Juniper Networks | Engineering Simplicity

Juniper Apstra 5.0 User Guide

Published
2024-10-29

Juniper Networks, Inc.
1133 Innovation Way
Sunnyvale, California 94089
USA
408-745-2000
www.juniper.net

Juniper Apstra 5.0 User Guide

The information in this document is current as of the date on the title page.

YEAR 2000 NOTICE

Juniper Networks hardware and software products are Year 2000 compliant. Junos OS has no known time-related limitations through the year 2038. However, the NTP application is known to have some difficulty in the year 2036.

END USER LICENSE AGREEMENT

The Juniper Networks product that is the subject of this technical documentation consists of (or is intended for use with) Juniper Networks software. Use of such software is subject to the terms and conditions of the End User License Agreement ("EULA") posted at https://support.juniper.net/support/eula/. By downloading, installing or using such software, you agree to the terms and conditions of that EULA.

Table of Contents

4 Staged Datacenter Blueprints

16

Platform

About This Guide

Welcome! Juniper Apstra automates all aspects of the data center network design, build, deploy, and operation phases. It leverages advanced intent-based analytics to continually validate the network, thereby eliminating complexity, vulnerabilities, and outages resulting in a secure and resilient network.

1

PART

Getting Started

Get Started

IN THIS SECTION

To get started, you'll install and configure the Apstra software. Then you'll replace the SSL certificate and default passwords to increase security. You can then start building the elements of your physical network. Depending on the complexity of your design, other tasks may be required in addition to the ones included in this general workflow.

Review Release Notes

Software Release Notification for Juniper Apstra Versions 5.0.1 & 5.0.0

Install Apstra Software

Install and configure Apstra software on one of the supported hypervisors.

If you're installing on an ESXi hypervisor, check out the Installing Apstra Software Quick Start Guide, which focuses on ESXi only.

Design

1. "Logical devices" on page 845 (Design > Logical Devices) are abstractions of physical devices. They allow you to specify device capabilities before selecting specific vendor hardware. Check the logical device design (global) catalog for ones that meet your requirements; create them if needed.
2. "Interface maps" on page 853 (Design > Interface Maps) combine device profiles and logical devices. Check the interface map design (global) catalog for ones that meet your requirements; create them if needed.
3. "Rack types" on page 862 (Design > Rack Types) are logical representations of racks. Check the rack type design (global) catalog for ones that meet your requirements; create them if needed.
4. "Templates" on page 889 (Design > Templates) are used to build rack designs (blueprints). Check the template design (global) catalog for one that meets your requirements; create it if needed.

Resources

Create resource pools ("ASNs" on page 928, "IPv4 addresses" on page 934, and "IPv6 addresses" on page 937 if needed) for your network. When you're ready to assign resources to your blueprint, you'll specify a resource pool, then the resources will automatically be assigned from that pool.

Devices

You can set up your devices anytime before you need to assign them in your blueprint.

Access the "Apstra GUI" on page 4 and get your devices ready.

1. "Device profiles" on page 783 (Devices > Device Profiles) represent the physical devices in your network. Many device profiles are predefined for you. Check the list, and if one that you need is not included, you can create it.
2. "Add devices" on page 651 to be managed by the Apstra environment.

Check out the Onboarding Data Center Switches with Apstra Quick Start Guide for additional information.

Blueprints

1. "Create a blueprint" on page 8 from one of the templates in the Design catalog.

2. "Assign resources" on page 58, "device profiles" on page 62, and "devices" on page 64 (S/Ns) to build the network (Blueprints > <your_blueprint_name> > Staged > Physical > Build)
3. Review the calculated cabling map (Blueprints > <blueprint_name> > Staged > Physical > Links), then cable up the physical devices according to the map. If you have a set of pre-cabled switches, ensure that you have configured interface maps according to the actual cabling so that calculated cabling matches actual cabling.
4. When you've finished building your network, "commit" on page 599 the blueprint (Blueprints > <your_blueprint_name> > Uncommitted). Committing a blueprint initiates work on the intent and pushes configuration changes on assigned devices to realize it on the network.
5. Review the "blueprint dashboard" on page 52 (Blueprints > Dashboard) for "anomalies" on page 616. If you have cabling anomalies, the likely reason is a mismatch in calculated cabling and actual cabling. Either re-cable the switches, recreate the blueprint with appropriate interface maps or use the "Apstra-CLI" on page 1420 utility to override the cabling in the blueprint with discovered cabling.

Next Steps

After your deployment is running, you can build the virtual environment with "virtual networks" on page 251 and "routing zones" on page 282, as needed.

Log in to Apstra GUI

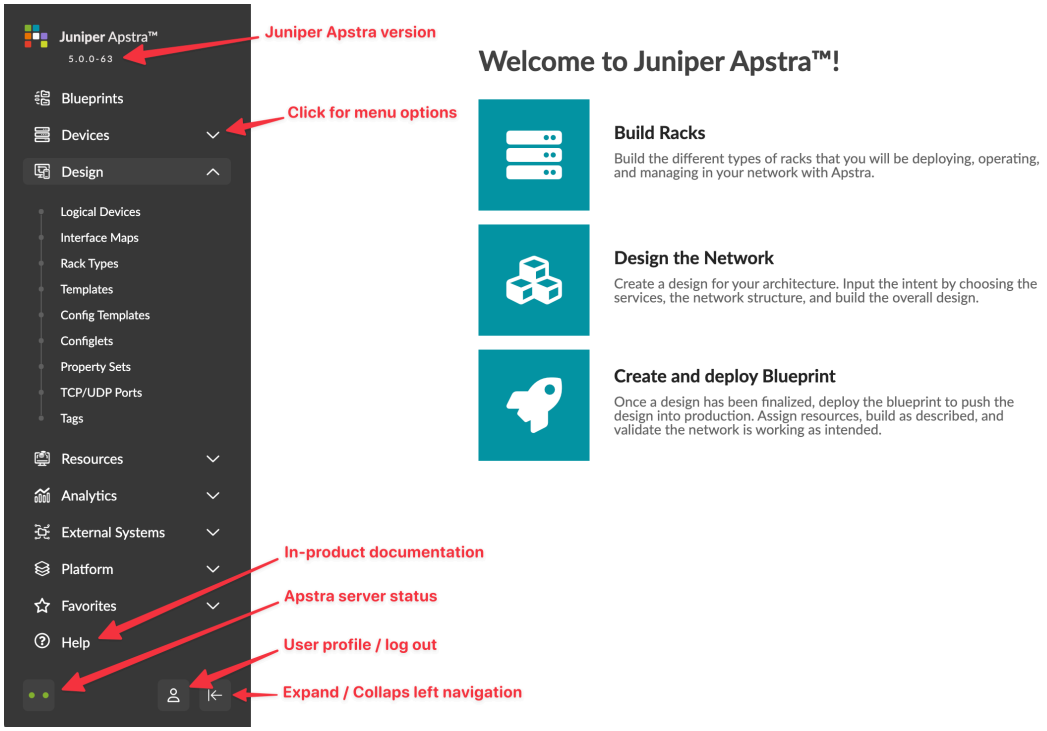
SUMMARY

Access the Apstra GUI to design, build, deploy, operate and validate your network.

1. From the latest web browser version of Google Chrome or Mozilla FireFox, enter the URL `https://<apstra_server_ip>` where `<apstra_server_ip>` is the IP address of the Apstra server (or a DNS name that resolves to the IP address of the Apstra server).
2. If a security warning appears, click **Advanced** and **Proceed to** the site. The warning occurs because the SSL certificate that was generated during installation is self-signed, and you didn't replace it with a signed one when you installed the software. We recommend, for security reasons, that you replace the SSL certificate. (See the Juniper Apstra Installation and Upgrade Guide for details.)

3. From the login page, enter username **admin** and the secure password that you set when you configured the Apstra server. (Entering the password incorrectly too many times locks you out for a few minutes depending on how password requirements have been configured. You can "update password complexity requirements" on page 1286.)

   The main screen appears.



Next Steps: See the "Get Started" on page 2 section of this guide for the general workflow for building your network, with links to more information.

Reset Apstra GUI Admin Password

If you reset (a lost) Apstra GUI admin password to the default, we highly recommend that you immediately change it to a secure one. User **admin** has full root access. Juniper is not responsible for security-related incidents because of not changing default passwords.

1. SSH into the Apstra server as user **admin** (ssh admin@<apstra-server-ip> where <apstra-server-ip> is the IP address of the Apstra server.)
2. Run the command `aos_reset_admin_password` as shown in the example below.

```
admin@aos-server:~$ aos_reset_admin_password
Resetting UI "admin" user password to default "admin"
```

```
Successfully reset admin's password
admin@aos-server:~$
```

3. Log in to the Apstra GUI (default password: **admin**), then navigate to **Platform > User Management > Users**.
4. Click username **admin**, then click the **Change Password** button (top-right)
5. Enter a secure password that meets the complexity requirements, then re-enter the new password.
6. Click **Change Password** to update the password.

2

PART

Blueprints Introduction

Create Blueprint

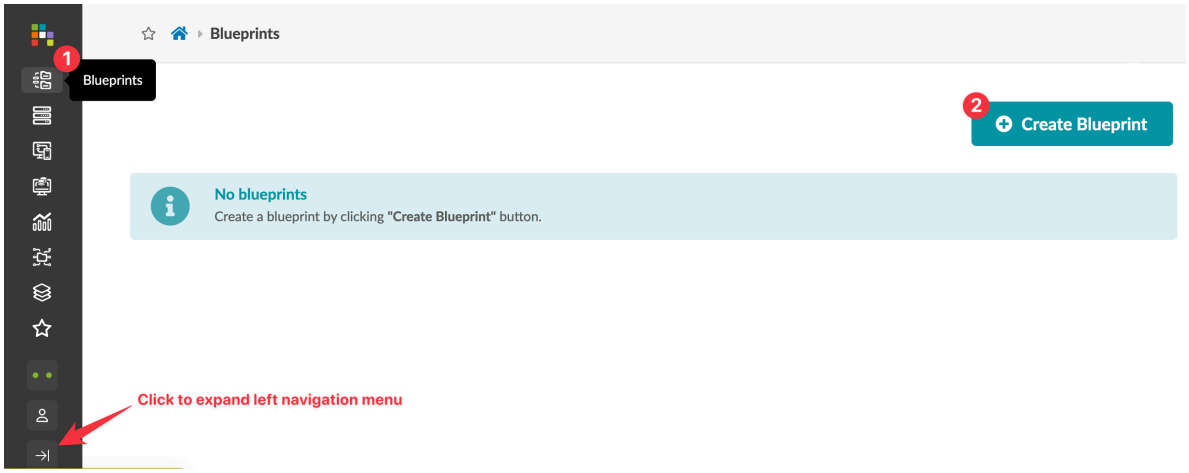
SUMMARY

Depending on your topology you'll create either a Datacenter blueprint or a Freeform blueprint. Datacenter blueprints are for spine and leaf, or collapsed (spineless) topologies, and Freeform blueprints are for all other topologies.

IN THIS SECTION

Create Datacenter Blueprint

1. From the left navigation menu in the Apstra GUI, click **Blueprints**, then click **Create Blueprint**.



2. Enter a unique name and leave **Datacenter** selected for **Reference Design**.
3. Select a "template" on page 889 from the **Template** drop-down list. A preview shows template parameters, topology preview, structure, logical structure, and policies. (The example shows the template for L2 Virtual.)

Create Blueprint

Blueprint parameters

Name *

datacenter

Reference Design *

- Datacenter
- Freeform

Filter Templates

- All
- RACK BASED
- POD BASED
- COLLAPSED

Template *

Select template from drop-down list

L2 Virtual

Spine to Leaf Links Underlay Type

- IPv4
- IPv6 RFC-5549
- IPv4-IPv6 Dual Stack

Spine to Superspine Links

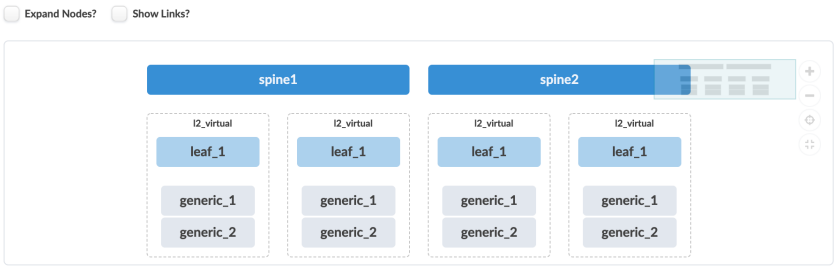
- IPv4
- IPv6 RFC-5549
- IPv4-IPv6 Dual Stack

Expanded View | Compact View

Template Parameters

Name	L2 Virtual
Type	RACK BASED

Topology Preview

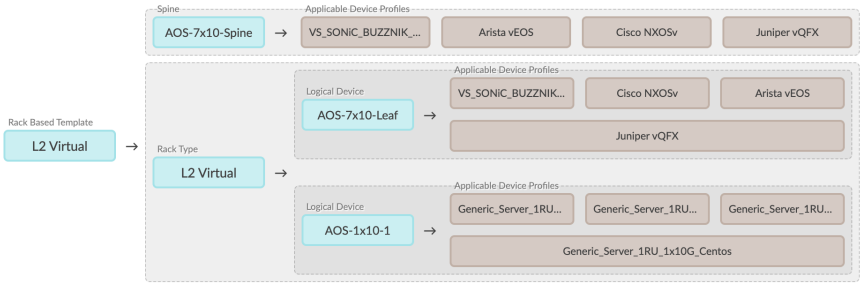


Structure

Spines	2 of AOS-7x10-Spine
Tags	
Rack Types	4 of L2 Virtual (2 generic systems)

Logical Structure

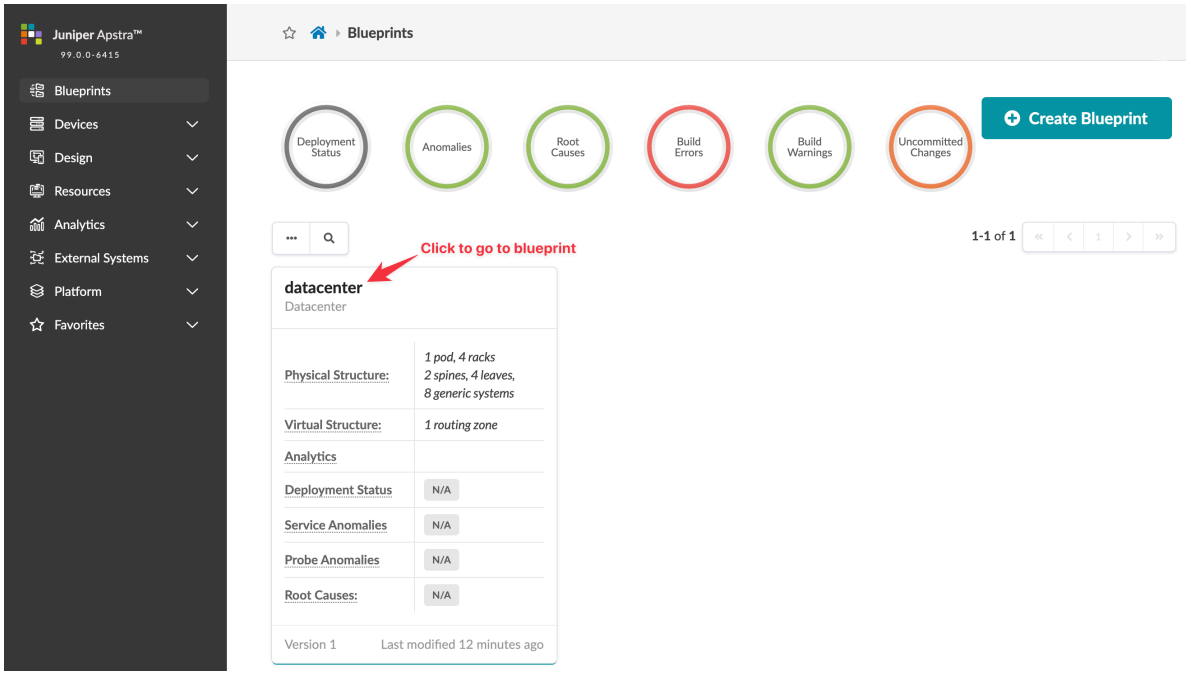
This shows you all the logical relationships and dependencies chosen so you can validatethat the design matches your expectations.



Policies

Overlay Control Protocol	Pure IP Fabric
ASN Allocation Policy (spine)	Unique

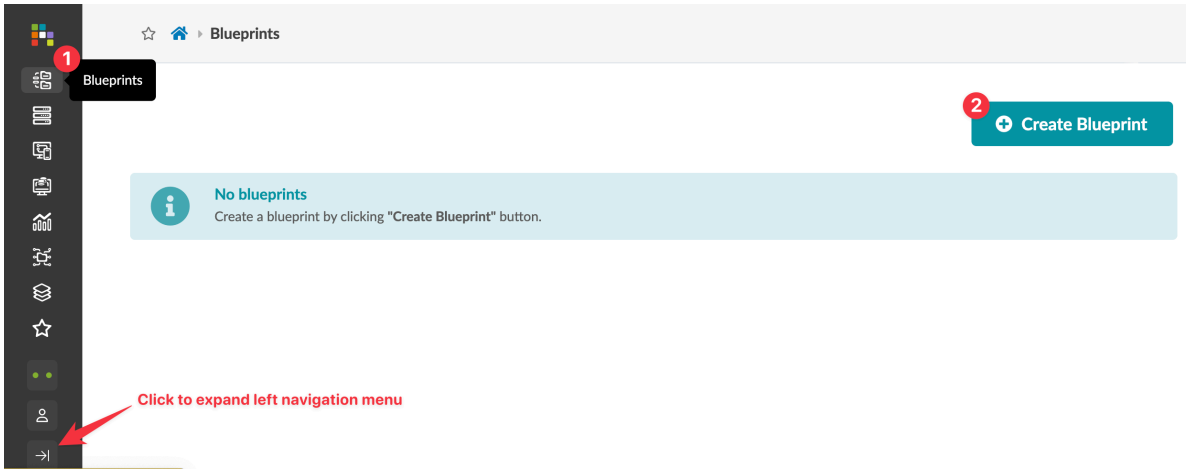
4. Click **Create** to create the blueprint and return to the **Blueprints** page where you'll see a summary of your new blueprint.



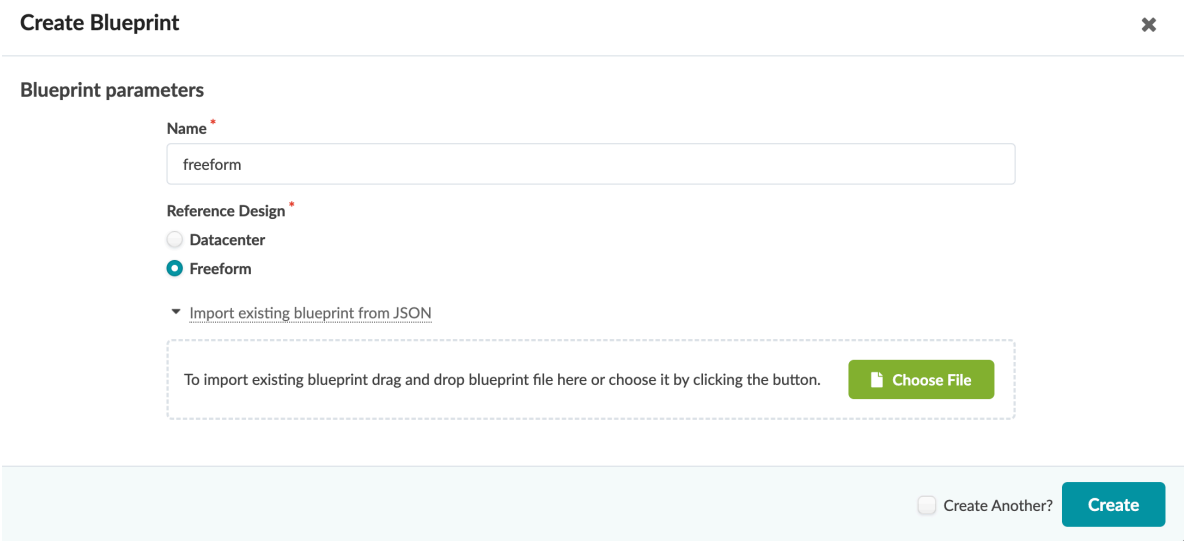
Next Steps: Build the Physical environment in Apstra by "assigning ASNs and IP Addresses" on page 58 , "interface maps (device profiles)" on page 62 and "system IDs" on page 64.

Create Freeform Blueprint

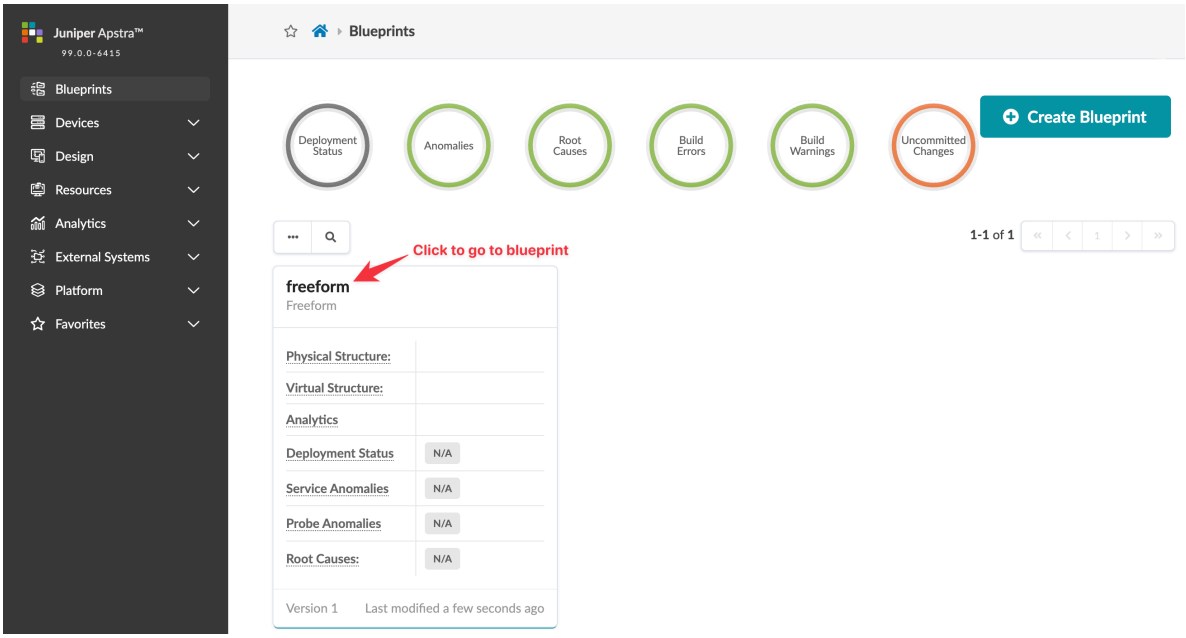
1. From the left navigation menu in the Apstra GUI, click **Blueprints**, then click **Create Blueprint**.



2. Enter a unique blueprint name and select **Freeform** for **Reference Design**.



3. If you've previously "exported" on page 503 a Freeform blueprint, you can use it as a "template" on page 889 for a new one. Click **Import existing blueprint from JSON**. Then either click **Choose File** and navigate to the downloaded file, or drag and drop the file into the dialog window. Otherwise, continue to the next step.
4. Click **Create** to create the blueprint and return to the **Blueprints** page where you'll see a summary of your new blueprint.



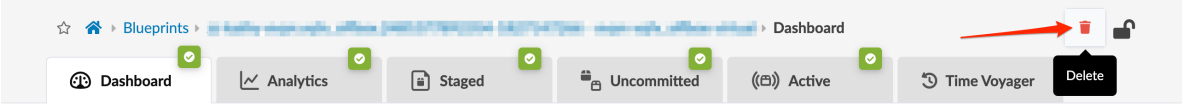
Next Steps:

- "Import device profiles" on page 583 into the Freeform blueprint catalog.
- You can "bring your devices under Apstra management" on page 651 anytime before deploying your network.

Delete Blueprint

You must have "permission" on page 1263 to delete blueprints. (Permissions are based on the roles you've been assigned as a user.)

1. From the blueprint, click **Dashboard**, then click the **Delete** button (top-right).



2. Enter the blueprint name, then click **Delete** to delete the blueprint and go to the **Blueprints** page.

3

PART

Blueprint Analytics

What are Blueprint Analytics

Managed devices generate large amounts of data over time. On their own these data are voluminous and unhelpful. With Intent-Based Analytics (IBA) you can combine intent from the "graph" on page 1764 with current and historic data from devices to reason about the network at-large.

Data generated by devices are ingested via "agents" on page 678 and sent to the Apstra server. With the use of "probes" on page 29, data can be aggregated across devices in response to operator configuration. Combining probes with intent from the blueprint graph generates a reduced set of data that can be more easily reasoned about. You can directly inspect advanced data from the Apstra GUI or from "REST API" on page 1351 to gain real-time insight about the network. It can also be streamed out with our existing streaming infrastructure. Also, based on the state of this advanced data, "anomalies" on page 27 can be raised.

While operating IBA at scale, using many probes, disk usage can grow significantly within the Apstra server VM. This is expected because the system will persist at least enough samples to maintain data for the requested duration for all time-series for all existing probes. Additionally, the system will create checkpoint (backup) files up to a configured limit. Settings in the `/etc/aos/aos.conf file` indicate how often to rotate logs and remove old checkpoint files. Using IBA can increase disk usage to tens of gigabytes. If this is an issue, you can adjust the log rotation settings to reduce disk usage.

Additional space may be used by system snapshots and old images from any in-place Apstra server upgrades. These can be deleted or moved off the system to increase free disk space.

Agents ingest data that devices generate and send them to the Apstra server. With IBA "probes" on page 29, you can aggregate data across devices based on how they are configured. Combining probes with intent from the blueprint graph generates a reduced set of data. You can directly inspect advanced data from the Apstra GUI or from "REST API" on page 1351 to gain real-time insight about the network. You can stream data out with our existing streaming infrastructure. Also, based on the state of this advanced data, probes can raise "anomalies" on page 27.

While operating IBA at scale, using many probes, disk usage can grow significantly within the Apstra server VM. This is expected because the system will persist at least enough samples to maintain data for the requested duration for all time-series for all existing probes. Additionally, the system will create checkpoint (backup) files up to a configured limit. Settings in the `/etc/aos/aos.conf file` indicate how often to rotate logs and remove old checkpoint files. Using IBA can increase disk usage to tens of gigabytes. If this is an issue, you can adjust the log rotation settings to reduce disk usage.

System snapshots and old images from in-place Apstra server upgrades may use additional space. You can delete them or move them off the system to increase free disk space.

CHAPTER 1

Dashboards

IN THIS CHAPTER

What are Blueprint Analytics Dashboards

IN THIS SECTION

Blueprint Analytics Dashboards

Analytics dashboards monitor the network and raise alerts. Widgets are used within each dashboard to monitor different aspects of the network and raise alerts to relevant anomalies.

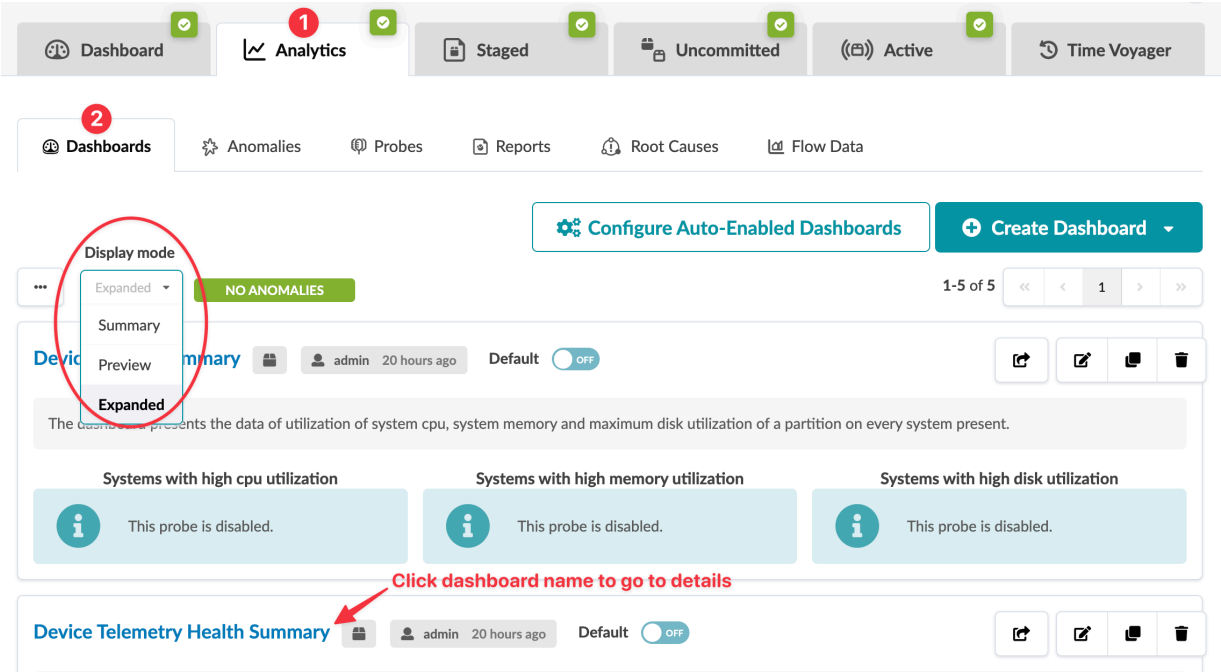
Some characteristics of analytics dashboards include:

- Specific dashboards are automatically created and enabled based on the state of the active (operational) blueprint.

- You can't configure the trigger logic that determines when dashboards are auto-created, but you *can* instantiate predefined dashboards and create your own dashboards.
- Probes that you've created and haven't modified are reused instead of creating duplicates of those probes.
- When you enable a dashboard, the required probes and widgets are instantiated. If you update or delete associated probes, the dashboard may enter an invalid state. Invalid dashboards are not automatically repaired.
- When upgrading the controller, the auto-creation behavior of dashboards occurs on preexisting active blueprints, in the same way as for newly-created blueprints.
- You can export dashboards from one blueprint and import them into other blueprints (new in Apstra version 5.0.0).

Blueprint Analytics Dashboards in the Apstra GUI

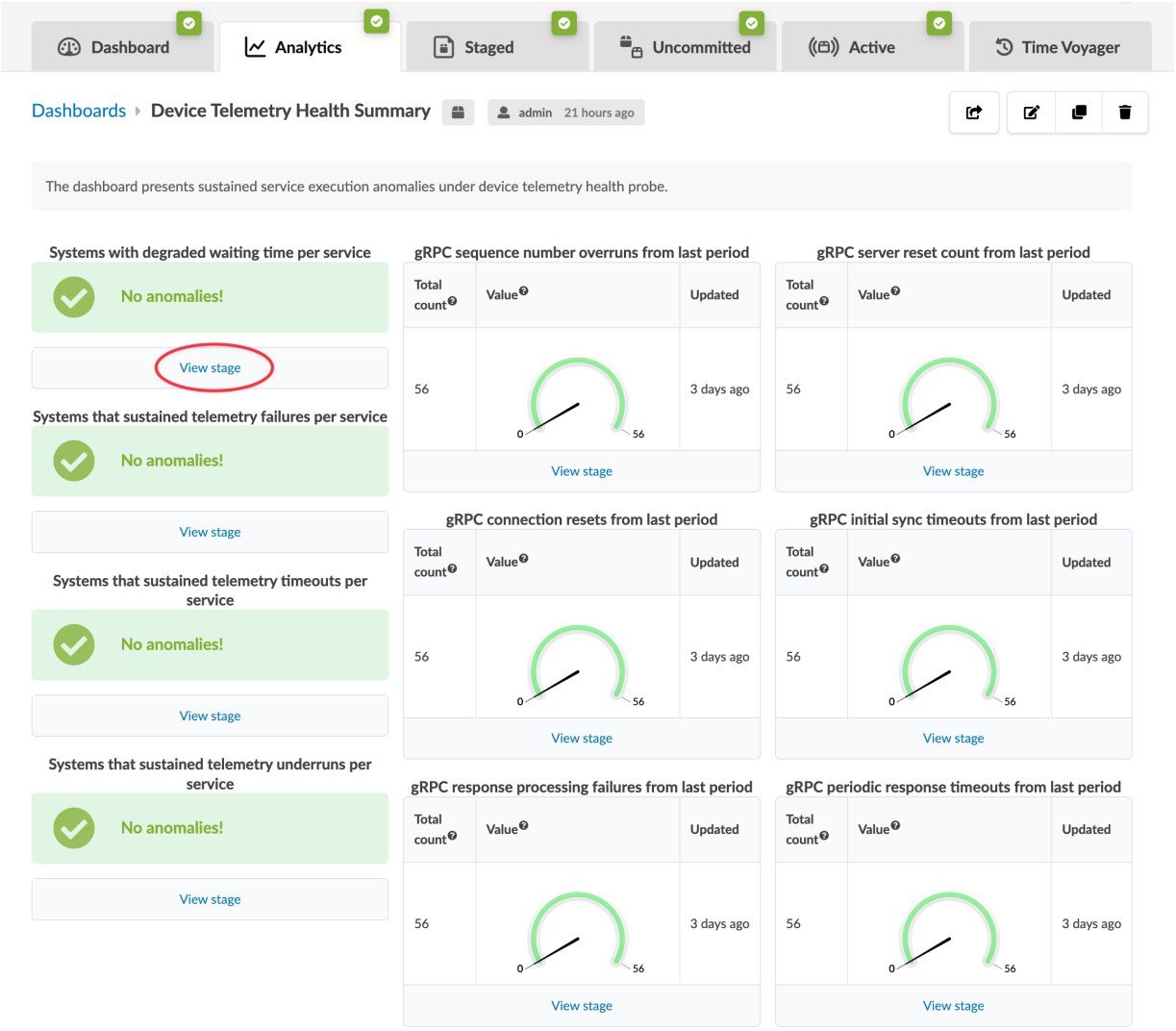
From the blueprint, navigate to **Analytics > Dashboards** to go to the analytics dashboard.



- The **Display Mode** defaults to the **Expanded** view. To see the dashboards in different levels of detail, select **Summary** or **Preview** from the drop-down list.
- System-generated dashboards are labeled with **System** and user-generated (and user-modified) dashboards are labeled with the user's name ('admin' in the screenshot above).

- You can display analytics dashboards on the "Blueprint Dashboard" on page 53 to have additional network monitoring information on that main screen. To add them, turn **ON** the analytics dashboards' default toggles.

To see details of a dashboard, click its name, and to see more details click **View stage** on one of the widgets.

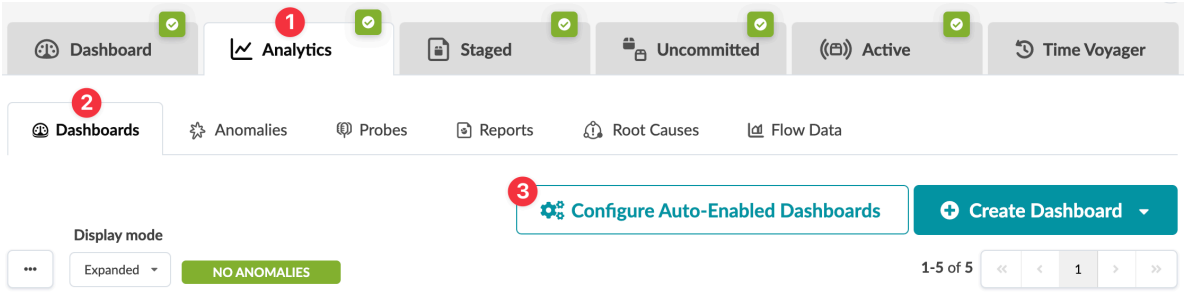


You can configure auto-enabled dashboards, instantiate predefined dashboards, create, update, export, import, and delete analytics dashboards, as described in subsequent pages.

Configure Auto-Enabled Blueprint Analytics Dashboards

Certain auto-enabled dashboards generate anomalies that are expected, so you may not want to see them. To suppress these anomalies, either proactively set the auto-enable toggle for the dashboard to **OFF** as described below, or "delete" on page 25 the dashboard after it has been enabled. Once a dashboard is disabled it won't be re-enabled unless the auto-enable toggle is set back to **ON** and the respective trigger is satisfied.

1. From the blueprint, navigate to **Analytics > Dashboards** and click **Configure Auto-Enabled Dashboards**.

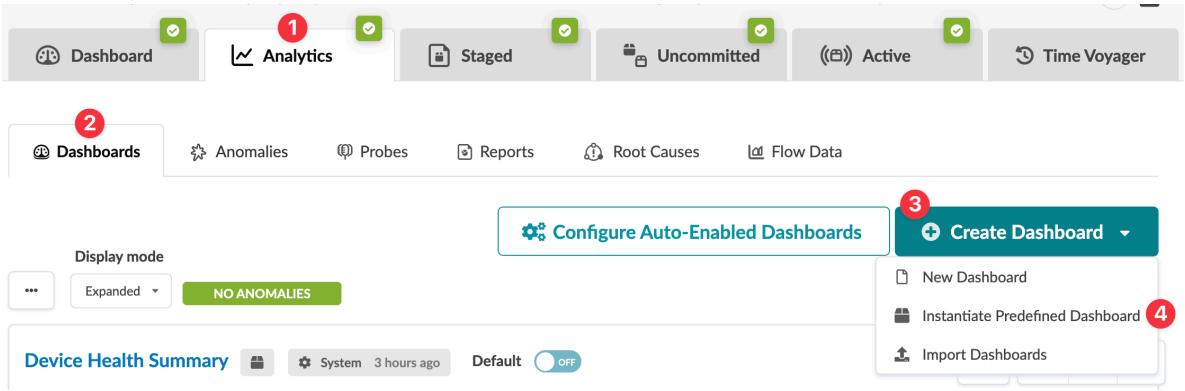


   The **Auto-Enabled Dashboards** page opens showing the table of automatically enabled dashboards, including their descriptions, widgets used, and toggles for automatic enablement.
2. Toggle the relevant dashboards **ON** to auto-enable them or **OFF** to disable auto-generation.

Instantiate Predefined Blueprint Analytics Dashboard

You can instantiate predefined dashboards and modify them to show analytics in multiple ways. You can instantiate more than one instance of any predefined dashboard.

1. From the blueprint, navigate to **Analytics > Dashboards**, click **Create Dashboard**, then select **Instantiate Predefined Dashboard** from the drop-down list.



2. Select a predefined dashboard from the drop-down list. For more information about individual predefined dashboards, see "Predefined Dashboards" on page 1532 in the References section.
3. Click **Create** to instantiate the dashboard and return to the **Analytics Dashboards** view.

Create Blueprint Analytics Dashboard

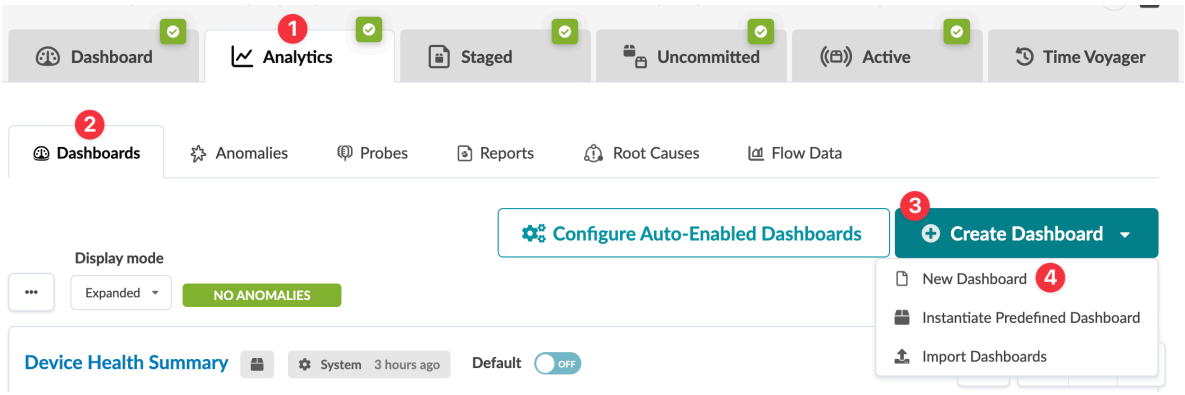
IN THIS SECTION

Some probes and dashboards are automatically created to give you immediate value. The probes auto-adjust based on the state of the blueprint (examples: undeployed or unassigned device, addition or removal of virtual infra managers). You can also create your own dashboards to display custom information from IBA probes and stages.

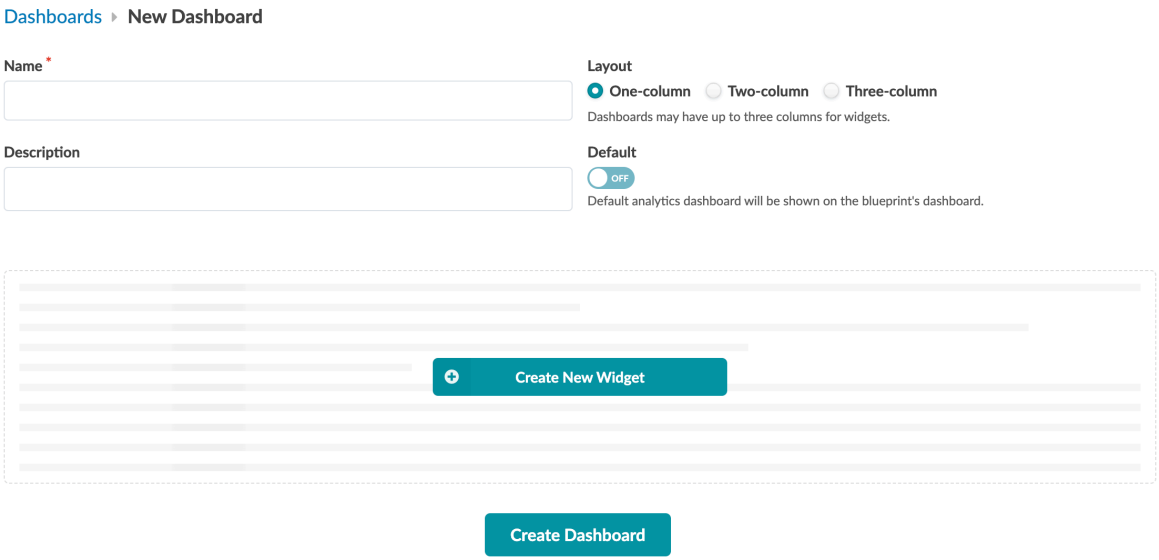
You can create a dashboard by manually adding widgets or by cloning an existing dashboard and customizing it.

Create New Dashboard

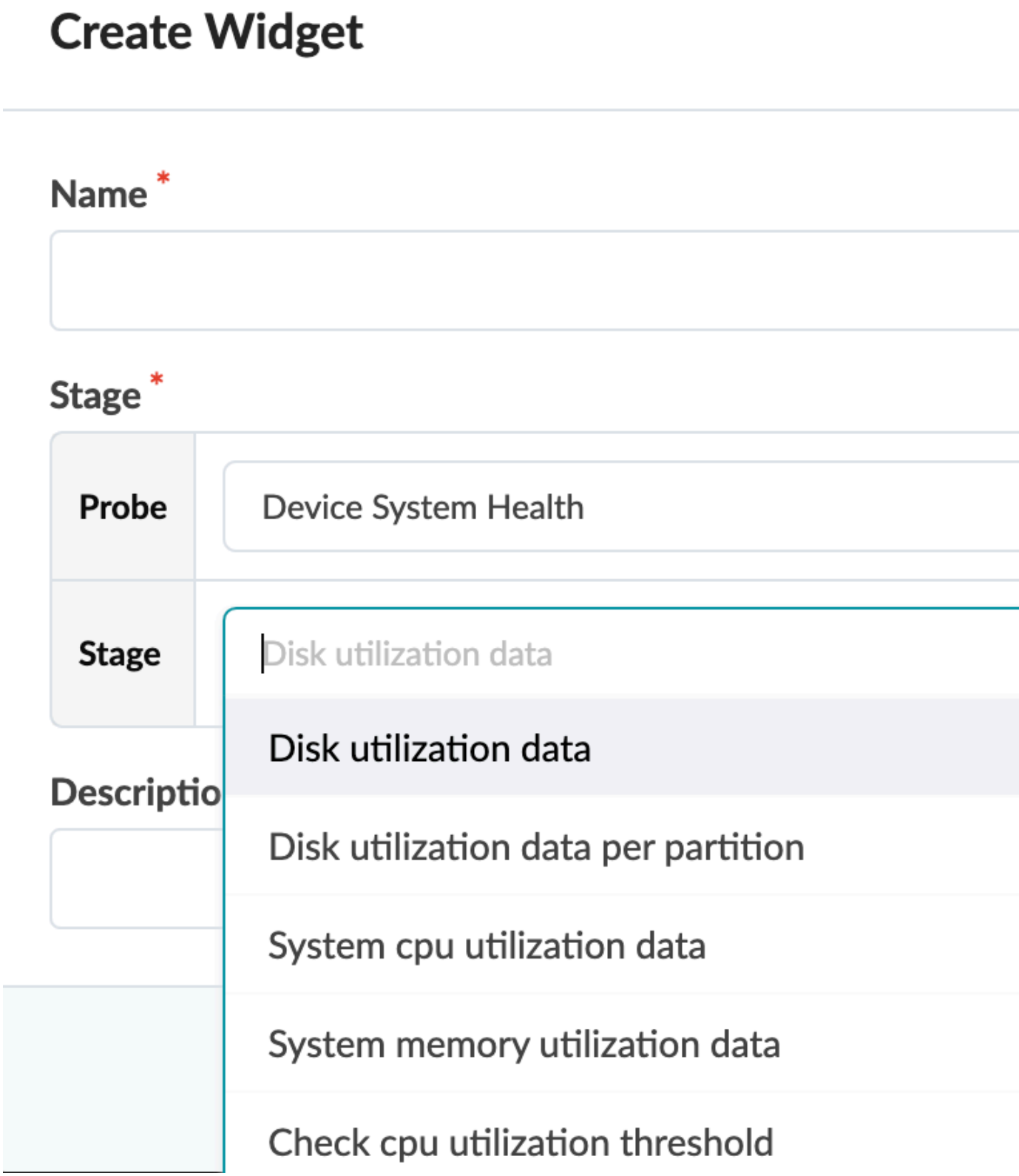
1. From the blueprint, navigate to **Analytics > Dashboards**, click **Create Dashboard**, then select **New Dashboard** from the drop-down list.



   The **New Dashboard** page opens.
2. Enter a unique name and (optional) description.



3. Select a layout (one-column, two-column, three-column) and if you want the dashboard to appear on the blueprint **Dashboard** tab, toggle on **Default**.
4. Click **Create New Widget**.
   The **Create Widget** dialog opens.
5. Enter a widget name, select a probe and one of its stages from the drop-down lists, then customize details as needed.



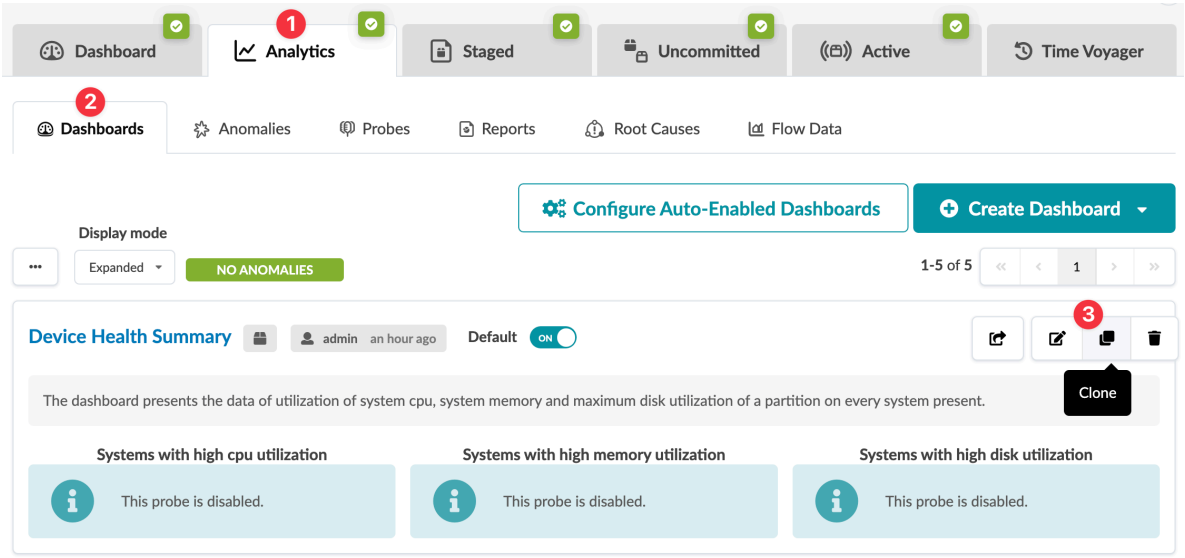
6. Add additional widgets in the same manner as above, as needed.

7. Click **Create Dashboard** to create the dashboard and go to the details view of the new dashboard.

   A large dashboard may take some time to create. You can monitor the status at the bottom of the screen under **Active Tasks**.

Clone Existing Dashboard

1. From the blueprint, navigate to **Analytics > Dashboard** and click the **Clone** button (top-right of dashboard) for the dashboard to copy.



   The **Clone** page for the selected dashboard opens.

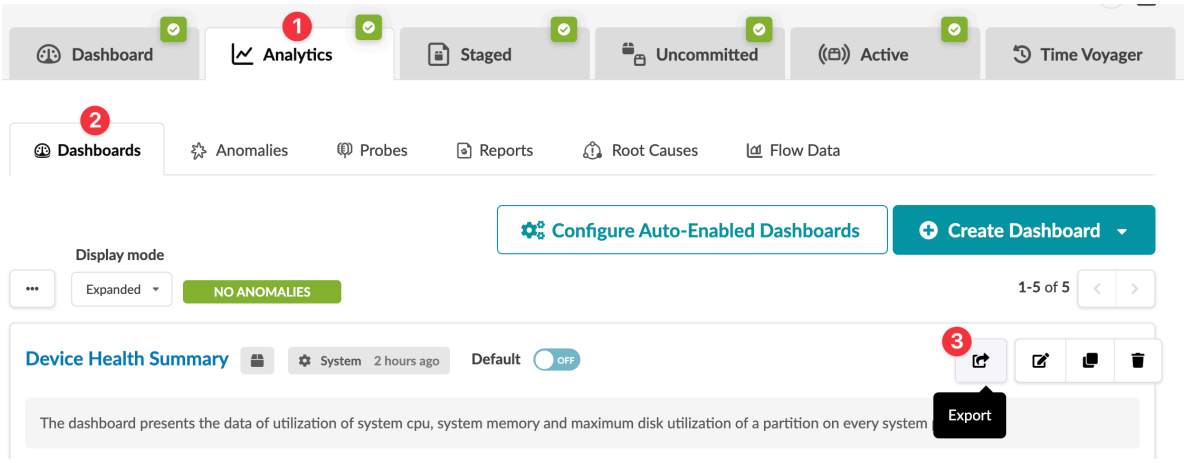
2. The dashboard name must be unique. Make your changes to customize the copied dashboard by creating, editing and/or removing widgets. Various other fields can be changed as well.
3. Click **Create Dashboard** to create the dashboard and go to the details view of the new dashboard.

   A large dashboard may take some time to create. You can monitor the status at the bottom of the screen under **Active Tasks**.

Export Blueprint Analytics Dashboard

You can export an analytics dashboard from one blueprint, then "import" on page 23 it into other blueprints (as of Apstra version 5.0.0).

1. From the blueprint, navigate to **Analytics > Dashboard** and click the **Export** button (top-right of dashboard) for the dashboard to export.



2. Click **Copy** to copy the contents, or click **Save As File** to download the file.

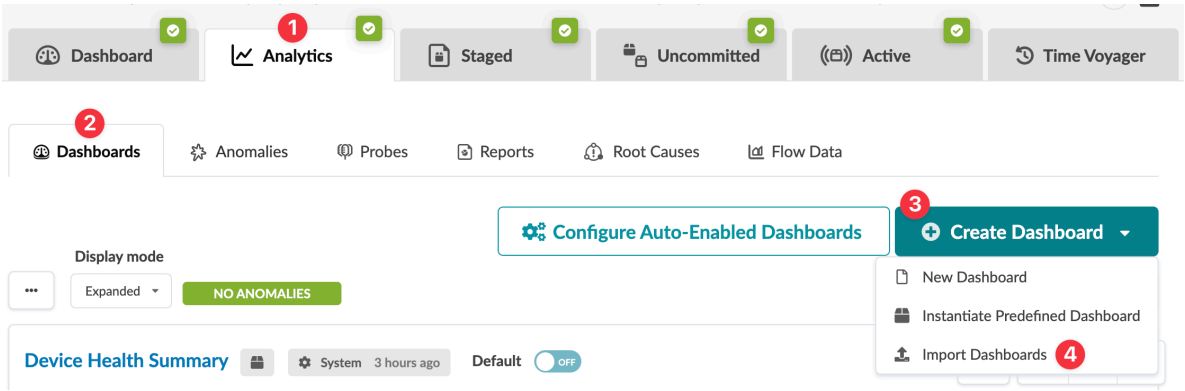
3. When you've copied or downloaded the dashboard, close the dialog to return to the **Analytics Dashboards** view.

You can "import" on page 23 exported dashboard into other blueprints.

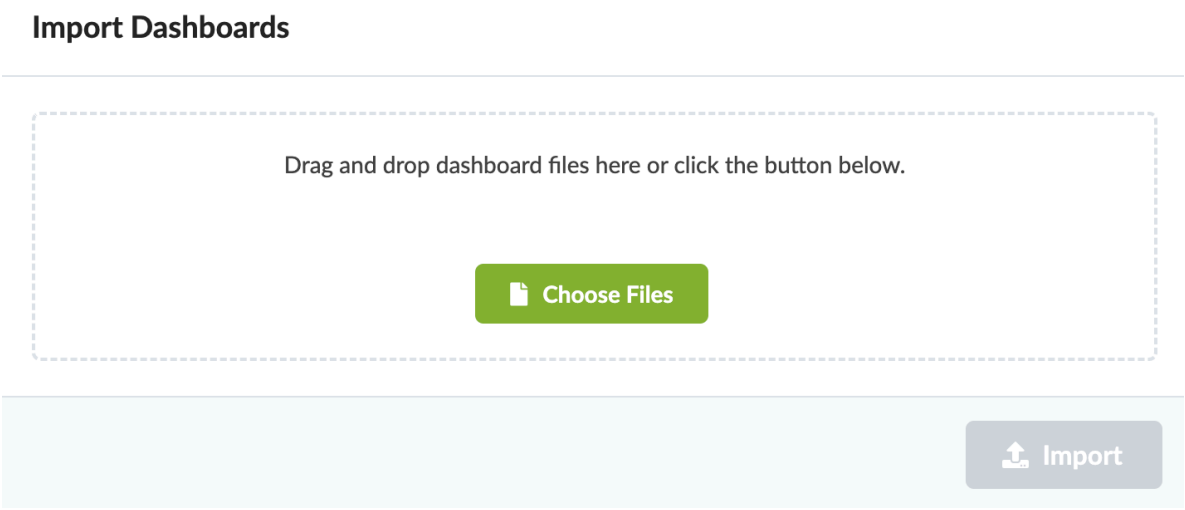
Import Blueprint Analytics Dashboard

After you've "exported" on page 22 an analytics dashboard from one blueprint, you can import it into other blueprints (as of Apstra version 5.0.0).

1. From the blueprint, navigate to **Analytics > Dashboard**, click **Create Dashboard**, then select **Import Dashboards** from the drop-down list.



2. Either click **Choose Files** and navigate to the file(s) on your computer, or drag and drop the file(s) from your computer into the dialog window.

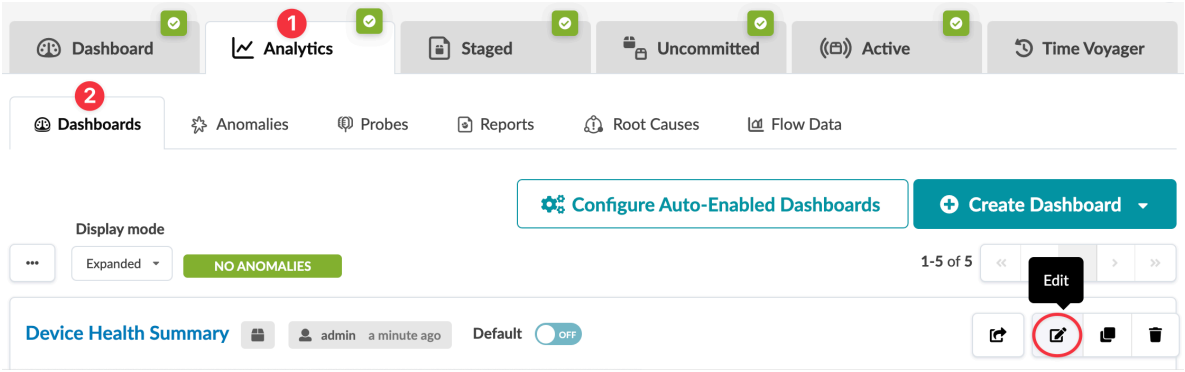


3. Click **Import** to import the dashboard(s) and return to the **Analytics Dashboards** view.

You can start using the analytics dashboard in this blueprint.

Update Blueprint Analytics Dashboard

1. From the blueprint, navigate to **Analytics > Dashboards** and click the **Edit** button for the dashboard to update.

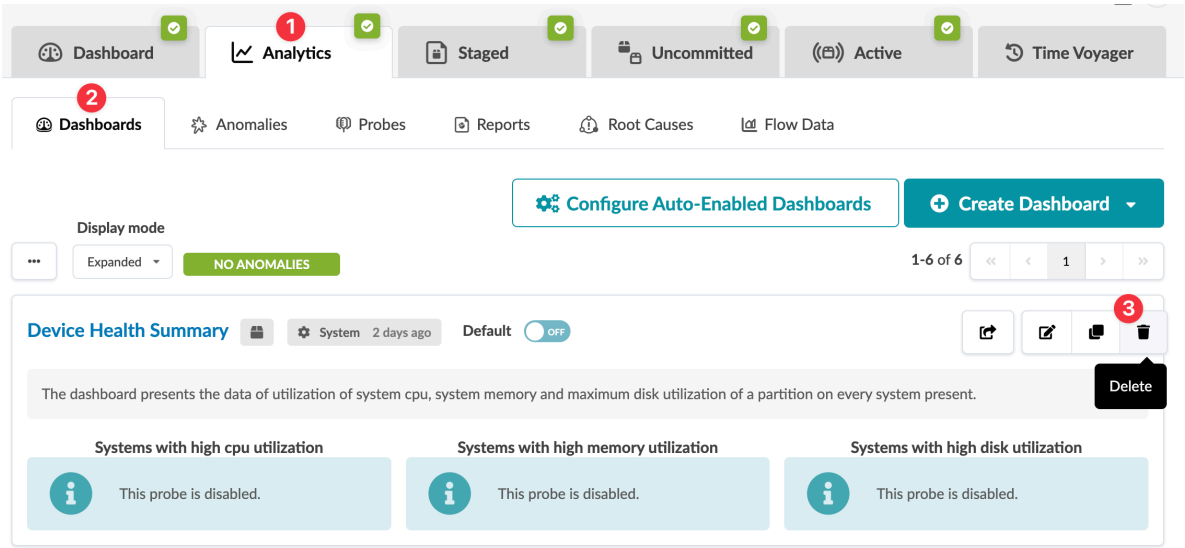


The **Edit** page opens for the selected dashboard.

2. Make your changes by creating, editing and/or removing widgets. Various other fields can be changed as well.
3. Click **Update Dashboard** to update the dashboard and return to the selected dashboard.

Delete Blueprint Analytics Dashboard

1. From the blueprint, navigate to **Analytics > Dashboards** and click the **Delete** button for the dashboard to delete.



The **Delete this Dashboard?** dialog opens.

2. To delete all probes that are exclusively used in this dashboard, select the **Delete probes used in the dashboard** check box. Deleting unnecessary probes frees up disk space.
3. Click **Delete Dashboard** to delete the dashboard and return to the **Analytics Dashboards** view.

If you're deleting a predefined dashboard (because it doesn't apply to your network for example), the auto-enable feature is disabled so it doesn't reappear automatically. If you want to re-establish the dashboard you can "instantiate" on page 18 it manually.

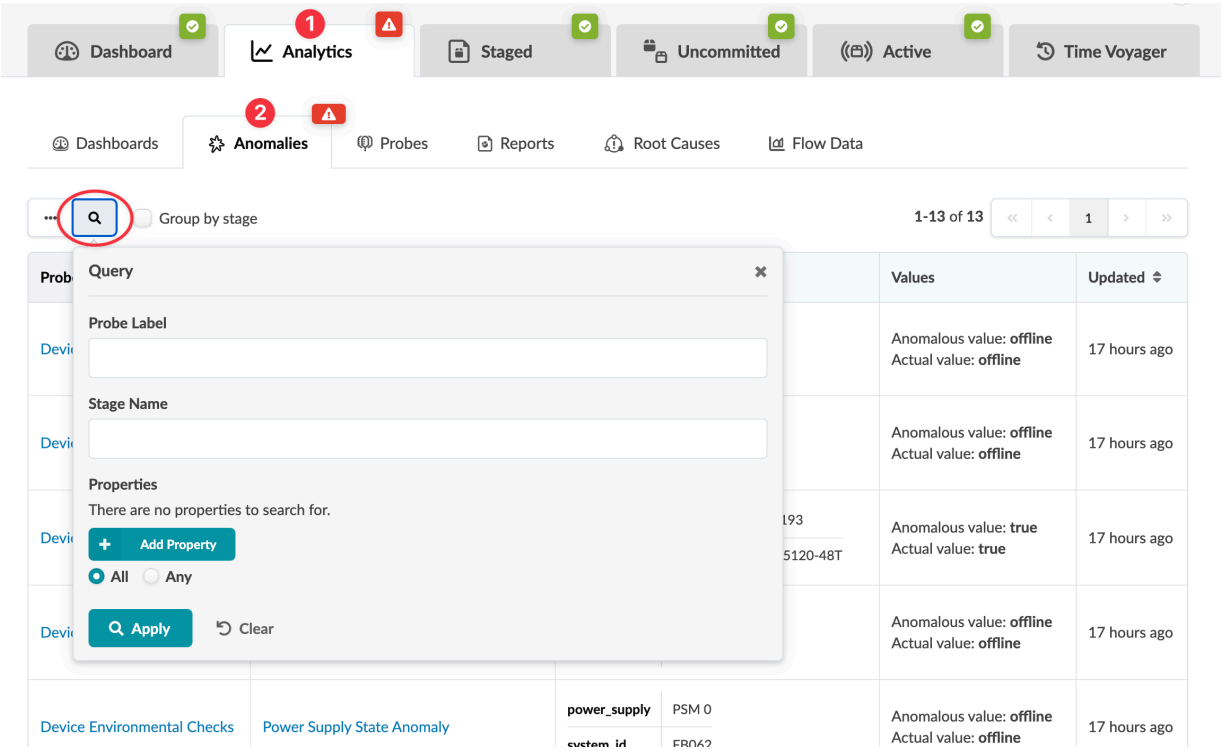
CHAPTER 2

Anomalies

IN THIS CHAPTER

Anomalies (Analytics)

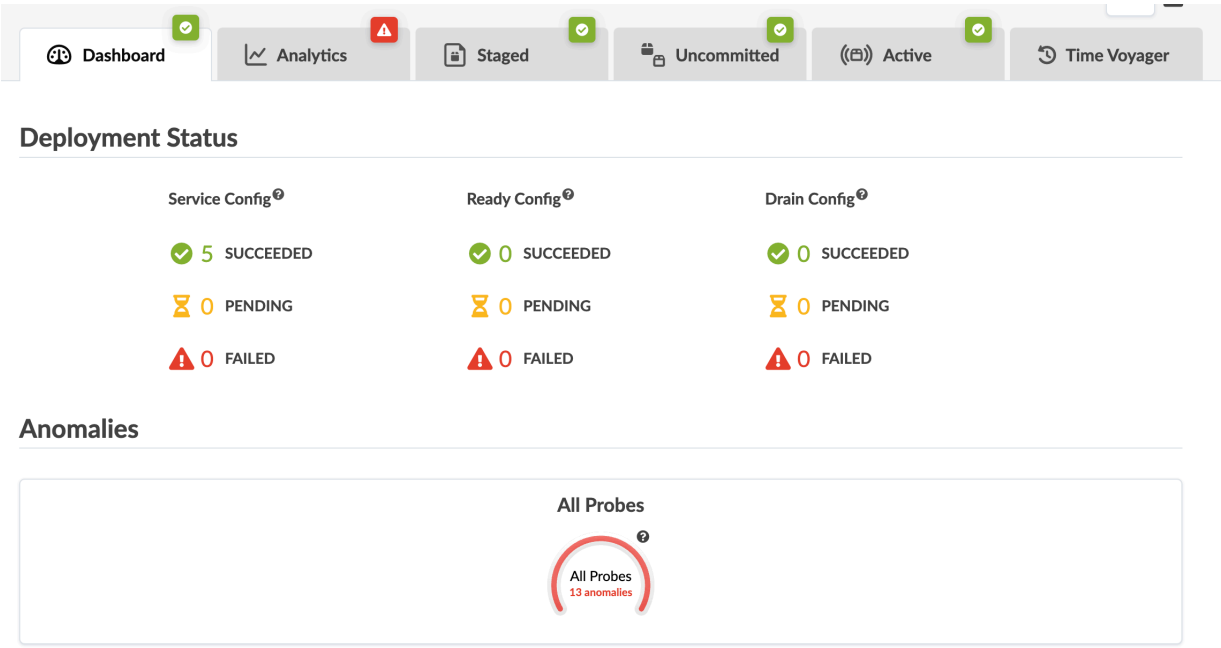
From the blueprint, navigate to **Analytics > Anomalies** to go to the list of anomalies that the IBA probes have detected. From the **Query** box, you can search for specific anomalies by filtering **Probe Label**, **Stage Name**, and other properties.



To display a condensed view of the anomaly count per probe/stage, check the **Group by stage** check box. Example: If three stages of the first of two probes are generating anomalies, and two stages of the

second probe are generating anomalies, **Group by Stage** shows five entries in a table, each one representing one stage with anomalies.

The "Blueprint Dashboard" on page 53 shows a summary of all anomalies including those that IBA probes generated. Clicking the **All Probes** gauge on the dashboard takes you to the list of anomalies (Analytics > Anomalies).



CHAPTER 3

Probes

IN THIS CHAPTER

What are Probes

IN THIS SECTION

Probes are the basic unit of abstraction in Intent-Based Analytics. Generally, a given probe consumes some set of data from the network, does various successive aggregations and calculations on it, and optionally specifies some conditions of said aggregations and calculations on which anomalies are raised.

Probes are Directed Acyclic Graphs (DAGs) where the nodes of the graph are processors and stages. Stages are data, associated with context, that can be inspected by the operator. Processors are sets of operations that produce and reduce output data from input data. The input to processors are one-or-

many stages, and the output from processors are also one-or-many stages. The directionality of the edges in a probe DAG represent this input-to-output flow.

Importantly, the initial processors in a probe are special and do not have any input stage. They are notionally generators of data. We shall refer to these as source processors.

IBA works by ingesting raw telemetry from collectors into probes to extract knowledge (ex: anomalies, aggregations and so on). A given collector publishes telemetry as a collection of metrics, where each metric has identity (viz, set of key-value pairs) and a value. IBA probes, often with the use of graph queries, must fully specify the identity of a metric to ingest its value into the probe. With this feature, probes can ingest metrics with partial specification of identity using ingestion filters, thus enabling ingestion of metrics with unknown identities.

Some probes are created automatically. These probes will not be deleted automatically. This keeps things simple operationally and implementation-wise.

Processors

The input processors of a probe handle the required configuration to ingest raw telemetry into the probe to kickstart the data processing pipeline. For these processors, the number of stage output items (one or many) is equal to the number of results in the specified graph query(s). If multiple graph queries are specified, for example. `graph_query: [A, B]`, and query A matches 5 nodes and query B matches 10 nodes, results of query A will be accessible using `query_result` indices from 0 to 4, and results of query B using indices from 5 to 14.

If a processor's input type and/or output type is not specified, then the processor takes a single input called **in**, and produces a single output called **out**.

Some processor fields are called **expressions**. In some cases, they are **graph queries** and are so noted. In other cases, they are Python **expressions** that yield a value. For example, in the Accumulate processor, duration may be specified as integer with seconds, for example `900`, or as an expression, for example `60 * 15`. However, expressions could be more useful: there are multiple ways to parameterize them.

Expressions support string values. Processor configuration parameters that are strings and support expressions should use special quoting when specifying static value. For example, `state: "up"` is not valid because it refers to the variable "up", not a static string, so it should be: `state: '"up"'`.

An expression is always associated with a graph query and is run for every resulting match of that query. The execution context of the expression is such that every variable specified in the query resolves to a named node in the associated match result. For more information, see "Service Collector" on page 1695 example.

Graph-based processors have been extended with query_tag_filter, which enables you to filter graph query results by tags. In IBA probes, tags are used only as filter criteria for servers and external routers, specifically for the ECMP Imbalance (External Interfaces) probe and the Total East/West Traffic probe. For specific processor information, see "Probe Processors" on page 1618 in the References section.

Ingestion Filters

With "ingestion filters" one query result can ingest multiple metrics into a probe. Table data types are used to store multiple metrics as part of a single stage output item. Table data types include `table_ns`, `table_dss`, `table_ts` - to correspond to existing types - `ns`, `dss`, `ts` -respectively.

IBA Collection Filter

Collection filters determine the metrics that are collected from the target devices.

A collection filter for a given collector on a given device, is simply a collection of ingestion filters present in different probes. You can also specify it as part of enabling a service outside the context of IBA or probes but existing precedence rules for service enablement apply here - only filters at a given precedence level are aggregated. When multiple probes specify an ingestion filter targeting a specific service on a specific device, the metrics collected are a union - in other words, a metric is published when it matches any of the filters. This is why, the data is also filtered by the controller component prior to ingesting into the IBA probes.

This filter is evaluated by telemetry collectors, often to better control even what subset of available metrics is fetched from the underlying device operating system (for example, to fetch only a subset of routes instead of getting all routes, which can be a huge number). In any case, only the metrics matching the collection filter are published as the raw telemetry.

As part of enabling a service on a device, you can now specify collection filters for services. This filter becomes an additional input provided to collectors as part of "self.service_config.collection_filters".

IBA Filter Format

Following are the design/usability goals for filters (ingestion and collection)

1. Ease of authoring - given probe authors are the ones specifying it
    - Most often cases are match any, match against a given list of possible values, equality match, range check if key has numeric values.
2. Efficient evaluation - given the filters are evaluated in the hot paths of collection or ingestion.
3. Aggregatable - multiple filters are aggregated so this aggregation logic need not become the responsibility of individual collectors.
4. Programming language neutral - components operating on filters can be in Python or C++ or some other language in future.
5. Programmable - be amenable to future programmability around the filters, by the controller itself and/or collectors, to enhance things like usability, performance and so on.

Considering the above goals, following is a suggested and illustrative schema for filter1. Refer to ingestion filter sections for specific examples to understand this better.

```
FILTER_SCHEMA = s.Dict(s.Object(
  'type': s.Enum(['any', 'equals', 'list', 'pattern', 'range', 'prefix']),
  'value': s.OneOf({
    'equals': s.OneOf([s.String(), s.Integer()]),
    'list': s.List(s.String(), validate=s.Length(min=1)),
    'pattern': s.List(s.String(), validate=s.Length(min=1)),
    'range': s.AnomalyRange(), validate=s.Length(min=1),
    'prefix': s.Object({
      'prefixsubnet': s.Ipv6orIpv4NetworkAddress(),
      'ge_mask': s.Optional(s.Integer()),
      'le_mask': s.Optional(s.Integer()),
      'eq_mask': s.Optional(s.Integer())
  })
), key_type=s.String(description=
  'Name of the key in metric identity. Missing metric identity keys are '
  'assumed to match any value'))
```

One instance of filter specification is interpreted as **AND** of all specified keys (aka per-key constraints). Multiple filter specifications coming from multiple probes are considered as **OR** at the filter level.

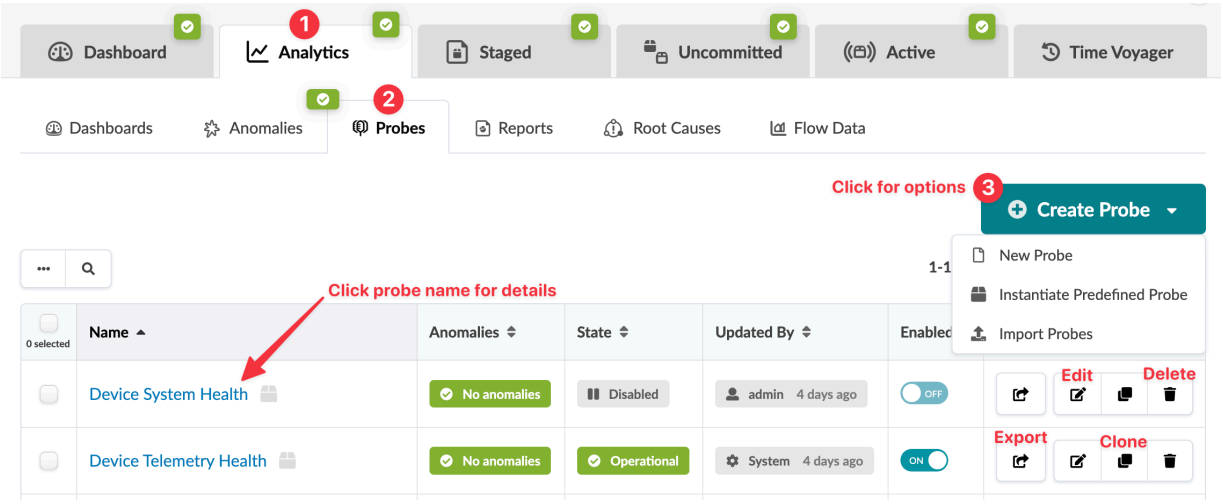
NOTE: The schema presented here is only for communicating the requirements. You can choose any way that accomplishes stated use cases.

Collector Processors additional_properties specified in collector processors' configuration can be accessed using the special `context.` namespace. For example, if a collector defines property `system_role`, it could be used this way:

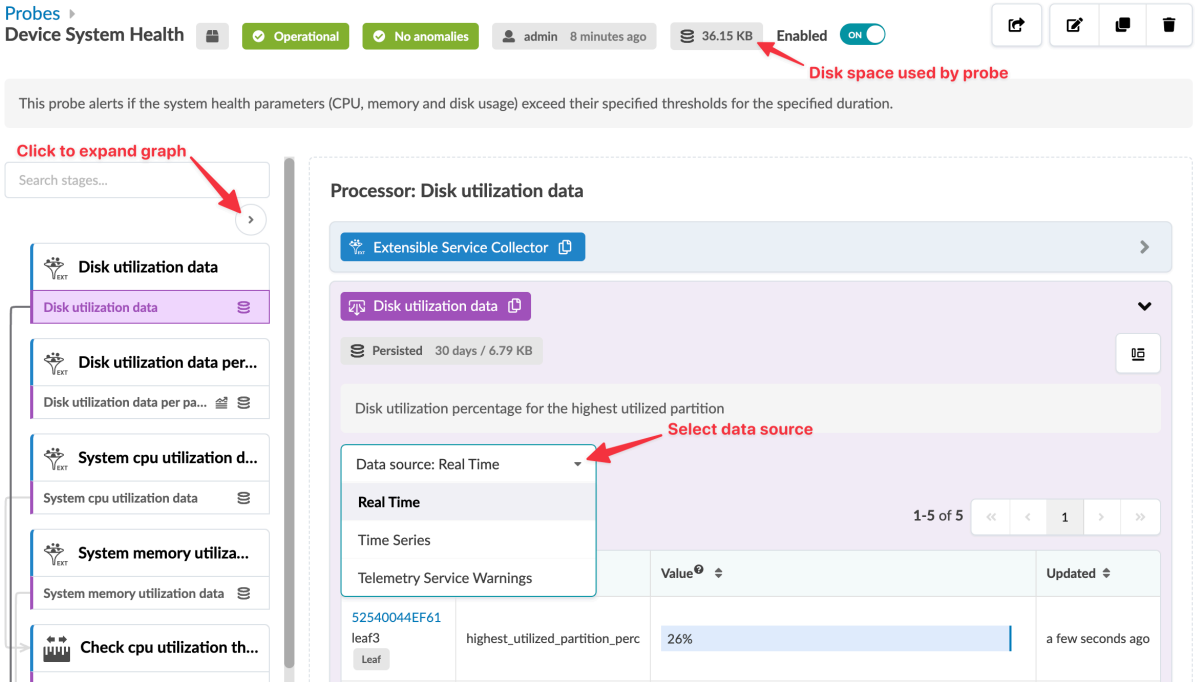
```
duration: 60 * (15 if context.system_role == "leaf" else 10)
```

NOTE: Items context is available as long as the items set is unchanged from the original set derived from the collector processor configuration. After data goes through a processor that changes this set, it's no longer available (for example, any grouping processor).

From the blueprint, navigate to **Analytics > Probes** to go to the probes table view. You can instantiate, create, clone, edit, delete, import, and export probes.

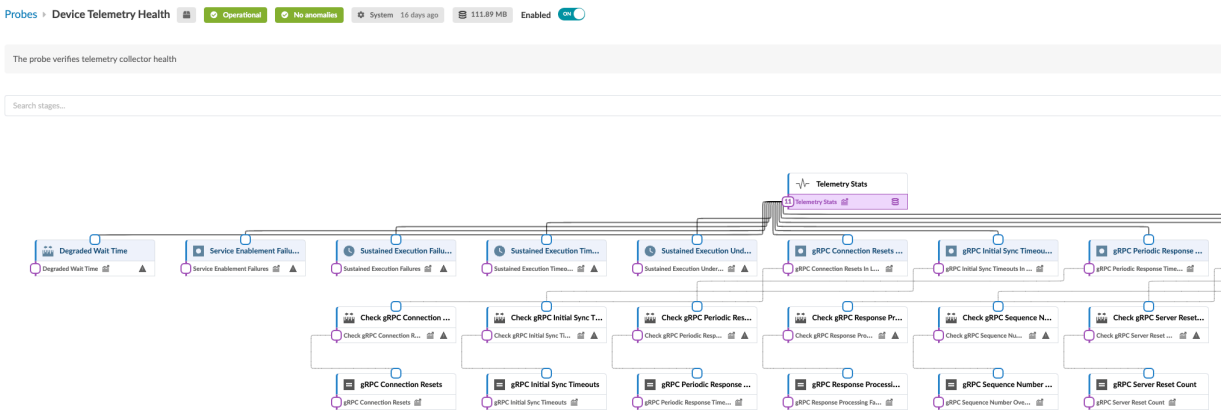


To go to a probe's details, click its name. You can enable/disable a probe from the table view or the detail view. You can display stages in some probes in various ways. For example, when you click the probe named **Device System Health**, you'll see the image below. You can select the data source (real time, time series, telemetry service warnings) then aggregate data in various ways. Also, you can see the disk space used on each probe, as applicable.



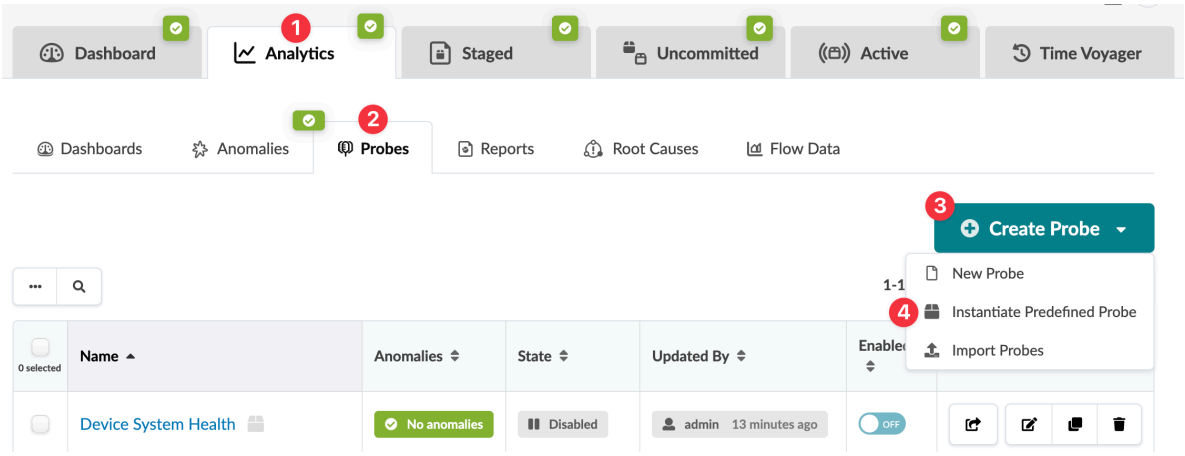
> **CAUTION**: If the Apstra controller has insufficient disk space, older telemetry data files are deleted. To retain older telemetry data, you can increase capacity with "Apstra VM Clusters" on page 1315.

The structure and logic of non-linear probes with tens of processors is not easily distinguished in the standard view. You can click the expand button (top of left panel) to see an expanded representation of how the processors are inter-related. For example, the image below shows part of the expanded view of the **Device Telemetry Health** probe.



Instantiate Predefined Probe

1. From the blueprint, navigate to **Analytics > Probes**, then click **Create Probe** and select **Instantiate Predefined Probe** from the drop-down list. For information on specific "predefined probes" on page 1535 see the References section.



The **Instantiate Predefined Probe** dialog opens.

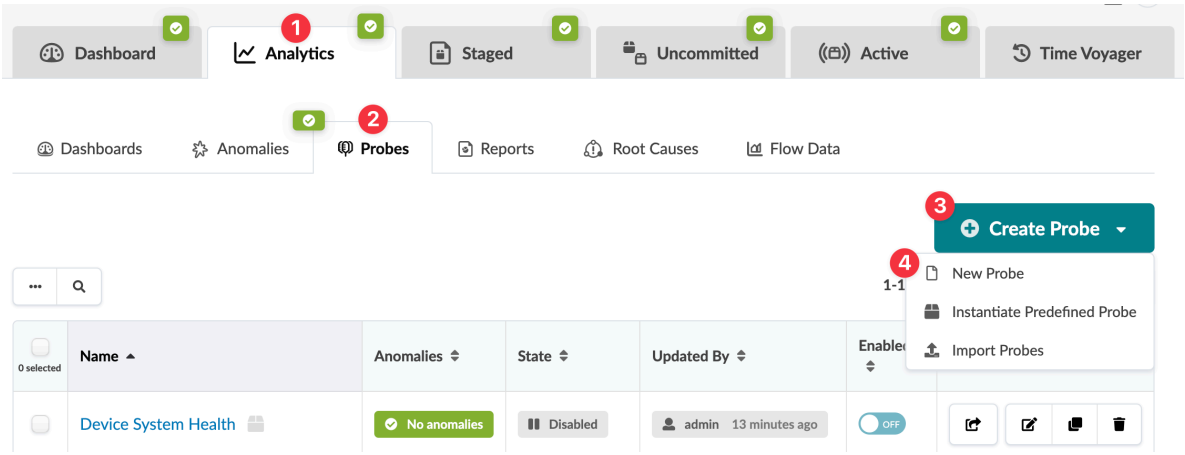
2. Select a predefined probe from the drop-down list. Probe names must be unique. If you instantiate the same probe more than once, you'll need to change its name.
3. Configure the probe to suit your anomaly detection requirements.
4. Click **Create** to instantiate the probe and return to the table view.

RELATED DOCUMENTATION

What are Probes | 29

Create Probe

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



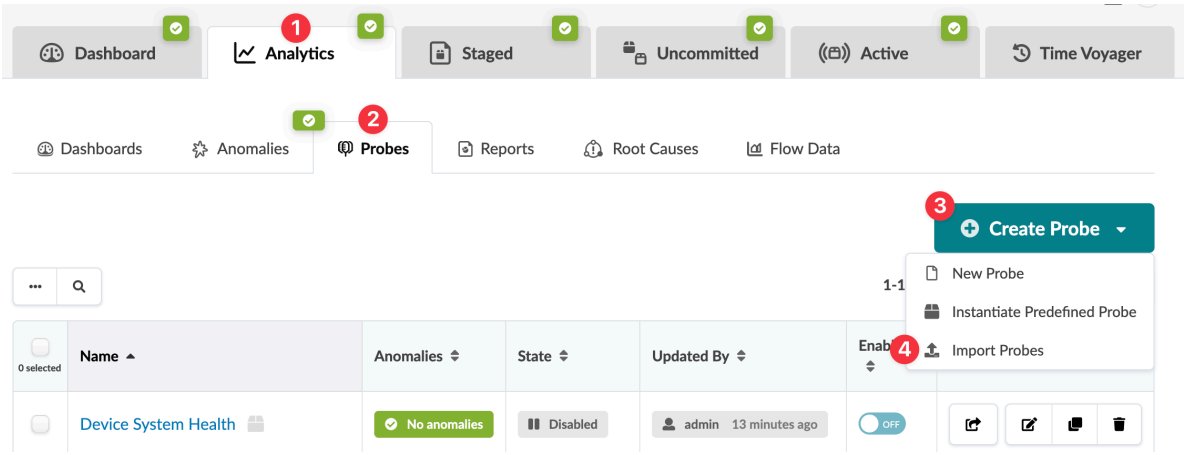
2. Enter a unique name and (optional) description.
3. Probes are enabled by default. This means that data is collected and processed (potentially creating anomalies) as soon as the probe is created. To disable the probe, toggle off **Enabled**. When you're ready to start collecting and processing data, "update the probe" on page 36 to enable it.
4. Click **Add Processor**, select a processor type, then click **Add** to add the processor to the probe. For more information about individual processors, see "Probe Processors" on page 1618 in the References section.
5. Customize inputs and properties as appropriate, or leave defaults as is.
6. Repeat the previous two steps until you've added all required processors for the new probe.
7. Click **Create** to create the probe and return to the table view.

RELATED DOCUMENTATION

What are Probes | 29

Import Probe

1. From the blueprint, navigate to **Analytics > Probes**, then click **Create Probe** and select **Import Probes** from the drop-down list.



   The **Import Probes** dialog opens.
2. Either click **Choose Files** and navigate to the JSON file(s) on your computer, or drag and drop the file(s) from your computer into the dialog window.
3. Click **Import** to import the probe and return to the table view.

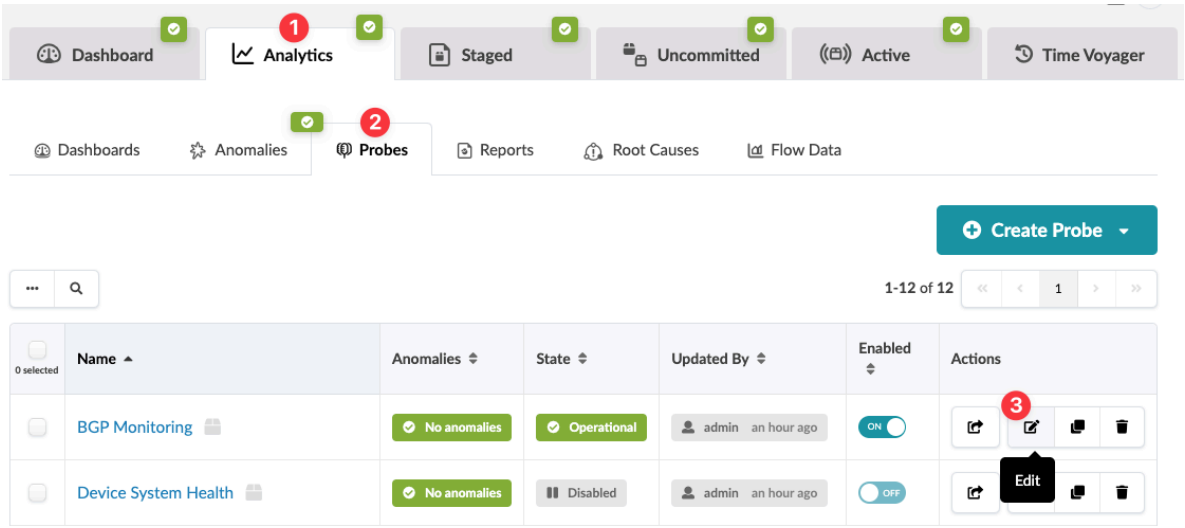
RELATED DOCUMENTATION

What are Probes | 29

Update Probe

Editing a probe affects any widgets and dashboards that are associated with it.

1. From the blueprint, navigate to **Analytics > Probes** and click the **Edit** button in the **Actions** panel for the probe to update.



The **Edit Predefined Probe** dialog opens.

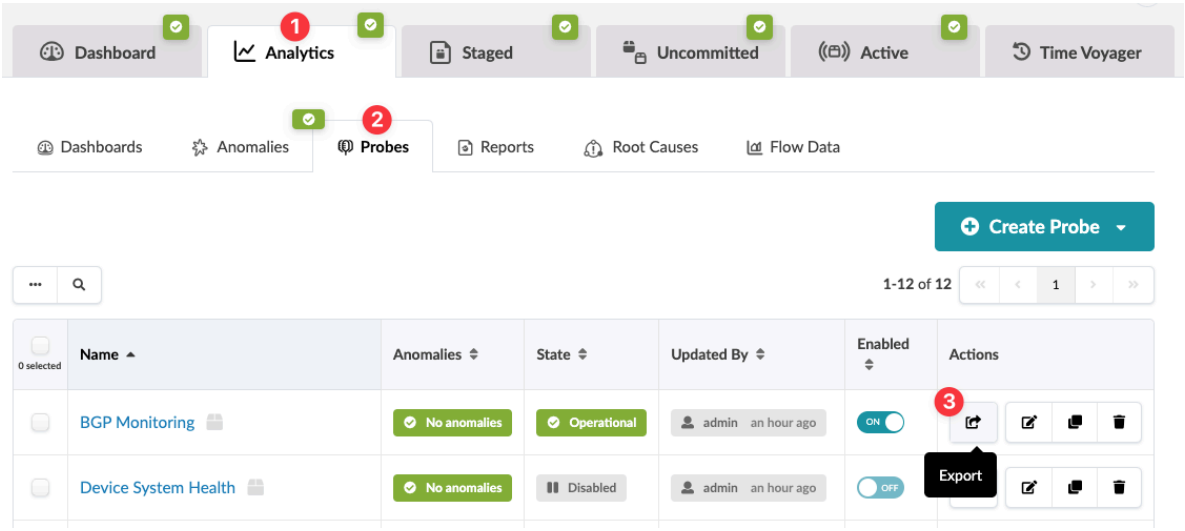
2. Make your changes.
3. Click **Update** to stage the changes and return to the table view.

RELATED DOCUMENTATION

What are Probes | 29

Export Probe

1. From the blueprint, navigate to **Analytics > Probes** and click the **Export** button in the **Actions** panel for the probe to export.



The **Export** dialog opens.

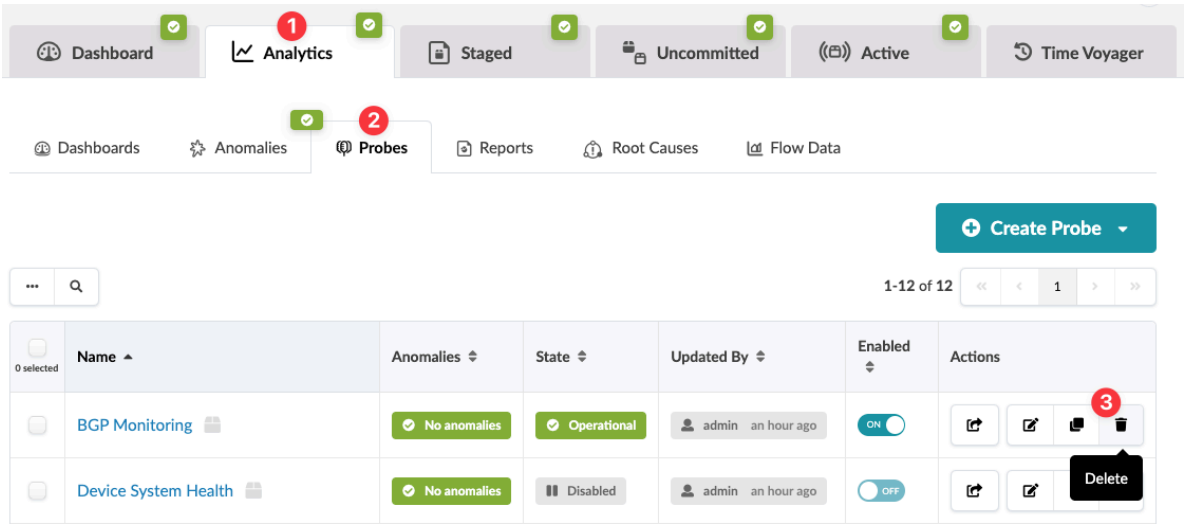
2. To copy the contents, click **Copy**, then paste it, or to download the JSON file to your local computer, click **Save as File**.
3. When you've copied and/or downloaded the file, click the **X** to close the dialog.

RELATED DOCUMENTATION

What are Probes | 29

Delete Probe

1. From the blueprint, navigate to **Analytics > Probes** and click the **Delete** button in the **Actions** panel for the probe to delete.



The **Delete this Probe?** dialog opens.

2. Click **Delete Probe** to stage the deletion and return to the table view.

CHAPTER 4

Predefined Reports

IN THIS CHAPTER

Analytics Reports Introduction

In the Apstra GUI, you can generate four types of predefined analytic reports for your blueprint.

Apstra supports the following predefined reports:

- Device Health Report

  The Device Health report analyzes the health of the device, including inventory overview, memory usage analysis, and CPU analysis. To generate the report, the probes for device system health and device telemetry health must be enabled.

- Optical Transceiver (XCVR) Report

  The Optical XCVR report analyzes optical transceivers interface statistics, and telemetry patterns and trends. To generate the report, the probes for optical transceivers and device traffic must be enabled.

  Traffic Report

- Traffic Report

  The Traffic report analyzes device traffic patterns and trends. To generate the report, the probes for device traffic and device system health must be enabled.

- Environmental Data Report

  The environmental data report analyzes the device power supply, fan, and temperature related measurements. To generate the report, the probes for device traffic and device system health must be enabled.

RELATED DOCUMENTATION

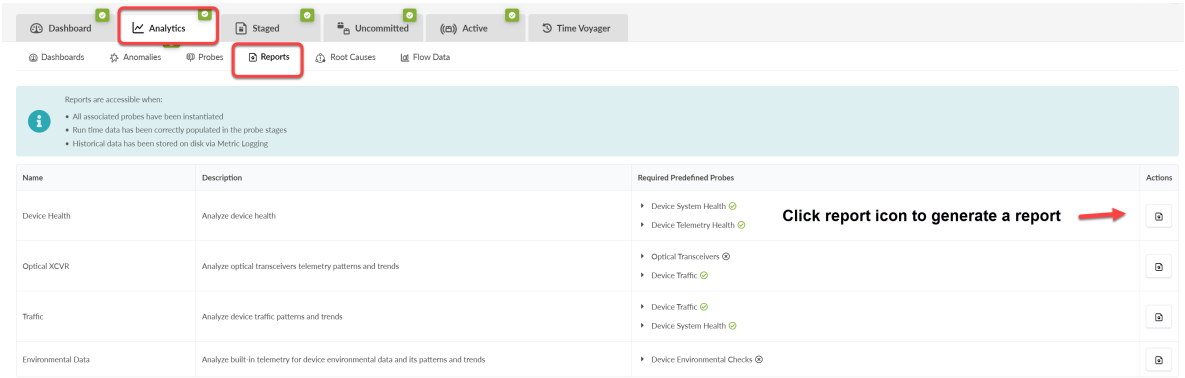
Generate an Analytics Report | 40

Generate an Analytics Report

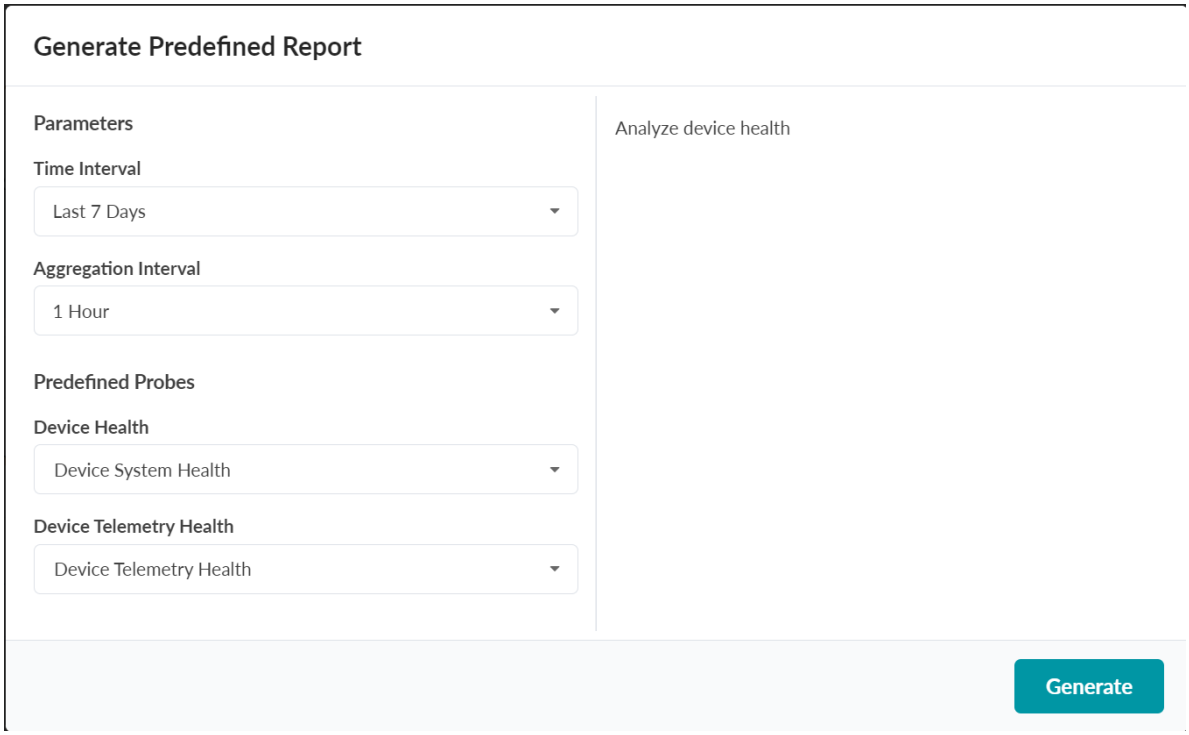
In the Apstra GUI, you can generate four types of analytics reports for your blueprint. These reports are based on historical data that are gathered by Apstra. Each report is dependent on data that are collected from the predefined probes. You must enable the required probes to generate a report.

To generate a predefined report:

1. From your blueprint, navigate to **Analytics > Reports**.
2. Click the **Generate Report** icon in the **Actions** panel for the report you want to generate.



3. Specify the time and aggregation interval parameters for your report.



- Time interval: Specify a predefined date range or custom range. The default time interval is 7 days.
- Aggregation interval: Specify the interval or frequency that data is returned. If no interval is defined, the data is returned in the default aggregation interval. The default is 1 hour.

> **NOTE**: By default, predefined probes are enabled for all reports.

4. Click **Generate**.

   A detailed report is generated and appears for your specified report.

RELATED DOCUMENTATION

Analytics Reports Introduction | 39

Environmental Data Analytics Report

IN THIS SECTION

This chapter provides an overview of the environmental data predefined report you can generate in the Apstra GUI. To learn how to generate the report, see "Generate an Analytics Report" on page 40.

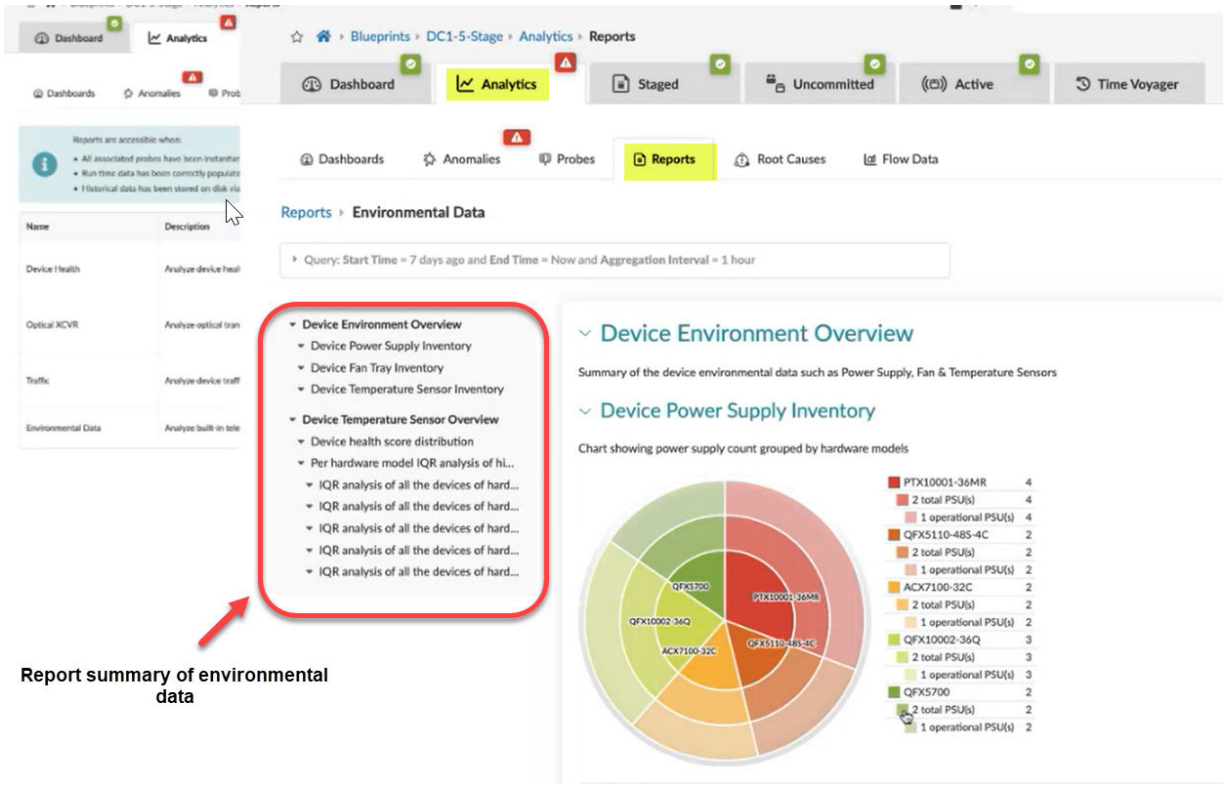
The environmental data predefined report shows environmental data for all Juniper devices including metrics for power supply, fans and temperature measurement. This report performs historical analytics on data from the "Device Environmental Checks Probe" on page 1544.

Device Environment Overview

IN THIS SECTION

When you generate an environmental data report, the report summary for the environmental data appears. The report summary, as shown in the figure below, provides an overview of the device environment (power supply, fan tray, and temperature inventory) and the device temperature sensor (device health score and device IQR analysis).

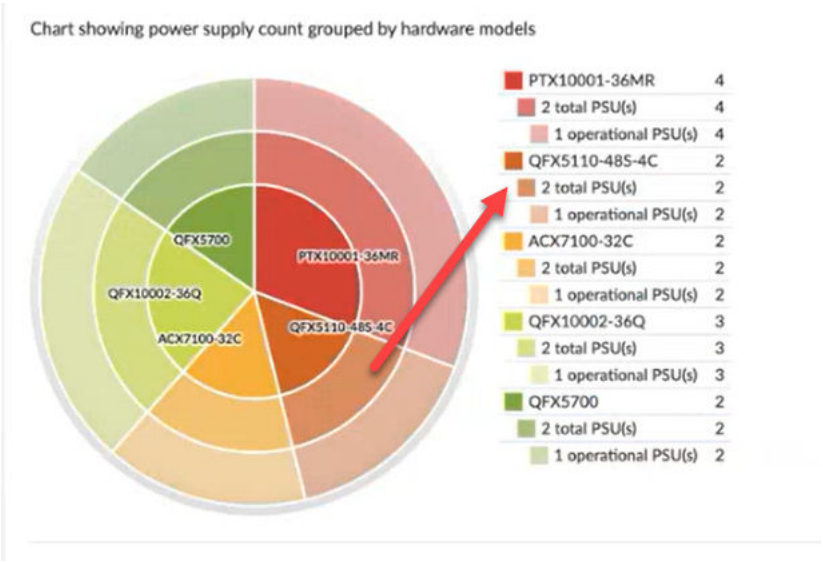
Figure 1: Environmental Report Summary



Device Power Supply Inventory

The "Device Power Supply Inventory" subsection shows the number of total and operational power supplies grouped by hardware models. Each color in the chart indicates a different platform. As shown in the chart below, the orange slice represents the QFX5110-48S-4C hardware model. In this example, the chart shows two QFX5100's switches, each containing two PSU's (power supply units), however only one PSU is operational.

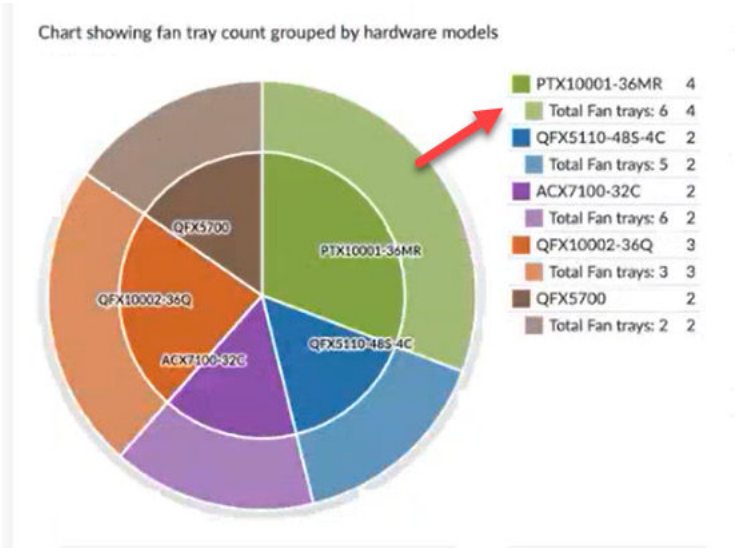
Figure 2: Device Power Supply Chart



Device Fan Tray Inventory

The "Device Fan Tray Inventory" subsection shows the number of fan trays for each hardware model. Each color in the chart indicates a different platform. As shown in the chart below, the green slice represents the PTX10001-36MR router. In this example, the chart shows four PTX10001 routers, each with six fan trays.

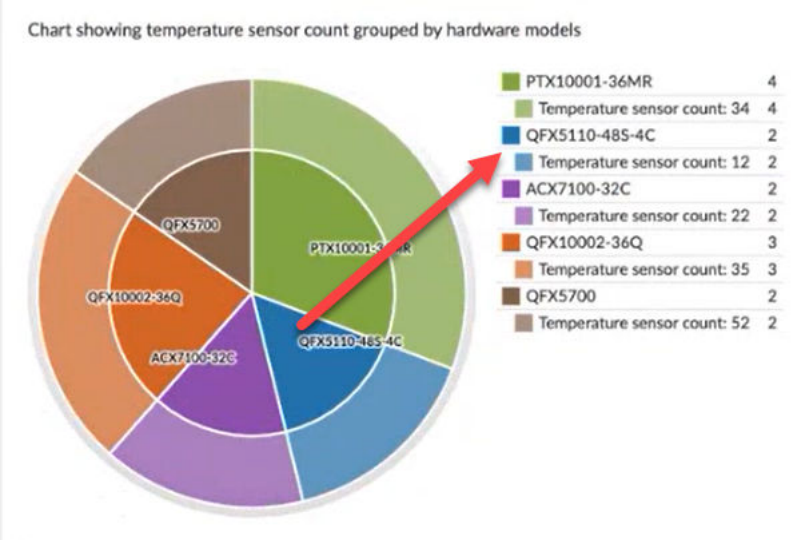
Figure 3: Device Fan Tray Chart



Device Temperature Sensor Inventory

The "Device Temperature Sensory Inventory" subsection shows the temperature sensor counts for each hardware model. As shown in the chart below, the blue slice represents the QFX5110-48S-4C hardware model. In this example, the chart shows two QFX5100's with a total of 12 temperature sensors per switch.

Figure 4: Device Temperature Sensor Chart



Device Temperature Sensor Overview

IN THIS SECTION

The Device Temperature Sensor analyzes the sensors for each hardware model. This overview includes the following sections:

Device Health Score Distribution

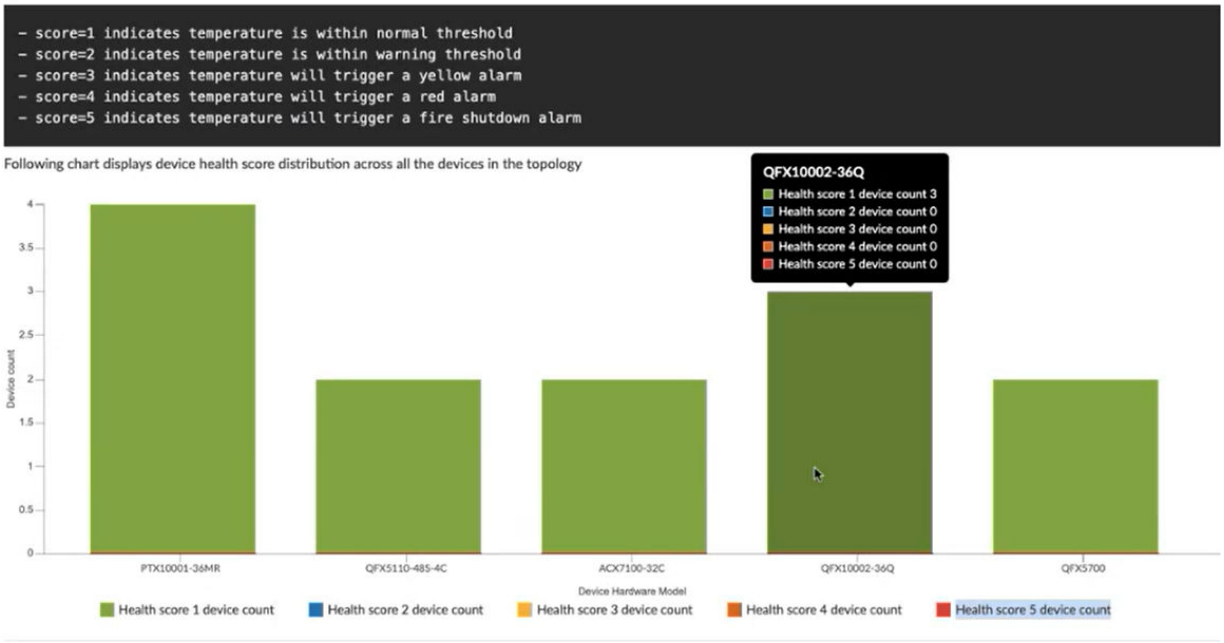
Every Juniper hardware model has pre-calculated temperature threshold values for all temperature sensors on the device. To calculate the device health score, the temperature measurement for each

temperature sensor is compared with this pre-calculated threshold value. A temperature health score 1 (best) to 5 (worst) is assigned for all devices based on that comparison.

For example, Figure 5 on page 46 shows the device health score that is distributed across all the devices in the topology. In this chart, all hardware models are shown as green. When you hover over each model, you can see the health score for each device. In this example, each device is assigned "score=1" indicating that the temperature is within the normal threshold.

NOTE: If a device is assigned a higher health score (score=3, 4, 5), the temperature sensor triggers an alarm or a shutdown for that device.

Figure 5: Device Health Score Distribution Chart



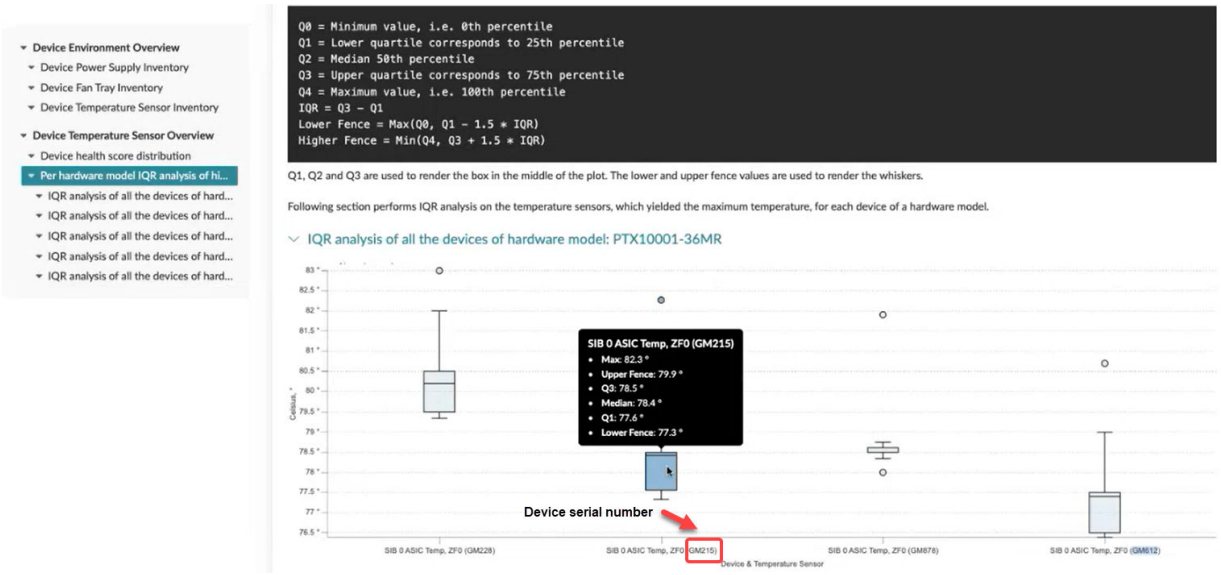
Per Hardware Model IQR Analysis of Highest Temperature Measured Over Time

The interquartile range (IQR) is a measure of statistical dispersion, which is the spread of data. An IQR analysis helps identify a normal range of values and how often extreme values are observed. The IQR range is the difference between the upper and lower quartile values in the data. The range is defined as the amount of spread in the middle 50 percent of the dataset.

When you run a report, IQR analysis is performed on the temperature sensors for each hardware model. The figure below shows an example of IQR analysis that was performed on the PTX10001 router. As

shown in the chart, all PTX1000's are identified by their serial number. When you hover over a PTX, you can view the SIB ASIC temperature values and percentages for that device.

Figure 6: Device IQR Analysis Chart



SEE ALSO

Generate an Analytics Report | 40

Probe: Device Environmental Checks | 1544

CHAPTER 5

Root Causes

IN THIS CHAPTER

Root Causes

IN THIS SECTION

Root Cause Overview

Root Cause Identification (RCI) is a technology integrated into Apstra software that automatically determines root causes of complex network issues. RCI leverages the Apstra datastore for realtime network status, and automatically correlates telemetry with each active blueprint intent. Root cause use cases include the following:

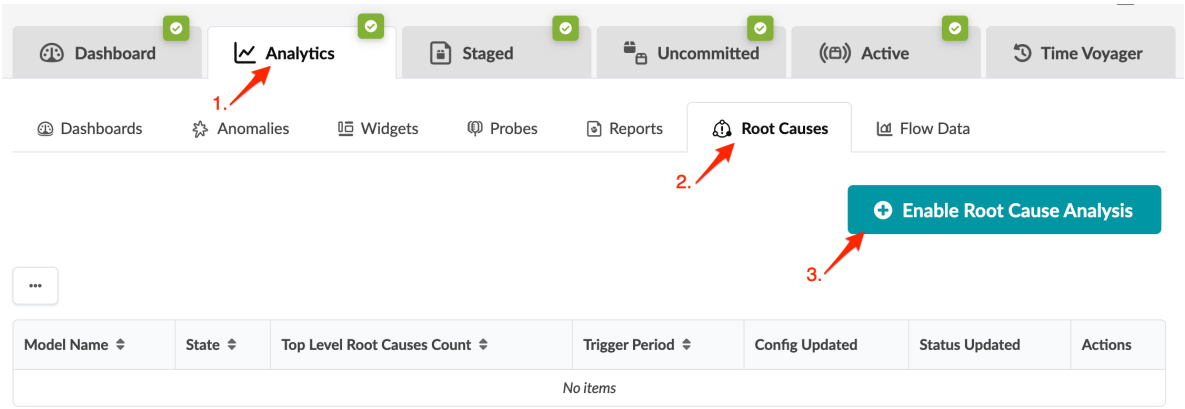
Root Cause	Description
Link broken	Symptoms: Both interfaces are operationally down, LLDP is missing on both sides, BGP peered across that link is operationally down.
Link miscabled	Symptoms: LLDP indicates wrong neighbors, BGP peered across that link is operationally down.

(Continued)

Root Cause	Description
Operator shut interface	Symptoms: Both interfaces on the link are operationally down; the interface in question is administratively down; LLDP missing on both sides, BGP peered across that link is operationally down.
Disconnection between 2 devices	Symptoms: Union of symptoms for link broken / link miscabled / operator shut interface for all constituent links between a spine and a leaf For instance, if there are 3 links between a spine and a leaf, then 2 could be miscabled and 1 is broken - this results in a disconnection between that spine and that leaf.

Enable Root Cause Analysis

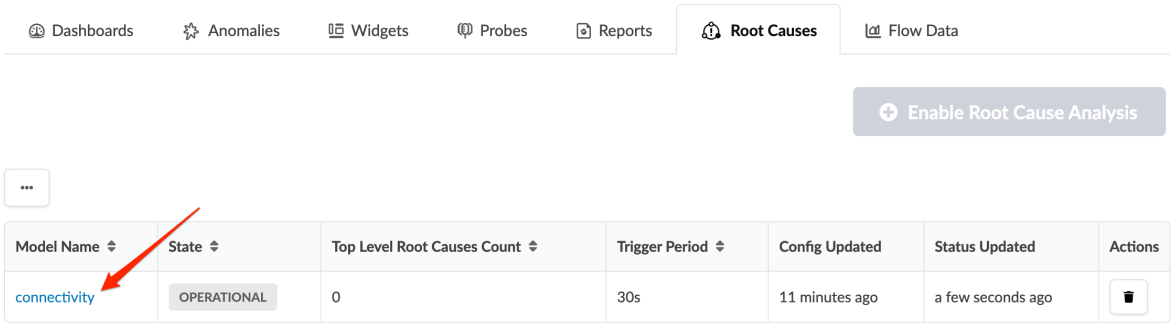
1. From the blueprint, navigate to **Analytics > Root Causes** and click **Enable Root Cause Analysis**.



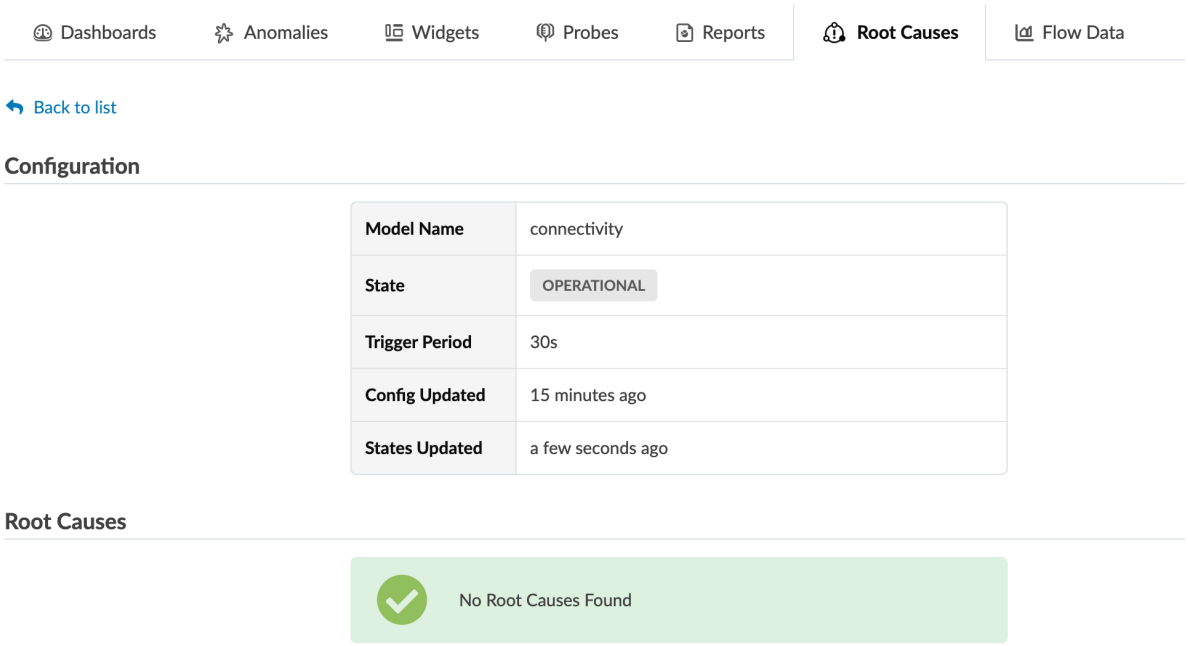
2. Enter a **Trigger Period** or leave the default, and click **Create** to enable root cause analysis and return to the table view.

View Root Cause Analysis

From the blueprint, navigate to **Analytics > Root Causes** and click the model name **connectivity** in the table.



Root cause analysis runs periodically and produces zero or more root causes. Any root causes that are found include a description, a timestamp of when it was detected and a list of symptoms.



4

PART

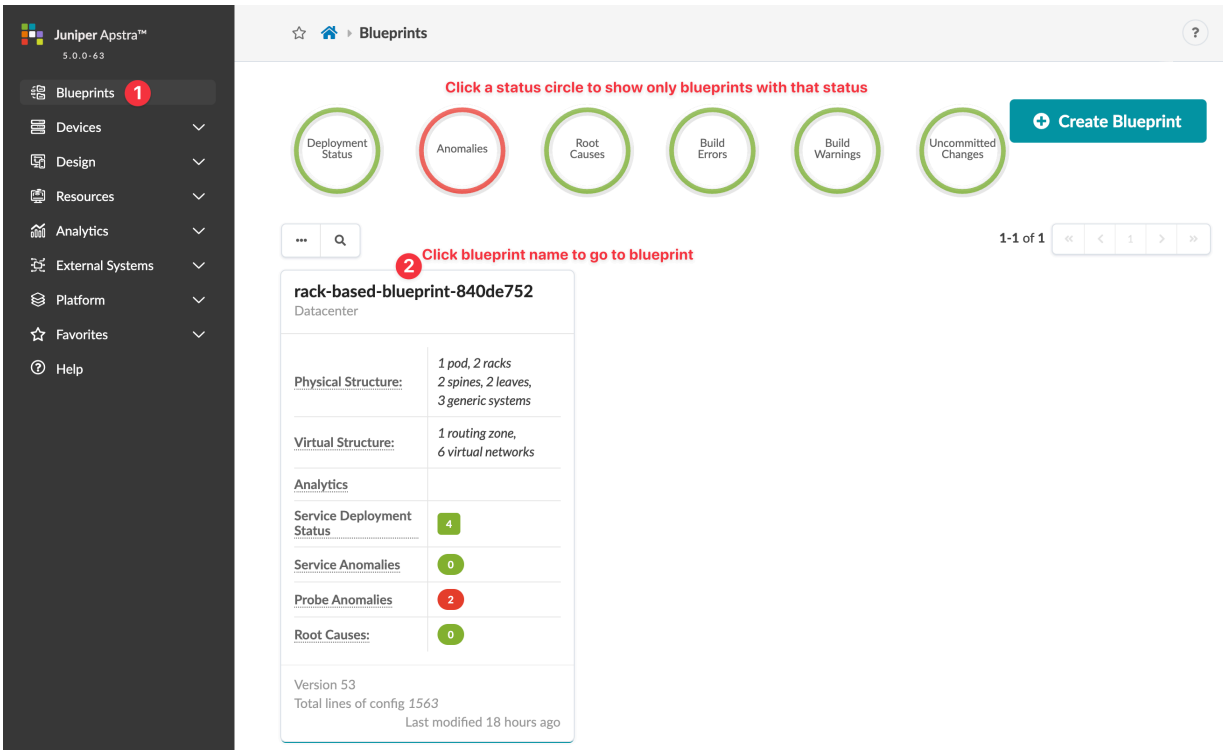
Staged Datacenter Blueprints

Blueprint Summaries and Dashboard

IN THIS SECTION

Blueprint Summaries

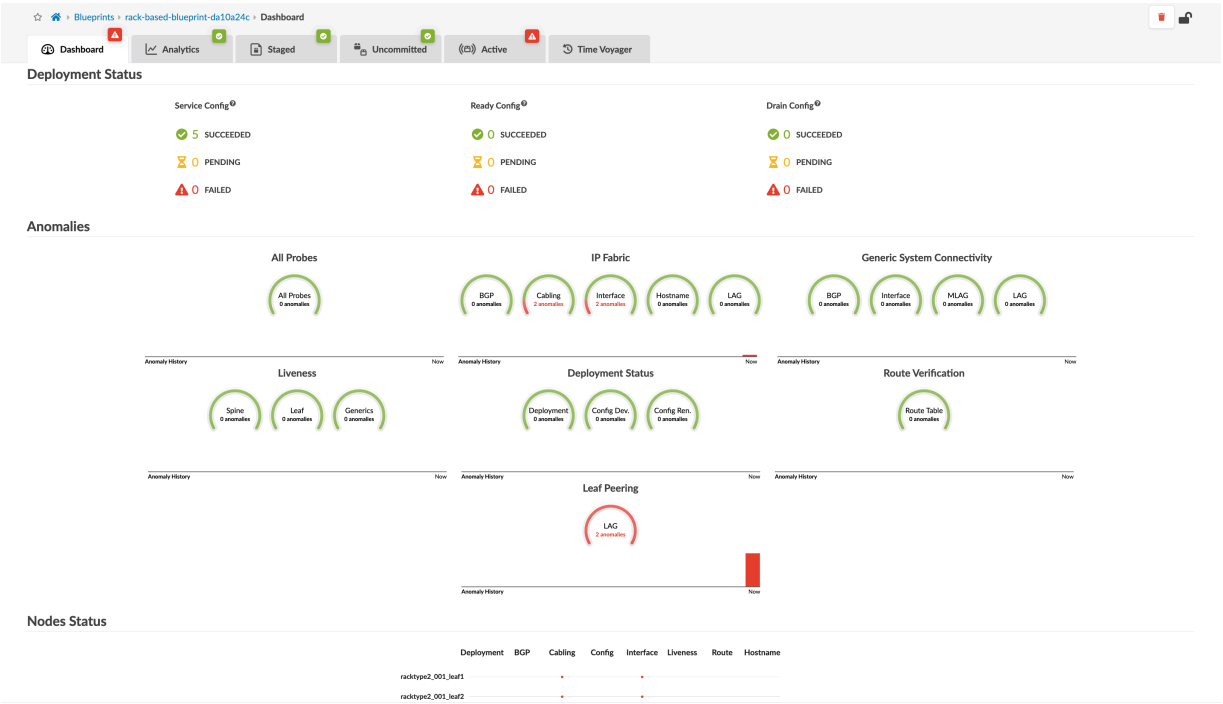
From the left navigation menu, click **Blueprints** to go to the blueprint summaries page. This page shows a summary of each individual blueprint. At the top of the page, indicators show various statuses across all blueprints (deployment status, anomalies, root causes, build errors and warnings, and uncommitted changes. This is useful when you have many blueprints in your Apstra instance. To quickly filter to show only blueprints that meet a certain criteria, click one of the indicators. If blueprints don't have any issues, the indicators are green. If there are any issues, the indicator is red. In the example below, clicking the red part in **Anomalies** results in displaying only the blueprints that include anomalies. (This Apstra instance has only one blueprint anyway, but you get the idea.)



Blueprint Dashboard

From the left navigation menu in the Apstra GUI, click **Blueprints**, then click the name of a blueprint to go to its dashboard.

The dashboard shows the overall health and status of a blueprint. Statuses are indicated by color: green for changes that succeeded, yellow for changes that are in progess, and red for changes that failed. The deployment status section includes statuses for service configuration, ready configuration, and drain configuration. The anomalies section includes statuses for all probes, IP fabric, generic system connectivity, liveness, deployment status, route verification, leaf peering, and more. The nodes status section includes statuses for deployment, BGP, cabling, config, interface, liveness, route, and hostname. You can see in the example below, we have some issues with the IP fabric and leaf peering. You can click the red indicators for details.



You can display analytics dashboards on the blueprint dashboard to have additional network information on one screen. To add them, navigate to **Analytics > Dashboards** and turn ON the analytics dashboards' default toggle.

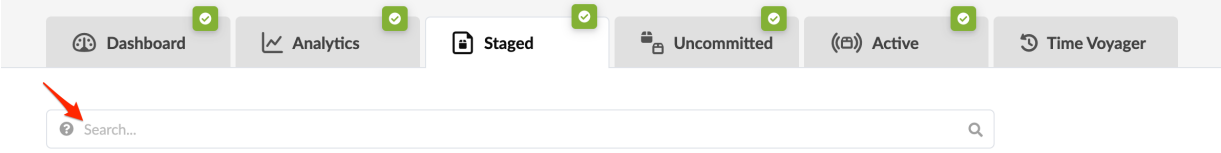
Blueprint-Wide Search

SUMMARY

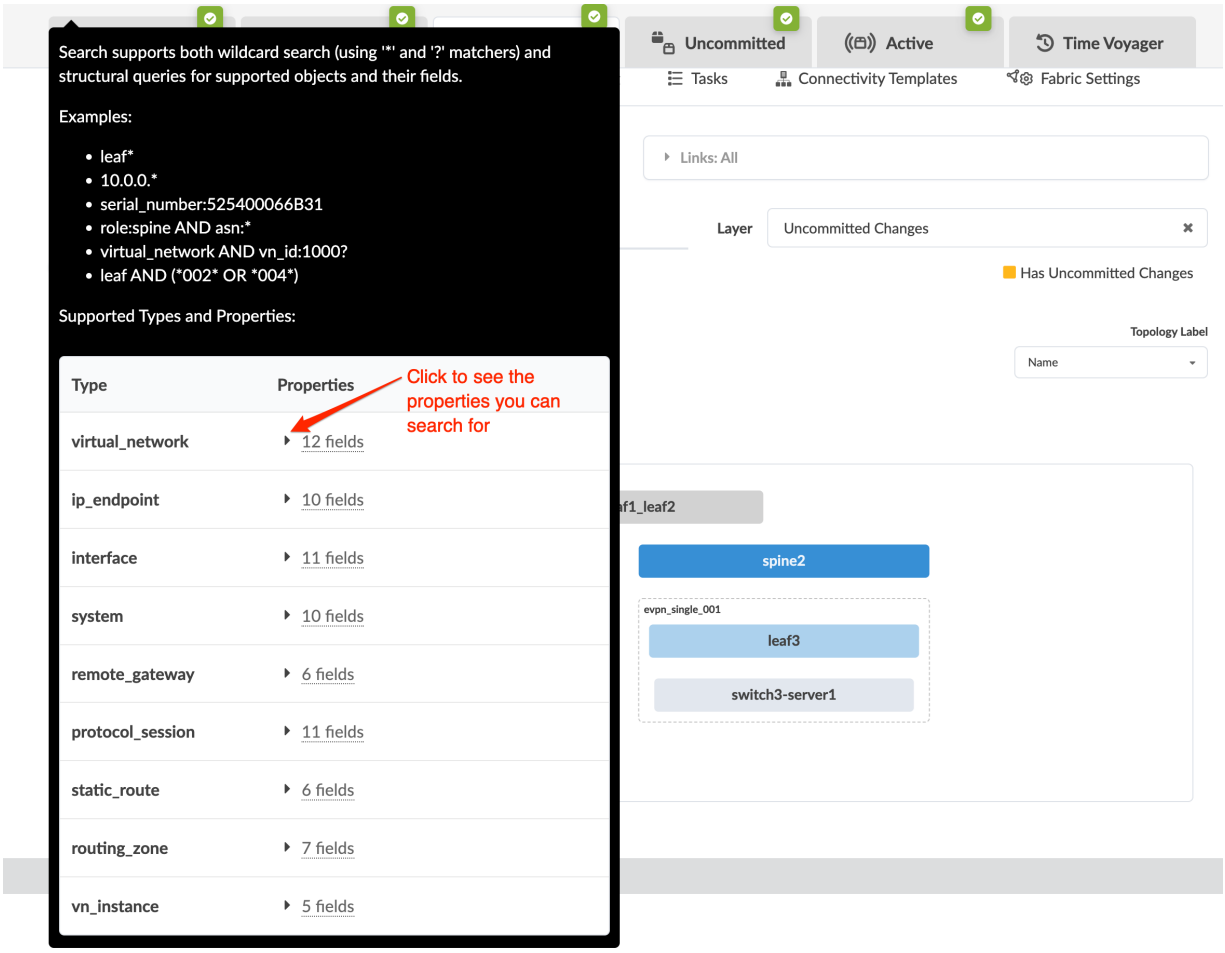
You can search the entire blueprint from the **Staged** (and **Active**) tabs.

IN THIS SECTION

To search the staged blueprint, enter your search criteria in the **Search** field at the top of the **Staged** tab.



For assistance with search, you can click the search field to see a tooltip describing the ways you can search.

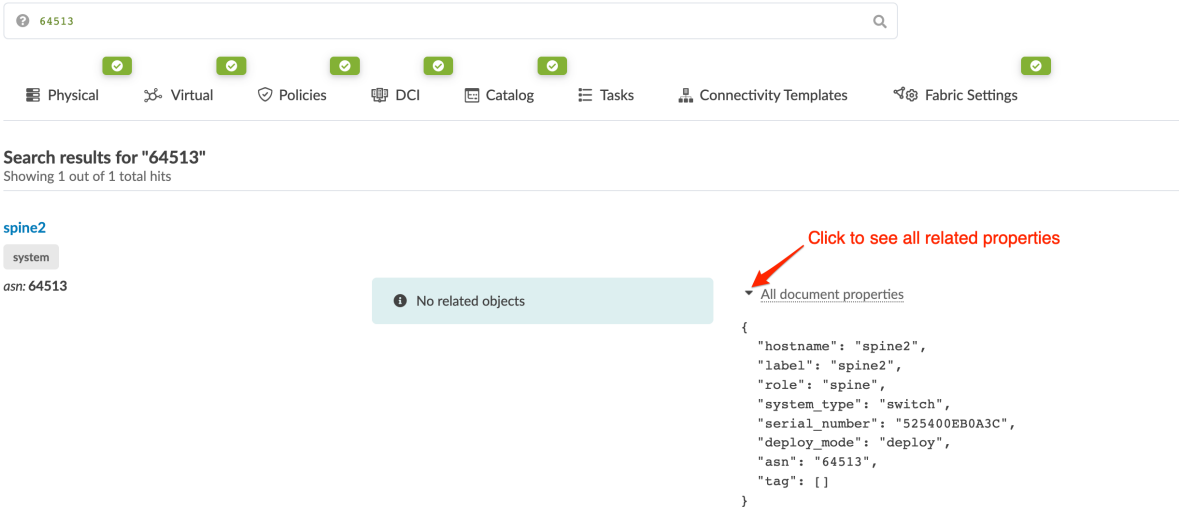


Exact Match

To find an exact match, enter the exact value for the object. For example, you can enter `64513` to find that ASN. The results in our example show that it's assigned to `spine2`.

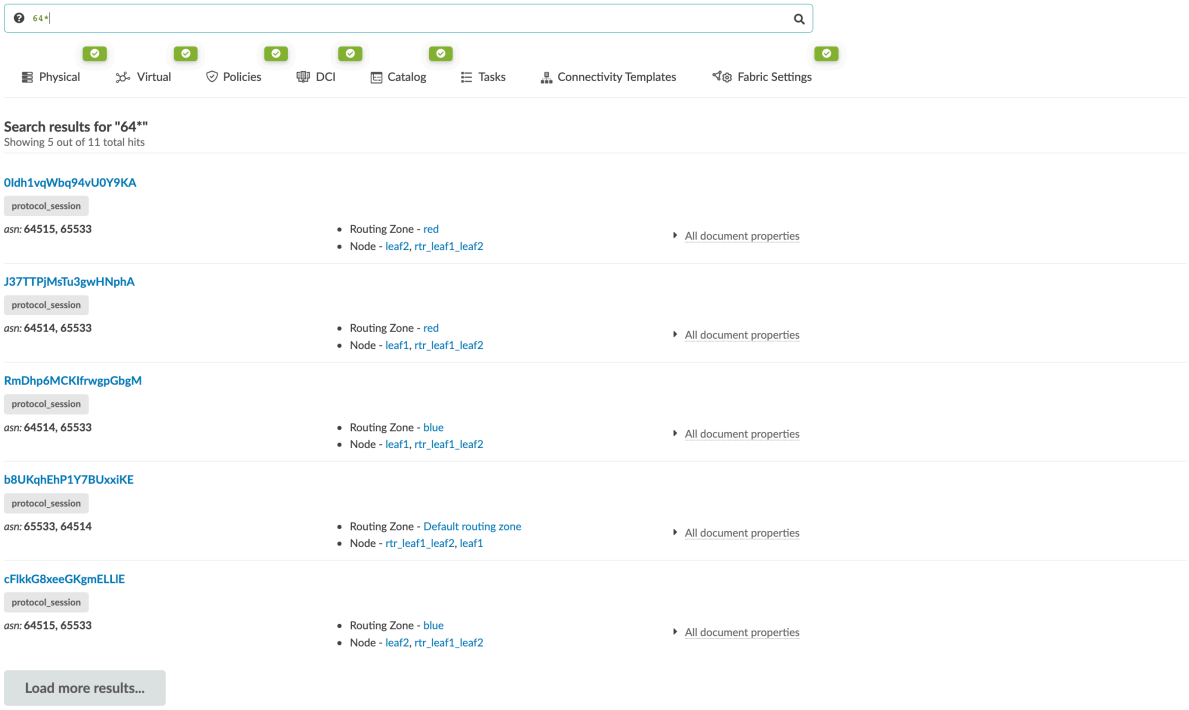
Additional metadata is returned that tells you what else is associated with the object. Click **All document properties** to see this information.

From your results you can click an object name (in blue) to go to its details.



Wildcards

You can search using wildcards. Let's say you want to search for ASNs that begin with `64`. By adding the wildcard character `*`, you get all objects that begin with 64. (Enter `64*`.) Five results are loaded by default. To see additional results, click **Load more results**. If there are no more results, **No more results** appears at the bottom.

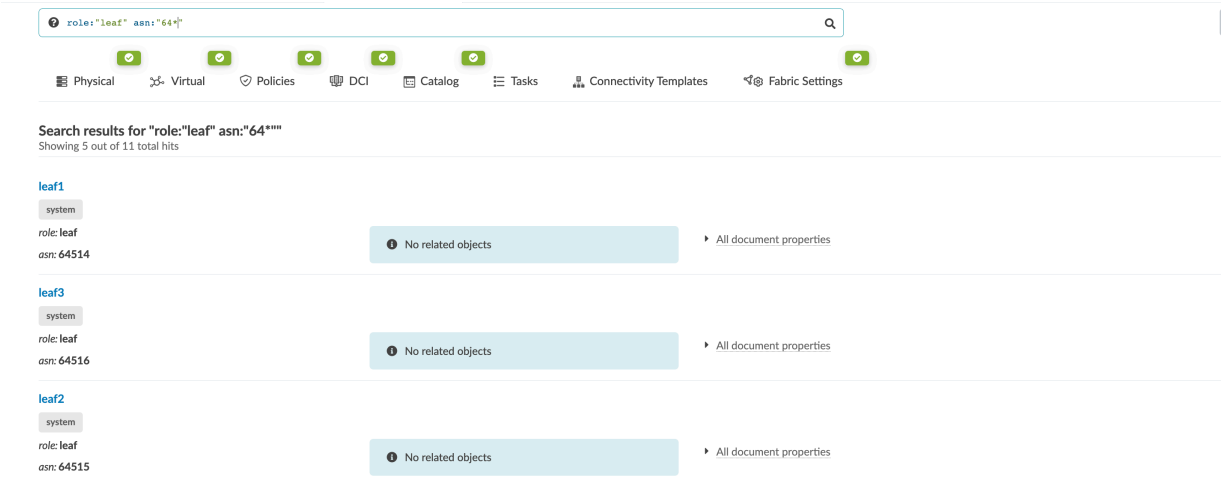


Field References

You can include a reference to a field in your search to receive more relevant results. With the ASN example, if you search `64*`, there may be other entities besides ASNs that begin with `64`. If you know you're looking for an ASN, you can enter a search query, such as `asn:"64*"`. As you begin typing results auto-fill to help you with the query. You can press the tab key to autocomplete.

Composite Queries

You can combine searches into one query. Returning to the ASN example, say you want to find ASNs beginning with 64 and that also have the `leaf` role. You can enter the search query, `role:"leaf" asn:"64*"`.



CHAPTER 6

Physical

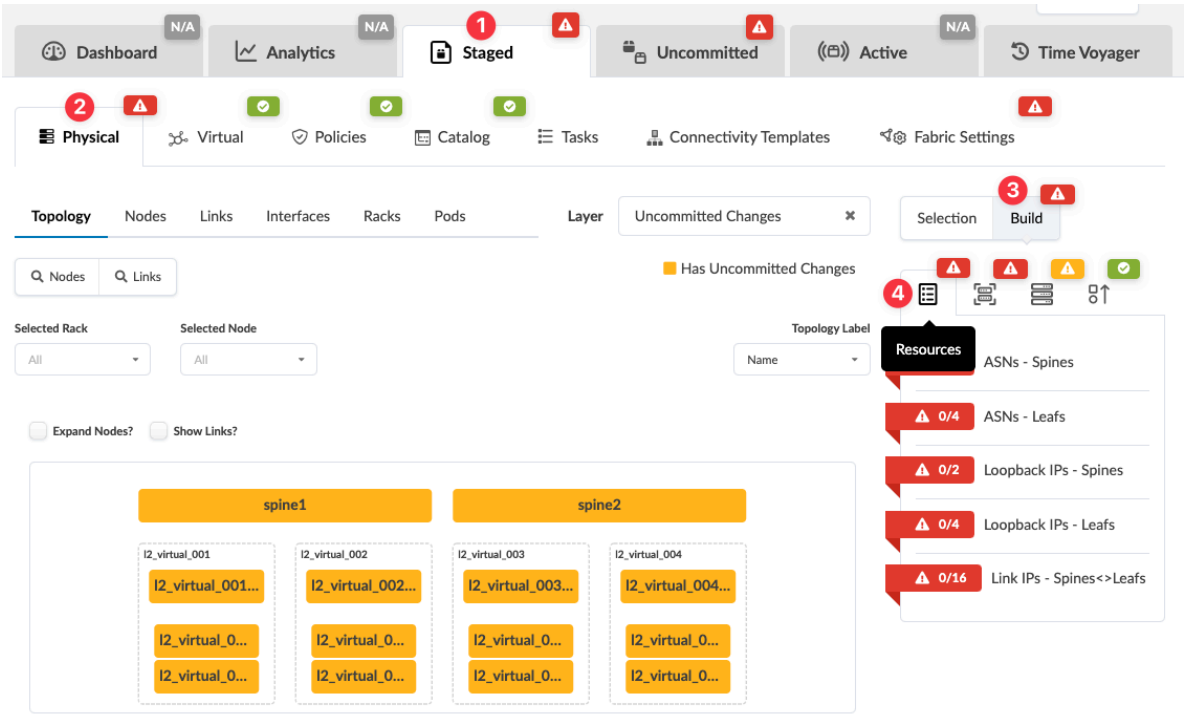
IN THIS CHAPTER

Build

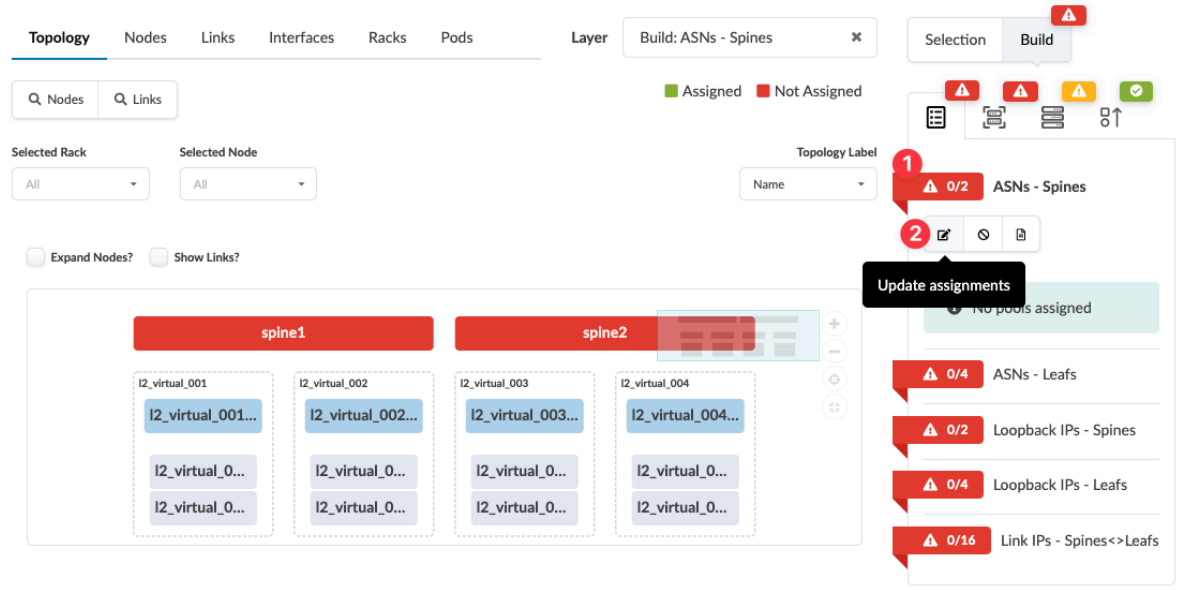
IN THIS SECTION

Assign ASNs and IP Addresses (Datacenter)

1. From the "blueprint" on page 53, navigate to **Staged > Physical > Build > Resources**. (The **Build** panel is on the right. **Resources** is the leftmost tab.)

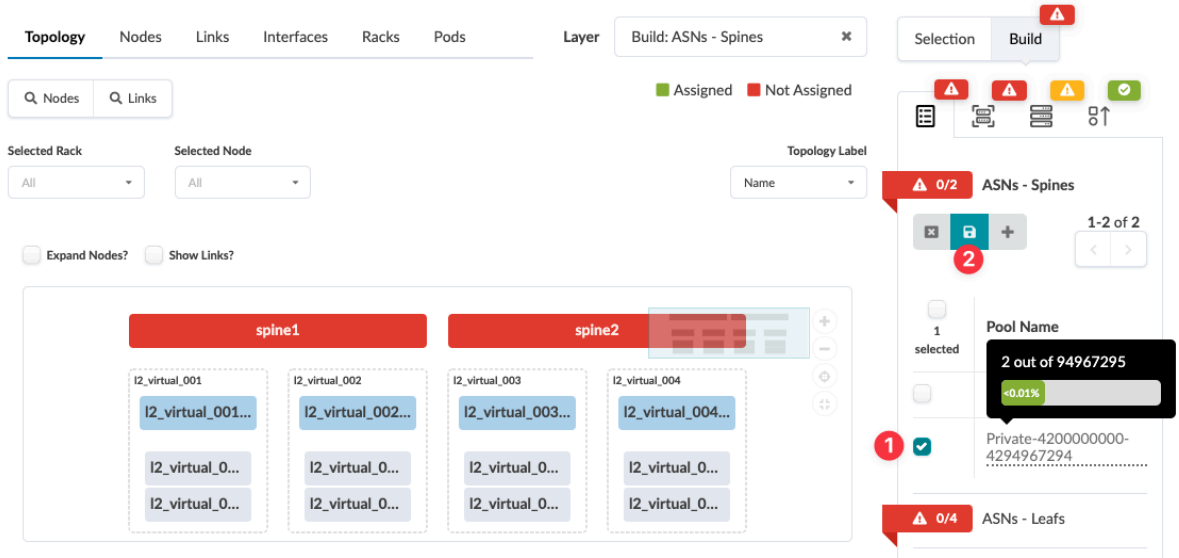


2. Red status indicators mean that resources are required. The first number in the indicator is the number of resources that have been assigned and the second number is the number of resources that are required. (For example, in the **ASNs - Spines** resource group, zero out of two resources have been assigned.) Click a red status indicator to see that no pools are assigned, then click the **Update assignments** button.



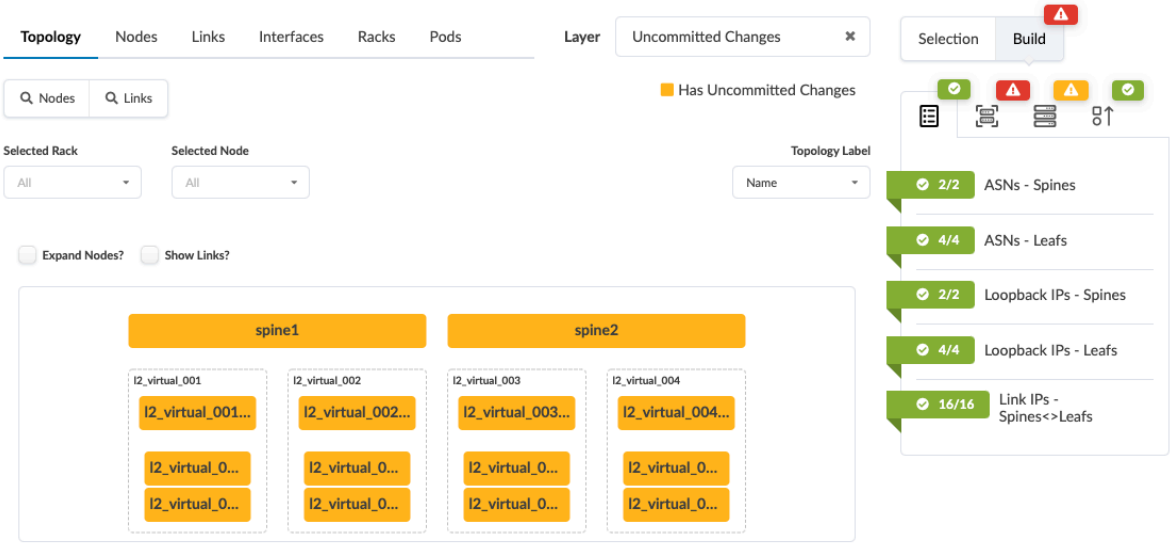
Available pools are listed. Theses are the same resource pools that you'll find in the global **Resources** catalog in the left navigation menu. (To see how much of the pool has already been assigned, hover over the resource pool name.)

3. Select the check box for the pool to use, then click the **Save** button.

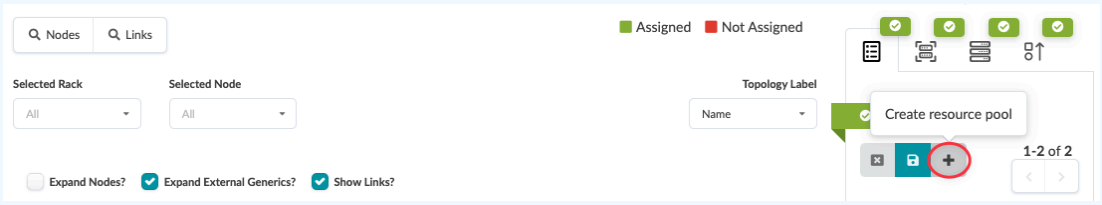


The required number of resources are automatically assigned to the resource group. When the red status indicator turns green, the resource assignment has been successfully staged.

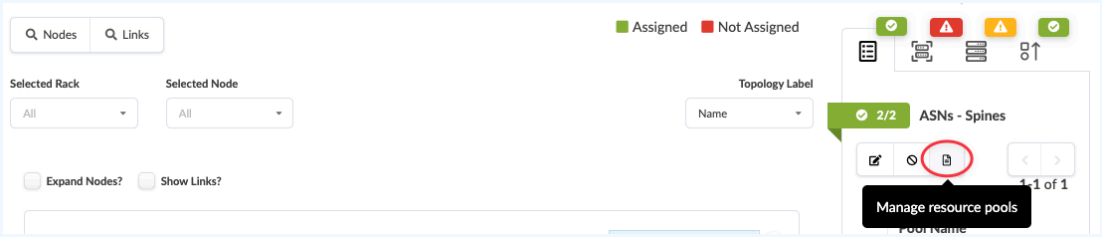
4. Continue assigning resources in the same manner until you've assigned all required resources.



NOTE: If not enough resources are available, you can create a resource pool directly from the blueprint instead of navigating to the **Resources** catalog (new in Apstra version 5.0.0). After clicking the **Update assignments** button, click the **Create resource pool** button (the plus sign), enter the details for the pool, then click **Create**. You can then select the check box for the new pool to use for assigning resources.



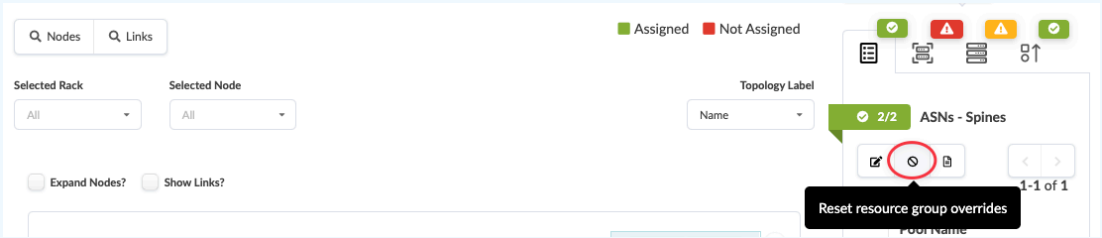
If you want to add, update, or delete resource pools you can go directly to the **Resources** catalog from the blueprint with one click. Before clicking the **Update assignments** button, click the **Manage resource pools** button. You're taken directly to resource pools. When you're ready to return to the blueprint, click the back button.



When you unassign resources (by removing a device or changing a resource), those unused resources remain reserved. Usually, you want to keep them reserved in case you need them again. Some examples are:

- Time Voyager Rollbacks that involve the use of resources - For example, if you delete a rack then subsequently roll back to a blueprint revision where the rack was not deleted, you'd need to use the same resources.
- Revert Operations - When you revert staged changes, the originally assigned resources appear in the table view to indicate that they've been retained, but the build section shows that no resources are assigned. Situations like this can (but don't always) result in build errors.
- Using the "Update Stated Cabling Map from LLDP" on page 200 feature.

To prevent potential issues, it's best to keep unused resources reserved. In the rare instances, when you need to release unused but reserved resources, use the **Reset resource group overrides** button. Those resources will then be available for assigning elsewhere.

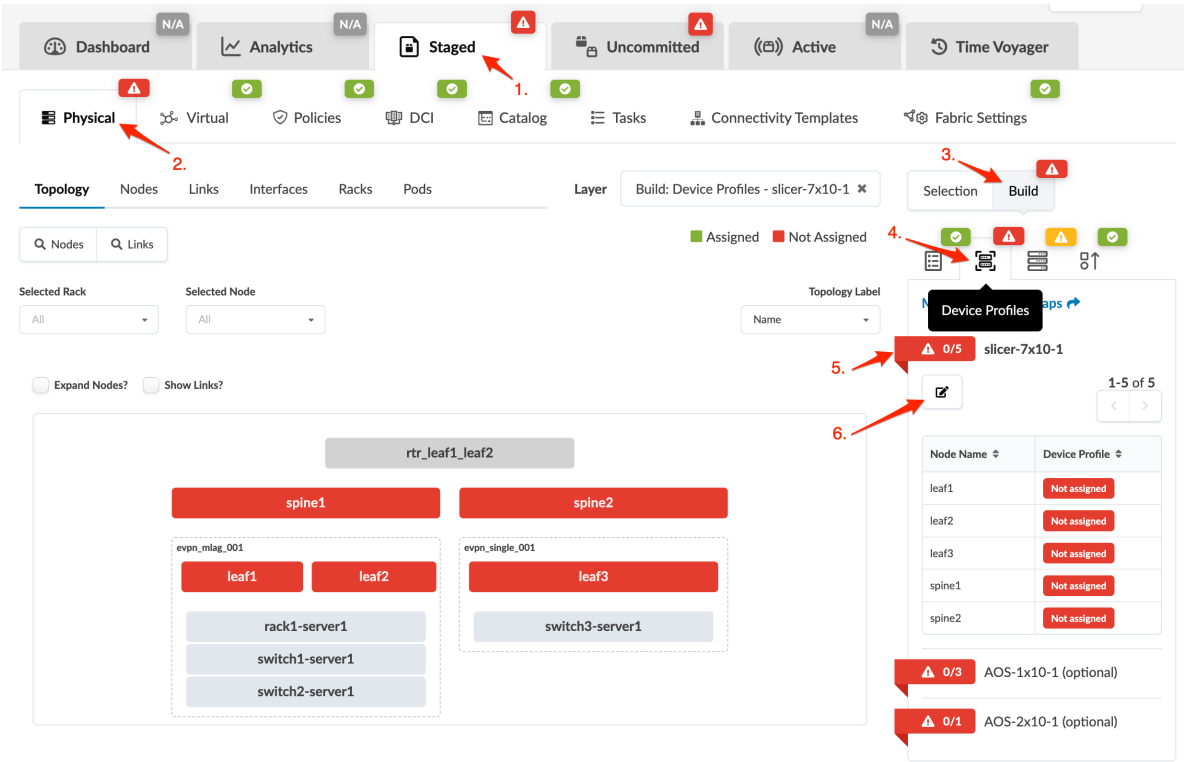


Next Steps: "Assign Interface Maps (Device Profiles)" on page 62.

Assign Interface Maps (Device Profiles)

When you "created your blueprint" on page 8 you selected a "template" on page 889. That template, among other things, defines the capabilities of your devices (via "logical devices" on page 845). It doesn't specify vendor hardware. Now that you're building your network in the blueprint, it's time to map those capabilities to actual device models ("device profiles" on page 783). You do that by assigning "interface maps" on page 853 to your "managed devices" on page 647. Remember, interface maps associate the interfaces between logical devices and device profiles.

1. From the blueprint, navigate to **Staged > Physical > Build > Device Profiles**, click one of the red status indicators, then click the **Change interface maps assignments** button (looks like an edit button).

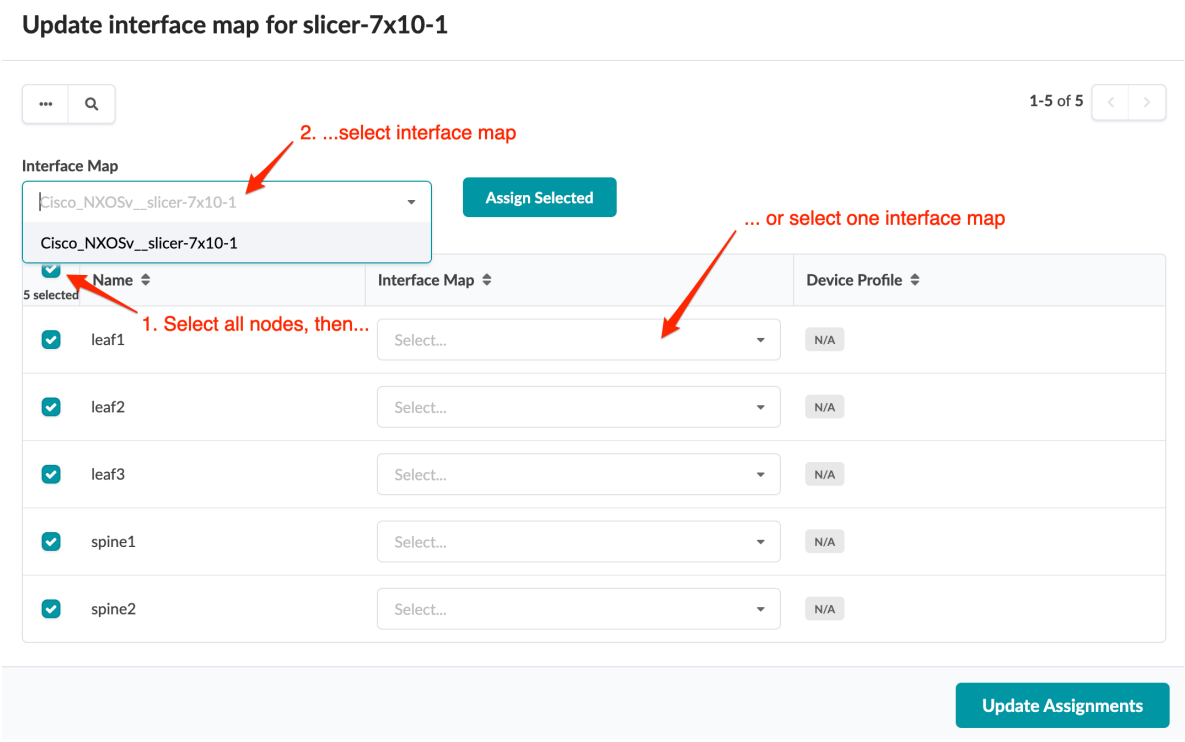


   The **Update Interface Map** dialog opens.

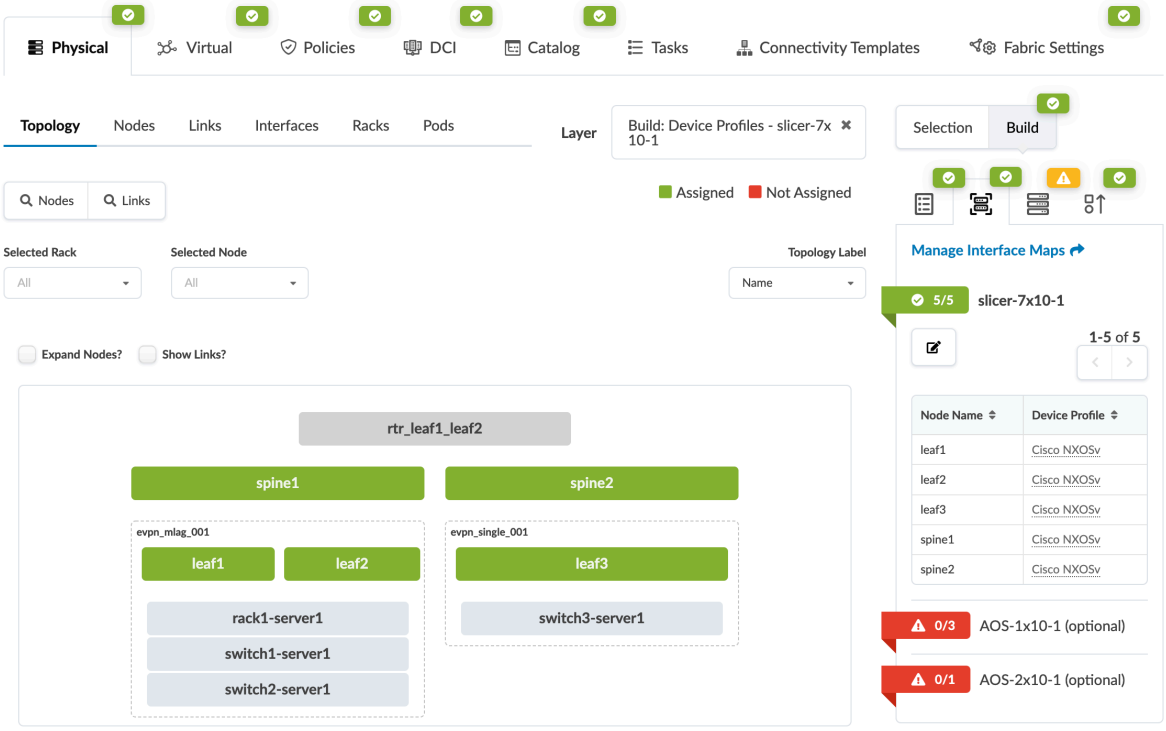
2. You can batch assign interface maps, or you can assign them singly.

   If you're using the same device model for all of your devices, you can quickly assign them all at once by selecting the check box in the upper-left corner of the table and selecting the interface map from the **Interface Map** drop-down list that appears above the table. The list includes interface maps from the blueprint catalog (Staged > Catalog > Interface Maps). (If you don't see the interface map that you need, you can "import" on page 439 it, then come back here and it will be included in the list.) Click **Assign Selected** to populate the table.

   If you're using various device models, you can select an interface map for a single device from the individual **Interface Map** drop-down list. Repeat for the other devices, as applicable.



3. Click **Update Assignments**. When the red status indicator turns green, the interface map assignments have been successfully staged.



NOTE: If you've already deployed your network and you want to change an interface map, see "Change Assigned Interface Map" on page 110. There are some important details about devices being automatically unassigned and undeployed.

Next step: "Assign System IDs" on page 64.

Assign Device IDs (Datacenter)

IN THIS SECTION

Device Assignment Overview

Before devices can be assigned to a blueprint, they must have interface maps assigned to them (from the Device Profiles tab). When a device is assigned to a blueprint, it performs discovery configuration. During this phase all interfaces are changed to L3-only mode allowing interfaces to be *up*. There is no BGP configuration, no routing expectations, nothing that can influence the network. A device in *discovery* mode is benign; it does not participate in the datacenter fabric, and it does not forward any packets through it. You can then perform critical validations of network health including viewing statistics for cabling, LLDP, transceivers and more. Any issues, such as miscabling or physical link errors, cause a telemetry alarm. You can address and correct the anomalies *before* deploying the device.

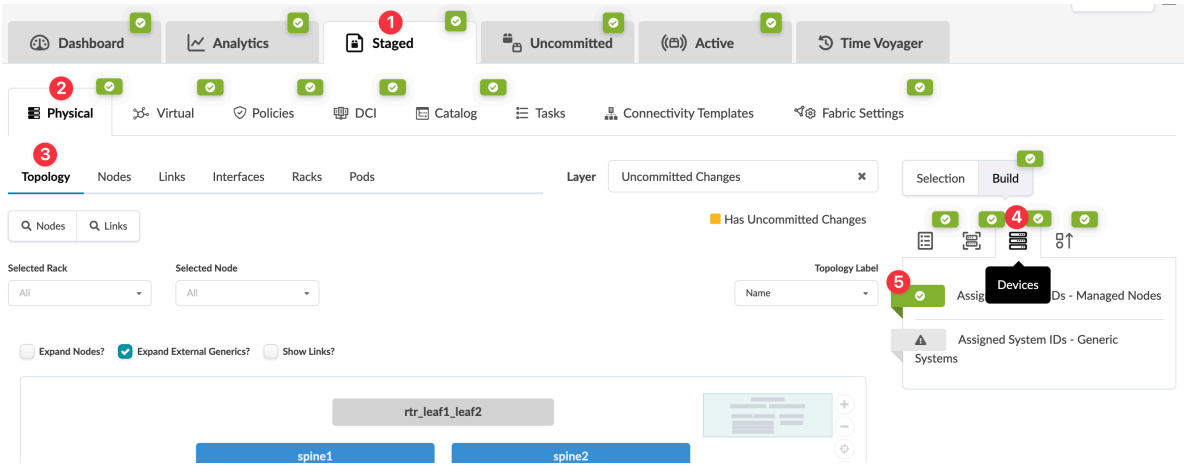
It's common to have a committed blueprint without any deployed devices. You can deploy devices as required, in batches, one by one, or all in one go. If you want to assign devices without deploying them, set the deploy mode to **Ready**, which puts devices in the **In Service Ready** state. This configuration is called **Ready Config** (previously known as Discovery 2 Config).

NOTE: When resetting system IDs (serial number) Discovery 1 configuration is re-applied. Before physically uninstalling the agent, it is good practice to fully erase the device configuration and uninstall the device agent.

Assign Device(s) (from Devices Build Panel)

> **NOTE**: You can also use apstra-cli to bulk-assign system IDs to devices either with a CSV text file or the `blueprint set-serial-numbers` command.

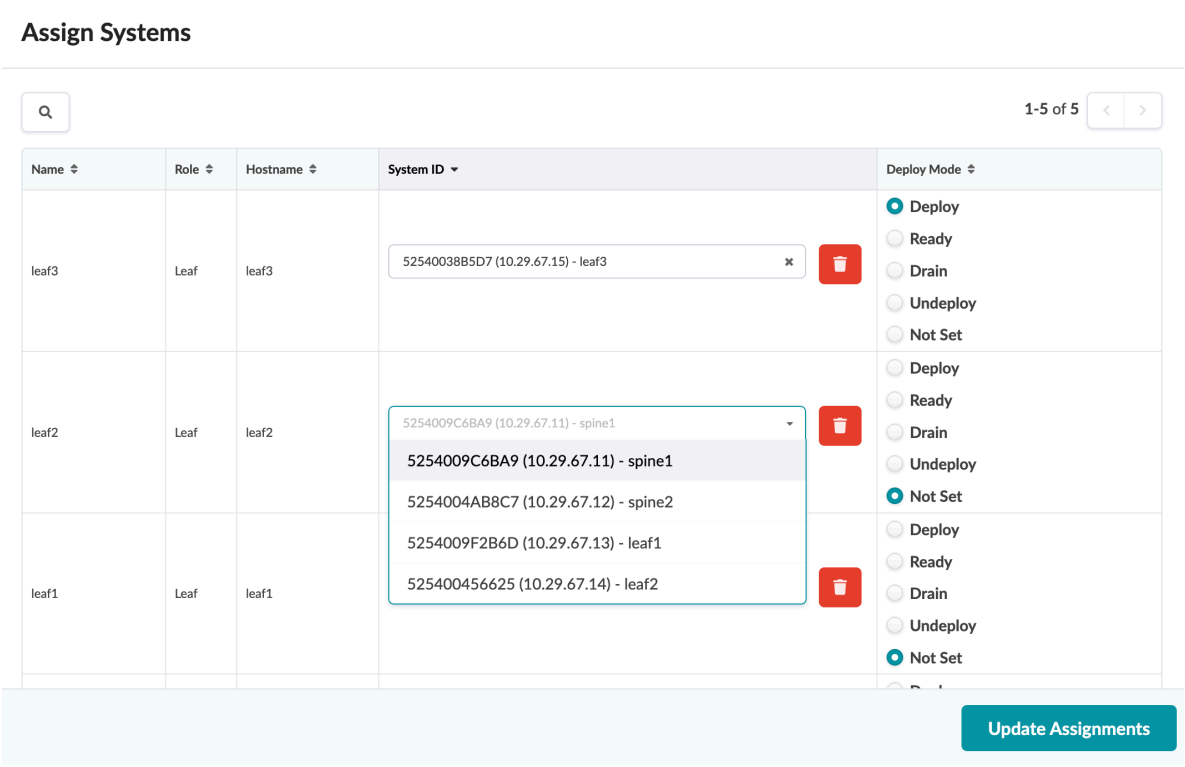
1. From the blueprint, navigate to **Staged > Physical > Topology** and click the **Devices** tab in the **Build** panel (on the right), then click the status indicator for **Assigned System IDs**.



   The **Assigned System IDs** list appears in the right panel.

2. Click the **Change System IDs assignments** button (the edit icon below Assigned System IDs) The **Assign Systems** dialog opens.

3. For each node, select system IDs from the drop-down list. (If you don't see an expected serial number (system ID), you may still need to acknowledge the device (Devices > Managed Devices).)



4. Change deploy modes, as appropriate. For deploy mode details, see "Device Configuration Lifecycle" on page 645.
5. Click **Update Assignments** to stage the changes. Before the task is completed you can click **Active Tasks** at the bottom of the screen to see its progress.
6. Commit changes to the blueprint to deploy device(s) into the active fabric. Device state changes to **In Service Active** and the configuration is called **Service Config**.

   As soon as you deploy a device, anomalies may appear on the dashboard. When telemetry data is verified against Intent, anomalies resolve themselves. This can take a fair amount of time in some cases, especially for BGP sessions and advertising routes.

   Deploying devices can have different implications depending on the device vendor. Juniper Junos devices, for example, have the following characteristics with regards to raising anomalies:

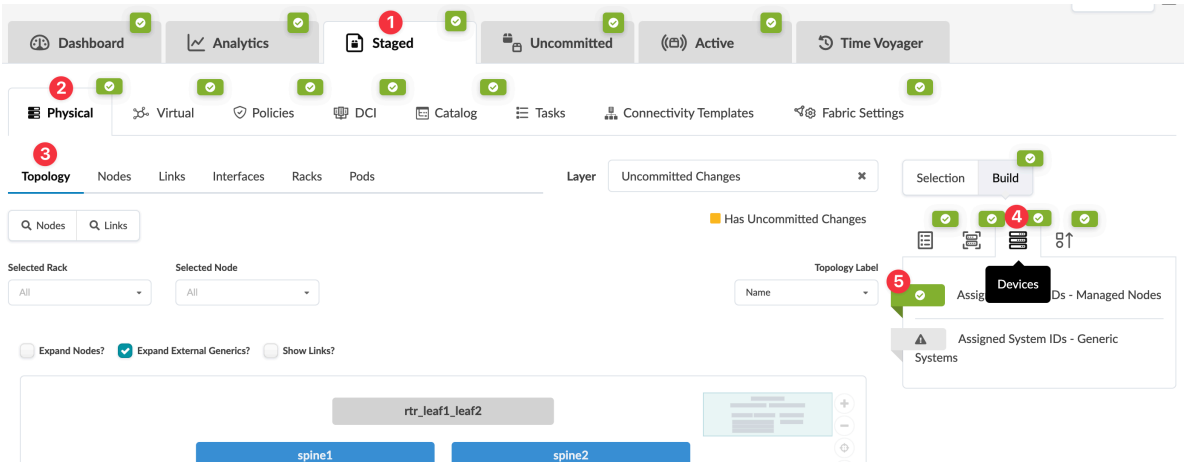
   - `show interface` commands don't list interfaces on ports that do not have a transceiver plugged in. This means *Interface Down* anomalies can't be raised for these interfaces. Such interfaces can be recognized using the `show virtual-chasses vc-port`, and have a status of 'Absent'.
   - If a virtual network endpoint is configured on a leaf interface, Apstra expects an EVPN type 3 route for that interface. If this interface is down, Junos does not advertise the RT-3, resulting in a "Missing Route" anomaly. If this anomaly is undesirable, we recommend that you remove the interface from the virtual network until the interface is up.

   After deploying devices a new running config is collected, called the **Golden Config**, which serves as Intent. Running configuration is continuously collected and compared against this Golden config.

When a deployment fails, Golden Config is unset. Protocol related anomalies like BGP or LLDP are only raised if devices at both ends are deployed.

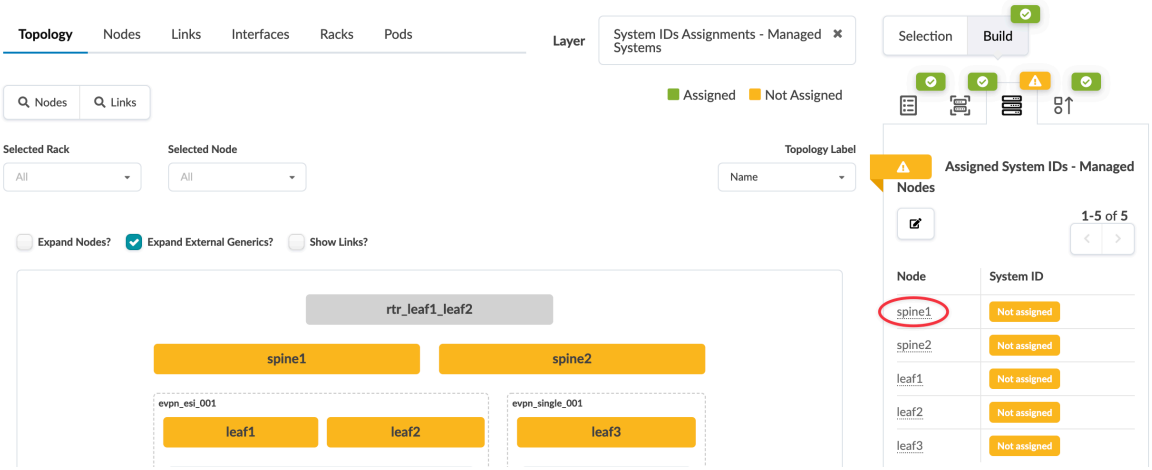
Assign One Device (from Devices Build Panel)

1. From the blueprint, navigate to **Staged > Physical > Topology** and click the **Devices** tab in the **Build** panel (on the right), then click the status indicator for **Assigned System IDs**.



The **Assigned System IDs** list appears in the right panel.

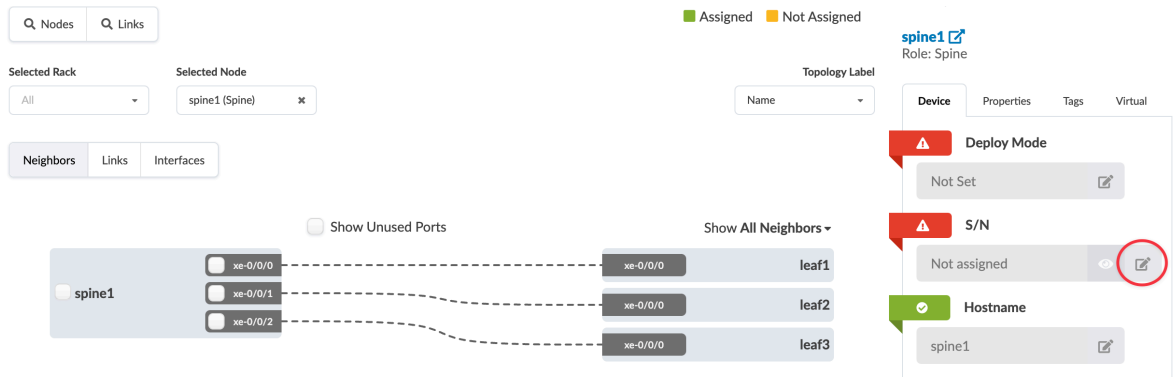
2. Click the name of the node that you want to assign.



NOTE: You can also select a node name in the **Selected Nodes** drop-down list (left-middle) to go to these device details.

The **Device** tab for the selected node opens in the right panel.

3. In the right panel, in the **Device** tab, click the **Edit** button for **S/N**.

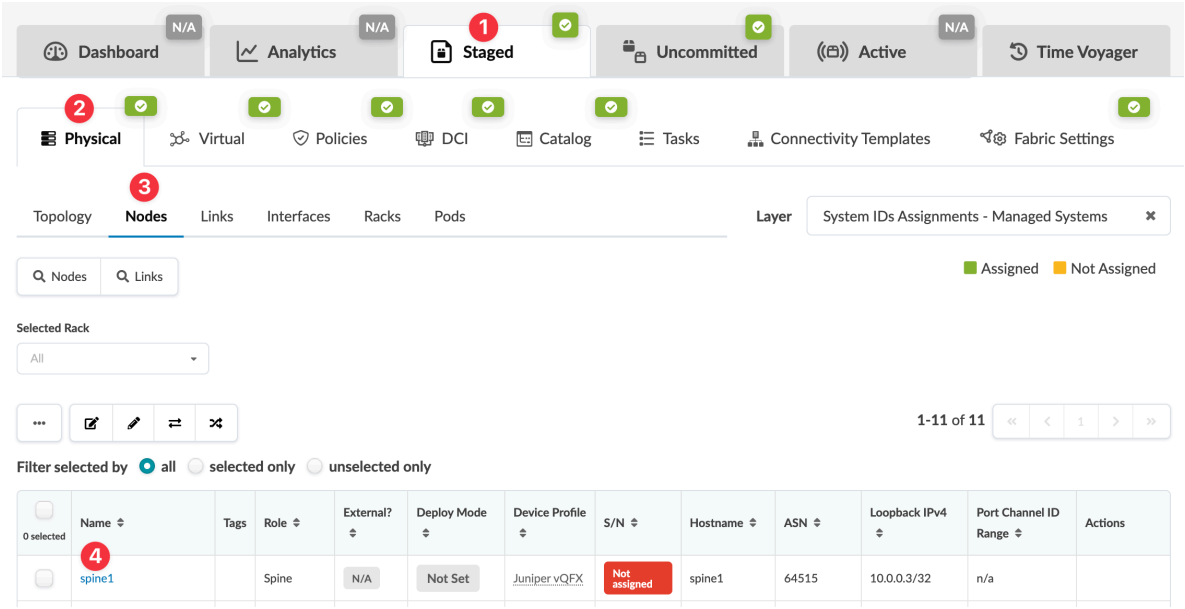


The **S/N** field becomes editable.

4. Select the system ID from the drop-down list, and click the **Save** button to stage the change. (If you don't see the expected serial number (system ID), you may still need to acknowledge the device (Devices > Managed Devices). (To remove an existing S/N instead of assigning one, click the **Edit** button for **S/N**, then click the **Save** button to stage the change.)

