

# Juniper Apstra 4.2.0 Custom Telemetry Collection Guide

Published  
2024-03-04

RELEASE

# Table of Contents

**Introduction**

**Apstra Telemetry and Intent-Based Analytics**

**Custom Telemetry Collection Overview**

**Creating a Custom Telemetry Collector**

Execute the CLI Command | 10

Identify the Key and Value of Interest from the CLI Output | 13

Create a Service Schema | 14

Create a Collector | 15

Validate That the Collector Is Working | 19

**Using Custom Telemetry Data in an IBA Probe**

Create a Probe | 22

Customize a Probe | 26

Performing Analytics | 27

Raising Anomalies and Storing Historical Data | 29

**Monitoring the Health of the Telemetry Service**

**Summary**

# Introduction

Juniper Apstra is a powerful automation solution that manages the full life cycle of data center switching fabrics. Apstra's Intent-Based Networking (IBN) approach to automation helps you design, build, deploy, operate and validate your network.

Apstra validates that:

- The user-supplied inputs are valid.
- The user inputs are consistent and compatible with the constraints of the network.
- The expected telemetry outputs are correct when the network is stable.
- There are no gaps between the expected and actual telemetry.

Once you deploy your network, Apstra collects various telemetry data from its managed devices. This data is automatically aggregated and validated against the intended state of each telemetry type, such as interfaces, LLDP, BGP, and so on. This capability in Apstra is called *Intent-Based Analytics*, or IBA. IBA is an invaluable tool for obtaining accurate and relevant data for robust 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 visualize and analyze your data.

In this document, you will learn:

- The fundamentals of IBA.
- How to define a custom telemetry service.
- How to create a new IBA probe to visualize and analyze data from your telemetry service.

We'll also walk you through an example use case that shows you how to:

- Define a custom telemetry service that gathers the BFD session state from managed devices.
- Create an IBA probe that ingests and visualizes the BFD session state data.
- Customize your IBA probe to raise anomalies for BFD sessions that are down.
- Store the history of anomalies in a time-series database.

Let's dive in!

# Apstra Telemetry and Intent-Based Analytics

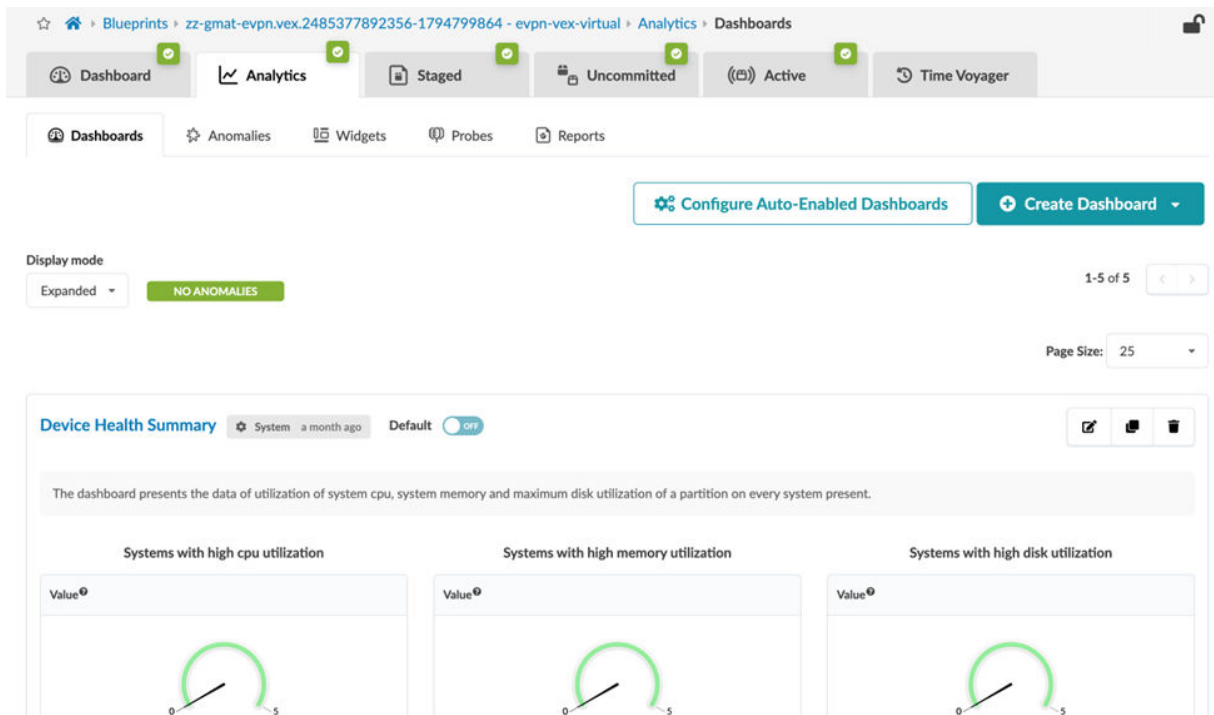
## IN THIS SECTION

- [What Is Intent-Based Analytics? | 2](#)
- [Telemetry Services | 4](#)
- [Auto-Enabled Probes | 4](#)
- [Predefined Probes Catalog | 5](#)
- [Custom Probes | 7](#)
- [What If Apstra Doesn't Collect the Data You're Looking For? | 8](#)

## What Is Intent-Based Analytics?

Intent-Based Analytics (IBA) helps you with any operational status changes in your infrastructure by extracting knowledge out of raw telemetry data.

You configure IBA in the Apstra GUI. From the left navigation menu, click **Blueprints**. Select your blueprint, then navigate to the **Analytics** tab in the dashboard as shown below.



## IBA Probes

In IBA, probes represent a single analytics pipeline. A probe is a configurable data-processing pipeline that enables you to set up conditions of interest (situations to watch). IBA probes fetch data, apply processing, 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.
- Generate 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 5 .

## 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. The show commands are also used to configure a gRPC sensor path (see the [Junos Telemetry Interface User Guide](#) for information).

To view the available telemetry services in the Apstra GUI, from the left navigation menu, click **Device > Telemetry > Services**.

☆ 🏠 > Devices > Services

<p><b>ARP</b></p> <p>Configured on: <u>5 devices</u></p> <p>Errors during enabling: 0 devices</p> <p>Last collection cycle errors: 0 devices</p> <p>Used by collectors: 0 collectors</p>	<p><b>BGP</b></p> <p>Configured on: <u>5 devices</u></p> <p>Errors during enabling: 0 devices</p> <p>Last collection cycle errors: 0 devices</p> <p>Used by collectors: 0 collectors</p>	<p><b>DISK UTIL</b></p> <p>Configured on: <u>5 devices</u></p> <p>Errors during enabling: 0 devices</p> <p>Last collection cycle errors: 0 devices</p> <p>Used by collectors: 0 collectors</p>	<p><b>HOSTNAME</b></p> <p>Configured on: <u>5 devices</u></p> <p>Errors during enabling: 0 devices</p> <p>Last collection cycle errors: 0 devices</p> <p>Used by collectors: 0 collectors</p>
<p><b>INTERFACE</b></p> <p>Configured on: <u>5 devices</u></p> <p>Errors during enabling: 0 devices</p> <p>Last collection cycle errors: 0 devices</p> <p>Used by collectors: 0 collectors</p>	<p><b>INTERFACE COUNTERS</b></p> <p>Configured on: <u>5 devices</u></p> <p>Errors during enabling: 0 devices</p> <p>Last collection cycle errors: 0 devices</p> <p>Used by collectors: 0 collectors</p>	<p><b>LAG</b></p> <p>Configured on: <u>2 devices</u></p> <p>Errors during enabling: 0 devices</p> <p>Last collection cycle errors: 0 devices</p> <p>Used by collectors: 0 collectors</p>	<p><b>LLDP</b></p> <p>Configured on: <u>5 devices</u></p> <p>Errors during enabling: 0 devices</p> <p>Last collection cycle errors: 0 devices</p> <p>Used by collectors: 0 collectors</p>
<p><b>MAC</b></p> <p>Configured on: <u>5 devices</u></p> <p>Errors during enabling: 0 devices</p>	<p><b>RESOURCE UTIL</b></p> <p>Configured on: <u>5 devices</u></p> <p>Errors during enabling: 0 devices</p>	<p><b>ROUTE</b></p> <p>Configured on: <u>5 devices</u></p> <p>Errors during enabling: 0 devices</p>	<p><b>XCVR</b></p> <p>Configured on: <u>5 devices</u></p> <p>Errors during enabling: 0 devices</p>







**NOTE:** The raw data that Apstra collects does not appear in 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 generates 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.

