

Juniper Apstra 5.0.1 / 5.0.0 Custom Telemetry Collection Guide

Published
2025-08-08

RELEASE

Table of Contents

Introduction

Apstra Telemetry and Intent-Based Analytics

Custom Telemetry Collection Overview

Create a Custom Telemetry Collector

Step 1. Execute the CLI Command | 9

Step 2. Identify the Keys and Values of Interest from the CLI Output | 12

Step 3. Create a Service Schema | 13

Step 4. Create a Telemetry Collector | 15

Use Custom Telemetry Data in an IBA Probe

Types of Processors | 22

Create a Probe | 24

Configure Additional Processors for Anomaly Generation | 29

Verify Your Configuration | 51

Monitor the Health of the Telemetry Service

Import and Export Services

Export and Import a Service from the Service Registry | 55

Exporting and Importing a Service from the Apstra Flow Dashboard (Recommended) | 57

Summary

Introduction

Juniper Apstra offers robust automation for data center switching fabrics management. Apstra's Intent-Based Networking (IBN) aids in network creation, deployment, operation, and validation.

Apstra validates that:

- User-supplied inputs are valid.
- User inputs align with network constraints.
- Expected telemetry outputs are correct in a stable network.
- There is no discrepancy between expected and actual telemetry.

After deploying your network, Apstra collects and validates telemetry data from managed devices. This data is automatically aggregated and cross-checked against the intended state of each telemetry type, such as interfaces, LLDP, and BGP. This capability, known as Intent-Based Analytics (IBA), provides accurate and relevant data for efficient operations and informed decision-making.

Starting with Release 4.2.0, Apstra introduces its *Custom Telemetry Collection*. This collection enables you to easily configure Apstra to collect new telemetry data from managed devices. Apstra then uses that data in IBA probes to automate knowledge extraction from this raw data through analysis and visualization.

This document guides you on IBA fundamentals and creating a custom telemetry service. You'll also learn how to create a new IBA probe for data visualization and analysis. This guide includes a use case example that shows how to:

- Define a telemetry service for power metrics collection from managed devices.
- Create an IBA probe to monitor, detect, and visualize power consumption.
- Store anomaly history in a time-series database.

Let's get started!

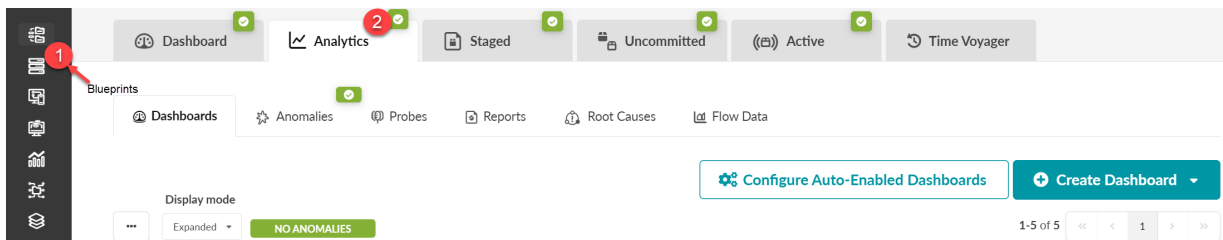
Apstra Telemetry and Intent-Based Analytics

IN THIS SECTION

- [What Is Intent-Based Analytics? | 2](#)
- [Telemetry Services | 3](#)
- [Auto-Enabled Probes | 4](#)
- [Predefined Probes Catalog | 4](#)
- [Custom Probes | 6](#)
- [What If Apstra Doesn't Collect the Data You're Looking For? | 7](#)

What Is Intent-Based Analytics?

Intent-Based Analytics (IBA) extracts knowledge from raw telemetry data, assisting you in monitoring operational status changes in your infrastructure. To configure IBA, select **Blueprints** from the left navigation menu in the Apstra GUI. Select your blueprint and navigate to the **Analytics** tab.



NOTE: You can also expand the icons to see the "names" for each category by clicking the left arrow icon at the bottom of the left navigation menu.

IBA Probes

An IBA probe is a real-time analytics pipeline that enables you to set up conditions of interest (situations to monitor). An IBA probe retrieves data, processes it, and then compares the result against expectations.

IBA probes:

- Collect different types of telemetry data from managed devices.
- Enrich the data with contextual information from the blueprint.
- Aggregate and process the raw data into more meaningful data such as average over time, state in time, standard deviation, and so forth.
- Raise anomalies when the network deviates from an intended state and streams the anomalies as alerts to external systems, as necessary.

Probes are available as either predefined probes or user-defined (custom) probes. When you deploy a blueprint, some predefined probes are enabled automatically. You can enable other predefined probes on-demand from a catalog, as described in the ["Predefined Probes Catalog" on page 4](#).

Telemetry Services

You can view a list of telemetry services currently activated in your Apstra deployment. Each service represents a different type of data Apstra collects from your managed devices. For each telemetry service, Apstra issues different CLI show commands over the device API to ingest the data utilized in IBA.

To view the available telemetry services in the Apstra GUI, from the left navigation menu, select **Analytics > Telemetry > Services**

The screenshot shows the Apstra GUI interface. On the left, a dark navigation sidebar contains various icons. A red arrow points to the 'Analytics' icon, which has opened a dropdown menu. In this menu, 'Telemetry' is selected, and 'Services' is highlighted with a red box. The main content area of the GUI shows the 'Analytics > Services' page. It features a breadcrumb trail at the top: '☆ > Analytics > Services'. Below this, there is a grid of service cards. Each card represents a different telemetry service and displays the following information:

- Service Name:** ARP, BGP, DISK UTIL, LLDP, MAC.
- Configured on:** 5 devices (for all services shown).
- Errors during enabling:** 0 devices.
- Last collection cycle errors:** 0 devices.
- Used by collectors:** 0 collectors.

The 'LLDP' and 'MAC' services also show '2 devices' for the 'Configured on' status.



NOTE: The raw data that Apstra collects does not appear on the Telemetry Services page. The raw data is only shown and visualized in IBA probes.

Auto-Enabled Probes

When you deploy a blueprint, several IBA probes are automatically enabled. IBA probes are used to monitor essential information about the managed fabric and generate anomalies when it detects degradations in the device health or fabric performance.

To view all existing probes for your blueprint, navigate to the **Analytics** dashboard, then click the **Probes** tab. For example:

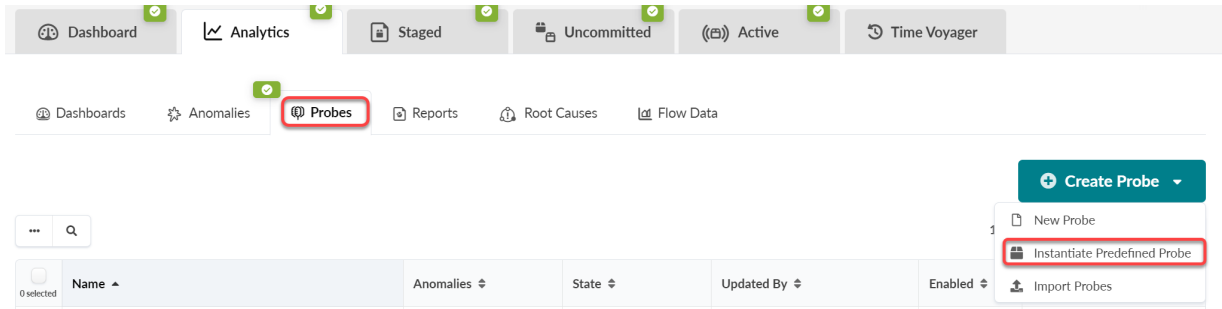
<input type="checkbox"/> 0 selected	Name ▲	Anomalies ▼	State ▼
<input type="checkbox"/>	Device System Health	<input checked="" type="checkbox"/> No anomalies	<input checked="" type="checkbox"/> Operational
<input type="checkbox"/>	Device Telemetry Health	<input checked="" type="checkbox"/> No anomalies	<input checked="" type="checkbox"/> Operational
<input type="checkbox"/>	Device Traffic	<input checked="" type="checkbox"/> No anomalies	<input checked="" type="checkbox"/> Operational
<input type="checkbox"/>	ECMP Imbalance (Fabric Interfaces)	<input checked="" type="checkbox"/> No anomalies	<input checked="" type="checkbox"/> Operational
<input type="checkbox"/>	ESI Imbalance	<input checked="" type="checkbox"/> No anomalies	<input checked="" type="checkbox"/> Operational
<input type="checkbox"/>	LAG Imbalance	<input checked="" type="checkbox"/> No anomalies	<input checked="" type="checkbox"/> Operational

Predefined Probes Catalog

In addition to auto-enabled probes, you can select predefined probes from a built-in catalog and enable these probes based on your monitoring requirements.

Some predefined probes (such as EVPN or Optical Transceivers probes) activate additional services. These probes collect the necessary data from the devices and ingest the data into the probe for analysis.

To access the list of predefined probes, navigate to the **Analytics** dashboard, then click the **Probes** tab. Select **Instantiate Predefined Probe** from the **Create Probe** drop-down to see the list of probes.



Instantiate Predefined Probe

Predefined Probe *

- EVPN Host Flapping
- EVPN Host Flapping**
- EVPN VXLAN Type-3 Route Validation
- EVPN VXLAN Type-5 Route Validation
- External Routes
- Hot/Cold Interface Counters (Fabric Interfaces)
- Hot/Cold Interface Counters (Specific Interfaces)

If MAC address is suppressed for more than or equal to percentage of Anomaly Time Window, an anomaly will be raised.

Collection period

2 Minutes

Controls how often flapping MAC addresses will be collected on devices.

☒ **Enable flapping hosts history**

If enabled, probe will keep history of which leaf suppresses flapping MAC addresses and which specific addresses were suppressed.

History retention period

7 Days

Duration to maintain flapping MAC addresses historical data.

On every leaf probe monitors MAC addresses that are being learned alternately from local and VTEP interfaces more often than it is allowed by constraints configured in the system.

☐ Create Another? **Create**

For more information about predefined probes, see [Predefined Probes](#) in the Juniper Apstra User Guide.

Custom Probes

If you have a monitoring use case not addressed by any of the default or predefined probes, you must create a new custom probe. To create a new probe, from the **Analytics** dashboard, select **Probes > Create New Probe**.

The screenshot shows the 'Probes > New Probe' form in the Analytics dashboard. At the top, there is a navigation bar with five tabs: 'Dashboards', 'Anomalies', 'Widgets', 'Probes' (which is selected and highlighted), and 'Reports'. Below the navigation bar, the breadcrumb 'Probes > New Probe' is displayed. The form contains two input fields: 'Name' with a red asterisk indicating it is required, containing the text 'My Probe', and 'Description', which is currently empty. Below these fields, there is a dashed box containing a plus icon and the text 'Add Processor'. To the right of this box is a teal button with an upward arrow icon and the text 'Import Probe'. Above the 'Import Probe' button, there is a small circular arrow icon and the text 'Start creation of a new probe by adding a processor.'

For the probe to be functional, you'll need to add at least one processor. A processor adds data to your probe from one of the existing telemetry services. A pipeline starts when the processor(s) injects the raw data into the pipeline. The raw data is then sent to the processor. For descriptions of the available processors, see ["Types of Processors" on page 22](#).

Here is an example of a source processor, whose processor type is **Interface Counters**.

Add Processor

Processor Type *
 Interface Counters

Processor Name *
 Interface Counters

Output Stage Name: out *
 Interface Counters

Interface Counters Processor.
Selects interfaces according to the configuration and outputs counter stats of the specified types (e.g. 'tx_bytes').
Has no inputs.

Add

Probes ▶ My Probe ✓ Operational ✓ No anomalies admin a few seconds ago Enabled ON

Search stages...

➤

13³₄ Interface Counters

Interface Counters

Stage: Interface Counters

Query: All

System ID ⓘ	Interface ⓘ	Value ⓘ
5254001BFC0D spine2 Spine	ge-0/0/0	1
5254001BFC0D spine2 Spine	ge-0/0/1	2
5254001BFC0D spine2 Spine	ge-0/0/2	1

What If Apstra Doesn't Collect the Data You're Looking For?

If Apstra did not collect the data you want to monitor, we recommend that you use Apstra's Custom Telemetry Collection feature. To learn about this feature, proceed to the next section "[Custom Telemetry Collection Overview](#)" on page 8.

Custom Telemetry Collection Overview

IN THIS SECTION

- [Example Use Cases | 8](#)

Juniper Apstra Custom Telemetry Collection is a new feature introduced in Apstra 4.2.0. With this feature, you can define new telemetry services for data monitoring and analysis in Apstra. It also allows you to customize analytics according to your business needs. This process simplifies the previously complex task of adding a telemetry service, which required advanced programming and knowledge of the telemetry SDK.

With the custom telemetry collection, you can do the following:

- Run the Junos CLI show commands that provide you with the data you want analyzed.
- Identify the specific key and value to extract from the show command based on its XML output.
- Create a telemetry collector definition.