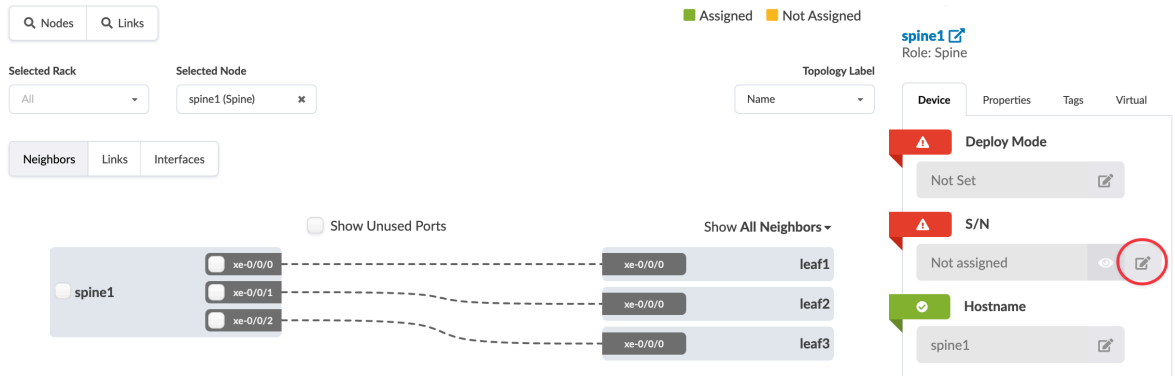
Assign One System ID (from Selection Panel)

1. From the blueprint, navigate to **Staged > Physical > Nodes** and select the node name to assign (not the check box).



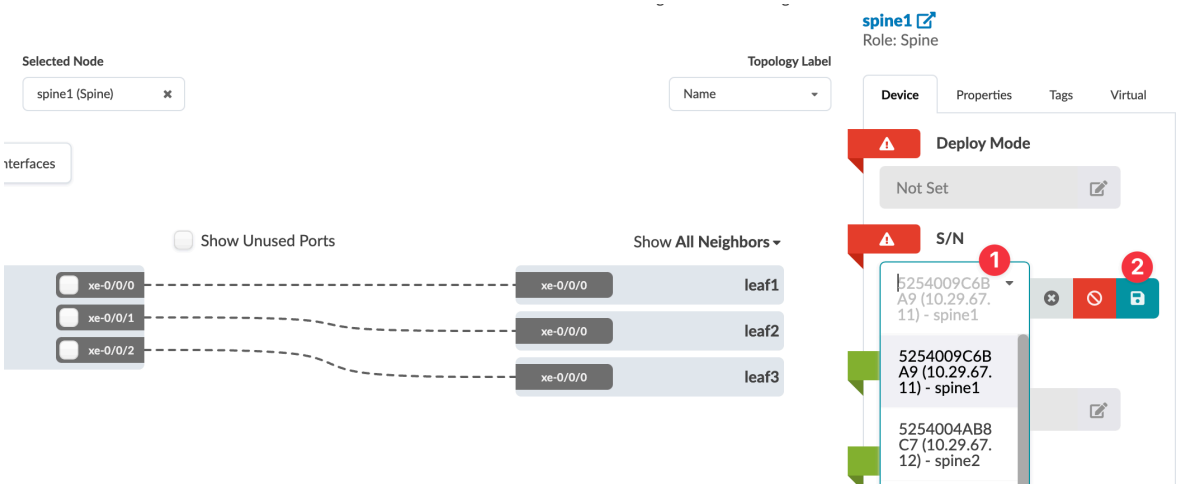
The **Device** tab for the selected node opens in the right panel.

2. In the right panel, in the **Device** tab, click the **Edit** button for **S/N**.



The **S/N** field becomes editable.

3. Select the system ID from the drop-down list, and click the **Save** button to stage the change. (If you don't see the expected serial number (system ID), you may still need to acknowledge the device (Devices > Managed Devices). (To remove an existing S/N instead of assigning one, click the **Edit** button for **S/N**, then click the **Save** button to stage the change.)



SEE ALSO

Commit / Revert Changes to Blueprint | 599

Manage Configlets

Configlets are vendor-specific. Apstra software automatically ensures that configlets of a specific vendor are not assigned to devices from a different vendor.

If the configlets you need are not in the blueprint catalog (Staged > Catalog > Configlets), then you need to import them.

RELATED DOCUMENTATION

Import Configlet | 443

Topology

IN THIS SECTION

- Topology (Datacenter) | 71

Topology (Datacenter)

IN THIS SECTION

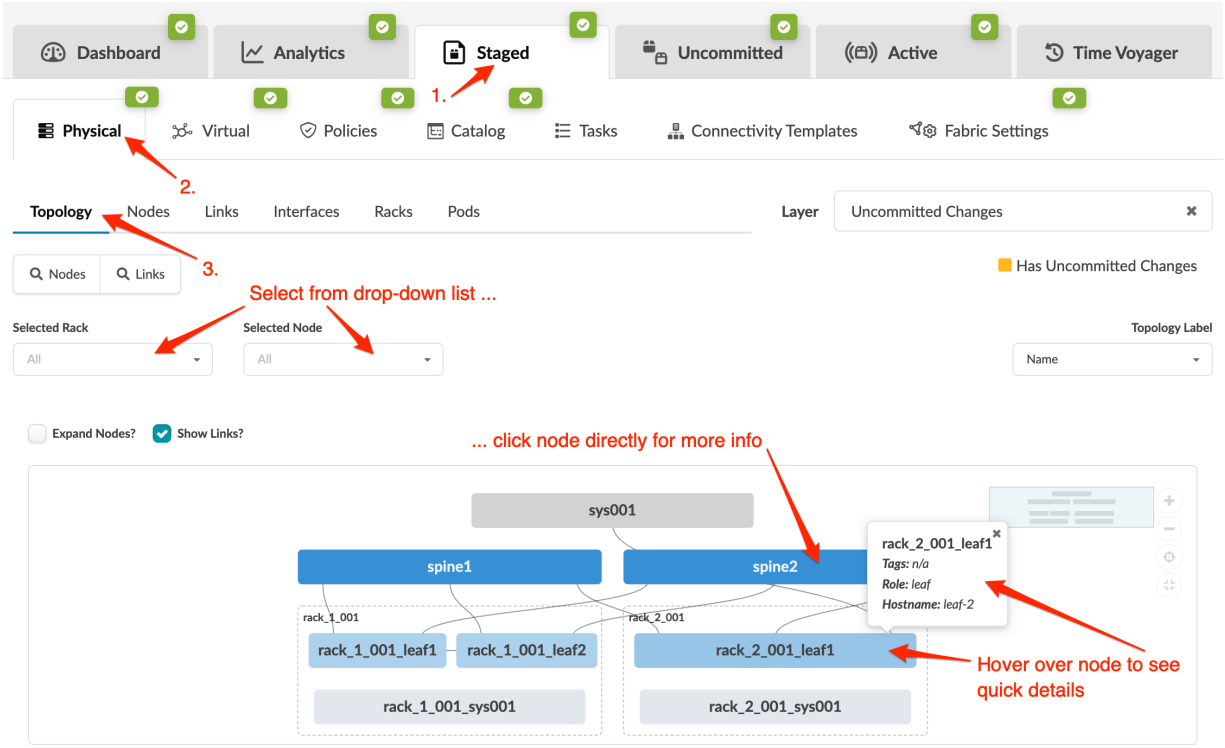
Before you push your changes to the active blueprint you can view progressive changes in the staged blueprint. This staging area allows you to validate that the pending changes are compliant with your intent, and that they work together with available resources and devices before you deploy the network.

Many node and link operations are performed from the **Topology** view. See "Nodes" on page 77 and "Links" on page 143 for more information.

You can view selections within topologies as neighbors, links, interfaces or virtual network endpoints, as applicable.

Main Topology View

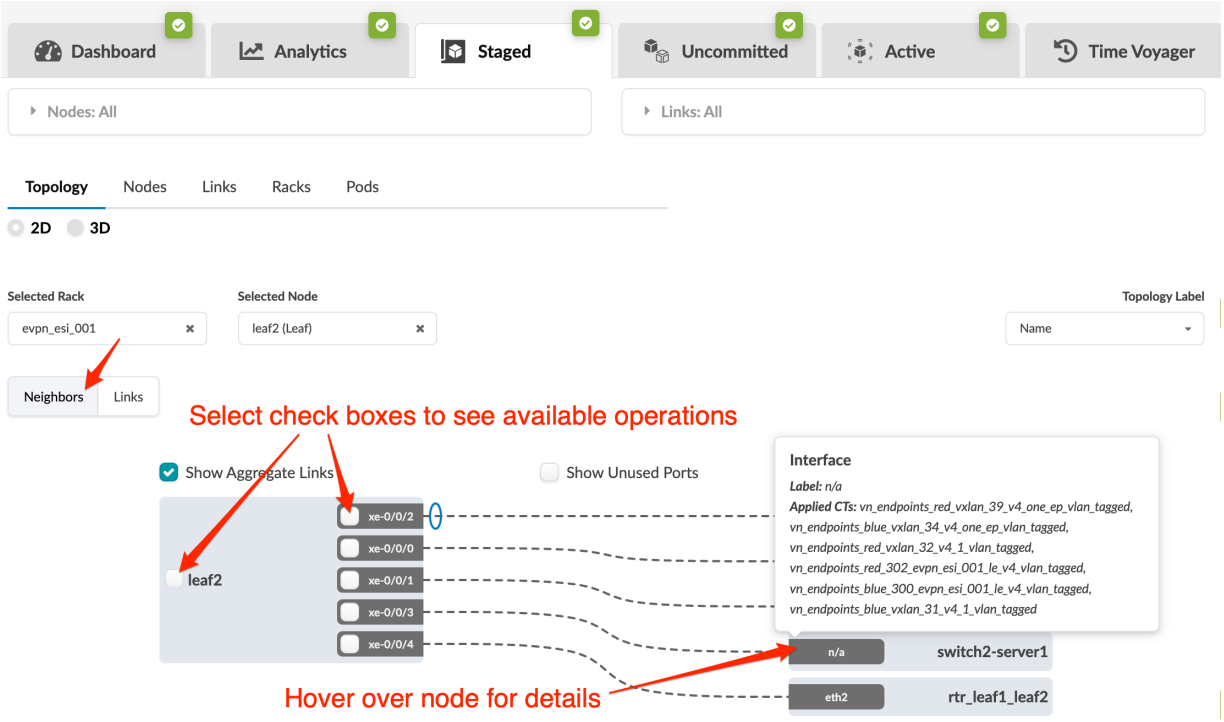
From the blueprint, navigate to **Staged > Physical > Topology**.



- To make topology elements larger, click the **Expand Nodes** check box.
- To display the links between elements, click the **Show Links** check box.
- To display a different layer, select the layer from the **Layer** drop-down list. **Uncommitted Changes** is an example of one of the layers you could display. The nodes with uncommitted changes are shown in yellow. The changes that apply to this layer are specific to the nodes themselves, such as ASN, loopback IP addresses and deploy modes. It doesn't apply to such changes as adding routing zones, virtual networks or connectivity templates on those nodes.
- To display additional information (node name, hostname, role, link, tags, as applicable), hover over a node or link.
- To display a different label (name, hostname, S/N), select a different label from the **Topology Label** drop-down list.
- To display a specific rack topology, click the rack element or select the rack from the **Selected Rack** drop-down list.
- To display a specific node topology, click the node element in the topology or select the node from the **Selected Node** drop-down list.

Neighbors Selection View

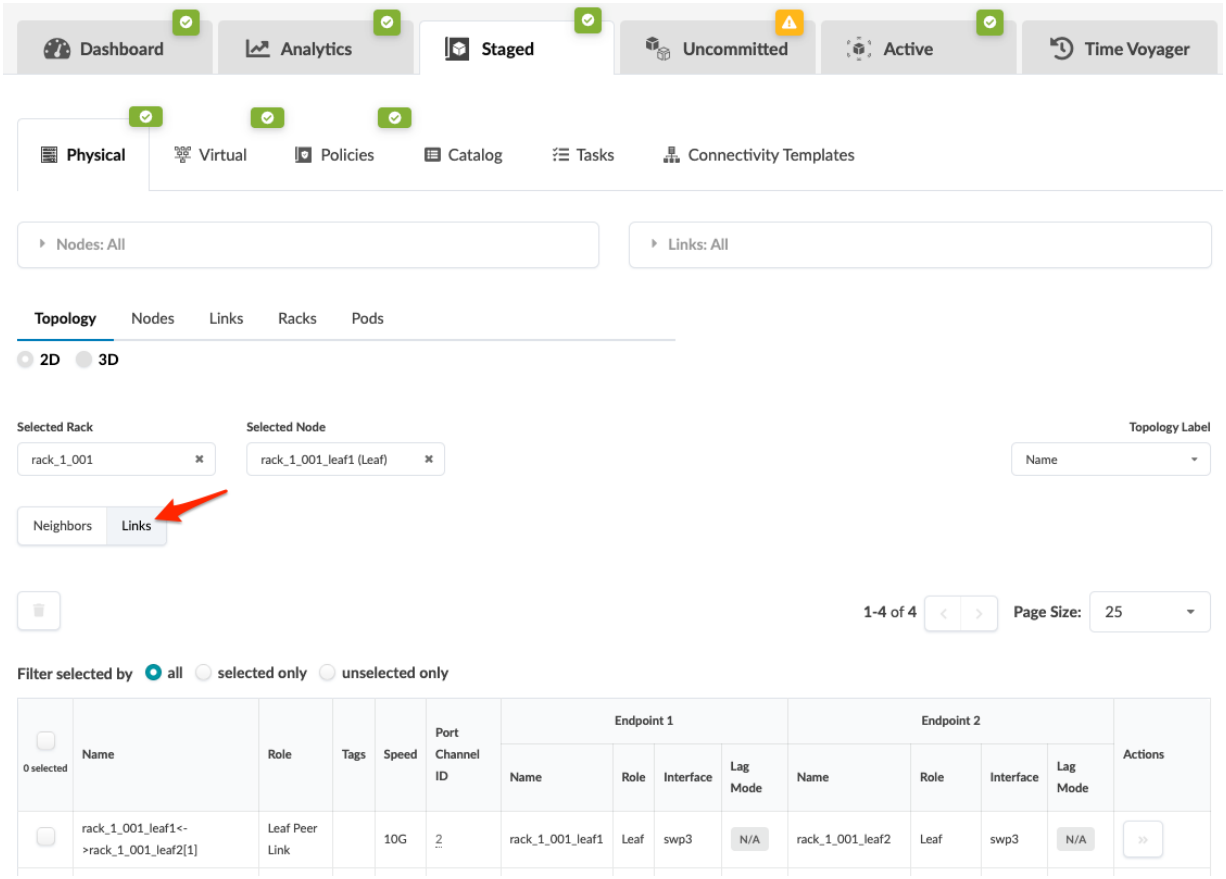
To see the neighbors view of a selection, click **Neighbors**.



- To display aggregate links, click the **Show Aggregate Links** check box.
- To display unused ports, click the **Show Unused Ports** check box.
- To display a different label (name, hostname, S/N), select a different label from the **Topology Label** drop-down list (right side).
- To display a particular neighbor type (all neighbors, generic, leaf, spine, and so on) select it from the **Show** drop-down list.
- To display available operations for a selected node or interface select the check box(es).
- To see details, hover over a node. Hovering over a generic system shows applied connectivity templates.

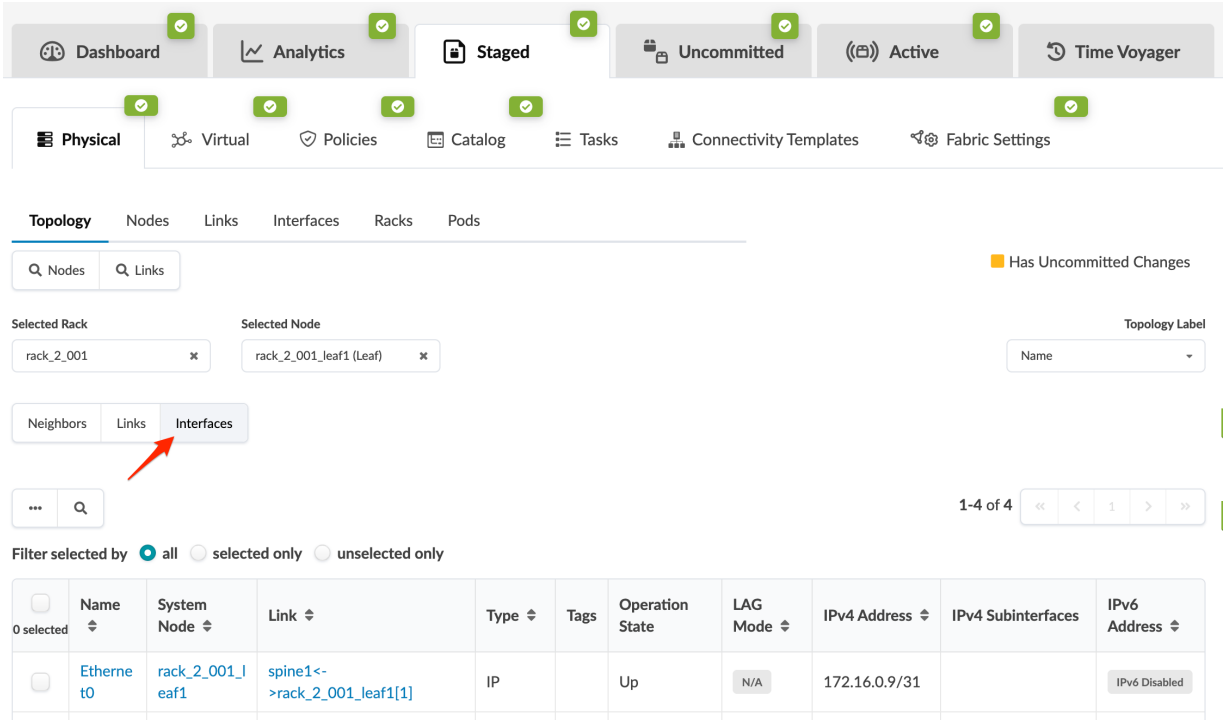
Links Selection View

To see the links view of a selection, click **Links**.



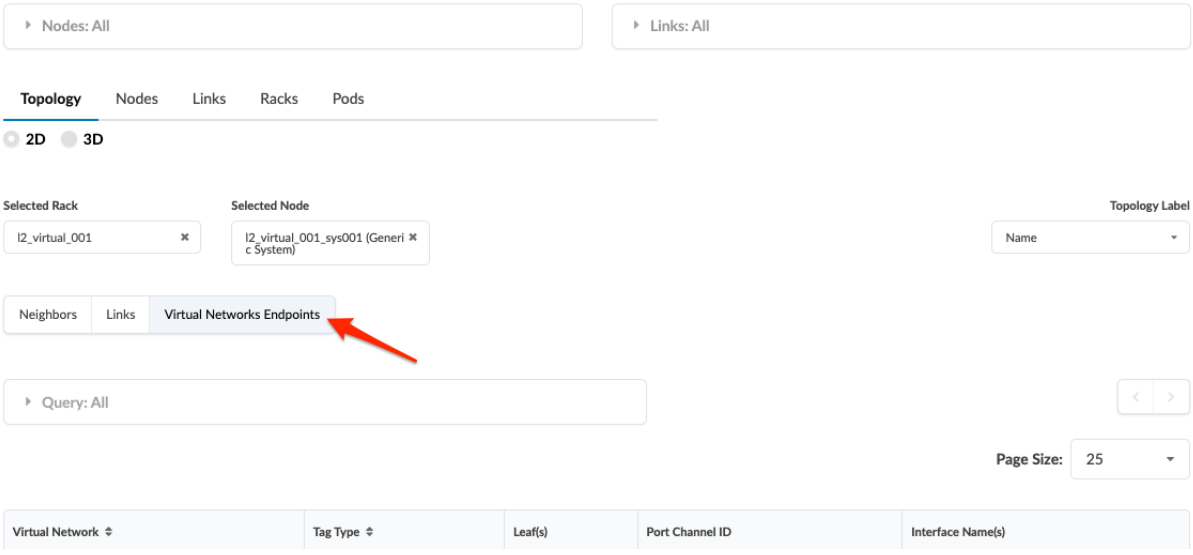
Interfaces Selection View

To see the interfaces view of a selection, click **Interfaces**.



Virtual Network Endpoints

To see the virtual network endpoints of a selection, click **Virtual Networks Endpoints**.

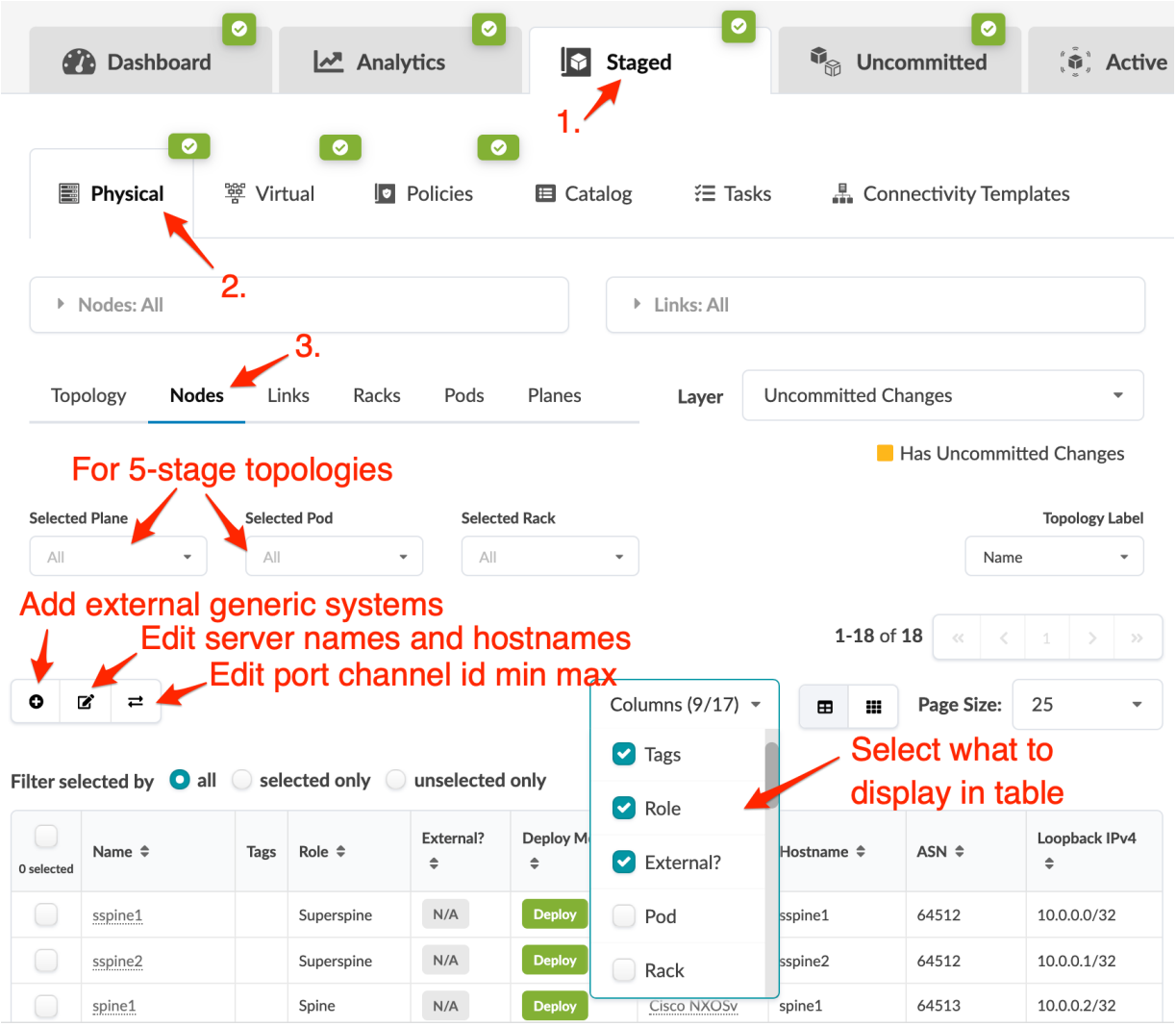


Nodes

IN THIS SECTION

Nodes (Datacenter)

From the blueprint, navigate to **Staged > Physical > Nodes** to go to the **Nodes** view.

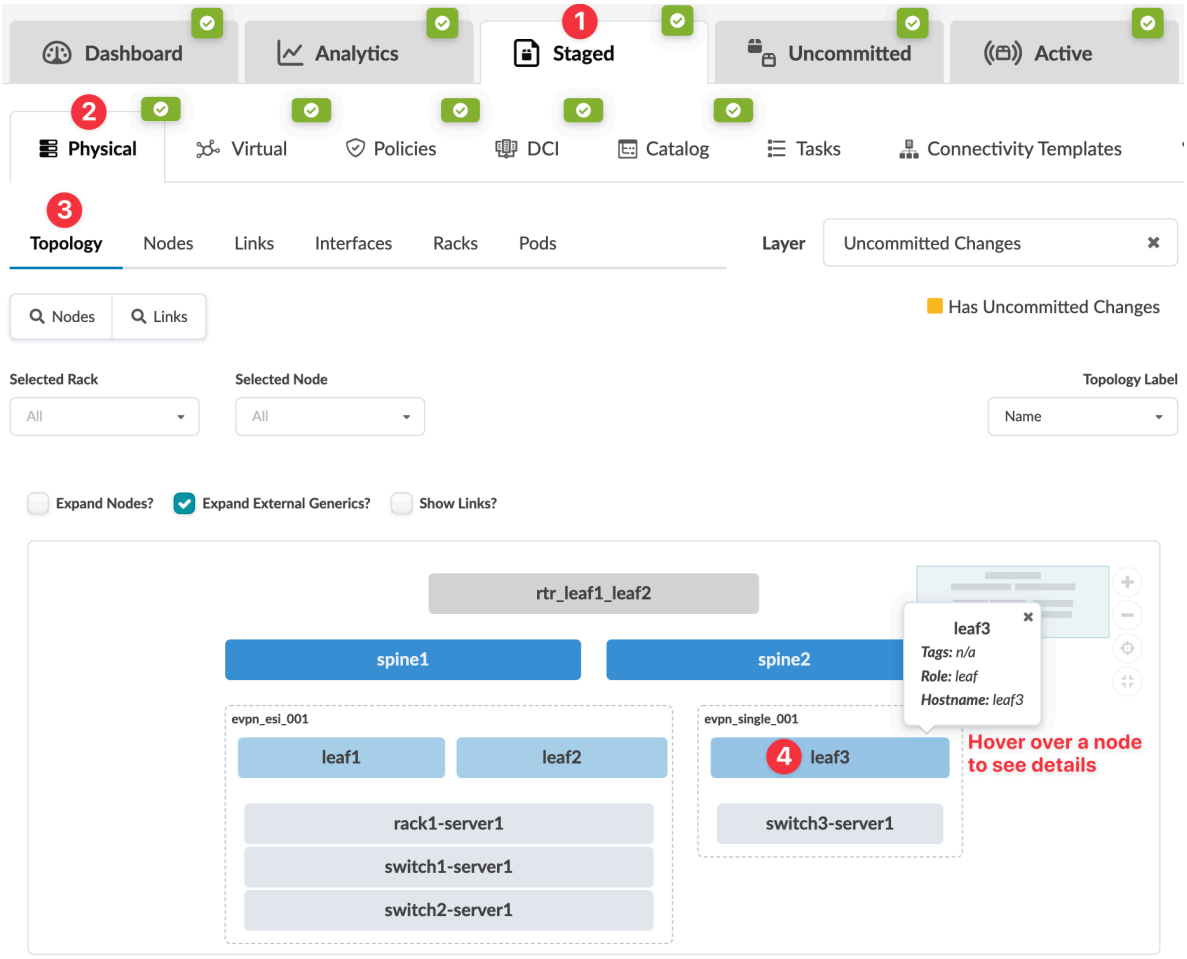


- You can view nodes in the table view or card view.
- In table view, you can select which details to display (from the drop-down list).
- You can click the name of a node in the table to display information in the right panel (such as telemetry, properties, and tags).

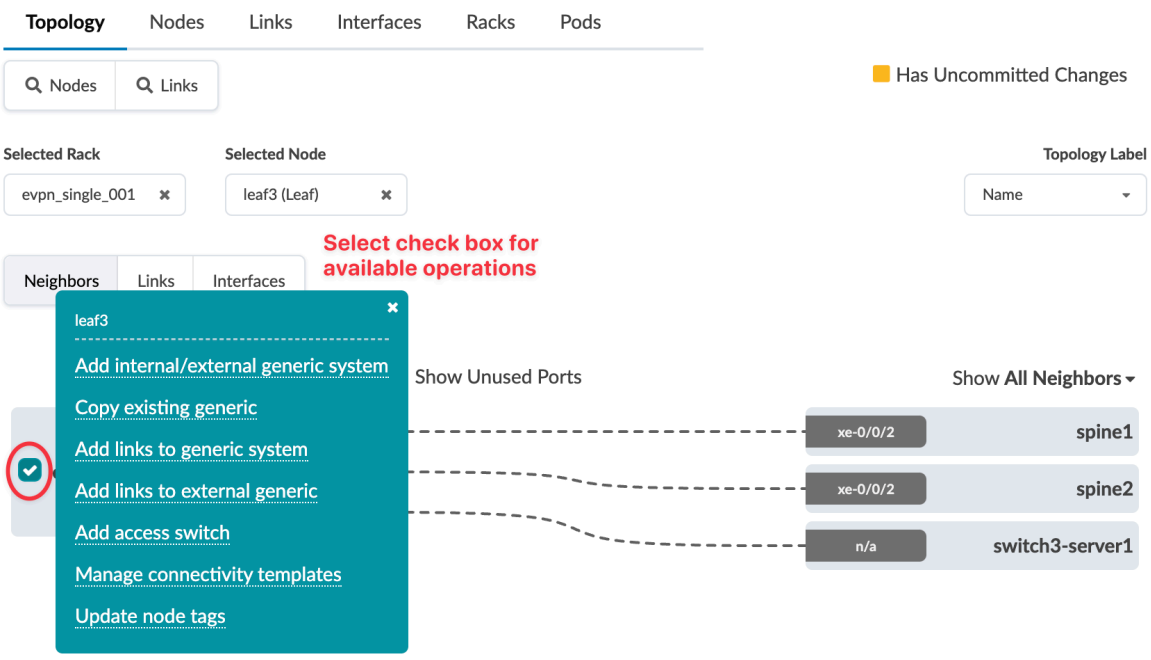
Many node operations are performed from the **Topology** view, and some can also be performed directly in the **Nodes** view. See the following sections for more information.

Create Access Switch

1. From the blueprint, navigate to **Staged > Physical > Topology** and select the leaf that you want to connect to the new access switch.

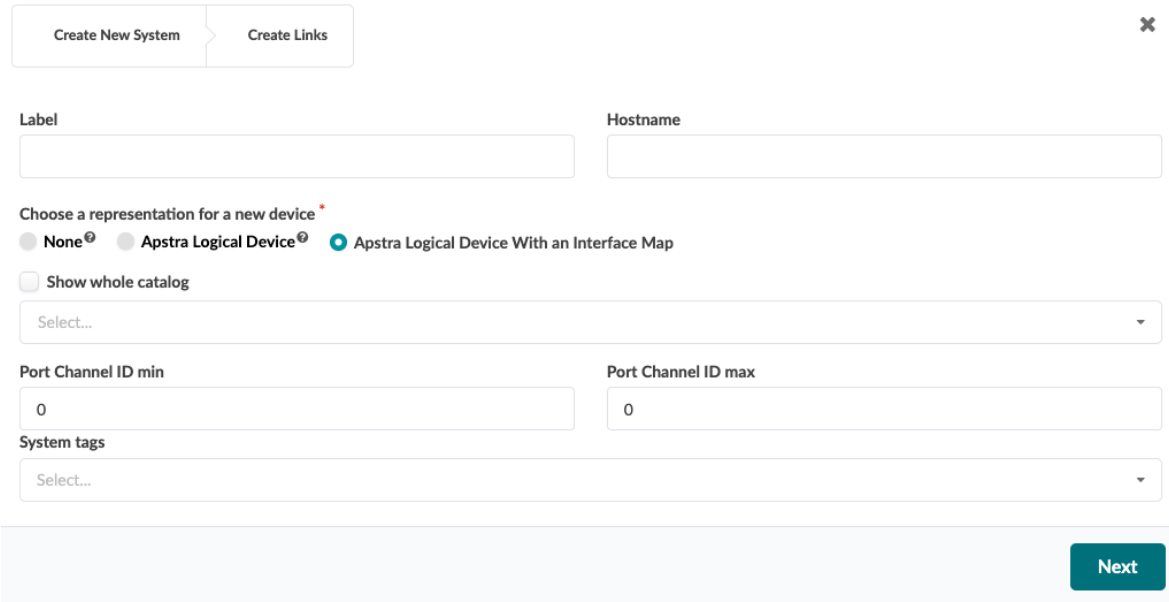


2. Select the node check box to see the operations available for that node (and that you have permissions for).



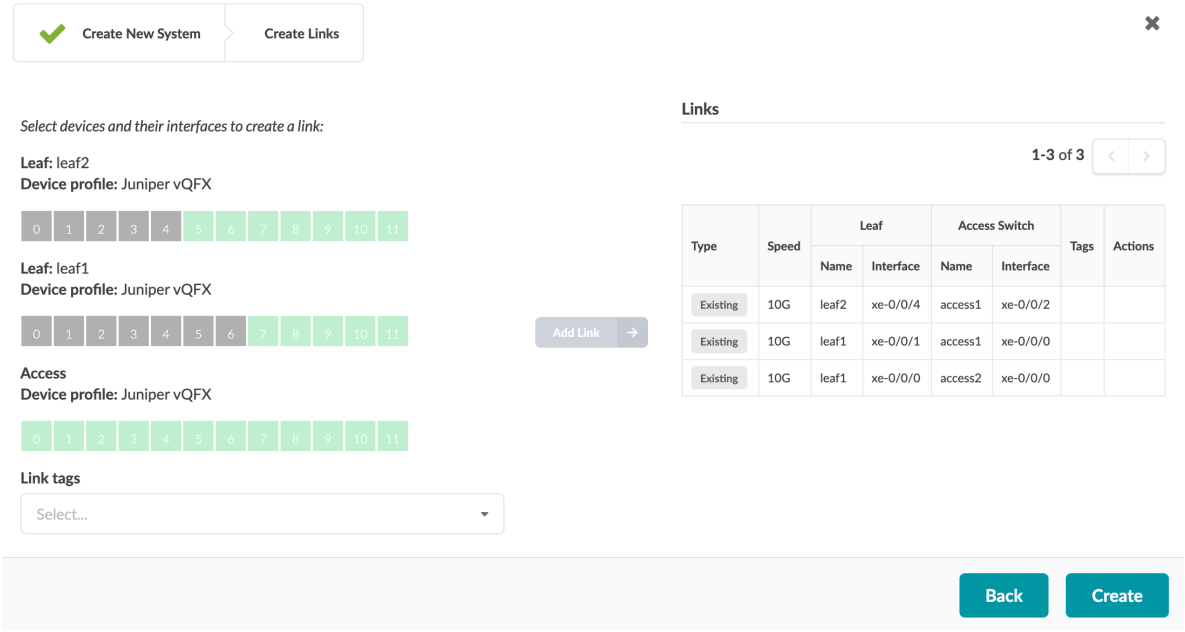
NOTE: You can also get to the selection page from the **Nodes** view. From the blueprint, navigate to **Staged > Physical > Nodes**, click the leaf name in the table, then click the leaf name that appears at the top of the **Selection** panel (on the right side of the page).

3. Click **Add access switch** and enter a unique label and hostname.

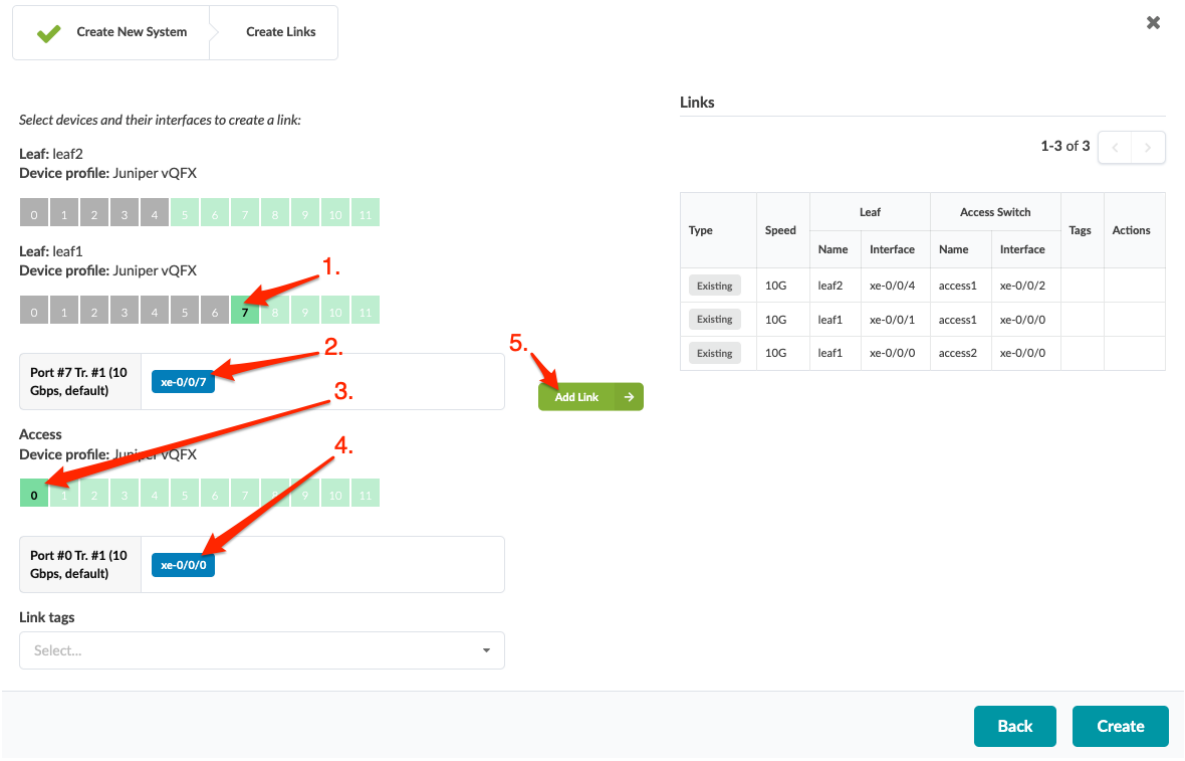


4. Select the appropriate interface map from the drop-down list.
5. Enter the port channel ID min and max.

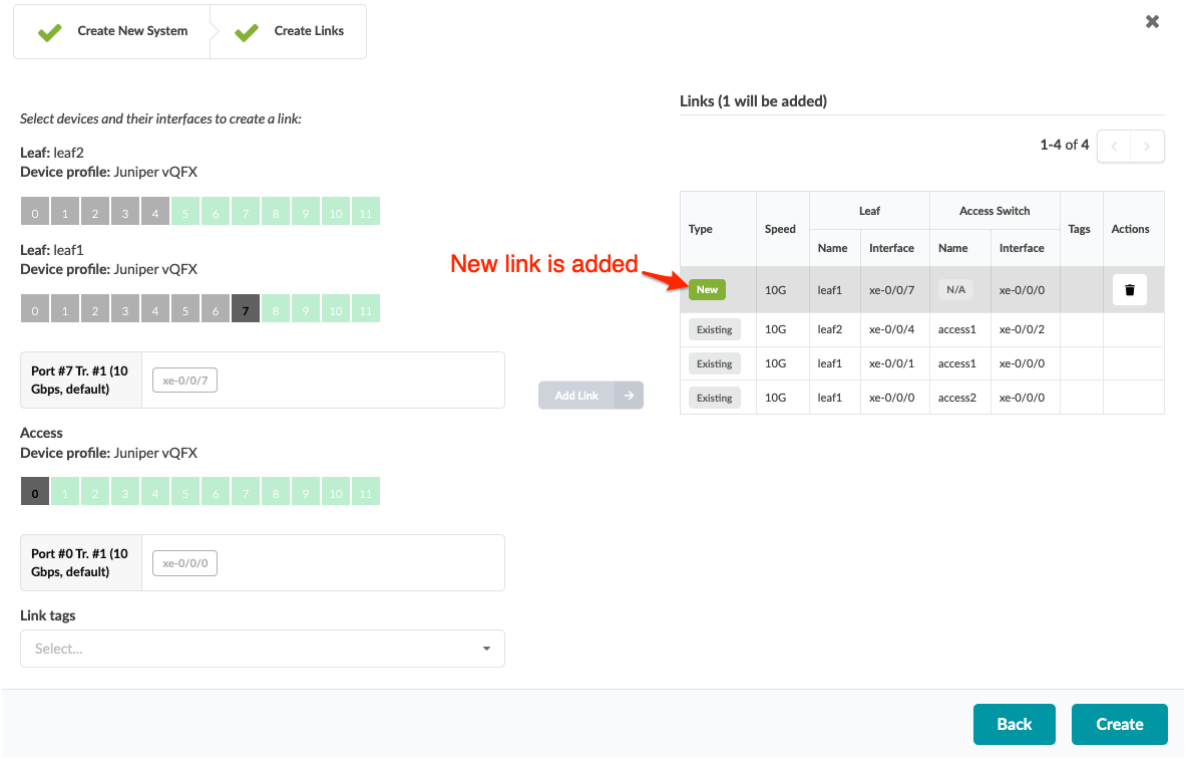
6. Enter tags (optional) to identify the role(s) of the new access switch, then click **Next**.



7. Select available ports and transformations, as applicable. The gray **Add Link** button turns green.



8. Click **Add Link**. The link is added to the link table.



9. Click **Create** to stage the change and return to the **Topology** view.

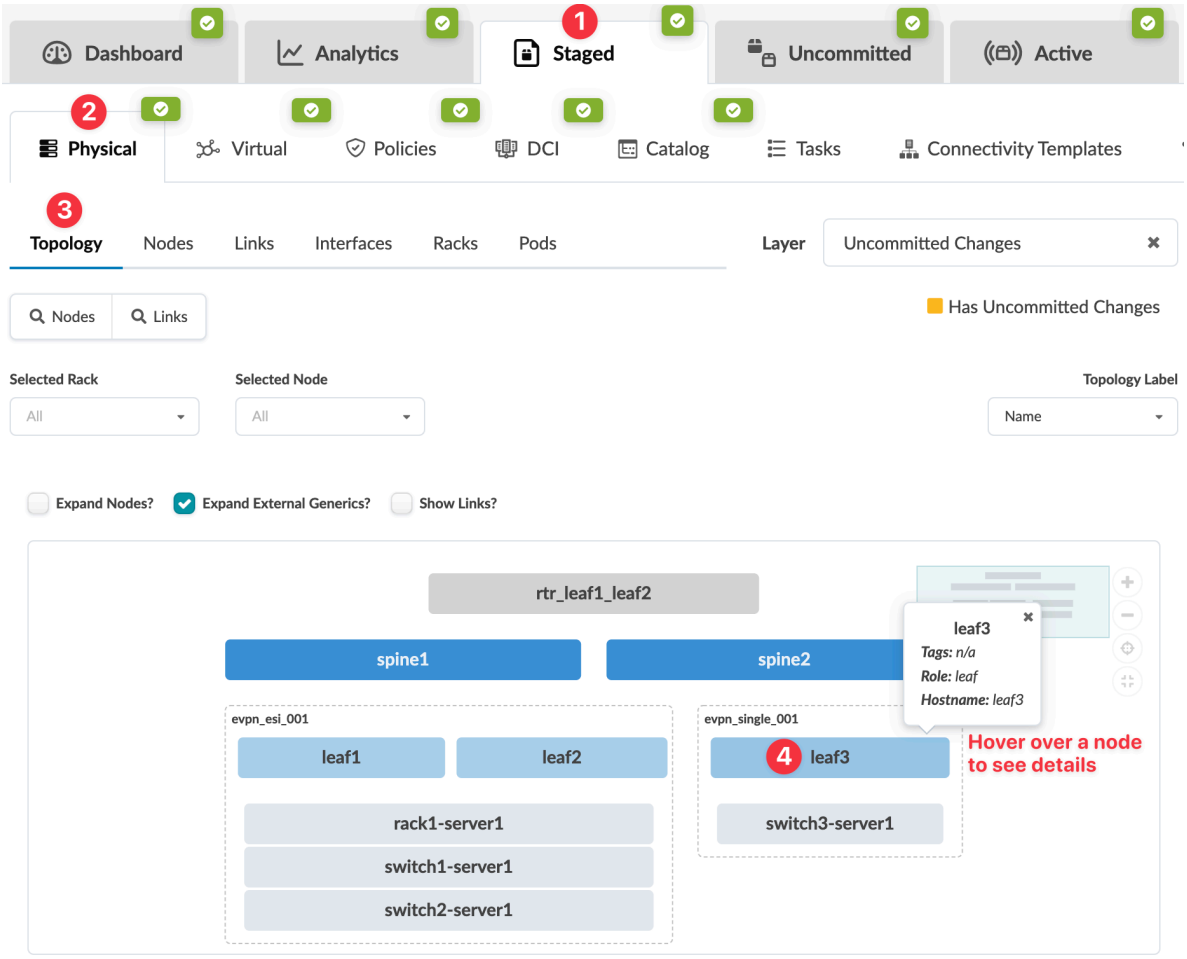
When you're ready to activate your changes, commit them from the **Uncommitted** tab.

RELATED DOCUMENTATION

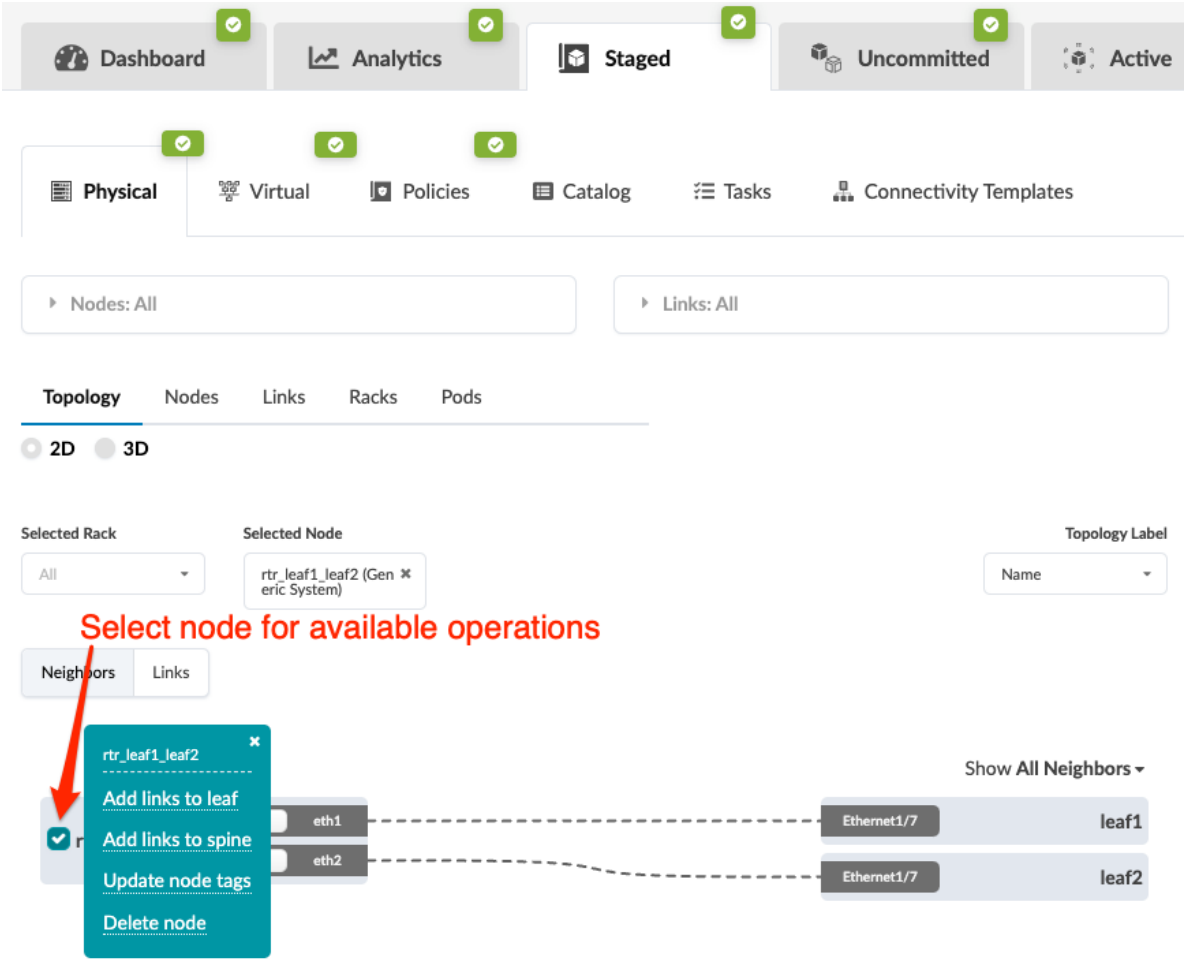
What are Rack Types | 862

Delete Node

1. From the blueprint, navigate to **Staged > Physical > Topology** and select the node to delete.

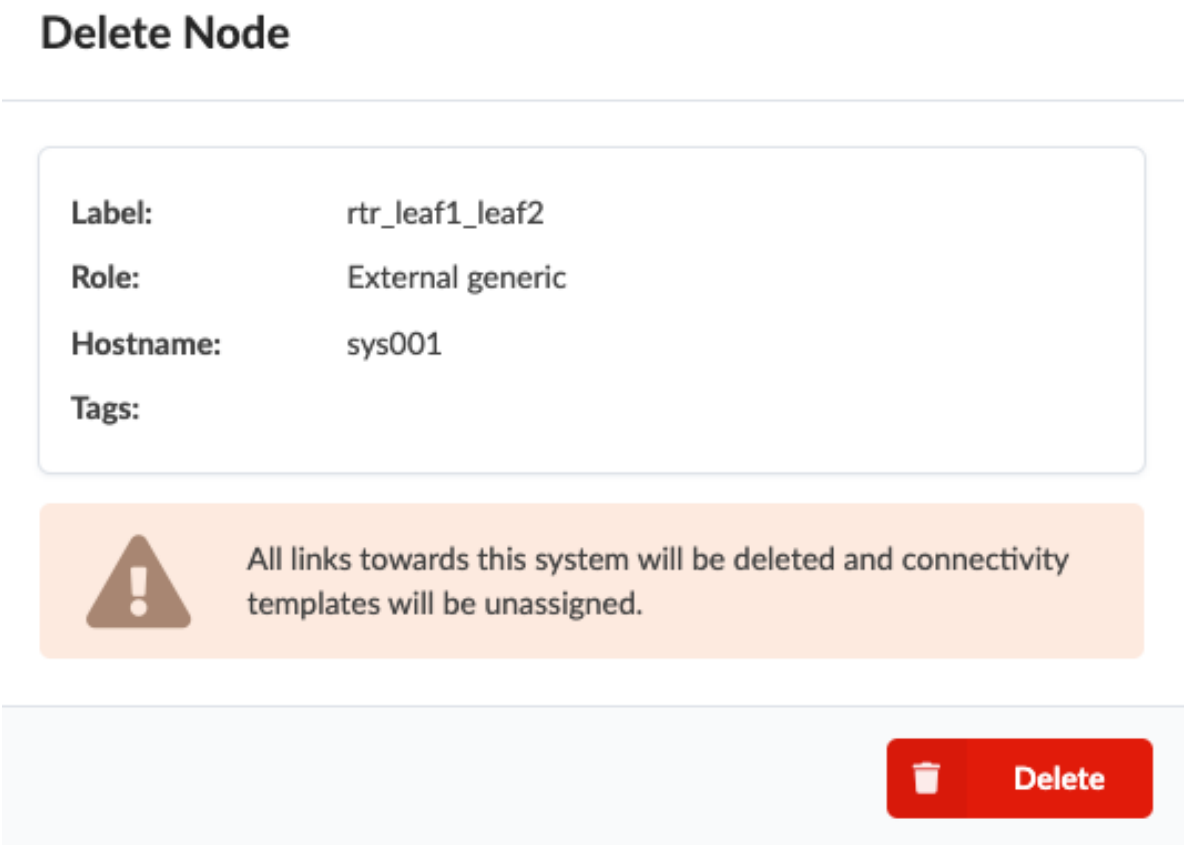


2. Select the check box to see the operations available for that node (and that you have permissions for).



NOTE: You can also get to the selection page from the **Nodes** view. From the blueprint, navigate to **Staged > Physical > Nodes**, click the node name in the table, then click the node name that appears at the top of the **Selection** panel (on the right side of the page).

3. Click **Delete node** to go to its dialog. All links towards the system will be deleted and connectivity templates will be unassigned for you.



4. Click **Delete** to stage the deletion and return to the **Topology** view.

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Update Deploy Mode (Datacenter)

IN THIS SECTION

Set Deploy Mode (from Build Panel)

1. From the blueprint, navigate to **Staged > Physical**, then in the **Build** panel (on the right) click the **Devices** tab.
2. If you don't see the nodes list, click the status indicator for **Assigned System IDs**.

3. Click a node name to see device details.
4. Click the **Edit** button for **Deploy Mode** and select a deploy mode. For deploy mode details, see "Device Configuration Lifecycle" on page 645.
5. Click the **Save** button to stage the change.

When you're ready to activate changes, commit them from the **Uncommitted** tab.

Set Deploy Mode (from Selection Panel)

1. From the blueprint, navigate to **Staged > Physical**.
2. Either from the **Topology** view or the **Nodes** view, select a node.
3. If it's not already selected, click the **Device** tab in the **Selection** panel (on the right).
4. Click the **Edit** button for **Deploy Mode** and select a deploy mode.
5. Click the **Save** button to stage the new deploy mode.

When you're ready to activate changes, commit them from the **Uncommitted** tab.

Set Deploy Mode (from Nodes View)

You can change the deploy mode for one or more nodes at the same time from the **Nodes** view.

1. From the blueprint, navigate to **Staged > Physical > Nodes** and check one or more check boxes for the node(s) to change. (You can narrow your search with the drop-down lists for planes, pods, and racks as applicable.)
2. Click the **Set Deploy Mode** button (fourth of five buttons above the nodes list) and select a deploy mode. (To filter selection before changing deploy mode, you can use the query.)
3. Click **Set Deploy Mode** to stage the change and return to the **Nodes** view.

When you're ready to activate changes, commit them from the **Uncommitted** tab.

SEE ALSO

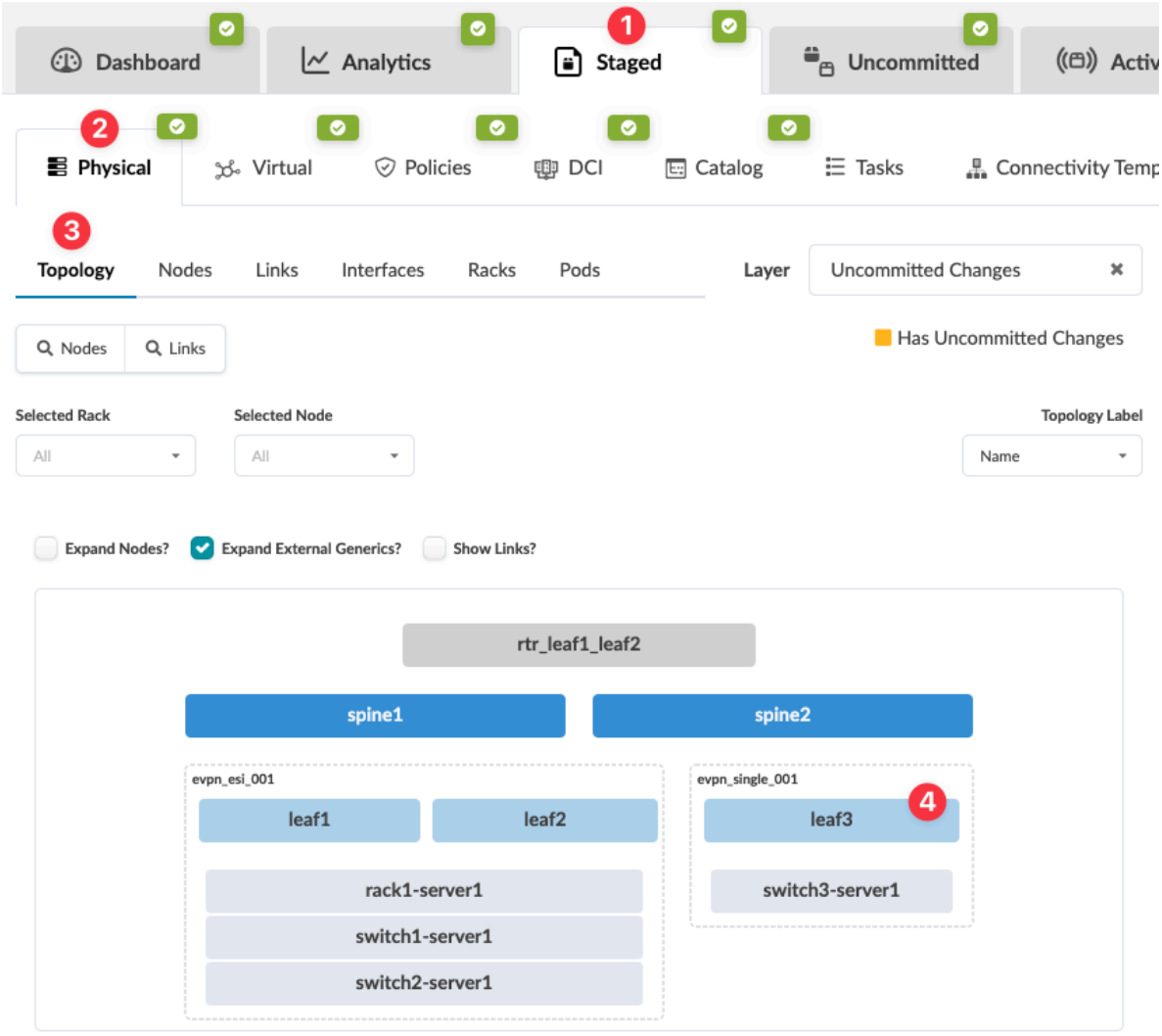
Commit / Revert Changes to Blueprint | 599

Unassign Device (Datacenter)

IN THIS SECTION

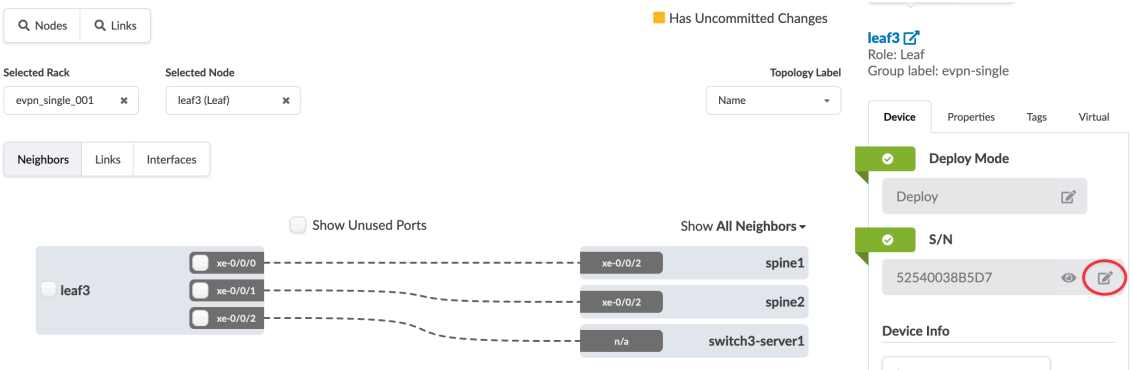
Unassign Device (from Device Selection Panel)

1. From the blueprint, navigate to **Staged > Physical > Topology**, and click the device to be removed.



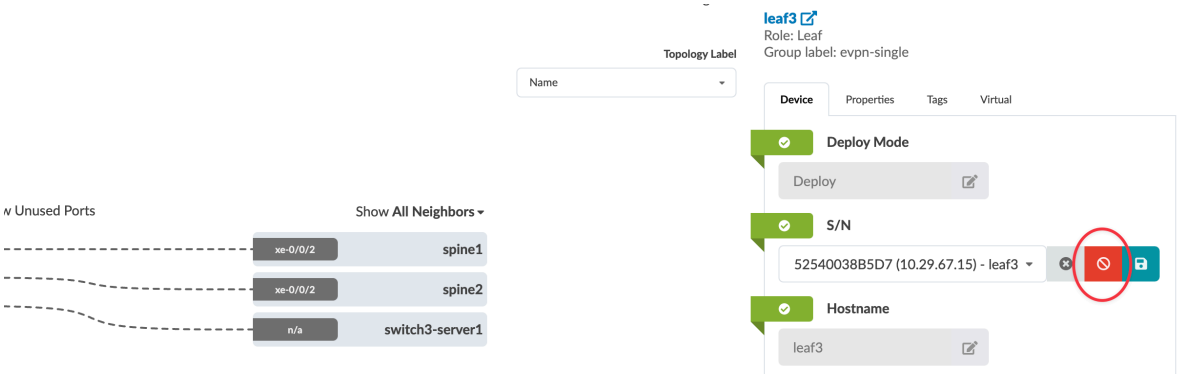
   The topology for the selection appears.

2. In the right panel, in the **Device** tab, click the **Edit** button for **S/N**.

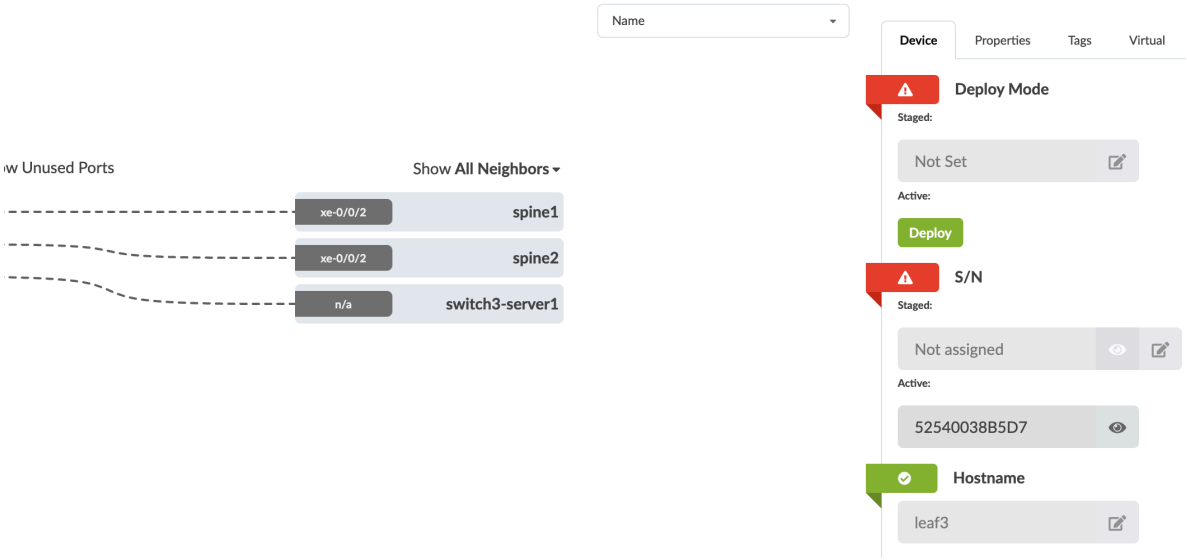


The S/N (system ID) field becomes editable

3. Click the red square in the **S/N** section to unassign the system ID.



The S/N is unassigned, and the deploy mode automatically changes to **Not Set** (new in Apstra version 5.0.0)

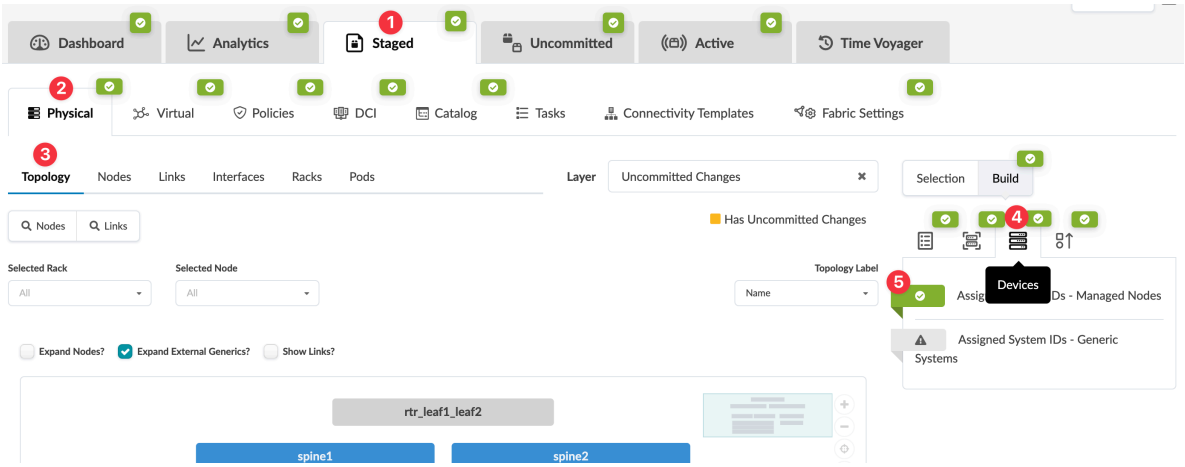


When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes. The device will still be under Apstra management. It's ready and available to be assigned to any blueprint.

To remove the device completely from Apstra management, "remove the device from Managed Devices" on page 673.

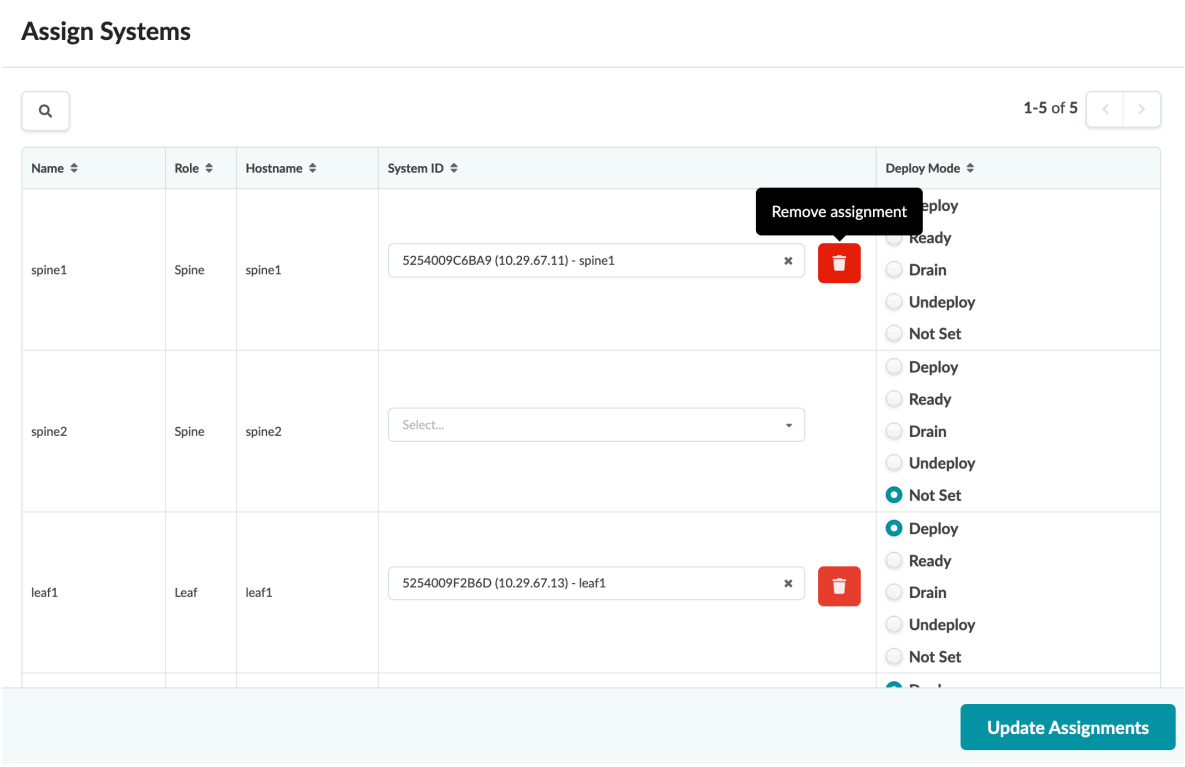
Unassign Device(s) (from Devices Build Panel)

1. From the blueprint, navigate to **Staged > Physical > Topology** and click the **Devices** tab in the **Build** panel (on the right), then click the status indicator for **Assigned System IDs**.



The **Assigned System IDs** list appears in the right panel.

2. Click the **Change System IDs assignments** button (the edit icon below Assigned System IDs) The **Assign Systems** dialog opens.
3. Click the **Remove assignment** button (trash can icon) for the device to unassign. The deploy mode is automatically set to **Not Set** (as of Apstra version 5.0.0).



4. Click **Update Assignments** (bottom-right in dialog) to stage the change and return to the **Topology** view.

When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes. The device will still be under Apstra management. It's ready and available to be assigned to any blueprint.

To remove the device completely from Apstra management, "remove the device from Managed Devices" on page 673.

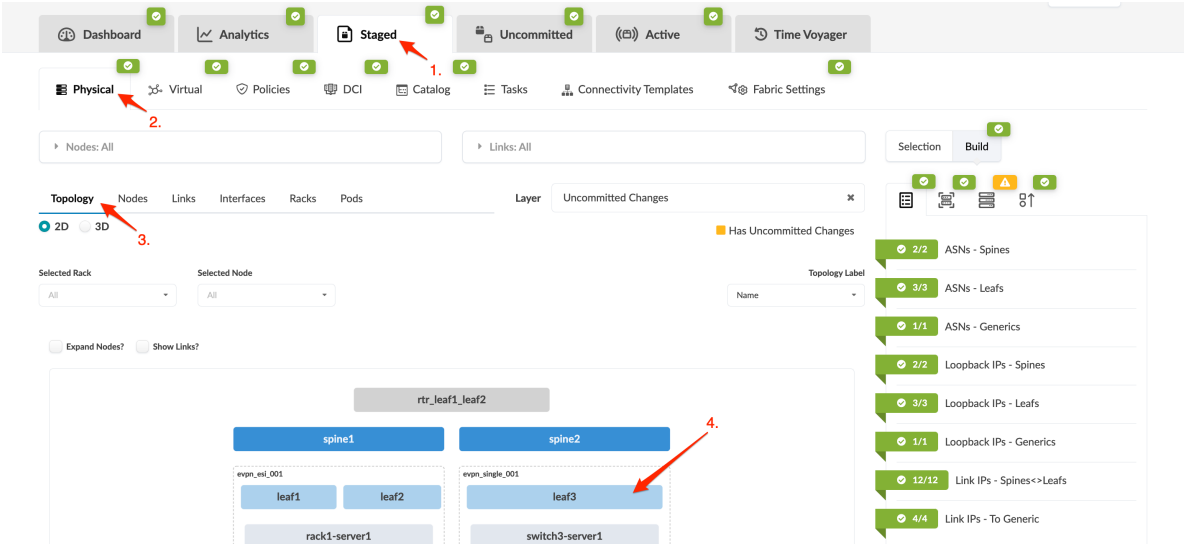
SEE ALSO

Commit / Revert Changes to Blueprint | 599

Execute CLI Show Command (Data Center Blueprint)

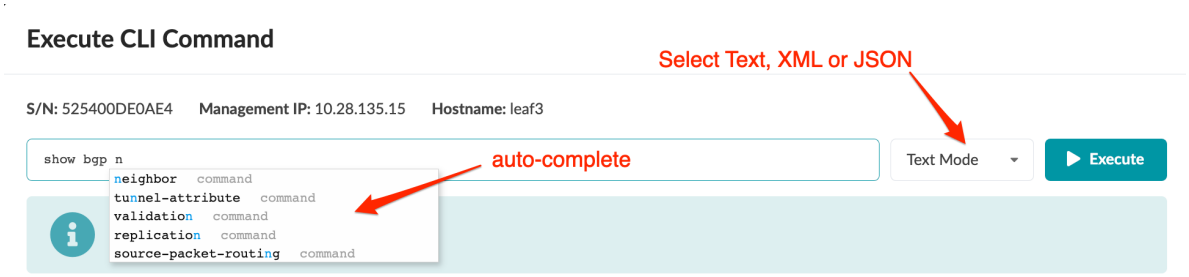
While in the Apstra environment, you may need device information that's obtained via CLI commands. Traditionally, you need to log in to a machine with access to the device management network, open a terminal, find device IP addresses, SSH to each of them, then run the required CLI commands. You can bypass these steps and run show commands for Juniper devices directly from the Apstra GUI. You can execute CLI commands from within the staged or active blueprint, or from the **Managed Devices** page. The steps below are for Datacenter blueprints.

1. From the blueprint, navigate to **Staged > Physical > Topology** (or **Staged > Physical > Nodes**) and select a Juniper device node.

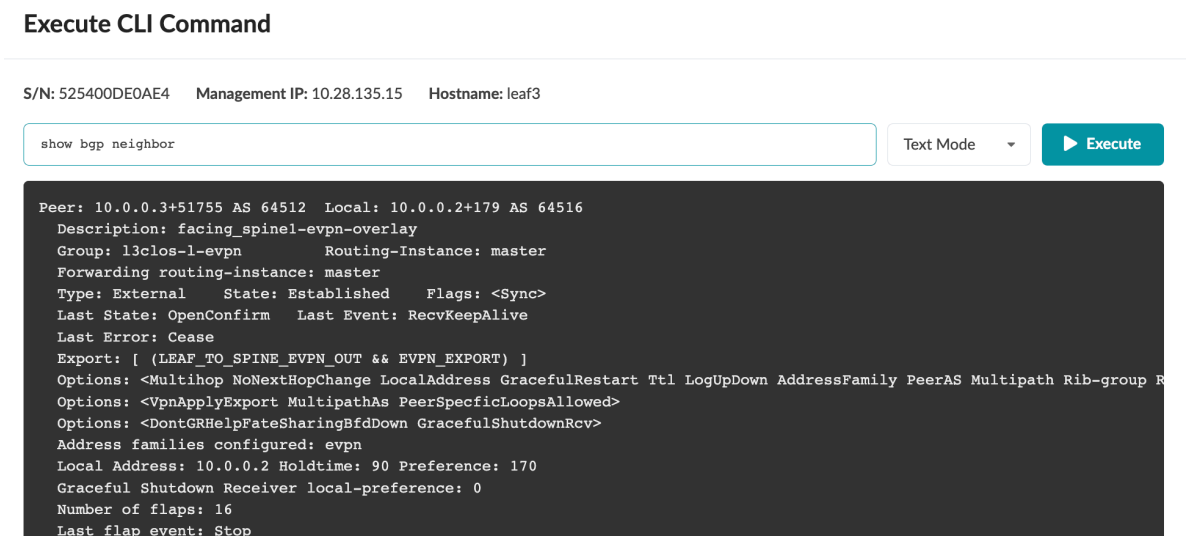


2. In the **Selection** section that appears in the right panel, on the **Device** tab, click **Execute CLI Command**.

3. In the dialog that opens type `show`, then press the space bar. Available commands appear that you can scroll through to select, or you can start typing the command and it will auto-fill. In our example we're looking for BGP neighbors. We typed `show`, space, then `b`, which filtered the commands to only include those with the letter b. We selected `bgp`, then pressed the space bar to show available arguments for bgp. We typed `n` to show commands including the letter n. We'll select `neighbor` to complete the command.



4. From the drop-down list, select how you want to view the results: text, XML or JSON.
5. Click **Execute** to return show command results. We used **Text Mode** for our example.



RELATED DOCUMENTATION

Execute CLI Show Command (Devices) | 671

Change Hostnames / Names

IN THIS SECTION

Change Leaf Hostname/Name

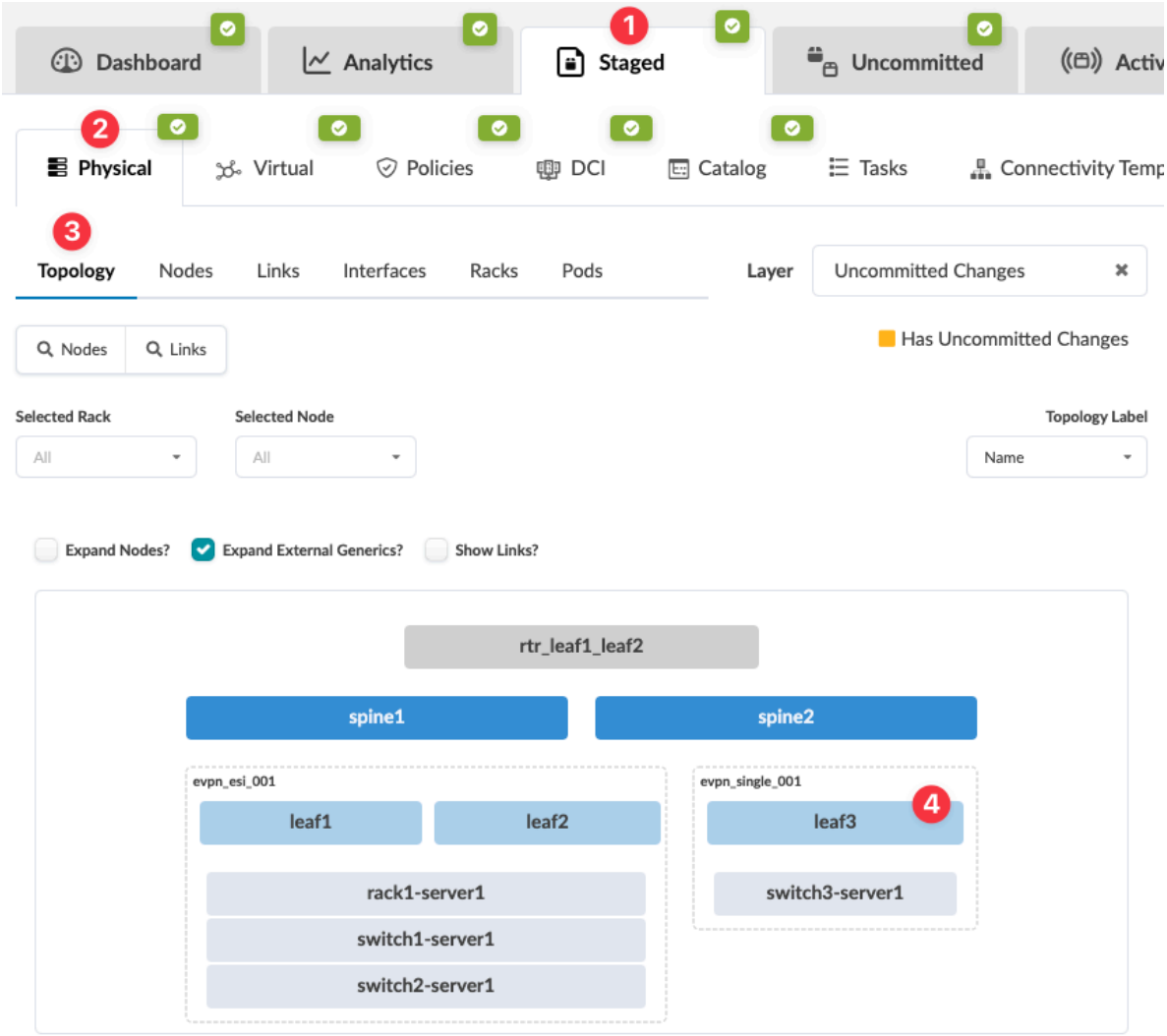
SUMMARY

You can change the hostname and/or name of a single leaf device or of multiple leaf devices simultaneously.

IN THIS SECTION

Change the Hostname and/or Name of a Single Leaf Device

1. From the blueprint, navigate to **Staged > Physical > Topology** and select the applicable leaf.

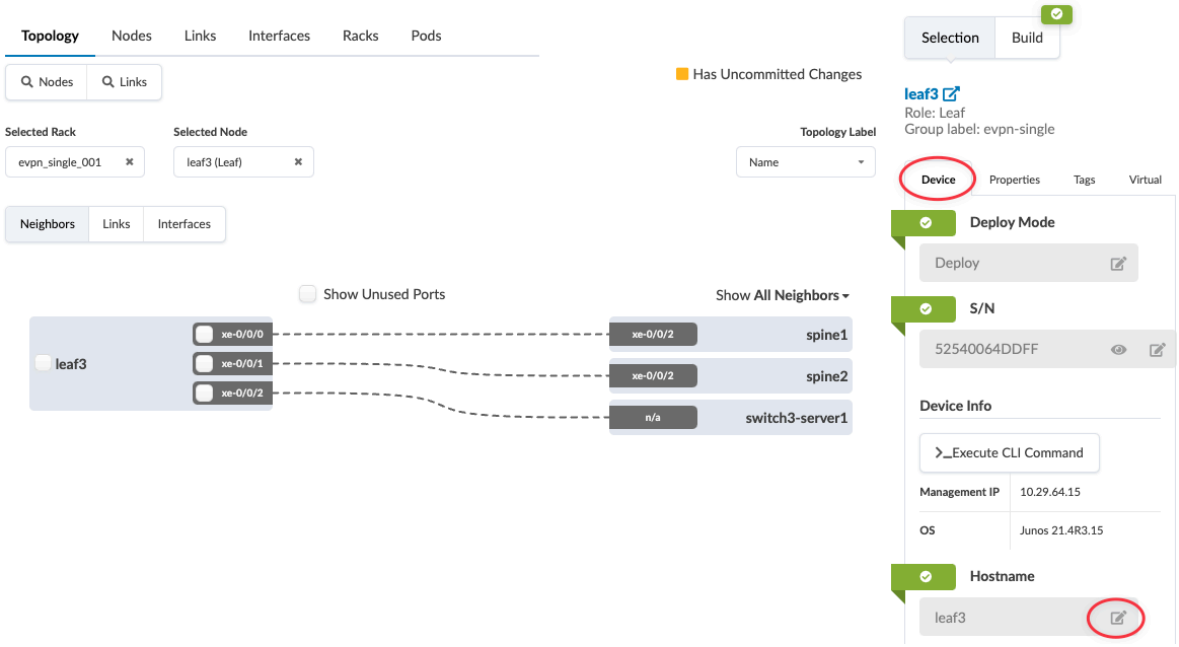


The **Selection** panel opens (on the right).

> **NOTE**: You can also get to the **Selection** panel by clicking the name of the leaf in the **Nodes** table (Staged > Physical > Nodes).

2. To change the leaf name - From the **Properties** tab in the **Selection** panel, click the **Edit** button for **Name**, change the name accordingly, then click the **Save** button. (Notice you can change other various parameter values from the **Properties** tab.)

3. To change the leaf hostname - click the **Device** tab in the **Selection** panel, click the **Edit** button for **Hostname**, change the name accordingly, then click the **Save** button. (Notice you can change other various parameter values from the **Device** tab.)



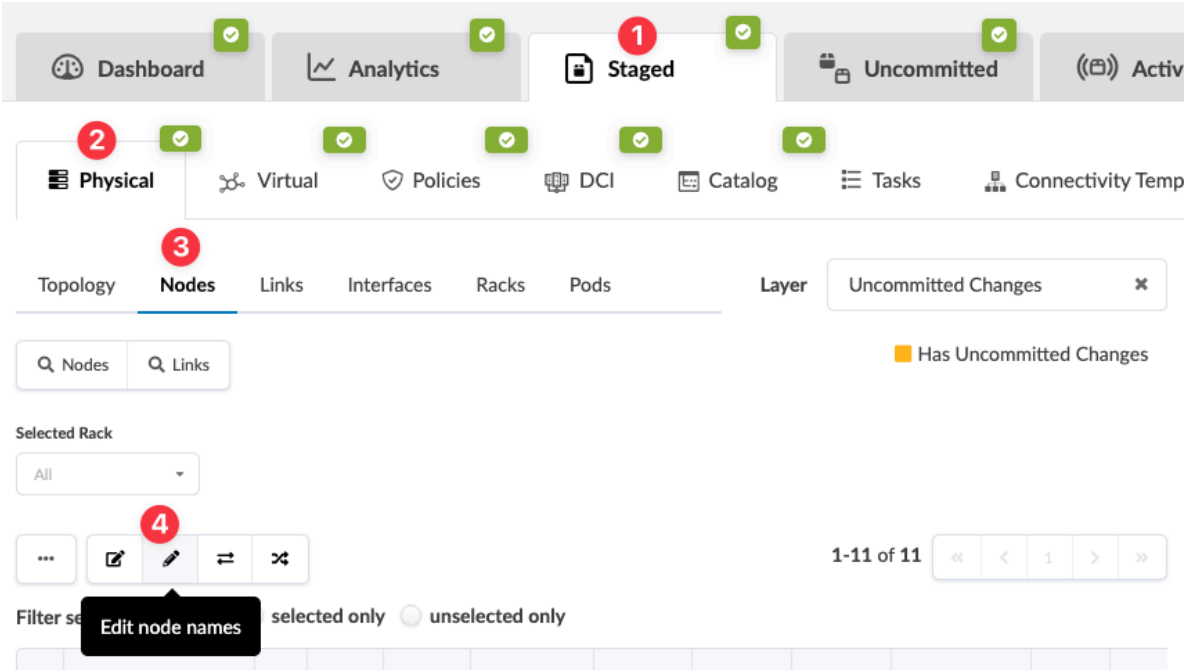
If you changed leaf names in a leaf pair, the leaf pair name does not change. You can manually "change the leaf pair" on page 95 name to correspond with the new leaf names. This is especially useful when assigning leaf pairs when you create virtual networks.

When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

Change Hostnames and/or Names of Multiple Leaf Devices

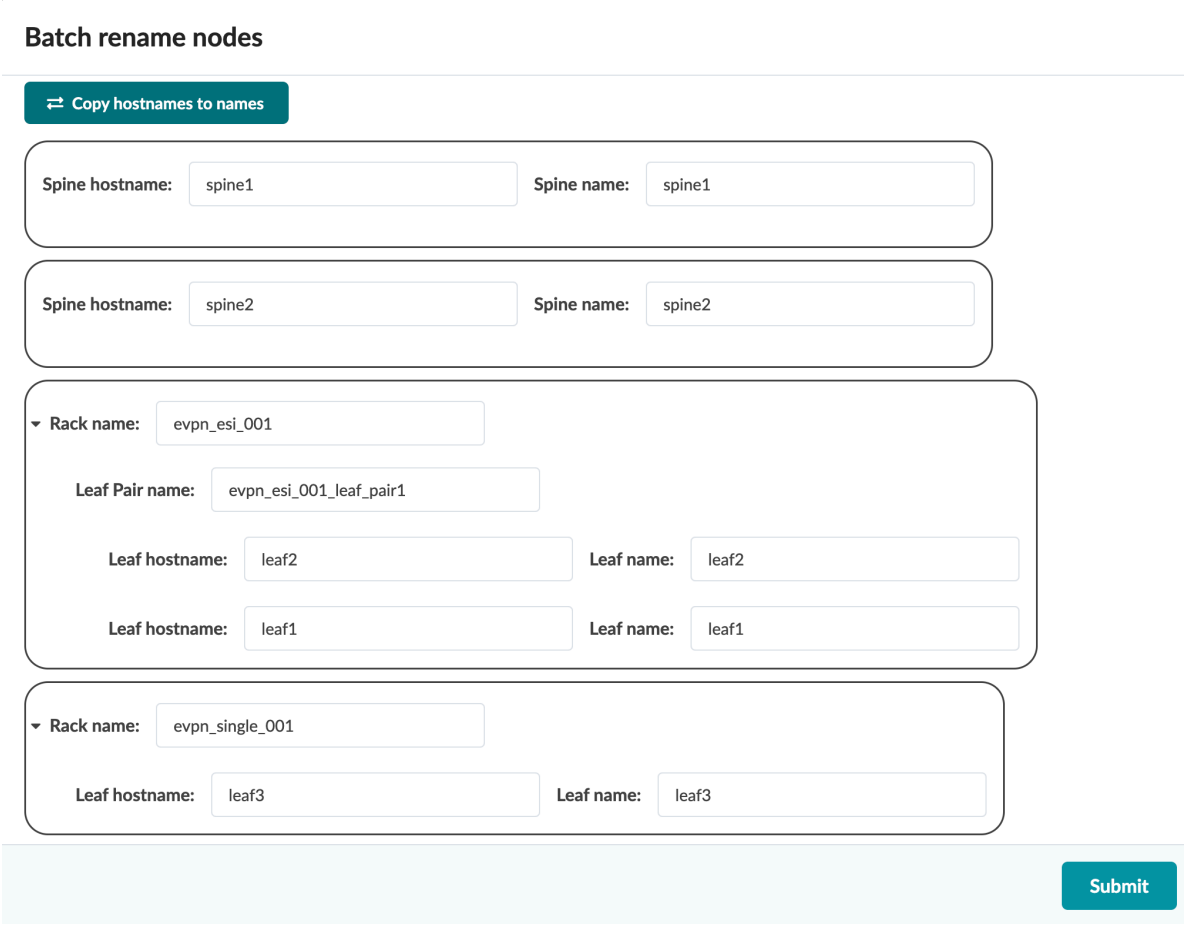
New feature in Apstra version 5.0.0.

1. From the blueprint, navigate to **Staged > Physical > Nodes** and click the **Edit Node Names** button, above the table.



The **Batch rename nodes** dialog opens.

2. Change leaf hostnames and/or names, as applicable. If you're changing leaf names in a leaf pair, the leaf pair name doesn't change automatically. You can change the leaf pair name to correspond with the new leaf names from this same dialog. To efficiently change the names to match the hostnames, after you've changed the hostnames click **Copy hostnames to names**. All names in the dialog change to match the hostnames.



3. Click **Submit** to stage the changes and return to the **Nodes** table.

When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

Change Leaf Pair Name

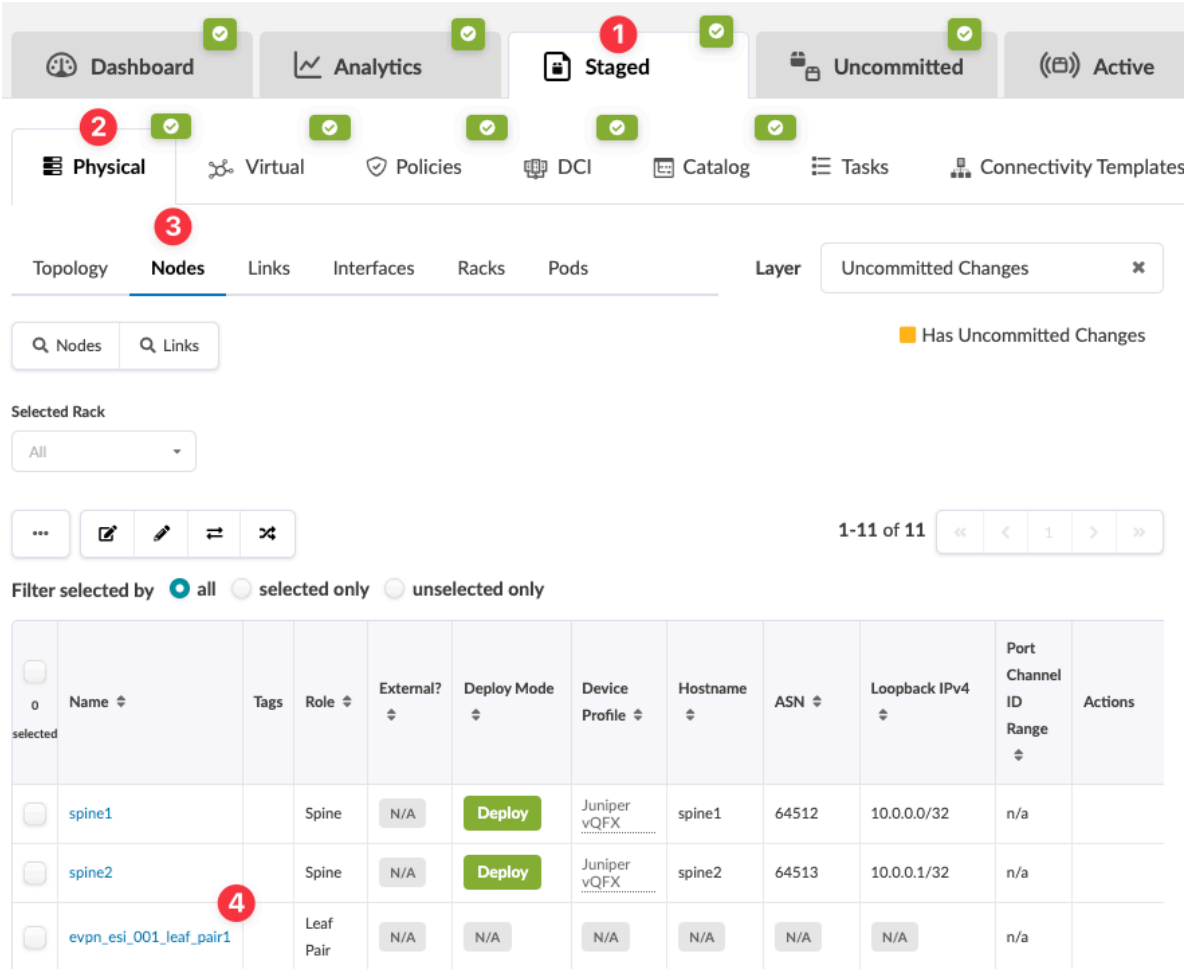
SUMMARY

You can change the leaf pair name of a single leaf pair or of multiple leaf pairs simultaneously.

IN THIS SECTION

Change Leaf Pair Name of a Single Pair

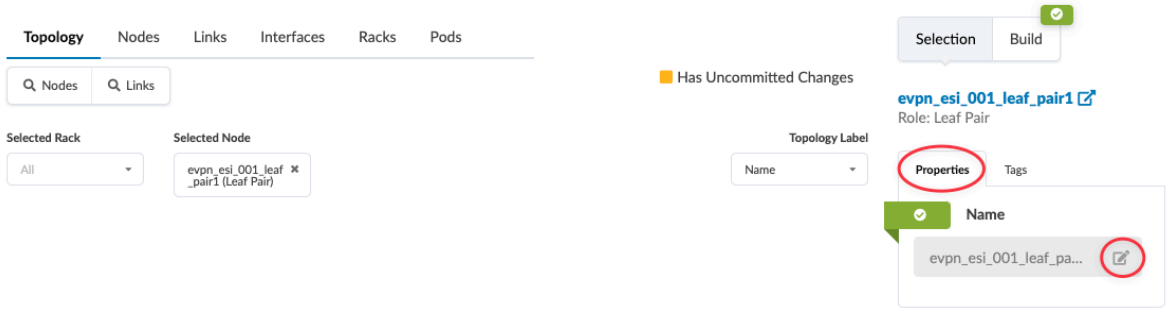
1. From the blueprint, navigate to **Staged > Physical > Nodes** and select the *name* of the applicable leaf pair (not the check box).



The **Selection** panel opens (on the right).

> **NOTE**: You can also get to the **Selection** panel from the **Topology** view (Staged > Physical > Topology) by selecting the leaf pair from the **Selected Node** drop-down list.

2. From the **Properties** tab in the **Selection** panel, click the **Edit** button for **Name**, change the name accordingly, then click the **Save** button.

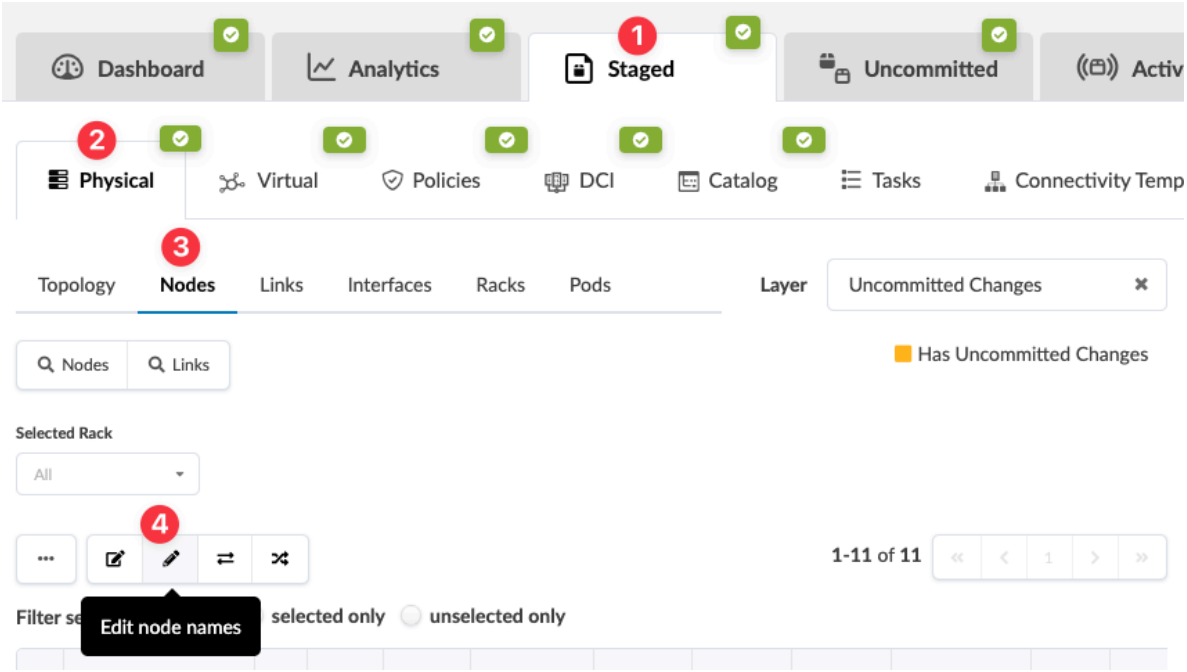


When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

Change Leaf Pair Names of Multiple Pairs

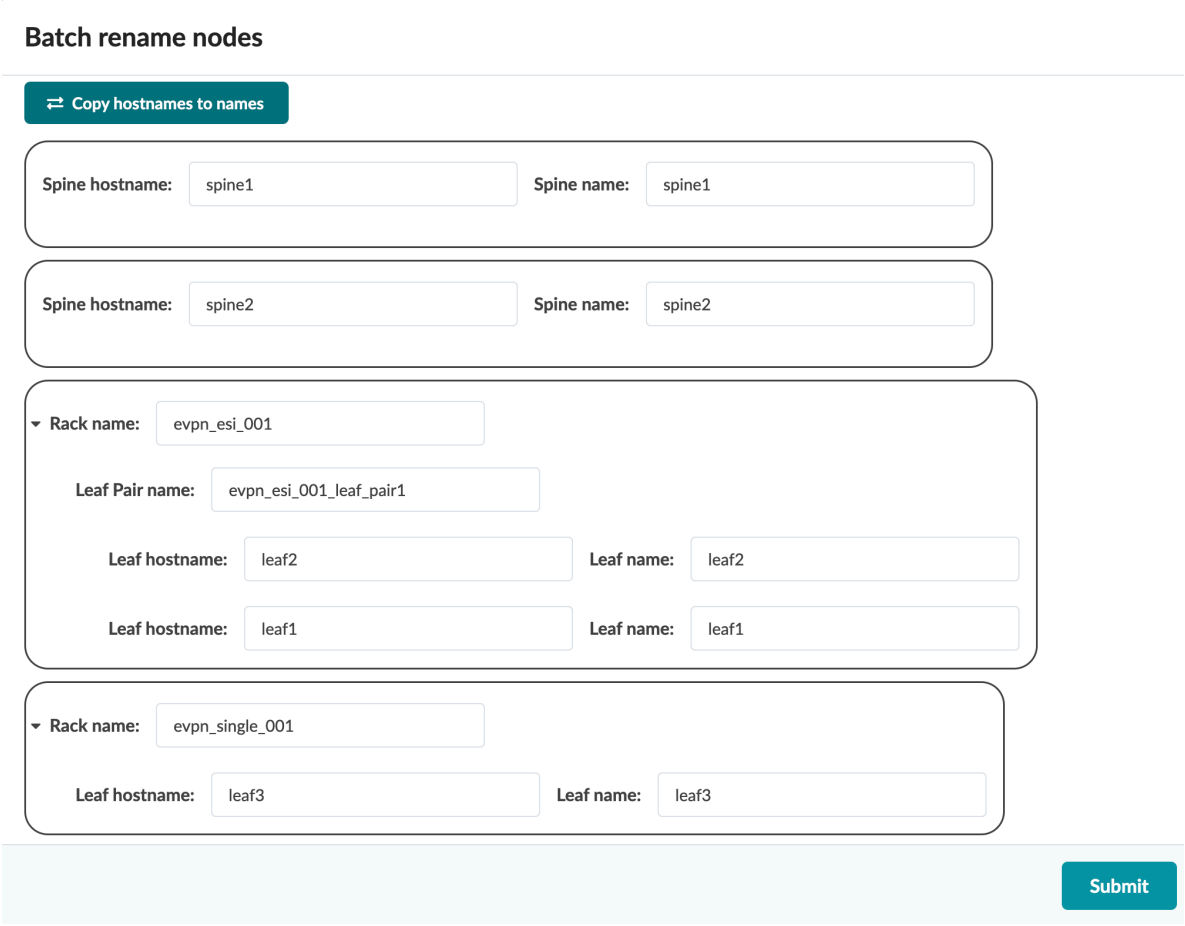
New feature in Apstra version 5.0.0.

1. From the blueprint, navigate to **Staged > Physical > Nodes** and click the **Edit Node Names** button, above the table.



   The **Batch rename nodes** dialog opens.

2. Change leaf pair names, as applicable. (Notice that you can change other various hostnames and names from this dialog as well.)



3. Click **Submit** to stage the changes and return to the **Nodes** table.

When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

Change Spine Hostname/Name

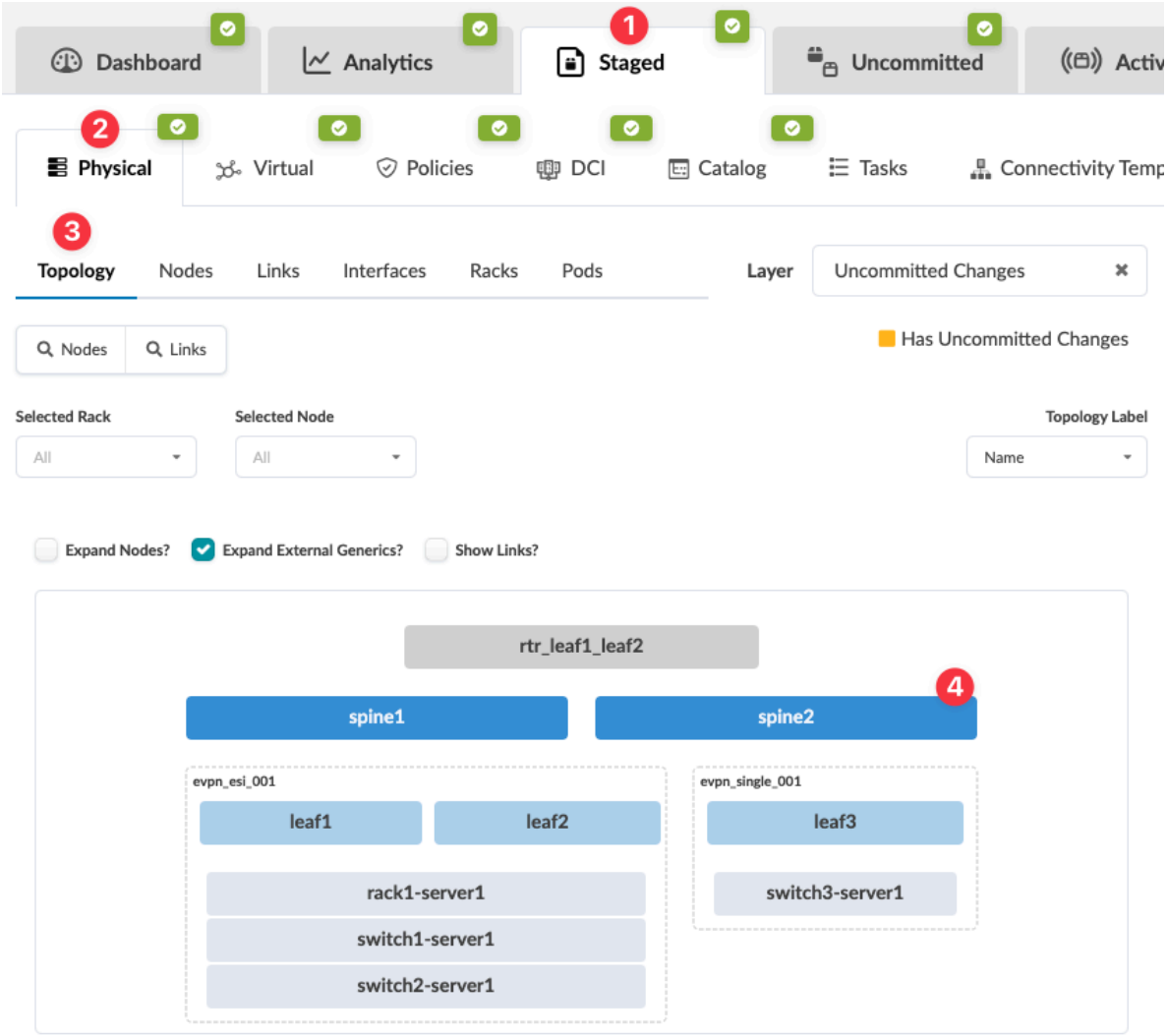
SUMMARY

You can change the hostname and/or name of a single spine or of multiple spines simultaneously.

IN THIS SECTION

Change the Hostname and/or Name of a Single Spine

1. From the blueprint, navigate to **Staged > Physical > Topology** and select the applicable spine.

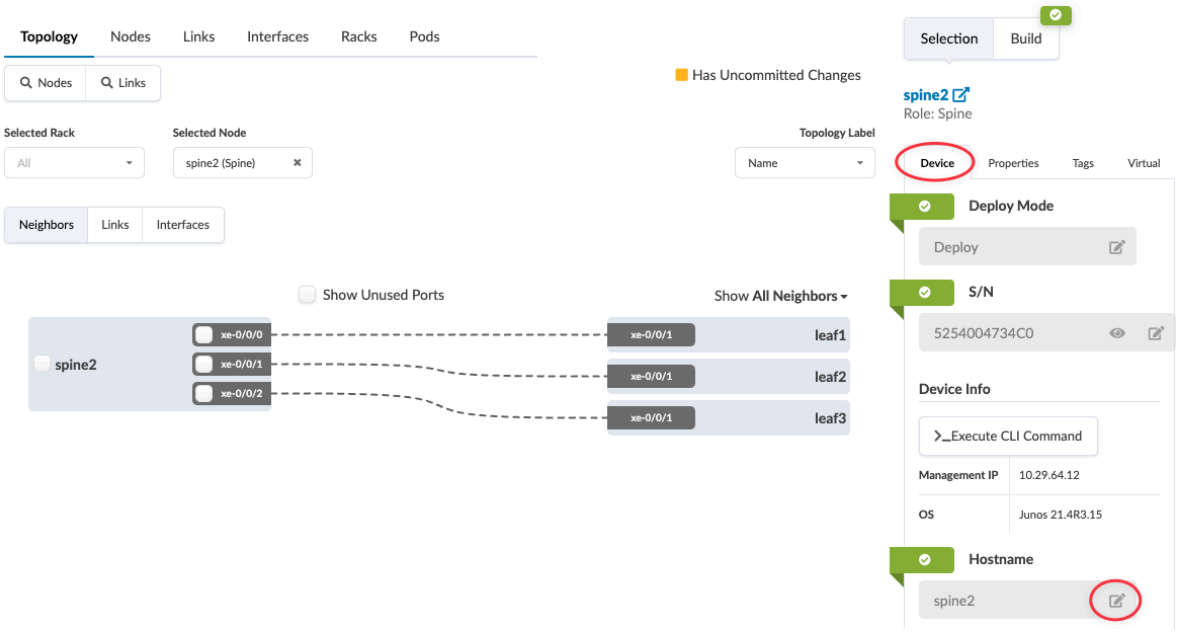


The **Selection** panel opens (on the right).

> **NOTE**: You can also get to the **Selection** panel by clicking the name of the spine in the **Nodes** table (Staged > Physical > Nodes).

2. To change the spine name - From the **Properties** tab in the **Selection** panel, click the **Edit** button for **Name**, change the name accordingly, then click the **Save** button. (Notice you can change other various parameter values from the **Properties** tab.)

3. To change the spine hostname - click the **Device** tab in the **Selection** panel, click the **Edit** button for **Hostname**, change the hostname accordingly, then click the **Save** button. (Notice you can change other various parameter values from the **Device** tab.)

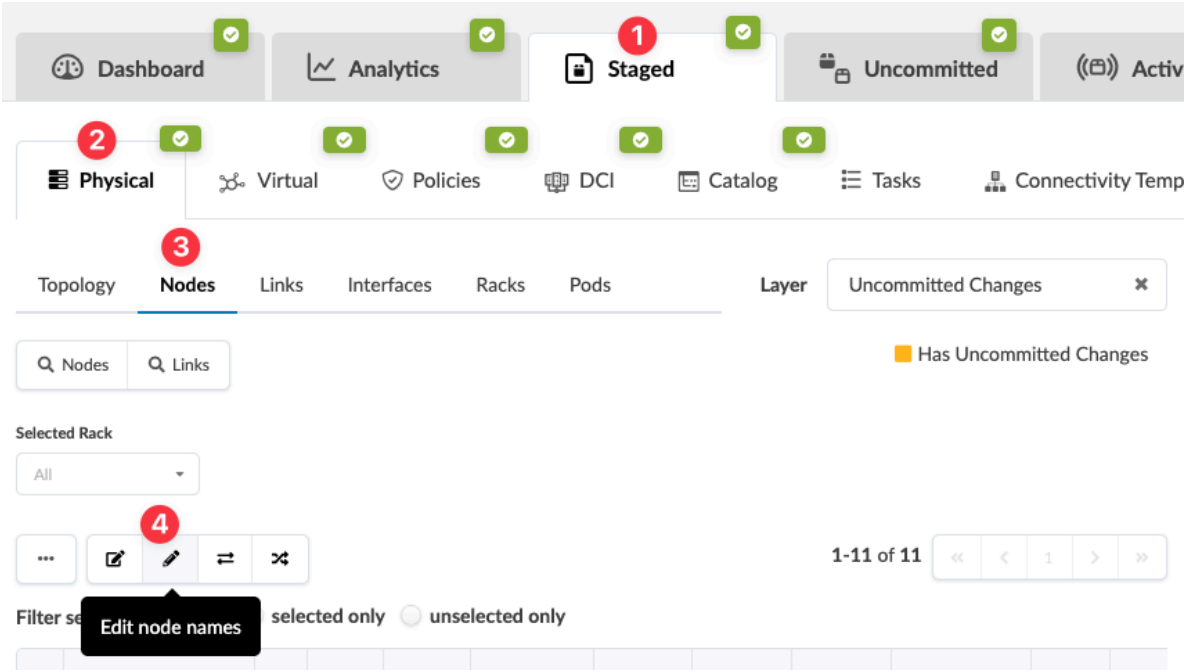


When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

Change Hostnames and/or Names of Multiple Spines

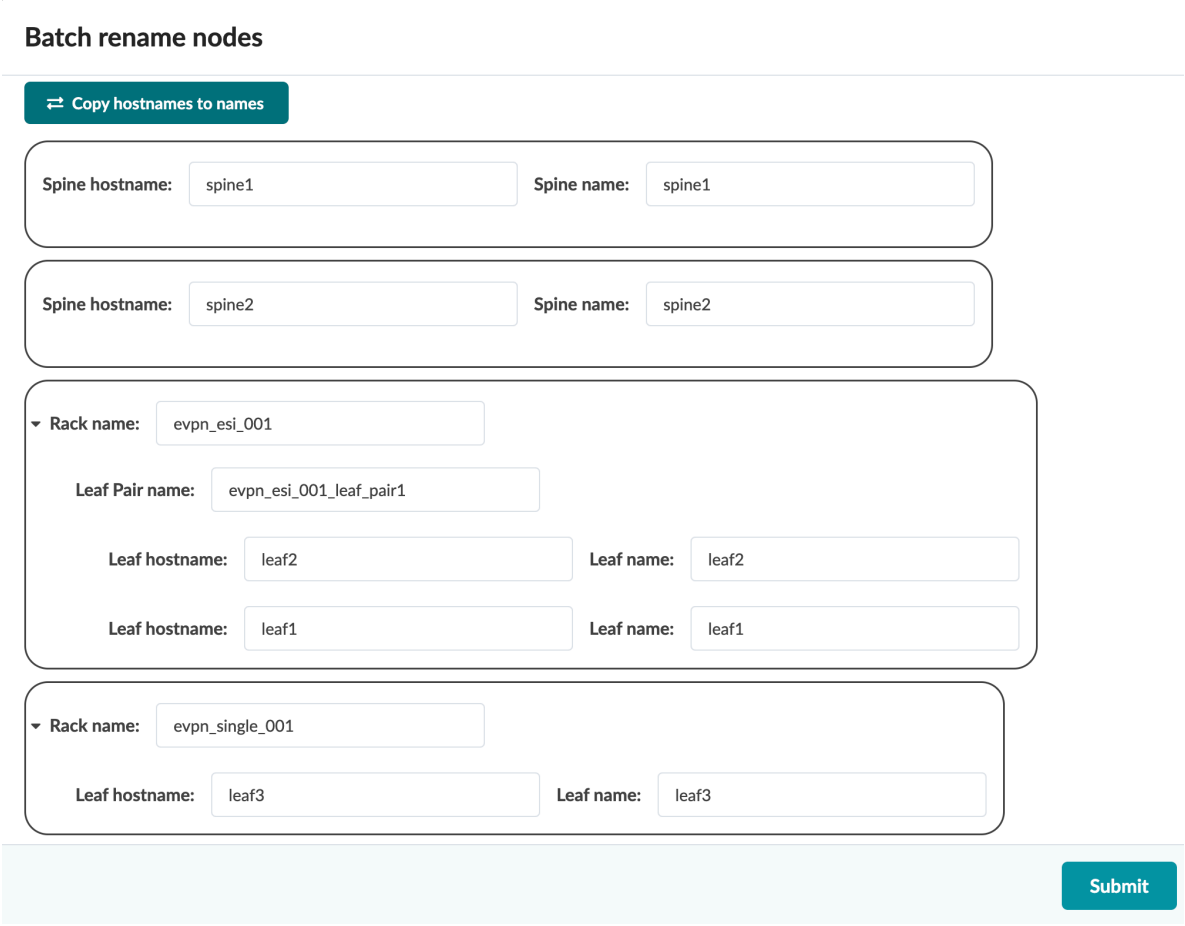
New feature in Apstra version 5.0.0.

1. From the blueprint, navigate to **Staged > Physical > Nodes** and click the **Edit Node Names** button, above the table.



The **Batch rename nodes** dialog opens.

2. Change spine hostnames and/or names. (Notice that you can change other various hostnames and names from this dialog as well.) To efficiently change names to match the hostnames, click **Copy hostnames to names** after you've changed the hostnames. All names in the dialog change to match the hostnames.



3. Click **Submit** to stage the changes and return to the **Nodes** table.

When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

Change Superspine Hostname/Name

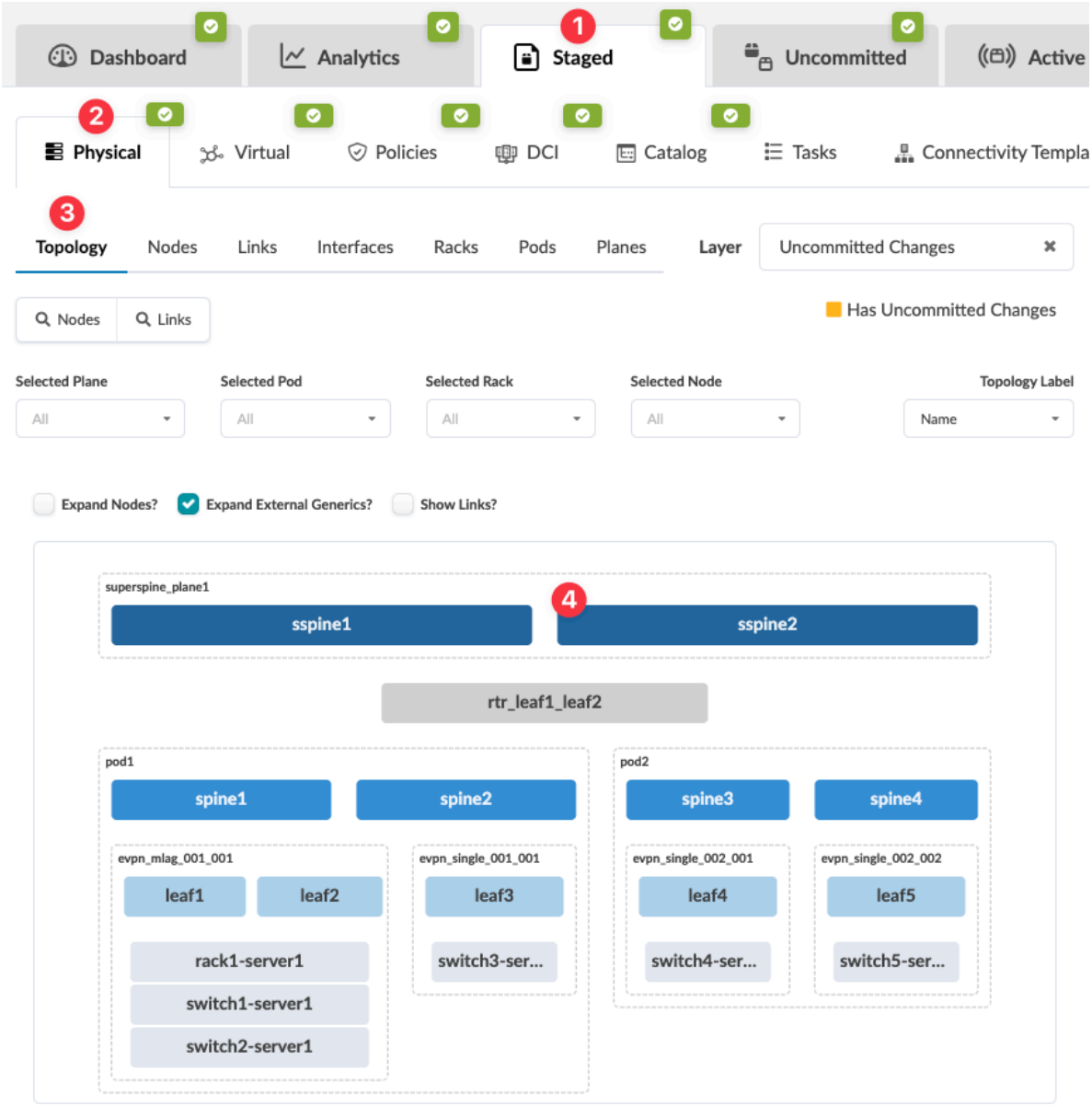
SUMMARY

You can change the hostname and/or name of a single superspine or of multiple superspines simultaneously.

IN THIS SECTION

Change the Hostname and/or Name of a Single Superspine

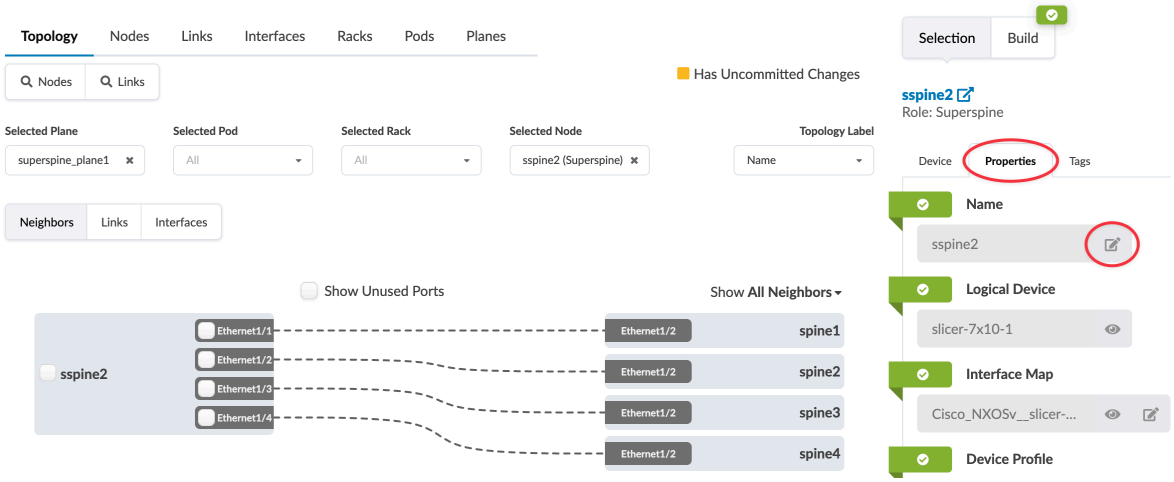
1. From the blueprint, navigate to **Staged > Physical > Topology** and select the applicable superspine.



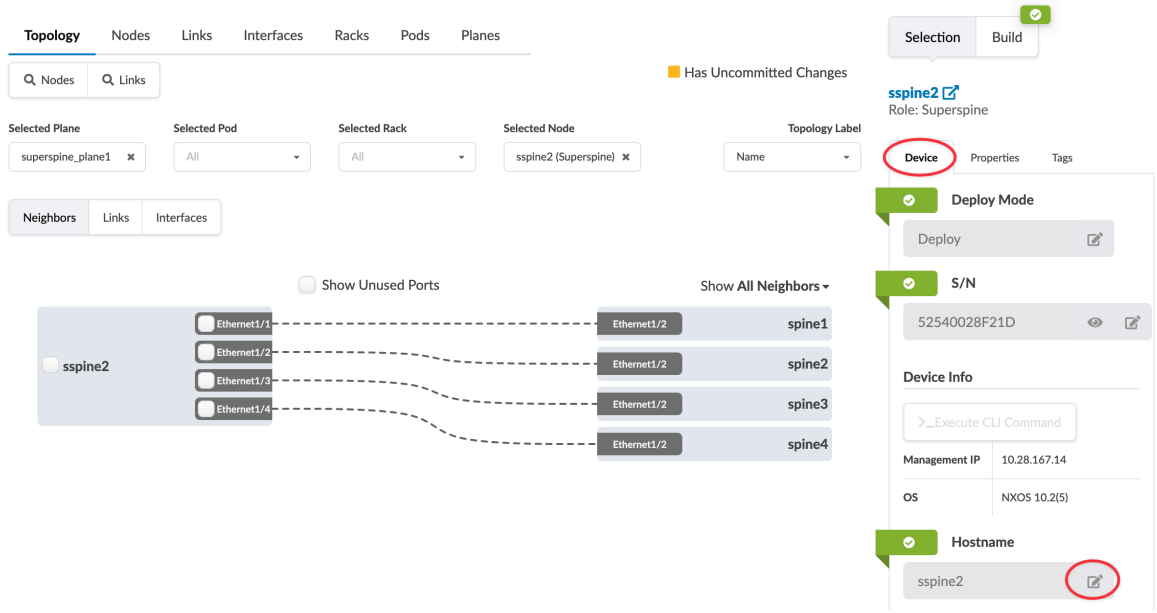
The **Selection** panel opens (on the right).

> **NOTE**: You can also get to the **Selection** panel by clicking the name of the superspine in the **Nodes** table (Staged > Physical > Nodes).

2. To change the superspine name - From the **Properties** tab in the **Selection** panel, click the **Edit** button for **Name**, change the name accordingly, then click the **Save** button. (Notice you can change other various parameter values from the **Properties** tab.)



3. To change the superspine hostname - click the **Device** tab in the **Selection** panel, click the **Edit** button for **Hostname**, change the hostname accordingly, then click the **Save** button. (Notice you can change other various parameter values from the **Device** tab.)

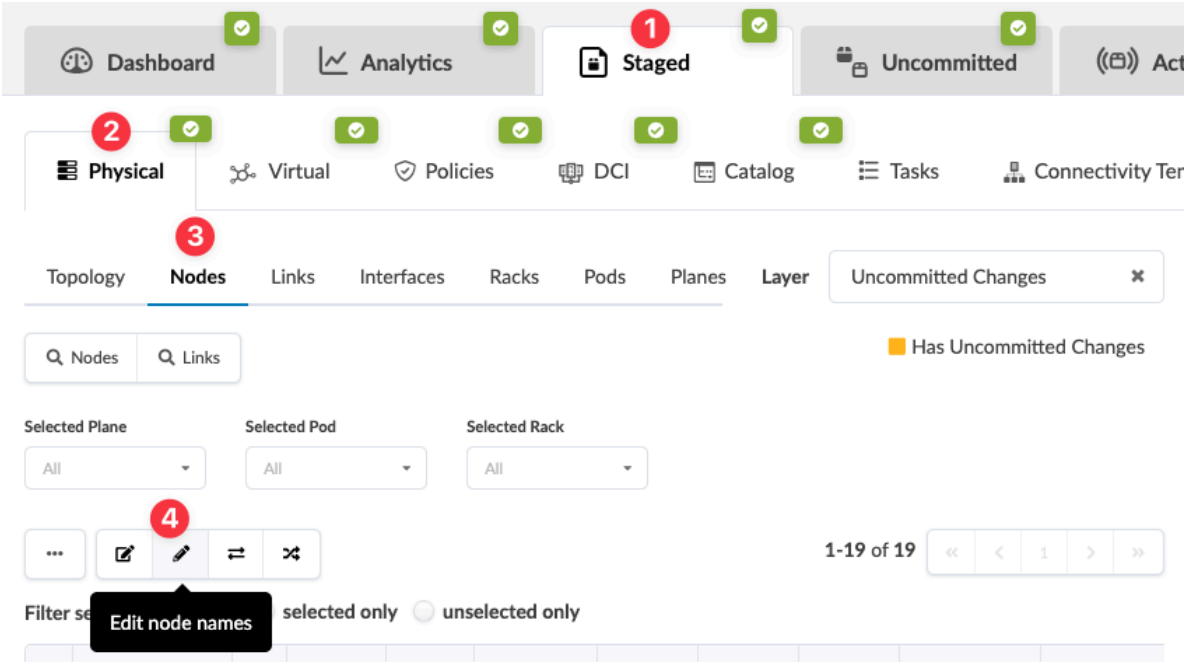


When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

Change Hostnames and/or Names of Multiple Superspines

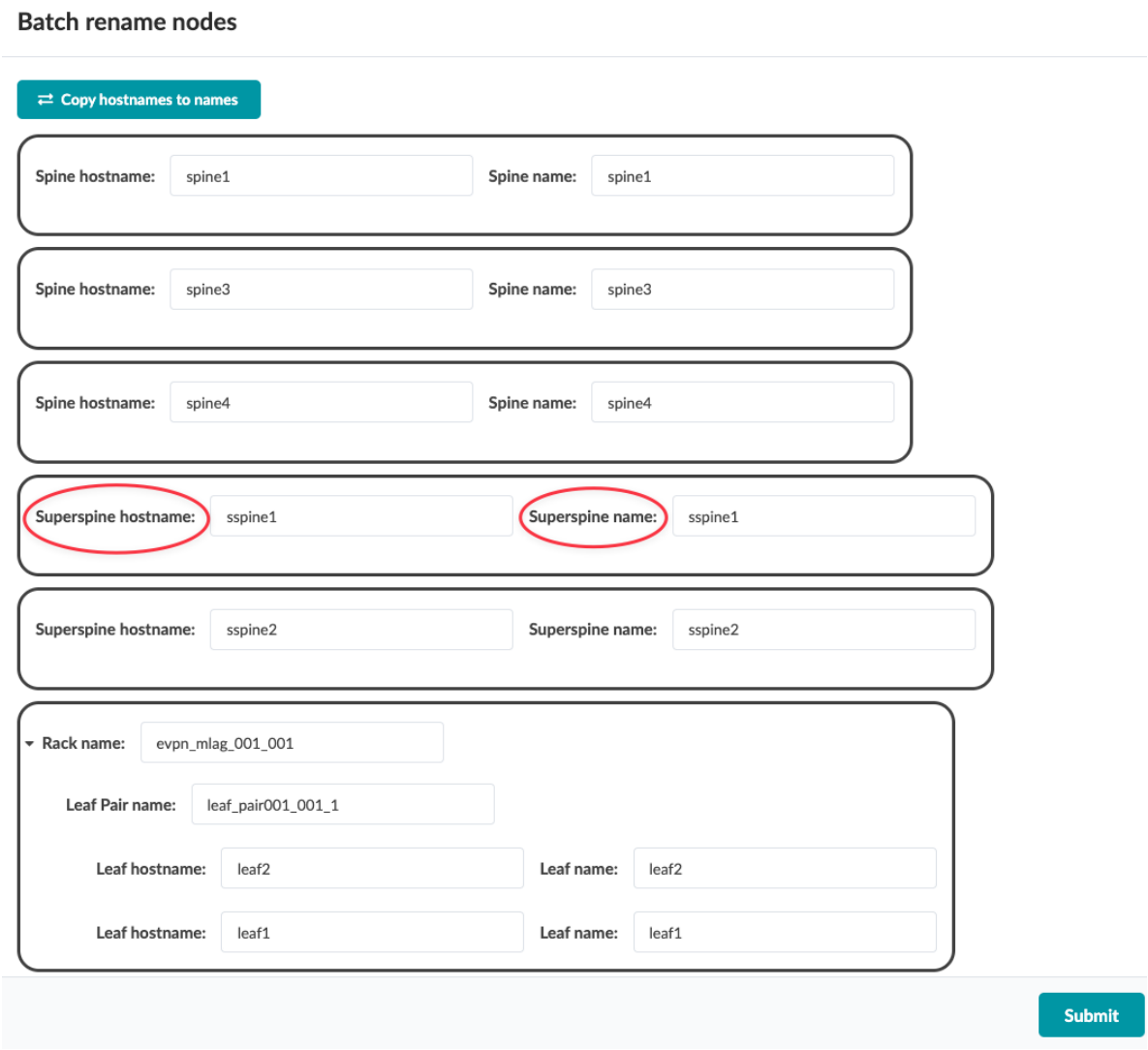
New feature in Apstra version 5.0.0.

1. From the blueprint, navigate to **Staged > Physical > Nodes** and click the **Edit Node Names** button, above the table.



The **Batch rename nodes** dialog opens.

2. Change superspine hostnames and/or names. (Notice that you can change other various hostnames and names from this dialog as well.) To efficiently change the names to match the hostnames, after you've changed the hostnames click **Copy hostnames to names**. All names in the dialog change to match the hostnames.



3. Click **Submit** to stage the changes and return to the **Nodes** table.

When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

Change Generic System Hostname/Name

SUMMARY

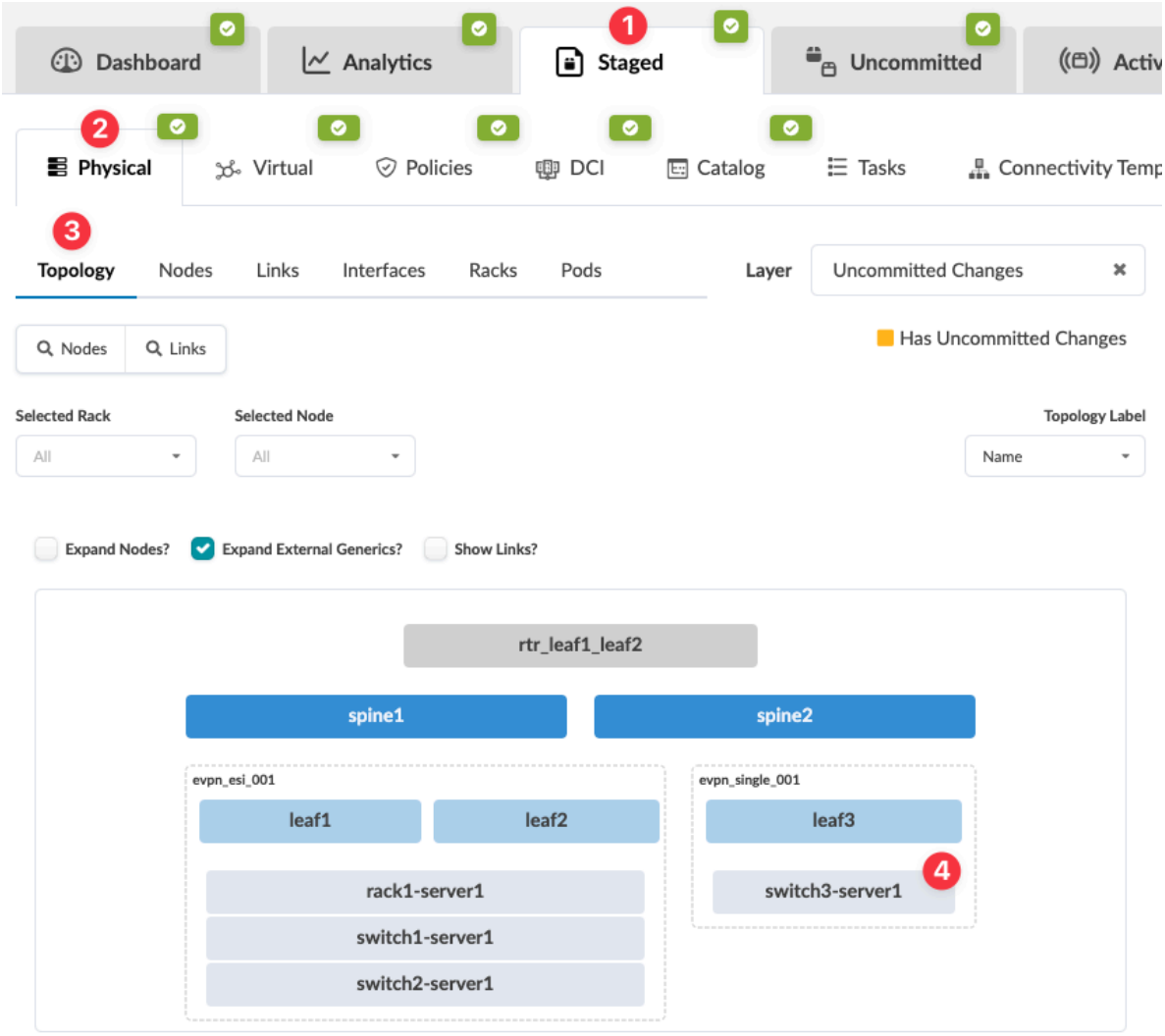
You can change the hostname and/or name of a single generic system or of multiple generic systems simultaneously.

IN THIS SECTION

Change Hostnames and/or Names of Multiple Generic Systems | 108

Change the Hostname and/or Name of a Single Generic System

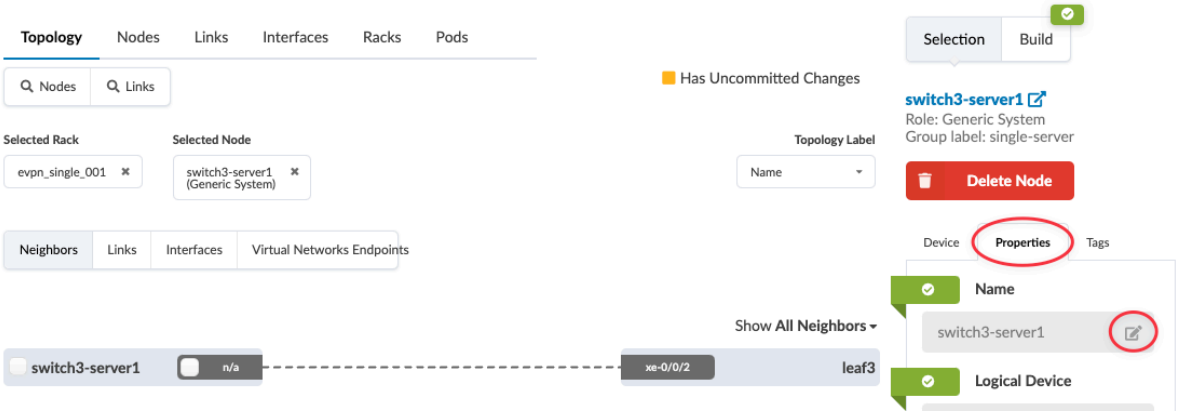
1. From the blueprint, navigate to **Staged > Physical > Topology** and select the applicable generic system.



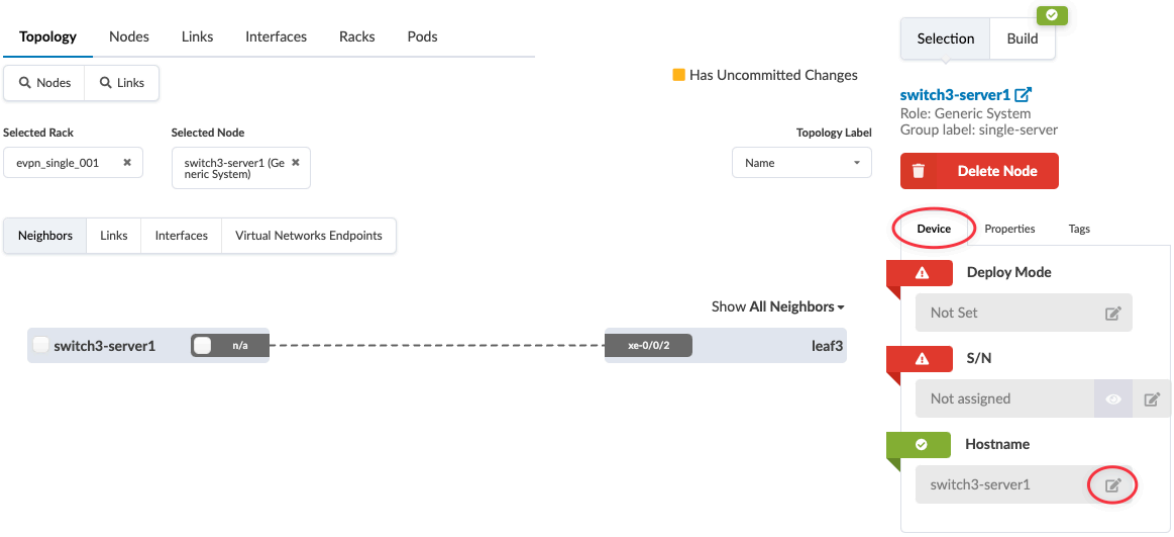
The **Selection** panel opens (on the right).

NOTE: You can also get to the **Selection** panel by clicking the name of the generic system in the **Nodes** table (Staged > Physical > Nodes).

2. To change the generic system name - From the **Properties** tab in the **Selection** panel, click the **Edit** button for **Name**, change the name accordingly, then click the **Save** button. (Notice you can change other various parameter values from the **Properties** tab.)



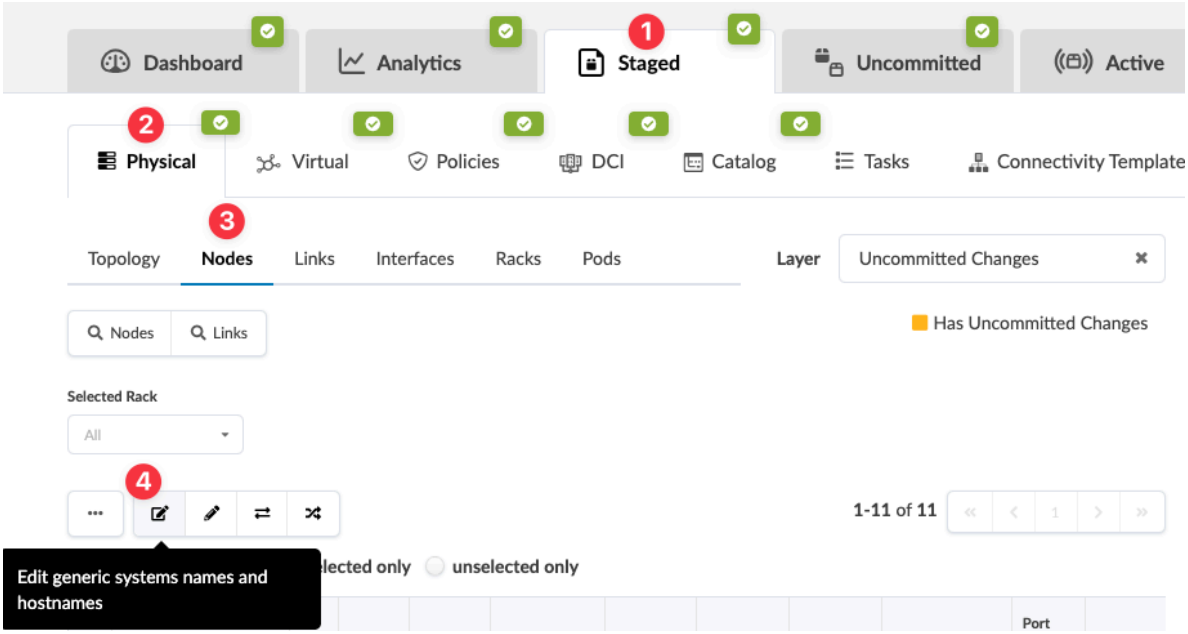
3. To change the generic system hostname - click the **Device** tab in the **Selection** panel, click the **Edit** button for **Hostname**, change the hostname accordingly, then click the **Save** button. (Notice you can change other various parameter values from the **Device** tab.)



Any associated link names do not automatically update to match the changed server names and/or hostnames. You can manually "change the link names" on page 199 to match so when you're reviewing an updated cabling map the names align. When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

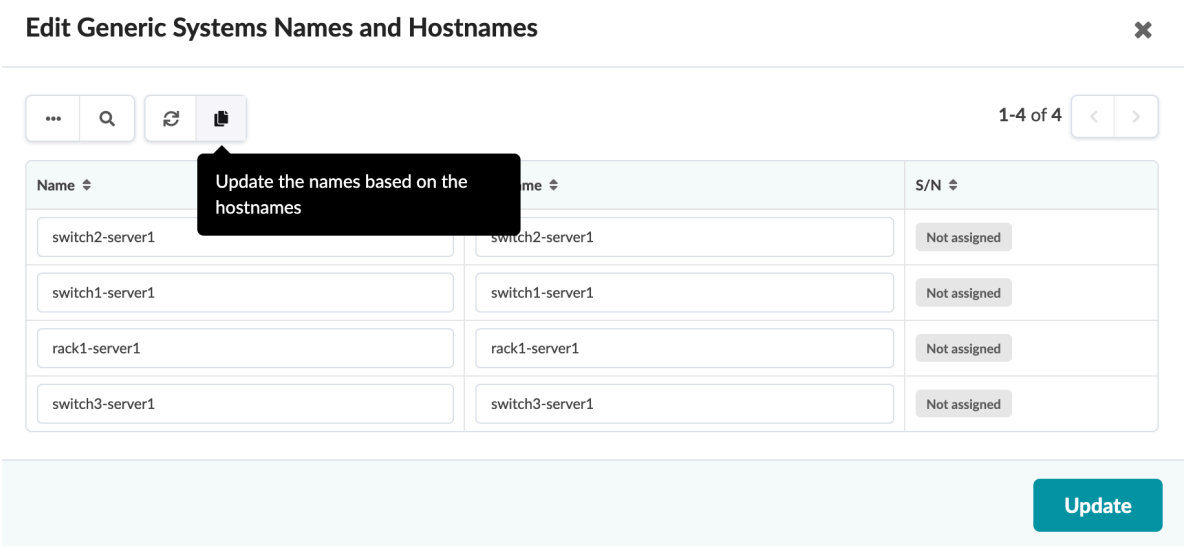
Change Hostnames and/or Names of Multiple Generic Systems

1. From the blueprint, navigate to **Staged > Physical > Nodes** and click the **Edit generic system names and hostnames** button, above the table.



The **Edit Generic Systems Names and Hostnames** dialog opens.

2. Change generic system names and/or hostnames. To efficiently change names to match the hostnames, after you've changed the hostnames click the **Update the names based on the hostnames** button. All names in the dialog change to match the hostnames.



3. Click **Update** to stage the changes and return to the **Nodes** table.

Any associated link names do not automatically update to match the changed server names and/or hostnames. You can manually "change the link names" on page 199 to match so when you're reviewing an updated cabling map the names align. When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

Change Assigned Interface Map

If you change the device profile that's assigned to a device (via the interface map) to one with a different selector field (manufacturer, model, OS family, version) the device is automatically undeployed and unassigned; you'll need to reassign and deploy the device. If the selector fields are unchanged, you can change the device profile without the device being undeployed and unassigned (as of Apstra version 5.0.0), but you will still need to change the device profile in **Managed Devices**.

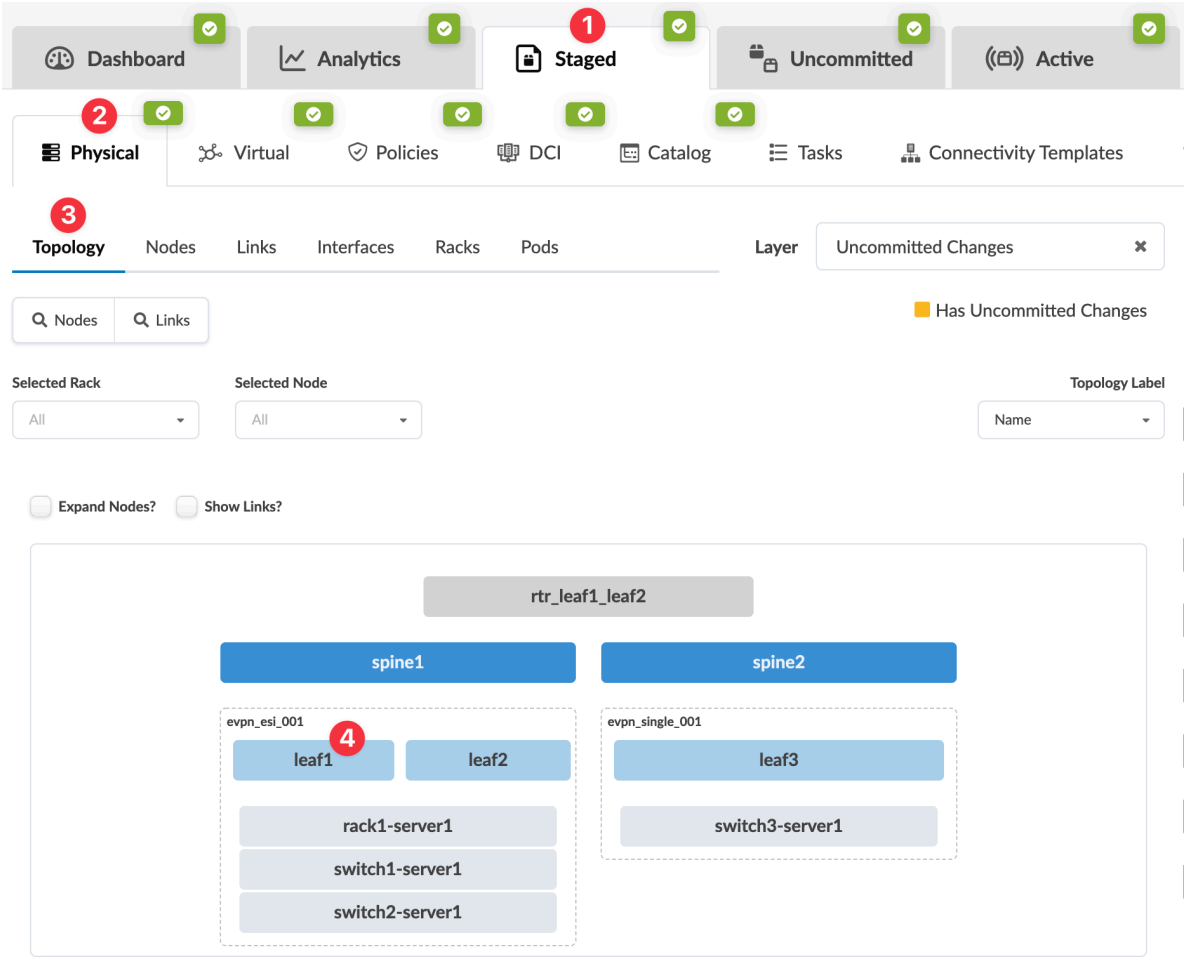
Some examples of when you might change the device profile without changing the selector fields are the following:

- To change the spine uplinks from the right side of the cabinet to the left side of the cabinet
- To change transformations

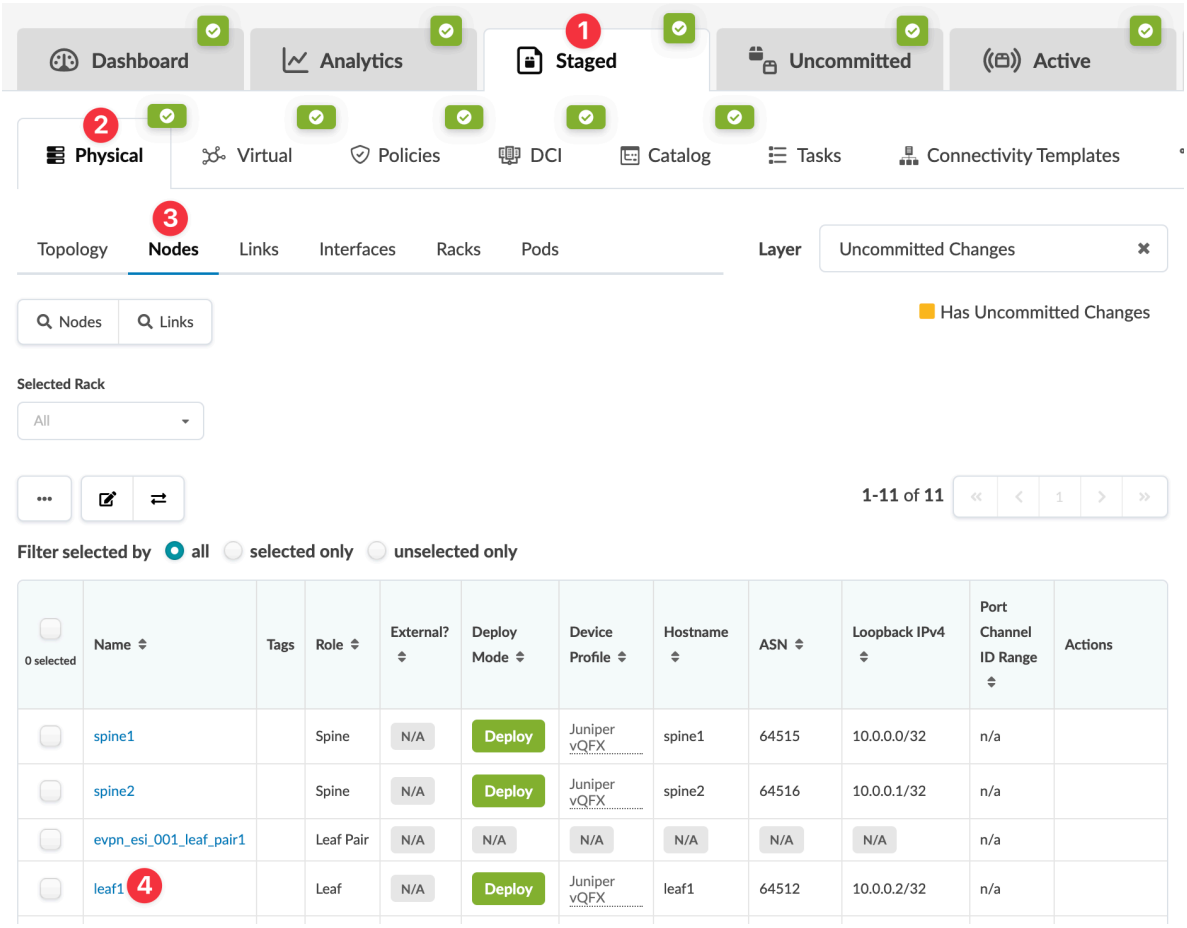
> **NOTE**: If you want to completely change a device that's used in your network, follow the steps for "removing a device" on page 673 and "adding a device" on page 651.

1. From the blueprint, navigate to **Staged > Physical**, then either from the **Topology** view or the **Nodes** table, select the device with the device profile to change:

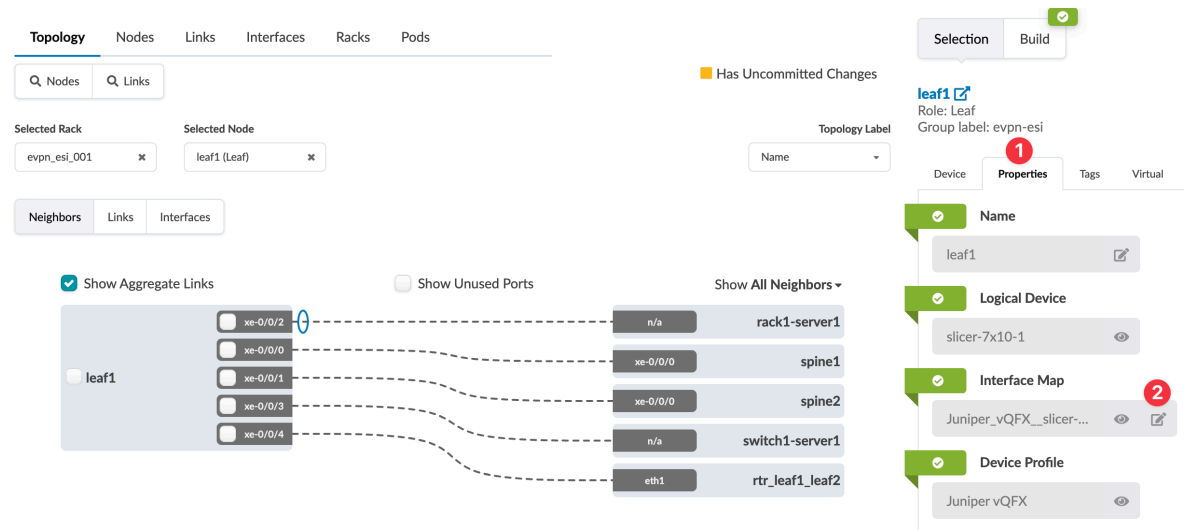
From Topology View



From Nodes Table



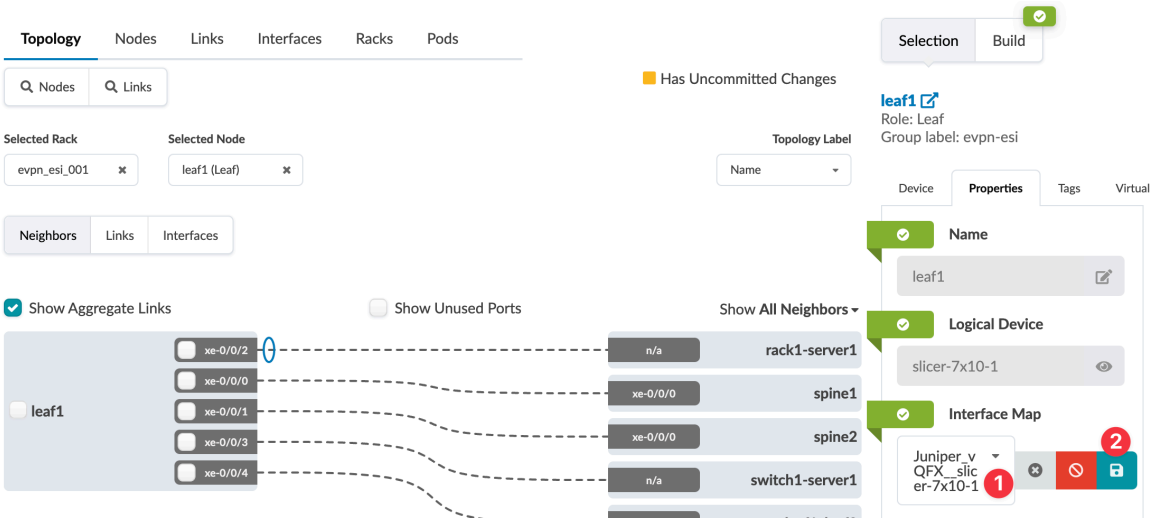
2. In the **Selection** panel on the right, click the **Properties** tab, then click the **Edit** button for **Interface Map**.



3. Select the new interface map from the drop-down list.

The drop-down list includes interface maps that have been "imported" on page 439 to the blueprint catalog that match the logical device. If you don't see the interface map you need, "import" on page 439 it from the **Design** (global) catalog, then come back here and it will be included in the list. If it's not in the **Design** catalog, you can "create the interface map" on page 856, then import it.

4. Click the **Save** button to stage the change.



The device profile used in the blueprint needs to match the device profile in the **Managed Devices** table. Before you commit, "change the assigned device profile in the **Managed Devices** table" on page 669 to match the device profile you just changed, or a build error will occur.

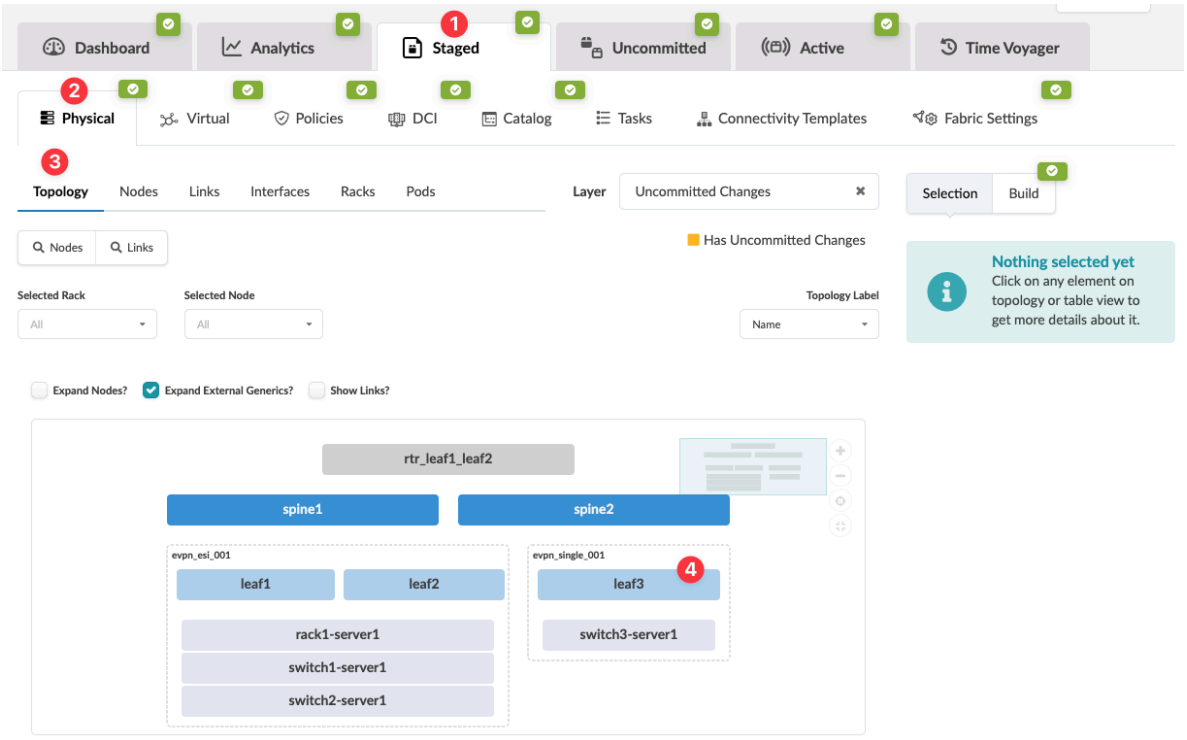
Change Assigned ASN (Datacenter)

SUMMARY

Assign or change an individual ASN

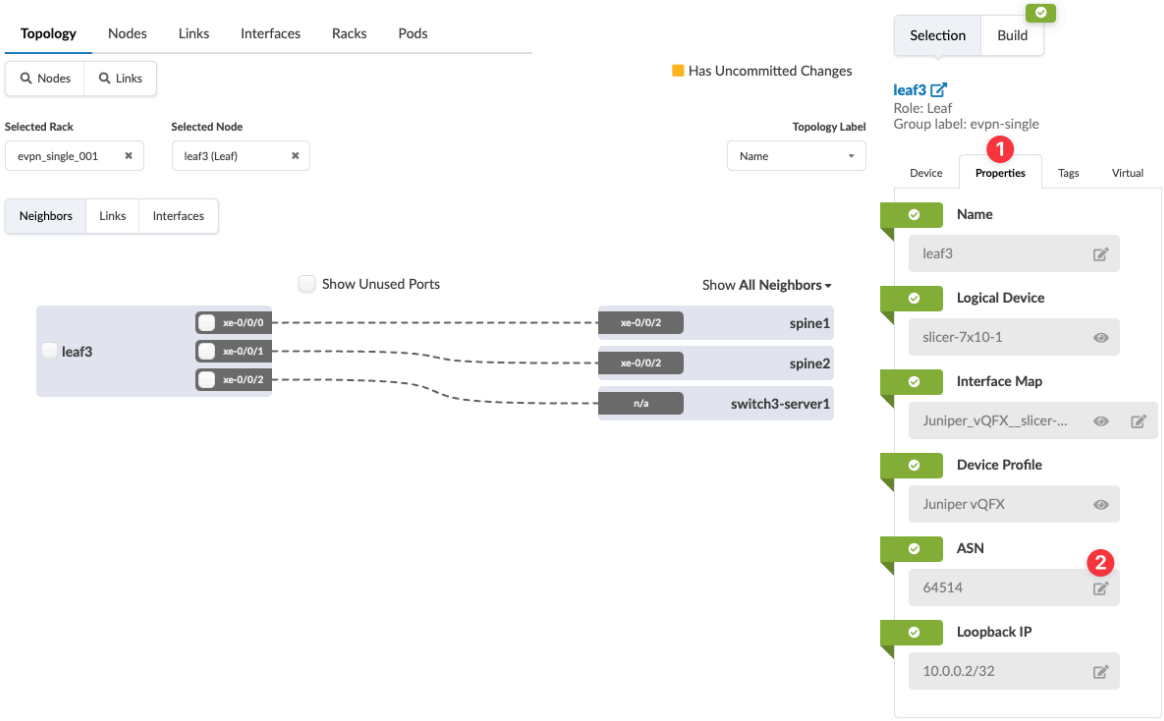
Normally, you would let Apstra "pull ASNs from resource pools" on page 58 for you, but there may be times when you need to assign a specific ASN or change one that was already assigned. In these cases, you can select the device and assign or change the ASN from the **Properties** tab. See below for details.

1. From the blueprint, navigate to **Staged > Physical > Topology** and select the device to update. (Alternatively, you can navigate to **Staged > Physical > Nodes** and select the device from the **Nodes** table.)



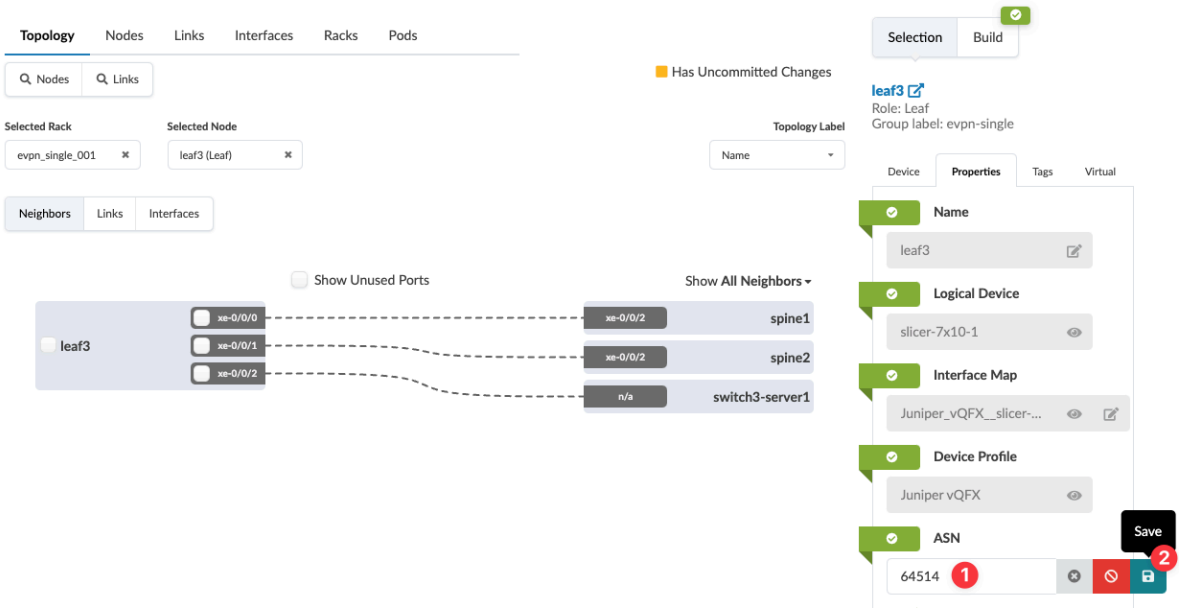
The **Selection** panel on the right becomes active and populates with information about the selection.

2. Click the **Properties** tab in the **Selection** panel, then click the **Edit** button for the **ASN** field.



The **ASN** field becomes available to change.

3. Enter the new ASN value, then click the **Save** button to stage your changes.



When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

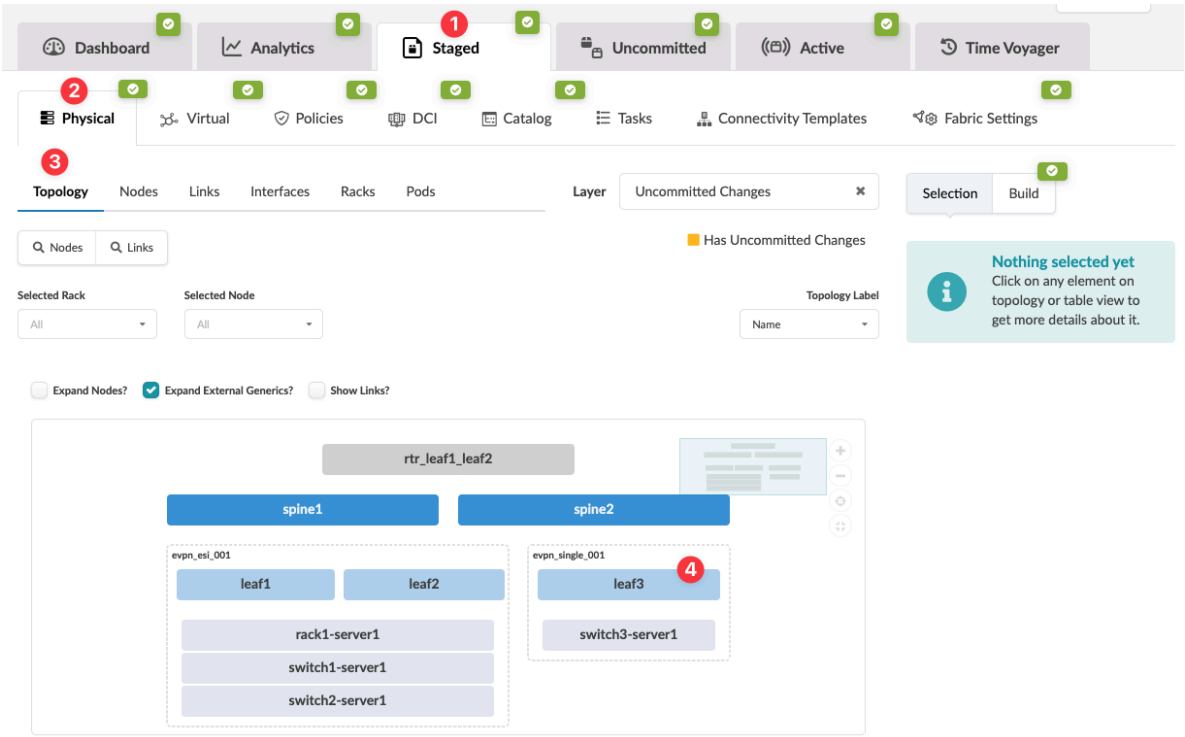
Change Assigned Loopback IP Address (Datacenter)

SUMMARY

Assign or change an individual loopback IP Address

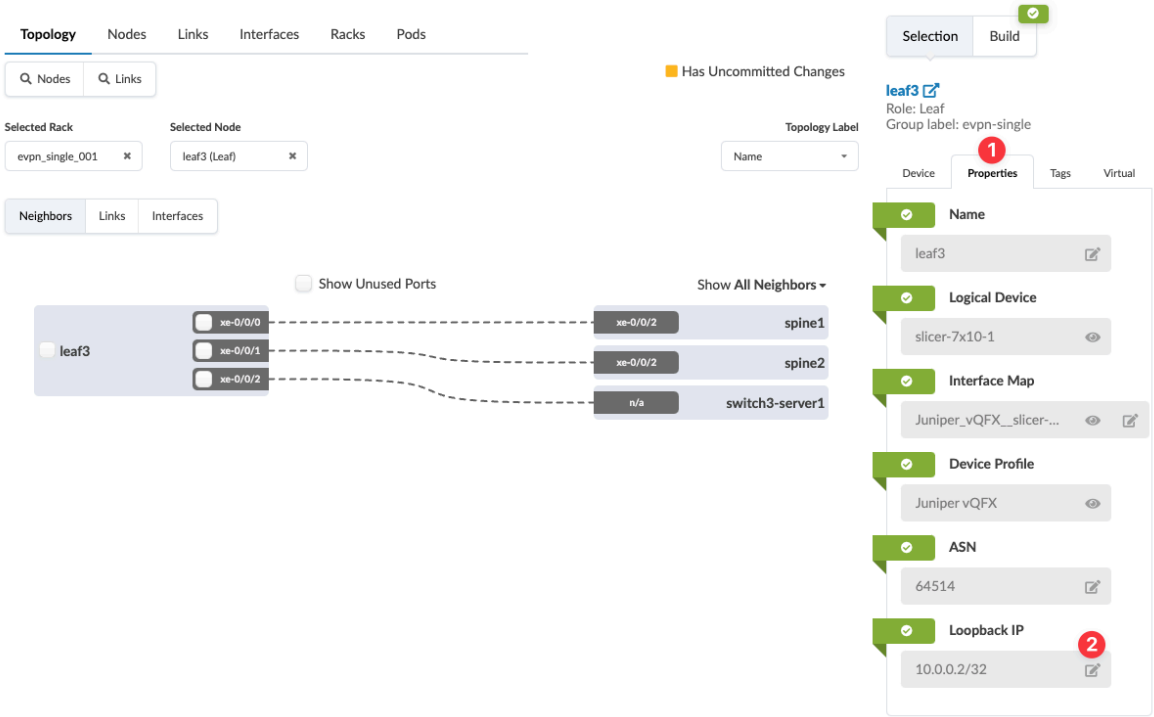
Normally, you would let Apstra "pull IP addresses from resource pools" on page 58 for you, but there may be times when you need to assign a specific IP address or change one that was already assigned. In these cases, you can select the device and assign or change the IP address from the **Properties** tab. See below for details.

1. From the blueprint, navigate to **Staged > Physical > Topology** and select the device to update. (Alternatively, you can navigate to **Staged > Physical > Nodes** and select the device from the **Nodes** table.)



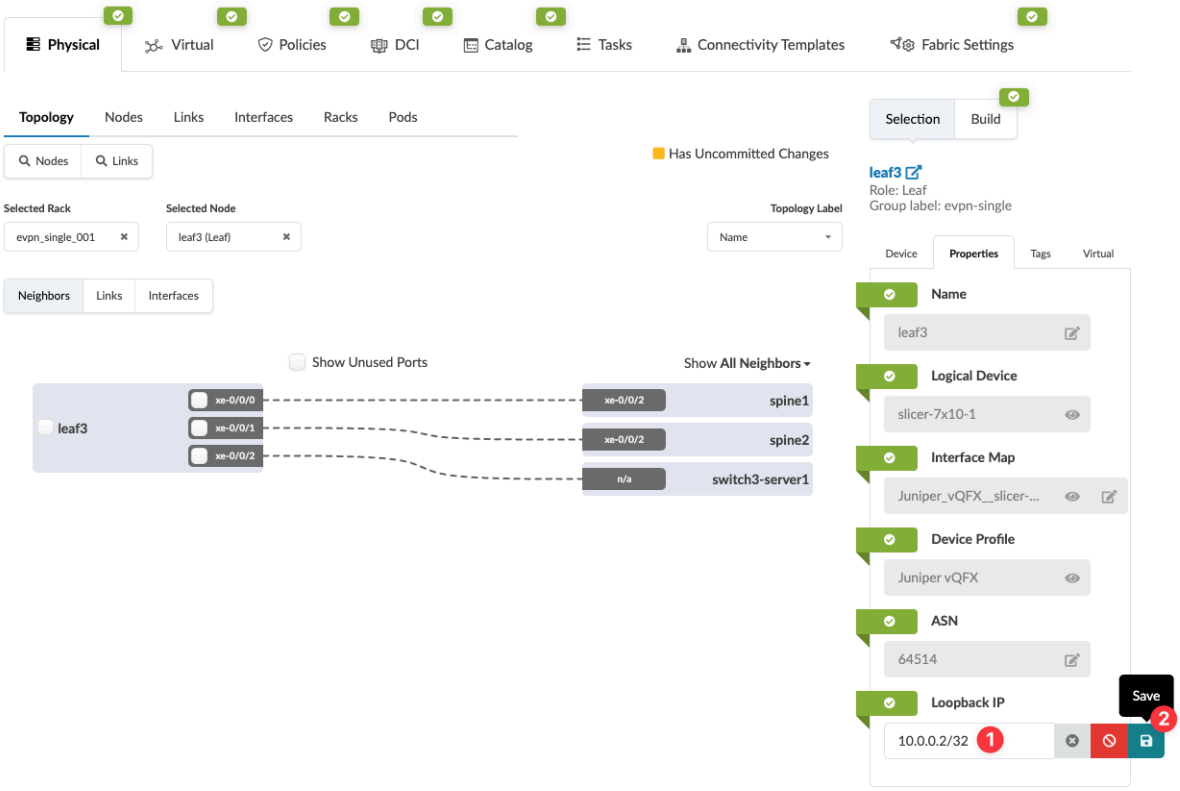
The **Selection** panel on the right becomes active and populates with information about the selection.

2. Click the **Properties** tab in the **Selection** panel, then click the **Edit** button for the **Loopback IP** field.



The **Loopback IP** field becomes available to change.

3. Enter the new Loopback IP address value, then click the **Save** button to stage your changes.



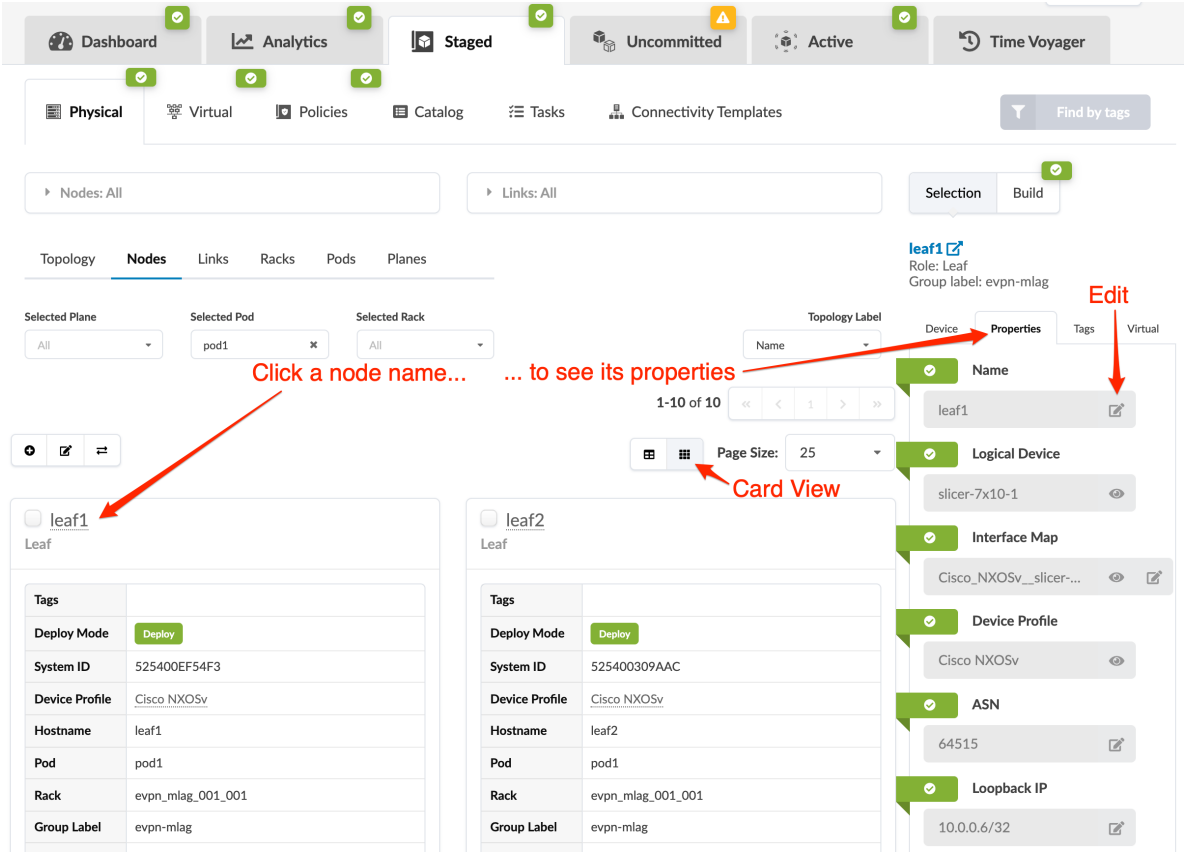
When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

Edit Device Properties (Datacenter)

You can change device properties such as name, interface map, ASN, and loopback IP, depending on the node chosen.

1. From the blueprint, navigate to **Staged > Physical > Nodes** and select a node name (not the check box). You can narrow your search with the drop-down lists for planes, pods, racks and access groups, as applicable.

2. Click the **Properties** tab in the right panel.



3. You can change device properties such as name (must be changed to a unique name), interface map, ASN, and loopback IP, depending on the node chosen. The attributes that can be edited have an **Edit** button associated with them. Change properties as applicable.

> **NOTE**: If you changed leaf names in a leaf pair, the leaf pair name does not change. You can manually change the leaf pair name to correspond with the new leaf names. This is especially useful when assigning leaf pairs when you create virtual networks.

4. Click the **Save** button to stage the changes.

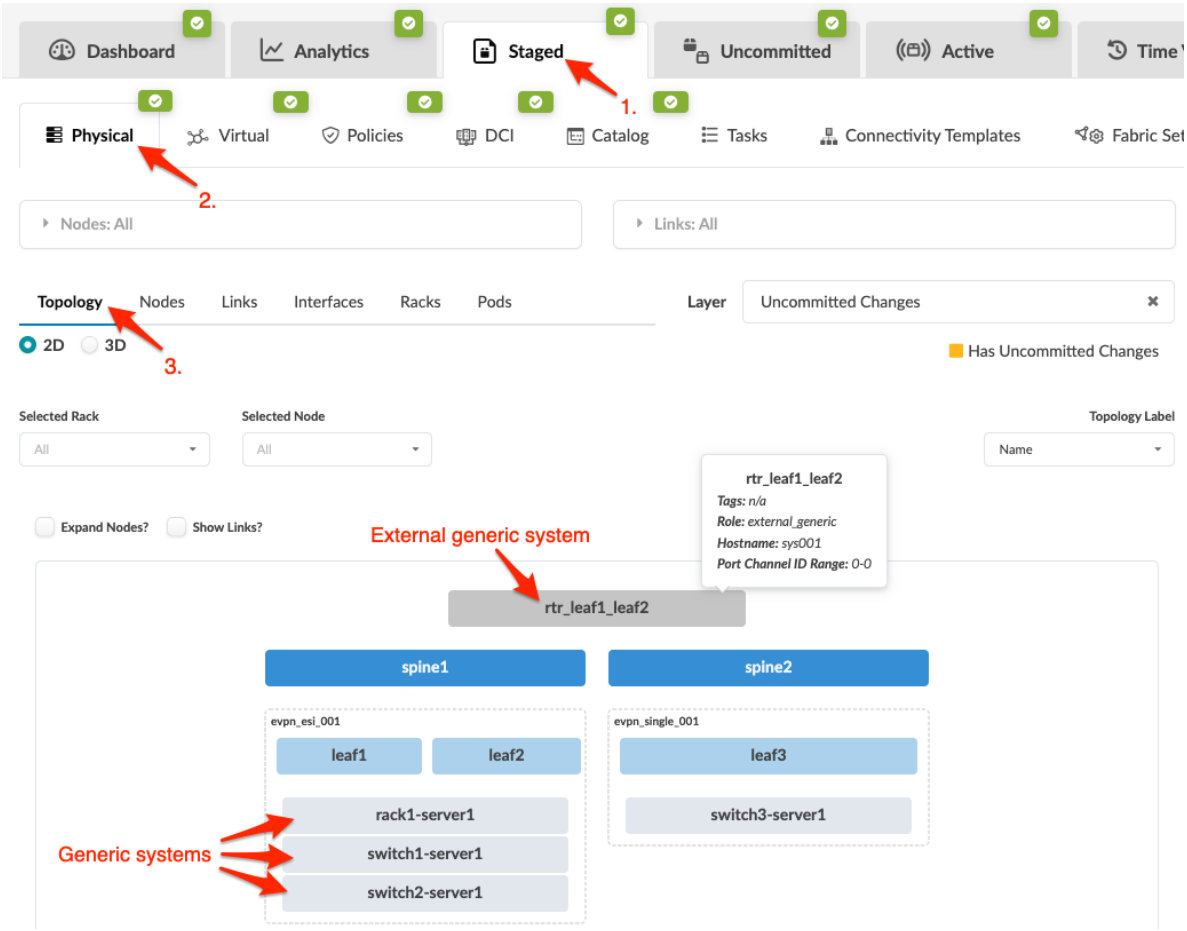
Update Port Channel ID Range

IN THIS SECTION

> ⚠ **CAUTION**: Changing port channel range is an invasive operation and may lead to reassigning existing port channel IDs.

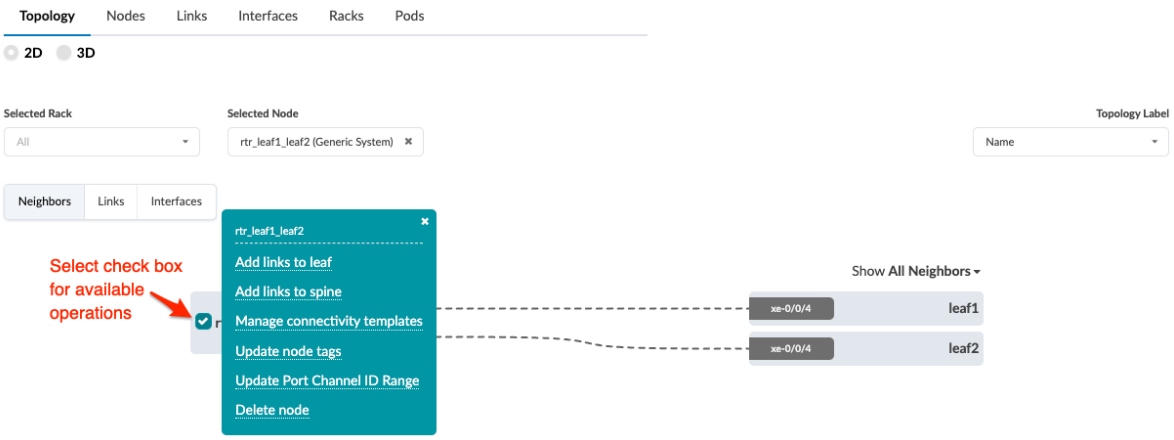
Update Port Channel ID Range (from Topology View)

1. From the blueprint, navigate to **Staged > Physical > Topology** and select the generic system or external generic system to update.