The following probes are enabled by default:

<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 adds the data into the probe for analysis.

You can access the list of predefined probes from the **Instantiate Predefined Probe** dialog box.

## + Create Probe ▾

-  New Probe
-  Instantiate Predefined Probe
-  Import 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.

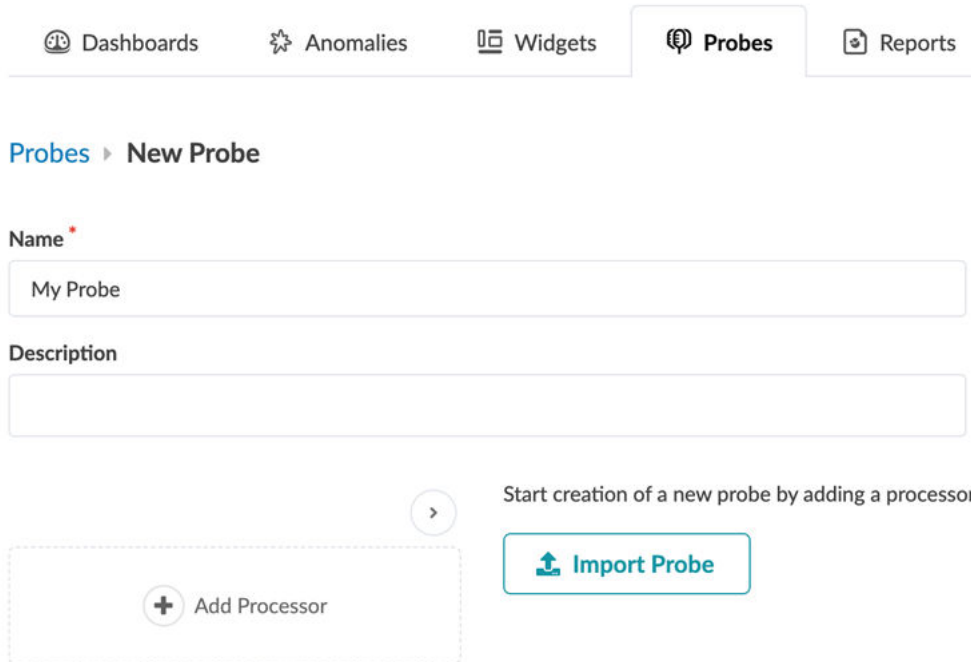
Create Another?
 Create

For detailed information about probes, see [Predefined Probes \(Analytics\)](#) 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 in the Apstra GUI.



The screenshot shows the 'Probes' section of the Apstra GUI. At the top, there is a navigation bar with icons and labels for 'Dashboards', 'Anomalies', 'Widgets', 'Probes' (which is highlighted), and 'Reports'. Below this, the breadcrumb 'Probes > New Probe' is visible. The form contains a 'Name' field with a red asterisk and the text 'My Probe'. Below it is a 'Description' field. To the right of the description field, there is a text prompt: 'Start creation of a new probe by adding a processor.' Below this prompt, there is a dashed box containing a plus sign icon and the text 'Add Processor'. To the right of this dashed box is a blue button with a download icon and the text 'Import Probe'.

For the probe to be functional, you'll need 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 analytics processor. Analytic processors are also referred to as *source processors*.

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

### Add Processor

**Processor Type \***  
 Interface Counters  
 Interface Counters Processor.  
 Selects interfaces according to the configuration and outputs counter stats of the specified types (e.g. 'tx\_bytes').

**Processor Name \***  
 Interface Counters  
 Has no inputs.

**Output Stage Name: out \***  
 Interface Counters

**Add**

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

Search stages...

- 13<sup>3</sup> Interface Counters
- Interface Counters

#### Stage: Interface Counters

▶ Query: All

System ID <sup>⊕</sup> <sub>⊕</sub>	Interface <sup>⊕</sup> <sub>⊕</sub>	Value <sup>⊕</sup> <sub>⊕</sub>
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 didn't 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 9 .

# Custom Telemetry Collection Overview

## IN THIS SECTION

- [Example Use Cases | 9](#)

Juniper Apstra Custom Telemetry Collection is a new feature introduced in Apstra 4.2.0. You can now define new telemetry services for monitoring data for Apstra to analyze. You can also tailor analytics on your data based on your specific business needs.

Previously, adding a telemetry service to collect new data involved substantial development work that required advanced programming and familiarity with the IBA software development kit (SDK).

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

- Run the Junos CLI show commands that provides 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.
- Create an IBA probe that utilizes the data from the telemetry collector.

## Example Use Cases

Here are some examples of what you can do with the custom telemetry collection:

- Monitor various counters (firewall filter match count, IRB interface statistics, and so forth).
- 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 forth).

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 BFD sessions as an example.

Let's go!

# Creating a Custom Telemetry Collector

## SUMMARY

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

## IN THIS SECTION

- [Execute the CLI Command | 10](#)
- [Identify the Key and Value of Interest from the CLI Output | 13](#)
- [Create a Service Schema | 14](#)
- [Create a Collector | 15](#)
- [Validate That the Collector Is Working | 19](#)

In this topic, we'll walk you through creating your own custom telemetry service using BFD. In our example, the telemetry service collects the state of the BFD sessions that you just configured. Our goal is to alert operations that a BFD session is down.

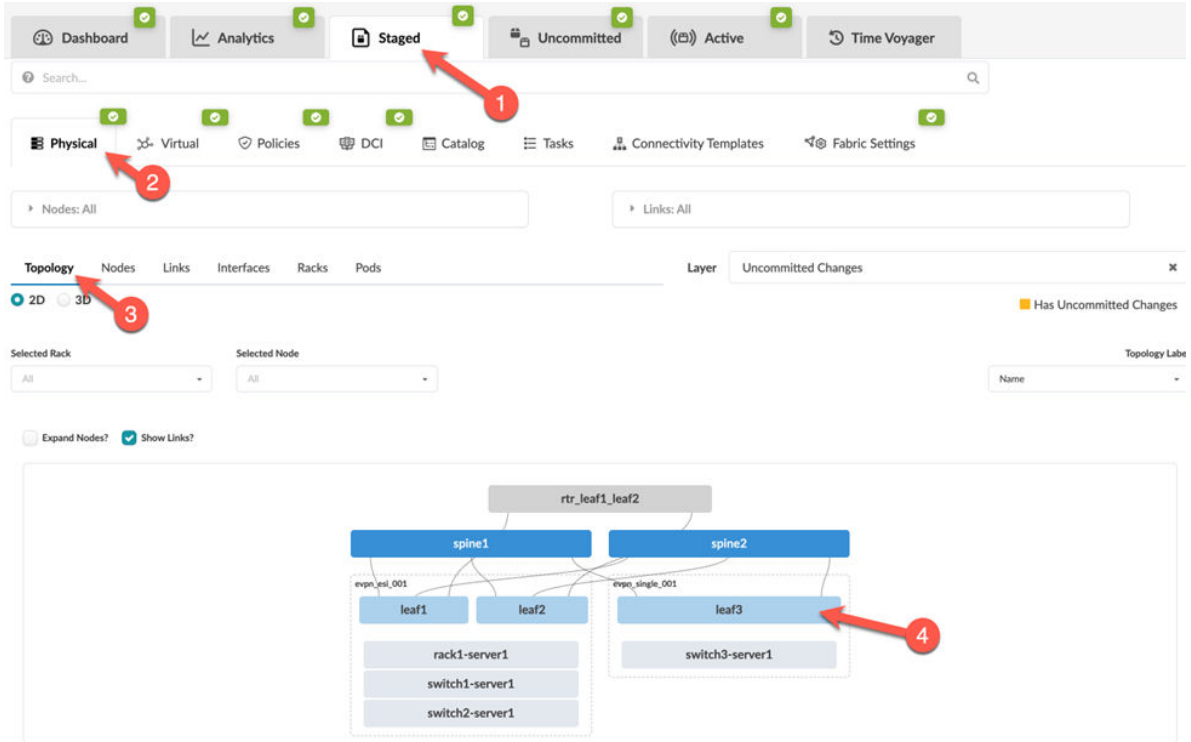
## Execute the CLI Command

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 the show commands without opening a CLI session, its primary purpose is to help you create your own custom telemetry collectors.