Example Use Cases

Here are some examples of what you can do with the custom telemetry collection. This list isn't exhaustive. Any show command on a switch can serve as an example.

- Monitor various counters (firewall filter match count, IRB interface statistics, and so on).
- Monitor device power usage and thermal output for capacity planning.
- Monitor device health (line card status or other environmental statuses).
- Monitor protocol status or features enabled with configlets (BFD, MACsec, QoS, multicast, OSPF, RPM and so on).
- Monitor power usage and thermal output for capacity planning.

In the following sections, we'll walk you through the end-to-end workflow of creating your own custom telemetry service. In this walkthrough, we'll monitor a device's power usage as an example.

Let's go!

Create a Custom Telemetry Collector

SUMMARY

This topic describes the steps required to create a custom telemetry collector.

IN THIS SECTION

- [Step 1. Execute the CLI Command | 9](#)
- [Step 2. Identify the Keys and Values of Interest from the CLI Output | 12](#)
- [Step 3. Create a Service Schema | 13](#)
- [Step 4. Create a Telemetry Collector | 15](#)

In this topic, we'll walk you through creating your own custom telemetry service using power-related metrics as an example.

Step 1. Execute the CLI Command

Starting in Apstra version 4.2.0, you can run CLI show commands for Junos devices directly from the Apstra GUI. Although you can run show commands without opening a CLI session, its primary purpose is to help you create your own custom telemetry collectors.

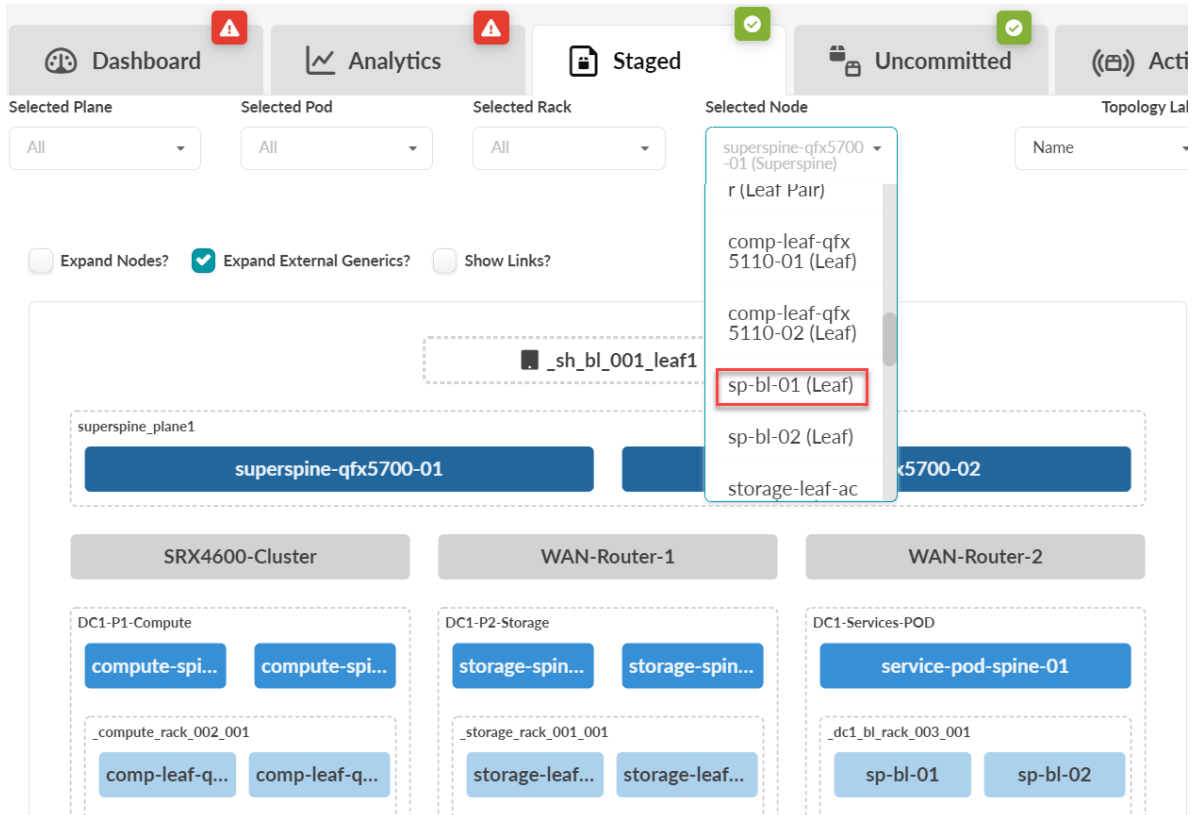
You can execute CLI commands from within a staged or active blueprint (shown in our example), or from the **Devices > Managed Devices** page.



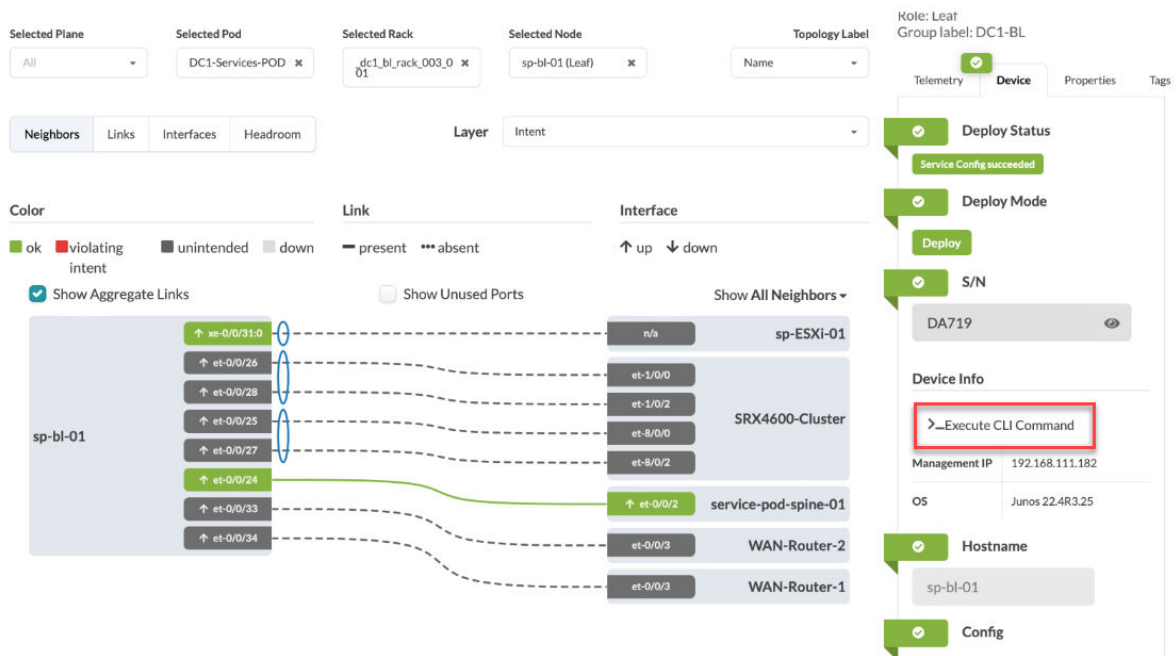
NOTE: Before you get started, identify the CLI command that you want to use for the output values. In this procedure, we're using the `show chassis power` command as an example.

To execute the CLI command:

1. From your deployed blueprint, select **Staged > Physical > Topology**, then select your Juniper device node. In this example, sp-bl-01 (Leaf).



2. In the **Selection** section that appears in the right panel, on the **Device** tab, select **Execute CLI Command**.



In the dialog box that opens, select how you want to view the results: Text Mode, XML Mode, or JSON mode.

Execute CLI Command

S/N: DA719 Management IP: 192.168.111.182 Hostname: sp-bl-01

Text Mode
Text Mode
XML Mode
JSON Mode

Execute

Only "show" commands are supported



NOTE: The CLI supports only Junos show commands. You cannot run commands that affect the device state, such as request system reboot. For information about the various show commands, see the [CLI User Guide for Junos OS](#).

Here is an example of the CLI output in **Text Mode**.

Execute CLI Command

S/N: YR3624060033 Management IP: 10.28.77.7 Hostname: spine1

Text Mode

Execute

Chassis Power	Voltage(V)	Power(W)
Total Input Power		232
PSM 0		
State: Online		
Input 1	202	232
Output	11.96	212.78
PSM 1		
State: Offline		
Input 1	0	0
Output	0	0
System:		
Zone 0:		
Capacity:	1600 W (maximum 1600 W)	
Allocated power:	1241 W (359 W remaining)	
Actual usage:	232 W	
Total system capacity:	1600 W (maximum 1600 W)	
Total remaining power:	359 W	

Here is an example of the CLI output in **XML Mode**.

```

<rpc-reply xmlns:junos="http://xml.juniper.net/junos/23.4R2.14-EV0/junos">
  <power-usage-information>
    <power-usage-voltage-total>
      <dc-power>233</dc-power>
    </power-usage-voltage-total>
    <power-usage-psm-item1>
      <slot>0</slot>
      <state>Online</state>
      <power-usage-psm-input1>
        <name>Input 1</name>
        <dc-voltage>202</dc-voltage>
        <dc-power>233</dc-power>
      </power-usage-psm-input1>
      <power-usage-psm-output>
        <name>Output</name>
        <dc-voltage>11.96</dc-voltage>
        <dc-power>214.28</dc-power>
      </power-usage-psm-output>
    </power-usage-psm-item1>
    <power-usage-psm-item1>
      <slot>1</slot>
      <state>Offline</state>
      <power-usage-psm-input1>
        <name>Input 1</name>
        <dc-voltage>0</dc-voltage>
        <dc-power>0</dc-power>
      </power-usage-psm-input1>
      <power-usage-psm-output>
        <name>Output</name>
        <dc-voltage>0</dc-voltage>
        <dc-power>0</dc-power>
      </power-usage-psm-output>
    </power-usage-psm-item1>
    <power-usage-system>
      <power-usage-zone-information>
        <zone>0</zone>
        <capacity-actual>1600</capacity-actual>
        <capacity-max>1600</capacity-max>
        <capacity-allocated>1241</capacity-allocated>
        <capacity-remaining>359</capacity-remaining>
        <capacity-actual-usage>233</capacity-actual-usage>
      </power-usage-zone-information>
      <power-usage-system-total>
        <capacity-sys-actual>1600</capacity-sys-actual>
        <capacity-sys-max>1600</capacity-sys-max>
        <capacity-sys-remaining>359</capacity-sys-remaining>
      </power-usage-system-total>
    </power-usage-system>
  </power-usage-information>
  <cli>
    <banner></banner>
  </cli>
</rpc-reply>

```

In this example, the output shows the power usage information for the device. This power information is what we'll use to create our telemetry collection service.

Step 2. Identify the Keys and Values of Interest from the CLI Output

The following steps show you how to use the CLI show command to view the power utilization for your devices.

1. Enter the CLI show command (in this example, show chassis power).
2. Click **Execute** to view the power utilization information.

S/N: DA719 Management IP: 192.168.111.182 Hostname: sp-bl-01

1 2 Text Mode Execute

```

PEM 0:
  State:      Offline
  Capacity:   1600 W (maximum 1600 W)
  DC output:  0 W (zone 0, 0.00 A at 0.00 V, 0% of capacity)

PEM 1:
  State:      Online
  Capacity:   1600 W (maximum 1600 W)
  DC output:  408 W (zone 0, 34.00 A at 12.00 V, 25% of capacity)

System:
  Zone 0:
    Capacity:      3200 W (maximum 3200 W)
    Allocated power: 408 W (2792 W remaining)
    Actual usage:   408 W
    Total system capacity: 3200 W (maximum 3200 W)
    Total remaining power: 2792 W
  
```

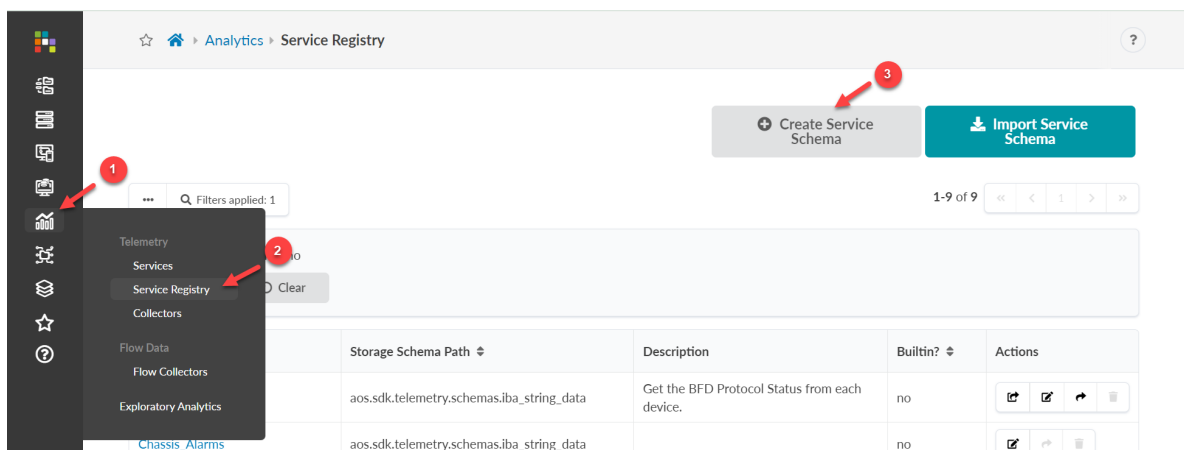
3. Continue to "Step 3. Create a Service Schema" on page 13.

