_1)</code>	5	VALUE	StaticThresholdAlgo	operator,LT

Drop a JSON field here to add a new row

Step 4: Test the Service Producer Configuration

Test both the Unified CPU and Unified Network Service Producers using the current JSON response. As in the previous Service Producers, the Output panel confirms that the configuration executes successfully and generates the expected service model.

Unlike the previous Service Producers, these configurations generate a single unified Nordic Retail business service containing the three retail locations beneath one parent service, while preserving the CPU and Network service elements created for each device.

Review the Output panel after each test to confirm that both Service Producers complete successfully before saving the configurations.

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	

Test Producer

JSON Structure

Selected: root.rows.0.device_score

Output

Executing SERVICE producer: 5-Service-CPU

Executing configuration at 2026-07-07T01:30:15.400

Total JSON Array items: 11, added: 11

ServiceModel: Nordic Retail Device Status - Grouped

ServiceElement: Berger/ROAC1571ZP/CPU

ServiceQualifiers: 2

Name	Value	Unit	Threshold	AlgorithmClassName	AlgorithmProperties	Asset
Cpu Usage	55	PERCENT	80	StaticThresholdAlgorithm	[operator=LT]	ROAC1571ZP
Cpu Queue Length	0	VALUE	10	StaticThresholdAlgorithm		ROAC1571ZP

ServiceElement: Berger/XOAC1571ZP/CPU

ServiceQualifiers: 2

Name	Value	Unit	Threshold	AlgorithmClassName	AlgorithmProperties	Asset
Cpu Usage	6	PERCENT	80	StaticThresholdAlgorithm	[operator=LT]	XOAC1571ZP
Cpu Queue Length	0	VALUE	10	StaticThresholdAlgorithm		XOAC1571ZP

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	

Test Producer

JSON Structure

rows [ARRAY]

[0] [OBJECT]

Output

Executing SERVICE producer: 6-Service-Network

Executing configuration at 2026-07-07T01:21:54.805

Total JSON Array items: 11, added: 11

ServiceModel: Nordic Retail Device Status - Grouped

ServiceElement: \$[group]/ROAC1571ZP/Network

ServiceQualifiers: 2

Name	Value	Unit	Threshold	AlgorithmClassName	AlgorithmProperties	Asset
Bytes sent per sec	346	BYTE_PER_SECOND	80	StaticThresholdAlgorithm	[operator=LT]	ROAC1571ZP
Latency	7	VALUE	5	StaticThresholdAlgorithm	[operator=LT]	ROAC1571ZP

ServiceElement: \$[group]/XOAC1571ZP/Network

ServiceQualifiers: 2

Name	Value	Unit	Threshold	AlgorithmClassName	AlgorithmProperties	Asset
Bytes sent per sec	2342	BYTE_PER_SECOND	80	AvailableAlgorithm		ROAC1571ZP
Latency	7	VALUE	5	StaticThresholdAlgorithm	[operator=LT]	XOAC1571ZP