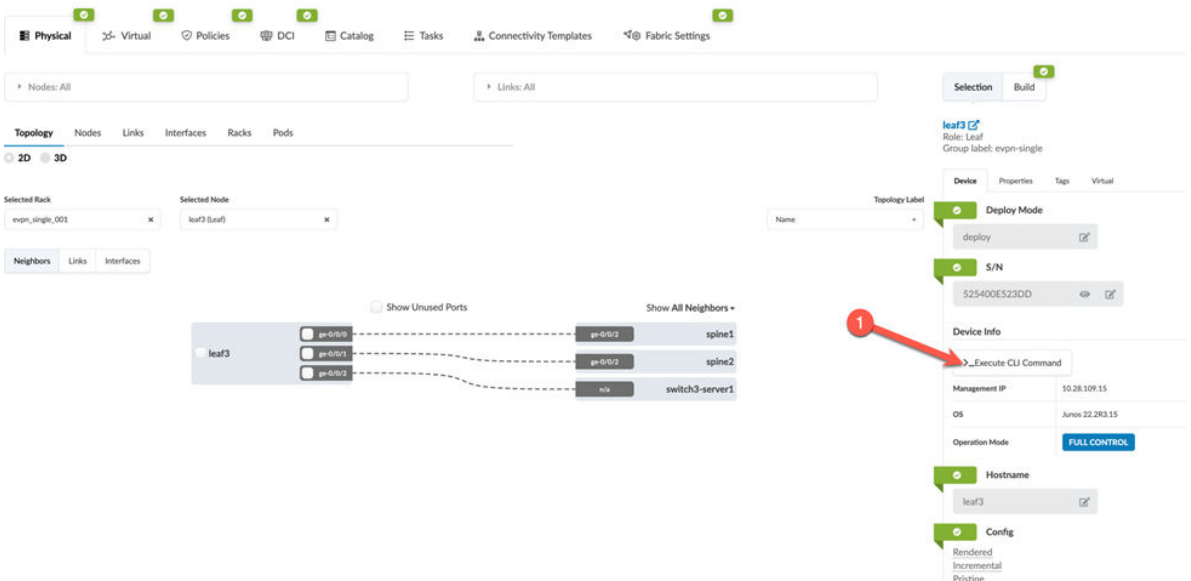
You can execute CLI commands from within a staged or active blueprint (shown in our example), or from the **Devices > Managed Devices** page.

To execute the CLI command:

1. From your deployed blueprint, select **Analytics > Staged Physical Topology** (or **Staged > Physical > Nodes**), then select your Juniper device node.



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



In the dialog box that opens, select how you want to view the results: Text Mode, XML Mode, or JSON mode. Below are examples of **Text Mode** and **XML Mode**.

Accepts "Show" commands only **1**

Supports auto-completion of arguments\*.

Most piping command supported, so you can filter the command's output through expressions. **2**

Execute CLI Command

S/N: 52540011DEFF Management IP: 10.28.126.13 Hostname: leaf1

show route summary | match " RIB | FIB | VRF"

Supports executing the command also in XML. **3**

Copy to clipboard **4**

```

Highwater Mark (All time / Time averaged watermark)
RIB unique destination routes: 685 at 2023-05-22 11:12:33 / 571
RIB routes : 1063 at 2023-05-22 11:13:17 / 870
FIB routes : 67 at 2023-05-22 11:09:36 / 36
VRF type routing instances : 4 at 2023-05-22 10:34:13

```

**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](#).

Now, run the same show command (show route summary), but select **XML Mode** this time.

**Execute CLI Command**

S/N: 525400E523DD Management IP: 10.28.109.15 Hostname: leaf3

show bfd session

XML Mode ▶ Execute

```

<rpc-reply xmlns:junos="http://xml.juniper.net/junos/22.2R0/junos">
  <bfd-session-information xmlns="http://xml.juniper.net/22.2R0/junos-bfd" junos:style="brief">
    <bfd-session>
      <session-neighbor>10.0.0.0</session-neighbor>
      <session-state>Up</session-state>
      <session-interface>ge-0/0/0</session-interface>
      <session-detection-time>9.000</session-detection-time>
      <session-transmission-interval>3.000</session-transmission-interval>
      <session-adaptive-multiplier>3</session-adaptive-multiplier>
    </bfd-session>
    <bfd-session>
      <session-neighbor>10.0.0.0</session-neighbor>
      <session-state>Up</session-state>
      <session-interface>ge-0/0/0</session-interface>
      <session-detection-time>9.000</session-detection-time>
      <session-transmission-interval>3.000</session-transmission-interval>
      <session-adaptive-multiplier>3</session-adaptive-multiplier>
    </bfd-session>
    <bfd-session>
      <session-neighbor>172.16.0.4</session-neighbor>
      <session-state>Up</session-state>
      <session-interface>ge-0/0/0.0</session-interface>
      <session-detection-time>3.000</session-detection-time>
      <session-transmission-interval>1.000</session-transmission-interval>
      <session-adaptive-multiplier>3</session-adaptive-multiplier>
    </bfd-session>
    <bfd-session>
      <session-neighbor>172.16.0.10</session-neighbor>
      <session-state>Up</session-state>
      <session-interface>ge-0/0/1.0</session-interface>
      <session-detection-time>3.000</session-detection-time>
      <session-transmission-interval>1.000</session-transmission-interval>
      <session-adaptive-multiplier>3</session-adaptive-multiplier>
    </bfd-session>
    <sessions-shop>2</sessions-shop>
    <sessions-hshop>2</sessions-hshop>
    <sessions-up>4</sessions-up>
    <sessions>4</sessions>
    <clients>4</clients>
    <cumulative-transmission-rate>2.7</cumulative-transmission-rate>
    <cumulative-reception-rate>2.7</cumulative-reception-rate>
  </bfd-session-information>
</cli>
<banner></banner>
</cli>
</rpc-reply>

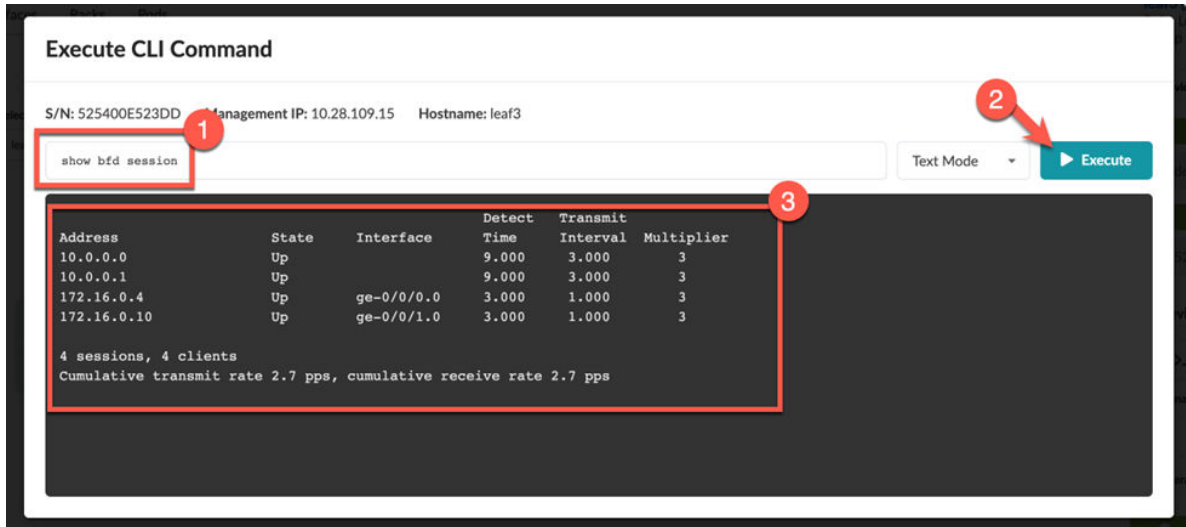
```

In the XML output, the XML path (BFD session) information is highlighted. This session information is what we'll use to create our telemetry collection service.