Step 3. Create a Service Schema

You must create a service schema to define how you want your data to be structured and stored. A service defines your schema. You can create a service for a schema with a single output value or multiple output values. This procedure shows how to create a schema with multiple output values.

To create a service schema:

1. From the left navigation menu in the Apstra GUI, navigate to **Analytics > Service Registry**, then click **Create Service Schema**.



2. Define your schema.

Create Service Schema

Name *
Power





Description
New power schema

Telemetry Keys

Key #1 *
PSM

+ Add Key

Telemetry Values

Value #1 * Allocated_Capacity	Value Type * integer	
Value #2 Max_Capacity	Value Type integer	
Value #3 State	Value Type string	
Value #4 Thermal_Output	Value Type integer	
Value #5	Value Type	

☐ Create Another? **Create**

Specify the **Telemetry Keys** and **Telemetry Values**.

In Apstra, the telemetry keys and value type are a collection of key-value pairs.

- The telemetry key represents the identity of your service, such as Interface_name or Power_Supply_Module . The Value name might be Tx_rate or, for power monitoring, Voltage or Power_Draw.
- The value type is the data produced by the telemetry service, which can be a String (text) or an Integer (whole number).



NOTE: Key names and value names cannot contain hyphens or other special characters, except for underscores (_).

In our example, we've created a service schema that has multiple output values. We specified the PSM (Power Supply Module) as the **Service Key** and added multiple **Service Values** for Allocated Capacity, Max_Capacity, State, Thermal_Output, and Used_Capacity.



NOTE: For more information about PSM's (also referred to as PEMs), see [Managing Power](#) in the Juniper Chassis-Level User Guide.

3. Click **Create** to finish creating your schema.

The new service schema is added to the **Service Registry** table.

☆ Analytics > Service Registry

Create Service Schema Import Service Schema

Filters applied: 1 1-1 of 1

Applied Query: BuiltIn? = no

Copy Clear

Service Name	Storage Schema Path	Description	BuiltIn?	Actions
Power	aos.sdk.telemetry.schemas.liba_data	Power Consumption	no	

Step 4. Create a Telemetry Collector

So far, you've defined the data you want to collect and determined how the data will be organized and structured. Our final step is to create a telemetry collector for our service.



NOTE: A service is composed of one or multiple collectors. In this example, the service has a single collector.

To create a telemetry collector:

1. From the left navigation pane, navigate to **Analytics > Collectors**, then click **Create Collector**.

Analytics > Collectors

1-7 of 7

	OS Type	OS Version	OS Variant	Model	Actions
	junos	22.2r2	junos junos-ex junos-qfx	QFX	
	junos_evo	21.2r2	ptx ptx1k qfx-ms-fixed		
	junos	22.2r2	junos junos-ex junos-qfx		
	junos_evo	21.2r2	acx acx-f acx-qfx-7k ptx ptx1k qfx-ms-fixed		
	junos_evo	21.2r2	acx acx-f acx-qfx-7k ptx ptx1k qfx-ms-fixed		
	junos	22.2r2	junos junos-ex junos-qfx		
	junos	22.2r2	junos junos-ex junos-qfx		

2. Select the existing service schema you just created (in our example, **Power**), then click **Next**.

Create Telemetry Collector

Service Platform Command Mapping

Select existing service: Power or Create a new service schema

Next

3. Select the platform and devices for your telemetry collection. Defining a mix of these inputs enables you to be very broad or very granular. For example, you might want to collect telemetry just from a specific model or series.

Edit Telemetry Collector ✕

☒ Service
 ☒ **Platform**
☐ Command
 ☐ Mapping

OS ✕

junos_evo

OS Variant ✕

acx ✕ acx-f ✕ acx-qfx-7k ✕ ptx ✕ ptx1k ✕
 qfx-ms-fixed ✕

Device OS Version ✕

21.2r2

Model ✕

Target Devices

1-10 of 10

Management IP	Device Key	Hostname	Vendor	OS	Hardware Model
192.168.111.48	JN5ED6048JHA	superspine-qfx5700-01-re0	Juniper	Junos 22.4R3.23-EVO	QFX5700
192.168.111.49	JN5ED4F7CJHA	superspine-qfx5700-02-re0	Juniper	Junos 22.4R3.23-EVO	QFX5700
192.168.111.71	YR3623140015	dc2-bl	Juniper	Junos 22.4R3.23-EVO	QFX5130-32CD
192.168.111.174	YR3623120056	spine1	Juniper	Junos 22.4R3.23-EVO	QFX5130-32CD
192.168.111.175	YR3623120059		Juniper	Junos 22.4R3.23-EVO	QFX5130-32CD
192.168.111.179	GM228	compute-spine-ptx-02	Juniper	Junos 22.4R3.23-EVO	PTX10001-36MR
192.168.111.180	GM215	storage-spine-ptx-01	Juniper	Junos 22.4R3.23-EVO	PTX10001-36MR
192.168.111.181	GM612	storage-spine-ptx-02	Juniper	Junos 22.4R3.23-EVO	PTX10001-36MR
192.168.111.185	YK3623220023	storage-leaf-acx-01	Juniper	Junos 22.4R3.22-EVO	ACX7100-32C
192.168.111.186	YK3623220046	storage-leaf-acx-02	Juniper	Junos 22.4R3.22-EVO	ACX7100-32C

- a. Select the **OS type**, either **junos** or **junos-evo**.



NOTE: If you do not specify the `junos_evo` collector for Junos OS Evolved devices, the collector uses the corresponding Junos definition. This means if you use the same command between the different devices, you only need to create a single Junos collector definition for that service. If the command resides only on `junos_evo`, you'll need to create a single collector definition specifically for `junos-evo`.

For more information about `junos-evo` (also known as Junos OS Evolved), see the [Junos OS Evolved](#) documentation.

- b. Select the **OS Variant** the device belongs to and determine the CLI schema for a given device. This field accepts multiple entries. If you select multiple entries, the intersection of schemas is used.
- c. Select the **minimum OS Version** the device must run for the collector to execute. If you have multiple collector definitions with different OS versions for the same service, the collector automatically chooses the one closest to the version the device is running.
- d. (Optional) Specify a **Model** or a regular expression to filter based on a device model or series. The table shows a list of target devices currently managed in Apstra and matches the applied combination of filters.
- e. Click **Next**.

4. Execute the CLI command.

Use the show command to collect data from the device (in our example, show chassis), then click **Execute** to load the CLI schema.

CLI Command output

```

Chassis Power      Voltage(V)      Power(W)
Total Input Power
PSM 0
  State: Online
  Input 1      211      392
  Output      12.04      353.7
PSM 1
  State: Offline
  Input 1      0      0
  Output      0      0

System:
Zone 0:
  Capacity:      3000 W (maximum 3000 W)
  Allocated power: 1500 W (1420 W remaining)
  Actual usages: 392 W
  Total system capacity: 3000 W (maximum 3000 W)
  Total remaining power: 1420 W
  
```

Click **Next**.

5. Map the Keys and Value.

So far, we've defined the service schema, the target platforms, and the CLI command the custom telemetry collector will execute. Next, we'll map the key(s) and value type we defined in ["Step 3. Create a Service Schema " on page 13.](#)

- To map the keys, click **Expand All** to search for the RPC value you want to map.

✓ Service

✓ Platform

✓ Command

Mapping

+

 power-usage-item

+

 power-usage-max-fru-item

+

 power-usage-pdu-item

-

 power-usage-psm-item1

+

 power-usage-psm-detail1

+

 power-usage-psm-input1

+

 power-usage-psm-output

name?

slot

state?

+

 power-usage-system

+

 power-usage-system2

+

 power-usage-system3

+

 power-usage-total

+

 power-usage-voltage-total

+

 power-usage-voltage-total1

Add Mapping ▾

Mapped to PSM ✕

Add Mapping ▾

☒ PSM

☐ Allocated_Capacity

☐ Max_Capacity

☐ State

☐ Thermal_Output

☐ Used_Capacity

Keys are indicated in green

Values are indicated in blue

b. Click **Add Mapping** to map each value to a key.

c. Assign the value to the key.

In our example, PSM is the key and **state** is one of the values. This value is populated based on the dynamic state field returned by the CLI command as shown in XML output below.

show chassis power

get-power-usage-information

Show Chassis Power Usages

CLI Command

show chassis power

Command Arguments

Output Fields

```
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/22.4R0/junos">
  <power-usage-information>
    <power-usage-voltage-total1>
      <dc-power>231</dc-power>
    </power-usage-voltage-total1>
    <power-usage-psm-item1>
      <slot>0</slot>
      <state>Online</state>
      <power-usage-psm-input1>
        <name>Input 1</name>
        <dc-voltage>207</dc-voltage>
        <dc-power>231</dc-power>
      </power-usage-psm-input1>
      <power-usage-psm-output>
        <name>Output</name>
        <dc-voltage>11.99</dc-voltage>
        <dc-power>206.45</dc-power>
      </power-usage-psm-output>
    </power-usage-psm-item1>
  </power-usage-information>
</rpc-reply>
```

- d. In the **Create Telemetry Collector** window, search for the **state** field, then click **Add Mapping**.

Create Telemetry Collector

Service Platform Command Mapping

Mapping View: Default Advanced Filter: All fields Command: show chassis power Check Schema

Search Field

state

state

Add Mapping

PSM

Allocated_Capacity

Max_Capacity

State

Thermal_Output

Used_Capacity

- e. Map the state field to each value, then click **Submit**.

6. Validate that the collector is working.

- a. Click the **Advanced** option button on the **Create Telemetry Collector** page.
- b. Verify that the query and test results match your expected results.

Command: show chassis power Test Query

Query Results - 1 rows

PSM	Allocated_Capacity	Max_Capacity	State	Thermal_Output	Us
0	3000	3000	Online	2741.6400000000003	80

192.168.111.48

192.168.111.49

192.168.111.71

192.168.111.174

192.168.111.175

192.168.111.179

192.168.111.180

192.168.111.181

192.168.111.185

192.168.111.186

Congratulations! You successfully created a collector.



NOTE: When you define the integer (number) values for a collector, you might need to enter a value expression for the collector to function. This is because Junos occasionally reports number data as a string. Before the collector can be processed, you must perform a conversion from *string* to *integer* on the Apstra side.

To define the integer (number) values for a collector, enter **int(value)** into the **Value Expression** field, then click **Submit**.

You can also use additional operators as shown in the figure below. Calculating the Thermal_Output, for example, multiplies the used capacity by 3.41. For more information, see [Calculating System Thermal Output](#).

Data Accessors

Name	Path
PSM	/power-usage-information/power-usage-psm-item1/slot
Allocated_Capacity	/power-usage-information/power-usage-system/power-usage-zone-information/capacity-allocated
Max_Capacity	/power-usage-information/power-usage-system/power-usage-zone-information/capacity-max
State	/power-usage-information/power-usage-psm-item1/state
Used_Capacity	/power-usage-information/power-usage-system/power-usage-zone-information/capacity-actual-usage

+ Add Accessor

Keys

Name	Data Expression
PSM	PSM

Values

Name	Data Expression
Allocated_Capacity	int(re_match("\d+", Allocated_Capacity)) if Allocated_Capacity else int(re_match("\d+", Max_Capacity))
Thermal_Output	int(re_match("\d+", Used_Capacity))*3.41
Max_Capacity	int(re_match("\d+", Max_Capacity))
State	State

Use Custom Telemetry Data in an IBA Probe

SUMMARY

This topic describes how to configure and use custom telemetry data in an IBA probe.

IN THIS SECTION

- [Types of Processors | 22](#)
- [Create a Probe | 24](#)
- [Configure Additional Processors for Anomaly Generation | 29](#)
- [Verify Your Configuration | 51](#)

We've created a custom telemetry collector service to specify data from your devices. Next, we'll ingest this data into IBA probes in your blueprint as described in [Create a Probe](#). This process allows Apstra to visualize and analyze the data

Types of Processors

An IBA probe, functioning as an analytics pipeline, begins with a source processor. This processor creates data and several outputs without requiring any input. The different types of source processors are described in [Table 1 on page 22](#).

You can add more processors in the probe for extra data analytics. This enhances your network's health insights. These extra processors, known as analytical processors, are described in [Table 2 on page 24](#).