2. To see the current port channel ID range (and other details) hover over the system.

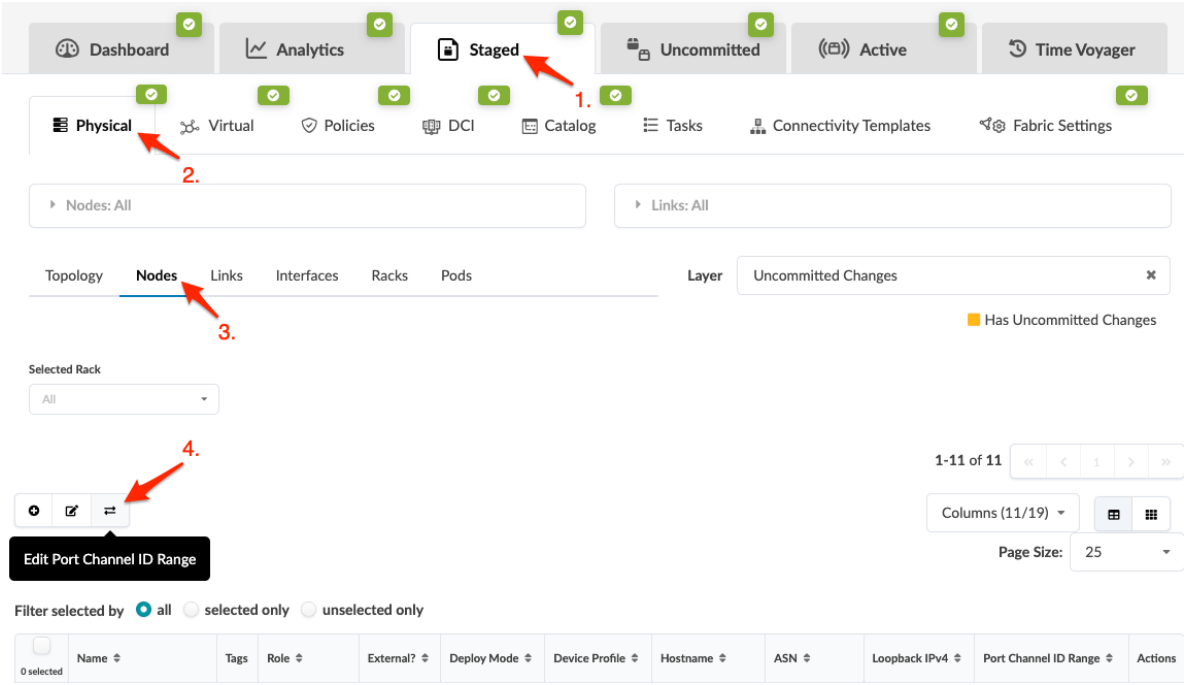
3. Select the check box for the system to see operations available for that system (and that you have permissions for).



4. Click **Update Port Channel ID Range** and edit the min and/or max values, as needed. Non-default port channel ranges need only be unique per *system*, not per *blueprint*.
5. Click **Update** to stage your changes and return to the **Topology** view.

Update Port Channel ID Range (from Nodes view)

1. From the blueprint, navigate to **Staged > Physical > Nodes** and click the **Edit Port Channel ID Range** button.

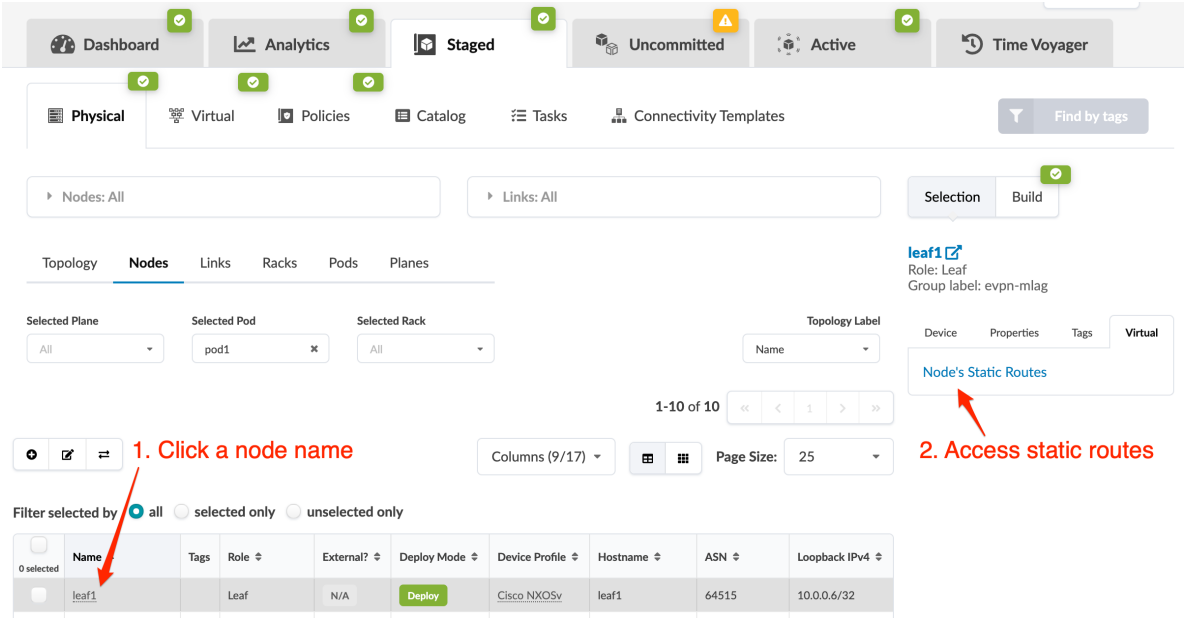


2. In the table of generic systems and external generic systems, edit the min and/or max port channel ID values, as needed. Non-default port channel ranges need only be unique per *system*, not per *blueprint*.
3. Click **Update** to stage your changes and return to the **Nodes** view.

View Node's Static Routes

1. From the blueprint, navigate to **Staged > Physical > Nodes** and select a node name (not the check box). (You can narrow your search with the drop-down lists for planes, pods, and racks as applicable, as of Apstra version 4.0.)

2. Click the **Nodes** tab in the right panel.



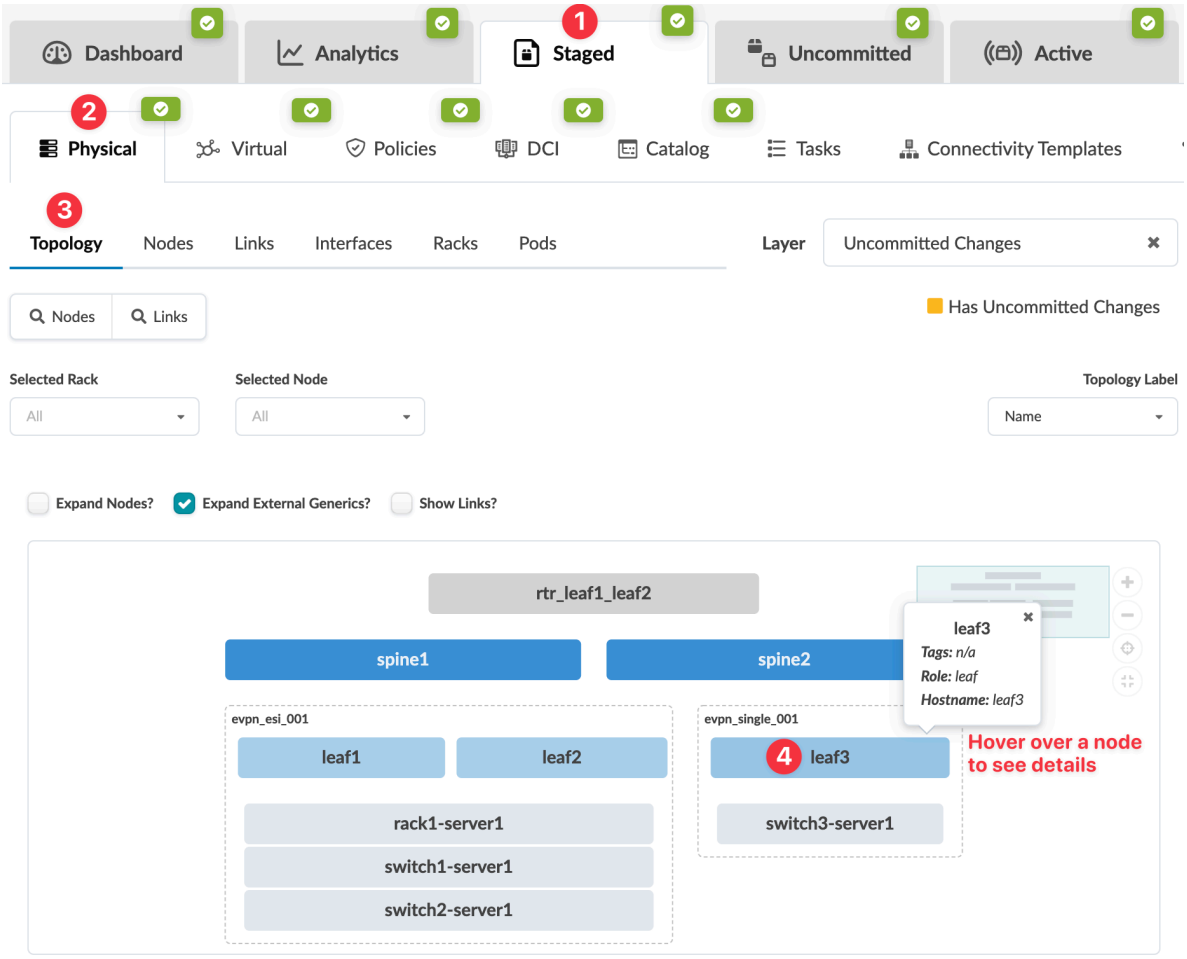
3. Click **Node's Static Routes** to go to **Staged > Virtual > Static Routes** where you can see that node's static routes.

Update Tags on Node (Datacenter)

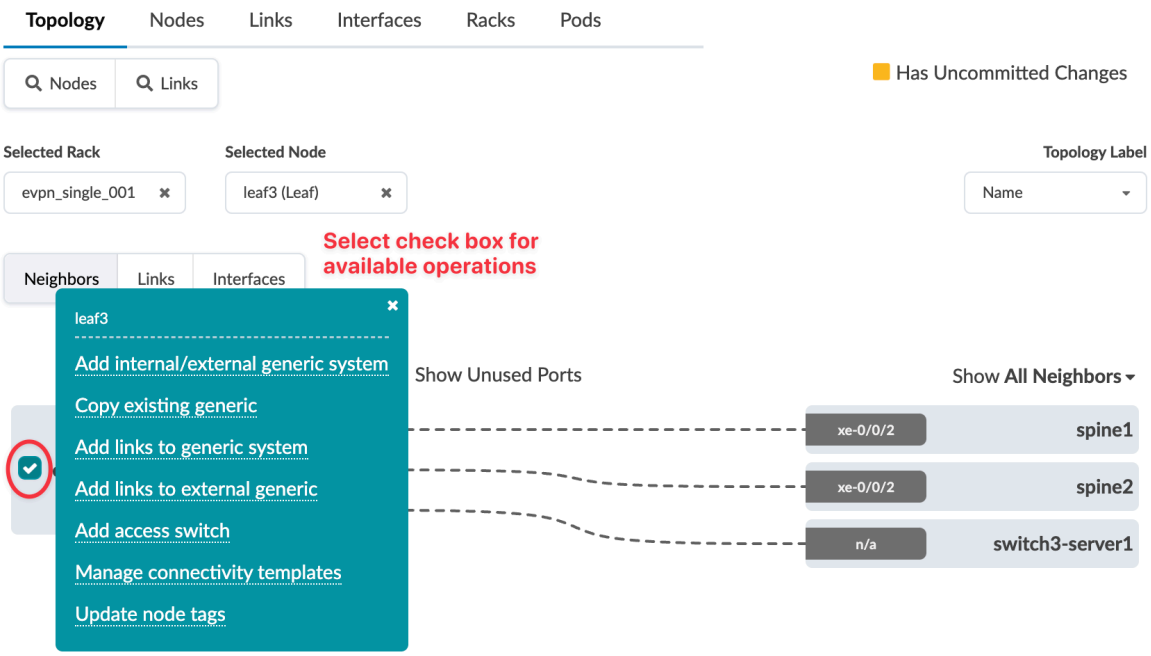
IN THIS SECTION

Update Node Tags (One Node)

1. From the blueprint, navigate to **Staged > Physical > Topology** and select the node that needs updated tags.



2. Select the node check box to see the operations available for that node (and that you have permissions for).

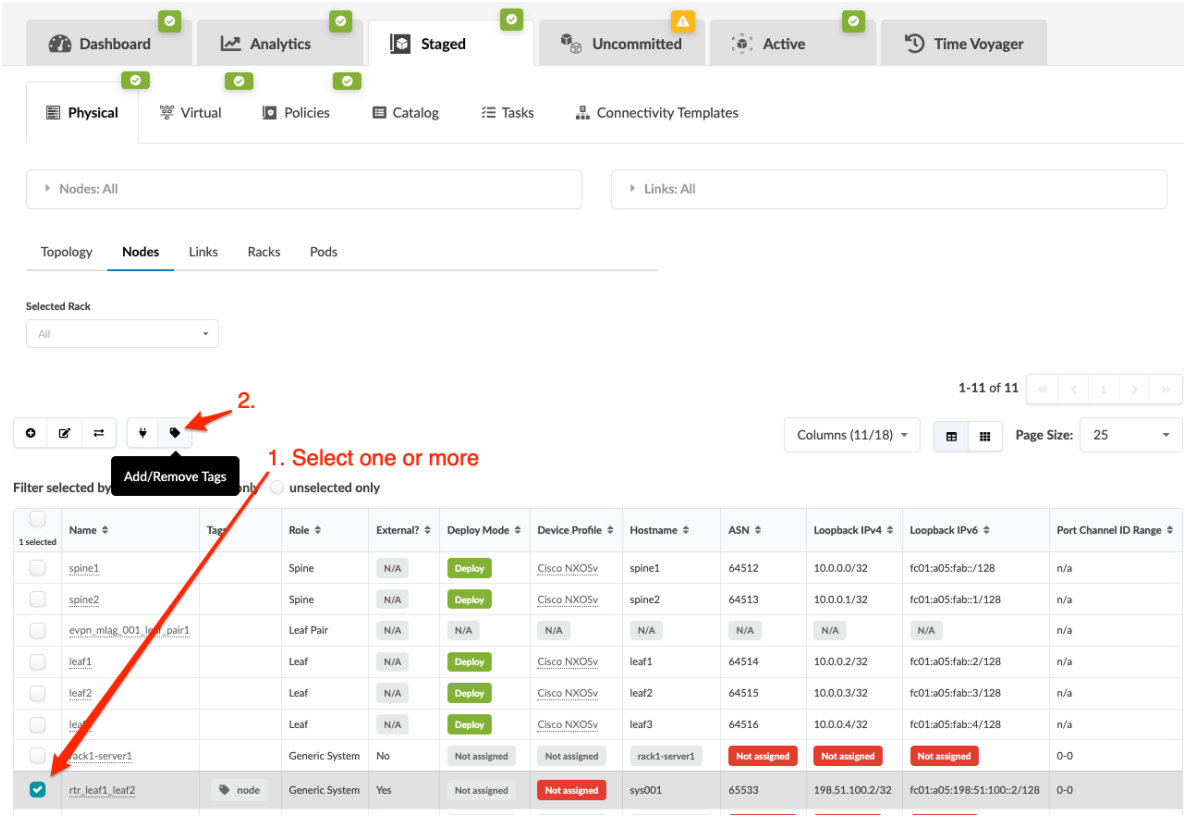


3. Click **Update node tags** and update node tags as needed.
4. Click **Update** to update the tags and return to the **Selection** view.

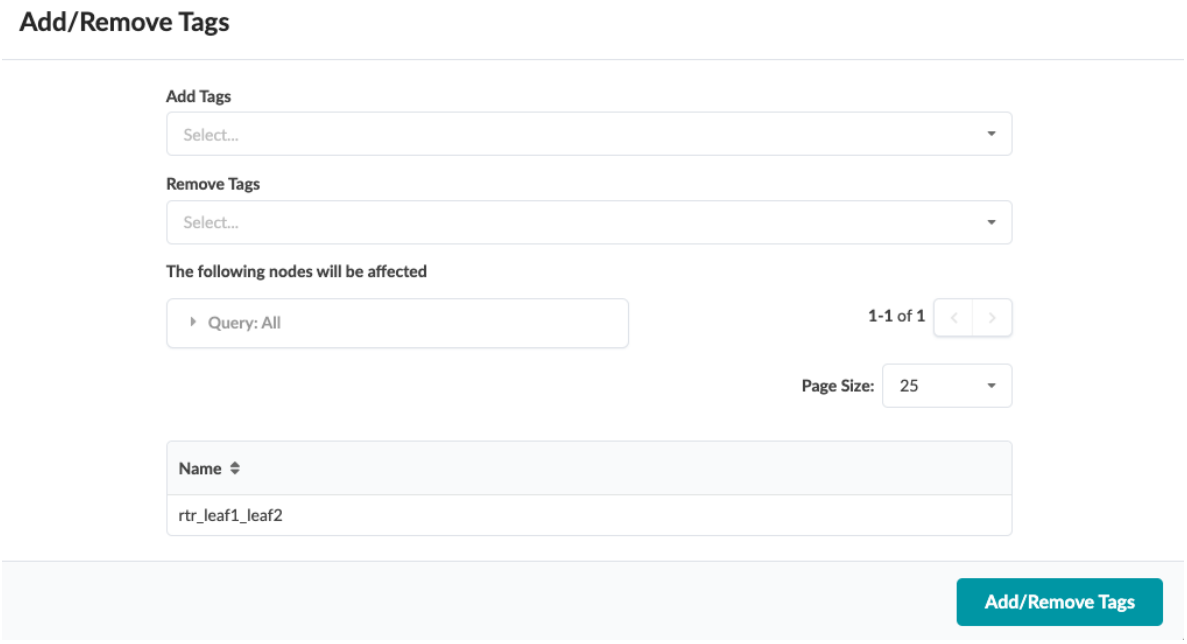
When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Update Node Tags (Multiple Nodes)

1. From the blueprint, navigate to **Staged > Physical > Nodes** and select one or more check boxes for the node(s) that need updated tags. The **Add/Remove Tags** button appears above the table.



2. Click the **Add/Remove Tags** button and update tags as needed. When you create new tags here they are added to the blueprint catalog.



3. Click **Add/Remove Tags** to stage the change and return to the **Nodes** view.

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Generic Systems (Internal/External)

IN THIS SECTION

Generic Systems vs. External Generic Systems

When to use a generic system and when to use an external generic system:

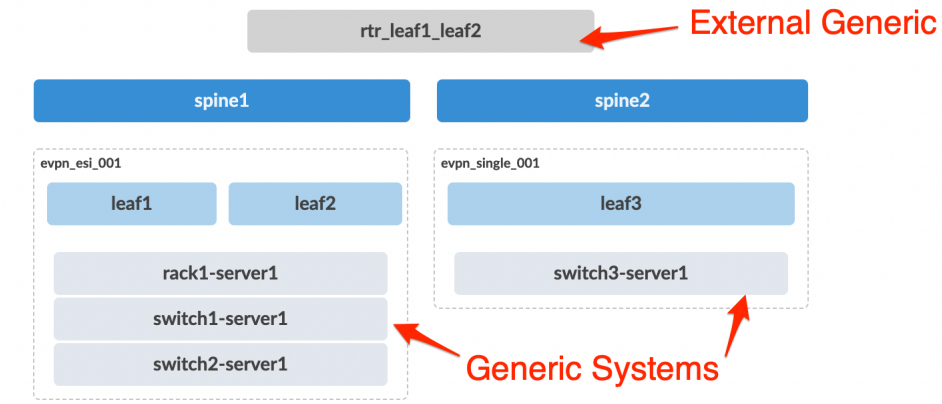
Generic System

- For attaching compute/storage
- Can only be connected to a single rack
- Appears in the topology as part of a rack

External Generic System

- For middleware devices, such as firewalls, load balancers, external routers and so on*
- Can be connected to multiple racks
- Appears in the topology outside of racks for easier identification

* In many cases, middleware boxes only connect to a single *border leaf pair* in a rack, but configuring it as an **external generic** system allows it to be visually separated outside of the rack. However, if there is a requirement such as connecting to an external router (MX) via BGP and you want to provide rack redundancy, then you would use an external generic system to allow this multi-rack connectivity.



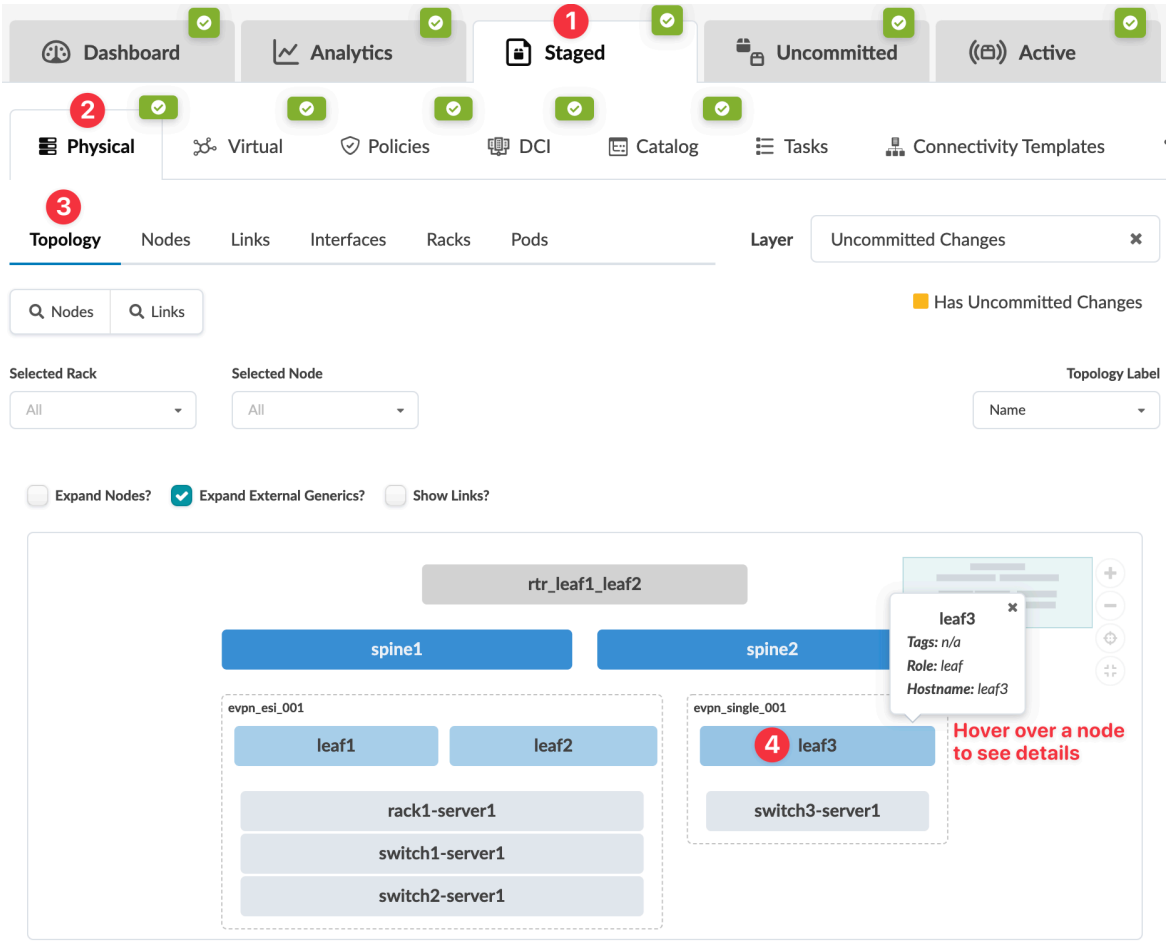
Create Internal Generic System

IN THIS SECTION

Systems that are not managed by Apstra and that are part of a rack topology are called (internal) generic systems. You specify their roles with tags. You can create an internal generic system by adding a new one or by copying an existing one.

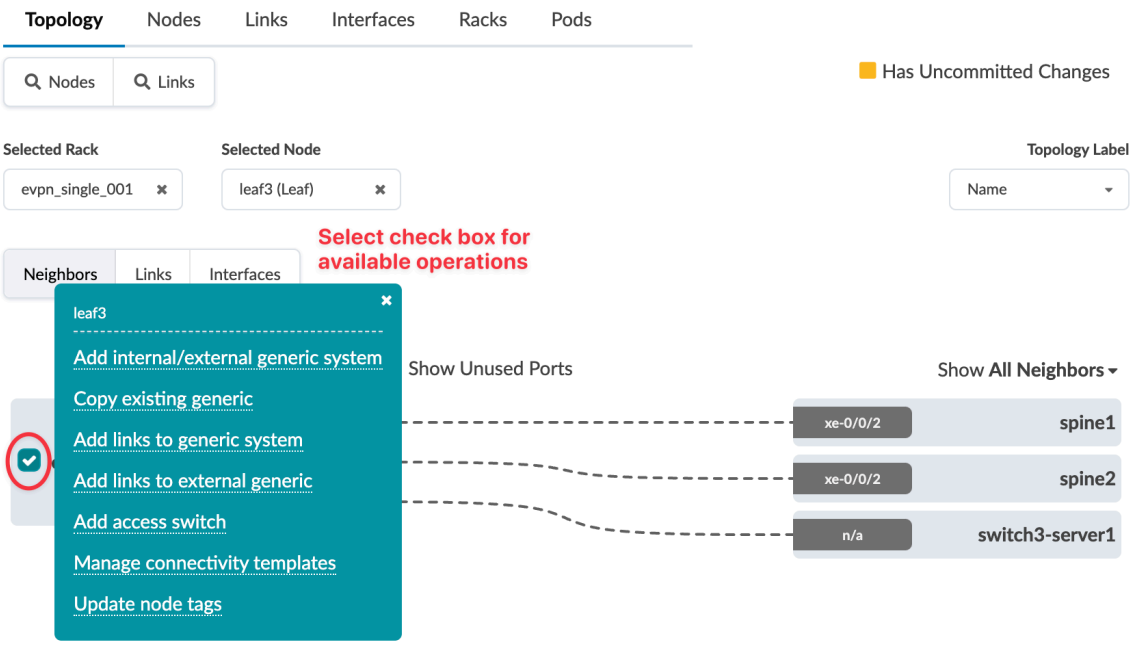
Add Internal Generic System

1. From the blueprint, navigate to **Staged > Physical > Topology** and select the leaf or access switch that you want to connect to the new internal generic system.



The **Topology** view of the selection appears.

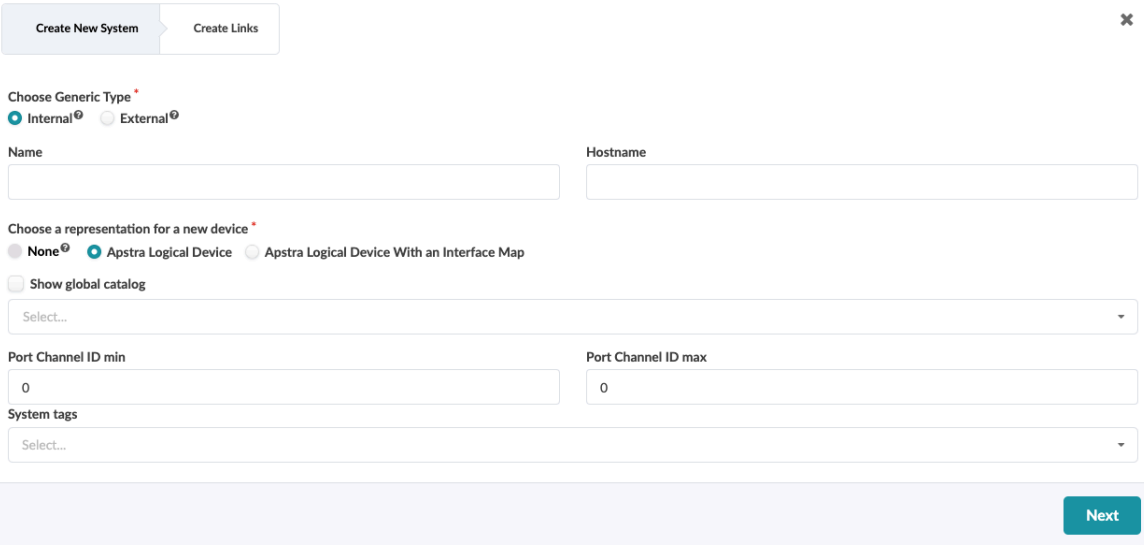
2. Select the node check box to see the operations available for that node (and that you have permissions for), then click **Add internal/external generic system**.



NOTE: You can also get to the selection page from the **Nodes** view. From the blueprint, navigate to **Staged > Physical > Nodes**, click the node name in the table, then click the node name that appears at the top of the **Selection** panel (on the right side of the page).

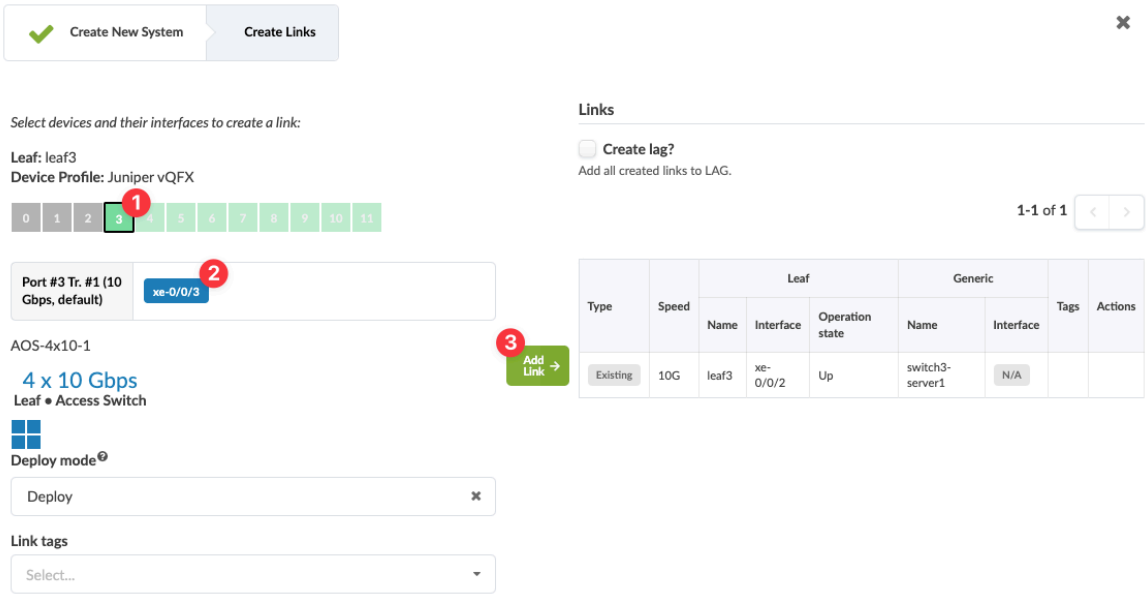
The **Create New System** dialog opens.

3. For **Generic Type**, select **Internal**.

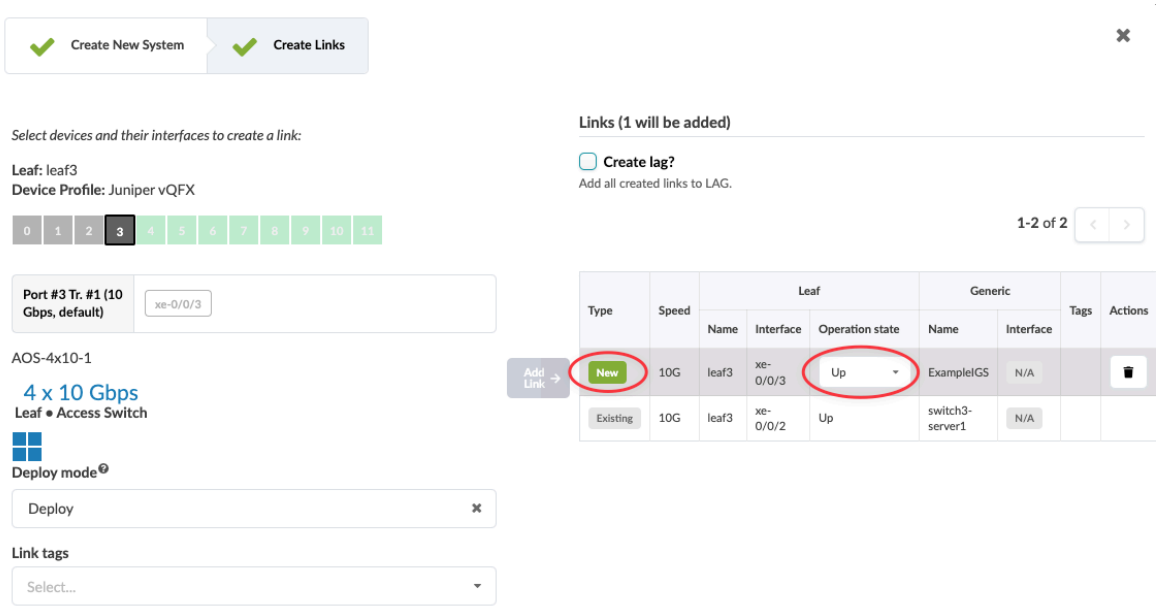


4. Enter a unique name. The optional **Hostname** automatically populates with the same name.

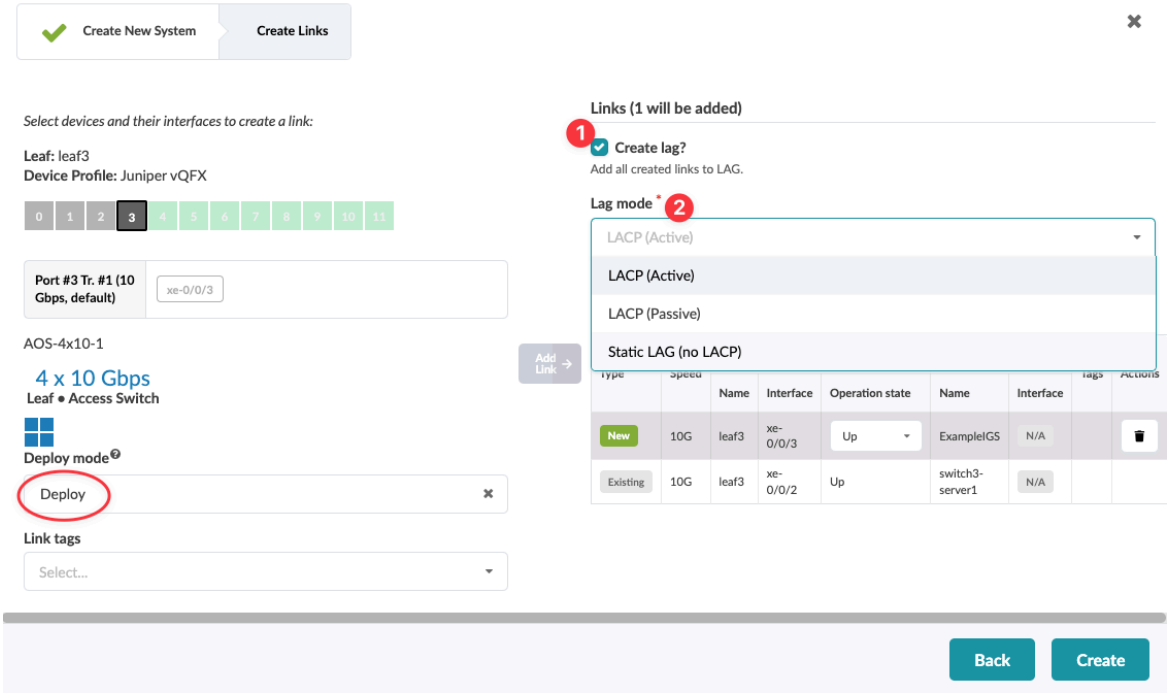
5. Select the representation for the new node (none, logical device, or logical device with interface map), then select the appropriate logical device or interface map from the drop-down list, as applicable. (Logical devices allow you to define port roles.)
6. Enter the port channel ID min and max. If you leave the values at zero, any available port-channel may be used. Non-default port channel ranges need only be unique per *system*, not per *blueprint*.
7. Enter tags (optional) to identify the role(s) of the new internal generic system.
8. Click **Next**.
   The **Create Links** dialog opens
9. Select an available port and transformation. The gray **Add Link** button turns green. Click **Add Link**.



   The link is added to the link table.
10. The operation mode (new in Apstra version 5.0.0) for new links defaults to **Up**. You have the option of setting it to **Down**, from the drop-down list.



11. You have the option of creating a LAG when you create a generic system (new in Apstra version 5.0.0). Click the **Create lag** check box, then select the LAG mode from the drop-down list.

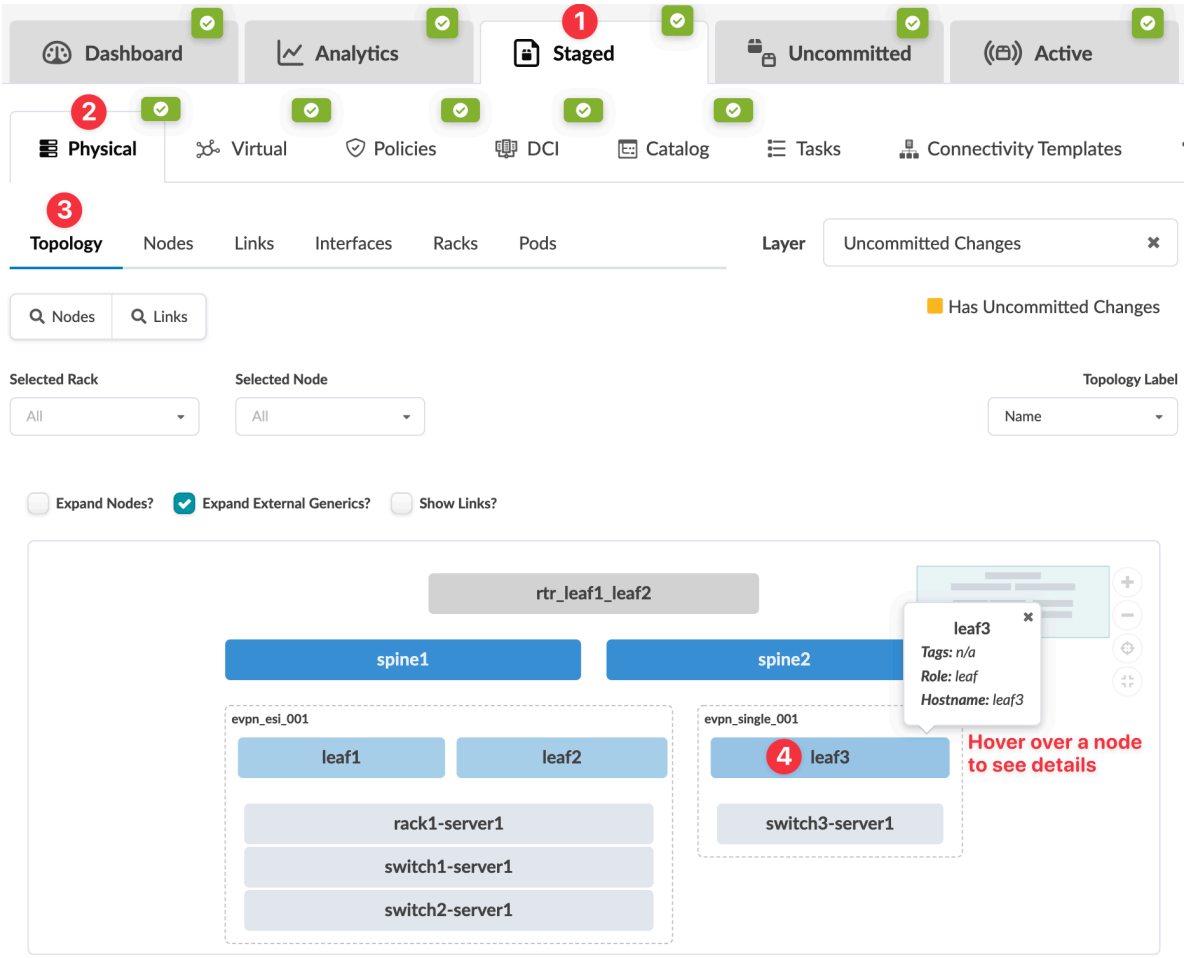


12. The **Deploy mode** (new in Apstra version 5.0.0) defaults to **Deploy**. You have the option of setting it to **Undeploy**, from the drop-down list. If all interfaces that are facing the generic system are **admin down**, then you can't set the deploy mode to **Deployed**.
13. Click **Create** to stage the change and return to the **Topology** view.

When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

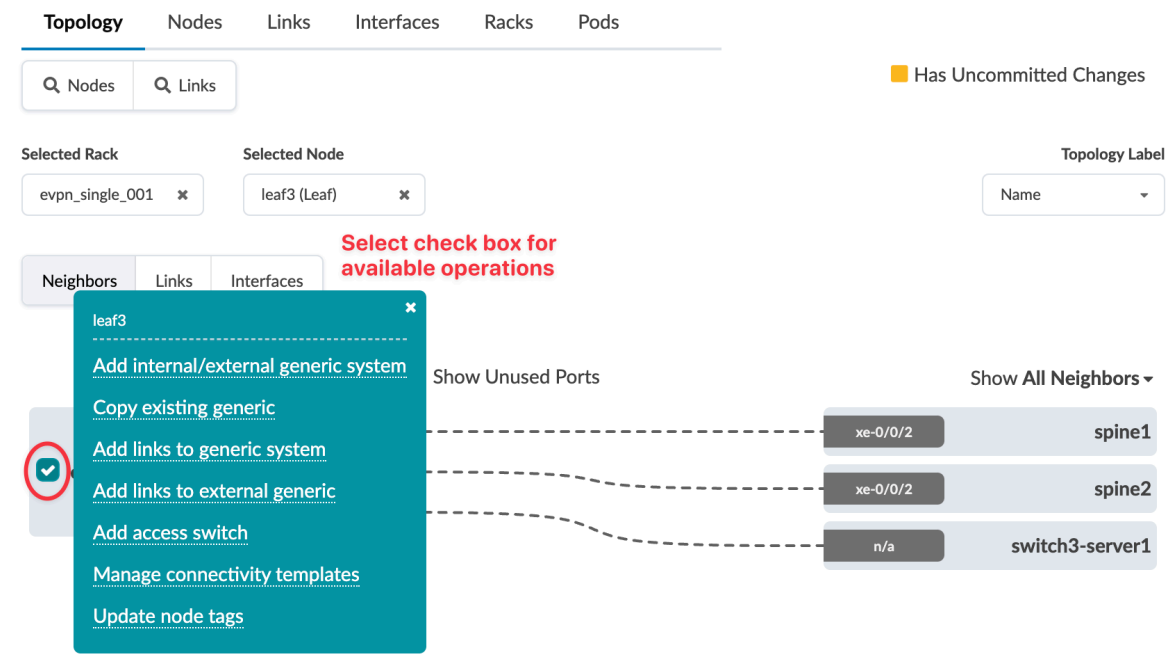
Copy Existing Generic System

1. From the blueprint, navigate to **Staged > Physical > Topology** and select the leaf or access switch that you want to connect to the new generic system.



The **Topology** view of the selection appears.

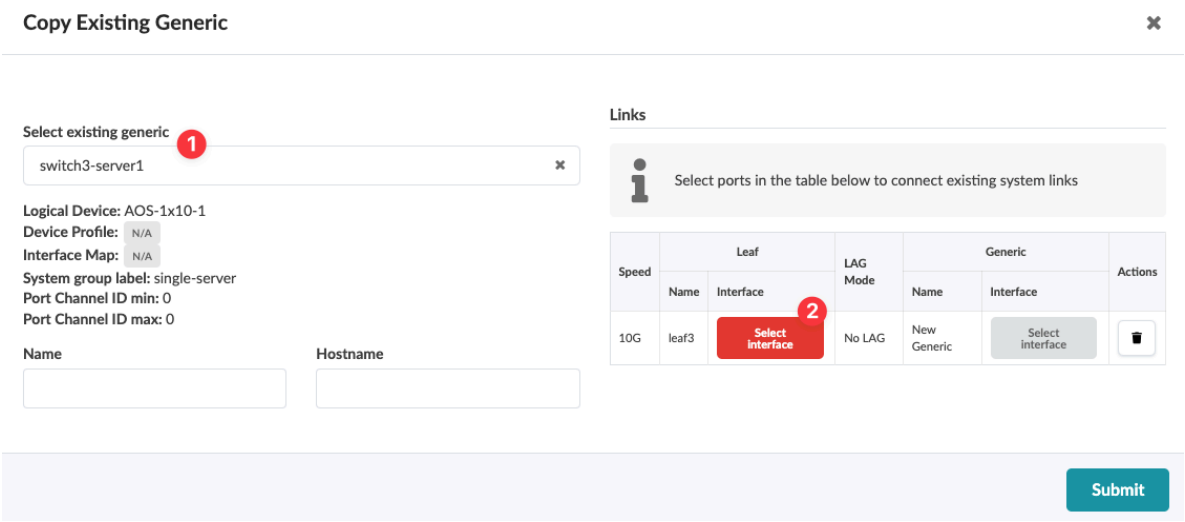
2. Select the node check box to see the operations available for that node (and that you have permissions for), then click **Copy existing generic**.



NOTE: You can also get to the selection page from the **Nodes** view. From the blueprint, navigate to **Staged > Physical > Nodes**, click the node name in the table, then click the node name that appears at the top of the **Selection** panel (on the right side of the page).

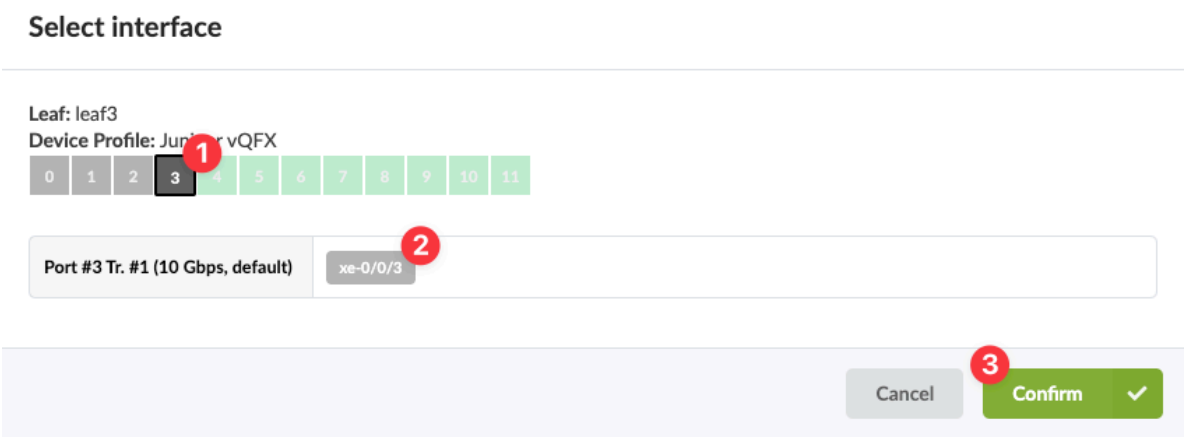
The **Copy Existing Generic** dialog opens.

3. From the drop-down list, select the existing internal generic system to copy, then in the **Links** table that appears, click **Select interface**.



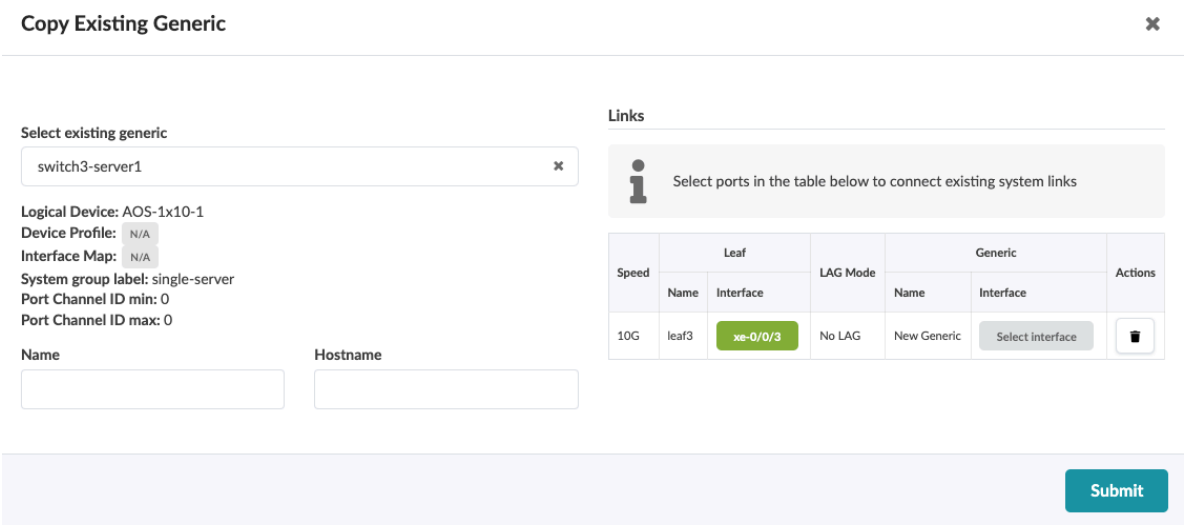
The **Select interface** dialog opens.

4. Select a port and transformation, then click **Confirm**.



The **Copy Existing Generic** dialog opens again.

5. Click **Submit** to stage the change and return to the **Topology** view.



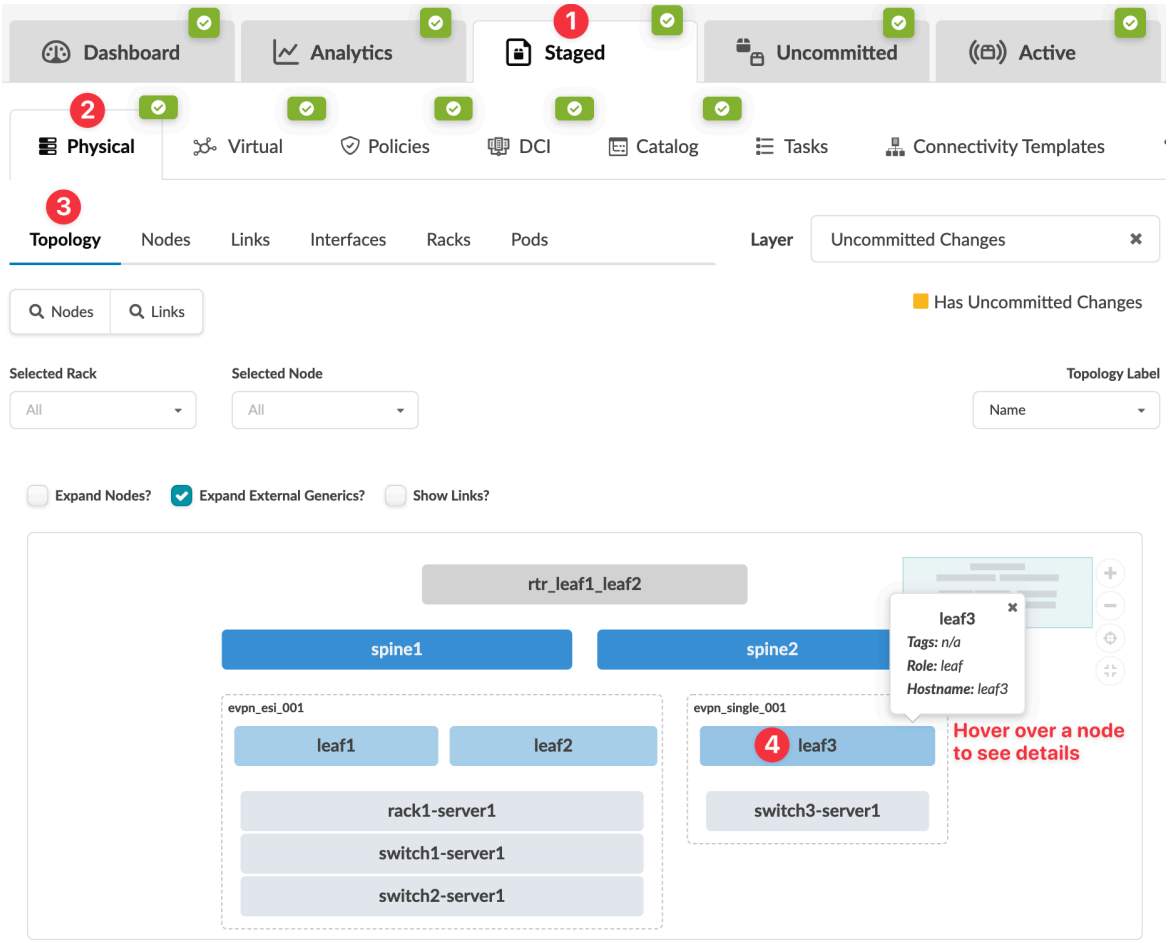
When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

NOTE: You can also create generic systems when you create rack types during the Design phase.

Create External Generic System

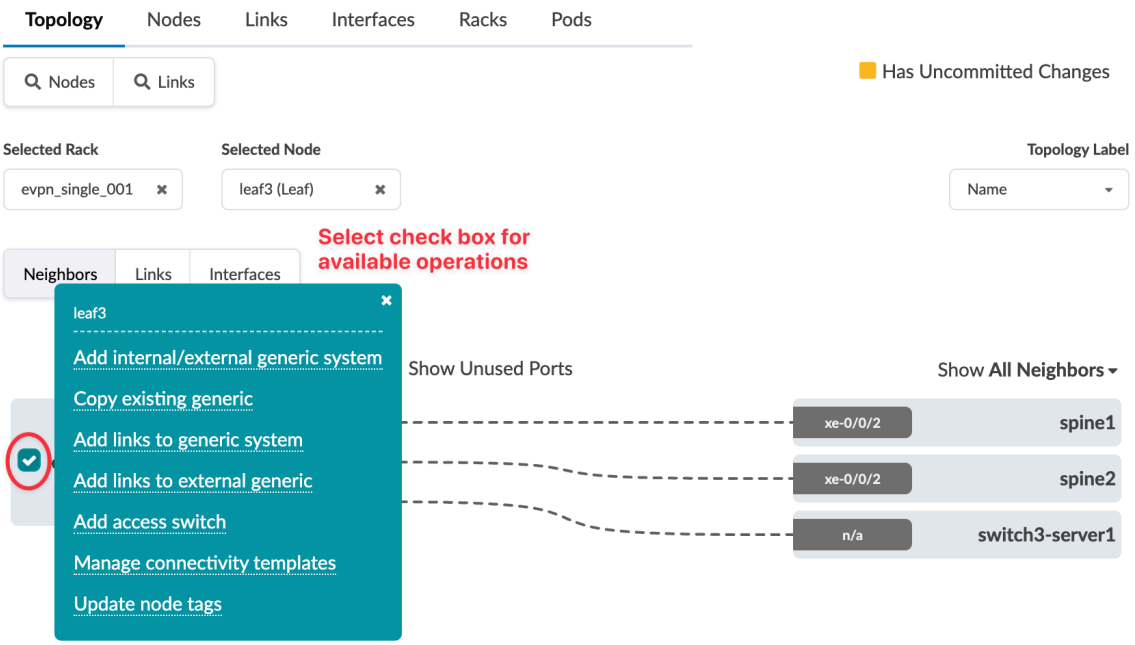
When you want to connect your Apstra-managed fabric to a system that's not managed in the Apstra environment, you use generic systems and external generic systems. These systems can be external routers, firewalls, or whatever else you want; you specify their roles with tags. If the system is part of a rack topology, we call it a generic system (or internal generic system). If the system is *not* part of a rack topology, we call it an external generic system.

1. From the blueprint, navigate to **Staged > Physical > Topology** and select the leaf, spine or superspine to connect to the new external generic system.



The **Topology** view of the selection appears.

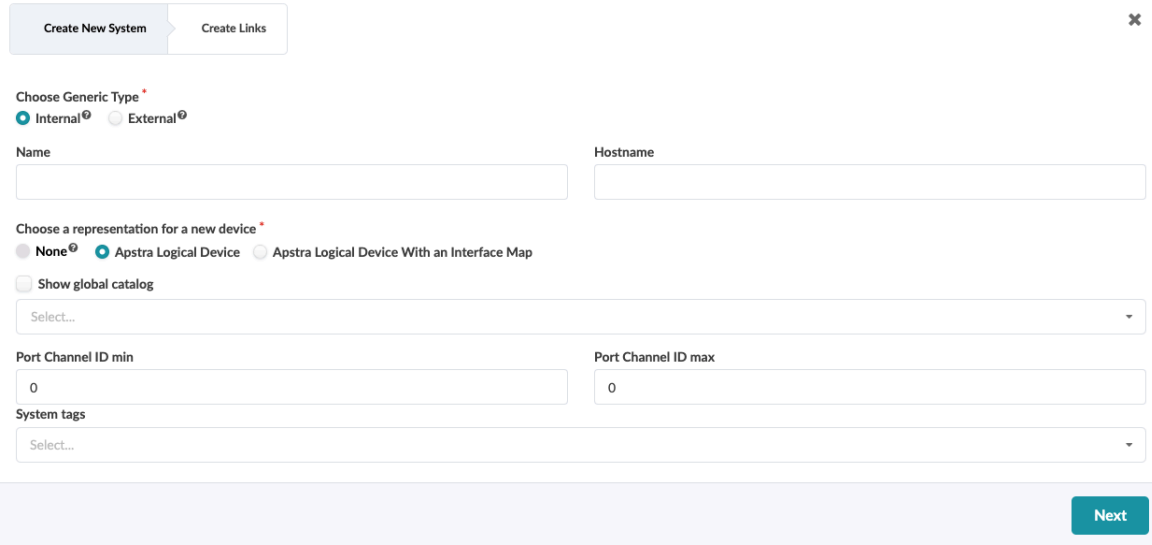
2. Select the node check box to see the operations available for that node (and that you have permissions for), then click **Add internal/external generic system**.



NOTE: You can also get to the selection page from the **Nodes** view. From the blueprint, navigate to **Staged > Physical > Nodes**, click the node name in the table, then click the node name that appears at the top of the **Selection** panel (on the right side of the page).

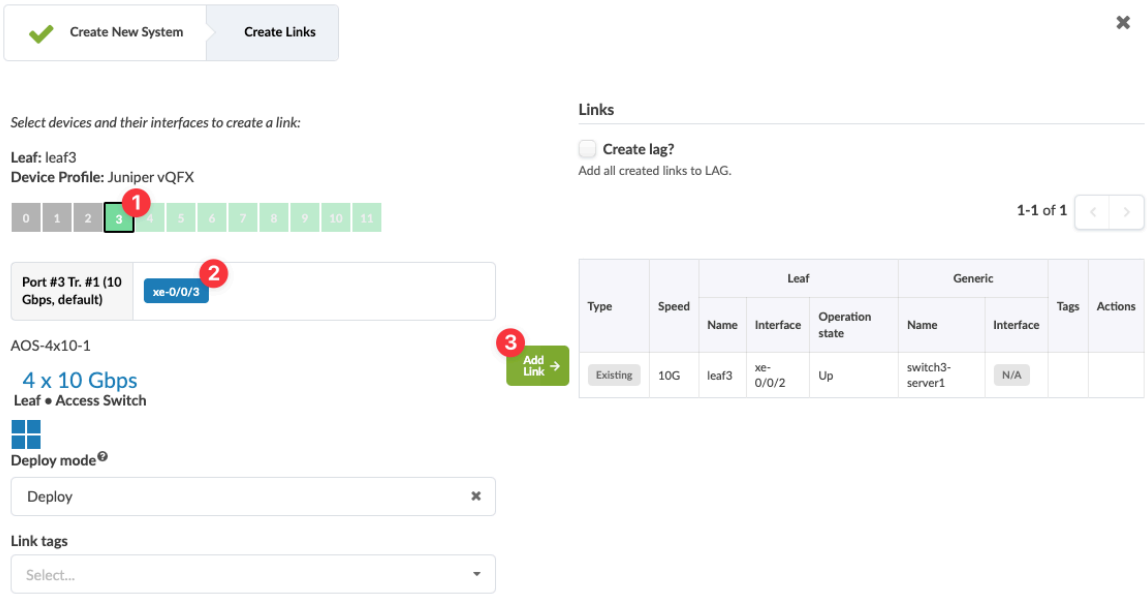
The **Create New System** dialog opens.

3. For **Generic Type**, select **External**.

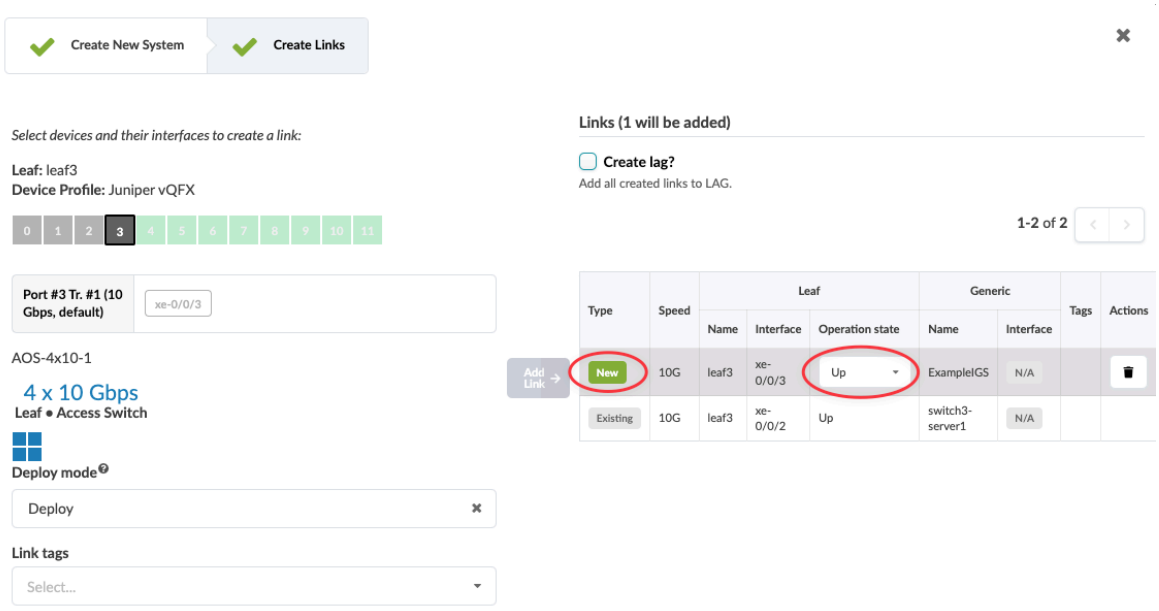


4. Enter a unique name. The optional **Hostname** automatically populates with the same name.

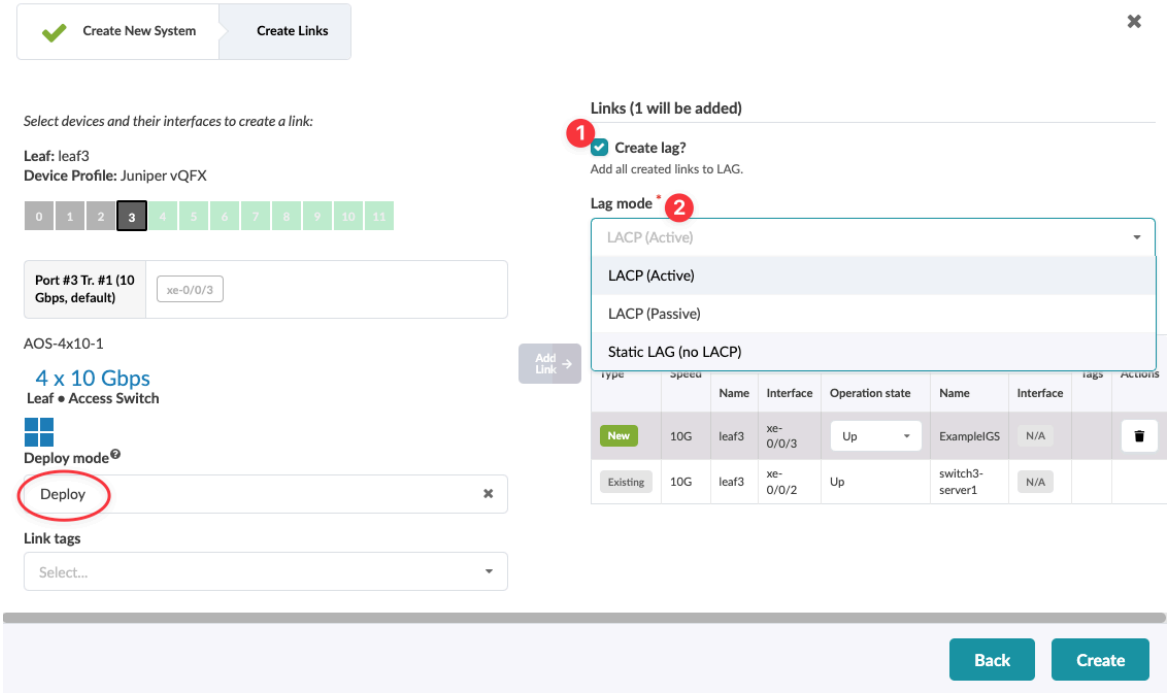
5. Select the representation for the new node (none, logical device, or logical device with interface map), then select the appropriate logical device or interface map from the drop-down list, as applicable. (Logical devices allow you to define port roles.)
6. Enter the port channel ID min and max. The values in the range are used to allocate PC IDs for all leafs, spines, and superspines attached to this external generic system. If you leave the values at zero, any available port-channel may be used.
7. Enter tags (optional) to identify the role(s) of the new external generic system.
8. Click **Next**.
   The **Create Links** dialog opens
9. Select an available port and transformation. The gray **Add Link** button turns green. Click **Add Link**.



   The link is added to the link table.
10. The operation mode (new in Apstra version 5.0.0) for new links defaults to **Up**. You have the option of setting it to **Down**, from the drop-down list.



11. You have the option of creating a LAG when you create a generic system (new in Apstra version 5.0.0). Click the **Create lag** check box, then select the LAG mode from the drop-down list.



12. The **Deploy mode** (new in Apstra version 5.0.0) defaults to **Deploy**. You have the option of setting it to **Undeploy**, from the drop-down list. If all interfaces that are facing the generic system are **admin down**, then you can't set the deploy mode to **Deployed**.
13. Click **Create** to stage the change and return to the **Topology** view.

When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

RELATED DOCUMENTATION

Add Links to External Generic System | 157

Change Generic System Type

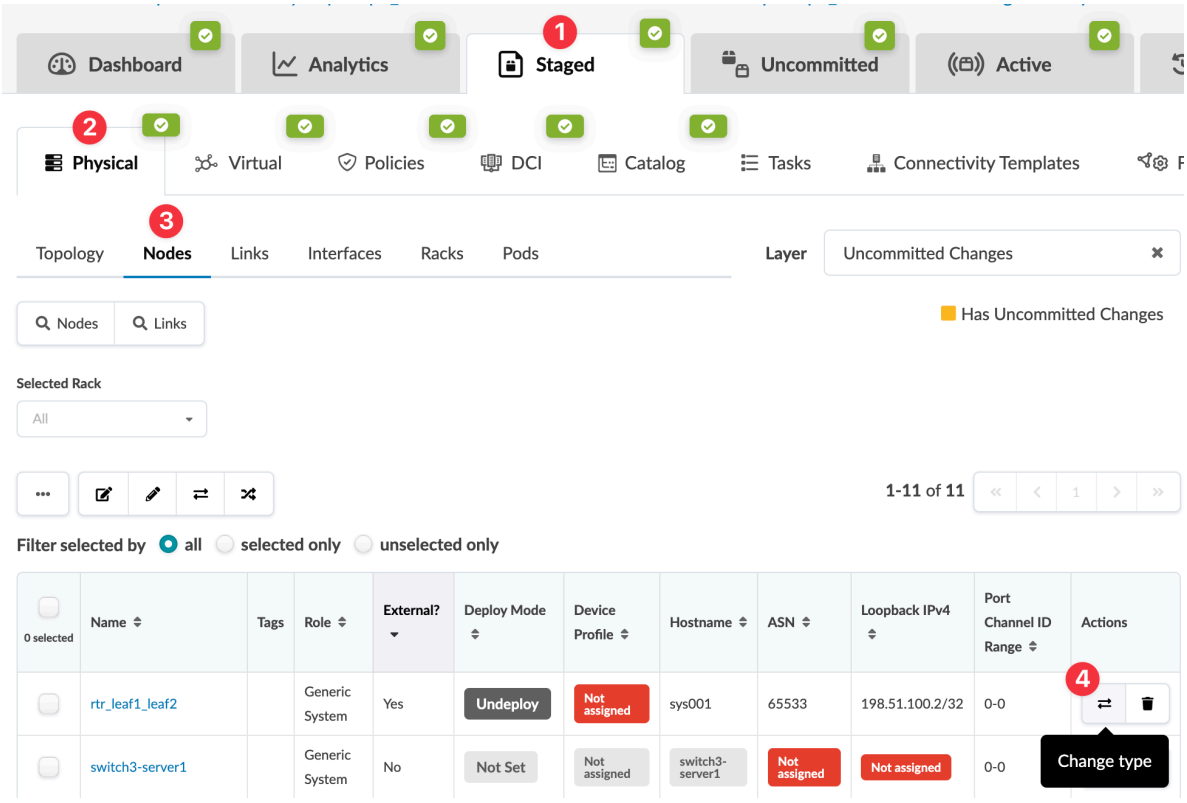
IN THIS SECTION

You can transfer generic systems from being internal (part of a rack) to being external (separate from a rack), or vice versa, with a single transaction (new in Apstra version 5.0.0). You can change one generic system at a time, or you can change multiple generic systems simultaneously. Previously, to change the generic system type, you had to delete the generic system from the blueprint and re-add it as the other type.

To transfer a generic system from external to internal, all associated external links must be located within the same rack.

Change Type on One Generic System

1. From the blueprint, navigate to **Staged > Physical > Nodes** and click the **Change type** button in the **Actions** panel, for the generic system to change.



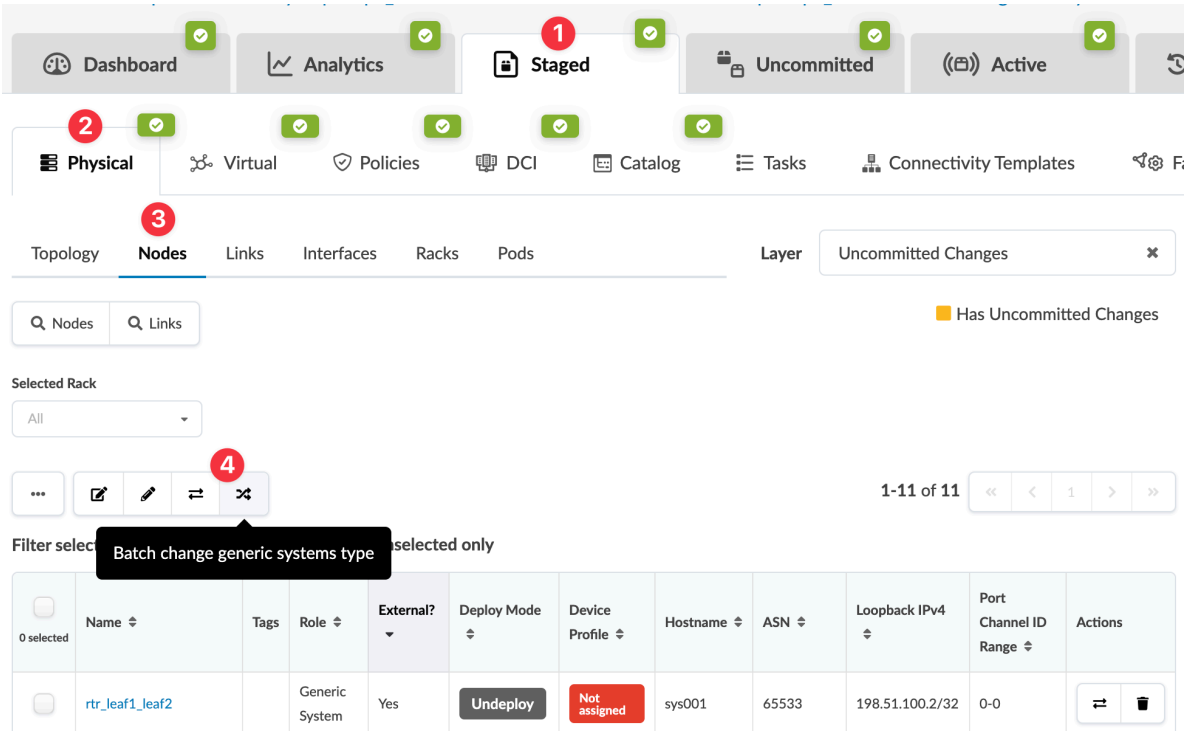
The **Change System Type** dialog opens.

2. Select the other generic system type.
3. Click **Submit** to stage the change and return to the **Nodes** view.

When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

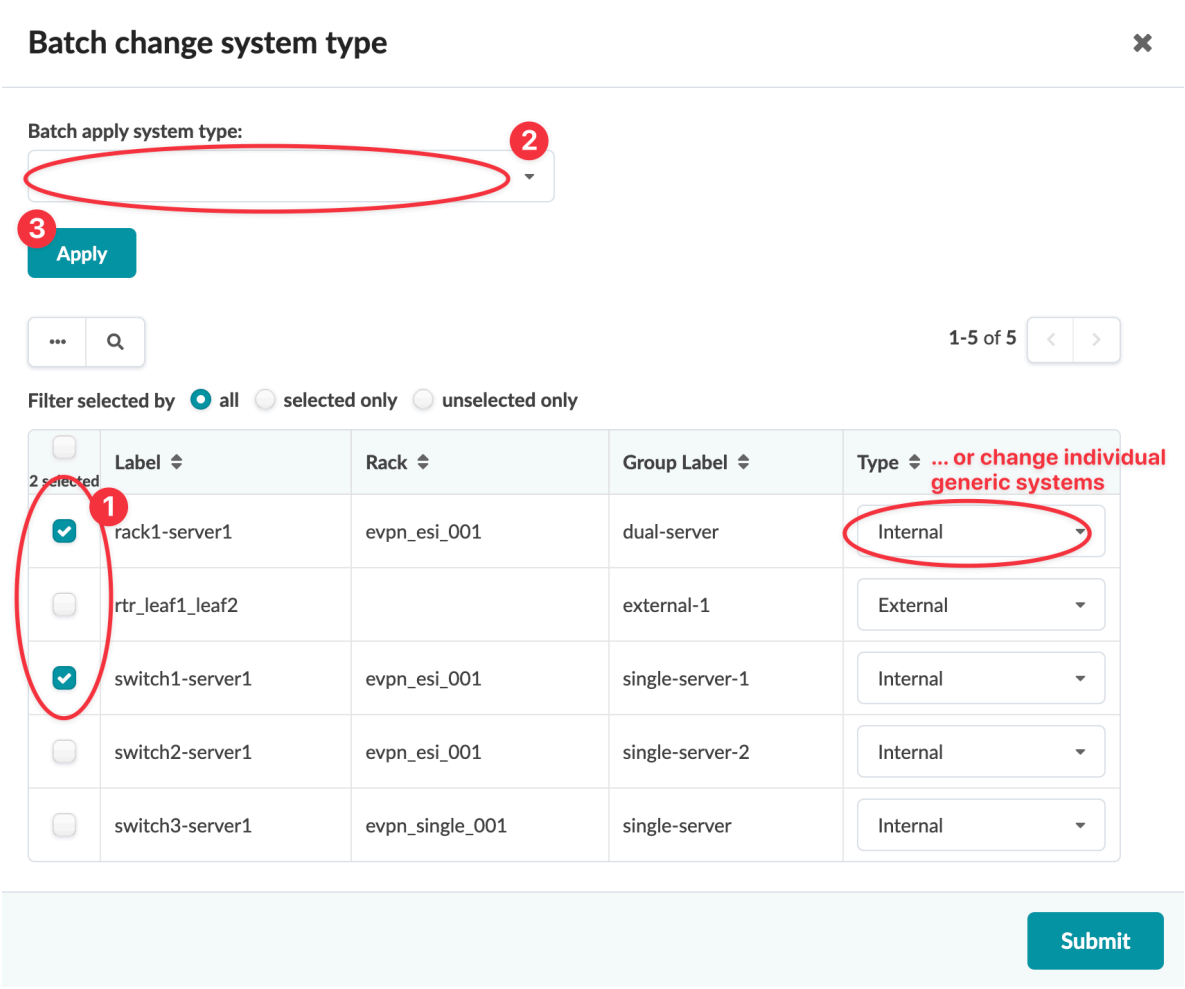
Change Type on Multiple Generic Systems

1. From the blueprint, navigate to **Staged > Physical > Nodes** and click the **Batch change generic systems type** button, above the table.



The **Batch change system type** dialog opens.

2. You can change multiple generic systems simultaneously by selecting their check boxes and selecting the generic system type from the **Batch apply system type** drop-down list, then clicking **Apply**; or you can change individual generic system by selecting the generic system type from the **Type** drop-down list for the specific generic system.



3. Click **Submit** to stage the change and return to the **Nodes** view.

When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

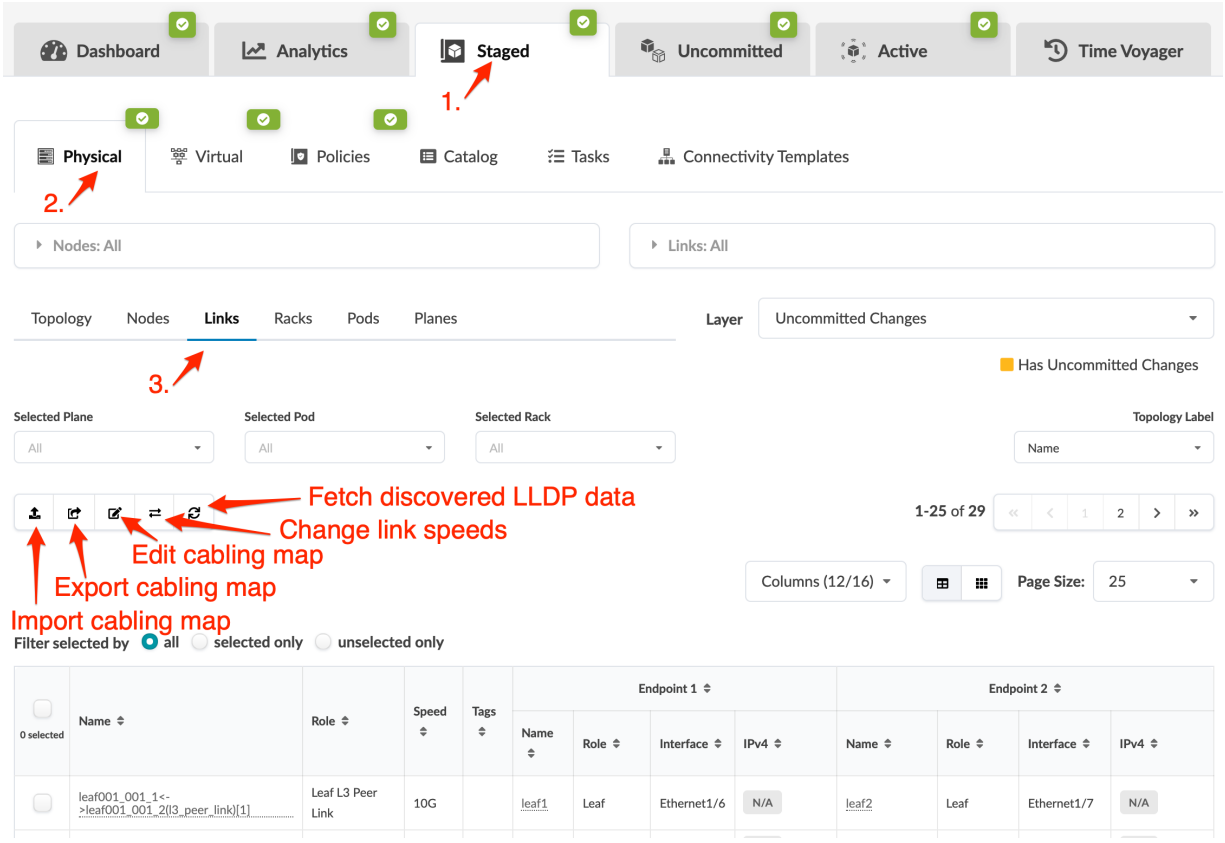
Links

IN THIS SECTION

Links (Datacenter)

IN THIS SECTION

From the blueprint, navigate to **Staged > Physical > Links** to go to the **Links** view.



Many link operations are performed from the **Topology** view, and some can also be performed directly in the **Links** view. See the following sections for more information.

Example of how links are assigned

Links Example

In this example, each link is assigned a unique /31 subnet from the IP Pool.

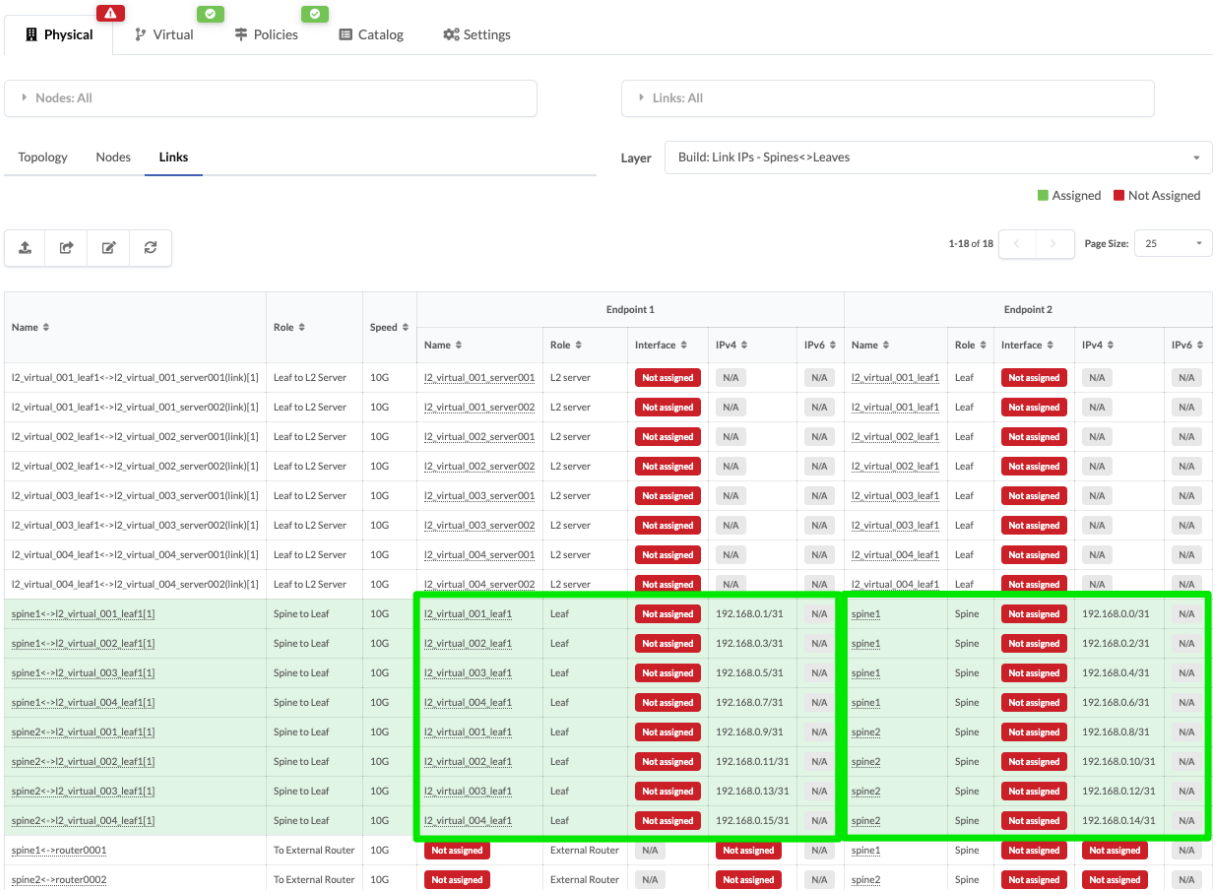
The smaller /31 IP is assigned to the spine interface.

he larger /31 IP is assigned to the leaf interface.

Subnets are assigned in increasing order in a spine-major order.

That is, the links between spine1 and all leafs (in ascending order) are assigned subnets first.

The links between spine2 and all leafs are then assigned subnets, and so on.

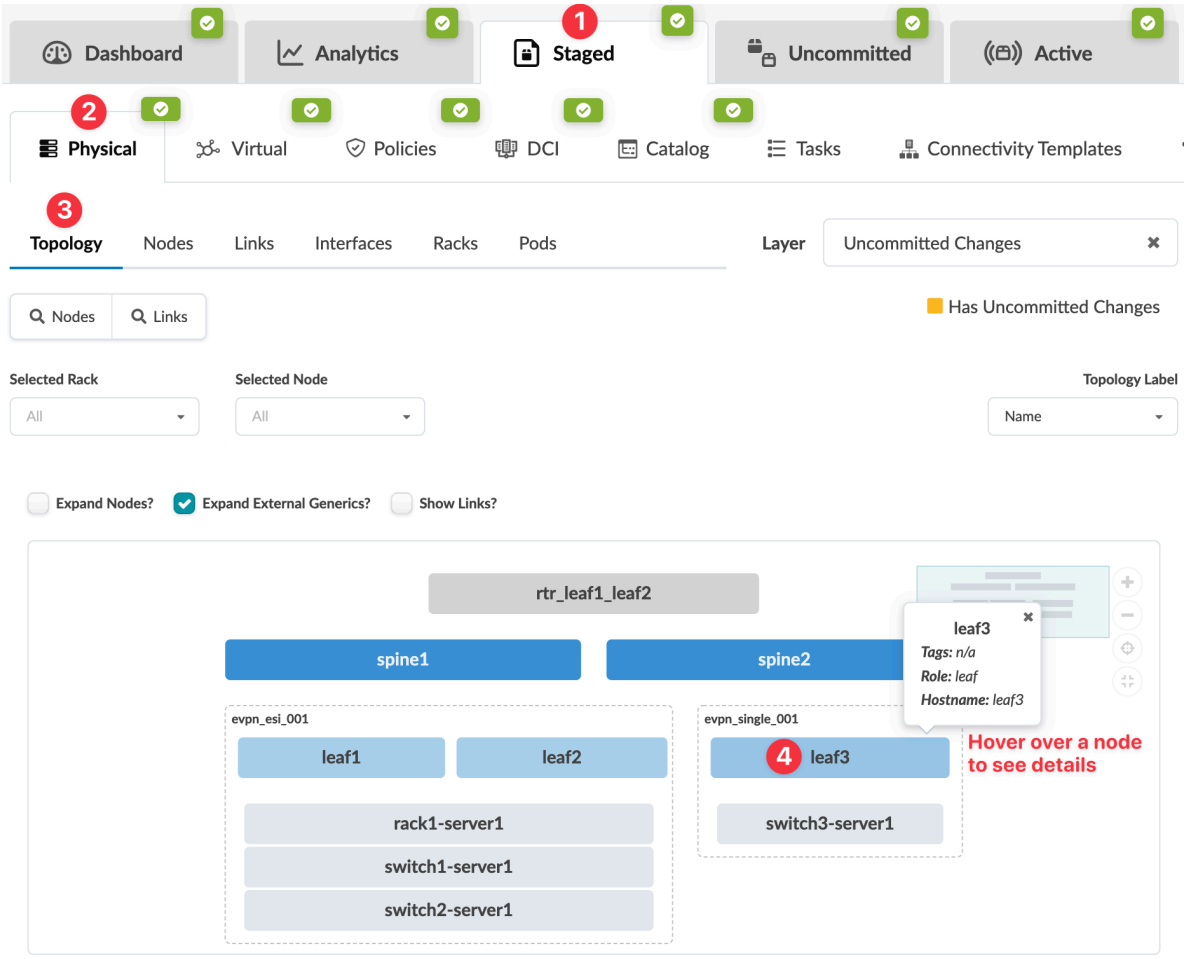


Add Links

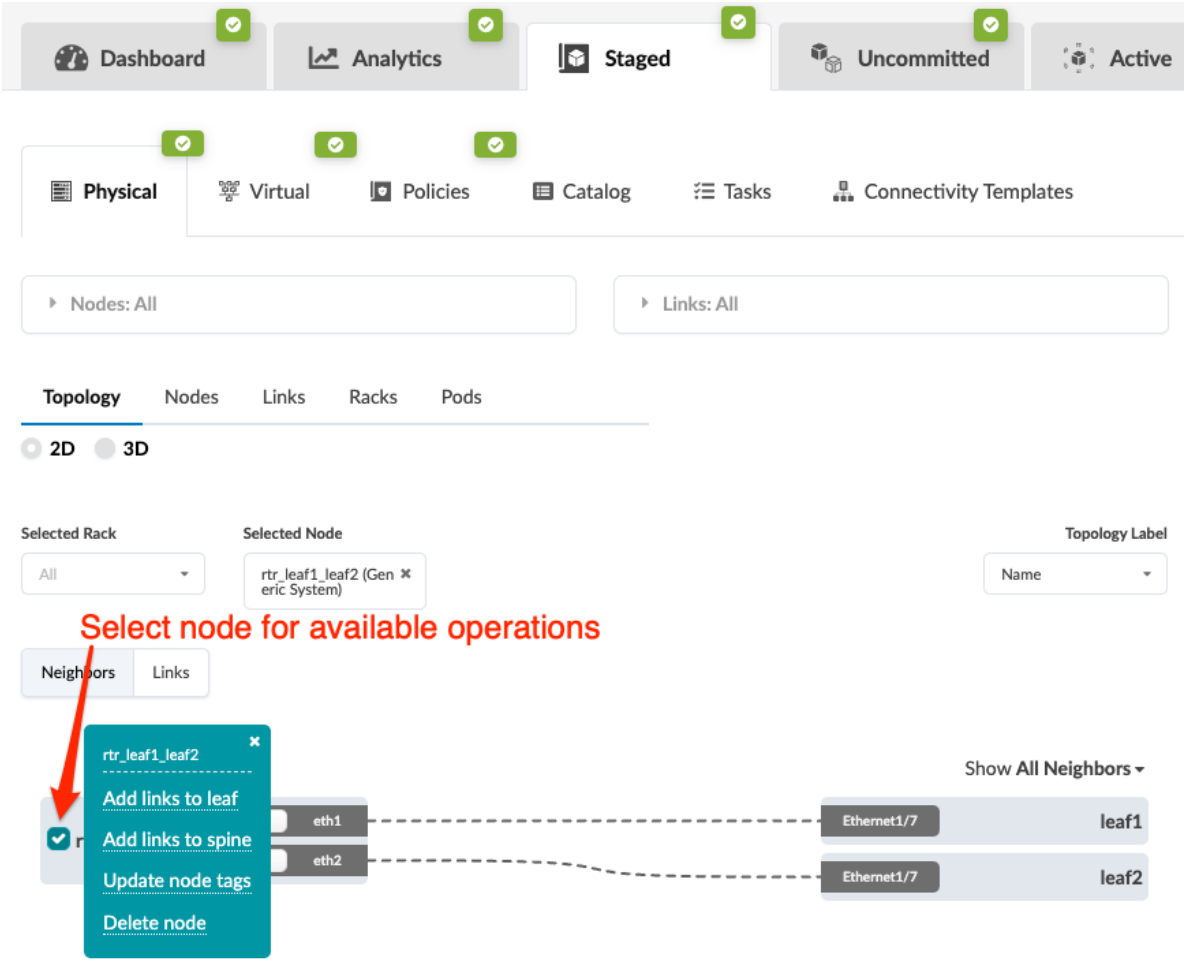
IN THIS SECTION

Add Links to Leaf

1. From the blueprint, navigate to **Staged > Physical > Topology** and select a node that can connect to a leaf.

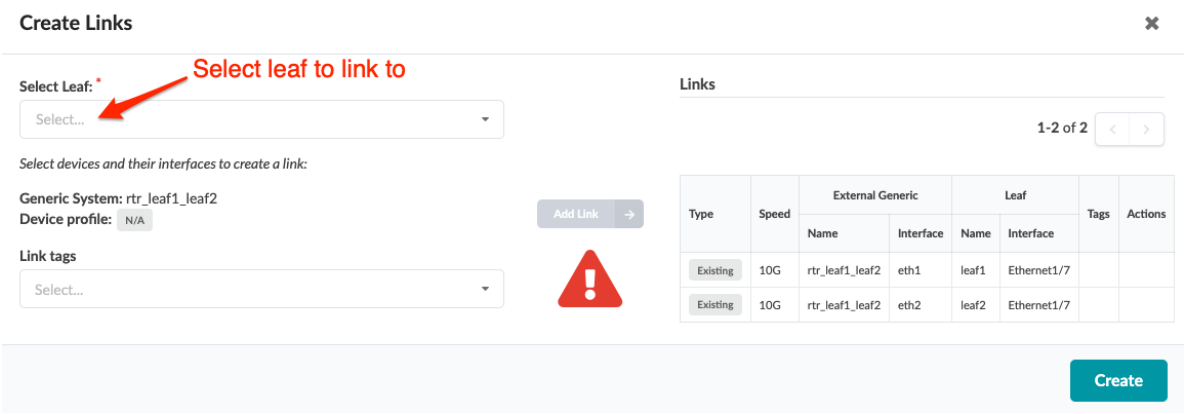


2. Select the node check box to see the operations available for that node (and that you have permissions for).

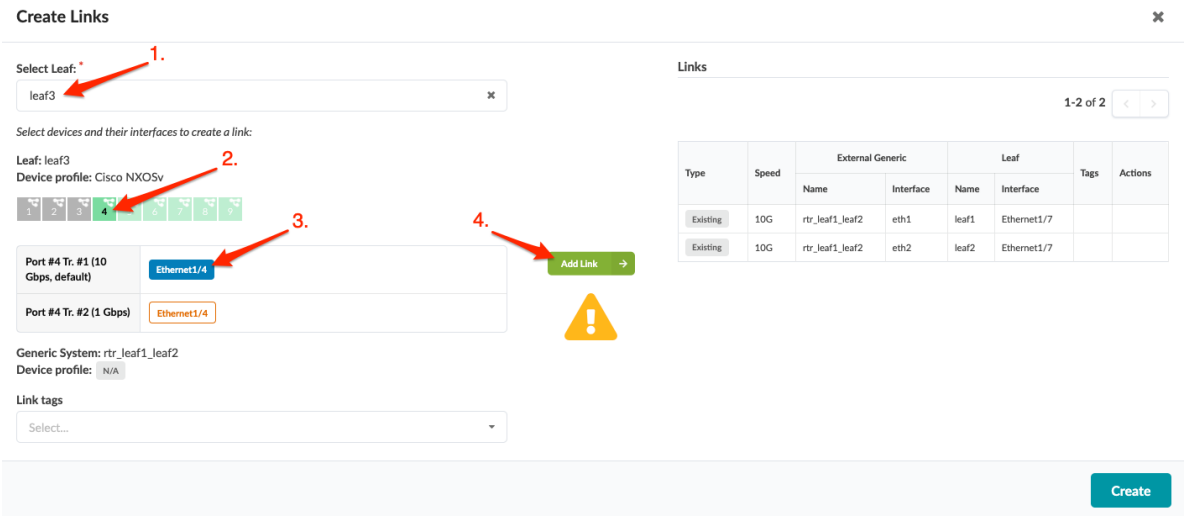


NOTE: You can also get to the selection page from the **Nodes** view. From the blueprint, navigate to **Staged > Physical > Nodes**, click the node name in the table, then click the node name that appears at the top of the **Selection** panel (on the right side of the page).

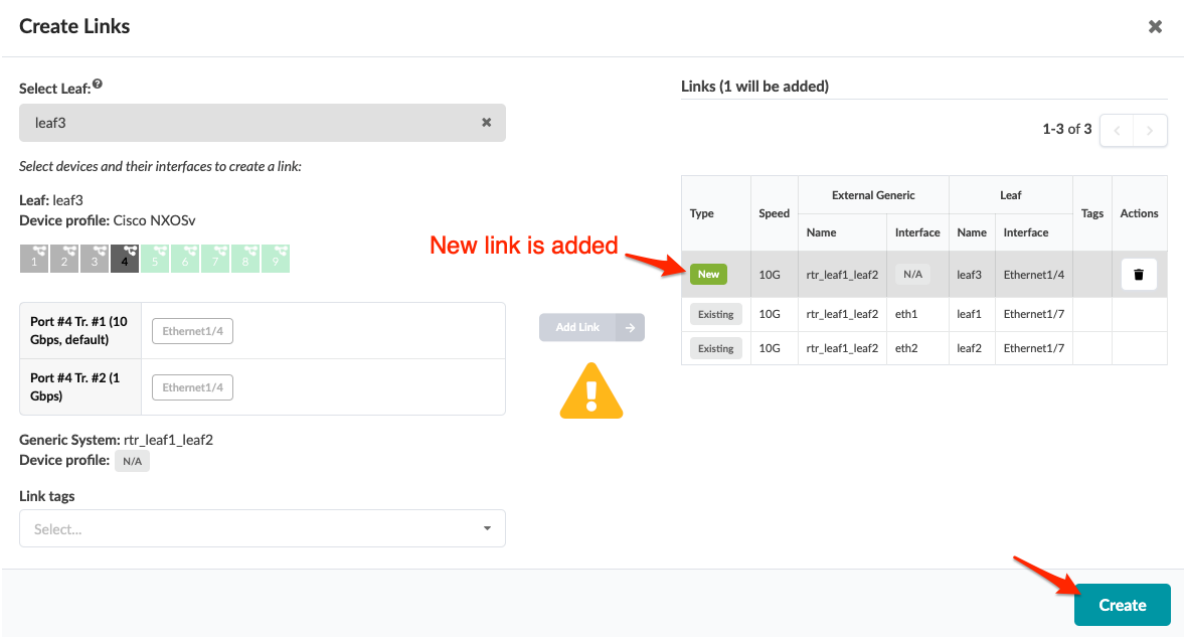
3. Click **Add links to leaf** to go to its dialog.



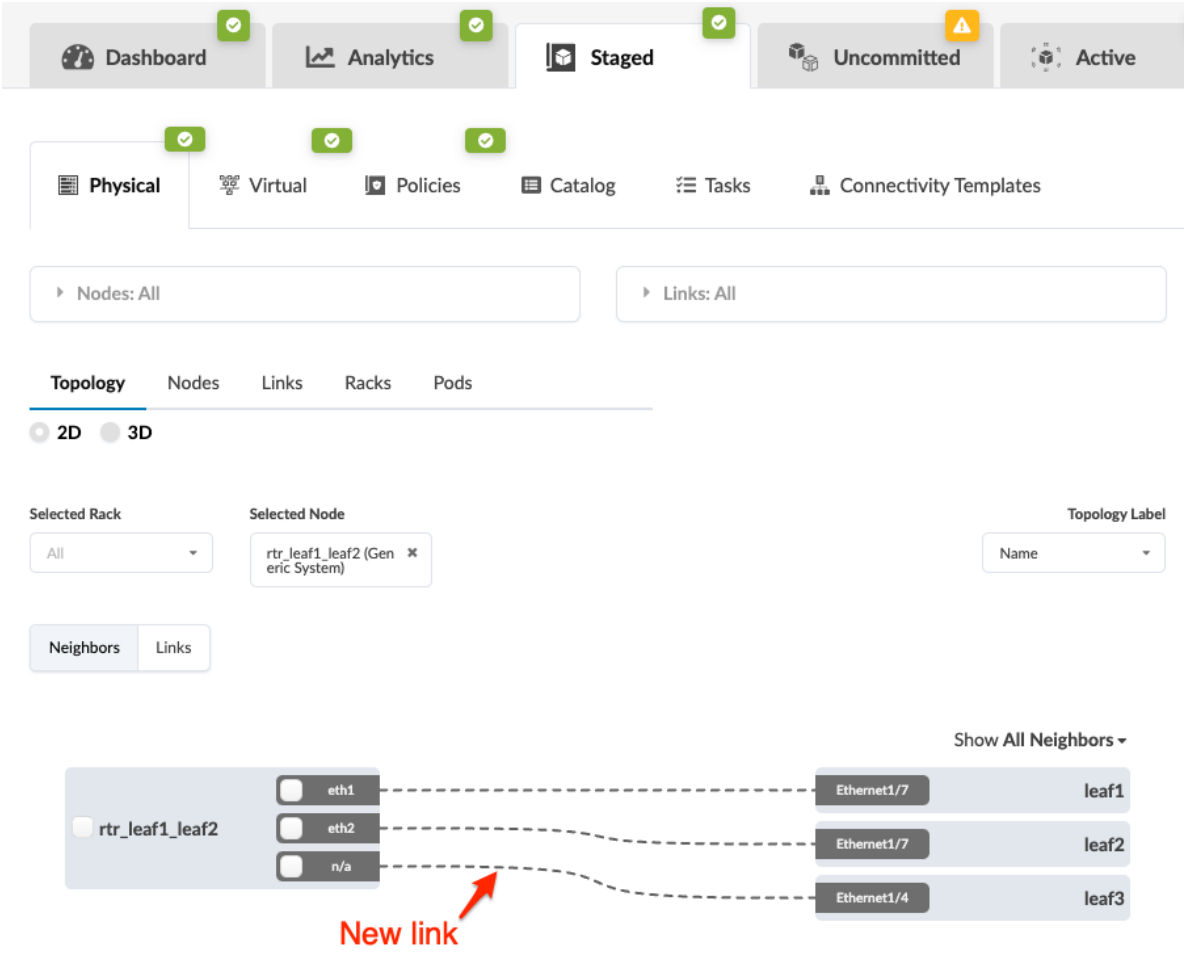
4. Select the leaf to link to from the drop-down menu, then select an available port and transformation. The gray **Add Link** button turns green.



5. Click **Add Link**. The link is added to the link table.



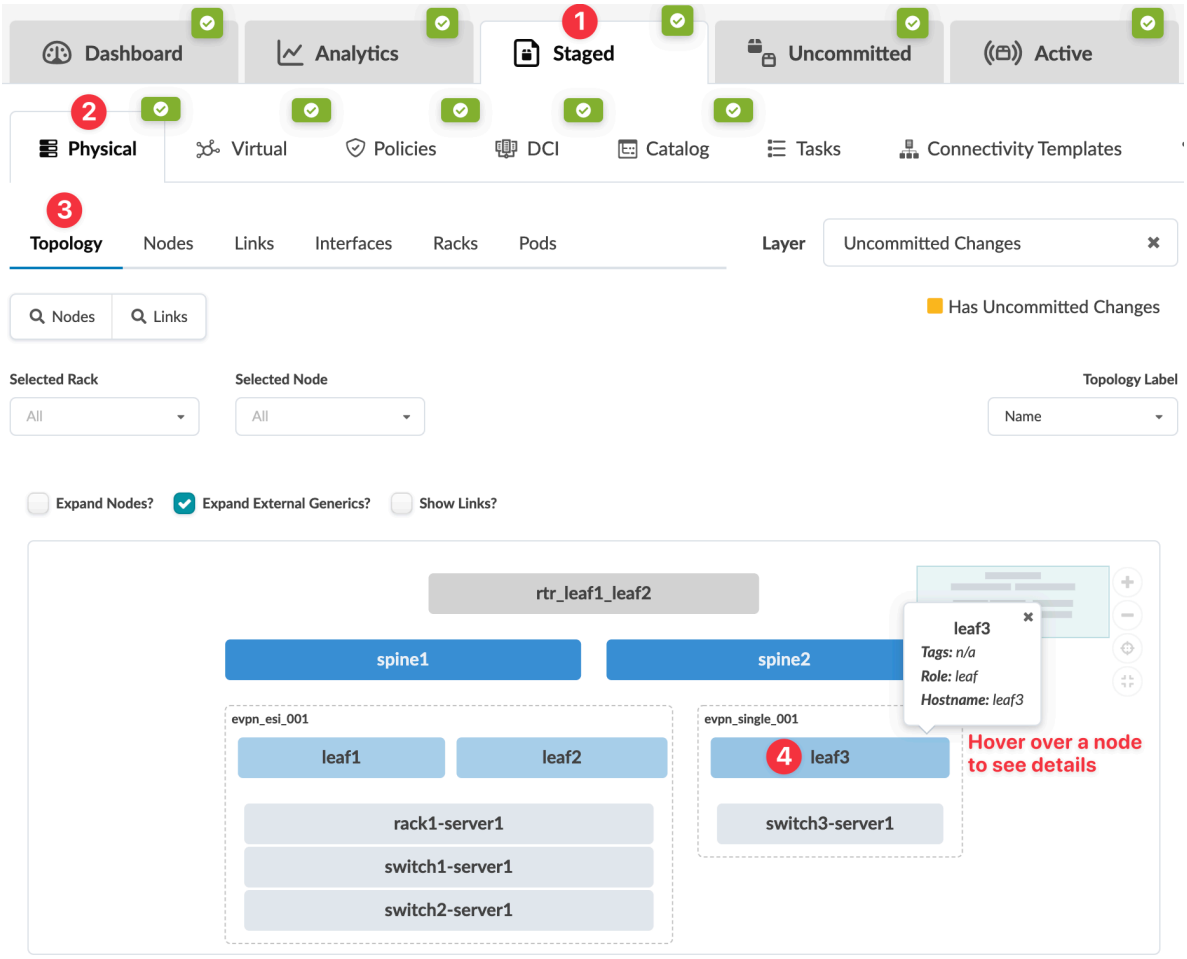
6. Click **Create** to stage the change and return to the **Topology** view.



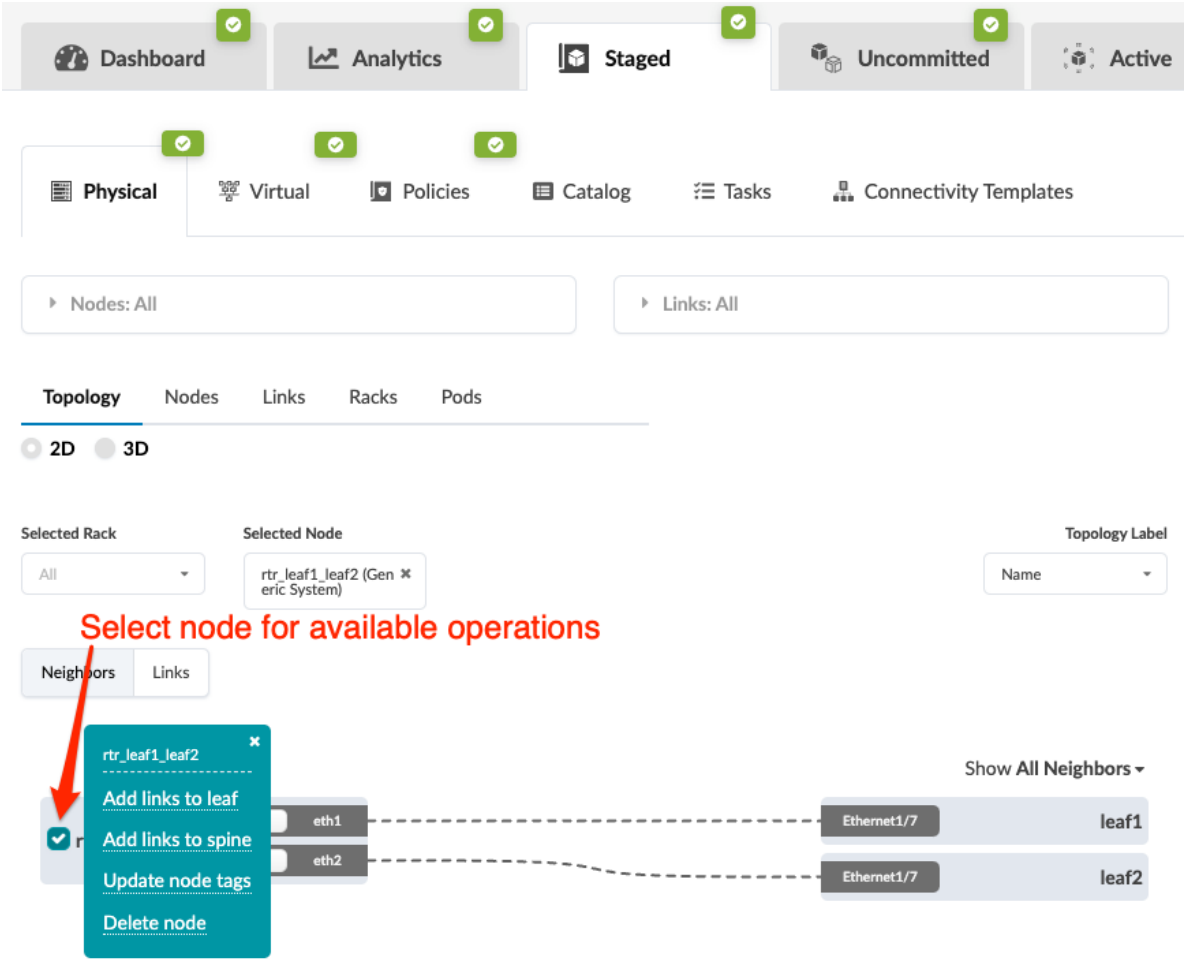
When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Add Links to Spine

1. From the blueprint, navigate to **Staged > Physical > Topology** and select a node that can connect to a spine.

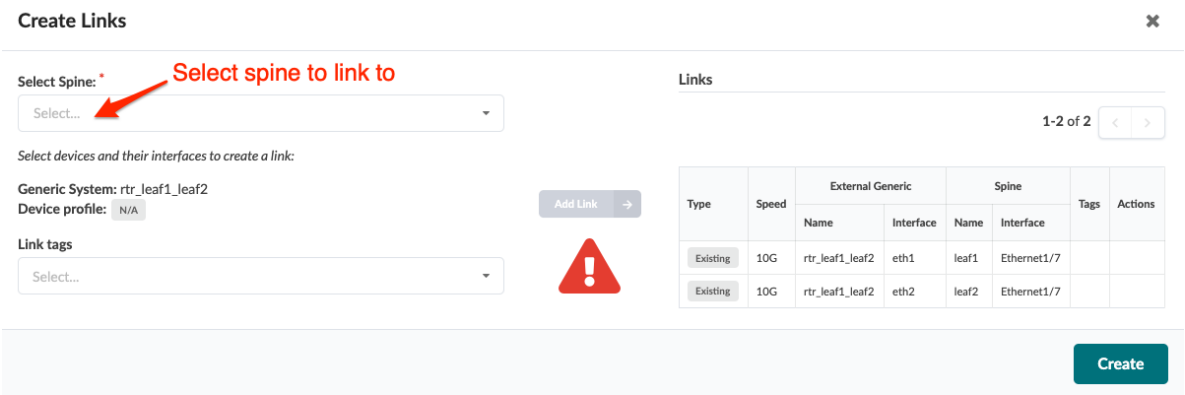


2. Select the node check box to see the operations available for that node (and that you have permissions for).

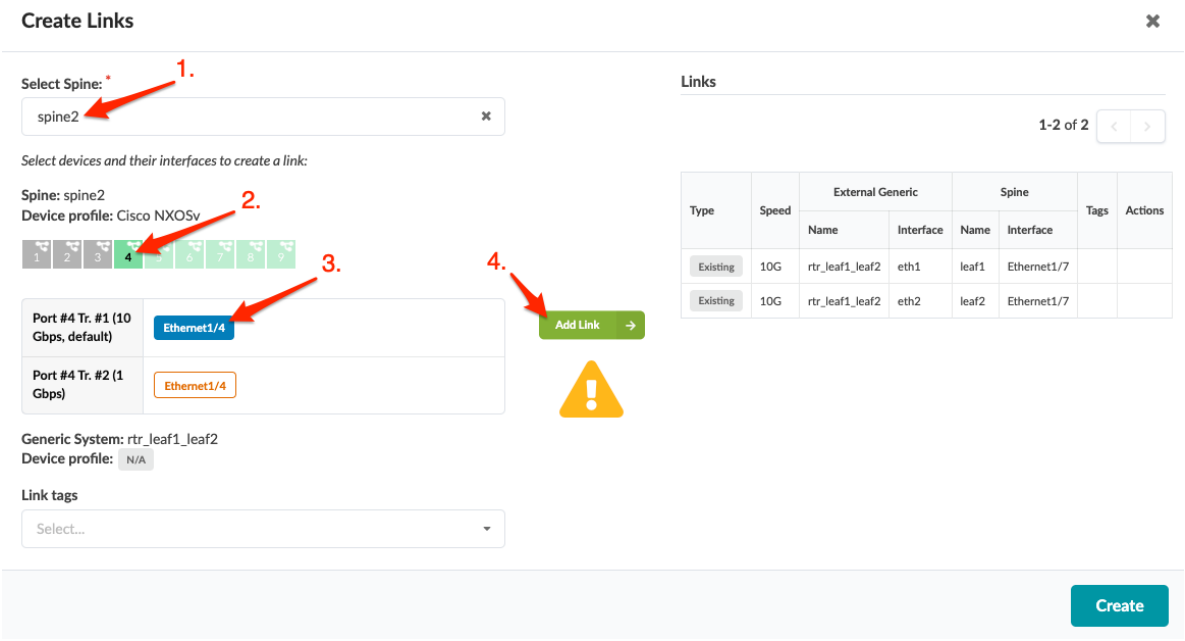


NOTE: You can also get to the selection page from the **Nodes** view. From the blueprint, navigate to **Staged > Physical > Nodes**, click the node name in the table, then click the node name that appears at the top of the **Selection** panel (on the right side of the page).

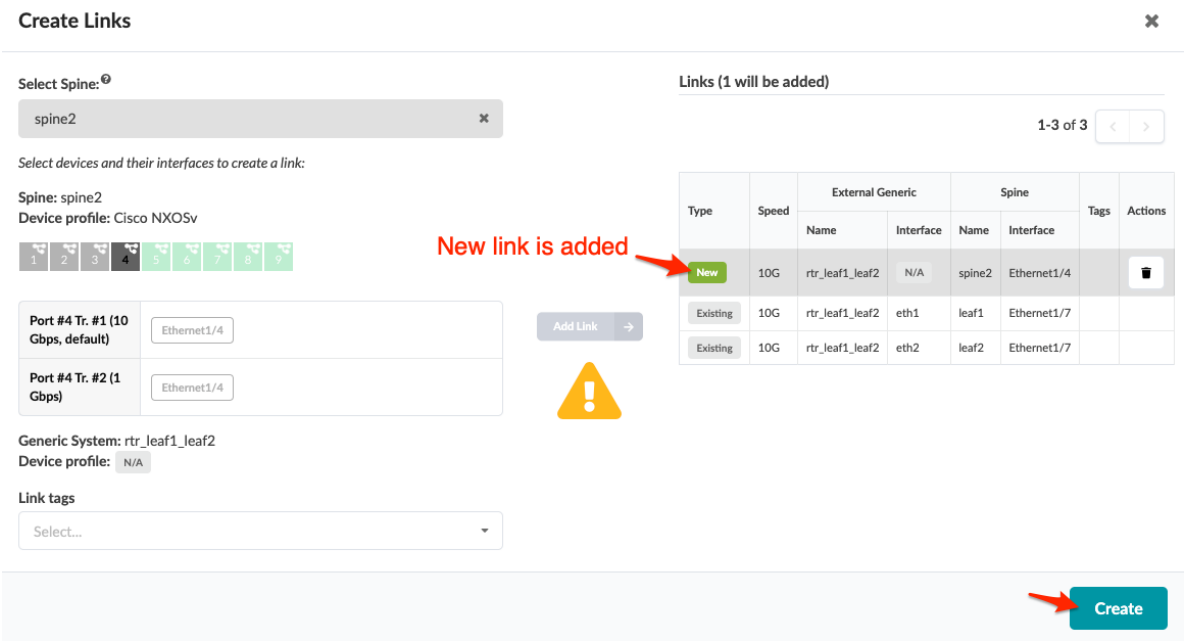
3. Click **Add links to spine** to go to its dialog.



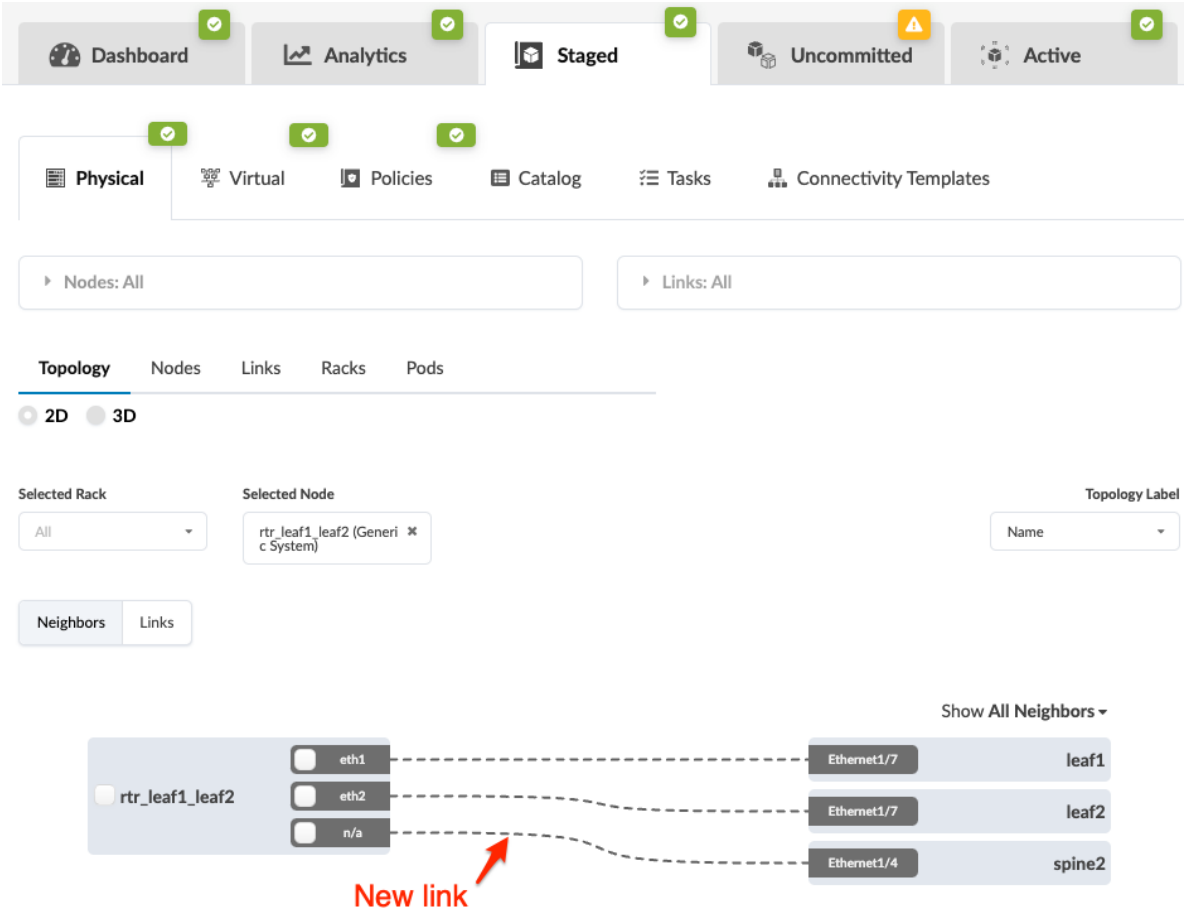
4. Select the spine to link to from the drop-down menu, then select an available port and transformation. The gray **Add Link** button turns green.



5. Click **Add Link**. The link is added to the link table.



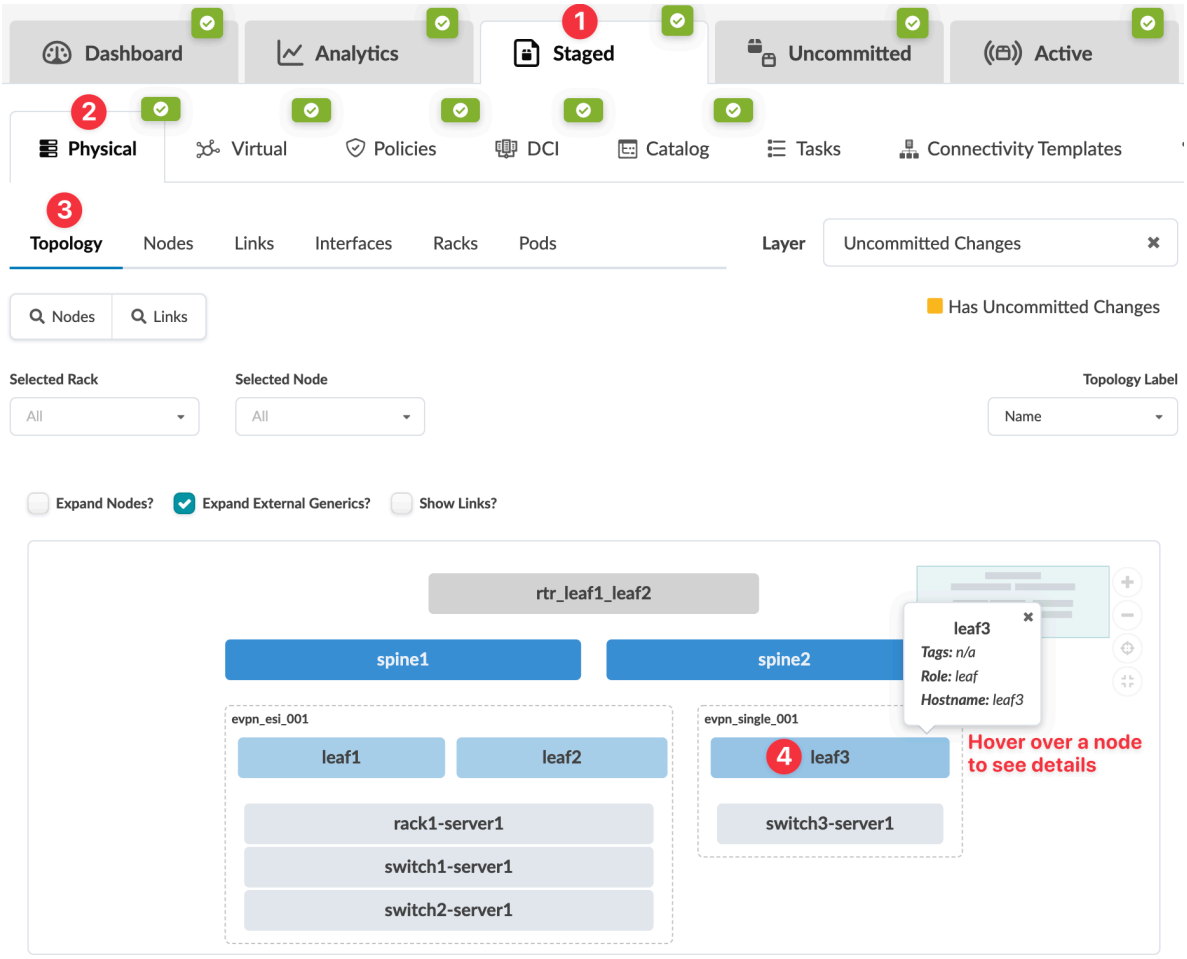
6. Click **Create** to stage the change and return to the **Topology** view.



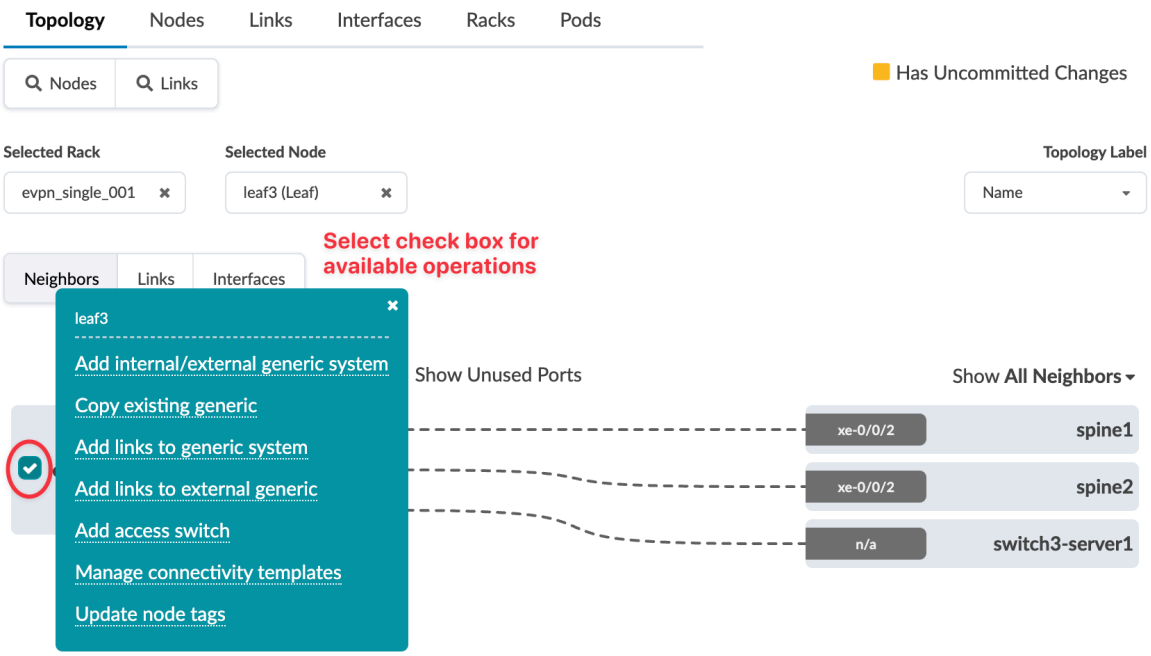
When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Add Links to Generic System

1. From the blueprint, navigate to **Staged > Physical > Topology** and select a node that can connect to a generic system.

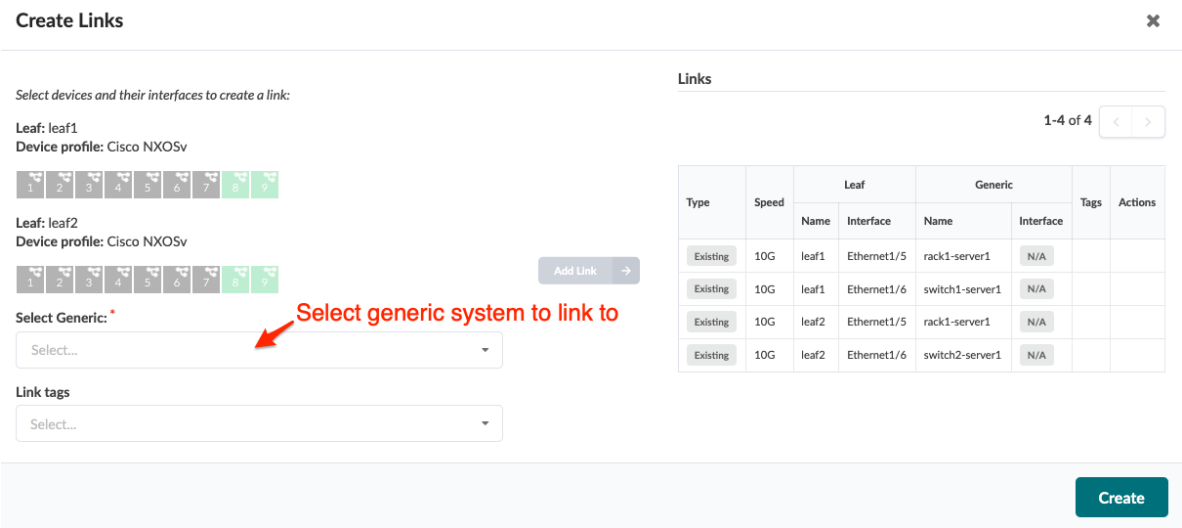


2. Select the node check box to see the operations available for that node (and that you have permissions for).

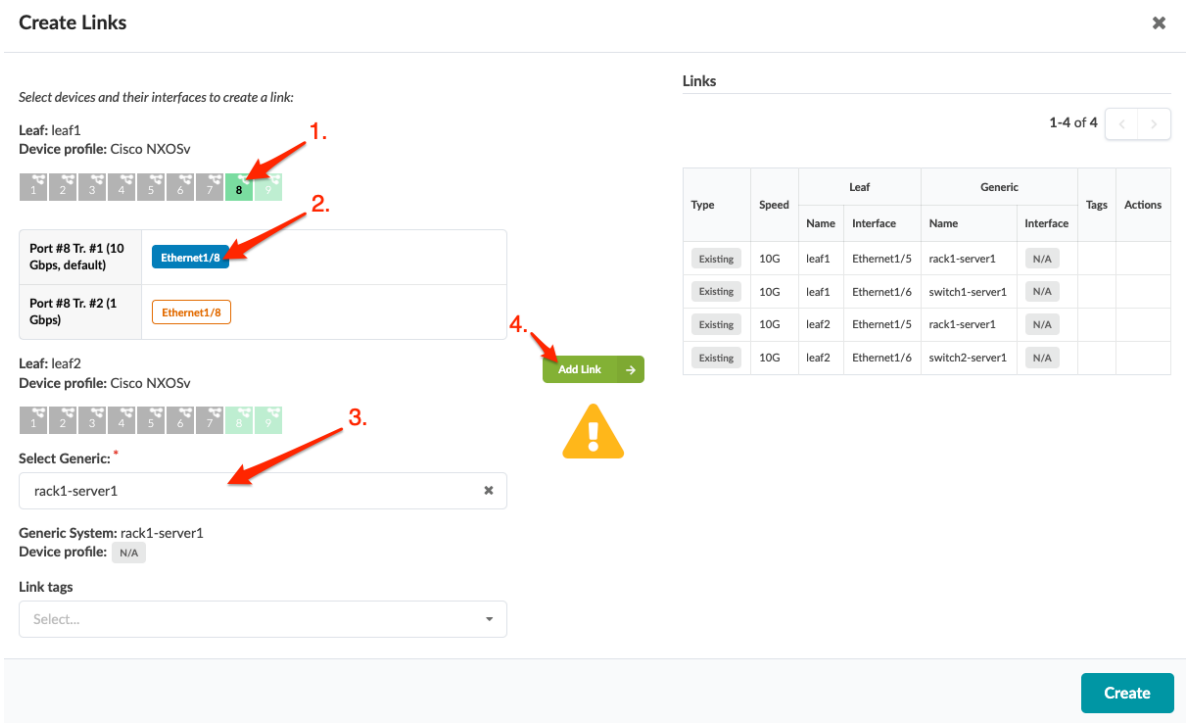


NOTE: You can also get to the selection page from the **Nodes** view. From the blueprint, navigate to **Staged > Physical > Nodes**, click the node name in the table, then click the node name that appears at the top of the **Selection** panel (on the right side of the page).

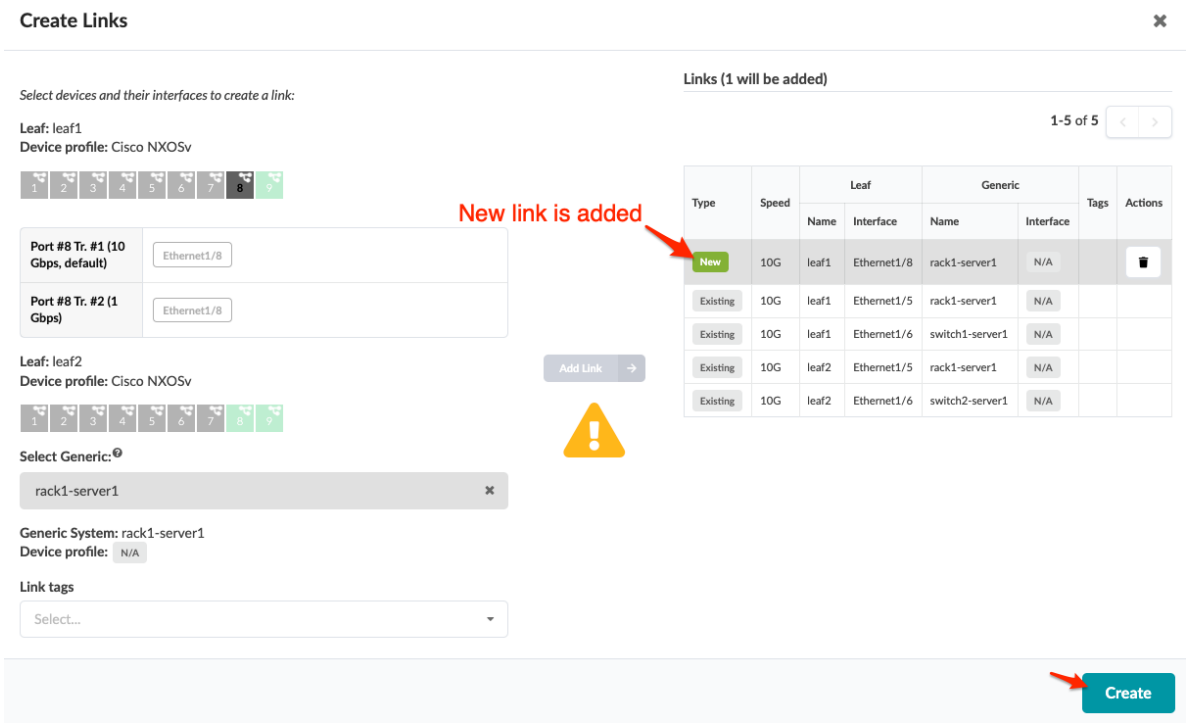
3. Click **Add links to generic system** to go to its dialog.



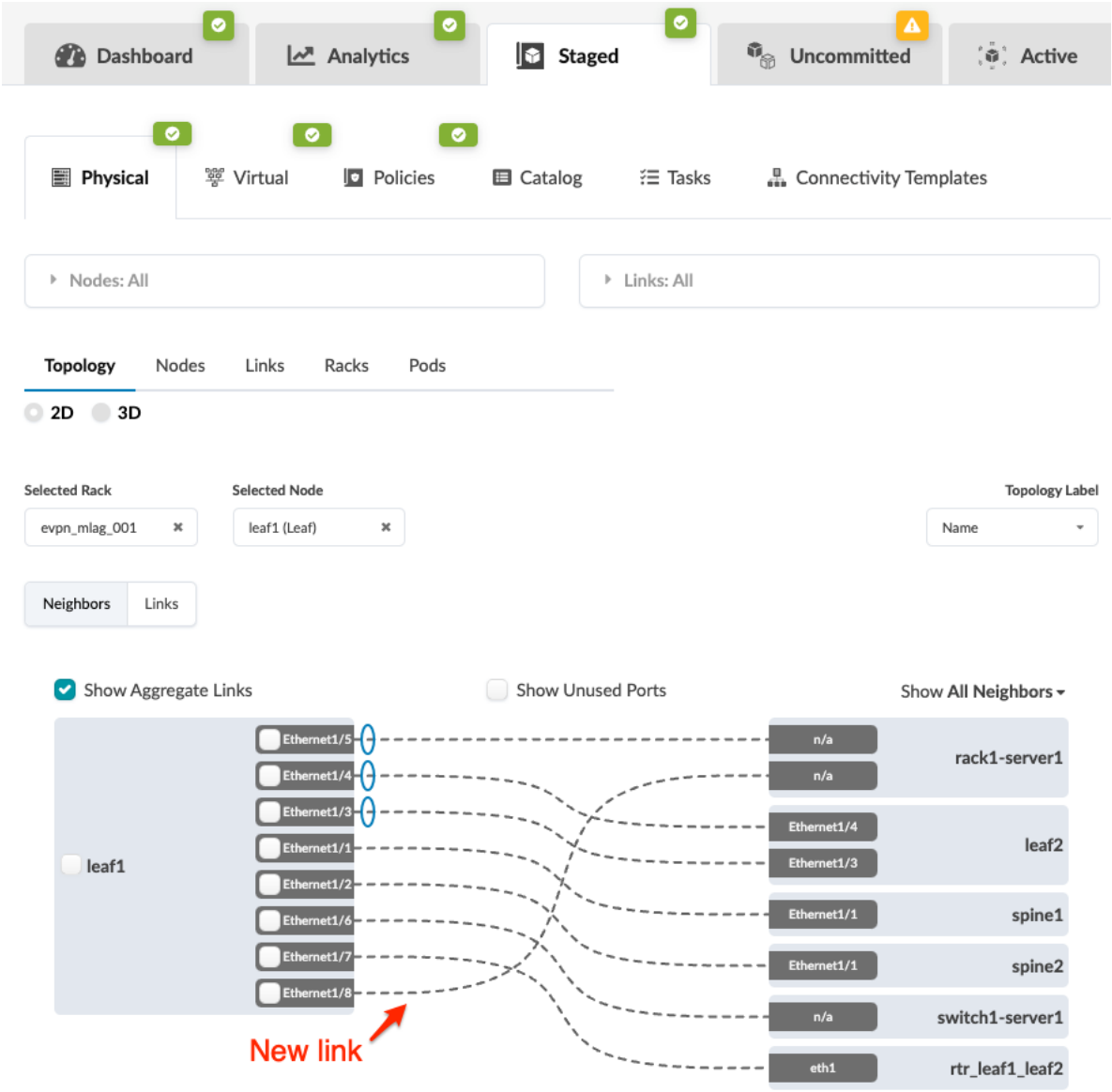
4. Select an available port, transformation, and the generic system to link to. The gray **Add Link** button turns green.



5. Click **Add Link**. The link is added to the link table.



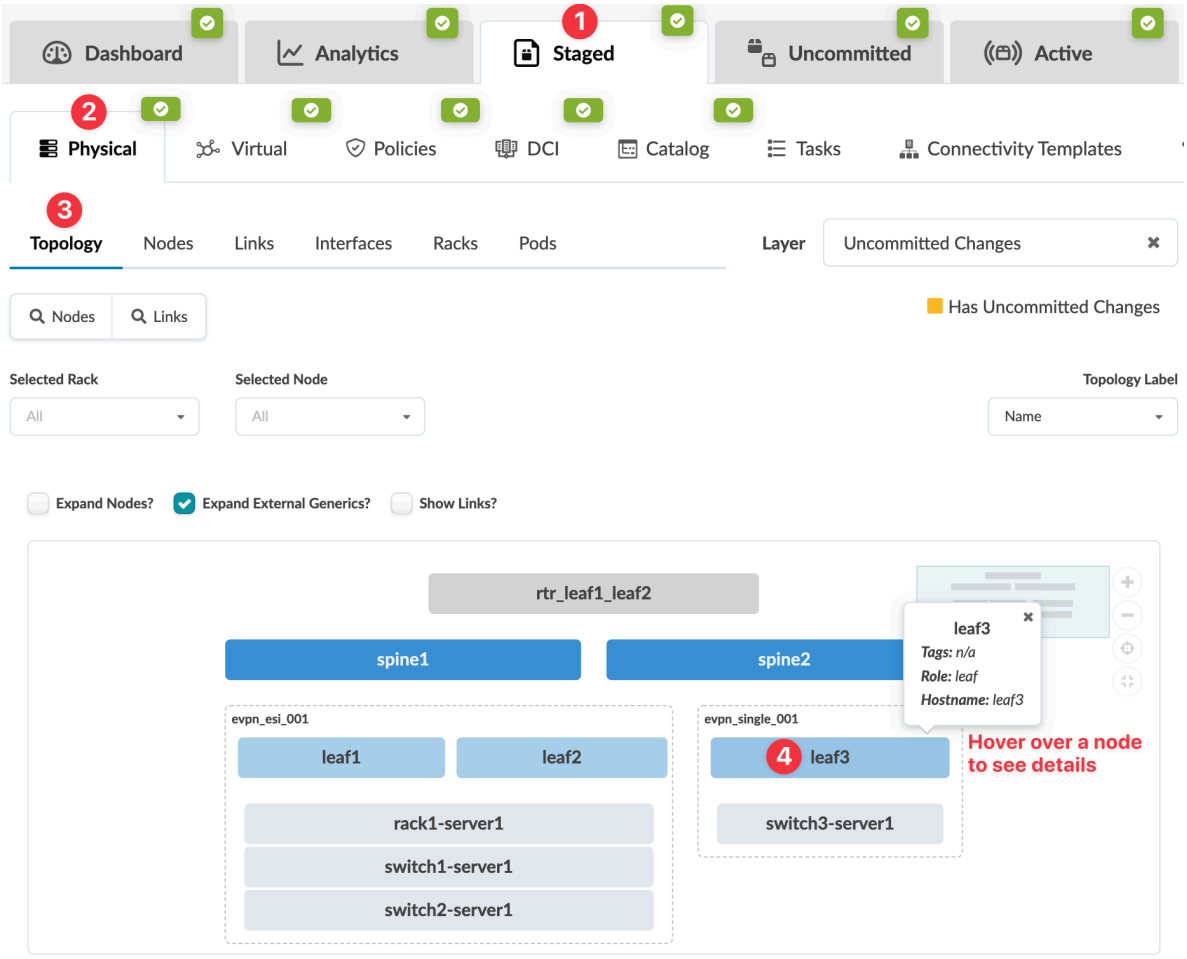
6. Click **Create** to stage the change and return to the **Topology** view.



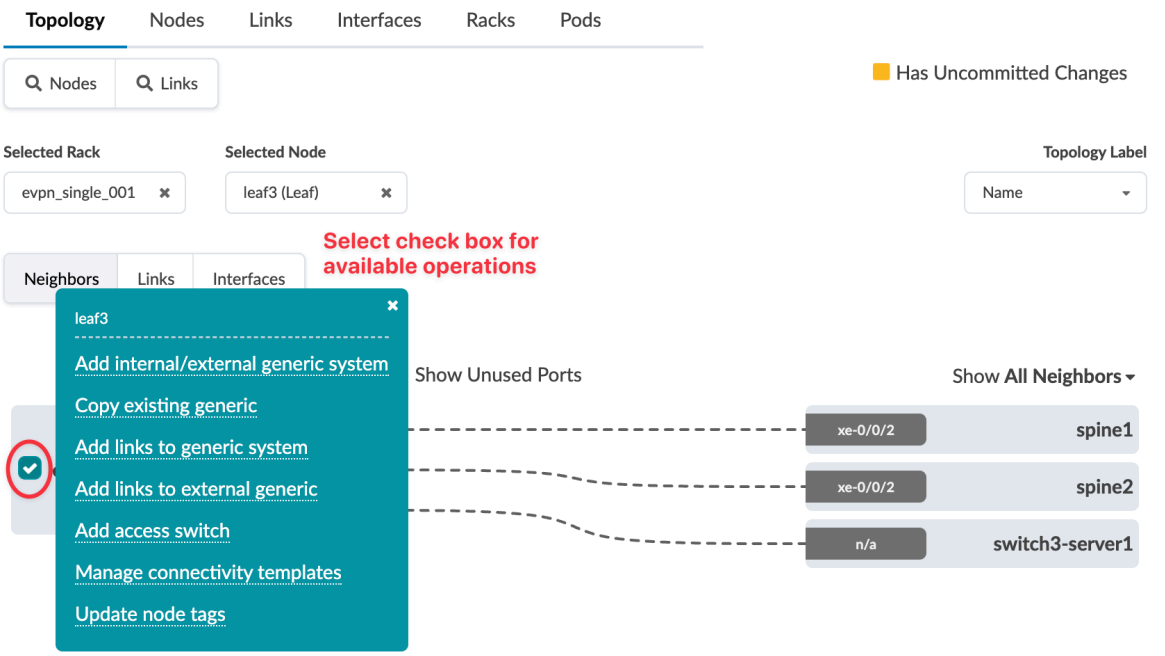
When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Add Links to External Generic System

1. From the blueprint, navigate to **Staged > Physical > Topology** and select a node that can connect to an external generic system.

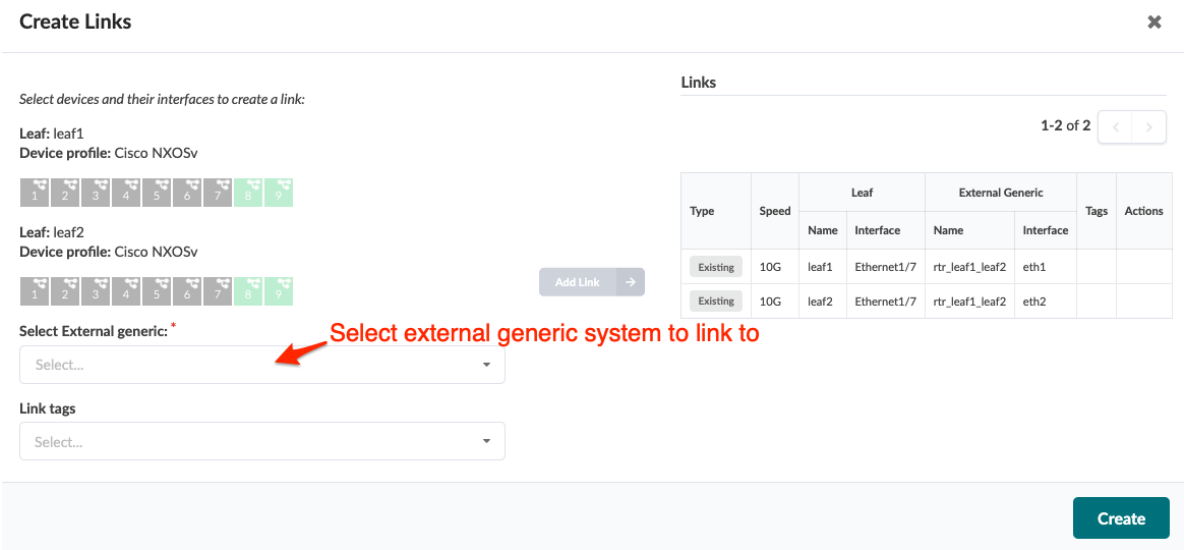


2. Select the node check box to see the operations available for that node (and that you have permissions for).

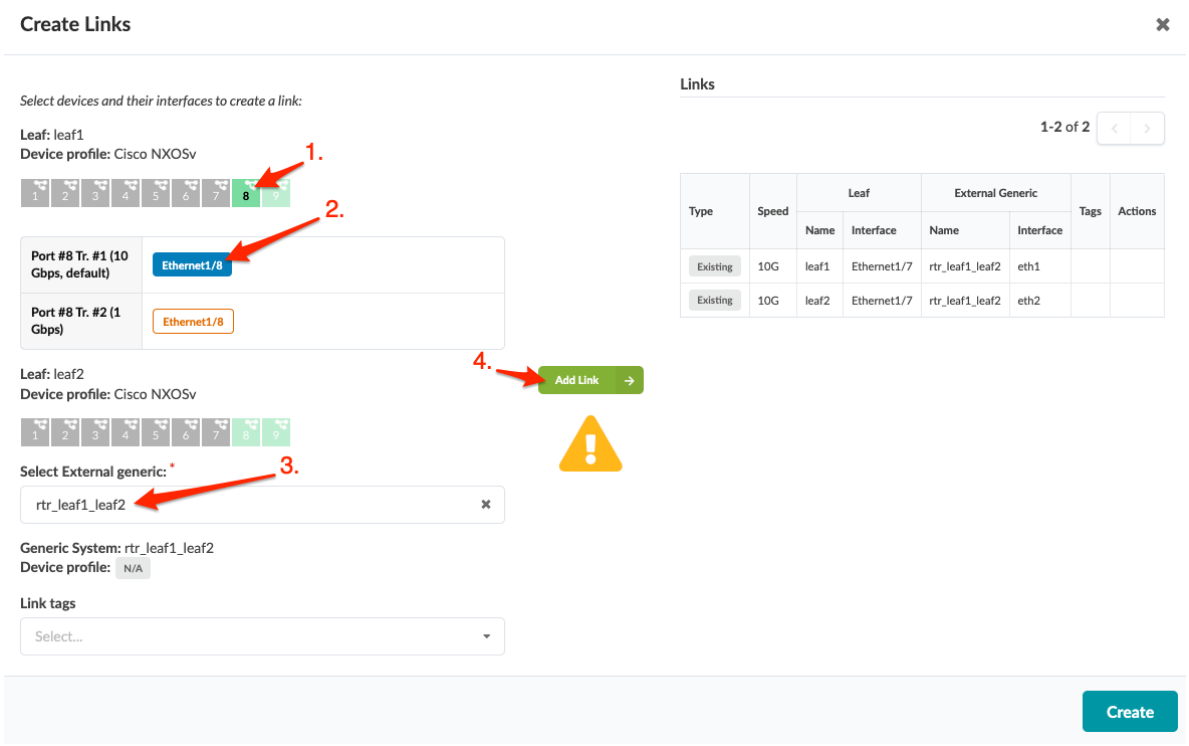


NOTE: You can also get to the selection page from the **Nodes** view. From the blueprint, navigate to **Staged > Physical > Nodes**, click the node name in the table, then click the node name that appears at the top of the **Selection** panel (on the right side of the page).

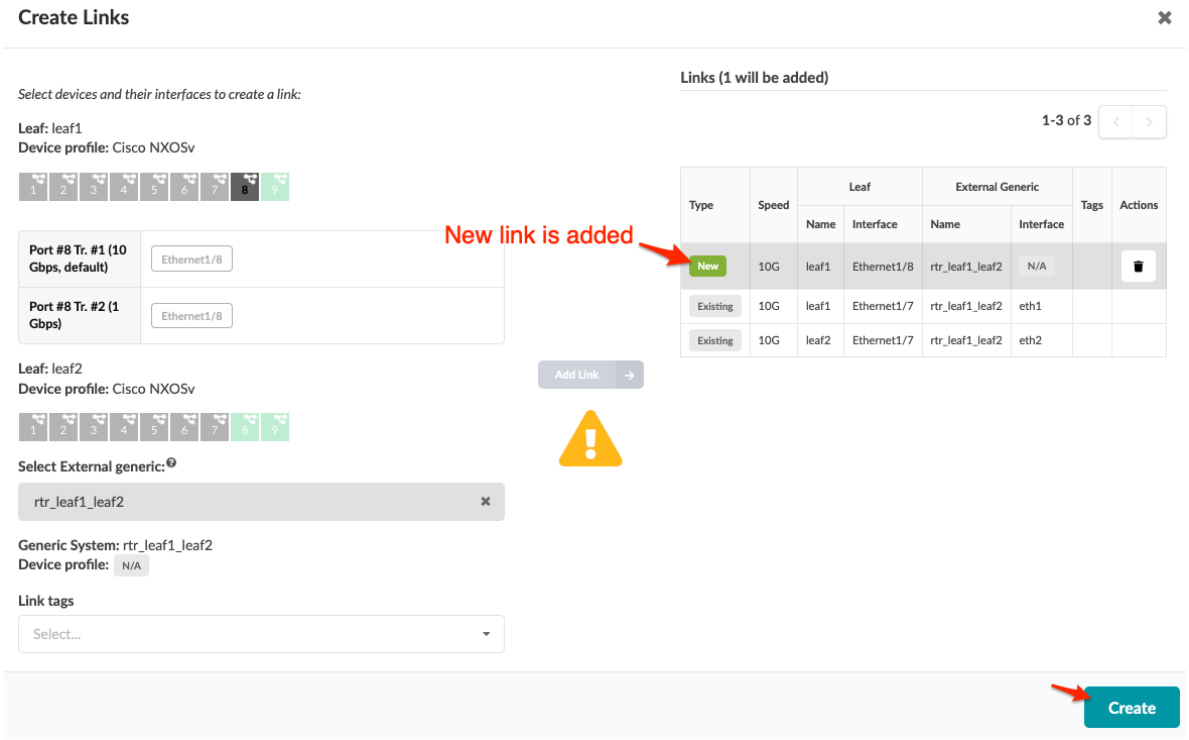
3. Click **Add links to external generic** to go to its dialog.



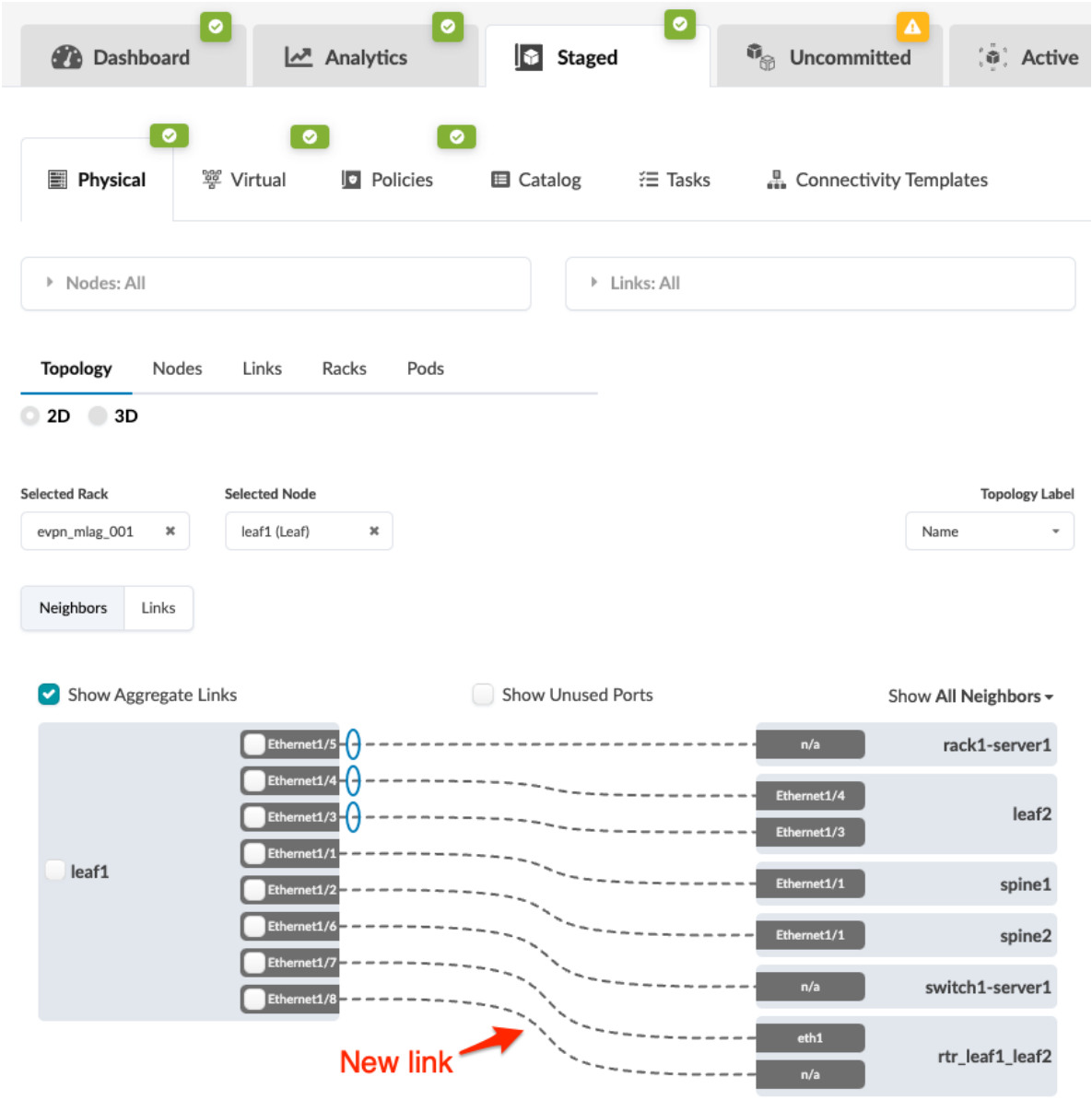
4. Select an available port, transformation, and the external generic system to link to. The gray **Add Link** button turns green.



5. Click **Add Link**. The link is added to the link table.



6. Click **Create** to stage the change and return to the **Topology** view.

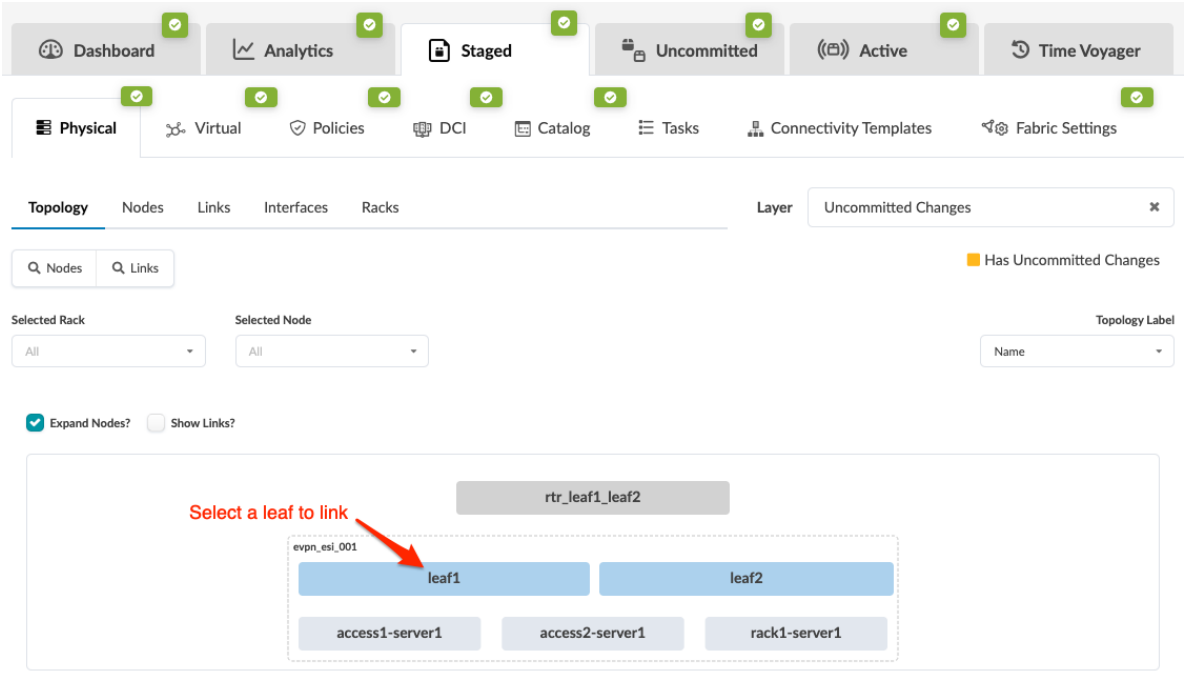


When you're ready to activate your changes, commit them from the **Uncommitted** tab.

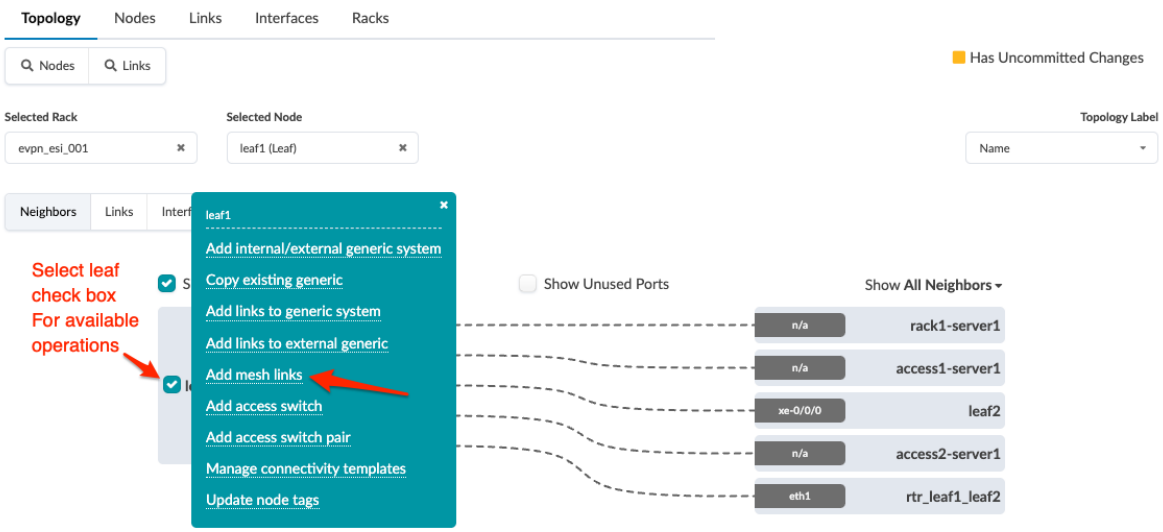
Add Mesh Links (Collapsed)

You can increase the number of mesh links in your collapsed fabric topology as a Day 2 operation. You might do this, for example, when you want to increase network capacity, or because the incorrect number of links were allocated when the template was created. As of Apstra version 5.0.0, you can perform this task from the Apstra GUI.

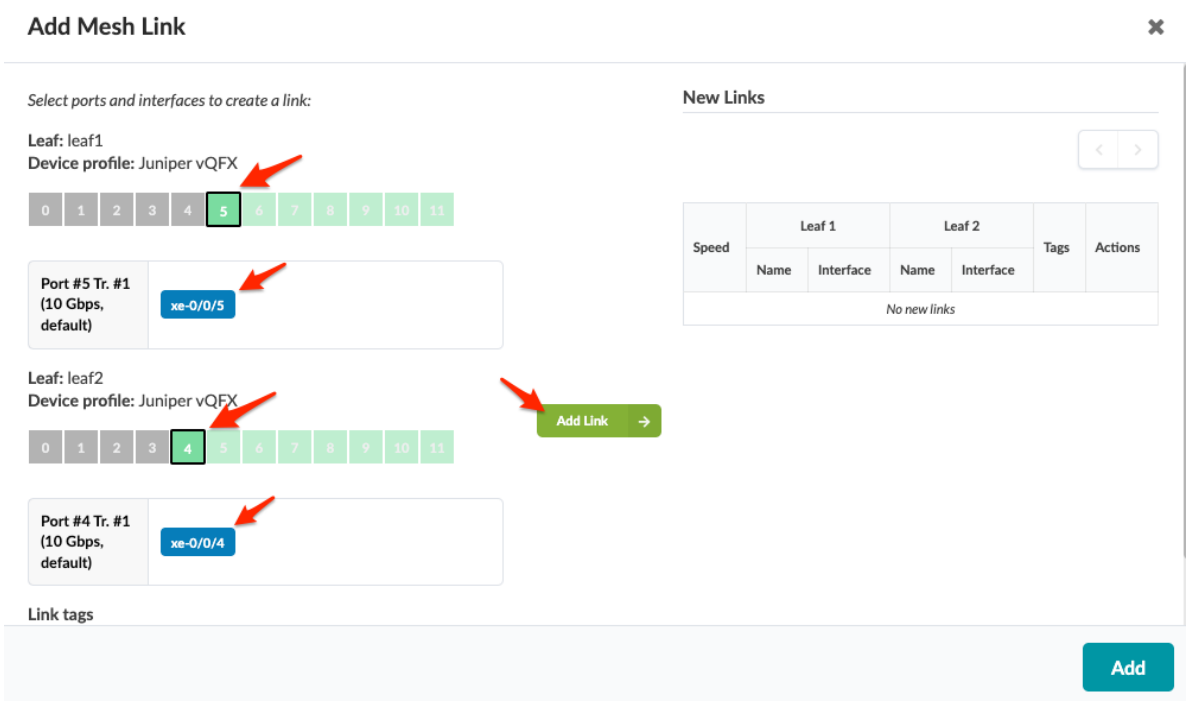
1. From the blueprint, navigate to **Staged > Physical > Topology** and select the leaf where you want to add a link.



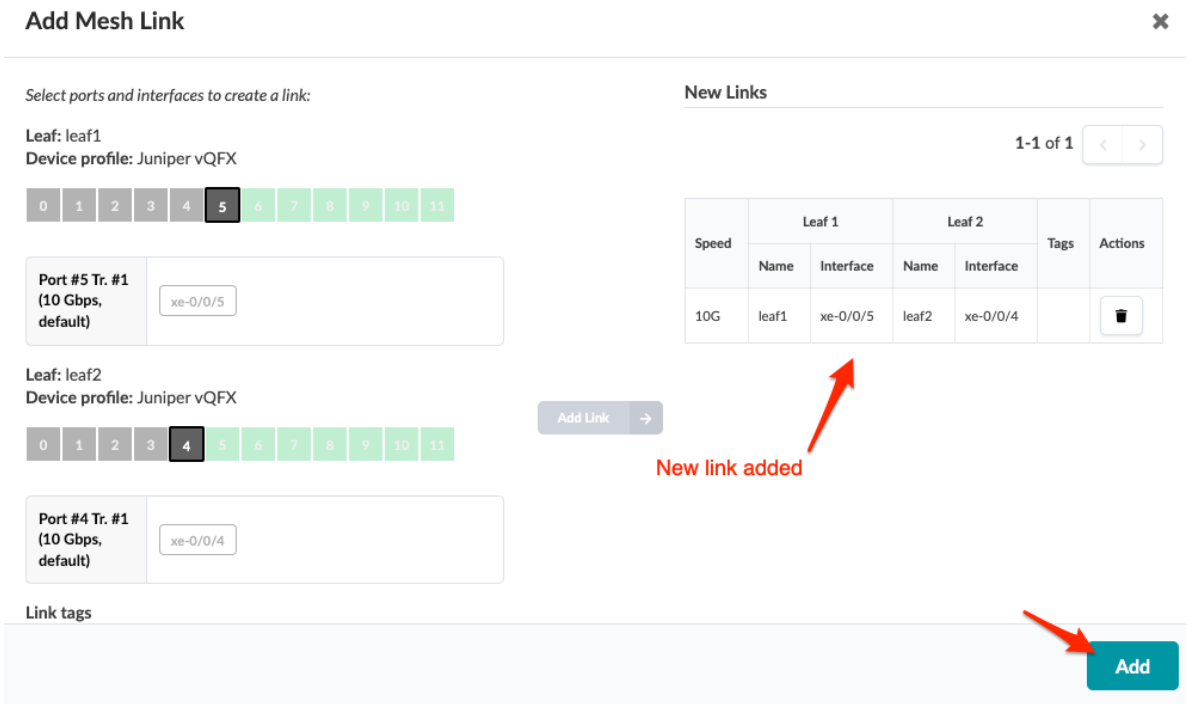
2. Select the leaf check box to see the operations available for that leaf (and that you have permissions for), then click **Add mesh links**.



3. In the dialog that opens, select the ports and transformations on each leaf, then click **Add Link**. (It'll turn green when valid ports and transformations are selected.)



4. Click **Add** to stage the addition and return to the **Topology** view.

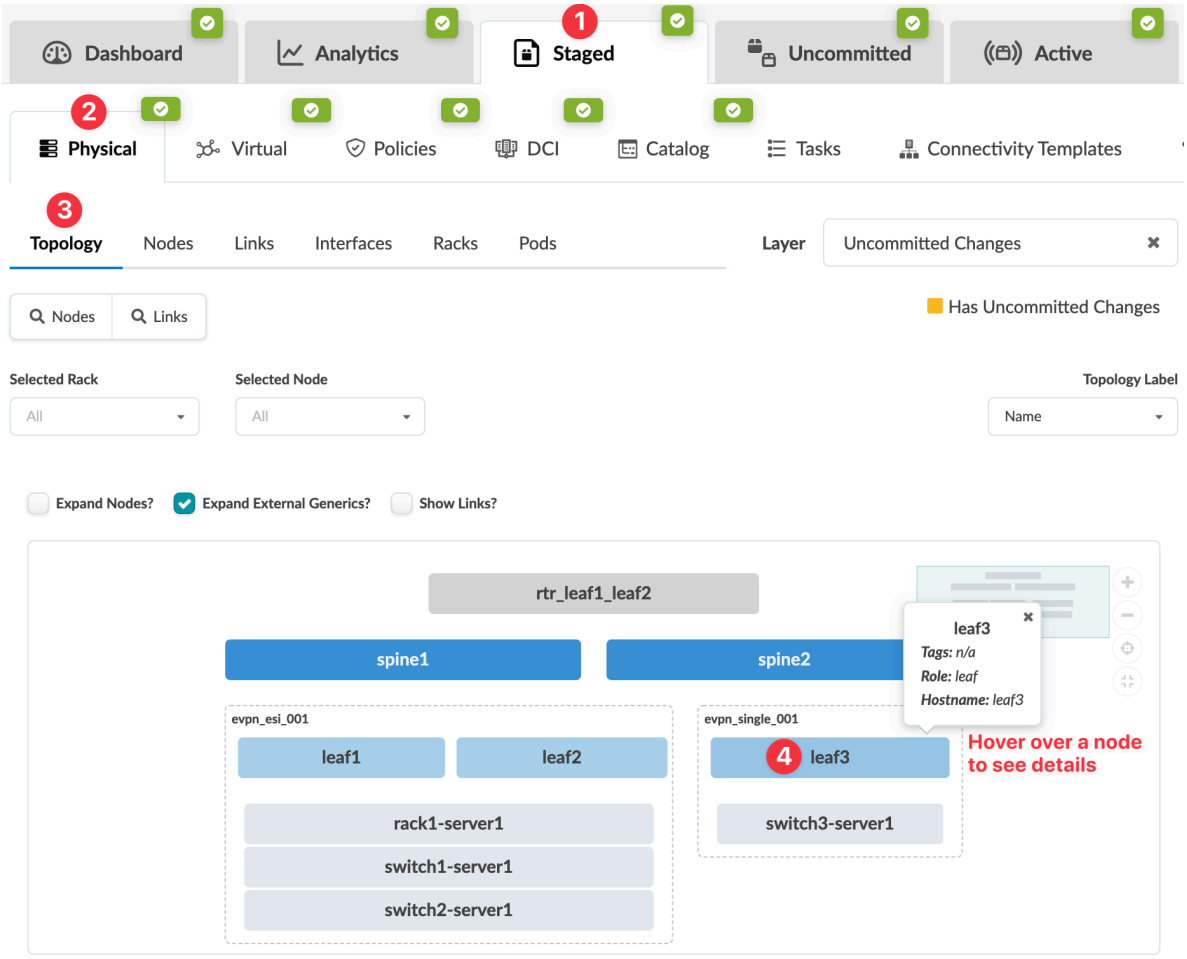


When you're ready to activate your changes, commit them from the **Uncommitted** tab.

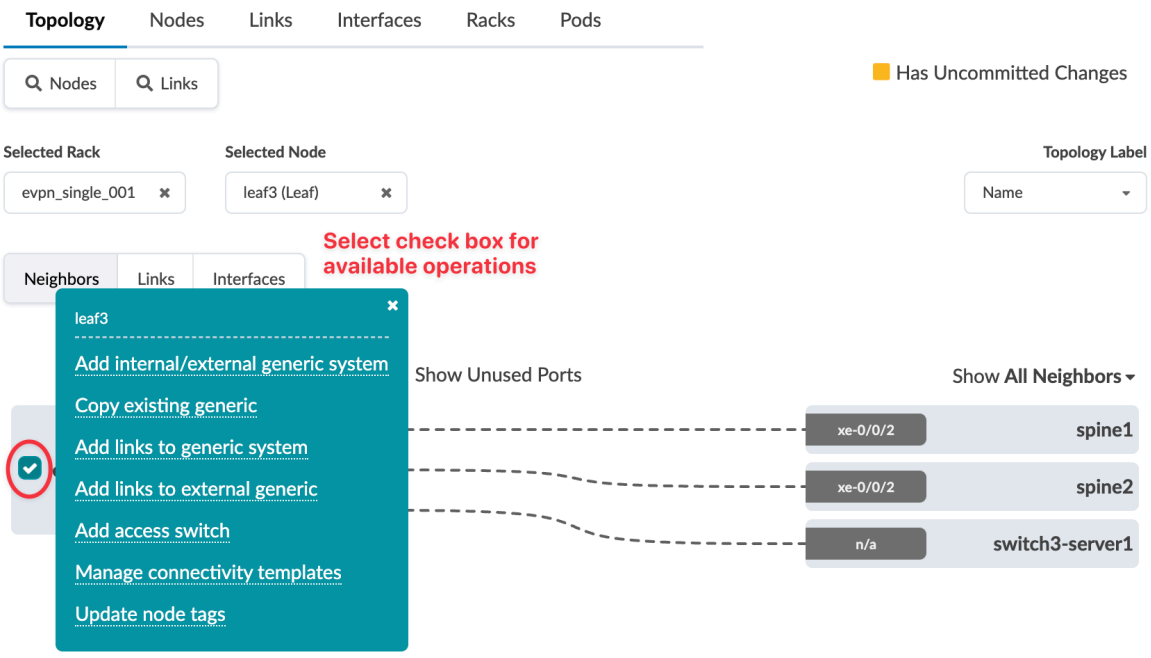
Add Leaf Peer Links

If your platform does not support it, do not attempt to create leaf peer links. Currently, Junos devices do not support any peer links, and SONiC devices do not support L3 peer links.

1. From the blueprint, navigate to **Staged > Physical > Topology** and select the MLAG member that needs a peer link.

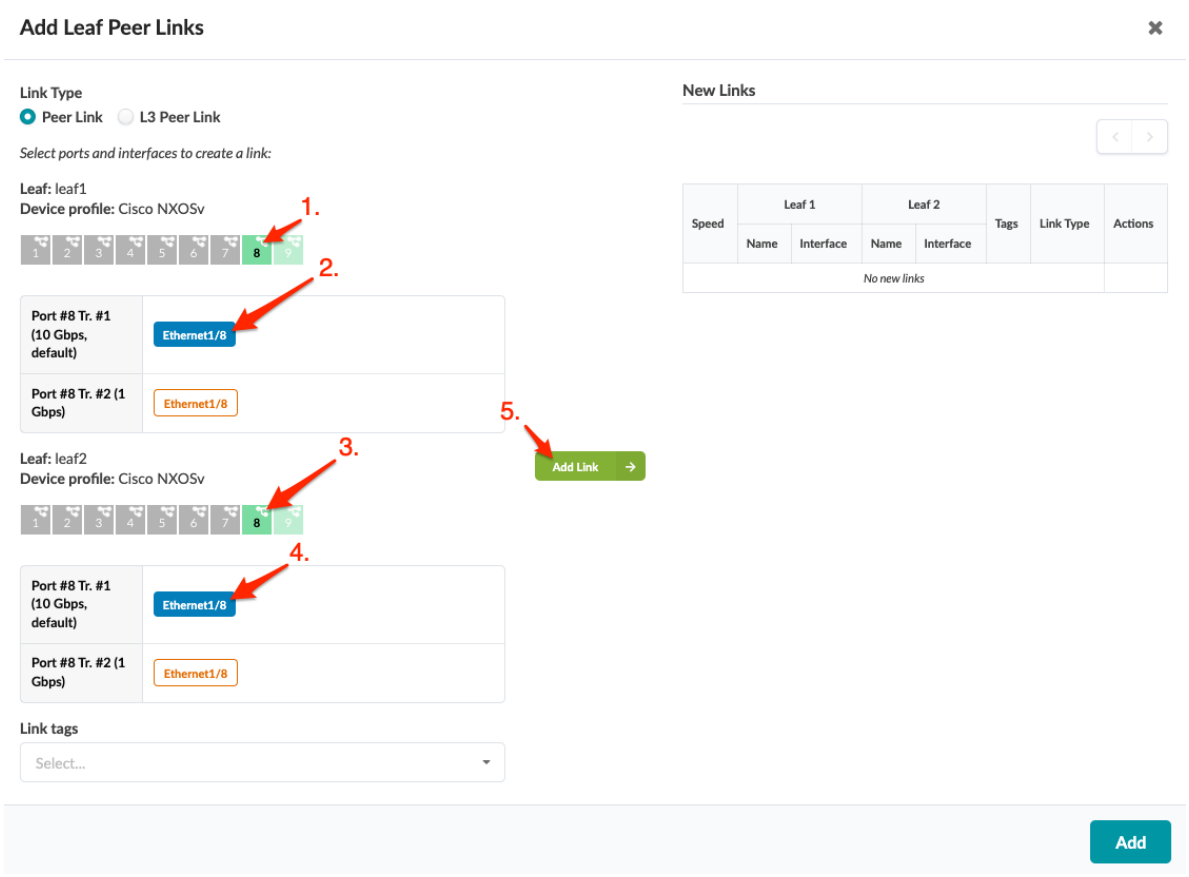


2. Select the node check box to see the operations available for that node (and that you have permissions for).

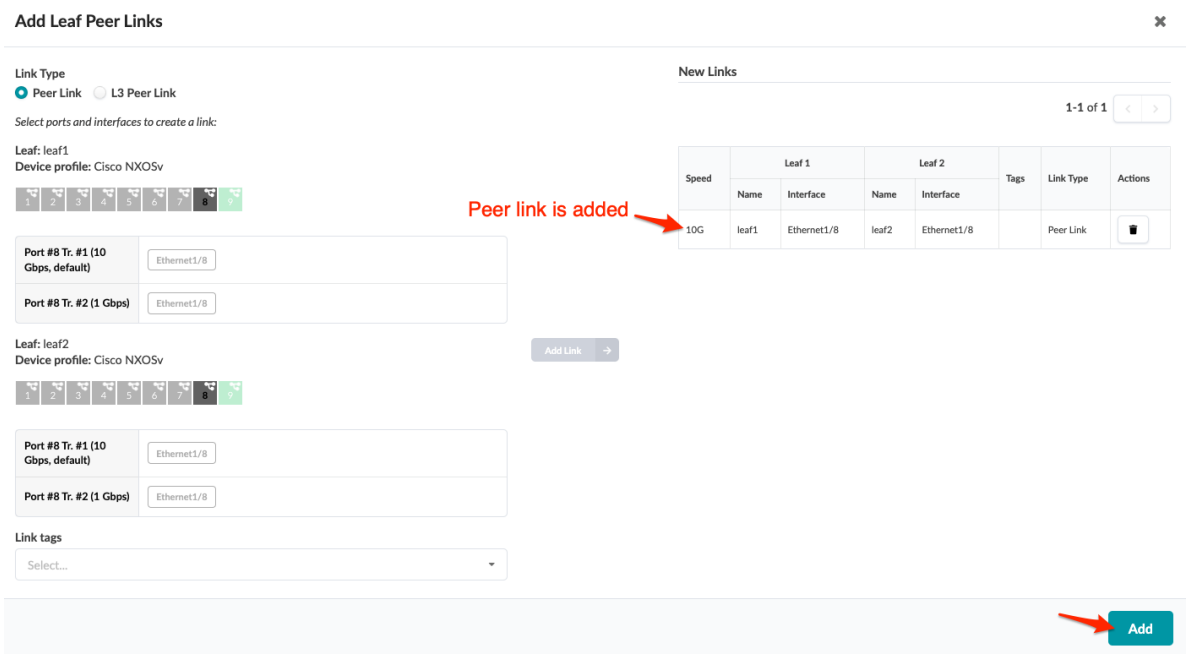


NOTE: You can also get to the selection page from the **Nodes** view. From the blueprint, navigate to **Staged > Physical > Nodes**, click the node name in the table, then click the node name that appears at the top of the **Selection** panel (on the right side of the page).

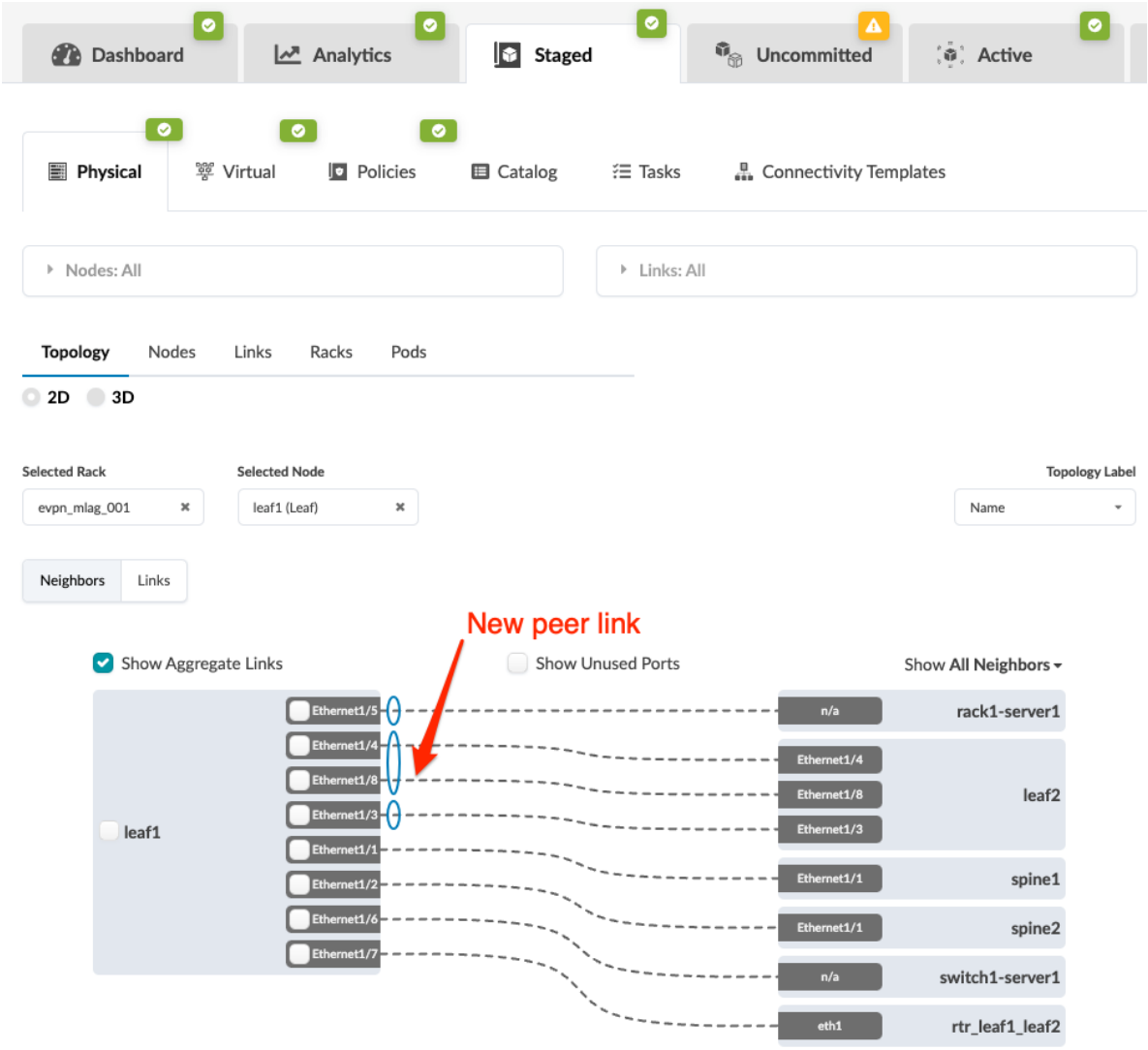
3. Click **Add leaf peer links** to go to its dialog.
4. Select the link type (peer link, L3 peer link) and an available port and transformation for each leaf member. (Only unused ports are selectable.) The gray **Add Link** button turns green.