## Identify the Key and Value of Interest from the CLI Output

This example shows you how to use the CLI show command to view the neighbor addresses and state information (**Up** or **Down**) for your BFD session.

1. Enter the CLI show command (in this example, show bfd session).
2. Click **Execute** to view the BFD session information.

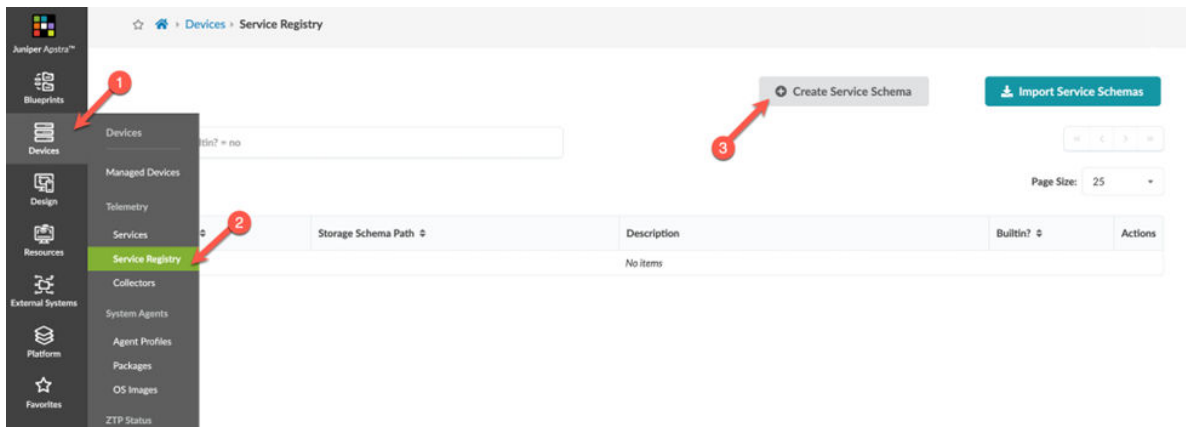


## Create a Service Schema

To create a custom service collector, you first need to create a service schema to define how you want your data to be structured and stored.

**NOTE:** A single telemetry service schema can have multiple collectors associated with it.

1. From the left navigation menu, navigate to **Devices > Service Registry**, then click the **Create Service Schema**.



2. In the dialog box that opens, define your schema. The schema determines how you want the collector output to be structured.



Execute CLI Command

S/N: 52540011DEFF Management IP: 10.28.126.13 Hostname: leaf1

show bfd session |

```

Address          State      Interface  Detect  Transmit
-----          -
192.168.0.1      Down      ne-0/0/4.1 0.000   2.000   3
192.168.0.5      Down      ne-0/0/4.2 0.000   2.000   3
192.168.0.9      Down      ne-0/0/4.2 0.000   2.000   3
  
```

3 sessions, 3 clients  
Cumulative transmit rate 1.5 gpps, cumulative receive rate 0.0 gpps

I want this as the Service's value

I want this as the Service's key

Edit Service Schema

Name\* BFD\_Status Value Type\* string 2

Description

Telemetry Keys

Key #1\* neighbor 1

+ Add Key

3 Update

### 3. Map the Telemetry Keys and Value Type.

The telemetry key and value type is collection of *key-value pairs* in Apstra and are defined as follows:

- Telemetry key: String that identifies the interface name.
- Value type: Piece of data that the probe executes against. The value type is usually a string (text), but could also be an integer (whole number).

As shown in our example in Step 2, we defined the **Telemetry key** as *neighbor* and the **Value Type** as *string*.

### 4. Click **Update** to finish creating your schema.

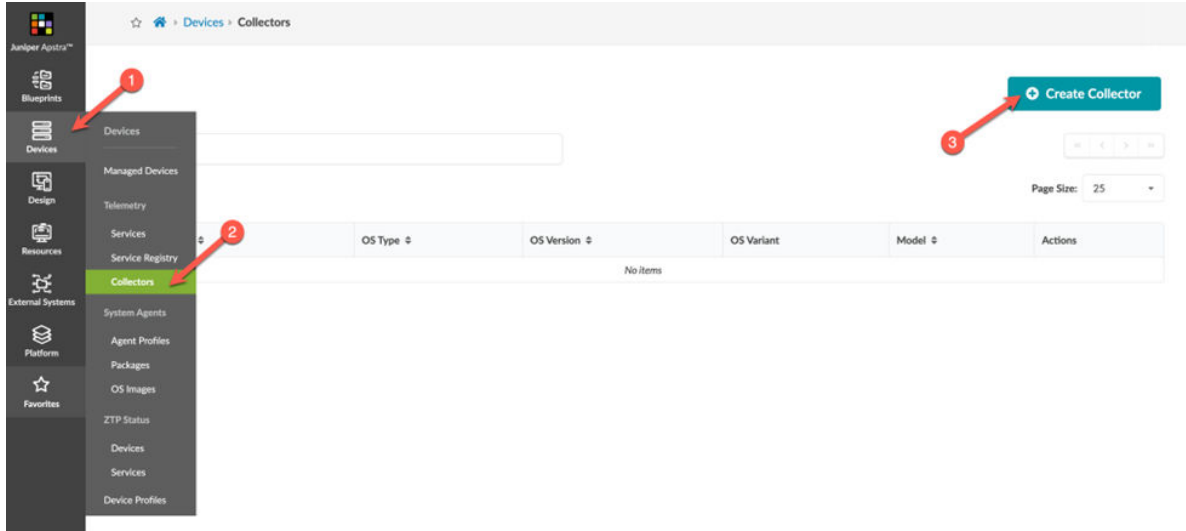
## Create a Collector

So far, you defined the data to want to collect and how the data will be organized and structured. Our final step is to create a collector.

