Juniper Networks | Engineering Simplicity

Juniper Apstra 4.2.0 Custom Telemetry Collection Guide

Published
2024-03-04

RELEASE

Table of Contents

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!

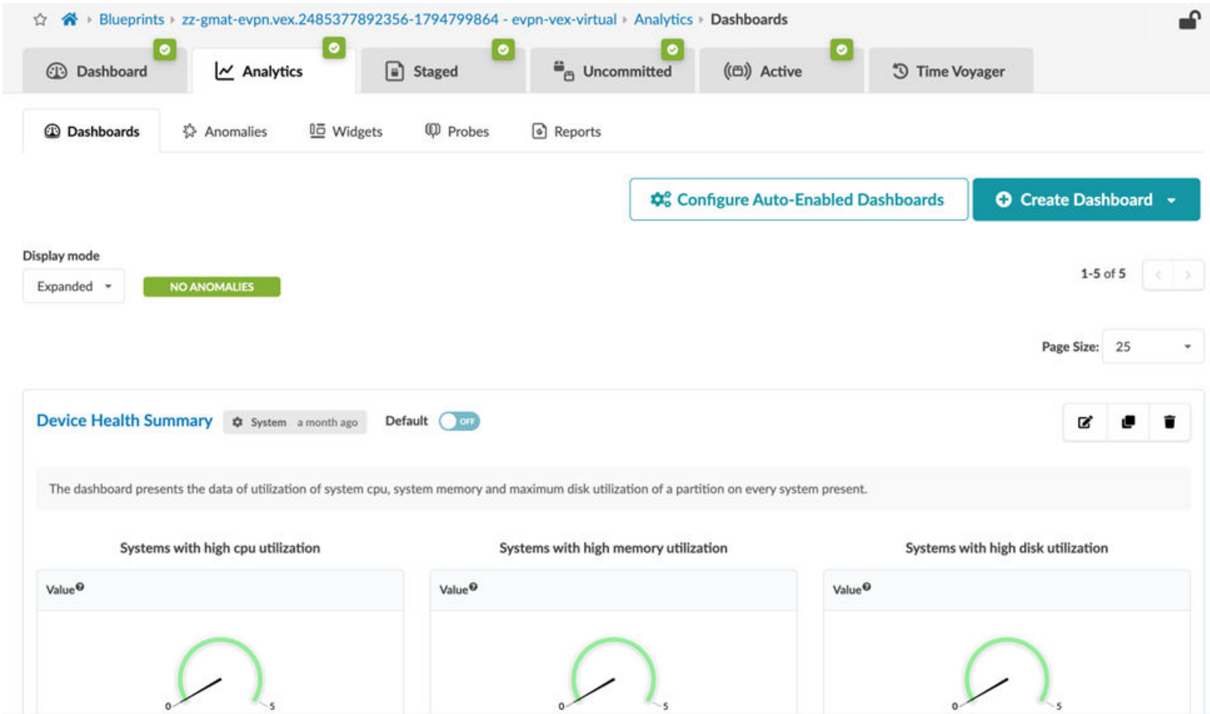
Apstra Telemetry and Intent-Based Analytics

IN THIS SECTION

What Is Intent-Based Analytics?

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:

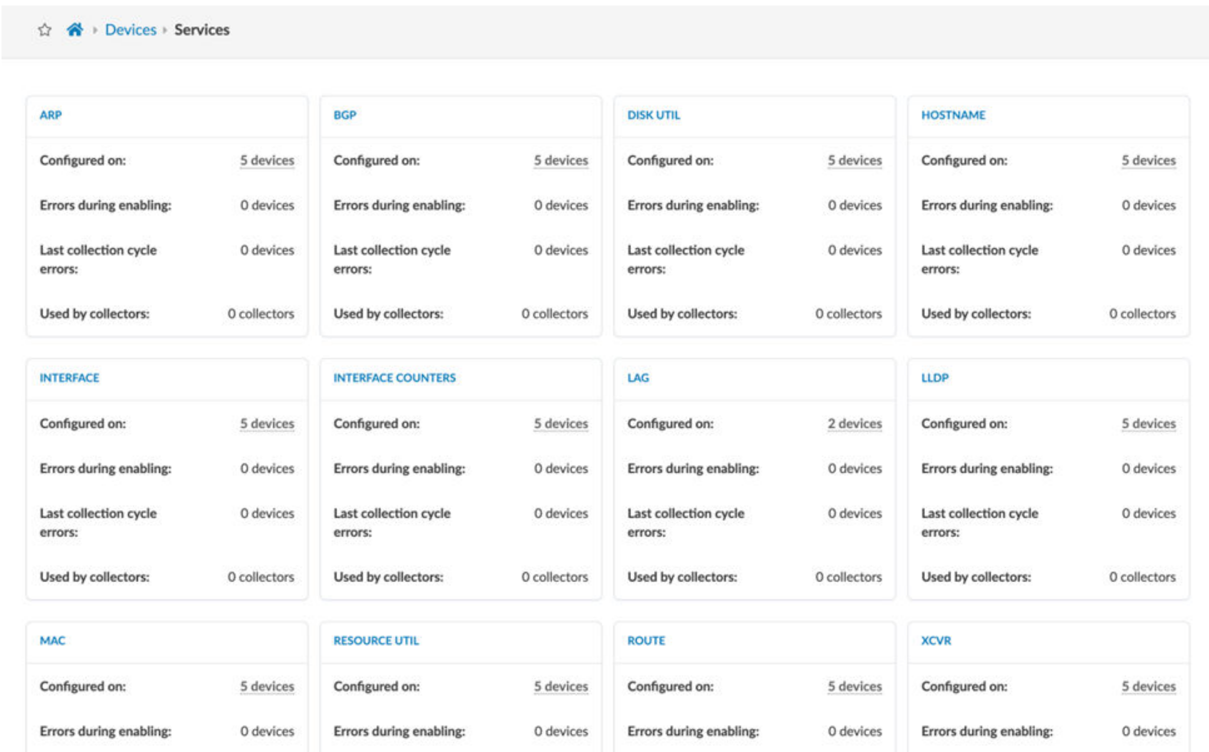
- Collect different types of telemetry data from managed devices.
- Enrich the data with contextual information from the blueprint.
- Aggregate and process the raw data into more meaningful data such as average over time, state in time, standard deviation, and so forth.
- Generate anomalies when the 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**.



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:

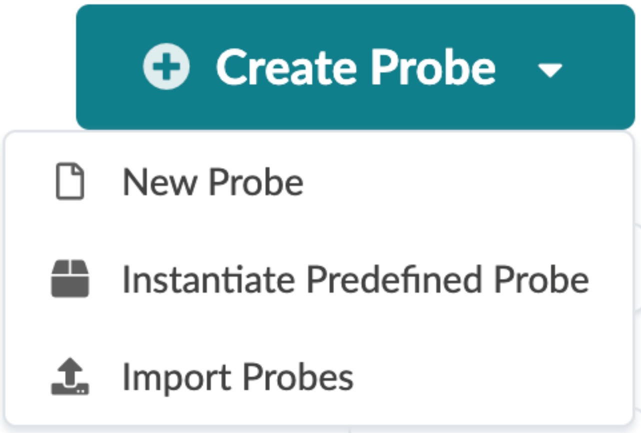
0 selected	Name	Anomalies	State
	Device System Health	No anomalies	Operational
	Device Telemetry Health	No anomalies	Operational
	Device Traffic	No anomalies	Operational
	ECMP Imbalance (Fabric Interfaces)	No anomalies	Operational
	ESI Imbalance	No anomalies	Operational
	LAG Imbalance	No anomalies	Operational

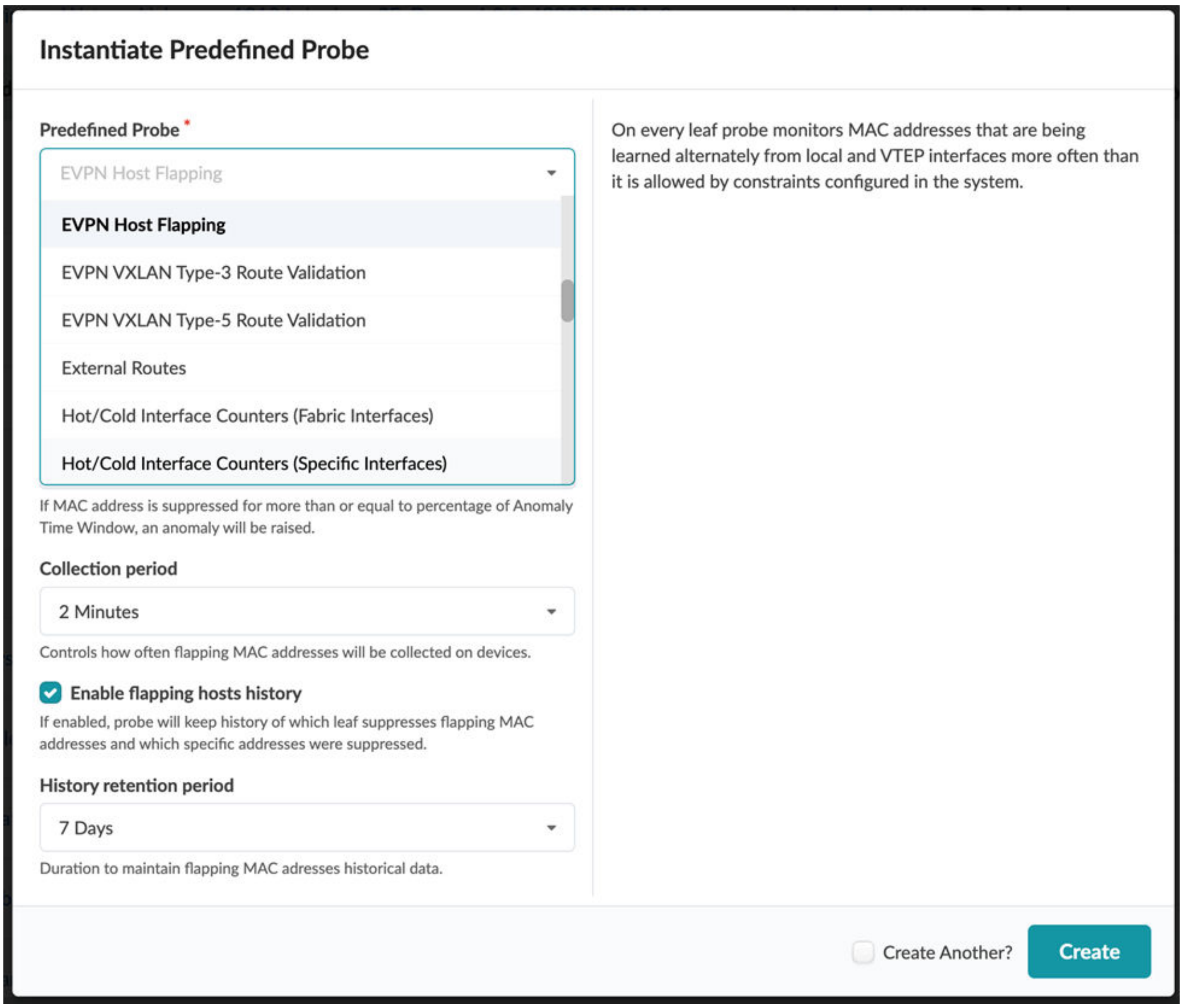
Predefined Probes Catalog

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.