ServiceElement: \$[group]/ROAC16C5KZ/Network

ServiceQualifiers: 2

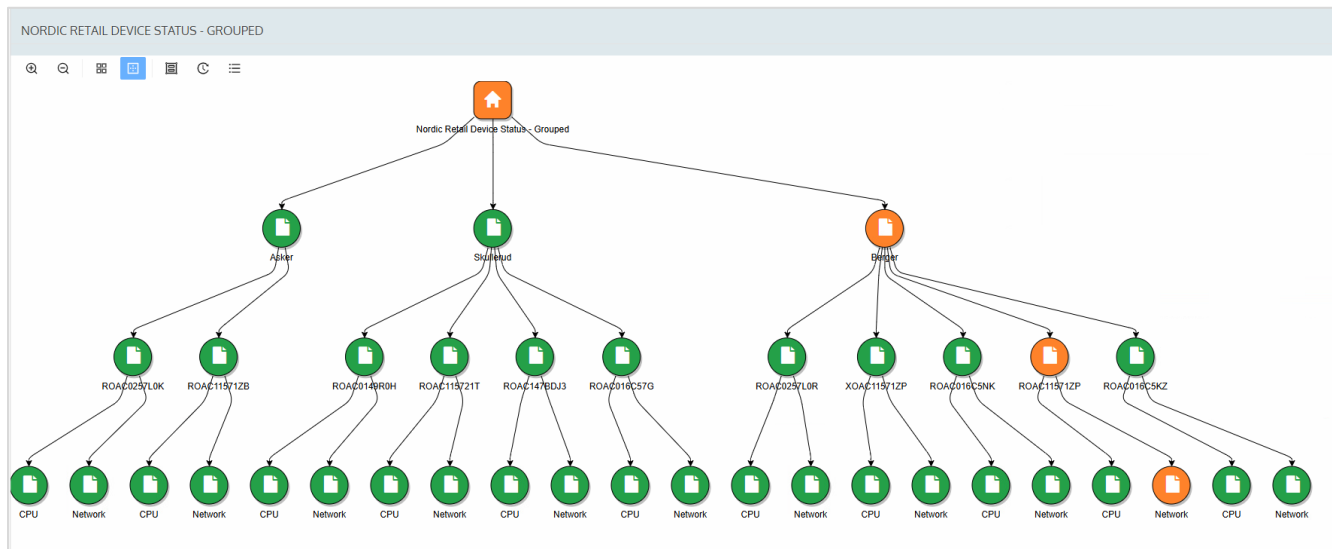
Name	Value	Unit	Threshold	AlgorithmClassName	AlgorithmProperties	Asset
Bytes sent per sec	1326	BYTE_PER_SECOND	80	AvailableAlgorithm		ROAC16C5KZ
Latency	1	VALUE	5	StaticThresholdAlgorithm	[operator=LT]	ROAC16C5KZ

Step 5: Update the REST Monitor

Click Update to save the REST Monitor configuration and activate the Unified CPU and Unified Network Service Producers. During each execution interval, the REST Monitor now generates and maintains the unified Nordic Retail business service alongside the previously created location/device service models.

Step 6: Verify the Generated Service Model

The unified Nordic Retail business Service now contains the three retail locations beneath a single parent Service. Each location includes the generated device hierarchy with both CPU and Network service elements attached beneath every device.



Step 7: Confirm the Updated REST Monitor

Return to the Nordic Retail Example REST Monitor. The Profile Designer now contains six Producers working together:

- Health Producer
- Metric Producer
- Location/Device CPU Service Producer
- Location/Device Network Service Producer
- Unified Business CPU Service Producer
- Unified Business Network Service Producer

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	

Continue the REST Monitor Learning Series

Congratulations—you've now transformed a single REST API into a rich operational model within Ceeview.

Across the first three tutorials, you've progressively extended the same REST Monitor by creating Health Assets, Metrics, and native Service Models. Together, these Producers demonstrate how a single JSON response can generate multiple operational views of the same environment without writing code.

 **KEY TAKEAWAYS**

After completing this tutorial, you now know how to:

-  **Create native Ceeview Service Models**
Automatically transform REST API data into hierarchical Service Models that become first-class Ceeview objects.
-  **Layer multiple operational views**
Use multiple Service Producers to build different operational perspectives from the same REST API without duplicating data collection.
-  **Extend Service Models incrementally**
Add new monitored resources—such as CPU and Network—to existing Service Models while preserving the overall operational structure.
-  **Build business-level operational services**
Compose multiple location-based Service Models into a single business service that reflects the operational hierarchy of the environment.
-  **Reuse the same REST API**
Generate Health Assets, Metrics, Service Models, and additional object types from one continuously monitored REST API.

 **One REST API can generate multiple native Ceeview object types, enabling progressively richer operational visibility without additional data collection.**

 **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
Next

The Nordic Retail Example REST Monitor now contains Health, Metric, and Service Producers working together from the same REST API. Health Producers create operational assets, Metric Producers record historical performance, and Service Producers organize those assets into hierarchical operational services.

In the next tutorial, you'll complete the operational model by adding a **Data Matrix Producer**, enabling REST API data to be presented in rich tabular views for dashboards, reporting, filtering, and operational analysis.