5. Click **Add Link**. The link is added to the link table.



6. Click **Add** to stage the change and return to the **Topology** view. (BGP session is added as applicable.)

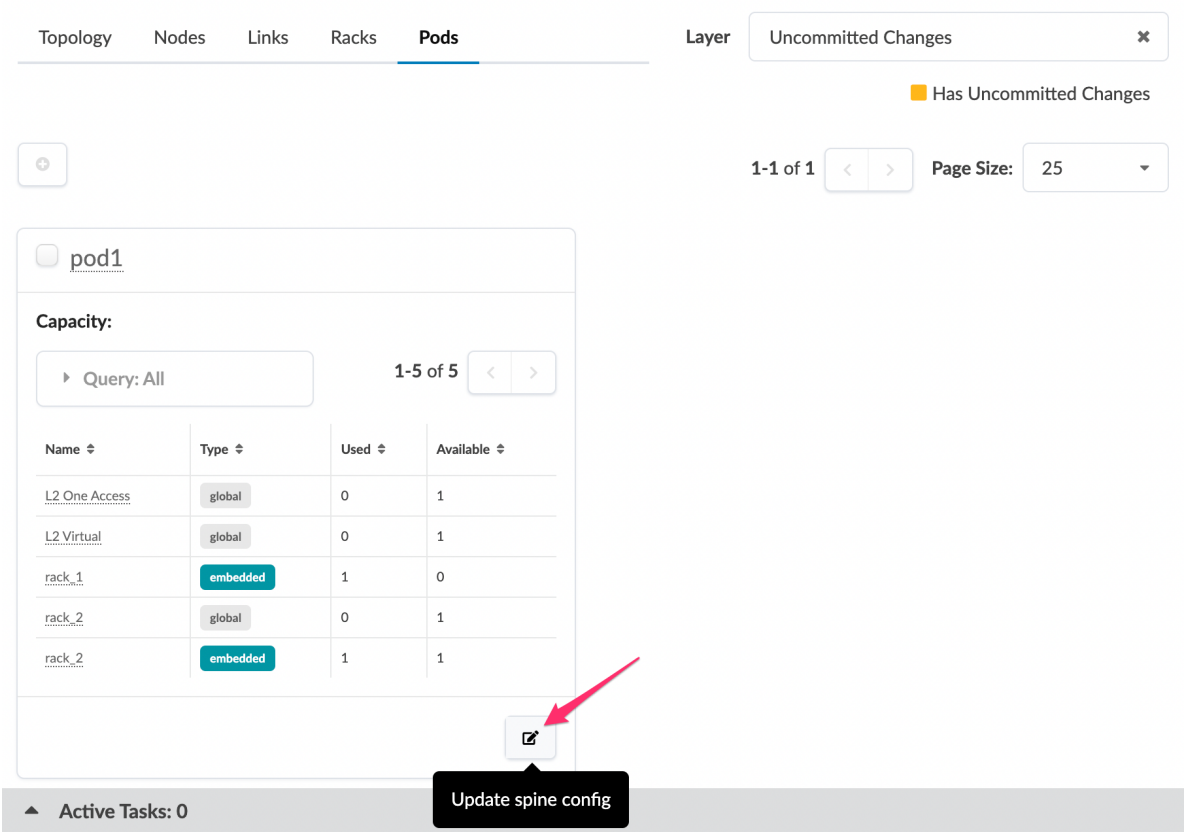


When you're ready to activate your changes, commit them from the **Uncommitted** tab.

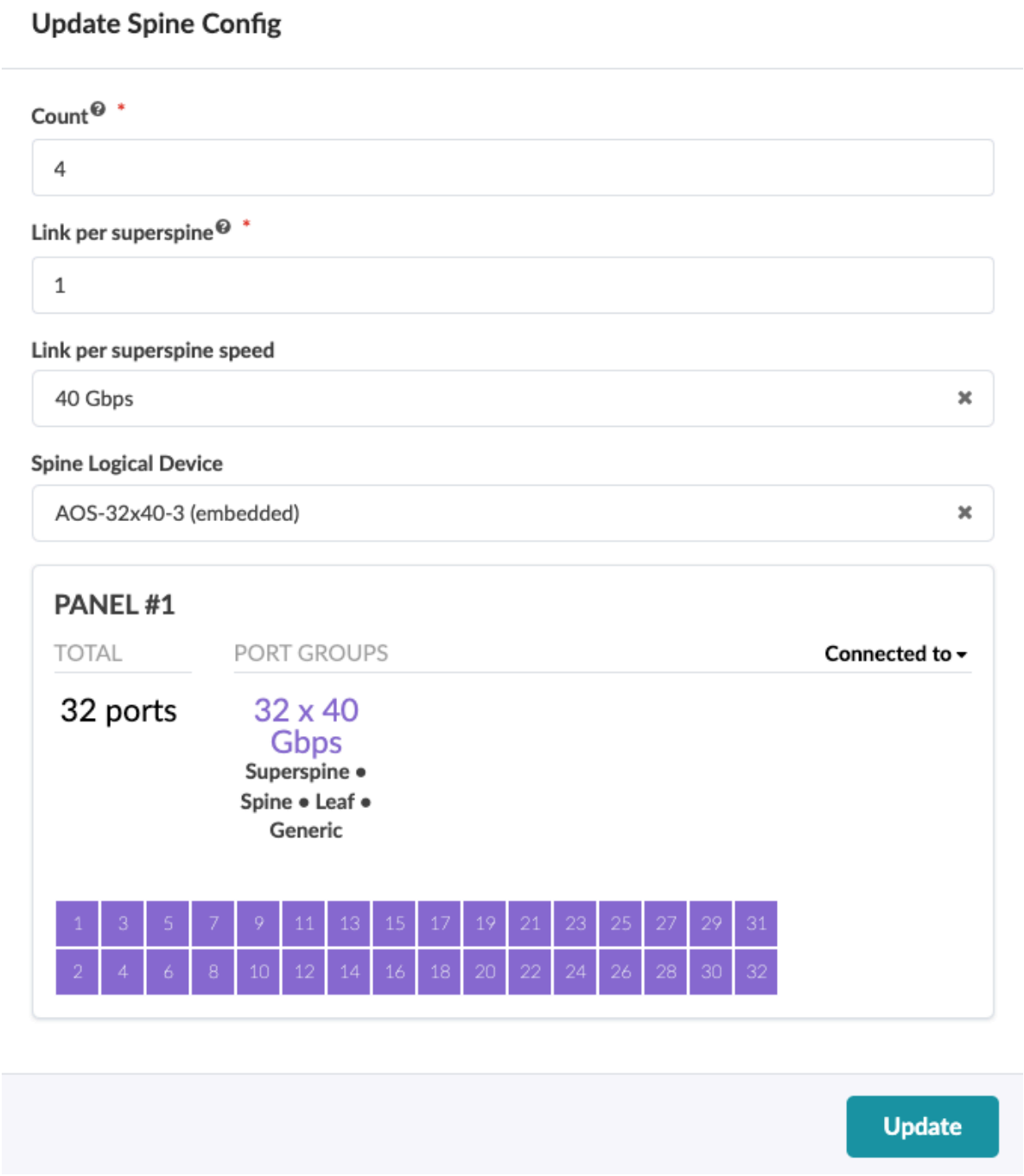
Add Link per Superspine (5-Stage)

As a Day 2 operation, you can add links per superspine on 5-stage blueprints.

1. From the blueprint, navigate to **Staged > Physical > Pods**.
2. Click the **Update spine config** button on the bottom-right of the card for the pod to change.



3. In the **Link per superspine** field, enter the total number of links you want between spines and superspines. You can only add links. Plan carefully. After you add links, you won't be able to remove them later.



4. Click **Update** to stage your changes and return to the **Pods** view.

When you're ready to activate changes, commit them from the Uncommitted tab.

RELATED DOCUMENTATION

Commit / Revert Changes to Blueprint | 599

Cabling Map

IN THIS SECTION

Export Cabling Map (Datacenter)

Data center technicians may find a printed cabling map useful when wiring in switches, or remote network operators may find it useful for viewing IP assignments. It's available in CSV and JSON formats. You can copy the contents or download the file to your local computer. If you're planning on importing the cabling map back into your blueprint, use the JSON format; you can't import a CSV file back into a blueprint.

1. From the blueprint, navigate to **Staged > Physical > Links** and click the **Export cabling map** button (second of five buttons above the links list), then select **JSON** or **CSV**. (Use the JSON format If you're planning on importing the cabling map back into your blueprint.)
2. Click **Copy** to copy the contents or click **Save As File** to download the file.
3. When you've copied or downloaded the cabling map, close the dialog to return to the **Links** view.

> **NOTE**: You can also export cabling maps from **Active > Physical > Links**.

Import Cabling Map (Datacenter)

You can export a cabling map either as a JSON file or a CSV file, but you can only import it as a JSON file.

1. From the blueprint, navigate to **Staged > Physical > Links** and click the **Import cabling map** button (first of five buttons above the links list).
2. Either click **Choose File** and navigate to the JSON file on your computer, or drag and drop the file onto the dialog window.
3. Click **Import** to import the cabling map and return to the links view.

Edit Cabling Map (Datacenter)

IN THIS SECTION

Situations when you might want to edit the cabling map include:

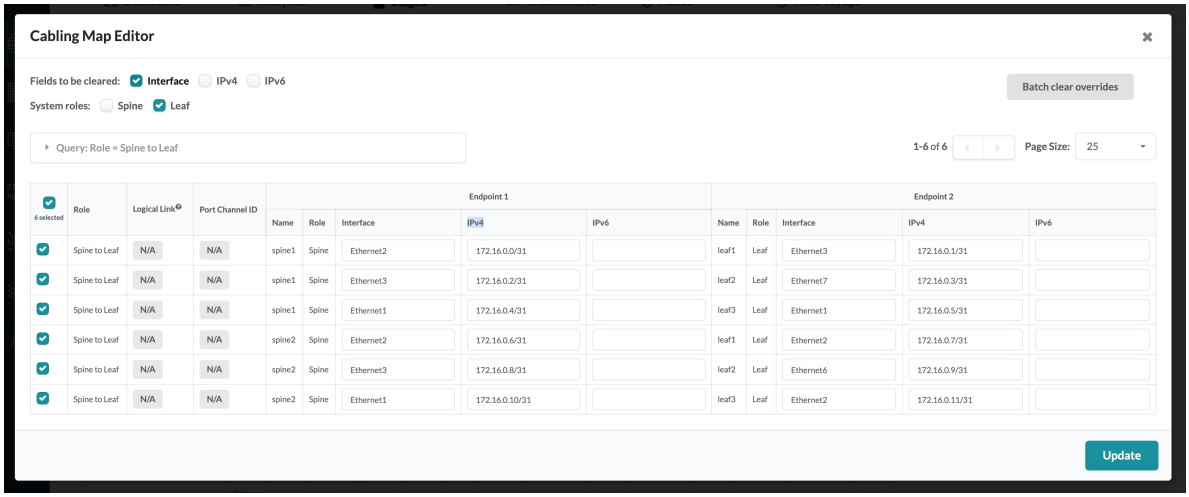
- to use existing network cabling instead of recabling to the Apstra-prescribed cabling
- to change interface names or IP addresses in the existing network cabling map
- to specify a different port from the one that the Apstra cabling algorithm selected
- to avoid the use of a defective interface

Device profiles must be assigned to blueprint nodes.

CAUTION: Overriding Apstra-generated cabling can be disruptive to the network. Use with extreme caution. For assistance with production networks, please contact "Juniper Support" on page 1374.

Edit Cabling Map (GUI)

1. From the blueprint, navigate to **Staged > Physical > Links** and click the **Edit cabling map** button (third of five buttons above the links list).
2. In the cabling map editor, change interface names and/or IP addresses, as applicable.
   - You can use **Batch clear override** to clear all Interface and IPv4/IPv6 values for a specific device type.
   - To drop the override for either an interface name or IPv4/IPv6 address, submit an empty value in the corresponding field.



3. Click **Update** to stage your changes and return to the **Links** view.

Next Steps:

When you're ready to activate your changes, commit them from the **Uncommitted**d tab.

Edit Cabling Map (JSON)

To change the cabling map using JSON, you'll export the JSON file, edit the file, then import it back into the Apstra environment.

1. From the blueprint, navigate to **Staged > Physical > Links** and click the **Export cabling map** button to see the dialog for exporting a cabling map.
2. Select **JSON** and click **Save As File** to download the file.
3. Change interface names (if_name) and/or IP addresses (ipv4_addr or ipv6_addr) in the file, as applicable. Do not change any other fields. If you do, the changes will be ignored or they will result in an error message.
4. From the cabling map (Staged > Physical > Links) click the **Import cabling map** button to see the dialog for importing a cabling map.
5. Either click **Choose File** and navigate to the revised file on your computer, or drag and drop the file onto the dialog window.
6. Click **Import**.

Next Steps:

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Link Speeds

IN THIS SECTION

Update Link Speed

IN THIS SECTION

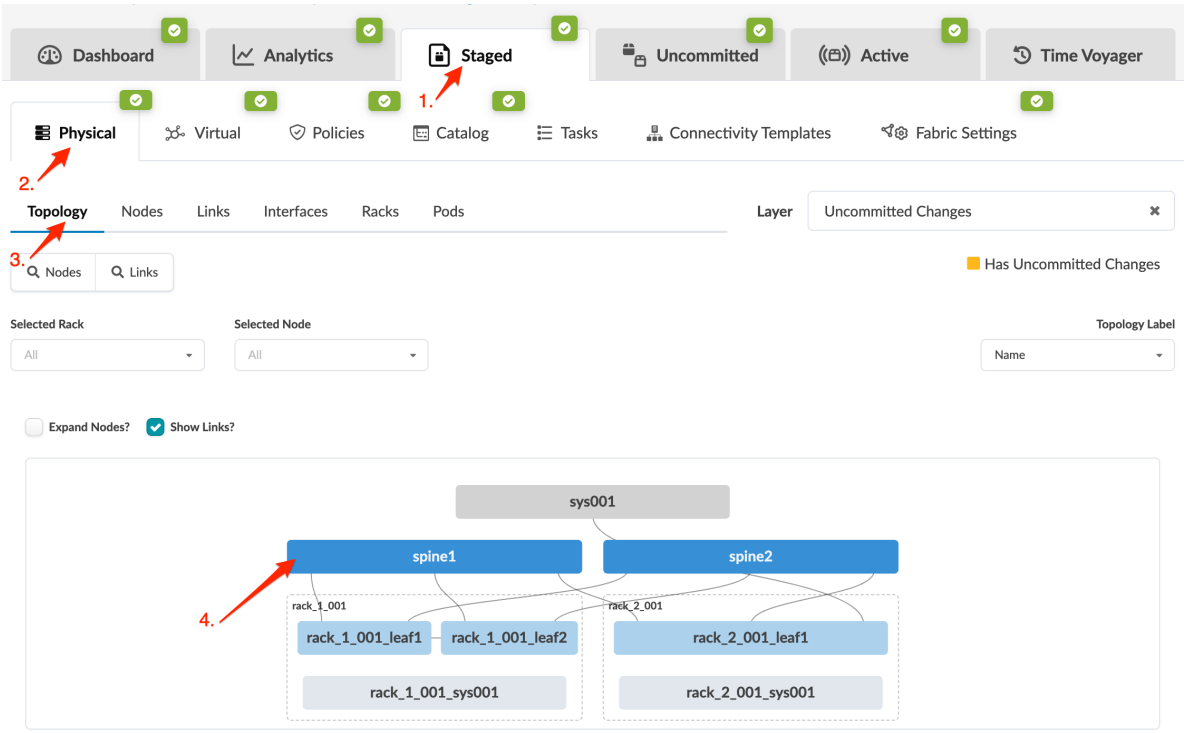
You can update link speeds between a spine device and leaf devices directly from the **Topology** view or from the **Links** view in the blueprint.

To change link speeds between spine devices and superspine devices, see "Update Link Speed per Superspine (5-Stage)" on page 176.

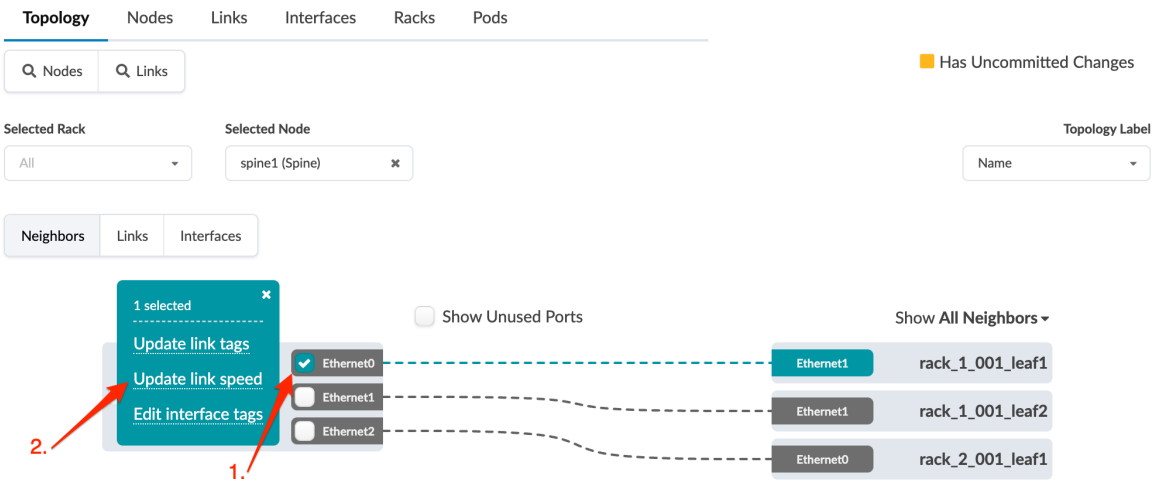
Update Link Speed (Topology View)

From the **Topology** view, you can update one link speed at a time. (You can update more than one link speed at a time from the **Links** view; see the next section.)

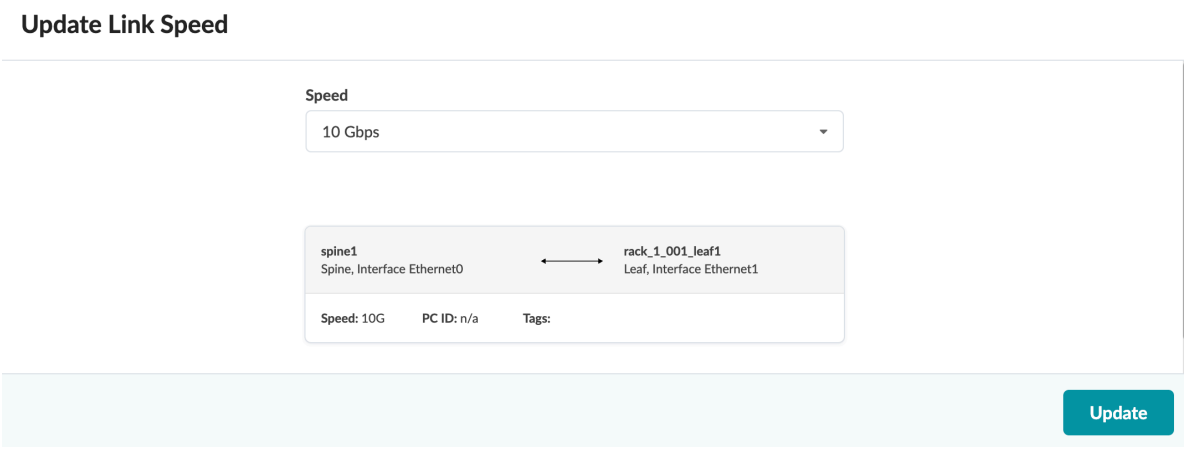
1. From the blueprint, navigate to **Staged > Physical > Topology** and select the node where you want to change link speed.



2. Select the interface check box to see the operations available for that interface (and that you have permissions for).



3. Click **Update link speed** and select the new link speed from the drop-down list. Only speeds that are available for that link/interface are listed.



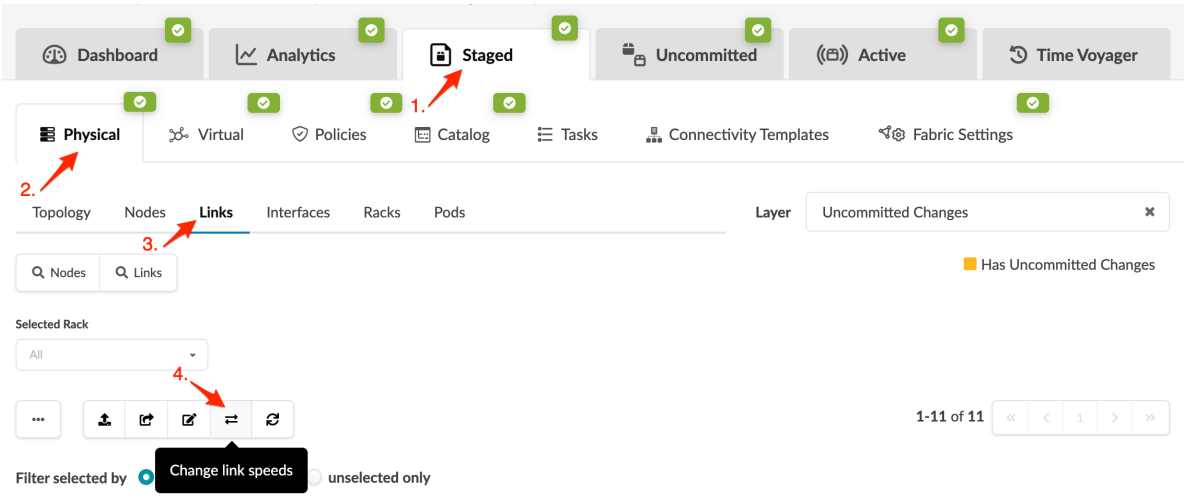
4. Click **Update** to stage your changes and return to the **Topology** view.

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

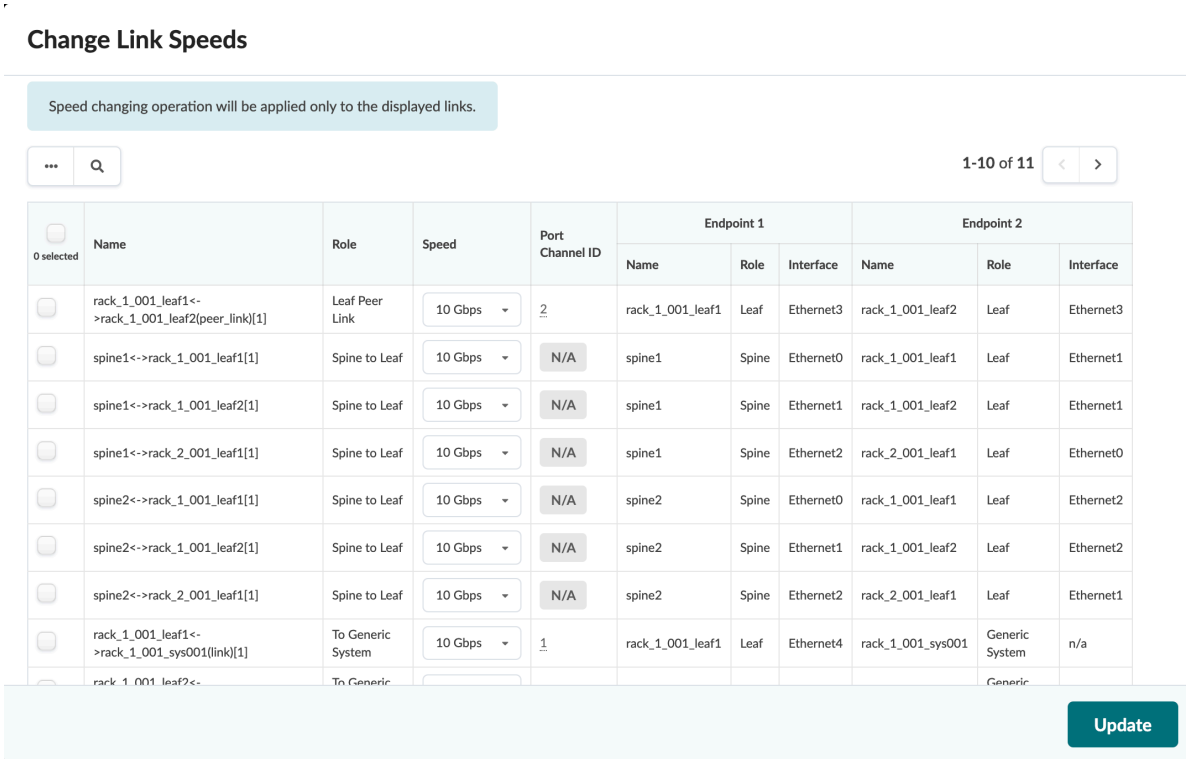
Update Link Speed (Links View)

From the **Links** view you can update one or more link speeds at the same time.

1. From the blueprint, navigate to **Staged > Physical > Links** and click the **Change link speeds** button.



2. Select new link speeds for one or more links from the **Speed** drop-down lists. Only speeds that are available for that link/interface are listed. You can change link speeds between multiple leaf devices and spine devices from this dialog. All leaf devices linked to a particular spine device must use the same link speed, so if you change a spine-to-leaf speed, be sure to change all leafs linked to that spine accordingly. This also applies to leafs in logical leaf devices (ESI pairs). They are considered as one leaf, so each leaf must use the same link speed.



Name	Role	Speed	Port Channel ID	Endpoint 1			Endpoint 2		
				Name	Role	Interface	Name	Role	Interface
rack_1_001_leaf1<->rack_1_001_leaf2(peer_link)[1]	Leaf Peer Link	10 Gbps	2	rack_1_001_leaf1	Leaf	Ethernet3	rack_1_001_leaf2	Leaf	Ethernet3
spine1<->rack_1_001_leaf1[1]	Spine to Leaf	10 Gbps	N/A	spine1	Spine	Ethernet0	rack_1_001_leaf1	Leaf	Ethernet1
spine1<->rack_1_001_leaf2[1]	Spine to Leaf	10 Gbps	N/A	spine1	Spine	Ethernet1	rack_1_001_leaf2	Leaf	Ethernet1
spine1<->rack_2_001_leaf1[1]	Spine to Leaf	10 Gbps	N/A	spine1	Spine	Ethernet2	rack_2_001_leaf1	Leaf	Ethernet0
spine2<->rack_1_001_leaf1[1]	Spine to Leaf	10 Gbps	N/A	spine2	Spine	Ethernet0	rack_1_001_leaf1	Leaf	Ethernet2
spine2<->rack_1_001_leaf2[1]	Spine to Leaf	10 Gbps	N/A	spine2	Spine	Ethernet1	rack_1_001_leaf2	Leaf	Ethernet2
spine2<->rack_2_001_leaf1[1]	Spine to Leaf	10 Gbps	N/A	spine2	Spine	Ethernet2	rack_2_001_leaf1	Leaf	Ethernet1
rack_1_001_leaf1<->rack_1_001_sys001(link)[1]	To Generic System	10 Gbps	1	rack_1_001_leaf1	Leaf	Ethernet4	rack_1_001_sys001	Generic System	n/a

3. Click **Update** to stage the changes and return to the **Links** view.

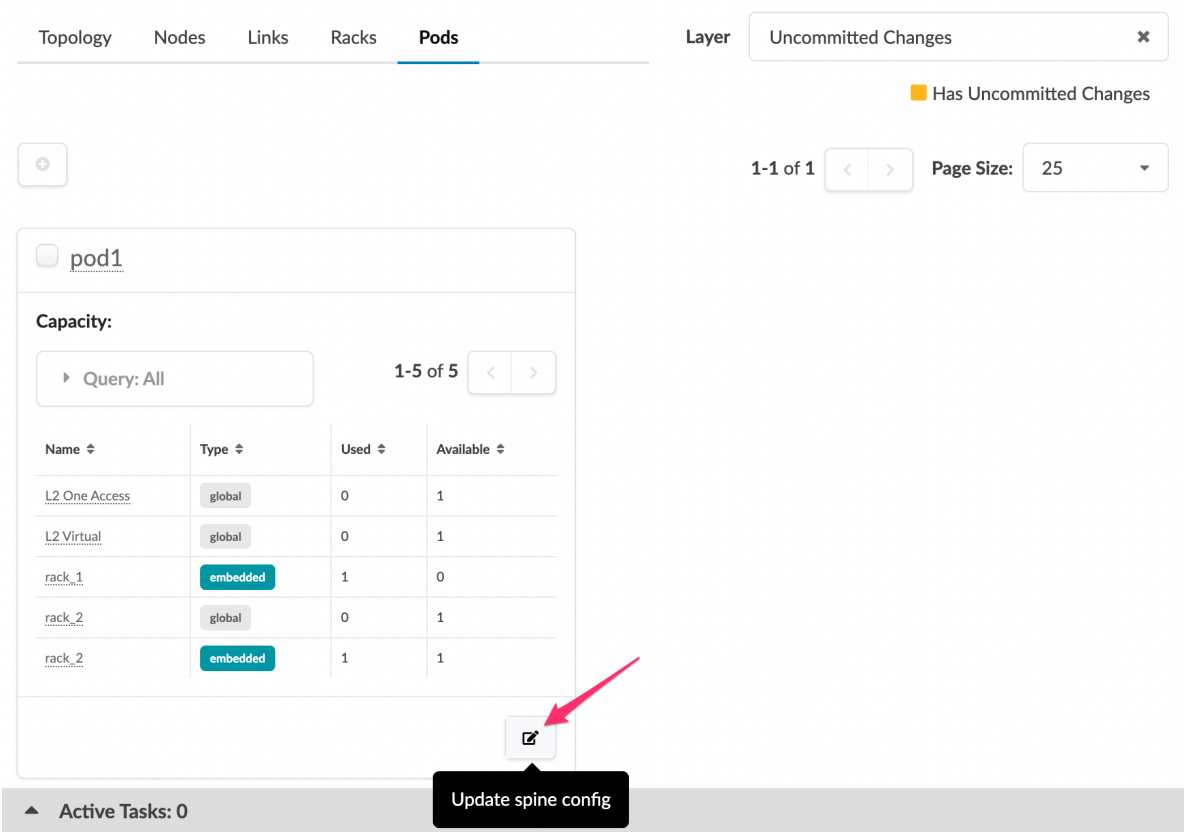
When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Update Link Speed per Superspine (5-Stage)

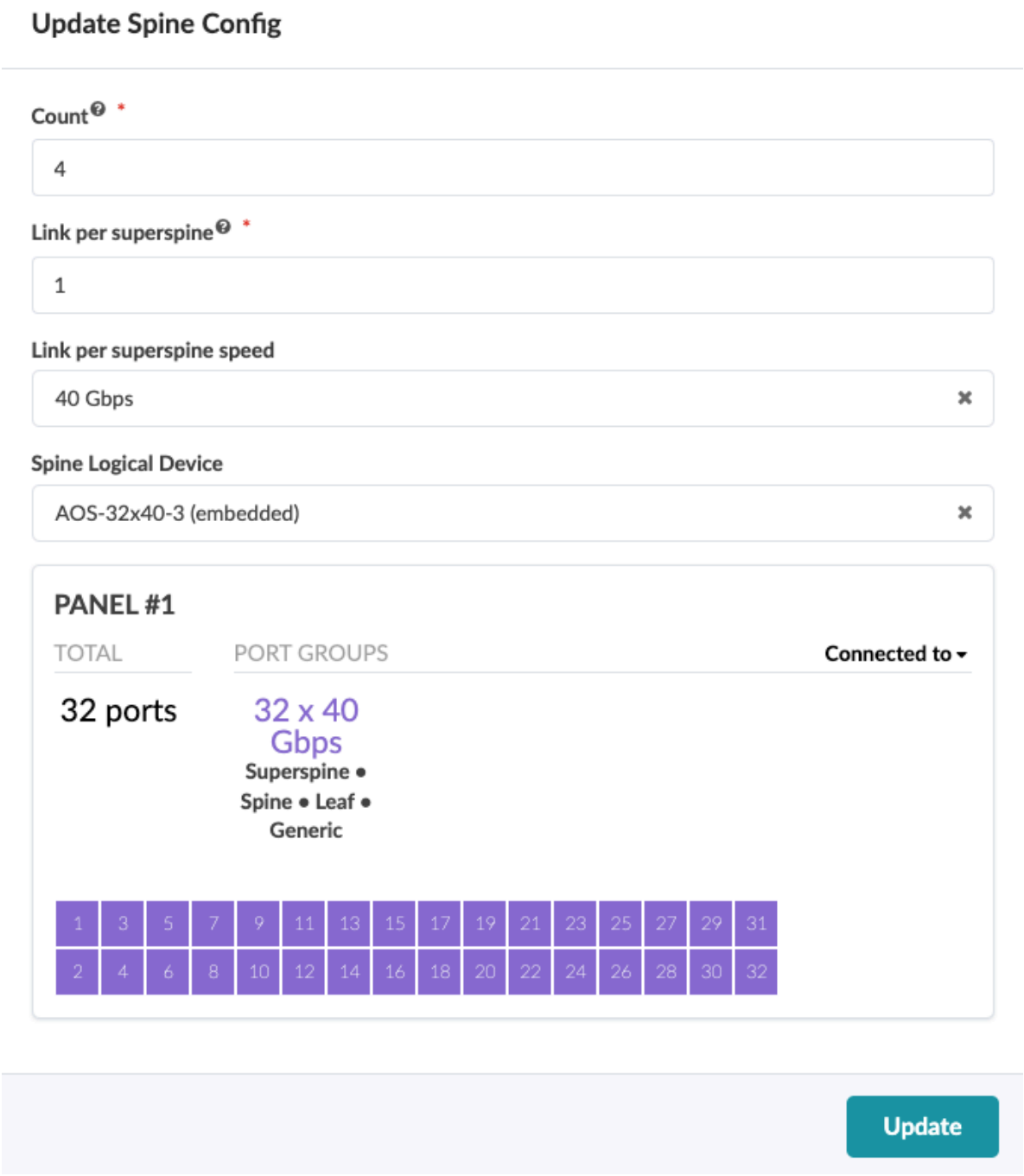
As a Day 2 operation, you can change the link speed between spines and superspines on 5-stage blueprints.

Make sure link speed is supported on the links / ports (speeds must be part of the port transformations).

1. From the blueprint, navigate to **Staged > Physical > Pods**.
2. Click the **Update spine config** button on the bottom-right of the card for the pod to change.



3. In the **Link per superspine speed** drop-down list, select the new link speed.



4. Click **Update** to stage your changes and return to the **Pods** view.

When you're ready to activate changes, commit them from the Uncommitted tab.

RELATED DOCUMENTATION

Commit / Revert Changes to Blueprint | 599

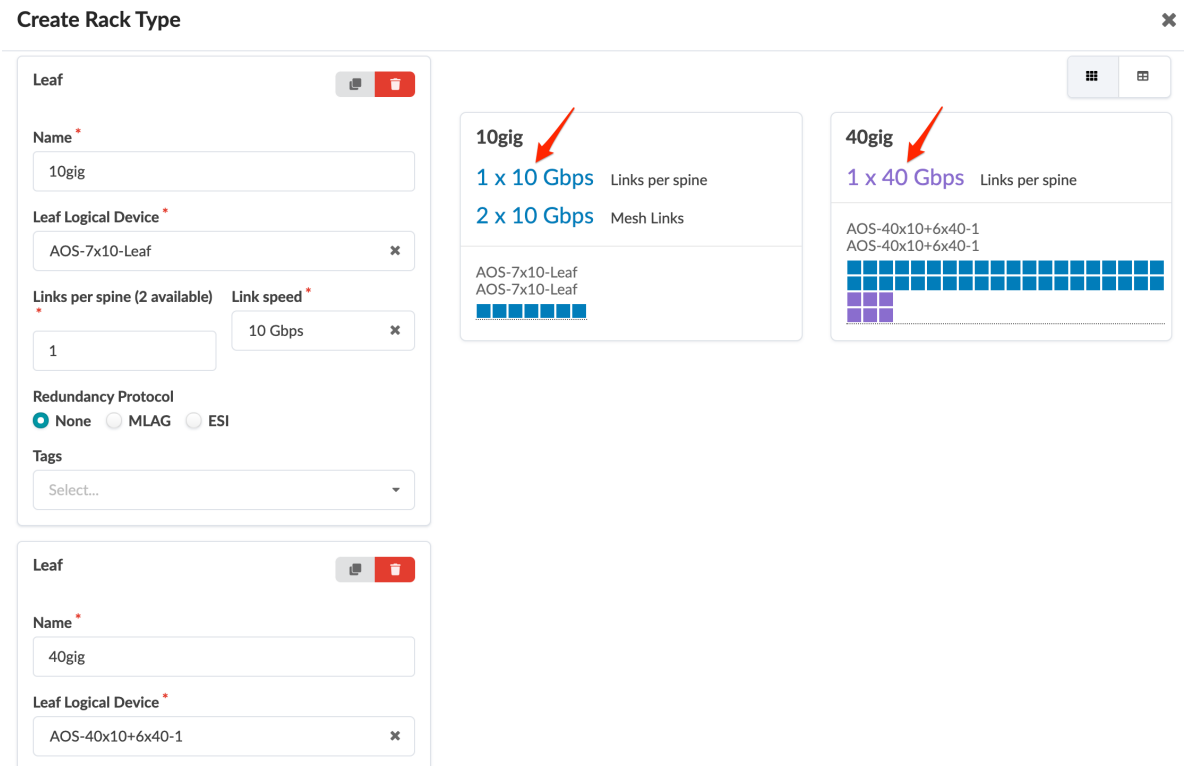
Mixed Link Speeds between Leaf and Spine

The leaf devices in your racks can have different uplink speeds to a spine. When designing for mixed speeds, make sure you plan sufficient ports for spine-to-leaf connections with mixed link speeds for Day 0, and for adding racks as a Day 2 operation. The spine logical device must have mixed port speeds defined that specify the port role as **Leaf** for the required number of ports. The following limitations apply:

- Parallel links between the same devices cannot have mixed speeds.
- You can't update spine logical devices if they're used in a blueprint. You could possibly use the AOS-CLI utility for manual patching. AOS-CLI is an experimental tool and it may not be able to provide a solution. For assistance, contact "Juniper Support" on page 1374.

The example below shows how to design rack types and templates with mixed speeds.

1. Create an **L3 Clos** rack type with logical devices **AOS-7x10-Leaf** and **AOS-40x10+6x40-1** for two leaf switches, having 10 GbE and 40GbE, respectively, as uplinks towards spine devices



Logical Device Preview

Name

AOS-7x10-Leaf

PANEL #1

TOTAL	PORT GROUPS	Connected to
7 ports	2 x 10 Gbps — Spine • Leaf 2 x 10 Gbps — Peer 2 x 10 Gbps — Access • Generic 1 x 10 Gbps — Generic	

1 2 3 4 5 6 7

Logical Device Preview

Name

AOS-40x10+6x40-1

PANEL #1

TOTAL	PORT GROUPS	Connected to
40 ports	40 x 10 Gbps — Access • Peer • Generic	

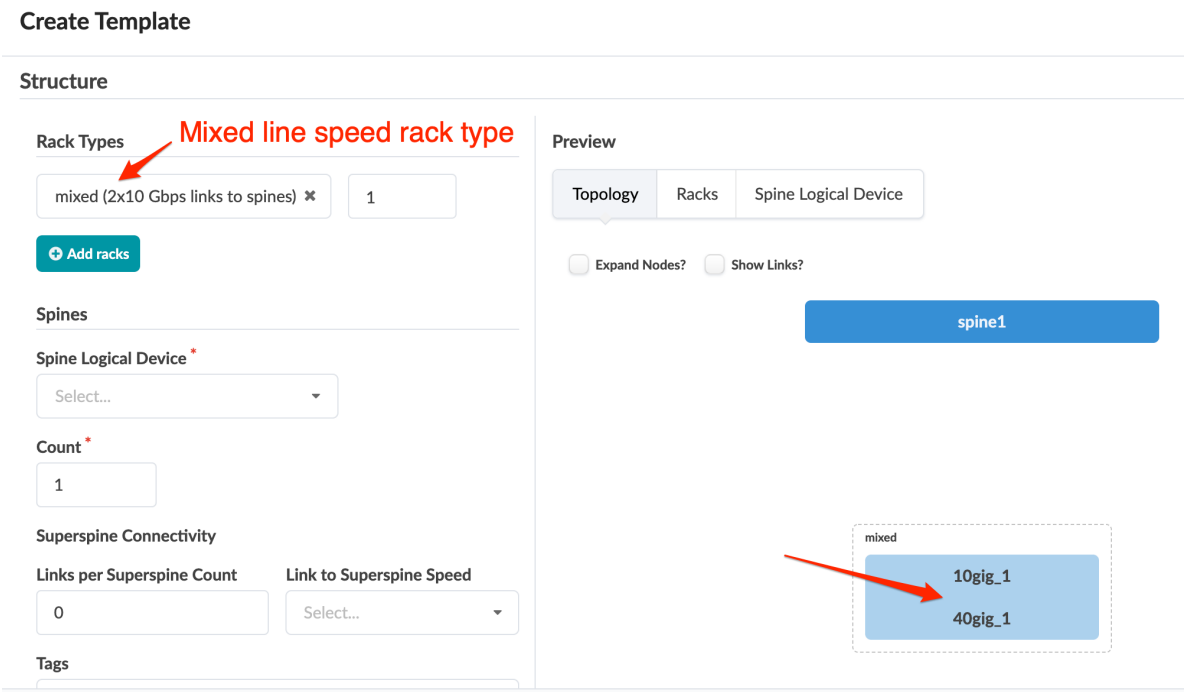
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40

PANEL #2

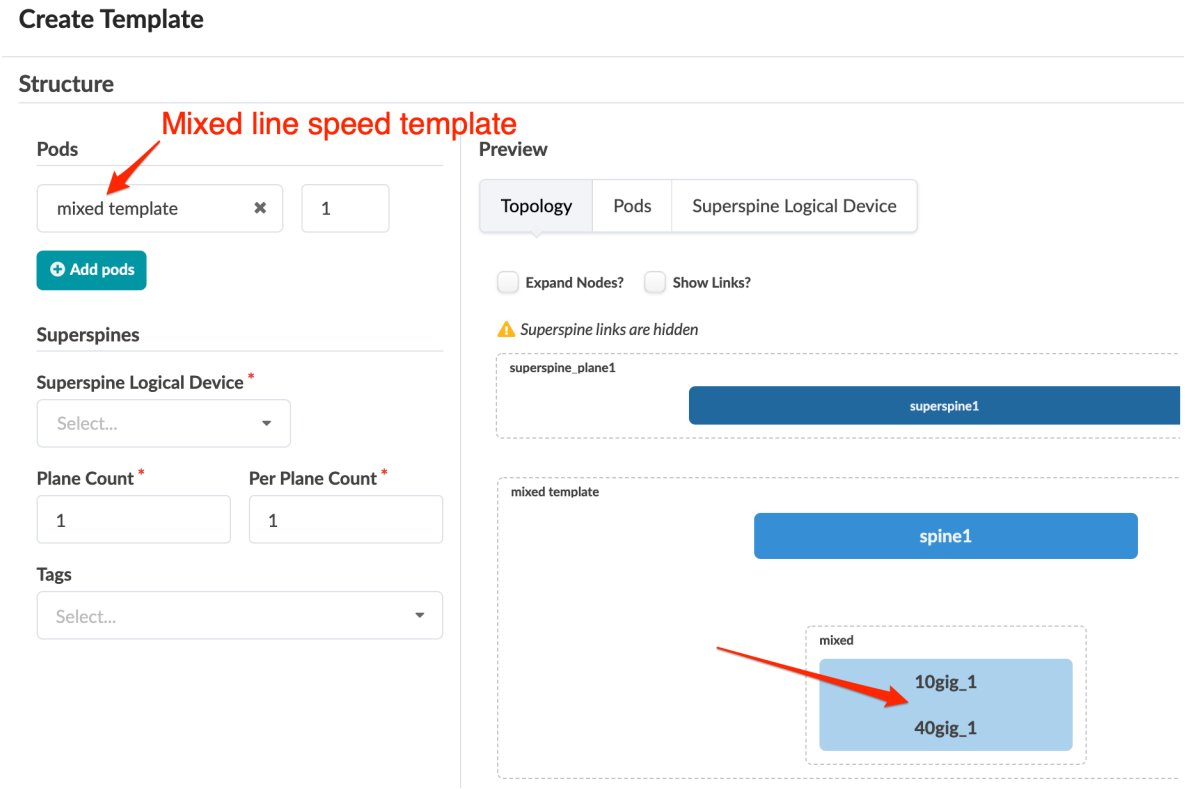
TOTAL	PORT GROUPS	Connected to
6 ports	6 x 40 Gbps — Spine • Generic	

1 3 5
2 4 6

2. Create a **Rack Based** template based on the mixed speed rack type.



3. You can create a **Pod Based** template based on the above rack based template.



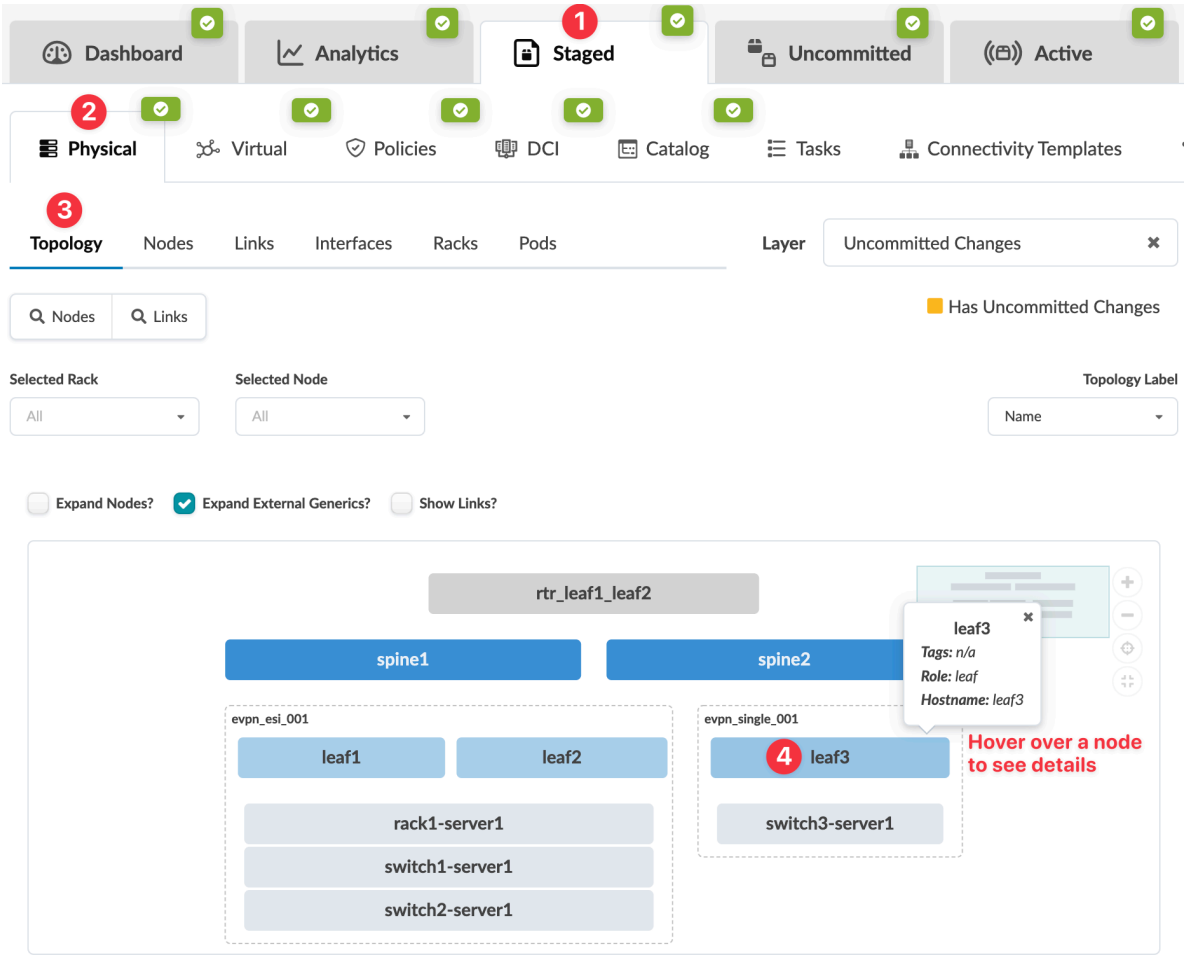
4. As a Day 0 operation you can "create a blueprint" on page 8 with one of the above templates; or as a Day 2 operation you can select a mixed speed rack type when "adding a rack" on page 233 to an existing blueprint.

LAG

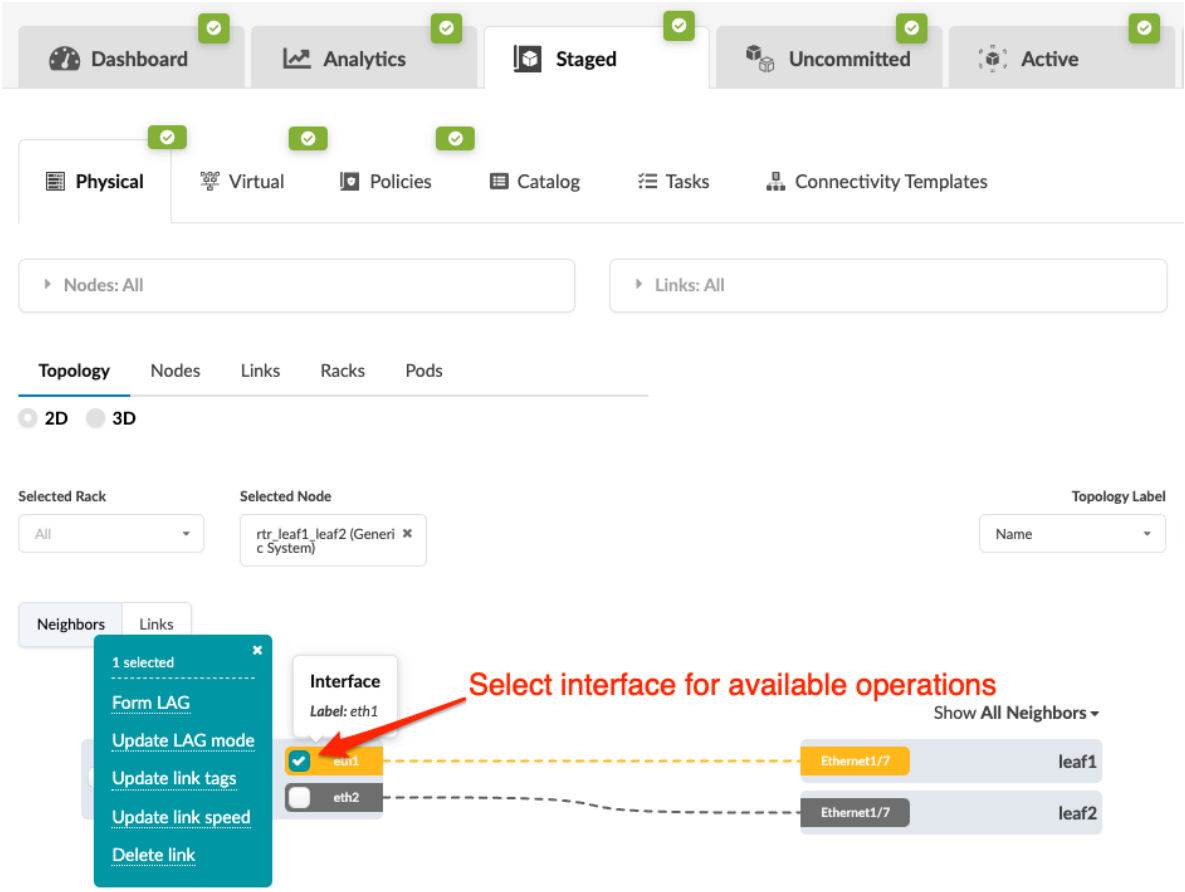
IN THIS SECTION

Form LAG

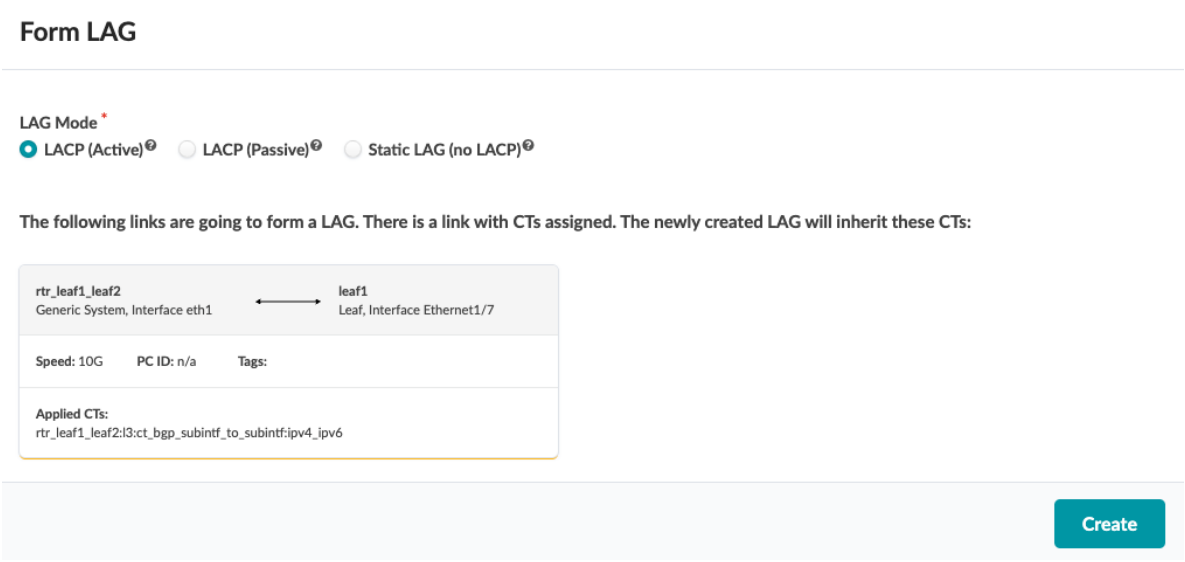
1. From the blueprint, navigate to **Staged > Physical > Topology** and select the node to add as a member of a LAG.



2. Select the interface check box to see the operations available for that interface (and that you have permissions for).



3. Click **Form LAG** and select the LAG mode:
   - **LACP (Active)** - actively advertises LACP BPDU even when neighbors do not.
   - **LACP (Passive)** - doesn't generate LACP BPDU until it sees one from a neighbor.
   - **Static LAG (no LACP)** - Static LAGs don't participate in LACP and will conditionally operate in forwarding mode.



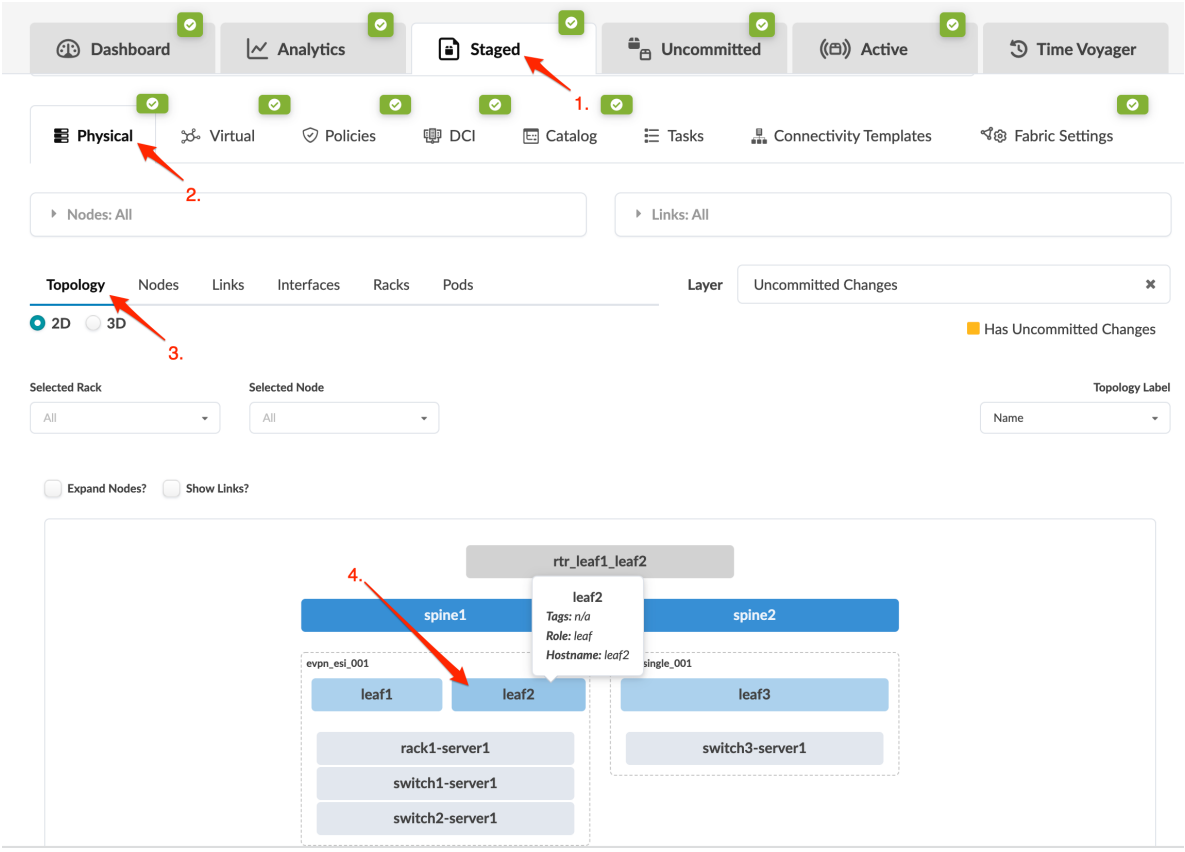
4. Click **Update** to stage your changes and return to the **Topology** view.
   When you form a LAG, it inherits any connectivity templates assigned on the individual links. The LAG is created, but LACP configuration won't be pushed to the device until connectivity templates are applied. When you form a LAG, it inherits any connectivity templates assigned on the individual links.

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

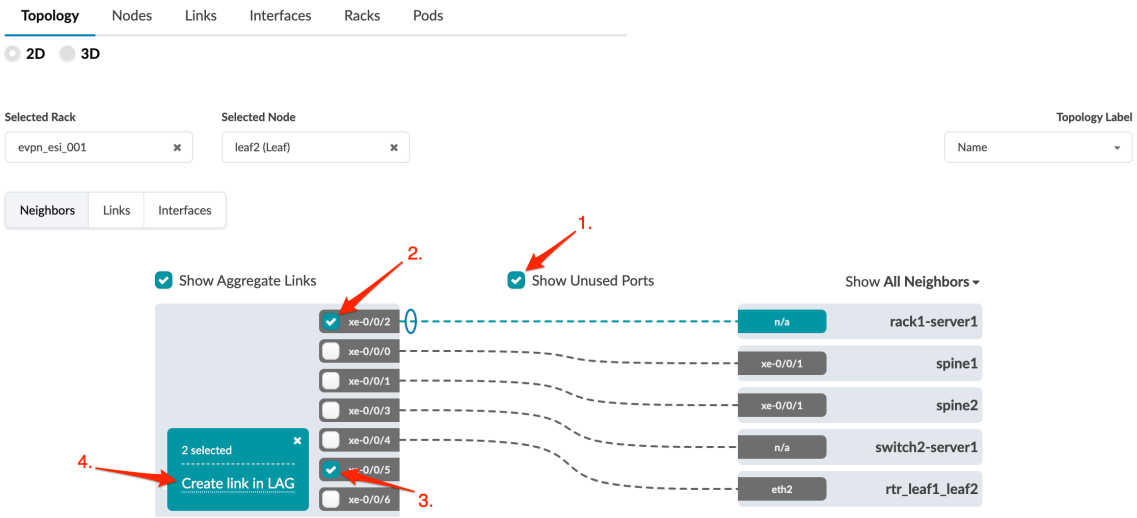
Create Link in LAG

You can add a link between a LAG and a generic system.

1. From the blueprint, navigate to **Staged > Physical > Topology** and select a leaf that is part of a LAG. (Alternatively, you can click the leaf name from the Nodes table at Staged > Physical > Nodes.)

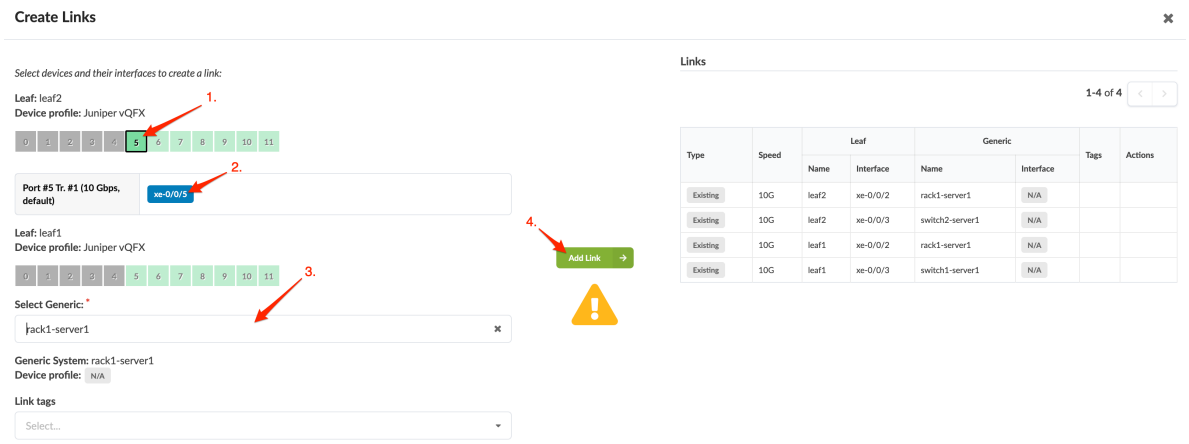


2. Check the **Show Unused Ports** check box, select a LAG interface and one of the unused port interfaces, then click **Create link in LAG**.



The **Create Link** dialog opens.

3. Select available ports and transformations, as applicable. The gray **Add Link** button turns green. From the drop-down list, select the generic system to link to, then click **Add Link**.



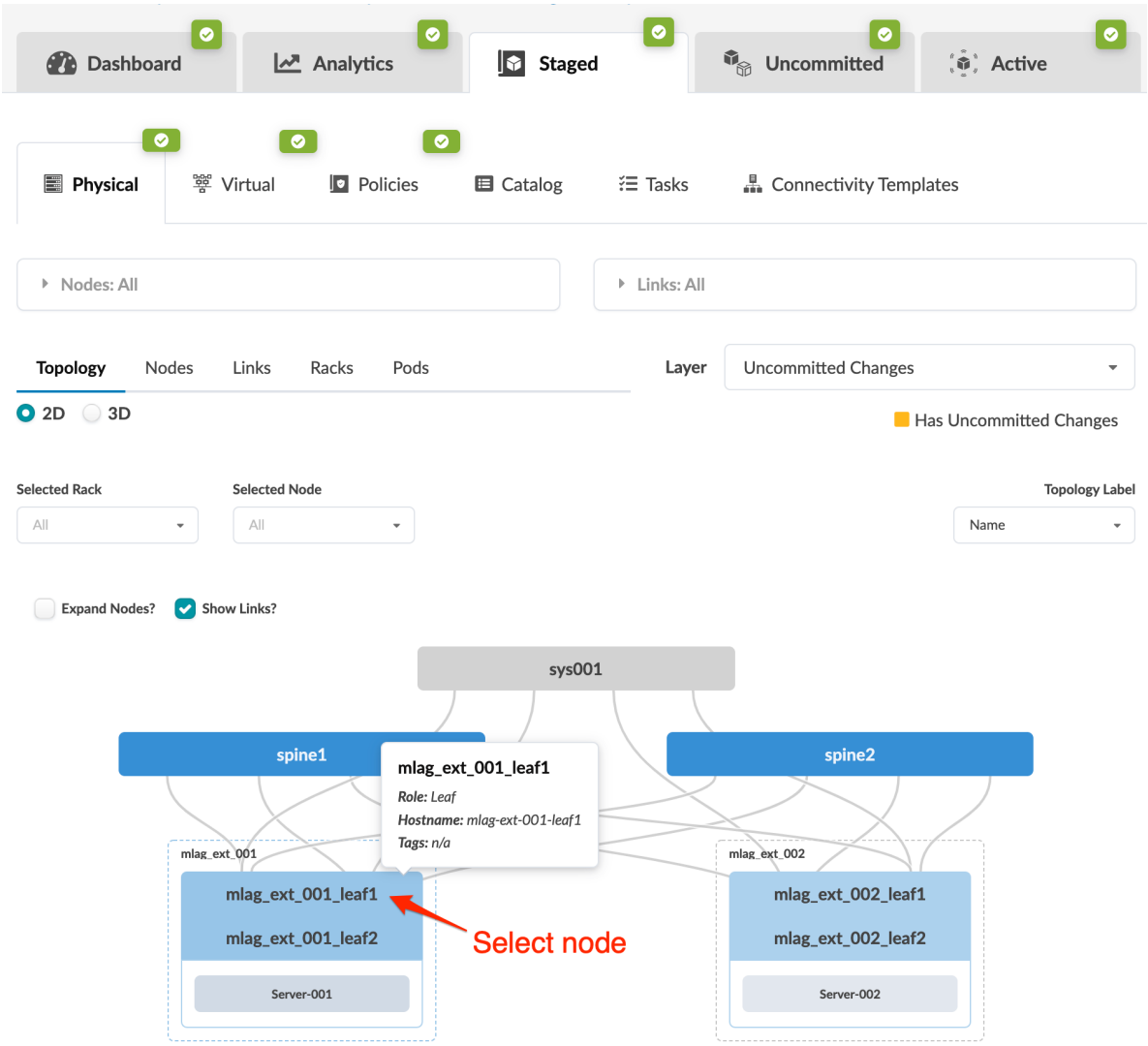
4. Dual-attached links to leaf groups (evpn-esi) must be symmetric. Add the second link, as applicable.
5. Click **Create** to create the link and return to the **Topology** view.

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

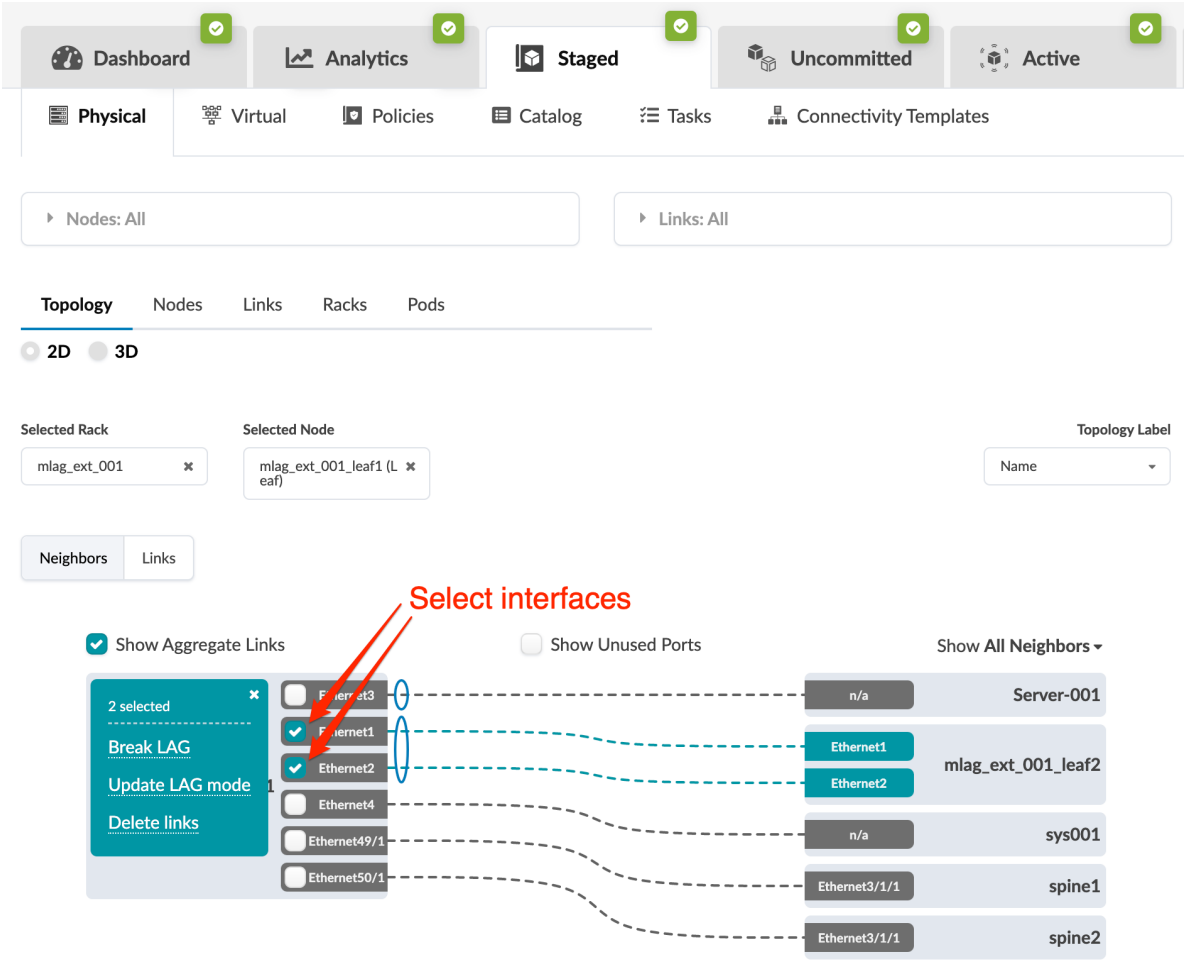
Break LAG

It's common to break a LAG towards a server into individual links, then reform the LAG from individual links, all while keeping the same VLAN allocation (when re-bootstrapping the server for example). You can break a LAG while preserving any assigned connectivity templates.

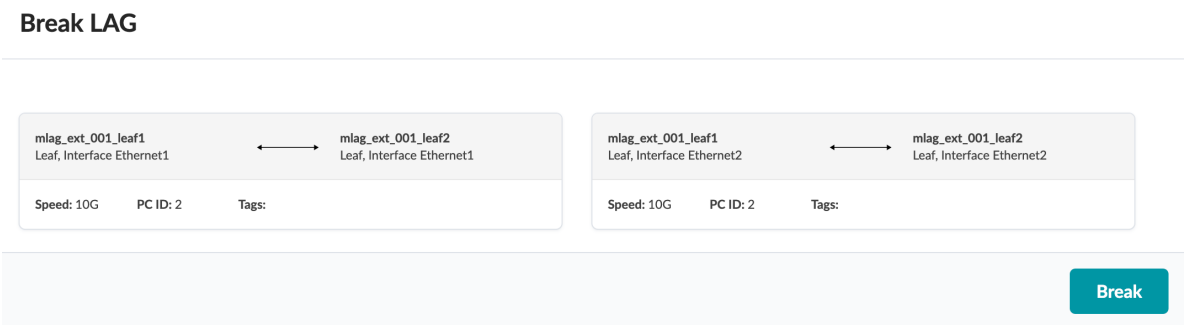
1. From the blueprint, navigate to **Staged > Physical > Topology** and select the node with the LAG to break.



2. Select the interface check boxes for the LAG (or click the port-channel representation) to see the operations available for those interfaces (and that you have permissions for).



3. Click **Break LAG** to go to its dialog with details on the LAG to break.

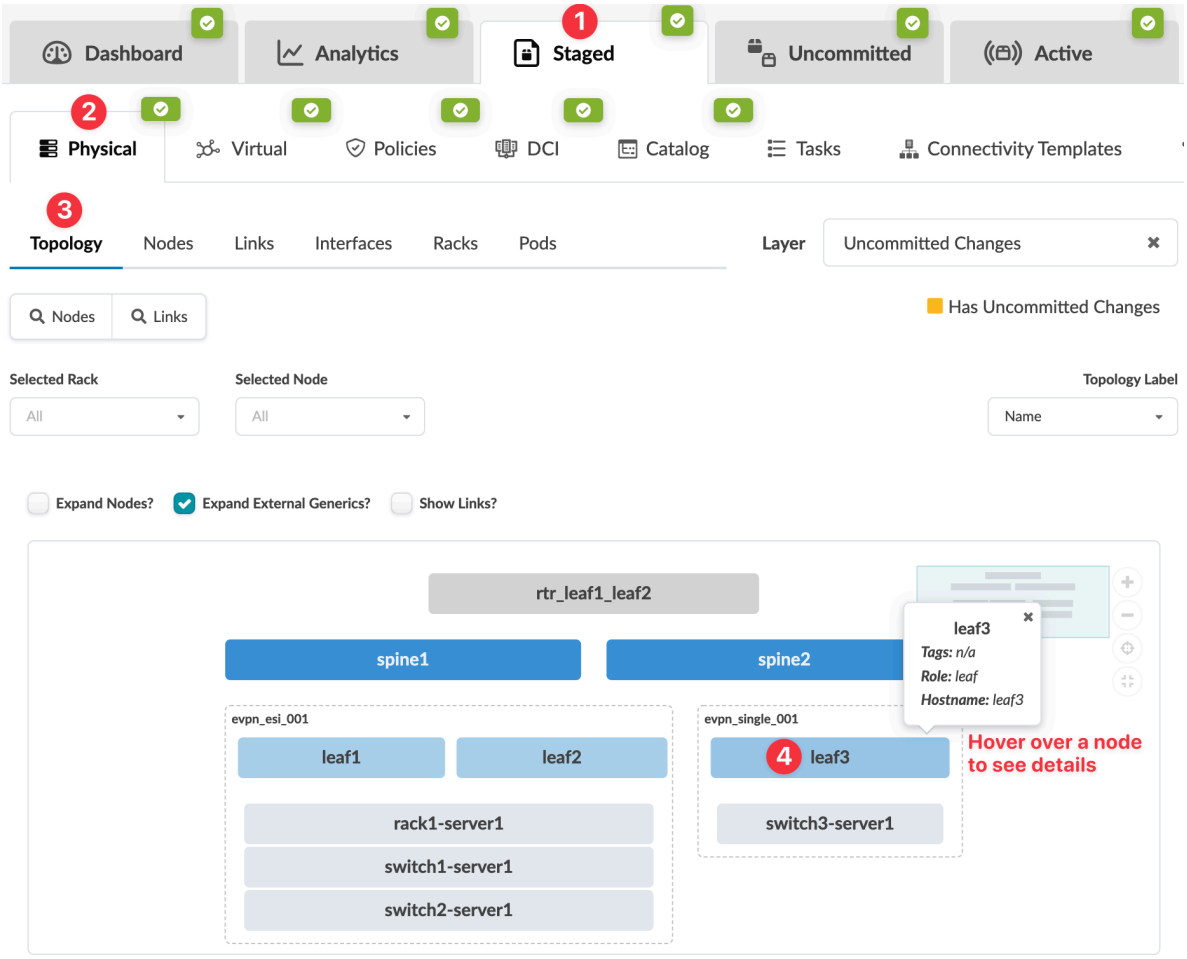


4. Click **Break** to stage your changes and return to the **Topology** view.

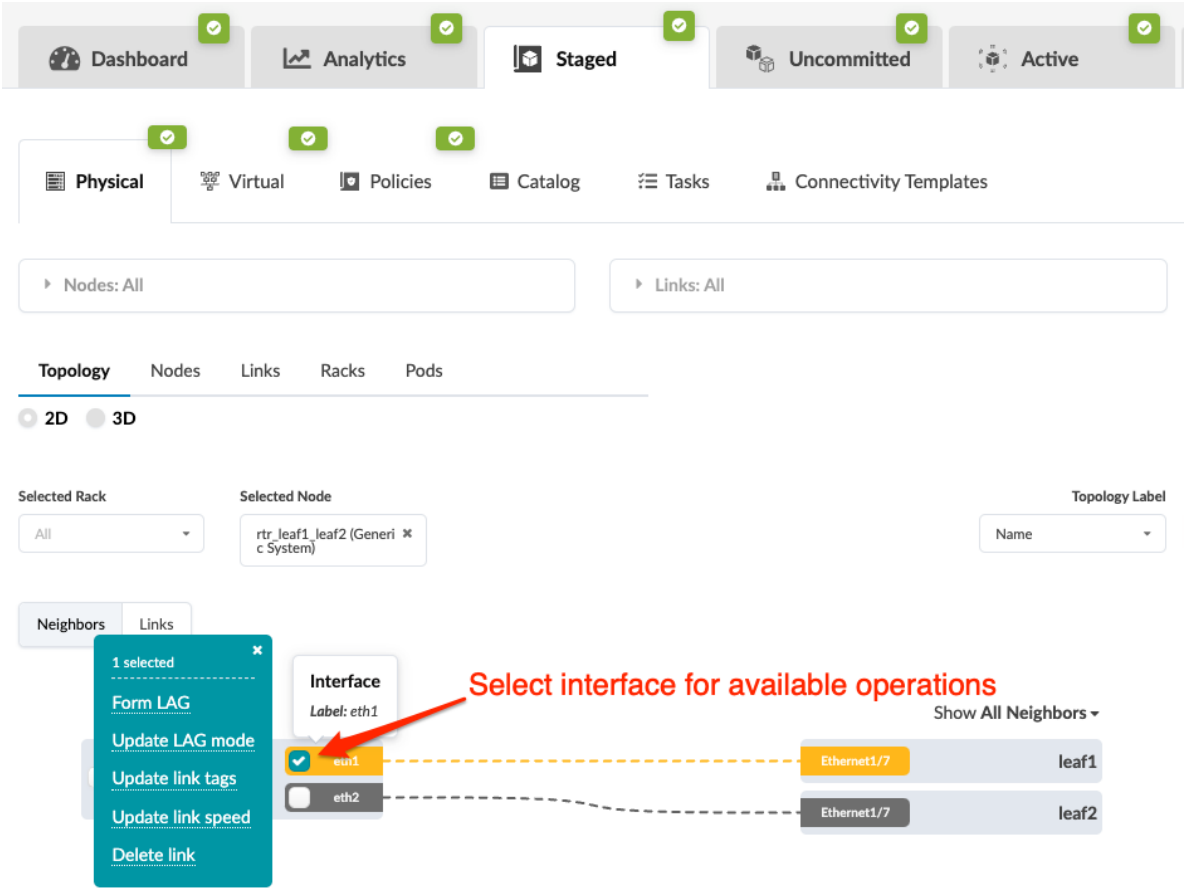
When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Update LAG Mode

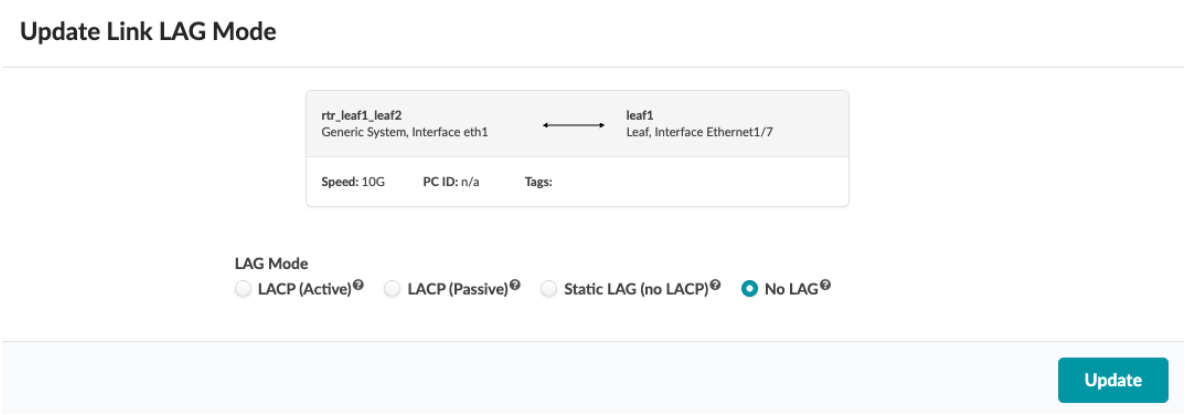
1. From the blueprint, navigate to **Staged > Physical > Topology** and select the MLAG member that needs an updated link LAG mode.



2. Select the interface check box to see the operations available for that interface (and that you have permissions for).



3. Click **Update LAG mode** and select the new LAG mode:
   - **LACP (Active)** - actively advertises LACP BPDU even when neighbors do not.
   - **LACP (Passive)** - doesn't generate LACP BPDU until it sees one from a neighbor.
   - **Static LAG (no LACP)** - Static LAGs don't participate in LACP and will conditionally operate in forwarding mode.
   - **No LAG** - The link is not part of a LAG.



4. Click **Update** to stage your changes and return to the **Topology** view.

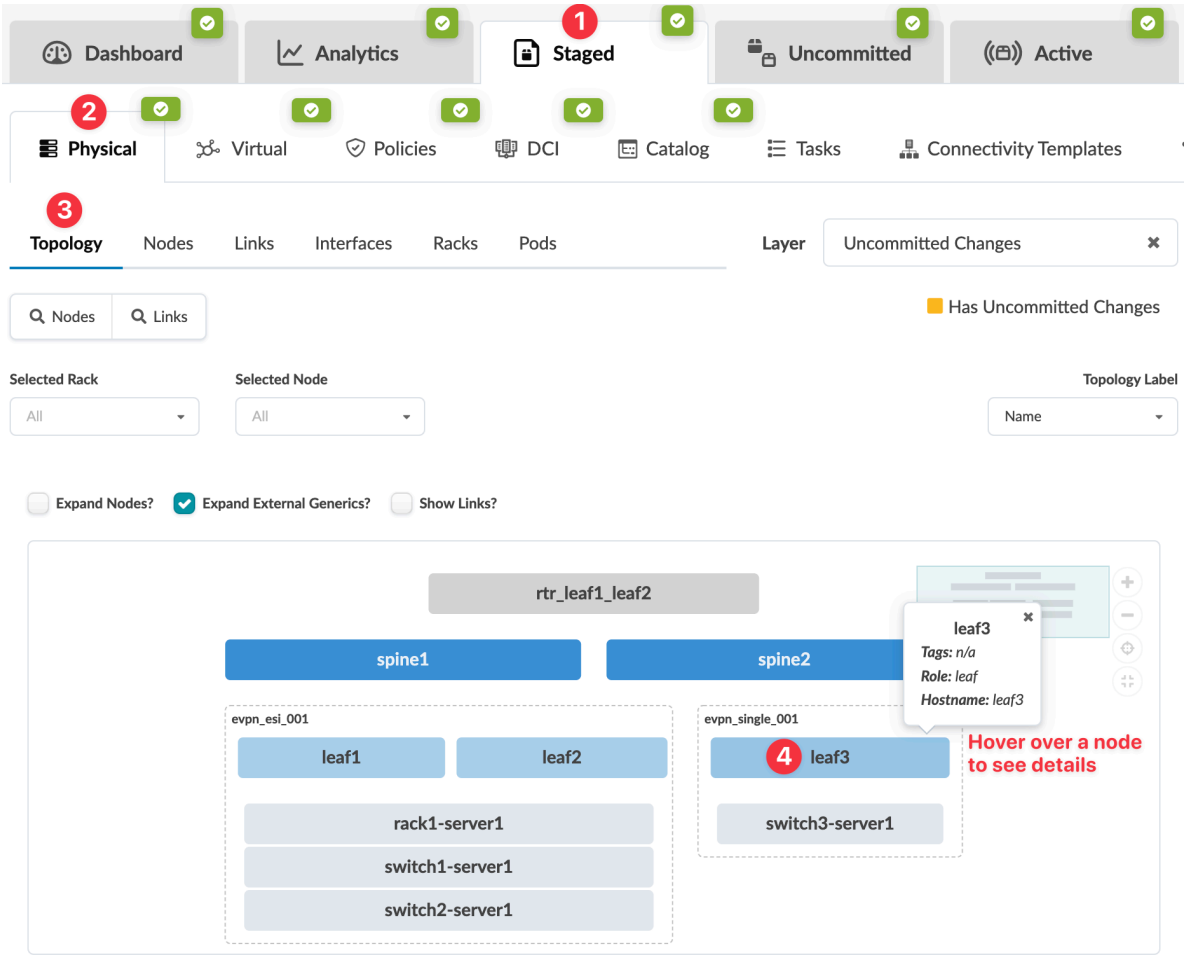
When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Update Tags on Link (Datacenter)

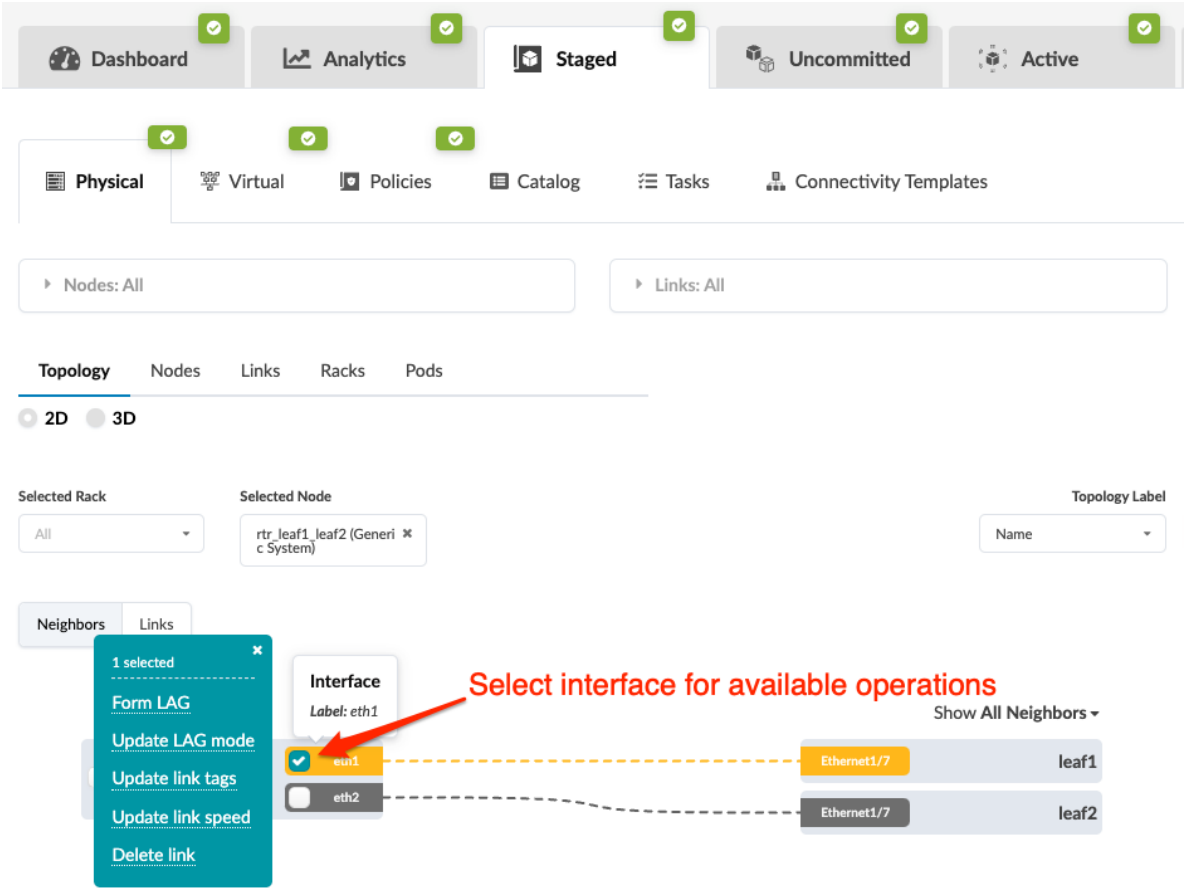
IN THIS SECTION

Update Link Tags (One Link - Topology View)

1. From the blueprint, navigate to **Staged > Physical > Topology** and select the node connected to the link that needs tags updated.



2. Select the interface check box to see the operations available for that interface (and that you have permissions for).



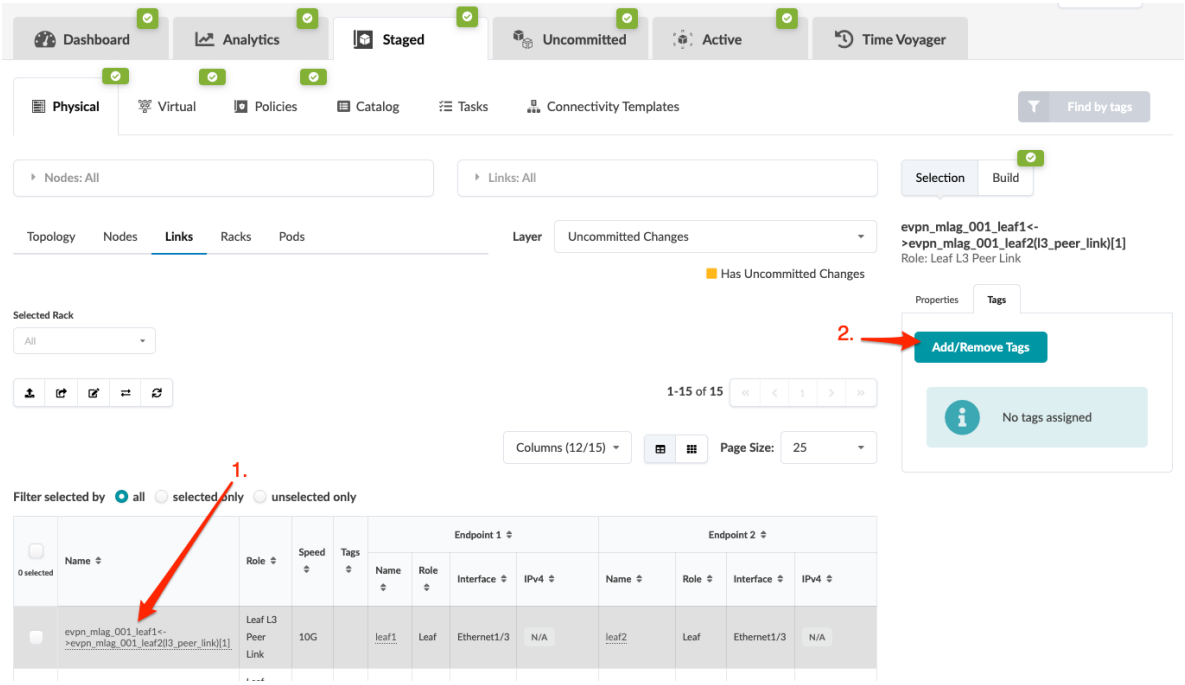
3. Click **Update link tags** and update link tags as needed.

4. Click **Update** to update link tags and return to the **Selection** view.

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Update Link Tags (One Link - Links View)

1. From the blueprint, navigate to **Staged > Physical > Links** and select the link name (not the check box) for the link that needs updated tags.

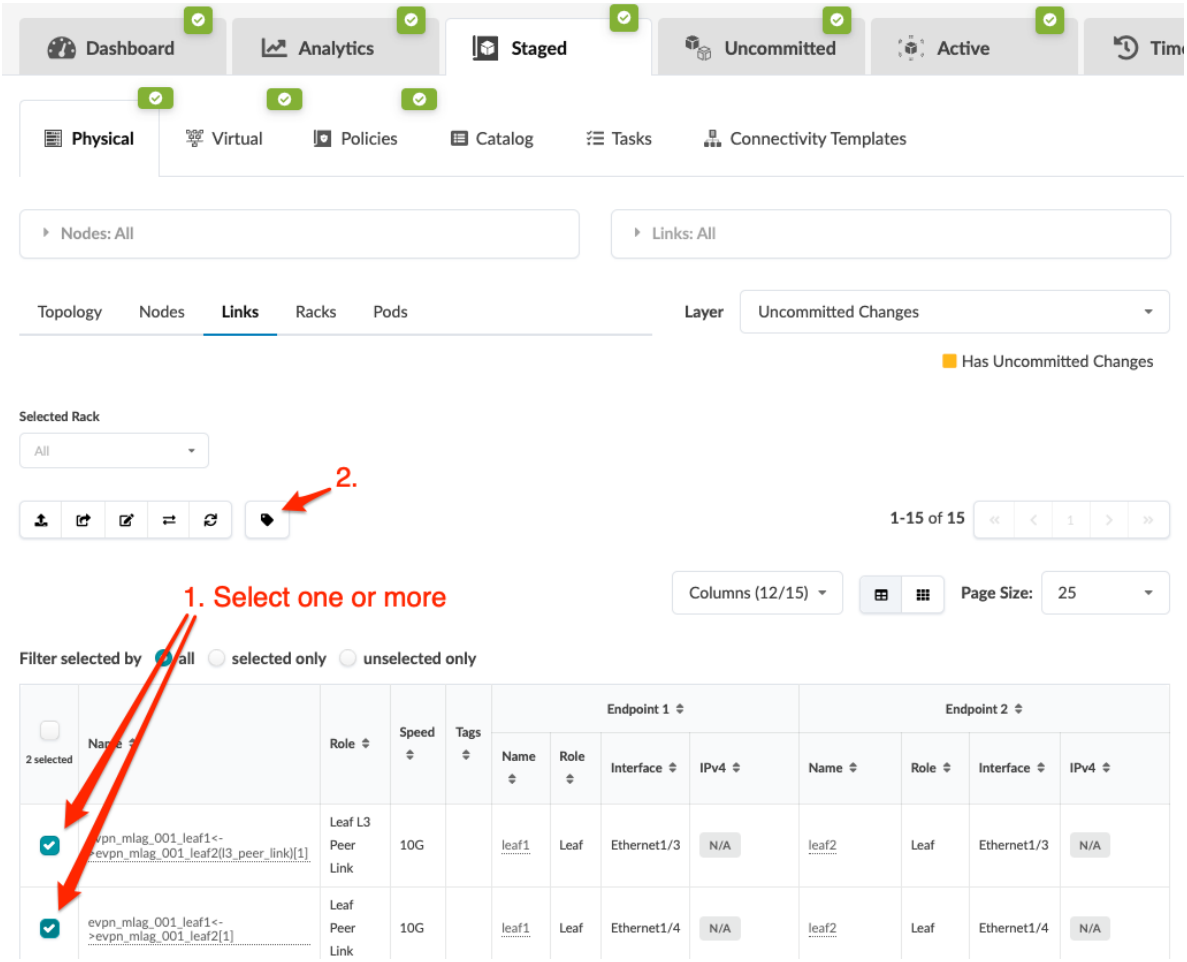


2. Click **Add/Remove Tags** to see tags that are in the blueprint catalog.
3. Select existing tag(s) or create new one(s) that will be tagged to the link and added to the blueprint catalog.
4. Click **Update Tags** to update the tags and return to the **Links** view.

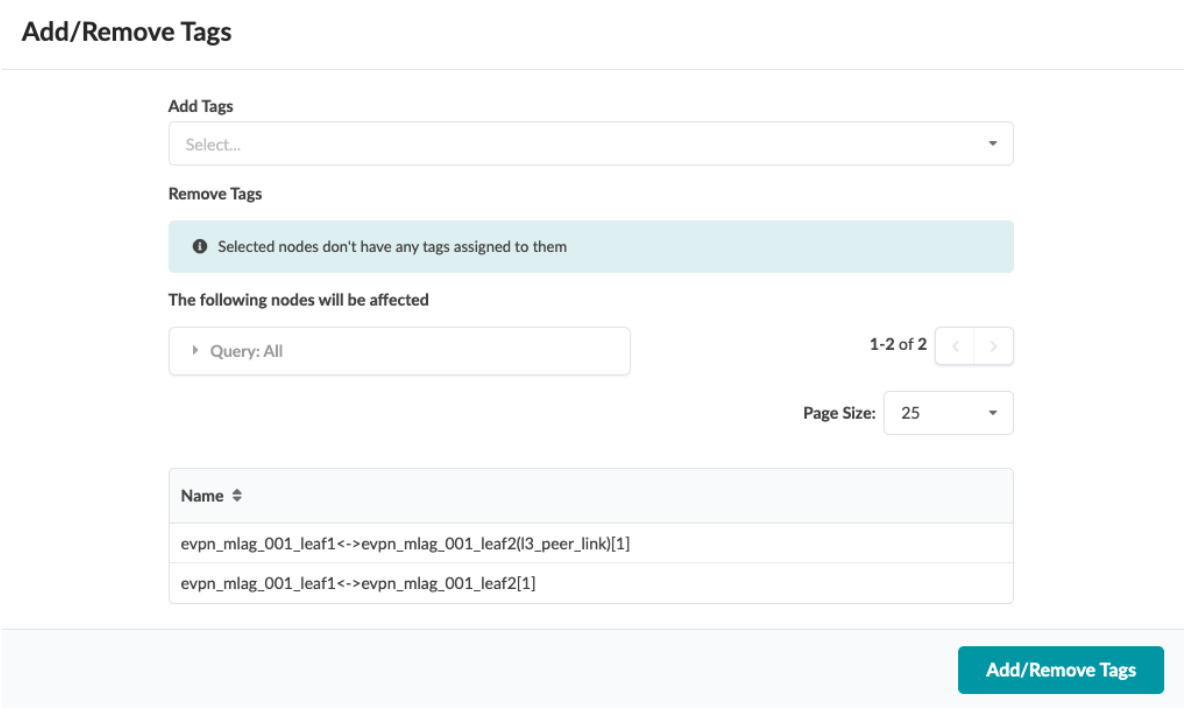
When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Update Link Tags (Multiple Link - Links View)

1. From the blueprint, navigate to **Staged > Physical > Links** and select one or more check boxes for the link(s) that need updated tags. The **Add/Remove Tags** button appears above the table.



2. Click the **Add/Remove Tags** button and update tags as needed. When you create new tags here they are added to the blueprint catalog.



3. Click **Add/Remove Tags** to stage the change and return to the **Links** view.

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

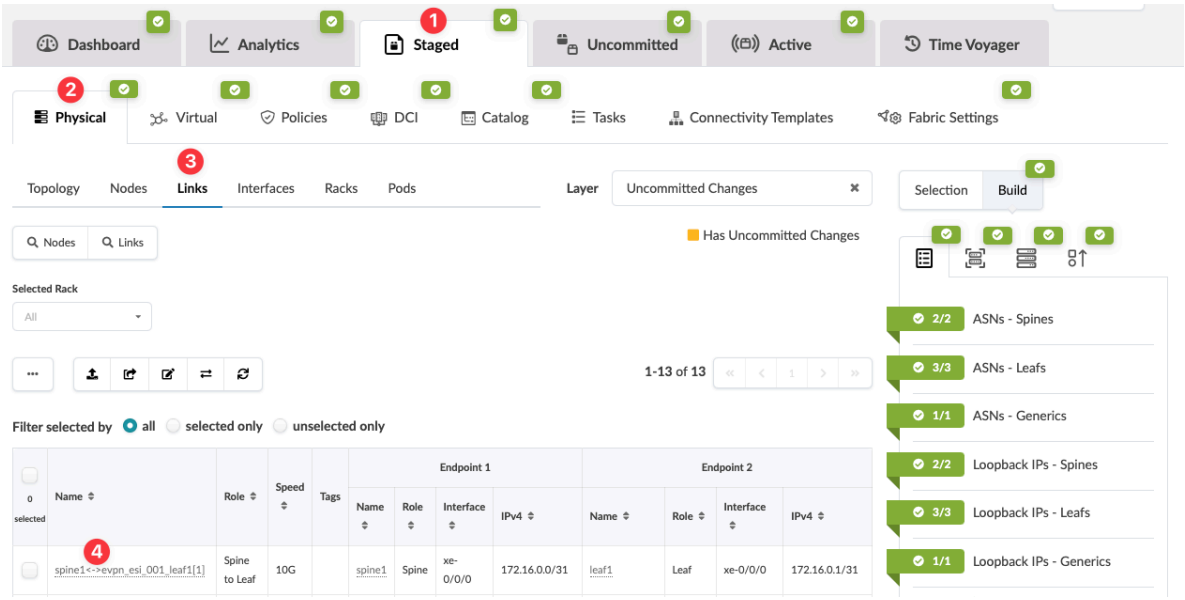
Change Assigned Link IP Addresses (Datacenter)

SUMMARY

Assign or change an individual Link IP Address

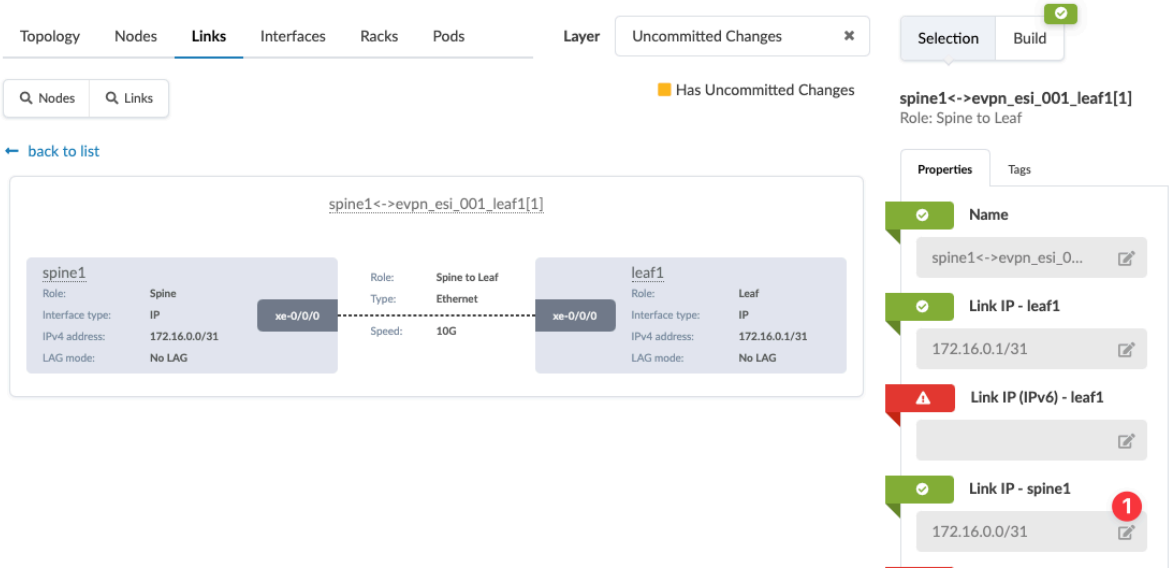
Normally, you would let Apstra "pull IP addresses from resource pools" on page 58 for you, but there may be times when you need to assign a specific IP address or change one that was already assigned. In these cases, you can select the device and assign or change the IP address from the **Properties** tab. See below for details.

1. From the blueprint, navigate to **Staged > Physical > Links** and select the link to update.



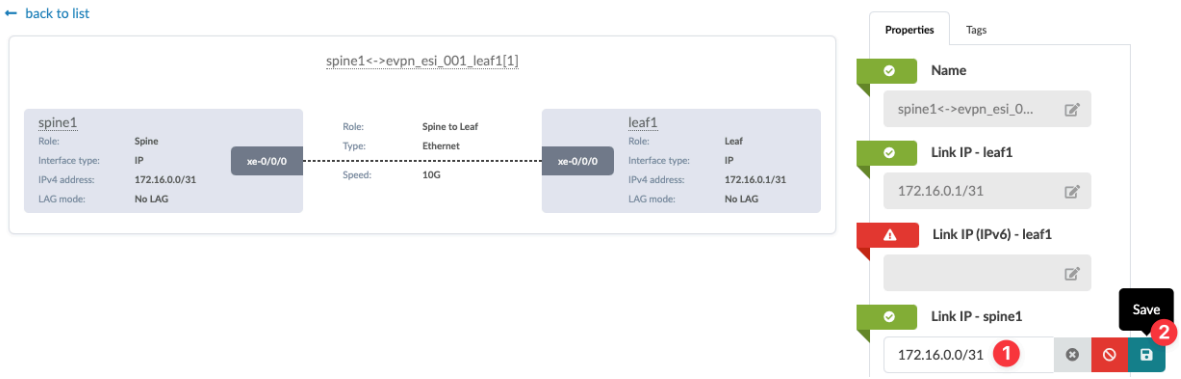
The **Selection** panel on the right becomes active and populates with information about the selection.

2. In the **Properties** tab in the **Selection** panel click the **Edit** button for the **Link IP** field to change.



The **Link IP** field becomes available to change.

3. Enter the new link IP address value, then click the **Save** button to stage your changes.



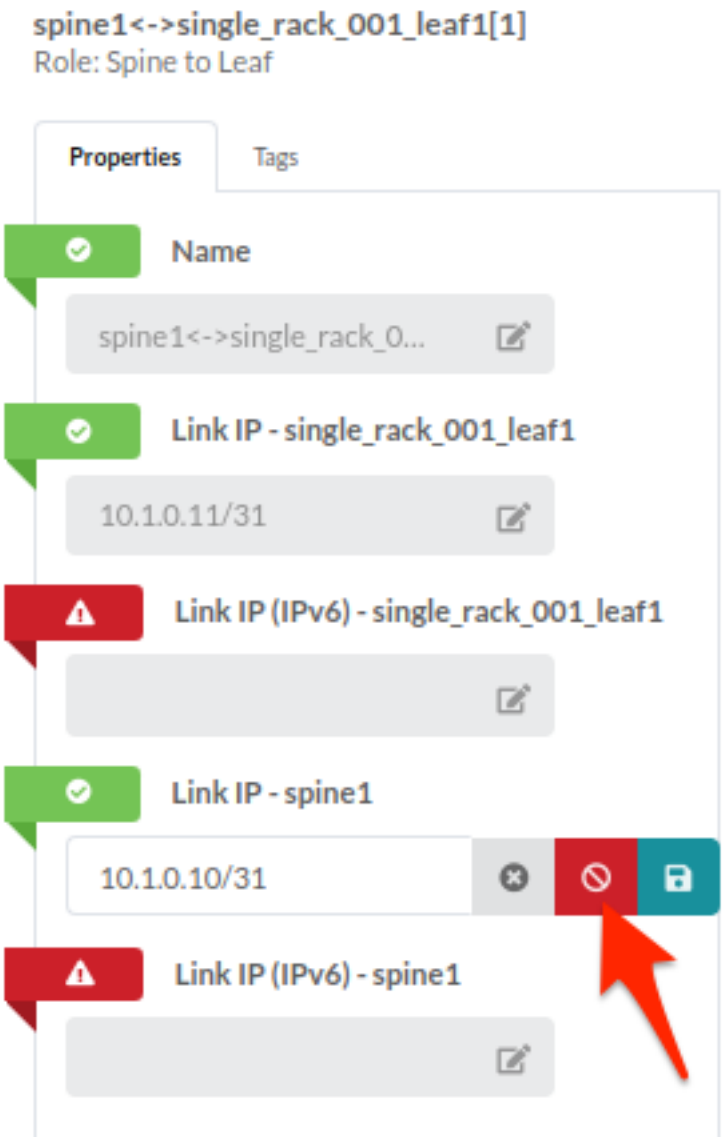
When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

Update Link Properties

If you have changed server names and/or hostnames for switches, any associated link names do not automatically update to match. This may cause confusion when reviewing an updated cabling map in the **Uncommitted** tab. You can change link names to match your other name changes. You can also change link IP for endpoints from here.

1. From the blueprint, navigate to **Staged > Physical > Links** and click the name of the link to change.
2. Go to the **Properties** tab in the right panel.
3. Depending on the link chosen, you can change link properties such as name and Link IP for endpoints. The attributes that can be edited have an **Edit** button associated with them. Change properties as applicable.

   When you change link IP for an endpoint, you must remove link IP from the other endpoint first. Otherwise you will get validation error “User-specified link IPv4 addresses not in the same subnet”.



When you assign new link IP to an endpoint, the link IP for the other endpoint is automatically assigned from the same subnet.

4. Click the **Save** button to stage the changes.

Fetch LLDP Data (Datacenter)

If you've already cabled up your devices, you can have Apstra discover your existing cabling instead of using the cabling map prescribed by Apstra. All system nodes in the blueprint must have system IDs assigned to them.

> ⚠ **CAUTION**: This is a disruptive operation. All links can potentially be renumbered.

1. From the blueprint, navigate to **Staged > Physical > Links** and click the **Fetch discovered LLDP data** button (fifth of five buttons above links list).
2. If staged data is *identical* to LLDP discovery results, you will see a message with that statement. Your actual cabling matches the Apstra cabling map. No further action is needed.
3. If staged data is *different* from LLDP discovery results, the message includes the number of links that are different.
4. Scroll to see details of the diffs (in red), or check the **Show only links with LLDP diff?** checkbox to see only the differences.
5. To accept the changes and update the map to match LLDP data, click **Update Stated Cabling Map from LLDP**. You might also need to `reset resource group overrides`.

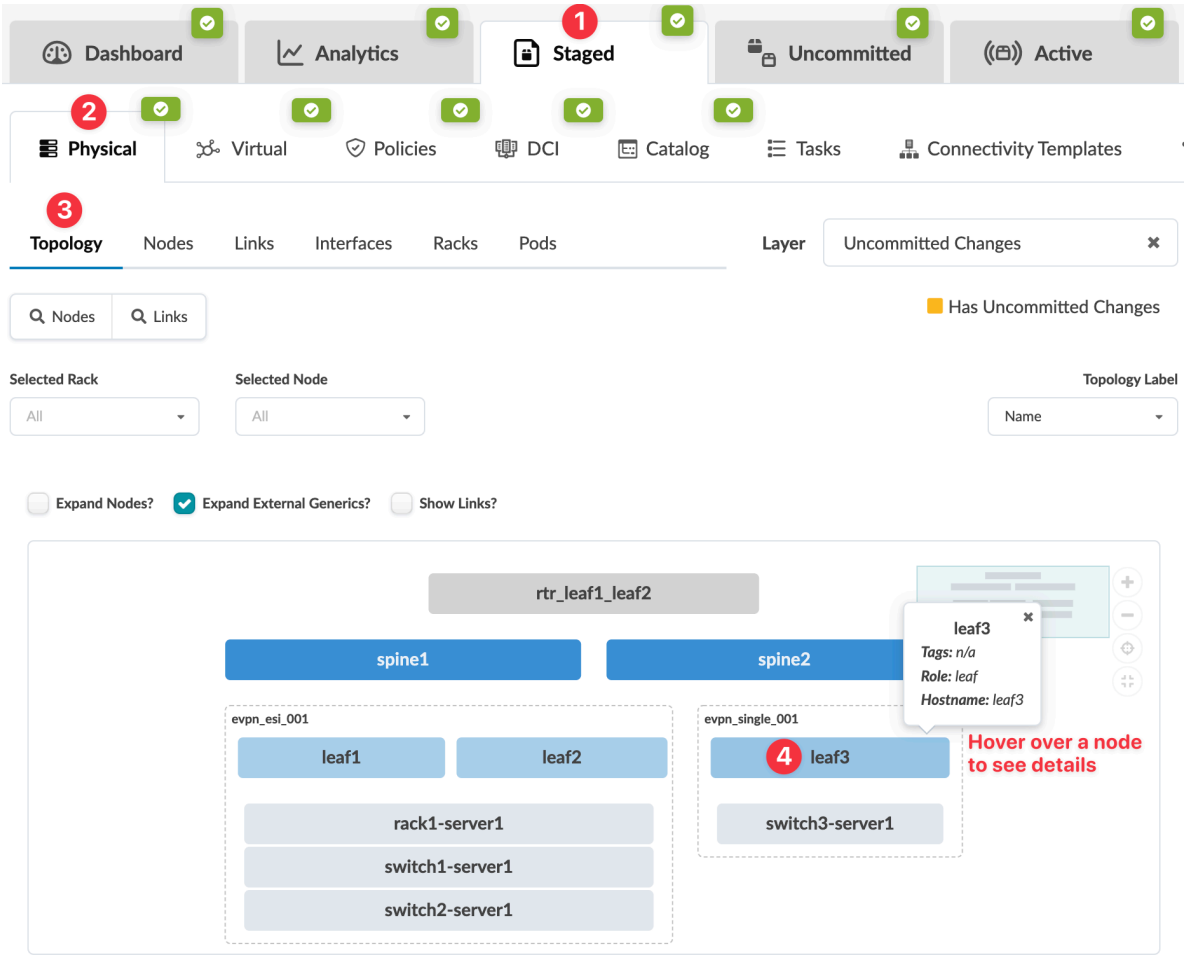
Delete Link (Datacenter)

IN THIS SECTION

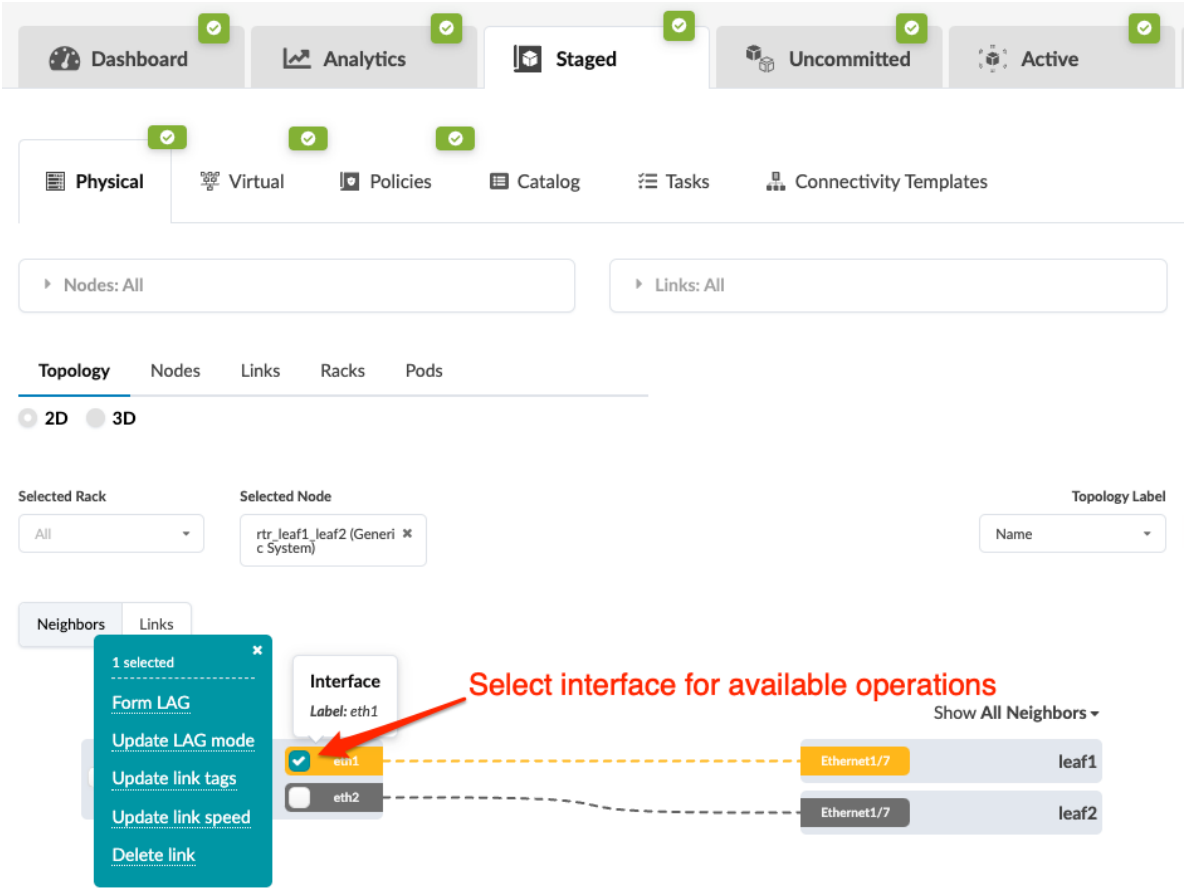
You can delete links from the **Neighbors** view or the **Links** view of a selection in a blueprint.

Delete Link (Neighbors View)

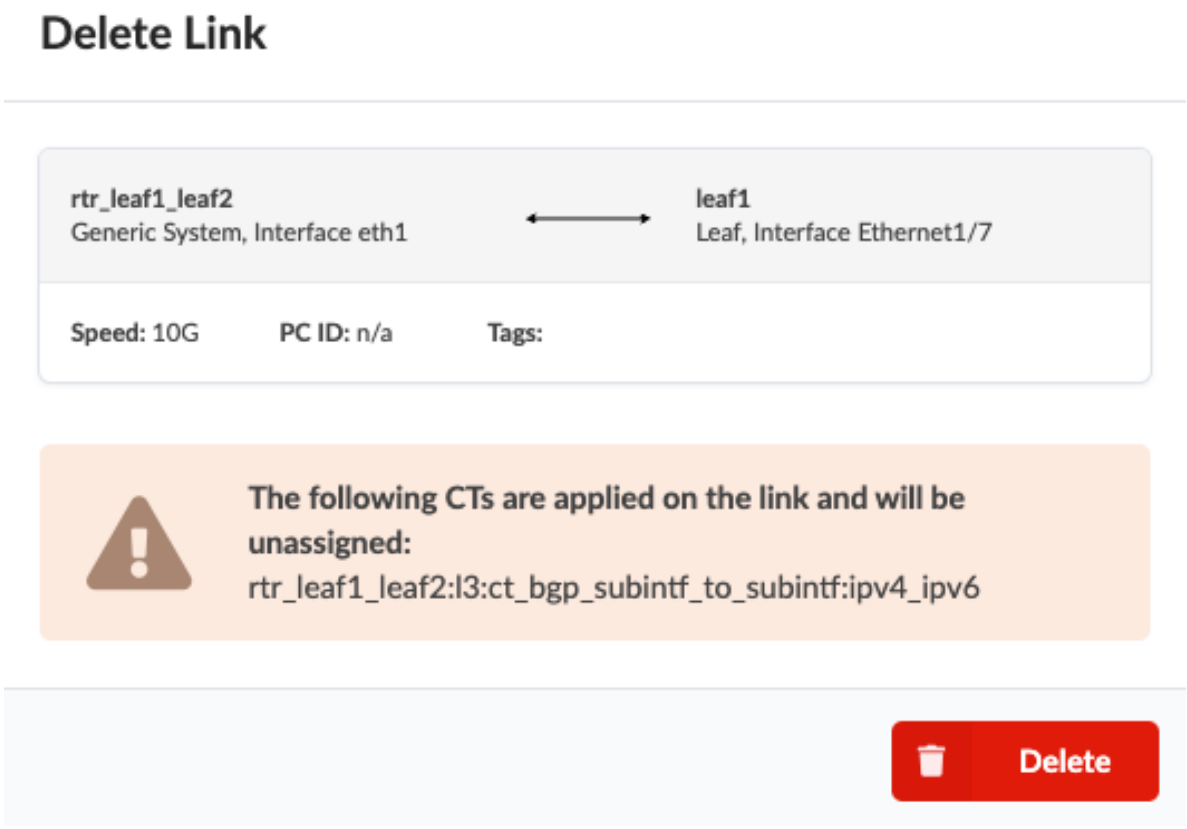
1. From the blueprint, navigate to **Staged > Physical > Topology** and select the node where you want to delete a link.



2. From the **Neighbors** view, select the node check box to see the operations available for that node (and that you have permissions for).



3. Click **Delete Link** to go to its dialog and review deletion details. Any connectivity templates that are applied on the link will be unassigned.



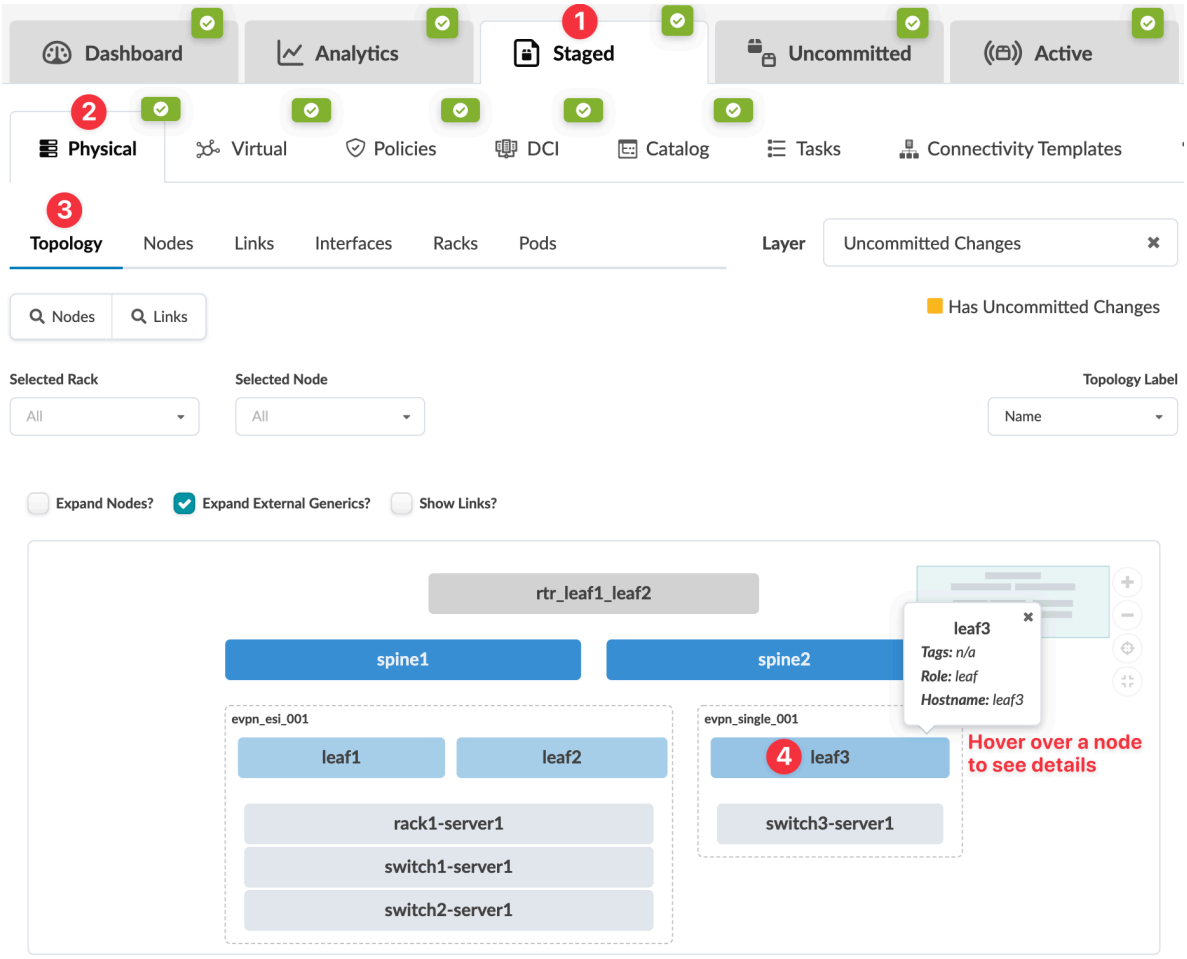
4. Click **Delete** to stage the deletion and return to the **Neighbors** view of the selected node.

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

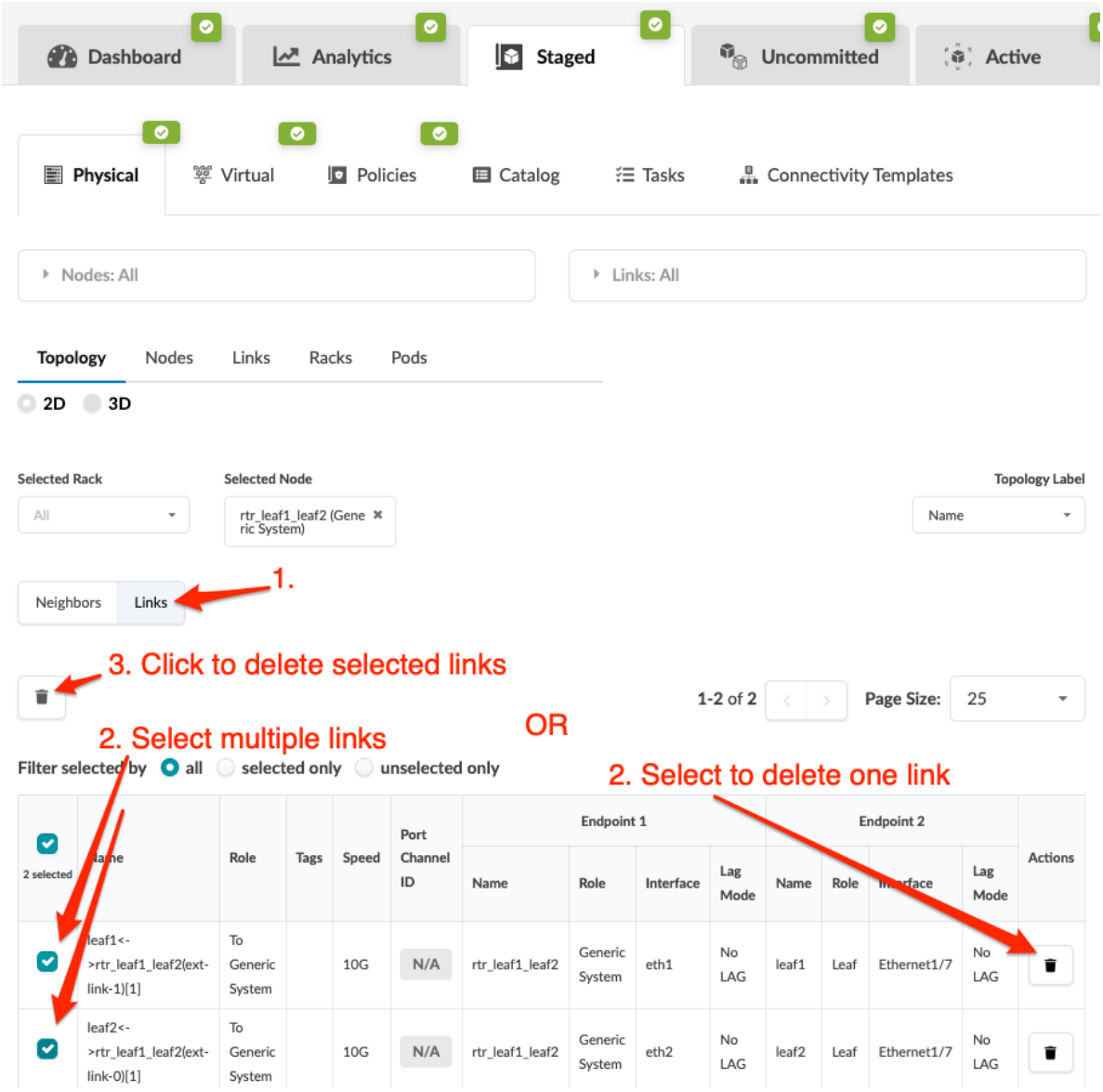
Delete Link (Links View)

From the **Links** view of your selected node you can delete one or more links at the same time.

1. From the blueprint, navigate to **Staged > Physical > Topology** and select the node where you want to delete a link.



2. Click **Links** to go to the Links table.



3. Select link(s) to delete in one of the following ways:
   - Select one or more links in the left column and click the **Delete** button above the table.
   - Click the **Delete** button in the right column for the one link to delete.
4. Review deletion details in the dialog that opens. Any connectivity templates that are applied on the link(s) will be unassigned.
5. Click **Delete** to stage the deletion and return to the **Links** view of the selected node.

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Interfaces

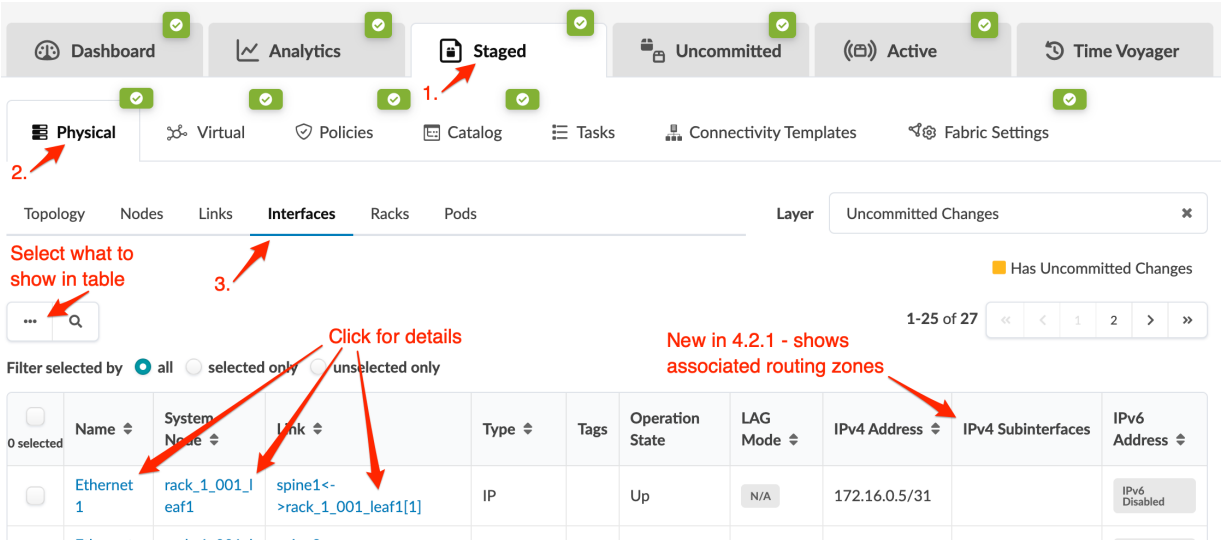
IN THIS SECTION

Interfaces Introduction

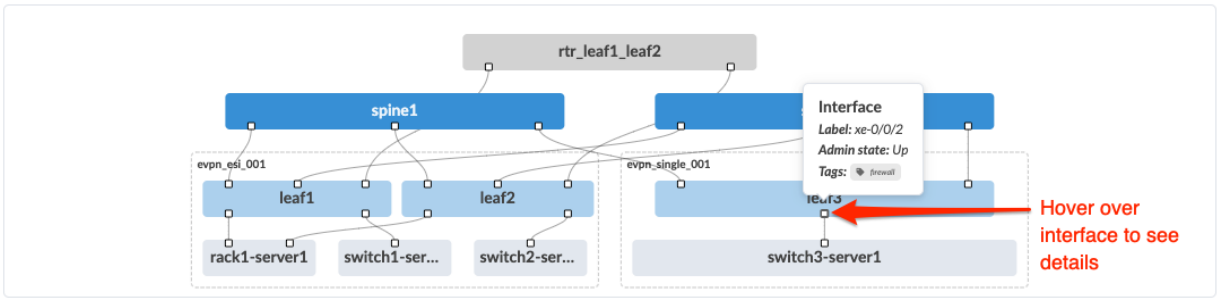
SUMMARY

Interfaces details in blueprints are in the interface table at Staged > Physical > Interfaces and in the interfaces section of specific routing zones at Staged > Virtual > Routing Zones.

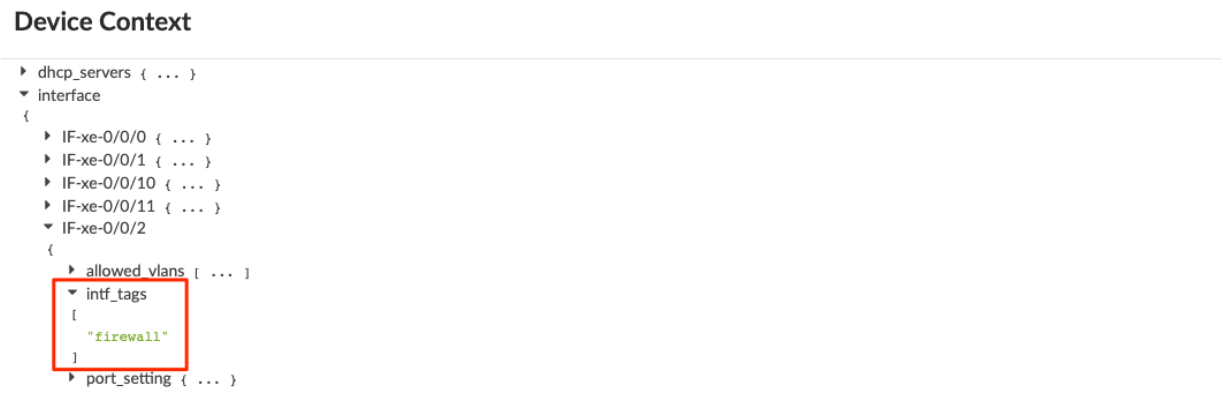
Interface details are listed in an interfaces table in the blueprint. To go to the table, navigate to **Staged > Physical > Interfaces**. From the table, you can access additional details about the associated system node, links and the interface itself. You can tag physical interfaces and aggregated logical interfaces (port channels). Tags become part of the graph, which means you can use them for configuration. You can administratively disable and enable interfaces from the GUI.



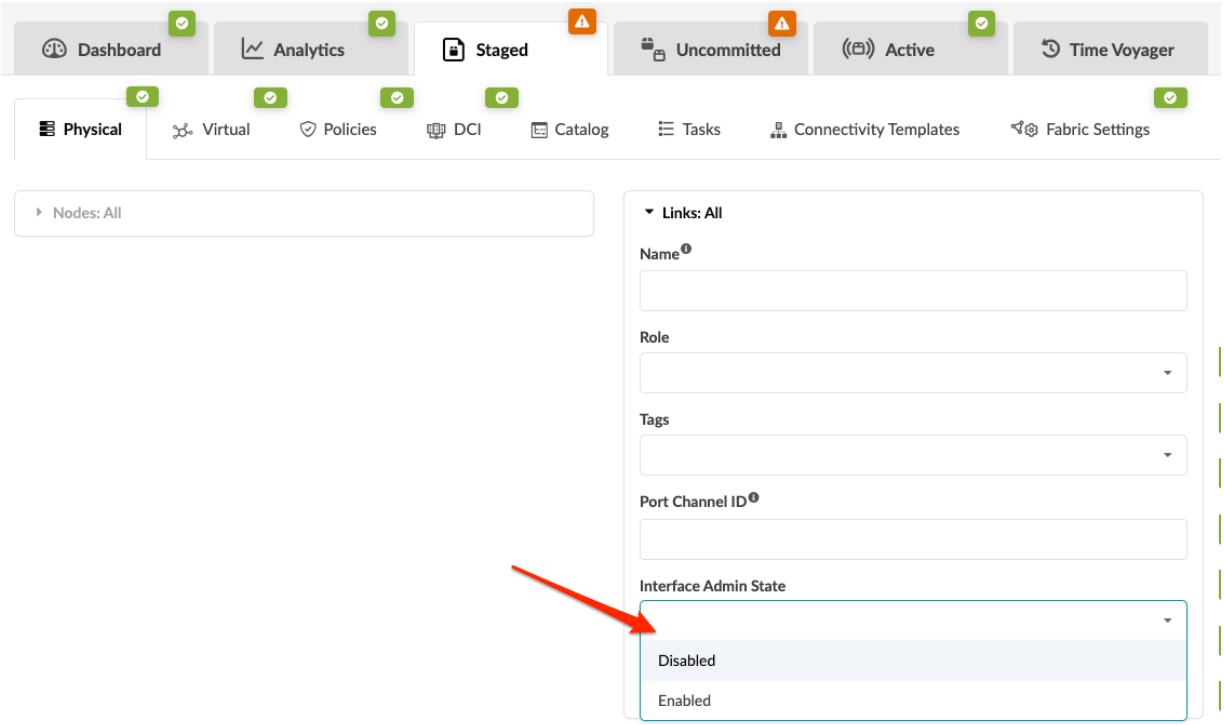
You can see interface tags from various locations in the GUI. The interfaces table includes a column for tags, as shown above. You can also see interface tags when you hover over an interface in the topology view (along with the admin state and other information).



The device context also includes interface tag information. (Select a node, then from the **Device** tab on the right, at the bottom, click **Device Context**.)



You can administratively disable and enable interfaces from the Apstra GUI. To show only the interfaces in the table that you've disabled or enabled, select the state from the **Interface Admin State** drop-down list in the **Links** filter.



RELATED DOCUMENTATION

Change Interface IP Address | 211

Add / Remove Tags on Interface (Datacenter) | 216

Add / Remove Tags on Port Channel (Datacenter) | 222

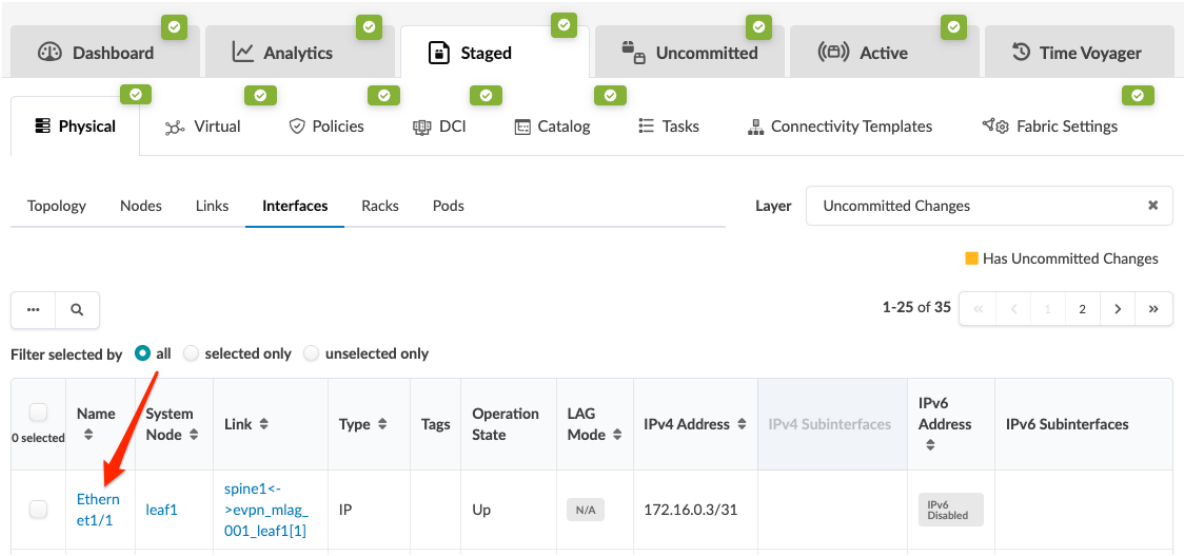
Administratively Disable Interface | 224

Administratively Enable Interface | 226

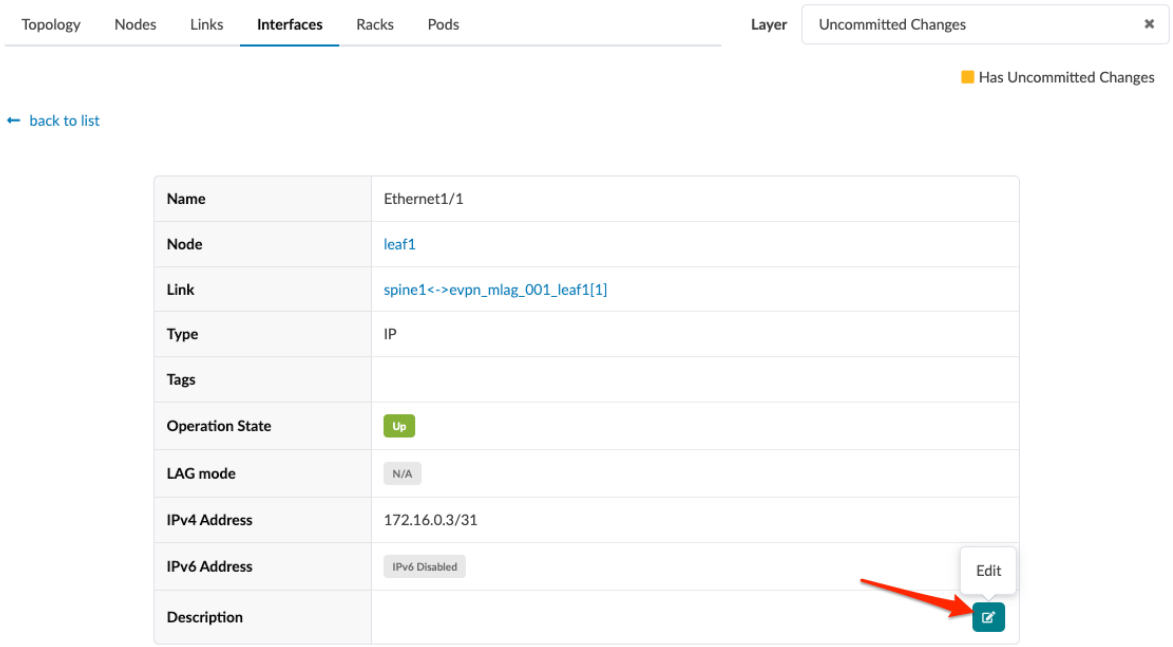
Change Interface Description

You can edit interface descriptions from the Apstra GUI (as of Apstra version 5.0.0). Previously, you could only change the description via an API call.

1. From the blueprint, navigate to **Staged > Physical > Interfaces**, and select the name of the interface that needs a new or changed description.



2. In the Interface details table, click the **Edit** button in the **Description** row.



3. In the dialog that opens, add or change the description, as needed.
4. Click **Submit** to save your changes and return to the Interfaces table.

To deploy the change to the active blueprint, commit from the **Uncommitted** tab.

Change Interface IP Address

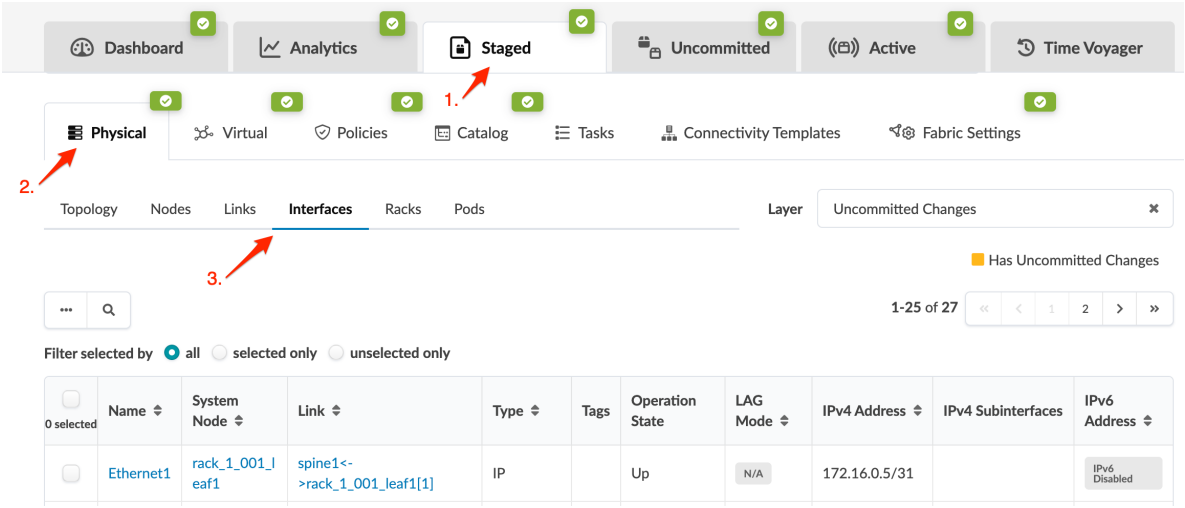
SUMMARY

You may need to change the interface IP address after it's been assigned. You can change it from the associated routing zone.

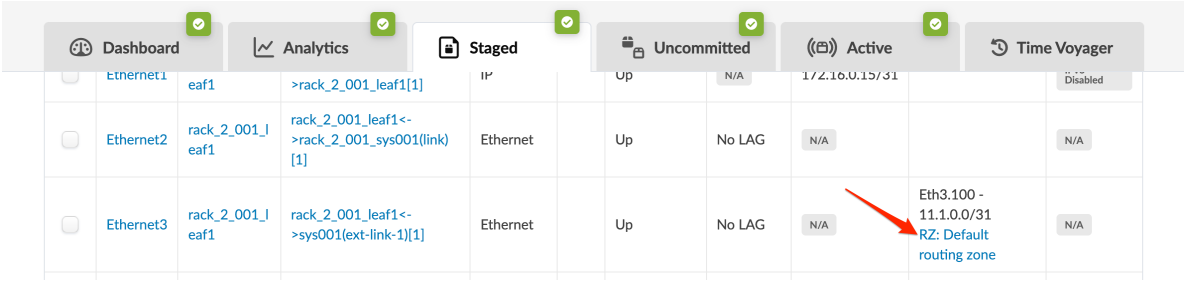
IN THIS SECTION

From Main Interfaces Table

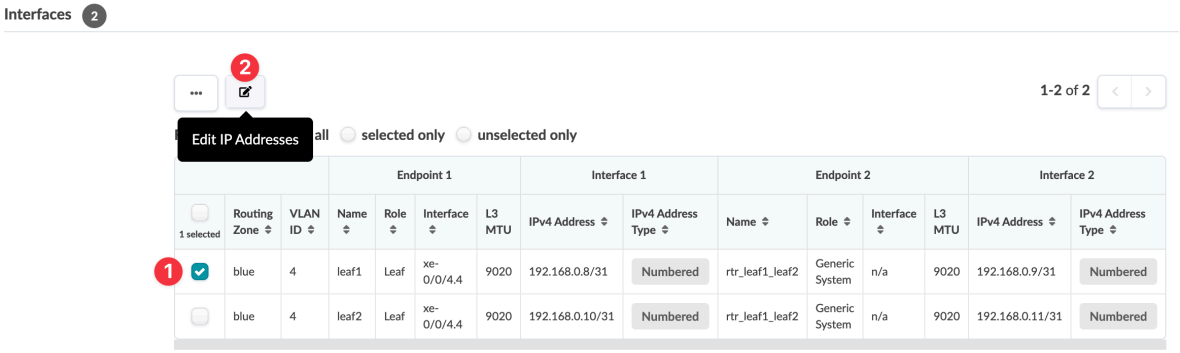
1. From the blueprint, navigate to **Staged > Physical > Interfaces**.



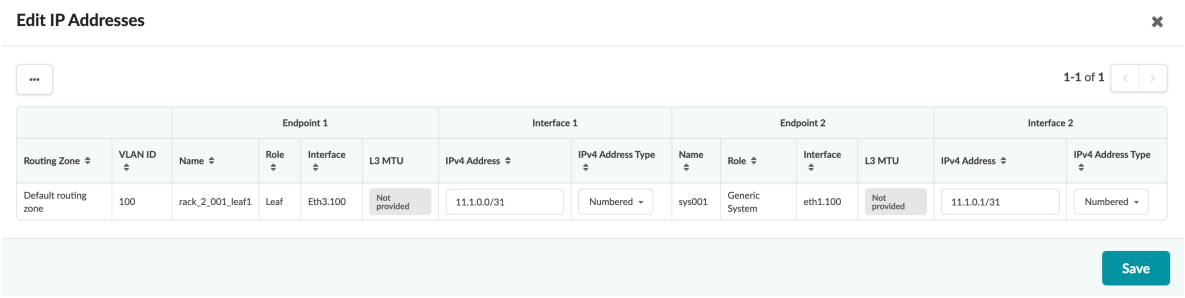
2. Find the applicable leaf device in the **System Node** column, then click the corresponding routing zone in the **IPv4 Subinterfaces** column.



3. From the routing zone details that appear, scroll to the **Interfaces** section, click the checkbox for the routing zone, then click the **Edit IP Addresses** button that appears above the table.



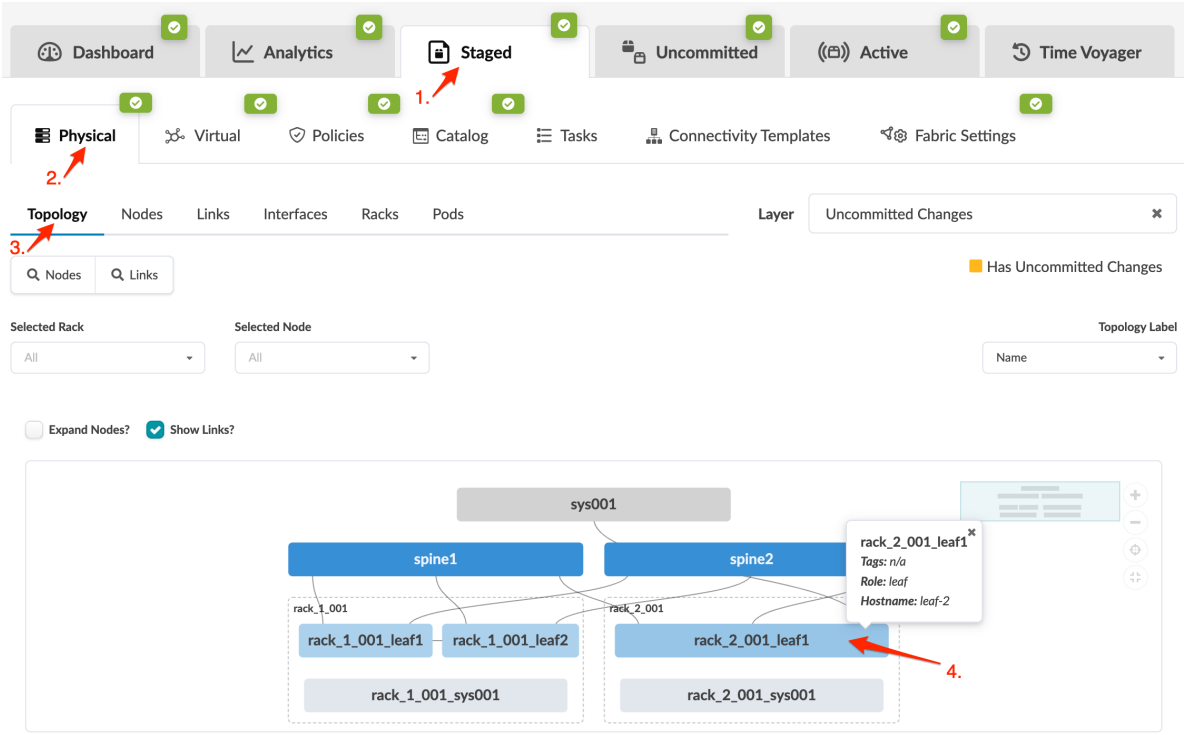
4. In the dialog that opens, change the IP addresses (and IP address type), as applicable, then click **Save** to save your changes and return to the previous screen.



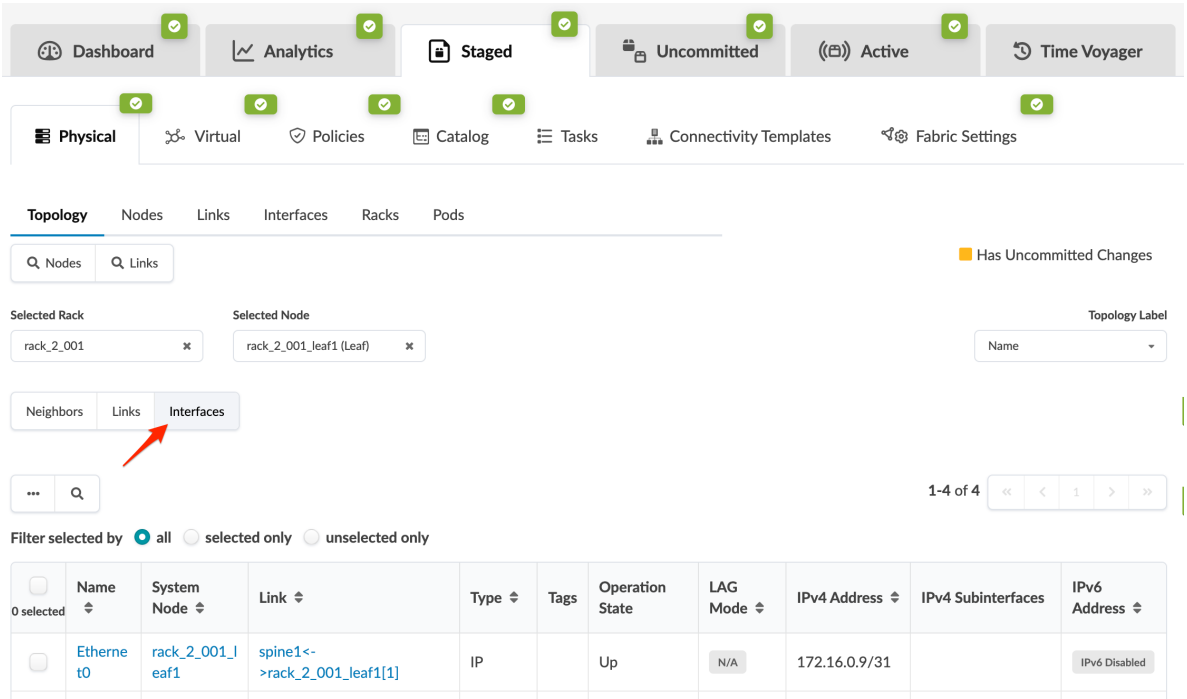
To deploy the change to the active blueprint, commit from the **Uncommitted** tab.

From Selection Interfaces Table

1. From the blueprint, navigate to **Staged > Physical > Topology** and select the leaf with the applicable interface.



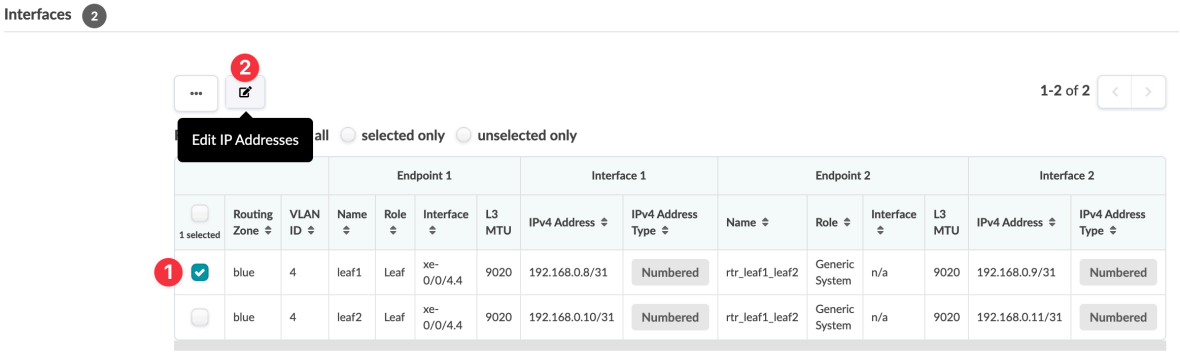
2. Above the topology for the selected node that appears, click **Interfaces** (next to **Neighbors** and **Links**) to go to the Interfaces table just for that node.



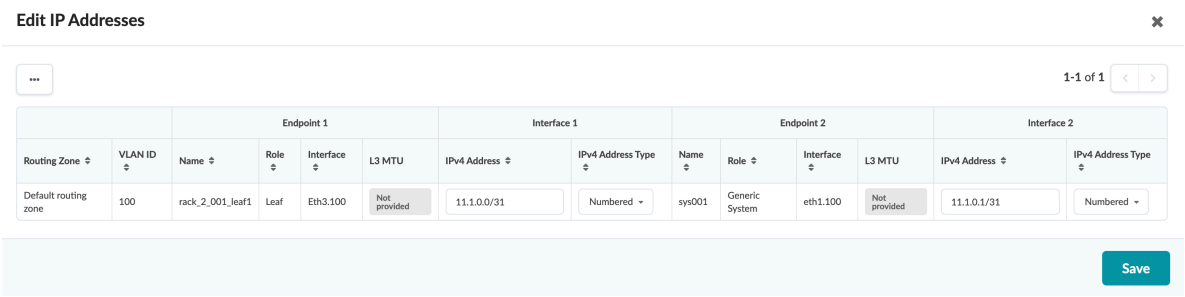
3. Click the applicable routing zone in the **IPv4 Subinterfaces** column.

0 selected	Name	System Node	Link	Type	Tags	Operation State	LAG Mode	IPv4 Address	IPv4 Subinterfaces	IPv6 Address
	Ethernet0	rack_2_001_leaf1	spine1<->rack_2_001_leaf1[1]	IP		Up	N/A	172.16.0.9/31		IPv6 Disabled
	Ethernet1	rack_2_001_leaf1	spine2<->rack_2_001_leaf1[1]	IP		Up	N/A	172.16.0.15/31		IPv6 Disabled
	Ethernet2	rack_2_001_leaf1	rack_2_001_leaf1<->rack_2_001_sys001(link)[1]	Ethernet		Up	No LAG	N/A		N/A
	Ethernet3	rack_2_001_leaf1	rack_2_001_leaf1<->sys001(ext-link-1)[1]	Ethernet		Up	No LAG	N/A	Eth3.100 - 11.1.0.0/31 RZ: Default routing zone	N/A

4. From the routing zone details that appear, scroll to the **Interfaces** section, click the checkbox for the routing zone, then click the **Edit IP Addresses** button that appears above the table.



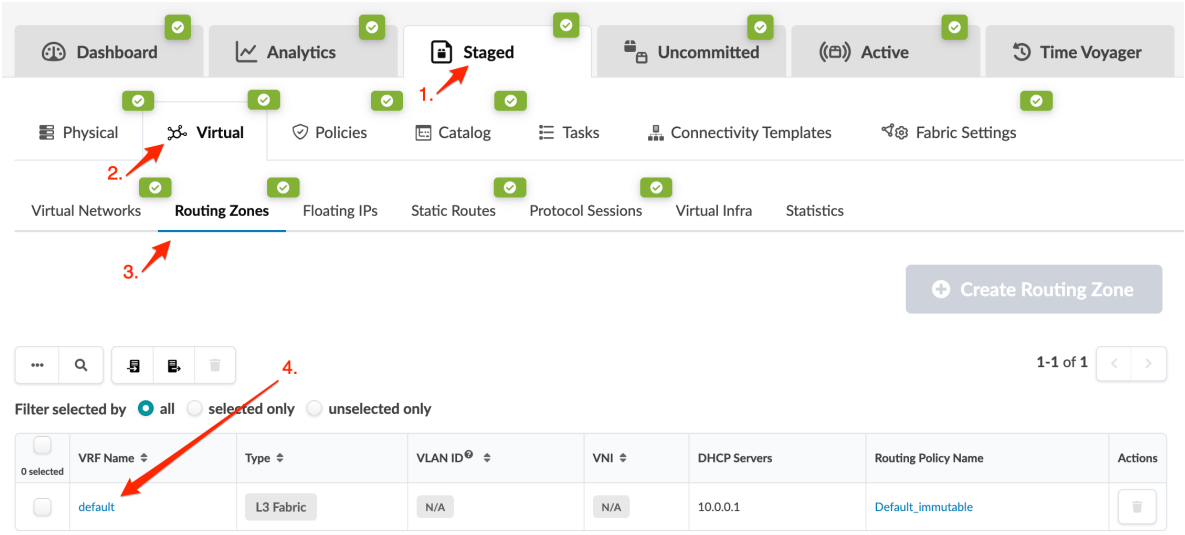
5. In the dialog that opens, change the IP addresses (and IP address type), as applicable, then click **Save** to save your changes and return to the previous screen.



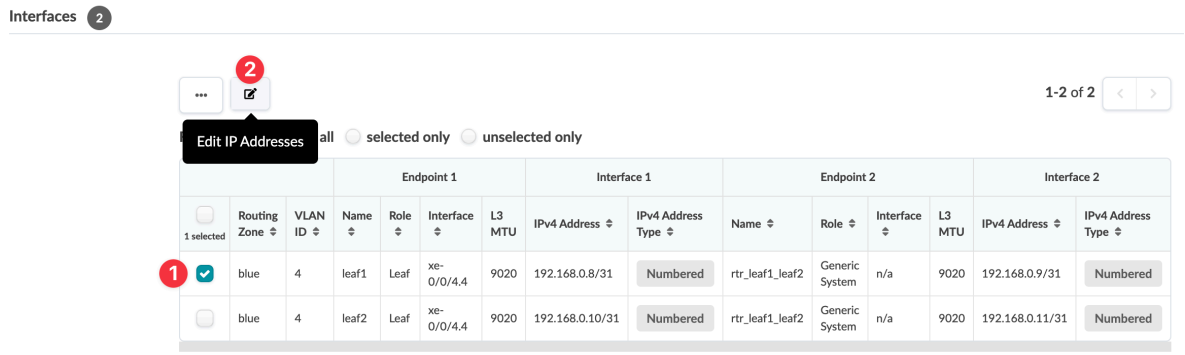
To deploy the change to the active blueprint, commit from the **Uncommitted** tab.

Directly from Routing Zone

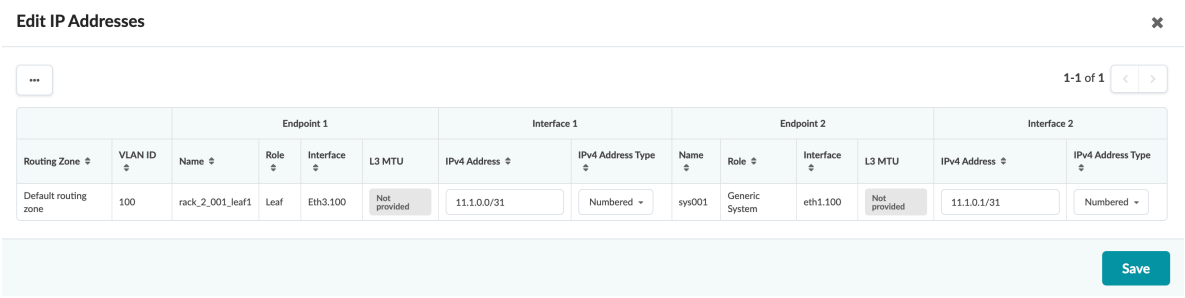
1. From the blueprint, navigate to **Staged > Virtual > Routing Zones**, then click the name of the VRF in the table.



2. From the routing zone details that appear, scroll to the **Interfaces** section, click the checkbox for the routing zone, then click the **Edit IP Addresses** button that appears above the table.



3. Change the IP addresses (and IP address type), as applicable, then click **Save** to save your changes and return to the previous screen.



To deploy the change to the active blueprint, commit from the **Uncommitted** tab.

Add / Remove Tags on Interface (Datacenter)

SUMMARY

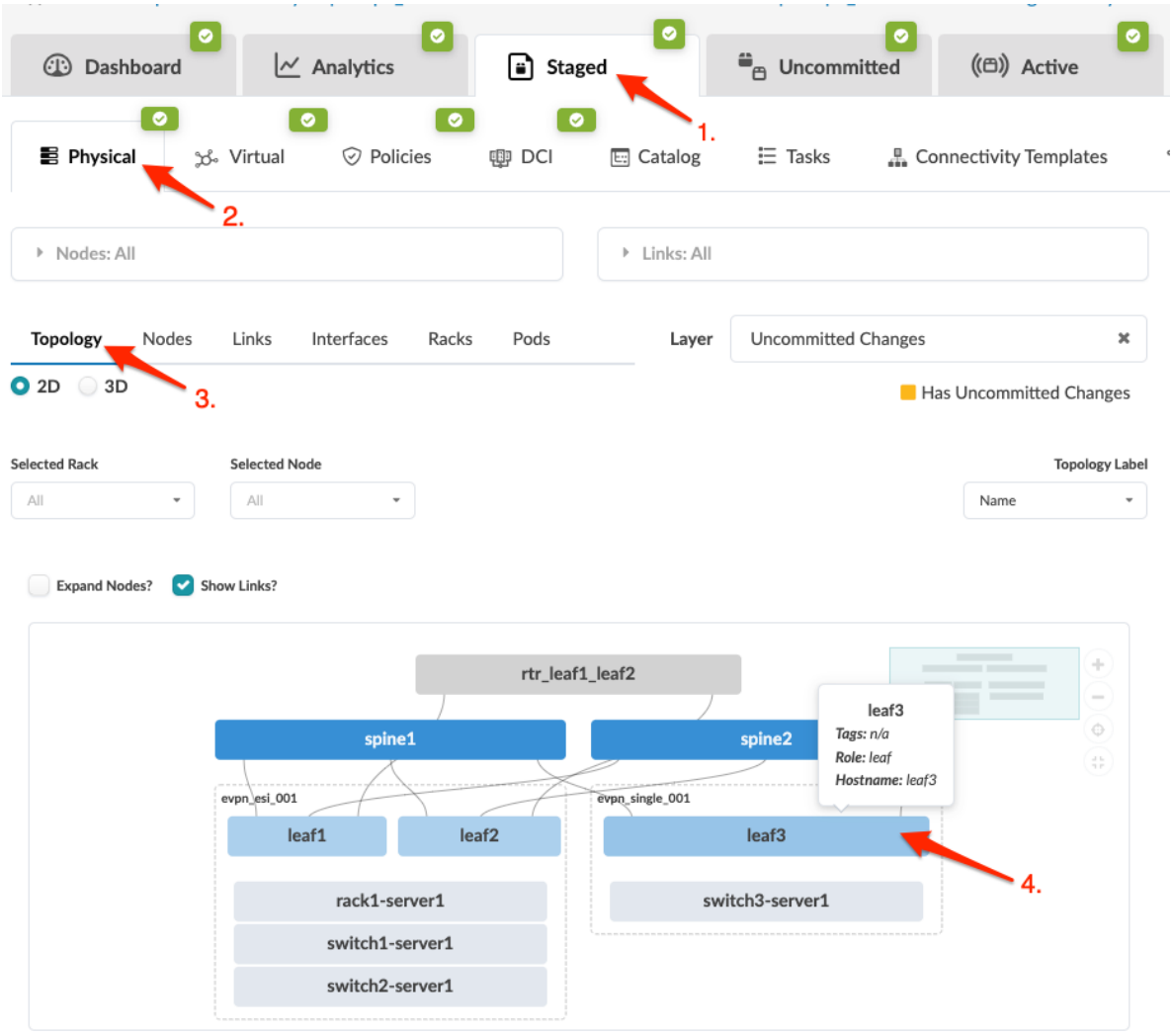
You can add tags to interfaces and use them for configuration.

IN THIS SECTION

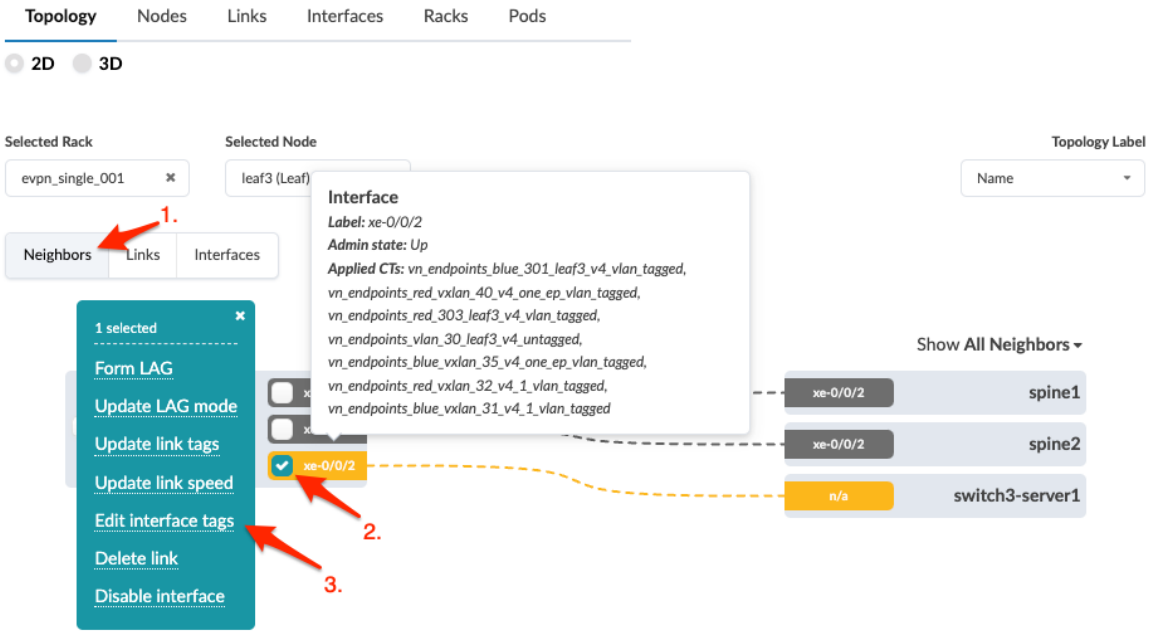
You can manage interface tags from various locations in the Apstra GUI.

Update from Topology Neighbors View

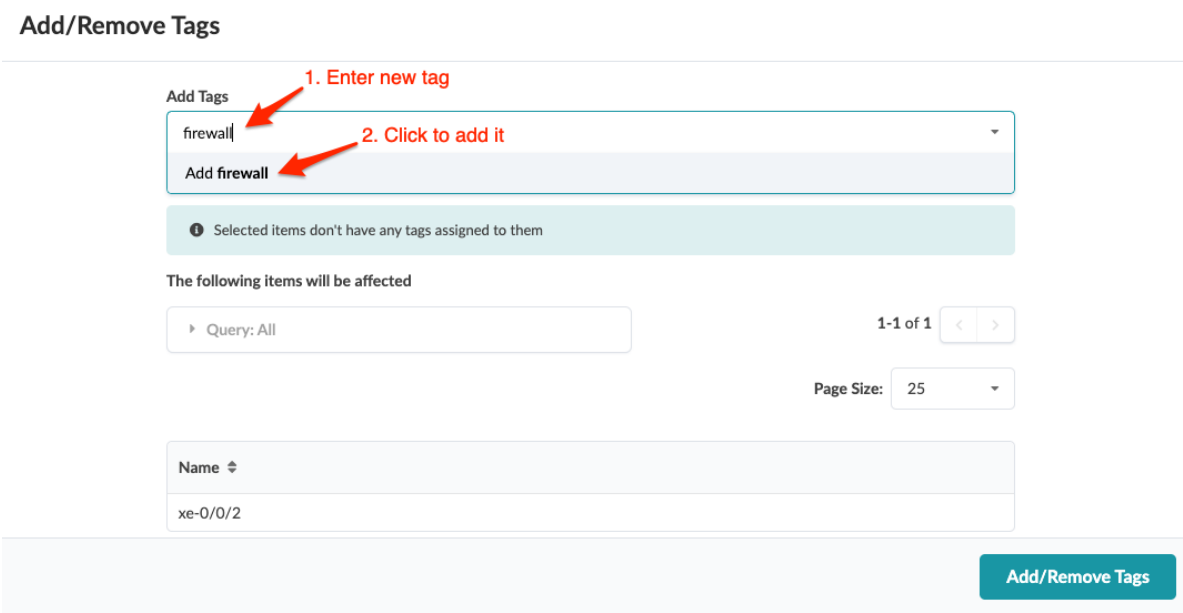
1. From the blueprint, navigate to **Staged > Physical > Topology** and select the leaf with the interface to tag.



2. If it's not already selected, click **Neighbors** view.
3. Select the check box for the applicable interface, then click **Edit interface tags**.



4. Select or deselect existing tags from the drop-down lists and/or add new tags, as needed.

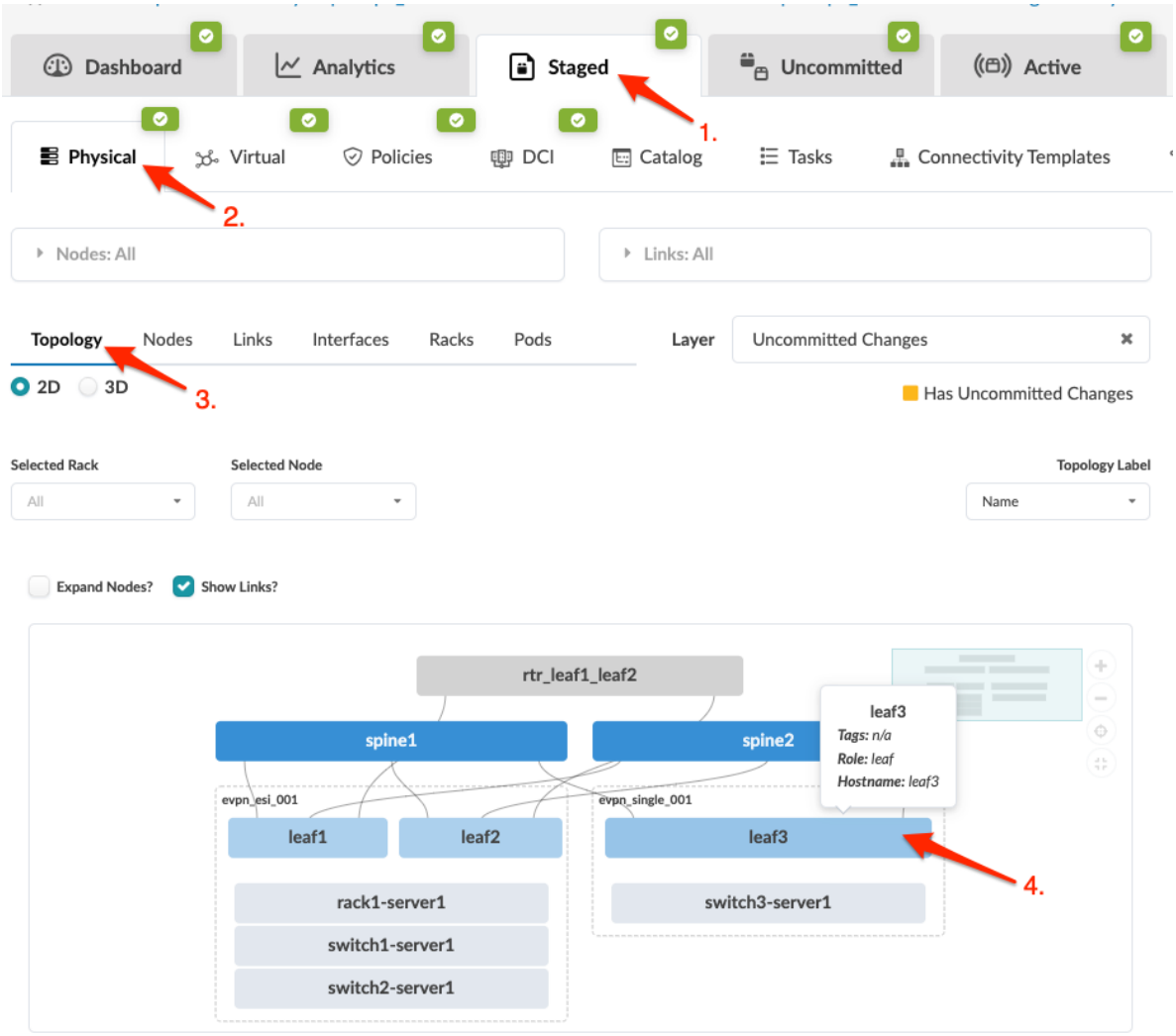


5. Click **Add/Remove Tags** to stage the tag changes and return to the interfaces table.

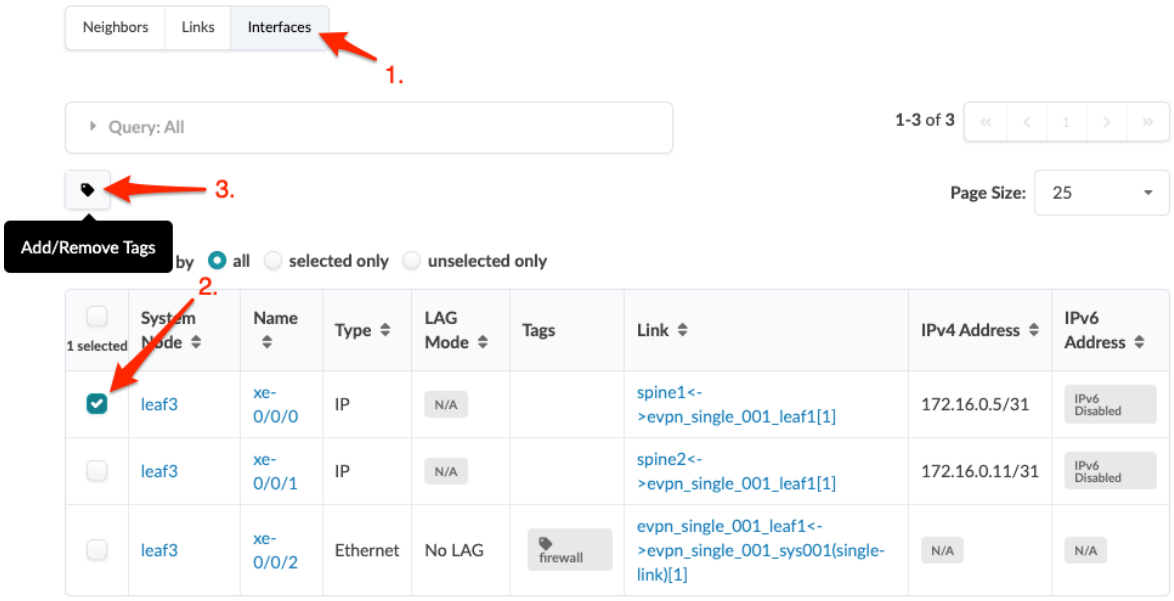
To deploy the change to the active blueprint, commit from the **Uncommitted** tab.

Update from Topology Interfaces View

1. From the blueprint, navigate to **Staged > Physical > Topology** and select the leaf with the interface to tag.



2. Click **Interfaces** view, select one or more check boxes for the interfaces to tag, then click the **Add/Remove Tags** button that appears above the table.

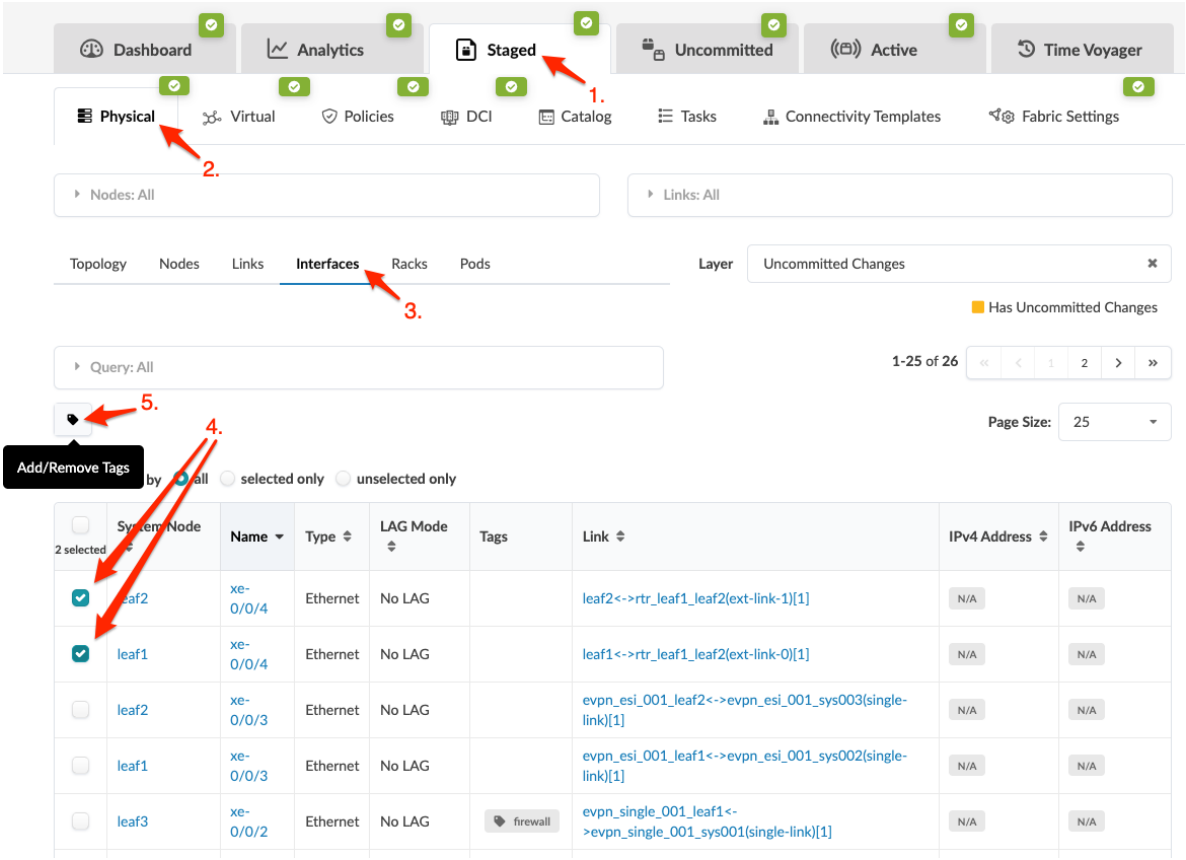


3. Select or deselect existing tags from the drop-down lists and/or add new tags, as needed.
4. Click **Add/Remove Tags** to stage the tag changes and return to the interfaces table.

To deploy the change to the active blueprint, commit from the **Uncommitted** tab.

Update from Interfaces Table

1. From the blueprint, navigate to **Staged > Physical > Interfaces** and select one or more check boxes for the interfaces to tag.



2. Click the **Add/Remove Tags** button that appears above the table.
3. Select or deselect existing tags from the drop-down lists and/or add new tags, as needed.
4. Click **Add/Remove Tags** to stage the tag changes and return to the interfaces table.

To deploy the change to the active blueprint, commit from the **Uncommitted** tab.

SEE ALSO

Interfaces Introduction | 207

What are Tags | 922

Commit / Revert Changes to Blueprint | 599

Add / Remove Tags on Port Channel (Datacenter)

SUMMARY

You can add tags to aggregated logical interfaces (port channels) and use them for configuration.