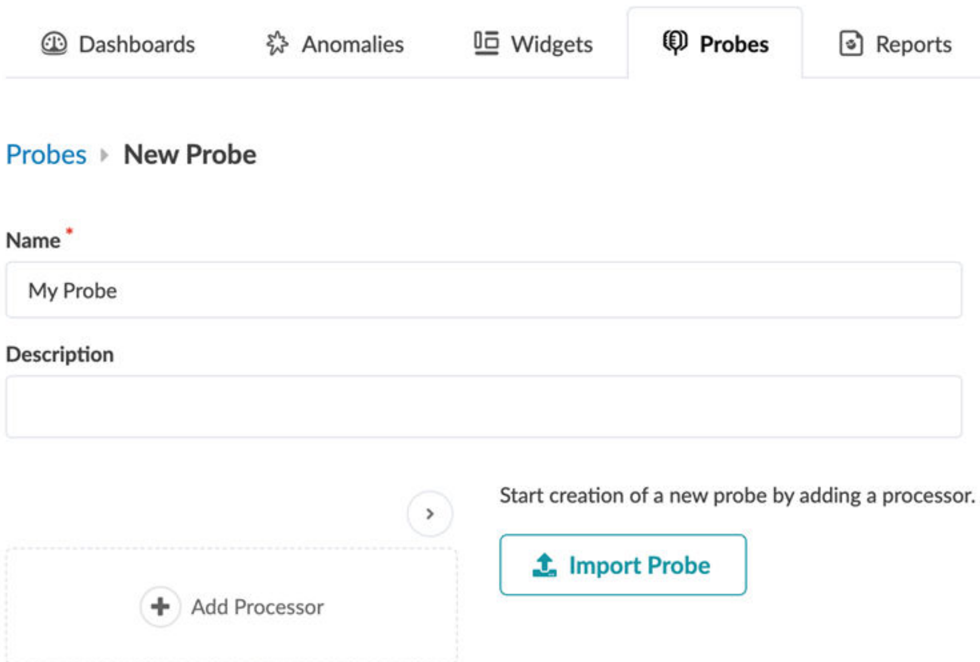




For detailed information about probes, see Predefined Probes (Analytics) in the Juniper Apstra User Guide.

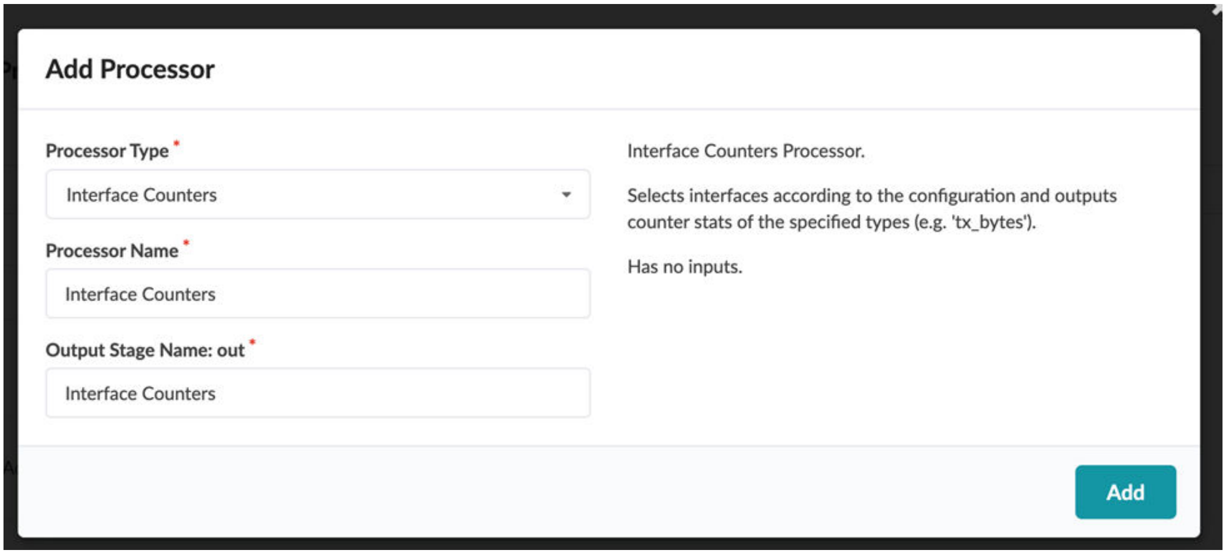
Custom Probes

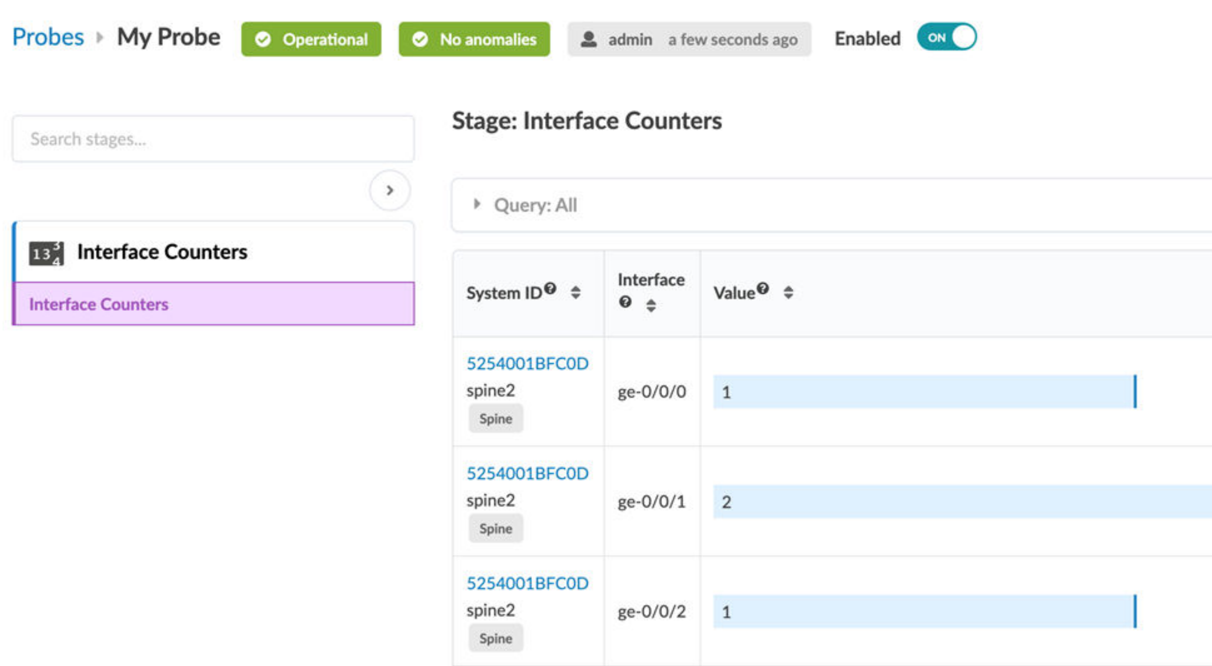
If you have a monitoring use case not addressed by any of the default or predefined probes, you must create a new custom probe in the Apstra GUI.



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**:





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

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

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.

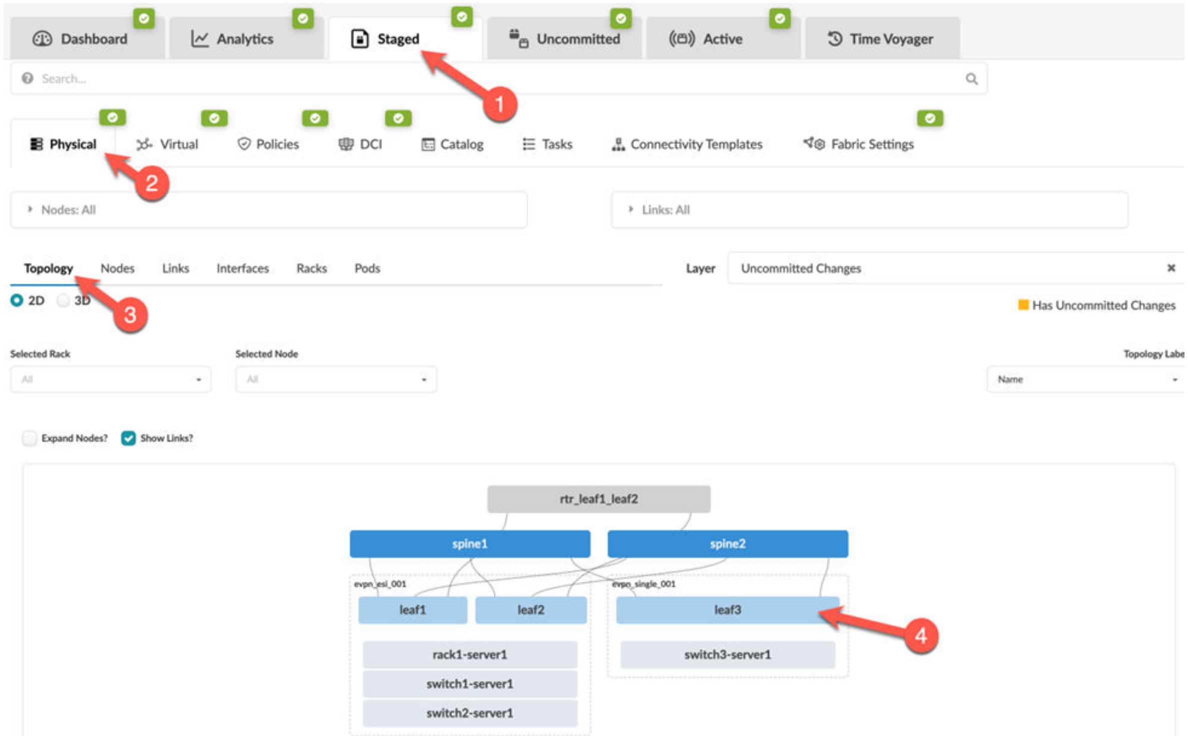
Execute the CLI Command

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.

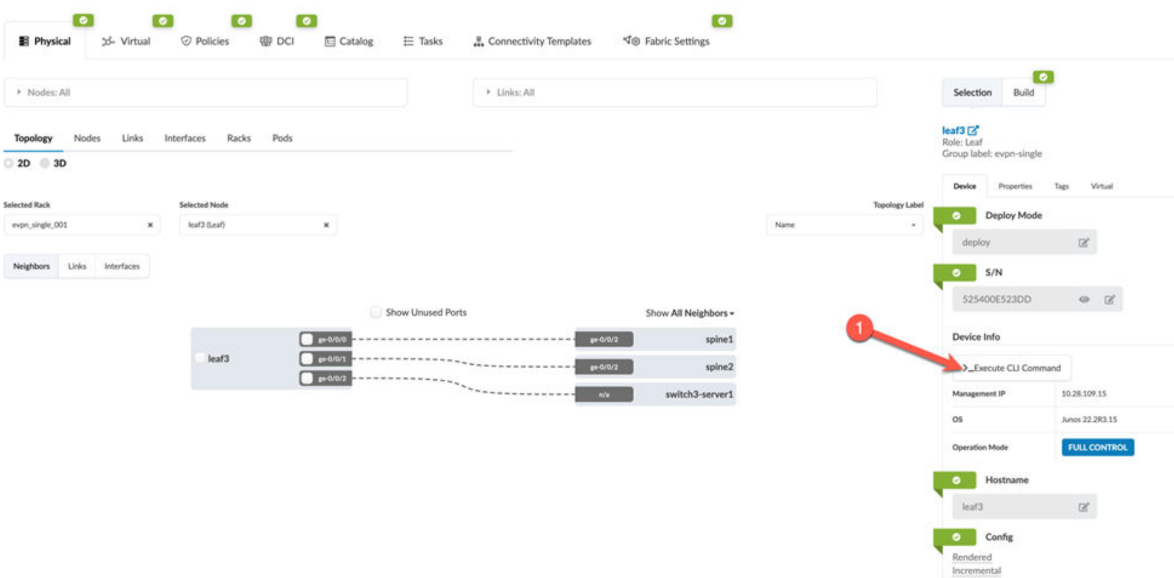
You can execute CLI commands from within a staged or active blueprint (shown in our example), or from the **Devices** > **Managed Devices** page.