Analytical processors let you compile, logically process your data, and identify an intended state to detect anomalies. These processors perform calculations such as averages, min/max, and standard deviations. This aggregated data is then compared with expected results to determine if it's within a preset range. Anomalies are only flagged when a specific threshold is surpassed for a certain duration, preventing flags for transient conditions. A `Time_In_State` processor configuration can achieve this.

Table 1: Source Processors

Built-in	Built-in processors categorize the various processors used to enable services like power monitoring. These processors determine the scope of service activation using a graph query.
----------	--

Extensible	<p>Data from IBA probes is ingested by extensible collectors. This data is then collected by processors using a graph query provided by a custom service. Service keys are mapped using graph elements, depending on whether the data type is static or dynamic. This process decides the scope of telemetry collection.</p> <ul style="list-style-type: none"> Extensible Service Data Collector The Extensible Service Data collector processes data from custom telemetry services. This processor is ideal for services using custom telemetry collectors. It supports both static series (graph-driven) and dynamic series (collector-driven) telemetry collection. Generic Service Data Collector The Generic Service Data collector ingests data from "generic" custom telemetry services. Its telemetry collection is graph-driven and only supports static series.
Graph	<p>Graph processors ingest data from the IBA probes source in the graph database. They do not consume device telemetry data.</p> <ul style="list-style-type: none"> Generic Graph Collector The Generic Graph collector collects data from the active graph. The value field expression provides a value for each item in the graph query response.

Table 2: Analytical Processors

	Types of Analytical Processors
Analytical Processors	<ul style="list-style-type: none"> Grouping processors Grouping processors reduce output size compared to input data. Check processors Check processors examine a particular condition, using Boolean output to identify anomalies. Periodic processors Periodic processors measure a specific input over a user-determined period. Arithmetic processors Arithmetic processors carry out operations like arithmetic (subtraction, division), comparison ('greater than', 'less than'), and logical (AND, OR). Specialized processors Specialized processors use distinct analytics functions to support specific probes. Miscellaneous processors Miscellaneous processors, like "Accumulate," store short-term and intermediate data.

For detailed descriptions of all types of processors, see [Probe Processor \(Analytics\)](#) in the Juniper Apstra User Guide.

Create a Probe

Now, we'll create a probe in your deployed blueprint. A probe allows Apstra to gather data from your service. We're using a simple configuration in this scenario to view power information and to create anomaly alerts based on power usage.



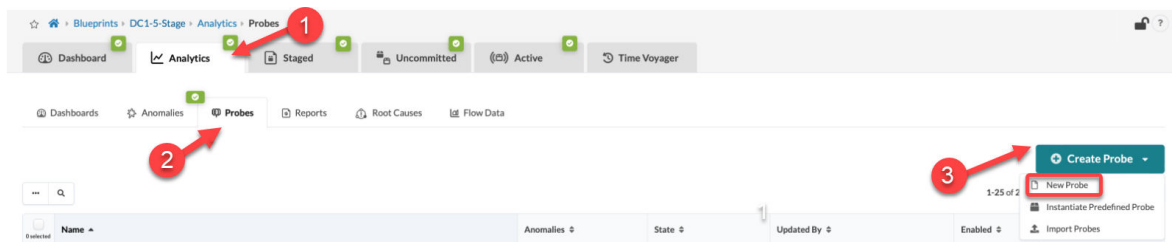
NOTE: As previously mentioned, an IBA probe, functioning as an analytics pipeline, begins with a source processor. It operates in two modes with the telemetry service: Static and Dynamic series modes. In Static series mode, all keys can be sourced entirely from the graph. Conversely, Dynamic series mode does not require key mapping. In instances where data like power supply isn't modeled in the graph, a probe is created without mapping to the graph. Subsequently, the data ingestion into the IBA pipeline is collector-driven rather than graph-driven.



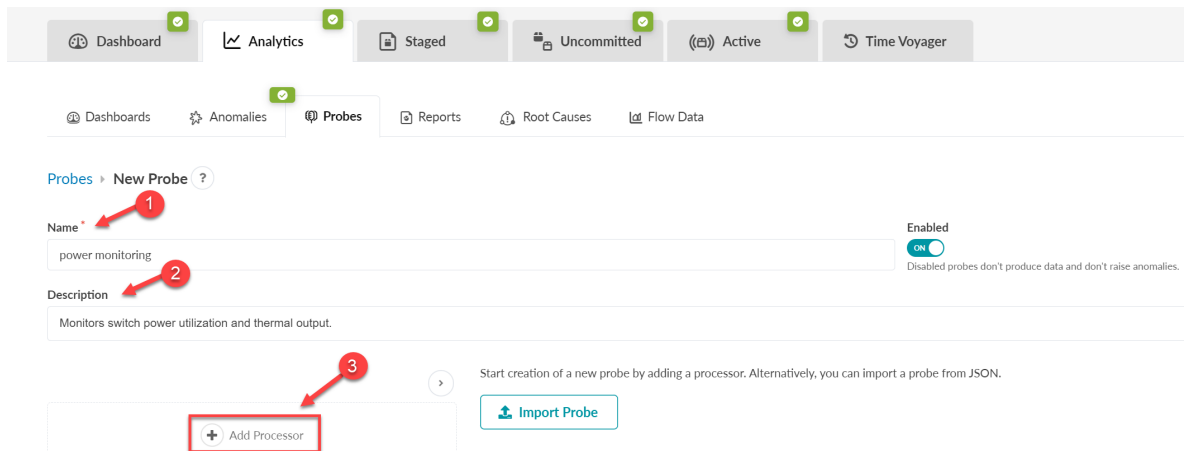
NOTE: Data Center and Freeform blueprints support IBA probes with the Custom Telemetry Collection.

To create a probe:

1. From your blueprint, navigate to **Analytics > Probes**, then click **Create Probe > New Probe**.

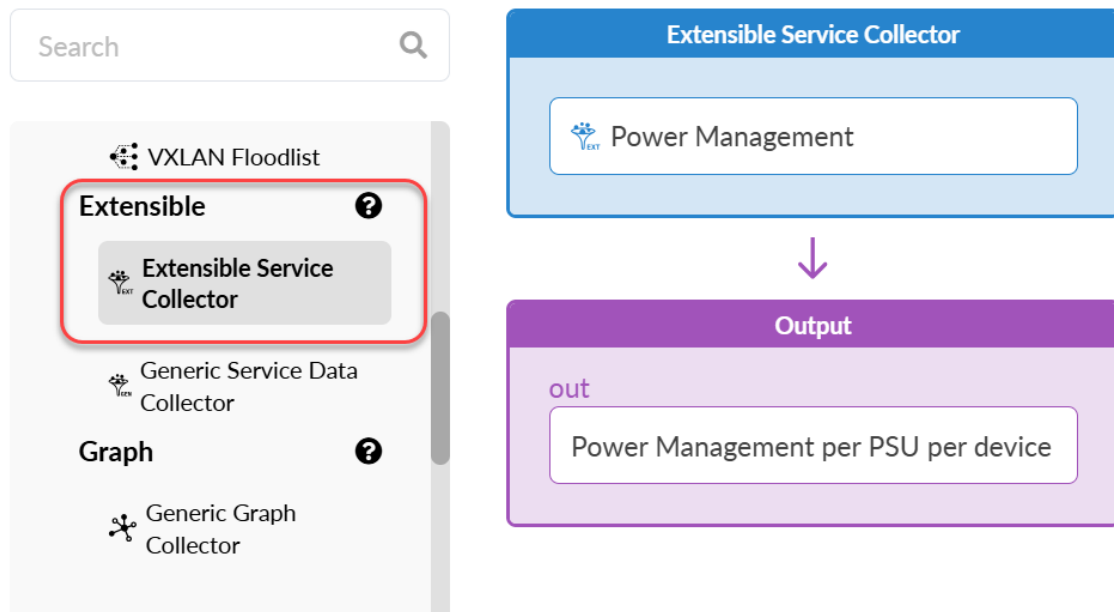


2. Enter a **Name** and **Description** (in this example, **power monitoring**), then click **Add Processor**.



3. Select a processor from the **Source Processors** list, then enter the collector's name and output information. In this example, we selected the **Extensible Service Collector** processor to consume the service you just created.

Add Processor



4. Click **Add** to add the processor to the probe.
5. To the right of the **Graph Query** field click the **Select a predefined graph query** button.

The Graph Query sets the blueprint's scope for the telemetry collection. If a device in the blueprint doesn't match the graph query, the telemetry collection for it won't start.

Processor: Power Management

⌵ ⌶ ✎ 🗑

Extensible Service Collector
⌵

Graph
⌵

Graph Query *

Click to select a predefined graph query

```

match(
  node('system', name='system', deploy_mode='deploy', role-is_in(['leaf', 'access', 'spine', 'superspine']))
)

```

📄 🔗

Node Type	Node Name
system	system

The graph query matches all system nodes in your blueprint graph database. Each managed device, whether a leaf switch or spine switch, is represented as a system node in the graph.

For example, in the **Graph Query**, the query matches all system, type nodes. In deploy mode, these nodes have roles such as leaf, access, spine, or superspine.

6. Select **DC – All managed devices (any role)** from the **Predefined Query** drop-down, then click **Update**.

Update Graph Query

Once selected, a pre-defined query is not kept synchronized, any change to the query is not automatically reflected here

Predefined Query **1**

DC - All managed devices (any role) ▼

```
match(
  node('system', name='system', deploy_mode='deploy', role
    =is_in(['leaf', 'access', 'spine', 'superspine']))
)
```

2 Update

7. From the Processor page, specify the following:

- In the **System ID** field, enter `system.system_id`.
This entry instructs the probe to match the graph query with your managed devices named `system`. The attribute `system_id` on each system node refers to the system ID of each device. This attribute is what Apstra uses to uniquely identify each device.
- Select **power** from the **Service name** drop-down list.
- Select the **Data Type**, either **Dynamic Value** or **Static Value**.
- Click **Create Probe**.

System ID * a

system.system_id

Expression mapping from graph query to a system_id.

Service name * b

Power

Name of the custom collector service.

Service interval

2 Minutes

Telemetry collection interval. Can be an expression.

Service input

.

Data to pass to telemetry collectors, if any. Can be an expression.

Data Type * c

Dynamic Value

Type of values produced from graph query results: numbers, strings or discrete states

Value Map

Value map is empty.

+ Add Column

A mapping of discrete-state values to human readable strings for all applicable output columns.

ⓘ Advanced

🔧 Power Management per PSU per device 📄

d Create Probe

Well done! You successfully create a probe!

We created a working probe that collects the power consumption for every device in your network. Now let's explore a few valuable customization options to refine your probe.

Service Interval

The telemetry collection service uses a service interval to fetch and ingest data from devices. This interval is crucial as a too aggressive one can overload your devices. The data type you collect determines the optimal interval.

Service interval

1 Minute

1 Minute

2 Minutes

5 Minutes

10 Minutes

30 Minutes

1 Hour

Query Tag Filter

Another useful customization option is the **Query Tag Filter**. Let's say you tagged some switches in your blueprint as **storage** for a specific monitoring use case. You can configure this filter to perform telemetry on devices with the matching tag, as shown in the following example:

Query Tag Filter

Tag Filter Operation

and

Depending on this parameter graph queries return results that satisfy all tag filters for "and" and at least only one of them for "or".

Node Name	Matcher	Tags
system	Is In	storage

+ Add Tag Filter

Filters named nodes in the graph queries by assigned tags.

Raw data from your custom telemetry collector might be difficult to interpret. Asptara, however, notifies you proactively if any anomaly is detected in your network.

In the next section, we'll enrich the power probe we created with additional processors to detect and raise anomalies.

Configure Additional Processors for Anomaly Generation

IN THIS SECTION

- Sum Processor to Aggregate Allocated Capacity | 31
- Sum Processor to Aggregate Thermal Output | 34
- Sum Processor to Aggregate Used Capacity | 37
- Ratio Processor to Calculate the Proportion of Used Capacity to Allocated Capacity | 40

- [Sum Processor to Calculate Total Thermal Output and User Power | 44](#)
- [Range processor to set threshold based anomaly | 49](#)

We'll now set up our power probe to identify any power anomalies. You can do this either individually or cumulatively by adding extra processors. The anomalies are then stored in a historical database for reference. We'll further augment the probe by aggregating power readings from all of the system power supplies.