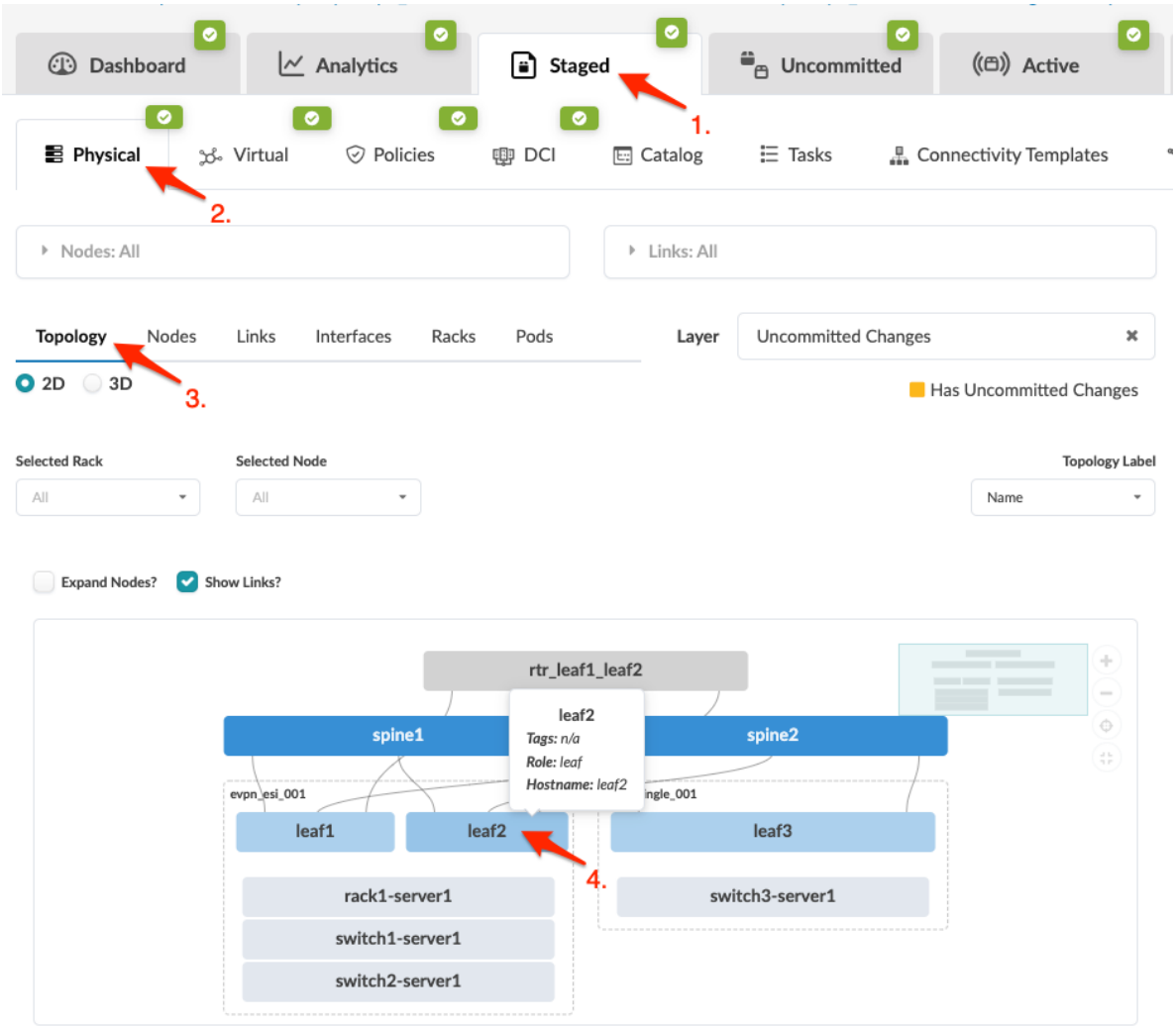
IN THIS SECTION

You can manage aggregated logical interfaces, also known as port channels, from various locations in the Apstra GUI.

Update from Topology Neighbors View

Update from Topology Interfaces View

1. From the blueprint, navigate to **Staged > Physical > Topology** and select the leaf with the interface to tag.



2. Click **Interfaces** view, then select one or more check boxes for the interfaces to tag.
3. Add and/or remove tags from the drop-down lists, as needed.
4. Click **Add/Remove Tags** to stage the tag changes and return to the interfaces table.

Update from Interfaces Table

1. From the blueprint, navigate to **Staged > Physical > Interfaces** and select one or more check boxes for the interfaces to tag.
2. Click the **Add/Remove Tags** button that appears above the table.
3. Add and/or remove tags from the drop-down lists, as needed.
4. Click **Add/Remove Tags** to stage the tag changes and return to the interfaces table.

SEE ALSO

Interfaces Introduction | 207

What are Tags | 922

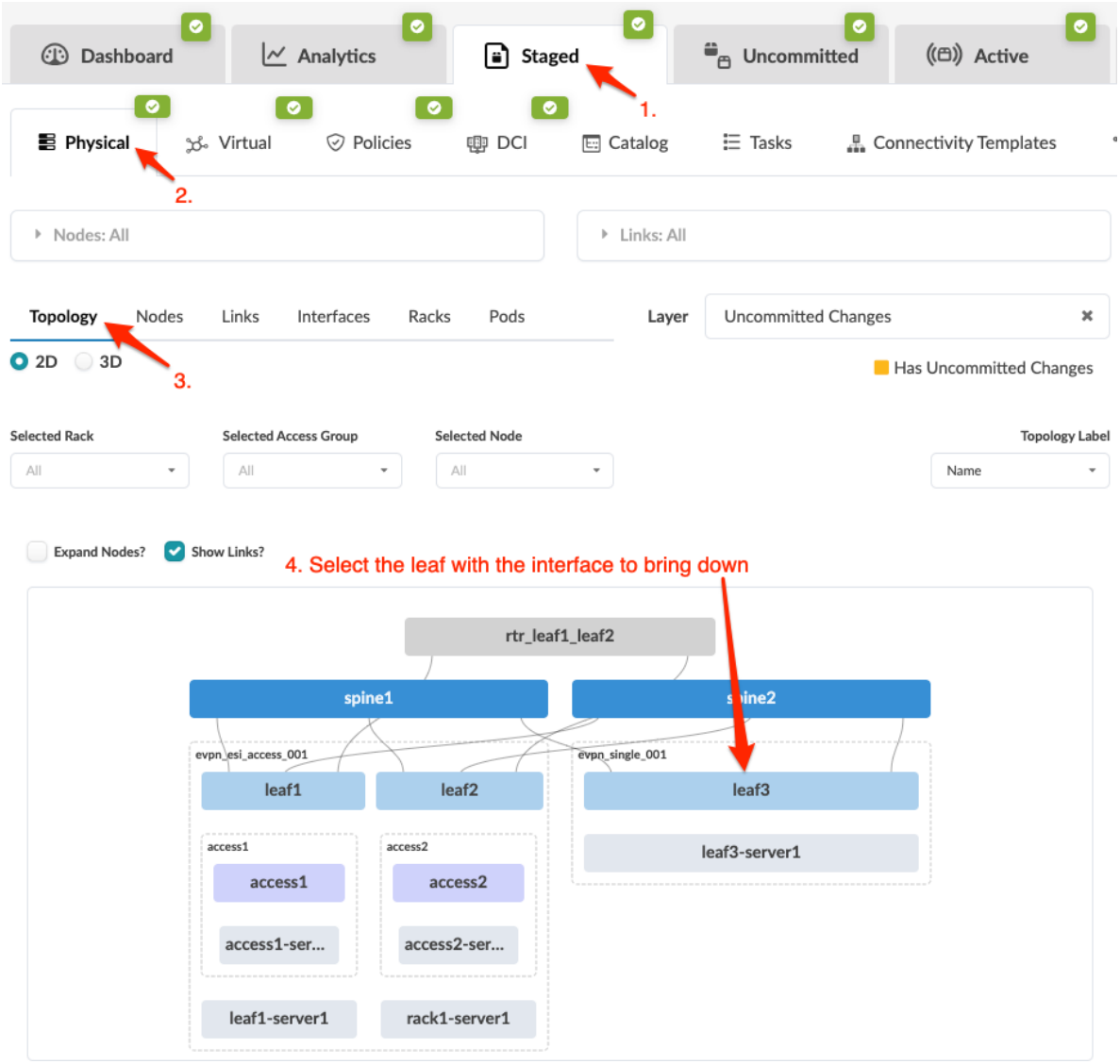
Commit / Revert Changes to Blueprint | 599

Administratively Disable Interface

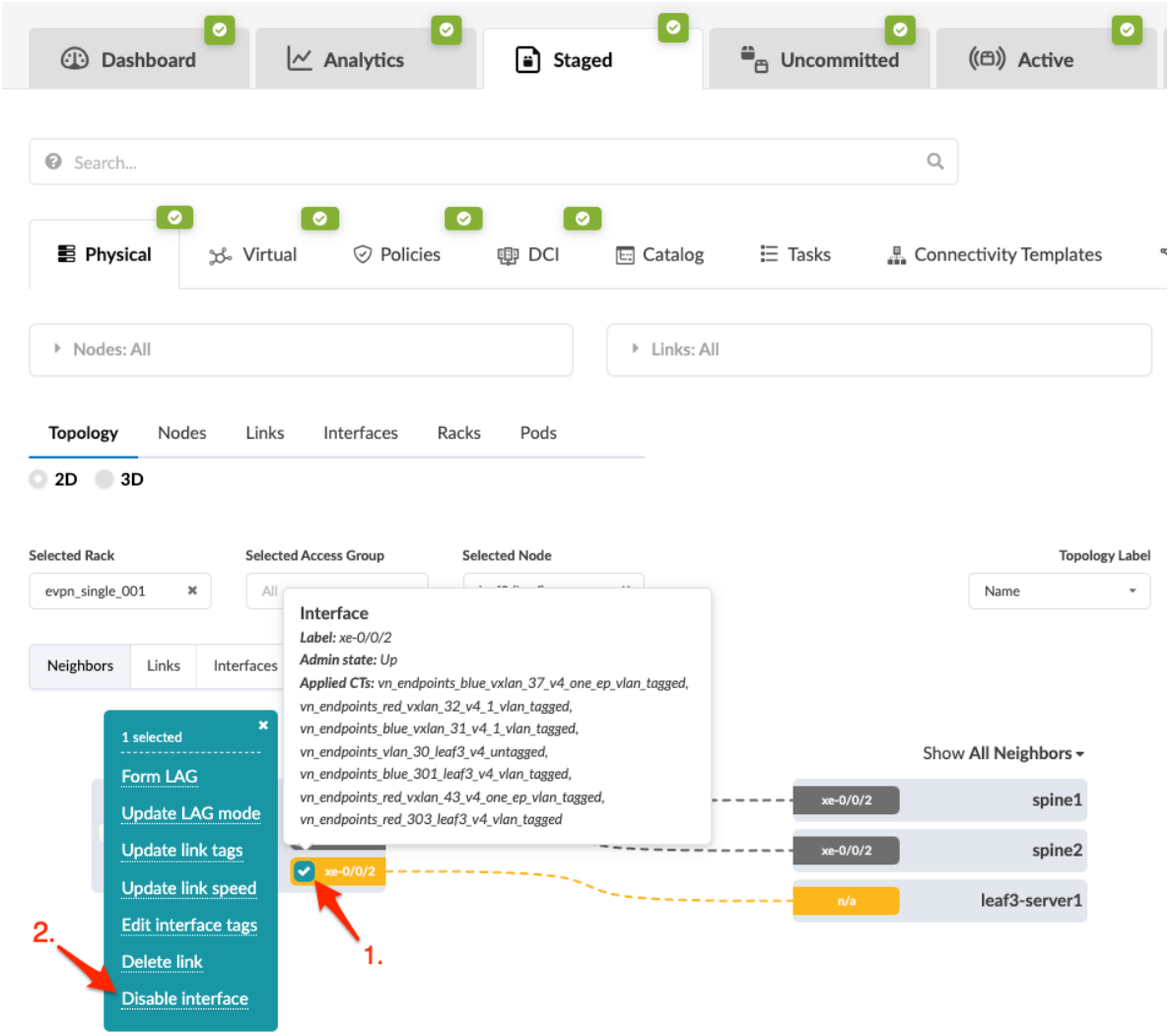
SUMMARY

You can administratively disable an interface from the Apstra GUI.

1. From the blueprint, navigate to **Staged > Physical > Topology** and select the leaf that's connected to the applicable generic system or external generic system.



2. Select the check box for the applicable interface, then click **Disable interface** to stage the change.



To deploy the change to the active blueprint, commit from the **Uncommitted** tab.

RELATED DOCUMENTATION

Interfaces Introduction | 207

Administratively Enable Interface | 226

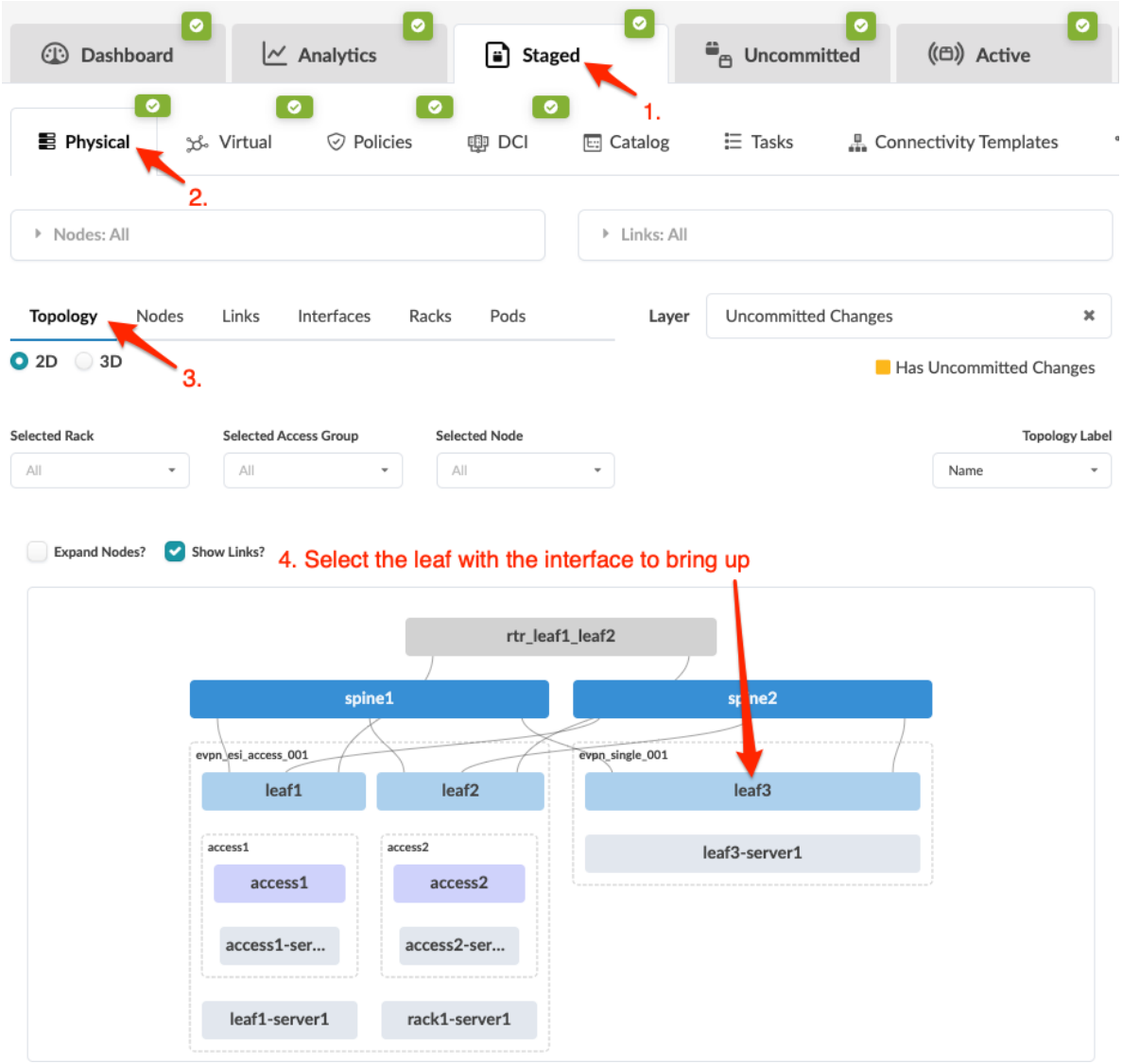
Commit / Revert Changes to Blueprint | 599

Administratively Enable Interface

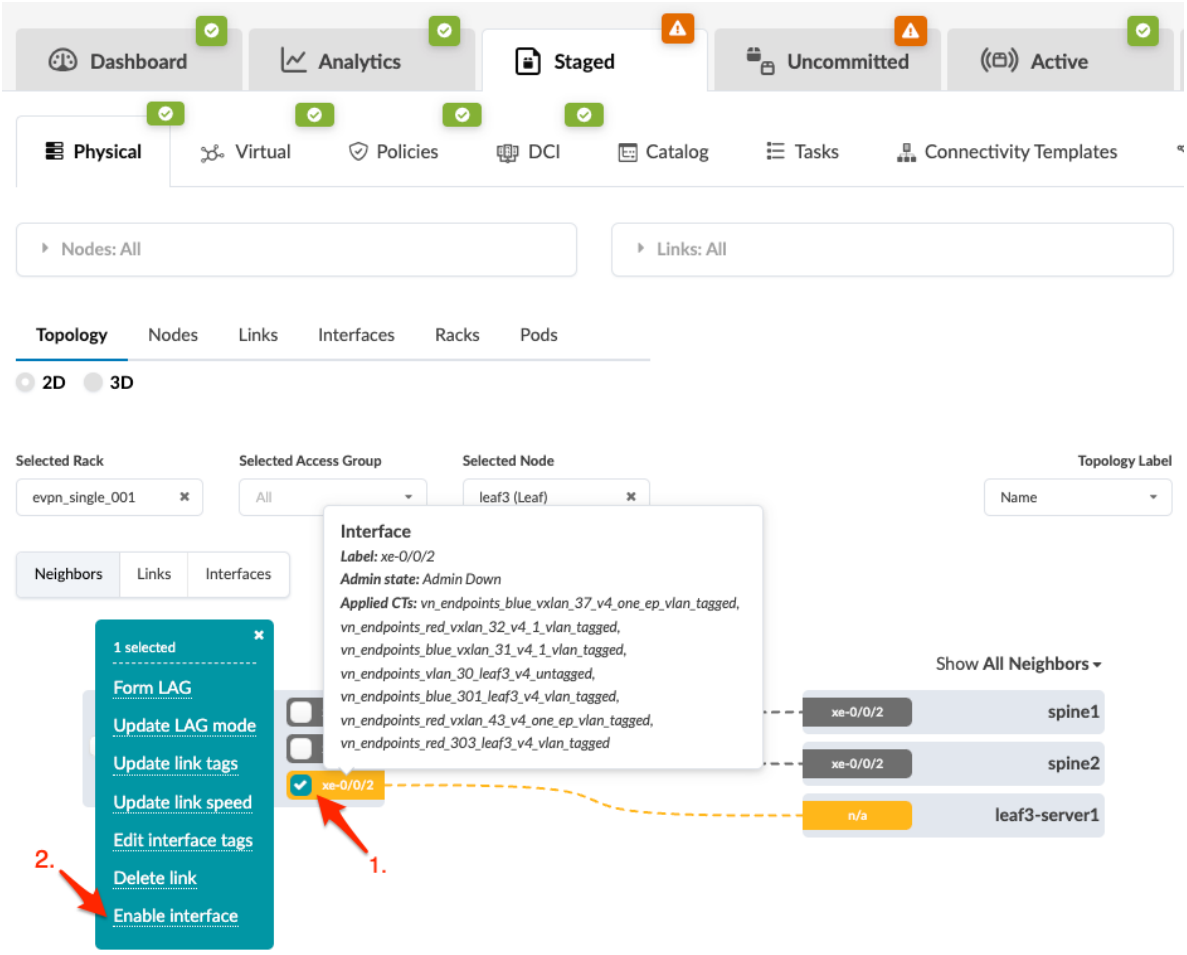
SUMMARY

You can administratively enable an interface from the Apstra GUI.

1. From the blueprint, navigate to **Staged > Physical > Topology** and select the leaf that's connected to the applicable generic system or external generic system.



2. Select the check box for the applicable interface, then click **Enable interface** to stage the change.



To deploy the change to the active blueprint, commit from the **Uncommitted** tab.

RELATED DOCUMENTATION

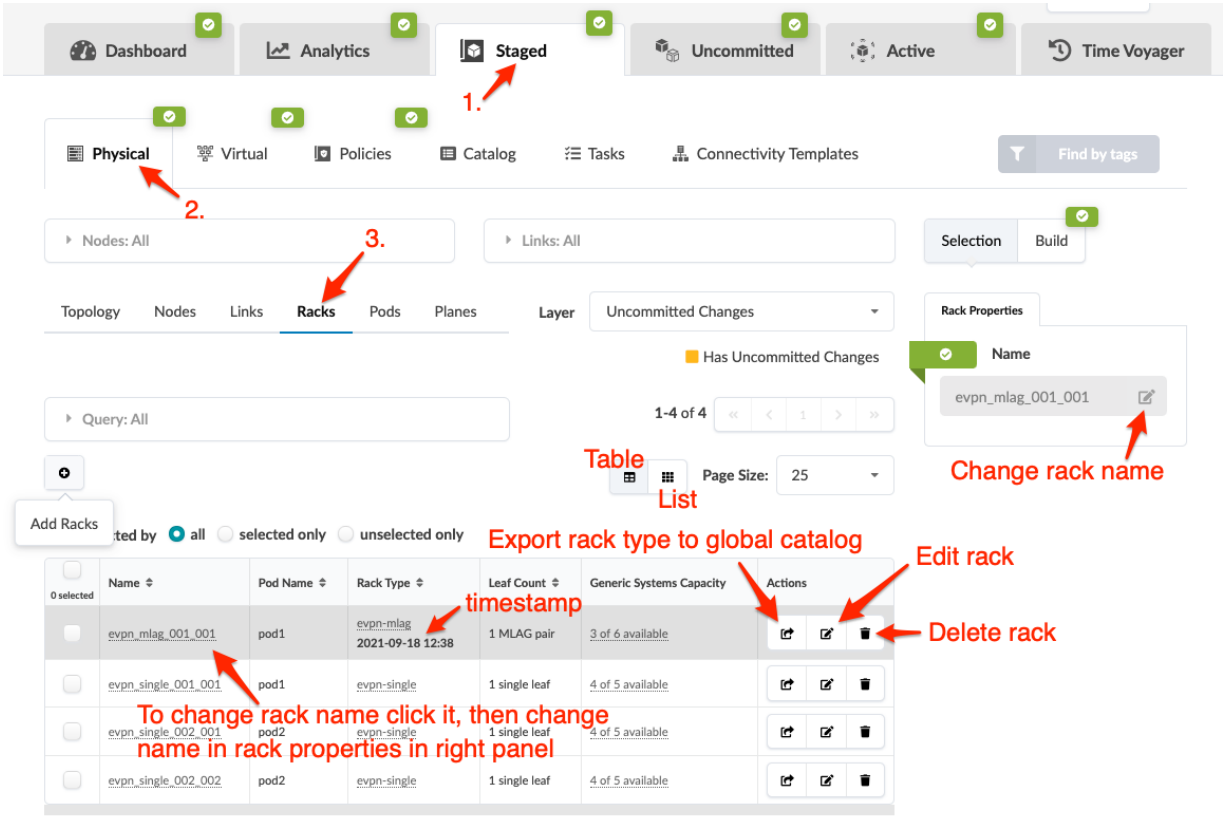
- Interfaces Introduction | 207
- Administratively Disable Interface | 224
- Commit / Revert Changes to Blueprint | 599

Racks

IN THIS SECTION

Racks (Datacenter)

From the blueprint, navigate to **Staged > Physical > Racks** to go to the **Racks** view.



- You can view racks in table view or card view.

- You can filter racks to show **all**, **selected only**, or **unselected only**.

You can control the growth of your network by adding, editing and deleting complete racks in a running blueprint. This flexible fabric expansion (FFE) feature is supported on both 3-stage and 5-stage Clos networks. (In 5-stage topologies, you can also "add and remove pods" on page 237, "increase the number of superpines per plane" on page 246. Although, you can't add or remove planes themselves.) You can also "change rack names" on page 230.

Rack types are *embedded* into blueprints from the global catalog. The rack type in the global catalog and the blueprint are initially the same. When you use FFE operations (for example to change link speeds, add generic systems or add/remove links) the rack type is modified and its timestamp is updated. The rack type name in the global catalog and the blueprint are still the same, but their contents are now different from each other.

See the following sections for more information on rack operations.

Change Rack Name

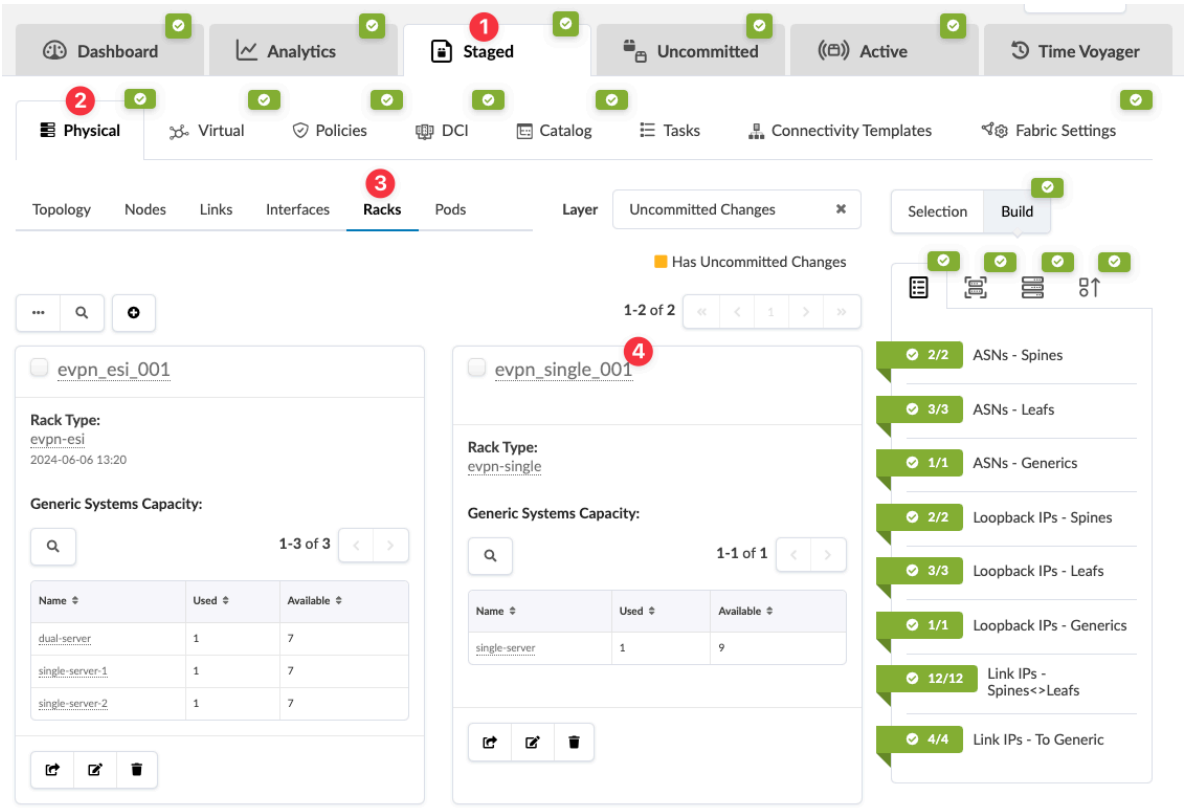
SUMMARY

You may want to use your own rack naming schema, to be based on their physical locations, for example. In these cases you can change the default rack names. You can change the name of a single rack or of multiple racks simultaneously.

IN THIS SECTION

Change the Rack Name of a Single Rack

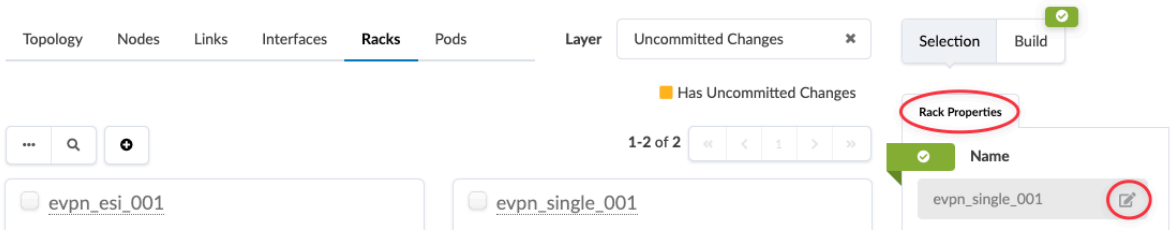
1. From the blueprint, navigate to **Staged > Physical > Racks** and click the *name* of the applicable rack (not the check box).



The **Selection** panel opens (on the right).

> **NOTE**: You can also get to the **Selection** panel from the **Topology** view (Staged > Physical > Topology) by selecting the rack from the **Selected Rack** drop-down list.

2. From the **Rack Properties** tab in the **Selection** panel, click the **Edit** button for **Name**, change the name accordingly, then click the **Save** button.

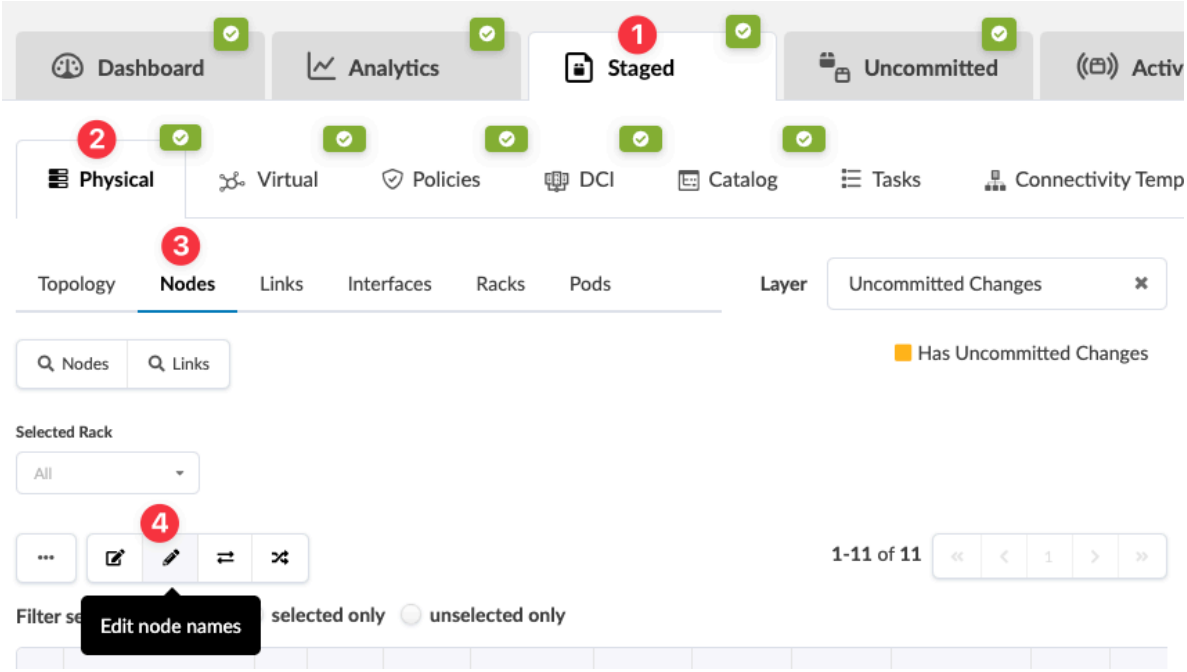


When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

Change Rack Names of Multiple Racks

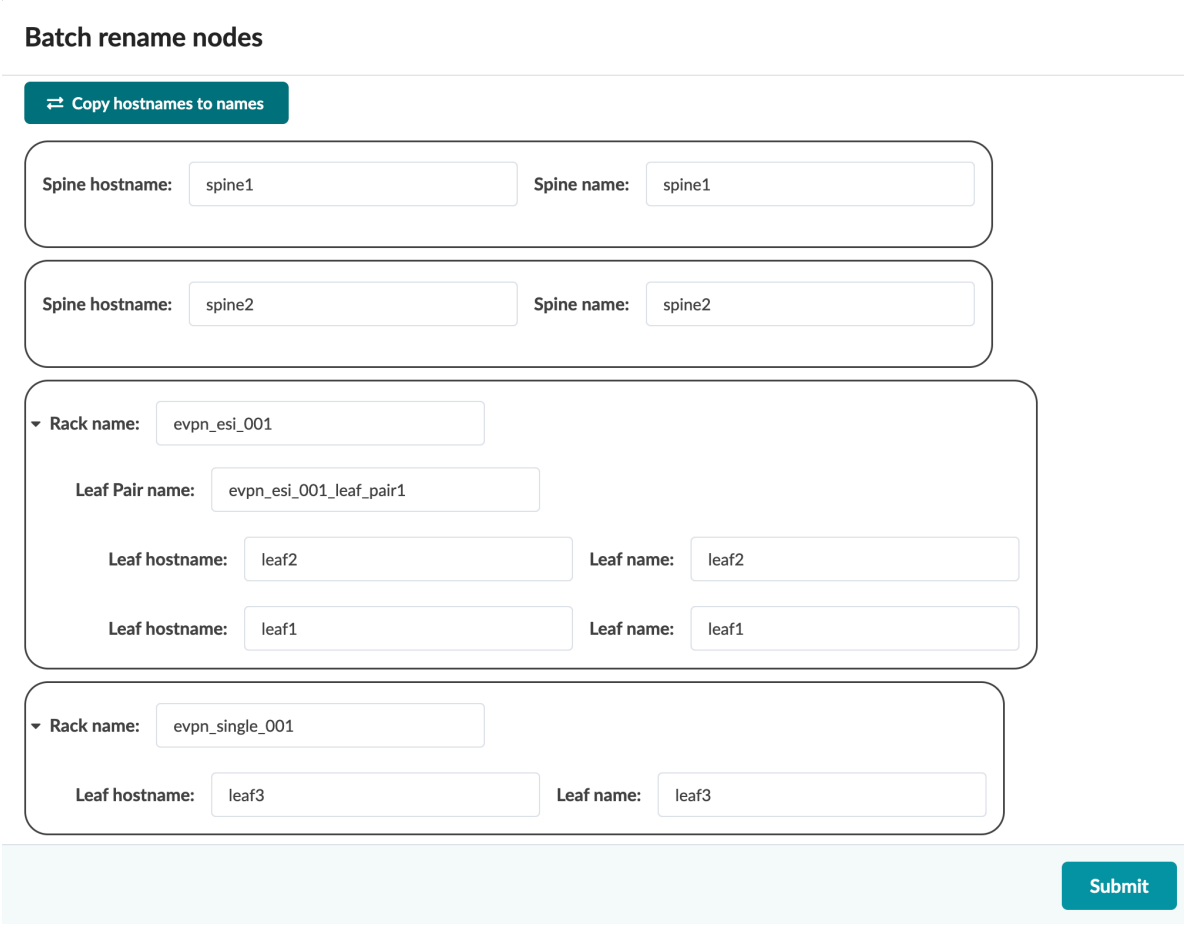
New feature in Apstra version 5.0.0.

1. From the blueprint, navigate to **Staged > Physical > Nodes** and click the **Edit Node Names** button, above the table.



   The **Batch rename nodes** dialog opens.

2. Change rack names, as applicable. (Notice that you can change other various hostnames and names from this dialog as well.)



3. Click **Submit** to stage the changes and return to the **Nodes** table.

When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

Add Rack

The easiest and fastest way to expand your network is to add a rack.

1. From the blueprint, navigate to **Staged > Physical > Racks** and click the **Add Racks** button (+).
2. If your blueprint is for a 5-stage topology, select the pod that needs a rack.
3. From the **Rack Type** drop-down list, select a rack type to preview and validate. (To go to a different preview, select a different rack type.)
4. Enter the number of racks to add.
5. If you uncheck **Keep existing cabling in the fabric after change**, port assignments are re-calculated and you may need to re-cable. When in doubt, leave this box checked.
6. Click **Add** to stage the rack addition and return to the table view.
7. "Assign device profiles" on page 62 and "system IDs" on page 64 (serial numbers) to the new rack(s).
8. Commit the changes to your blueprint to configure the rack(s) and complete the fabric expansion.

Next Steps:

To assign virtual networks to your new rack, see "Assign / Unassign Virtual Networks" on page 268. You can assign many VNs at the same time to one or more nodes.

Export Rack Type

If you can't make certain changes directly in the blueprint rack, you can export the rack type to the global catalog and update it there.

1. From the blueprint, navigate to **Staged > Physical > Racks** and click the **Export rack to global catalog** button (first of three buttons).

   NOTE: If the rack type is inconsistent with the same-named one in the global (design) catalog, you won't be able to export the rack type. Rack types are embedded in blueprints from the global catalog. When you use Flexible Fabric Expansion (FFE) operations (for example to change link speeds, add generic systems or add/remove links) the blueprint rack type is modified. The rack type name in the global catalog and the blueprint are still the same, but their contents are now different from each other. When rack types are inconsistent, you can create a rack type in the global catalog that meets your new requirements.

2. Enter a unique **Rack Type** name.
3. Click **Export** to export the rack type to the global catalog.

Next Steps: From the left navigation menu, navigate to **Design > Rack Types** and edit the rack type in the global catalog. (Or, if you couldn't export the rack type, create one that meets your new requirements.) Then from the blueprint, "Update the rack" on page 234 to use the revised (or new) rack type from the global catalog.

Change Rack

You can change running racks while preserving many rack characteristics (such as leaf/server/link names and virtual network (VN) endpoints if labels have not changed). To edit a rack, you export its rack type to the global catalog with a unique name, update that rack type in the global catalog, then, in the blueprint, select the updated rack type to replace the one in the blueprint.

VN endpoints remain as long as the server and link labels between the old and new rack type are the same.

CAUTION: If it's not possible to retain VN endpoints, you must re-assign them. Review pending changes on the **Uncommitted** tab before committing. If you don't want to commit the changes, you can **revert** them.

NOTE: If you don't need to retain rack details, we recommend that you "delete the rack" on page 235 and "add a replacement rack" on page 233, instead of editing the rack.

Typically, a rack edit operation involves the following steps:

1. Ensure that the global catalog or the blueprint includes a suitable rack type for replacement.
2. From the blueprint, navigate to **Staged > Physical > Racks** and click the **Edit** button for the rack to edit (second of three buttons).
3. From the **New Rack Type** drop-down list, select the required rack type.
4. If you added new devices, "assign device profiles" on page 62 and "system IDs" on page 64 (serial numbers) to them.

   CAUTION: This action is service-impacting since it requires a full config push.

5. You have the option of reviewing the **Incremental Config** to see the changes that will be pushed to the device(s). If devices were assigned, a full config push is performed.
6. Commit the changes to the blueprint to push all required configuration changes to the devices in the modified rack.

RELATED DOCUMENTATION

What are Rack Types | 862

Delete Rack

Before deleting a rack that has live traffic on it, you may want to take its devices out-of-service by draining them. For information, see "Drain Device Traffic" on page 652.

1. To delete a rack from the blueprint, navigate to **Staged > Physical > Racks** and click the **Delete** button for the rack to delete (third of three buttons).
   - If you will be adding a rack back into your system, leave the **Keep existing cabling in the fabric after change** box checked.
   - If you will *not* be replacing the rack in your system, uncheck the **Keep existing cabling in the fabric after change** box. Otherwise, the intent will not match the actual topology anymore, and you will encounter anomalies, such as for cabling and BGP.
2. Click **Delete Rack** to stage the deletion and return to the table view.

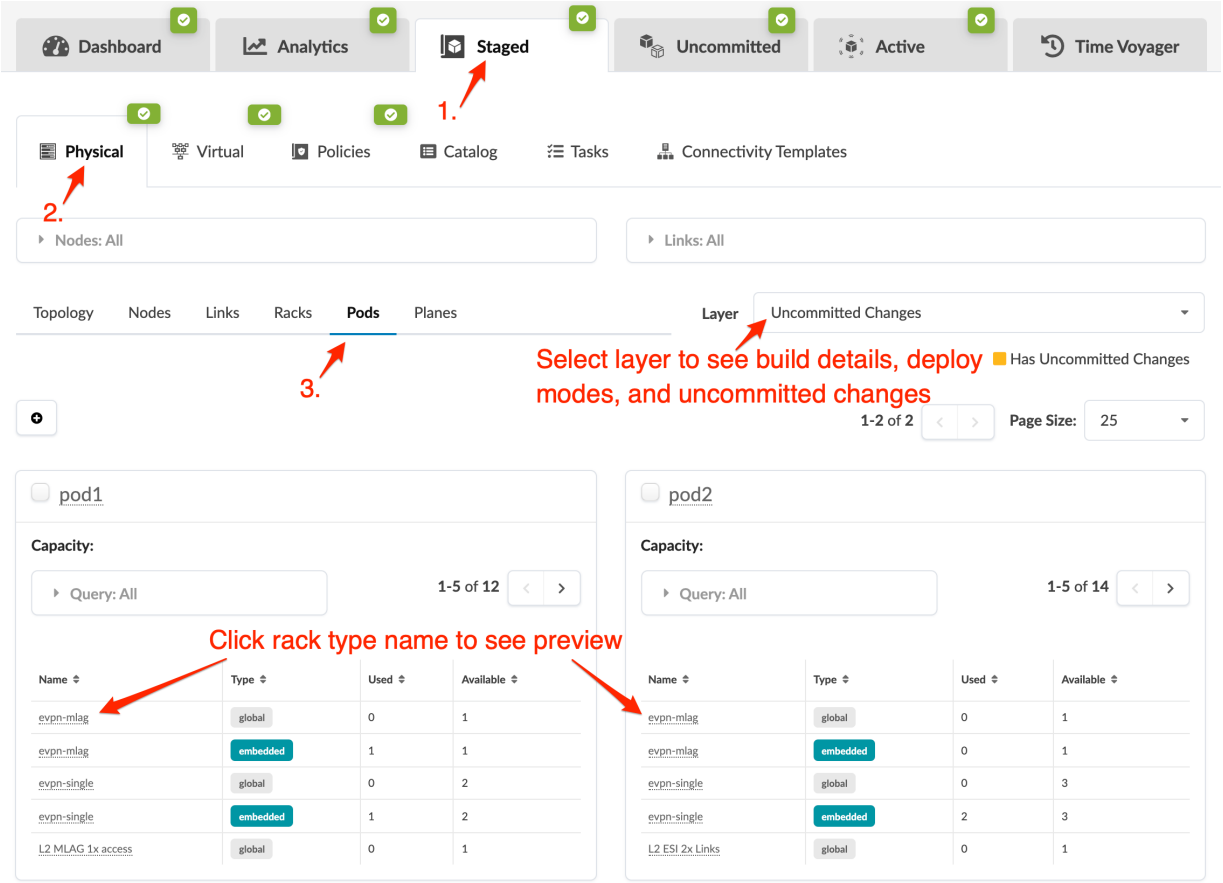
3. Commit the changes to the blueprint. Configuration on any running devices will be erased and the devices will be ready to be decommissioned.

Pods

IN THIS SECTION

Pods (Datacenter)

From the blueprint, navigate to **Staged > Physical > Pods** to go to the **Pods** view.



You can search for specific nodes or links.

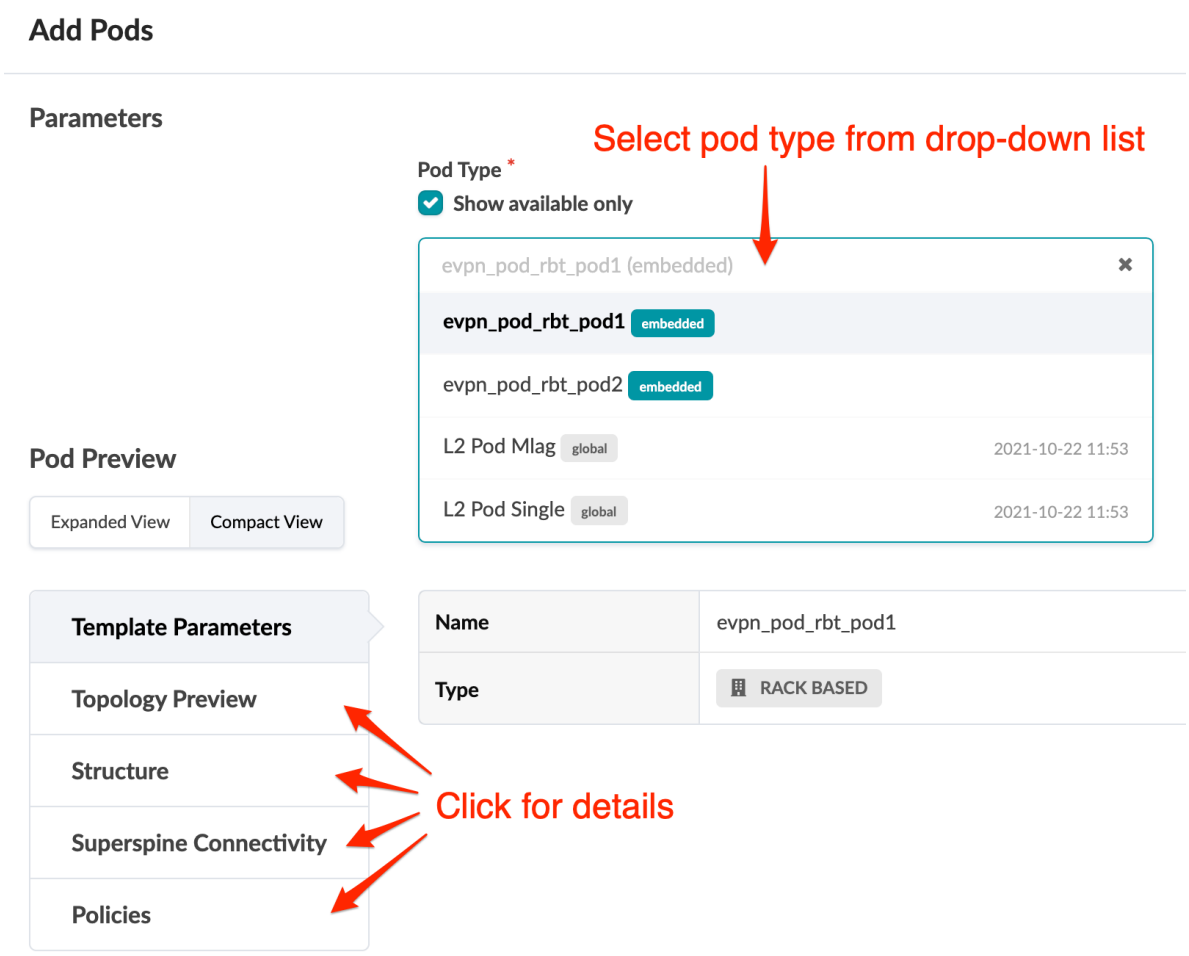
From the **Pods** view, you can view pod capacity and change pod names. 3-stage topologies can have only one pod. If your topology is for 5-stage, you can add and remove entire pods. The ability to add pods to your running blueprint allows for organic growth of large networks without having to pre-design every pod. For more information about building 5-stage topologies, see "5-stage Clos Architecture" on page 1425.

See the following sections for more information about adding, editing and deleting pods.

Add Pod (5-Stage Only)

You can add pods to 5-stage topologies, but not to 3-stage topologies.

1. From the blueprint, navigate to **Staged > Physical > Pods**, and click the **Add Pods** button (+) (center-left). (This button is disabled on 3-stage topologies.)
2. From the **Pod Type** drop-down list, select a pod type to preview and validate. To go to a different preview, select a different pod type.



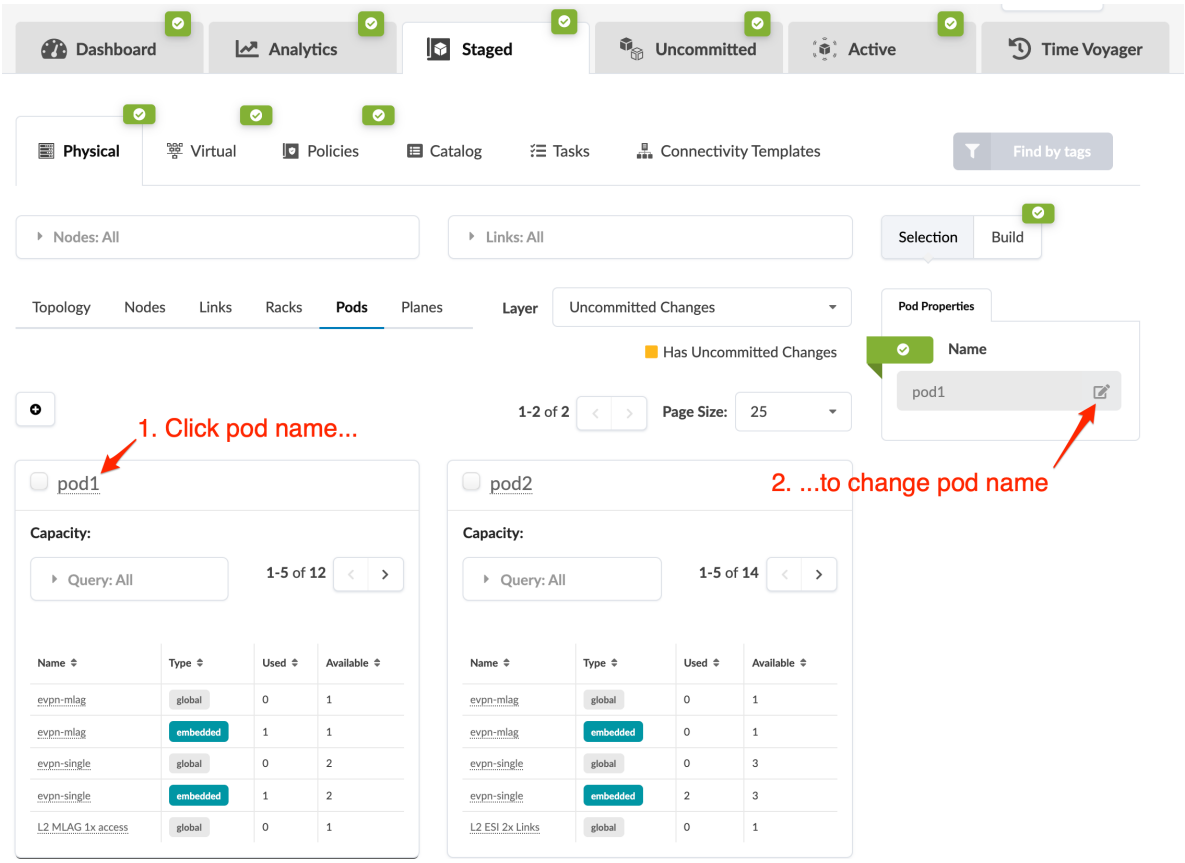
3. Enter the number of pods to add.
4. Click **Add** to stage the pod addition and return to the table view.
5. Commit the changes to your blueprint to complete the fabric expansion.

RELATED DOCUMENTATION

Commit / Revert Changes to Blueprint | 599

Change Pod Name

1. From the blueprint, navigate to **Staged > Physical > Pods** and click the pod name to change.



2. In **Pod Properties** (right panel) click the **Edit** button for the name.
3. Change the name and click the **Save** button to stage the change.
4. Commit the changes to your blueprint to activate the name change.

RELATED DOCUMENTATION

Commit / Revert Changes to Blueprint | 599

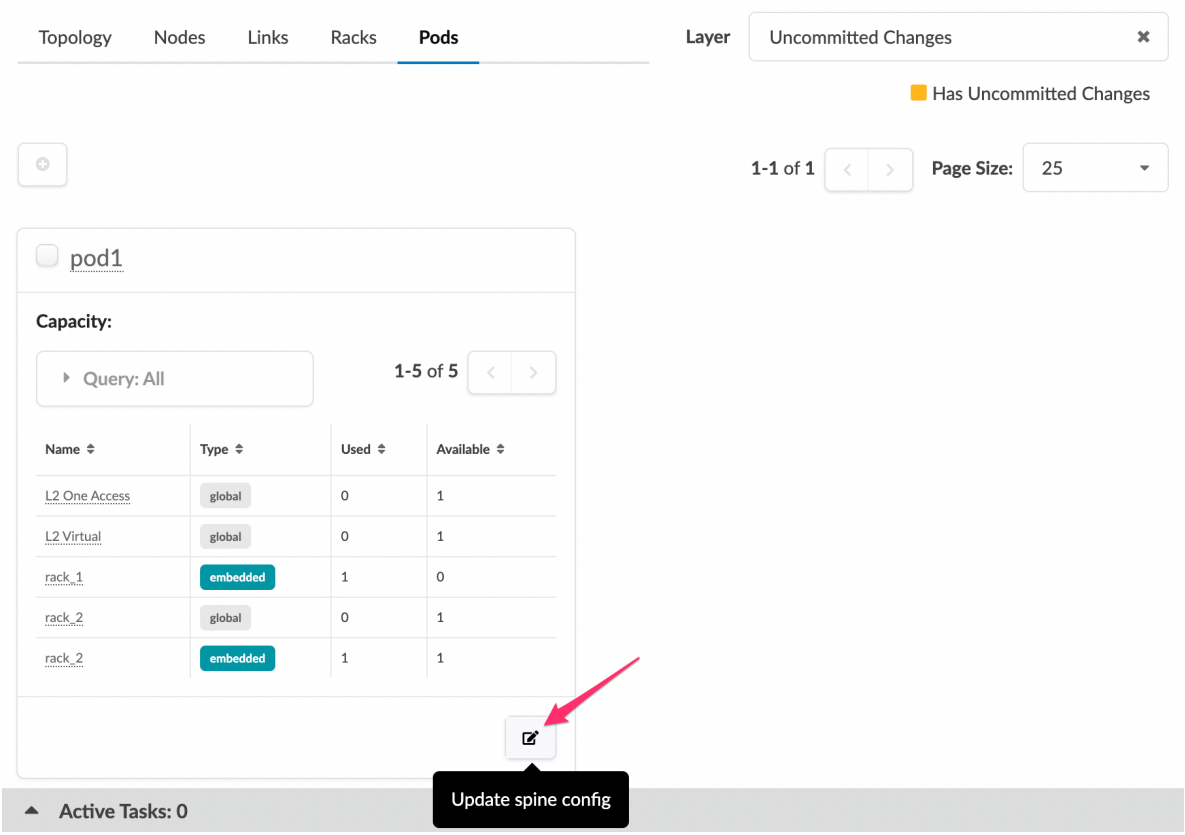
Add Spine per Pod

As a Day 2 operation, you can add spines per pod on both 3-stage and 5-stage blueprints.

> **CAUTION**: Plan carefully. After you've added spines, you won't be able to remove them.

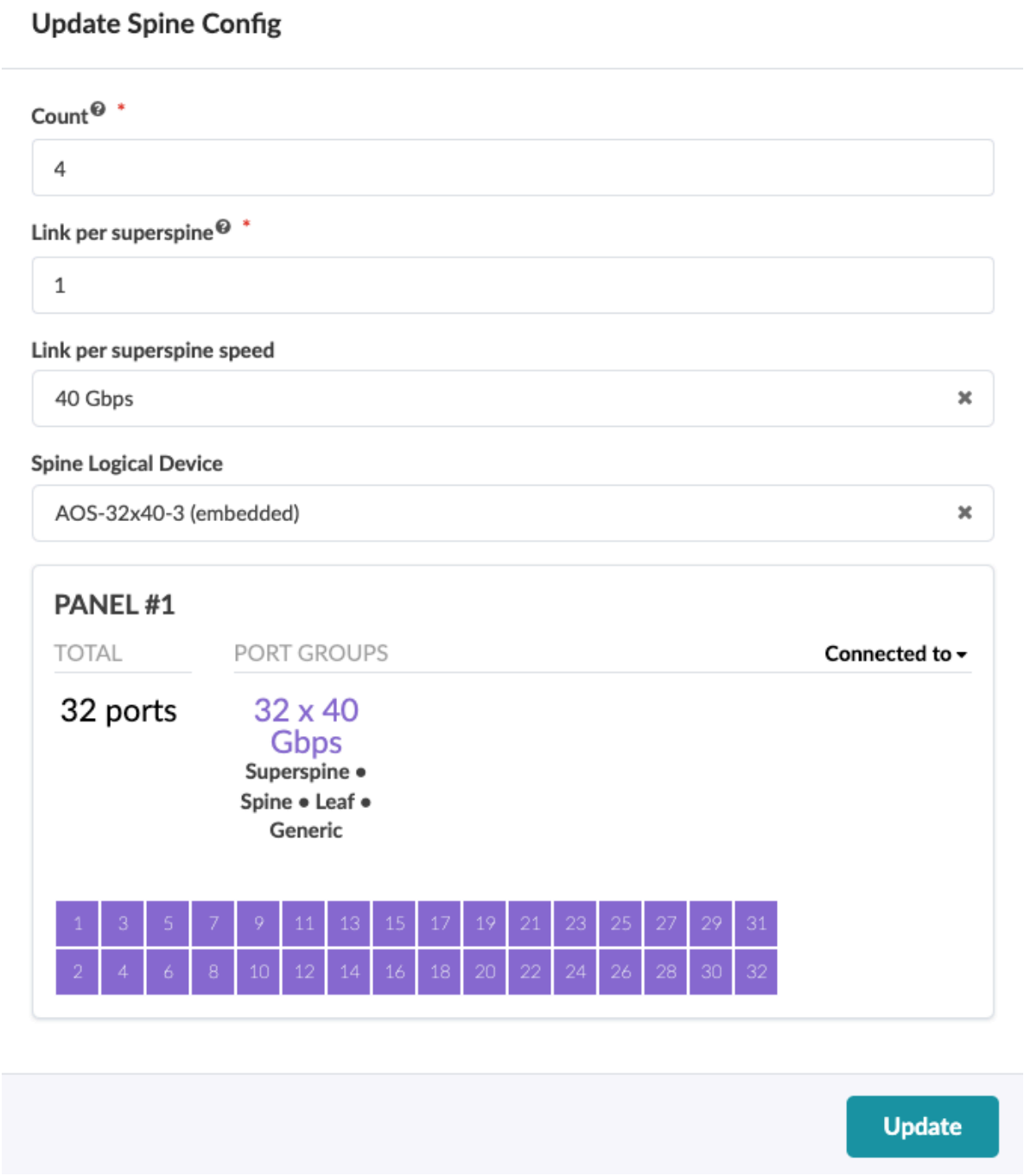
Make sure you have enough ports with specific roles and speeds for additional spine(s).

1. From the blueprint, navigate to **Staged > Physical > Pods**.
2. Click the **Update spine config** button on the bottom-right of the card for the pod to change.



3. In the **Count** field, enter the total number of spines you want:
   - You can only *increase* the number of spines.
   - On 5-stage blueprints, the number of spines must be a multiplier of the number of superspine planes.

> ⚠ **CAUTION**: Plan carefully. After you've added spines, you won't be able to remove them.



4. Click **Update** to stage your changes and return to the **Pods** view.

When you're ready to activate changes, commit them from the Uncommitted tab.

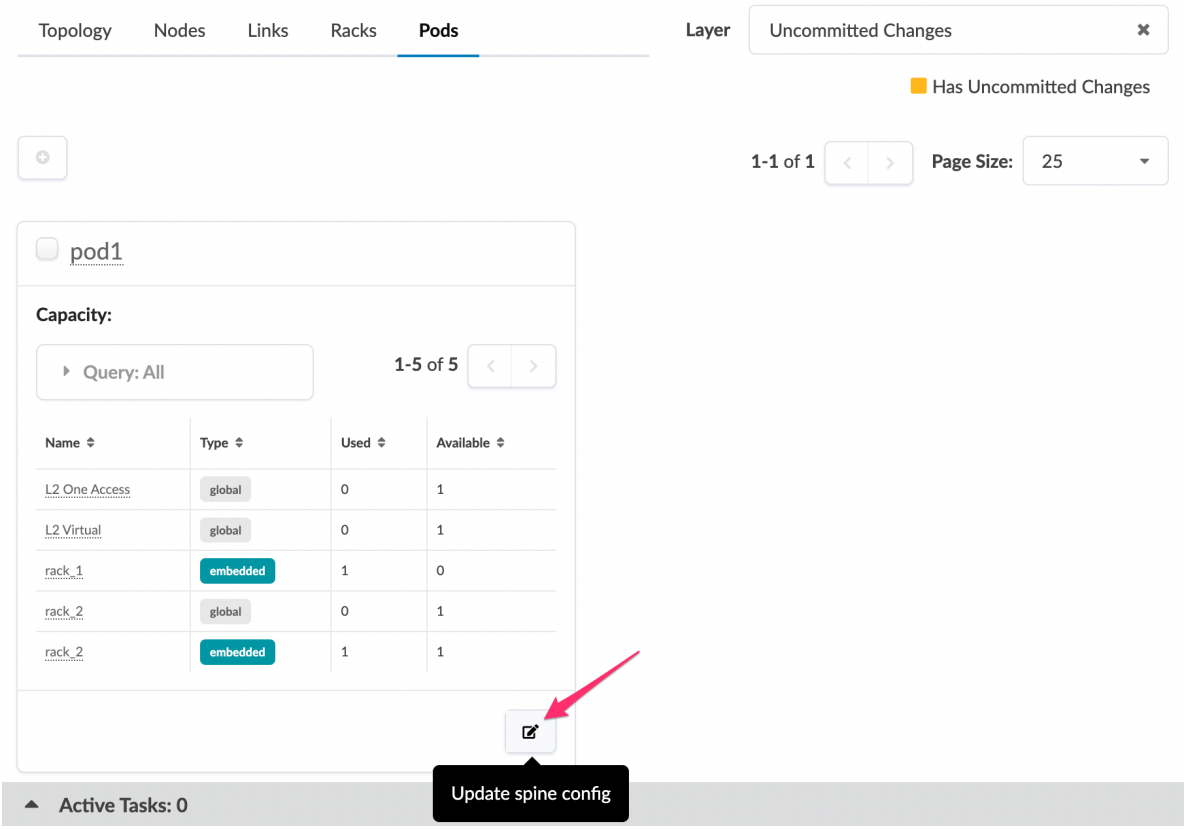
RELATED DOCUMENTATION

Commit / Revert Changes to Blueprint | 599

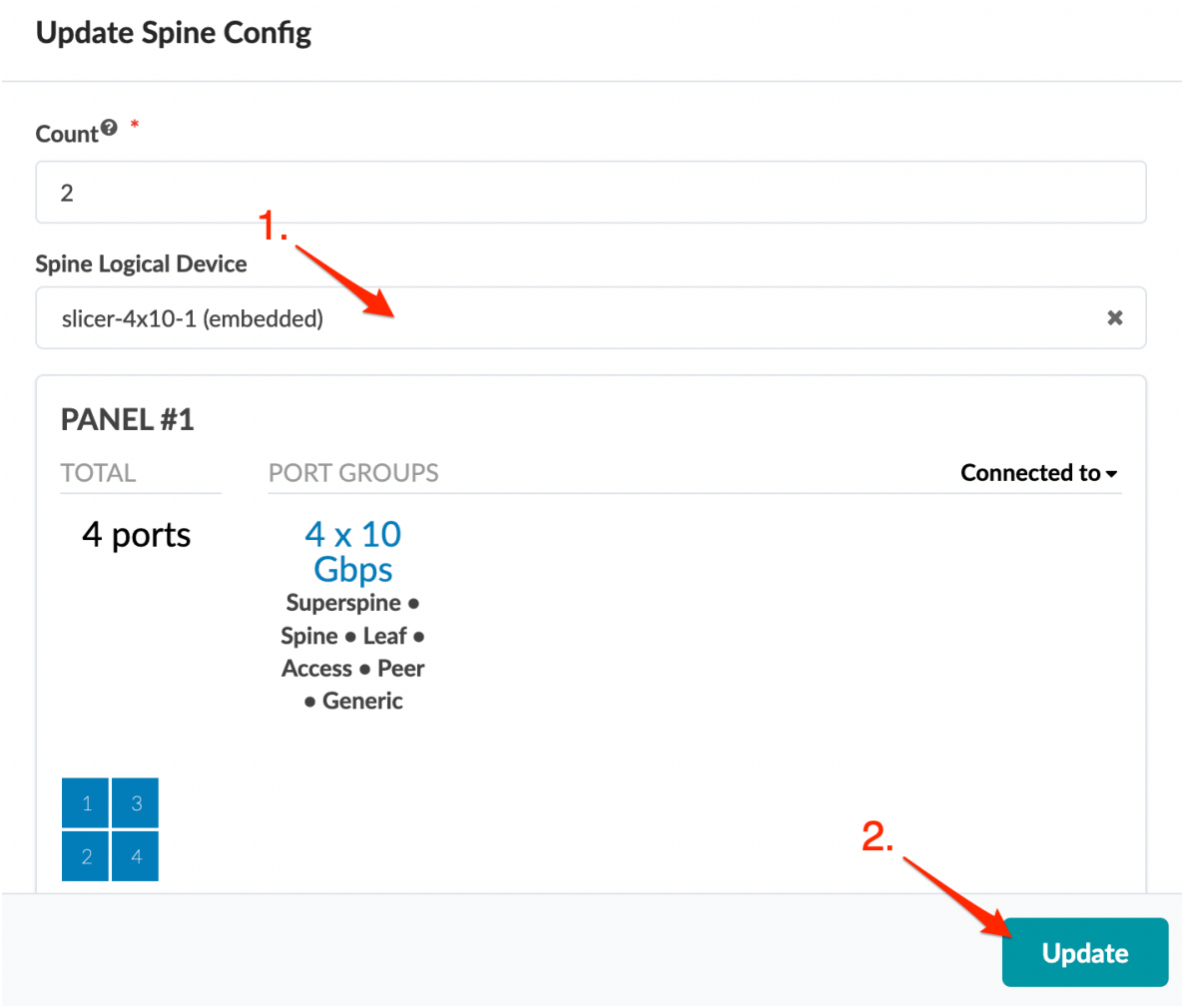
Change Spine Logical Device (Pod)

As a Day-2 operation, you can increase capabilities with a different spine logical device on both 3-stage and 5-stage blueprints. (On 5-stage topologies you can also "change the superspine logical device" on page 249.) Changes affect the entire pod, not just a node. Based on the change, this could be disruptive.

1. From the blueprint, navigate to **Staged > Physical > Pods**.
2. Click the **Update spine config** button on the bottom-right of the card for the pod to change.



3. From the **Spine Logical Device** drop-down list, select a different logical device.



4. Click **Update** to stage your changes and return to the **Pods** view.
   Build errors appear because interface maps need to be assigned.
5. Click the **Device Profiles** tab in the right panel and assign interface maps, as needed.

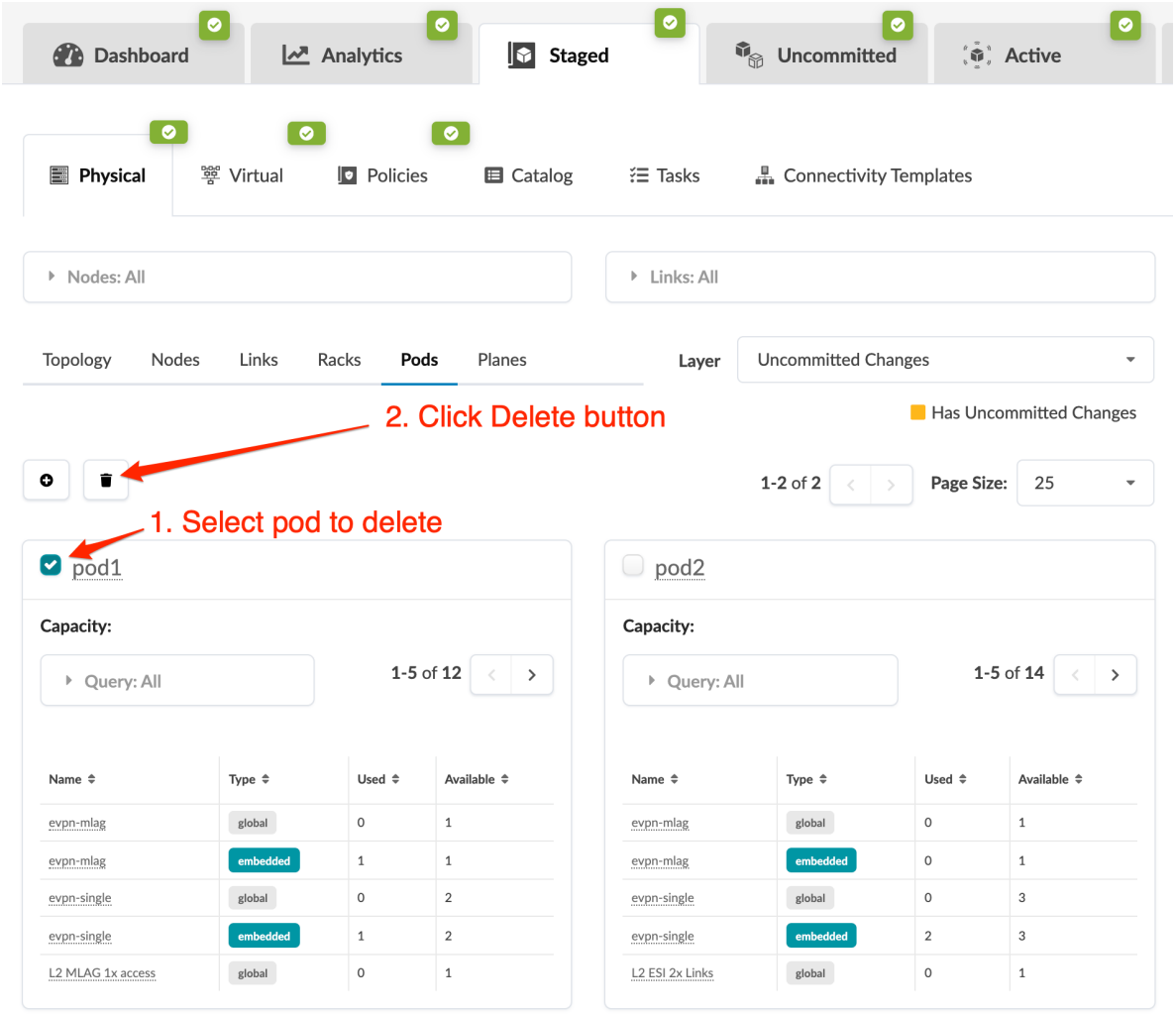
RELATED DOCUMENTATION

What are Logical Devices | 845

Delete Pod

When you delete a pod, all of its devices are removed from the blueprint; this could be highly impactful. Before deleting a pod that has live traffic on it, you may want to take its devices out-of-service by draining them. For more information, see the "Drain Device Traffic" on page 652 page.

1. From the blueprint, navigate to **Staged > Physical > Pods**.
2. Select the check box(es) for the pod(s) to delete. (You must keep at least one pod.)



3. Click the **Delete** button (trash can) for the pod(s) to delete.
4. Click **Delete Pod** to stage the deletion and return to the table view.
5. Commit the changes to your blueprint. Configuration on any running devices is erased and the devices are ready to be decommissioned.

RELATED DOCUMENTATION

Commit / Revert Changes to Blueprint | 599

Planes

IN THIS SECTION

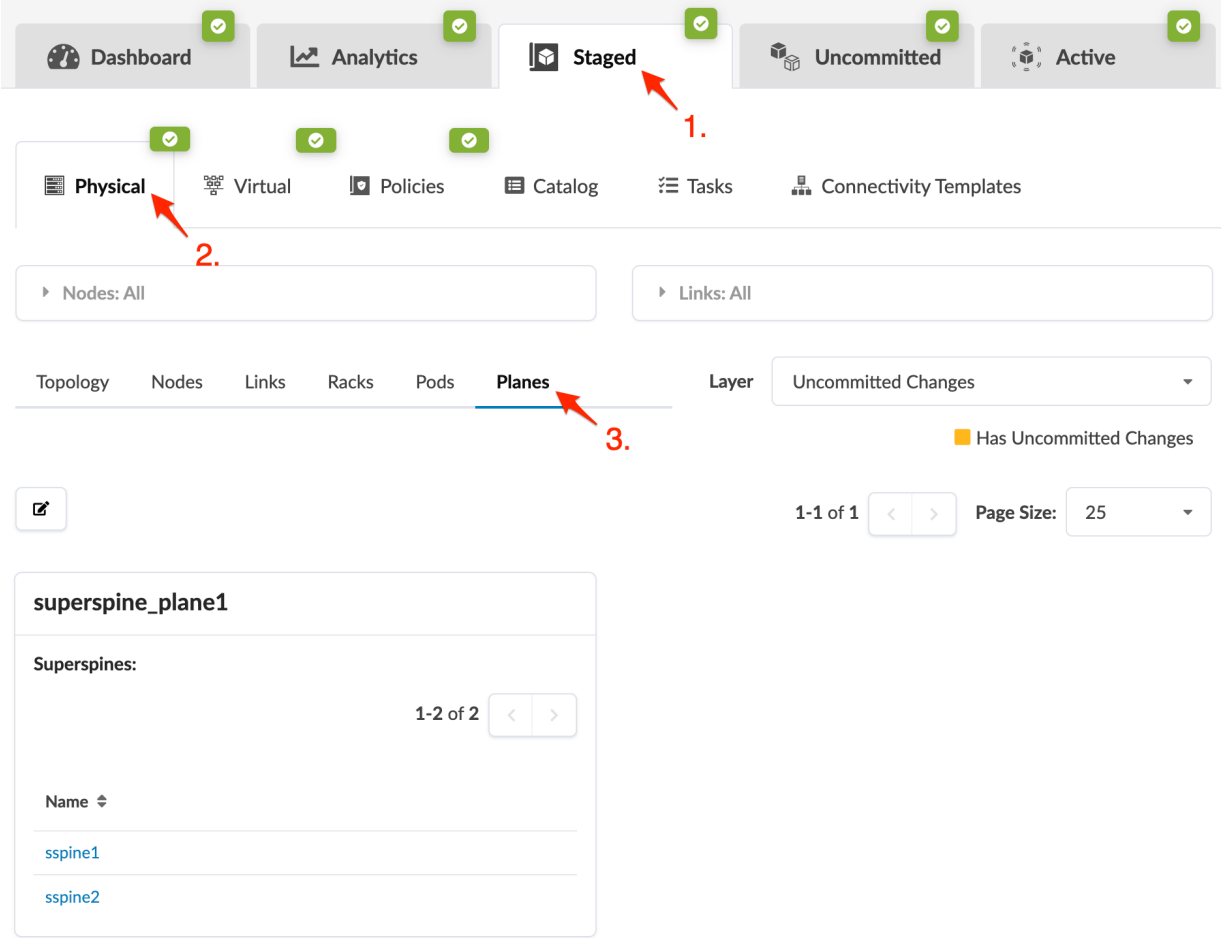
Planes (Datacenter)

Planes are groups of superspines in 5-stage blueprints. Every 5-stage topology has at least one plane.

As a Day 2 operation, you can add superspines to planes in 5-stage Clos networks. The maximum number of superspines is limited by the number of available spine ports of type **superspine**. When you add superspines, additional superspine nodes are created with the same logical devices that are used in the existing blueprint template. You must manually "assign the interface maps for the device profiles" on page 62 of each new node. When the devices are physically ready, you can "assign" on page 64 each node with their corresponding system IDs (serial numbers). When you commit pending changes, the superspines are configured and they become part of the control and data plane, taking part of forward traffic between pods.

You can also change the superspine logical device on planes to add or update superspine port capacity on 5-stage blueprints. This change is for *all* planes (not per plane) which, based on the change, could be disruptive. Changing the logical device requires that you specify a different interface map, and possibly a new device profile.

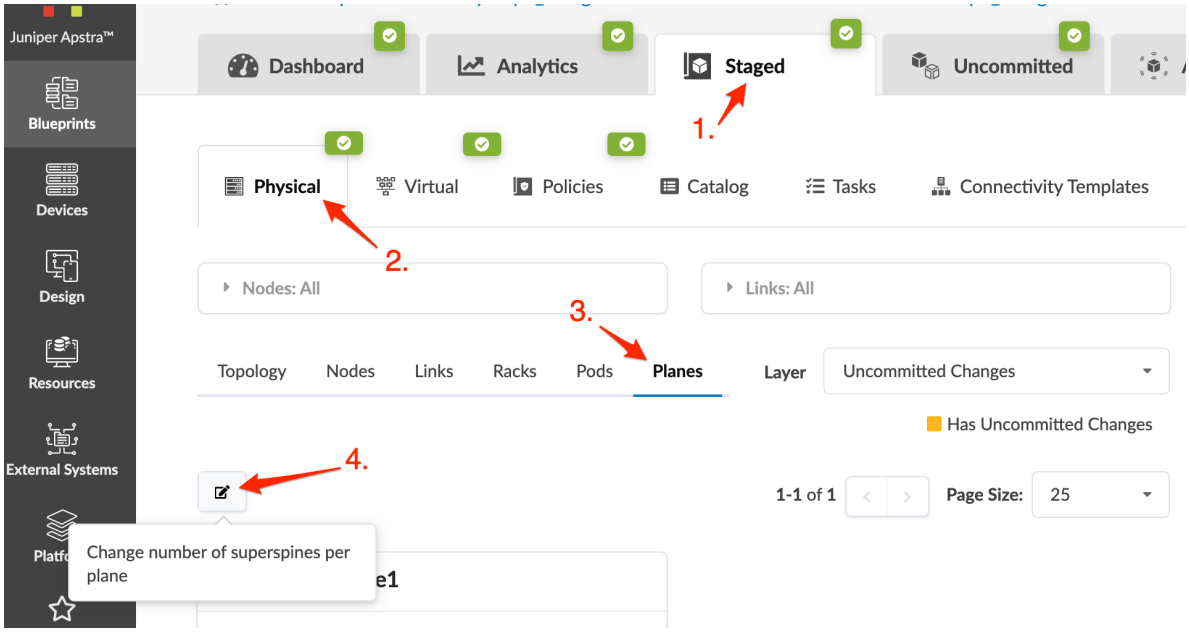
From the blueprint, navigate to **Staged > Physical > Planes** to go to the **Planes** view.



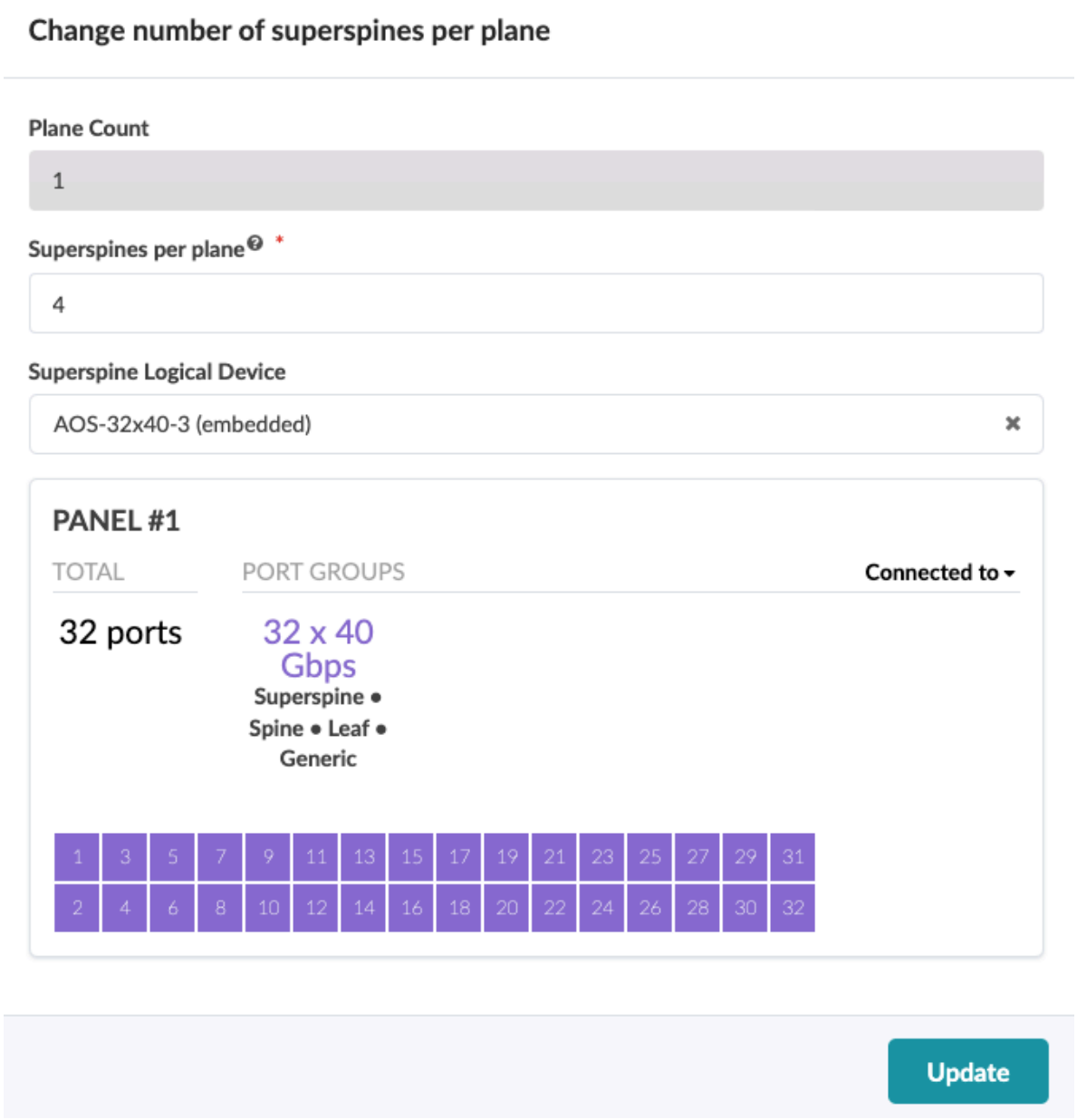
Add Superspine per Plane

As a Day 2 operation, you can add superspines per plane on 5-stage blueprints.

1. From the 5-stage blueprint, navigate to **Staged > Physical > Planes** and click the **Change number of superspines per plane** button.



2. In the **Superspines per plane** field, enter the total number of superspines you want. You can only add superspines per plane. Plan carefully. After you add superspines, you won't be able to remove them later.



3. Click **Update** to stage your changes and return to the **Planes** view.

When you're ready to activate changes, commit them from the Uncommitted tab.

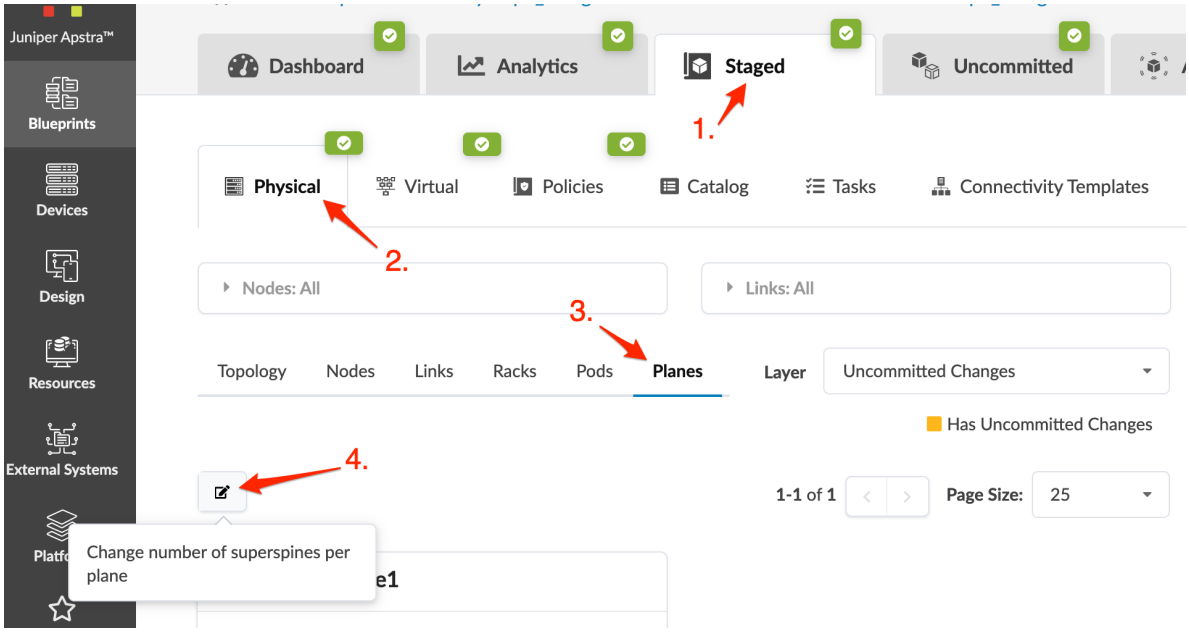
RELATED DOCUMENTATION

Commit / Revert Changes to Blueprint | 599

Change Superspine Logical Device (Plane)

As a Day 2 operation, you can change the superspine logical device on planes to add or update superspine ports capacity on 5-stage blueprints. This change is for *all* planes (not per plane) which, based on the change, could be disruptive. Changing the logical device requires that you specify a different interface map, and possibly a new device profile.

1. From the 5-stage blueprint, navigate to **Staged > Physical > Planes** and click the **Change number of superspines per plane** button.



2. Select a diferent logical device from the **Superspine Logical Device** drop-down list.
3. Click **Update** to stage your changes and return to the **Planes** view.
   Build errors appear because interface maps need to be assigned.
4. Click the **Device Profiles** tab in the right panel and assign interface maps, as needed.

RELATED DOCUMENTATION

What are Logical Devices | 845

Commit / Revert Changes to Blueprint | 599

CHAPTER 7

Virtual

IN THIS CHAPTER

Virtual Networks

IN THIS SECTION

What are Virtual Networks

You can create an overlay network in an Apstra blueprint by creating virtual networks (VN)s to group physically separate endpoints into logical groups. These collections of Layer 2 forwarding domains are either VLANs or VXLANs.

VLANs have the following characteristics:

- Single rack (rack-local)
- Single leaf devices or leaf pairs
- Can deploy in Layer 2-only mode (for example, isolated cluster networks for database replication)
- Can deploy with Layer 3 gateway (SVI) IP address on rack leaf, hosted with or without first-hop redundancy

VXLANs have the following characteristics:

- Fabric-wide for ubiquitous Layer 2 (inter-rack)
- Combination of single rack leaf devices or leaf pairs (MLAG)
- Can deploy in Layer 2-only mode
- Can deploy with Layer 3 gateway functionality
- The control plane selected (Pure IP Fabric or MP-EBGP EVPN) when configuring the template for your blueprint determines what is configured in the VN. (MP-EBGP EVPN provides a control plane for VXLAN routing.)
- VXLAN-EVPN capabilities for VXLAN VNs are dependent on network device makes and models. For more information see the `evpn_support_addendum:Apstra EVPN Support Addendum`.

For complete VN feature compatibility for supported Network Operating Systems (NOS), see the Apstra Feature Matrix for the applicable release (in the Reference section). For detailed capability information for a device, contact your network device vendor or "Juniper Support" on page 1374.

VNs contain the following details:

Table 1: Virtual Network Parameters

Name	Description
Name	30 characters or fewer. Underscore, dash, and alphanumeric characters only.

Table 1: Virtual Network Parameters *(Continued)*

Name	Description
Description	The way the description is rendered in a configuration depends on the NOS of the device: • **Junos** - Description is rendered under VLAN configuration (It's always present.) • **SONiC** - No description is rendered. • **NX-OS** - Description is rendered only on SVI, unless the IP address is disabled, then no description is rendered. • **EOS** - Description is rendered only on SVI, unless the IP address is disabled, then no description is rendered.
Type	• VLAN (rack-local VN) • VXLAN (EVPN) (inter-rack VN)
Tags	Tags
Routing Zone	• VLAN - default routing zone only (used for the underlay network) • VXLAN - default routing zone or user-defined routing zone
Tenant	

Table 1: Virtual Network Parameters *(Continued)*

Name	Description

Default VLAN ID (VLAN only)	• Layer 2 VLAN ID on the switch that the VN is assigned to.
	• If left blank, it's auto-assigned from static pool (2-4094).
	• If you assign it, we don't recommend assigning VLAN ID 1 for active VNs.
	• Cisco NX-OS reserves VLAN IDs 3968-4094.
	• Arista reserves 1006-4094 for internal VLANs for routed ports. You can modify "reserved" VLAN ID range with the EOS `vlan internal allocation policy` configuration command. You can apply it to all EOS devices using a **SYSTEM** configlet before configuring and deploying VNs.
	`l2-virtual-ext-002-leaf1(config)#vlan internal allocation policy ascending range 3001 3999`
	`l2-virtual-ext-002-leaf1(config)#exit`
	`l2-virtual-ext-002-leaf1#show vlan internal allocation policy`
	`Internal VLAN Allocation Policy: ascending`
	`Internal VLAN Allocation Range: 3001-3999`
	`l2-virtual-ext-002-leaf1#`
	• Using reserved VLAN IDs may cause deployment errors, but not build errors.

VNI(s) (VXLAN only)	Layer 2 VXLAN ID on the switch that the VN is assigned to. If left blank, it's auto-assigned from resource pools. Create up to 40 VNs at once by entering ranges or individual VNI IDs separated by commas (for example: 5555-5560, 7777). Commit the first 40 VNs before creating additional ones.
VLAN ID (on leaf devices)	VLAN ID
Reserve across blueprint (VXLAN only)	Option to use same VLAN ID on all leaf devices
DHCP server	Enabled/Disabled - DHCP relay forwarder configuration on SVI. Implies L3 routing on SVI
IPv4 Connectivity	Enabled/Disabled - for SVI routing

Table 1: Virtual Network Parameters *(Continued)*

Name	Description
IPv4 subnet (if connectivity is enabled)	• IPv4 subnet - (for example: 192.168.100.0/24) (can't use batching VLANs) • IPv4 CIDR length - automatically assigns a subnet with the specified length (for example: /26) • If left blank, it's auto-assigned a /24 subnet network from resource pools
Virtual Gateway IPv4	The IPv4 address, if enabled
IPv6 Connectivity	Enabled/Disabled - IPv6 connectivity for SVI routing. You must enable IPv6 in blueprint. If the template uses IPv4 spine-to-leaf link types, you can't use IPv6 in default routing zone and for VLAN type VNs.
IPv6 subnet (if connectivity is enabled)	• IPv6 subnet (for example: 2001:4de0::/64) • IPv6 CIDR length - automatically assigns a subnet with the specified length (for example: /56) • If left blank, it's auto-assigned a /64 subnet network from resource pools. • If assigned automatically, the IP is derived from the assigned VNs SVI pools. • To assign multiple VLAN networks, leave blank or specify CIDR length.
Virtual Gateway IPv6	The IPv6 address, if enabled
Create connectivity templates for	• Tagged • Untagged
L3 MTU	Default value is from Virtual Network Policy. You can update the value here for these specific virtual networks.
Assigned to	The racks that the VN is assigned to. For more information, see table below.

Table 2: Virtual Network Rack (or Pod) Details

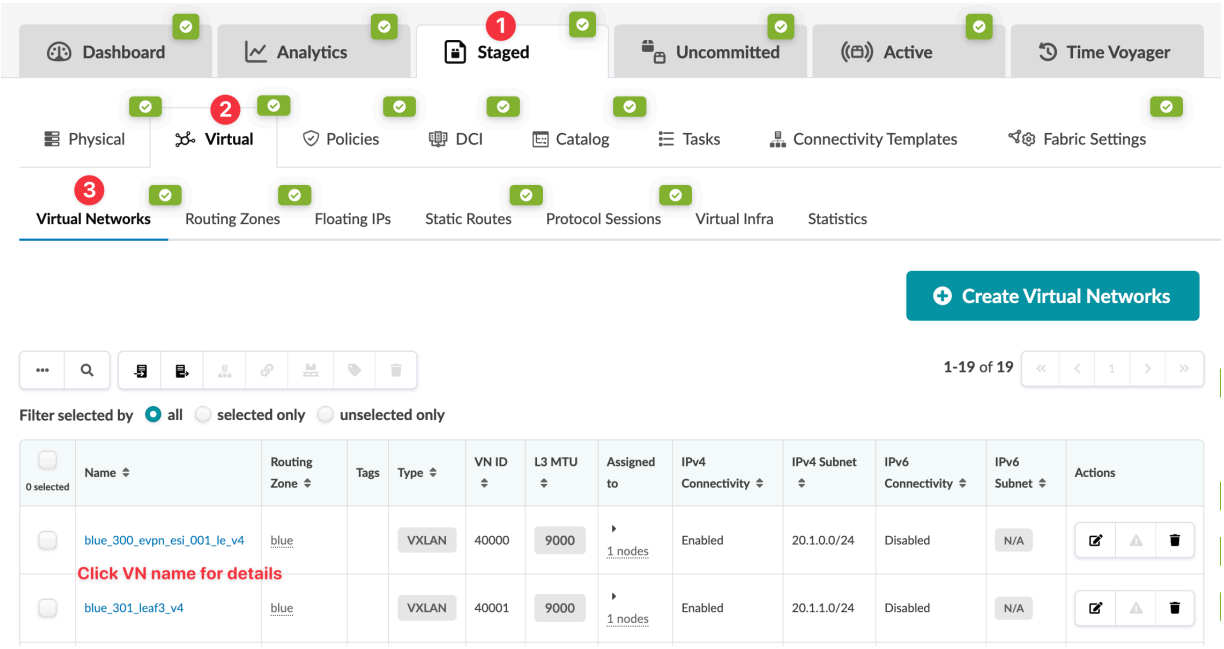
Assigned To Details	Description
Pod Name (5-stage)	5-stage Clos networks include pods, and you can select leaf devices within each pod to extend VNs to those devices.
Bound to	The racks assigned. For MLAG racks, the leaf pair is shown. For VLANs, if more than one rack is selected, multiple rack-local VLAN-based VNs are created.
Tags	Leverage system tags for filtering when you create virtual networks. This helps speed-up the definition of a virtual network footprint in large-scale deployment. It nicely complements the tag-driven interface assignment at the connectivity template level.
Link Labels	Label assigned to rack (for example, ext-link-1, single-link, single-link, ext-link-0)
VLAN ID	Can use for batch creating VNs

Secondary IP Allocation Mode	• **Enabled** (default) - Apstra decides whether a secondary IP address is needed.
	  • Automatically allocate if an assigned connectivity template requires an address (BGP, Static routes).
	  • VN of type VXLAN:
	  • Some NOS types automatically allocate unicast IPv4 addresses when an anycast IPv4 gateway is present: (Junos when in an ESI pair).
	  • If a NOS type forbids co-existence of an anycast IPv4 address with an unicast IPv4 address, a blueprint error will be raised (Sonic).
	  • VN of type VLAN - All NOS types require unicast IPv4 addresses when the IPv4 anycast address is enabled.
	• **Forced** - A secondary IP address is rendered irrespective of whether or not a connectivity template requires it.
	  • If a NOS type forbids co-existence of an anycast IPv4 address with a unicast IPv4 address, a blueprint error will be raised.
	  • Permits you to manually create an optional unicast IPv4 address for purposes such as BGP peering or static routing.

Table 2: Virtual Network Rack (or Pod) Details *(Continued)*

Assigned To Details	Description
IPv4 Address / IPv6 Address	You can set the first-hop-redundancy IP address for the SVI (VRRP, VARP and so on). If left blank, the SVI IP address is assigned from the selected pool. When you bind an EVPN connectivity template to a Layer 2 application point, the SVI IP address is used as the source / destination for the BGP session, static routes and so on.

From the blueprint, navigate to **Staged > Virtual > Virtual Networks** to go to the **Virtual Networks** table view.



To go to the details of a virtual network, click its name in the table.

Parameters

Name	blue_300_evpn_esi_001_le_v4
Description	blue_300_evpn_esi_001_le_v4
Type	VXLAN
Tags	
Routing Zone	blue
VNI	40000
DHCP Service	Enabled
L3 MTU	9000
IPv4 Connectivity	Enabled
IPv4 Subnet	20.1.0.0/24
Virtual Gateway IPv4	20.1.0.1
Route Target	40000:1

Assigned To

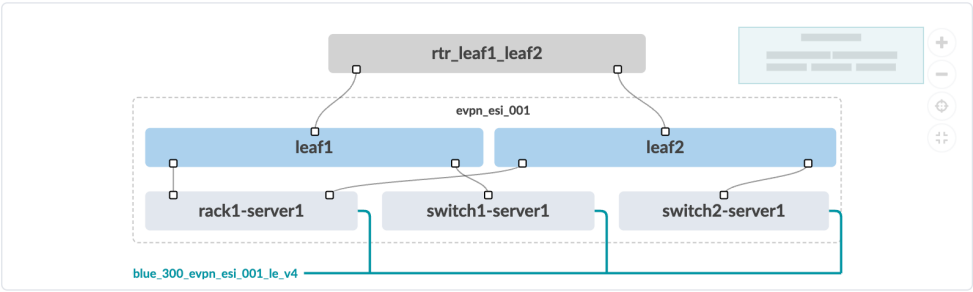
1-1 of 1

Name	VLAN ID	Secondary IP Allocation Mode	IPv4 Address
evpn_esi_001_leaf_pair1	300	leaf1 enabled leaf2 enabled	leaf1 Not assigned leaf2 Not assigned

Endpoints

Ports View | Links View

Expand Nodes? Expand External Generics? Show Links?



Port Maps:

Ports: Unassigned Untagged VLAN Tagged Partial

1 of 1 Page 1

Size: 5

You can create, edit, import, export, tag, and delete virtual networks.

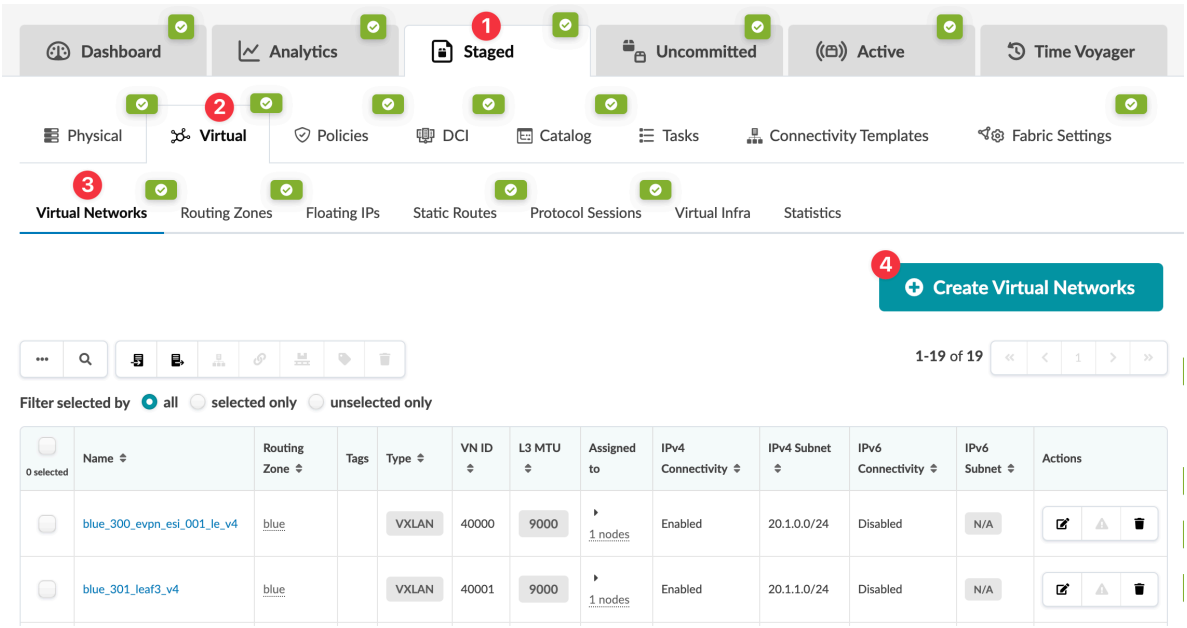
Create Virtual Network

IN THIS SECTION

Create Virtual Networks (using GUI)

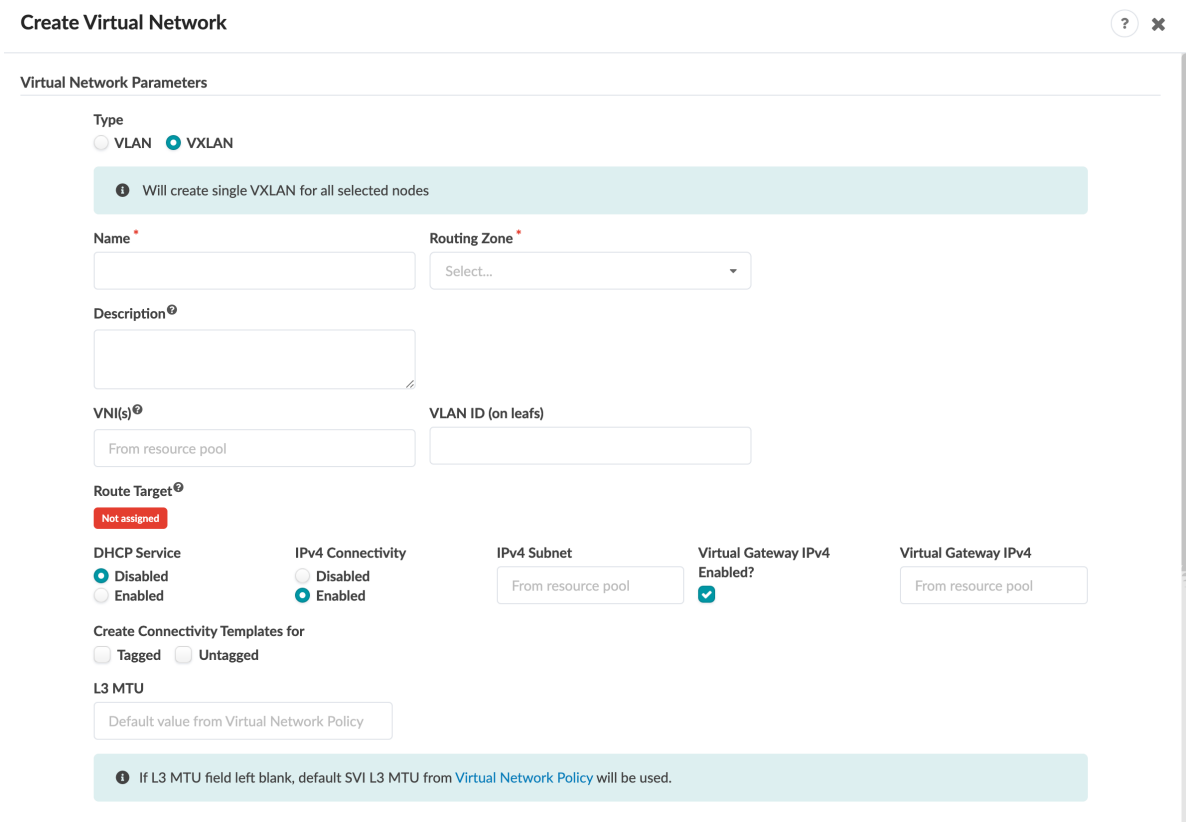
If you'll be assigning your virtual network(s) to a routing zone other than the default one, make sure you've "created" on page 258 them before proceeding.

1. From the blueprint, navigate to **Staged > Virtual > Virtual Networks** and click **Create Virtual Networks**.



   The **Creating Virtual Network** dialog opens.

2. Enter **Virtual Networks Parameters** details as described below. (For more details about each parameter, see "What are Virtual Networks" on page 251.)



a. Select the type of virtual network(s) to create (VLAN, VXLAN), enter a unique name, and (optional) description. (Description is new in Apstra version 5.0.0.) The way the description is rendered in a configuration depends on the NOS of the device:

   - **Junos** - Description is rendered under VLAN configuration (It's always present.)
   - **SONiC** - No description is rendered.
   - **NX-OS** - Description is rendered only on SVI, unless the IP address is disabled, then no description is rendered.
   - **EOS** - Description is rendered only on SVI, unless the IP address is disabled, then no description is rendered.

b. Select the "routing zone" on page 282 from the drop-down list to associate with the virtual network(s). (If the routing zone that you need is not in the list, you probably still need to "create" on page 258 it.) (VLANs must use the default routing zone.)

c. If you're creating VXLANs, specify VNIs. If you're creating VLANs, specify VLAN ID(s). You have the option of leaving this field blank to allow auto-assignment from a resource pool.

d. If you're creating VXLANs and you enter a **VLAN ID (on leaf devices)**, you can select the check box (that appears after entering a value) to **Reserve across blueprint**. This enforces the same rule

across the fabric and helps you to honor the same VLAN policy across racks when adding new racks.

e. The **Route Target** field represents the built-in Apstra Route-target for the L2 VNI. When using EVPN DCI Gateway features, this route-target must be included in L2 VNI import and export route-targets for EVPN fabrics outside of the Apstra blueprint. You can't modify the build-in route-target. If this value displays **Not Assigned**, it means a VNI must be associated with the virtual network.

f. If you enable **DHCP Service**, enter a subnet. A DHCP relay forwarder is configured on the SVI. This option also implies Layer 3 routing on this SVI. (You assign the DHCP server in the routing zone.)

g. If you enable **IPv4 Connectivity**, enter a subnet, unless you're batch-creating virtual networks. If so, enter an IPv4 CIDR length, or leave subnet blank to allow auto-assignment.

h. If you enable **Virtual Gateway IPv4**, enter an IPv4 address.

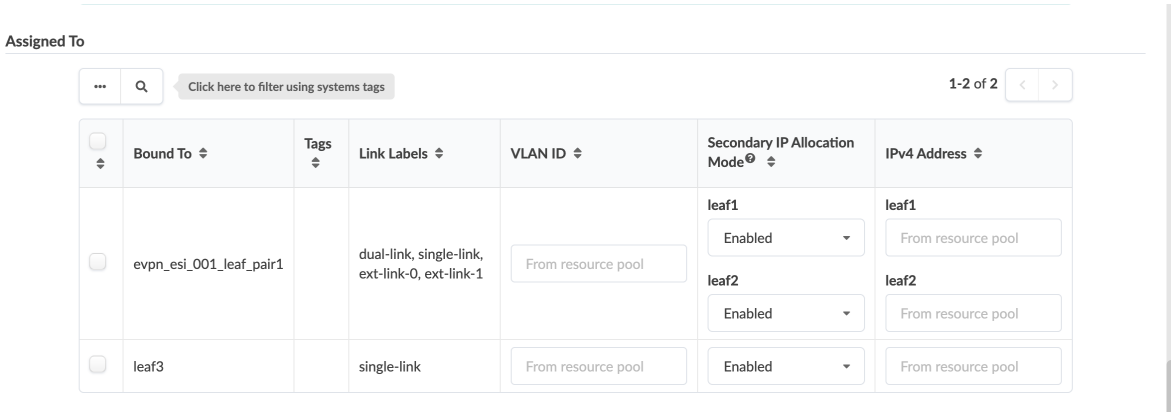
i. If IPv6 is enabled in the blueprint (Policies > Fabric Addressing Policy), and you enable **IPv6 Connectivity**, enter a subnet, unless you're batch- creating VNs. If so, enter an IPv6 CIDR length, or leave subnet blank to allow auto-assignment.

j. If you enable **Virtual Gateway IPv6**, enter an IPv6 address.

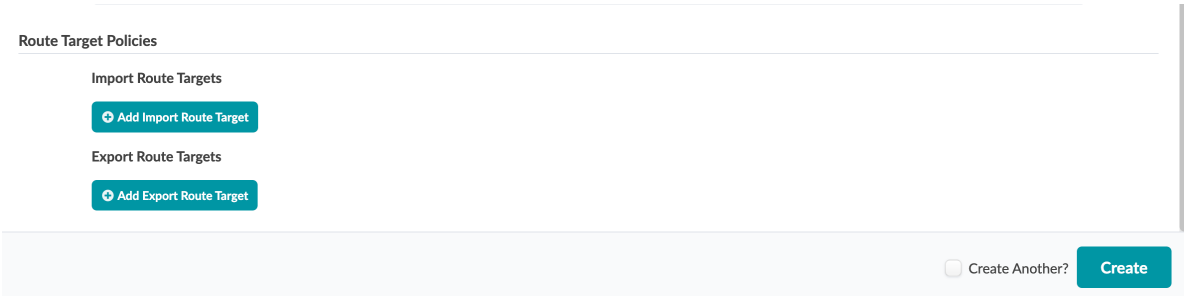
k. To create connectivity templates for the VN(s), check the box for **Tagged** and/or **Untagged**, as applicable.

l. To override the default MTU value, enter a value for **L3 MTU**.

3. In the **Assigned To** section of the dialog, select the applicable rack(s) to assign the virtual networks to and configure them. As of Apstra version 5.0.0, you can leverage system tags for filtering when you create virtual networks. This helps speed-up the definition of a virtual network footprint in large-scale deployment. It nicely complements the tag-driven interface assignment at the connectivity template level. As of Apstra version 5.0.0, you can skip virtual network assignment during creation and assign them later.



4. In the **Route Target Policies** section of the dialog, add import and export route targets, as applicable



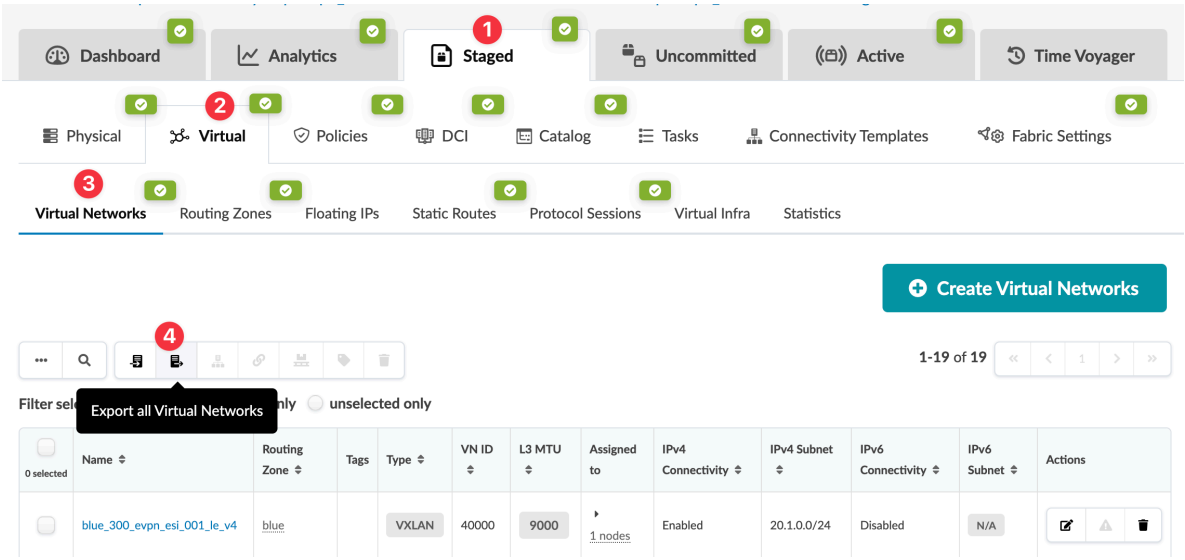
5. Click **Create** to stage the new virtual network(s) and return to the **Virtual Networks** table view.
6. Assign IPv4 (IPv6) resources for SVI subnets. Navigate to **Staged > Virtual > Virtual Networks** and "assign resources" on page 58 in the **Build** panel (right-side).
7. For VXLAN only: Assign VTEP IPs. Navigate to **Staged > Virtual > Virtual Networks** and assign resources in the **Build** panel (right-side). (You can display the VTEPs list in the nodes table (Staged > Physical > Nodes). Select the type of VTEP to display from the **Columns** drop-down list (above the table).)
   - **Single Leaf Nodes** require one VTEP IP and an anycast VTEP IP for all switches in the VN.
   - **MLAG Leaf-pair Nodes** require a common VTEP IP for the leaf-pair and an anycast VTEP IP for all switches in the VN.

When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

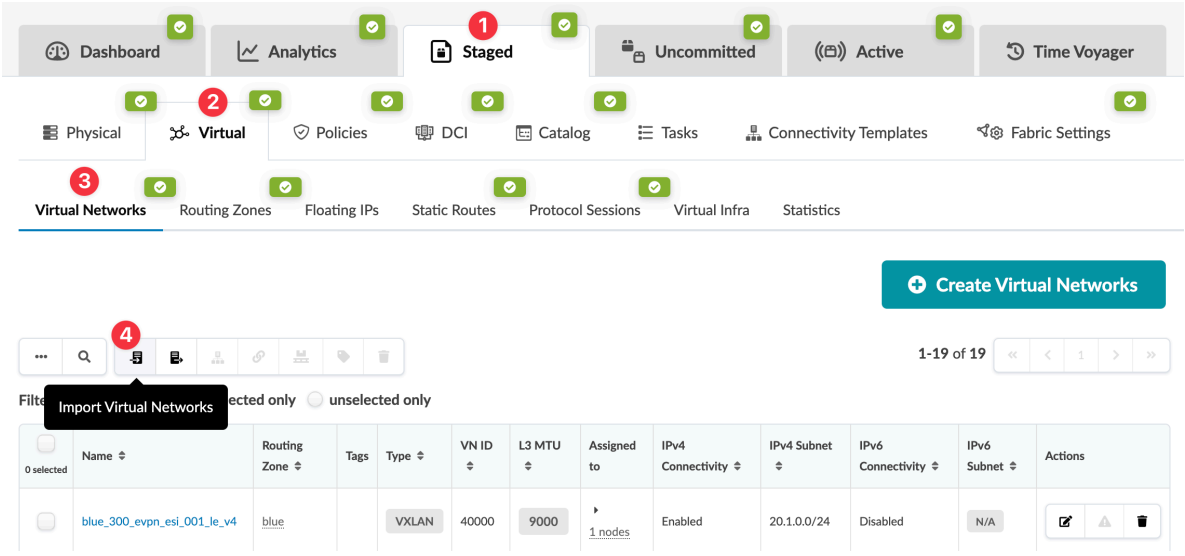
Create Virtual Networks (using CSV File)

You can create many virtual networks at once with a CSV file. First, you'll "export" on page 266 the virtual network schema from your blueprint, then open and populate the file in a spreadsheet program. And finally, you'll import the file back into your blueprint.

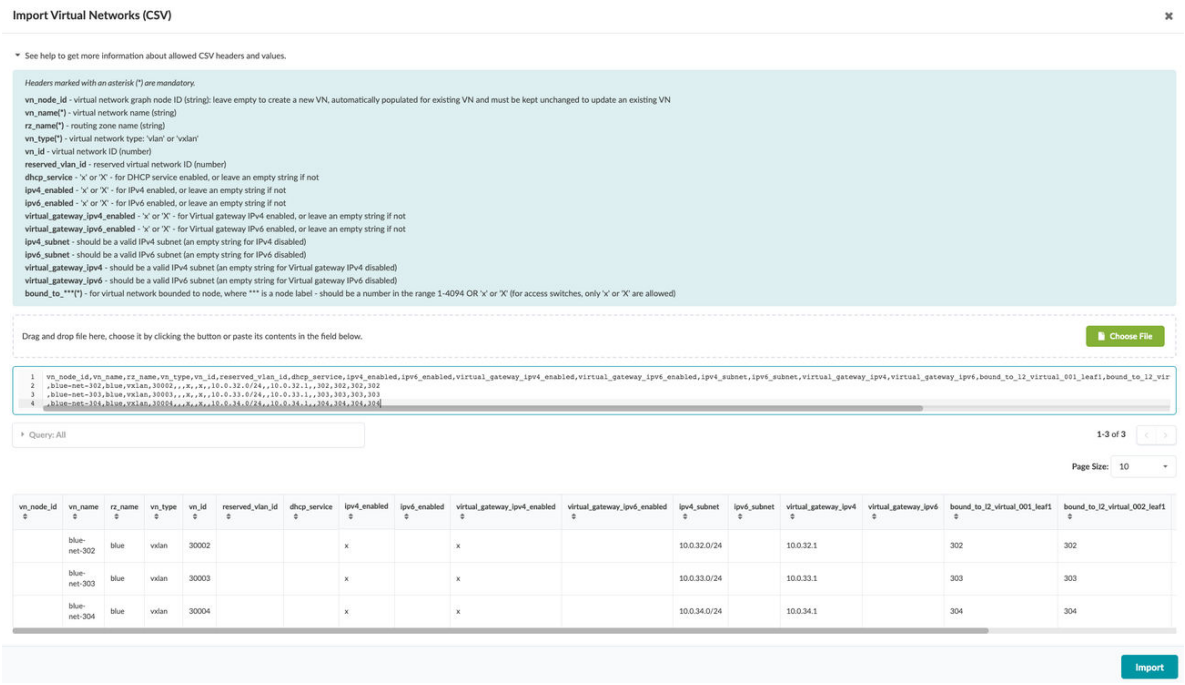
1. From the blueprint, navigate to **Staged > Virtual > Virtual Networks** and click **Export all virtual networks**.



2. Click **Copy** to copy the contents or click **Save As File** to download the file. (Close the dialog to return to the table view.)
3. Paste the contents, or open the CSV file, in a spreadsheet program (such as Google Sheets or Microsoft Excel). (Any virtual networks that were previously created are included in the file.)
4. Enter virtual networks details into the spreadsheet leaving the `vn_node_id` field blank for new VNs, then save the file.
5. In the Apstra GUI, from the blueprint, navigate to **Staged > Virtual > Virtual Networks** and click **Import virtual networks**.



6. Either click **Choose File** and navigate to the file on your computer, drag and drop the file onto the dialog window, or as shown in the screenshot below, directly paste CSV file contents. Virtual network details are displayed for your review.



7. Click **Import** to import the virtual networks, stage the changes, and return to the **Virtual Networks** table view.
8. Assign IPv4 (IPv6) resources for SVI subnets. Navigate to **Staged > Virtual > Virtual Networks** and "assign resources" on page 58 in the **Build** panel (right-side).
9. For VXLAN only: Assign VTEP IPs. Navigate to **Staged > Virtual > Virtual Networks** and assign resources in the **Build** panel (right-side). (You can display the VTEPs list in the nodes table (Staged > Physical > Nodes). Select the type of VTEP to display from the **Columns** drop-down list (above the table).)
   - **Single Leaf Nodes** require one VTEP IP and an anycast VTEP IP for all switches in the VN.
   - **MLAG Leaf-pair Nodes** require a common VTEP IP for the leaf-pair and an anycast VTEP IP for all switches in the VN.

When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

Update Virtual Network Resource Assignments

SUMMARY

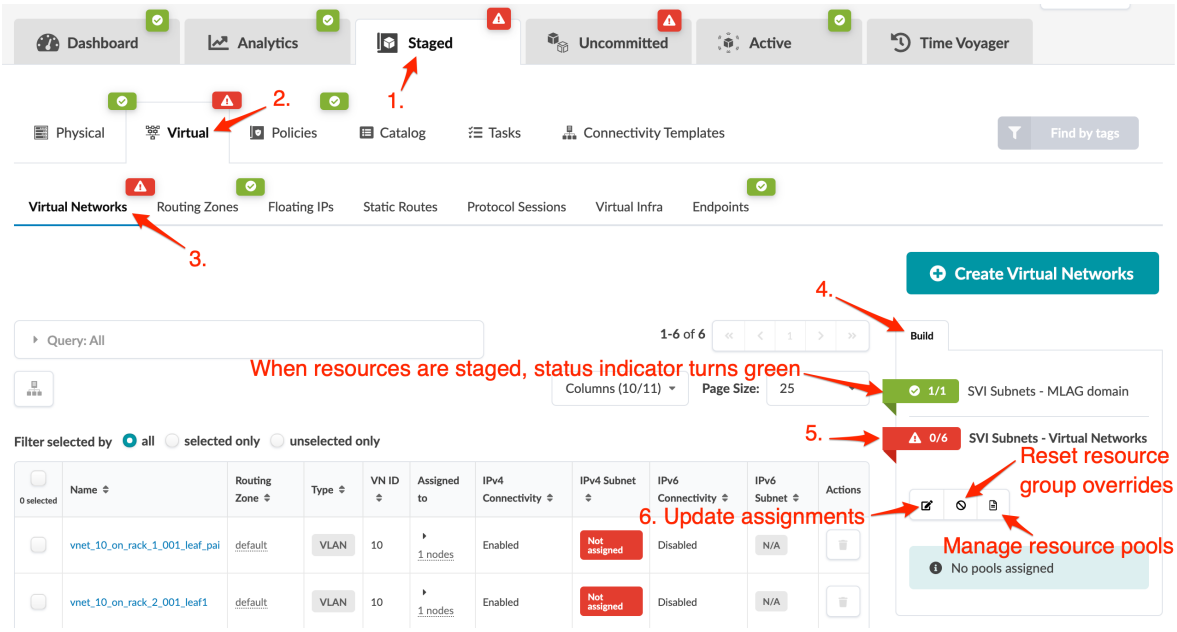
IN THIS SECTION

Assign Virtual Network Resources from Resource Pools

You can assign resources, release previously used resources and go to resource pool management from the virtual build panel. The resource assignment section has a convenient shortcut button, **Manage resource pools**, that takes you to resource pool management. From there, you can monitor resource usage and create additional resource pools, as needed.

A red status indicator in the build panel means that resources need to be assigned. Resources may include virtual network SVI subnets for routing zones, SVI subnets for MLAG domain, SVI subnet for virtual networks, VNI Virtual Network IDs, and VTEP IPs.

1. From the blueprint, navigate to **Staged > Virtual > Virtual Networks > Build**. (The build panel is on the right side.)



2. Red status indicators mean that resources need to be assigned. Click a red status indicator, then click the **Update assignments** button.
3. Select a pool from which to pull the resources, then click the **Save** button. The required number of resources are automatically assigned to the resource group. When the red status indicator turns green, the resource assignment has been successfully staged.

When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

Assign Specific Virtual Network Resource

When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

Reset Virtual Network Resource Group Override

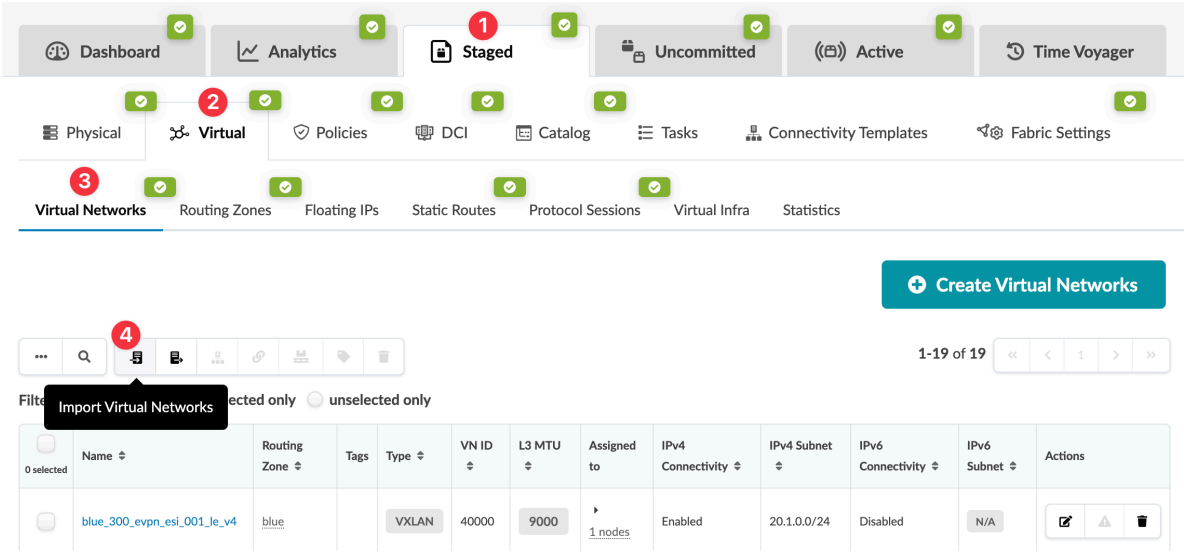
Certain blueprint operations require resource allocations to be retained even when a device has been removed from a blueprint. For example, if you decide to reuse a device, previously allocated resources need to be re-used as well. If resources were not retained, build errors may occur because the expected resources would no longer be available to the device. To minimize build errors, resource allocations persist by default. If you know that a device won't be re-instated, you don't need to keep its resources allocated to it. Click the **Reset resource group overrides** button to reset the resource group and release the resources.

a.

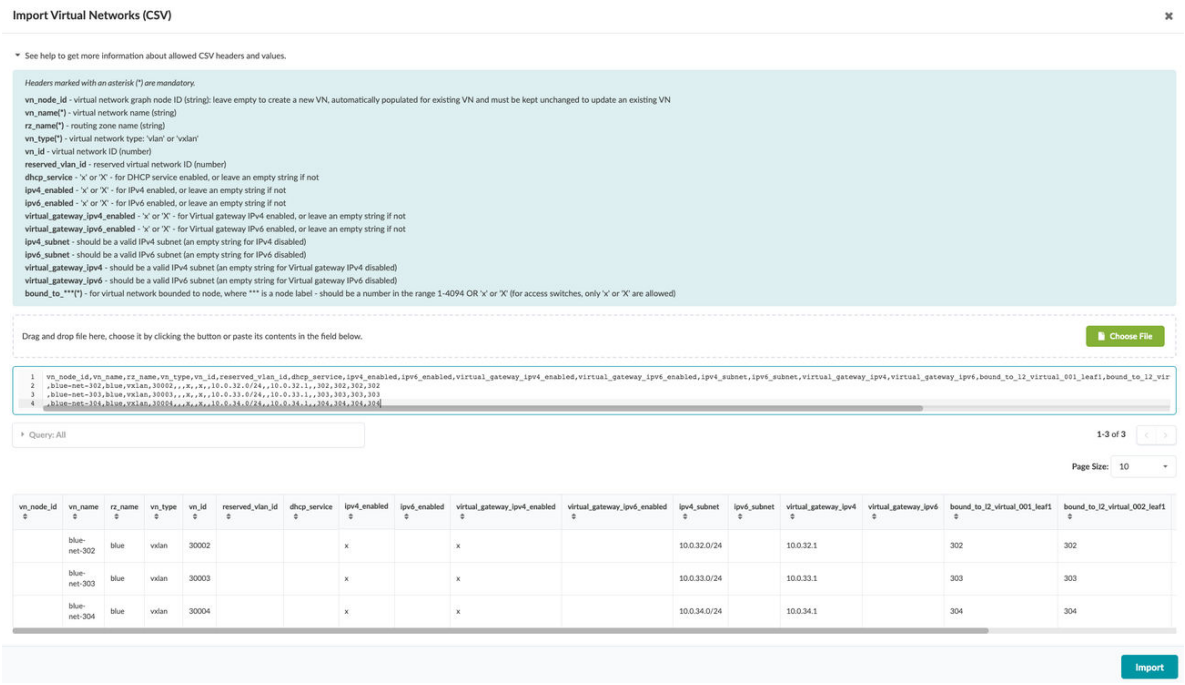
Import Virtual Network

You can import multiple virtual networks (as a CSV file) into your blueprint. (Tip: First "export virtual networks" on page 266 so you'll have the schema set up for you in the CSV file.)

1. From the blueprint, navigate to **Staged > Virtual > Virtual Networks** and click the **Import virtual networks** button.



2. Either click **Choose File** and navigate to the file on your computer, drag and drop the file onto the dialog window, or as shown in the screenshot below, directly paste CSV file contents. Virtual network details are displayed for your review.



3. Click **Import** to import the virtual networks, stage the changes, and return to the table view.

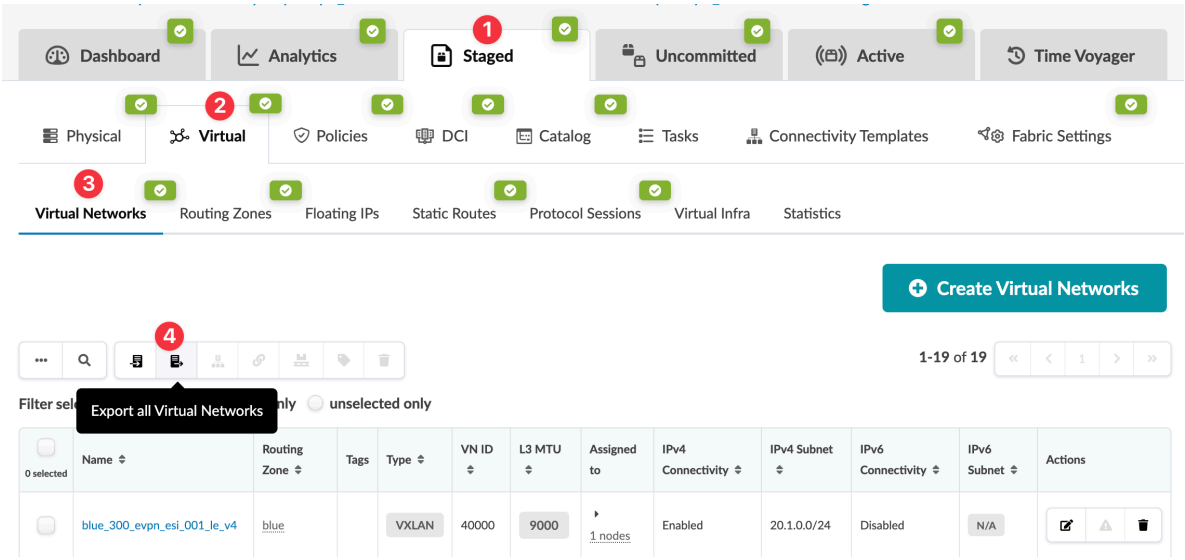
When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

Export Virtual Network to CSV File

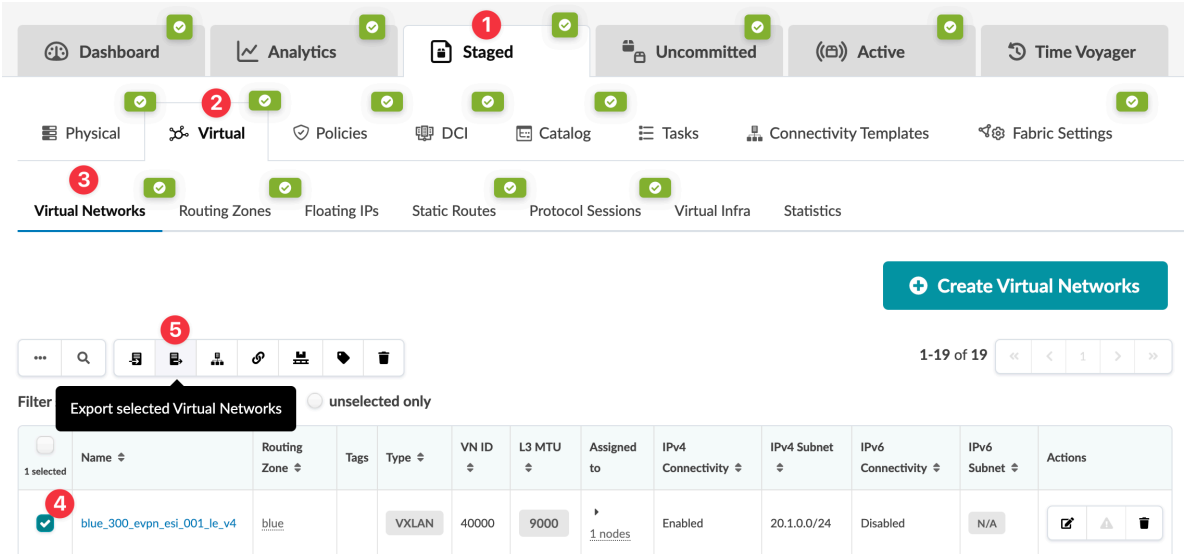
SUMMARY

You can update many virtual networks quickly by exporting them as a CSV file, updating the file, then importing the file back into the blueprint.

1. From the blueprint, navigate to **Staged > Virtual > Virtual Networks**.



2. To export all virtual networks, click the **Export all virtual networks** button as shown in the screenshot above.
3. Or to export specific virtual networks instead of all of them, select their check boxes, then click the same button as in the previous step (now called **Export selected virtual networks**)



4. Click **Copy** to copy the contents or click **Save As File** to download the file.
5. When you've copied or downloaded the virtual networks, close the dialog to return to the **Virtual Networks** table view.

Next Steps: Update the CSV file (with a spreadsheet program), then "import" on page 265 it back into your blueprint.

Update Virtual Network Assignments

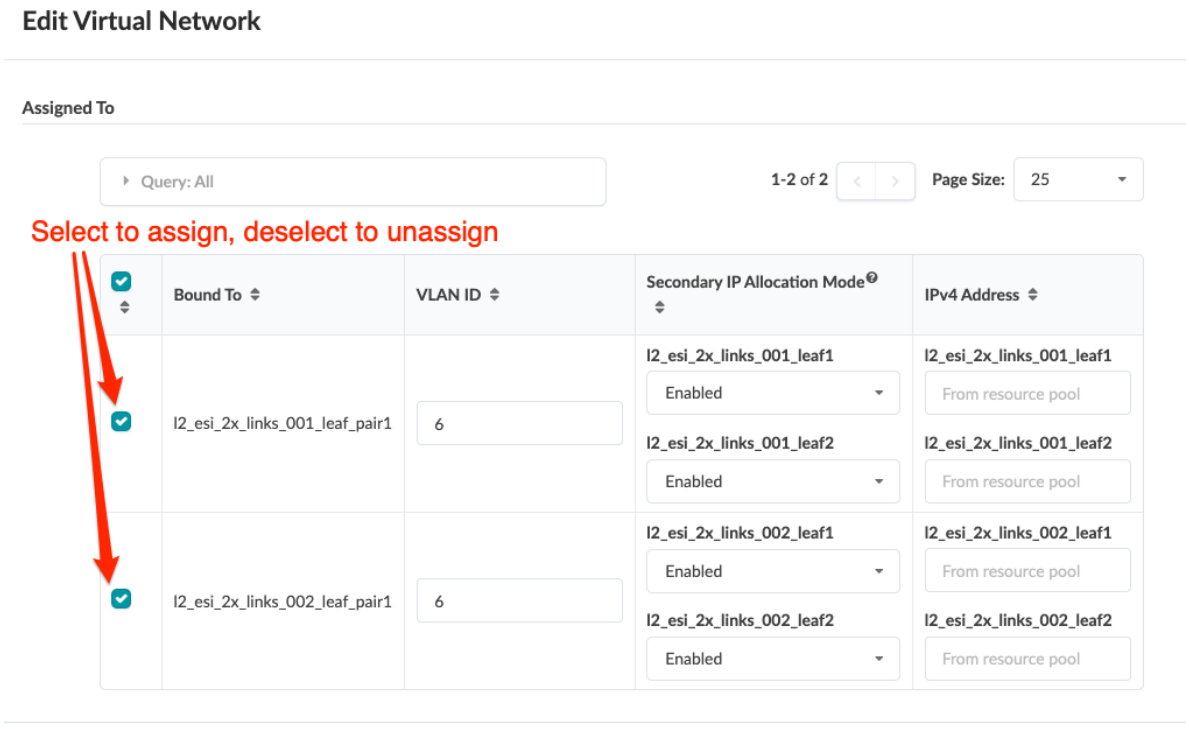
IN THIS SECTION

You can assign (and unassign) multiple VXLAN virtual networks at the same time from the Apstra GUI.

Assign / Unassign One Virtual Network

When you create a virtual network, you assign it to one or more nodes. You can edit the VN to assign it to additional nodes and/or to unassign it from nodes that it's already assigned to.

1. Either from the table view (Staged > Virtual > Virtual Networks) or the details view, click the **Edit** button for the virtual network to update.
2. In the dialog that opens, scroll past the **Virtual Network Parameters** section to the **Assigned To** section:
   - Assign the VN to one or more nodes by selecting the applicable node check box(es).
   - Unassign the VN from one or more nodes by deselecting the applicable node check box(es).

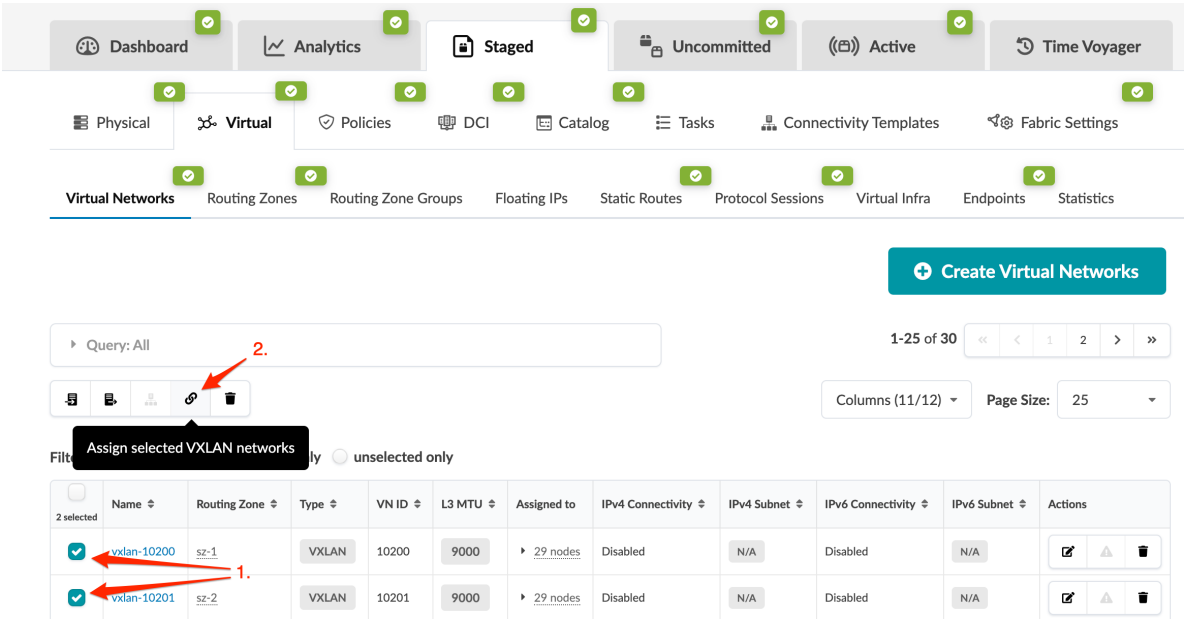


3. Click **Update** to stage the changes and return to the table view.

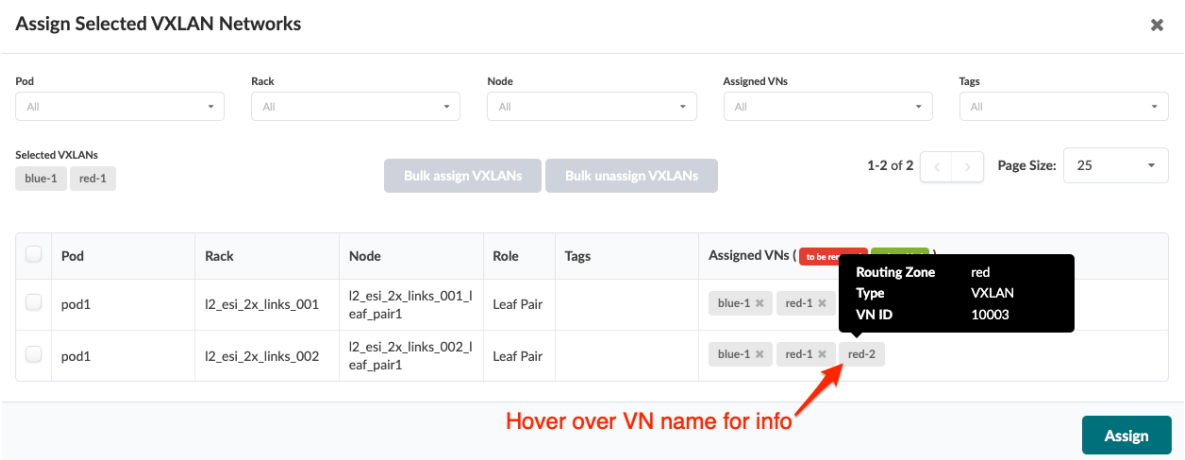
Assign / Unassign Multiple Virtual Networks

You can assign/unassign many virtual networks at the same time. This is especially useful when you've added a rack as a Day 2 operation and you need to assign a lot of virtual networks to it.

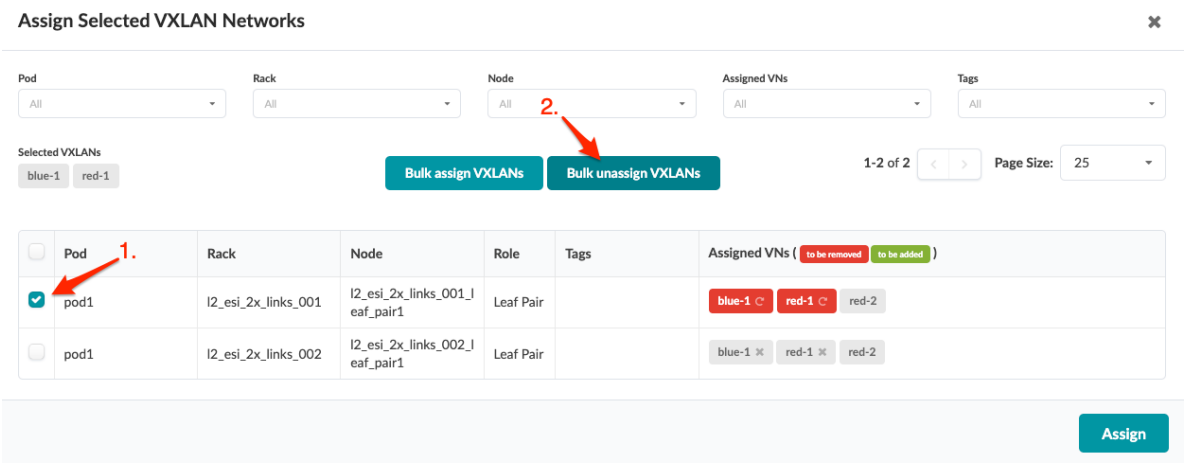
1. From the table view (Staged > Virtual > Virtual Networks) select one or more check boxes for the VNs to update.
2. Click the **Assign selected VXLAN networks** button that becomes available above the table (fourth of five buttons).



3. In the dialog that opens, you can see the associated routing zone, VN type and VN ID by hovering over the VNs that are already assigned.



4. Your selected VXLANs appear above the table on the left. The table shows the VNs that are already assigned to nodes in the network. Select the check boxes for one or more nodes. The **Bulk assign VXLANs** and **Bulk unassign VXLANs** buttons become available.



5. Assign and unassign virtual networks, as needed:
   - To assign your selected VXLANs to the nodes you just selected, click the **Bulk assign VXLANs** button. The VNs to be assigned turn green.
   - To unassign your selected VXLANs that are already assigned to the nodes you just selected, click the **Bulk unassign VXLANs** button. The VNs to be unassigned turn red (as shown in the screenshot example above).
6. Click **Assign** to stage your changes and return to the table view.

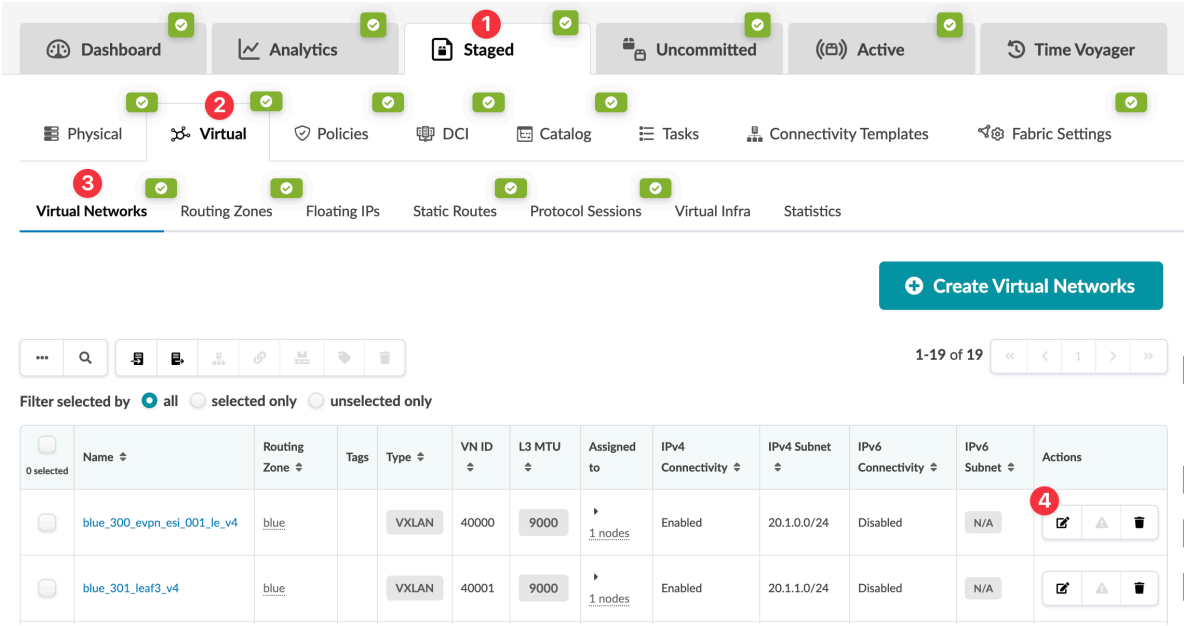
Move Virtual Network to Different Routing Zone

IN THIS SECTION

You can move one or more virtual networks from one routing zone to another (as of Apstra version 5.0.0). This is traffic-impacting for any endpoints that are part of the selected virtual networks (since the VNs need to be deleted from one RZ and recreated in the other one).

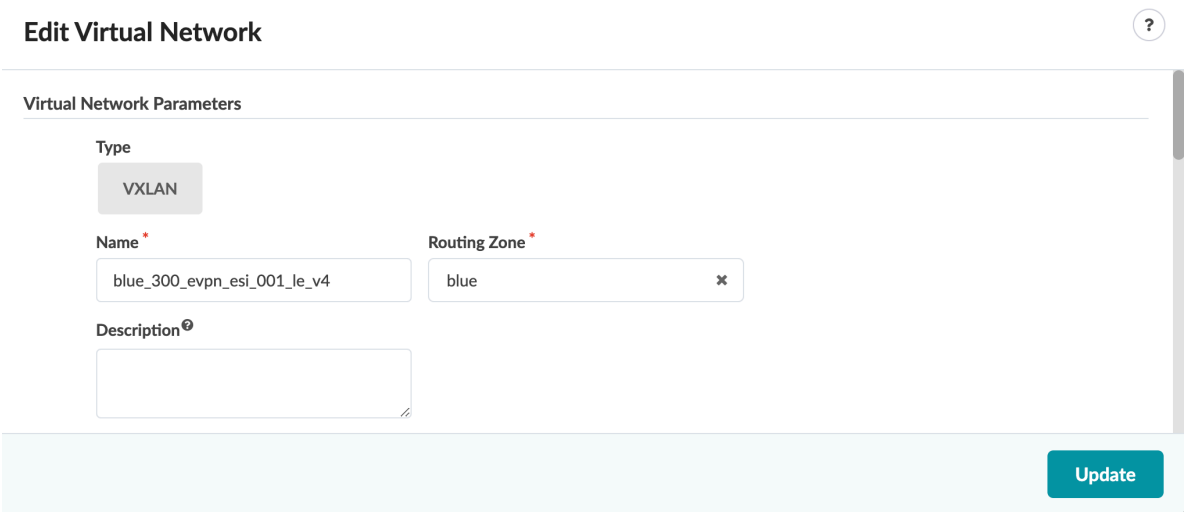
Move One Virtual Network to a Different Routing Zone

1. From the blueprint, navigate to **Staged > Virtual > Virtual Networks** and click the **Edit** button in the **Actions** panel for the virtual network to move.



The **Edit Virtual Network** dialog opens.

2. From the **Routing Zone** drop-down list, select the routing zone that you want to move the virtual network to. (The virtual network subnet IP, gateway IP (if the VN is IP-enabled) and VLAN ID of the source virtual network will be preserved, so these need to "free of use" in the destination routing zone.)

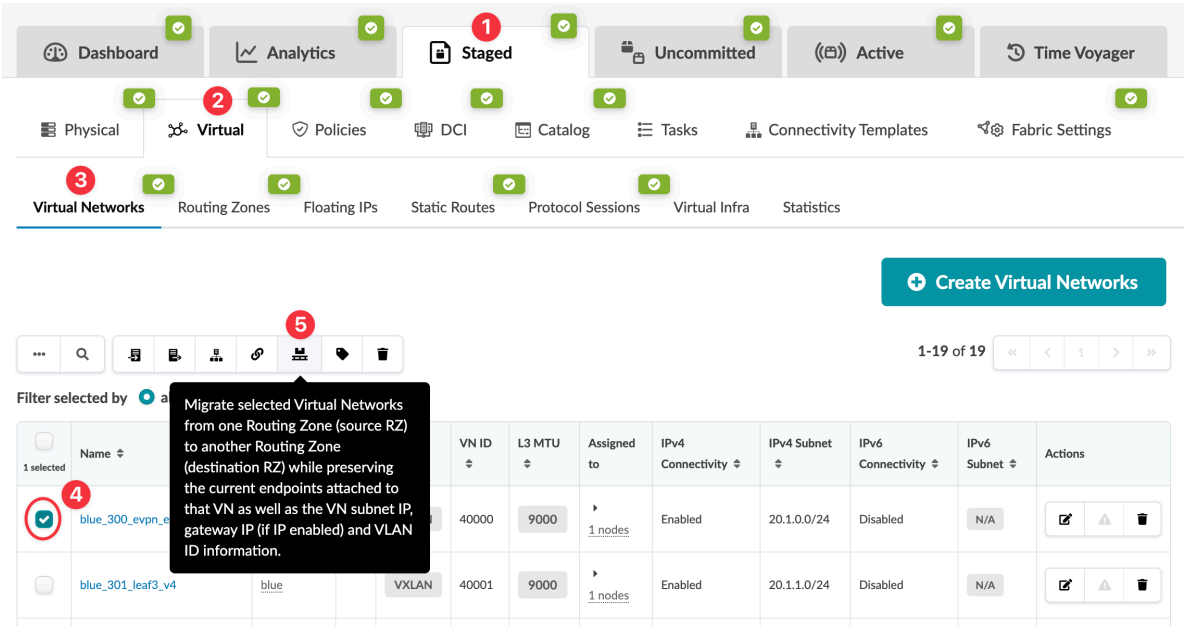


3. Click **Update** to stage the change and return to the **Virtual Networks** table view.

When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

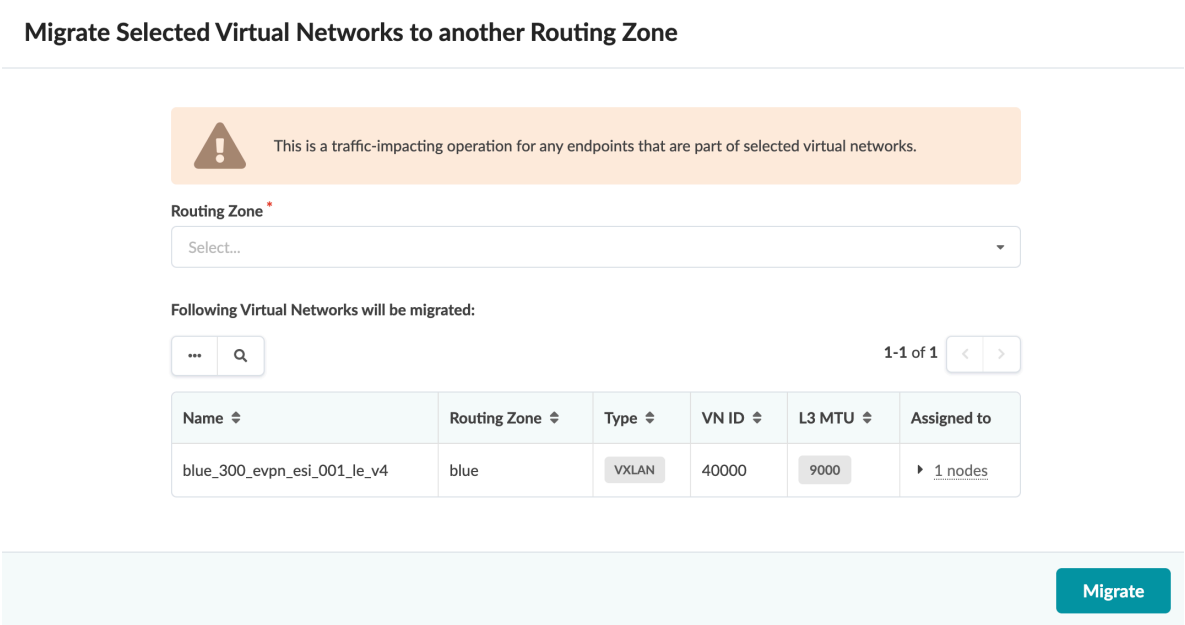
Move Multiple Virtual Networks to a Different Routing Zone

1. From the blueprint, navigate to **Staged > Virtual > Virtual Networks**, select the check boxes for one or more virtual networks to move, then click the **Migrate selected Virtual Networks...** button that becomes available above the table.



   The **Migrate Selected Virtual Networks to another Routing Zone** dialog opens.

2. From the **Routing Zone** drop-down list, select a different routing zone. (The virtual network subnet IP, gateway IP (if the VN is IP-enabled) and VLAN ID of the source virtual networks will be preserved, so these need to "free of use" in the destination routing zone.)



3. Click **Migrate** to stage the change and return to the **Virtual Networks** table view.

When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

Update Virtual Network Tags

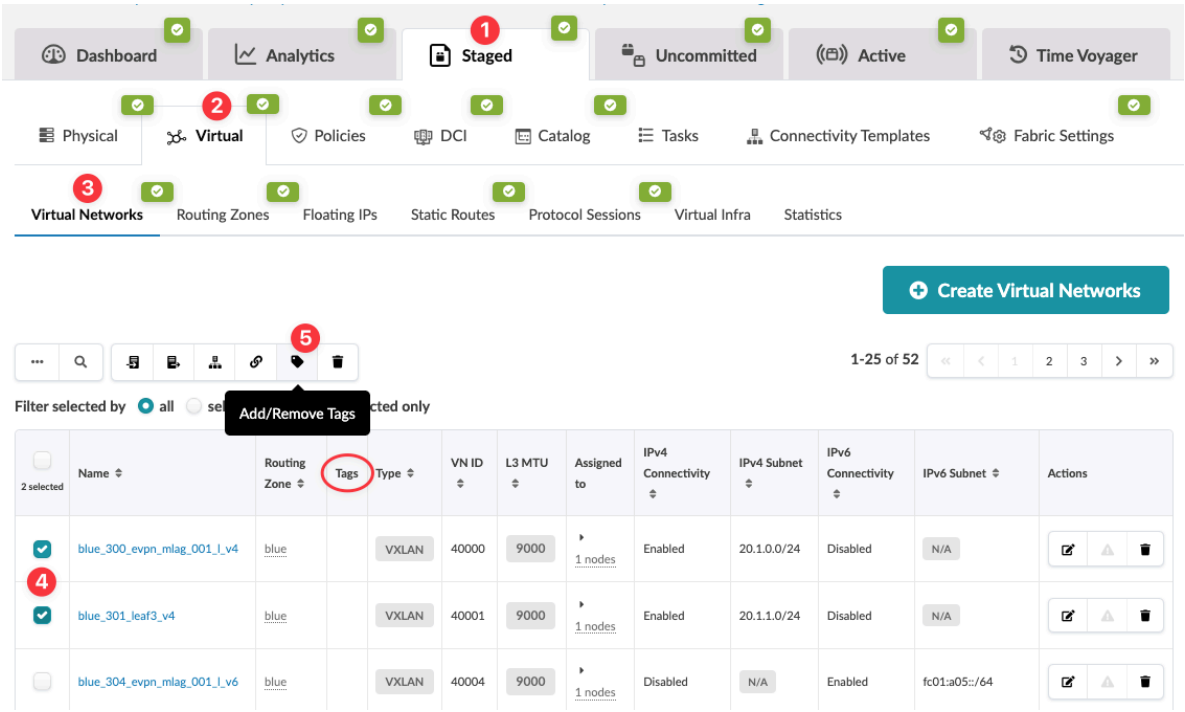
You can group and search virtual networks based on tags that you apply to them (new in Apstra version 5.0.0). Using tags on virtual networks can also enhance building dynamic configlets for virtual networks-based requirements (for example, virtual network QoS or determining what virtual networks could be available for route leaking).

If you'd like to apply a tag from the **Design** (global) catalog (Design > Tags), "import" on page 454 it to the **Blueprint** catalog beforehand.

You can also "create" on page 453 a tag directly in the **Blueprint** catalog before you add it to a virtual network, which also makes it available to the other taggable elements in the blueprint (nodes, links, interfaces, port channels, routing zones, connectivity templates).

Finally, you can create a tag on the fly as you're adding it to a virtual network and it will automatically be added to the catalog as well.

1. From the blueprint, navigate to **Staged > Virtual > Virtual Networks** and select one or more check boxes for the virtual networks(s) that need the same tag updates. The **Add/Remove Tags** button above the table becomes available.



2. Click the **Add/Remove Tags** button, then add and/or remove tags, as needed.

The **Add Tags** drop-down list includes all tags that are in the catalog (Staged > Catalog > Tags). Enter the tag name, and if it's not in the catalog you'll get the option to add it; it will be applied to the selected virtual network and added to the catalog.

The **Remove Tags** drop-down list includes all tags that are applied to the virtual network(s). Select tag(s) from the list to remove them. (If no tags are applied, the drop-down list isn't shown in the dialog.)

3. Click **Add/Remove Tags** to stage the change and return to the **Virtual Networks** table.

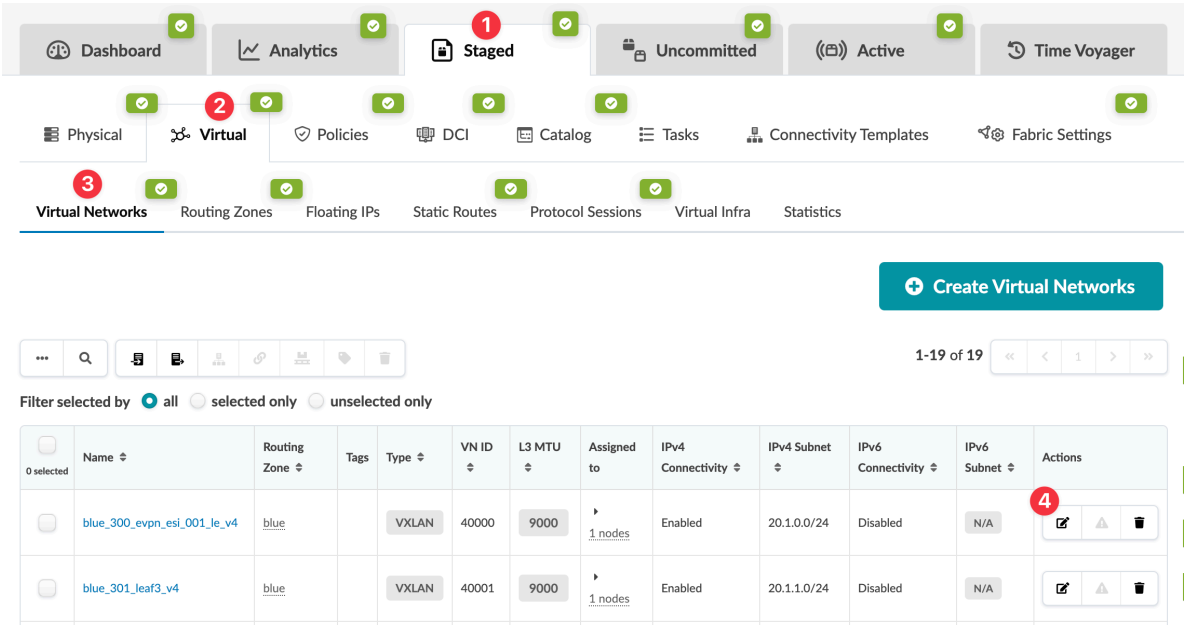
   If you have **Tags** selected in **Table Settings** (above the table, leftmost button with 3 dots), you'll see them in the **Tags** column in the table.

When you need to find something based on tags, you can search virtual networks with the **Search** button above the table, or for a wider search, use the **Search** field at the top of the **Staged** tab or the **Find by tags** button in the upper-right.

Change Virtual Network Description

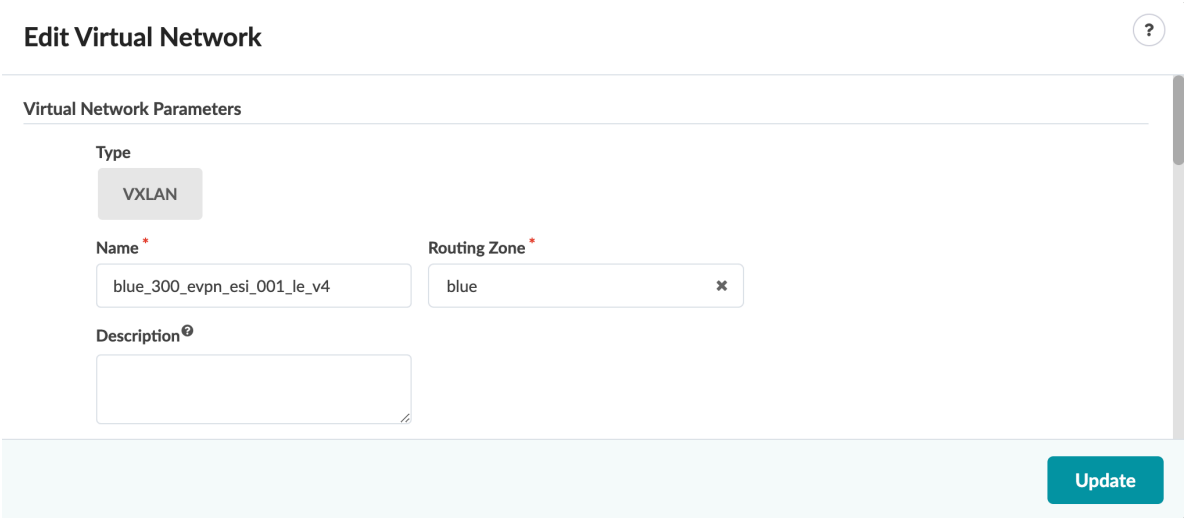
You can add and change a description on a virtual network (as of Apstra version 5.0.0).

1. From the blueprint, navigate to **Staged > Virtual > Virtual Networks** and click the **Edit** button in the **Actions** panel for the virtual network to update.



   The **Edit Virtual Network** dialog opens.

2. Change (or add) the description. (You can change various other parameter details from this dialog as well.)



3. Click **Update** to stage your changes and return to the **Virtual Networks** table view.

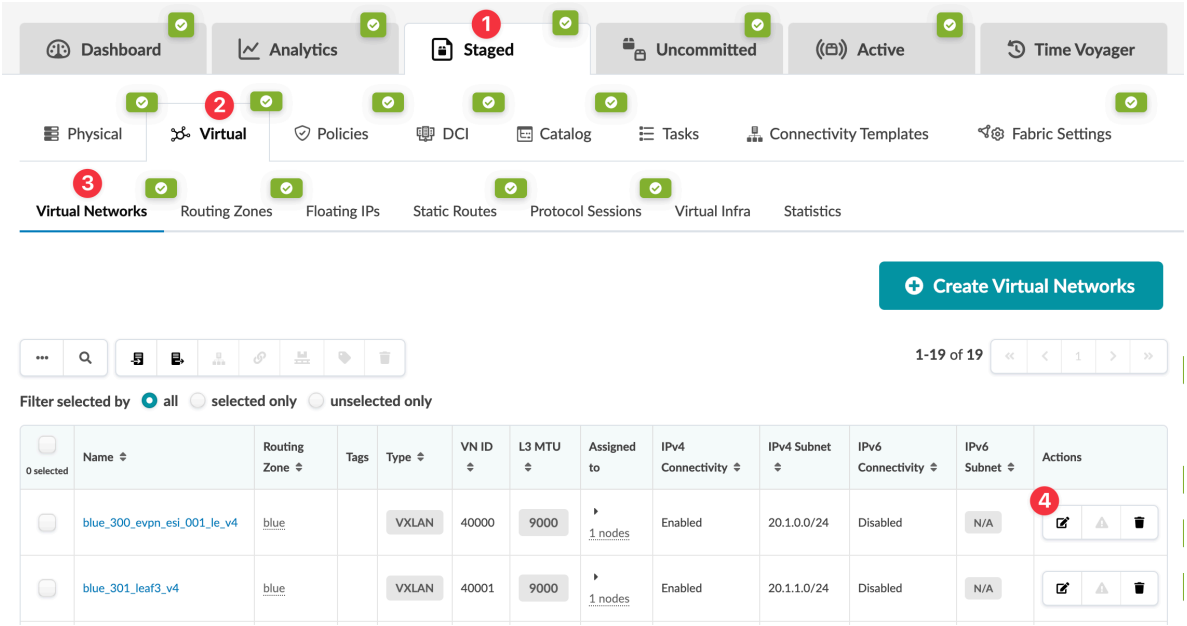
When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

Change Virtual Network Details

IN THIS SECTION

Edit One Virtual Network

1. From the blueprint, navigate to **Staged > Virtual > Virtual Networks** and click the **Edit** button in the **Actions** panel for the virtual network to edit.



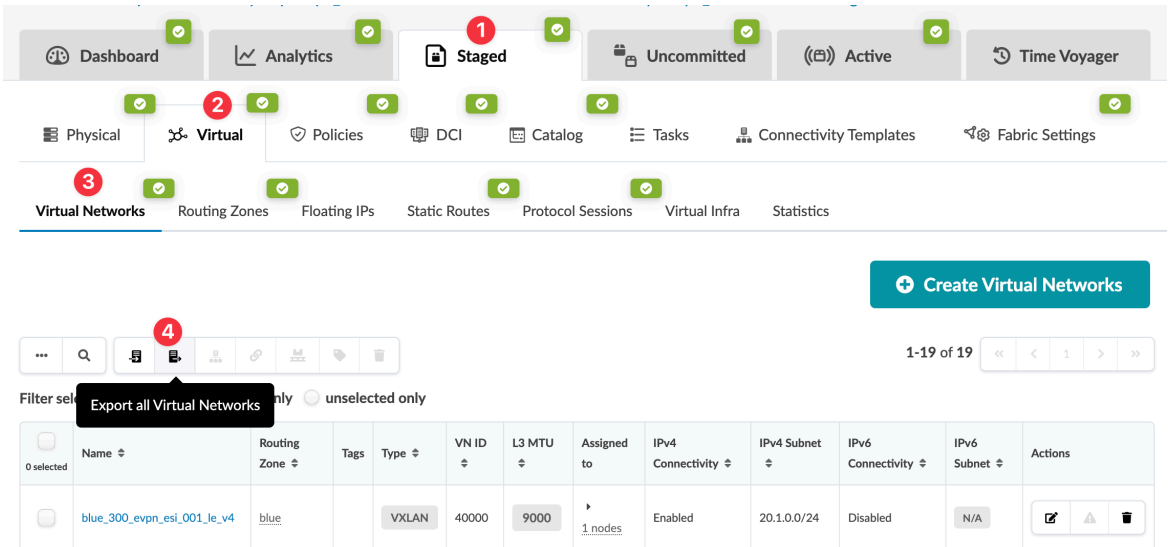
2. Make your changes.
3. Click **Update** to stage your changes and return to the table view.

When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

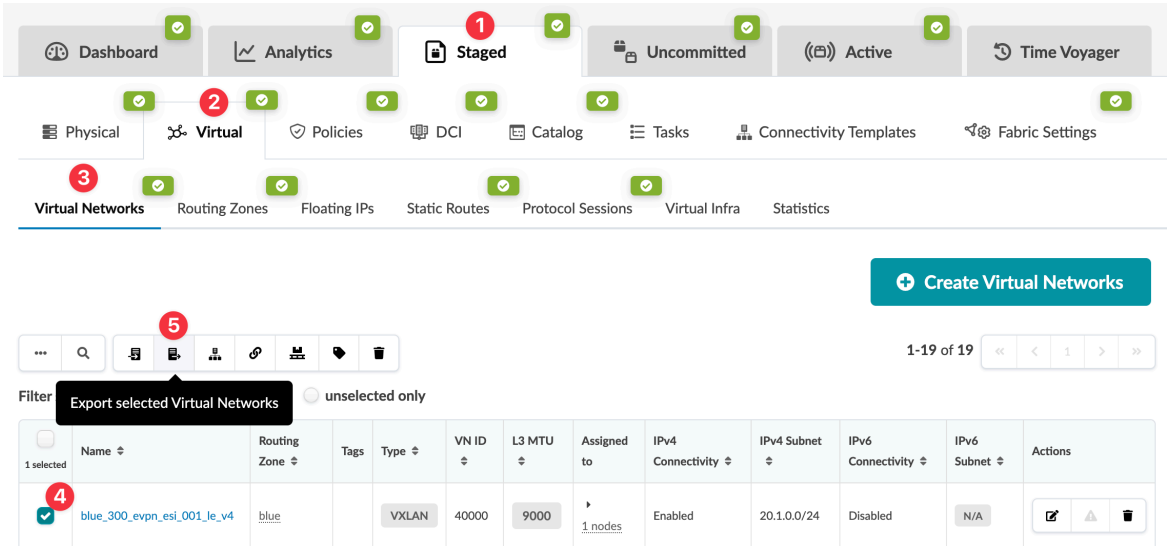
Edit Multiple Virtual Networks

You can update many virtual networks quickly by exporting them in a CSV file, updating the file, then importing the file back into your blueprint.

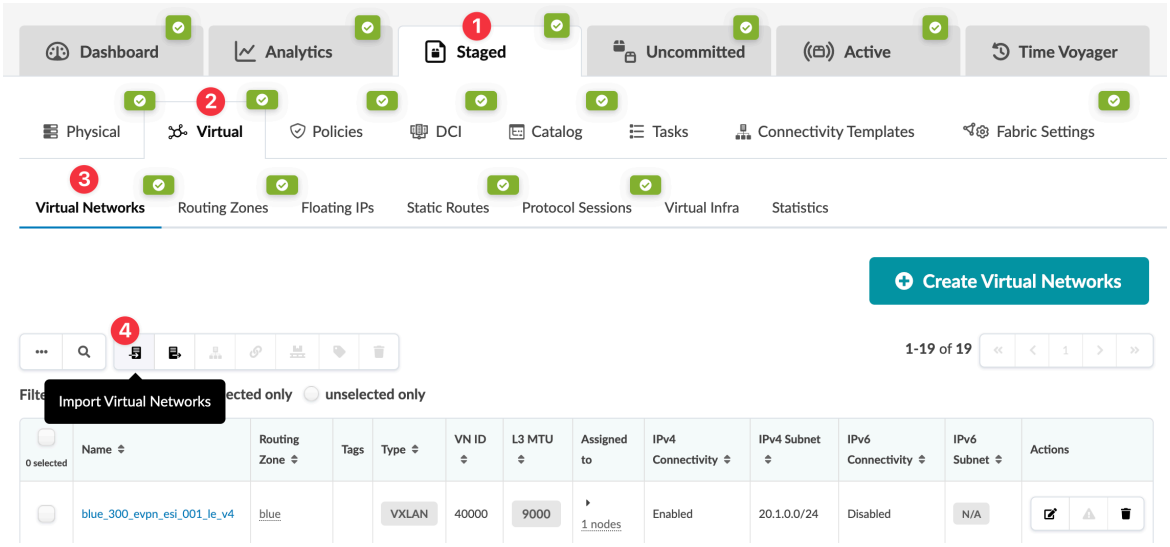
1. From the blueprint, navigate to **Staged > Virtual > Virtual Networks**.
2. To export all virtual networks, click the **Export all virtual networks** button.



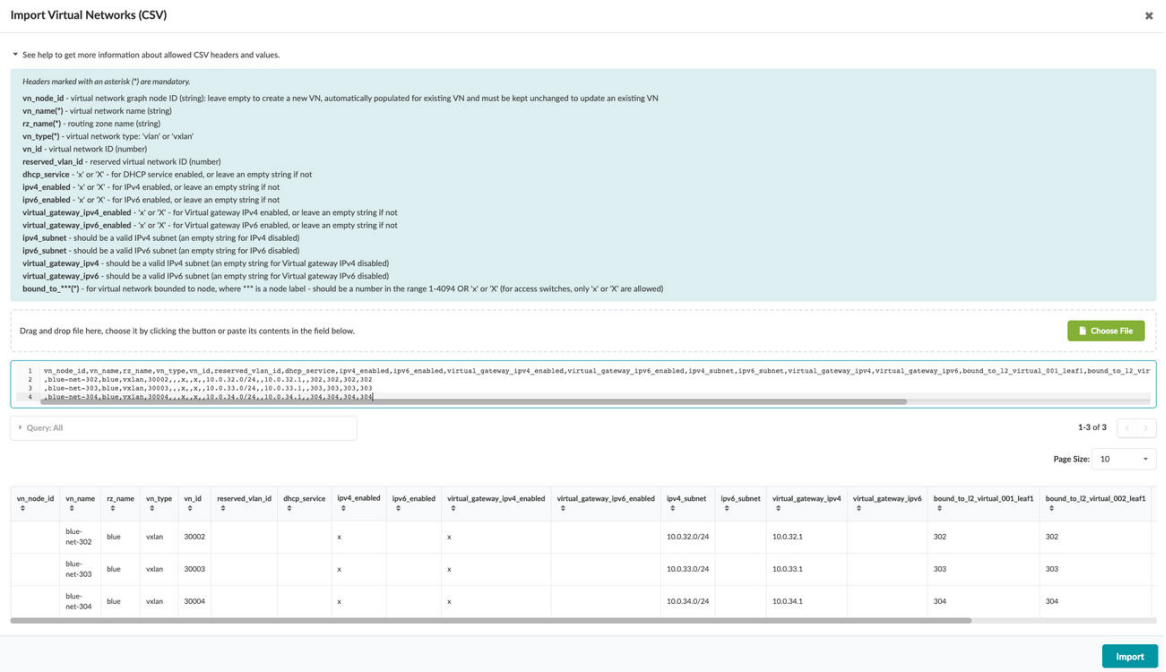
3. Or to export specific virtual networks instead of all of them, check their check boxes, then click the same button as in the previous step (now called **Export selected virtual networks**).



4. Click **Copy** to copy the contents, or click **Save As File** to download the file.
5. When you've copied or downloaded the virtual networks, close the dialog to return to the table view.
6. Paste the contents, or open the CSV file, in a spreadsheet program (such as Google Sheets or Microsoft Excel).
7. Update virtual networks as needed, then save the file.
8. In the Apstra GUI, navigate to **Staged > Virtual > Virtual Networks** and click the **Import virtual networks** button.



9. Either click **Choose File** and navigate to the file on your computer, or drag and drop the file onto the dialog window, or as shown in the screenshot below, directly paste CSV file contents. Virtual network details are displayed for your review.



10. Click **Import** to import the virtual networks, stage the changes, and return to the table view.

When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

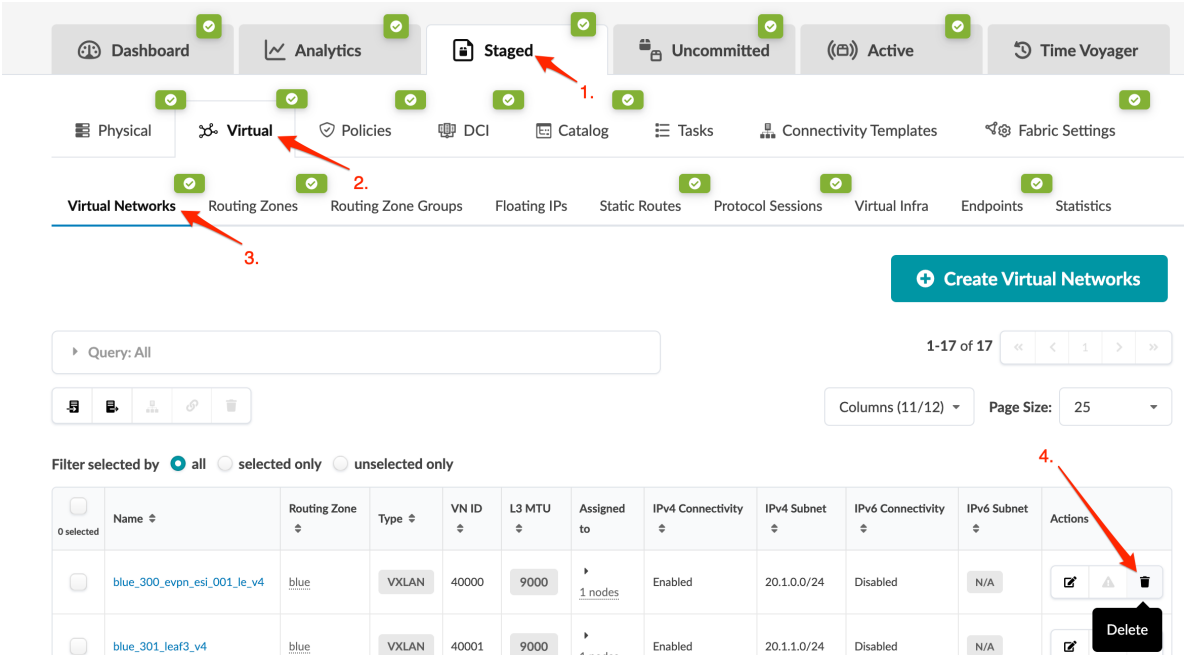
Delete Virtual Network

IN THIS SECTION

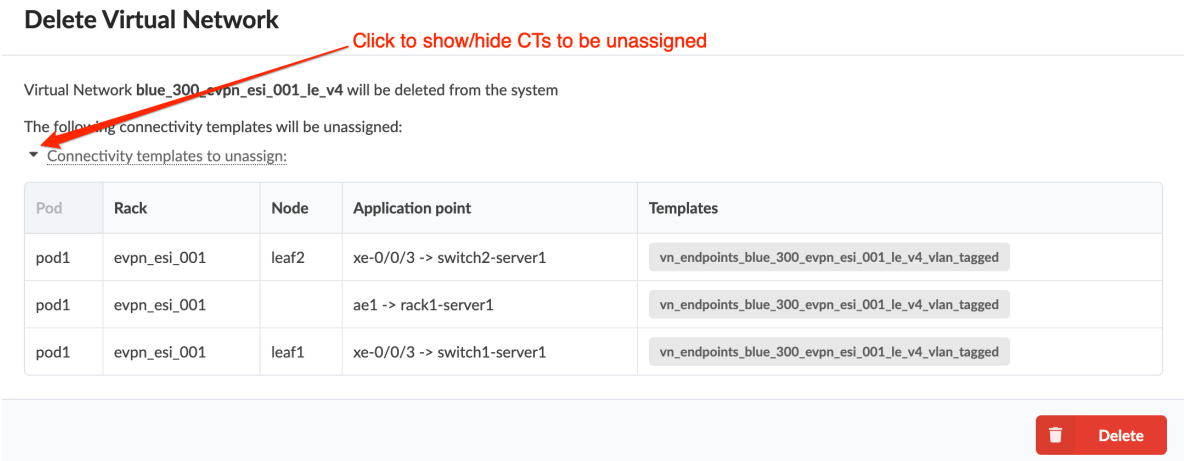
When you delete virtual networks, any connectivity templates that are assigned to those virtual networks are automatically unassigned. Those unassigned connectivity templates become available to be assigned elsewhere, or to be deleted. (In previous versions, you had to manually find and unassign connectivity templates before you could delete virtual networks.)

Delete One Virtual Network

1. From the blueprint, navigate to **Staged > Virtual > Virtual Networks** and click the **Delete** button in the **Actions** panel for the VN to delete.



2. The **Delete Virtual Network** dialog that opens shows the virtual network to be deleted. Click the drop-down triangle to show (or hide) the connectivity templates that will be unassigned.



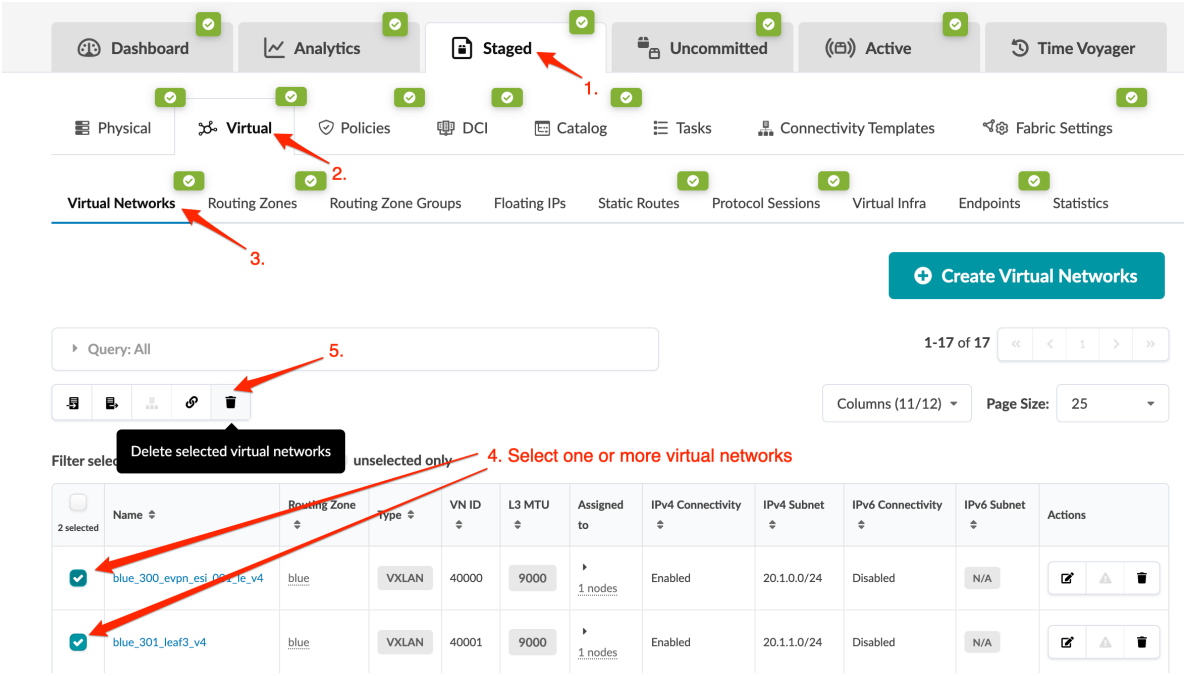
3. Click **Delete** to stage the deletion and return to the table view.