To execute the CLI command:

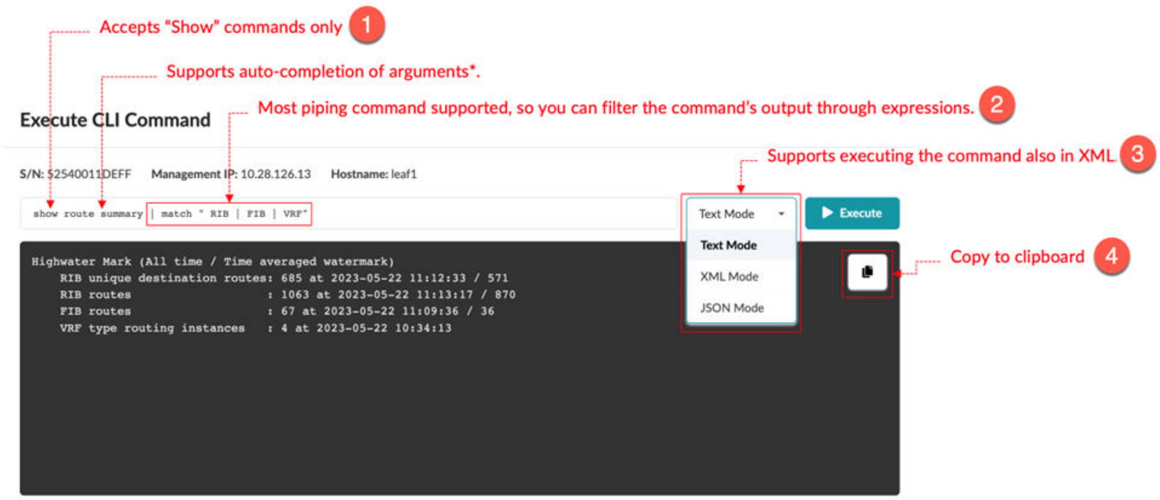
1. From your deployed blueprint, select **Analytics** > **Staged Physical Topology** (or **Staged** > **Physical** > **Nodes**), then select your Juniper device node.



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

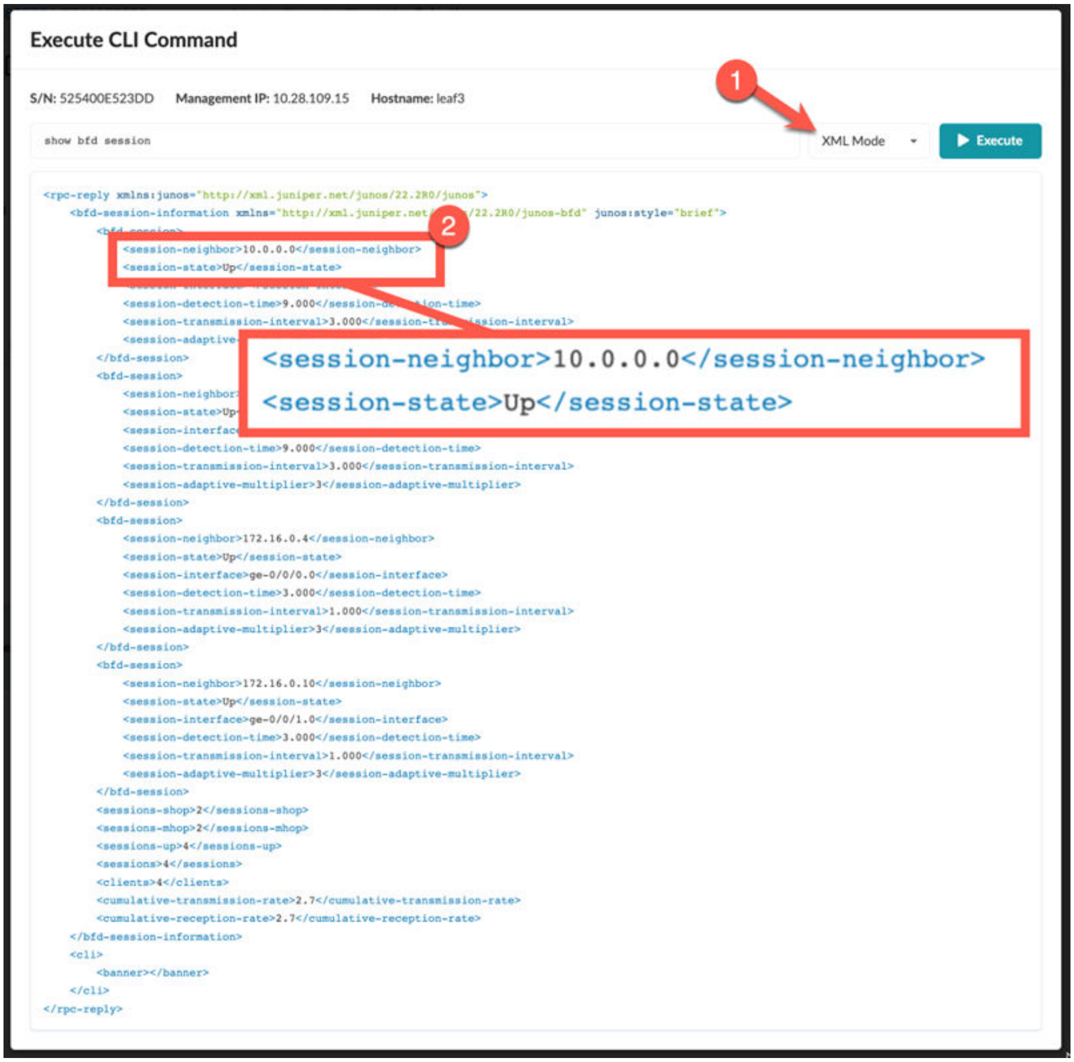


In the dialog box that opens, select how you want to view the results: Text Mode, XML Mode, or JSON mode. Below are examples of **Text Mode** and **XML Mode**.



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.



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.

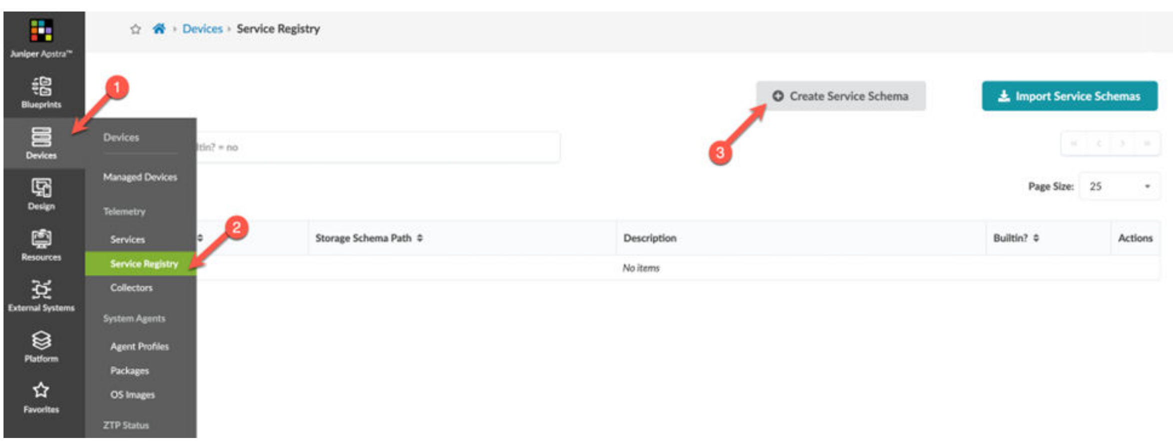
1. Enter the CLI show command (in this example, `show bfd session`).
2. Click **Execute** to view the BFD session information.

Create a Service Schema

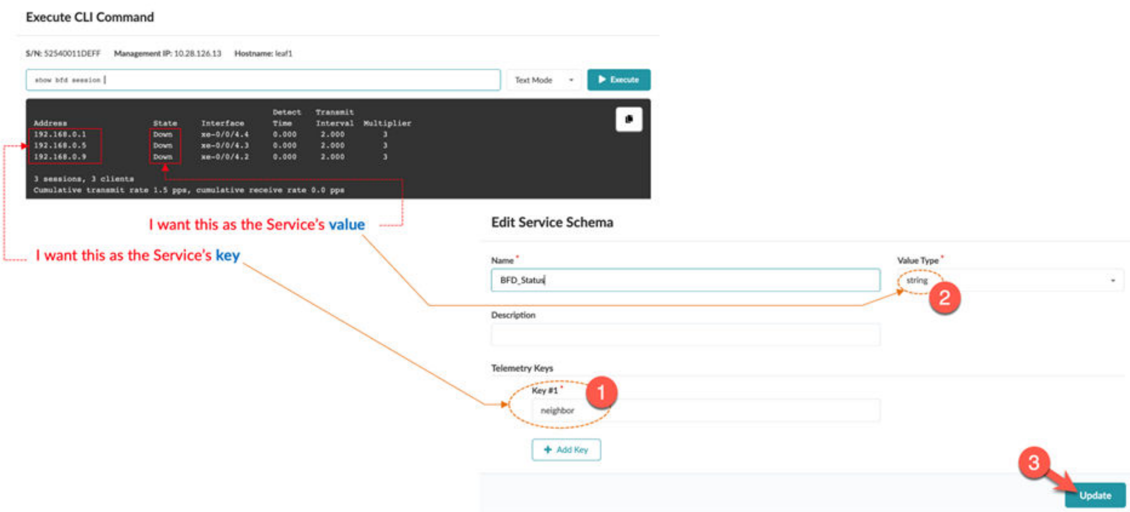
To create a custom service collector, you first need to create a service schema to define how you want your data to be structured and stored.

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.



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.

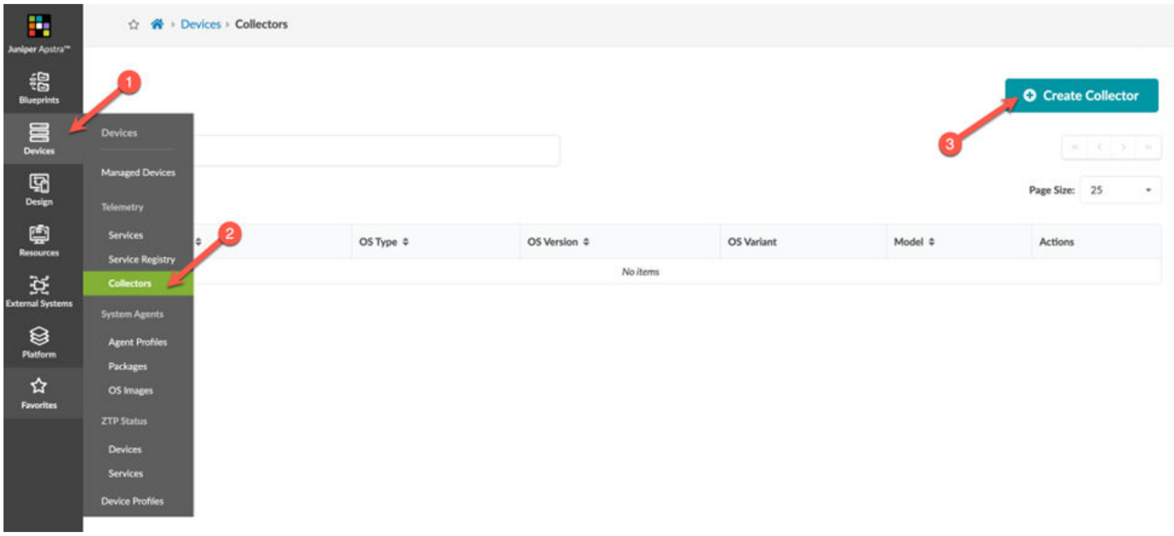
Create a Collector

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.

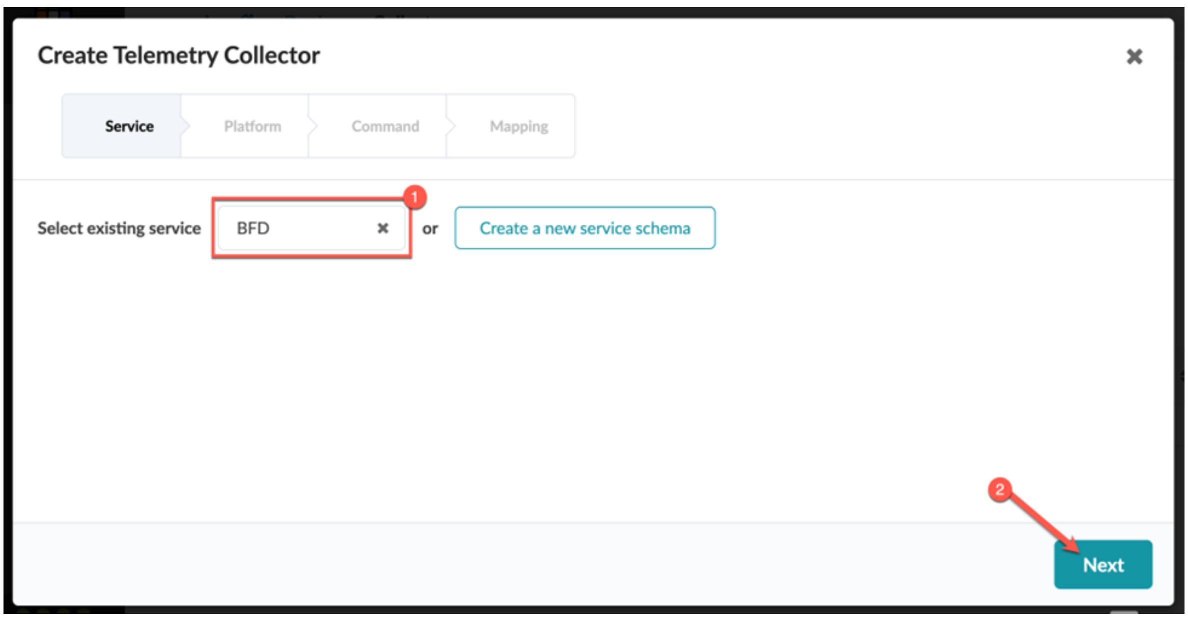
NOTE: A single telemetry service schema can have multiple collectors associated with it.