To get started:

1. Click the **Edit** button next to the power monitoring probe you created in ["Create a Probe" on page 24](#).

	Dashboard	Analytics	Staged	Uncommitted	Active	Time Voyager	
<input type="checkbox"/> Device Traffic			✓ No anomalies	✓ Operational	System 14 days ago	ON	
<input type="checkbox"/> ECMP Imbalance (Fabric Interfaces)			✓ No anomalies	✓ Operational	System 14 days ago	ON	
<input type="checkbox"/> ESI Imbalance			✓ No anomalies	✓ Operational	System 14 days ago	ON	
<input type="checkbox"/> LAG Imbalance			✓ No anomalies	✓ Operational	System 14 days ago	ON	
<input type="checkbox"/> MAC Monitor			✓ No anomalies	✓ Operational	System 14 days ago	ON	
<input checked="" type="checkbox"/> Power Monitoring			✓ No anomalies	✓ Operational	admin 8 minutes ago	ON	Edit Button

2. Click the **Add Processor** button to start adding processors.



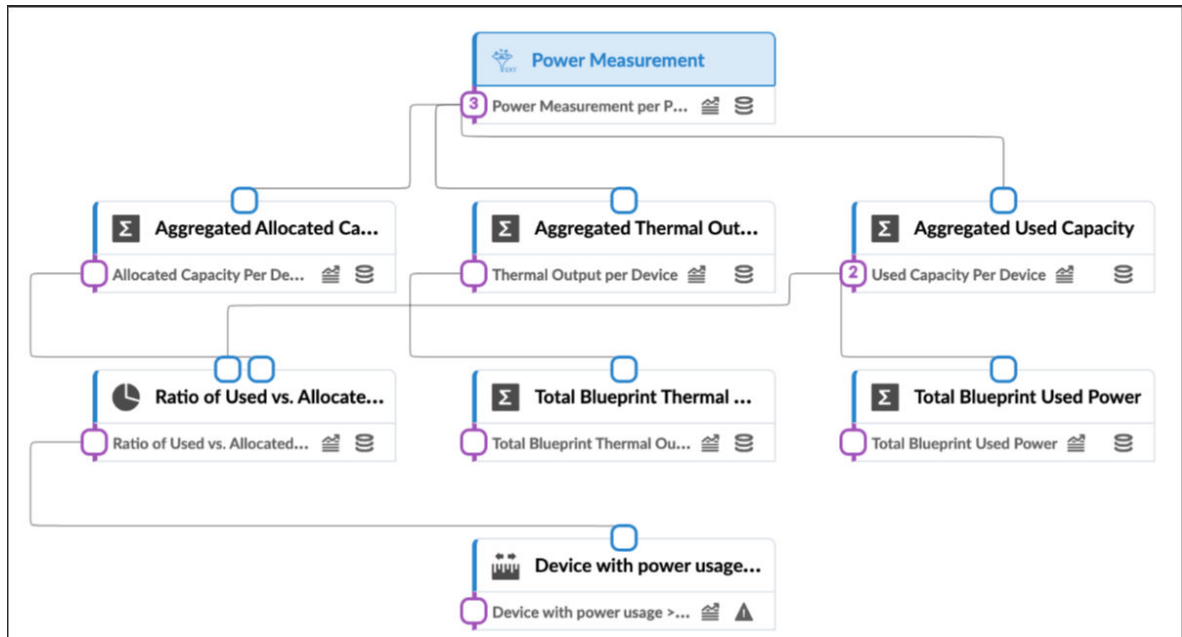
We'll now show you some examples of some different types of processors you can add to your probe. See [Figure 1 on page 31](#).



NOTE: Processors fall into two categories: source and analytical. Each category hold various sub-categories. Every probe requires at least one source processor. Although

you can use any type of processors, most probes use analytical processors, as shown in the following examples.

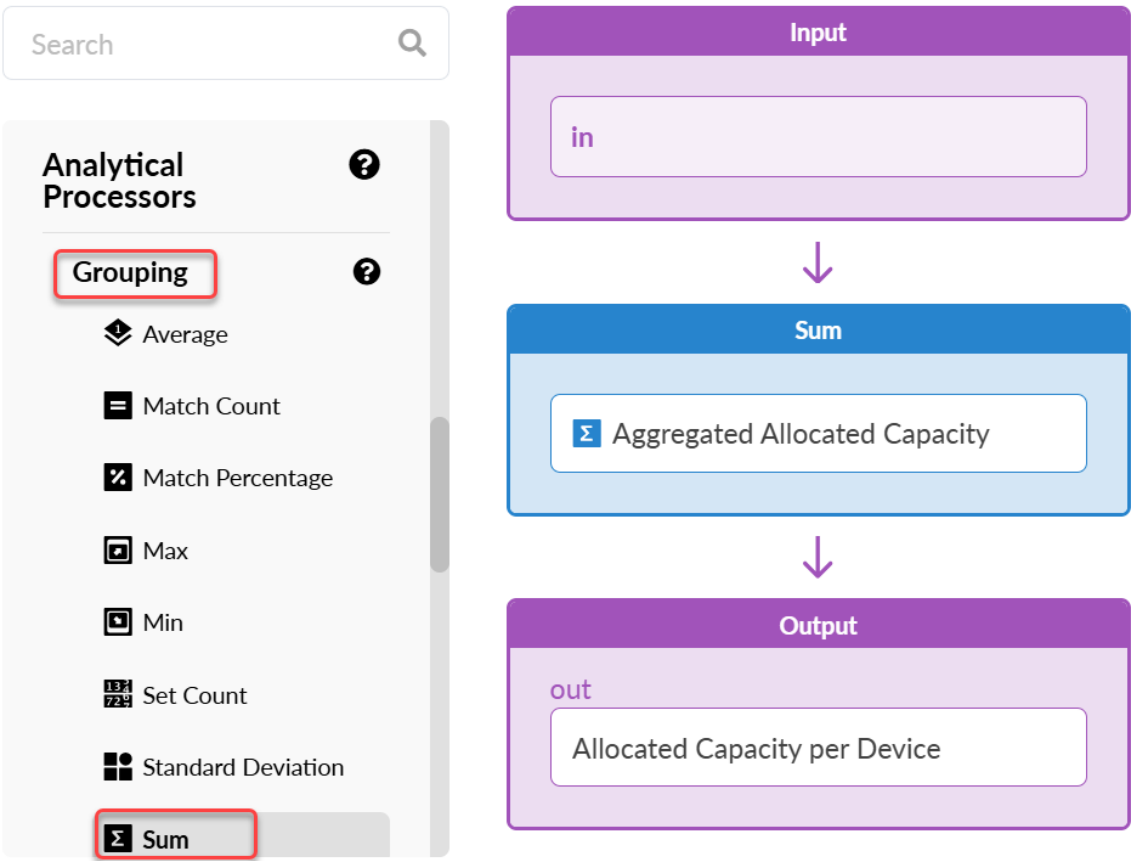
Figure 1: Example of Power Processors



Sum Processor to Aggregate Allocated Capacity

1. From the **Analytical Processors** list, under the **Grouping** category, select **Sum**. In the Sum and Output fields, enter **Aggregated Allocated Capacity** and **Allocated Capacity Per Device**.

Add Processor



2. Edit the processor.

Enter **Power Measurement per PSU per Device** for the Stage Name and **Allocated_Capacity** for the Column Name. We'll then group this data by `system_id` to provide a consolidated view of the allocated capacity for each device's power supplies.

Sum

Inputs

Input Stage

Input Name	Stage Name	Column Name
in	Power Measurement per PSU per Device	Allocated_Capacity

Processing



Group by


system_id

Accepts a list of property names to group input items into output items, produces only one output group for the empty list.

Advanced

3. Check **Enable Metric Logging** to enable logging, then enter the value such as Watt, to measure the **Allocated Capacity Per Device**. This action allows tracking of aggregate usage history for the past 30 days.

 Allocated Capacity Per Device 

 Dynamic

Properties

☒ Enable Metric Logging

Save changes in this stage to MetricDB time series database.

Retention Duration


Last 30 Days

Retain data in MetricDB for specified duration.

Description

Graph Annotation Properties

There are no annotation properties defined.

 Add Annotation Property

Annotate graph nodes with properties associated with stage values.

Units

Value	Units
Total Count	
Value	Watt

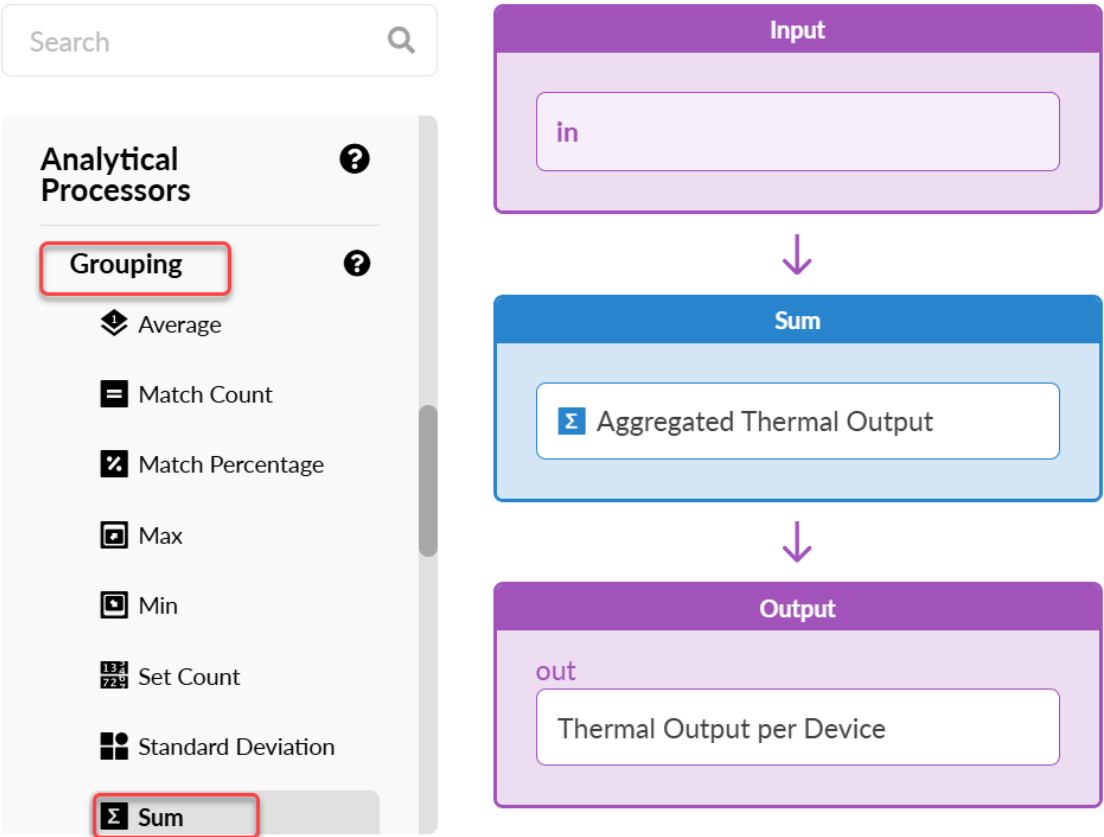
Units for values.

Sum Processor to Aggregate Thermal Output

The thermal output of devices is an important metric to monitor. It plays a key role in data center capacity planning and power usage. Having knowledge of the actual power utilization and thermal output for devices and your EVPN fabric helps the DC Manager plan for power and cooling capacity expansion. The thermal output data from our custom telemetry collector provides a comprehensive fabric total.

1. From the **Analytical Processors** list, under the **Grouping** category, select **Sum**. In the **Sum** and **Output** fields, enter **Aggregated Thermal Output** and **Thermal Output Per Device**.

Add Processor



2. Edit the processor.

Enter **Power Measurement per PSU per Device** for the Stage Name and **Thermal_Output** for the Column Name. We'll then group this data by system id to provide a consolidated view of the thermal output for each device.

Processor: Aggregated Thermal Output



Sum

Inputs

Input Stage

Input Name

in

Stage Name

Power Measurement per PSU per Device

Column Name

Thermal_Output

Processing

Group by

system_id

Accepts a list of property names to group input items into output items, produces only one output group for the empty list.

Advanced

3. Check **Enable Metric Logging** to enable logging Then specify the power consumption value, such as BTUs, to measure the thermal output.

Thermal Output per Device

Dynamic

Properties

☒ Enable Metric Logging

Save changes in this stage to MetricDB time series database.

Retention Duration

Last 30 Days

Retain data in MetricDB for specified duration.

Description

Graph Annotation Properties

There are no annotation properties defined.

+ Add Annotation Property

Annotate graph nodes with properties associated with stage values.