> **NOTE**: If you get an error, it's probably because there's a dependency that you need to remove manually. If a connectivity template refers to an object (like a virtual network endpoint) that is created by *another* connectivity template, you need to unassign that dependent object. Then you can return to step 1.

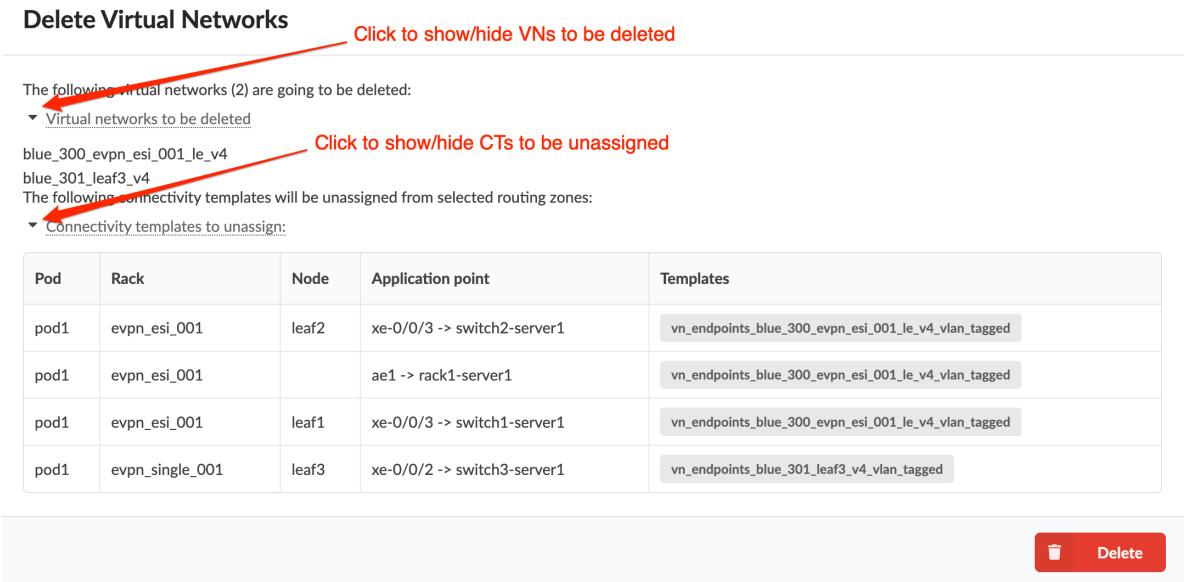
When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Delete Multiple Virtual Networks

1. From the blueprint, navigate to **Staged > Virtual > Virtual Networks**, check the check boxes for the virtual networks to delete, then click the **Delete selected virtual networks** button that becomes available above the table. (Tip: Use the Query function to filter specific virtual networks.)



2. In the **Delete Virtual Networks** dialog that opens, click the drop-down triangles to show (or hide) the virtual networks to be deleted and the connectivity templates to be unassigned.



3. Click **Delete** to stage the deletion and return to the table view.

> **NOTE**: If you get an error, it's probably because there's a dependency that you need to remove manually. If a connectivity template refers to an object (like a virtual network endpoint) that is created by *another* connectivity template, you need to unassign that dependent object. Then you can return to step 1.

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

SEE ALSO

- What are Virtual Networks | **251**
- Create Virtual Network | **258**
- Update Connectivity Template Assignments | **478**
- Delete Connectivity Template | **486**

Routing Zones

IN THIS SECTION

What are Routing Zones

A routing zone is an L3 domain, the unit of tenancy in multi-tenant networks. You create routing zones for tenants to isolate their IP traffic from one another, thus enabling tenants to reuse IP subnets. In addition to being in its own VRF, each routing zone can be assigned its own DHCP relay server and external system connections. You can create one or more virtual networks within a routing zone, which means a tenant can stretch its L2 applications across multiple racks within its routing zone. For virtual networks with Layer 3 SVI, the SVI is associated with a Virtual Routing and Forwarding (VRF) instance for each routing zone isolating the virtual network SVI from other virtual network SVIs in other routing zones. If you're using multiple routing zones, external system connections must be from leaf switches in the fabric. Routing between routing zones must be accomplished with external systems. All SVIs configured for virtual networks in this zone are in the default VRF. This is the same VRF used for the underlay or fabric network routing between network devices. All blueprints include a default routing policy. The number of routing zones is limited only by the network devices being used.

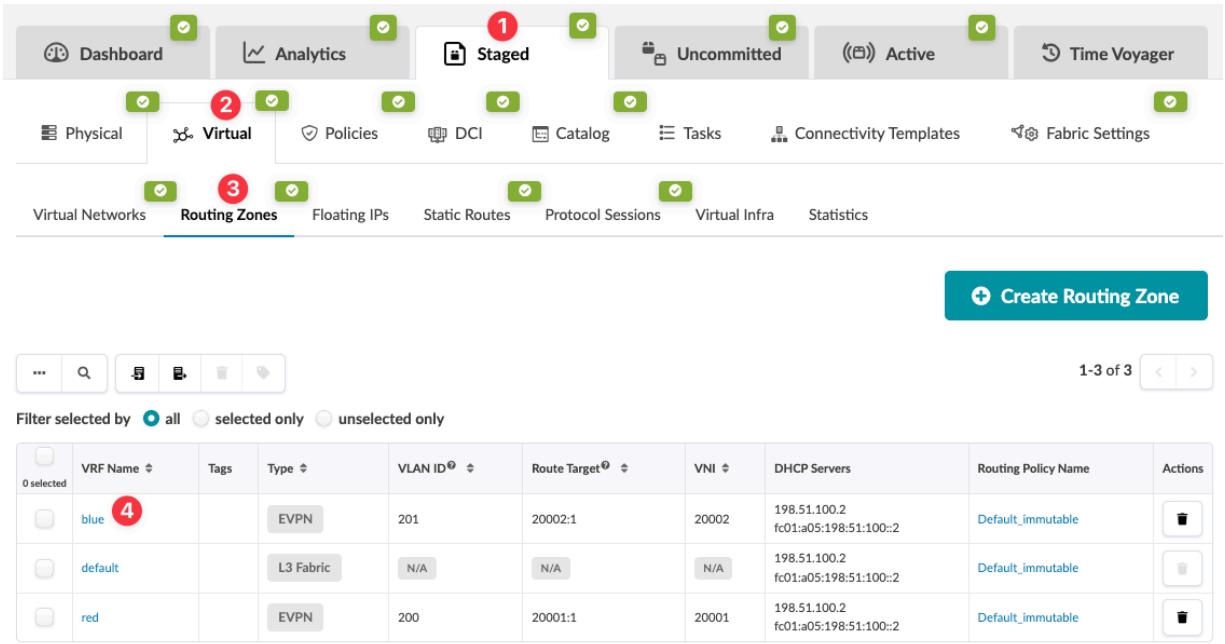
Routing zones include the following details:

Parameter	Description
VRF Name	15 characters or fewer. Underscore, dash and alphanumeric characters only
VRF Description	Optional field to add additional information/context to the routing zone configuration that is also available on the switch itself. 240 characters or fewer.
Type	L3 Fabric or EVPN
VLAN ID	Used for VLAN tagged Layer 3 links on external connections. Leave this field blank to have it automatically assigned from a static pool in the range of 2-4094), or enter a specific value.
VNI	VxLAN VNI associated with the routing zone. Leave this field blank to have it automatically assigned from a resource pool, or enter a specific value.
Tenant	

(Continued)

Parameter	Description
Route Target	Only EVPN routing zones use route targets. The rendered EVPN L3-VNI route target represents the built-in, automatic route target that is associated with the EVPN routing zone VRF. When using EVPN remote gateway features for Data Center Interconnect, this route target must be imported by the EVPN fabric external to this fabric. This route target is composed of "<VNI_ID>:1" where "1" is hard-coded. If route target is not assigned, then a VNI must be assigned.
DHCP Servers	
Routing Policies	Non-EVPN blueprints must use the default policy. EVPN blueprints can use non-default policies. For more information, see "What are Routing Policies" on page 396.
Route Target Policies	• Import Route Targets • Export Route Targets
Resources	Resources include loopback IP addresses, peer link IP addresses, and SVI subnets.
Virtual Networks	The virtual networks that are assigned to the routing zone
Interfaces	The interfaces that are associated with the routing zone

From the blueprint, navigate to **Staged > Virtual > Routing Zones** to go to the routing zones table view. You can create, edit, import, export and delete routing zones and assign DHCP servers to them.



To go to routing zone details, click a VRF name in the table. As of Apstra version 5.0.0, assigned loopback IP addresses are included in a new section at the bottom of the details page called **System Loopback Interfaces**.

Parameters

VRF Name	blue
VRF Description	
Type	EVPN
Tags	
VLAN ID	201
VNI	20002
Tenant	tenant_east
Route Target	20002:1
Junos EVPN IRB Mode	Symmetric
DHCP Servers	198.51.100.2 fc01:a05:198:51:100::2

Routing Policy

Name	Default_immutable
Description	Associated with routing zones by default, cannot be updated or deleted.
Import Policy	Default
Extra Import Routes	Not provided
Spine Leaf Links	no
Spine Superspine Links	no
L2 Edge Subnets	yes
Loopbacks	yes
Static Routes	no
Extra Export Routes	Not provided
Aggregate Prefixes	Not provided
Expect Default IPv4 Route	yes
Expect Default IPv6 Route	yes

Route Target Policies

Import Route Targets	Not provided
Export Route Targets	Not provided

Resources

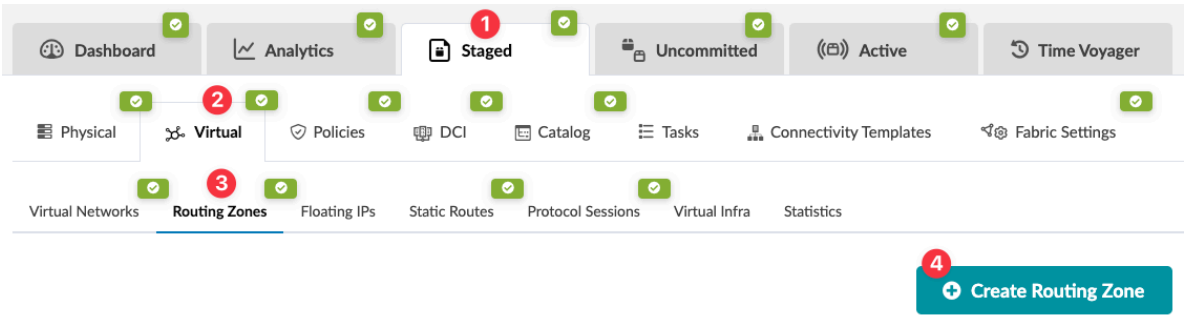
Create Routing Zone

IN THIS SECTION

If your blueprint is using **MP-EBGP EVPN** overlay control protocol, you can create routing zones. If it's using **Pure IP Fabric**, you must use the default routing zone. (Overlay control protocol is specified in "templates" on page 889.) You can create one routing zone at a time, or with the help of CSV files, you can create multiple routing zones.

Create Routing Zones (using GUI)

1. From the blueprint, navigate to **Staged > Virtual > Routing Zones** and click **Create Routing Zone**.



2. Enter a unique VRF name (15 characters or fewer) and (optional) description (up to 240 characters).
3. You can leave the remaining fields as is to use default values and have resources assigned from pools, or you can configure them manually. See "What are Routing Zones" on page 282 for details.
4. Click **Create** to create the routing zone and return to the table view.

Next Steps:

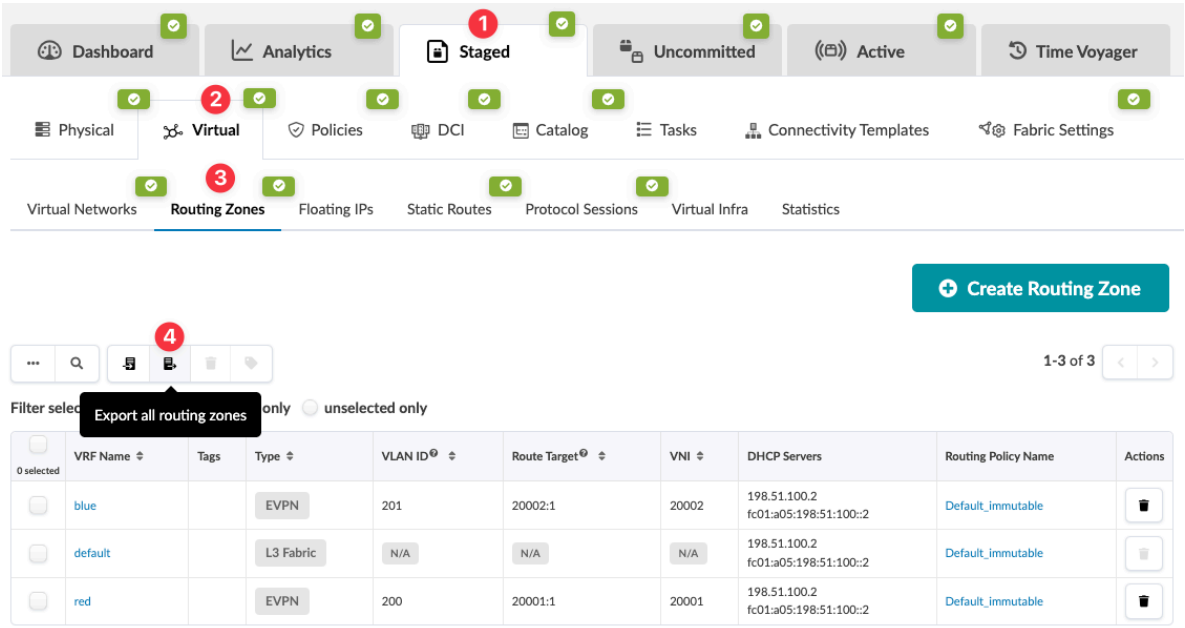
Assign resources. Each leaf network device in each routing zone requires a loopback IP address. If IPv6 is enabled on the blueprint, you must also assign IPv6 addresses to the routing zone. After you've assigned connectivity templates to your external generic systems, you'll also need to assign IP addresses.

As of Apstra version 5.0.0, assigned loopback IP addresses are included in a new section at the bottom of the routing zone details page called **System Loopback Interfaces**.

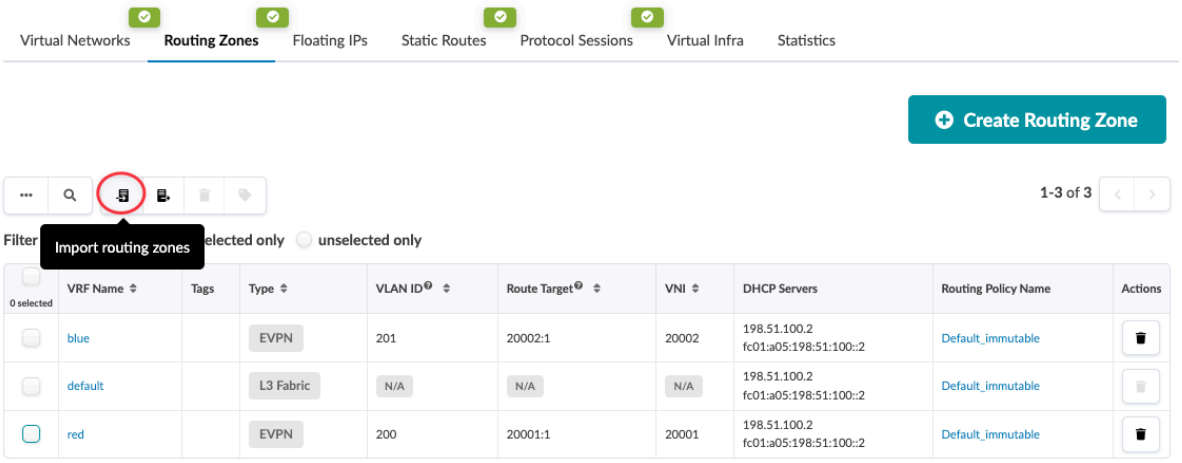
Create Routing Zones (using CSV File)

You can create many routing zones at once with a CSV file. First, you'll export the routing zone schema from your blueprint, then open and populate the file in a spreadsheet program. And finally, you'll import the file back into your blueprint.

1. From the blueprint, navigate to **Staged > Virtual > Routing Zones** and click the **Export all routing zones** button, which is above the table.



2. In the dialog that opens, click **Copy** to copy the contents or click **Save As File** to download the file.
3. Paste the contents, or open the CSV file, in a spreadsheet program (such as Google Sheets or Microsoft Excel).
4. Enter routing zone details into the spreadsheet, then save the file. For details about each field, see "What are Routing Zones" on page 282.
5. In the Apstra GUI, navigate to **Staged > Virtual > Routing Zones** and click the **Import routing zones** button, which is above the table.



6. Either click **Choose File** and navigate to the file on your computer, drag and drop the file onto the dialog window, or directly paste CSV file contents into the dialog window. Routing zone details are displayed for your review.
7. Click **Import** to import the routing zones, stage the changes, and return to the table view.

Next Steps:

Assign resources. Each leaf network device in each routing zone requires a loopback IP. If IPv6 is enabled on the blueprint, you must also assign IPv6 addresses to the routing zone. After you've assigned connectivity templates to your external generic systems, you'll also need to assign IP addresses.

As of Apstra version 5.0.0, assigned loopback IP addresses are included in a new section at the bottom of the routing zone details page called **System Loopback Interfaces**.

Update Routing Zone Tags

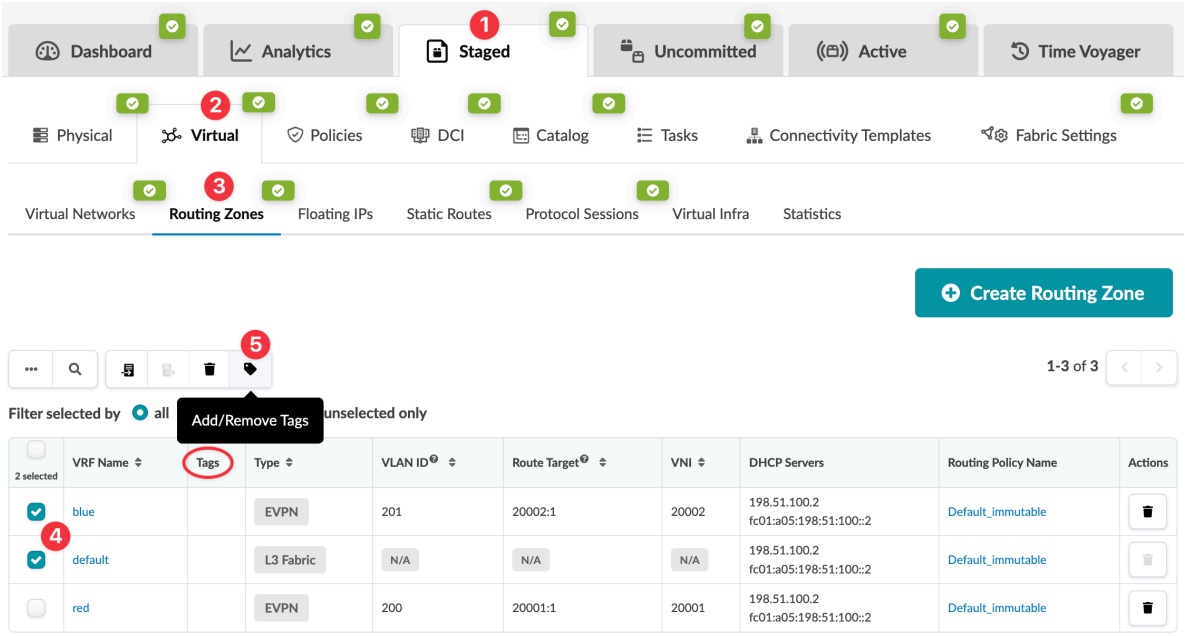
You can group and search routing zones based on tags that you apply to them (new in Apstra version 5.0.0). Using tags on routing zones can also enhance building dynamic configlets for VRF-based requirements.

If you'd like to apply a tag from the **Design** (global) catalog (Design > Tags), "import" on page 454 it to the **Blueprint** catalog beforehand.

You can also "create" on page 453 a tag directly in the **Blueprint** catalog before you add it to a routing zone, which also makes it available to the other taggable elements in the blueprint (nodes, links, interfaces, port channels, virtual networks, connectivity templates).

Finally, you can create a tag on the fly as you're adding it to a routing zone and it will automatically be added to the catalog as well.

1. From the blueprint, navigate to **Staged > Virtual > Routing Zones** and select one or more check boxes for the routing zone(s) that need the same tag updates. The **Add/Remove Tags** button above the table becomes available.



2. Click the **Add/Remove Tags** button, then add and/or remove tags, as needed.

   The **Add Tags** drop-down list includes all tags that are in the catalog (Staged > Catalog > Tags). Enter the tag name, and if it's not in the catalog you'll get the option to add it; it will be applied to the routing zone and added to the catalog.

   The **Remove Tags** drop-down list includes all tags that are applied to the selected routing zone(s). Select tag(s) from the list to remove them. (If no tags are applied, the drop-down list isn't shown in the dialog.)

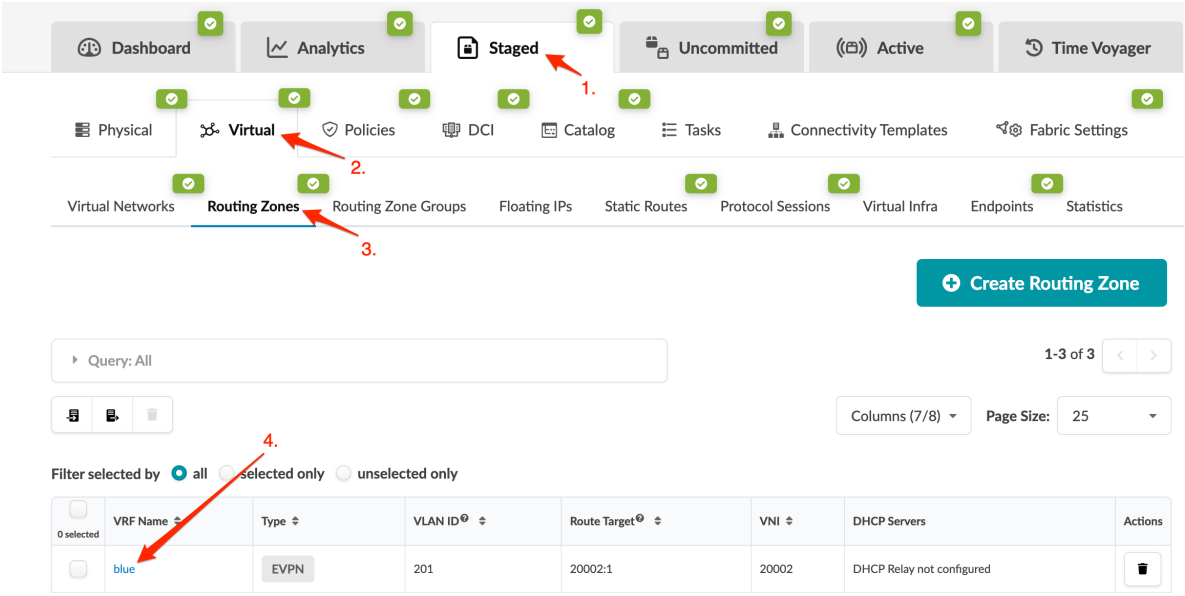
3. Click **Add/Remove Tags** to stage the change and return to the **Routing Zones** table.

   If you have **Tags** selected in **Table Settings** (above the table, leftmost button with 3 dots), you'll see them in the **Tags** column in the table.

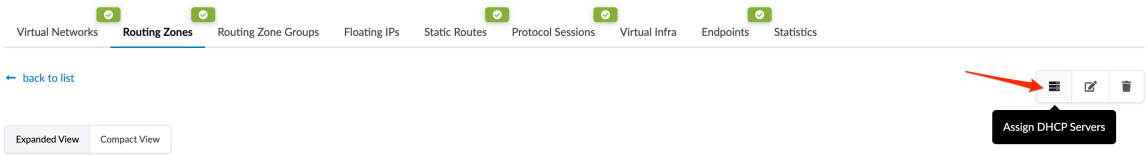
When you need to find something based on tags, you can search routing zones with the **Search** button above the table, or for a wider search, use the **Search** field at the top of the **Staged** tab or the **Find by tags** button in the upper-right.

Assign DHCP Server to Routing Zone

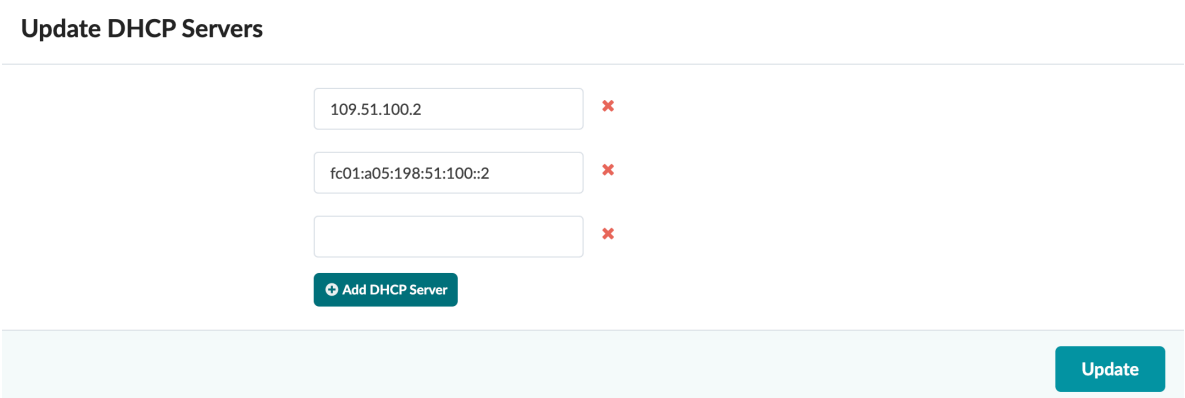
1. From the blueprint, navigate to **Staged > Virtual > Routing Zones** and click the name of the routing zone that needs a DHCP server assigned to it. (The screenshot below is for Apstra version 4.2.0. Newer versions include a column for Routing Policy Name that you can link to directly; and you can select which columns to show in the table.)



2. Click the **Assign DHCP Servers** button (upper-right).



3. Enter the IPv4 address (or IPv6 address) for the DHCP server and click **Add DHCP Server**. To add an additional server, enter the IP address and click **Add DHCP Server** again.



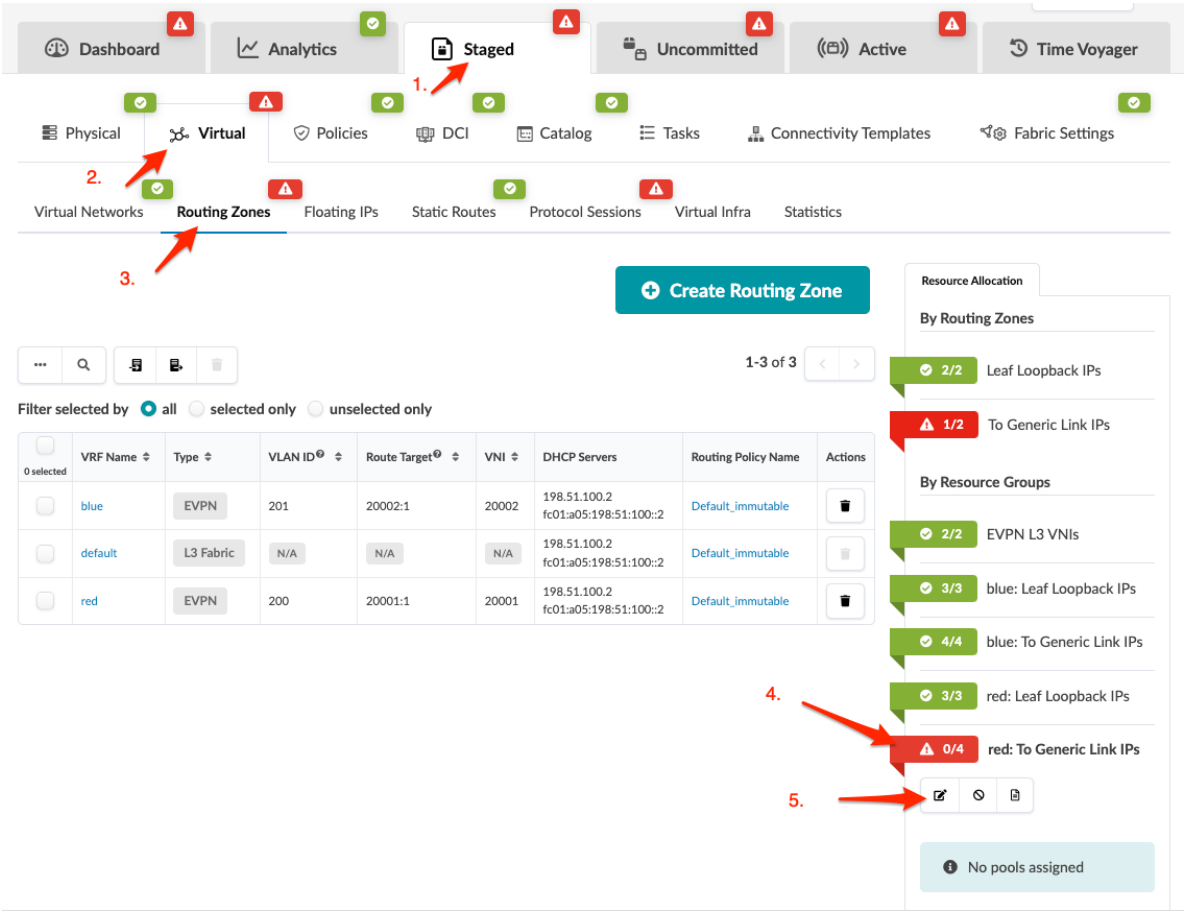
4. Click **Update** to stage the assignment and return to the routing zone detail view.

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Assign Routing Zone Resources

Each leaf network device in each routing zone requires a loopback IP. If IPv6 is enabled on the blueprint, you must also assign IPv6 addresses to the routing zone. After you've assigned connectivity templates to your external generic systems, you'll also need to assign IP addresses.

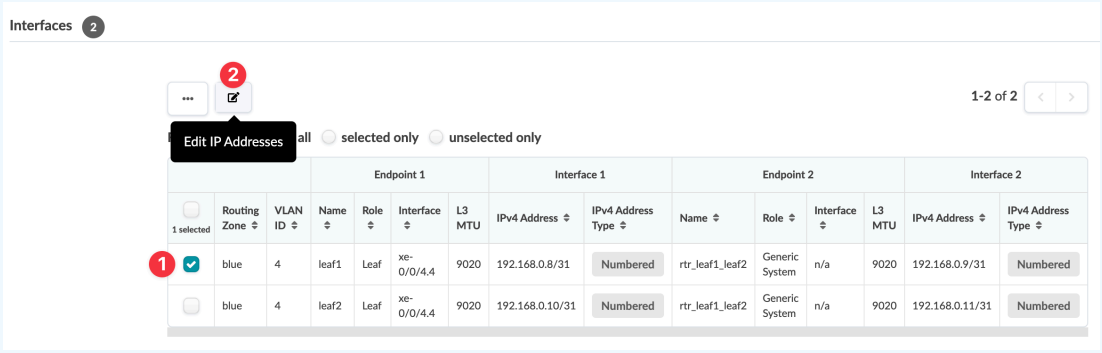
1. From the blueprint, navigate to **Staged > Virtual > Routing Zones**.
2. Red status indicators in the **Build** panel (on the right) indicate that resources need to be assigned. Click a red indicator and click the **Update assignments** button.



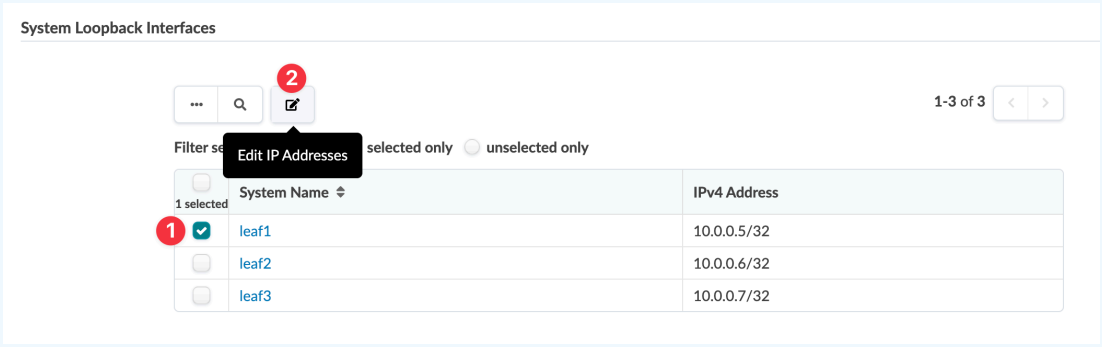
3. Select a pool from which to pull the resources, then click the **Save** button. (For information about IP address pools, see "IP Pools" on page 934.) When the red status indicator turns green, the required resources are successfully assigned.
4. Repeat the steps to assign resources from pools until all required resources have been assigned.

NOTE: You can also assign or change individual IP addresses to links by clicking the name of the routing zone in the table view, scrolling down to the **Interfaces** section, selecting one or

more check boxes, clicking the **Edit IP addresses** button, and entering them from there.



As of Apstra version 5.0.0, you can also assign or change individual loopback IP addresses by clicking the name of the routing zone in the table view, scrolling down to the **System Loopback Interfaces** section, selecting one or more check boxes, clicking the **Edit IP addresses** button, and entering them from there.



Reset Routing Zone Resource Group Overrides

IN THIS SECTION

Benefits of <feature-name>

-

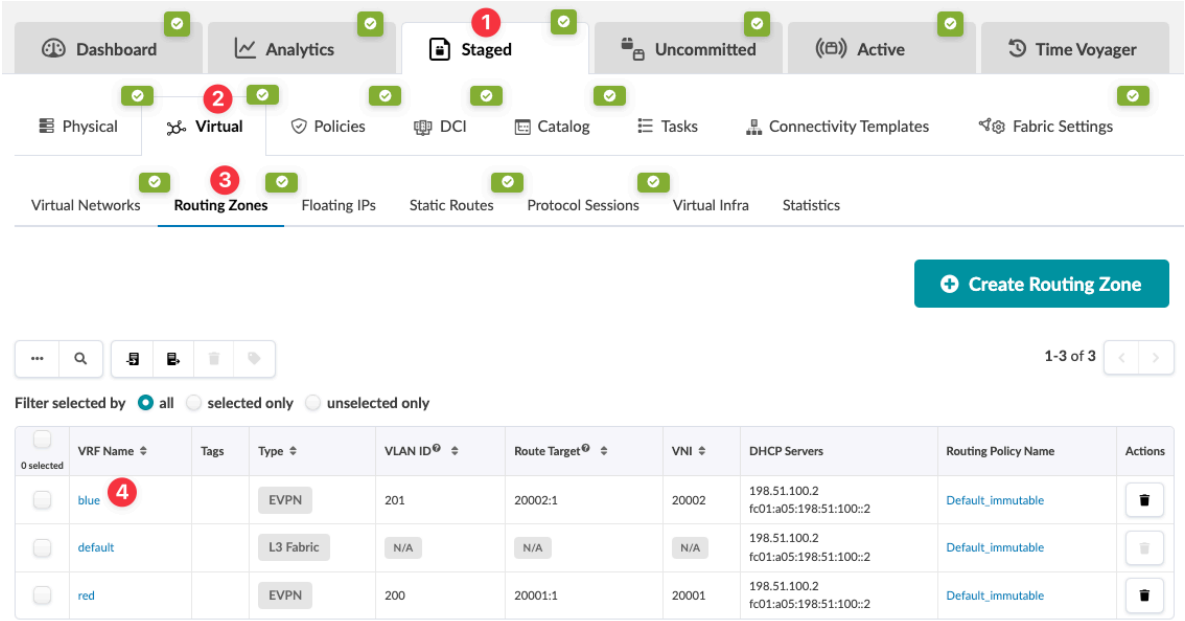
Change Routing Zone Details

IN THIS SECTION

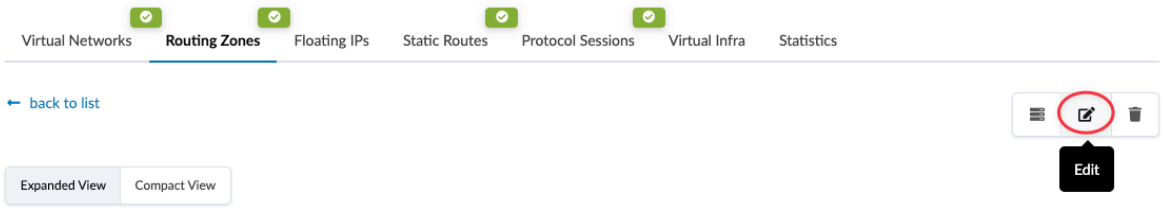
You can change a routing zone's description, VLAN ID, VNIs, routing policies and symmetric IRB mode for Junos EVPN. You can change one routing zone at a time or, with the help of CSV files, you can edit multiple routing zones.

Update One Routing Zone

1. From the blueprint, navigate to **Staged > Virtual > Routing Zones** and click the name of the routing zone to edit.



2. Click the **Edit** button (upper-right) for the selected routing zone.



3. In the dialog that opens, make the required changes. For details about each field, see "What are Routing Zones" on page 282.

Edit Routing Zone

VRF Name
blue

VRF Description

VLAN ID
201

VNI
20002

Route Target
20002:1

Routing Policies

Default_immutable

Name	Default_immutable
Description	Associated with routing zones by default, cannot be updated or deleted.
Import Policy	Default
Extra Import Routes	Not provided
Spine Leaf Links	no
Spine Superspine Links	no
L2 Edge Subnets	yes
Loopbacks	yes
Static Routes	no
Extra Export Routes	Not provided
Aggregate Prefixes	Not provided
Expect Default IPv4 Route	yes
Expect Default IPv6 Route	yes

Route Target Policies

Import Route Targets

Add Import Route Target

Export Route Targets

Add Export Route Target

Symmetric IRB mode for Junos EVPN

Changing this value will result in a disruption of EVPN Type2 routes while they are re-generated

Enables Symmetric IRB Routing for EVPN on Junos devices. This makes use of an L3 VNI for inter-subnet routing which is embedded into EVPN Type2-routes. This supports better scaling for networks with large amounts of VLANs. The default model is 'asymmetric' indicating asymmetric EVPN mode. This option is only applicable to security zones of sz_type 'evpn'

(•) Asymmetric ( ) Symmetric

Update

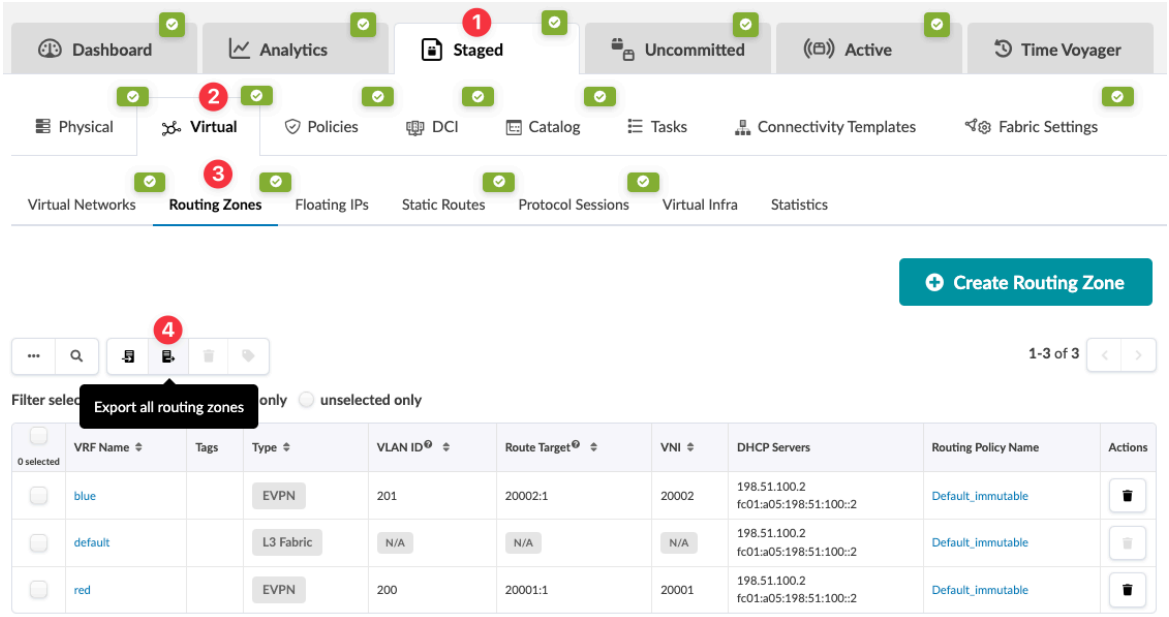
4. Click **Update** to stage your changes and return to routing zone details.

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

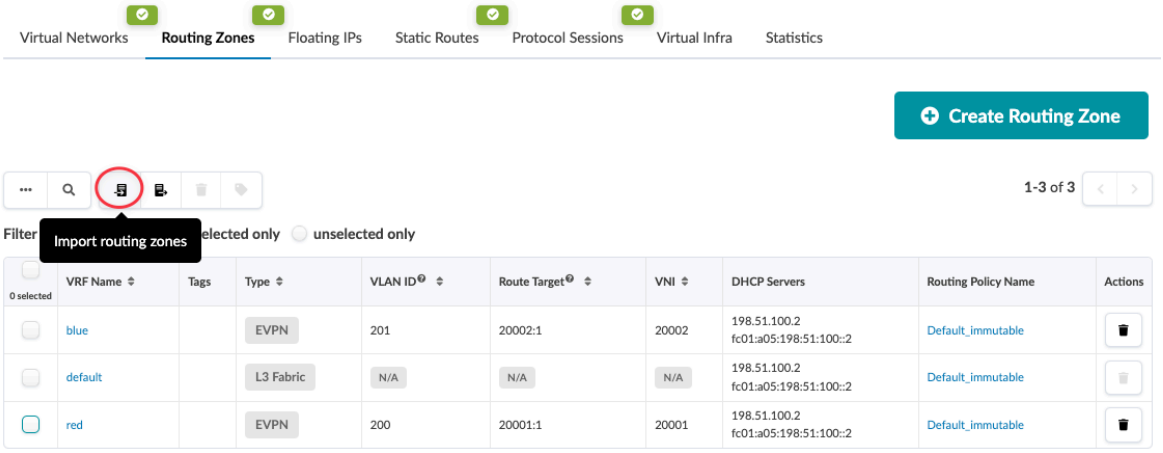
Update Multiple Routing Zones

You can update all routing zones, or a selected few, by exporting them to a CSV file, updating the file, then importing the file back into your blueprint.

1. From the blueprint, navigate to **Staged > Virtual > Routing Zones**.
2. To export all routing zones, click the **Export all routing zones** button, which is above the table.



3. Or to export specific routing zones instead of all of them, select their check boxes, then click the same button as in the previous step (now called **Export selected routing zones**).
4. Click **Copy** to copy the contents, or click **Save As File** to download the file.
5. When you've copied or downloaded the routing zones, close the dialog to return to the table view.
6. Paste the contents, or open the CSV file, in a spreadsheet program (such as Google Sheets or Microsoft Excel).
7. Update routing zones as needed, then save the file. For details about each field, see "What are Routing Zones" on page 282.
8. In the Apstra GUI, navigate to **Staged > Virtual >Routing Zones** and click the **Import routing zones** button, which is above the table.



9. Either click **Choose File** and navigate to the file on your computer, or drag and drop the file onto the dialog window. Routing zone details are displayed for your review.
10. Click **Import** to import the routing zones, stage the changes, and return to the table view.

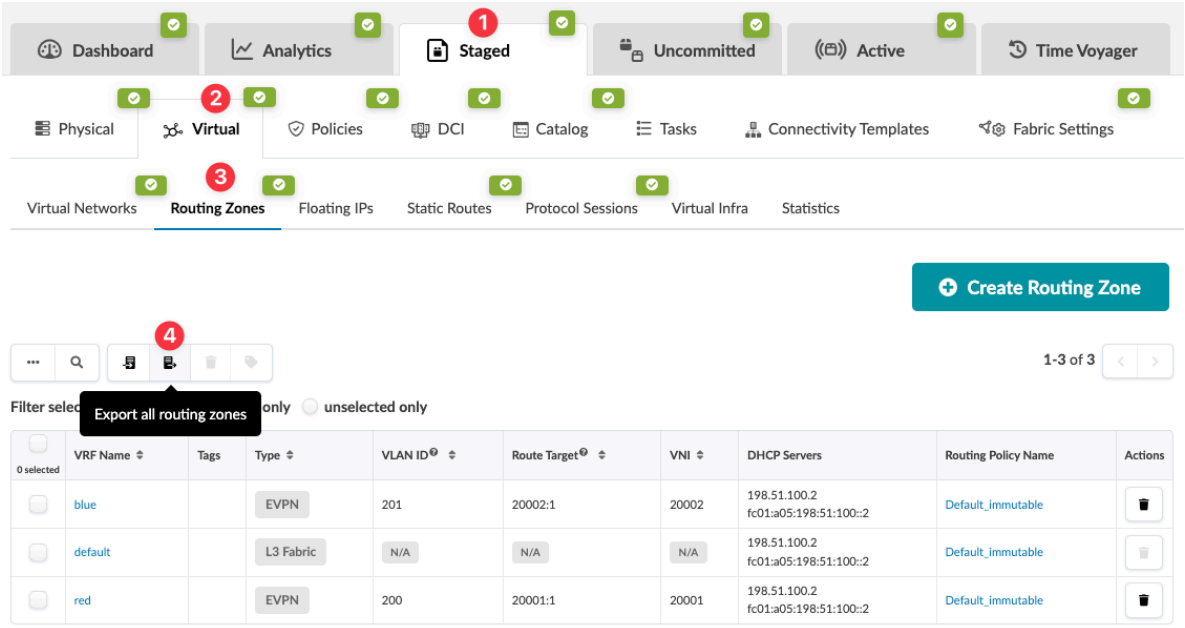
When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Export Routing Zone

SUMMARY

You can update all routing zones, or a selected few, by exporting them to a CSV file, updating the file, then importing the file back into the blueprint.

1. From the blueprint, navigate to **Staged > Virtual > Routing Zones**.
2. To export all routing zones, click the **Export all routing zones** button that's above the table.



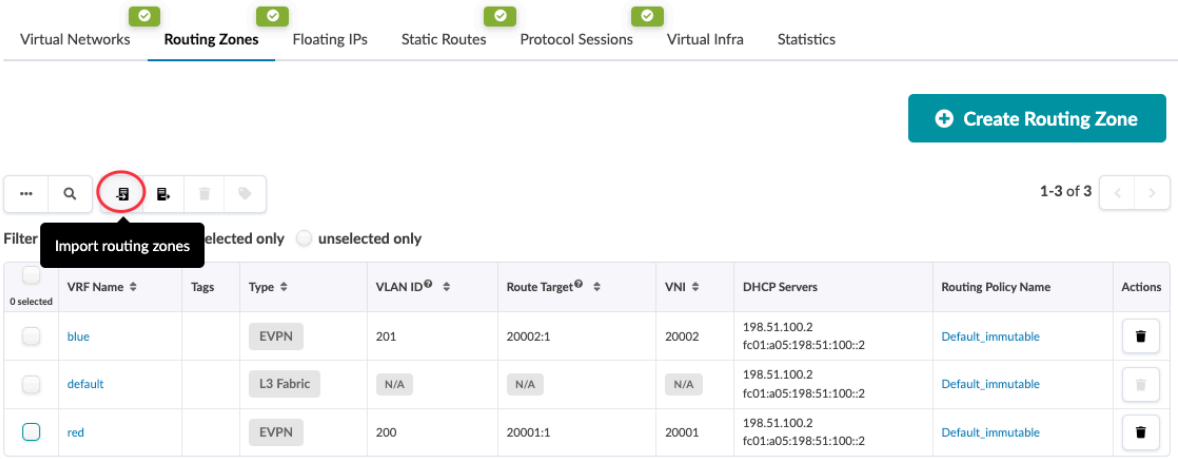
3. Or to export specific routing zones instead of all of them, check their check boxes, then click the same button as in the previous step (now called **Export selected routing zones**).
4. Click **Copy** to copy the contents or click **Save As File** to download the file.
5. When you've copied or downloaded the routing zones, close the dialog to return to the table view.

Next Steps: Update the CSV file with a spreadsheet program, then "import" on page 298 it back into your blueprint.

Import Routing Zone

You can import multiple routing zones (as a CSV file) into your blueprint. (Tip: First "export routing zones" on page 297 so you'll have the schema set up for you in the CSV file.)

1. From the blueprint, navigate to **Staged > Virtual > Routing Zones** and click the **Import routing zones** button.



2. Either click **Choose File** and navigate to the file on your computer, drag and drop the file onto the dialog window, or directly paste CSV file contents into the dialog window. Routing zone details are displayed for your review.
3. Click **Import** to import the routing zones, stage the changes, and return to the table view.

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

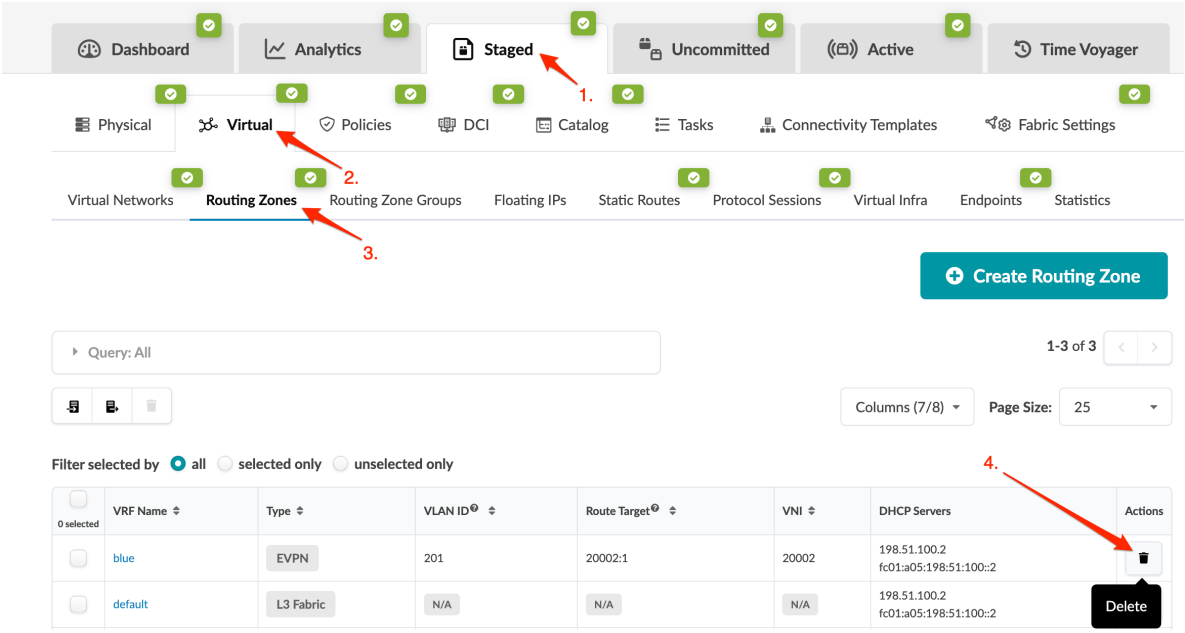
Delete Routing Zone

IN THIS SECTION

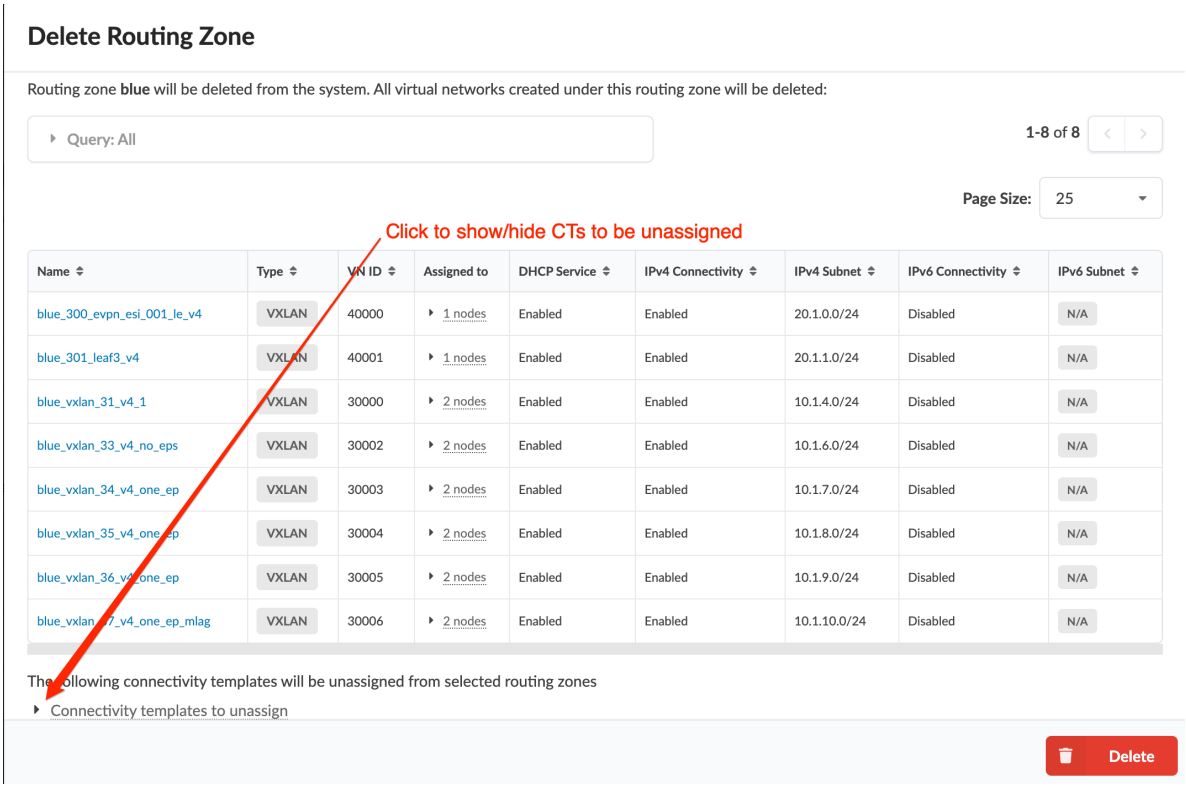
When you delete routing zones, all virtual networks created under the routing zone are also deleted, and the connectivity templates that are assigned are automatically unassigned. Those unassigned connectivity templates become available to be assigned elsewhere, or to be deleted. (In previous versions, you had to manually find and unassign connectivity templates before you could delete routing zones.)

Delete One Routing Zone

1. From the blueprint, navigate to **Staged > Virtual > Routing Zones** and click the **Delete** button in the **Actions** panel for the RZ to delete. (The screenshot below is for Apstra version 4.2.0. Newer versions include a column for Routing Policy Name that you can link to directly; and you can select which columns to show in the table, and some menu tabs have been renamed, moved, and/or added.))



2. The **Delete Routing Zone** dialog that opens shows the routing zone and virtual networks to be deleted. Click the drop-down triangle to show (or hide) the connectivity templates that will be unassigned.



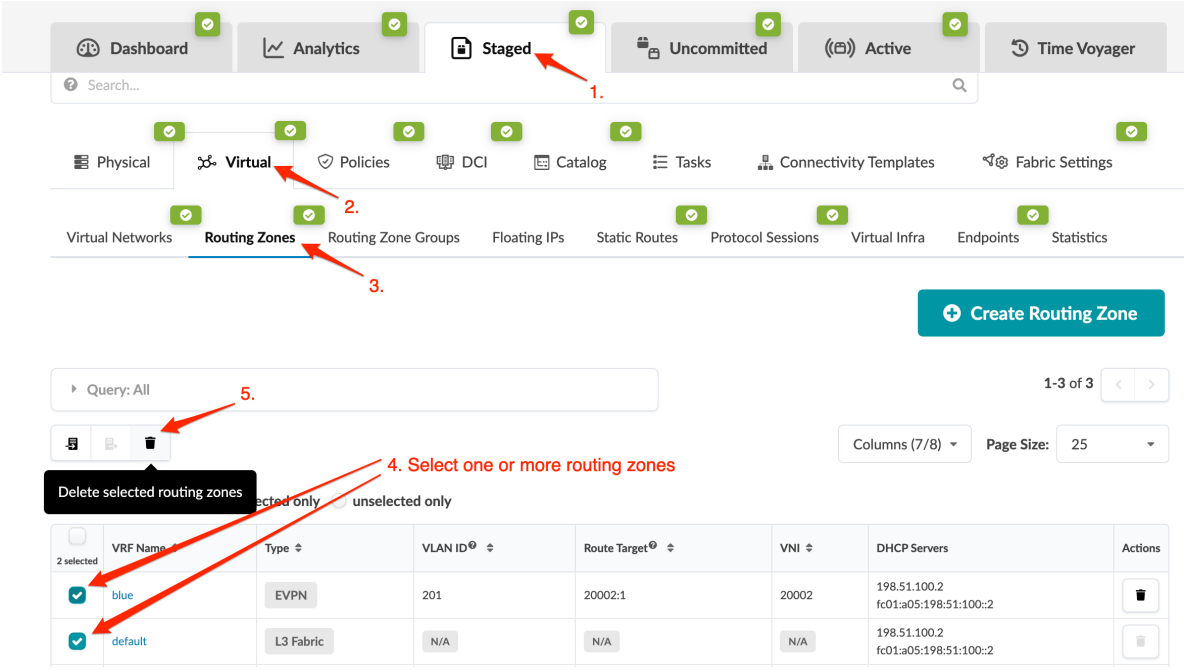
3. Click **Delete** to stage the deletion and return to the table view.

> **NOTE**: If you get an error, it's probably because there's a dependency that you need to remove manually. If a connectivity template refers to an object (like a virtual network endpoint) that is created by another connectivity template, you need to unassign that dependent object. Then you can return to step 1.

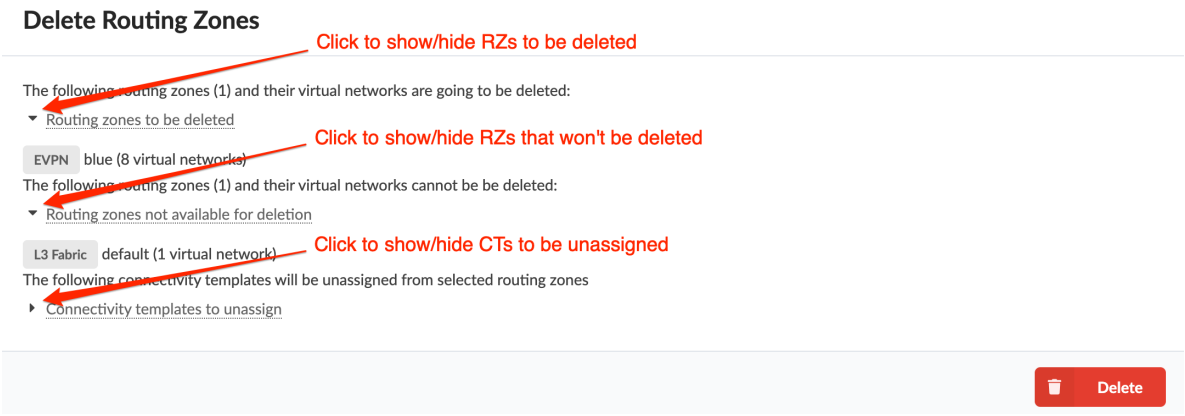
When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Delete Multiple Routing Zones

1. From the blueprint, navigate to **Staged > Virtual > Routing Zones**, select the check boxes for the routing zones to delete, then click the **Delete selected routing zones** button that becomes available above the table. (Tip: Use the Query function to filter specific routing zones.) (The screenshot below is for Apstra version 4.2.0. Newer versions include a column for Routing Policy Name that you can link to directly; and you can select which columns to show in the table, and some menu tabs have been renamed, moved, and/or added.)



2. In the **Delete Routing Zones** dialog that opens, click the drop-down triangles to show (or hide) the routing zones to be deleted and the connectivity templates to be unassigned



3. Click **Delete** to stage the deletion and return to the table view.

> **NOTE**: If you get an error, it's probably because there's a dependency that you need to remove manually. If a connectivity template refers to an object (like a virtual network endpoint) that is created by another connectivity template, you need to unassign that dependent object. Then you can return to step 1.

SEE ALSO

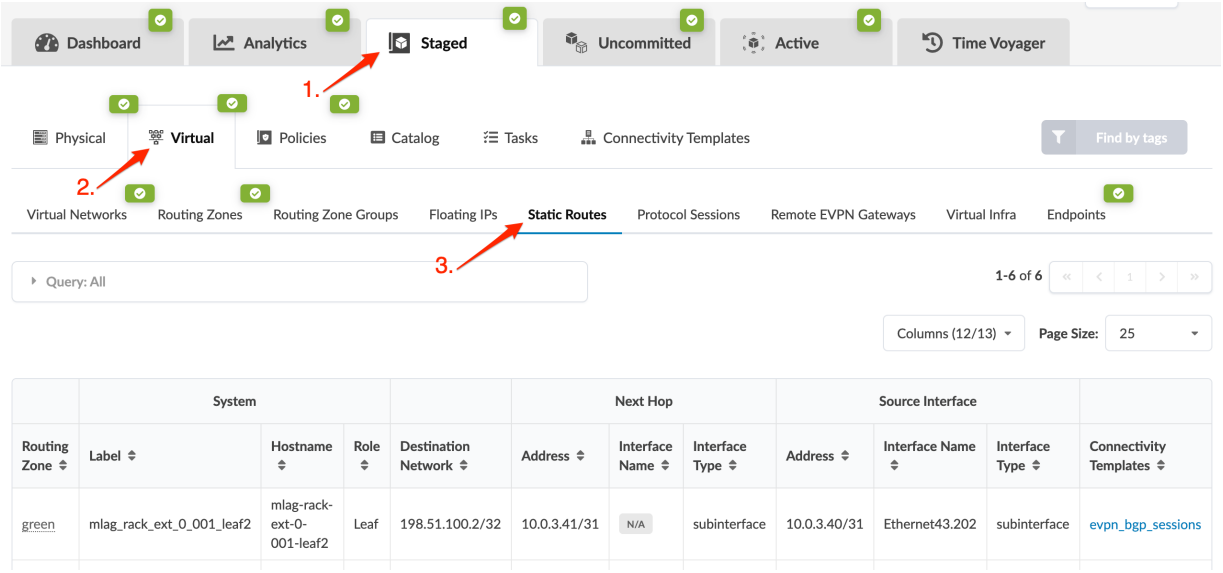
- What are Routing Zones | 282
- Create Routing Zone | 286
- Update Connectivity Template Assignments | 478
- Delete Connectivity Template | 486

Static Routes

IN THIS SECTION

Static Routes (Virtual)

When you create connectivity templates, static routes are created. From the blueprint, navigate to **Staged > Virtual > Static Routes** to go to static routes.

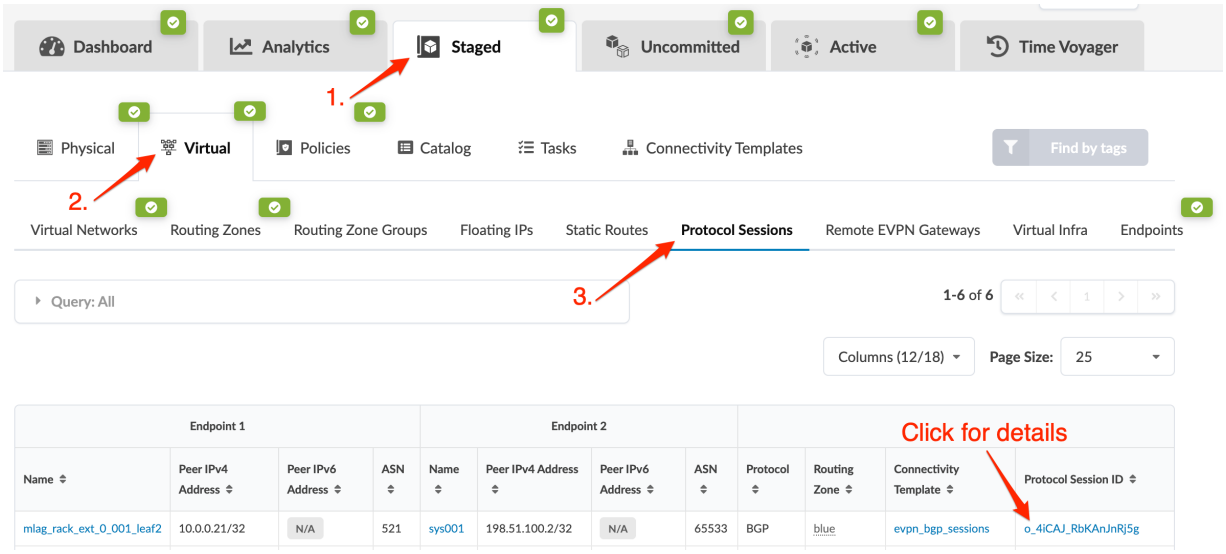


Protocol Sessions

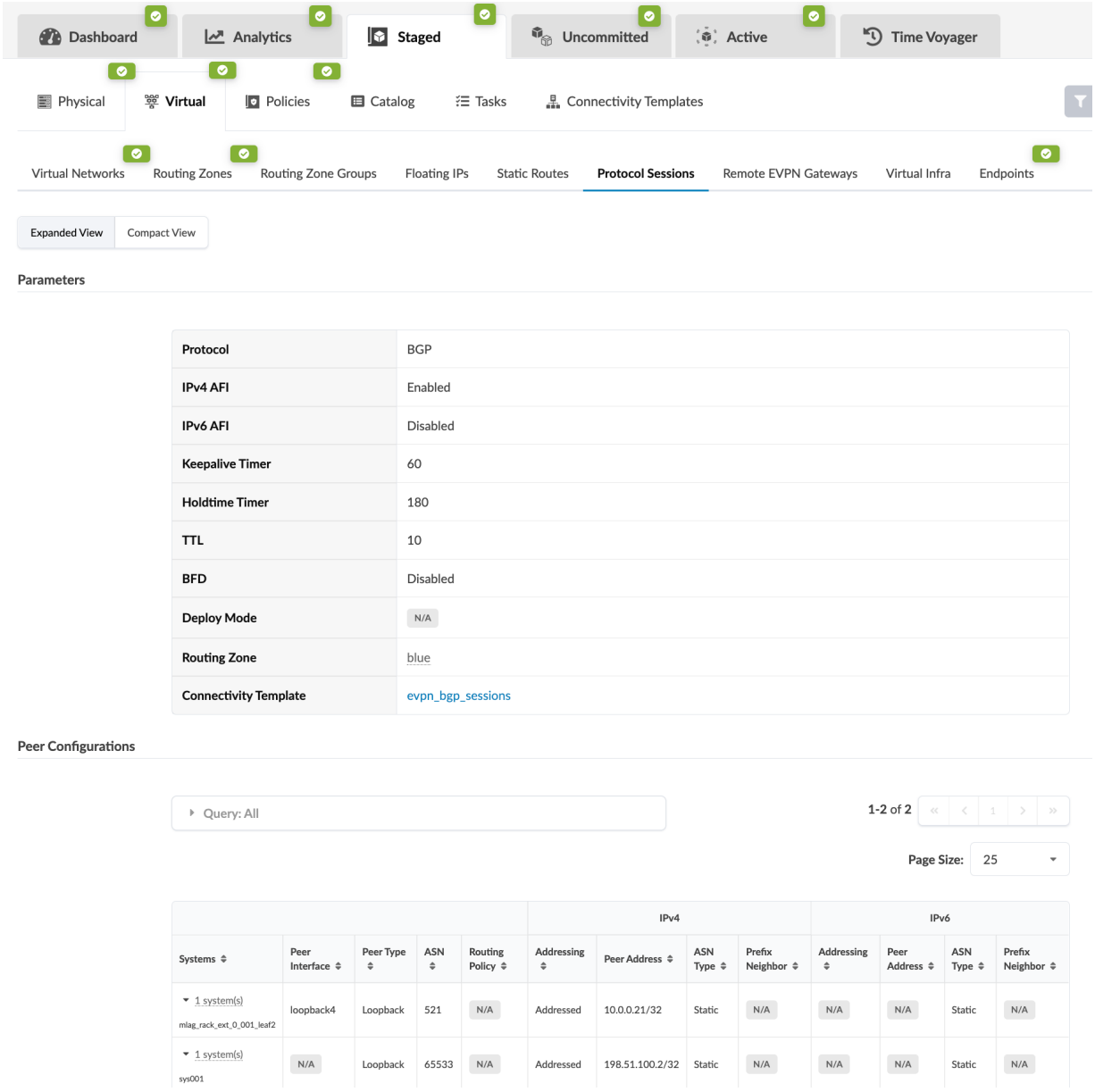
IN THIS SECTION

Protocol Sessions (Virtual)

When you create connectivity templates, protocol sessions (BGP sessions) are created. (As of Apstra version 4.0, protocol sessions replace security zone external connectivity points.) From the blueprint, navigate to **Staged > Virtual > Protocol Sessions** to go to protocol sessions.


Dashboard
Analytics
Staged
Uncommitted
Active
Time Voyager
1.
Physical
Virtual
Policies
Catalog
Tasks
Connectivity Templates
Find by tags
2.
Virtual Networks
Routing Zones
Routing Zone Groups
Floating IPs
Static Routes
Protocol Sessions
Remote EVPN Gateways
Virtual Infra
Endpoints
3.
Query: All
1-6 of 6
Columns (12/18)
Page Size:
25
Endpoint 1
Endpoint 2
Click for details
Name
Peer IPv4 Address
Peer IPv6 Address
ASN
Name
Peer IPv4 Address
Peer IPv6 Address
ASN
Protocol
Routing Zone
Connectivity Template
Protocol Session ID
mlag_rack_ext_0_001_leaf2
10.0.0.21/32
N/A
521
sys001
198.51.100.2/32
N/A
65533
BGP
blue
evpn_bgp_sessions
o_4iCAJ_RbKAnJnRj5g

To see details including peer configuration, click the **Protocol Session ID**.



Virtual Infrastructure

IN THIS SECTION

vCenter Virtual Infra

IN THIS SECTION

VMware vSphere Integration Overview

IN THIS SECTION

With Apstra vCenter integration, you have VM visibility of your virtualized environments. This feature helps to troubleshoot various VM connectivity issues. Inconsistencies between virtual network settings (VMware Port Groups) and physical networks (Apstra Virtual Networks) that might affect VM connectivity are flagged.

To accomplish this, the Apstra software identifies the ESX/ESXi hosts and thereby the VMs connected to Apstra-managed leaf switches. LLDP information transmitted by the ESX/ESXi hosts is used to

associate host interfaces with leaf interfaces. For this feature to work, LLDP transmit must be enabled on the VMware distributed virtual switch.

The Apstra software also connects to vCenter to collect information about VMs, ESX/ESXi hosts, port groups and VDS. Apstra extensible telemetry collectors collect this information. The collector runs in an offbox agent and uses pyVmomi to connect to vCenter. On first connect, it downloads all of the necessary information and thereafter polls vCenter every 60 seconds for new updates. The collector updates the discovered data into the Apstra Graph Datastore allowing VM queries and alerts to be raised on physical/virtual network mismatch.

Supported Versions

Apstra VMware integration is supported on the following VMware vCenter Server/vSphere versions: 8.0, 7.0U1, 6.7, 6.5.

The specific test and qualification for version 7.0 is three vCenter servers on three different routing zones: zone 1 supports 3000 VMs, zone 2 supports 1000 VMs, and zone 3 supports 1000 VMs. We support vCenter managed data center stretched clusters. vCenter segregation is based on workload, not location.

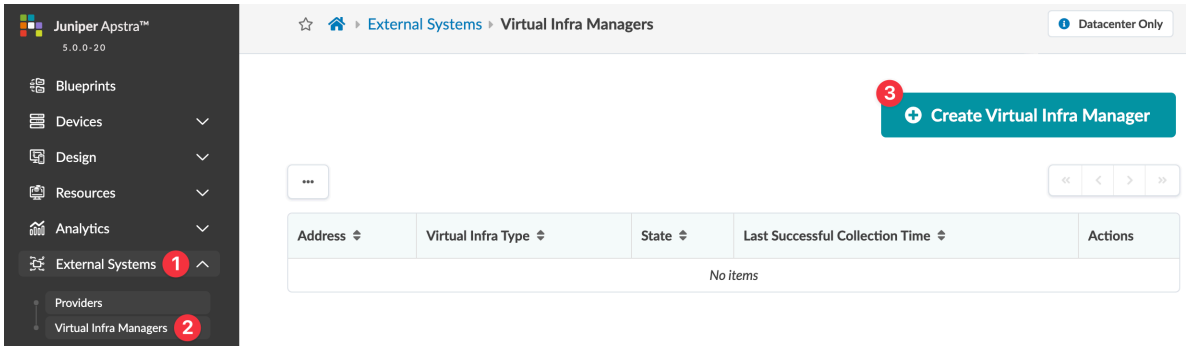
Limitations

vCenter integration does not support DVS port group with VLAN type Trunking.

Enable vCenter Integration

You only need **Read** permissions to enable vSphere Integration.

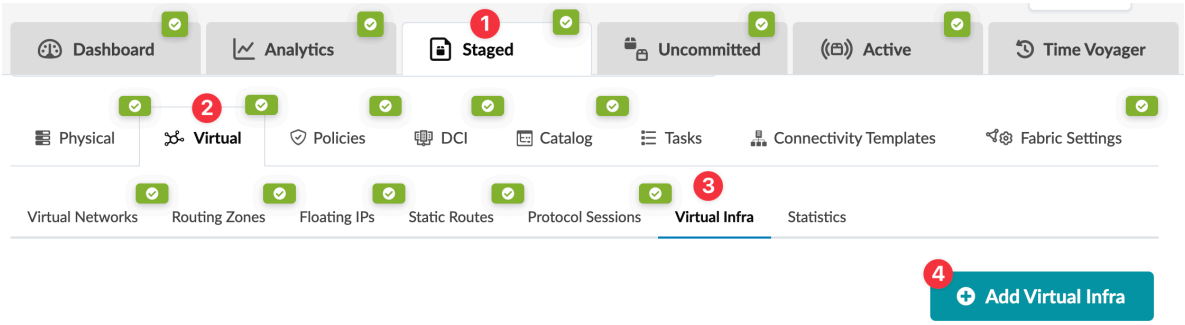
1. From the left navigation menu, navigate to **External Systems > Virtual Infra Managers** and click **Create Virtual Infra Manager**.



   The **Create Virtual Infra Manager** dialog opens.

2. Enter the vCenter IP address (or DNS name), select **VMware vCenter Server**, then enter a username and password.

3. Click **Create** to launch an offbox container running vCenter. While the container is connecting, the state is DISCONNECTED. When the container successfully connects, the state changes to CONNECTED.
4. When vCenter is connected, from the blueprint, navigate to **Staged > Virtual > Virtual Infra** and click **Add Virtual Infra**.



   The **Add Virtual Infra Manager** dialog opens.
5. Select the vCenter Server from the **Virtual Infra Manager** drop-down list, then click **Create** to stage the change.

When you're ready to deploy, commit the changes from the **Uncommitted** tab.

VM Visibility

When Apstra software manages virtual infra, you can query VMs by name. From the blueprint, navigate to **Active > Query > VMs** and enter search criteria. VMs include the following details:

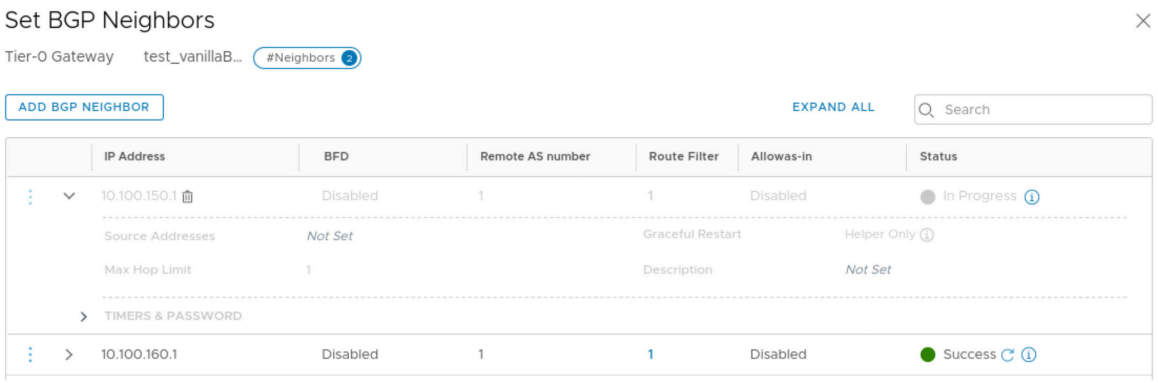
Parameter	Description
Hosted On	The ESX host that the VM is on
VM IP	The IP address as reported by vCenter after installation of VM tools. If the IP address is not available this field is empty. If the IP address is available, the VM IP address is displayed.
Leaf:Interface	The leaf and the interface ESX host is connected to
Port Group Name:VLAN ID	The VNIC's portgroup and the VLAN ID associated with the portgroup

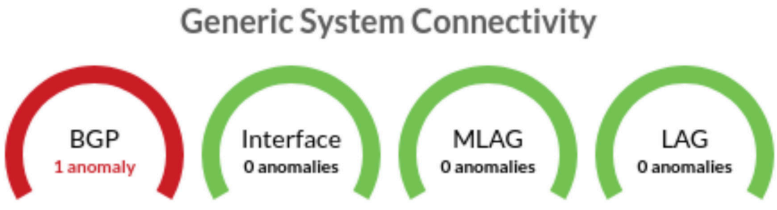
(Continued)

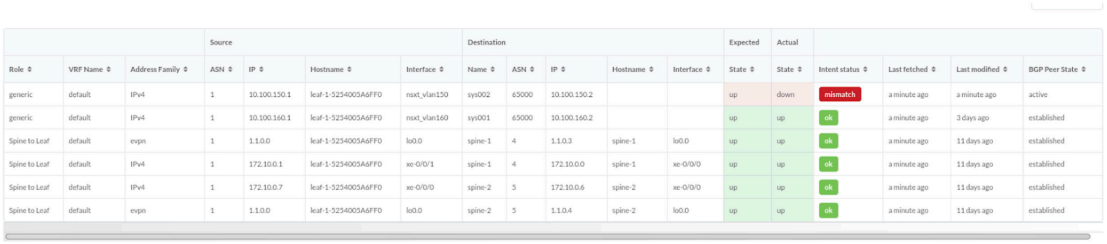
Parameter	Description
MAC Addresses	MAC address of the VNIC
Virtual Infra Address	IP address of the vCenter the VM is part of

Validate Virtual Infra Integration

You can validate virtual infra with intent-based analytics. Apstra validates BGP session towards NSX-T Edge. In case BGP neighborship in NSX-T Manager is deleted then respective anomalies can be seen in Apstra dashboard.







Two predefined analytics dashboards (as listed below) are available that instantiate predefined virtual infra probes.

Virtual Infra Fabric Health Check Dashboard

- "Hypervisor MTU Mismatch Probe" on page 1584
- "Hypervisor MTU Threshold Check Probe" on page 1584
- "Hypervisor & Fabric LAG Config Mismatch Probe" on page 1576
- "Hypervisor & Fabric VLAN Config Mismatch Probe" on page 1577
- "Hypervisor Missing LLDP Config Probe" on page 1585
- "VMs without Fabric Configured VLANs Probe" on page 1613

Virtual Infra Redundancy Check Dashboard

- "Hypervisor Redundancy Checks Probe" on page 1586

SEE ALSO

> What are Blueprint Analytics | **14**

Auto-Remediation Overview

Automatic remediation of virtual network anomalies is available without user intervention. This can reduce operational cost when network operators don't need to investigate each anomaly and check for details and intervene to mitigate anomalies. VxLAN auto-remediation is a policy configured while adding vCenter/NSX-T to a blueprint. Anomaly remediation is done in accordance with this policy.

A policy-based auto-remediation approach automatically notifies you if there is a mismatch between vSphere DPG (VMware Port Groups) and VN in a particular blueprint, or if there is a VLAN mismatch between virtual infra and the Apstra fabric, or if there is a mismatch in LAG configuration on hypervisors and the corresponding leaf ports. Apstra software provides automatic guided remediation of such anomalies.

Some of the constraints and validations that take place before the remediation happens are listed below:

- When remediation policy is set to VLAN, that is rack-local, routing zone can only be the default one.
- If VLAN ID for virtual network spanning multiple hypervisors is the same, a single layer 2 broadcast domain is assumed. For such scenarios, the VLAN remediation policy must be set to VXLAN as for missing VLAN anomalies it is checked on all the ToR leaf devices connected to different hypervisors

having virtual network with the same VLAN ID. If this is mistakenly chosen as VLAN type, validation errors are generated.

- Errors are flagged for different types of remediation policies (For example, if one is VXLAN type and other is VLAN type) are found attached to different virtual infras (such as two different vCenter servers) having the same VLAN ID in anomalies.
- If two different virtual infra servers are mapped in a blueprint and they have the same VLAN IDs then it is checked as two separate virtual networks by VXLAN auto-remediation policy.

Enable Auto-Remediation

1. From the blueprint, navigate to **Staged > Virtual > Virtual Infra** and click **Add Virtual Infra**.
2. Select the **Virtual Infra Manager** from the drop-down list.
3. Click **VLAN Remediation Policy** to see the attributes to configure.
4. Select the **VN Type** from the drop-down list.
   - **VXLAN** (inter-rack) (default) Assumes VXLAN virtual network and looks for VN mismatch in all of the related ToRs in the Apstra fabric.
   - **VLAN** (rack-local) Select VLAN if the VLAN footprint on local vSphere does not extend to other ToR leaf devices in a fabric.
5. Select the **Routing zone**. (If VN type is **rack-local** only the default routing zone is allowed.)
6. Click **Create**.

After enabling the VLAN remediation policy as inter-rack, Apstra software searches for matching local VLANs in all ToRs connecting any member host (hypervisor for example) participating in the virtual infra virtual network. If such a VN is found, it simply extends that VN to also be bound to the ToR in question with the same local VLAN. If it's not found, a new inter-rack VN is created in the specified routing zone.

Remediate Probe Anomalies

The **Remediate Anomalies** feature works in conjunction with the **Virtual Network (Single)** primitive in connectivity templates. It can't be used with the **Virtual Network (Multiple)** primitive.

Apstra policy-based remediation has the following features:

- VLAN mismatch anomalies create one virtual network for one vCenter Distributed Virtual Switch (vDS) port group that is attached to hypervisors connected to leaf ports of ToRs in Apstra fabric.
- You cannot delete a routing zone that is being referenced in remediation policy.

NOTE: For an EVPN-enabled fabric, we recommend that you have VN type as inter-rack or VXLAN in a specific routing zone.

1. From the blueprint, navigate to **Analytics > Probes** and click one of the instantiated predefined probe names.
2. Click **Remediate Anomalies** on a given stage. The Apstra software automatically updates the staged blueprint by **adding/removing/updating VN endpoints** and **VNs** to resolve the anomalies.
3. Review the staged configuration in terms of virtual network parameters, then commit the configuration. The Apstra software indicates if there are no detected changes. This could happen if you invoke remediation more than once.
4. Review and commit the changes on the **Uncommitted** tab.
5. Return to the predefined probe to view any remaining anomalies.

Disable Virtual Infra Integration

Virtual infra integrations are disabled by deleting them from the blueprint and external systems.

1. From the blueprint, navigate to **Staged > Virtual > Virtual Infra** and click the **Delete** button for the virtual infra to disable.
2. Click **Uncommitted** (top menu) and commit the deletion.
3. From the left navigation menu, navigate to **External Systems > Virtual Ingra Managers** and click the **Delete** button for the virtual infra to disable.

NSX-T Integration

IN THIS SECTION

VMware NSX-T Integration Overview

IN THIS SECTION

You can integrate NSX-T with Apstra software to help deploy fabric VLANs that are needed for deploying NSX-T in the data center or for providing connectivity between NSX-T overlay networks and fabric underlay networks. You can accelerate NSX-T deployments by making sure the fabric is ready in terms of LAG, MTU and VLAN configuration as per NSX-T transport node requirements. This feature also helps network operators with fabric visibility in terms of seeing all the NSX-T VMs, VM ports, and physical gateway ports. NSX-T integration helps identify issues on the fabric and on the virtual infrastructure. It eliminates manual config validation tasks between the NSX-T nodes side and the ToR switches.

When NSX-T VM is attached into VLAN Transport, VM query shows TOR switch/interface information together. When NSX-T VM is attached into Overlay Transport, VM query doesn't show TOR switch/interface information. Be sure to add ESXi host in generic systems, not external generic systems.

You can create Virtual Infra Managers for NSX-T Manager version 3.2.x using DVS mode. You can also add multiple Virtual Infra Managers per blueprint. This is useful when you have multiple NSX-T Managers or multiple vCenter Servers hosted in the same fabric blueprint. You'll need to provide the vCenter compute managers information (address and credentials) when you add the NSX-T Virtual Infra.

Supported Version in 5.0.0

VMware NSX/NSX-T Manager version 4.1.X

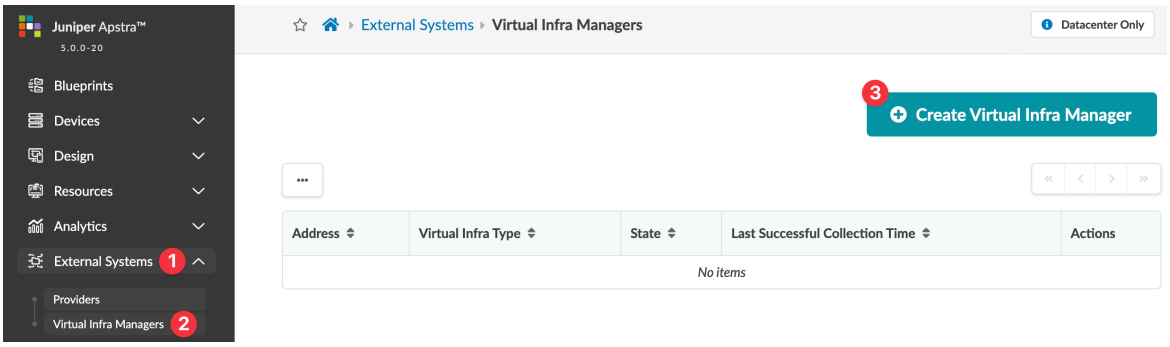
Supported Configurations

- vCenter as virtual manager in one blueprint
- Standalone NSX-T manager as virtual manager with vCenters added to it in one blueprint
- NSX-T Edge VM migration is supported only within a rack. Attempting to migrate between racks results in BGP disruption. You can migrate the NSX-T Edge VM from the ESXi host connected to leaf pair (that is, ToR-Leaf and ToR-Right) to the other ESXi host which is connected to single leaf with the rack.

Enable NSX-T Integration

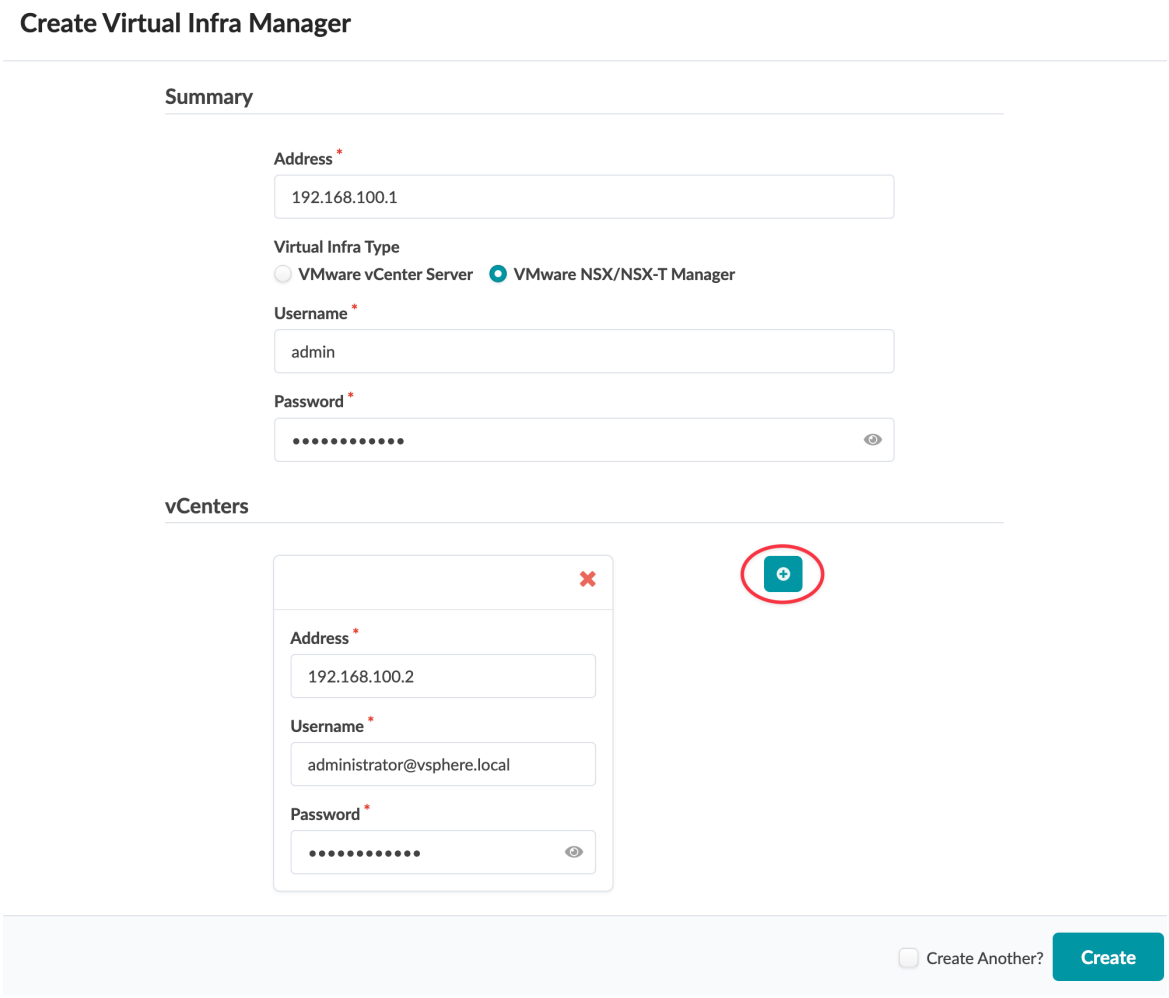
We recommend that you "create a user profile" on page 1273 dedicated to managing NSX-T integration activities.

1. From the left navigation menu, navigate to **External Systems > Virtual Infra Managers** and click **Create Virtual Infra Manager**.

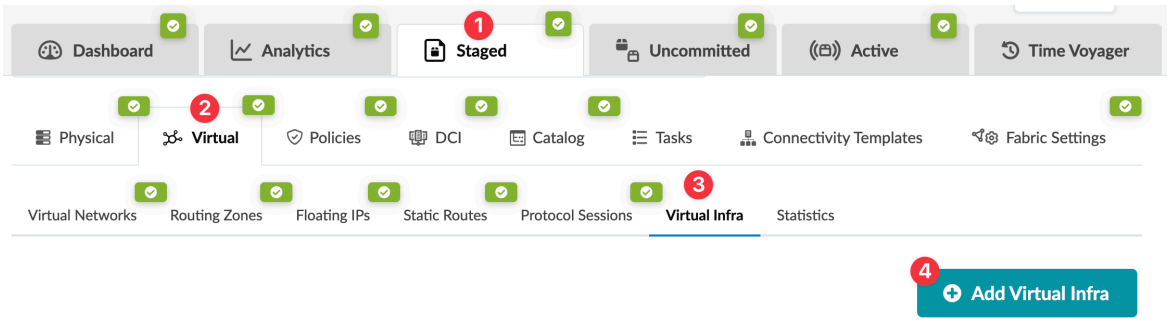


   The **Create Virtual Infra Manager** dialog opens.

2. Enter the NSX/NSX-T manager IP address (or DNS name), select **VMware NSX/NSX-T Manager** for **Virtual Infra Type** and enter a username and password.

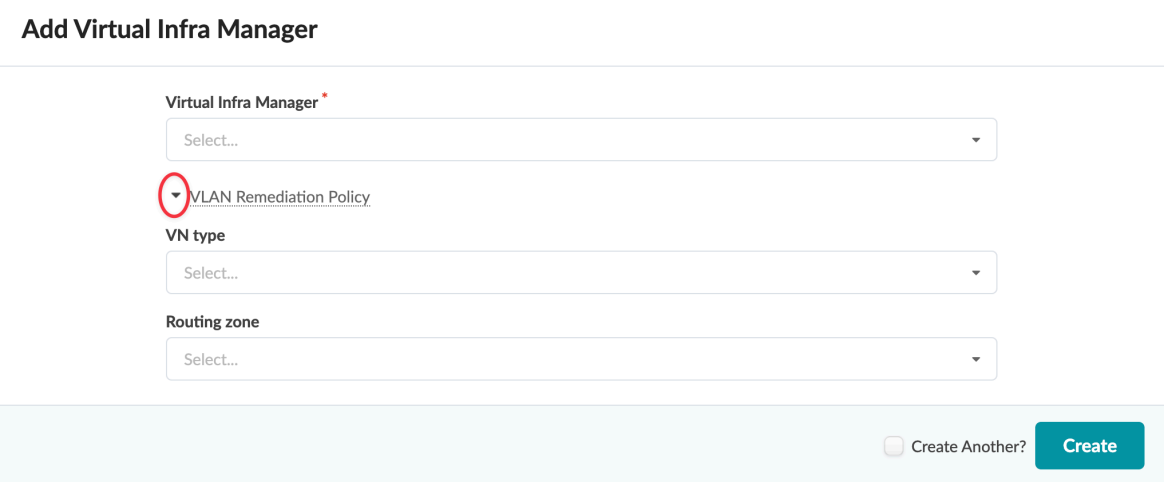


3. Click the plus sign in the **vCenters** section and enter the IP address, username and password. To add another vCenter, click the plus sign again.
4. Click **Create** to create the virtual infra manager and return to the table view.
When the connection is successful, the connection state changes from DISCONNECTED to CONNECTED.
5. When NSX-T is connected, from the blueprint, navigate to **Staged > Virtual > Virtual Infra** and click **Add Virtual Infra**.



The **Add Virtual Infra Manager** dialog opens.

6. Select the NSX-T manager from the **Virtual Infra Manager** drop-down list, then click **VLAN Remediation Policy** to expose additional fields.



7. Select the VN type and routing zone.
   - If you select VLAN (rack-local), you must use the default routing zone.
   - If you select VXLAN (inter-rack - when VN extends to different ToRs in the fabric) you can select a different routing zone.

   The information entered above is used in Intent-based analytics (IBA) probes that can remediate anomalies.
8. Click **Create** to stage the virtual infra manager and return to the table view. The new virtual infra manager appears in the table.
9. Click **Uncommitted** (top menu) to review changes, then click **Commit** (top-right) to add the NSX-T manager to the active blueprint.
10. Create a **Routing Zone** in the blueprint and specify the **VLAN ID**, **VNI** and **Routing Policies**. Routing Zone maps to a VRF on which BGP peering towards NSX-T Edge node is established.

Create Routing Zone

VRF Name *

NSX_VRF

VRF Description

VLAN ID

111

VNI

11011

Routing Policies

imp_all

Name	imp_all
Description	
Import Policy	All
Extra Import Routes	Not provided
Spine Leaf Links	yes
Spine Superspine Links	no
L2 Edge Subnets	yes
Loopbacks	yes
Static Routes	yes
Extra Export Routes	Not provided
Aggregate Prefixes	Not provided
Expect Default IPv4 Route	yes
Expect Default IPv6 Route	yes

Route Target Policies

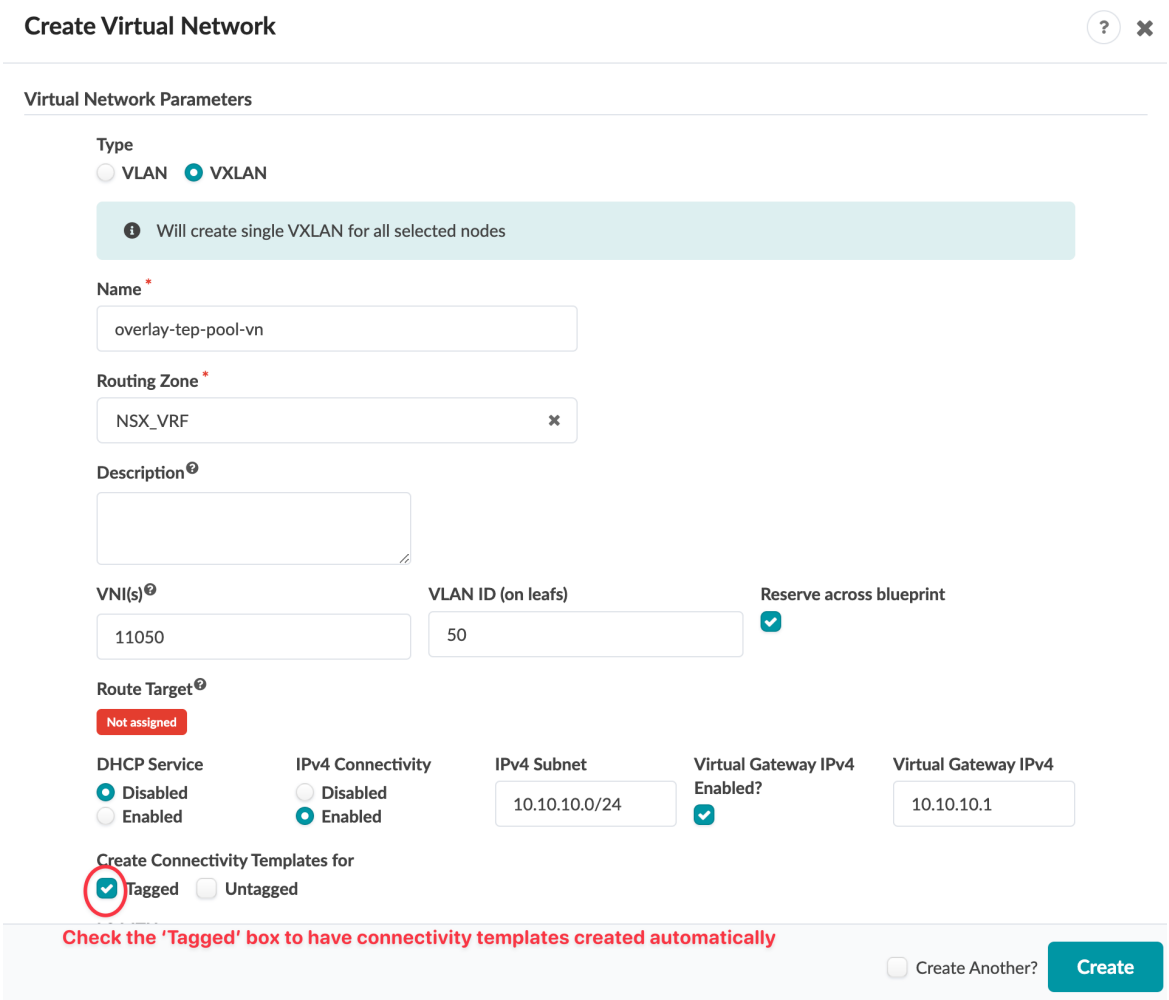
Import Route Targets

Add Import Route Target

Export Route Targets

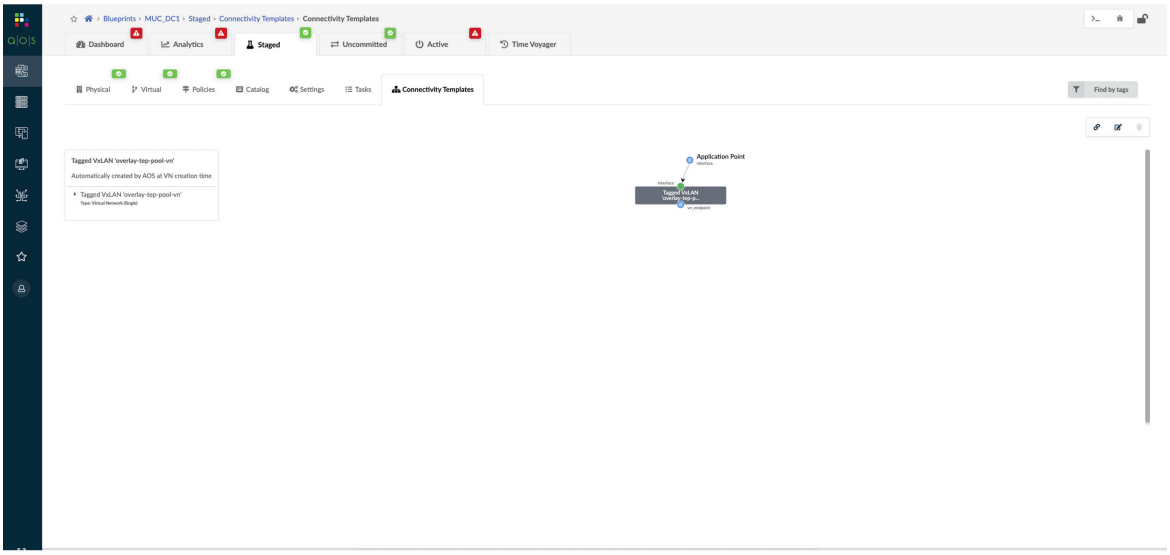
Add Export Route Target

11. For the GENEVE Tunnels to come up between the Transport Nodes in NSX-T, connectivity must be established via Juniper Apstra Fabric. This will be ensured by creating VXLAN VN in Apstra and assigning correct port mapping in ToR leaf devices towards Transport Node. VLAN ID for Overlay VXLAN VN defined in Apstra must match the one mapped in Overlay Profile in NSX-T for Transport Nodes. Also, the same IP subnet as that of the TEP Pool in NSX will be used.

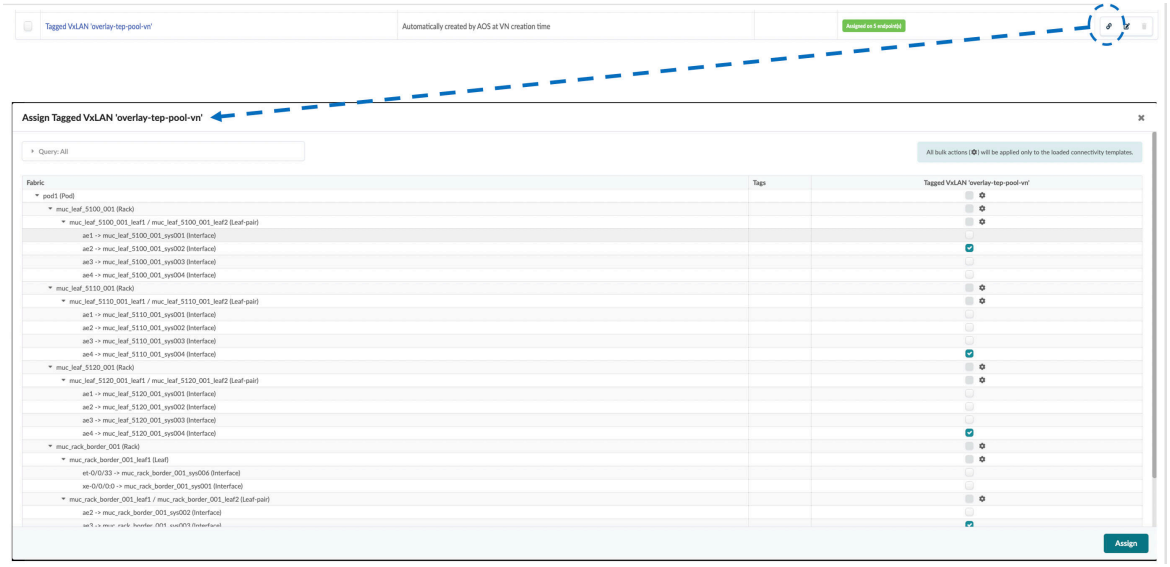


12. Since we checked the box to **Create Connectivity Template for** in last step during VXLAN VN creation in Apstra a **Connectivity Template** of type **Virtual Network** is automatically created under

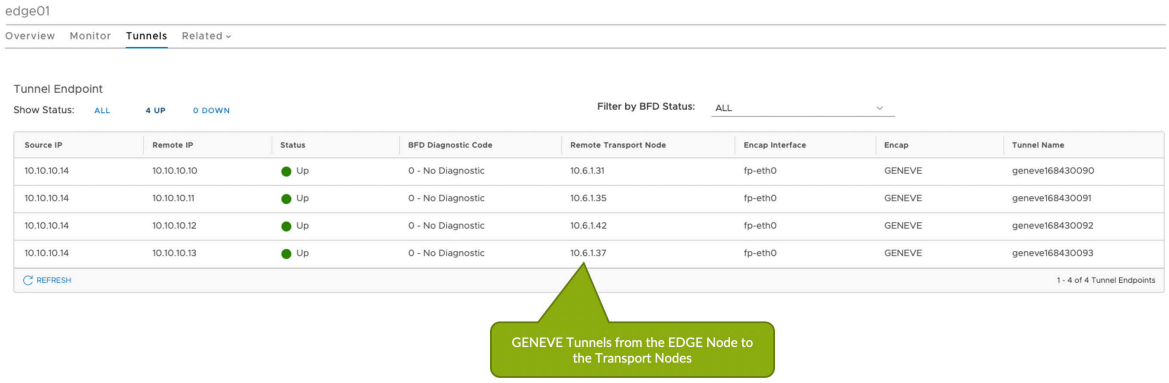
Blueprints > Staged > Connectivity Templates as shown below:



13. Assign the interfaces to the **Connectivity Template** created above towards Transport nodes in NSX-T side.



14. Once the configuration is rendered towards devices we can observe GENEVE Tunnels between Transport and Edge nodes are UP in NSX-T Manager.

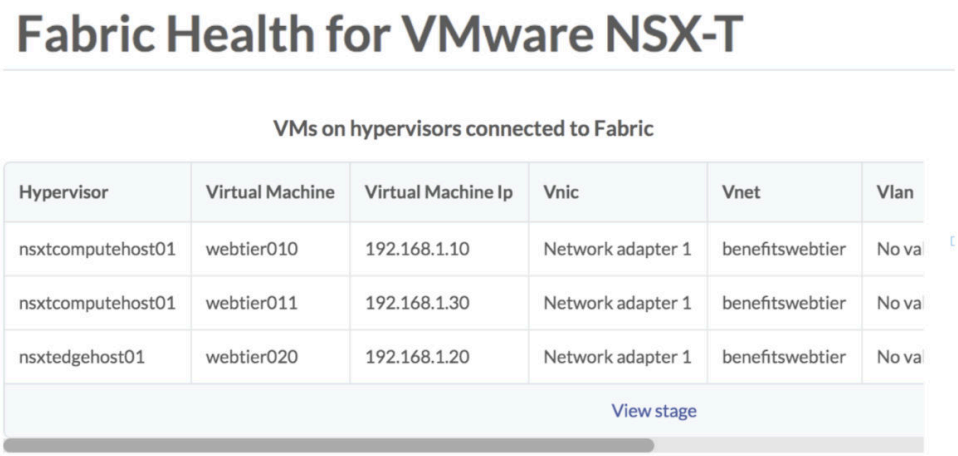


NOTE: When you install the NSX Edge as a virtual appliance or host Transport Node, use the default uplink profile. If the Failover teaming policy is configured for an uplink profile, then you can only configure a single active uplink in the teaming policy. Standby uplinks are not supported and must not be configured in the failover teaming policy.

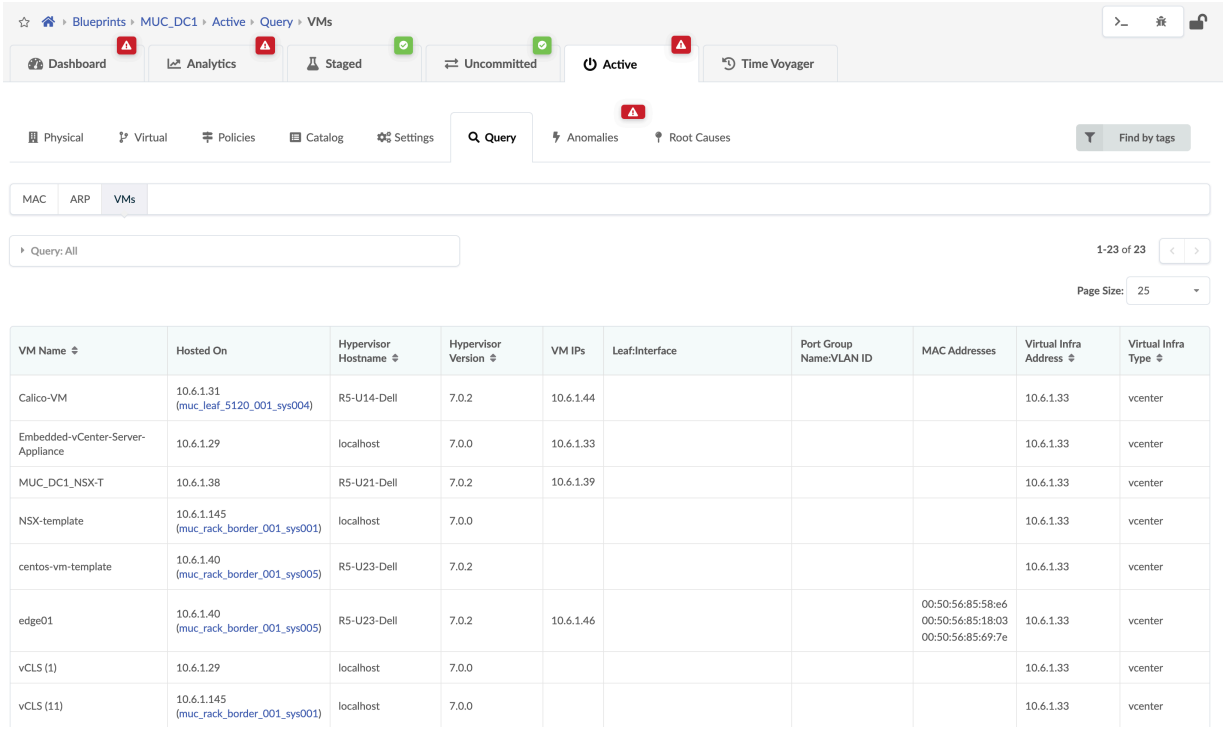
Virtual Infrastructure Visibility

When you've successfully integrated NSX-T, you have visibility of NSX-T VMs and transport nodes in the virtual infrastructure. You can query the status of the VMware fabric health.

To see a list of the VMs connected to the hypervisor, navigate to the dashboard and scroll to fabric health for VMware option.



You can also query VMs that are hosted on hypervisors connected to ToR leaf devices. From the blueprint, navigate to **Active > Query > VMs**.



VM Name	Hosted On	Hypervisor Hostname	Hypervisor Version	VM IPs	Leaf:Interface	Port Group Name:VLAN ID	MAC Addresses	Virtual Infra Address	Virtual Infra Type
Calico-VM	10.6.1.31 (muc_leaf_5120_001_sys004)	R5-U14-Dell	7.0.2	10.6.1.44				10.6.1.33	vcenter
Embedded-vCenter-Server-Appliance	10.6.1.29	localhost	7.0.0	10.6.1.33				10.6.1.33	vcenter
MUC_DC1_NSX-T	10.6.1.38	R5-U21-Dell	7.0.2	10.6.1.39				10.6.1.33	vcenter
NSX-template	10.6.1.145 (muc_rack_border_001_sys001)	localhost	7.0.0					10.6.1.33	vcenter
centos-vm-template	10.6.1.40 (muc_rack_border_001_sys005)	R5-U23-Dell	7.0.2					10.6.1.33	vcenter
edge01	10.6.1.40 (muc_rack_border_001_sys005)	R5-U23-Dell	7.0.2	10.6.1.46			00:50:56:85:58:e6 00:50:56:85:18:03 00:50:56:85:69:7e	10.6.1.33	vcenter
vCLS (1)	10.6.1.29	localhost	7.0.0					10.6.1.33	vcenter
vCLS (11)	10.6.1.145 (muc_rack_border_001_sys001)	localhost	7.0.0					10.6.1.33	vcenter

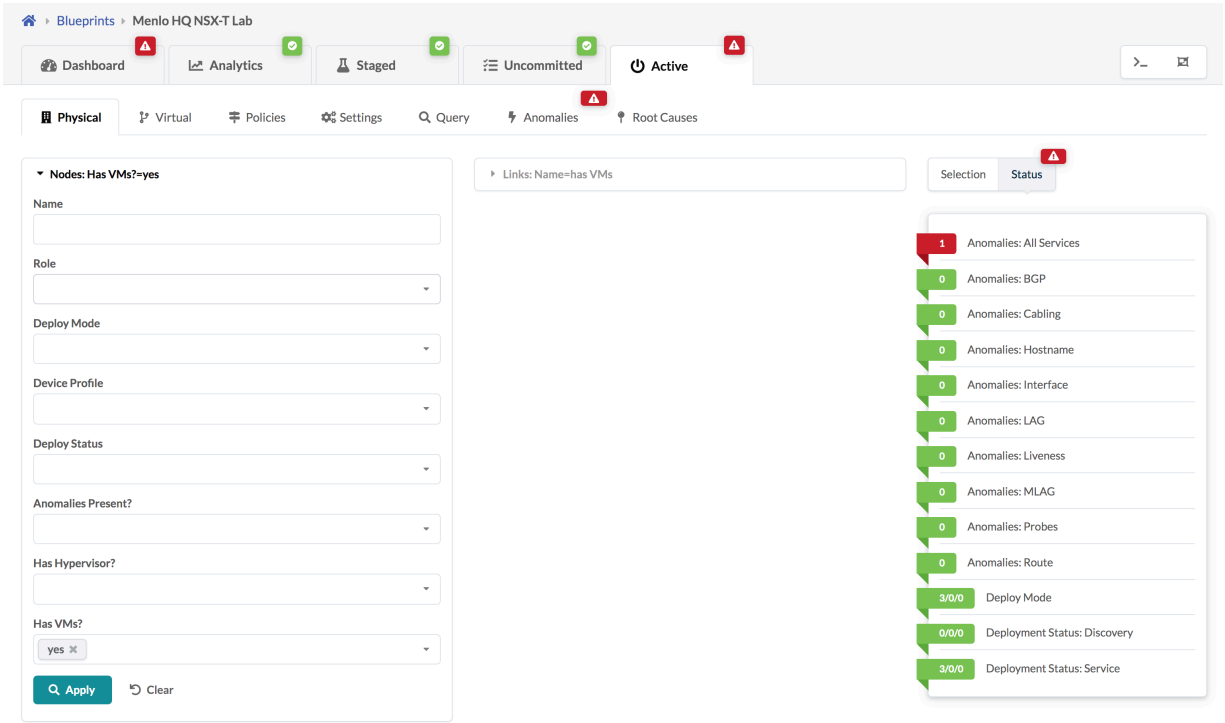
VMs include the following details:

Parameter	Description
VM Name	The Virtual Machine name which is hosted on NSX managed hypervisor.
Hosted On	The ESXi host on which Virtual Machine is hosted.
Hypervisor Hostname	The hypervisor hostname on which Virtual Machine is hosted and is connected to the leaf TORs in a fabric.
Hypervisor Version	The software version of OS running on the hypervisor.
VM IP	The IP address as reported by NSX-T after the installation of VM tools. If the IP address is not available this field is empty. Apstra displays VM IP if the IP address is available on installation VM tools on the VM.

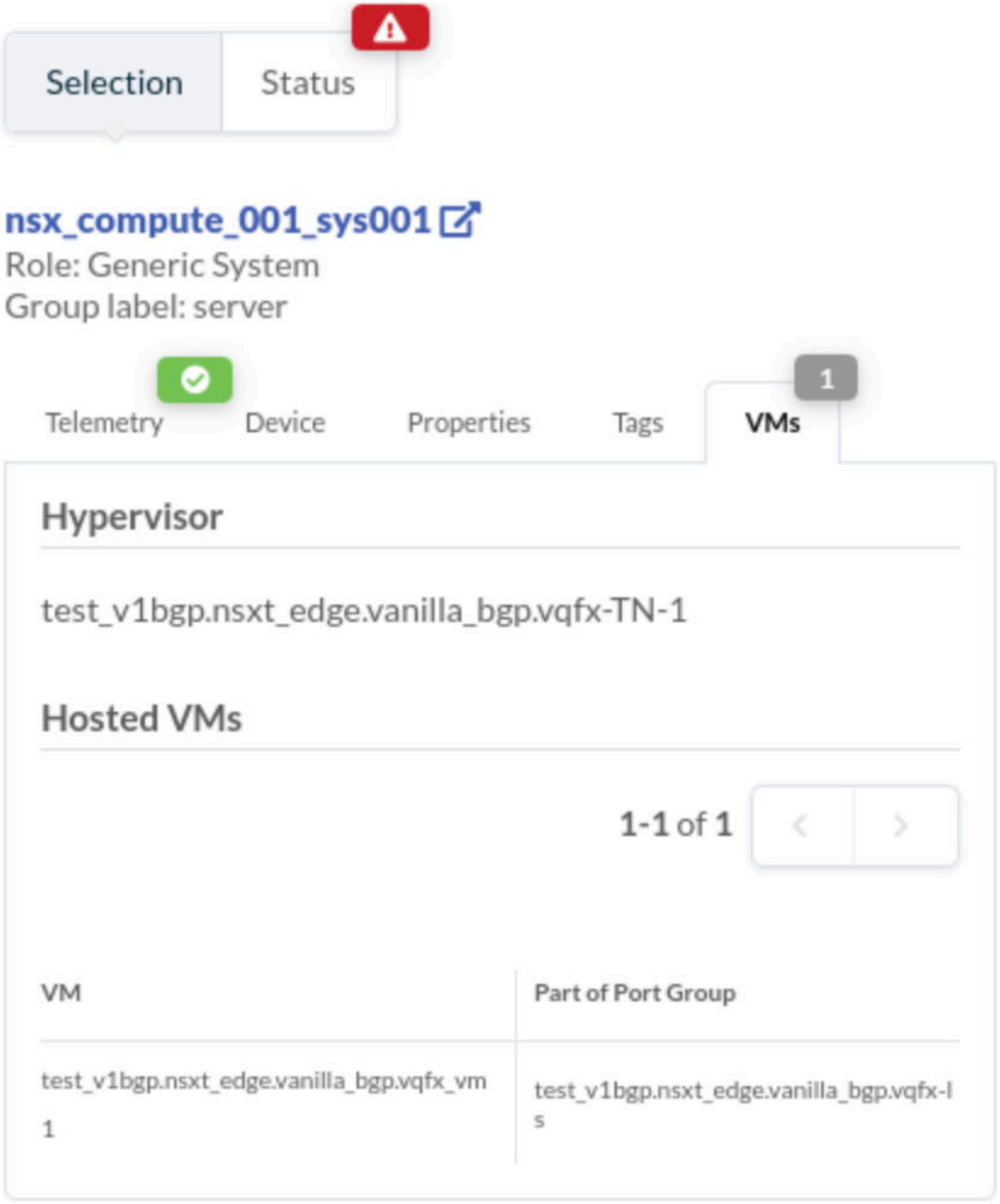
(Continued)

Parameter	Description
Leaf:Interface	System ID for the interface on the leaf to which ESXi host is connected and on which VM resides.
Port Group Name:VLAN ID	The VLAN ID which NSX-T port groups are using. Overlay VM to VM traffic in a NSX-T enabled Data Center tunnels between transport nodes over this Virtual network.
MAC Addresses	MAC address of the VM connected to the Apstra Fabric.
Virtual Infra address	IP address of the NSX-T infra added to a Blueprint

To search for nodes in the physical topology that have VMs, navigate to **Active > Physical** and select **Has VMs?** from the **Nodes** drop-down list.

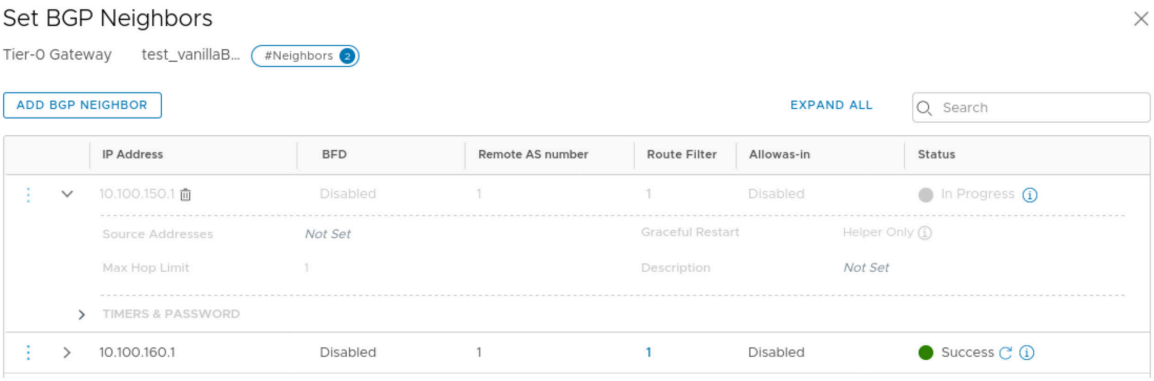


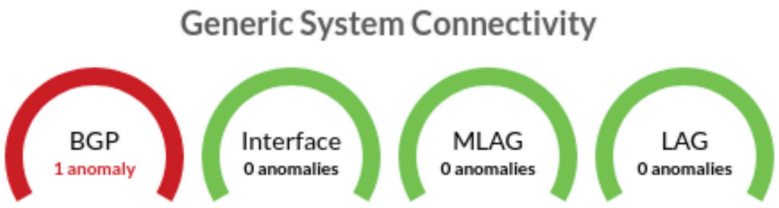
If the VM is moved from one Transport node to another in NSX-T it can be visualized in Apstra under **Active > Physical > Nodes > Generic System (Node_name)**. Select the **VMs** tab as shown below:

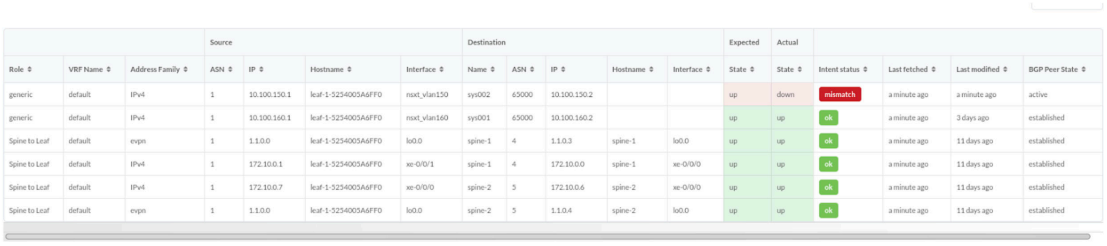


Validate Virtual Infra Integration

You can validate virtual infra with intent-based analytics. Apstra validates BGP session towards NSX-T Edge. If BGP neighborship in NSX-T Manager is deleted, then respective anomalies are displayed in the Apstra dashboard.







Two predefined analytics dashboards (as listed below) are available that instantiate predefined virtual infra probes.

Virtual Infra Fabric Health Check Dashboard

- "Hypervisor MTU Mismatch Probe" on page 1584
- "Hypervisor MTU Threshold Check Probe" on page 1584
- "Hypervisor & Fabric LAG Config Mismatch Probe" on page 1576
- "Hypervisor & Fabric VLAN Config Mismatch Probe" on page 1577
- "Hypervisor Missing LLDP Config Probe" on page 1585
- "VMs without Fabric Configured VLANs Probe" on page 1613

Virtual Infra Redundancy Check Dashboard

- "Hypervisor Redundancy Checks Probe" on page 1586

SEE ALSO

What are Blueprint Analytics | **14**

Disable Virtual Infra Integration

To disable virtual infra integrations, delete them from the blueprint and external systems.

1. From the blueprint, navigate to **Staged > Virtual > Virtual Infra** and click the **Delete** button for the virtual infra to disable.
2. Click **Uncommitted** (top menu) and commit the deletion.
3. From the left navigation menu, navigate to **External Systems > Virtual Infra Managers** and click the **Delete** button for the virtual infra to disable.

NSX-T Edge and Connectivity Templates

IN THIS SECTION

Overview

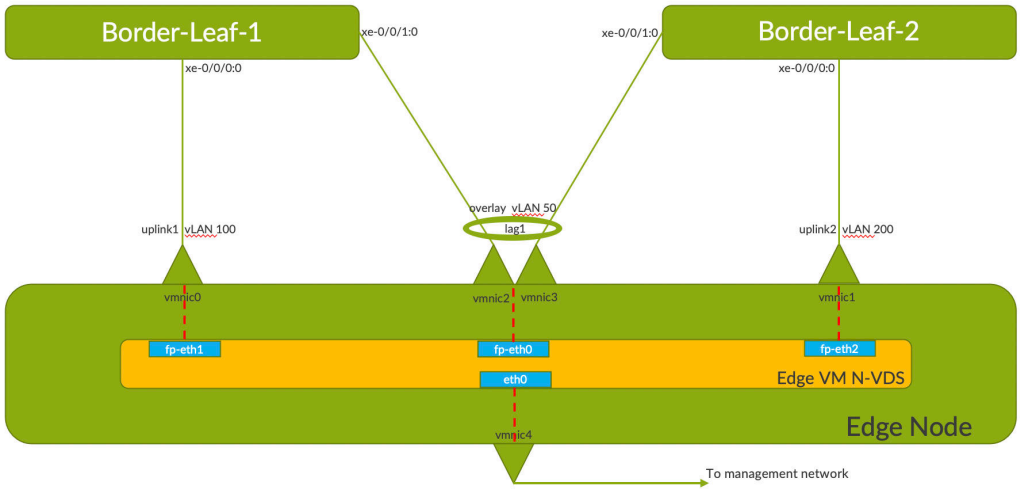
Juniper Apstra supports NSX-T Edge connectivity requirements using connectivity templates. Connectivity templates can be used both where NSX-T Edge is hosted on Bare Metal or when used as a virtual machine.

We support VRF lite enabled Tier-0 edge Gateway using connectivity templates.

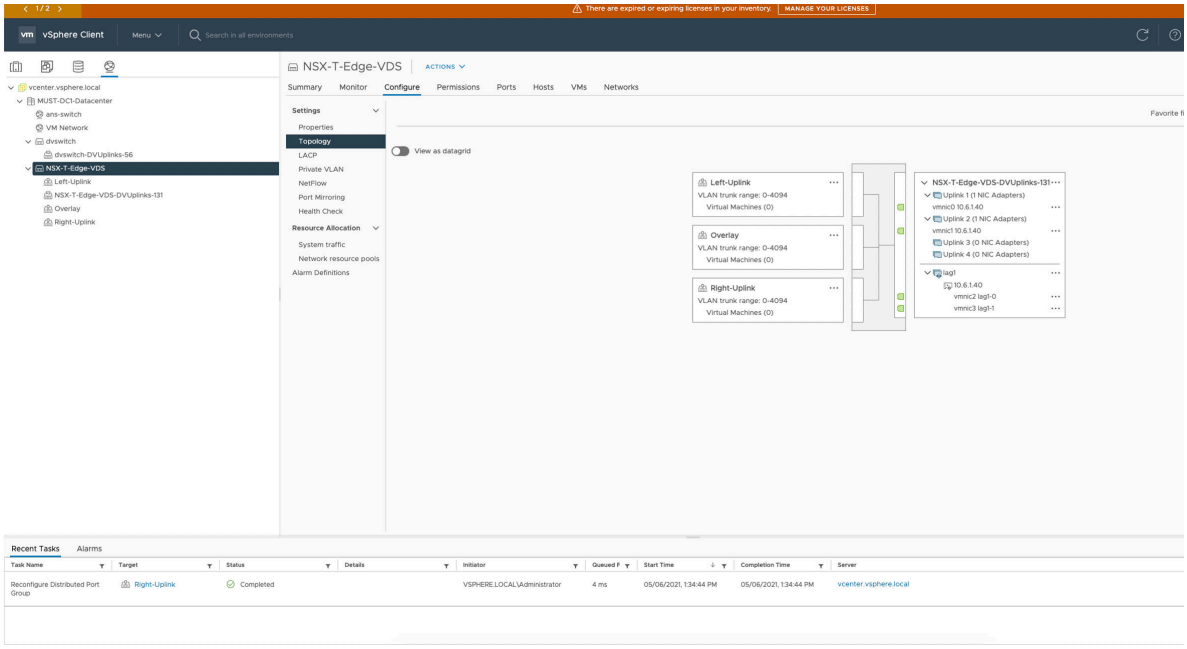
The use cases below relate to connectivity templates for NSX-T 3.0 Edge:

Set Up NSX-T Tier-0 Router BGP peering

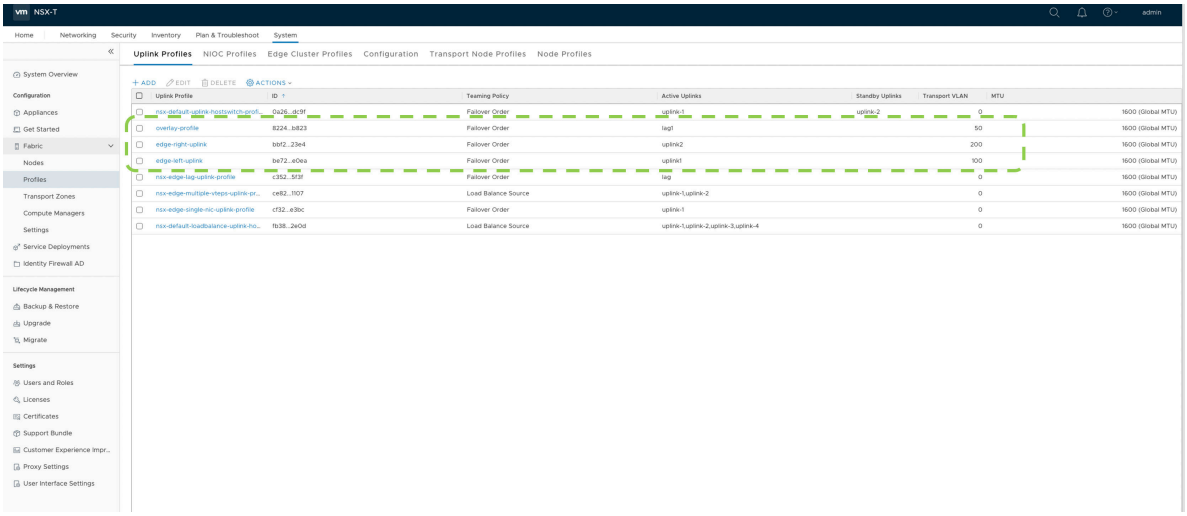
Let's say NSX-T Edge VM uplinks are connected to ToR leaf devices via VLAN Transport Zone which provides uplink network connectivity to physical infrastructure. Then Edge VM will also have vmnics as per below screenshot which will help for tunnelling traffic between Transport Nodes. This is called Overlay Transport Zone.



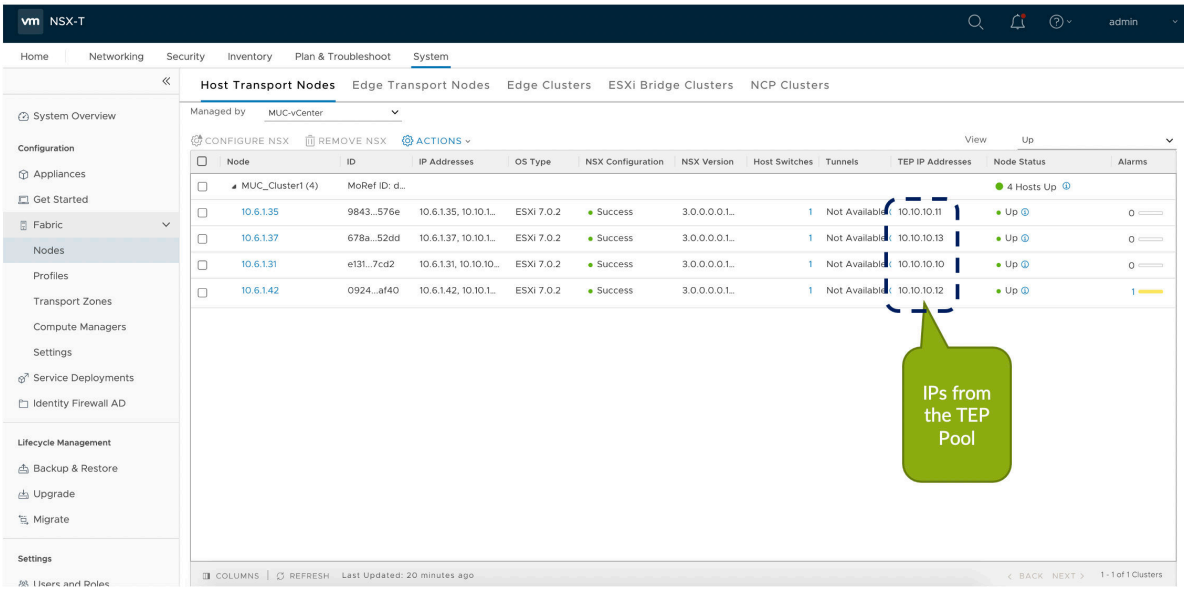
- Create three Distributed Port Groups for respective vmnics and VLAN Trunking to be enabled on all the Nodes as per the networking depicted in previous screenshot.



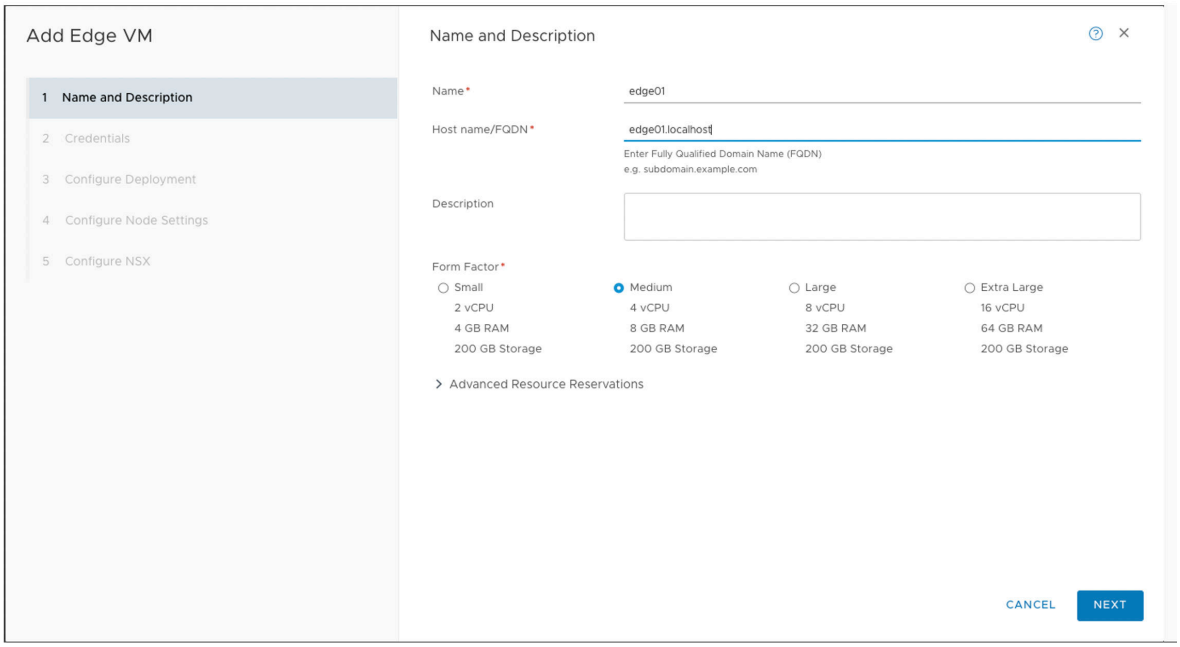
- Create respective Uplink profiles for Overlay and VLAN Transport Zones in NSX Manager(UI).

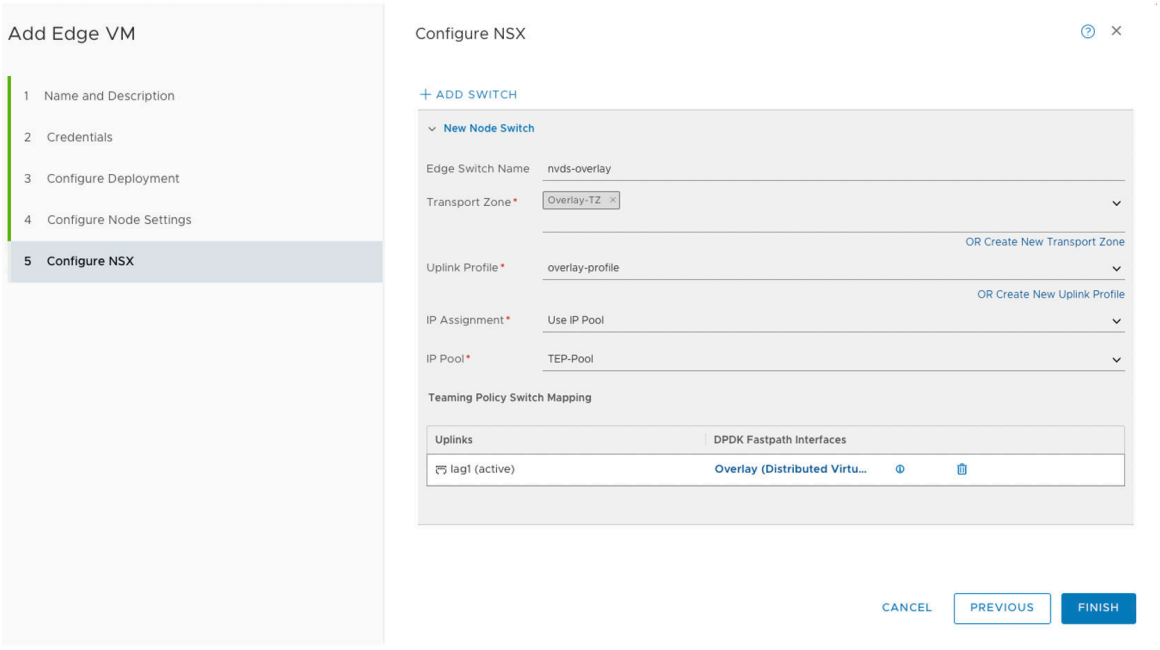


- After NSX-T is configured on the Transport nodes, a Tunnel endpoint(TEP) IP pool is created in the NSX UI as below:



- Now create the NSX-T Edge VM in NSX Manager UI as below. It is used as the device for north-south communication and BGP peering with Juniper Apstra Fabric. Also configure VDS on the Edge Nodes under NSX Manager(UI) for respective overlay and Uplink interfaces.

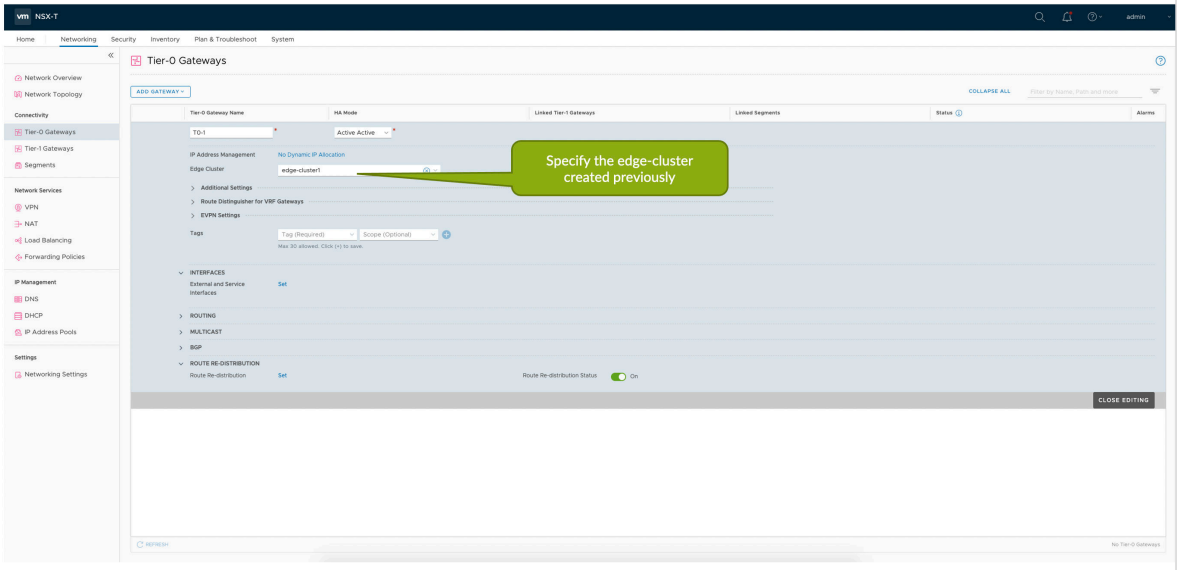




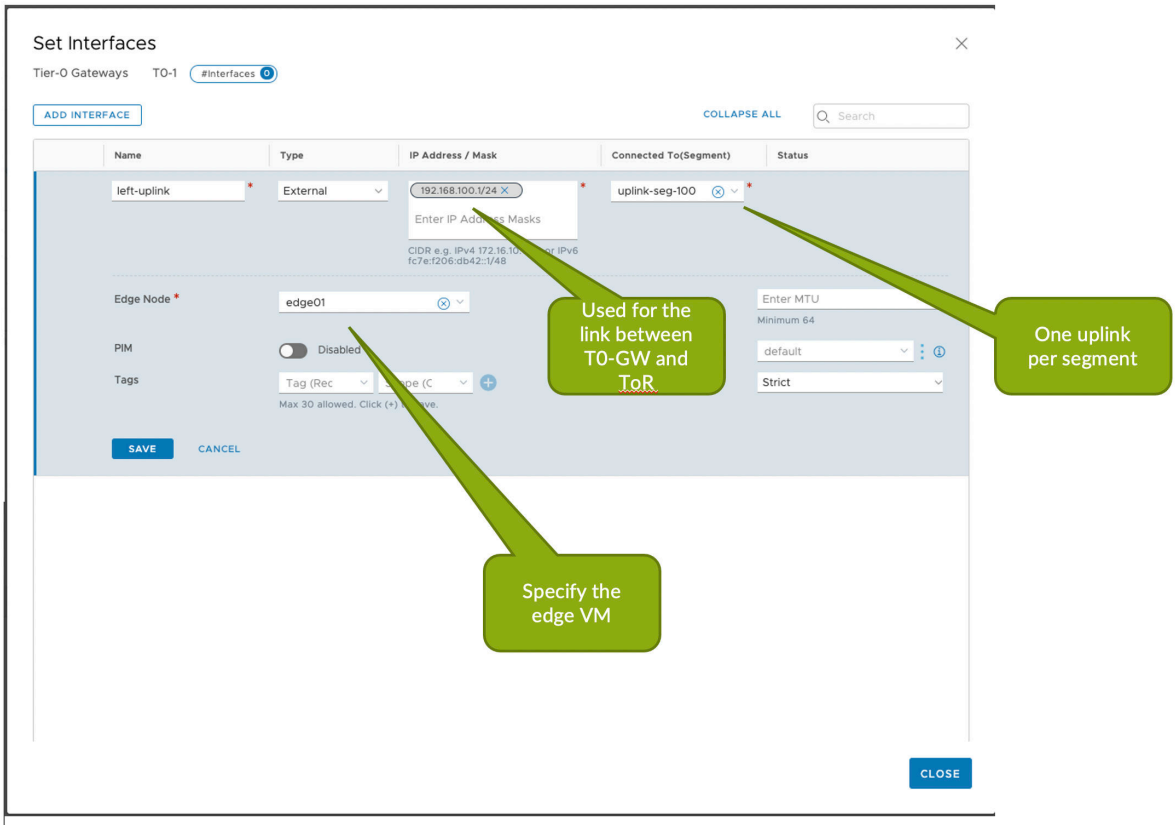
Overlay

- Tier-0 Gateway in the NSX-T Edge cluster provides a gateway service between the logical and physical network. In NSX Manager create T0 Gateway which connects to the ToR Leaf via BGP to

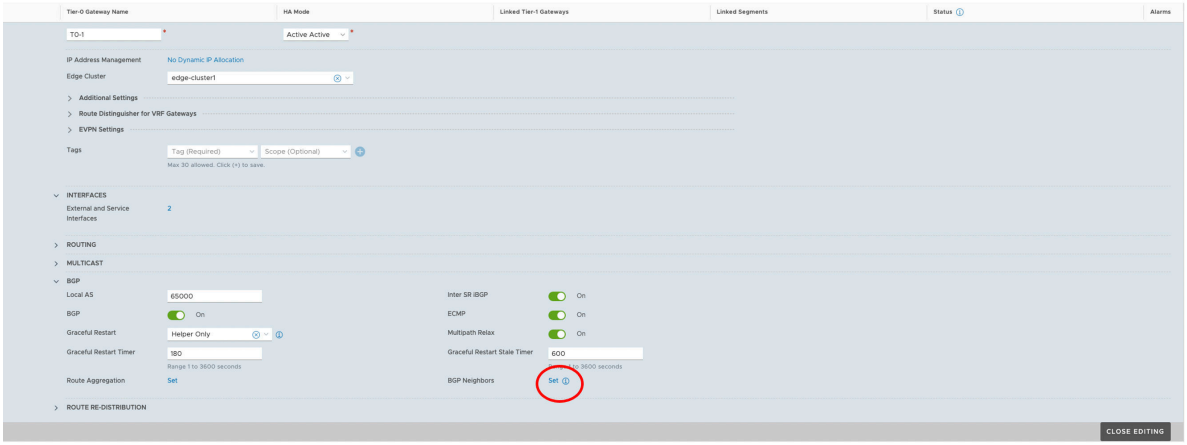
communicate with the rest of Juniper Apstra Fabric.



- Add External interfaces to the T0 GW which maps to the Uplink segments

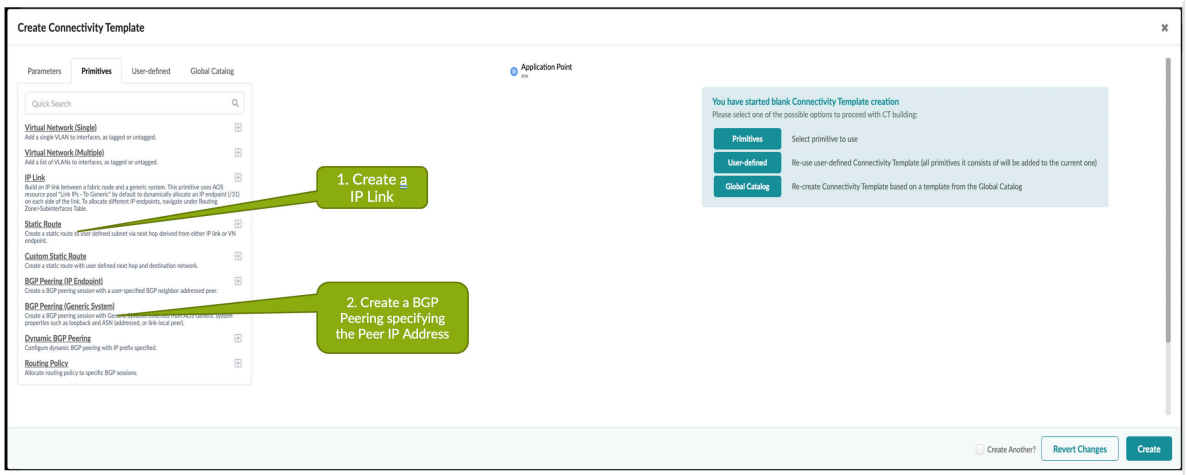


- Configure BGP peering on NSX T0 GW towards Juniper Apstra Fabric in NSX Manager.

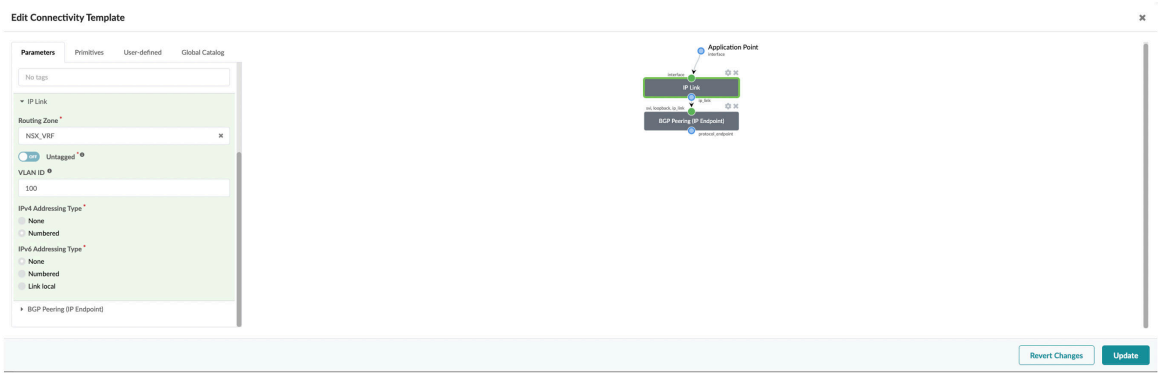


- For NSX-T integration with Juniper Apstra, see "NSX-T Integration" on page 312

  First create a **Routing Zone** in Juniper Apstra UI which maps to a VRF. Then need to setup **IP Link Primitive** based connectivity template to establish BGP peering from the NSX-T Edge node to Fabric as below:



  Specify the routing zone on which the IP link will be added and respective VLAN ID.

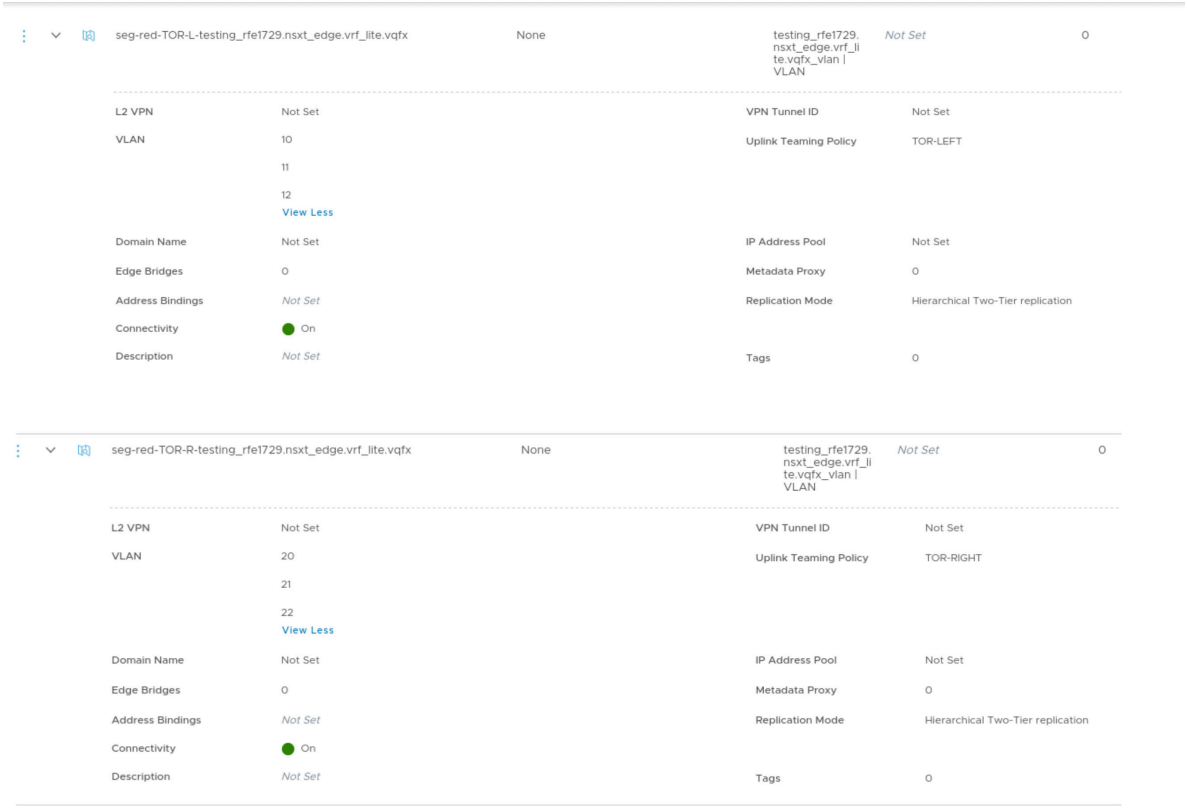


Set Up NSX-T VRF Lite

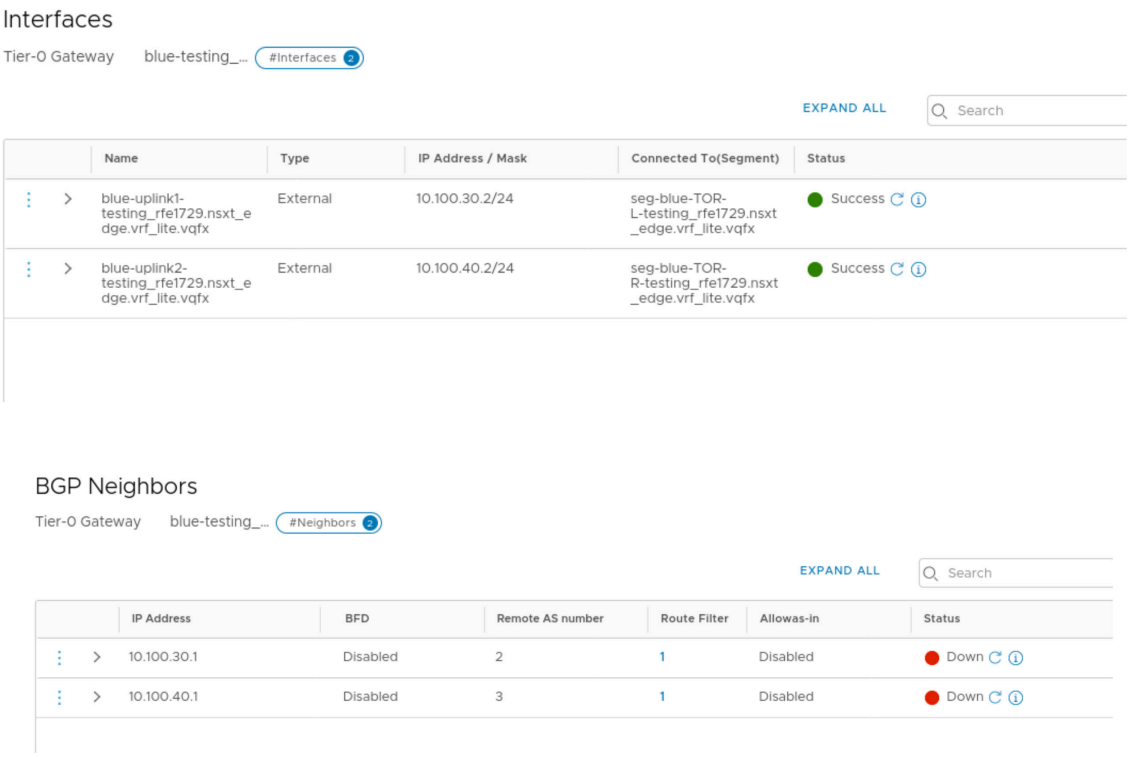
With NSX-T VRF Lite we are able to configure per tenant data isolation. Each VLAN can be considered as a separate channel for data plane under VRF gateways.

BGP peering can be built over these VLANs in VRF gateways for route exchange with the upstream Juniper Apstra fabric. Inter-VRF traffic is routed through the physical Juniper Apstra fabric.

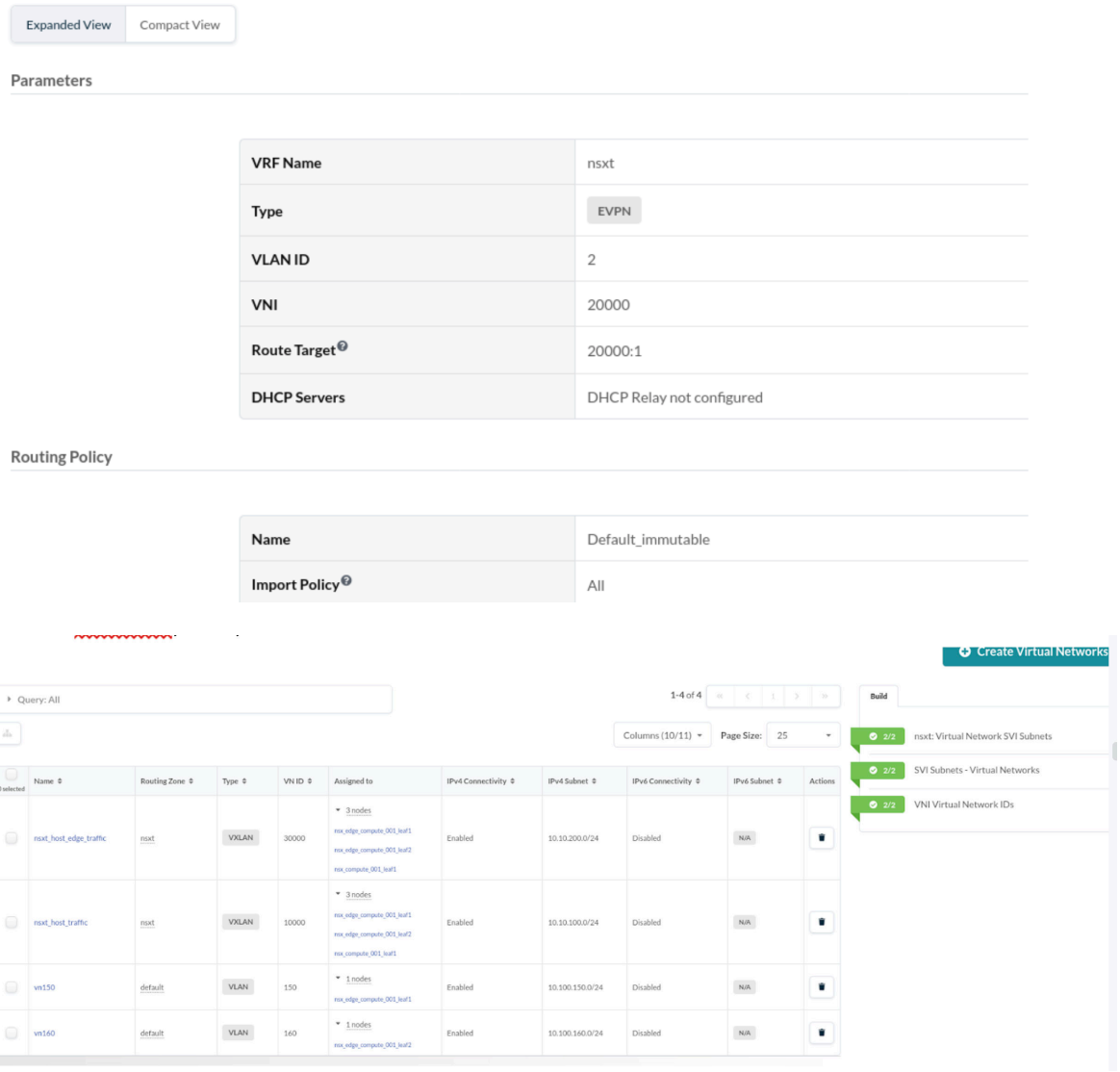
- In NSX-T Manager create the VLAN Segments for the Uplink networks for the tenants.



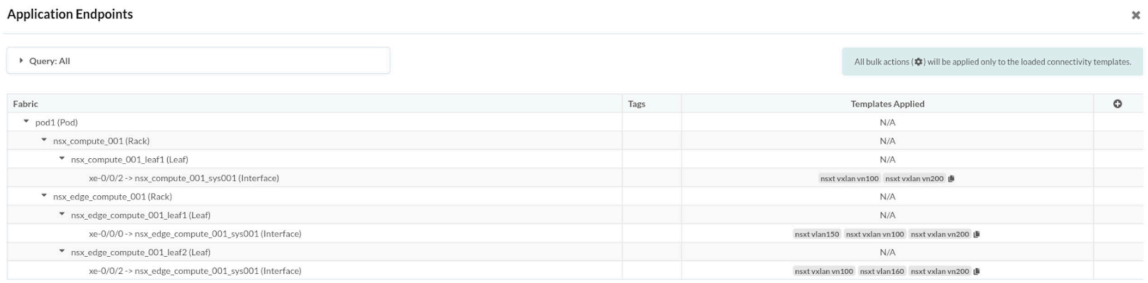
- In NSX-T Manager create the VRF-enabled Tier-0 Gateway for the tenants and add the uplink interfaces on the VRF enabled Gateways. Thereafter add the BGP neighbors.



- From the Apstra GUI, setup the **Routing Zone** and the respective VNs on which BGP session will be established towards ToR leaf devices as below:



- Create connectivity template under Staged option for the VNs created before and assign the respective interfaces towards NSX-T Edge VM.



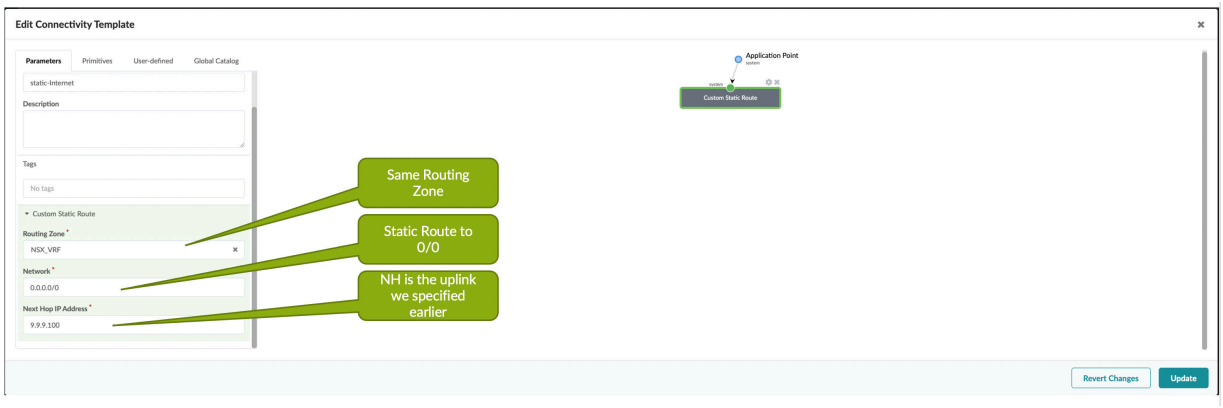
Type	Action	Name
Connectivity Point	ADDED	nsxt vlan150
Connectivity Point	ADDED	nsxt vlan160
Connectivity Point	ADDED	nsxt vxlan vn100
Connectivity Point	ADDED	nsxt vxlan vn200
Virtual Network	CHANGED	vn160
Virtual Network	CHANGED	nsxt_host_edge_traffic
Virtual Network	CHANGED	nsxt_host_traffic
Virtual Network	CHANGED	vn150

Set Up Default Static Route towards NSX-T Edge

Static default could be required in NSX-T edge setup to provide Internet connectivity. It can be taken care of by adding a default route(0/0) with the next hop pointing towards uplink ToR leaf using a connectivity template.

In the connectivity templates, assign the correct uplink:

Navigate to **Staged > Connectivity Templates > Add Template > Primitives > Custom Static Route** to inject default route:

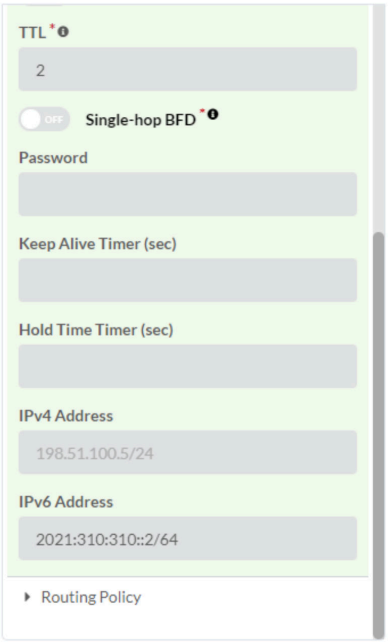


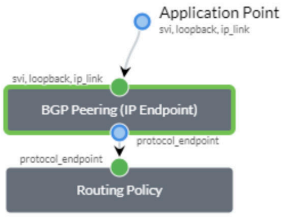
Set Up BGP IPv6 towards NSX-T Edge

We can enable IPv6-based BGP neighborship between T0 Gateway and ToR leaf using connectivity templates.

See "Set up NSX-T VRF Lite" section for details on creating uplink VLAN interfaces on T0 Gateway. This VLAN should be IPv6-enabled.

Create a connectivity template for each of the VXLAN VN and enable BGP towards IPv6 neighbor on NSX-T Edge as below:





Un-assign BGP on VXLAN VN towards NSX-T Edge

Let's say BGP neighborship from Tier-0 Gateway in NSX-T needs to be torn down towards ToR Leaf. In this case we need to unassign the interfaces in the **Virtual Network** based Connectivity Template used for BGP peering so that it is in the **Ready** state, and then delete the connectivity template:

Type	Action	Name
Floating IPs	REMOVED	5362661d-6e64-41c3-937b-ac974f20b5c0
Protocol Sessions	REMOVED	9005b70b-67e5-4db1-aa4c-70e408614726
Virtual Network	CHANGED	nsxt_vlan150

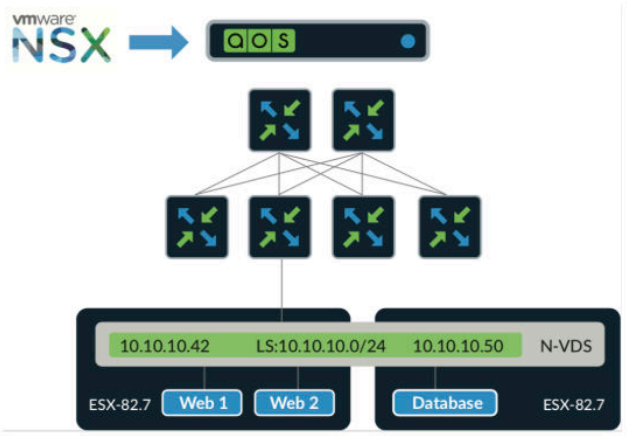
NSX-T Inventory Mapping to Apstra Virtual Infrastructure

IN THIS SECTION

Overview

Apstra software can connect to the NSX-T API to gather information about the inventory in terms of hosts, clusters, VMs, portgroups, vDS/N-vDS, and NICs within the NSX-T environment. Apstra can integrate with NSX-T to provide Apstra admins visibility into the application workloads (aka VMs) running and alert them about any inconsistencies that would affect workload connectivity. **Apstra Virtual Infrastructure visibility** helps provide underlay/overlay correlation visibility and use IBA analytics for overlay/underlay.

You cannot view the NSX Inventory in Apstra until the NSX-T manager is associated to a blueprint.



As per above screenshot inventory collection for NSX-T is done via Apstra extensible telemetry collector.

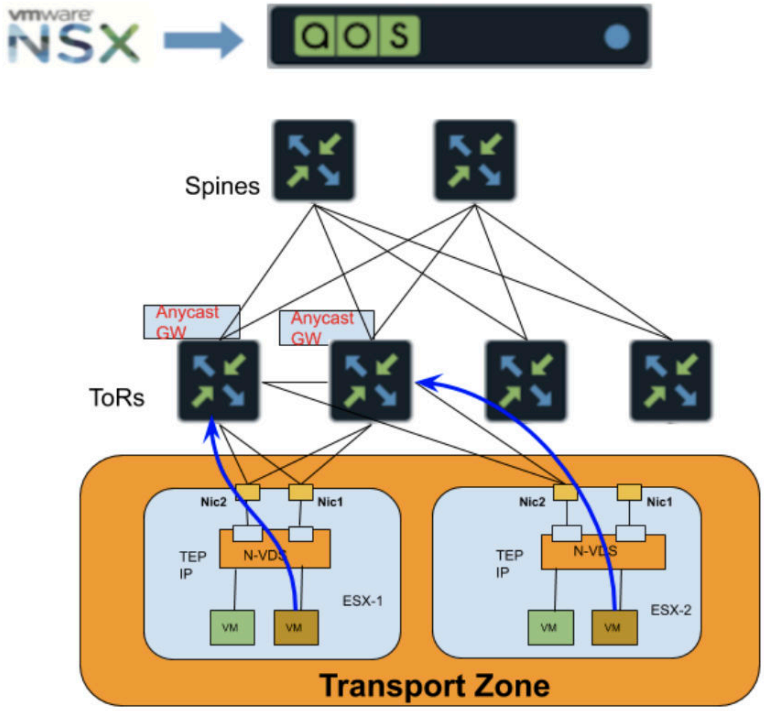
NSX-T Networking Terminology and correlation

IN THIS SECTION

NSX-T uses the following terminology for their control plane and data plane components. Also please find respective correlation with respect to Apstra.

Transport Zones

Transport Zones (TZ) define a group of ESXi hosts that can communicate with one another on a physical network.

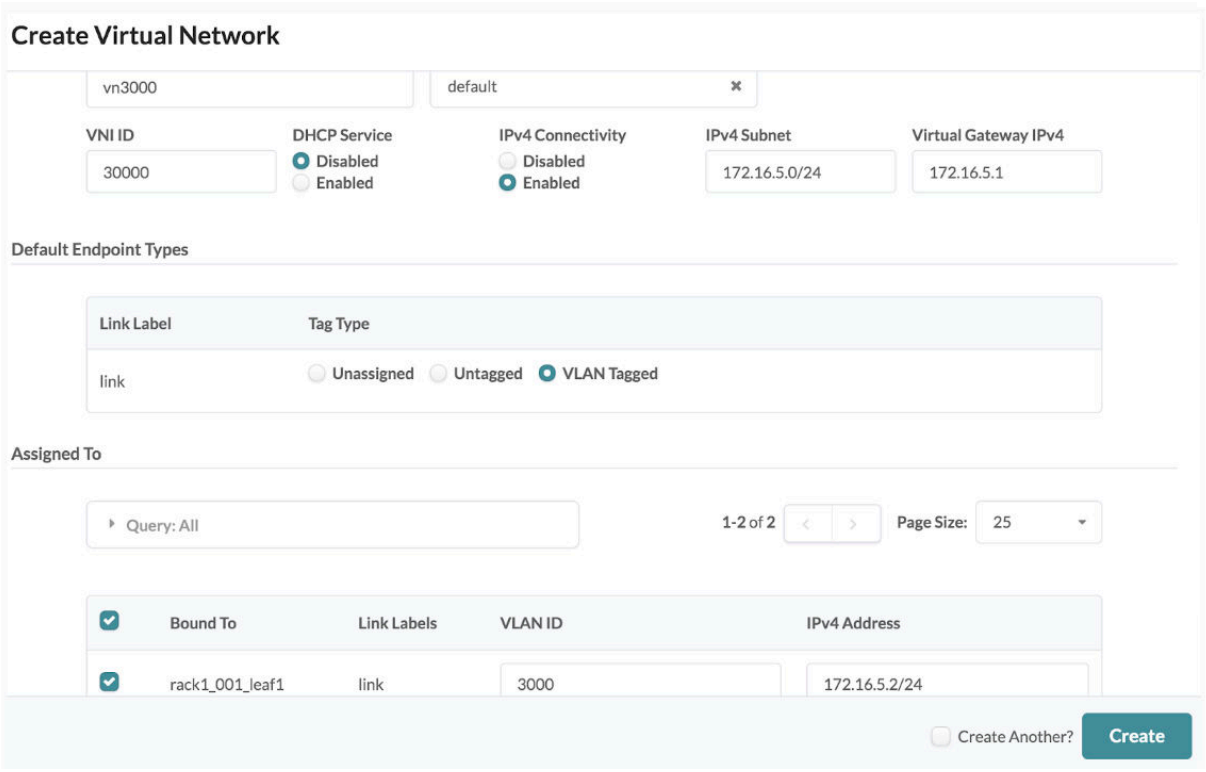


There are two types of Transport Zones:

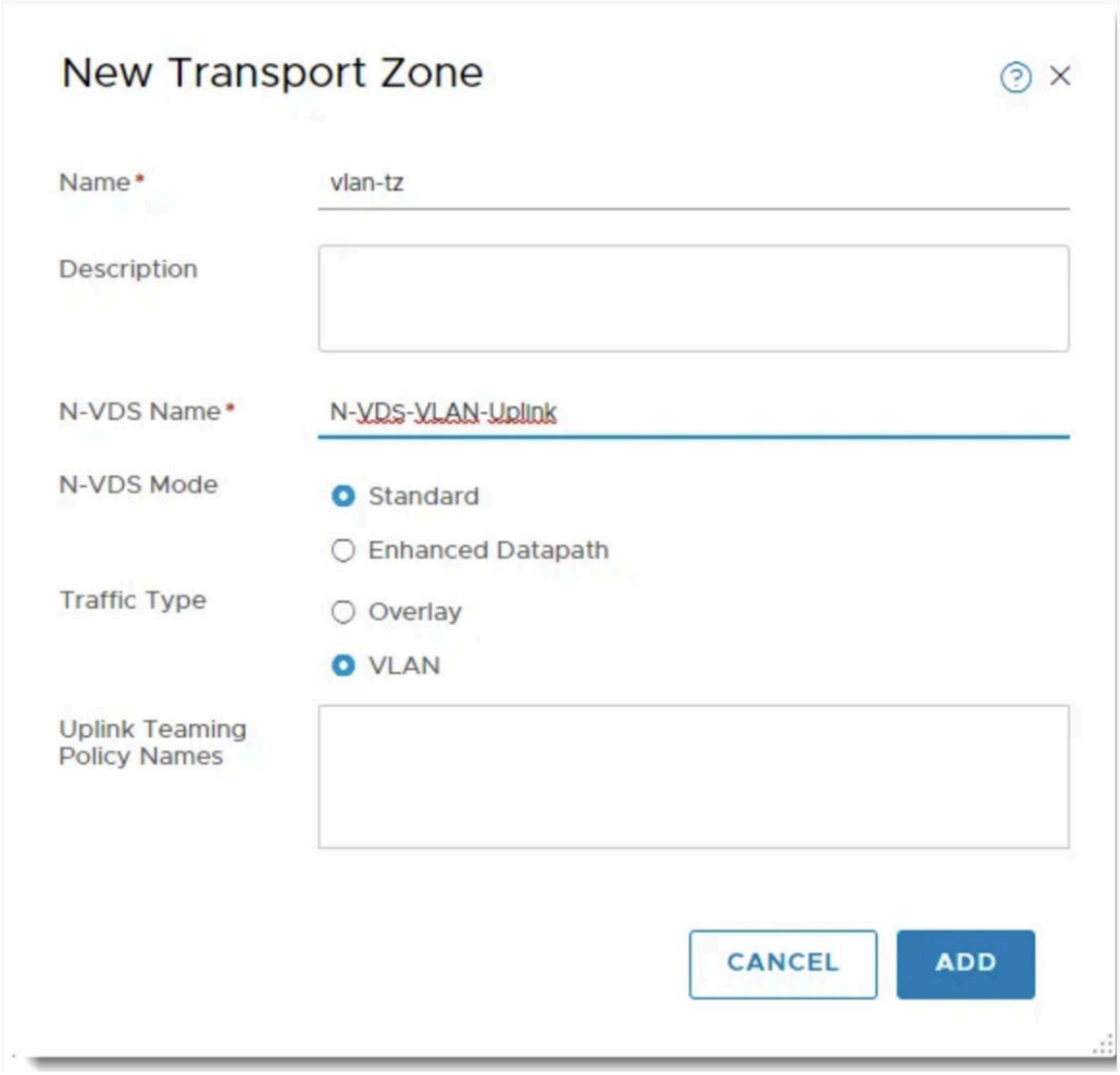
1. Overlay Transport Zone: This transport zone can be used by both transport nodes or NSX edges.When an ESXi host or NSX-T Edge transport node is added to an Overlay transport zone, an N-VDS is installed on the ESXi host or NSX Edge Node.
2. VLAN Transport Zone: It can be used by NSX Edge and host transport nodes for its VLAN uplinks.

Each Hypervisor Hosts can only belong to one Transport Zone at a given point of time.

A newly created VLAN VN tagged towards an interface in Apstra fabric corresponds to a VLAN based transport zone as per the screenshots below:



Here tagged VLAN VN is mapped to the respective Transport Zone in NSX-T with traffic type as VLAN.



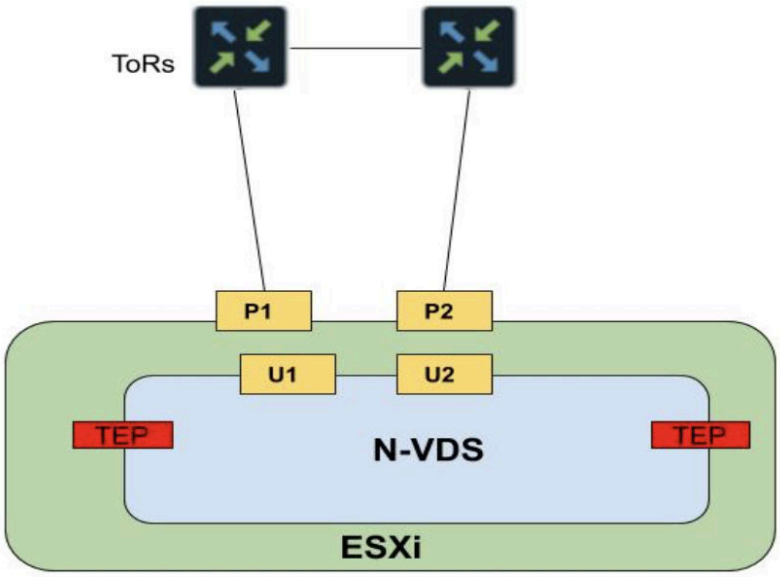
N-VDS

An NSX-managed virtual distributed switch provides the underlying forwarding and is the data plane of the transport nodes.

A few notables about N-VDS virtual switches include:

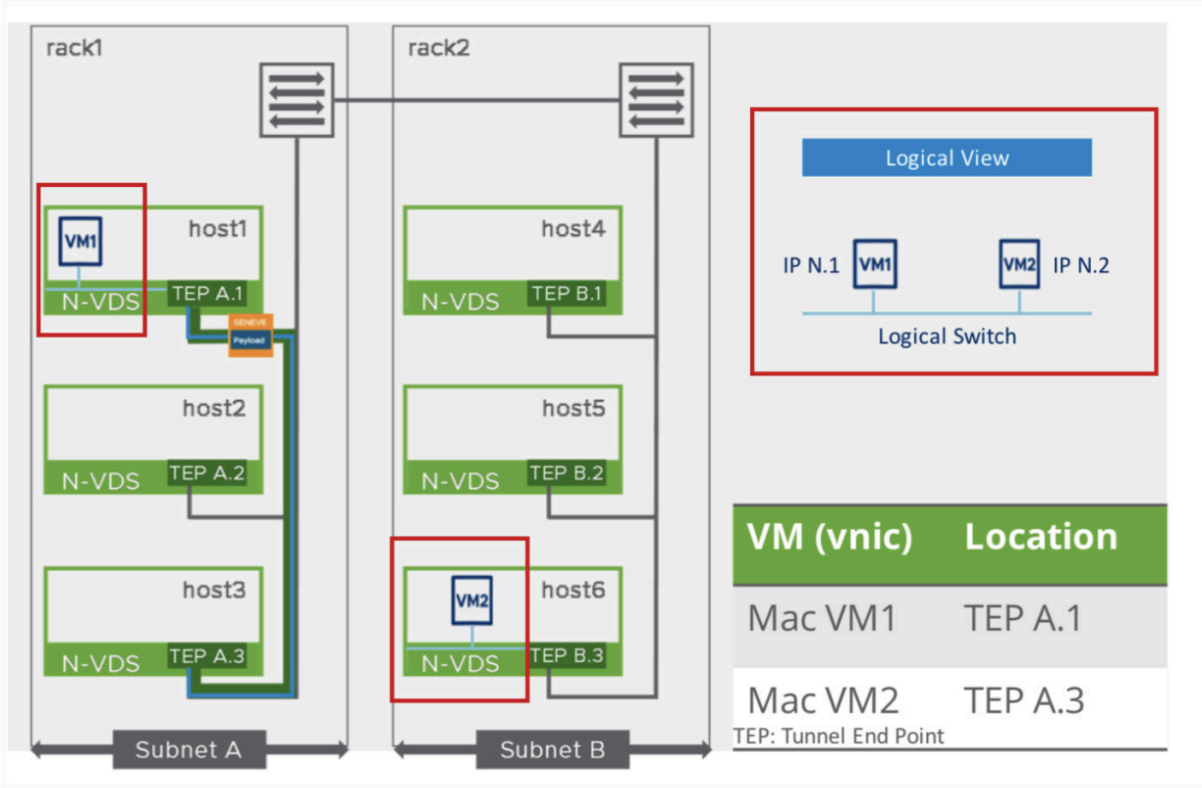
- pnics are physical ports on the ESXi host
- pnics can be bundled to form a link aggregation (LAG)
- uplinks are logical interfaces of an N-VDS

- uplinks are assigned pnics or LAGs



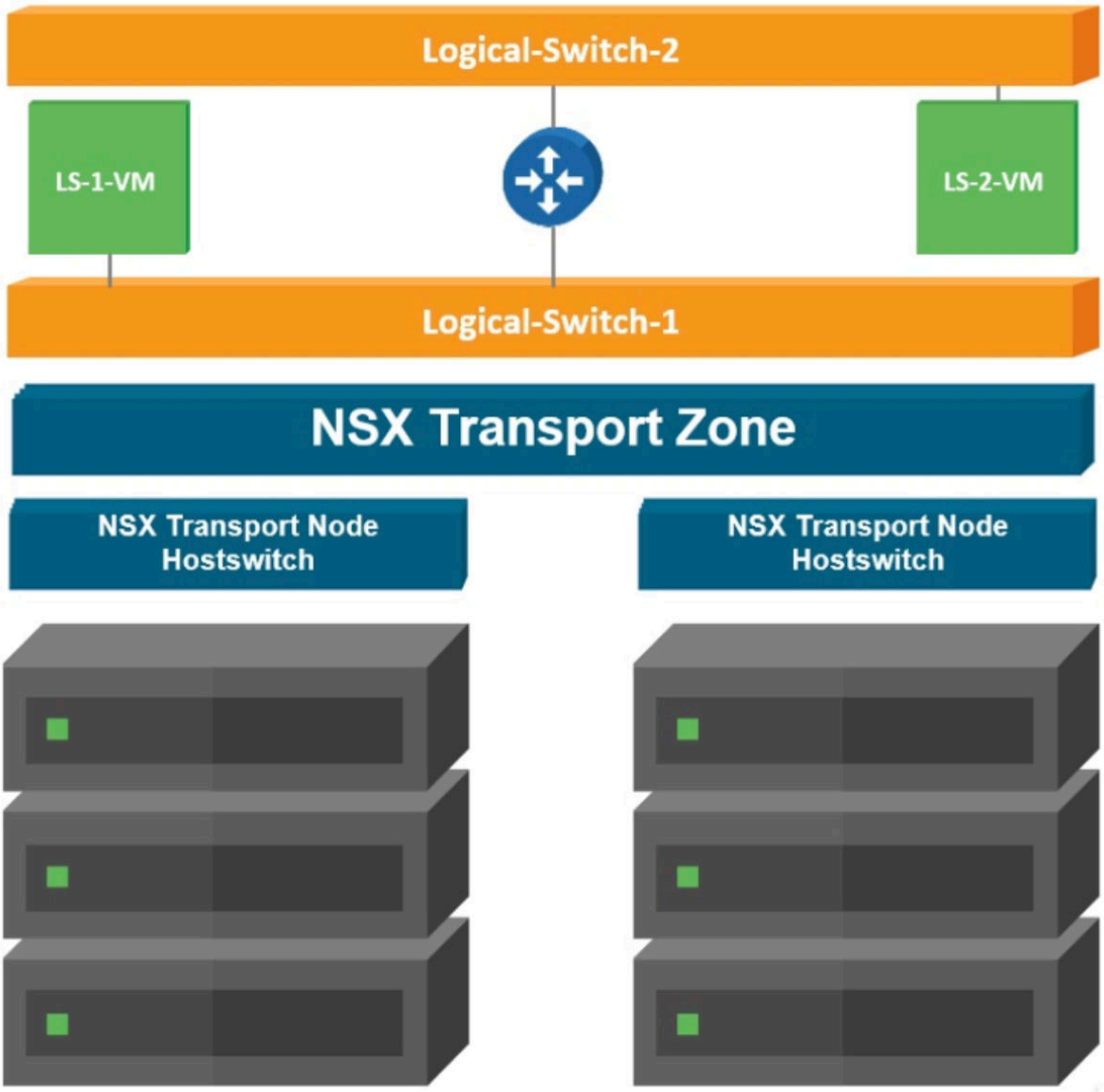
Here TEP are Tunnel Endpoints used for the NSX overlay networking (geneve encapsulation/ decapsulation). P1/P2 are pNICs mapped to the uplink profile(U1/U2).