To create a collector:

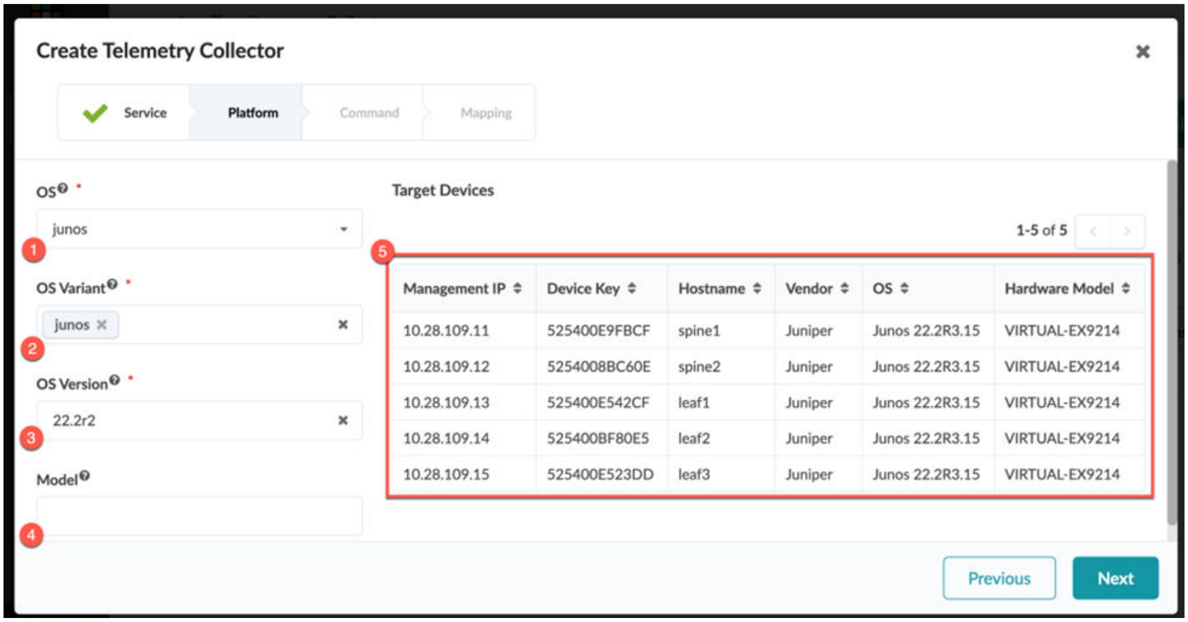
1. From the left navigation pane, navigate to **Devices** > **Collectors** > **Create Collector**.



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.



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.

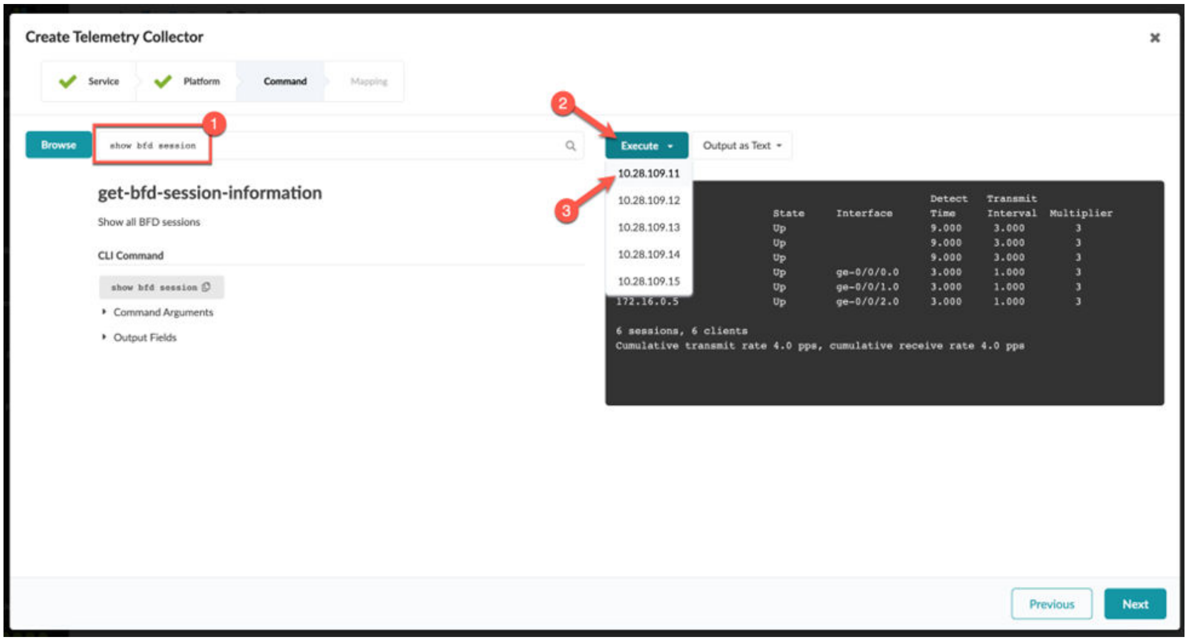
d. (Optional) Specify a **Model** or a regular expression to filter based on a device model or series.

   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.

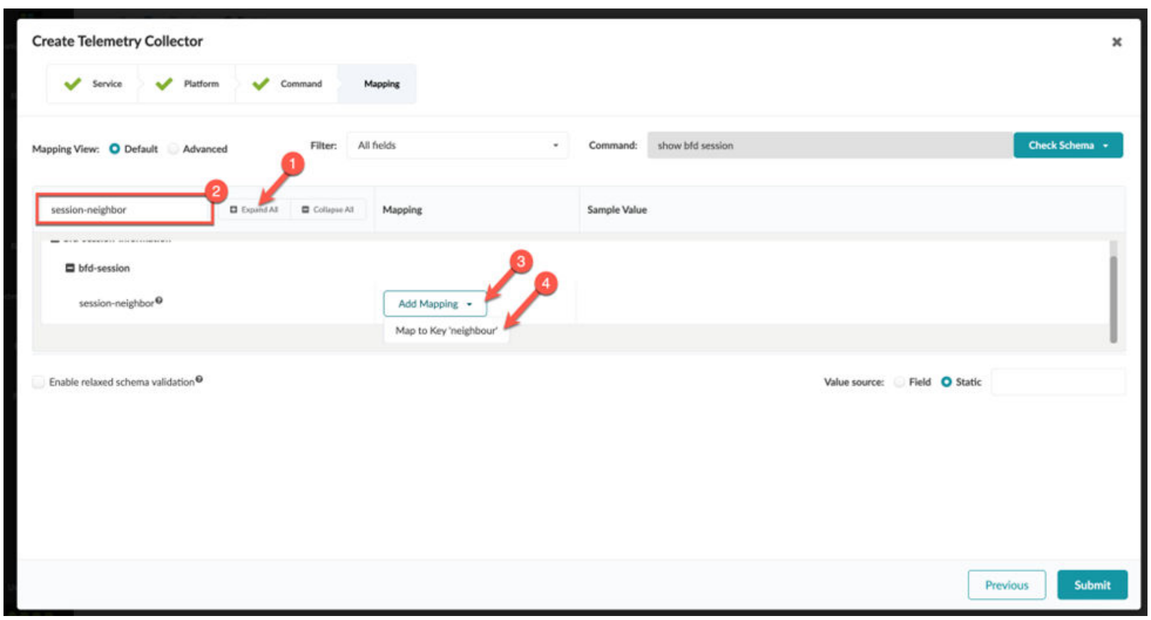
   Use the show command to gather the data you want to collect from the device (in this example, `show bfd session`).



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.

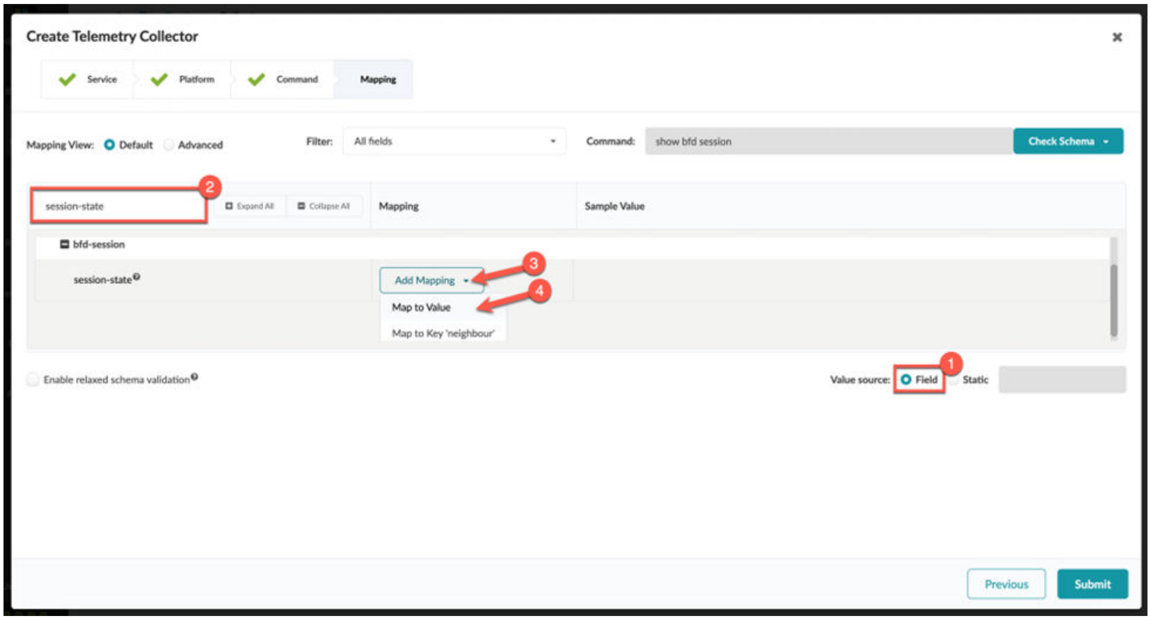
   a. To map the keys, click **Expand All** to search for the RPC value you want to map.



   b. Click **Add Mapping**.

   c. 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.