Units

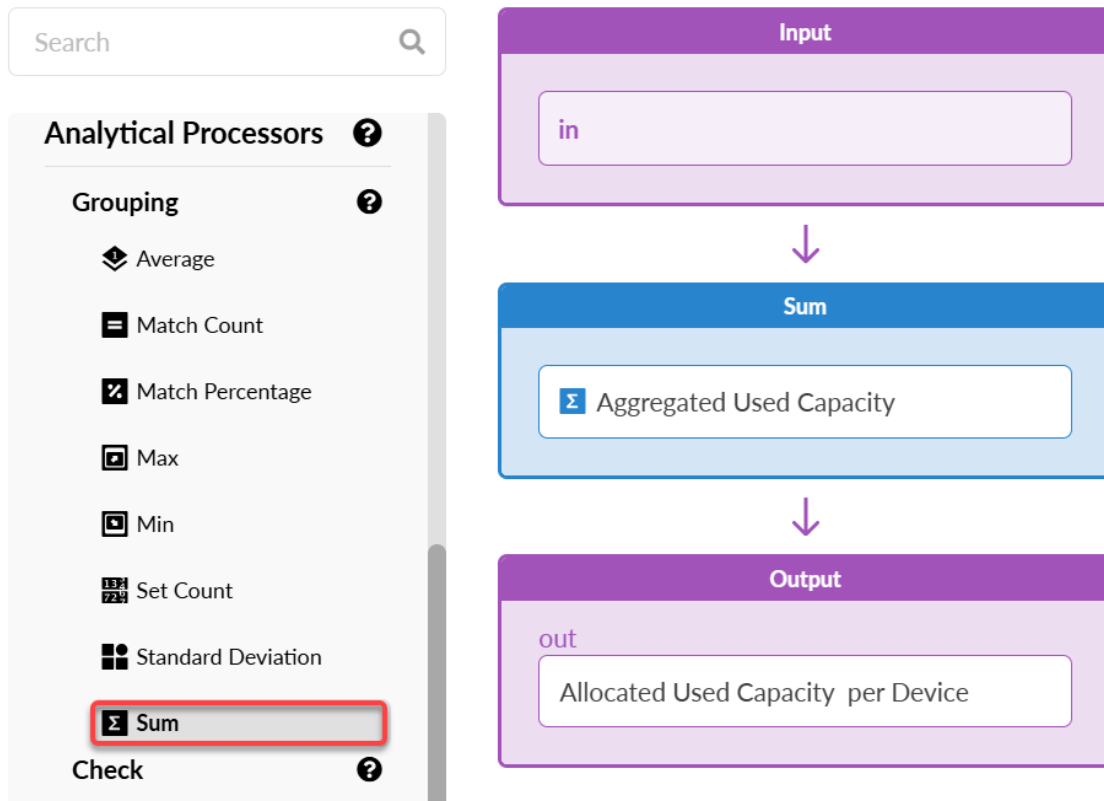
Value	Units
Total Count	
Value	BTUs / H

Units for values.

Sum Processor to Aggregate Used Capacity

- 1. From the **Analytical Processors** list, under the **Grouping** category, select **Sum**. In the Sum and Output fields, enter **Aggregated Used Capacity** and **Allocated Used Capacity per Device**.

Add Processor



2. Edit the processor.

Enter **Power Measurement per PSU per Device** for the Stage Name and **Used_Capacity** for the Column Name. We'll then group this data by `system id` to provide a consolidated view of the actual power usage for each device.

Processor: Aggregated Used Capacity



Σ Sum

Inputs

Input Stage *

Input Name

in

Stage Name

Power Measurement per PSU per Device

Column Name

Used_Capacity

⚙ Processing

Group by

system_id

Accepts a list of property names to group input items into output items, produces only one output group for the empty list.

📘 Advanced

3. Check **Enable Metric Logging** to enable logging. Then specify the power consumption value, such as Watt, to measure the used capacity.

Used Capacity Per Device

Dynamic

Properties

☒
Enable Metric Logging

Save changes in this stage to MetricDB time series database.

Retention Duration

Last 30 Days

Retain data in MetricDB for specified duration.

Description

Graph Annotation Properties

There are no annotation properties defined.

+

Add Annotation Property

Annotate graph nodes with properties associated with stage values.

Units

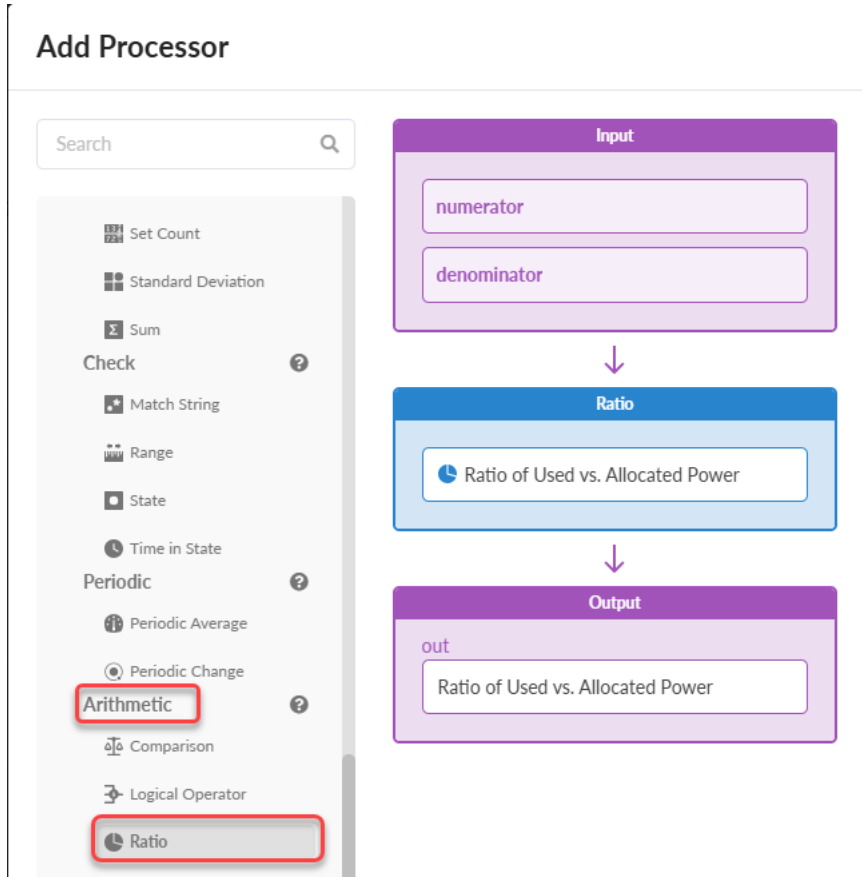
Value	Units
Total Count	
Value	Watt

Units for values.

Ratio Processor to Calculate the Proportion of Used Capacity to Allocated Capacity

The Arithmetic processor, an Analytical processor type, calculates power usage ratios for each device. This offers a clear view of power distribution.

1. From the Analytical Processor list, under the Arithmetic category, select **Ratio**. In the Sum and Output fields, enter **Ratio of Used vs. Allocated Power**.



2. Enter **Used Capacity Per Device** as the numerator and **Allocated Capacity Per Device** as the denominator. Use **100** as the Multiplier value so that the result shows as a percentage.
3. Set the **Result type** based on input and type.

Processor: Ratio of Used vs. Allocated Power

Inputs

Input Stage	Input Name	Stage Name	Column Name
numerator	Used Capacity Per Device	value	
denominator	Allocated Capacity Per Device	value	

Processing

Significant Keys

No keys specified

List of keys to map items from the inputs for applying the specified operation. If not specified, keys for all the inputs have to be the same.

Multiplier

100

Multiply numerator by a given value before calculating ratio

Denominator

Integer or an expression that evaluates to integer that is used as denominator. Should not be 0.

Result type



based_on_input_and_type


Type of result values.

Based on Input and Type means:

- Static Type:
 - Integer Input - Integer Output
 - Float Input - Float Output
- Dynamic Type:

4. Check **Enable Metric Logging** to enable logging. Then specify the power consumption value in percentages (%) to measure the Ratio of Used vs. Allocated Power for each device.

 Ratio of Used vs. Allocated Power 

 Dynamic

Properties

☒ **Enable Metric Logging**

Save changes in this stage to MetricDB time series database.

Retention Duration

Last 30 Days

Retain data in MetricDB for specified duration.

Description

Graph Annotation Properties

There are no annotation properties defined.

+

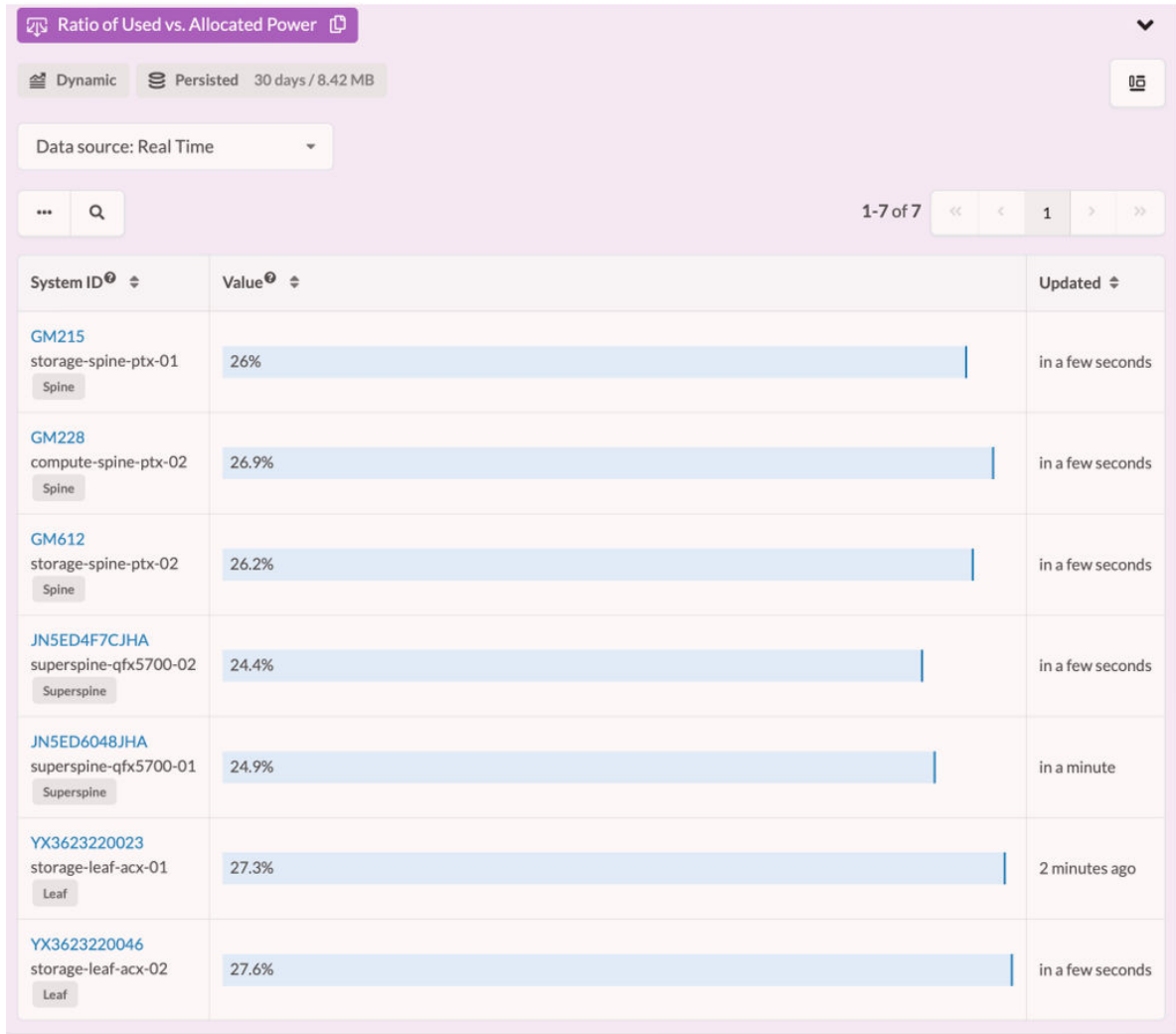
Add Annotation Property

Annotate graph nodes with properties associated with stage values.

Units

Value	Units
Value	%

Units for values.



Sum Processor to Calculate Total Thermal Output and User Power

For capacity planning purposes, it's useful to know the total power usage and thermal output for your entire fabric. For this information, we'll create two more Sum processors for **Total Blueprint Thermal Output** and **Total Blueprint Used Power**.