N-VDS are instantiated at the Hypervisor level and can be thought of Virtual switch connected to the ToR physical leaf devices as below:



Transport Node

It is a node capable of participating in an NSX-T Data Center overlay or VLAN networking.



VMs hosted on different Transport nodes communicate seamlessly across the overlay network. A transport node can belong to:

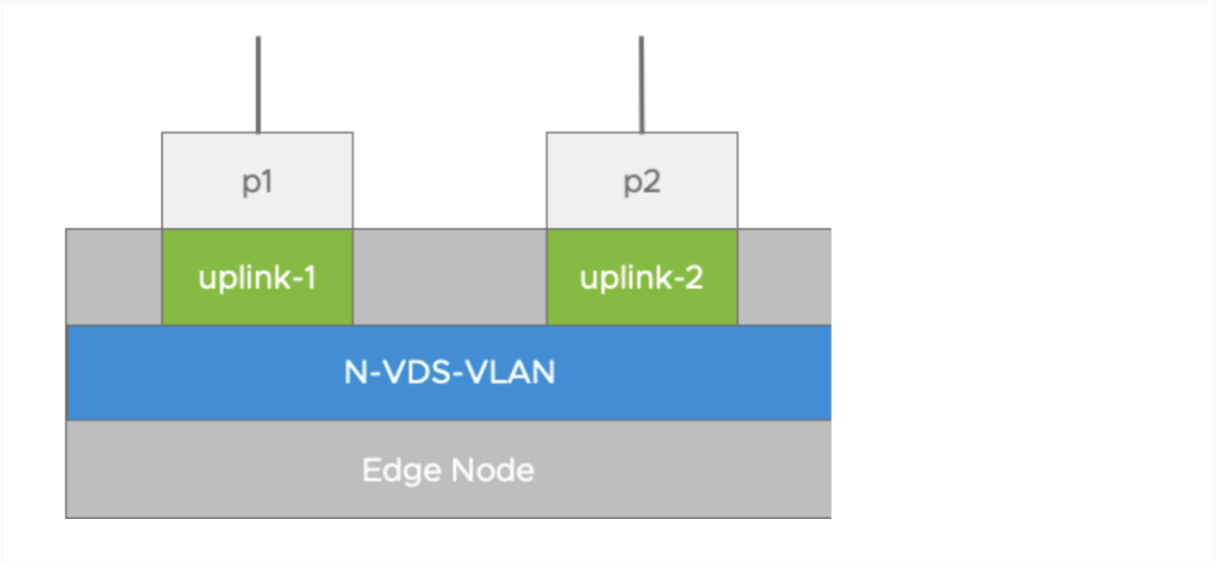
- Multiple VLAN transport zones.
- At most one overlay transport zone with a standard N-VDS.

This can be compared to setting end hosts(servers) in an Apstra blueprint to be part of VLAN (leaf-local) or VXLAN (inter-leaf) Virtual Network.

NSX Edge Node

The NSX Edge provides routing services and connectivity to networks that are external to the NSX-T deployment. It is required for establishing external connectivity from the NSX-T domain, through a Tier-0 router via BGP or static routing.

NSX Edge VMs have uplinks towards ToR leaves needing a separate VLAN transport zone. Apstra fabric must be configured with the corresponding VLAN Virtual Network.



NOTE: NSX-T Edge Bare Metal or VM form factors are Transport nodes and discovered as hypervisors in Apstra. However, VM edge Transport nodes can't be correlated to the connected ToR Leaf.

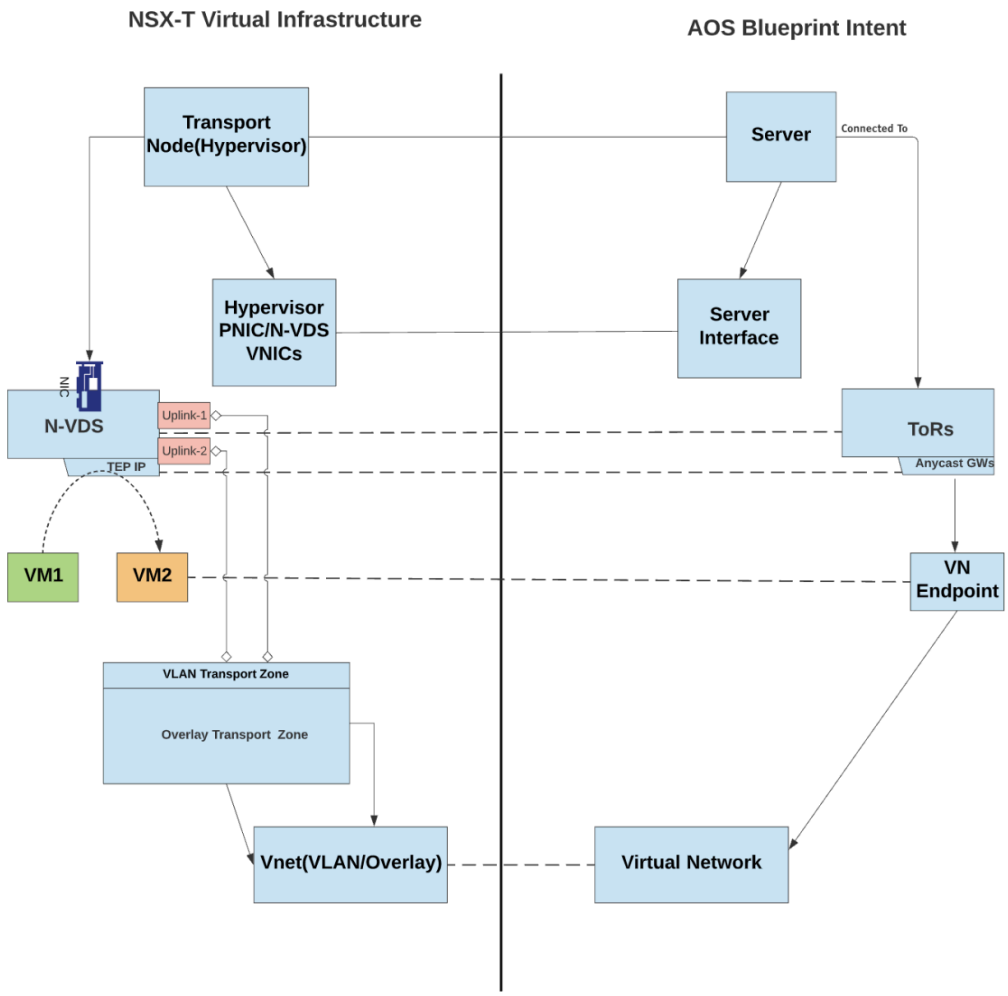
NSX Controller Cluster

It provides control plane functions for NSX-T Data Center logical switching and routing components.

NSX Manager

It is a node that hosts the API services, the management plane, and the agent services.

NSX Inventory Model



- In NSX-T Transport nodes are hypervisor hosts and they can be correlated to server nodes in a Blueprint connected to the ToR leaf devices. In NSX-T Data Center, ESXi hosts are prepared as Transport Node which allows nodes to exchange traffic for virtual networks on Apstra Fabric or amongst network on nodes. You must ensure hypervisors (ESXi) networking stack is sending LLDP packets to aid the correlation of ESXi hosts with server nodes in the blueprint.
- PNIC is the actual physical network adapter on ESXi or hypervisor host. Hypervisor PNICs can be correlated to the server interface on the Blueprint. LAG or Teaming configuration is done on the links mapped to these physical NICs. This can be correlated to bond configuration done on the ToR leaf devices towards the end servers.
- In NSX-T integration with Apstra VM virtual networks are discovered. These can be correlated to blueprint virtual networks. In case VMs need to communicate with each other over tunnels between hypervisors VMs are connected to the same logical switch in NSX-T(called N-VDS). Each logical

switch has a virtual network identifier (VNI), like a VLAN ID. This corresponds to VXLAN VNIs as in Apstra fabric physical infrastructure.

- The NSX-T Uplink Profile defines the network interface configuration facing the fabric in terms of LAG and LACP config on PNIC interfaces. The uplink profile is mapped in Transport node for the links from the hypervisor/ESXi towards top-of-rack switches in Apstra Fabric.
- VNIC defines Virtual Interface of transport nodes or VMs. N-VDS switch does mapping of physical NICs to such uplink virtual interfaces. These Virtual Interfaces can be correlated to server interface ports of Apstra Fabric.

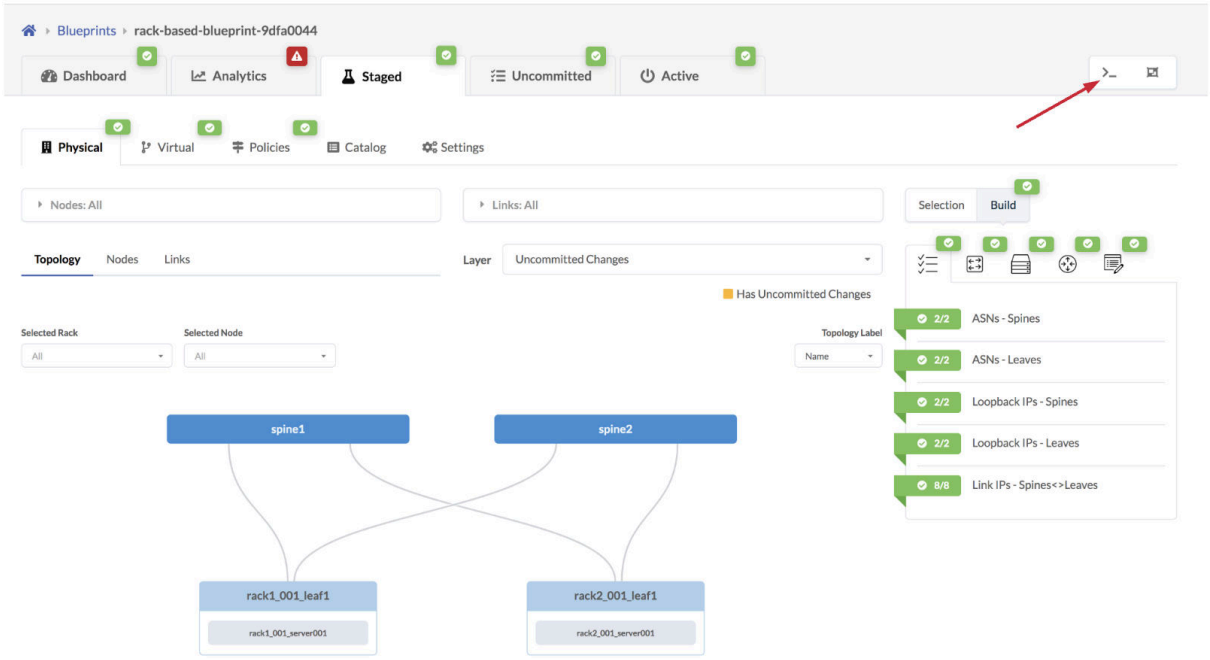
Model Details and Relationship

IN THIS SECTION

Hypervisor

- **Hostname:** FQDN attribute of transport node
- **Hypervisor_id:** Id attribute of transport node
- **Label:** Display name attribute of transport node
- **version:** NSX-T version installed on the transport node

To obtain NSX-T API response for respective hypervisor hosts and understand the correlation you can use graph query. To open the GraphQL Explorer, click the ">_" button



After that in the graph explorer we can type a graph query on the left as per the screenshot below using GraphQL:

To check for respective Label for the transport nodes below query can be used:

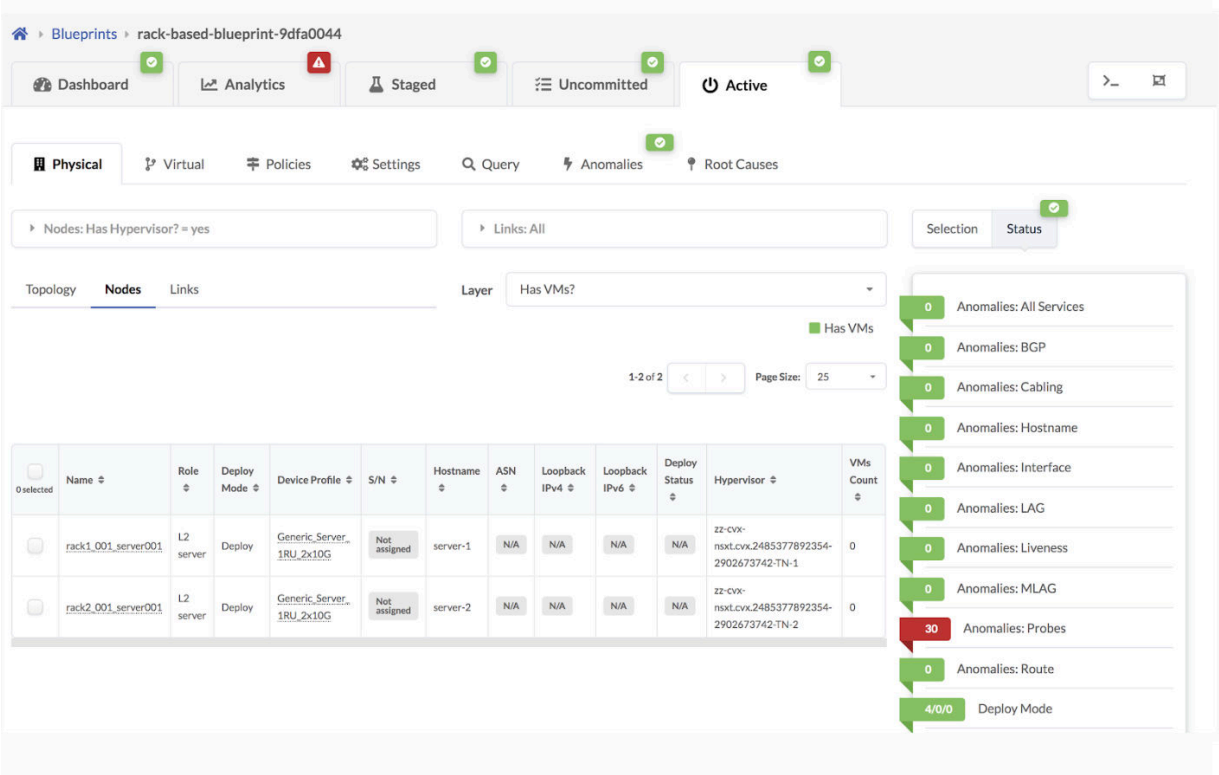
Request:

```
{
 hypervisor_nodes{
 label
 }
 }
```

Response:


```
{
 "data": {
   "hypervisor_nodes": [
     {
       "label": "zz-karun-nsxt.cvx.2485377892354-357746820-TN-2"
     },
     {
       "label": "zz-AndyF-nsxt.cvx.2485377892354-4240714876-TN-2"
     }
   ]
 }
}
```


Hypervisors which act as Transport Nodes can be visualized in Apstra under **Active** tab with **Has Hypervisor = Yes** option as below:



To obtain respective hostname for the transport nodes below query can be used:

Request:

```
{
 hypervisor_nodes {
 hostname
 }
 }
```

Response:

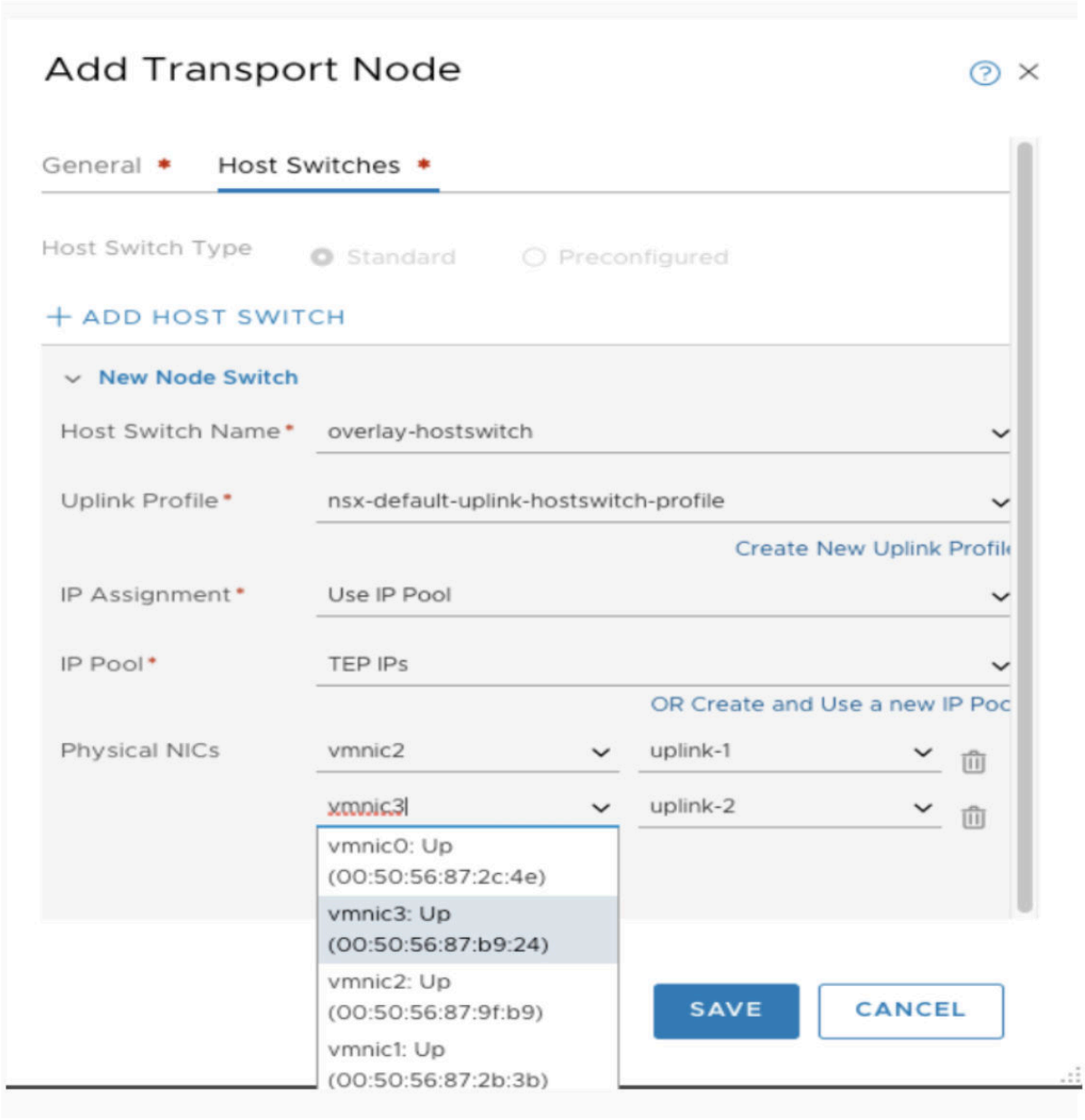
```
{
 "data": {
   "hypervisor_nodes": [
     {
       "hostname": "localhost"
     },
     {
       "hostname": "ubuntu-bionic-nsxt"
     }
   ]
 }
}
```

Hypervisor PNIC

- **MAC address:** Physical address attribute of transport node's interface
- **Switch_id:** Switch name attribute of transport node's transport zone
- **Label:** Interface id attribute of transport node's interface
- **Neighbor_name:** System name attribute of transport node's interface lldp neighbor
- **Neighbor_intf:** Name attribute of transport node's interface lldp neighbor
- **MTU:** MTU attribute of transport node's interface

Physical NICs are selected for uplink profile dedicated for the Overlay Network. NSX-T Uplink Profile defines the network interface configuration for the PNIC interfaces facing the Apstra fabric in terms of

LAG and LACP config.



So the uplink profile is mapped in Transport node for the links from the NSX-T logical switch of the hypervisor/ESXi hosts. It points towards top-of-rack switches in Apstra Fabric.

NSX-API Request/Response to check MAC address for the Transport node interfaces.

Request:

```
{
pnic_nodes {
id mac_address
```

```
}
}
```

Response:

```
{
  "data": {
   "pnic_nodes": [
     {
       "id": "1e2162c3-9ce6-4f35-afc2-217bb48ced49",
       "mac_address": "52:54:00:88:41:28"
     },
     {
       "id": "9752a438-1939-4648-bc8e-0494addf7c7e",
       "mac_address": "52:54:00:04:d5:4f"
     }
   ]
 }
}
```

The MAC address shown in above example is learned on a LAG interface in Apstra Fabric towards the NSX-T Transport Node. It is the MAC address of the ESXi host pNICs having LAG bond towards ToR leaf devices in Apstra fabric.

The NSX-API Request/Response below checks the switch name attribute of transport node's transport zone.

Request:

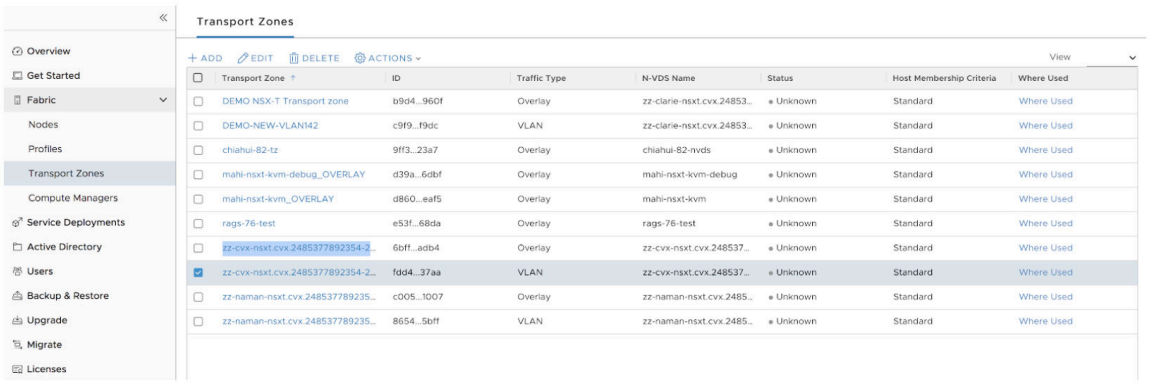
```
{
pnic_nodes {
id switch_id
}
}
```

Response:

```
{
 "data": {
   "pnic_nodes": [
     {
```

```
        "id": "82586be7-2998-401f-82ba-11afa5bb9730",
        "switch_id": "zz-cvx-nsxt.cvx.2485377892354-2902673742"
      },
      {
        "id": "0043d742-405a-454f-9e9b-695d5dd14608",
        "switch_id": "zz-cvx-nsxt.cvx.2485377892354-2902673742"
      }
    ]
  }
}
```

Switch ID attribute of the respective transport zone are read by NSX-T API from NSX manager as below:



NSX-API Request/Response to check Transport node's interface.

Request:

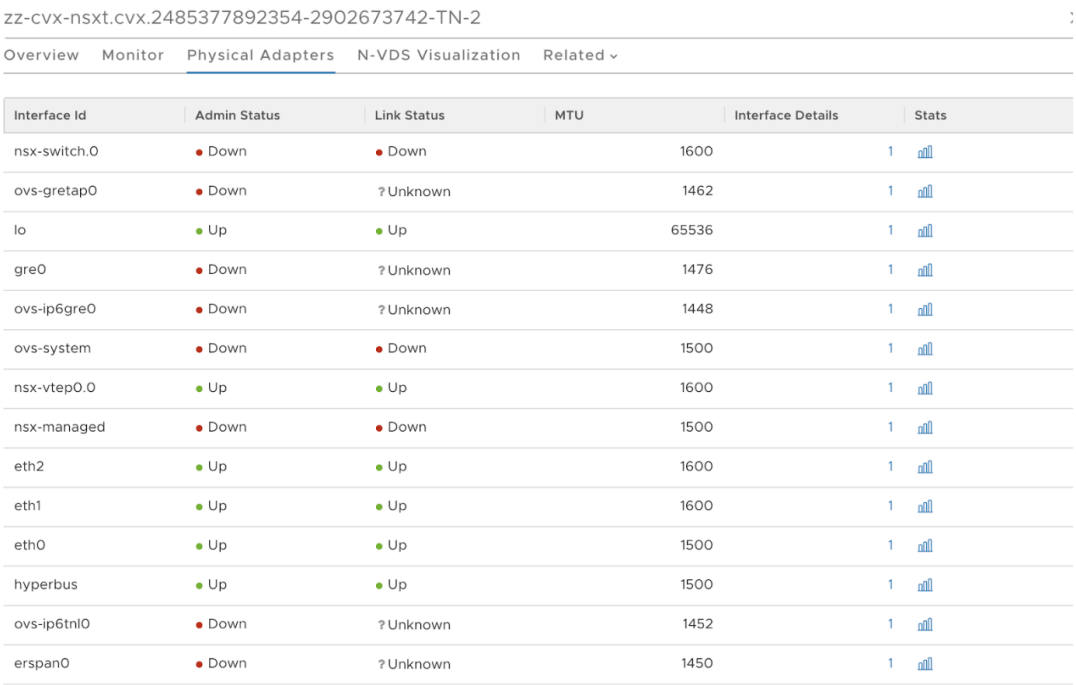
```
{
pnic_nodes {
id label
}
}
```

Response:

```
{
  "data": {
   "pnic_nodes": [
     {
```

```
        "id": "82586be7-2998-401f-82ba-11afa5bb9730",
        "label": "eth2"
      },
      {
        "id": "0043d742-405a-454f-9e9b-695d5dd14608",
        "label": "eth1"
      },
      {
        "id": "b91a5725-7500-489b-a454-e05d7c311525",
        "label": "eth0"
      }
    ]
  }
}
```


Transport nodes has the mapping of physical NICs which can be seen returned as labels according to above NSX-T API response.



zz-cvx-nsxt.cvx.2485377892354-2902673742-TN-2

Overview Monitor Physical Adapters N-VDS Visualization Related

Interface Id	Admin Status	Link Status	MTU	Interface Details	Stats
nsx-switch.0	Down	Down	1600	1	
ovs-gretap0	Down	Unknown	1462	1	
lo	Up	Up	65536	1	
gre0	Down	Unknown	1476	1	
ovs-ip6gre0	Down	Unknown	1448	1	
ovs-system	Down	Down	1500	1	
nsx-vtep0.0	Up	Up	1600	1	
nsx-managed	Down	Down	1500	1	
eth2	Up	Up	1600	1	
eth1	Up	Up	1600	1	
eth0	Up	Up	1500	1	
hyperbus	Up	Up	1500	1	
ovs-ip6tnl0	Down	Unknown	1452	1	
erspan0	Down	Unknown	1450	1	

Please find below NSX-API Request/Response to check Transport node's LLDP neighbor System name attribute.

Request:

```
{
pnic_nodes {
id neighbor_name
}
}
```

Response:


```
{
  "data": {
    "pnic_nodes": [
      {
        "id": "82586be7-2998-401f-82ba-11afa5bb9730",
        "neighbor_name": "leaf-2-525400C6DD2B"
      },
      {
        "id": "0043d742-405a-454f-9e9b-695d5dd14608",
        "neighbor_name": "leaf-2-525400C6DD2B"
      },
      {
        "id": "b91a5725-7500-489b-a454-e05d7c311525",
        "neighbor_name": "spine-1"
      },
      {
        "id": "f77575fb-44ea-4ec7-9913-1c75b7af87bc",
        "neighbor_name": "leaf-1-5254004D5560"
      },
      {
        "id": "628d0f86-4bc1-4faf-8f3f-f1deb92ceee2",
        "neighbor_name": "leaf-2-525400C6DD2B"
      },
      {
        "id": "1e2162c3-9ce6-4f35-afc2-217bb48ced49",
        "neighbor_name": "leaf-1-5254004D5560"
      }
    ]
  }
 }
```

Here Leaf1/2 are LLDP neighbors to the Transport nodes.

To obtain respective transport node's LLDP neighbor interface name attribute below query can be used:

Request:

```
{
pnic_nodes {
id neighbor_intf
}
}
```

Response:


```
{
  "data": {
    "pnic_nodes": [
      {
        "id": "82586be7-2998-401f-82ba-11afa5bb9730",
        "neighbor_name": "leaf-2-525400C6DD2B"
      },
      {
        "id": "0043d742-405a-454f-9e9b-695d5dd14608",
        "neighbor_name": "leaf-2-525400C6DD2B"
      },
      {
        "id": "b91a5725-7500-489b-a454-e05d7c311525",
        "neighbor_name": "spine-1"
      },
      {
        "id": "f77575fb-44ea-4ec7-9913-1c75b7af87bc",
        "neighbor_name": "leaf-1-5254004D5560"
      },
      {
        "id": "628d0f86-4bc1-4faf-8f3f-f1deb92ceee2",
        "neighbor_name": "leaf-2-525400C6DD2B"
      },
      {
        "id": "1e2162c3-9ce6-4f35-afc2-217bb48ced49",
        "neighbor_name": "leaf-1-5254004D5560"
      }
    ]
```

```
  }
 }
```

NSX-API Request/Response to check the MTU attribute of Transport node's interface.

Request:

```
{
pnic_nodes {
id neighbor_intf
}
}
```

Response:

```
{
  "data": {
   "pnic_nodes": [
     {
       "id": "82586be7-2998-401f-82ba-11afa5bb9730",
       "neighbor_intf": "swp4"
     },
     {
       "id": "0043d742-405a-454f-9e9b-695d5dd14608",
       "neighbor_intf": "swp3"
     },
     {
       "id": "b91a5725-7500-489b-a454-e05d7c311525",
       "neighbor_intf": "eth0"
     }
   ]
 }
}
```

MTU size of 1600 or greater is needed on any network that carries Geneve overlay traffic must. Hence in the NSX-T reply we can notice MTU value 1600 on network interfaces towards Transport nodes.

VNIC

- **MAC address:** Physical address attribute of transport node's or VM's Virtual interface
- **Label:** VNIC label attribute of transport node

- **Ipv4_addr:** IP address attribute of transport node's virtual interface
- **Traffic_types:** It is derived from transport node's virtual interface type
- **MTU:** MTU attribute of transport node's virtual interface

You can check the VNIC mac address attribute with the below NSX-API Request/Response. This can be of transport node's interface Virtual Interface or can be for the Virtual Interface of the VMs. For transport nodes under Host Switches select the Virtual NIC that matches the MAC address of the VM NIC attached to the uplink port group.

Request:

```
{
vnic_nodes{
id mac_address
}
}
```

Response:

```
{
  "data": {
   "vnic_nodes": [
     {
       "id": "c84d8636-c28b-4db3-8747-37fadca4c7aa",
       "mac_address": "1e:5c:3b:a2:ea:c3"
     },
     {
       "id": "7d5826d8-0622-4a45-88d7-6b1e88bac62f",
       "mac_address": "ca:0f:93:24:24:43"
     }
   ]
 }
}
```

NSX-API Request/Response to check VNIC label which signifies interface id attribute of transport node's virtual interface or device name attribute of virtual machine's virtual interface.

Request:

```
{
vnic_nodes{
id label
}
}
```

Response:

```
{
  "data": {
   "vnic_nodes": [
     {
       "id": "c84d8636-c28b-4db3-8747-37fadca4c7aa",
       "label": "hyperbus"
     },
     {
       "id": "7d5826d8-0622-4a45-88d7-6b1e88bac62f",
       "label": "nsx-switch.0"
     },
     {
       "id": "473c2b7d-ab2f-41cd-9a4b-fcf2eb248fd6",
       "label": "nsx-switch.0"
     },
     {
       "id": "9553390b-754e-45ef-8976-e63396d554ee",
       "label": "nsx-vtep0.0"
     },
     {
       "id": "a00bb649-5032-462f-97e7-b6c4f5f1ac86",
       "label": "nsx-vtep0.0"
     }
   ]
 }
}
```

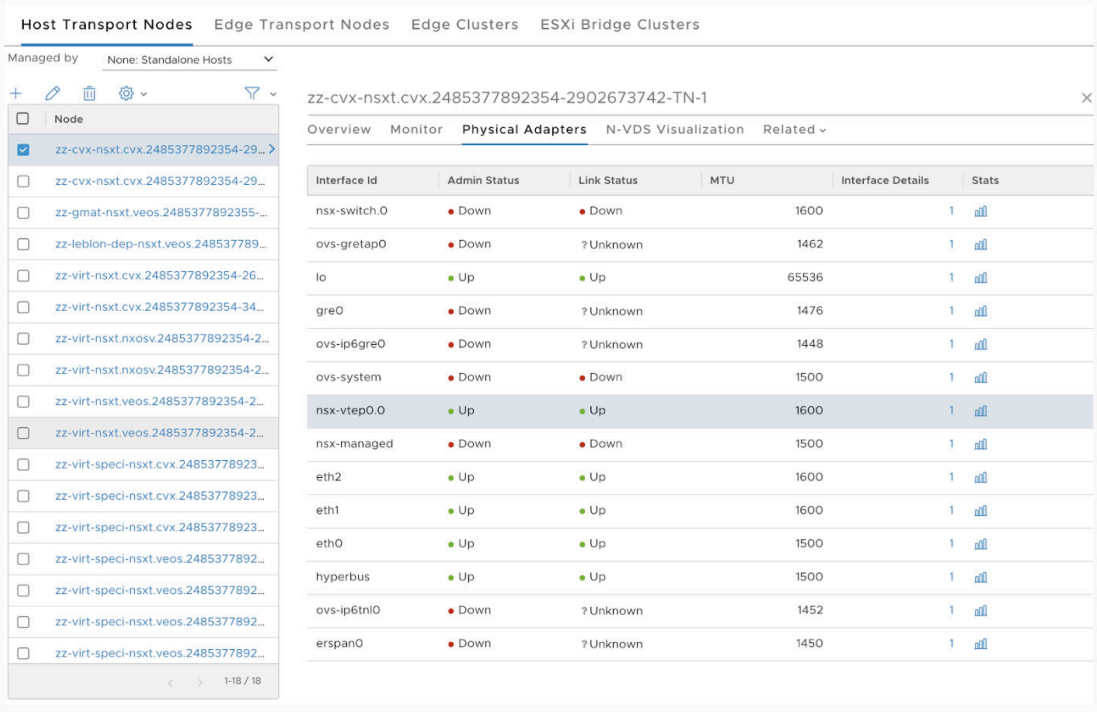
Below is the NSX-API Request/Response to check VNIC Ipv4 address which signifies ip address attribute of transport node's virtual interface or for the virtual interface of logical port.

Request:

```
{
vnic_nodes{
id ipv4_addr
}
}
```

Response:

```
{
  "data": {
   "vnic_nodes": [
     {
       "id": "9553390b-754e-45ef-8976-e63396d554ee",
       "ipv4_addr": "192.168.1.13"
     },
     {
       "id": "a00bb649-5032-462f-97e7-b6c4f5f1ac86",
       "ipv4_addr": "192.168.1.12"
     }
   ]
 }
}
```



Here “192.168.1.13” and “192.168.1.12” are ipv4 addresses for the bridge interface of the host transport nodes i.e **"nsx-vtep0.0"** which acts as a virtual tunnel endpoint (VTEP) of the transport node. Each hypervisor has a Virtual Tunnel Endpoint (VTEP) responsible for encapsulating the VM traffic inside a VLAN header and routing the packet to a destination VTEP for further processing. This can be compared to VXLAN Virtual Network anycast GW VTEP IP.

```
nsx-vtep0.0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1600
    inet 192.168.1.12  netmask 255.255.255.224  broadcast 192.168.1.31
    inet6 fe80::c8ec:50ff:fe69:536  prefixlen 64  scopeid 0x20<link>
    ether ca:ec:50:69:05:36  txqueuelen 1000  (Ethernet)
    RX packets 60312  bytes 3975194 (3.9 MB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 31215  bytes 2675310 (2.6 MB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
admin@localhost:~$
```

NSX-API Request/Response to check traffic types for the transport node’s virtual interface. Traffic type for the transport node can be overlay type as per the example below or it can be of VLAN type. One can add both the VLAN and overlay NSX Transport Zones to the Transport Nodes.

VLAN based Transport zone is mainly for uplink based traffic. In case VMs on different Hypervisor hosts need to communicate to each other then overlay network should be used. It can be compared to VXLAN Virtual network in Apstra Fabric.

Request:

```
{
vnic_nodes{
id traffic_types
}
}
```

Response:

```
{
  "data": {
   "vnic_nodes": [
     {
       "id": "9553390b-754e-45ef-8976-e63396d554ee",
       "traffic_types": [
         "overlay"
       ]
     },
     {
       "id": "a00bb649-5032-462f-97e7-b6c4f5f1ac86",
       "traffic_types": [
         "overlay"
       ]
     }
   ]
 }
}
```

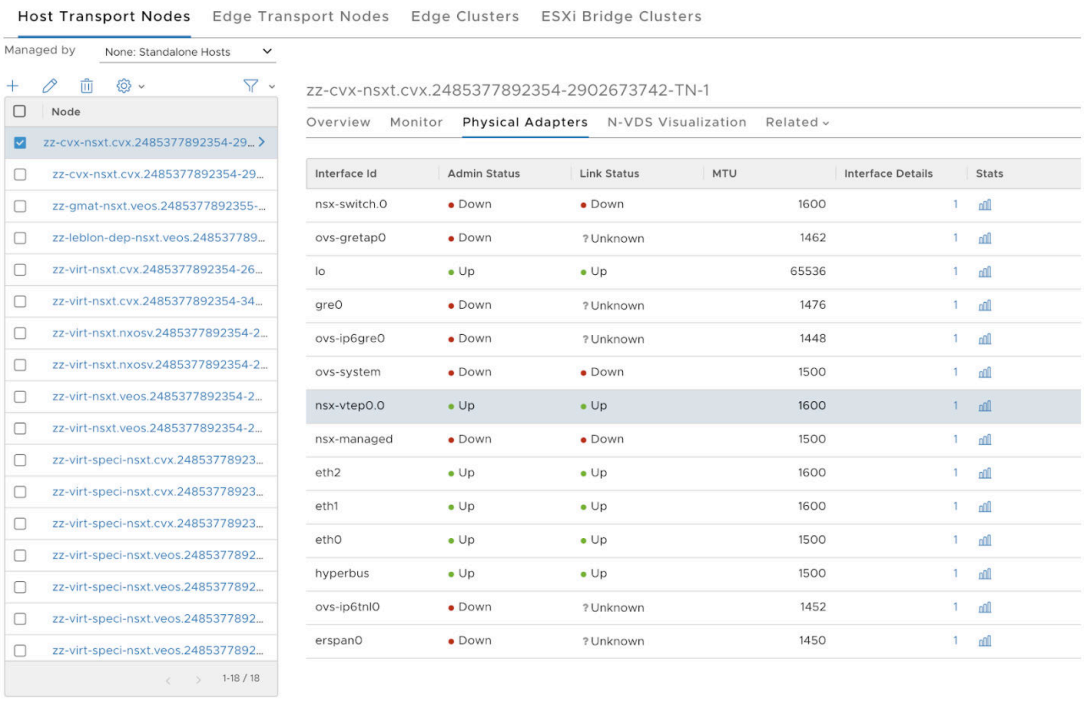
NSX-API Request/Response to obtain the mtu size for the transport node. MTU size for networks that carry overlay traffic must be size of 1600 or greater as it carries Geneve overlay traffic. N-VDS and TEP kernel interface all should have the same jumbo frame MTU size(i.e 1600 or greater).

Request:

```
{
vnic_nodes{
id mtu
}
}
```

Response:


```
{
  "data": {
   "vnic_nodes": [
     {
       "id": "9553390b-754e-45ef-8976-e63396d554ee",
       "mtu": 1600
     },
     {
       "id": "a00bb649-5032-462f-97e7-b6c4f5f1ac86",
       "mtu": 1600
     }
   ]
 }
}
```

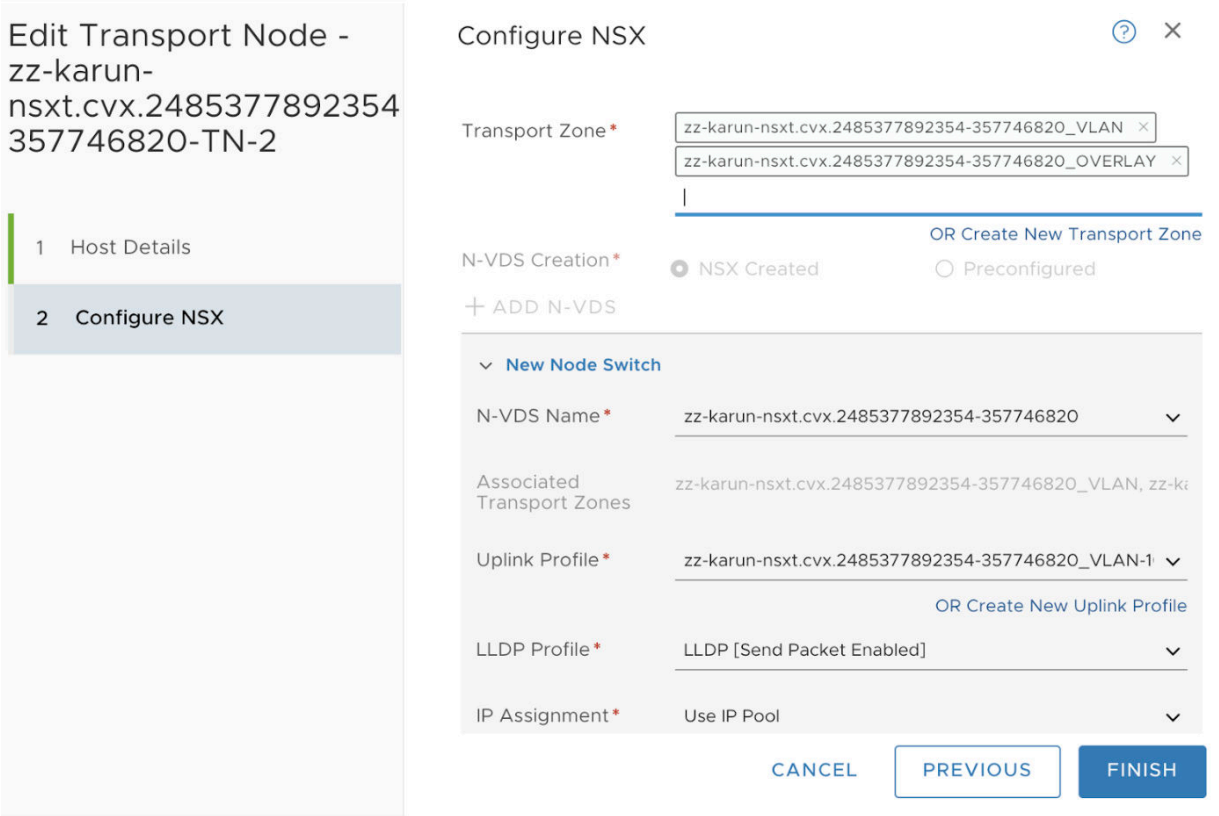
So Virtual Interface i.e NSX VTEP and vswitch should have mtu of 1600 as per screenshot above.

Port Channel Policy

- **Label:** Name attribute of the host switch uplink lag profile

- **Mode:** Mode attribute of host switch uplink lag profile
- **Hashing_algorithm:** Load balance algorithm attribute of host switch uplink lag profile

An uplink profile is mapped in a Transport node on the NSX-T side with policies for the links from the hypervisor hosts to NSX-T logical switches.



The links from the Hypervisor hosts to NSX-T logical switches can comprise of the LAG or Teaming configuration which must be tied to physical NICs.

NSX-API Request/Response to check the logical switch uplink LAG profile attribute.

Request:

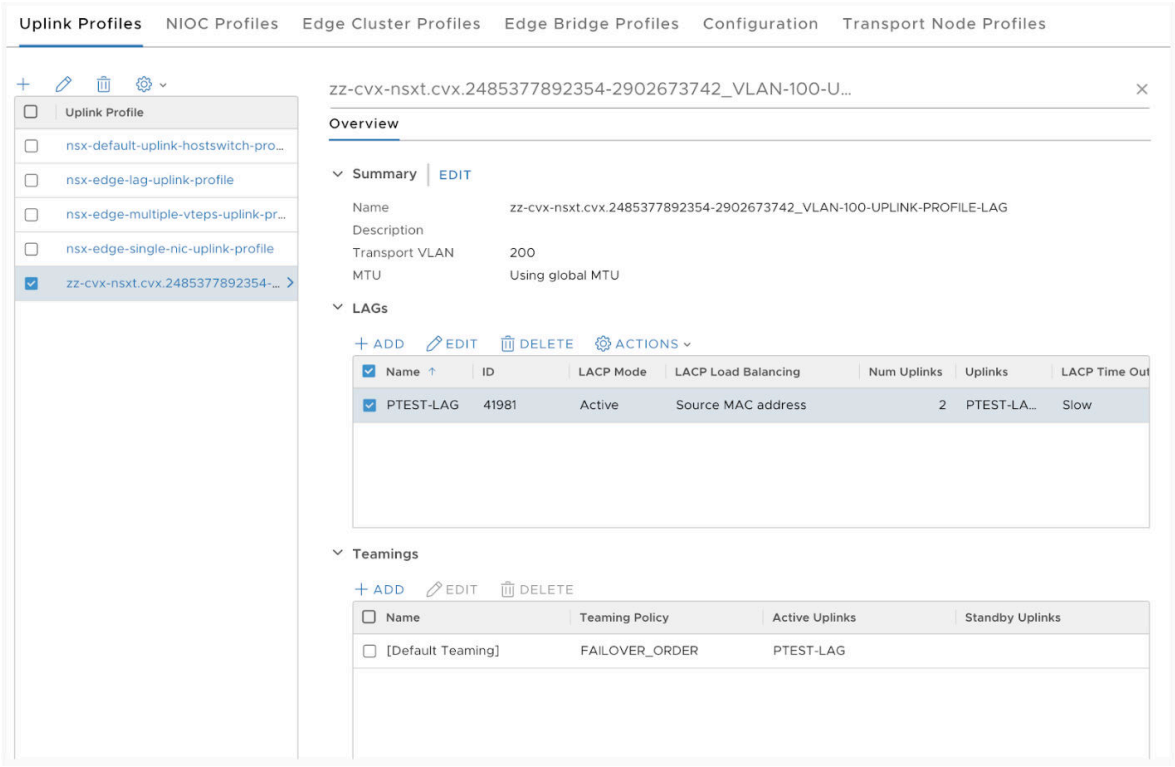
```
{
port_channel_nodes {
id label
} id port_channel_policy_nodes {
id label
```

```
}
}
```

Response:


```
{
 "data": {
   "port_channel_nodes": [
     {
       "id": "bd86666b-239d-4baa-8715-d73ca40d7100",
       "label": null
     },
     {
       "id": "ff5a5b6b-a103-471a-bbfd-ee3dc8c6e1c7",
       "label": null
     }
   ],
   "id": "rack-based-blueprint-9dfa0044",
   "port_channel_policy_nodes": [
     {
       "id": "59f60d47-ca48-441d-a4a4-e570af7bdb72",
       "label": "PTEST-LAG"
     }
   ]
 }
}
```

Uplink profile label can also be matched with one retrieved from the GUI in NSX-T Manager as below:



Below is NSX-API Request/Response to check the LACP mode attribute for the uplink LAG profile.

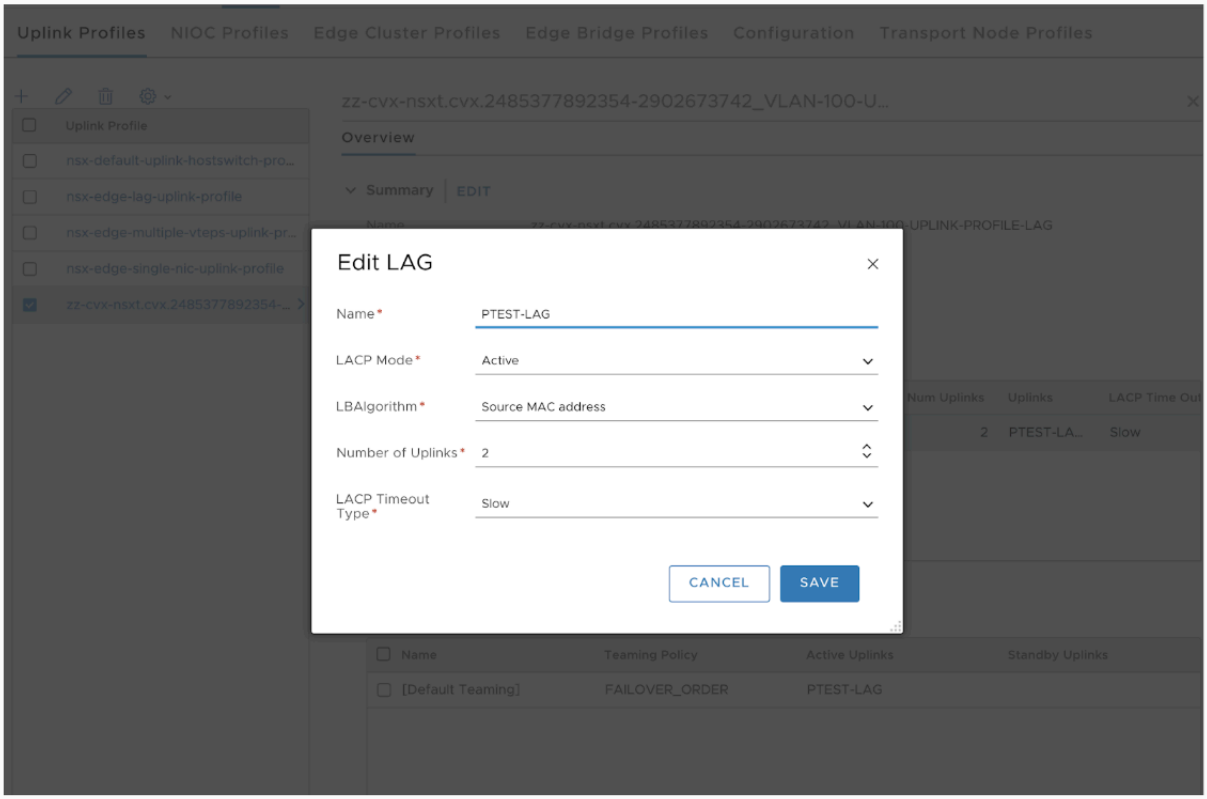
Request:

```
{
port_channel_nodes {
id
} id port_channel_policy_nodes {
id mode
}
}
```

Response:

```
{
  "data": {
   "port_channel_nodes": [
     {
       "id": "bd86666b-239d-4baa-8715-d73ca40d7100"
     },
```

```
      {
        "id": "ff5a5b6b-a103-471a-bbfd-ee3dc8c6e1c7"
      }
    ],
    "id": "rack-based-blueprint-9dfa0044",
    "port_channel_policy_nodes": [
      {
        "id": "59f60d47-ca48-441d-a4a4-e570af7bdb72",
        "mode": "active"
      }
    ]
  }
}
```



NSX-API Request/Response to check load balancing algorithm attribute of host switch uplink profile.

Request:

```
{
port_channel_nodes {
```

```
id
} id port_channel_policy_nodes {
id hashing_algorithm
}
}
```

Response:

```
{
 "data": {
   "port_channel_nodes": [
     {
       "id": "bd86666b-239d-4baa-8715-d73ca40d7100"
     },
     {
       "id": "ff5a5b6b-a103-471a-bbfd-ee3dc8c6e1c7"
     }
   ],
   "id": "rack-based-blueprint-9dfa0044",
   "port_channel_policy_nodes": [
     {
       "id": "59f60d47-ca48-441d-a4a4-e570af7bdb72",
       "hashing_algorithm": "srcMac"
     }
   ]
 }
}
```

From the LAG profile screenshot above it can be validated that it is using Source MAC Address based load balancing algorithm.

Vnet

- **Vn_type:** Transport type attribute of transport zone
- **Label:** Display name attribute of logical switch
- **switch_label:** Switch name attribute of transport zone
- **Vlan:** Vlan attribute of logical switch for vlan transport zone
- **Vni:** vni attribute of logical switch for overlay transport zone

To obtain respective transport type attribute of the transport zone below query can be used. This mainly signifies the type of traffic for a transport zone which can be Overlay or VLAN type.

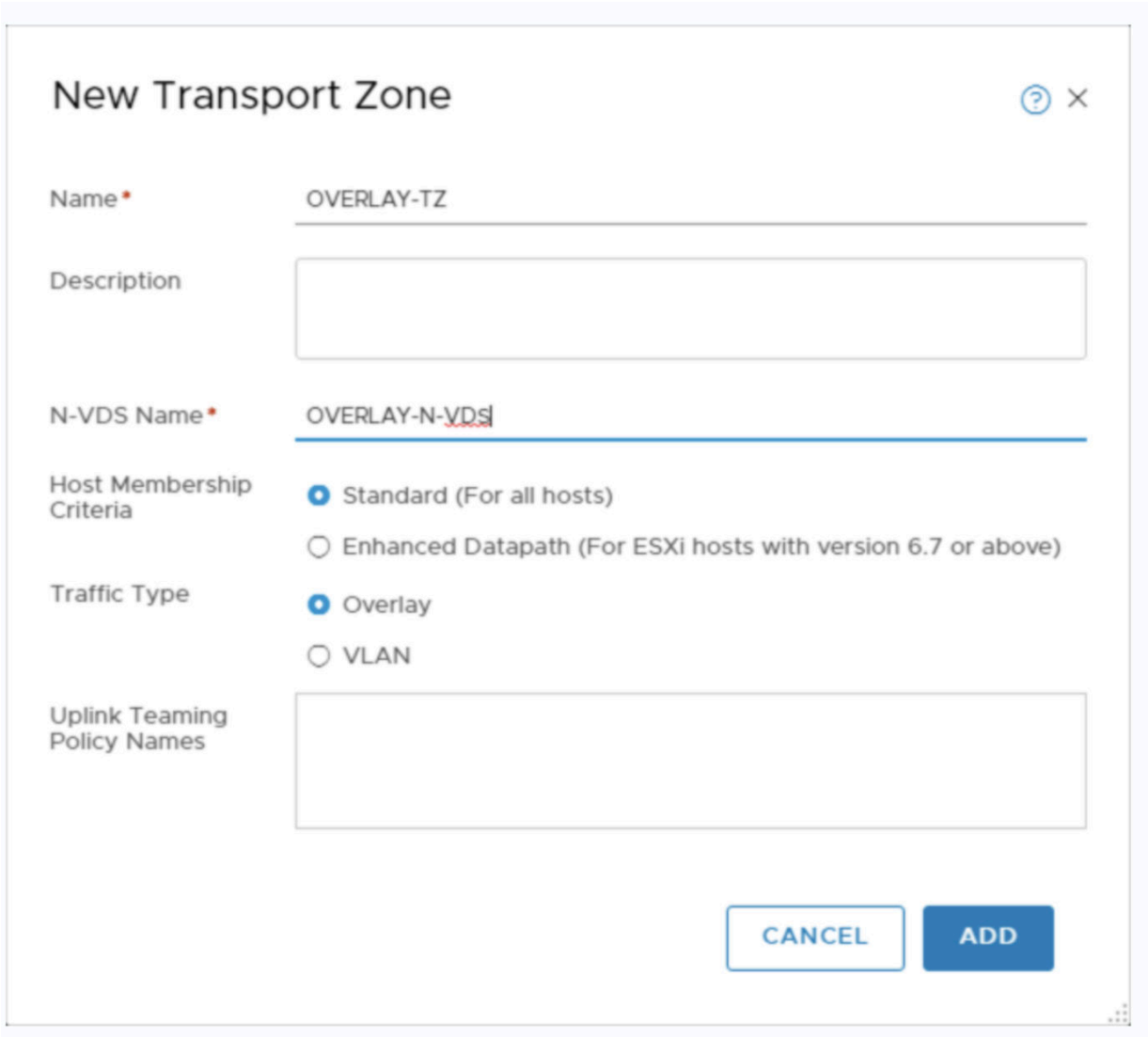
Request:

```
{
vnet_nodes {
id vn_type
} id
}
```

Response:

```
{
  "data": {
    "vnet_nodes": [
      {
        "id": "a3320cc6-601e-4a81-abe9-8464ae054f18",
        "vn_type": "overlay"
      },
      {
        "id": "6bdd7cd9-82eb-433d-8360-076d9daddd1b",
        "vn_type": "vlan"
      }
    ],
    "id": "rack-based-blueprint-9dfa0044"
  }
}
```

Traffic type can also be identified in NSX-T Manager GUI as below:



NSX-API Request/Response to check the display name of the N-VDS logical switch.

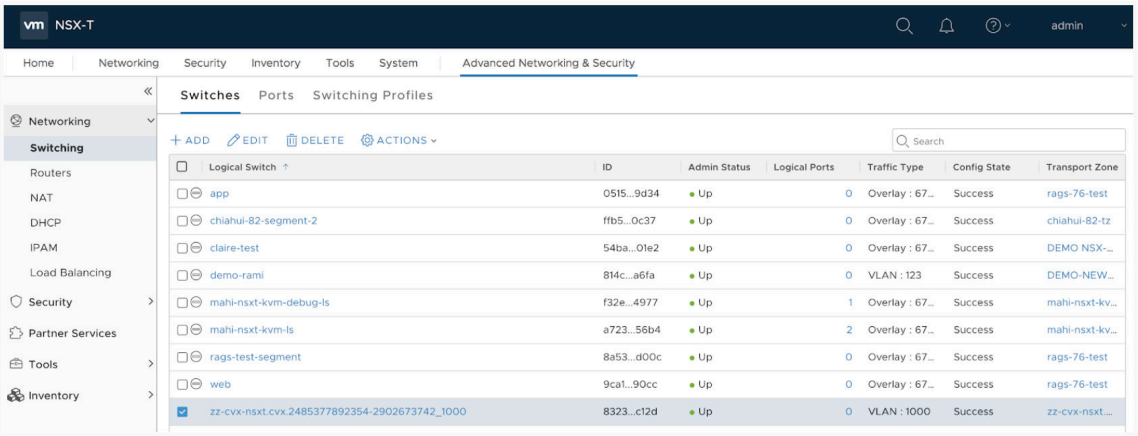
Request:

```
{
vnet_nodes {
id label
} id
}
```

Response:


```
{
 "data": {
   "vnet_nodes": [
     {
       "id": "241ce8e1-b31d-4093-a1a3-2f99a29ac2f9",
       "label": "mahi-nsxt-kvm-ls"
     },
     {
       "id": "fef41435-ac20-4c4d-81c0-b7f3059d977b",
       "label": "zz-cvx-nsxt.cvx.2485377892354-2902673742_1000"
     },
     {
       "id": "6bdd7cd9-82eb-433d-8360-076d9daddd1b",
       "label": "zz-cvx-nsxt.cvx.2485377892354-2902673742_VLAN-100-UPLINK-PROFILE-LAG"
     }
   ],
   "id": "rack-based-blueprint-9dfa0044"
 }
}
```


Here as per API response above “zz-cvx-nsxt.cvx.2485377892354-2902673742_1000” is the respective logical switch associated with the transport zone.



Below is the NSX-API Request/Response to check VLAN ID attribute of a VLAN based logical switch for the transport zone.

Request:

```
{
vnet_nodes {
 id vlan
 } id
 }
```

Response:

```
{
 "data": {
   "vnet_nodes": [
     {
       "id": "e0b29951-7739-4ecb-8c87-5725a61f669a",
       "vlan": 123
     },
     {
       "id": "cdd0c6d5-fecb-44d8-84c4-06c685e8ef14",
       "vlan": 2000
     },
     {
       "id": "fef41435-ac20-4c4d-81c0-b7f3059d977b",
       "vlan": 1000
     },
     {
       "id": "6bdd7cd9-82eb-433d-8360-076d9daddd1b",
       "vlan": 200
     }
   ],
   "id": "rack-based-blueprint-9dfa0044"
 }
}
```

Here in Apstra Fabric VNI IDs 1000 and 2000 represent such VXLAN Virtual network for east-west L2 stretched traffic. Bridge backed logical switch on NSX-T should have the same VLAN IDs defined.

NSX-API Request/Response to check the VNI attribute of logical switch of NSX-T

Request:

```
{
vnet_nodes {
 id vni
 } id
 }
```

Response:

```
{
 "data": {
   "vnet_nodes": [
     {
       "id": "a3320cc6-601e-4a81-abe9-8464ae054f18",
       "vni": 67595
     },
     {
       "id": "b7923224-659b-4075-b69b-3edeb5726a32",
       "vni": 67589
     },
     {
       "id": "18b81c81-8ae1-46b1-83ca-05cd5b364a1c",
       "vni": 67584
     }
   ],
   "id": "rack-based-blueprint-9dfa0044"
 }
}
```

CHAPTER 8

Policies

IN THIS CHAPTER

Endpoints

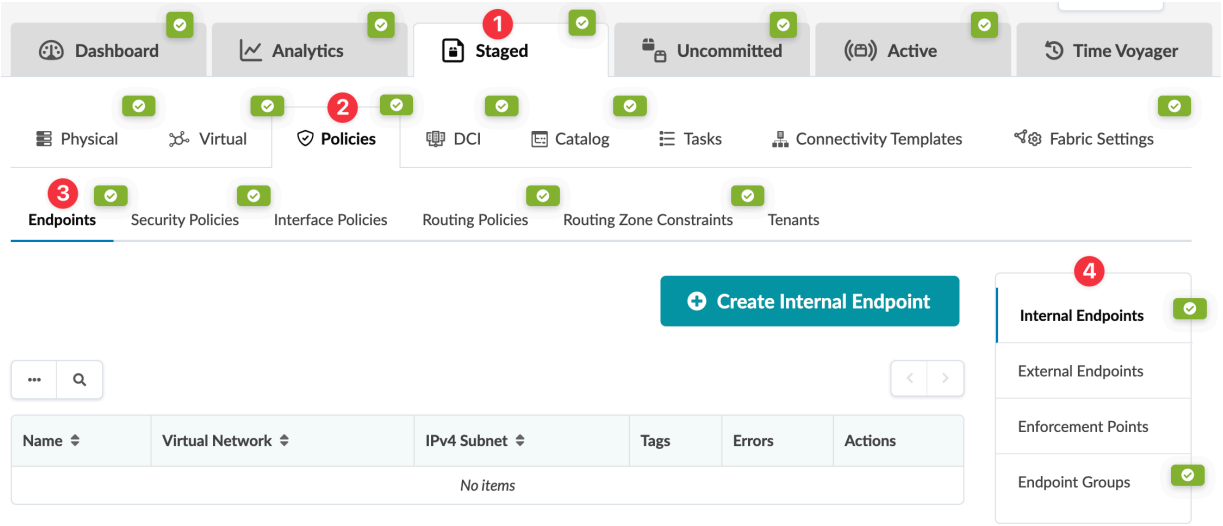
IN THIS SECTION

What are Endpoints

When you want more granularity in your security policies than virtual networks and routing zones can provide, you'll use endpoints. Endpoints can be internal or external to the fabric. You can also combine endpoints into groups.

Endpoints and security policies can be applied to Layer 2 IPv4 blueprints. (Blueprints with IPv6 applications enabled are not supported.) For more information about working with security policies, see "Security Policies" on page 379.

From the blueprint, navigate to **Staged > Policies > Endpoints** to go to endpoints. Click the name of a section to go to its table view. You can create, edit and delete endpoints. Then, when you create a security policy you'll select the endpoints that you've created.



Internal Endpoints

IN THIS SECTION

Create Internal Endpoint

1. From the blueprint, navigate to **Staged > Policies> Endpoints > Internal Endpoints** and click **Create Internal Endpoint**.
2. Configure the endpoint as described below:

Parameter	Description
Name	A unique name, 32 characters or fewer. Alphanumeric characters, underscores and dashes only.
Virtual Network	Select the virtual network where the endpoint is located.
IPv4 Subnet	Enter the IPv4 Subnet/CIDR.
Tags (optional)	You can add tags for filtering or grouping beyond membership custom groups or virtual networks (for example “web server”, “db” and so on).

3. Click **Create** to stage the endpoint addition and return to the table view. Validation is performed to ensure that the IP address is within the L2 subnet of the virtual network and that no endpoint with the same IP address is within the same routing zone.

Update Internal Endpoint

1. From the blueprint, navigate to **Staged > Policies > Endpoints > Internal Endpoints** and click the **Edit** button for the endpoint to update.
2. Make your changes.
3. Click **Update** to stage the endpoint change and return to the table view.

Delete Internal Endpoint

1. From the blueprint, navigate to **Staged > Policies > Endpoints > Internal Endpoints** and click the **Delete** button for the endpoint to delete.
2. Click **Delete** to stage the endpoint removal and return to the table view.

External Endpoints

IN THIS SECTION

Create External Endpoint

1. From the blueprint, navigate to **Staged > Policies > Endpoints > External Endpoints** and click **Create External Endpoint**.
2. Configure the endpoint as described below:

Parameter	Description
Name	A unique name, 32 characters or fewer. Alphanumeric characters, underscores and dashes only.
IPv4 Subnet	Enter the IPv4 Subnet/CIDR.
Tags (optional)	You can add tags for filtering or grouping beyond membership custom groups or virtual networks (for example “web server”, “db” and so on).
Enforcement Points (optional)	Enforcement points are supported on external-facing interfaces on border leaf devices only. They are external-facing points where access lists that involve external endpoints are applied. Any external generic system, external connectivity points and enforcement groups can be added.

3. Click **Create** to stage the endpoint addition and return to the table view.

Update External Endpoint

1. From the blueprint, navigate to **Staged > Policies> Endpoints > External Endpoints** and click the **Edit** button for the endpoint to update.
2. Make your changes.
3. Click **Update** to stage the endpoint change and return to the table view.

Delete External Endpoint

1. From the blueprint, navigate to **Staged > Policies> Endpoints > External Endpoints** and click the **Delete** button for the endpoint to delete.
2. Click **Delete** to stage the endpoint removal and return to the table view.

Enforcement Points

Enforcement points are supported on external-facing interfaces on border leaf devices only. They are automatically created when you add external generic systems or external connectivity points to a blueprint.

From the blueprint, navigate to **Staged > Policies > Endpoints > Enforcement Points** to go to enforcement points.

Endpoint Groups

IN THIS SECTION

Create Endpoint Group

1. From the blueprint, navigate to **Staged > Policies > Endpoints > Endpoint Groups** and click **Create Endpoint Group**.
2. Configure the endpoint group as described below:

Parameter	Description
Name	A unique name, 32 characters or fewer. Alphanumeric characters, underscores and dashes only
Type	Select the type of endpoint group to create: **Internal Endpoint Group**, **External Endpoint Group**, or **Enforcement Point Group**.

(Continued)

Parameter	Description
Members	Depending on the type of endpoint group you are creating, options for selecting members are presented. • **Internal Endpoint Group** - Select multiple internal endpoints or other internal endpoint groups. • **External Endpoint Group** - Select multiple external endpoints or other external endpoint groups, then select enforcement points or enforcement point groups to associate with the external endpoint group. • **Enforcement Points Group** - Select multiple enforcement points or other enforcement point groups.

3. Click **Create** to stage the endpoint group addition and return to the table view.

Update Endpoint Group

1. From the blueprint, navigate to **Staged > Policies > Endpoints > Endpoint Groups** and click the **Edit** button for the endpoint group to update.
2. Make your changes.
3. Click **Update** to stage the endpoint group change and return to the table view.

Delete Endpoint Group

1. From the blueprint, navigate to **Staged > Policies > Endpoints > Endpoint Groups** and click the **Delete** button for the endpoint group to delete.
2. Click **Delete** to stage the endpoint group removal and return to the table view.

Security Policies

IN THIS SECTION

Security Policies

IN THIS SECTION

Security Policy Overview

Endpoint connectivity is determined by reachability (the correct forwarding state in the network) and security (connectivity must be permitted). Policies must be specified between L2 and L3 domains and between more granular L2/L3 IP endpoints. Security policies allow you to permit or deny traffic between the more granular endpoints. They control inter-virtual network traffic (ACLs on SVIs) and external-to-internal traffic (ACLs in border leaf devices, external endpoints only). ACLs are rendered in the appropriate device syntax and applied on enforcement points. Adding a new VXLAN Endpoint (for example, adding a rack or adding a leaf to a virtual network) automatically places the ACL on the virtual network interface. Adding a new generic system External Connectivity Point (ECP) (enforcement point) automatically places ACL for external endpoint groups. You can apply security policies to Layer 2 IPv4-enabled blueprints (IPv6 is not supported). For supported devices, refer to the **Connectivity (from Leaf Layer)** table in the Feature Matrix in the Reference section.

Security policies consist of a source point (subnet or IP address), a destination point (subnet or IP address), and rules to allow or deny traffic between those points based on protocol. Rules are stateless, meaning responses to allowed inbound traffic are subject to the rules for outbound traffic (and vice versa).

Rules can include traffic logging. The ACL is configured to log matches using whatever mechanism is supported on the device. Log configuration is local to the network device; It's not on the Apstra server. Parsing these logs is outside the scope of this document.

For a bi-directional security policy, you would create two instances of the policy, one for each direction.

You can apply more than one policy to each subnet/endpoint, which means the ordering of rules has an impact on behavior. An implicit hierarchy exists between routing zones, virtual networks, and IP endpoints, so you must consider how policies are applied at different levels of hierarchy. When one rule's match set contains the other's match set (full containment), the rules can conflict. You can set the rules to execute more specific rules first (“exception” focus/mode) or less specific first (“override” focus/mode).

Rules can also conflict when there is a full containment situation between the rules but the action is the same. In this case, there is potential for compression by using the less specific rule, and the more specific rule becomes a “shadow” rule. When conflicting rules are detected, you are alerted and shown the resolution.

A few cases where conflicting rules are identified are described below:

- Rules in policies between different pairs of IP endpoints (even if one is common to both pairs) are non-overlapping given that the pairs of IP addresses are different. This causes a disjoint match set from a source IP / destination IP perspective (different “IP signature”).
- Rules in policies between the same IP endpoints can overlap fields (such as destination port); Apstra software checks for this.
- Rules in policies between different pairs of virtual networks (even if one virtual network is common to both pairs) are non-overlapping given that the pairs of subnets are different. This causes a disjoint match set from the source IP / destination IP perspective (different “IP signature”).
- Rules in policies between the same virtual networks can overlap fields (such as destination port); Apstra software checks for this.
- When IP endpoint groups are used, they result in a set of IP endpoint pairs so the above discussion related to IP endpoint pairs applies.
- Rules in policies between a pair of IP endpoints and a pair of parent virtual networks have containment from an IP signature perspective. Apstra software analyzes destination port / protocol overlap and classifies it as full-containment or non-full-containment conflict.

- Rules in policies between a pair of IP endpoints and a pair of virtual networks where at least one virtual network is not parent are non-conflicting (different "IP signature").
- Rules in policies between a pair of IP endpoints and an IP endpoint - virtual network pair where the virtual network is a parent have full containment from an IP signature perspective; Apstra software analyzes the remaining fields.
- Rules in policies that contain external IP endpoints or endpoint groups must be analyzed from an IP signature perspective as external points are not bound by any hierarchical assumptions.
- A routing zone is a set of virtual networks and IP endpoints so the above discussions apply.

Endpoints are not supported in security policies when:

- Source point is an external endpoint or external endpoint group
- Destination point is internal (internal endpoint, internal endpoint group, virtual network, routing zone)

To make composition tractable, both from an analysis point of view as well as from comprehending the resulting composition it may be useful to limit the number of security policies that may apply to any given endpoint/group.

Security Policy Parameters

Security policies include the following details:

Parameter	Description
Name	32 characters or fewer, underscore, dash and alphanumeric characters only
Description	optional
Enabled	• **ON** to enable security policy (default) • **OFF** to disable security policy
Tags	optional

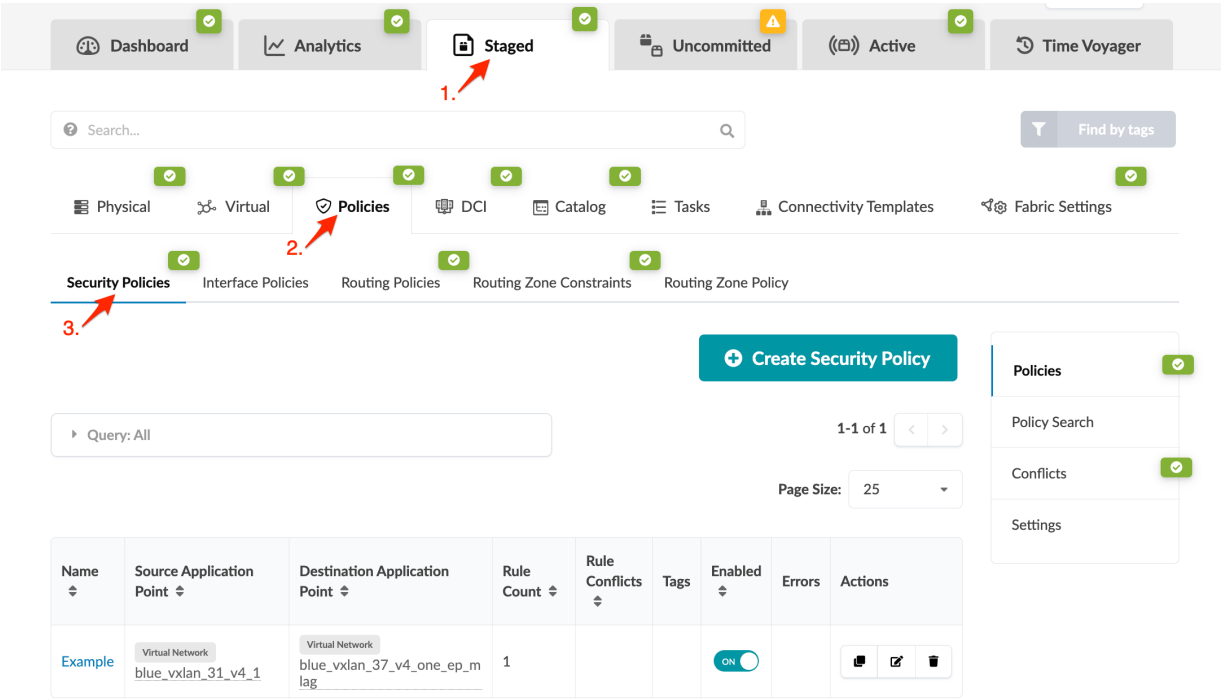
(Continued)

Parameter	Description
Source Point Type	• Internal Endpoint (associated with VNs - contain IP /32 address) • External Endpoint (contains /32 or subnet) • External Endpoint Group • Internal Endpoint Group • Virtual Network (contains subnet) • Routing Zone (logical collection of all virtual networks and internal IP endpoints)
Source Point	• Internal Endpoint • External Endpoint • External Endpoint Group • Internal Endpoint Group • Virtual Network • Routing Zone
Destination Point Type	Source point (previously created)
Destination Point	Destination point (previously created)
Rule Actions	• Deny • Deny & Log • Permit • Permit & Log

(Continued)

Parameter	Description
Rule Protocols	• TCP • UDP • IP • ICMP
Source Port	For TCP and IP protocols
Destination Port	For TCP and IP protocols

From the blueprint, navigate to **Staged > Policies > Security Policies > Policies** to go to security policies. You can create, clone, edit and delete security policies.



Create Security Policy

Before creating security policies, create "routing zones" on page 286, "virtual networks" on page 251, "endpoints and endpoint groups" on page 373, in that order. They are the basis for creating security policies.

To create security policies:

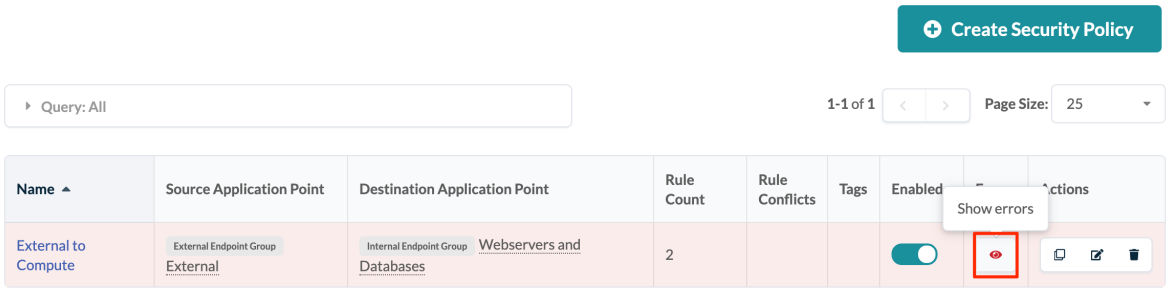
1. From the blueprint, navigate to **Staged > Policies > Security Policies > Policies** and click **Create Security Policy**.
2. Enter a name, and if you want the policy to be enabled leave the default. Otherwise, click the **Enabled** toggle to disable it.
3. Select a source point type, and enter the source point.
4. Select a destination point type, and enter the destination point.
5. Click **Add Rule**, then enter a name and (optional) description.
6. Select an action from the drop-down list (Deny, Deny & Log, Permit, Permit & Log).
7. Select a protocol from the drop-down list (TCP, UDP, IP ICMP).
8. If you selected TCP or UDP, enter a port (or port range) for source and destination. (If you created "TCP/UDP port aliases" on page 920, they appear in the drop-down list).
9. To add another rule, click **Add Rule** and configure as above.

> **NOTE**: To the right of the **Add Rule** button you can automatically create a blocklist-type policy by clicking **Deny All** or an allowlist-type policy by clicking **Permit All**.

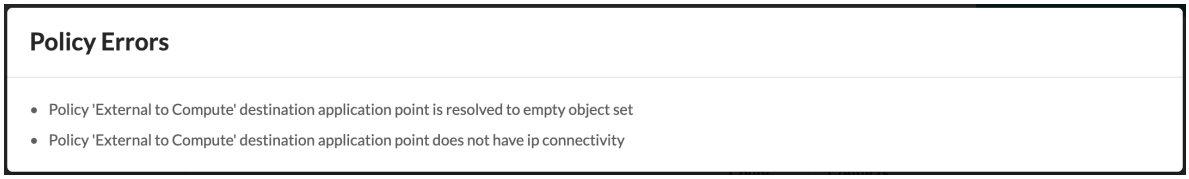
10. You can adjust the rule order by clicking the **Move up** or **Move Down** buttons in each rule.
11. Click **Create** to stage the policy and return to the table view.

Policy Errors

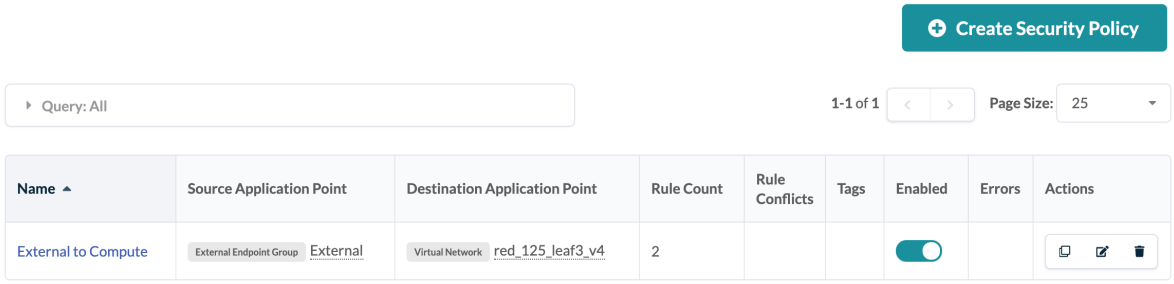
1. Check the security policy in the table view for errors, which are highlighted in red.



2. To see details, click the **Show errors** button.



3. When you resolve errors, the policy is no longer highlighted red and the **Errors** field is blank.



To activate staged changes, commit them to the blueprint.

Edit Security Policy

1. From the left navigation menu, navigate to **Staged > Policies > Security Policies > Policies** and click the **Edit** button for the policy to edit.
2. Make your changes.
3. Click **Edit** to stage the changes and return to the table view.

Delete Security Policy

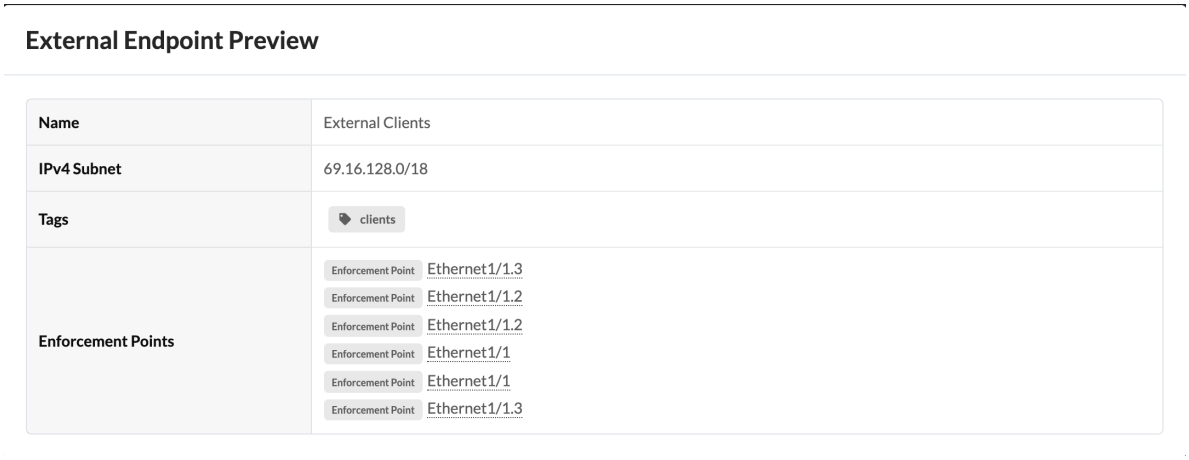
1. From the left navigation menu, navigate to **Staged > Policies > Security Policies > Policies** and click the **Delete** button for the policy to delete.
2. Click **Delete** to stage the deletion and return to the table view.

Security Policy Search

You can find security policies that are applied to specific subnets or points.

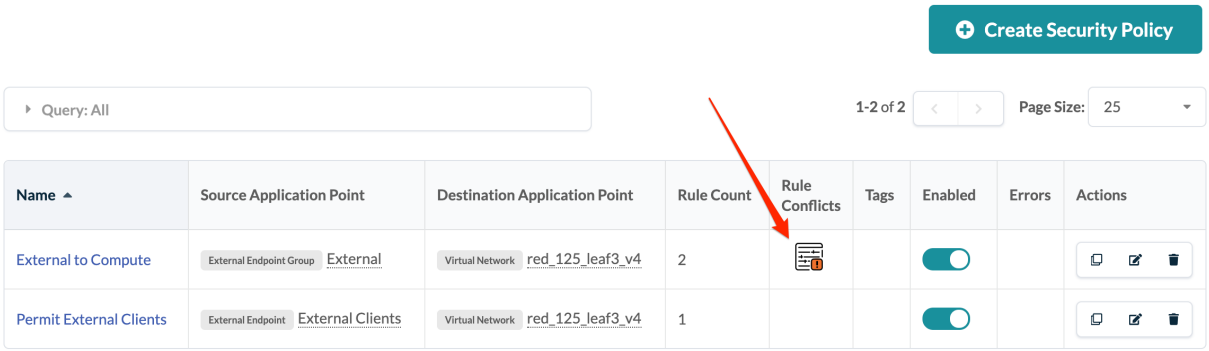
1. From the blueprint, navigate to **Staged > Policies > Security Policies > Policy Search**.
2. Select a source point type and enter a subnet or source point, as applicable.
3. Select a destination point type and enter a subnet or source point, as applicable.

4. Click **Search** to display associated security policies.

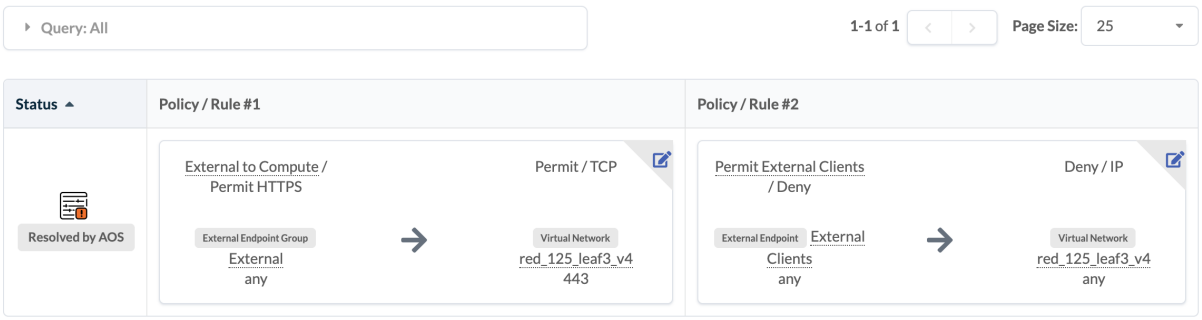


Security Policy Conflicts

From the blueprint, navigate to **Staged > Policies > Security Policies > Conflicts** to see any conflicts that have been detected (**Rule Conflicts** column). Conflicts are resolved automatically whenever possible. By default, more specific policies are applied before less specific ones, but you can change these security policy settings. To see conflict details, click the icon in the **Rule Conflicts** column.



If the conflict was resolved automatically, **Resolved by AOS** appears in the **Status** column.



Security Policy Settings

You can configure how you want to resolve conflicts and whether to permit or deny traffic.

1. From the blueprint, navigate to **Staged > Policies > Security Policies > Settings**.
2. Select options as appropriate.
   - Conflict resolution
     - **More specific first** - more specific IP policy is used (default)
     - **More generic first** - less specific IP policy is used
     - **Disabled** - disables conflict resolution
   - Default action
     - **Permit** - permits traffic (default)
     - **Permit & Log** - permits traffic and logs it
     - **Deny** - denies traffic
     - **Deny & Log** - denies traffic and logs it
3. Click **Save Changes** to stage the changes.

To activate staged changes, commit them to the blueprint.

SEE ALSO

Commit / Revert Changes to Blueprint | 599

Interface Policies

IN THIS SECTION

Interface Policies

IN THIS SECTION

802.1X Server Port Authentication

IEEE 802.1X is an IEEE Standard for network port-based Network Access Control. It is part of the IEEE 802.1 group of networking protocols. It provides an authentication mechanism to devices wishing to attach to a LAN.

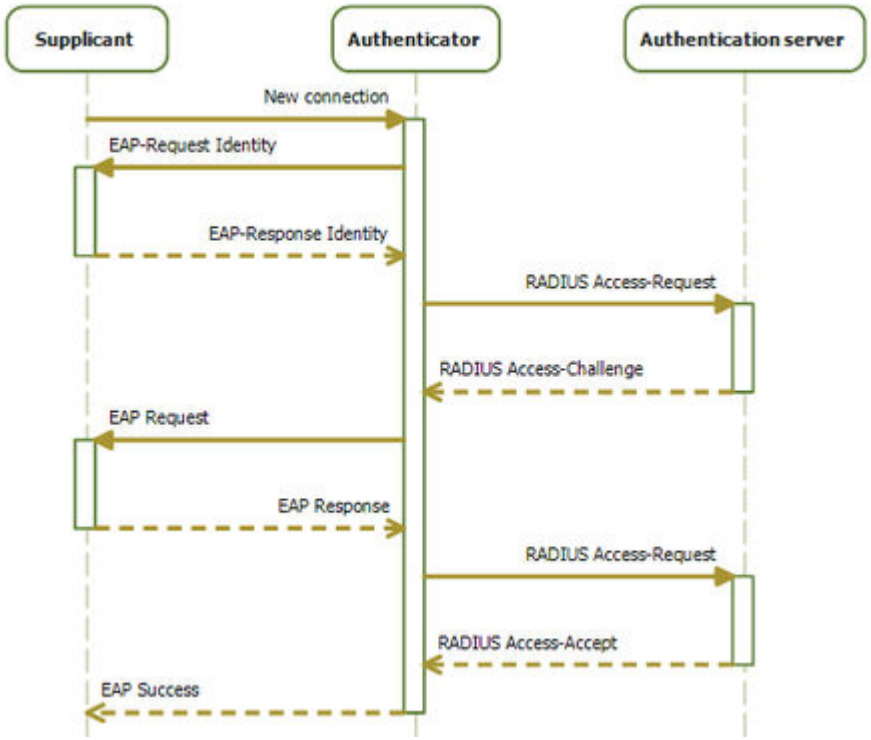
IEEE 802.1X defines the encapsulation of the Extensible Authentication Protocol (EAP) over IEEE 802, which is known as "EAP over LAN" or EAPOL.

802.1X authentication involves three parties: a supplicant, an authenticator, and an authentication server. The **supplicant** is a client device (such as a server) that wishes to attach to the LAN. The term 'supplicant' is also used interchangeably to refer to the software running on the client that provides credentials to the authenticator. The **authenticator** is a network device which provides a data link between the client and the network and can allow or block network traffic between the two, such as an Ethernet switch or wireless access point; and the **authentication server** is typically a trusted server that can receive and respond to requests for network access, and can tell the authenticator if the connection is to be allowed, and various settings that should apply to that client's connection or setting. Authentication servers typically run software supporting the RADIUS and EAP protocols. In some cases, the authentication server software may be running on the authenticator hardware.

The authenticator acts as a security guard to a protected network. The supplicant (i.e., client device) is not allowed access through the authenticator to the protected side of the network until the supplicant's identity has been validated and authorized. With 802.1X port-based authentication, the supplicant must initially provide the required credentials to the authenticator - these will have been specified in advance by the network administrator and could include a user name/password or a permitted digital certificate. The authenticator forwards these credentials to the authentication server to decide whether access is to be granted. If the authentication server determines the credentials are valid, it informs the authenticator,

which in turn allows the supplicant (client device) to access resources located on the protected side of the network.

Extensions to 802.1X can also allow the authentication server to pass port-configuration options to the authenticator. An example is using RADIUS value-pair attributes to pass a VLAN ID, allowing the supplicant access to one of several VLANs.



(Source: Wikipedia, revised by Apstra)

You can manage 802.1X configuration on network devices with 802.1X server port authentication, a collection of interface policy settings.

802.1X interface policy is supported on Junos (as a Tech Preview) and Arista EOS physical network devices only. Juniper Evolved does not at this time support this feature.

NOTE: 802.1X interface policy on Junos has been classified as a Juniper Apstra Technology Preview feature. These features are "as is" and voluntary use. Juniper Support will attempt to resolve any issues that customers experience when using these features and create bug reports on behalf of support cases. However, Juniper may not provide comprehensive support services to Tech Preview features.

For additional information, refer to the "Juniper Apstra Technology Previews" on page 1781 page or contact "Juniper Support" on page 1374.

This policy setting enables the network to require L2 servers in a blueprint to authenticate to a RADIUS server before being provided access to the network.

The network operator may require clients to authenticate using EAP-TLS, Certificates, simple username & password, or MAC Authentication bypass.

> **NOTE**: Support for encryption protocols, certificates, EAP, is negotiated between RADIUS supplicant and RADIUS server, and is not controlled by the switch.

After authentication occurs, a RADIUS server may optionally set a VLAN ID attribute at authentication time to move the supplicant into a defined VLAN, known by a leaf-specific VLAN ID.

This section describes the necessary tasks to create Interface Policies to be used with 802.1X server port authentication and dynamic VLAN allocation.

Common Scenarios

The following are some common scenarios for 802.1X port authentication.

Device supports 802.1X, credentials and VLAN are configured in Radius

1. Device (Supplicant) connects to a port
2. Switch (Authenticator) mediates EAP negotiation between supplicant and Radius (Authentication Server)
3. Upon authentication, Radius sends an Access-Accept message to the switch which includes the VLAN number for the device
4. The switch adds the device port to the specified VLAN

Device supports 802.1X, but credentials are not configured in Radius

1. Device (Supplicant) connects to a port
2. Switch (Authenticator) mediates EAP negotiation between supplicant and Radius (Authentication Server)
3. Finding no credential for the supplicant, Radius sends an Access-Reject message to the switch
4. The switch adds the device port to a designated Fallback (aka AuthFail/Parking) VLAN

Device does not support 802.1X, but the device MAC address is configured in Radius

1. Device (Non-Supplicant) connects to a port
2. Switch (Authenticator) does not receive a reply to its EAP-Request Identity message, indicating no 802.1X support
3. Switch authenticates device's MAC address to Radius (Authentication Server)
4. Radius sends an Access-Accept message to the switch which includes the VLAN number for the device
5. The switch adds the device port to the specified VLAN

Device does not support 802.1X, and device MAC address is not configured in Radius

1. Device (Non-Supplicant) connects to a port
2. Switch (Authenticator) does not receive a reply to its EAP-Request Identity message, indicating no 802.1X support
3. Switch authenticates device's MAC address to Radius (Authentication Server)
4. Radius does not find a record for the MAC address
5. Radius sends an Access-Reject or Access-Accept message to the switch without a VLAN
6. The switch adds the device port to a designated Fallback (aka AuthFail/Parking) VLAN

802.1X Interface Policy Workflow

1. Create virtual networks (e.g. Data VLAN, Fallback VLAN, Dynamic VLAN)
2. Create AAA servers
3. Create 802.1X interface policy
4. Assign ports and fallback VLANs

Create Virtual Networks for Interfaces

Create virtual networks for the interface policy per the table below. We suggest creating these virtual networks with a consistent VLAN ID among all leaf devices (instead of using a resource pool). For more information about creating VLANs, see "Virtual Networks" on page 258.

Parameter	Description
Data VLAN (assigned to ports)	Interfaces will have 802.1X configuration if at least one VLAN is assigned to the port. If a port does not have any VLANs assigned, 802.1X configuration will not be rendered on the interface. The interface will be configured as a routed port.
Dynamic VLAN (optional, assigned to leaf devices, not ports)	The RADIUS server itself optionally chooses the VLAN ID dynamically when the user (supplicant) is authenticated and authorized. Apstra software does not have control over Dynamic VLAN assignment. This decision is made by RADIUS configuration, not by the switch configuration.
Fallback VLAN (optional, assigned to leaf devices, not ports)	Fallback VLAN can be assigned to the user (supplicant) in case of authentication failure. For fallback, the VLAN is controlled by the switch configuration. A RADIUS dynamic VLAN or fallback VLAN must exist on the switch, but there is no requirement to bind any endpoints to that VLAN. It only needs to exist on the switch.

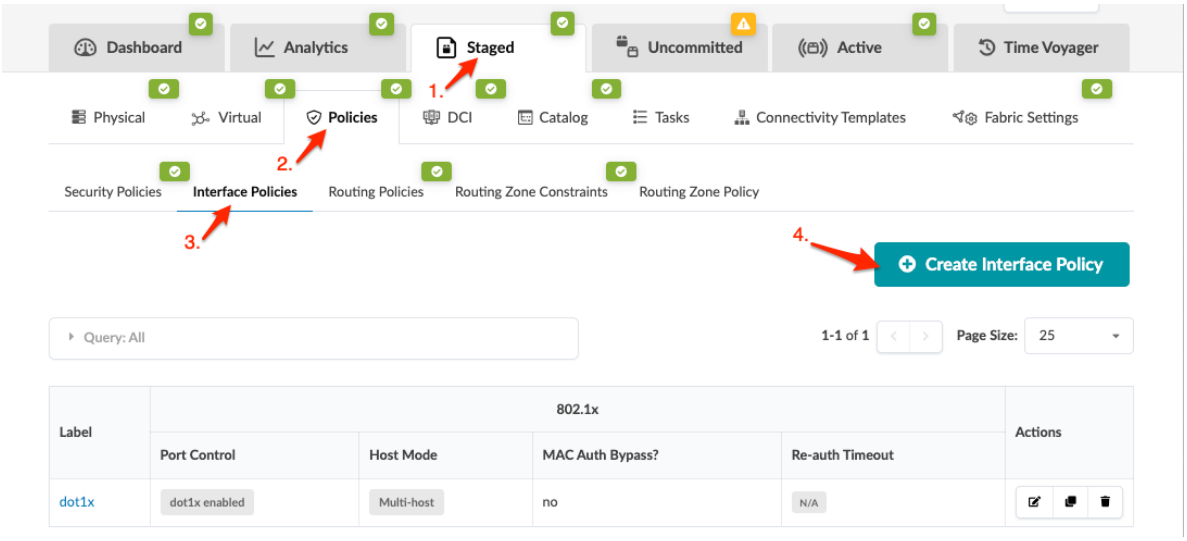
Create AAA Server for Interface Policy

Create the AAA server. For more information, see "AAA Servers (Blueprint)" on page 447.

Create 802.1x Interface Policy

You must create the policy before you can assign interfaces or fallback VLANs to it.

1. From the blueprint, navigate to **Staged > Polices > Interface Policies** and click **Create Interface Policy**.



2. Enter a name and select **802.1x** from the drop-down list.

3. Select the **Port Control**.

   - **dot1x enabled** - Requires ports to authenticate EAPOL before being given access to the network.

   - **Deny access** - Completely blocks the port; no network access is permitted. No other parameters are needed. Example: as a quarantine configuration to quickly deactivate ports that may be infected.

4. Select the **Host Mode**.

   - **Multi-host**** (default) - Allows all MAC addresses on the port to authenticate after the first successful authorization. After the first host deauthorizes, all MACs on the port are de-authenticated.

   - **Single-host** - Permits a single host to authenticate; all other MACs are not permitted.

5. If you want to enable **MAC Auth Bypass** on Arista EOS, check the **Enabled?** box. Enabling MAC auth bypass allows a switch to send the MAC address to the RADIUS server if the port does not authenticate within the authentication timeout period. MAC Auth bypass (MAB) requests are only sent if the client does not respond to RADIUS requests, or if the client fails authentication.

> **NOTE**: MAC Auth bypass must be configured along with 802.1X port control.

> ⚠ **CAUTION**: MAC auth bypass failure behavior may be different between switch vendors and major switch models.

6. Enter **Re-auth Timeout** (optional) to configure a time period (seconds). Re-authentication timeout causes the switch to request any clients to re-authenticate to the network after the timeout expires. This also re-triggers MAC Auth bypass.

   If re-authentication timeout is not configured, then no related configuration is rendered on the switch. This means the switchport will be whatever the OS vendor default is. If a value is configured, 802.1X re-authentication will be enabled on the port, and a time value will be configured.

7. Click **Create** to create the interface policy and return to the table view.

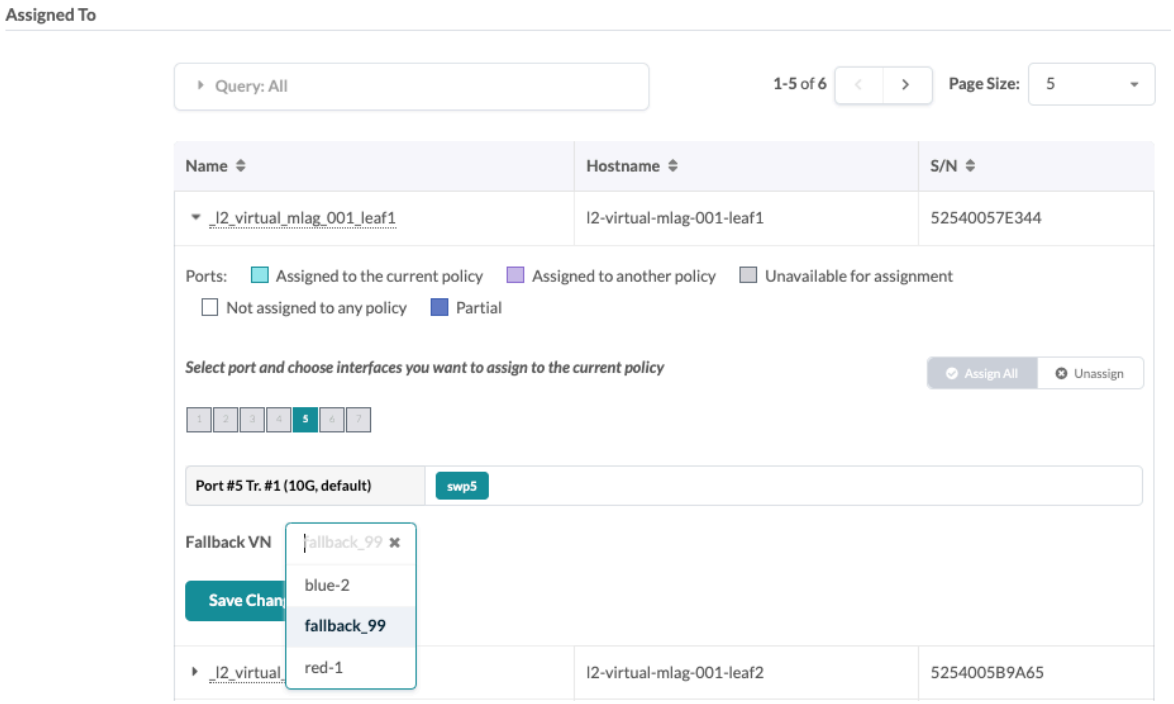
Assign Ports and Fallback VNs to Interface Policy

This steps adds interfaces or dynamic VLANs to the interface policy.

1. From the blueprint, navigate to **Staged > Polices > Interface Policies**, select the interface policy name and scroll down to the **Assigned To** section.

2. **Assign ports and interfaces:** Click leaf names to expand interfaces, then click ports and interfaces to assign them. Note that you cannot assign ports that are assigned to conflicting policies.

3. **Assign fallback VN:** Assigning the fallback virtual network is leaf-specific. To re-use the fallback on multiple leaf devices, you have to assign it to each leaf. Any VN that is assigned to the leaf may be used as a fallback virtual network, there are no restrictions.



4. After the policy is configured, the settings are now visible, including interfaces those settings apply to.

NOTE: AAA, Dot1x, and Dot1x interface configurations are now pushed to the leaf devices. The following is a part of sample config rendered for Arista EOS switch.

```
leaf1#sh running-config section dot1x
logging level DOT1X errors
!
aaa group server radius AOS_RADIUS_DOT1X
   server 172.20.191.5 vrf management
!
aaa authentication dot1x default group AOS_RADIUS_DOT1X
aaa accounting dot1x default start-stop group AOS_RADIUS_DOT1X logging
!
interface Ethernet5
   switchport trunk allowed vlan 99
   switchport mode trunk
   switchport
   ipv6 enable
   ipv6 address auto-config
   ipv6 nd ra rx accept default-route
   dot1x pae authenticator
   dot1x reauthentication
   dot1x port-control auto
   dot1x timeout reauth-period 30
!
..snip..
!
dot1x system-auth-control
dot1x dynamic-authorization
```

Routing Policies

IN THIS SECTION

What are Routing Policies

Routing policies enable you to control (filter) which routes a routing protocol imports into the routing table and which routes a routing protocol exports from the routing table. Routing policies in Apstra include the following details:

Parameter	Description
Name	18 characters or fewer. Alphanumeric, _ and - only.
Import Policy	Specify the Import policies to import, either default route only, all routes, or extra routes only. • **Default** - The default BGP route (0.0.0.0/0, ::/0) is permitted. If you select default here and add extra routes under **Extra Import Routes** below, both the default and the extra routes apply. • **All** - Any BGP route is permitted. • **Extra Only** - Only user-defined extra import routes (see below) are permitted or denied.

(Continued)

Parameter	Description
Extra Import Routes (user-defined)	User-defined import routes. If you want the default route to be included, make sure you've selected **Default** for **Import Policy** (above). If you want only the extra import routes to apply, make sure you've selected **Extra Only** for **Import Policy**. • **Prefix** - IPv4 or IPv6 network address (format: network/prefixlen) or IP address (interpreted as /32 network address). • **GE Mask and LE Mask** - GE Mask matches less-specific prefixes from a parent prefix, up from the GE mask to the prefix length of the route. (IPv4 range: 0-32. IPv6 range: 0-128). If you don't specify GE mask, then make sure that the prefix-list entry is an exact match. You can use this option in combination with LE Mask. GE mask must be longer than the subnet prefix length. If both the LE mask and GE mask are specified, then the LE mask must be greater than the GE mask. • **Action** - Permit or Deny

(Continued)

Parameter	Description
Export Policy	• **Spine Leaf Links** - Exports all spine-leaf (fabric) links within a VRF. EVPN routing zones don't have spine-leaf addressing, so this generated list may be empty. For routing zones of type Virtual L3 Fabric, subinterfaces between spine-leaf are included. • **Spine Superspine Links** - Exports all spine-superspine (fabric) links within the default routing zone (VRF). • **L2 Edge Subnets** - Exports all virtual networks (VLANs) that have L3 addresses within a routing zone (VRF). • **Loopbacks** - Exports all loopbacks within a routing zone (VRF) across spine, leaf, and L3 servers. • **Static Routes** - Exports all subnets in a VRF associated with static routes from all fabric systems to generic systems associated with this routing policy.

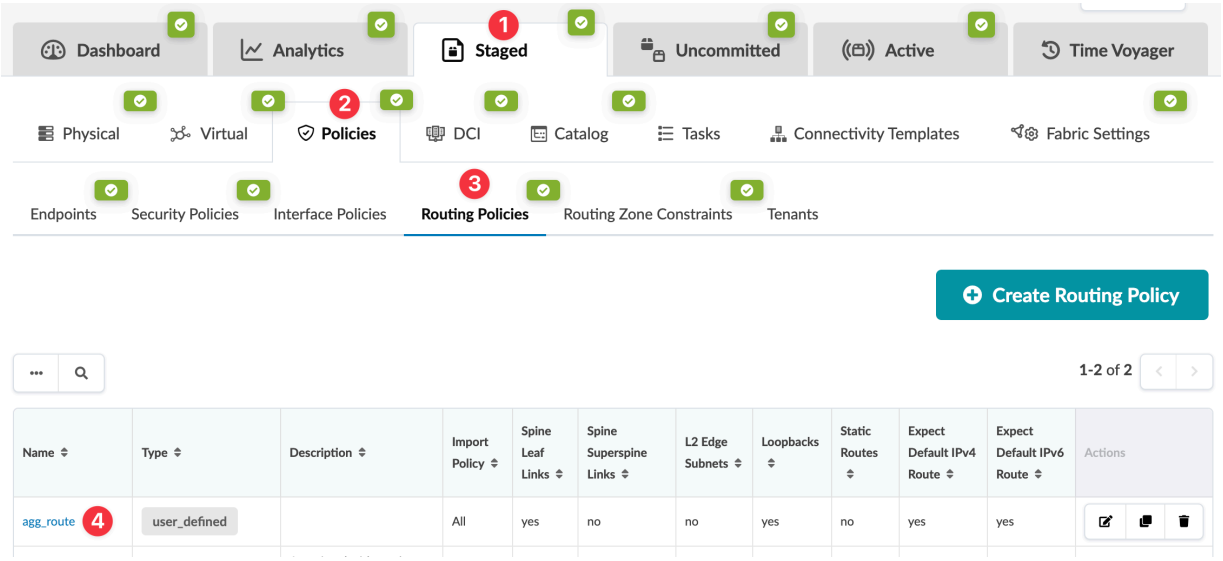
(Continued)

Parameter	Description
Extra Export Routes (user-defined)	User-defined export routes. If you want other export routes to be included, make sure you've selected them under **Export Policy** (above). If you want only the extra export routes to apply, make sure none of the export policies are selected under **Export Policy**. **NOTE**: To enable default route for EVPN host routes, go to **Staged > Fabric Settings > Fabric Policy**. Then, in the **Route Options** section, enable the **Generate EVPN host routes from ARP/IPv6 ND ARP** option. • **Prefix** - IPv4 or IPv6 network address (format: network/prefixlen) or IP address (interpreted as /32 network address). • **GE Mask and LE Mask** - GE Mask matches less-specific prefixes from a parent prefix, up from the GE mask to the prefix length of the route. (IPv4 range: 0-32. IPv6 range: 0-128). If you don't specify GE mask, then the prefix-list entry should be an exact match. You can use this option in combination with LE Mask. GE mask must be longer than the subnet prefix length. If both the LE mask and GE mask are specified, then the LE mask must be greater than the GE mask. • **Action** - Permit or Deny

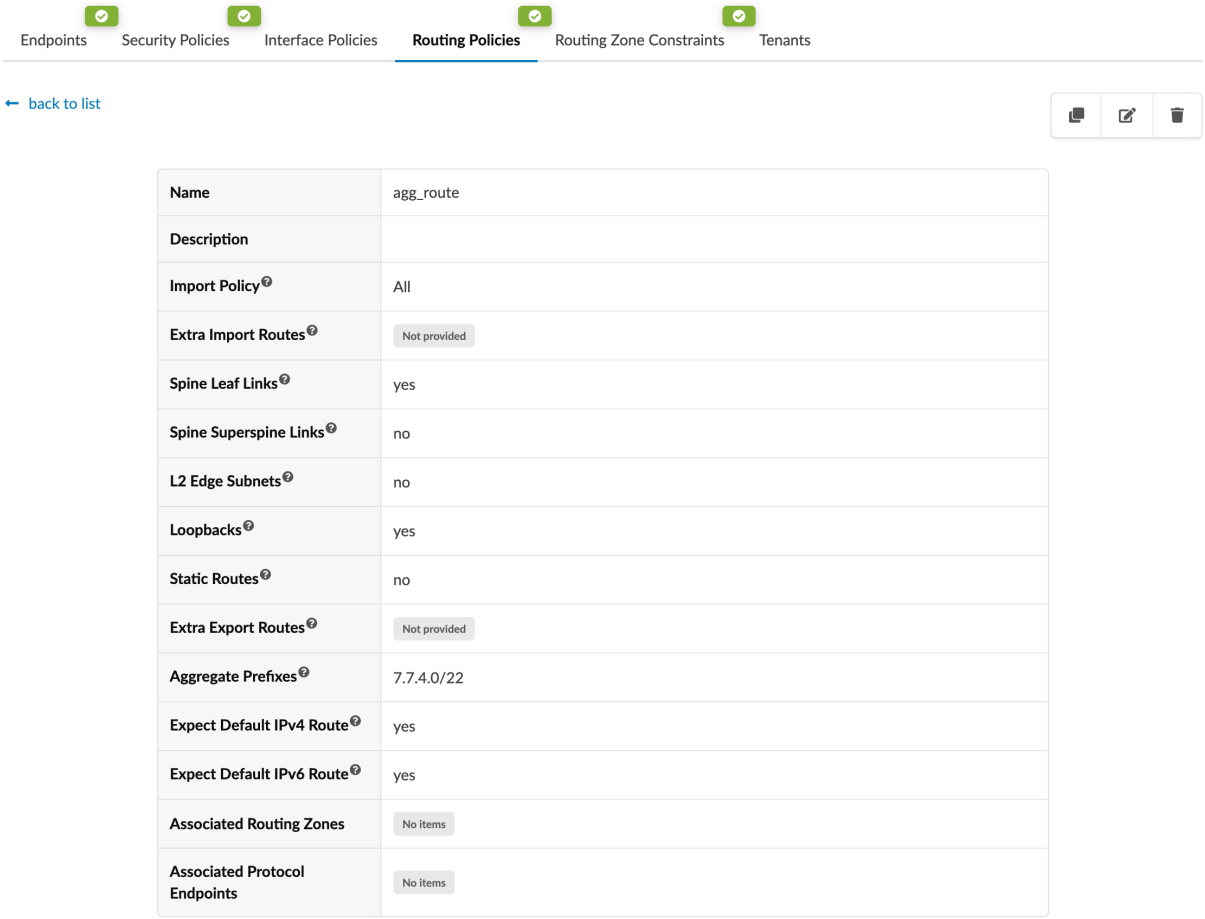
(Continued)

Parameter	Description
Aggregate Prefixes	If you have routing zones associated with your routing policy, and aggregate prefixes are supported on the platform (see the "feature matrix" on page 1481) you can specify aggregate prefixes. These are the BGP aggregate routes to be imported into the routing zone (VRF) on all border switches. The aggregated routes are sent to all generic system peers in a routing zone (VRF). **CAUTION**: Routing policies with aggregate prefixes are applied to the entire routing zone. You cannot configure them individually for BGP sessions (per connectivity point). If you do attempt to apply them via a connectivity template (CT), you could receive the error “Protocol endpoint routing policy aggregate prefixes should be empty”.
Expect Default IPv4 Route	To add the expectation that the default route is used in the default routing zone, select this check box when you create the policy. (This field applies to the default route in the default routing zone only.) Checking this box does not change any configuration; it generates the expectation and raises an anomaly when the default route is not present.
Expect Default IPv6 Route	To add the expectation that the default route is used in the default routing zone, select this check box when you create the policy. (This field applies to the default route in the default routing zone only.) Checking this box does not change any configuration; it generates the expectation and raises an anomaly when the default route is not present.
Associated Routing Zones	Lists any routing zones that are associated with the routing policy.
Associated Protocol Endpoints	Lists any protocol endpoints that are associated with the routing policy.

From the blueprint, navigate to **Staged > Policies > Routing Policies** to go to routing policies in the blueprint.



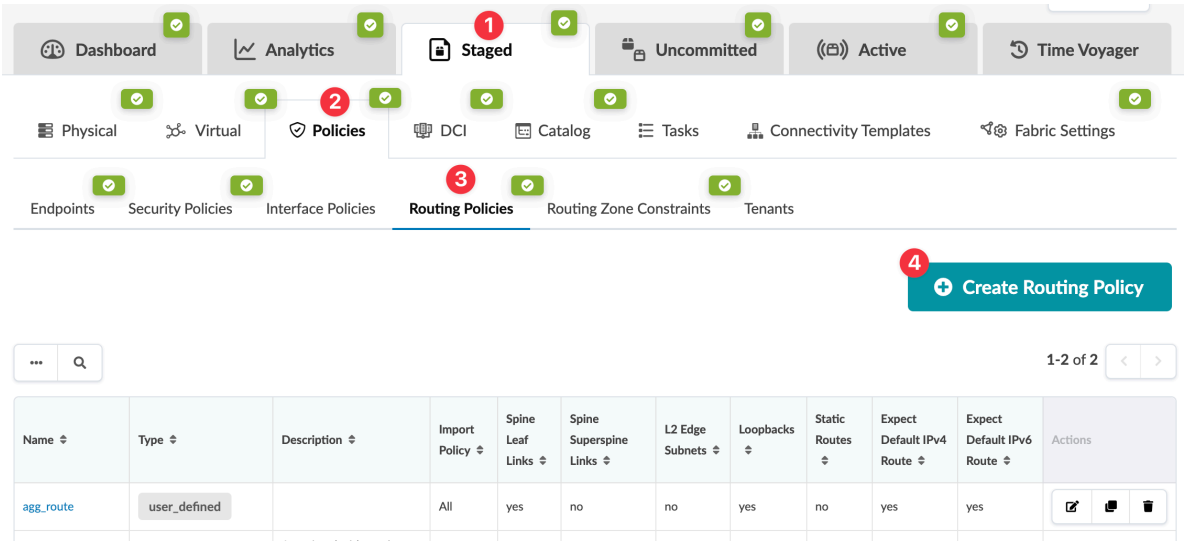
The default routing policy (not shown in table) is associated with the default routing zone. To see details of a routing policy, click its name.



You can't change the default routing policy, but you can create, clone, edit, and delete other routing policies as described in subsequent pages.

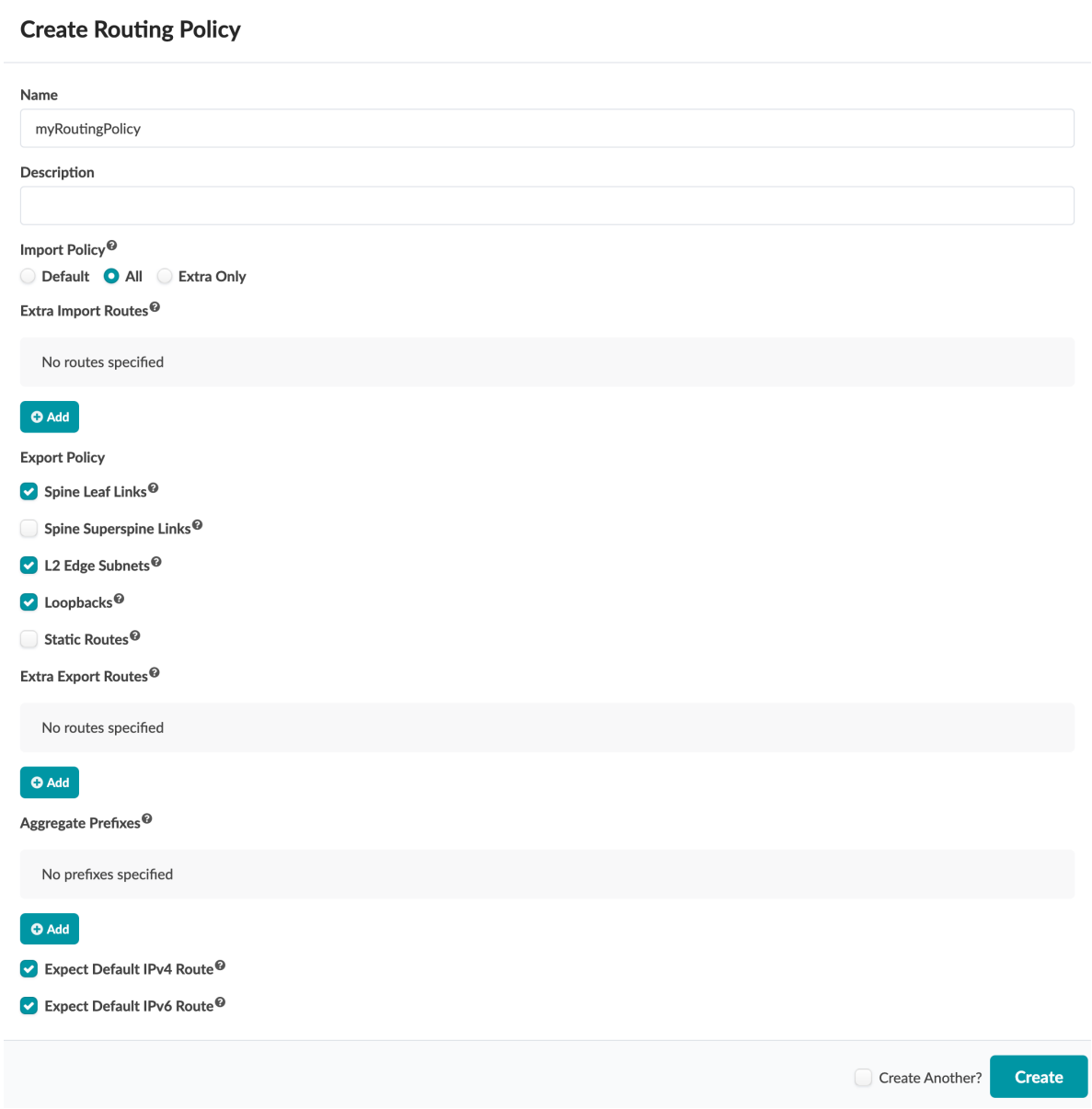
Create Routing Policy

1. From the blueprint, navigate to **Staged > Policies > Routing Policies** and click **Create Routing Policy** (or to copy an existing policy and customize it, click the **Clone** button (between the **Edit** and **Delete** buttons) for the policy to copy.)

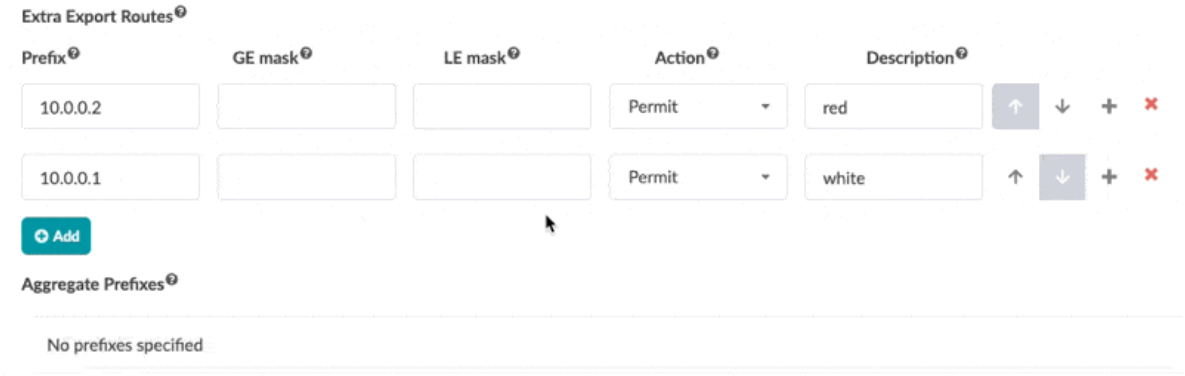


The **Create Routing Policy** dialog opens (or the **Clone Routing Policy** dialog opens, as applicable).

2. Configure your policy. For parameter details, see "What are Routing Policies" on page 396.



For extra routes, (both import and export) you can add route descriptions and reorder them (as of Apstra version 5.0.0).

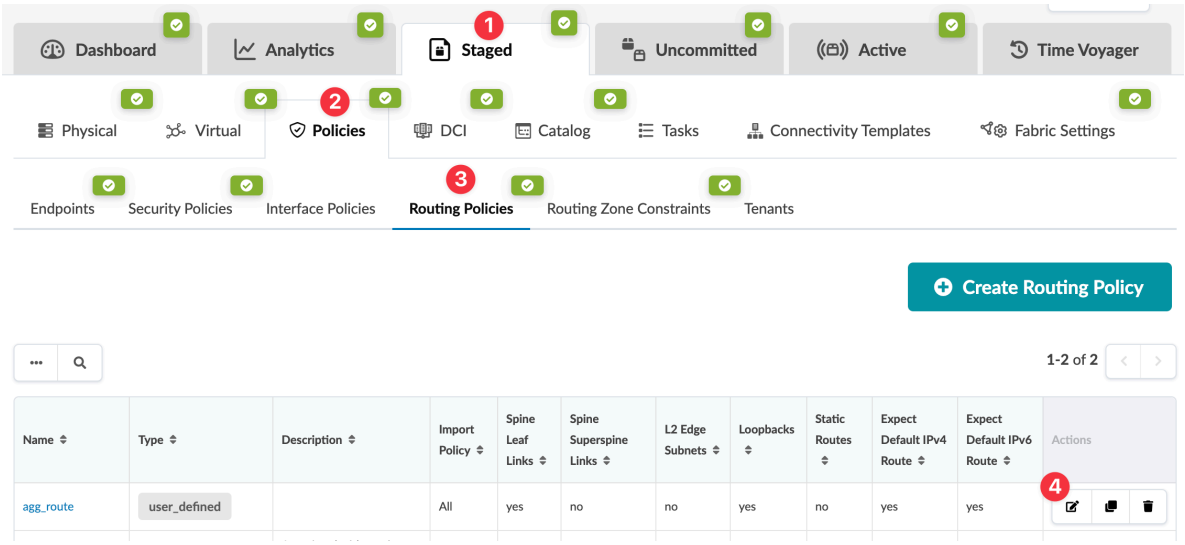


3. Click **Create** to stage the policy addition and return to the table view.

When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

Update Routing Policy

1. From the blueprint, navigate to **Staged > Policies > Routing Policies**, then in the table, click the **Edit** button for the policy to update. (You can also update a routing policy from the details view of that routing policy.)

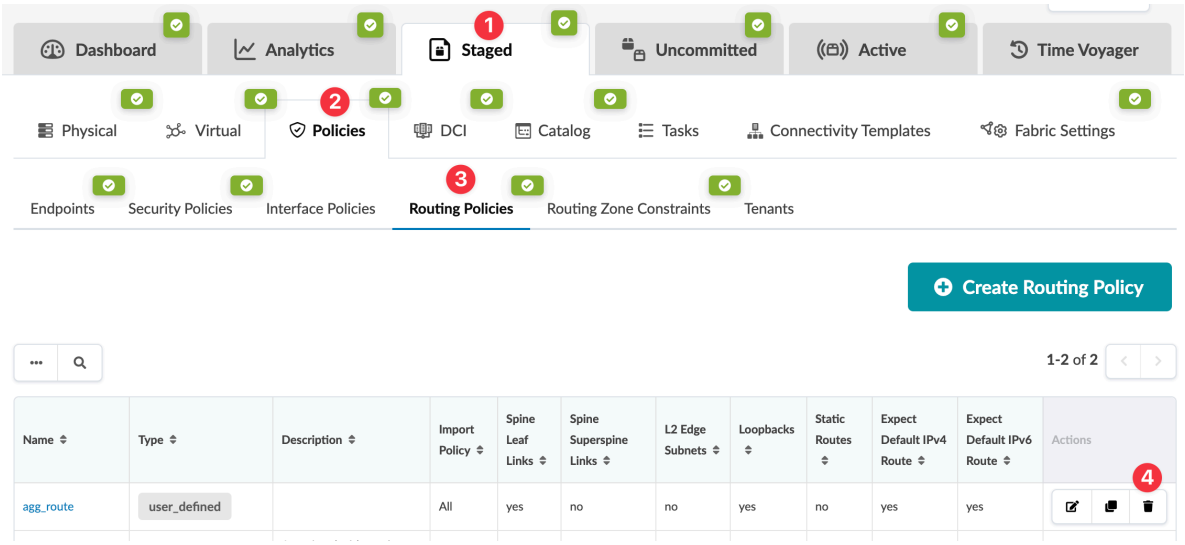


   The **Edit Routing Policy** dialog opens.
2. Make your changes. For parameter details, see "What are Routing Policies" on page 396.
3. Click **Update** (bottom-right) to stage the policy update and return to the table view.

When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

Delete Routing Policy

1. From the blueprint, navigate to **Staged > Policies > Routing Policies**, then in the table, click the **Delete** button for the policy to delete. (You can also delete a routing policy from the details view of a routing policy.)



   The **Delete this resource?** dialog opens.

2. Click **Delete** to stage the policy removal and return to the table view.

When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

Routing Zone Constraints

IN THIS SECTION

Routing Zone (VRF) Constraints

IN THIS SECTION

Routing zone constraints allow you to constrain server-facing interfaces that connect to specific routing zones. Day-2 operators would be prevented from connecting a server to the wrong network, and assure that a given server never gets added to the wrong network. The constraint can be defined in various ways such as a list of allowed VRFs, a list of excluded VRFs, a maximum number of VRFs allowed, and so on. Once the constraint is defined, you can enforce the constraint on server-facing interfaces using connectivity templates of the type **Routing Zone Constraint**.

Create Routing Zone Groups (Optional)

If you want to constrain more than one routing zone to a single port, you can group them, then specify the group as a constraint when you create the routing zone constraint policy.

1. From the blueprint, navigate to **Staged > Virtual > Routing Zone Groups** and click **Create Routing Zone Group**.
2. Enter a group name and (optional) tags.
3. In the **Routing Zone** drop-down list, select a routing zone to add to the group and click **Add**. The routing zone is added to the **Members** list.
4. Repeat the previous step until you've added all the routing zones that you want in the group.
5. Click **Create** to create the group and return to the table view.

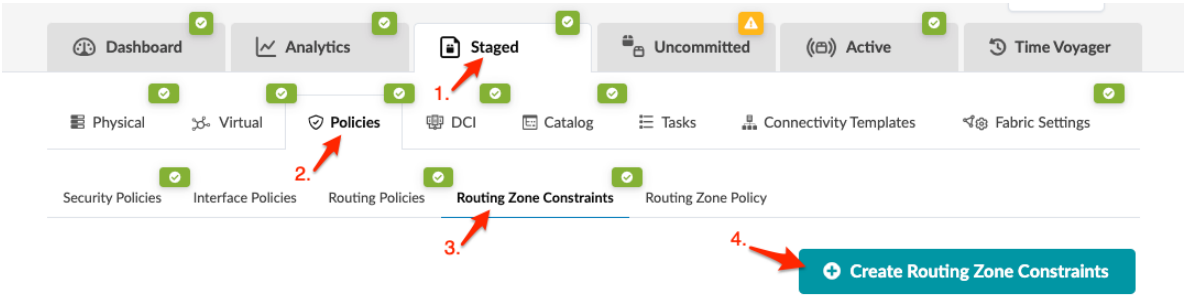
Create Routing Zone Constraint Policy

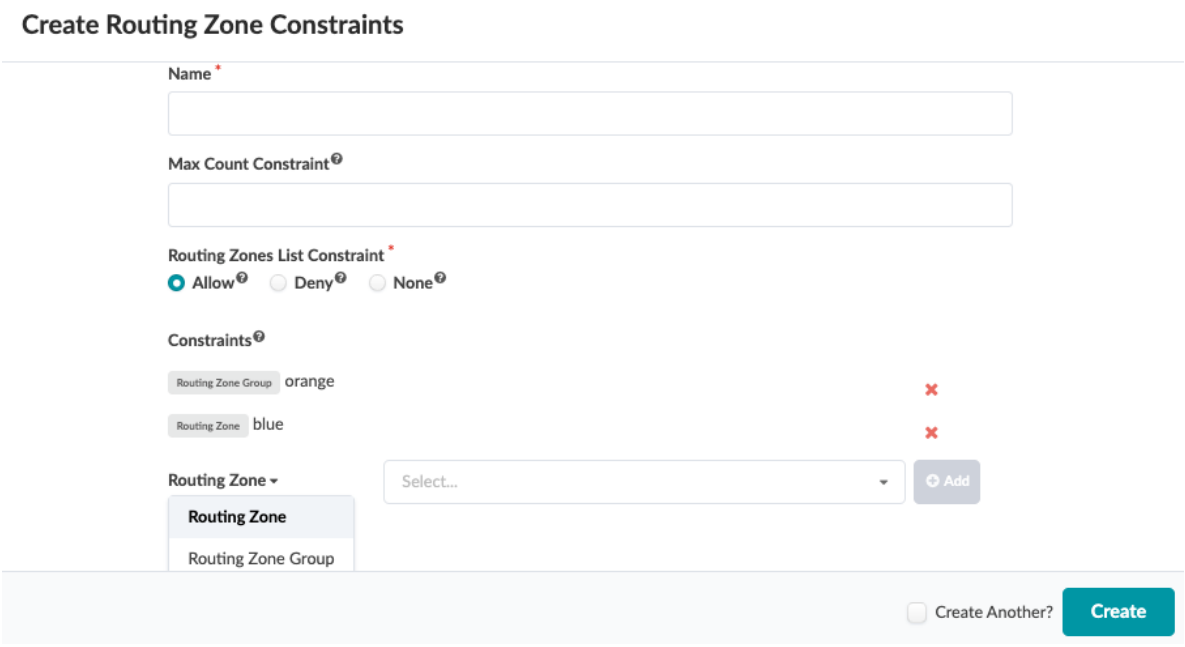
You can create a routing zone constraint policy, then later when you create a connectivity template you can apply the policy to an application point. Some examples of how you could constrain VRFs include:

- One VRF maximum
- Any VRF except Management
- Only VRFs Blue and Red

- Only VRF Group Orange

1. From the blueprint, navigate to **Staged > Policies > Routing Zone Constraints** and click **Create Routing Zone Constraints**.





2. Enter a name and (optional) maximum number of routing zones that the application point can be part of.
3. Set the (optional) **Routing Zones List Constraint**.
   a. **Allow** - only allow the specified routing zones (add specific routing zones to allow)
   b. **Deny** - denies allocation of specified routing zones (add specific routing zones to deny)
   c. **None** - no additional constraints on routing zones (any routing zones)
4. Click **Create** to create the policy and return to the table view.

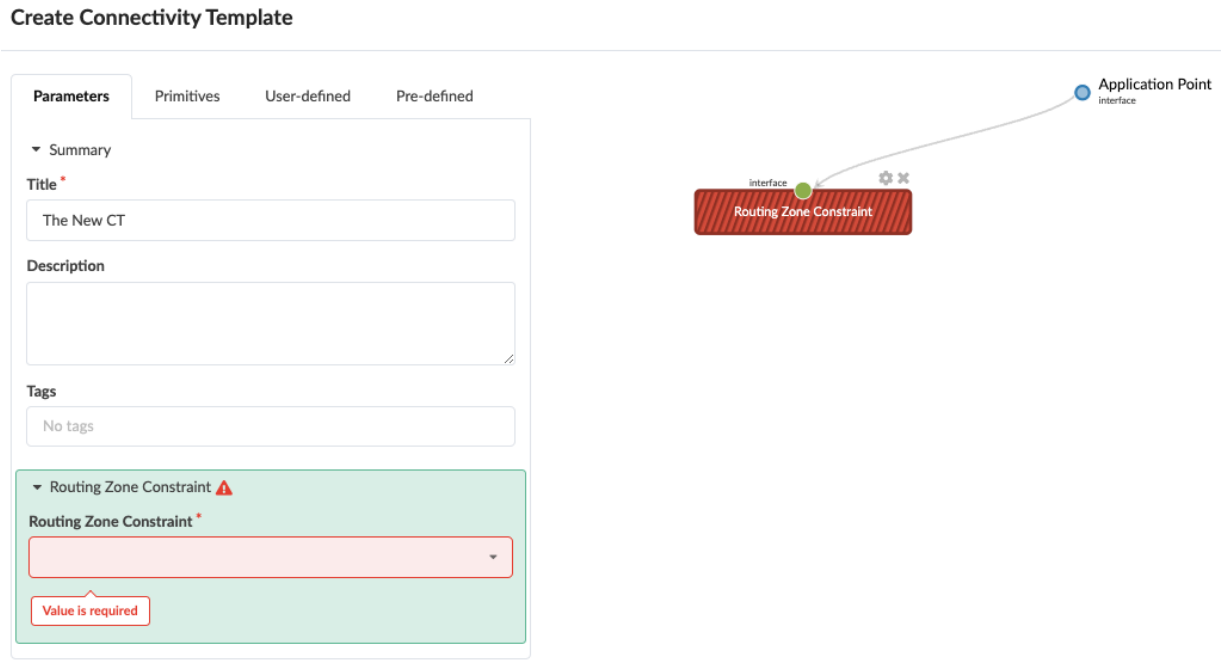
Edit / Delete Routing Zone Constraint Policy

If you need to, you can change or delete the policy after you've created it.

- If you edit the policy to increase the number of routing zones, you don't need to unassign participating ports from the restriction.
- If you edit the policy to reduce the number of routing zones, ensure that all participating ports are in compliance with the new restrictions before you save. Otherwise, you will receive an error.
- You can delete a constraint policy to free up any restrictions on the participating ports. These ports should behave as if the constraint was never applied.

Apply Routing Zone Constraint

When you want to apply the constraint to an application point, add the **Routing Zone Constraint** primitive to the connectivity template and specify the routing zone or routing zone group. For more information about connectivity templates, see "Connectivity Templates" on page 456.

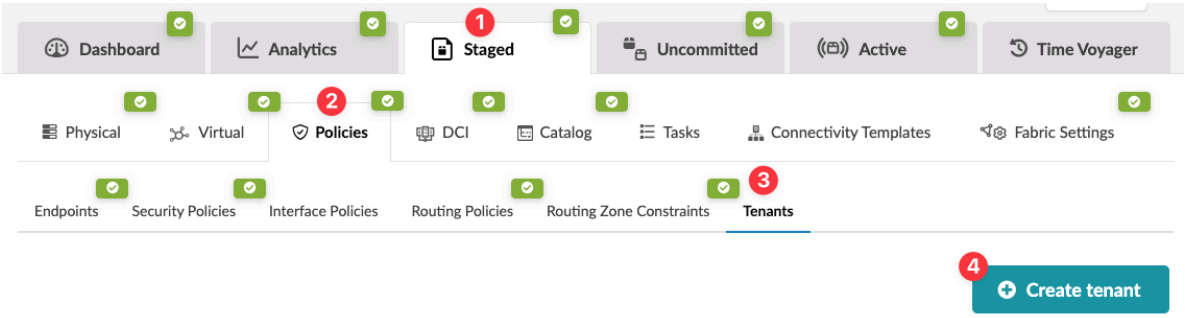


Tenants

IN THIS SECTION

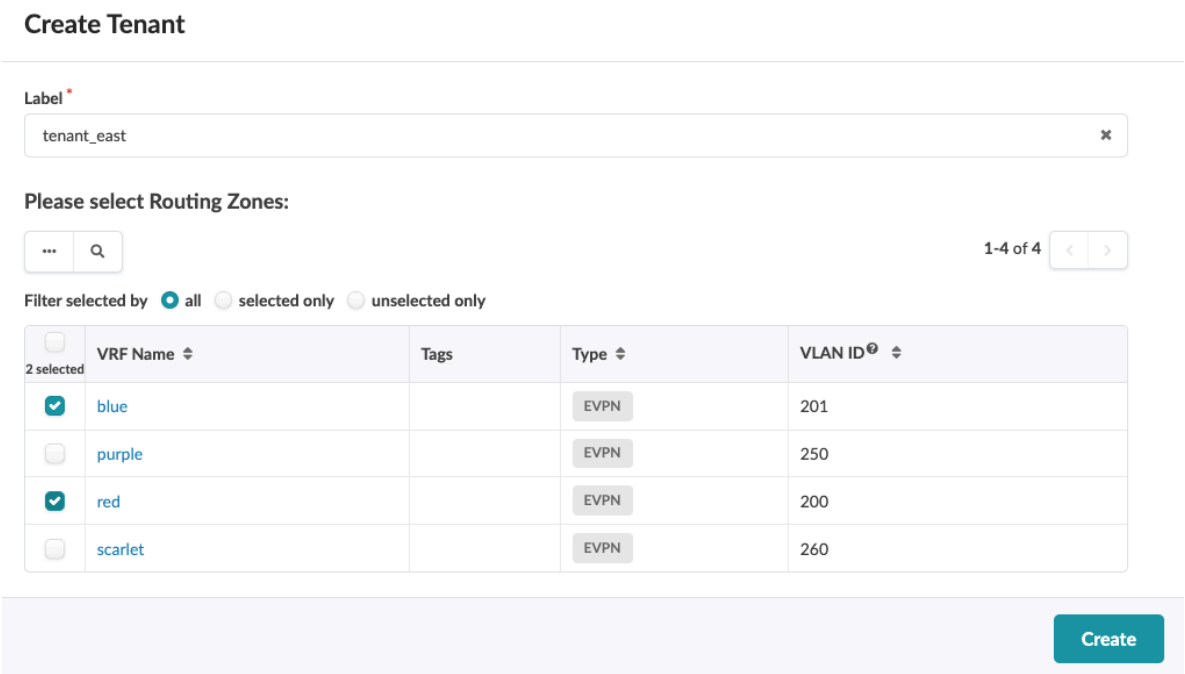
Create Tenant

1. From the blueprint, navigate to **Staged > Policies > Tenants** and click **Create Tenant**.



   The **Create Tenant** dialog opens.

2. Enter a label that doesn't start or end with space characters and that doesn't contain quotes. The regular expression for the allowed names is a-zA-Z0-9_-. Tenant labels serve as primary keys. After you've create a tenant, you can't change its label.



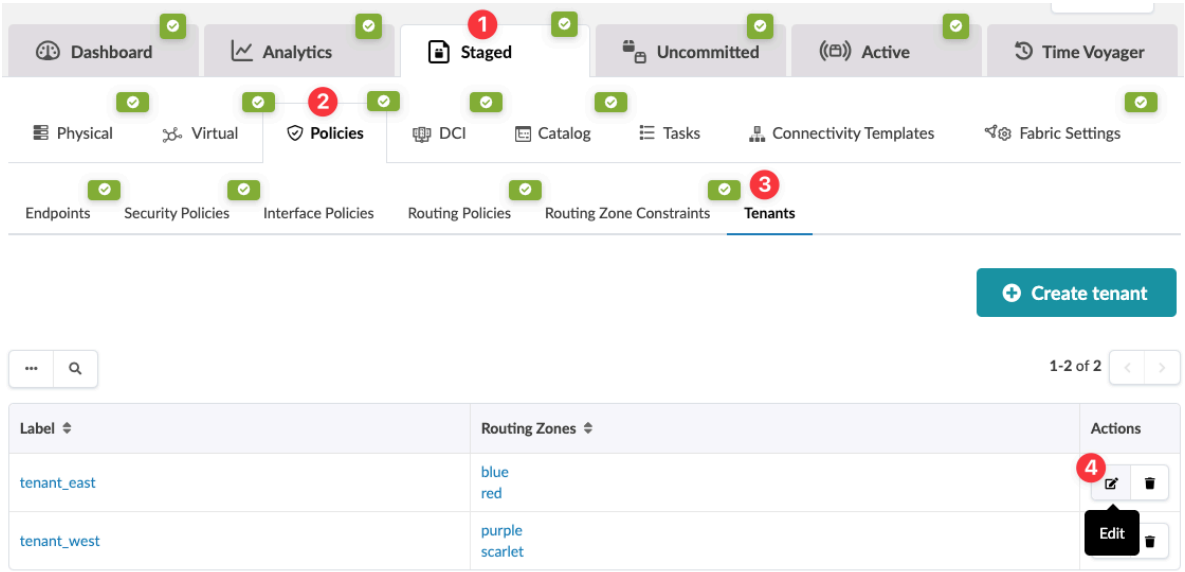
3. Select check boxes for one or more routing zones of type EVPN. (The Default Routing Zone is excluded from this feature given the number of design elements it holds which are structural to the fabric underlay.) You can search for routing zones by VRF name, tags, and/or type of routing zone (EVPN).

4. Click **Create** to stage the tenant and return to the tenant table view.

When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

Update Tenant

1. From the blueprint, navigate to **Staged > Policies > Tenants** and click the **Edit** button for the tenant to update.

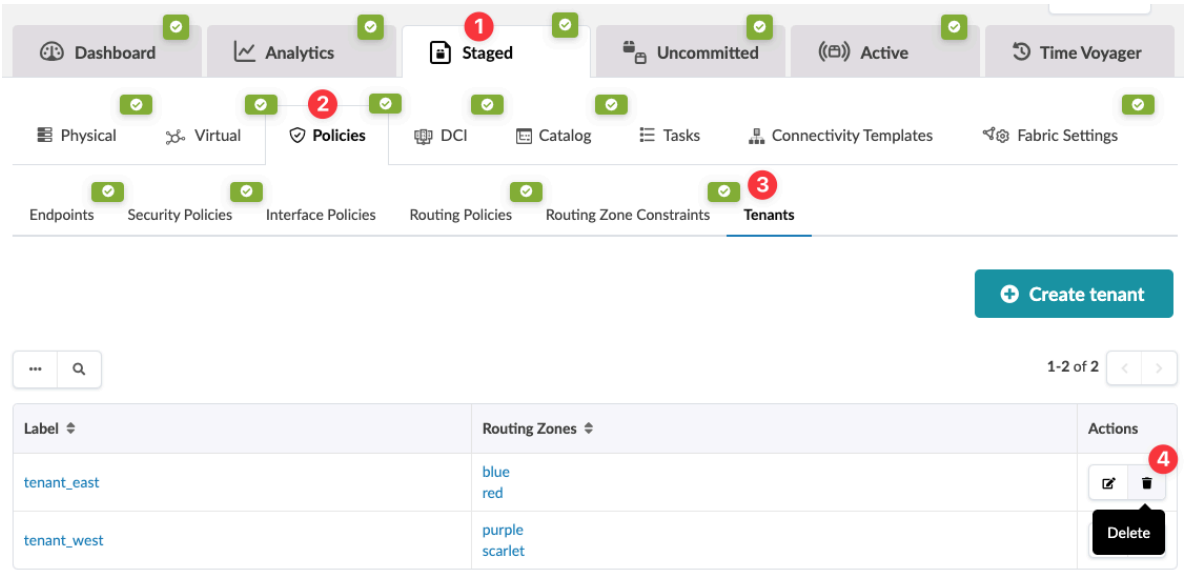


   The **Edit Tenant** dialog opens.
2. Add and/or remove routing zones, as needed, by checking or unchecking the appropriate boxes for the tenant(s).
3. Click **Submit** to stage the change and return to the tenant table view.

When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

Delete Tenant

1. From the blueprint, navigate to **Staged > Policies > Tenants** and click the **Delete** button for the tenant to delete.



The **Delete this resource?** dialog opens.

2. Click **Delete** to stage the deletion and return to the tenant table view.

When you're ready to activate your changes, go to the **Uncommitted** tab to review and commit (or discard) your changes.

CHAPTER 9

Data Center Interconnect (DCI)

IN THIS CHAPTER

Integrated Interconnect

IN THIS SECTION

Integrated DCI (VXLAN Stitching)

IN THIS SECTION

Overview

NOTE: Apstra also supports two other types of DCI:

- External Handoff where an external connection is set with a standard Layer 2 VLAN handoff external connection with traditional Flood MA VLAN learning. This extends a single Network/ Broadcast domain with a traditional demarcation point.
- OTT (over the top) Extending the Single EVPN-VXLAN domain between data centers.

Initial Apstra Integrated DCI is being qualified with the following models using Junos OS 23.2R: QFX5700, QFX5130, QFX1000.

Integrated DCI, also known as VXLAN stitching, allows Apstra users to extend EVPN Type 2 and Type 5 routes between data centers using designated border leaf(s) to act as DCI gateways at each data center.

- Apstra's Integrated DCI reference design follows RFE-9014 and draft-sharma-bess.
- Each data center is treated as its own independent domain.

Configuring Integrated DCI within Apstra:

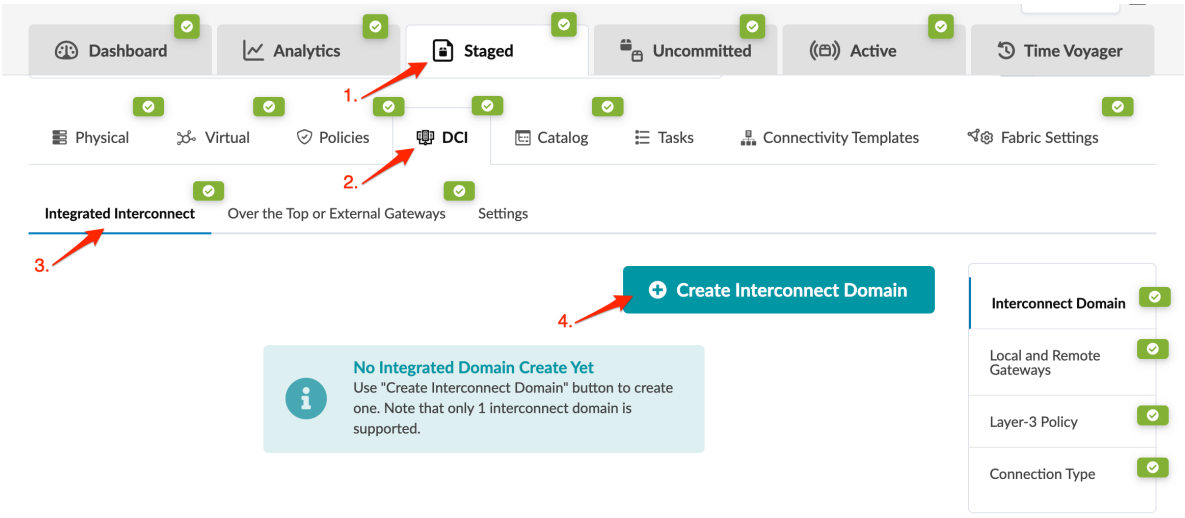
- DCI configuration must be configured as part of each data center deployment/blueprint and use the same Interconnect Route Target (iRT).
- The steps below guide you through the process for each blueprint and deployment.

1. Configure ESI MAC MSB

All data centers must be configured to use a different ESI MAC MSB (most significant byte). Refer to "Update ESI MAC MSB" on page 435 for details.

2. Create Interconnect Domain

1. From the blueprint, navigate to **Staged > DCI > Integrated Interconnect** and click **Create Interconnect Domain**.

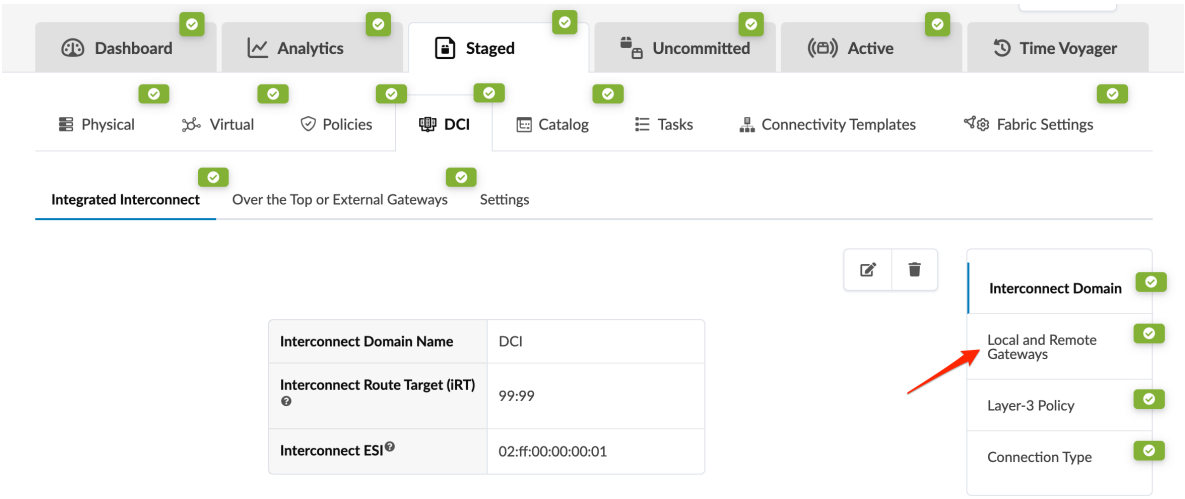


The **Create Interconnect Domain** dialog opens.

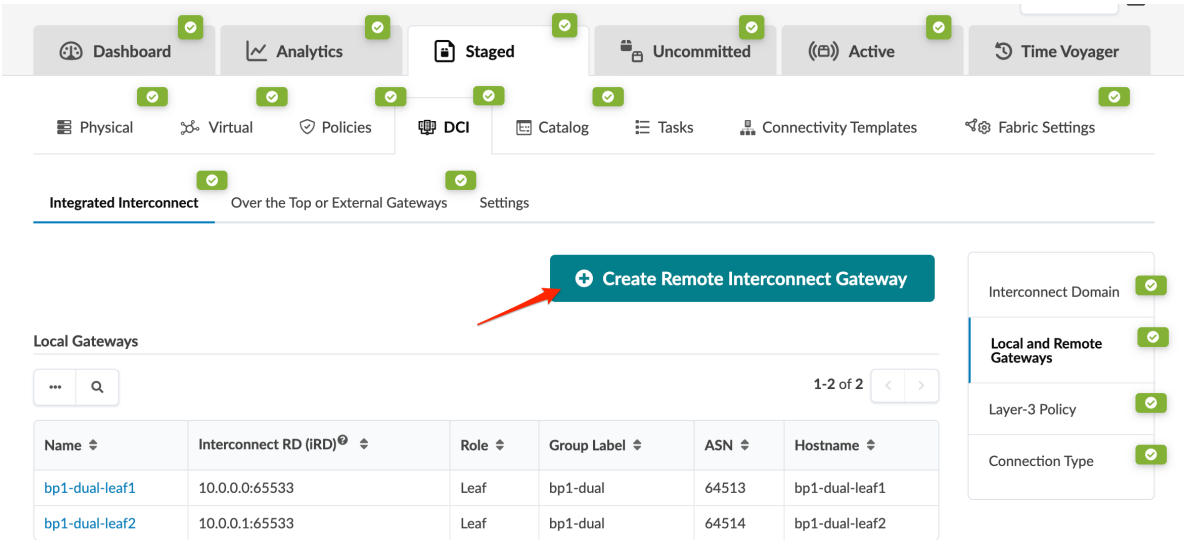
2. Enter the following information:
   - **Interconnect Domain Name**
   - **Interconnect Route Target (iRT)** - All interconnect gateways must use the same iRT. The iRT is an additional unique RT for the interconnect domain. The iRT must be globally unique; it must not be in use for every DCI-connected data center.
   - **Interconnect ESI** (optional) - Each site requires a unique site ID iESI at the MAC-VRF level, either auto-derived or set manually.
3. Click **Update** to create the interconnect domain and return to the **Integrated Interconnect** page.

3. Create Remote Interconnect Gateway

1. From the **Integrated Interconnect** page, click **Local and Remote Gateways**.

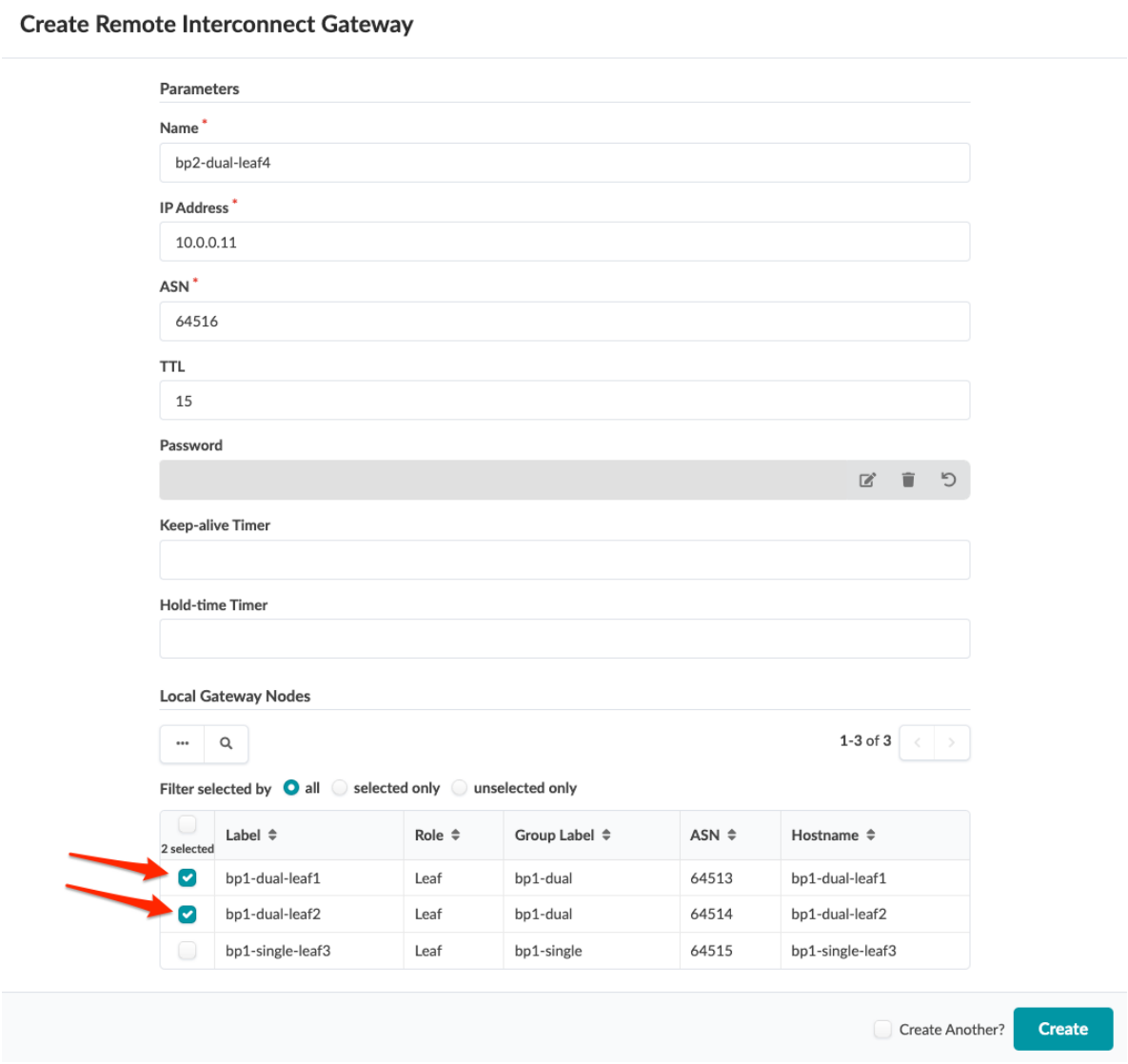


2. Click **Create Remote Interconnect Gateway**.



The **Create Remote Interconnect Gateway** dialog opens.

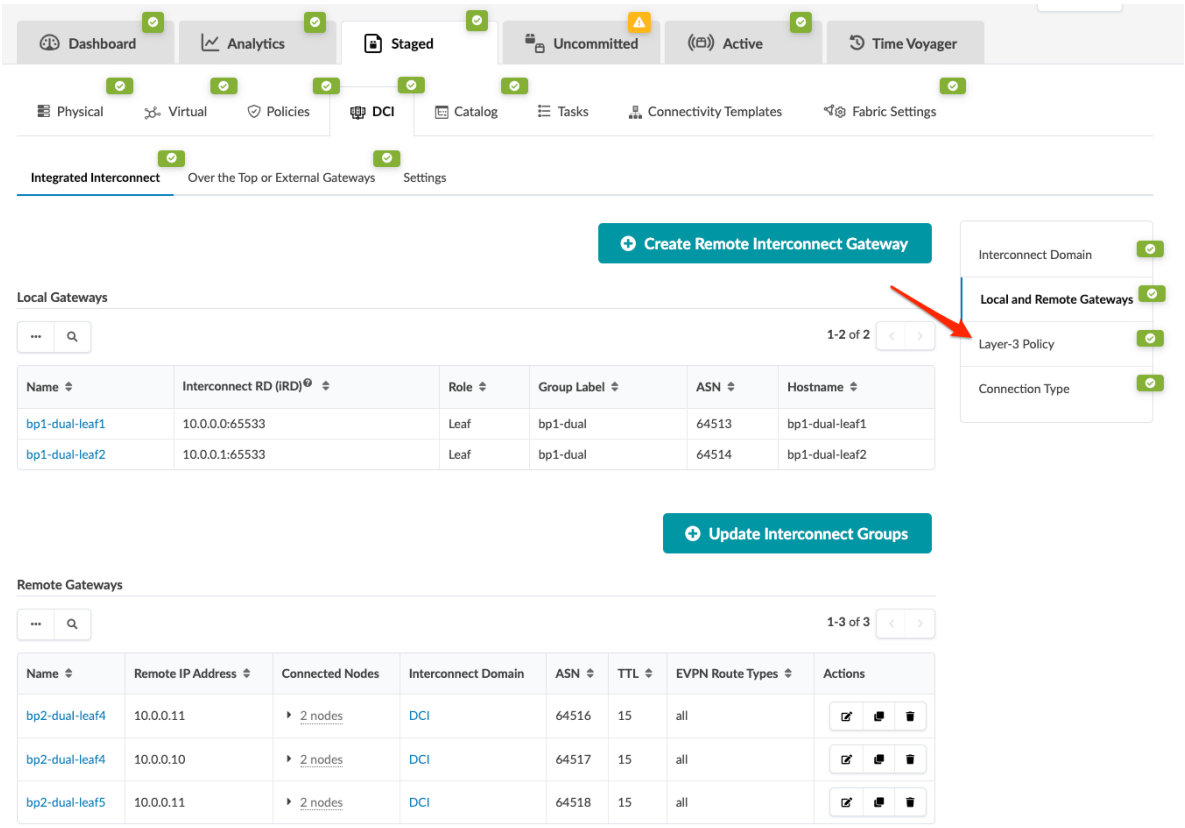
3. Enter a name for the remote border leaf interconnect gateway and add the remote BGP IP/ASN. Select the local border leaf devices that will establish a session with this remote DCI gateway.



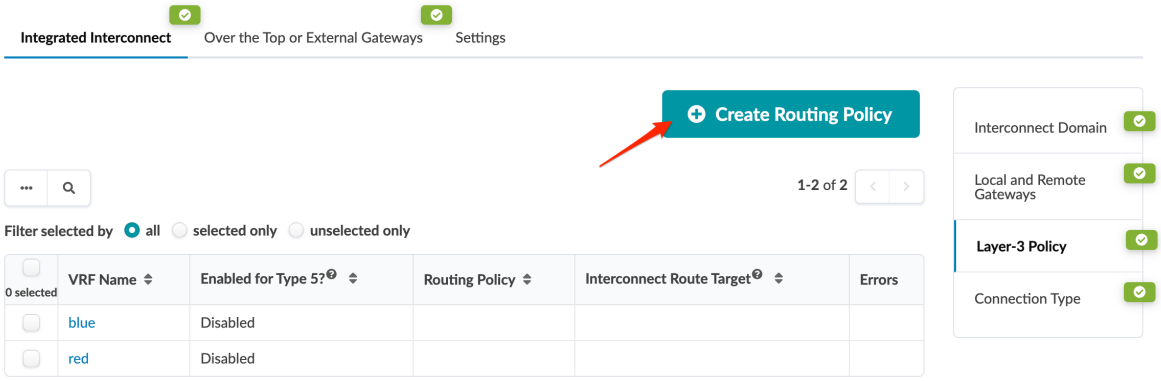
4. Click **Create** to create the gateway and return to the **Integrated Interconnect** page.

4. Create Routing Policy

1. From the **Integrated Interconnect** page, click **Layer-3 Policy**.

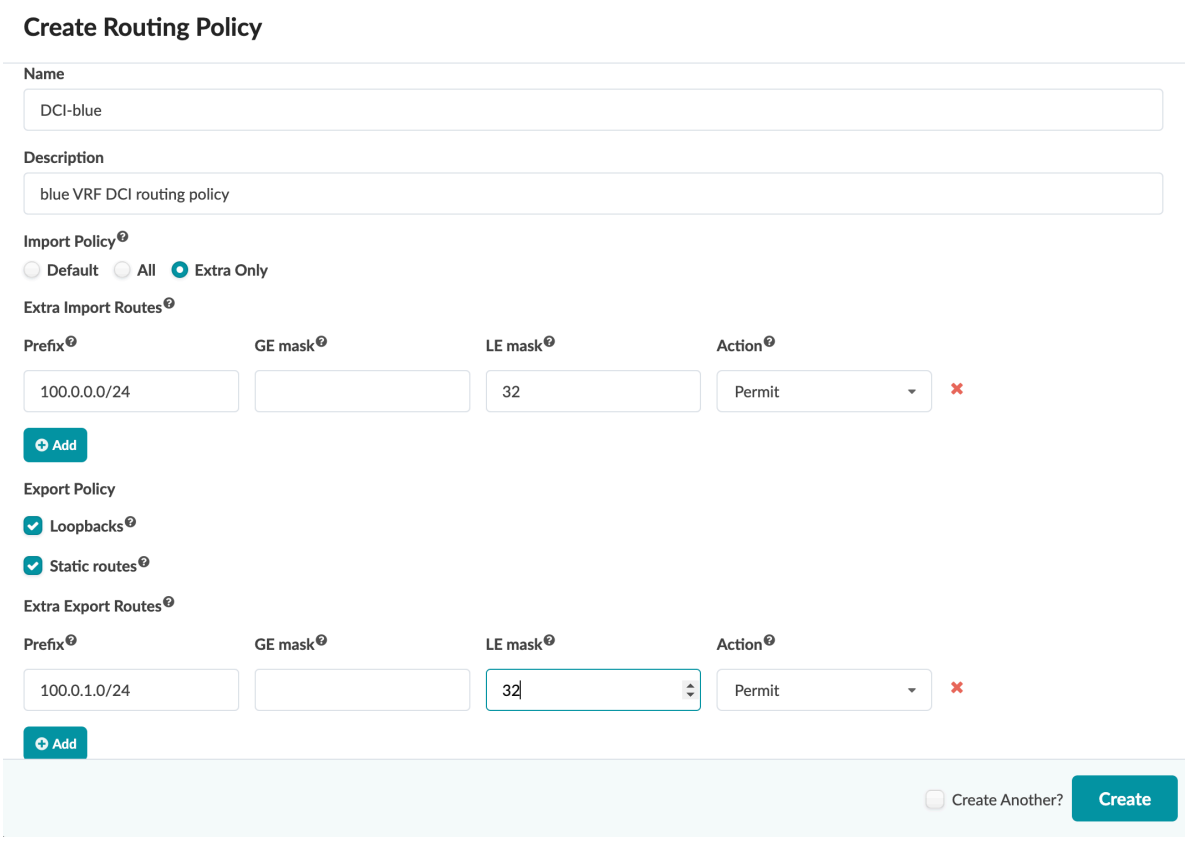


2. Click **Create Routing Policy**.

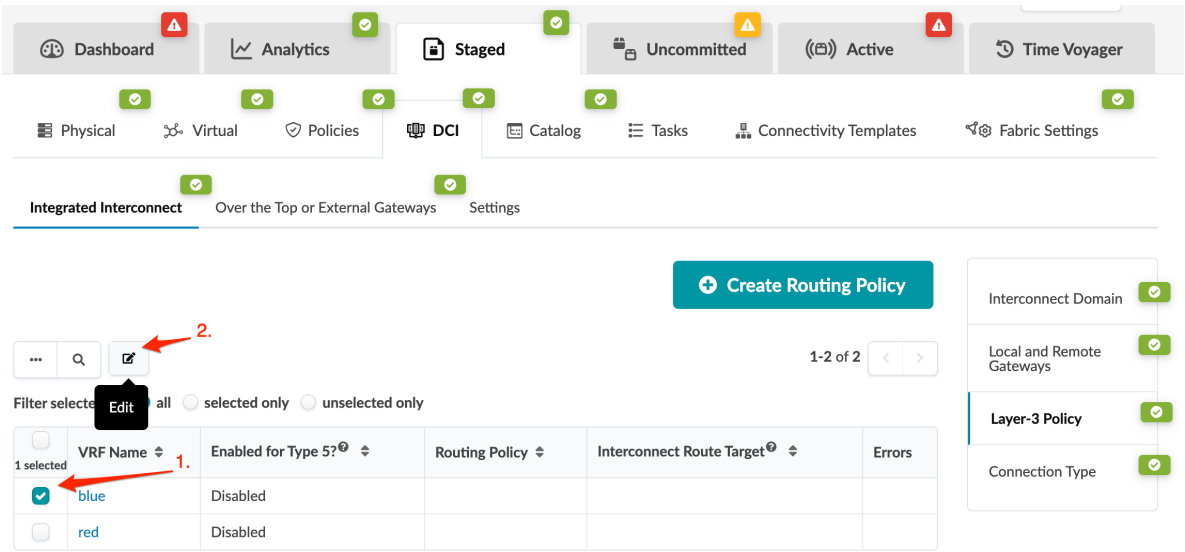


The **Create Routing Policy** dialog opens.

3. You can create a routing policy that can be used with VRF routes to exchange and extend the EVPN Type 5 routes via Integrated DCI. If you only plan to extend VNI and exchange EVPN Type 2 routes then you don't need to enable the VRFs for DCI EVPN Type 5 route exchange. The example below is for a route-map policy for the blue VRF.

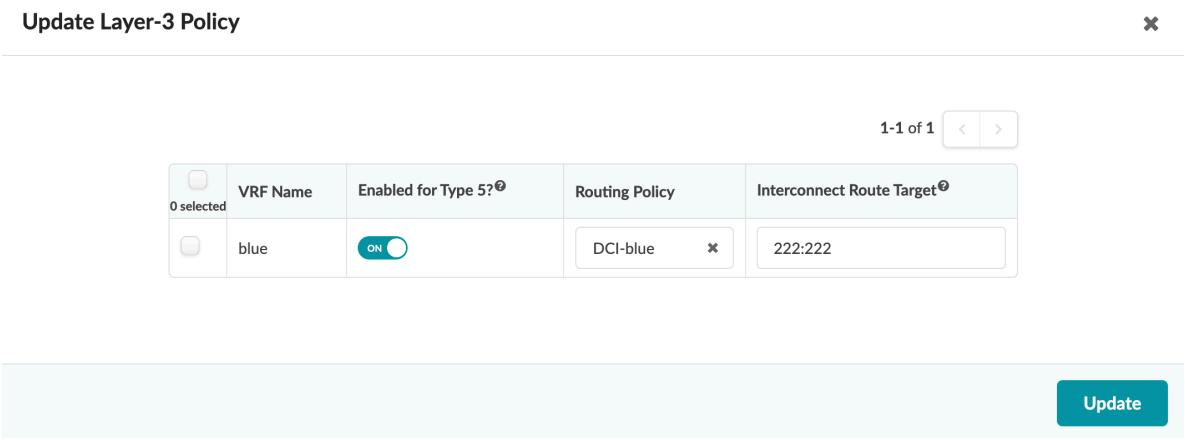


4. After entering routing policy details, click **Create** to create the routing policy and return to the **Integrated Interconnect** page.
5. To assign the routing policy and enable the VRF for DCI Type 5 route exchange (as applicable), select the check box for the VRF, then click the **Edit** button that appears above the table.



The **Update Layer-3 Policy** dialog opens

6. Toggle on Type 5 enablement (if applicable), select the routing policy from the drop-down list, and enter the iRT. Interconnect gateways must use the same Interconnect Route Target (iRT). The iRT is an additional unique route target for the interconnect domain.

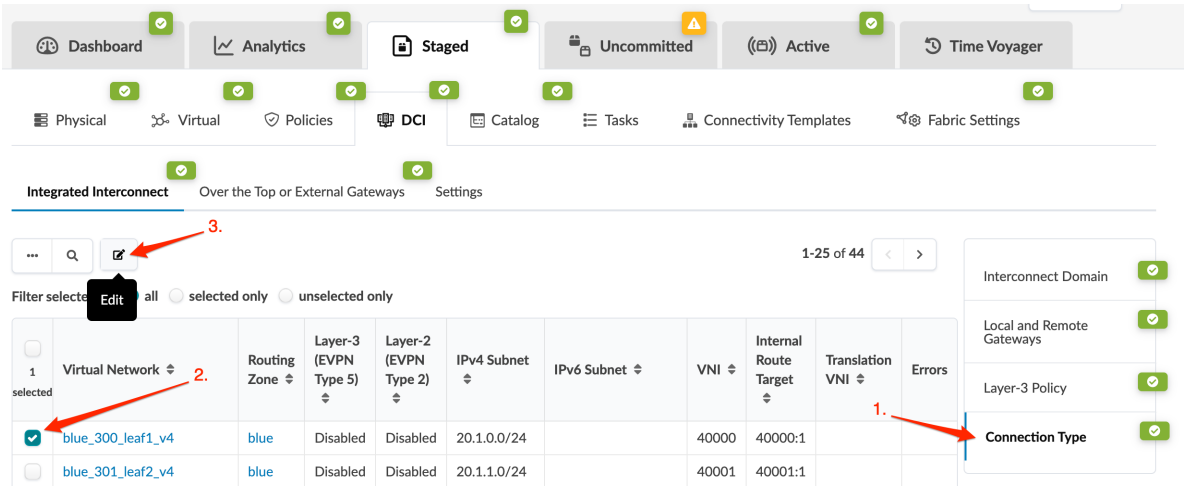


7. Click **Update** to save your changes and return to the **Integrated Interconnect** page.

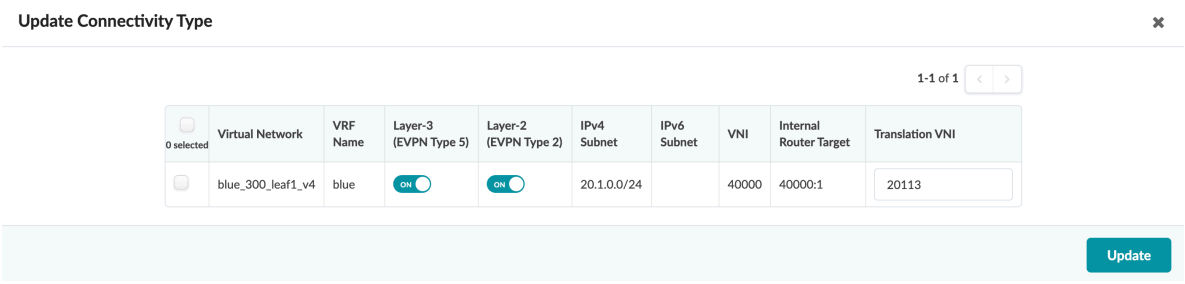
5. Update Connectivity Type

This configuration section allows extending VNI EVPN Type 2 routes across data centers via Integrated DCI. Let's configure virtual networks to be exchanged via Integrated DCI.

1. From the **Integrated Interconnect** page, click **Connection Type** (in right panel).
2. Select the check boxes for the Layer 2 virtual networks and Layer 3 VRFs that need to extend to the remote DCI gateway. Remember, for every VRF that is to be enabled for Type 5 EVPN route exchange, it must be enabled on the **Layer-3 Policy** tab.



3. Click the **Edit** button that appears above the table.
   The **Update Connectivity Type** dialog opens.
4. Enable Layer 2 EVPN Type 2 route exchange. Translation VNI is the intermediate VNI to be used. It isn't required, but it needs to match the remote VNI either by translation on the other side or by both data centers using the same VNI for the virtual network that's being extended.



5. Click **Update** to save your changes and return to the **Integrated Interconnect** page.
6. After enabling all virtual networks and VRFs that will be extended to the remote DCI gateway, ensure that the **Uncommitted** tab is green and that you can commit the changes to enable the local Integrated DCI gateways that have been configured. (You'll repeat all these steps for the remote DCI gateway.)

Repeat the process for all Layer 2 virtual networks and Layer 3 VRFs that need to extend to the remote DCI gateway. Remember, every VRF that is to be enabled for Type 5 EVPN route exchange must be enabled on the **Layer-3 Policy** tab.

6. Configure Remote DCI Gateway

Repeat the above steps to configure the remote DCI gateway.

Over the Top or External Gateways

IN THIS SECTION

Data Center Interconnect Introduction

IN THIS SECTION

ESI MAC Most Significant Byte

By default, ESI MAC msb (most significant byte) is set to 2 on all blueprints. Every Apstra blueprint that's connected must have a unique msb to prevent service-impacting issues. Before creating remote EVPN gateways, change ESI MAC msb accordingly. (You can leave one of them at the default value.)

Apstra is programmed to assign a unique ESI MAC address starting with the value 00.00.00.00.00.01.

This feature allows you to manually configure the most significant byte (MSB) of the MAC address.

Updating this value results in the regeneration of all ESI MACs in the blueprint. This is necessary to address the data center interconnect (DCI) use case requirement where ESI values must be unique across multiple fabrics (blueprints). For example, if you have data centers DC1, DC2, and DC3 all managed by Apstra and connected via Apstra DCI, by default, each of them will have the same internally generated ESI MAC. You would use this feature to provide a unique value to DC2 and DC3.

RELATED DOCUMENTATION

Update ESI MAC msb | 435

Data Center Interconnect (DCI) / Remote EVPN Gateways

IN THIS SECTION

DCI / EVPN Gateway Overvew

Historically, enterprises have leveraged Data Center Interconnect (DCI) technology as a building block for business continuity, disaster recovery (DR), or Continuity of Operations (COOP). These service availability use cases primarily relied on the need to connect geographically separated data centers with Layer 2 connectivity for application availability and performance.

With the rise of highly virtualized Software-Defined Data Centers (SDDC), cloud computing, and more recently, edge computing, additional use cases have emerged:

- **Colocation Expansion**: Share compute and storage resources to colocation data center facilities.

- **Resource Pooling**: Share and shift applications between data centers to increase efficiency or improved end-user experience.
- **Rapid Scalability**: Expand capacity from a resource-limited location to another facility or data center.
- **Legacy Migration**: Move applications and data off older and inefficient equipment and architecture to more efficient, higher-performing, and cost-effective architecture.

With Apstra software, you can deploy and manage a vendor inclusive DCI solution that is simple, flexible, and Intent-Based. Apstra utilizes the standards-based MP-BGP EVPN with VXLAN, which has achieved broad software and hardware adoption in the networking industry. You can choose from a vast selection of cost-effective commodity hardware from traditional vendors to white-box ODMs and software options ranging from conventional vendor integrated Network Operating Systems (NOS) to disaggregated open source options.

EVPN VXLAN is a standards-based (RFC-7432) approach for building modern data centers. It incorporates both data plane encapsulation (VXLAN) and a routing control plane (MP-BGP EVPN Address Family) for extending Layer 2 broadcast domains between hosts as well as Layer 3 routed domains in spine-leaf networks. Relying on a pure Layer 3 underlay for routing of VXLAN tunneled traffic between VXLAN Tunnel Endpoints (VTEPs), EVPN introduces a new address family to the MP-BGP protocol family and supports the exchange of MAC/IP addresses between VTEPs. The advertisement of endpoint MACs and IPs, as well as "ARP/ND-suppression", eliminates the need for a great majority of Broadcast/Unknown/Multicast (BUM) traffic and relies upon ECMP unicast routing of VXLAN, from Source VTEP to Destination VTEP. This ensures optimal route selection and efficient load-sharing of forwarding paths for overlay network traffic.

Just as EVPN VXLAN works within a single site for extending Layer 2 between hosts, the DCI feature enables Layer 2 connectivity between sites. The Apstra DCI feature enables the extension of Layer 2 or Layer 3 services between data centers for disaster recovery, load balancing of active-active sites, or even for facilitating the migration of services from one data center to another.

Limitations:

- EVPN-GW (DCI) between different vendors' EVPN fabric is not supported.
- IPv6 is not supported on Remote EVPN Gateways. (Actual EVPN routes can contain IPv6 Type 2 and Type 5.)

DCI Deployment Options

IN THIS SECTION

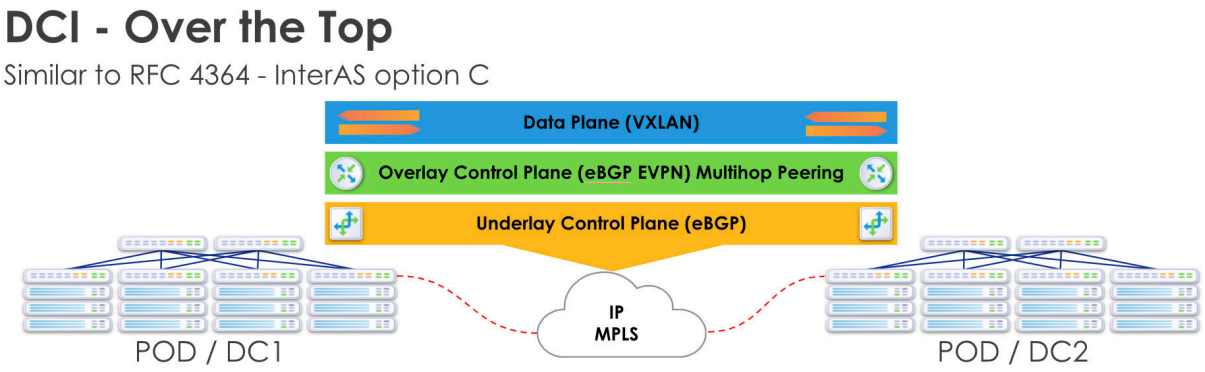
The following characteristics apply to all deployment options:

- You can extend Apstra DCI to other Apstra-managed data centers, non-Apstra managed data centers, or even to legacy non-spine-leaf devices.
- Apstra implementation and behavior is the same in all three cases.
- Whether the remote end is another DCI GW or an ASBR, it is transparent to Apstra.
- Apstra manages neither the GWs nor ASBRs.

You can implement Data Center Interconnect using the following methods. For assistance with selecting the best option for your organization, consult your Apstra Solutions Architect (SA) or Systems Engineer (SE).