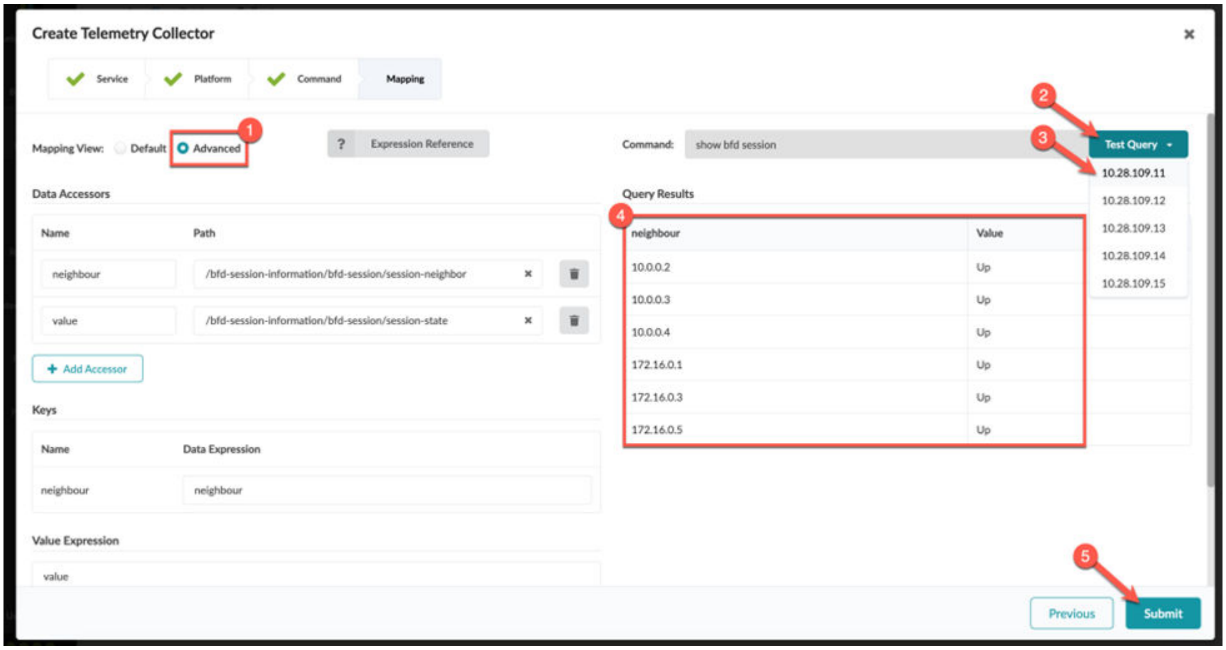


e. Search for the **session-state** field, then click **Add Mapping**.

f. Assign `session-state` to map the value, then click **Submit**.

Validate That the Collector Is Working

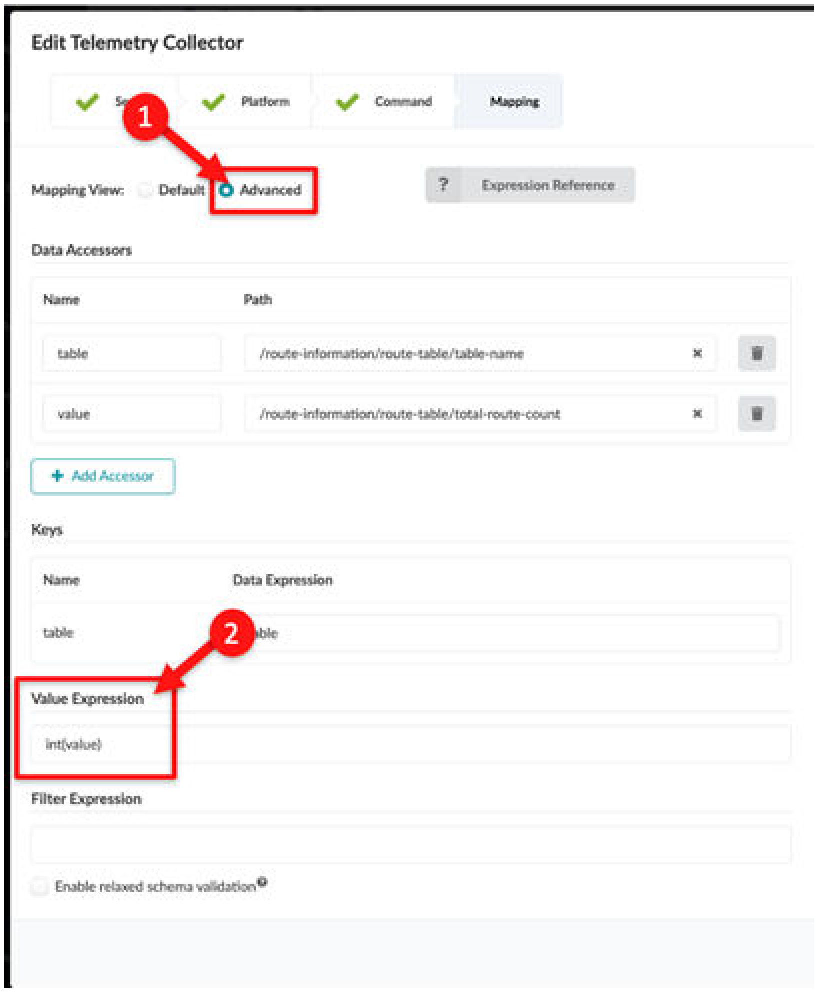
Finally, in **Advanced** view, validate that the collector is working. Verify that the query and test results match your expected results.



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**.



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

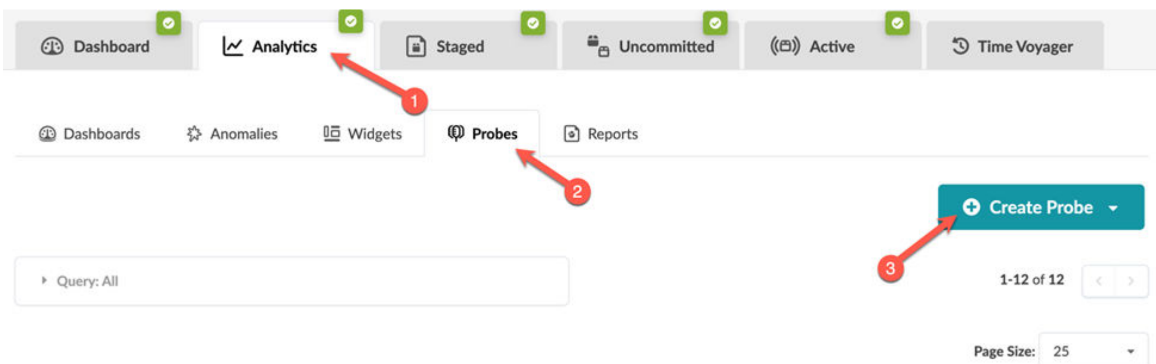
So far in our walkthrough, we've created a custom telemetry collector service that defines the data you want to collect from your devices. Now let's ingest this data into IBA probes in your blueprint so that Apstra can visualize and analyze the data.

Create a Probe

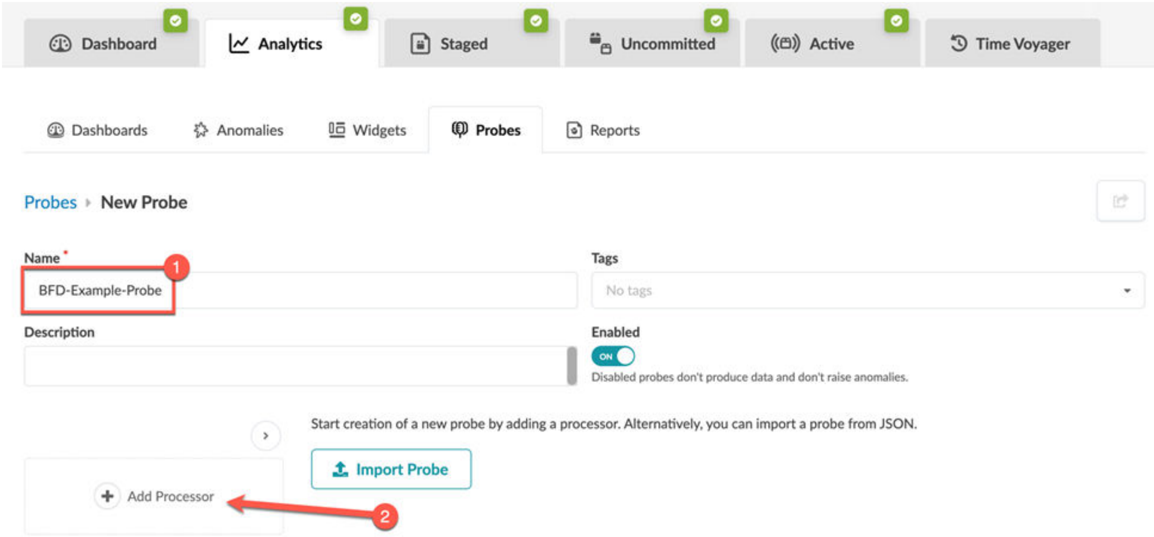
First, we'll create a new probe in your deployed blueprint so that Apstra can ingest data from your custom telemetry collector. In this example, we'll focus on a minimal set of configurations for the simple use case of visualizing BFD session data and generating anomalies (alerts) when sessions are down.

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

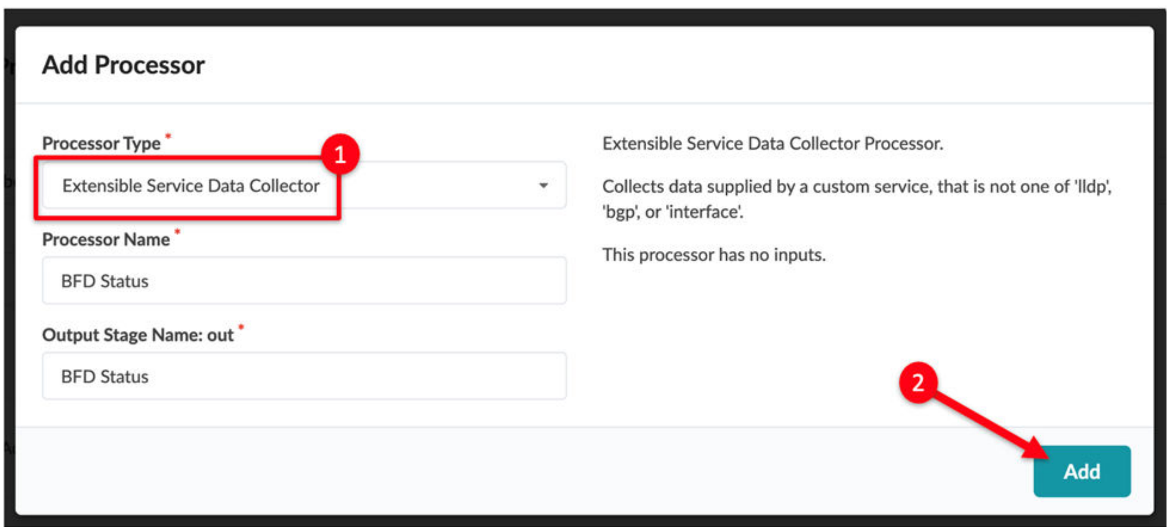
1. From your blueprint, navigate to **Analytics** > **Probes**, and then click **Create Probe** > **New Probe**.



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

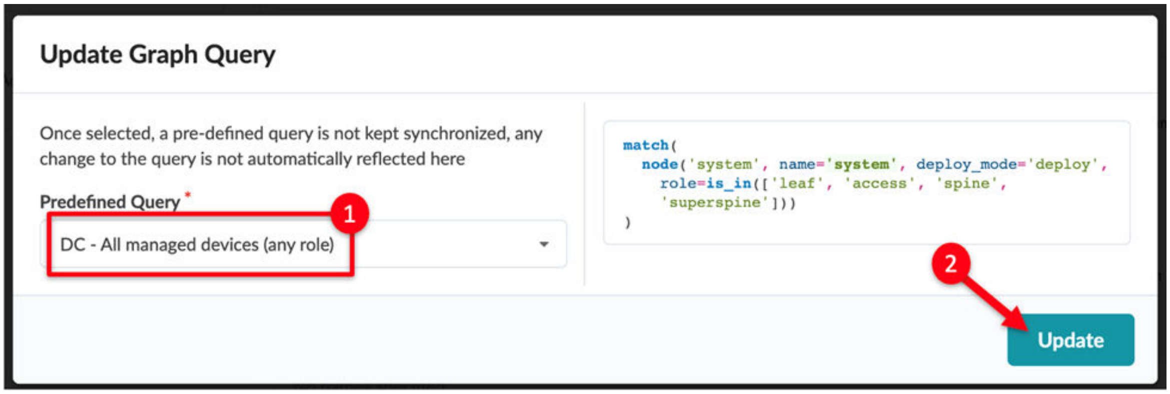


3. Select a processor type. For our example, we selected the **Extensible Service Data Collector** processor.



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.