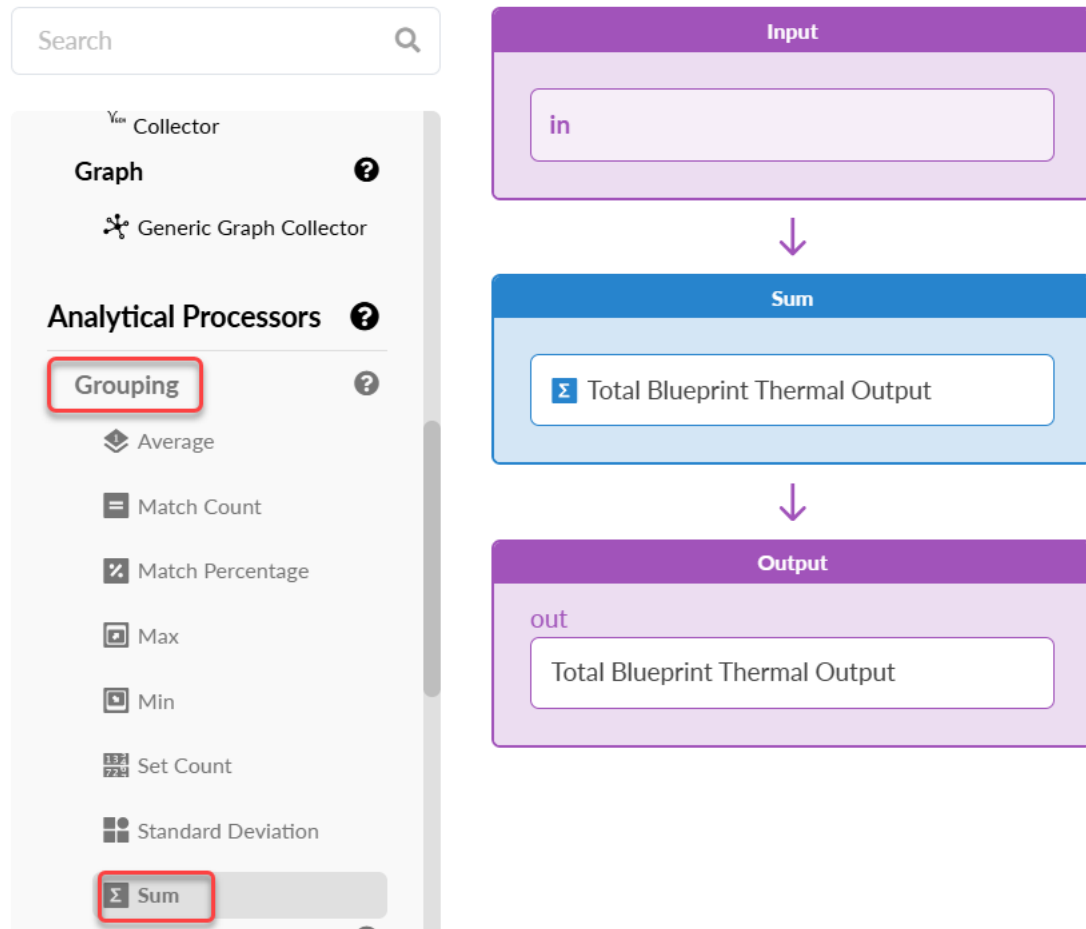


NOTE: In both processors, the **Group by** field remains empty. This approach allows for an aggregate across the blueprint.

To add a processor for total thermal output:

1. From the **Analytical Processor** list, under the **Grouping** category, select **Sum**. In the Sum and Output fields, enter **Total Blueprint Thermal Output**.

Add Processor



2. Edit the processor.

Enter **Thermal Output per Device** for the Stage Name and value for the **Column Name**.

Σ Sum

▼

Inputs

Input Stage *

Input Name

in

Stage Name

Thermal Output per Device

Column Name

value

⚙ Processing

▼

Group by

No properties specified, one output group will be produced

▼

Accepts a list of property names to group input items into output items, produces only one output group for the empty list.

🔍 Advanced

▶

3. Check **Enable Metric Logging** to enable logging.

Total Blueprint Thermal Output

Dynamic

Properties

☒ Enable Metric Logging

Save changes in this stage to MetricDB time series database.

Retention Duration

Last 30 Days

Retain data in MetricDB for specified duration.

Description

Graph Annotation Properties

There are no annotation properties defined.

+ Add Annotation Property

Annotate graph nodes with properties associated with stage values.

Units

Value	Units
Total Count	
Value	BTU/H

Units for values.

4. View your result.

Total Blueprint Thermal Output

Dynamic

Persisted 30 days / 3.01 MB

Data source: Real Time

1 of 1

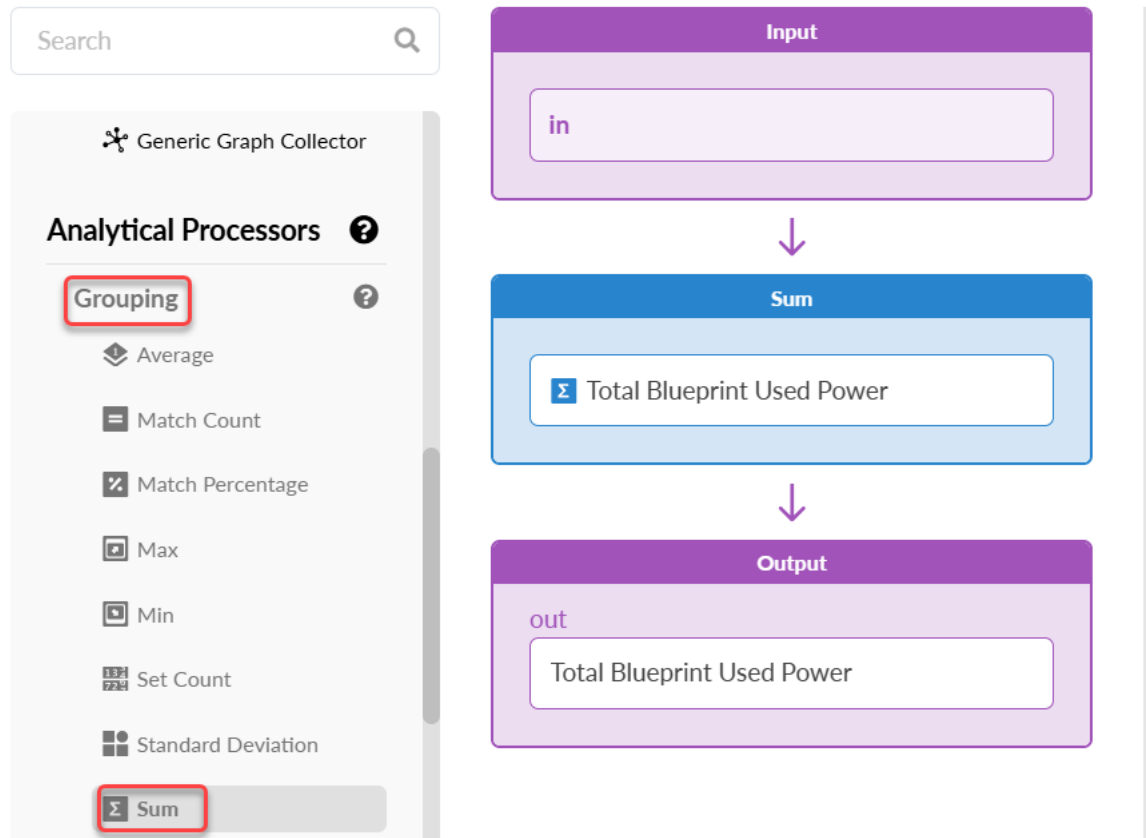
<< < 1 > >>

Total count	Value	Updated
7	13769	in a few seconds

5. Next, we'll create a processor for **Total Blueprint Used Power**.

From the **Analytical Processor** list, under **Grouping**, select **Sum**. In the Sum and Output fields, enter **Total Blueprint Used Power**.

Add Processor



6. Edit the processor.

Enter **Used Capacity Per Device** for the Stage Name and **value** for the **Column Name**.

Σ Sum

Inputs

Input Stage*

Input Name

Stage Name

Column Name

in

Used Capacity Per Device

value

⚙️ Processing

Group by

No properties specified, one output group will be produced

Accepts a list of property names to group input items into output items, produces only one output group for the empty list.

🔍 Advanced

7. View your result.

Total Blueprint Used Power

Dynamic

Persisted

30 days / 2.82 MB

Data source: Real Time

1 of 1

<<

<

1

>

>>

Total count	Value	Updated
7	4034	in a minute

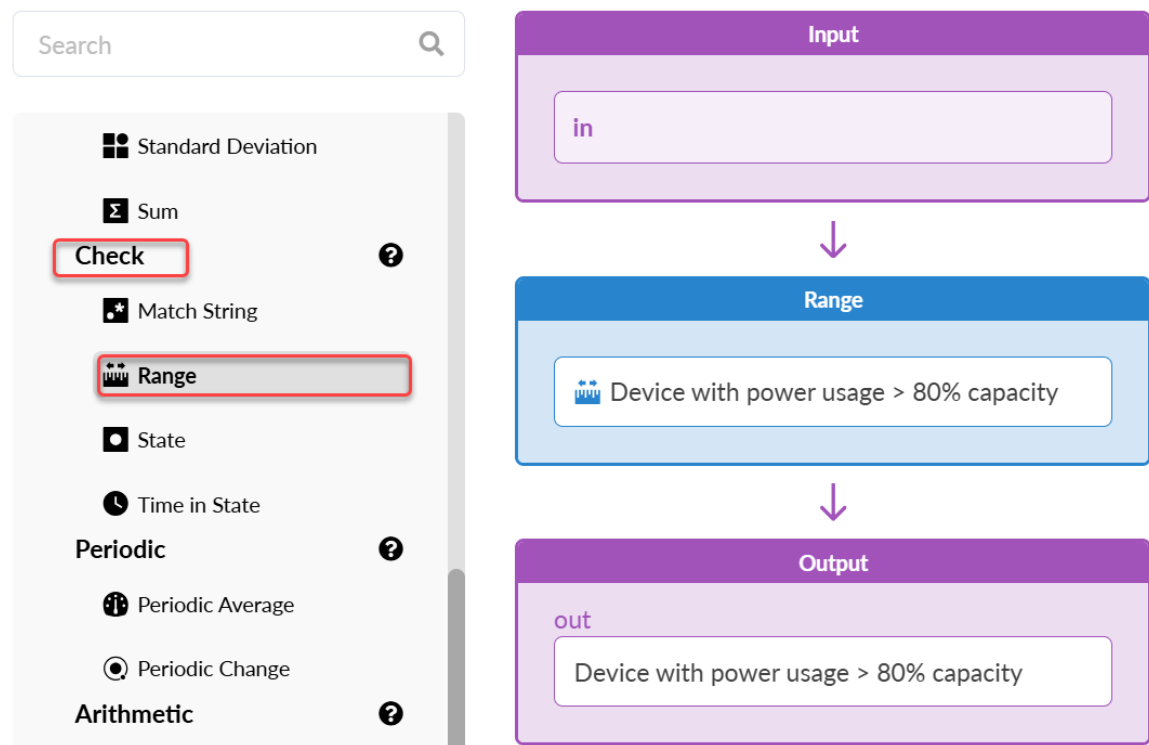
Range processor to set threshold based anomaly

You can create a range processor to generate an alert when a device uses more than 80 percent of its allocated capacity. A range processor checks the value against a defined range.

To create a range processor:

1. From the **Analytical Processor** list, under the **Check** category, select **Range**. In the Range and Output fields, enter **Device with power usage > 80 % capacity**.

Add Processor



2. Set the range processor to the ratio of used versus allocated power and specify an anomalous range of **80** percent or more. Check **Raise Anomaly** to receive alerts in the Apstra GUI when this range is exceeded. This allows for efficient power management and early detection of potential issues.

Range

Inputs

Input Stage *

Input Name

Stage Name

Column Name

in

Ratio of Used vs. Allocated Power

value

Processing

Anomalous Range *

More than or equal to

80

Numeric range, either min or max is optional. Float type is acceptable only with property "std_dev", other property values require integers. Min and max can be expressions evaluated into numeric values.

Property

value

A property of input items which is used to check against the range.

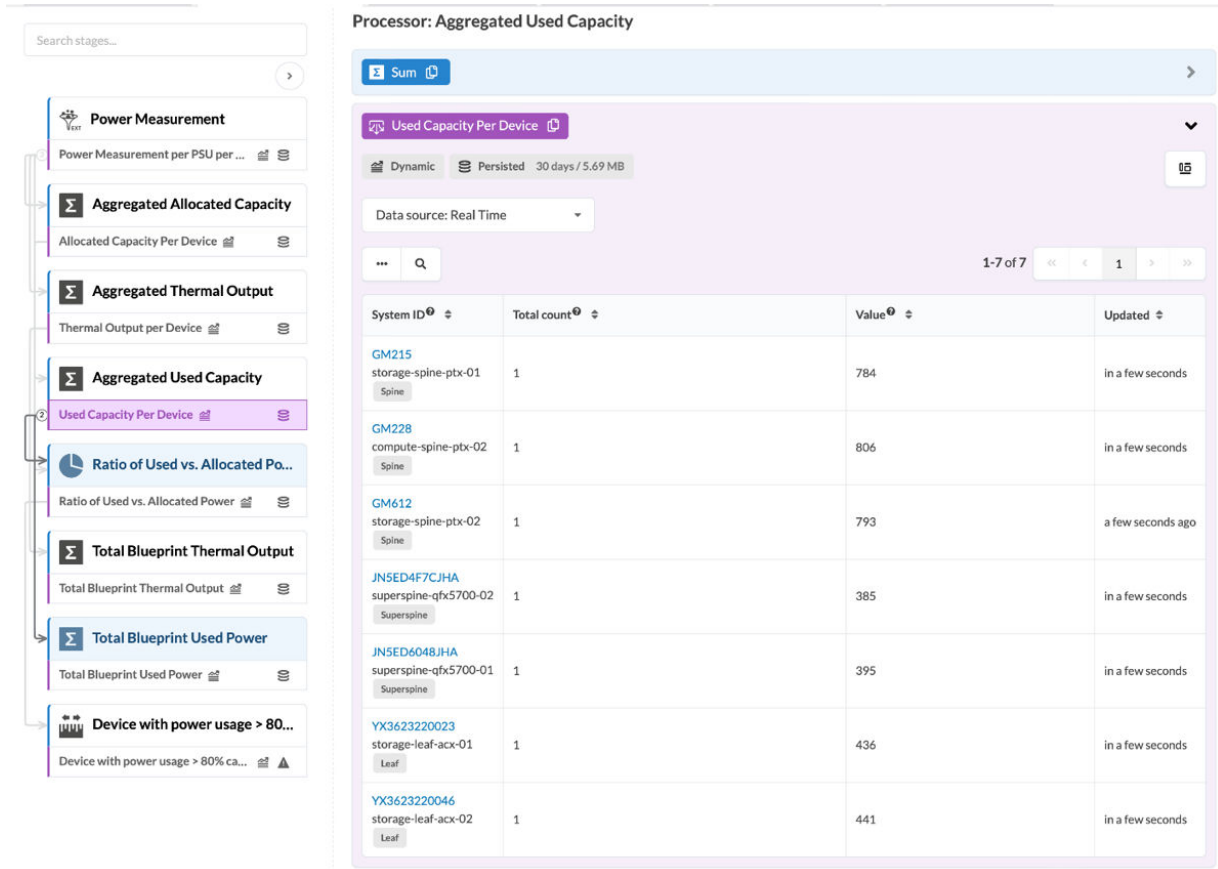
☒ Raise Anomaly

Whether to raise an anomaly

Advanced

Verify Your Configuration

You can view your configuration by selecting your probe from the table under **Analytics > Probes**. The following example shows the processor we created for **Aggregated Used Capacity** that details the capacity used for each device.