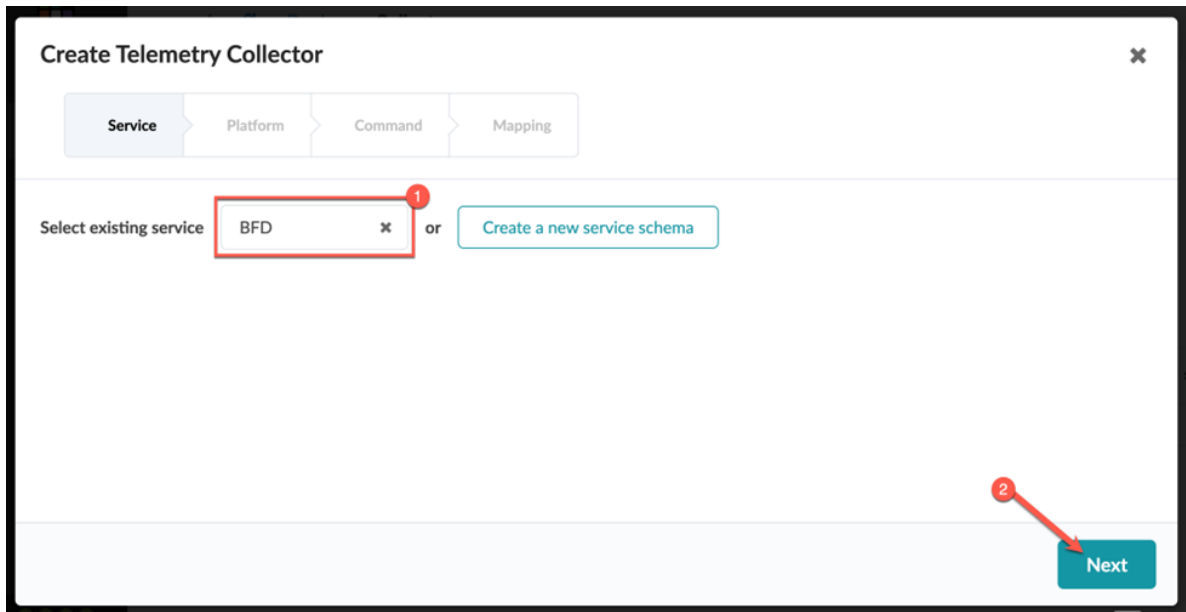
**NOTE:** A single telemetry service schema can have multiple collectors associated with it.

To create a collector:

1. From the left navigation pane, navigate to **Devices > Collectors > Create Collector**.



- 2. Select the existing service schema (BFD) you created in "Create a Service Schema" on page 14 , the click **Next**.



- 3. Select the platform (OS, OS Variant, OS version, and Model) and devices to target for your telemetry collection. Defining a mix of these inputs enables you to be very broad or very granular. For example, you might have a use case where you want to apply telemetry just on the border leaf devices.

**Create Telemetry Collector**

Service **Platform** Command Mapping

OS <sup>1</sup> junos

OS Variant <sup>2</sup> junos

OS Version <sup>3</sup> 22.2r2

Model <sup>4</sup>

**Target Devices** 1-5 of 5

Management IP	Device Key	Hostname	Vendor	OS	Hardware Model
10.28.109.11	525400E9FBCF	spine1	Juniper	Junos 22.2R3.15	VIRTUAL-EX9214
10.28.109.12	5254008BC60E	spine2	Juniper	Junos 22.2R3.15	VIRTUAL-EX9214
10.28.109.13	525400E542CF	leaf1	Juniper	Junos 22.2R3.15	VIRTUAL-EX9214
10.28.109.14	525400BF80E5	leaf2	Juniper	Junos 22.2R3.15	VIRTUAL-EX9214
10.28.109.15	525400E523DD	leaf3	Juniper	Junos 22.2R3.15	VIRTUAL-EX9214

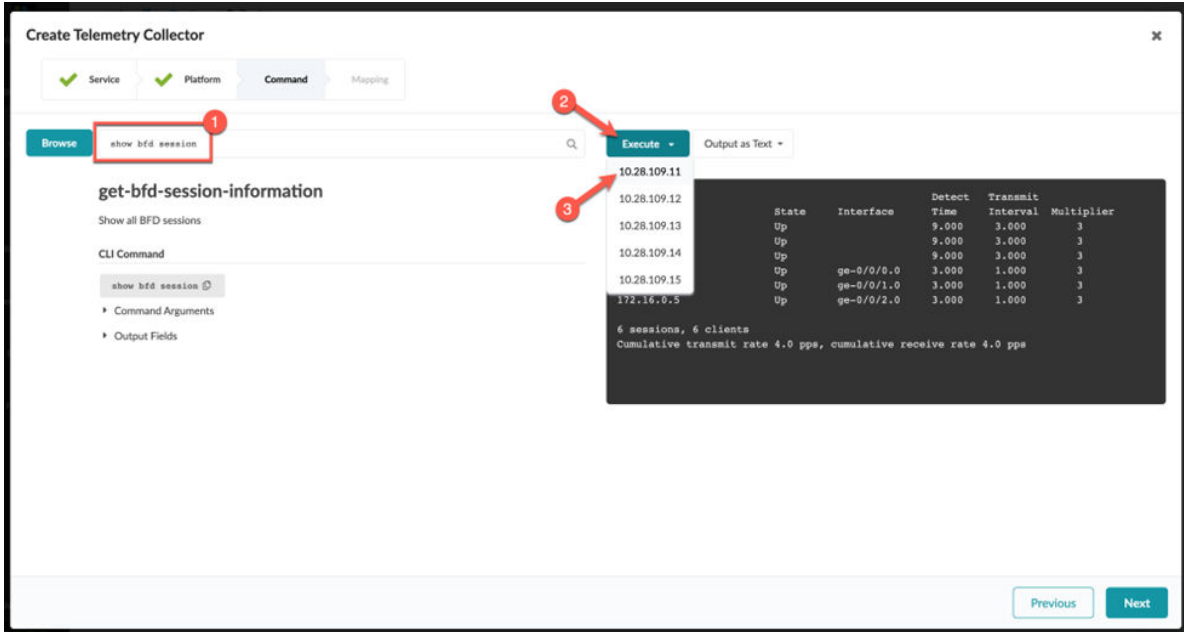
Previous Next

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

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

**NOTE:** If you do not define a Junos-evo collector for Junos-evo devices, the collector uses the corresponding Junos definition. This means, if you use the same command between Junos and Junos-evo, you can create a single Junos collector definition for that service. If the command resides only on Junos-evo, you'll want to create a single collector definition for Junos-evo.

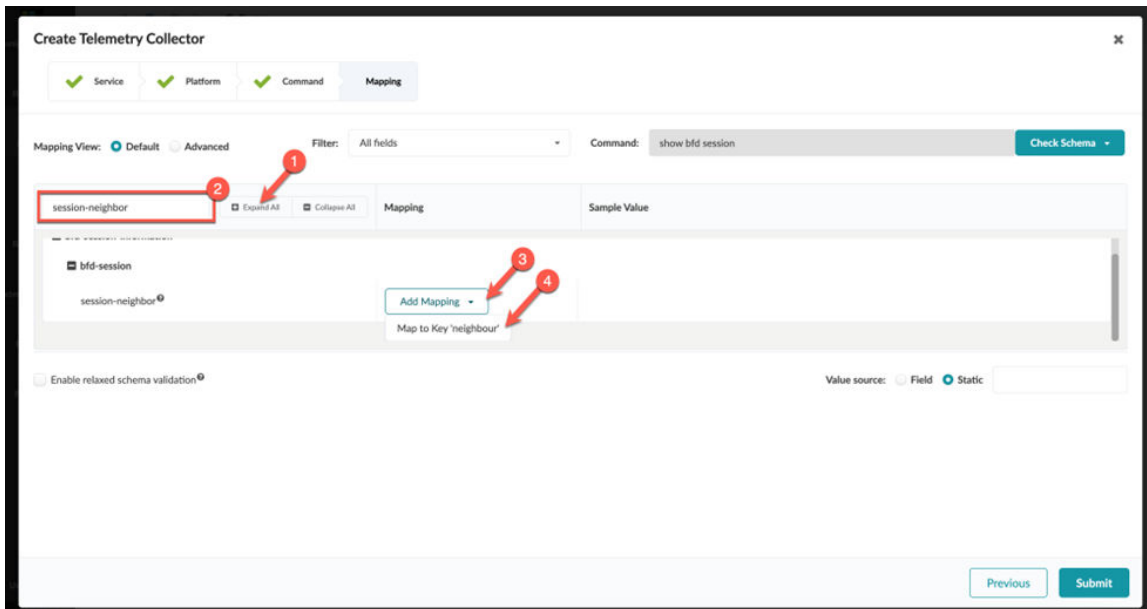
- b. Select the **OS Variant** the device belongs to and determine the CLI schema for a given device.
- c. Select the **minimum OS Version** the device must run for the collector to execute. If multiple collector definitions, with different OS versions exist for the same service, the collector automatically 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 gather the data you want to collect from the device (in this example, show bfd session).