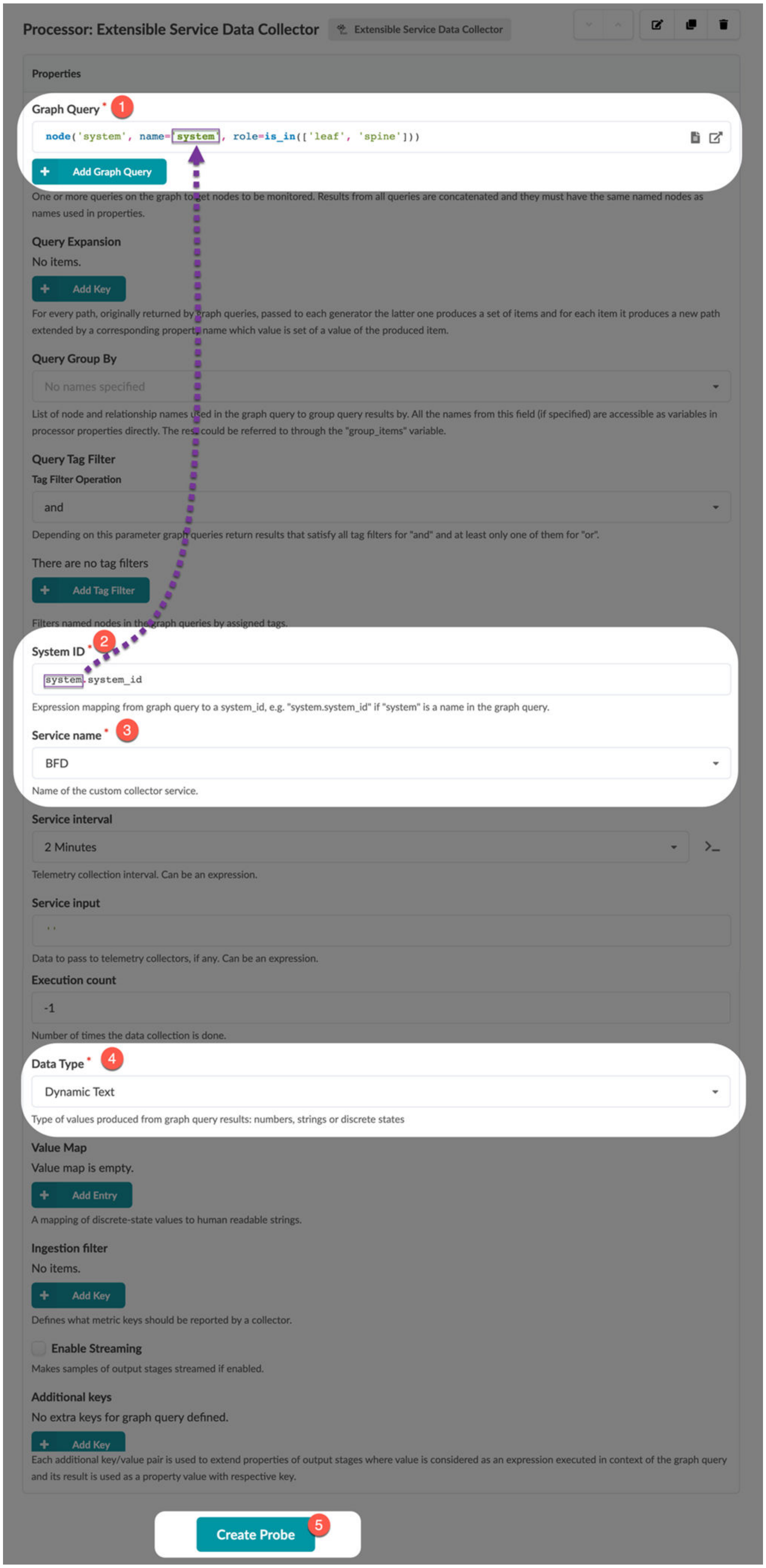
   This query determines the scope within the blueprint in which the telemetry collection is executed. This means if a device in your blueprint is not matched by the graph query, the telemetry collection service will not start for that device.



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`.

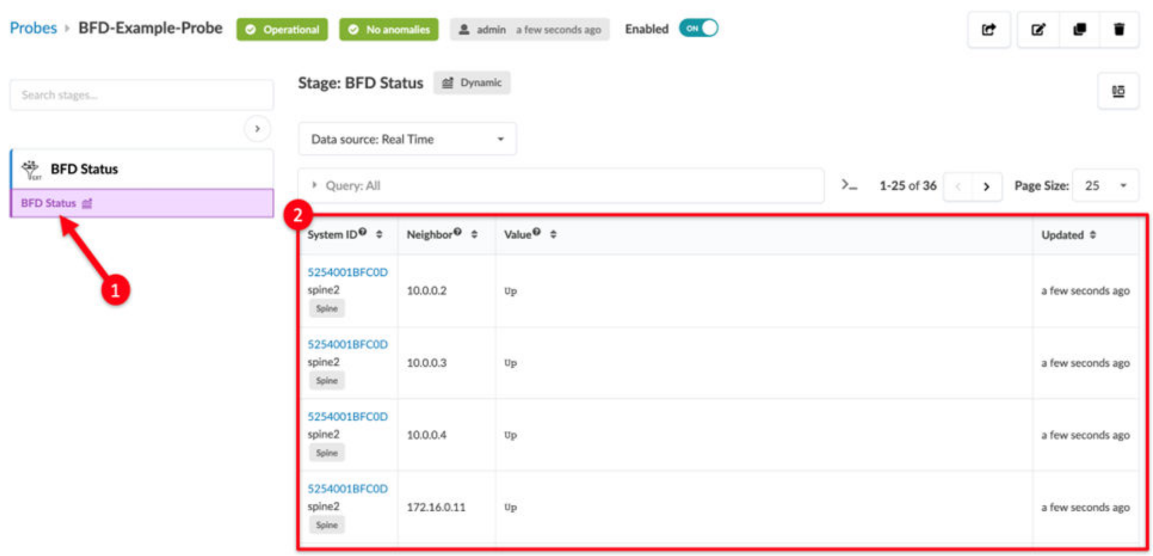
7. Click **Update** to return to the table view.


Processor: Extensible Service Data Collector
Extensible Service Data Collector
Properties
Graph Query *
1
node('system', name='system', role=is_in(['leaf', 'spine']))
Add Graph Query
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.
Add Key
For every path, originally returned by graph queries, passed to each generator the latter one produces a set of items and for each item it produces a new path extended by a corresponding property name which value is set of a value of the produced item.
Query Group By
No names specified
List of node and relationship names used in the graph query to group query results by. All the names from this field (if specified) are accessible as variables in processor properties directly. The rest could be referred to through the "group_items" variable.
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".
There are no tag filters
Add Tag Filter
Filters named nodes in the graph queries by assigned tags.
System ID *
2
system.system_id
Expression mapping from graph query to a system_id, e.g. "system.system_id" if "system" is a name in the graph query.
Service name *
3
BFD
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.
Execution count
-1
Number of times the data collection is done.
Data Type *
4
Dynamic Text
Type of values produced from graph query results: numbers, strings or discrete states
Value Map
Value map is empty.
Add Entry
A mapping of discrete-state values to human readable strings.
Ingestion filter
No items.
Add Key
Defines what metric keys should be reported by a collector.
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 considered as an expression executed in context of the graph query and its result is used as a property value with respective key.
Create Probe
5

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.



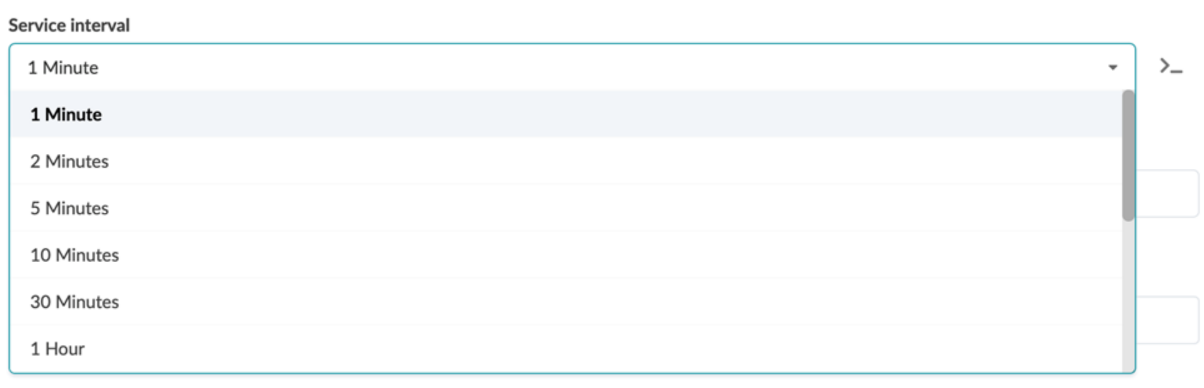
Congratulations! You successfully create a probe!

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.

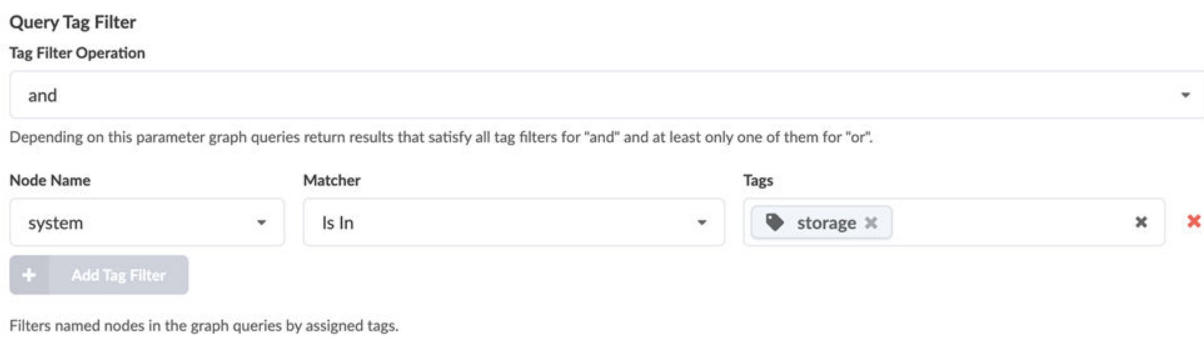
Service Interval

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.



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:



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 Asptra, 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
Range processors Processor names: Range, State, Time_In_State, Match_String	Range processors define reference state and generate anomalies.
Grouping processors Processor names: Match_Count, Match_perc, Set_Count, Sum, Avg, Min, Max, and Std_Dev	Group processors aggregate and process data before feeding into the range processors. These processors can: • Produce a per-device count of protocol states. • Produce a sum of counters from multiple devices to represent a total over the fabric.
Multi-input processors Processor names: Match_Count, Match_perc, Set_Count, Sum, Avg, Min, Max, and Std_Dev	Analytics processors take input from multiple stages. These processors can: • Produce a single output data set that is a union of input from multiple stages. • Perform a logical comparison between input from multiple stages.

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.

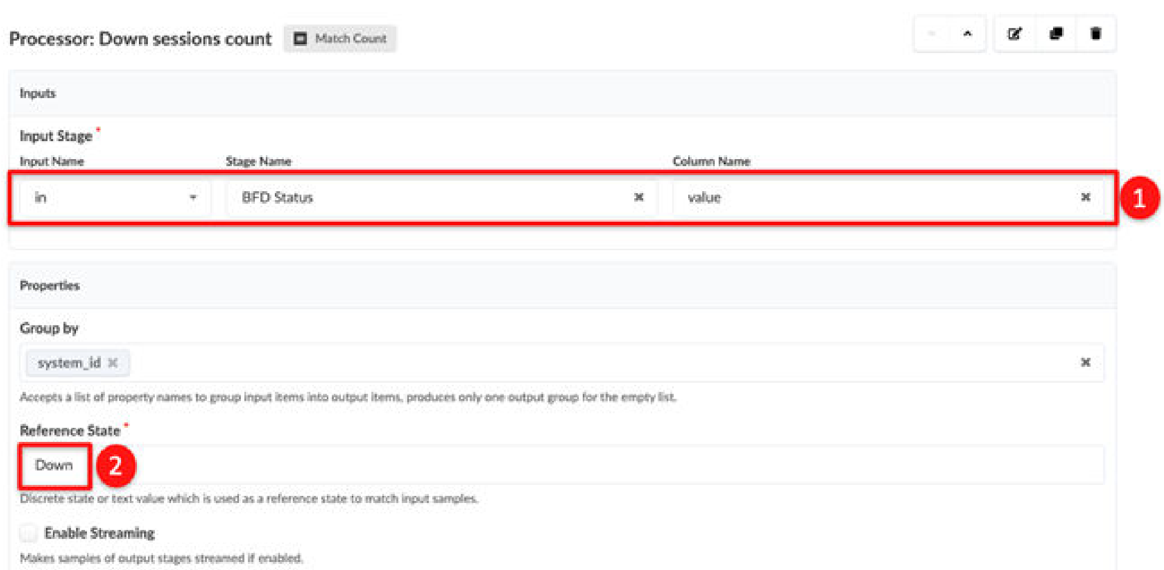
Raising Anomalies and Storing Historical Data

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.