Monitor the Health of the Telemetry Service

It is important to consider the load on your devices when creating a custom telemetry collection. Telemetry services could overload your devices based on the CLI show commands and collected data. Short intervals for collector execution can also impact impacting traffic forwarding. By default, Apstra provides an IBA telemetry health probe to monitor service health, including customer services and collectors.

To monitor the health of your telemetry service:

1. From your blueprint, navigate to **Analytics > Probes**.
2. Select the **Device Telemetry Health** probe from the table.

Dashboard | Analytics | Staged | Uncommitted | **Active** | Time Voyager

1-8 of 8

Name	Anomalies	State	Updated By	Enabled	Actions
Device System Health	No anomalies	Operational	System 5 days ago	ON	[Icons]
Device Telemetry Health	No anomalies	Operational	System 5 days ago	ON	[Icons]
Device Traffic	No anomalies	Operational	System 5 days ago	ON	[Icons]

3. To filter the telemetry health, click the magnifying glass icon.

To display data for your new custom telemetry service, select a service name from the **Service name** drop-down filter (in this example, **Power**).

Dashboard | Analytics | Staged | Uncommitted | **Active** | Time Voyager

Processor: Telemetry Stats

Telemetry Service Health

Dynamic | Persisted 30 days / 19.19 MB

Data source: Real Time

1-25 of 58

System	System ID	System Hostname	System Role	Collection Type	Has Service Started	Did Last Execution Fail	Did Last Execution Timeout	Did Last Execution Underrun	Execution Time
5254-leaf3	leaf3		Leaf	Spine	Power	false	false	false	0.699228305951
5254-spine1	spine1		Spine	Leaf		false	false	false	0.730011203035
5254-leaf1	leaf1		Leaf	Leaf		false	false	false	0.552752138988

4. Click **Apply**. The table now shows the health metric for your custom telemetry service.

Applied Query: Service name = "Power"

Clear

System ID ⓘ ⚙	Service name ⓘ ⚙	Collection Type ⓘ ⚙	Has Service Started ⓘ ⚙	Run Count ⓘ ⚙	Success Count ⓘ ⚙	Failure Count ⓘ ⚙	Timeout Count ⓘ ⚙	Underrun Count ⓘ ⚙	Did Last Execution Fail ⓘ ⚙	Did Last Execution Timeout ⓘ ⚙	Did Last Execution Underrun ⓘ ⚙
DA719 sp-bl-01 Leaf	Power	polling	false	0	0	0	0	0	false	false	false
DB757 service-pod-spine-01 Spine	Power	polling	false	0	0	0	0	0	false	false	false
DT505 sp-bl-02 Leaf	Power	polling	false	0	0	0	0	0	false	false	false
GM215 storage-spine-ptx-01 Spine	Power	polling	true	11256	11256	0	0	0	false	false	false
GM228 compute-spine-ptx-02 Spine	Power	polling	true	11260	11260	0	0	0	false	false	false

Check the following:

- Ensure that the **Success Count** value has increased. If the value remains the same, your service might be failing. Alternatively, your custom collector could be misconfigured.
- Check the **Execution Time**.

If the execution time resembles or exceeds the service interval, there might be an issue. If so, adjust your probe settings and increase the service interval. For instructions on setting the service interval, see ["Create a Probe" on page 24](#).

Similarly, a sustained nonzero **Waiting Time** can indicate that the device is taking too long to complete your service request.

5. To see how your metrics are trending, switch to **Time Series** view under the **Data Source** drop-down. The following graph shows the metrics for Power service.

System ID ⓘ ⚙	Service name ⓘ ⚙	Execution Time ⓘ
GM215 storage-spine-ptx-01 Spine	Power	
GM228 compute-spine-ptx-02 Spine	Power	
GM612 storage-spine-ptx-02 Spine	Power	
JN5ED4F7CJHA superspine-qfx5700-02 Superspine	Power	
JN5ED6048JHA superspine-qfx5700-01 Superspine	Power	

For more information about each of these columns and their definitions, see [Telemetry Collection Statistics](#) in the Juniper Apstra User Guide.

Import and Export Services

SUMMARY

IN THIS SECTION

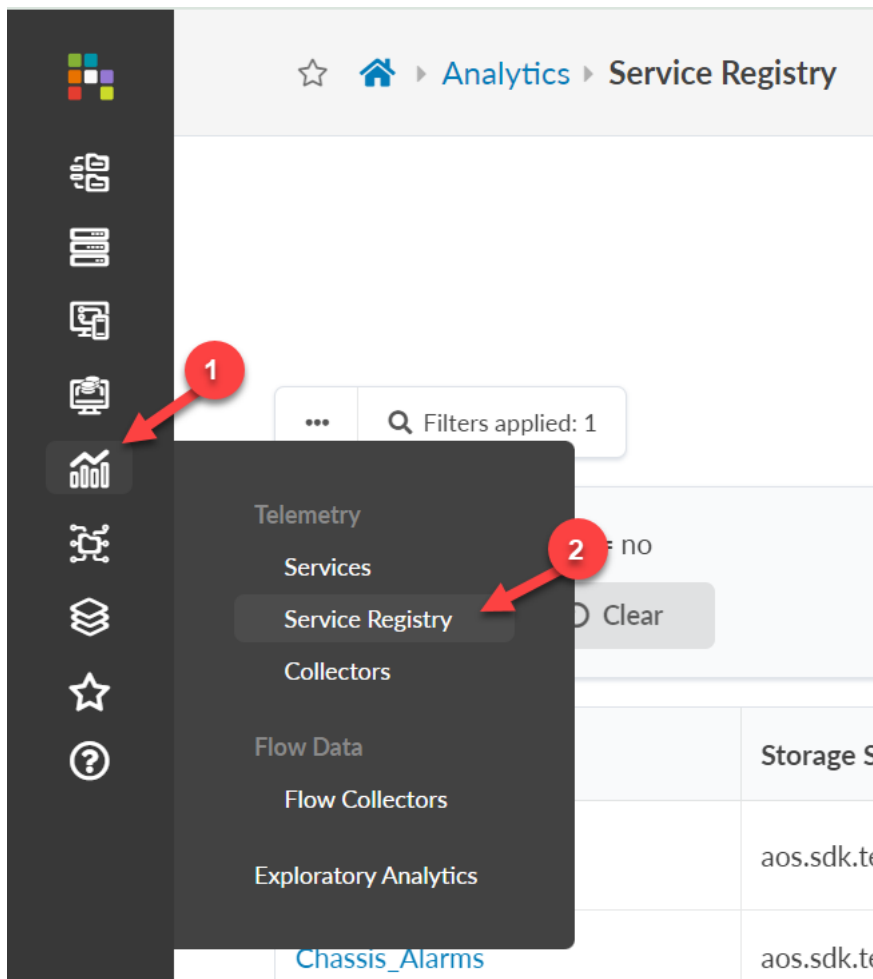
- [Export and Import a Service from the Service Registry | 55](#)
- [Exporting and Importing a Service from the Apstra Flow Dashboard \(Recommended\) | 57](#)

Export and Import a Service from the Service Registry

The service registry, which holds the service schema and the collector, allows you to export and import services. In our example, we're exporting one collector, but a service can include multiple collectors.

To export a service from the service registry:

1. From the left navigation pane in the Apstra GUI, navigate to **Analytics > Service Registry**.



2. Select your service from the registry table (in this example, Power), then click the **Export** button.

Service Name ↕	Storage Schema Path ↕	Description	Builtin? ↕	Actions
bfd_status	aos.sdk.telemetry.schemas.iba_string_data	Get the BFD Protocol Status from each device.	no	
Chassis_Alarms	aos.sdk.telemetry.schemas.iba_string_data		no	
DDoS_Protection_Protocols	aos.sdk.telemetry.schemas.iba_data		no	
EV-T2-routes	aos.sdk.telemetry.schemas.iba_data		no	
Interface_Queue_Queued_PPS	aos.sdk.telemetry.schemas.iba_integer_data		no	
Interface_Queue_Transmitted_PPS	aos.sdk.telemetry.schemas.iba_integer_data		no	
intf-usage	aos.sdk.telemetry.schemas.iba_data	Junos PFE Intf Usage	no	
nh-usage	aos.sdk.telemetry.schemas.iba_data	Junos PFE Next-Hop Usage	no	
Power	aos.sdk.telemetry.schemas.iba_data		no	

3. The schema that you export contains the service schema and collector info, as you can see in the example below. You can then import this file into a different instance.

Export Power

```

{
  "description": "",
  "application_schema": {
    "properties": {
      "key": {
        "properties": {
          "PSM": {
            "type": "string"
          }
        },
        "required": [
          "PSM"
        ],
        "type": "object"
      },
      "value": {
        "properties": {
          "Allocated_Capacity": {
            "type": "integer"
          },
          "Max_Capacity": {
            "type": "integer"
          },
          "State": {
            "type": "string"
          },
          "Thermal_Output": {
            "type": "integer"
          },
          "Used_Capacity": {

```

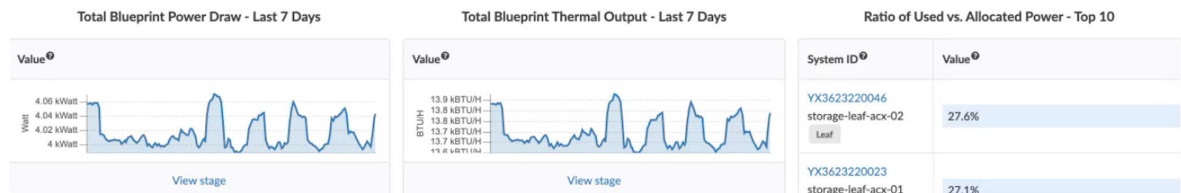
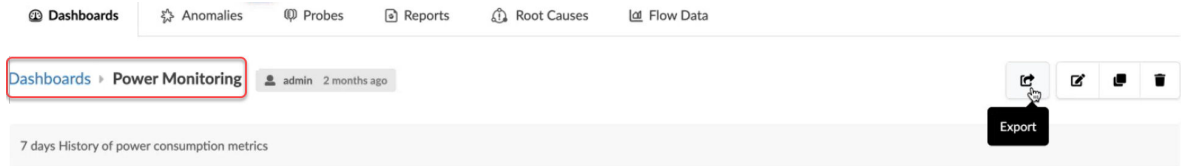
Save As File

Exporting and Importing a Service from the Apstra Flow Dashboard (Recommended)

You can export and import a service from the Apstra Flow dashboard. The dashboard holds the probe definitions.

To export a service from the dashboard:

1. From the dashboard table, select your probe (Power monitoring, for example). Then click the **Export** button to see the power monitoring dashboard.



- Exporting your dashboard generates a schema with the probe information as shown in the following example. You can then import this file to a different instance.

Export Power

```

{
  "description": "",
  "application_schema": {
    "properties": {
      "key": {
        "properties": {
          "PSM": {
            "type": "string"
          }
        },
        "required": [
          "PSM"
        ],
        "type": "object"
      },
      "value": {
        "properties": {
          "Allocated_Capacity": {
            "type": "integer"
          },
          "Max_Capacity": {
            "type": "integer"
          },
          "State": {
            "type": "string"
          },
          "Thermal_Output": {
            "type": "integer"
          },
          "Used_Capacity": {

```

Save As File

Summary

This document taught you the basics of Apstra Intent-Based Analytics. It also explained how to establish a custom telemetry service for data collection from managed devices. Additionally, you learned to create an IBA probe for data visualization, analysis, and anomaly detection.

For more information about Apstra and the Apstra GUI, see the [Juniper Apstra User Guide](#).