## 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 you defined in your schema earlier.

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



- Click **Add Mapping**.

- Assign **session-neighbor** to the key (in this example, **neighbor**).

- d. To map the value, select **Field** as the **Value source**. In our example, we populated the value based on the dynamic session-state field returned by the CLI command.

The screenshot shows the 'Create Telemetry Collector' interface in the 'Mapping' tab. The 'session-state' field is highlighted with a red box and a '2'. The 'Value source' dropdown is set to 'Field' and is also highlighted with a red box and a '1'. A dropdown menu is open for 'Add Mapping', with 'Map to Value' selected, highlighted with a red box and a '4'. A red arrow points from '3' to the 'Add Mapping' dropdown. Another red arrow points from '4' to the 'Map to Value' option. The 'Command' field contains 'show bfd session' and the 'Check Schema' button is visible. The 'Previous' and 'Submit' buttons are at the bottom right.

- e. Search for the **session-state** field, then click **Add Mapping**.
- f. Assign session-state to map the value, then click **Submit**.

## Validate That the Collector Is Working

Finally, in **Advanced** view, validate that the collector is working. Verify that the query and test results match your expected results.

Mapping View:  Default  Advanced

Command: show bfd session

Data Accessors

Name	Path
neighbour	/bfd-session-information/bfd-session/session-neighbor
value	/bfd-session-information/bfd-session/session-state

Keys

Name	Data Expression
neighbour	neighbour

Value Expression

value

Query Results

neighbour	Value
10.0.0.2	Up
10.0.0.3	Up
10.0.0.4	Up
172.16.0.1	Up
172.16.0.3	Up
172.16.0.5	Up

Test Query

10.28.109.11  
10.28.109.12  
10.28.109.13  
10.28.109.14  
10.28.109.15

Previous Submit

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 and click **Submit**.

**Edit Telemetry Collector**

✓ Settings ✓ Platform ✓ Command **Mapping**

Mapping View:  Default  **Advanced** ? Expression Reference ...

Data Accessors

Name	Path
table	/route-information/route-table/table-name
value	/route-information/route-table/total-route-count

+ Add Accessor

Keys

Name	Data Expression
table	table
Value Expression	int(value)

Filter Expression

Enable relaxed schema validation<sup>®</sup>

## Using Custom Telemetry Data in an IBA Probe

### SUMMARY

This topic describes how to create an IBA probe and detect and store any anomalies in a historical database for reference.

### IN THIS SECTION

- [Create a Probe | 22](#)
- [Customize a Probe | 26](#)
- [Performing Analytics | 27](#)
- [Raising Anomalies and Storing Historical Data | 29](#)

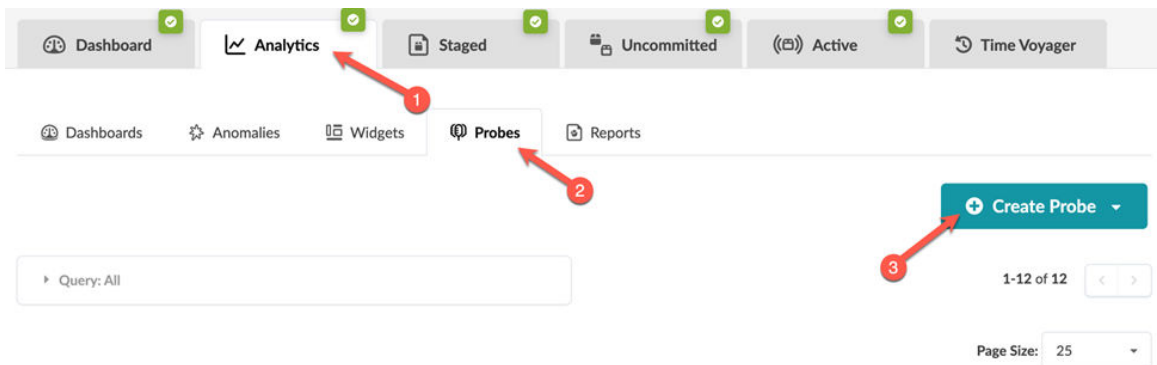
So far in our walkthrough, we've created a custom telemetry collector service that defines the data you want to collect from your devices. Now let's ingest this data into IBA probes in your blueprint so that Apstra can visualize and analyze the data.

## Create a Probe

First, we'll create a new probe in your deployed blueprint so that Apstra can ingest data from your custom telemetry collector. In this example, we'll focus on a minimal set of configurations for the simple use case of visualizing BFD session data and generating anomalies (alerts) when sessions are down.

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

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



2. Enter a name and (optional) description (in this example, **BFD-Example-Probe**), then click **Add Processor**.



Dashboard Analytics Staged Uncommitted Active Time Voyager

Dashboards Anomalies Widgets **Probes** Reports

Probes > New Probe

Name\* BFD-Example-Probe

Tags No tags

Description

Enabled  Disabled probes don't produce data and don't raise anomalies.

Start creation of a new probe by adding a processor. Alternatively, you can import a probe from JSON.

+ Add Processor Import Probe

3. Select a processor type. For our example, we selected the **Extensible Service Data Collector** processor.

### Add Processor

Processor Type\* Extensible Service Data Collector Processor.

Extensible Service Data Collector Processor.  
Collects data supplied by a custom service, that is not one of 'lldp', 'bgp', or 'interface'.

Processor Name\* BFD Status  
This processor has no inputs.

Output Stage Name: out\* BFD Status

Add

4. Click **Add** to add the processor to the probe. See the [Juniper Apstra User Guide](#) for information about the different processors.
5. Click **Create** to create the probe and return to the table view.
6. To the right of the Graph Query field click the **Select a predefined graph query** button, then select **DC – All managed devices (any role)** from the **Predefined Query** drop-down.  
This query determines the scope within the blueprint in which the telemetry collection is executed. This means if a device in your blueprint is not matched by the graph query, the telemetry collection service will not start for that device.

### 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 \*

DC - All managed devices (any role)

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

Update

The graph query specifically matches all system nodes in the graph database of your blueprint. Each managed device, such as a leaf switch or spine switch, shows as a system node in the graph.

In the **Predefined Query** we selected above, the query matches all nodes of the type system, which in deploy mode has a role of leaf, access, spine, or superspine.

7. Click **Update** to return to the table view.

Processor: Extensible Service Data Collector

Graph Query <sup>1</sup>

```
node('system', name='system', role=is_in(['leaf', 'spine']))
```

One or more queries on the graph to get nodes to be monitored. Results from all queries are concatenated and they must have the same named nodes as names used in properties.

Query Expansion

No items.

Query Group By

No names specified

Query Tag Filter

Tag Filter Operation

and

System ID <sup>2</sup>

```
system.system_id
```

Service name <sup>3</sup>

BFD

Service interval

2 Minutes

Service input

Execution count

-1

Data Type <sup>4</sup>

Dynamic Text

Value Map

Ingestion filter

Additional keys

Create Probe <sup>5</sup>

8. In the **System ID** field, enter `system.system_id`. This entry tells the probe that the graph query will match on your managed devices under the name `system` (`name='system'`).

The attribute `system_id` on each system nodes refers to the system ID of each device. This attribute is what Apstra uses to uniquely identify each device.

9. Select **BFD** from the **Service name** drop-down list.

10. Select the **Data Type**.

- Select **Dynamic Text** if your telemetry service collects string as the value type.
- Select **Dynamic Number** if the service collects integer as the value type.

In our example, we chose **Dynamic Text** because the BFD session state contains the string values Up and Down.

11. Click **Create Probe**.

12. Navigate to the output stage of the data collector processor to verify that the probe is correctly ingesting data from your custom telemetry collector.