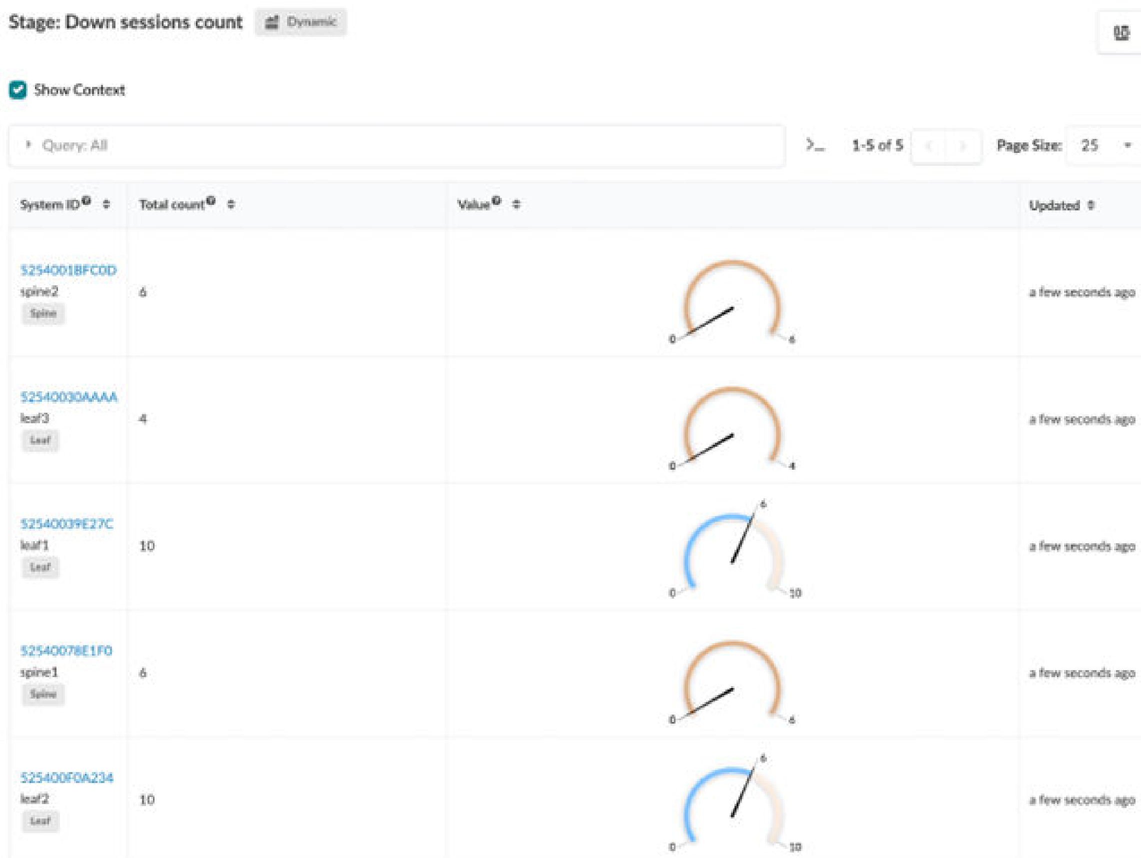
1. First, add a second processor to the probe you created in "Create a Probe" on page 22 , then click **Add Processor**.
2. Select the **Match Count** processor and give the processor a descriptive name, such as Down sessions count.

   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.

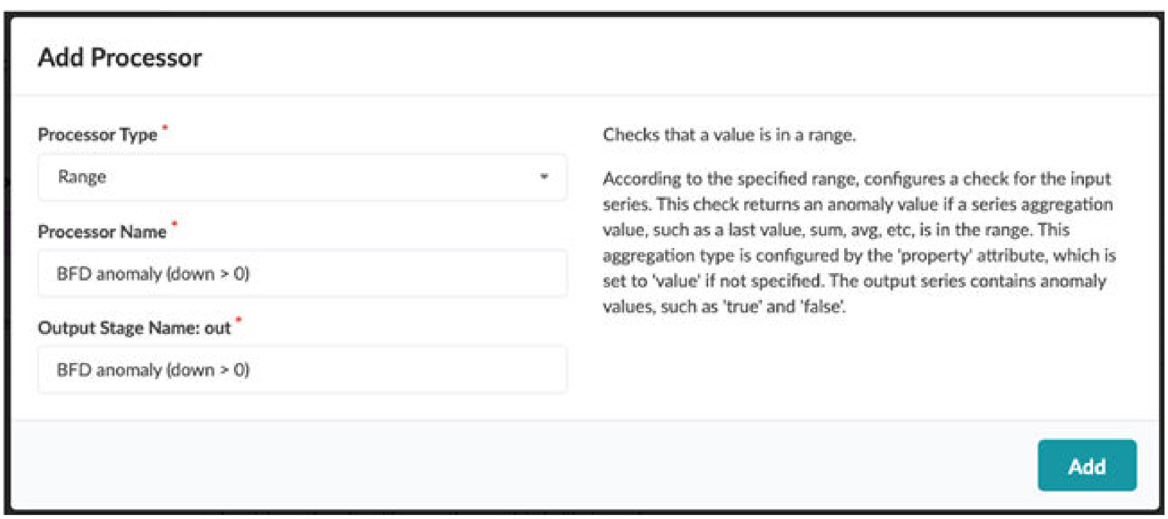


   When you update the probe, the output shows the number of BFD sessions in the **Down** state by each device.

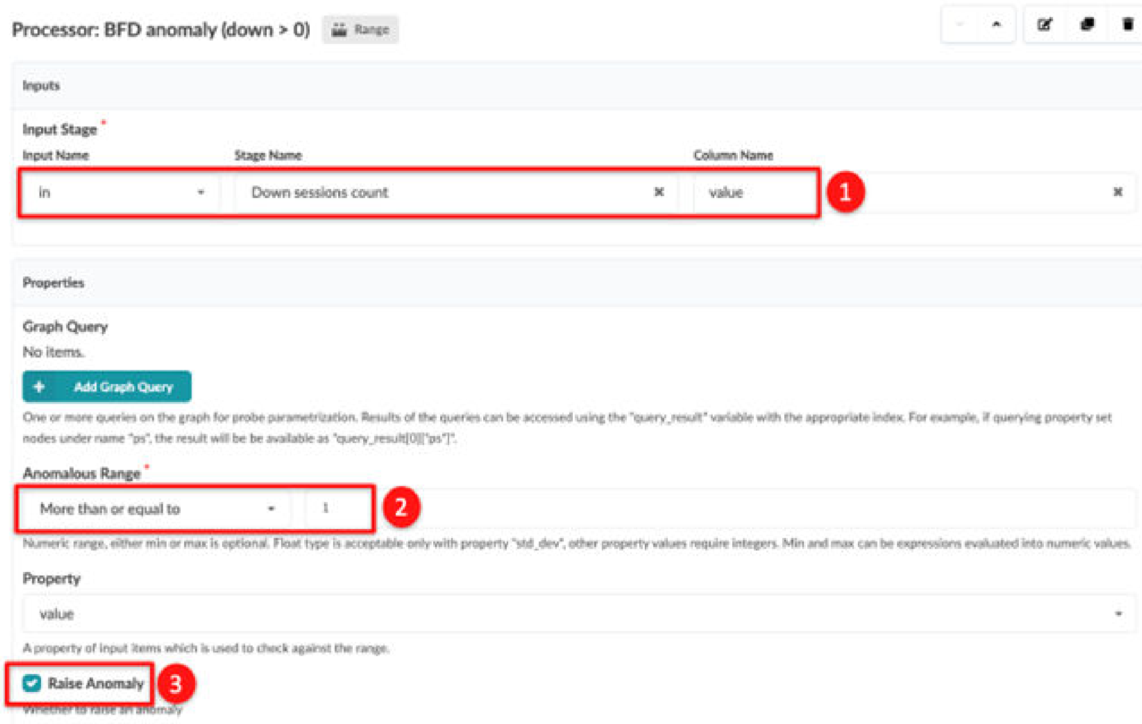


4. Add the third and final processor. This processor produces anomalies to alert you when there are one or more BFD sessions in the Down state.
5. Click **Add Processor**, then select the **Match Count** processor.

   Give the processor a descriptive name (in this example, **BFD anomaly (down > 0)**, then click **Add**.



6. Configure the third processor.



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**.

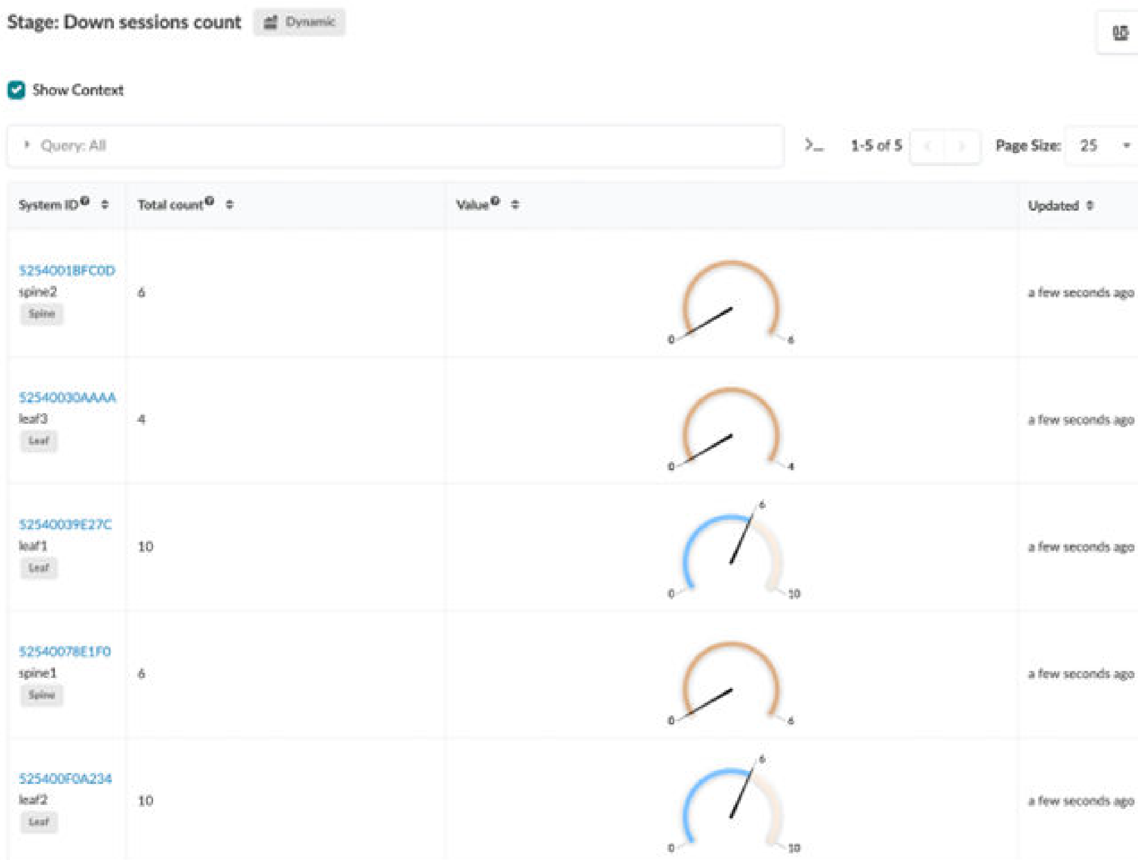
b. Set the **Anomalous Range** to **More than equal to** and **1**.

c. Click **Raise Anomaly**.

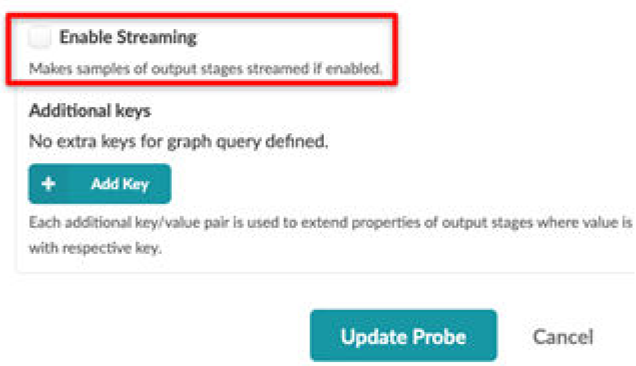
7. While still in the probe configuration interface, click **Enable Metric Logging**, then select the output stage for your second processor. This action enables historical logging of data.

8. Click **Update the Probe**.

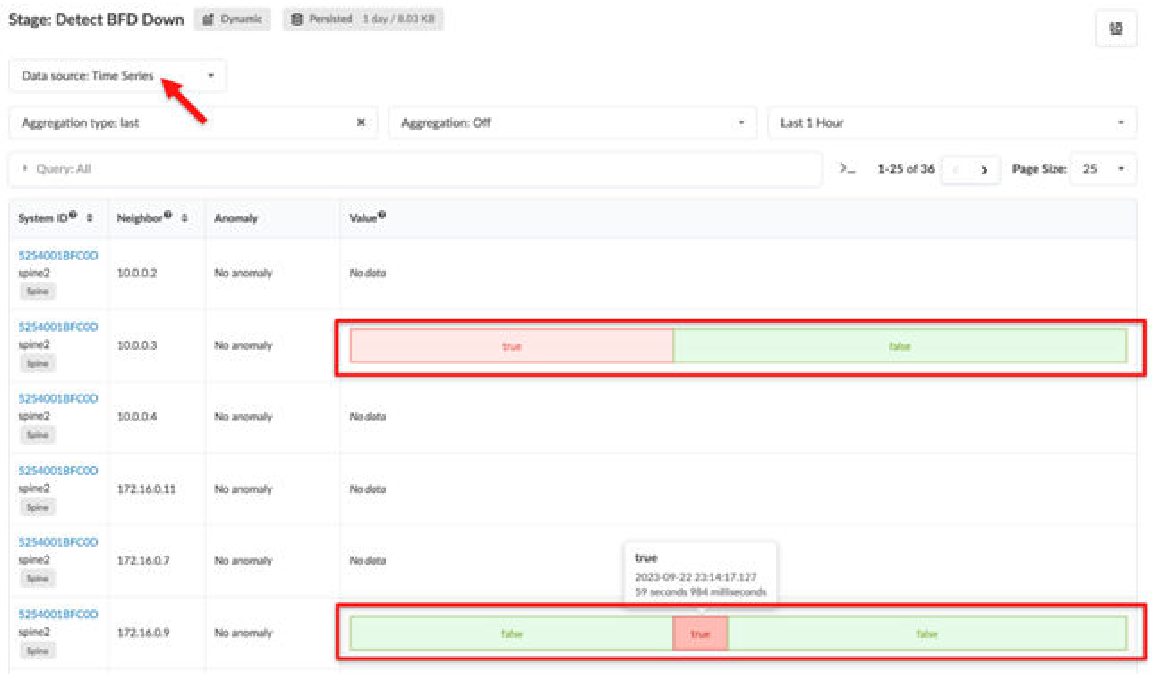
   If you have any BFD sessions in the Down state, the probe generates anomalies for the BFD sessions.



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



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



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.

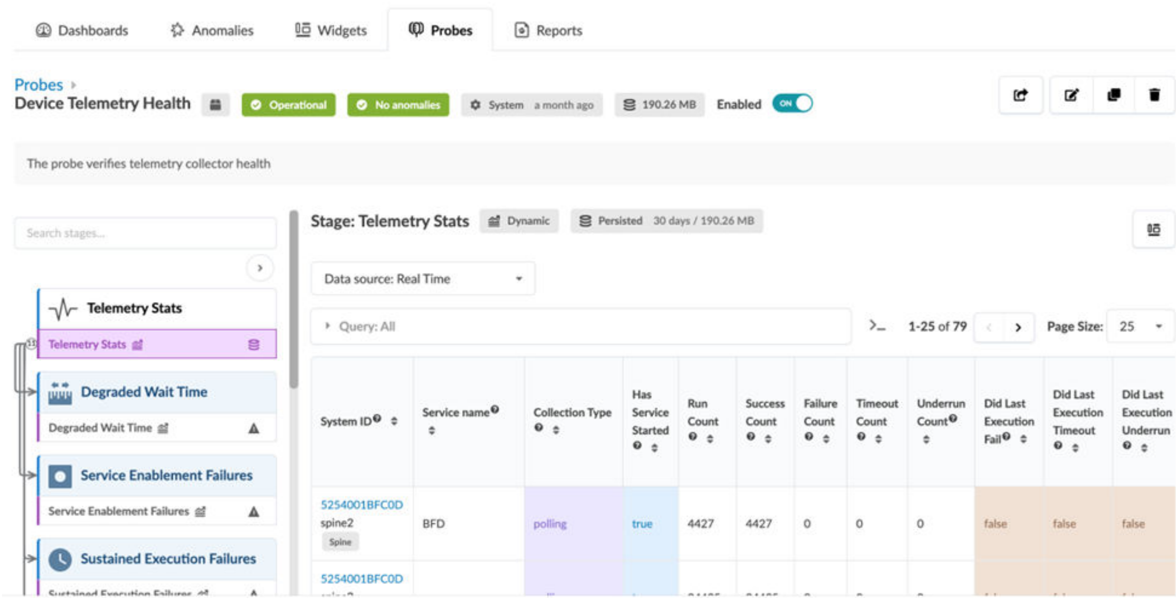
By default, Apstra provides an IBA telemetry health probe that enables you to monitor the health of telemetry services, including any custom services and collectors you configured.

To monitor the health of your telemetry services:

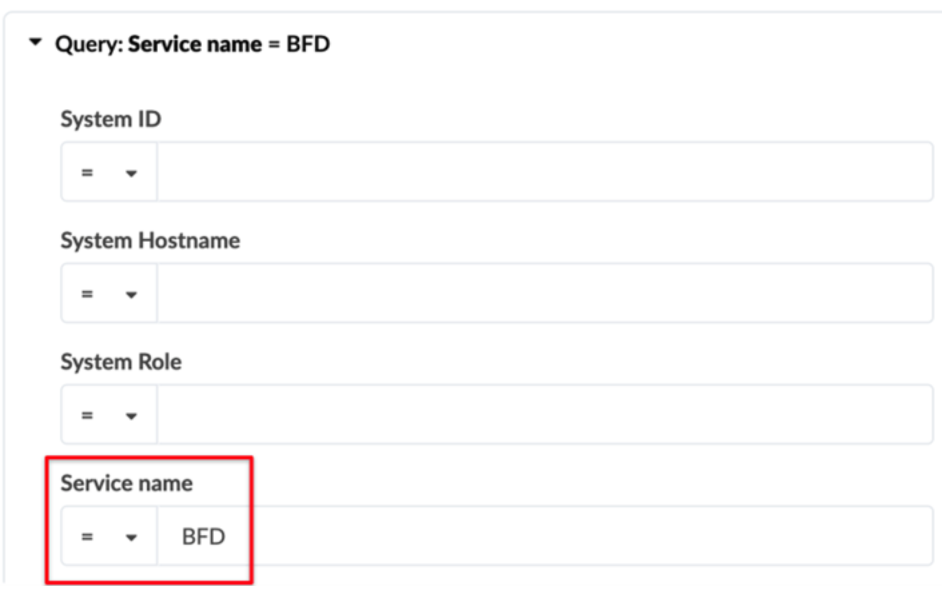
1. From your blueprint, navigate to **Analytics** > **Probes**.

2. Select the **Device Telemetry Health** probe from the table.

3. Click **Query: All** to filter the data in the table.



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**.



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 Spine	BFD	polling	true	4439	4439	0	0	0	false	false	false	0.1608336661
52540030AAAA leaf3 Leaf	BFD	polling	true	4732	4732	0	0	0	false	false	false	0.1845839512
52540039E27C leaf1 Leaf	BFD	polling	true	4439	4439	0	0	0	false	false	false	0.1687121880
52540078E1F0 spine1 Spine	BFD	polling	true	4439	4439	0	0	0	false	false	false	0.1853731549
525400F0A234 leaf2 Leaf	BFD	polling	true	4439	4439	0	0	0	false	false	false	0.2065631197

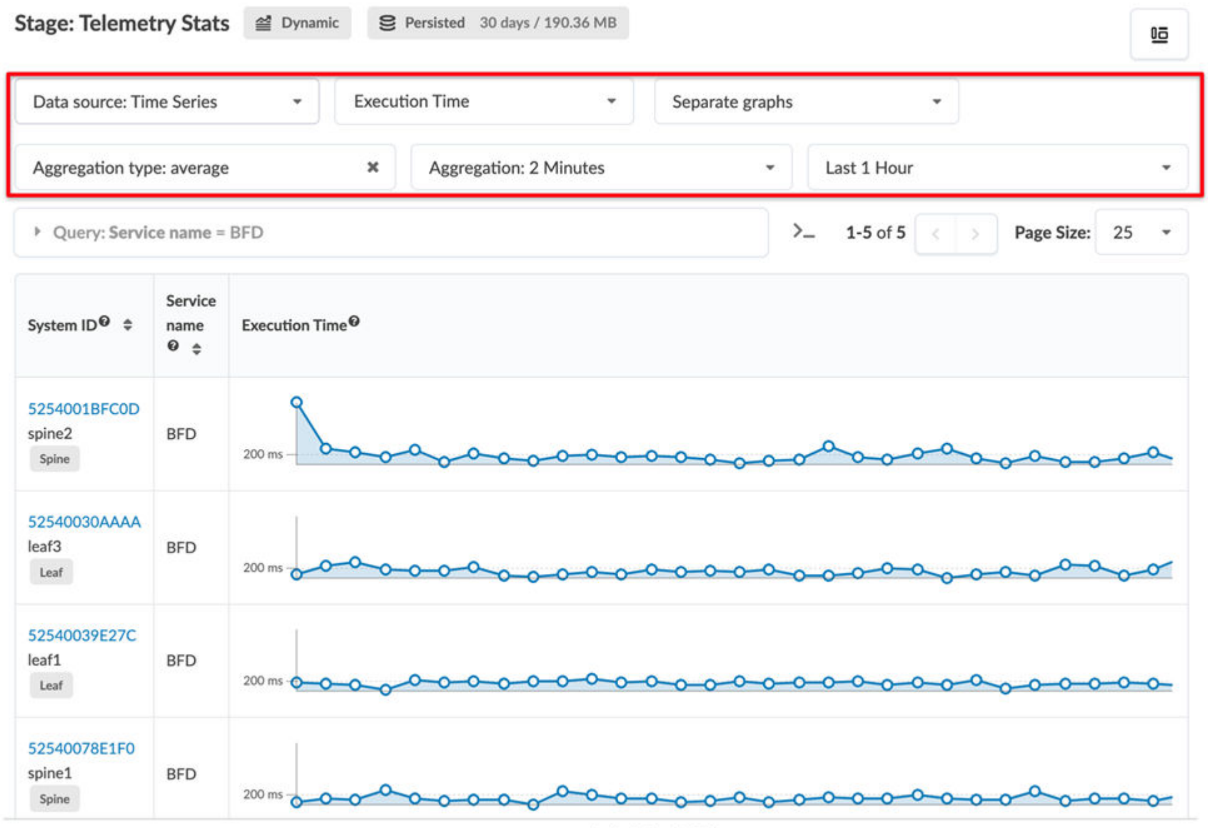
Check the following:

- 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.