The screenshot shows the Apstra interface for a probe named "BFD-Example-Probe". The probe is operational and has no anomalies. The "Stage: BFD Status" is selected, and the data source is "Real Time". The output table shows the following data:

System ID	Neighbor	Value	Updated
5254001BFCOD spine2 Spine	10.0.0.2	Up	a few seconds ago
5254001BFCOD spine2 Spine	10.0.0.3	Up	a few seconds ago
5254001BFCOD spine2 Spine	10.0.0.4	Up	a few seconds ago
5254001BFCOD spine2 Spine	172.16.0.11	Up	a few seconds ago

Congratulations! You successfully create a probe!

## Customize a Probe

We created a working probe that collects the BFD state for every device in your network. Now let's explore a couple of useful customization options to fine-tune your probe.

### Service Interval

The service interval determines how often your telemetry collection service fetches data from devices and ingests them into the probe. This interval is an important parameter to be aware of because an

overly aggressive interval can cause excessive load on your devices. The optimal interval will depend on the data you are collecting. For example, a collector fetching the content of a large routing table with thousands of entries can cause a higher load than collecting the status of a handful of BFD sessions.

#### Service interval

### 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 the telemetry collection only 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

system

##### Matcher

Is In

##### Tags

storage

+ Add Tag Filter

Filters named nodes in the graph queries by assigned tags.

Displaying the raw data from your custom telemetry collector shows only the raw data, so it may be difficult to conclude whether it signifies your network's normal or anomalous state. With Aspra, you are proactively notified when any anomaly is detected.

## Performing Analytics

An IBA probe functions as an analytics pipeline. All IBA probes have at least one source processor at the start of their pipeline. In our example, we added an **Extensible Service Data Collector** processor that ingests data from your custom telemetry collector.

You can chain additional processors in the probe to perform additional analytics on the data to provide more meaningful insight into your network's health. These processors are referred to as *analytics processors*.

Analytics processors enable you to aggregate and apply logic to your data and define an intended state (or a reference state) to raise anomalies. For instance, you might not be interested in instantaneous values of raw telemetry data, but rather in an aggregation or trends.

Analytics processors aggregate information such as calculating average, min/max, standard deviation, and so on. You can then compare the aggregated data against expectations so that you can identify whether the data is inside or outside a specified range, in which case an anomaly is raised. You might also want to check whether this anomaly is sustained for a period of time and exceeds a specific threshold. An anomaly is flagged only when the threshold is exceeded to avoid flagging anomalies for transient or temporary conditions. You can achieve this by configuring a Time\_In\_State processor.

[Table 1 on page 28](#) describes the different types of analytics processors.

**Table 1: Analytics Processors**

Type of Processor	Description
<p>Range processors</p> <p>Processor names: Range, State, Time_In_State, Match_String</p>	<p>Range processors define reference state and generate anomalies.</p>
<p>Grouping processors</p> <p>Processor names: Match_Count, Match_perc, Set_Count, Sum, Avg, Min, Max, and Std_Dev</p>	<p>Group processors aggregate and process data before feeding into the range processors. These processors can:</p> <ul style="list-style-type: none"> <li>• Produce a per-device count of protocol states.</li> <li>• Produce a sum of counters from multiple devices to represent a total over the fabric.</li> </ul>
<p>Multi-input processors</p> <p>Processor names: Match_Count, Match_perc, Set_Count, Sum, Avg, Min, Max, and Std_Dev</p>	<p>Analytics processors take input from multiple stages. These processors can:</p> <ul style="list-style-type: none"> <li>• Produce a single output data set that is a union of input from multiple stages.</li> <li>• Perform a logical comparison between input from multiple stages.</li> </ul>

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

**NOTE:** Multi-input processors are not supported for dynamic data types (dynamic text or dynamic number).

In the next section, we'll configure our BFD example probe to detect and raise anomalies.

## Raising Anomalies and Storing Historical Data

Now we'll configure our example probe to detect and raise anomalies if a BFD session goes down and store the anomalies in a historical database for reference.

1. First, add a second processor to the probe you created in ["Create a Probe" on page 22](#) , then click **Add Processor**.
2. Select the **Match Count** processor and give the processor a descriptive name, such as Down sessions count.

The match count processor counts the number of BFD sessions in the Down state and groups the count by device.

3. Configure the second processor, then Enter **Down** in the **Reference State** field.

This processor configures the probe pipeline so that data from the previous processor is fed into each other.

Processor: Down sessions count Match Count

Inputs

Input Name	Stage Name	Column Name
in	BFD Status	value

Properties

Group by

system\_id

Reference State

Down

Enable Streaming

When you update the probe, the output shows the number of BFD sessions in the **Down** state by each device.

Stage: Down sessions count Dynamic

Show Context

Query: All 1-5 of 5 Page Size: 25

System ID	Total count	Value	Updated
S2540018FC0D spine2 <small>Spine</small>	6		a few seconds ago
S2540030AAAA leaf3 <small>Leaf</small>	4		a few seconds ago
S2540039E27C leaf1 <small>Leaf</small>	10		a few seconds ago
S2540078E1F0 spine1 <small>Spine</small>	6		a few seconds ago
S25400F0A234 leaf2 <small>Leaf</small>	10		a few seconds ago

- Add the third and final processor. This processor produces anomalies to alert you when there are one or more BFD sessions in the Down state.
- Click **Add Processor**, then select the **Match Count** processor.  
Give the processor a descriptive name (in this example, **BFD anomaly (down > 0)**), then click **Add**.

### Add Processor

**Processor Type** \*

Range ▼

**Processor Name** \*

BFD anomaly (down > 0)

**Output Stage Name: out** \*

BFD anomaly (down > 0)

Checks that a value is in a range.  
According to the specified range, configures a check for the input series. This check returns an anomaly value if a series aggregation value, such as a last value, sum, avg, etc, is in the range. This aggregation type is configured by the 'property' attribute, which is set to 'value' if not specified. The output series contains anomaly values, such as 'true' and 'false'.

**Add**

- Configure the third processor.



Processor: BFD anomaly (down > 0) Range

Inputs

Input Stage <sup>+</sup>

Input Name	Stage Name	Column Name
in	Down sessions count	value

Properties

Graph Query

No items.

[+ Add Graph Query](#)

One or more queries on the graph for probe parametrization. Results of the queries can be accessed using the "query\_result" variable with the appropriate index. For example, if querying property set nodes under name "ps", the result will be available as "query\_result[0][ps]".

Anomalous Range <sup>+</sup>

More than or equal to = 1

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

- a. Enter the **Input Stage – Stage Name**, then select **value** for the **Column name**. In our example, we defined the stage name as **Down sessions count**.
  - b. Set the **Anomalous Range** to **More than equal to** and **1**.
  - c. Click **Raise Anomaly**.
7. While still in the probe configuration interface, click **Enable Metric Logging**, then select the output stage for your second processor. This action enables historical logging of data.
  8. Click **Update the Probe**.

If you have any BFD sessions in the Down state, the probe generates anomalies for the BFD sessions.

Stage: Down sessions count Dynamic

Show Context

Query: All 1-5 of 5 Page Size: 25

System ID	Total count	Value	Updated
S2540018FC0D spine2 <small>Spine</small>	6		a few seconds ago
S2540030AAAA leaf3 <small>Leaf</small>	4		a few seconds ago
S2540039E27C leaf1 <small>Leaf</small>	10		a few seconds ago
S2540078E1F0 spine1 <small>Spine</small>	6		a few seconds ago
S25400F0A234 leaf2 <small>Leaf</small>	10		a few seconds ago

9. Check **Enable Streaming** in the probe configuration.

**Enable Streaming**  
Makes samples of output stages streamed if enabled.

**Additional keys**  
No extra keys for graph query defined.

+ Add Key

Each additional key/value pair is used to extend properties of output stages where value is with respective key.

Update Probe Cancel

10. Finally, select the **Data source: Time Series** view to see the history of changes in the data value monitored by this stage.

Stage: Detect BFD Down Dynamic Persisted: 1 day / 8.03 KB

Data source: Time Series

Aggregation type: last × Aggregation: Off Last 1 Hour

Query: All >\_ 1-25 of 36 Page Size: 25

System ID	Neighbor	Anomaly	Value
S254001BFC00 spine2	10.0.0.2	No anomaly	No data
S254001BFC00 spine2	10.0.0.3	No anomaly	<div style="border: 2px solid red; padding: 2px;"> <span style="background-color: #f8d7da; padding: 2px;">true</span> <span style="background-color: #d4edda; padding: 2px;">false</span> </div>
S254001BFC00 spine2	10.0.0.4	No anomaly	No data
S254001BFC00 spine2	172.16.0.11	No anomaly	No data
S254001BFC00 spine2	172.16.0.7	No anomaly	<div style="border: 1px solid gray; padding: 2px;"> <span style="background-color: #d4edda; padding: 2px;">true</span>            2023-09-22 23:14:17.127            59 seconds 984 milliseconds         </div>
S254001BFC00 spine2	172.16.0.9	No anomaly	<div style="border: 2px solid red; padding: 2px;"> <span style="background-color: #d4edda; padding: 2px;">false</span> <span style="background-color: #f8d7da; padding: 2px;">true</span> <span style="background-color: #d4edda; padding: 2px;">false</span> </div>

## Monitoring the Health of the Telemetry Service

An important factor to consider when creating your custom telemetry collection is to ensure that the service does not cause excessive load on your devices. Some telemetry services can cause a higher load on your devices depending on the CLI show command and the data you are collecting. When you configure a collector to execute at short intervals you can possibly overload your devices, potentially impacting traffic forwarding.

By default, Apstra provides an IBA telemetry health probe that enables you to monitor the health of telemetry services, including any custom services and collectors you configured.

To monitor the health of your telemetry services:

1. From your blueprint, navigate to **Analytics > Probes**.
2. Select the **Device Telemetry Health** probe from the table.
3. Click **Query: All** to filter the data in the table.

The screenshot shows the 'Probes' section of a dashboard, specifically the 'Device Telemetry Health' probe. The probe is operational and has no anomalies. The 'Stage: Telemetry Stats' is active, showing a table of data for various services. The table has columns for System ID, Service name, Collection Type, Has Service Started, Run Count, Success Count, Failure Count, Timeout Count, Underrun Count, and three 'Did Last Execution' columns. The first row shows data for 'BFD' with a Run Count of 4427 and Success Count of 4427. The 'Service name' filter is set to 'BFD'.

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
5254001BFC0D spine2 Spine	BFD	polling	true	4427	4427	0	0	0	false	false	false

For example, to display data for your new custom telemetry service, select a service name from the **Service name** drop-down filter. In our example, the service name is **BFD**.

The screenshot shows the query builder interface for the 'Service name = BFD' filter. The 'Service name' field is highlighted with a red box, indicating it is the selected filter.

▼ Query: Service name = BFD






























System ID  
=

System Hostname  
=

System Role  
=

Service name  
= BFD

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

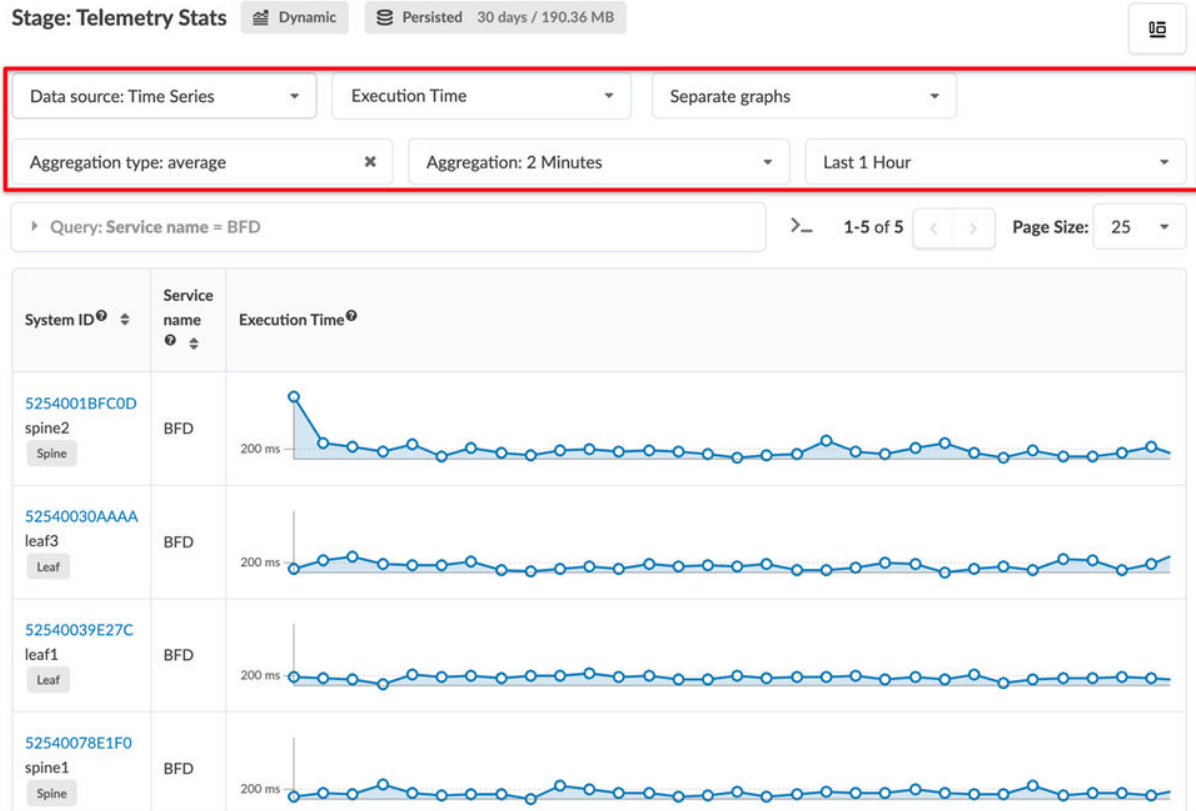
System ID  	Service name  	Collection Type  	Has Service Started  	Run Count  	Success Count  	Failure Count  	Timeout Count  	Underrun Count  	Did Last Execution Fail  	Did Last Execution Timeout  	Did Last Execution Underrun  	Execution Time
5254001BFC0D spine2 	BFD	polling	true	4439	4439	0	0	0	false	false	false	0.16083366610
52540030AAAA leaf3 	BFD	polling	true	4732	4732	0	0	0	false	false	false	0.18458395120
52540039E27C leaf1 	BFD	polling	true	4439	4439	0	0	0	false	false	false	0.16871218800
52540078E1F0 spine1 	BFD	polling	true	4439	4439	0	0	0	false	false	false	0.18537315490
525400F0A234 leaf2 	BFD	polling	true	4439	4439	0	0	0	false	false	false	0.20656311970

Check the following:

- Ensure that the **Success Count** value has increased. If not, this could mean that your service is failing or that your custom collector is misconfigured.
- Check the **Execution Time**. Although the execution time can vary, if the time is close to or higher than the service interval, this can indicate a problem. If this is the case, tune your probe settings and set a higher service interval. For instructions on setting the service interval, see "[Customize a Probe](#)" on page 26 .

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

To see how your metrics are trending, switch to **Time Series** view under the **Data Source** drop-down.



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

## Summary

Congratulations! In this document, you learned:

- The fundamentals of Apstra Intent-Based Analytics.
- How to define a custom telemetry service to collect data from managed devices.
- How to create an IBA probe that visualizes and analyzes your data, and detect anomalies.

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

---

Juniper Networks, the Juniper Networks logo, Juniper, and Junos are registered trademarks of Juniper Networks, Inc. in the United States and other countries. All other trademarks, service marks, registered marks, or registered service marks are the property of their respective owners. Juniper Networks assumes no responsibility for any inaccuracies in this document. Juniper Networks reserves the right to change, modify, transfer, or otherwise revise this publication without notice. Copyright © 2024 Juniper Networks, Inc. All rights reserved.