Over the Top

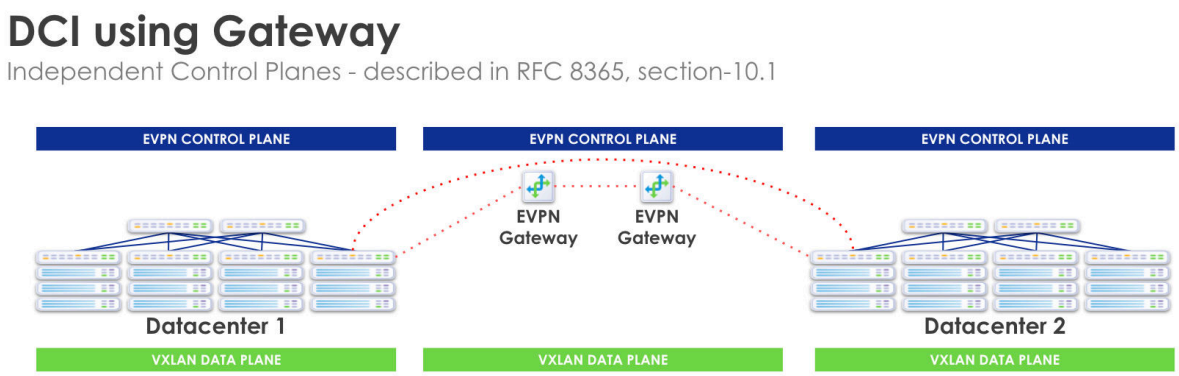
DCI "Over the Top" is a transparent solution, meaning EVPN routes are encapsulated into standard IP and hidden from the underlying transport. This makes the extension of services simple and flexible and is often chosen because data center teams can implement it with little to no coordination with WAN or Service Provider groups. This reduces the implementation times and internal company friction. However, the tradeoff is scalability and resilience.



Gateway (GW)

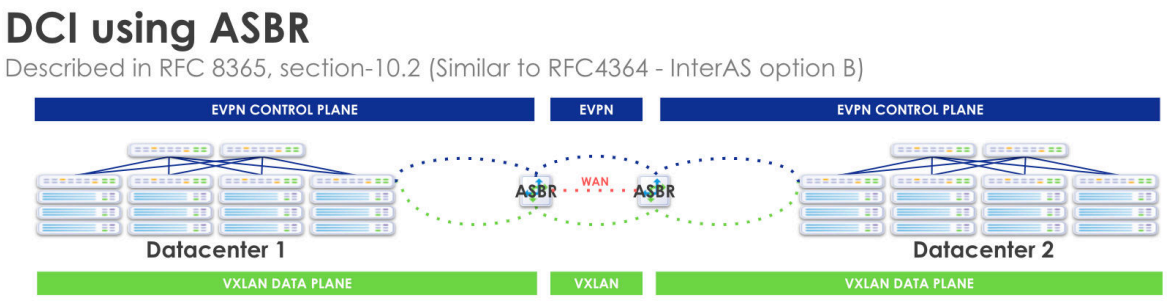
Building upon the Apstra **Remote EVPN Gateway** capability, you can optionally specify that the **Remote EVPN Gateway** is an external generic system (tagged as an external router) in the same site, thus extending the EVPN attributes to said gateway. This solution creates a fault domain per site, preventing

failures from affecting convergence in remote sites and creating multiple fault domains. IP/MAC endpoint tables for remote sites are processed and held in state on a generic system (tagged as external router) gateway. You can also implement WAN QoS and security, along with optimizations that the transport technology makes available (MPLS TE for example). However, this solution is more operationally complex, requiring additional hardware and cost.



Autonomous System Border Router (ASBR)

Using the Apstra **Remote EVPN Gateway** capability, you can optionally specify that the **Remote EVPN Gateway** is an ASBR WAN Edge Device. This end-to-end EVPN enables uniform encapsulation and removes the dedicated GW requirement. It is operationally complex but has greater scalability as compared to both "DCI Using Gateway" and "Over the Top".



Implementation

IN THIS SECTION

You can extend routing zones and virtual networks (VN) to span across Apstra-managed blueprints (across pods) or to remote networks (across data centers) that Apstra doesn't manage. This feature introduces the EVPN Gateway (GW) role, which could be a switch that participates in the fabric or RouteServer(s) on a generic system (tagged as a server) that is connected to the fabric.

EVPN Gateways Use Cases

- Span Layer 3 isolation domains (VRFs / routing zones) to multiple Apstra-managed pods (blueprints) or extend to remote EVPN domains.
- Provide Layer 2 domain extensions for L2VNI / virtual networks.
- Help extend EVPN domain from Apstra to Apstra-managed and Apstra to unmanaged pods.
- No VXLAN traffic termination on the spine devices - connect external generic systems (tagged as external routers) on spine devices. This is to support IPv4 (underlay) external connectivity. Here spine devices don't need to terminate VXLAN traffic, unlike border leaf devices, when connected to external generic systems (tagged as external routers). In a nutshell, using this can exchange IPv4 routes to remote VTEPs (in the default routing zone/VRF) and only Layer 3 connectivity is required:

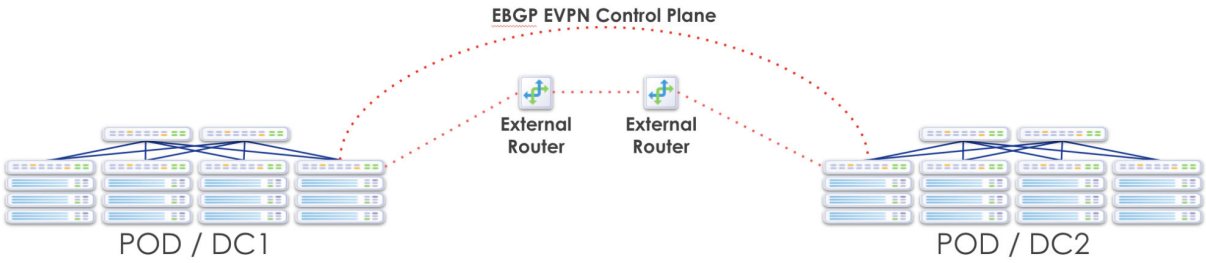
Over the Top

When BGP EVPN peering is done "over the top", the Data Center Gateway (DC-GW) is a pure IP transport function and BGP EVPN peering is established between gateways in different data centers.

The next sections describes the procedures for interconnecting two or more BGP-based Ethernet VPN (EVPN) sites in a scalable fashion over an IP network. The motivation is to support extension of EVPN sites without having to rely on typical Data Center Interconnect (DCI) technologies like MPLS/VPLS, which are often difficult to configure, sometimes proprietary, and likely legacy in nature.

"Over the Top" is a simple solution that only requires IP routing between data centers and an adjusted MTU to support VXLAN encapsulation between gateway endpoints. In such an implementation, EVPN routes are extended end-to-end via MP-BGP between sites. Multi-hop BGP is enabled with the assumption that there will be multiple Layer 3 hops between sites over a WAN. Otherwise the default TTL decrements to 0 and packets are discarded and don't make it to the remote router. Apstra

automatically renders the needed configuration to address these limitations.



This design merges the separate EVPN-VXLAN domains and VXLAN tunnels between sites. Merging of previously separate EVPN domains in different sites realizes the benefit of extending Layer 2 and Layer 3 (VRF) services between sites, but also renders the sites as a single fault domain. So a failure in one site is necessarily propagated. Also, anytime you stretch Layer 2 across the WAN between sites, you are also extending the flood domain and along with it, all broadcast traffic over your costly WAN links. At this time, this solution does not offer any filtering or QoS.

NOTE: When separate Apstra blueprints manage individual sites (or when only one site is Apstra-managed) you must create and manage extended routing zones (VRFs) and virtual networks (Layer 2 and/or Layer 3 defined VLANs/subnets) independently in each site. You must manually map VRFs and VNs between sites (creating administrative overhead).

NOTE: If you're setting up P2P connections between two data centers (blueprints) in the same Apstra controller, each blueprint must pull resources from different IP pools to avoid build errors. To do this, create two IP pools with the same IP subnet, but with different names.

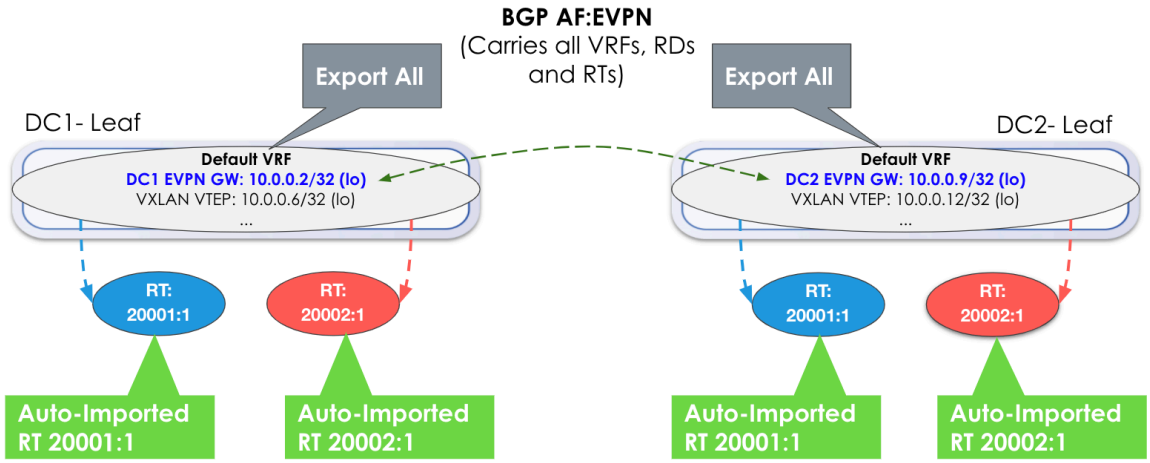
This "Over the Top" solution is the easiest to deploy, requires no additional hardware and introduces no additional WAN config other than increasing the MTU. It is the most flexible and has the lowest barrier to entry. However, the downside is that there is a single EVPN control plane and a routing anomaly in one site will affect convergence and reachability in the other site(s). The extension of Layer 2 flood domains also implies that a broadcast storm in one site extends to the other site(s).

With any DCI implementation, careful resource planning and coordination is required. Adding more sites requires an exponential increase in such planning and coordination. VTEP loopbacks in the underlay need to be leaked. VNIDs must match between sites and in some cases, additional Route Targets (RTs) must be imported. This is covered in detail later in this document.

Data Plane Extension: Layer 3

VXLAN Network IDs (VNIDs) are a part of the VXLAN header that identify unique VXLAN tunnels, each of which are isolated from the other VXLAN tunnels in an IP network. Layer 3 packets can be encapsulated into a VXLAN packet or Layer 2 MAC frames can be encapsulated directly into a VXLAN packet. In both cases, a unique VNID is associated with either the Layer 3 subnet, or the Layer 2 domain. When extending either Layer 3 or Layer 2 services between sites, you are essentially stitching VXLAN tunnels between sites. VNIDs therefore need to match between sites.

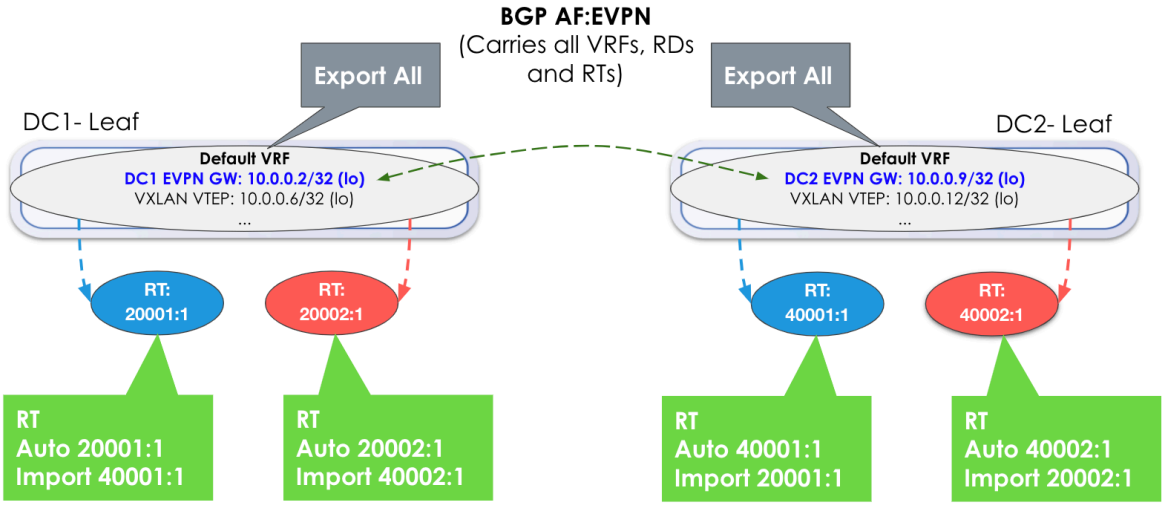
It is important to understand that a particular VNID will be associated with only one VRF (or routing zone in Apstra terminology). VNIDs exist within a VRF. They are tied to a VRF. For Layer 3 services, the stitching, or extending, of each VNID is done with the export and import of RTs within a routing zone (VRF). Layer 3 subnets (routes) are identified via RTs. All VNIDs are exported automatically at the EVPN gateway (edge) towards the WAN. Conversely, RTs of the same value are automatically imported at the EVPN gateway (edge) coming into the fabric. So if you coordinate the Layer 3 VNIDs at one site to match the other, no additional configuration is needed.



In the image above, no additional export or import is required. Everything is automatically exported (Export All) and because the RTs match, they are automatically imported.

However, if a VNID in DC1 is different from a VNID in DC2, then you must import the RTs respectively. Each respective gateway still automatically imports RTs of the same value. In the example below, an

additional step of manually adding the RTs from the other site is required.



Data Plane Extension: Layer 2

A virtual network can be a pure Layer 2 service (Layer 3 anycast gateway is not instantiated). It can be rack-local (VLAN on server-facing ports contained within a rack) or VXLAN (select the racks to extend the Layer 2 flood and broadcast domain between racks. This Layer 2 domain has its own VNID, and the MAC frames (as opposed to IP packets) are encapsulated into the VXLAN header with the VNID of the Layer 2 domain.

The same principles apply in that all VNIDs are exported at the EVPN gateway (in this case Type-2 routes/MAC addresses), and matching RTs are automatically imported. However, the location of importing and exporting RTs is not at the routing zone level, but instead at the virtual network itself.

Apstra Workflow

IN THIS SECTION

Control Plane Extension: EVPN Gateway

Apstra uses the concept of an an "EVPN Gateway". This device can theoretically be a leaf, spine or superspine fabric node, as well as the DCI device. EVPN Gateways separate the fabric-side from the network that interconnects the sites and masks the site-internal VTEPs.

In Apstra, an EVPN Gateway is a device that belongs to and resides at the edge of an EVPN fabric which is also attached to an external IP network. In an Apstra EVPN blueprint, this is always a border-leaf device. The EVPN Gateway of one data center, establishes BGP EVPN peering with a reciprocal EVPN gateway, or gateways, in another data center. The "other" EVPN gateway is the "Remote EVPN Gateway" in Apstra terminology. The Local EVPN Gateway is assumed to be one of the Apstra-managed devices in the blueprint, and is selected when creating the "Remote EVPN Gateway". The Local EVPN Gateway will be the border-leaf switch with one or more external routing connections for traffic in and out of the EVPN Clos fabric.

Due to this capability, you can configure a Local EVPN Gateway (always an Apstra-managed switch) to peer with a non Apstra-managed, or even a non Spine-Leaf device, in another DC. The EVPN Gateway BGP peering is used to carry all EVPN attributes from inside a pod, to outside the pod. In the Apstra environment, each blueprint represents a data center. If two or more sites are under Apstra management, you still must configure each site to point to the "Remote EVPN Gateway(s)" in other sites. We recommend that you create multiple, redundant EVPN Gateways for each data center. There is also currently a full mesh requirement between EVPN gateways, although in future releases this requirement will be removed.

Underlay VTEP Route Advertisements

The underlay reachability to VTEP IP addresses, or an equivalent summary route, must be established reciprocally. Each site must advertise these VTEP loopbacks from within the default routing zone into the exported BGP (IPv4) underlay advertisements. Loopbacks in the routing policy are enabled by default.

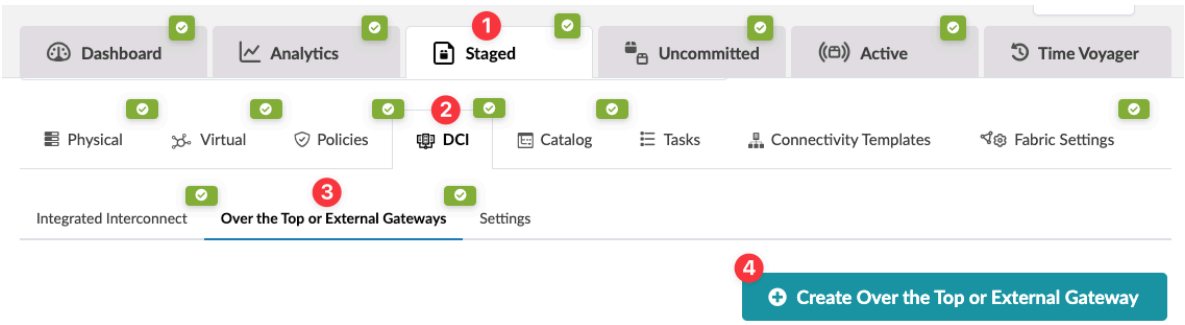
Create Remote EVPN Gateways

CAUTION: By default, ESI MAC msb (most significant byte) is set to 2 on all blueprints. Every Apstra blueprint that's connected must have a unique msb to prevent service-impacting issues. Before creating gateways, "change ESI MAC msb" on page 435 accordingly. (You can leave one of them at the default value.)

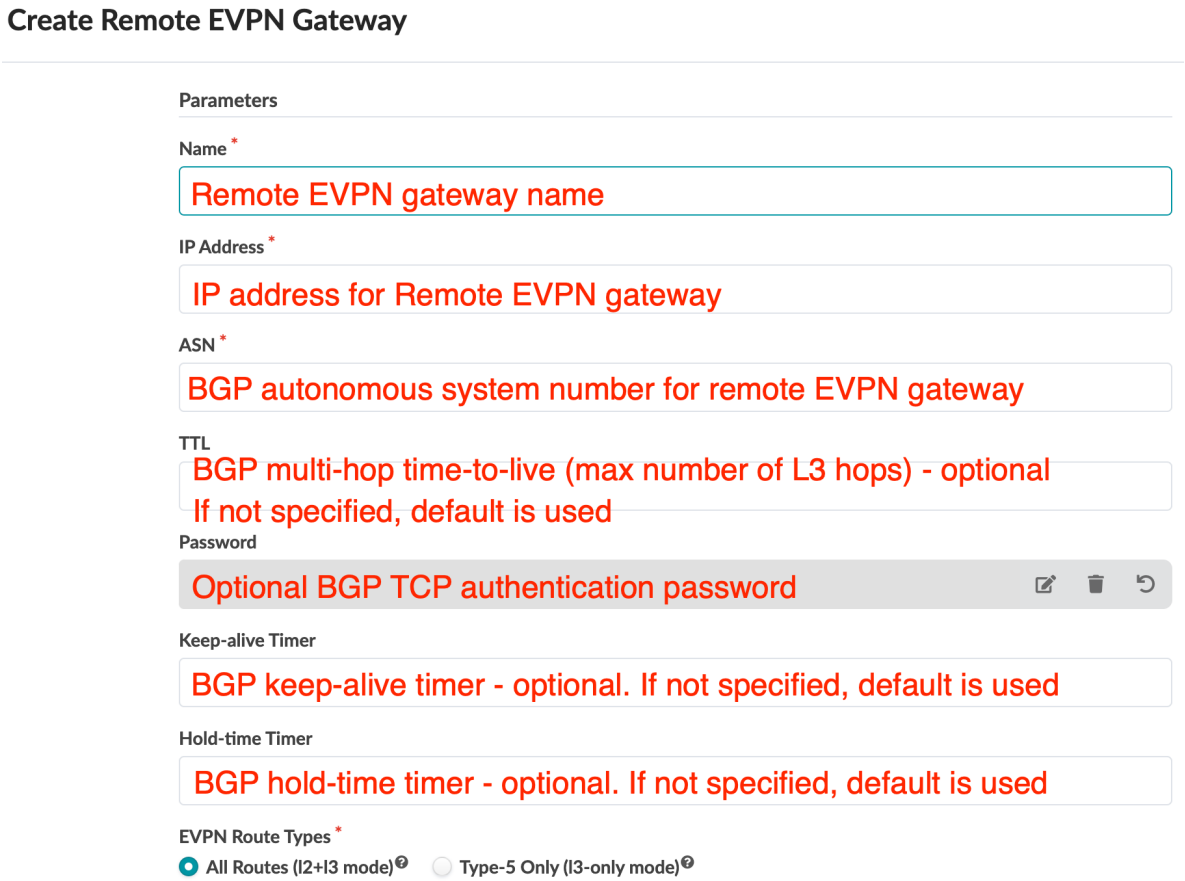
Remote EVPN Gateway is a logical function that you could instantiate anywhere and on any device. It requires BGP support in general, L2VPN/EVPN AFI/SAFI specifically. To establish a BGP session with an EVPN gateway, IP connectivity, as well as connectivity to TCP port 179 (IANA allocates BGP TCP ports), should be available.

NOTE: For resilience, we recommend having at least two remote gateways for the same remote EVPN domain.

1. From the blueprint, navigate to **Staged > DCI > Over the Top or External Gateways** and click **Create Over the Top or External Gateway**.

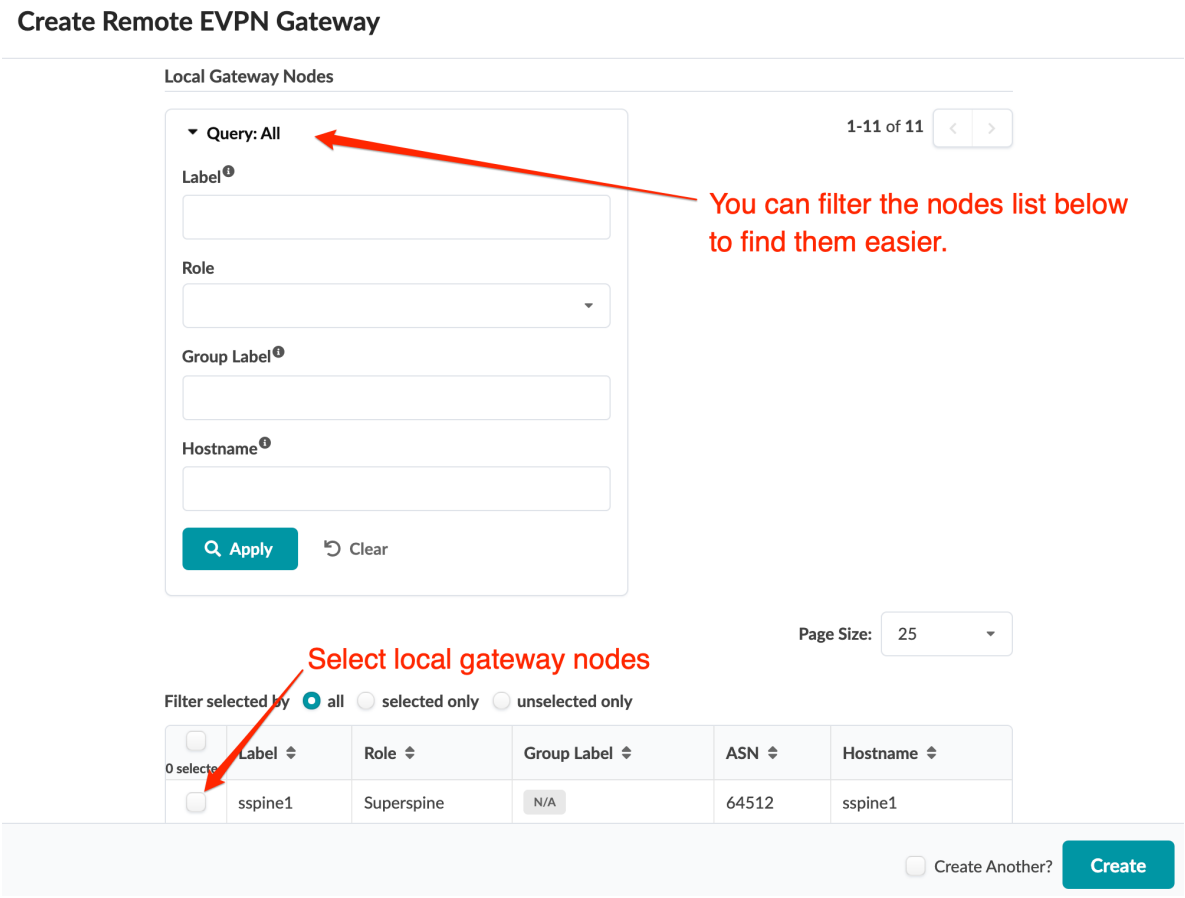


2. In the dialog that opens, fill in the following information for the remote EVPN gateway.



When extending L2 networks between data center fabrics you have the option to exchange only EVPN Route Type RT-5 prefixes (interface-less model). This is useful when there is no need to exchange all host routes between data center locations. This results in smaller requirements for the routing information base (RIB), also known as the routing table, and the forwarding information base (FIB), also known as the forwarding table, on DCI equipment.

3. Select the **Local Gateway Nodes**. These are the devices in the blueprint that will be configured with a Local EVPN Gateway. You can select one or more devices to peer with the configured remote EVPN gateway. You can use the query function to help you locate the appropriate nodes. We recommend using multiple border-leaf devices which have direct connections to external generic systems (tagged as external routers).



4. Click **Create** to stage the gateway and return to the table view.

5. When you are ready to deploy the devices in the blueprint, commit your changes.

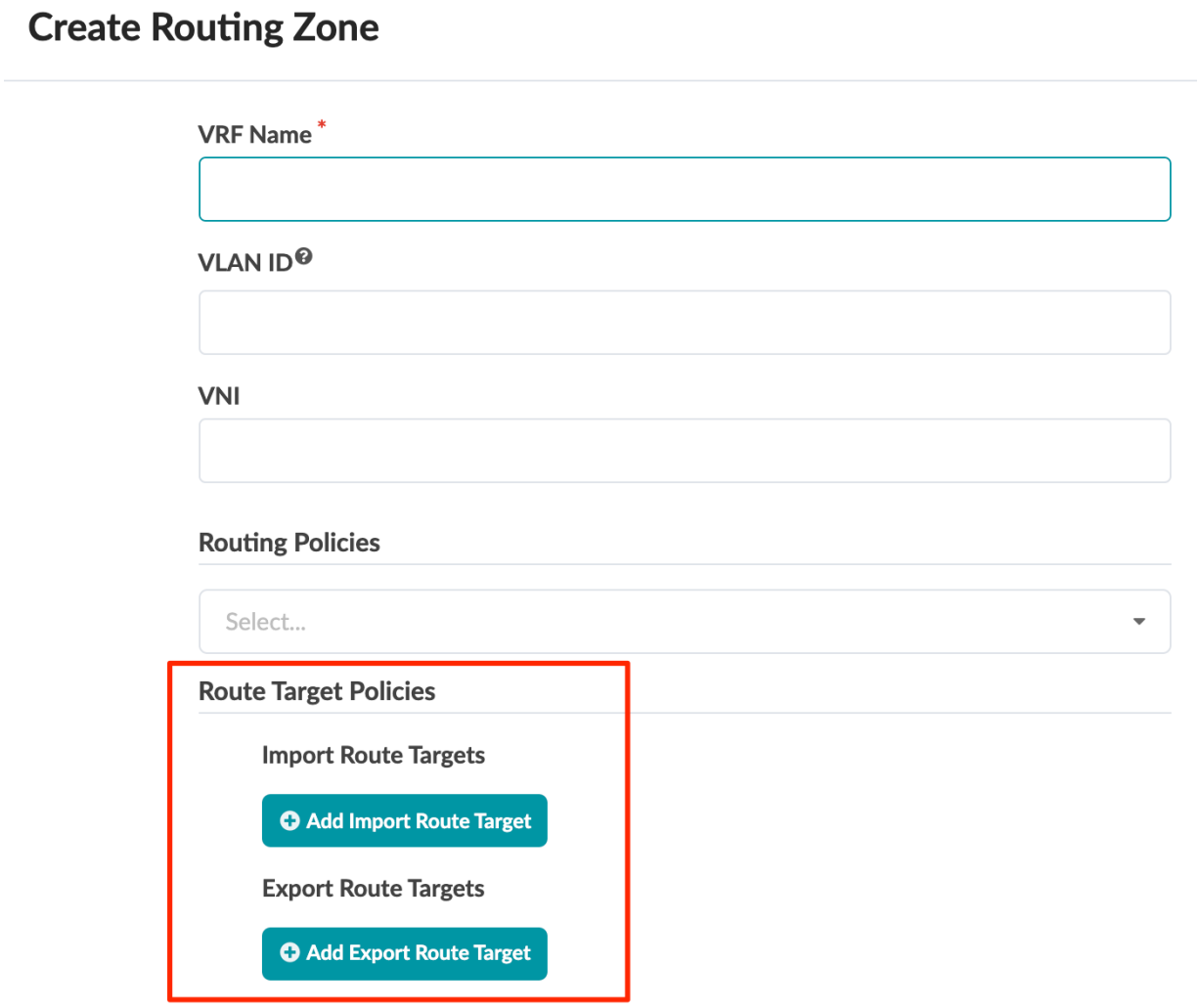
We recommend using multiple remote EVPN gateways. To configure additional remote EVPN gateways, repeat the steps above.

If you are configuring the Remote EVPN Gateway(s) to another Apstra blueprint, you must configure and deploy the remote EVPN gateway(s) separately in that blueprint.

Once the change is deployed, Apstra monitors the BGP session for the remote EVPN gateways. To see any anomalies from the blueprint, navigate to **Active > Anomalies**.

Enhanced Routing Zone

RT (route-target) import/export policies on devices that are part of extended service govern EVPN route installation. Specify route target policies to add import and export route-targets that Apstra uses for routing zones/VRFs. You do this when you create routing zones. Navigate to **Staged > Virtual > Routing Zones** and click **Create Routing Zone**. For more information, see "Routing Zones" on page 282.



NOTE: The generated default route-target for routing zones is **<L3 VNI>:1**. You can't change this default value.

To confirm that correct routes are received at VTEP make sure L3VNIs and route target are identical between the blueprint and remote EVPN domains.

Enhanced Virtual Networks

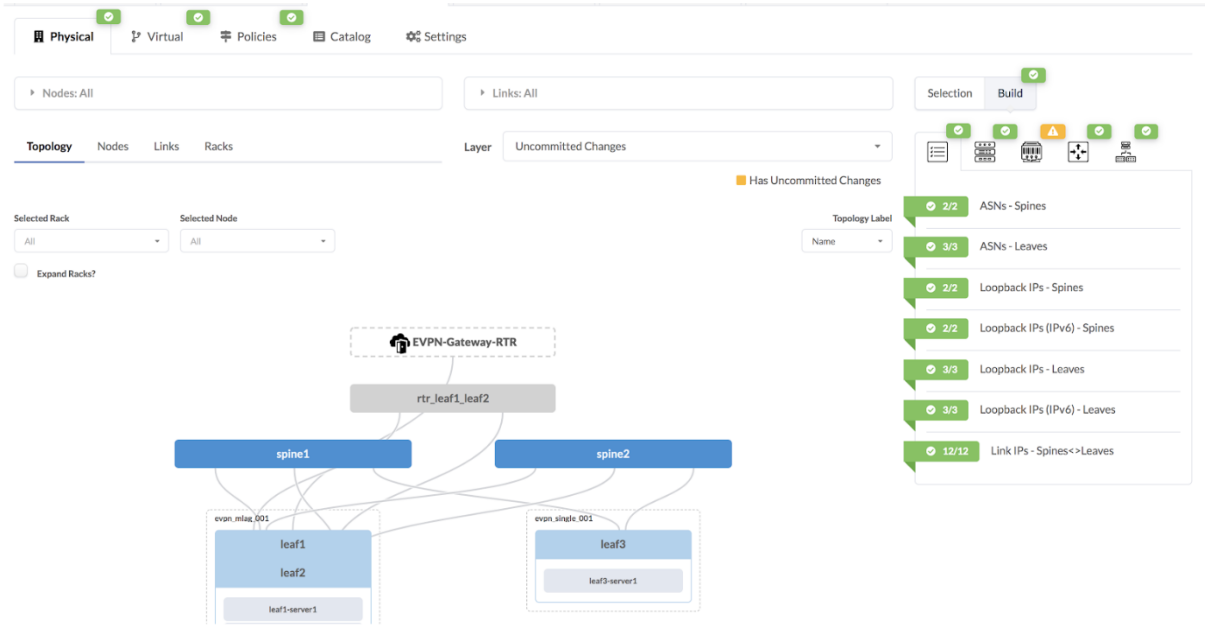
You can add additional import and export route-targets that Apstra uses for virtual networks.

> **NOTE**: The default route target that Apstra generates for virtual networks is **<L2 VNI>:1**. You can't alter this.

For Intra-VNI communication L2VNI specific RT is used. The import RT is used to determine which received routes are applicable to a particular VNI. To establish connectivity, Layer 2 VNIs must be the same between the blueprint and the remote domains. SVI subnets must be identical across domains.

Remote Gateway Topology Representation

Remote EVPN gateways are represented on the topology view as cloud elements with dotted line connections to the blueprint elements with which BGP sessions are established as shown in the image below. (Image below is slightly different from more recent versions.)



SEE ALSO

Commit / Revert Changes to Blueprint | 599

Settings

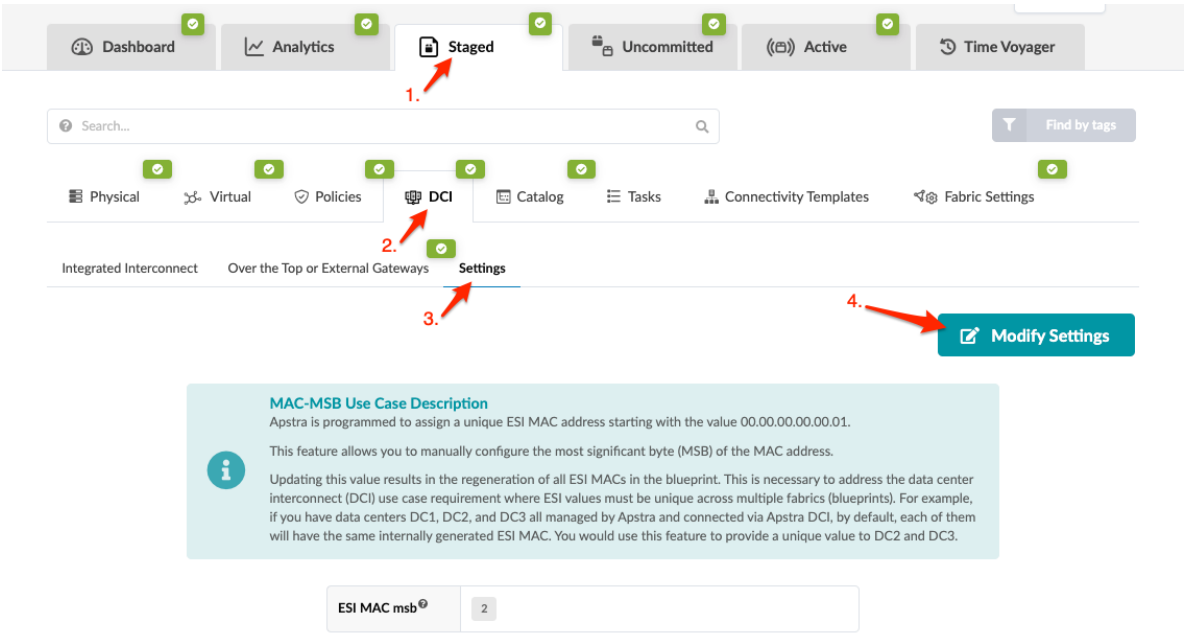
Update ESI MAC msb

> **CAUTION**: Updating the Most Significant Byte (msb) value regenerates all existing ESI MACs in the blueprint.

To enable ESI (EVPN) LAG multihoming, an Ethernet segment identifier (ESI) is mandatory. ESIs identify ESI LAGs. Apstra automatically generates ESI MAC addresses using most significant byte (msb) values. Configuration of the ESI value is rendered as 10 octets. The first octet is 0. The second octet is the most significant byte value. To ensure that multicast MACs are not generated, the second octet must be an even number between 0 and 254. The second through sixth octets are used as the LACP system ID.

Apstra is programmed to assign a unique ESI MAC address starting with the value 00.00.00.00.00.01. The msb value in each Apstra blueprint defaults to the value **2**. If you aren't connecting blueprints (IP fabrics) you can leave the value as is. You can manually configure the most significant byte (msb) of the MAC address. Updating this value results in the regeneration of all ESI MACs in the blueprint. This is necessary to address the data center interconnect (DCI) use case requirement where ESI values must be unique across multiple fabrics (blueprints). For example, if you have data centers DC1, DC2, and DC3 all managed by Apstra and connected via Apstra DCI, by default, each of them will have the same internally generated ESI MAC. You would use this feature to provide a unique value to DC2 and DC3.

1. From the blueprint, navigate to **Staged > DCI > Settings** and click **Modify Settings**.



2. Change the ESI MAC most significant byte to an even number between 0 and 254 that is different from the msbs for all connected blueprints.
3. Click **Save Changes** to save your changes and return to the DCI Settings view.

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

RELATED DOCUMENTATION

Data Center Interconnect Introduction | 421

CHAPTER 10

Catalog

IN THIS CHAPTER

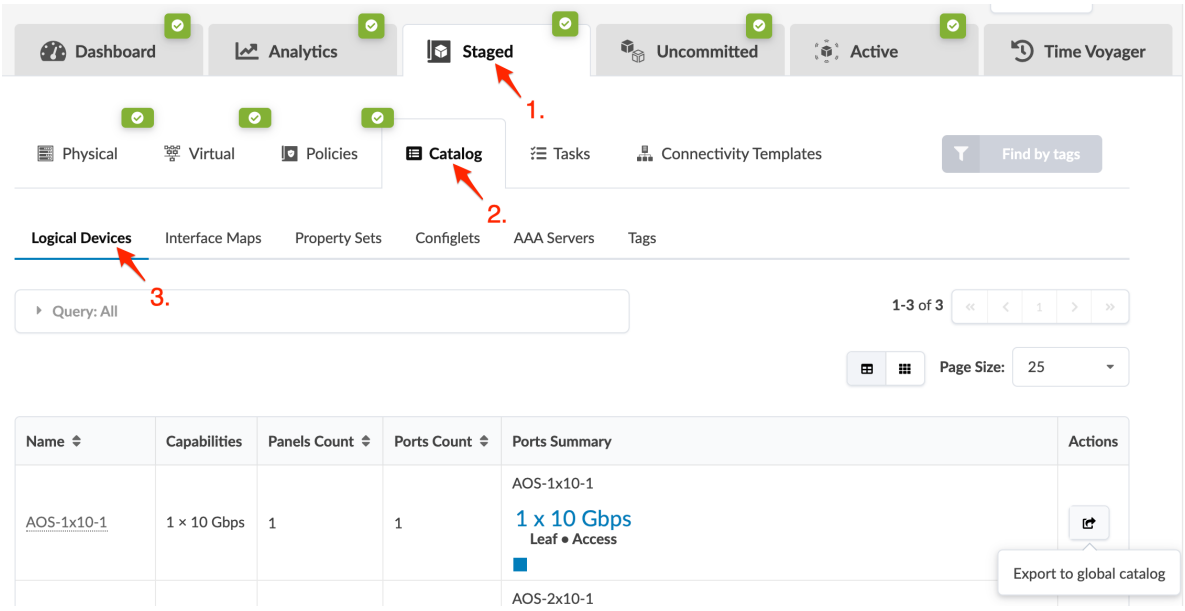
Logical Devices

IN THIS SECTION

Export Logical Device

The logical devices in the blueprint catalog are from the template used to create your blueprint .

1. From the blueprint, navigate to **Staged > Catalog > Logical Devices** and click the **Export to global catalog** button for the logical device to export (in the Actions column on the right side).



2. Select how you want to export the logical device:
   - **Export as new** - to create a new logical device based on the current one in the global catalog. This option doesn't keep references to interface maps. Even if you already have a logical device with the same name in the global catalog you can still export it. Exported logical devices with the same name are identified by the ID instead of by the logical device name.
   - **Export existing** - to create interface maps for this logical device in the global catalog that you can re-import into the blueprint. If you already have a logical device with the same name in the global catalog, you can't use this option. When you export a logical device with this option, the logical device ID and logical device name are the same.
3. Click **Export** to export the logical device and return to the table view.

RELATED DOCUMENTATION

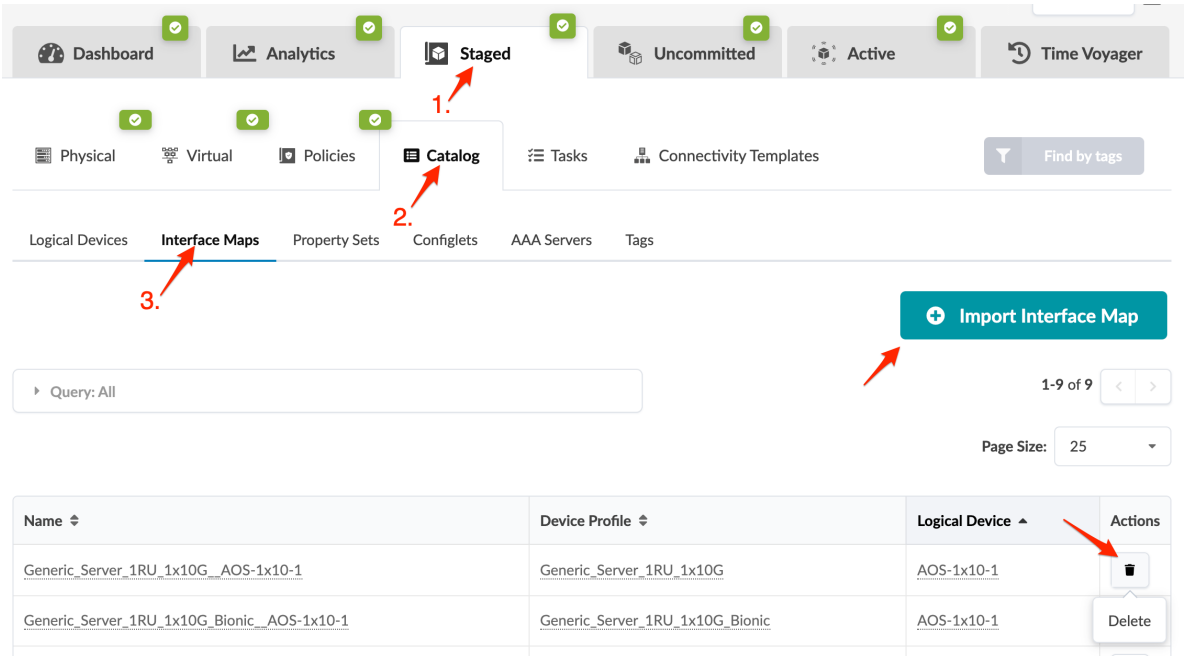
What are Logical Devices | 845

Interface Maps

IN THIS SECTION

Import Interface Map

1. Make sure the "interface map" on page 853 that you want to import is in the design (global) catalog.
2. From the blueprint, navigate to **Staged > Catalog > Interface Maps** and click **Import Interface Map**.



3. Select a logical device and an interface map from the drop-down lists. A preview of your selection appears.
4. Click **Import Selected Interface Map** to stage the import and return to the table view.

RELATED DOCUMENTATION

- What are Interface Maps | 853

Delete Interface Map (Blueprint)

1. From the blueprint, navigate to **Staged > Catalog > Interface Maps** and click the **Delete** button for the interface map to delete (in the Actions column on the right side).
2. Click **Delete** to stage the deletion and return to the table view.

RELATED DOCUMENTATION

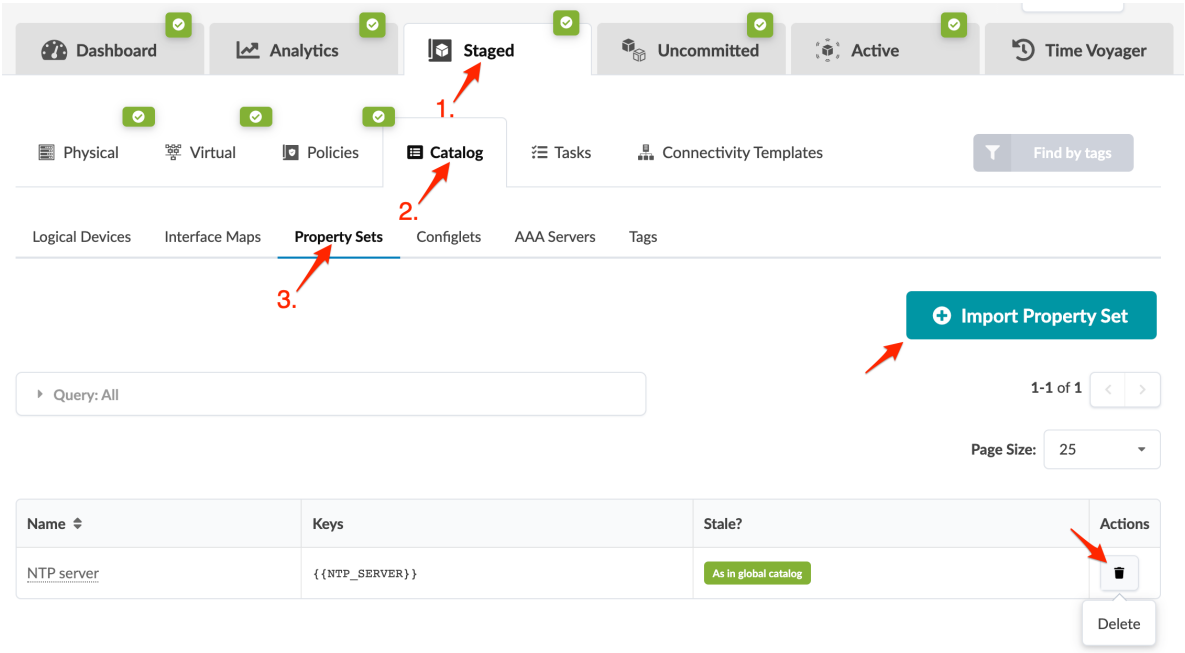
What are Interface Maps | 853

Property Sets

IN THIS SECTION

Import Property Set (Datacenter Blueprint)

1. Make sure the "property set" on page 916 that you want to import is in the design catalog.
2. From the blueprint, navigate to **Staged > Catalog > Property Sets** and click **Import Property Set**.



3. From the drop-down list, select a property set from the design catalog, then click **Import Property Set** to stage the import and return to the table view.

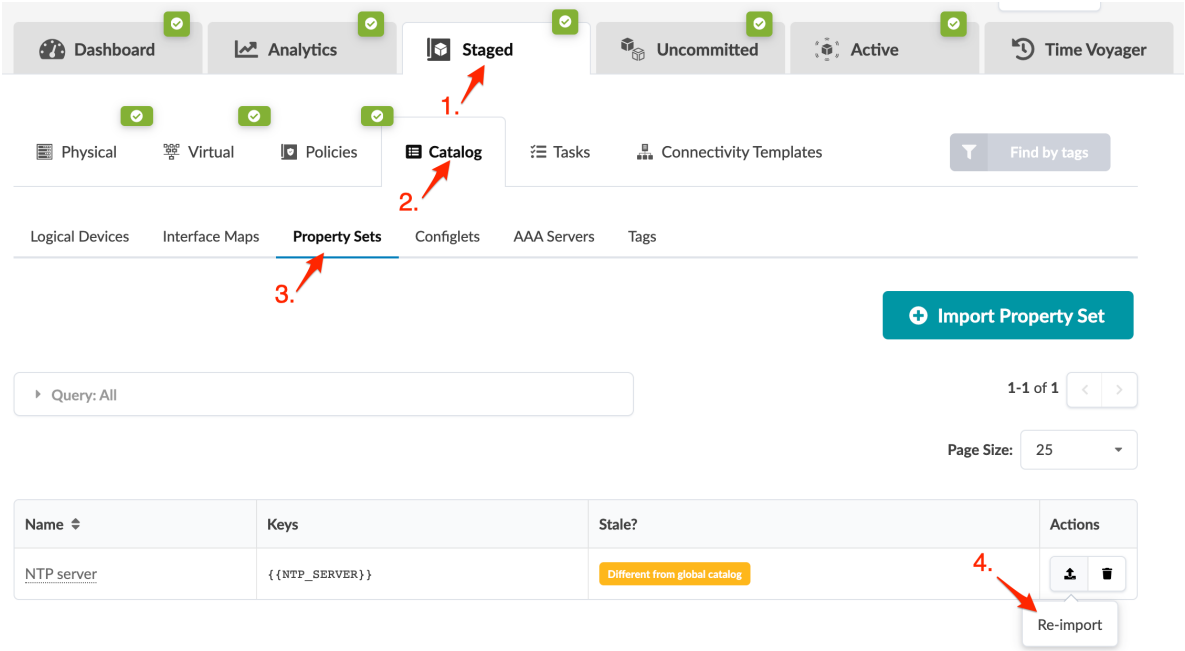
RELATED DOCUMENTATION

What are Property Sets (Datacenter Design) | 916

Re-import Property Set (Datacenter Blueprint)

If a property set that's used in a blueprint is updated in the design (global) catalog, a message appears in the blueprint catalog stating that the property set in the blueprint catalog is **Different from global catalog**. If you want the blueprint to use the updated property set, re-import it.

1. From the blueprint, navigate to **Staged > Catalog > Property Sets**.



2. Click the **Re-import** button for the "stale" property set, then click **Re-import Property Set** to stage the update and return to the table view.

RELATED DOCUMENTATION

What are Property Sets (Datacenter Design) | 916

Delete Property Set (Datacenter Blueprint)

As long as a property set is not used in a configlet, you can unassign it from a device at any time. If it is used in a configlet, a build error occurs and you won't be able to commit the change until you remove the property set from the configlet which resolves that build error.

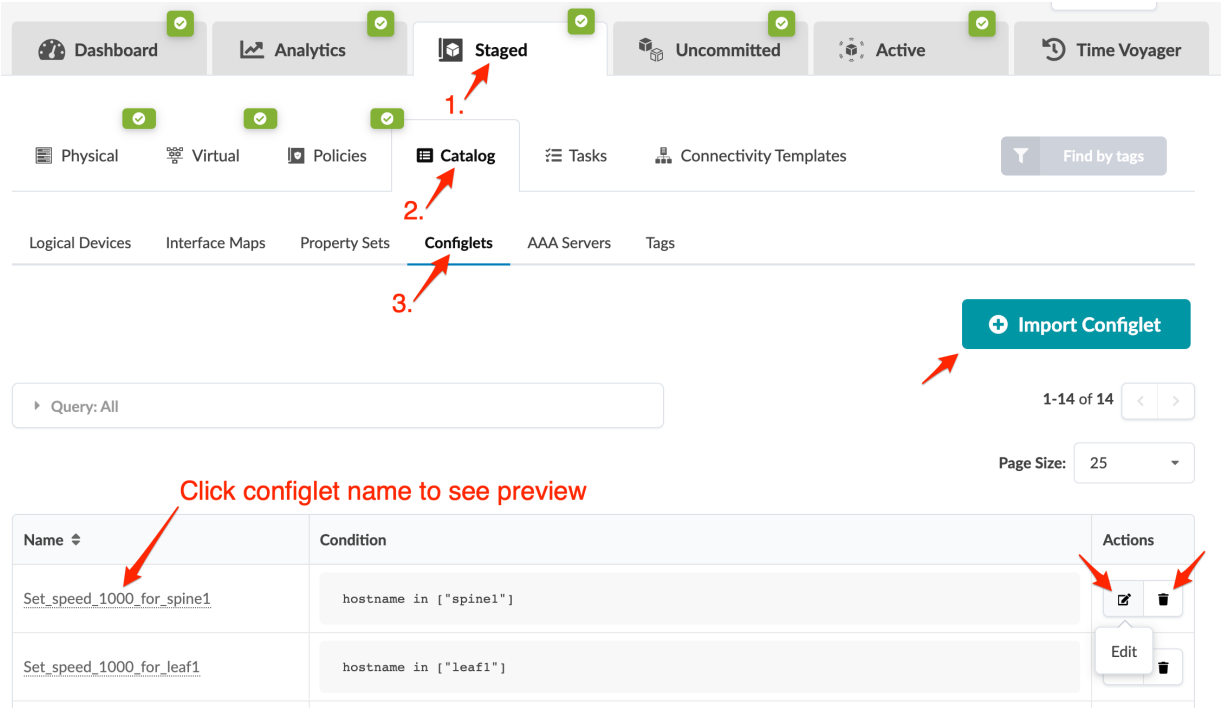
1. From the blueprint, navigate to **Staged > Catalog > Property Sets** and click the **Delete** button for the property set to delete.
2. Click **Delete** to stage the deletion and return to the summary table view.

Configlets

IN THIS SECTION

Configlets (Datacenter Blueprint)

From the blueprint, navigate to **Staged > Catalog > Configlets** to go to blueprint configlets. Configlets are vendor-specific. Apstra software automatically ensures that configlets of a specific vendor aren't applied to devices from a different vendor. You can import, edit, and delete configlets from the blueprint catalog.



NOTE:
service config deployment"Anomalies (Service)" on page 616

NOTE: If an improperly-configured configlet causes the disruption of connectivity between the device and Apstra controller, the device deployment state remains in PENDING forever and will never time out and fail.

For example, if a configlet with misconfigured routing engine firewall filter entry blocks the NETCONF port (tcp 830), the Junos offbox agent can't connect to the device to retrieve the running config. The device deployment remains in PENDING state indefinitely and will never time out and fail. Even if you manually change the device config to unblock NETCONF port (tcp 830), Apstra again re-sends the configuration from the last commit which results in a continuing failure. To recover, you have to re-onboard the device. For more details and the workaround, see the Juniper Support Knowledge Base article KB37291.

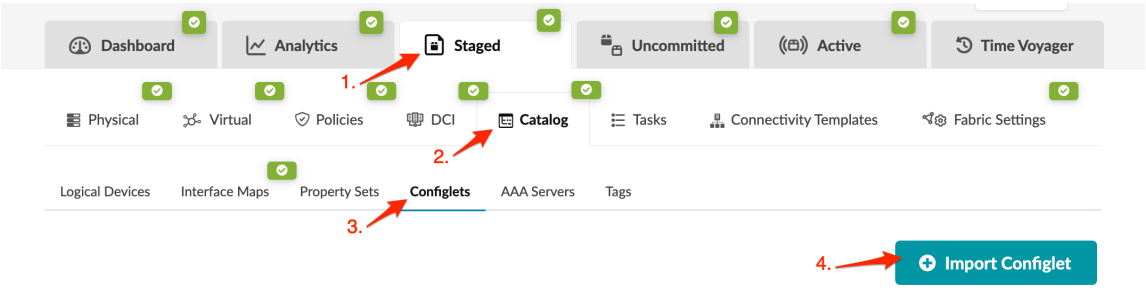
Import Configlet

SUMMARY

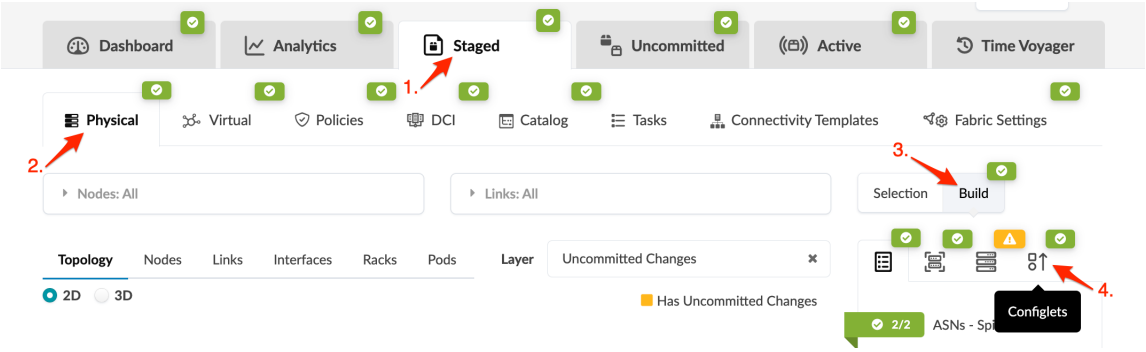
Import a configlet and specify where to apply it in the blueprint.

You've created a configlet in the design (global) catalog. Now you'll import it into your blueprint catalog, set conditions and specify where to apply it in the blueprint.

1. Go to the configlets catalog in the blueprint. You can get to it in a couple of ways:
   - From the blueprint, navigate to **Staged > Catalog > Configlets** to go to the configlets catalog.



   - Or, navigate to **Staged > Physical > Build > Configlets**, then click **Manage Configlets** to go to the configlets catalog.

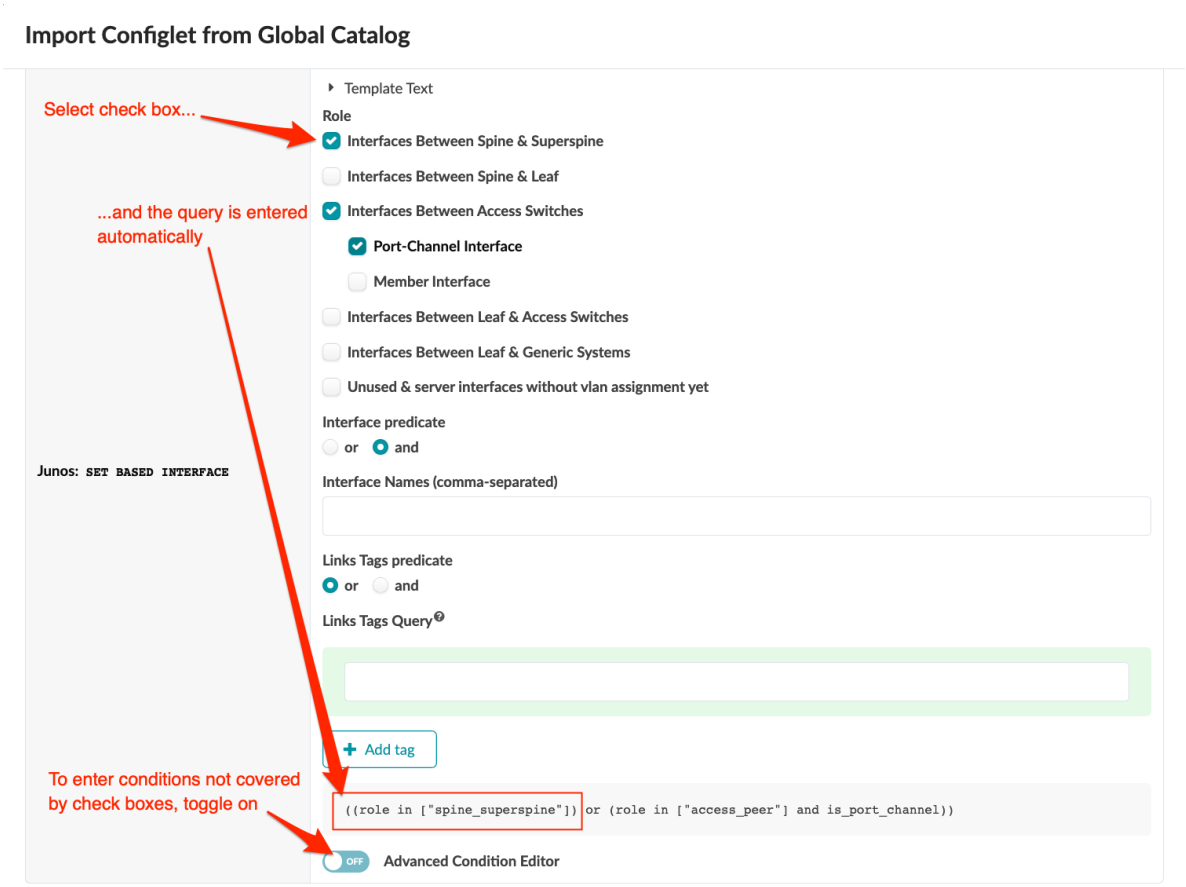


2. Click **Import Configlet**.

   The **Import Configlet from Global Catalog** dialog opens.

3. The **Configlet** drop-down list includes configlets from the global catalog. Select a configlet from the list. (An empty list means you haven't created any configlets yet.)

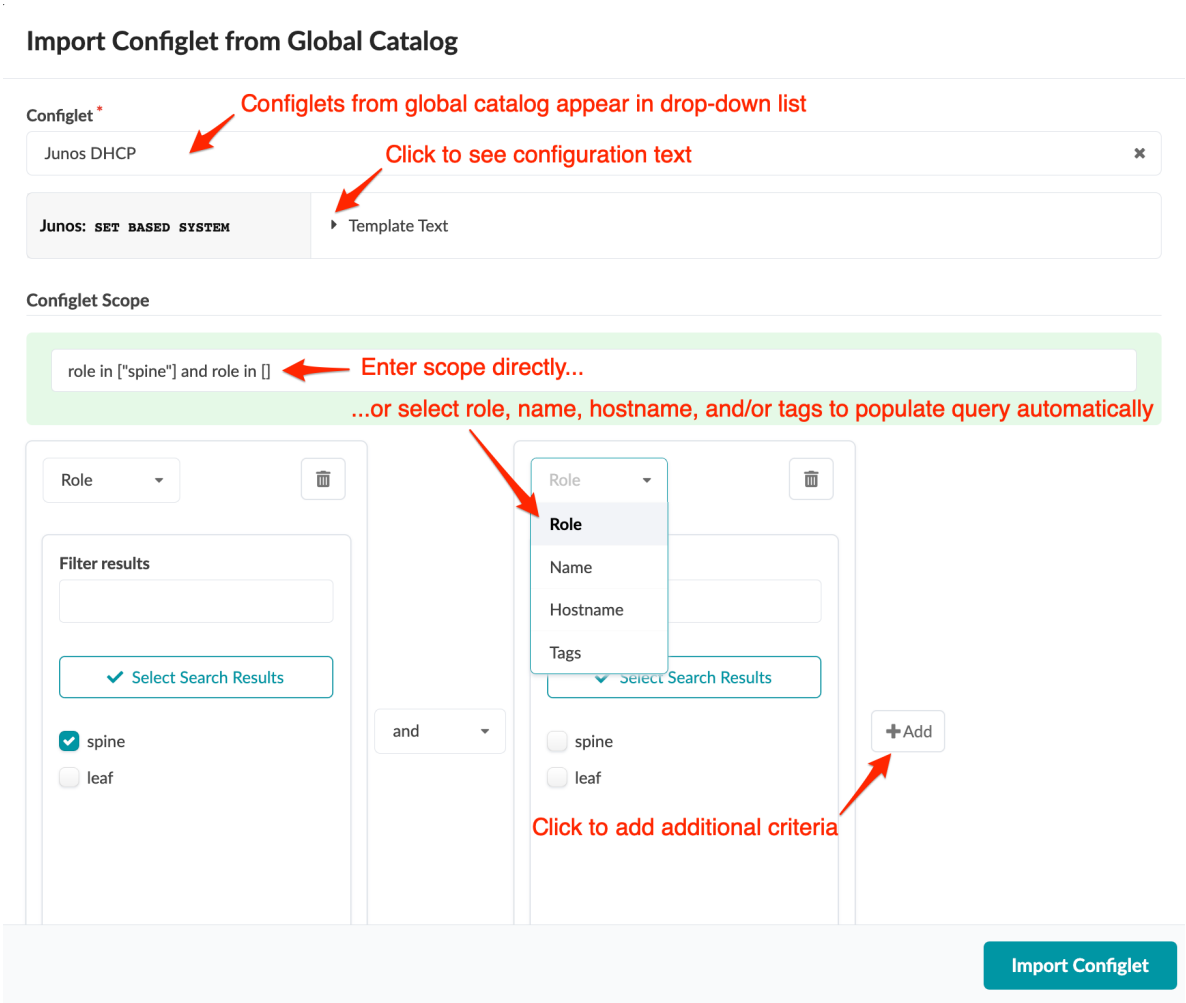
4. If you're importing a configlet with a section for interfaces, you can define the conditions for applying the configlet. Select check boxes for one or more interface roles, enter specific interface names and/or link tags. If conditions are not covered by the check boxes, toggle on the **Advanced Condition Editor**. The conditions field becomes editable and you can enter specific conditions.



5. For all configlet types, specify the application scope. For example, you may want to apply the configlet to all generic systems that you've tagged as firewalls. Instead of listing all applicable generic

systems, you can just add one tag to the scope. You can define the scope in a couple of different ways:

- Enter scope (query) directly. Auto-complete assists you as you type.
- Select Role, Name, Hostname or Tags from the drop-down list, then select the check boxes that apply. (You can "apply tags to interfaces" on page 216.) To add an additional definition, click **+Add**.



6. Click **Import Configlet** to stage the configlet and return to the configlet catalog.

RELATED DOCUMENTATION

Configlets Introduction | 908

Create Configlet (Design) | 913

Configlets (Datacenter Blueprint) | 442

Edit Configlet (Blueprint)

IN THIS SECTION

Edit Where Configlet is Applied

When you import a configlet into a blueprint catalog, you specify where in the blueprint to apply it based on roles, IDs, hostnames and/or tags. After you've imported a configlet, you can change this scope.

1. From the blueprint, navigate to **Staged > Catalog > Configlets** and click the **Edit** button for the configlet to edit.
2. Make your changes to the configlet scope. The options are the same as for "importing a configlet" on page 443.
3. Click **Update** to stage the update and return to the table view.

Edit Configlet Contents

You can't change configlet generators (template text, negation template text, filename) directly in blueprints. If an existing configlet is no longer relevant, you can delete it and import a new or revised one.

1. "Edit" on page 914 or "create" on page 913 a configlet in the design (global) catalog.
2. "Delete" on page 447 the configlet from the blueprint catalog.
3. "Import" on page 443 the configlet into the blueprint catalog from the design (global) catalog.
4. Commit the changes.

SEE ALSO

Configlets (Datacenter Blueprint) | 442

Import Configlet | 443

Commit / Revert Changes to Blueprint | 599

Delete Configlet (Blueprint)

When you delete a configlet, it's removed from all devices within its scope.

1. From the blueprint, navigate to **Staged > Catalog > Configlets** and click the **Delete** button for the configlet to delete.
2. Click **Delete** to stage the deletion and return to the table view.

RELATED DOCUMENTATION

Configlets (Datacenter Blueprint) | 442

Import Configlet | 443

AAA Servers

IN THIS SECTION

AAA Servers (Datacenter Blueprint)

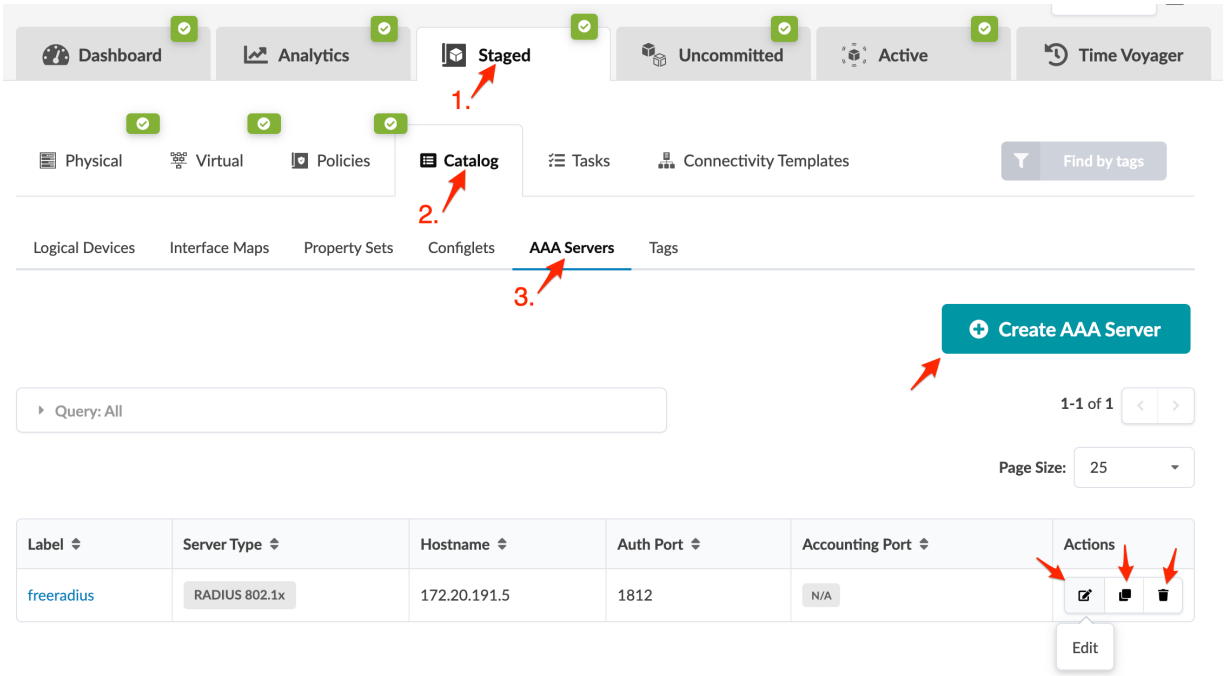
IN THIS SECTION

AAA Servers Overview

AAA servers are used with "interface policies" on page 388. AAA servers include the following details:

Parameter	Description
Label	To identify the AAA server
Server Type	• **RADIUS 802.1x** - If an 802.1x policy is bound to at least one interface on a switch, all defined AAA RADIUS 802.1x servers will be added to that switch. The server is not rendered unless it is needed. • **RADIUS COA (Change of Authorization)** - Used by switches to enable Dynamic Authorization Server (DAS) requests from RADIUS servers. This enables the switch to 'trust' the given RADIUS server to do assign dynamic VLANs after authentication instead of during auth. All RADIUS COA implementations are hard-coded to auth port 3799.
Hostname	
Auth Ports	
Accounting Port	optional

From the blueprint, navigate to **Staged > Catalog > AAA Servers** to go to the AAA servers catalog. You can create, clone, edit, and delete AAA servers.



Create AAA Server

1. From the blueprint, navigate to **Staged > Catalog > AAA Servers** and click **Create AAA Server**.
2. Enter a label, select the server type (RADIUS 802.1x, RADIUS COA), enter a hostname, key, auth port, and (optional) accounting port.
3. Click **Create** to stage the server and return to the table view.

Edit AAA Server

1. From the blueprint, navigate to **Staged > Catalog > AAA Servers** and click the **Edit** button for the AAA server to edit.
2. Make your changes, then click **Update** to stage the update and return to the table view.

Delete AAA Server

1. From the blueprint, navigate to **Staged > Catalog > AAA Servers** and click the **Delete** button for the AAA server to delete.
2. Click **Delete** to stage the deletion and return to the table view.

Configure AAA RADIUS Server

Configuring AAA RADIUS servers are external to Apstra software. The example below shows the files to configure for *FreeRADIUS*.

/etc/freeradius/clients.conf -- has credentials for each switch

```
client Arista-7280SR-48C6-1 {
    shortname = Arista-7280SR-48C6-1
    ipaddr    = 172.20.191.10
    secret    = testing123
    nastype   = other
}
```

/etc/freeradius/users -- has users and MAC addresses to authenticate. Tunnel-Private-Group-Id shows a dynamic VLAN ID, which is optional.

```
leaf1-server1 ClearText-Password := "password"

"52:54:00:37:d5:e1" Cleartext-Password := "52:54:00:37:d5:e1"
    Tunnel-Type = VLAN,
    Tunnel-Medium-Type = IEEE-802,
    Tunnel-Private-Group-Id = "50"
```

This example shows a simple credential; when you configure you may use any EAP method that both the client and RADIUS server support.

Configure Client Supplicant

Configuring client supplicant is external to Apstra software. The following is an example for *wpa_supplicant*.

/etc/wpa_supplicant/aos_wpa_supplicant.conf

```
# Ansible managed
ctrl_interface=/var/run/wpa_supplicant
# Default version is 0 - ensure we're using modern protocols.
eapol_version=2
# Don't scan for wifi.
ap_scan=0
# Hosts will be configured to authenticate with usernames that match their
# Slicer DUT name, configured in radius_server playbook.
network={
```

```
    key_mgmt=IEEE8021X
    eap=TTLS MD5
    identity="leaf1-server1"
    anonymous_identity="leaf1-server1"
    password="password"
    phase1="auth=MD5"
    phase2="auth=PAP password=password"
    eapol_flags=0
}
```

Tags

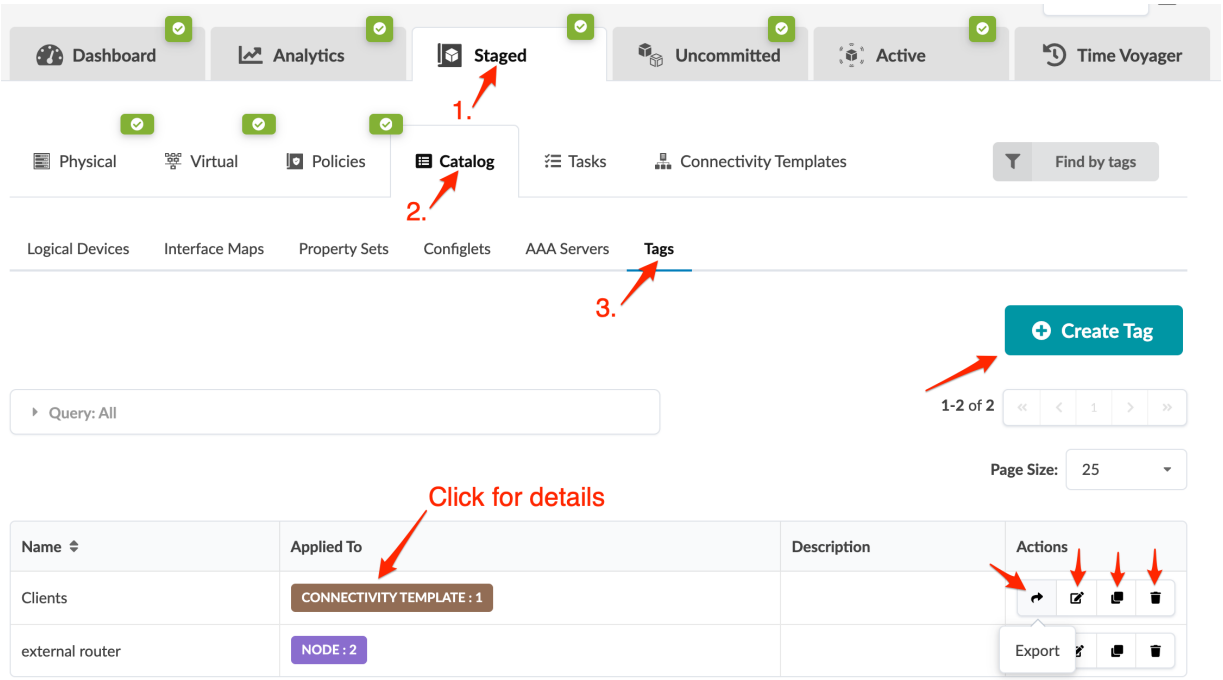
IN THIS SECTION

Tags (Datacenter)

IN THIS SECTION

Tags Overview

You can apply tags to nodes, links and connectivity templates in your blueprint. When you create a blueprint, if you added tags to the design elements used to create that blueprint (rack types and templates), those tags are added to the blueprint **Tags** catalog. From the blueprint, navigate to **Staged > Catalog > Tags** to go to the tags blueprint catalog. You can add, clone, edit and delete blueprint tags. You can also import global catalog tags to the blueprint catalog and export blueprint tags to the global catalog.



Search Tags

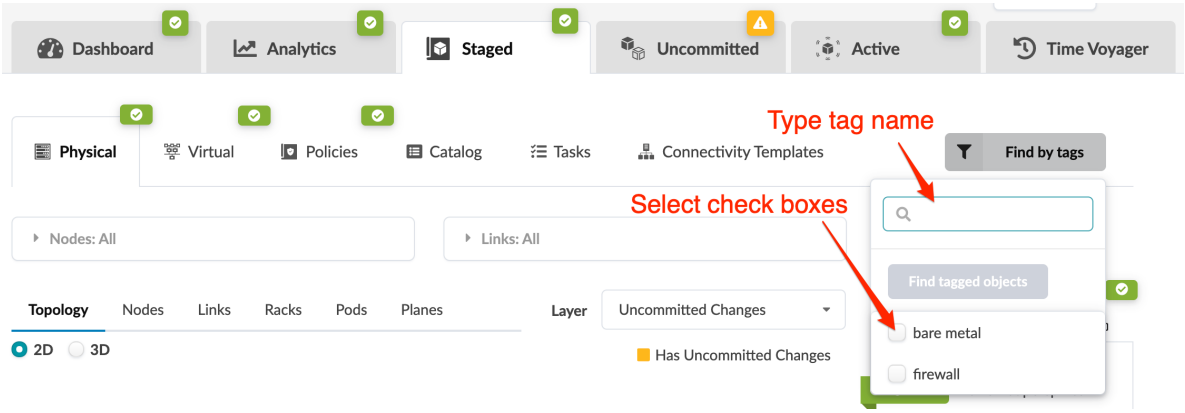
You can filter tagged elements based on tag names and/or element types.

1. From the blueprint, navigate to **Staged > Catalog > Tags** and click **Query** to open the dialog.
2. Enter search criteria:
   - To see elements associated with tags, enter tag name(s) in the **Name** field.
   - To see tags that elements are associated with, select element type(s) from the drop-down list in the **Applied To** field.
   - To filter both by tag name and element type, enter details in both fields.
3. Click **Apply** to see filtered results in the table.
4. To go to the table view for a filtered element type, click the element type in the **Applied To** column. From there you can drill down for more details on a specific element.

Find by Tags

With **Find by Tags**, you can search the entire blueprint for nodes, links, and connectivity templates that have associated tags.

1. From any page in the staged (or active) blueprint click **Find by Tags** (right side).
2. Either start typing to filter tags for selection, or select one or more check boxes.



3. Click **Find tagged objects** to display all objects with those tags.

Create Tag in Catalog (Datacenter)

1. From the blueprint, navigate to **Staged > Catalog > Tags** and click **Create Tag**.
2. Select **New** and enter a name and (optional) description. Names are case-insensitive.
3. Click **Create** to stage the tag addition and return to the table view. The newly created tag appears in the table.

RELATED DOCUMENTATION

What are Tags | 922

Export Tag from Catalog (Datacenter)

1. From the blueprint, navigate to **Staged > Catalog > Tags** and click the **Export** button for the tag to export. If a tag exists in the global catalog with the same name you won't be able to export it. (The export button will be nonfunctional.)
2. Click **Export** to export the tag to the global catalog and return to the table view.

RELATED DOCUMENTATION

What are Tags | 922

Import Tag to Catalog (Datacenter)

1. From the blueprint, navigate to **Staged > Catalog > Tags** and click **Create Tag**.
2. Select **Import from Global Catalog**, select a tag from the drop-down list and enter an (optional) description.
3. Click **Create** to stage the tag import and return to the table view. The newly created tag appears in the table.

RELATED DOCUMENTATION

What are Tags | 922

Change Tag Description (Datacenter)

1. From the blueprint, navigate to **Staged > Catalog > Tags** and click the **Edit** button for the tag to edit.
2. Change the description.
3. Click **Update** to stage the change and return to the table view.

RELATED DOCUMENTATION

What are Tags | 922

Delete Tag from Catalog (Datacenter)

1. From the blueprint, navigate to **Staged > Catalog > Tags** and click the **Delete** button for the tag to delete.
2. Click **Delete** to stage the deletion and return to the table view.

RELATED DOCUMENTATION

What are Tags | 922

CHAPTER 11

Tasks

IN THIS CHAPTER

Tasks (Datacenter) Staged

From the blueprint, navigate to **Staged > Tasks** to go to task history. Blueprint task details include type of task, task status (succeeded, failed, in progress), date/time started, date/time last updated, and the duration of the task. For any failed tasks, you can click to see error messages.

CHAPTER 12

Connectivity Templates

IN THIS CHAPTER

Connectivity Templates Introduction

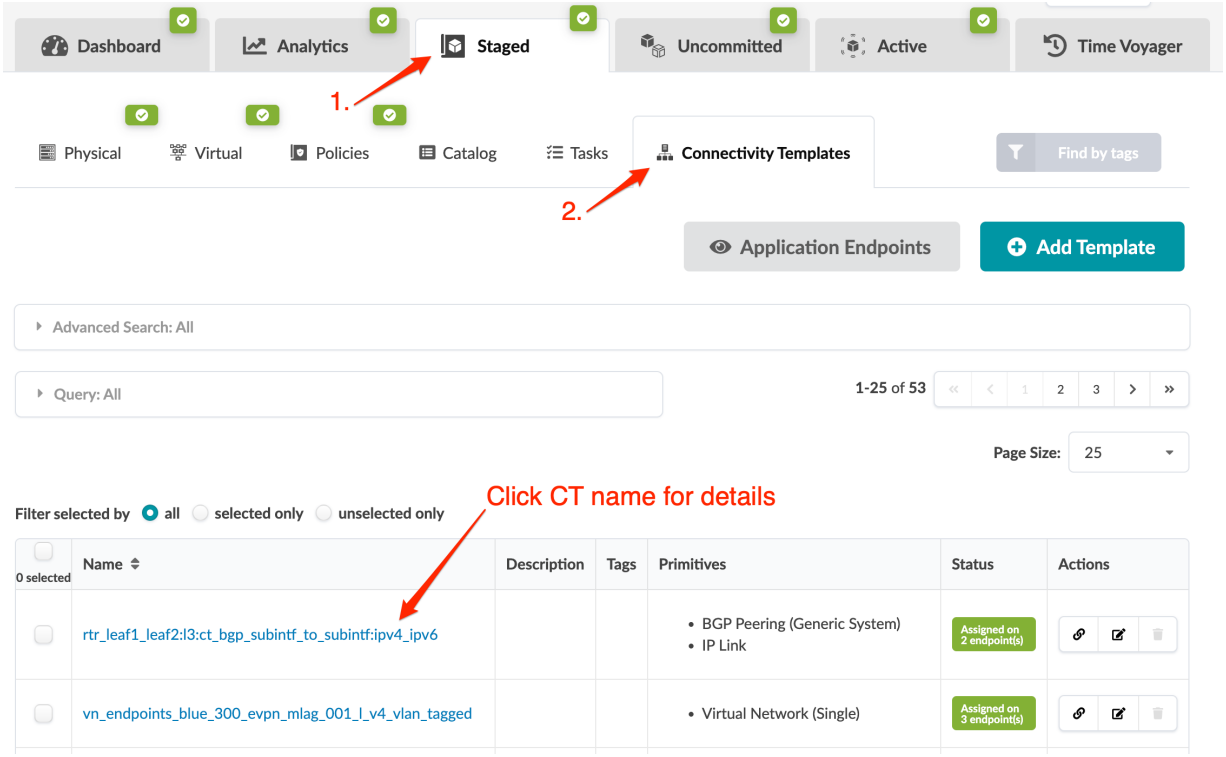
Connectivity templates enable you to apply various network configurations to devices connected to generic systems, as a Day 2 operation. Devices could be leaf devices, spine devices, or in 5-stage Clos topologies, superspine devices. Some use cases for connectivity templates include the following:

- Assigning Apstra virtual network endpoints (tagging and untagging VLAN ports) to connect Layer 2 servers.
- Creating Layer 3 interfaces and VLAN-tagged sub-interfaces with BGP routing between Apstra fabric border-leaf devices and external routers.

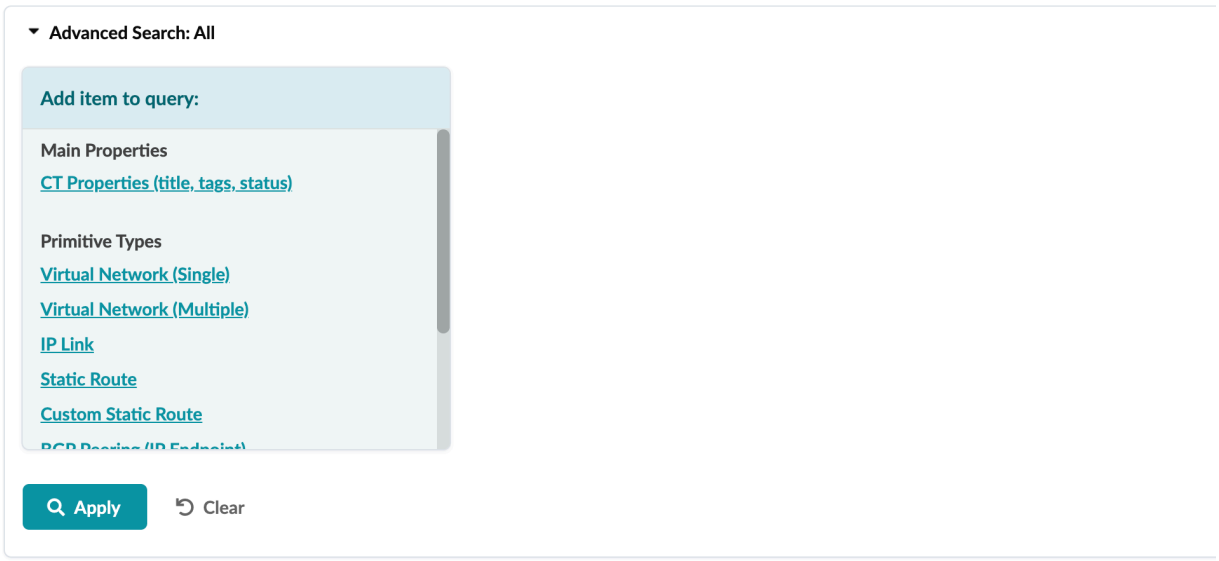
Connectivity templates consist of combinations of primitives as described in later sections.

Use connectivity templates to configure the required external routing connections to routing zones. To see static routes and protocol sessions, navigate to **Staged > Virtual** in the blueprint.

From the blueprint, navigate to **Staged > Connectivity Templates** to go to the connectivity template table view. You can create, assign, edit, and delete connectivity templates.



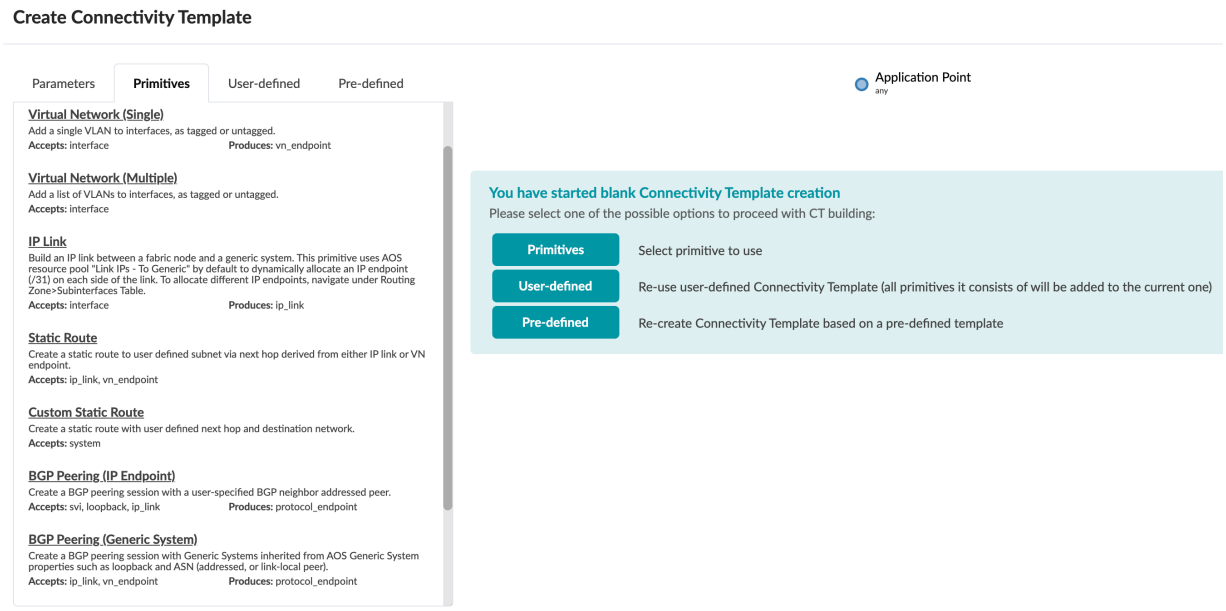
With advanced search you can filter based on primitive types, and based on the types, you can show parameters and filter on those parameters. You can take this search to multiples levels. For example, you can search for all the logical links in routing zone green or all the static routes with the same next hop.



Primitives

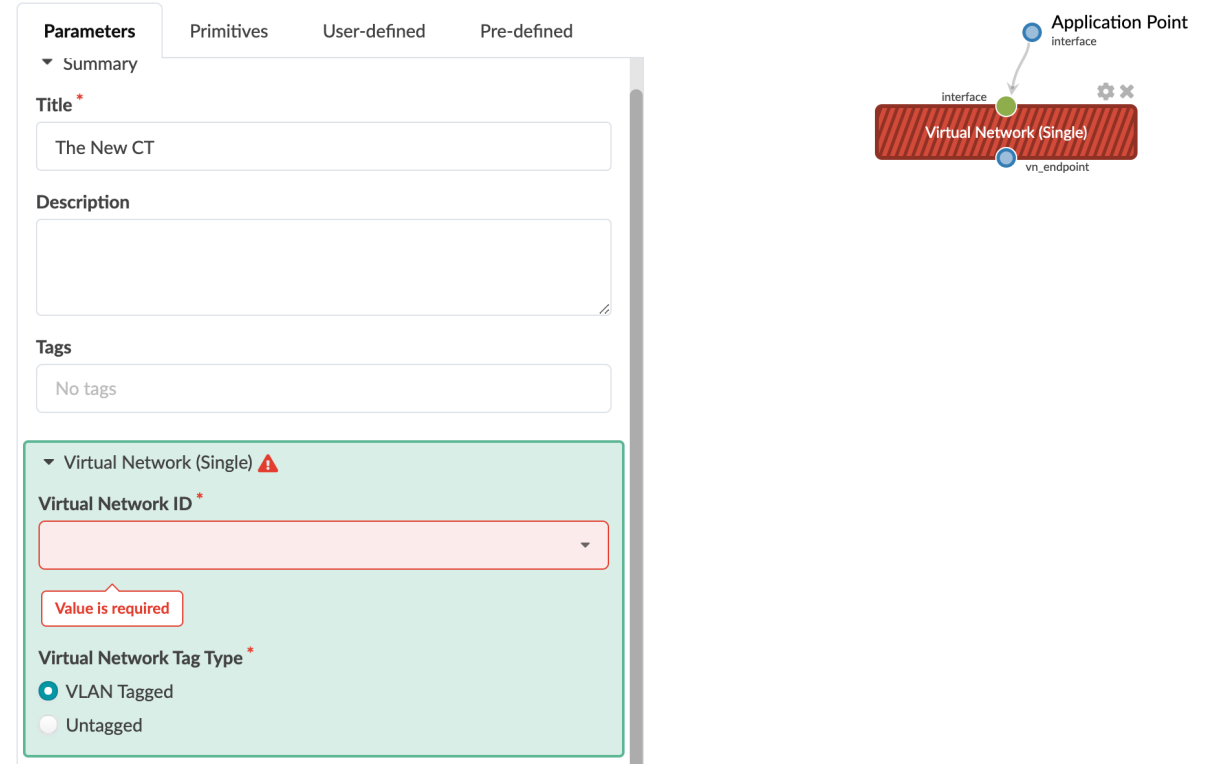
IN THIS SECTION

The **Primitives** tab includes the supported configuration functions that can be added to connectivity templates.



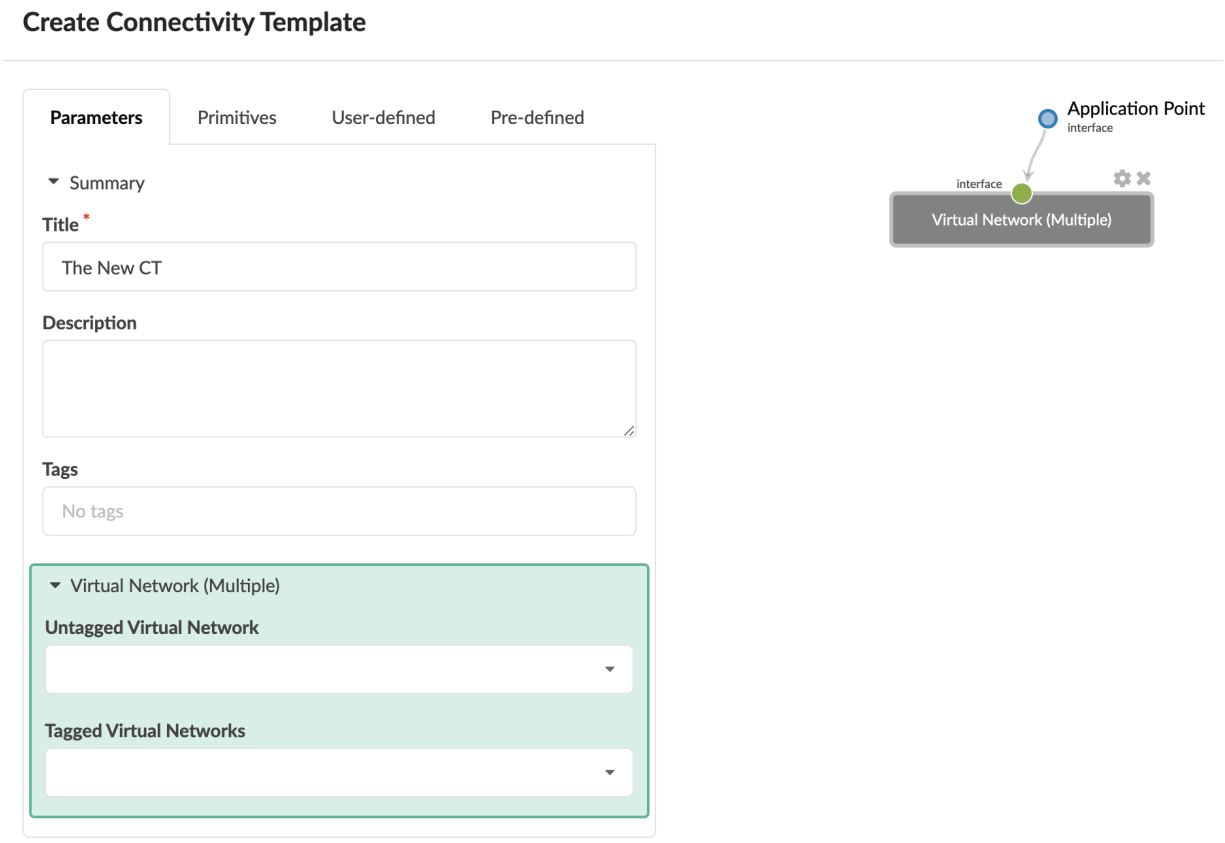
Primitive: Virtual Network (Single)

The virtual network (single) primitive ends with a **vn_endpoint** point that can optionally connect to another compatible primitive, such as BGP peering (generic system).



Primitive: Virtual Network (Multiple)

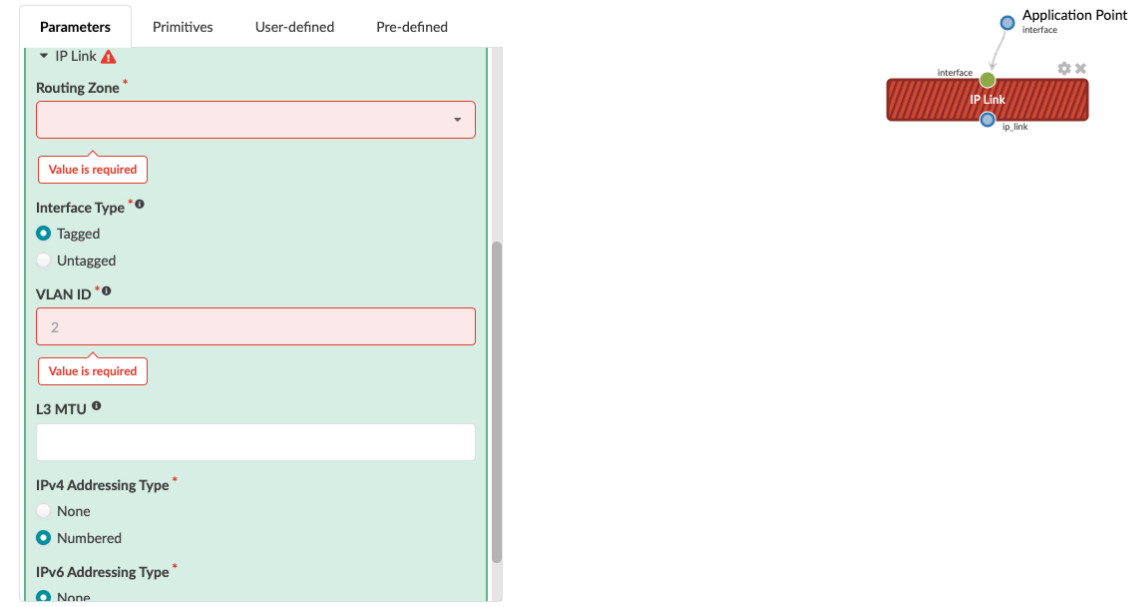
Unlike the virtual network (single) primitive, the virtual network (multiple) primitive cannot connect another primitive.



Primitive: IP Link

IP link uses Apstra resource pool **Link IPs - To Generics** (by default) to dynamically allocate an IP endpoint (/31) on each side of the link. You can create an IP link for any routing zone, including the default routing zone. You can use an untagged link, even if it's for a non-default routing zone. If you select a tagged interface, the VLAN ID is required.

The IP link primitive ends with an **ip_link** point that can optionally connect to another compatible primitive, such as BGP peering (generic system).



The **L3 MTU** field enables you to update the MTU on subinterfaces.

If you select **Numbered** for **IPv4 Addressing Type**, an IPv4 address is automatically assigned from a resource pool. You may need to know what the IP address is, so you can set the correct address on the load balancer connected to the IP link, for example. To see (and change) that IPv4 address, navigate to **Staged > Routing Zones**, then select the associated routing zone name and scroll down to the **Interfaces** section. You can also see the IPv4 addresses in the table at **Staged > Physical > Interfaces**. If you need to change the link IP address, you can link from there directly to the routing zone and change it from there.

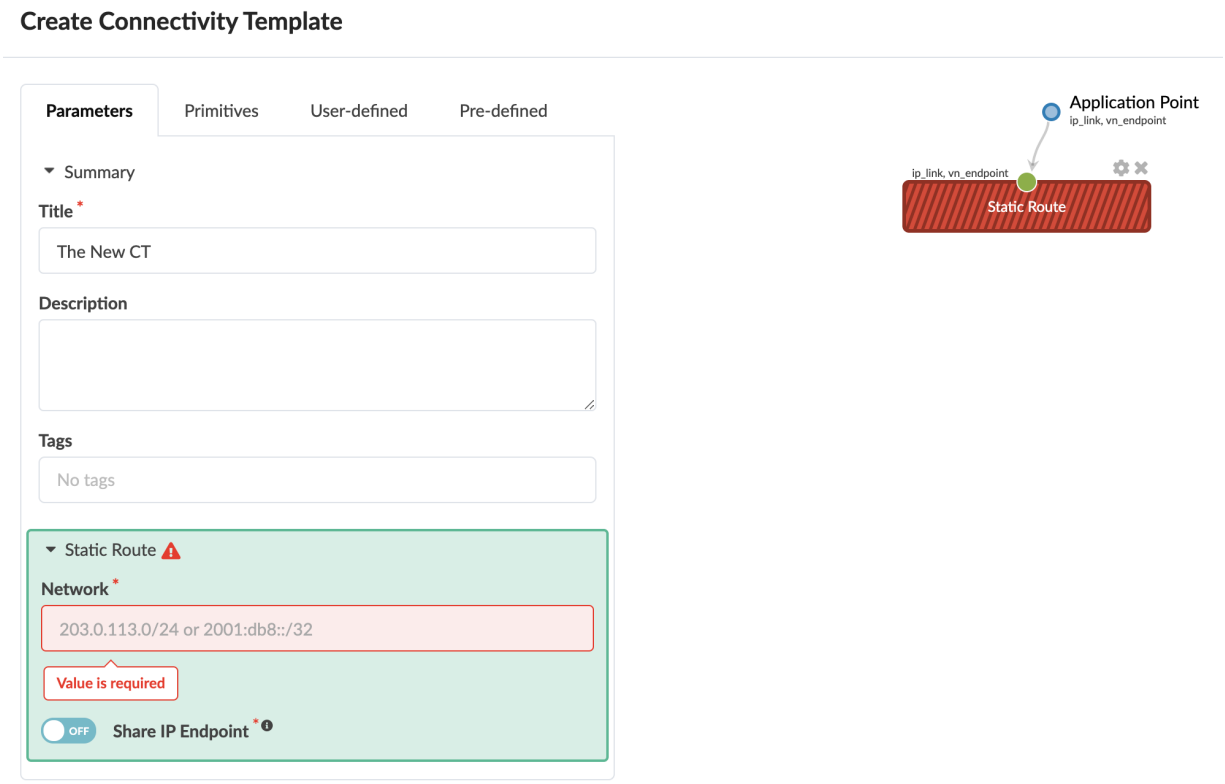
RELATED DOCUMENTATION

Change Interface IP Address | 211

Primitive: Static Route

Next-hop is derived from either the IP link or virtual network endpoint. If the remote peer IP is shared across the generic system, then share the IP endpoint.

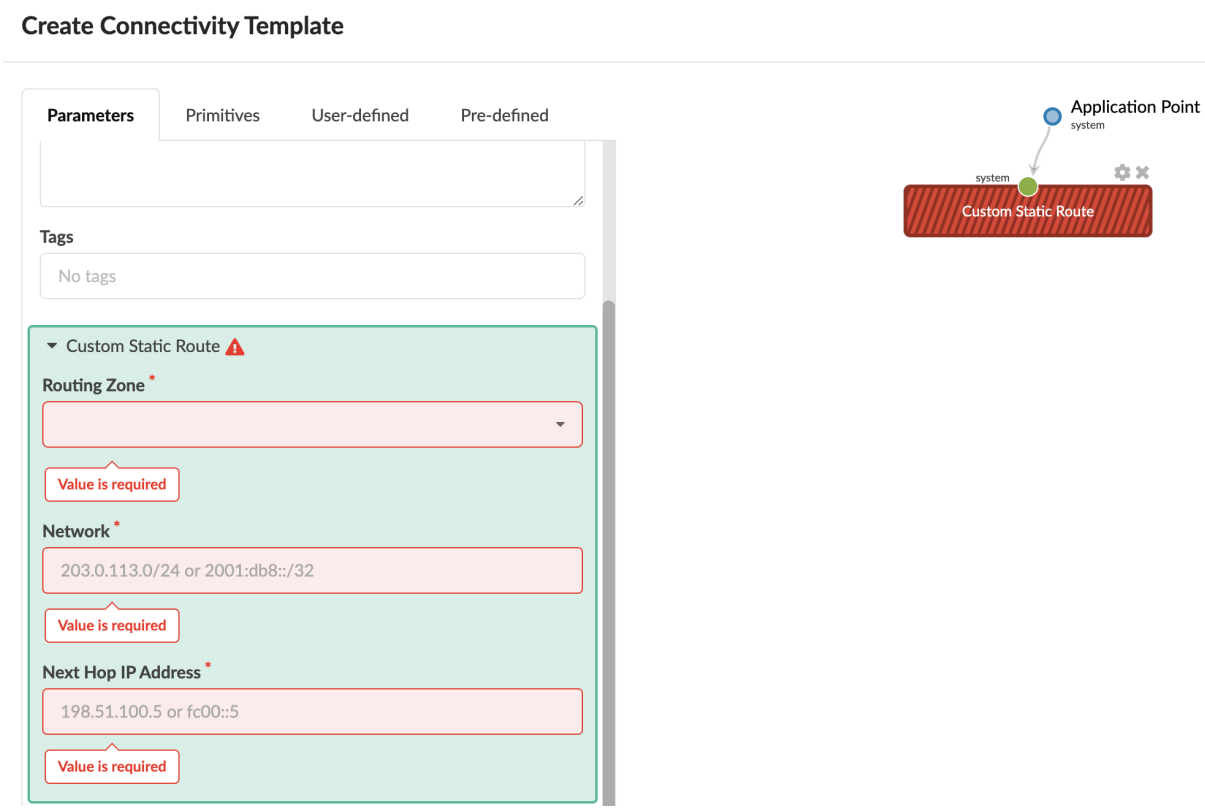
The **Static Route** primitive uses the next available IP address as the next-hop. To use a specific next-hop IP address, use the **Custom Static Route** instead.



Primitive: Custom Static Route

If the next-hop IP address is not accessible, the static route will not be installed. Apstra software cannot monitor the next-hop IP and will not alert you if it is not accessible. It is your responsibility to configure the custom static route primitive correctly.

Connectivity templates using this primitive can only be assigned to leaf systems and cannot be combined with interface primitives.



Primitive: BGP Peering (IP Endpoint)

The BGP peering (IP endpoint) primitive creates a BGP peering session with a user-specified BGP neighbor addressed peer. You can use this to create a BGP peering session to a Layer 3 server running BGP connected to an Apstra virtual network.

The following parameters must be configured:

- Neighbor ASN type (static, dynamic)
- ASN (if Neighbor ASN type is static)
- IPv4 AFI
- IPv6 AFI
- TTL - BGP Time to Live
  - When you set TTL to 0, nothing is configured and the device defaults are used.
  - When you set TTL to 1, Cisco NX-OS and FRR-based BGP (SONiC) render disable-connected-check. Otherwise, TTL values render ebgp-multihop on specific BGP neighbors.

- Enable BFD - Enable BFD with interval: 1 sec, multiple: 3 sec
  - This enables BFD for the BGP peering. Multihop BFD is only supported for Junos, which is activated by default. For non-Junos devices, set TTL to 1.
- BGP Password
- BGP Keep Alive Timer (seconds)
- BGP Hold Time Timer (seconds)
- Local ASN - Configured on a per-peer basis. It allows a router to appear to be a member of a second AS by prepending a local-as ASN, in addition to its real ASN, announced to its eBGP peer, resulting in an AS path length of two.
- IPv4 address of peer (if IPv4 AFI is enabled)
- IPv6 address of peer (if Ipv6 AFI is enabled)

You can connect a routing policy primitive to a BGP peering (IP endpoint)

Parameters | Primitives | User-defined | Pre-defined

Search...

▼ Summary

Title *

The New CT

Description

Tags

No tags

▼ BGP Peering (IP Endpoint)

Neighbor ASN Type *

- (•) **Static**
- ( ) **Dynamic**

ASN

64496

ON **IPv4 AFI** *

OFF **IPv6 AFI** *

TTL *

Primitive: BGP Peering (Generic System)

The BGP peering (generic system) primitive creates a BGP peering session with a generic system. The generic system is inherited from Apstra generic system properties, such as loopback and ASN (addressed, link-local peer). This primitive connects to a virtual network (single) or IP link connectivity point primitive.

The following parameters must be configured:

- IPv4 AFI
- IPv6 AFI
- BGP Time to Live (TTL)
  - When you set TTL to 0, nothing is configured and the device defaults are used.
  - When you set TTL to 1, Cisco NX-OS and FRR-based BGP (SONiC) renders disable-connected-check. Otherwise, TTL values render ebgp-multihop on specific BGP neighbors.
- Enable BFD - Enable BFD with interval: 1 sec, multiple: 3 sec
  - This enables BFD for the BGP peering. Multihop BFD is only supported for Junos, which is activated by default. For non-Junos devices, set TTL to 1.
- BGP Password
- BGP Keep Alive Timer (seconds)
- BGP Hold Time Timer (seconds)
- IPv4 Addressing Type (none, addressed)
- IPv6 Addressing Type (none, (addressed if IPv6 applications are enabled) link local)
- Local ASN - Configured on a per-peer basis. It allows a router to appear to be a member of a second autonomous system (AS) by prepending a local-as AS number, in addition to its real AS number, announced to its eBGP peer, resulting in an AS path length of two.
- Neighbor ASN Type (static, dynamic)
- Peer From (loopback, interface)
- Peer To (loopback, interface/IP endpoint, interface/shared IP endpoint)
  - Loopback: use this option to peer with the loopback address of a single remote system.
  - Interface/IP endpoint: use this option to peer with the IP address of a single remote system link or routed vlan interface.

- Interface/Shared IP endpoint: use this option for any scenario where the remote peer IP address is shared across multiple remote systems.

You can connect a routing policy primitive to a BGP peering (generic system).

Parameters | Primitives | User-defined | Pre-defined

Search...

Summary

Title *

The New CT

Description

Tags

No tags

BGP Peering (Generic System)

ON **IPv4 AFI** *

OFF **IPv6 AFI** *

TTL *

2

OFF **Enable BFD** *

Password

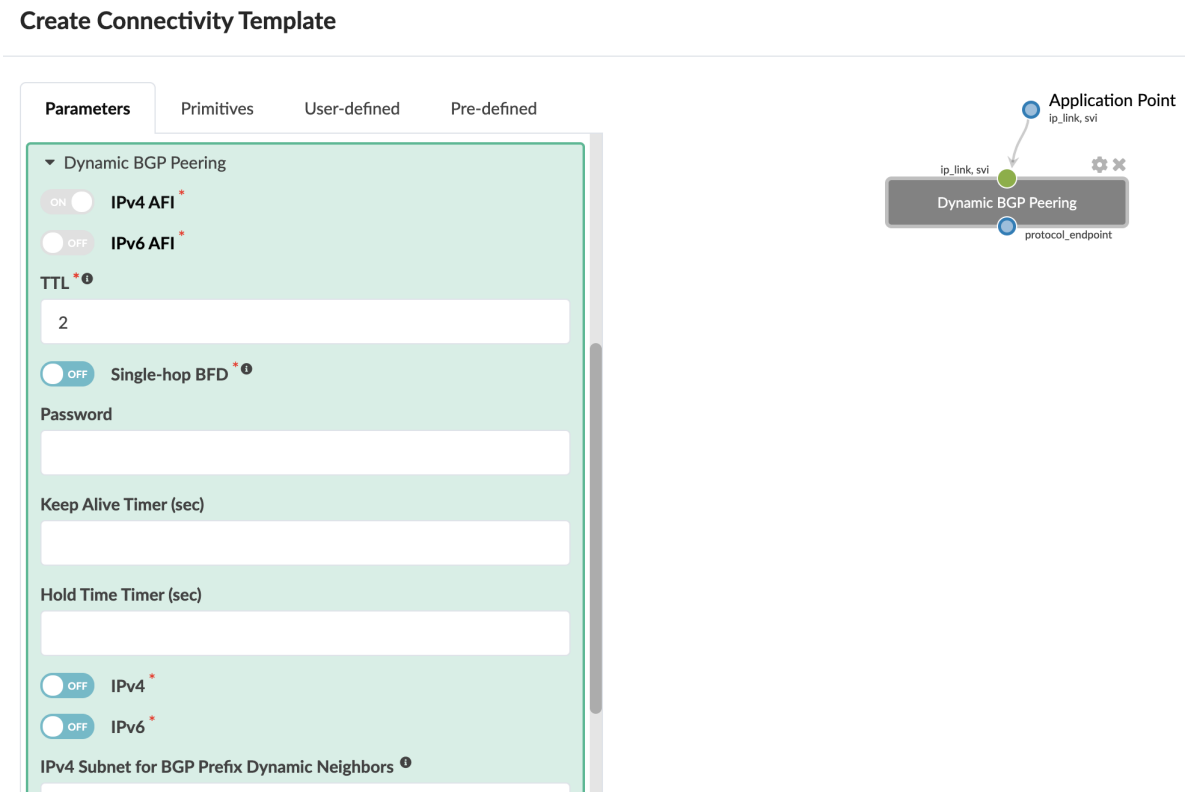
Primitive: Dynamic BGP Peering

The dynamic BGP peering primitive enables dynamic peering on selected devices and virtual networks.

The following parameters must be configured:

- IPv4 AFI
- IPv6 AFI
- BGP Time to Live (TTL)
  - When you set TTL to 0, nothing is configured and the device defaults are used.
  - When you set TTL to 1, Cisco NX-OS and FRR-based BGP (SONiC) renders disable-connected-check. Otherwise, TTL values render ebgp-multihop on specific BGP neighbors.
- Single-hop BFD
  - This enables BFD for the BGP peering. Multihop BFD is only supported for Junos, which is activated by default.
- BGP Password
- BGP Keep Alive Timer (seconds)
- BGP Hold Time Timer (seconds)
- IPv4
- IPv6
- IPv4 subnet for BGP prefix dynamic neighbors. If you leave this field blank, Apstra uses the local virtual network (from when you assigned the connectivity template) as the subnet value. In this case, if the virtual network only has a virtual gateway IP address, and it doesn't have any specific IP address per leaf switch, then it also renders an additional IP address on the leaf SVI besides the virtual gateway IP address.

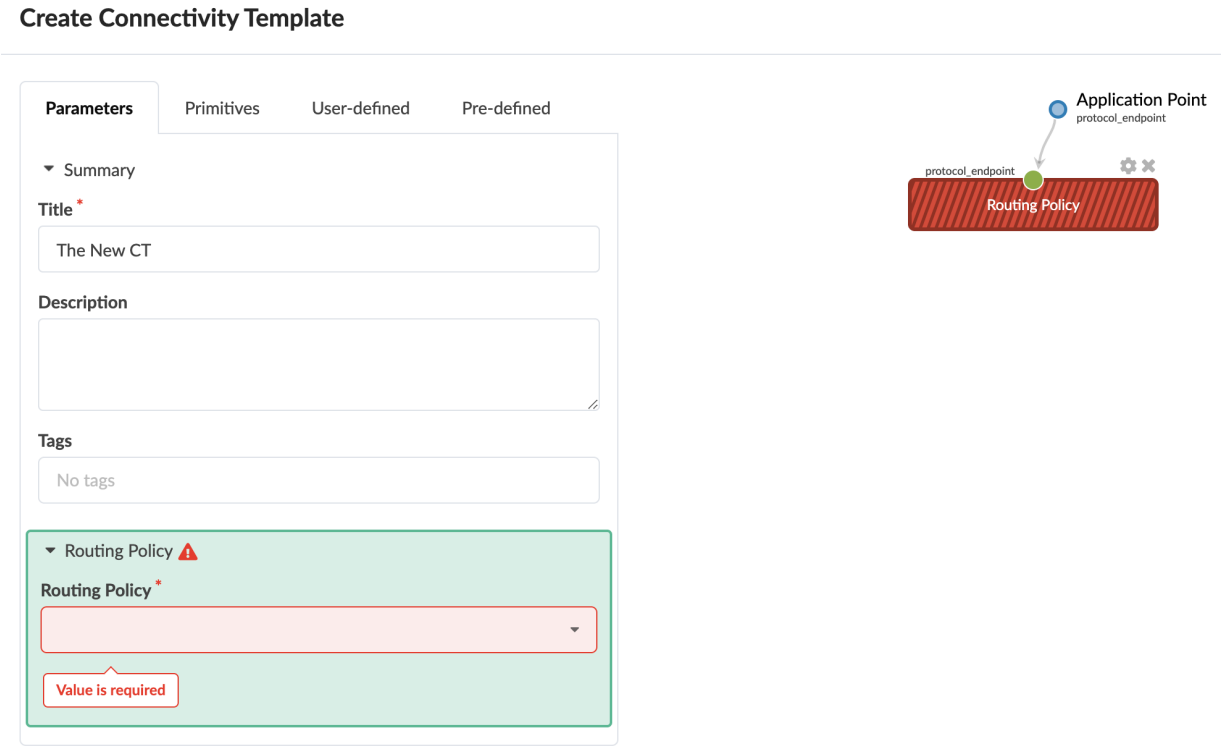
- IPv6 subnet for BGP prefix dynamic neighbors. If you leave this field blank, Apstra derives the subnet from the application point.



Primitive: Routing Policy

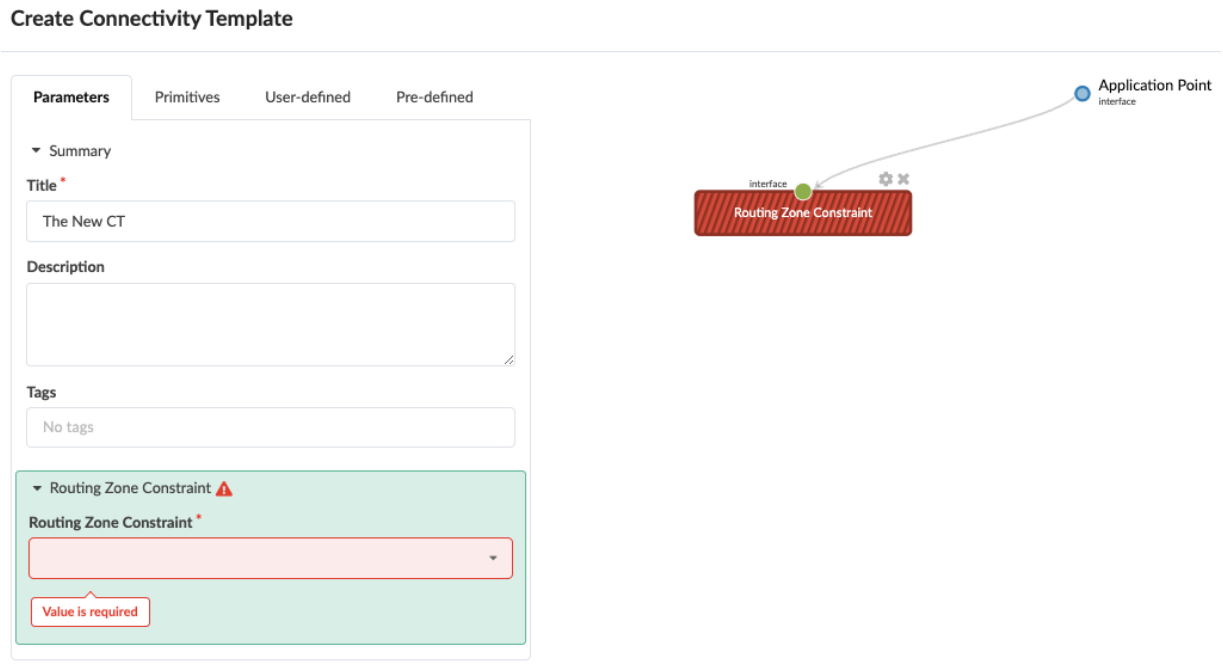
The routing policy primitive applies a routing policy to an application endpoint. This overrides the routing policy configured for the routing zone. You must select the routing policy that was defined in the

blueprint (Staged > Policies > Routing Policies).



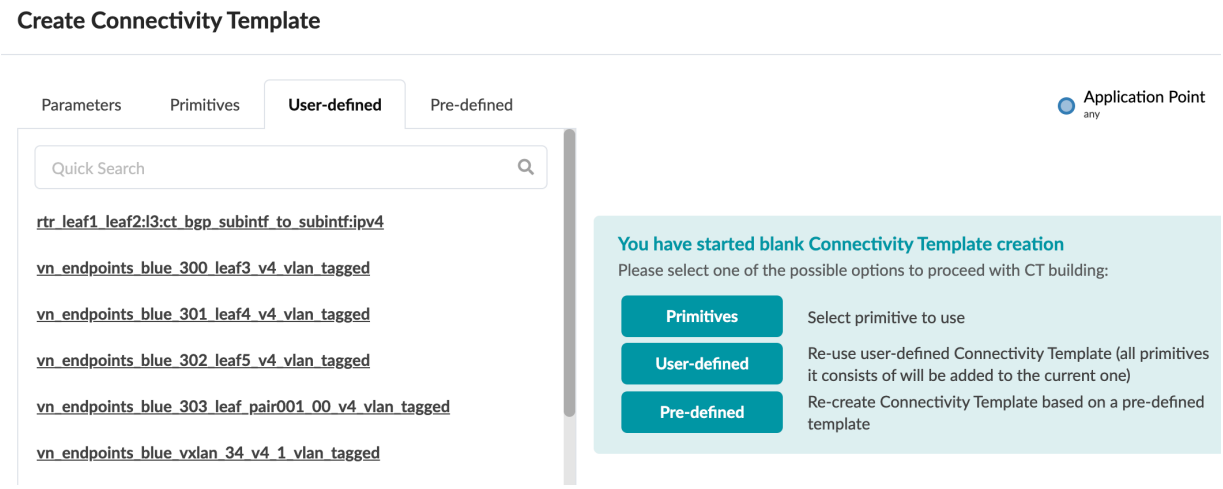
Primitive: Routing Zone Constraint

When you want to apply the routing zone constraint to an application point, add the Routing Zone Constraint primitive to the connectivity template and specify the routing zone or routing zone group.



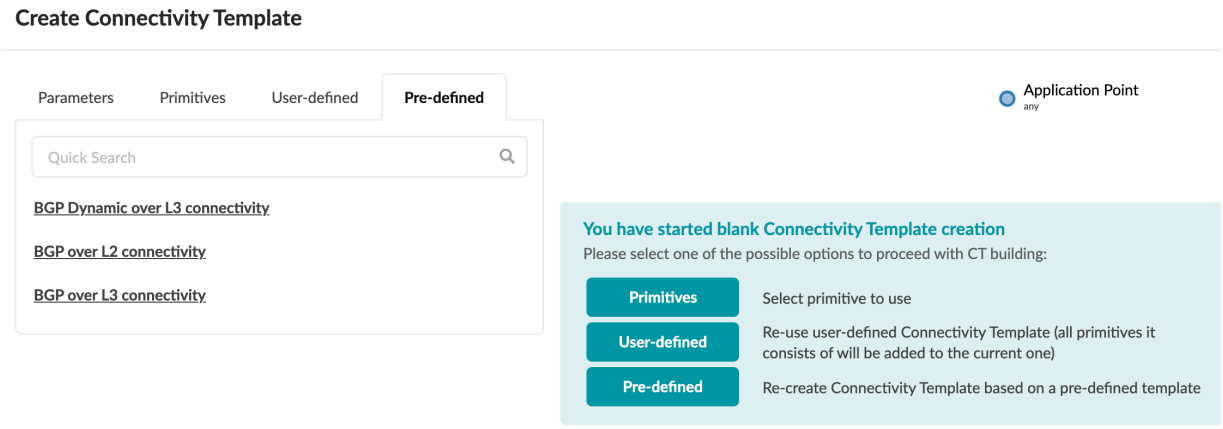
User-defined

From the **User-defined** tab, you can add grouped primitives that you previously created as connectivity templates.



Pre-defined

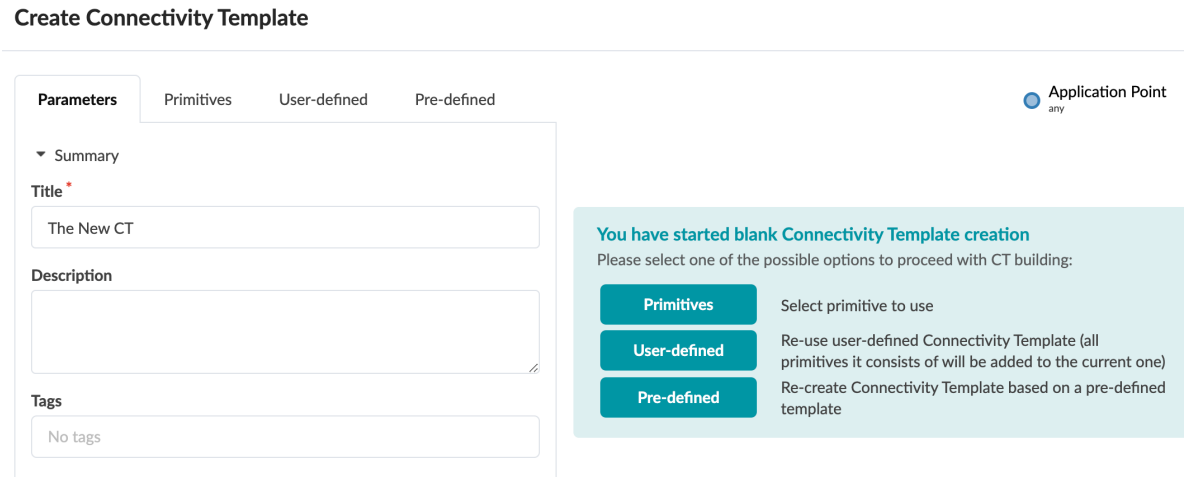
From the **Pre-defined** tab, you can add grouped primitives that ship with the Apstra software.



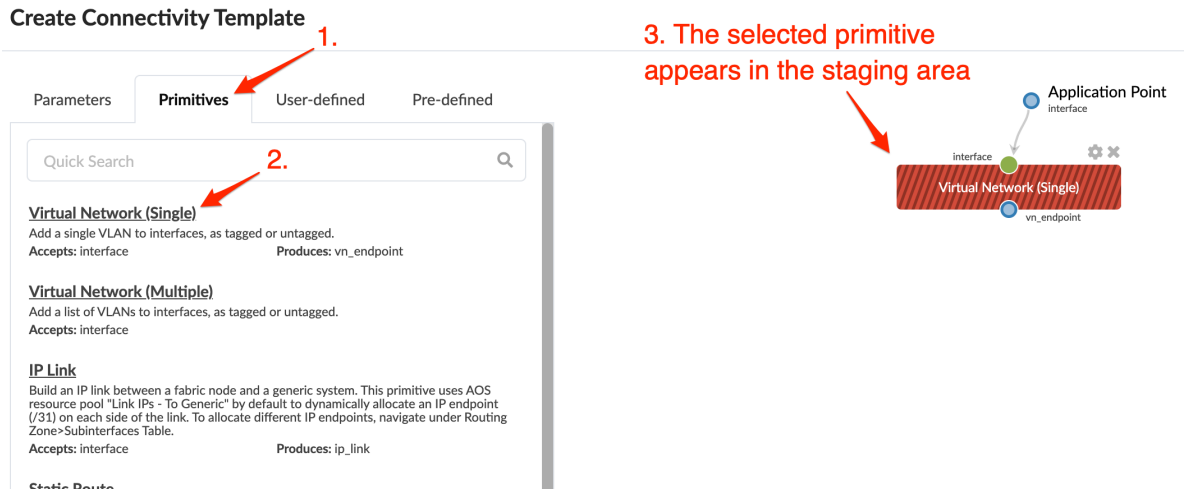
Create Connectivity Template for Multiple VNs on Same Interface (Example)

To create connectivity templates you add primitives (either singly or in groups) to a staging area, then you configure the parameters of those primitives. You can include up to 64 primitives in each connectivity template (increased from 18 as of Apstra version 4.0.1). We'll use examples to illustrate the process. First we'll show you how to create multiple virtual networks for the same interface.

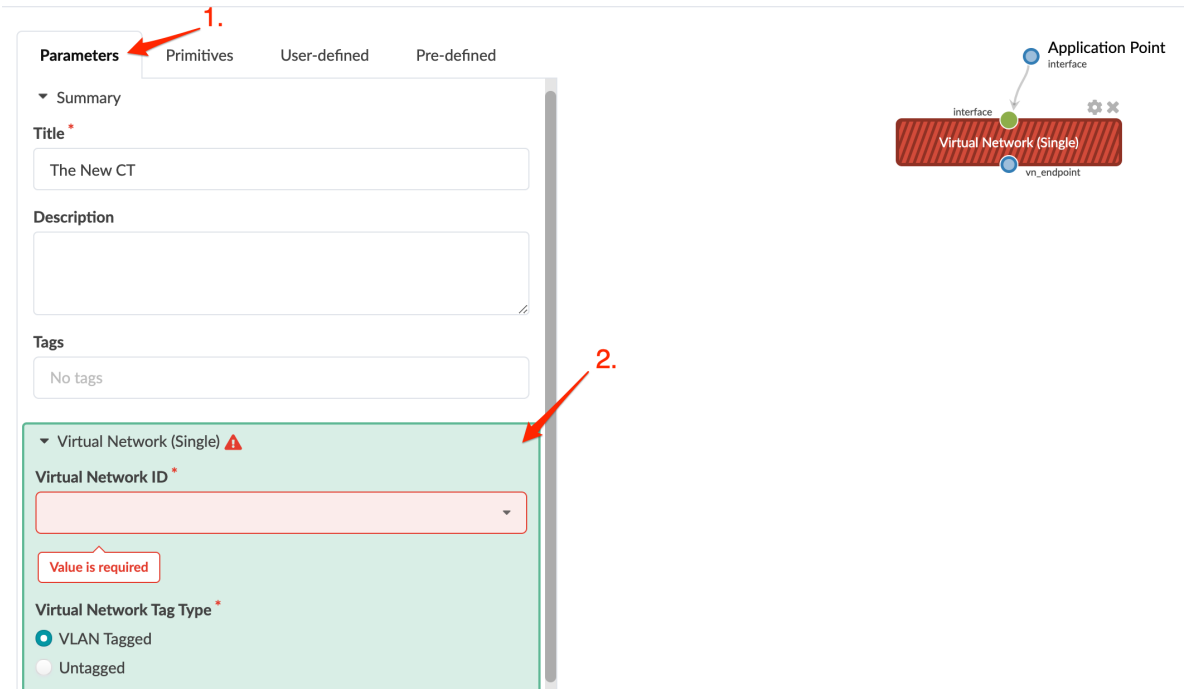
1. From the blueprint, navigate to **Staged > Connectivity Templates** and click **Add Template**. The staging area on the right contains the application point.



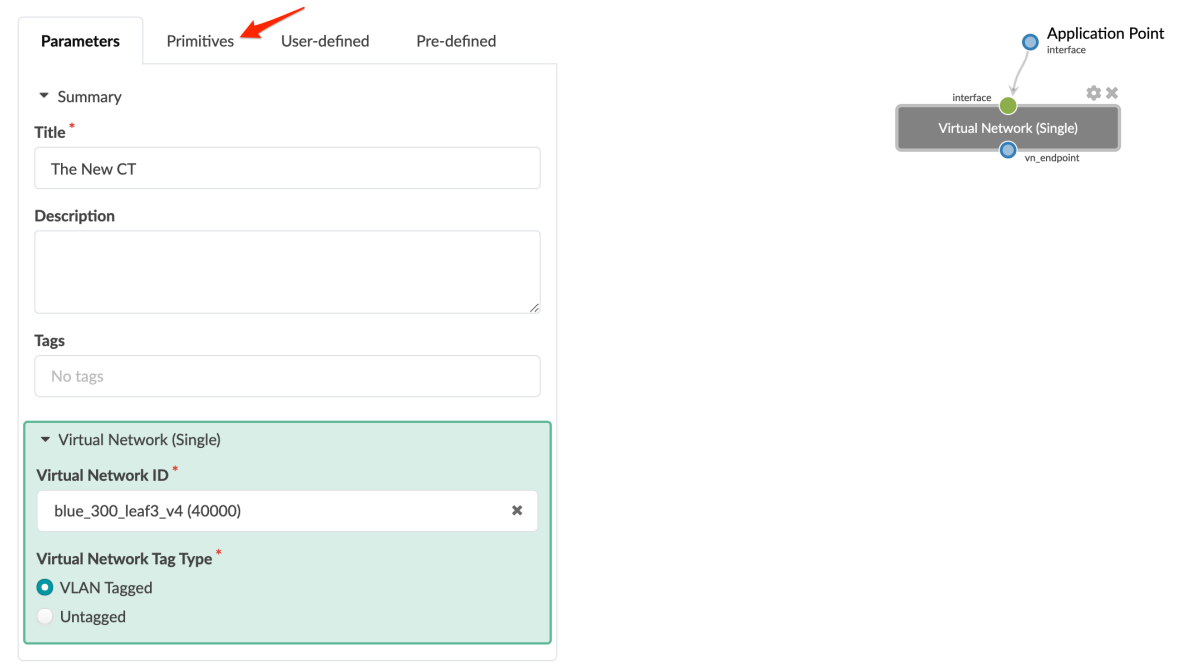
2. In the **Parameters** tab, enter a connectivity name in the **Title** field. You can optionally enter a description, and tags that you can use during subsequent searches.
3. The tabs **Primitives**, **User-defined**, and **Pre-defined** all contain primitives either singly or in groups. They are described in more detail in the overview. For this example, we'll add primitives one at a time from the **Primitives** tab. Click the **Primitives** tab, then click **Virtual Network (Single)**. It's added to the staging area, and it's connected to the application point.



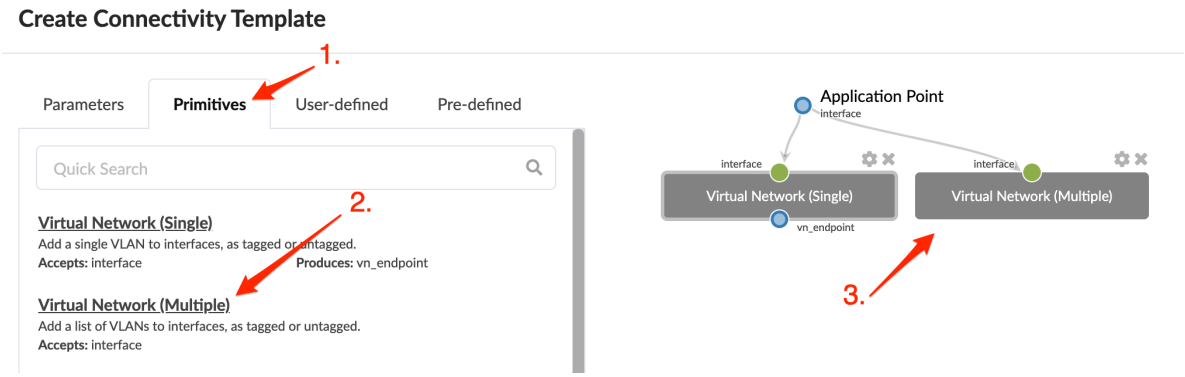
4. Click the **Parameters** tab to see what you need to configure for that primitive. In this example, you need to select a virtual network and specify whether it is VLAN tagged or untagged.



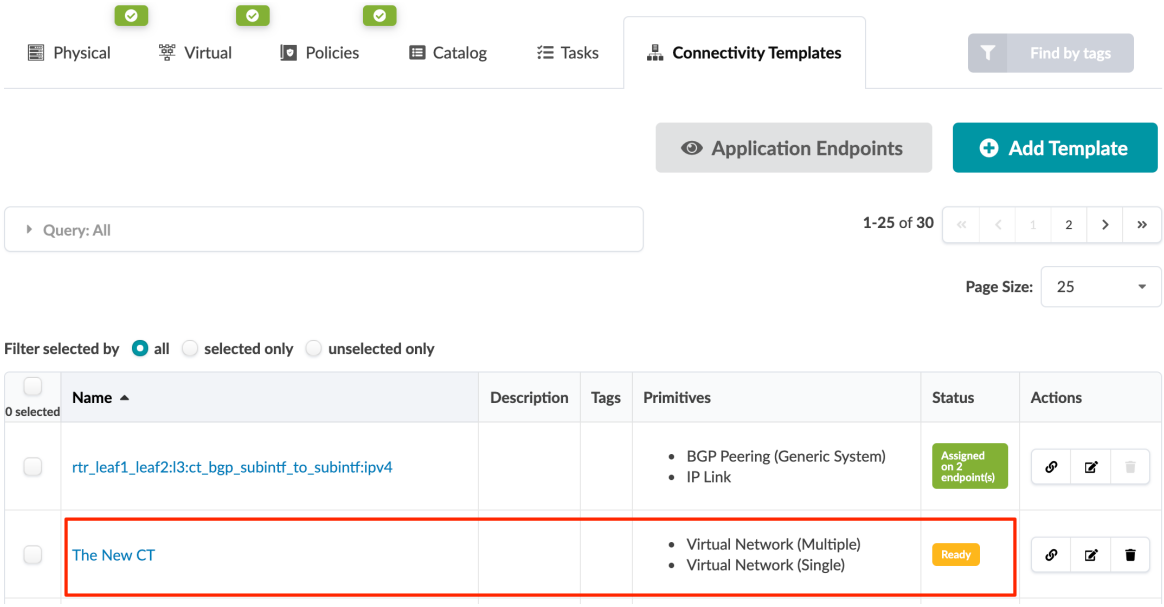
5. When it's successfully configured, the color of the selected primitive changes from red to gray. Click the **Primitives** tab.



6. From the **Primitives** tab, click **Virtual Network (Multiple)**.



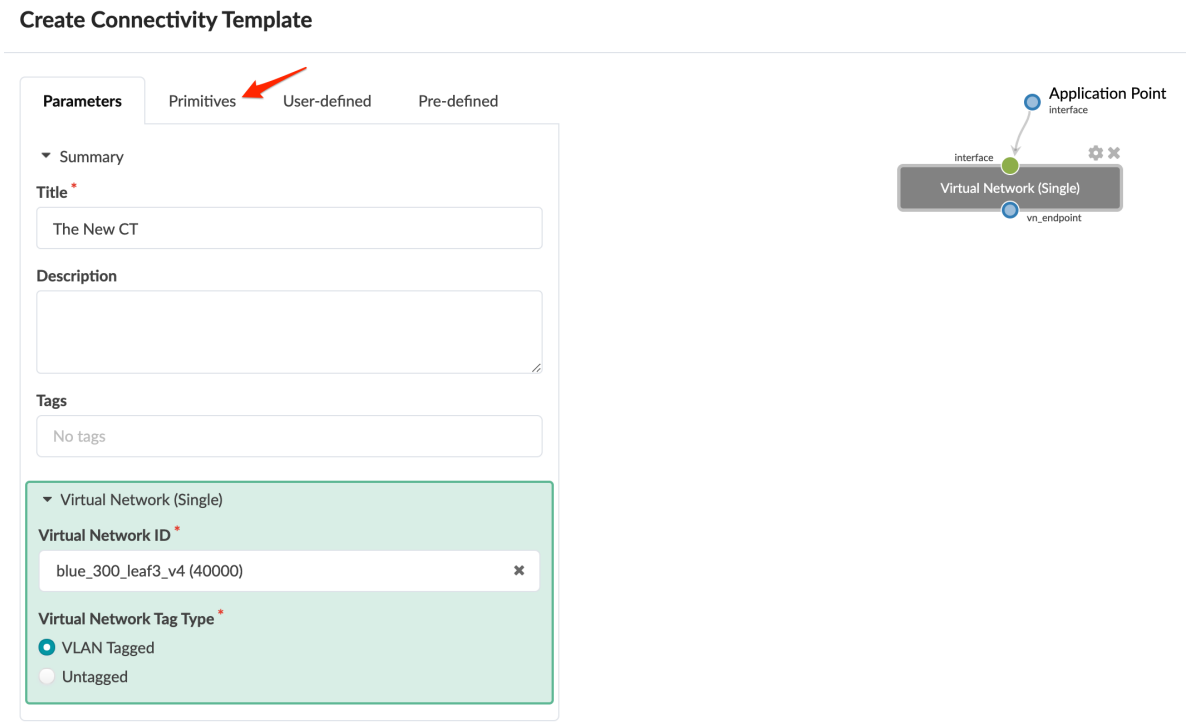
7. In the staging area, click **Virtual Network (Multiple)** (to make sure it's selected), click the **Parameters** tab and configure the primitive.
8. Click **Create** to create the connectivity template and return to the table view where you'll see your newly created connectivity template.



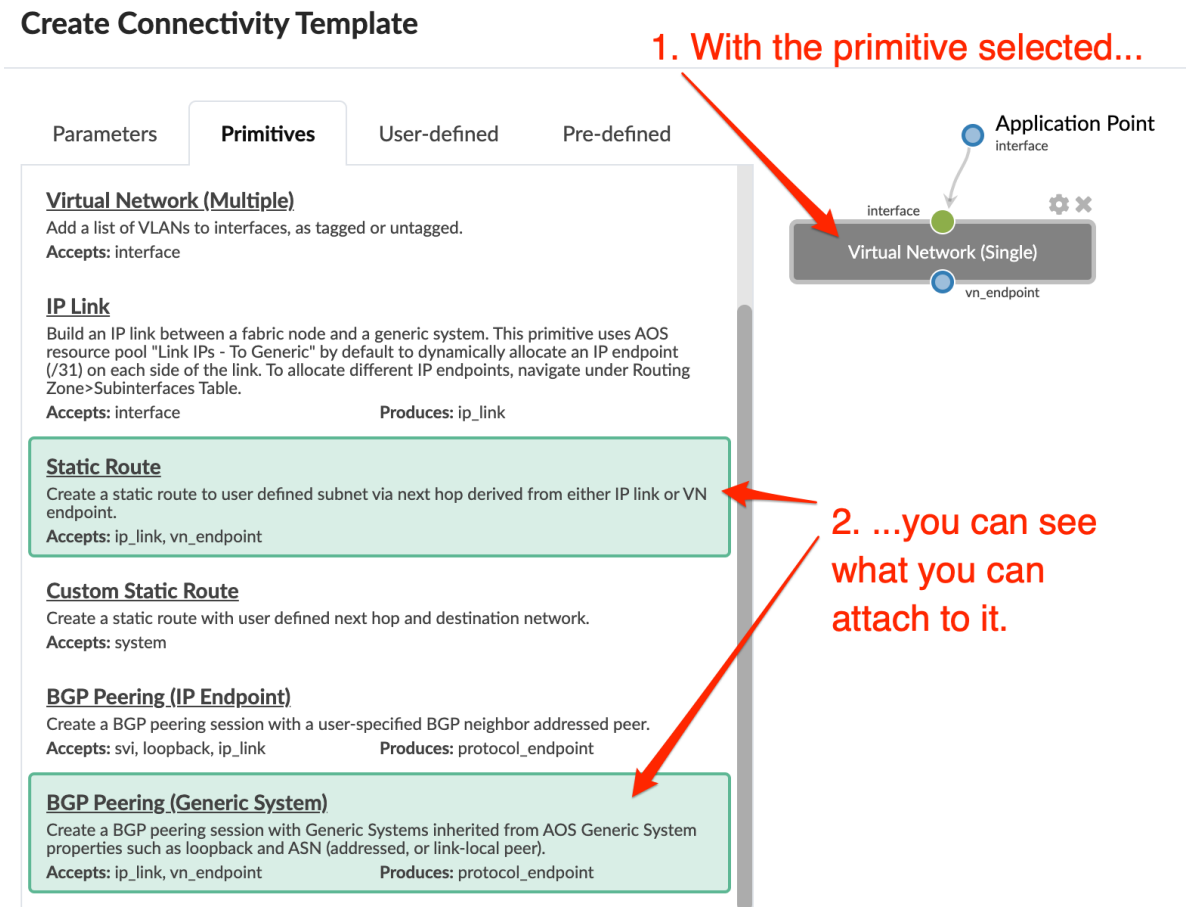
Create Connectivity Template for Layer 2 Connected External Router (Example)

In addition to applying multiple primitives to the application point interface, you can connect compatible primitives to each other. For example, let's configure a Layer 2 connected external router.

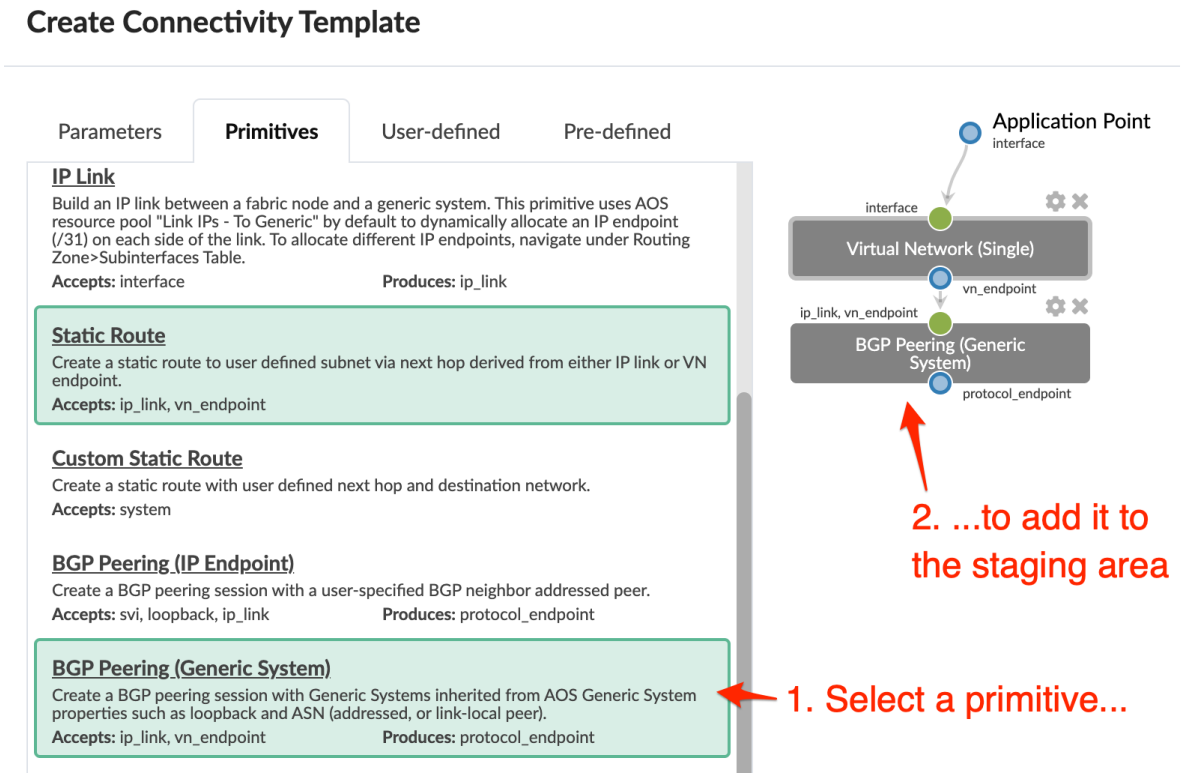
1. From the **Create Connectivity Template** dialog, click **Primitives**, click **Virtual Network (Single)**, and configure it on the **Parameters** tab (similar to the first example).



2. Click **Primitives**. When a primitive is selected, the other primitives that you can add to it are highlighted (new in Apstra version 4.0).



3. With **Virtual Network (Single)** selected in the staging area, click **BGP Peering (Generic System)** to add it to the staging area and connect it to the virtual network.



4. Proceed with configuring the parameters and click **Create** to create the template.

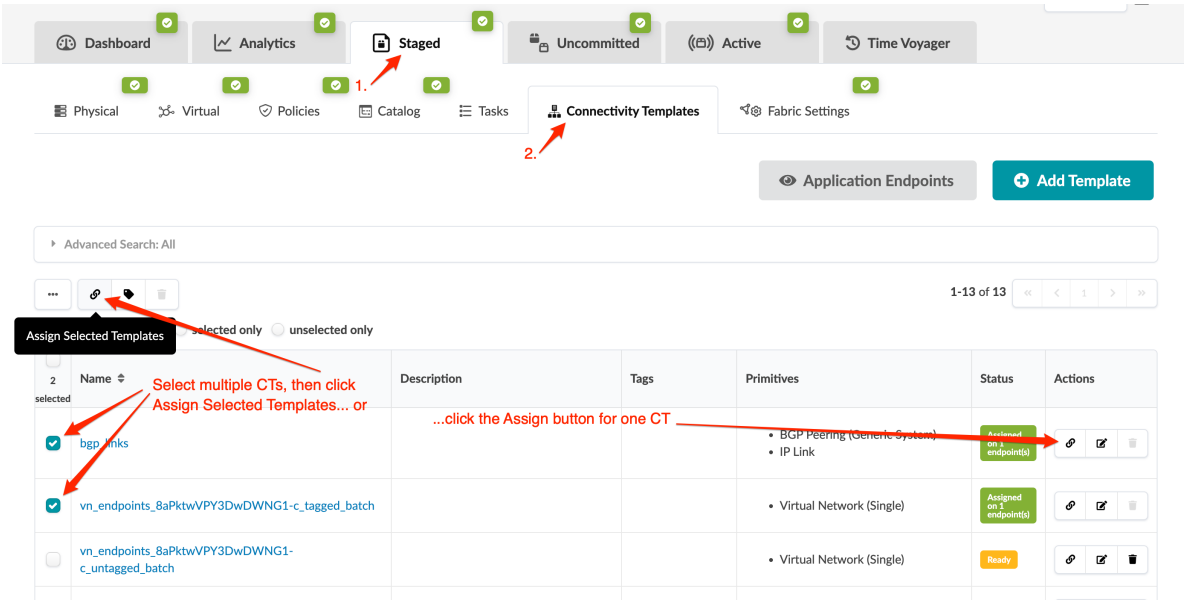
Update Connectivity Template Assignments

IN THIS SECTION

You can assign connectivity templates that have an active **Assign** button. These include connectivity templates in the **Ready** or **Assigned** status. (**Incomplete** status means that more configuration is required.) You can assign connectivity points directly from connectivity template(s) or from application endpoints.

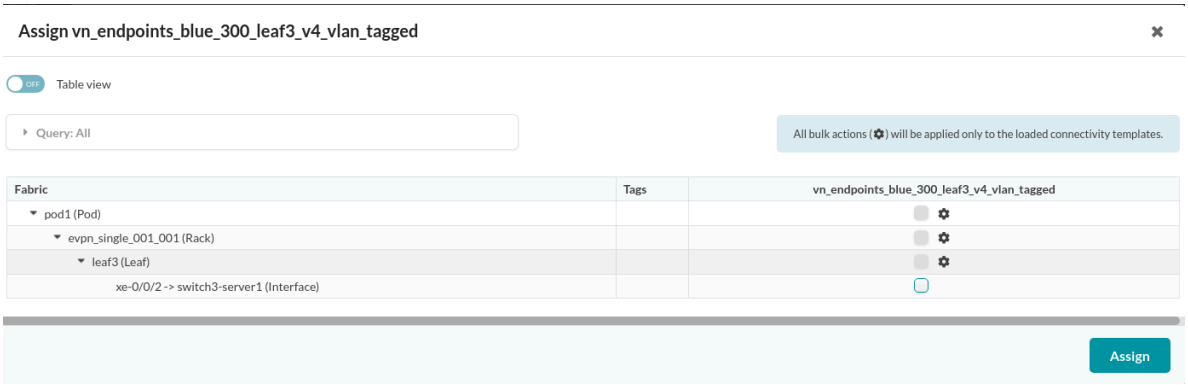
From Connectivity Templates

1. From the blueprint, navigate to **Staged > Connectivity Templates** and click the **Assign** button (in the **Actions** section on the right) for the connectivity template to assign, or select multiple connectivity templates, to the left of the CT name, then click the **Assign** button that appears above the table.



   The available fabric application endpoints appears in the dialog that opens.

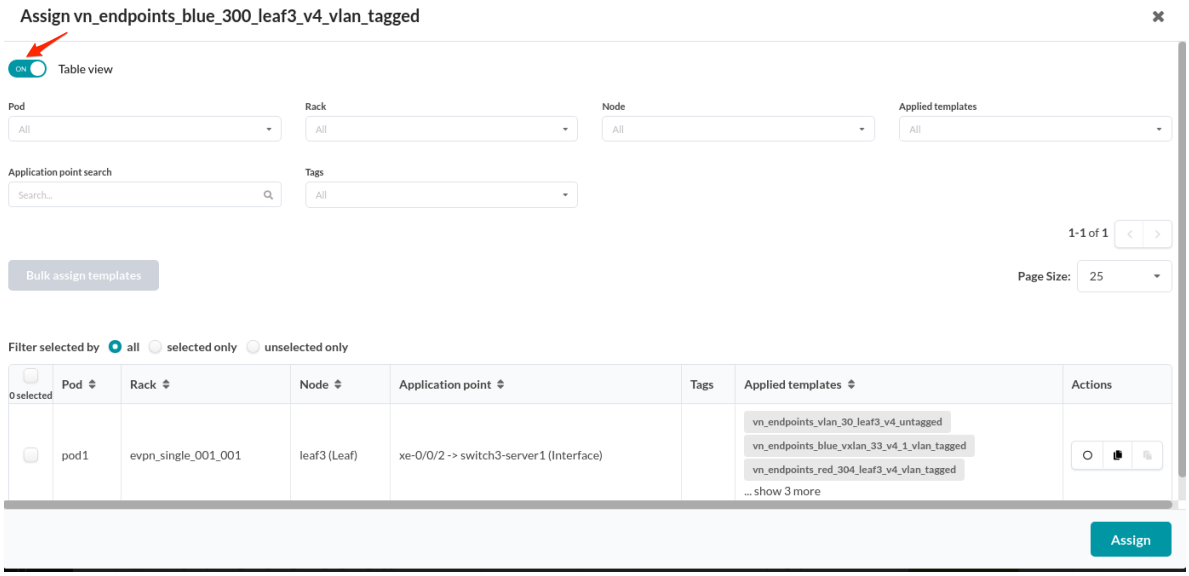
2. Click boxes on the connectivity template column to assign the connectivity template to the application endpoint. The **Tags** column shows the tags that are applied to each available application point. You can click the **Query** dialog to search by tags or labels.



   You can use "bulk actions" to select multiple "children" application endpoints.

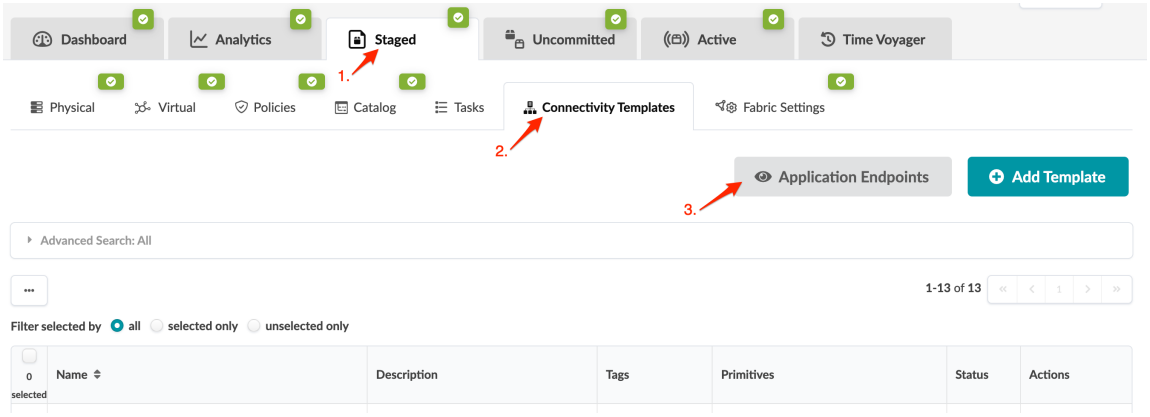
3. Click **Assign** to complete the connectivity template assignments.
4. You can view application endpoints in **Table view**. From the table view, you can filter application endpoints by pod, rack, node, applied connectivity templates, or tags. You can also copy/paste

connectivity template assignments from the table view.

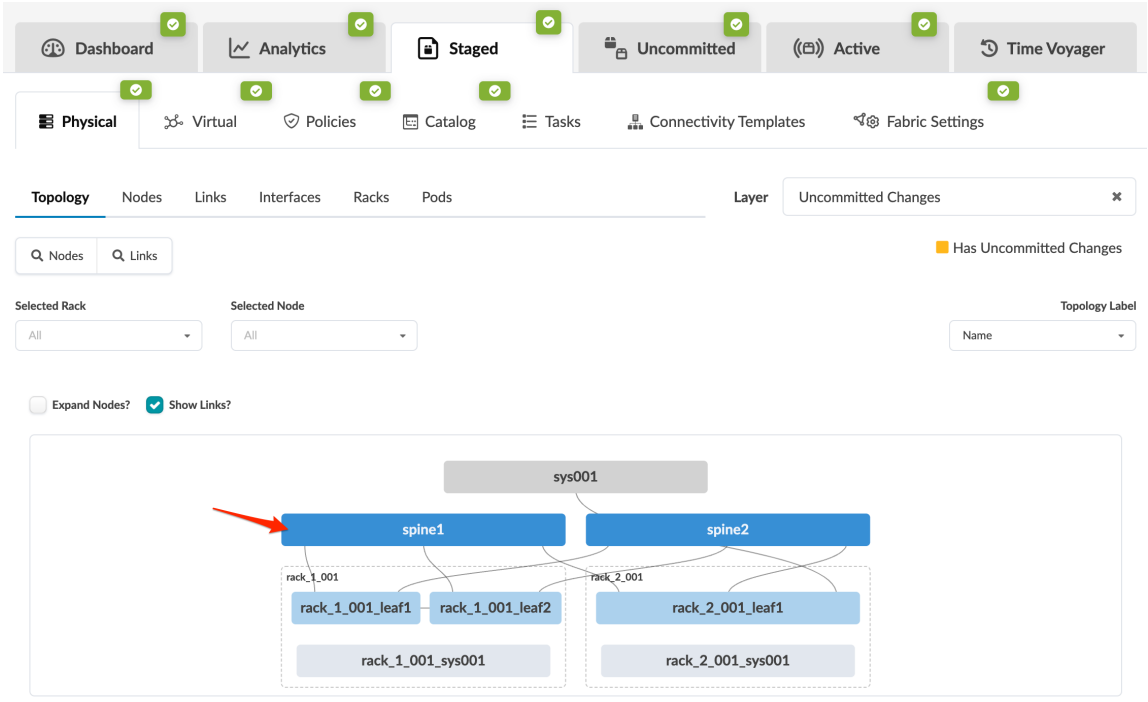


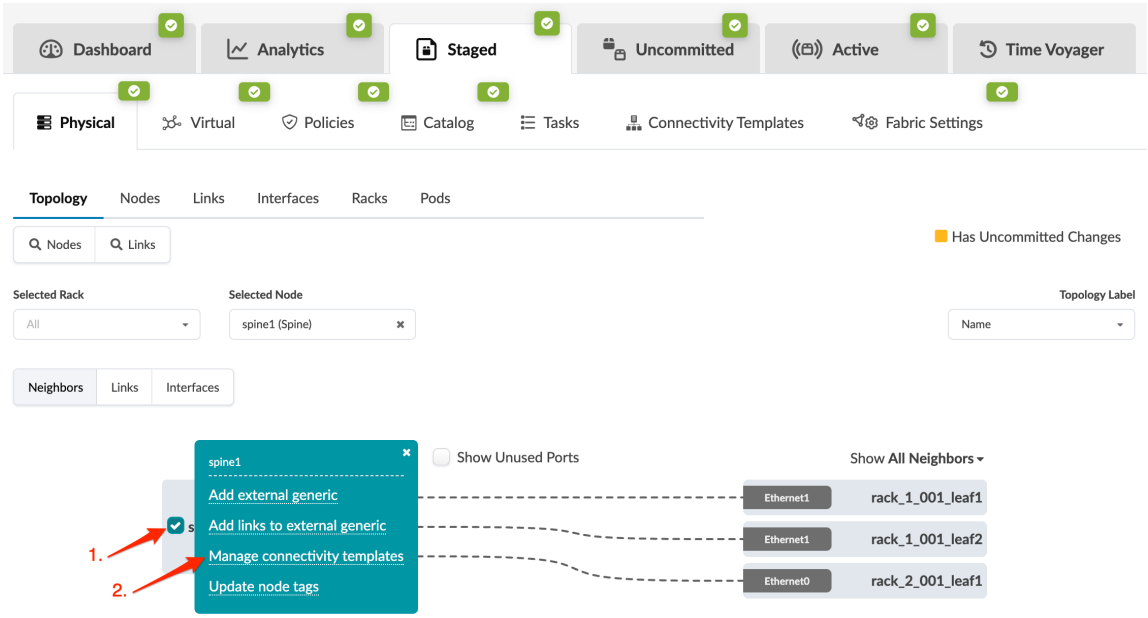
From Application Endpoints

1. You can access the application endpoints dialog from multiple places:
   - From the blueprint, navigate to **Staged > Connectivity Templates** and click **Application Endpoints**.

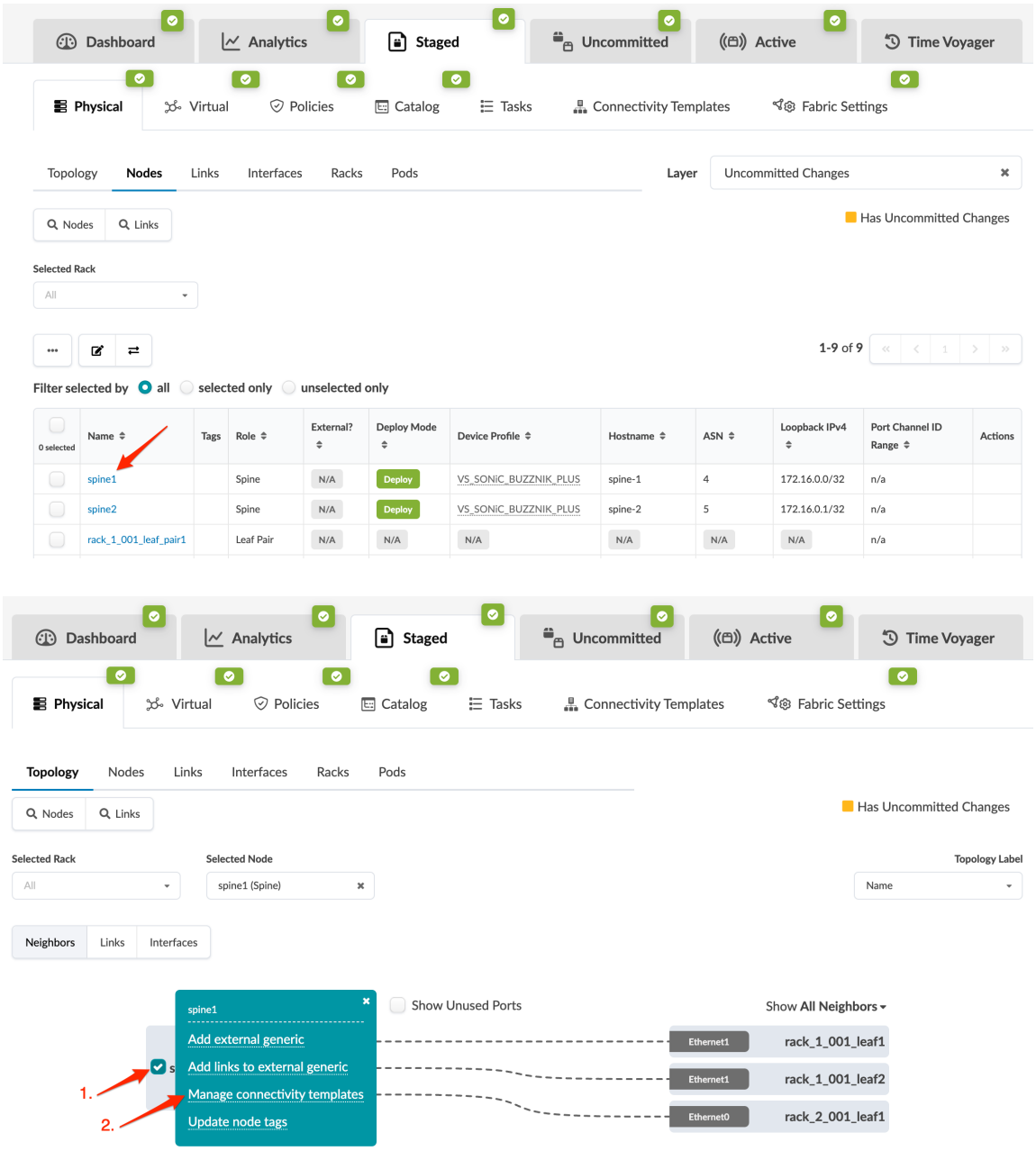


   - From the blueprint, navigate to **Staged > Physical >Topology**, click a node, click the check box for the node, then click **Manage Connectivity Templates**

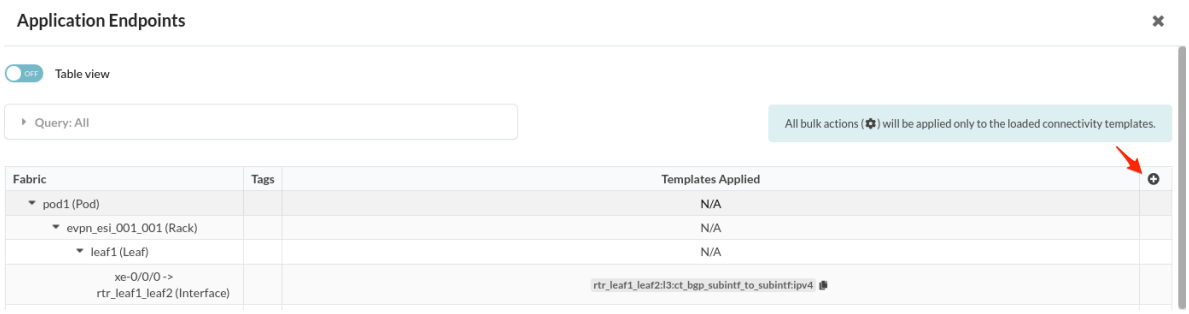




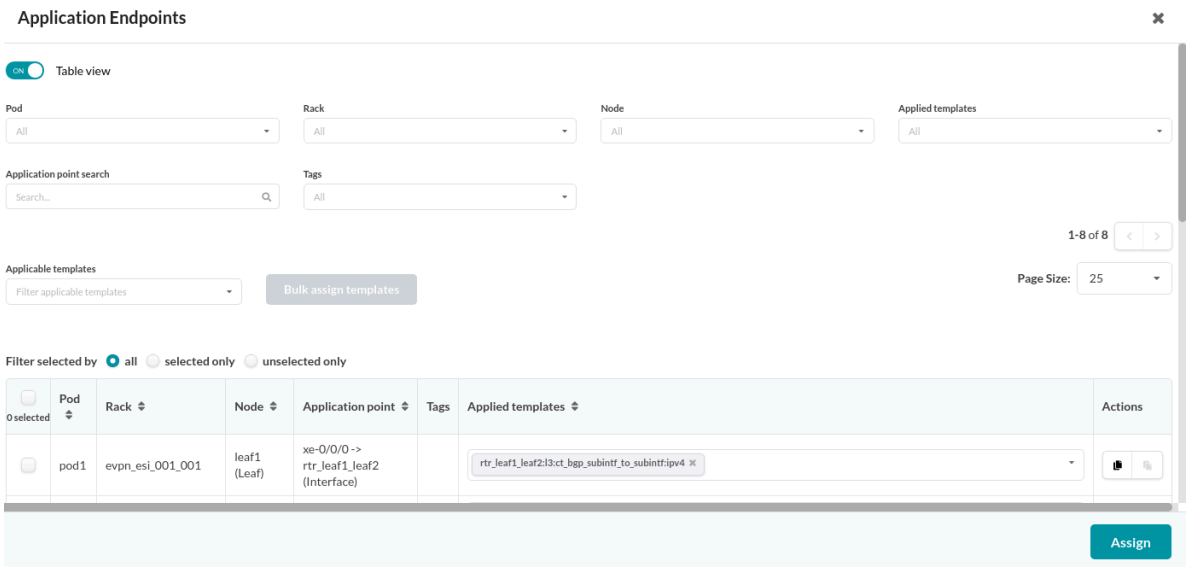
- From the blueprint, navigate to **Staged > Physical > Nodes**, click the name of a node, click the check box for the node, then click **Manage Connectivity Templates**.



2. You can click the + button to add a column for multiple connectivity templates.



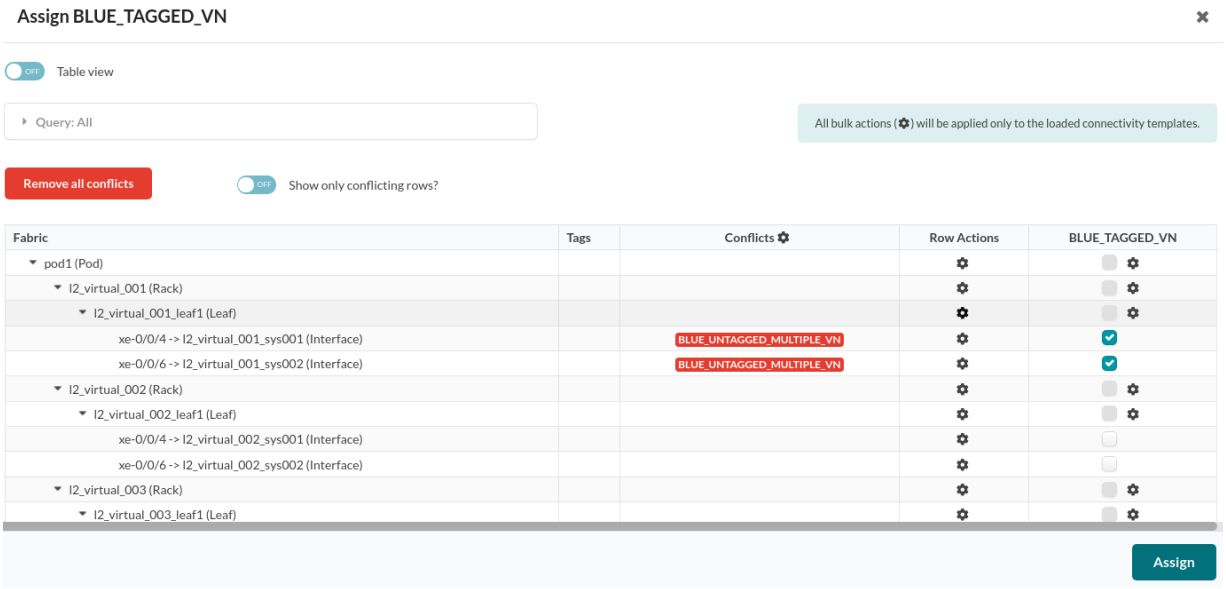
3. You can then query and select the desired assignment combination of connectivity templates and application endpoints.
4. After a connectivity template is applied, its configuration may require additional resources in the blueprint. For example, if you're adding Layer 3 links to connect a generic system (such as an external router), you must assign **Generic Link IPs**.
5. You can view as a **Table view**. From the table view, you can filter application endpoints by pod, rack, node, applied connectivity templates, or tags. You can also copy/paste connectivity template assignments from the table view.



Force Assign VN Templates

When a virtual network (single) or virtual network (multiple) template is already assigned to a port and you want to assign a new VN template, you'll receive a validation error indicating that the port already has a VN template assigned to it. As of Apstra version 4.0.1 you can force assign the new VN template, which automatically unassigns the existing VN template(s) and assigns the new one(s) on the selected port(s). You don't need to manually unassign the existing VN template.

To force assign VN templates, from the CT assignment screen, click **Remove all conflicts**, then click **Assign**.



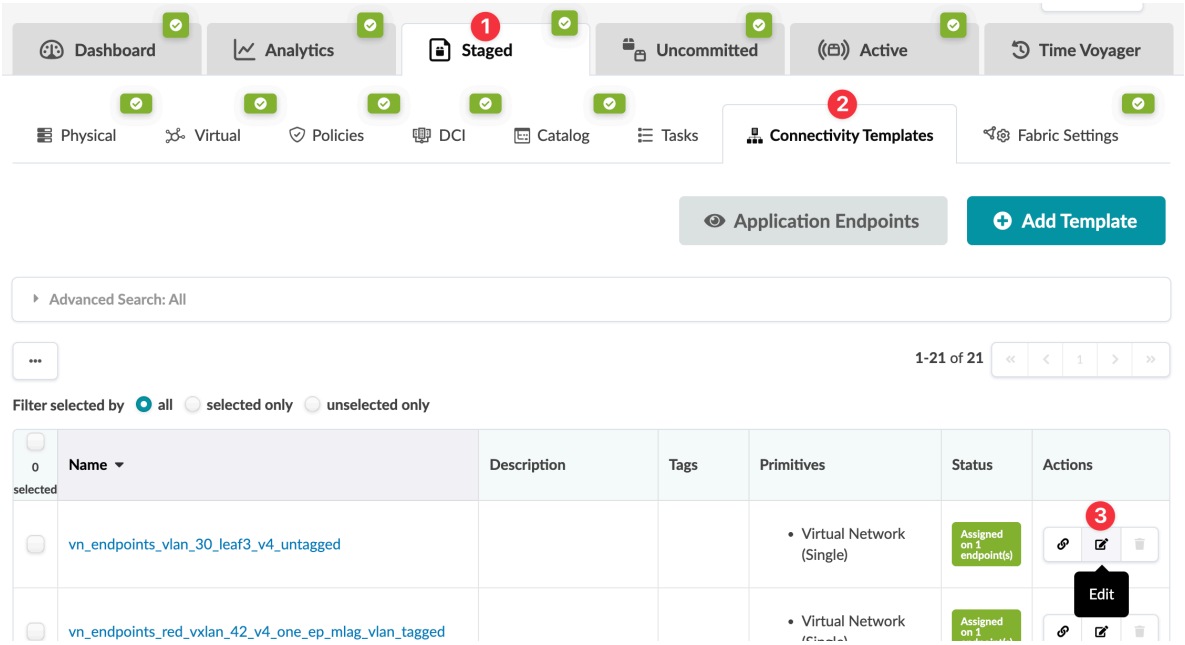
Add / Remove Tags on Connectivity Template

SUMMARY

IN THIS SECTION

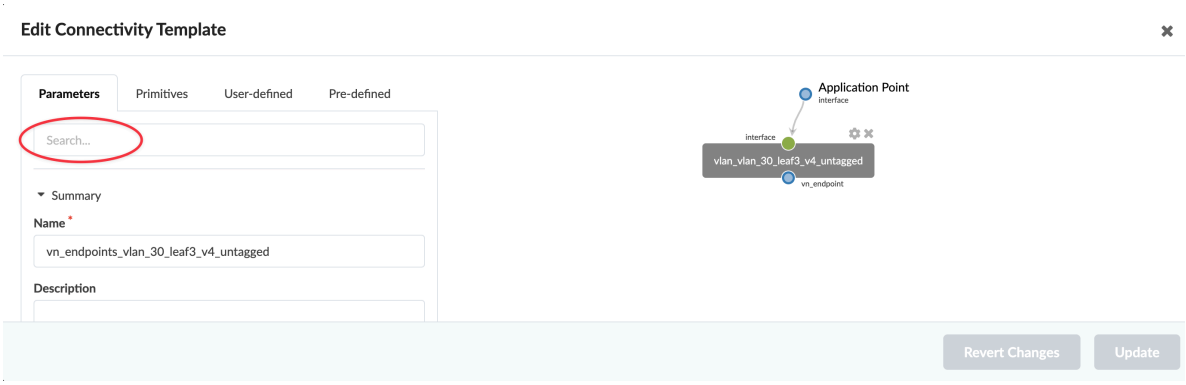
Update Connectivity Template

1. Either from the table view (Staged > Connectivity Templates) or the details view, click the **Edit** button for the connectivity template to update.



   The **Edit Connectivity Template** dialog opens.

2. Make your changes. The **Parameters** tab includes a search function (new in Apstra version 5.0.0) that you can use to search for and highlight primitives based on their parameters, for example a given IP address, routing zone, or virtual network.



3. Click **Update** to update the connectivity template and return to the table view. (If you decide not to change the connectivity template, click **Revert Changes** to discard your changes.)

Delete Connectivity Template

You cannot delete connectivity templates that have been assigned.

1. Either from the table view (Staged > Connectivity Templates) or the details view, click the **Delete** button for the connectivity template to delete.
2. Click **Delete** to delete the connectivity template and return to the table view.

CHAPTER 13

Fabric Settings

IN THIS CHAPTER

Fabric Policy

IN THIS SECTION

Update Fabric MTU

You can update the fabric-wide MTU setting.

1. From the blueprint, navigate to **Staged > Fabric Settings > Fabric Policy** and click **Modify Settings**.
2. Enter the new fabric MTU in the Fabric MTU field.
3. Click **Save Changes** to save your changes and return to the **Fabric Policy** page.

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

NOTE:

- To update the global default settings for SVIs and IP link MTUs, update the **Default IP Links to Generic Systems MTU** and/or the **Default SVI L3 MTU** fields in the Virtual Network Policy.

- To update the MTU on a specific SVI, update the **L3 MTU** field in the corresponding virtual network.
- To update the MTU on subinterfaces, update the **L3 MTU** field in the **IP Link** connectivity template primitive.

RELATED DOCUMENTATION

No Link Title

Change Virtual Network Details | 275

Primitive: IP Link | 460

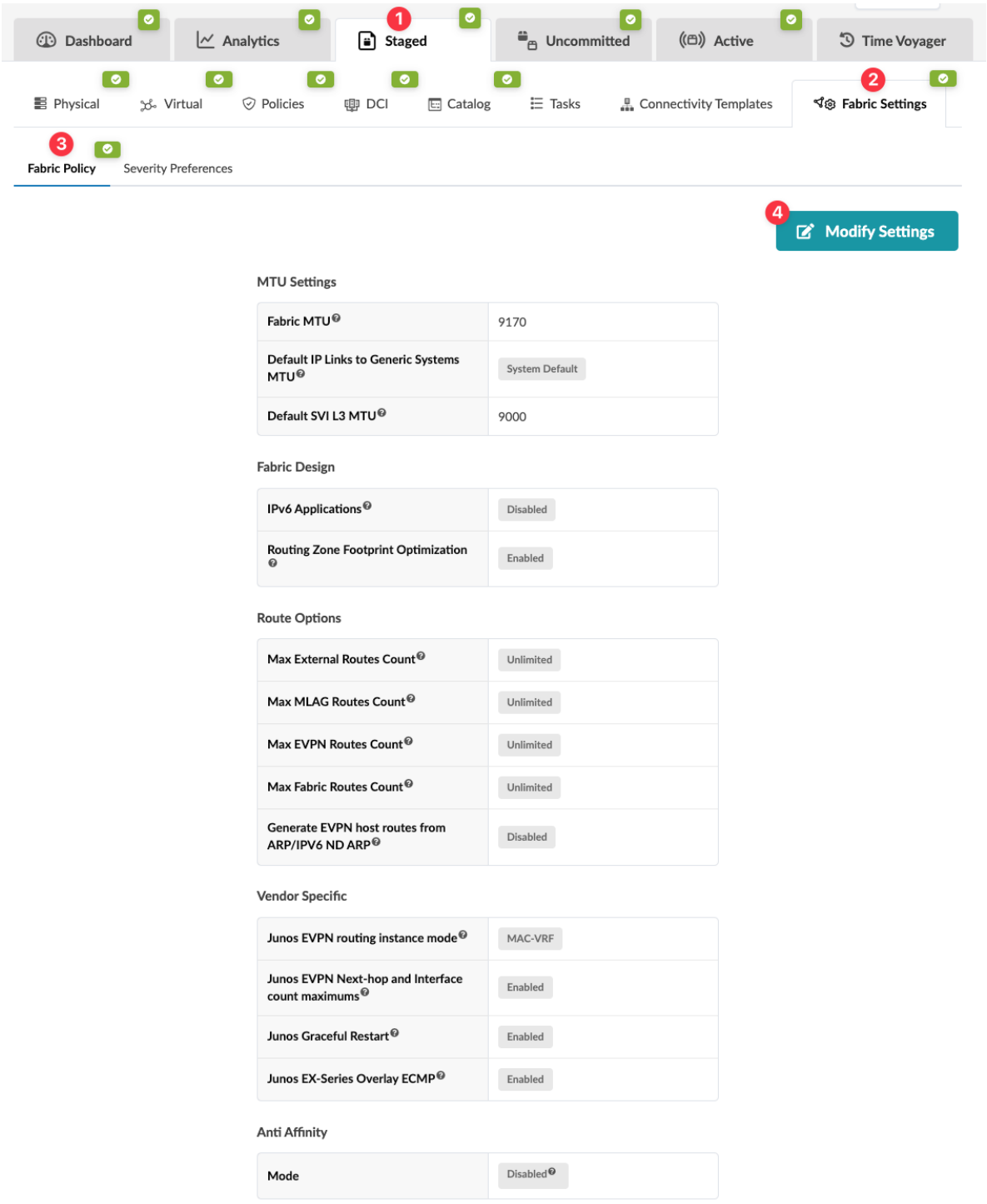
Enable IPv6 Applications

CAUTION: After IPv6 has been enabled in a blueprint, it cannot be disabled. Although, you could use **Time Voyager** to rollback to a revision before IPv6 was enabled.

Enabling support for IPv6 virtual networks on EVPN L2 deployments or L3 deployments adds resource requirements and device configurations. This includes IPv6 loopback addresses on leaf devices and spine devices, IPv6 addresses for MLAG SVI subnets and IPv6 addresses for leaf L3 peer links. The following caveats apply:

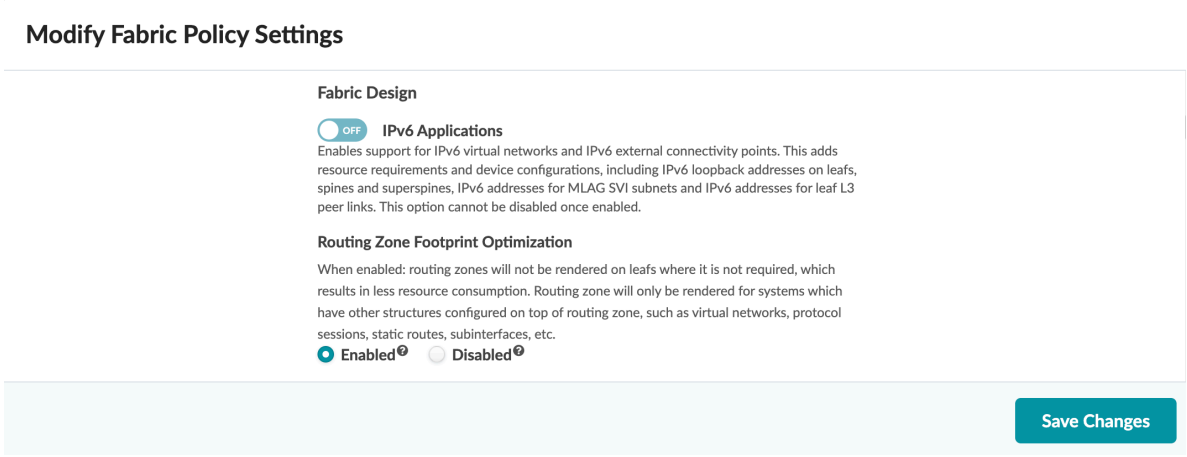
- This feature does not include IPv6 support in the fabric.
- IPv6 support is not available on non-EVPN L2 networks.
- When IPv6 is enabled on EVPN L2 deployments, security policy functionality is not available.

1. From the blueprint, navigate to **Staged > Fabric Settings > Fabric Policy** and click **Modify Settings**.



The **Modify Fabric Policy Settings** dialog appears.

2. In the **Fabric Design** section, click the toggle **ON** to enable IPv6 applications.



3. Click **Save Changes**.

Next Steps:

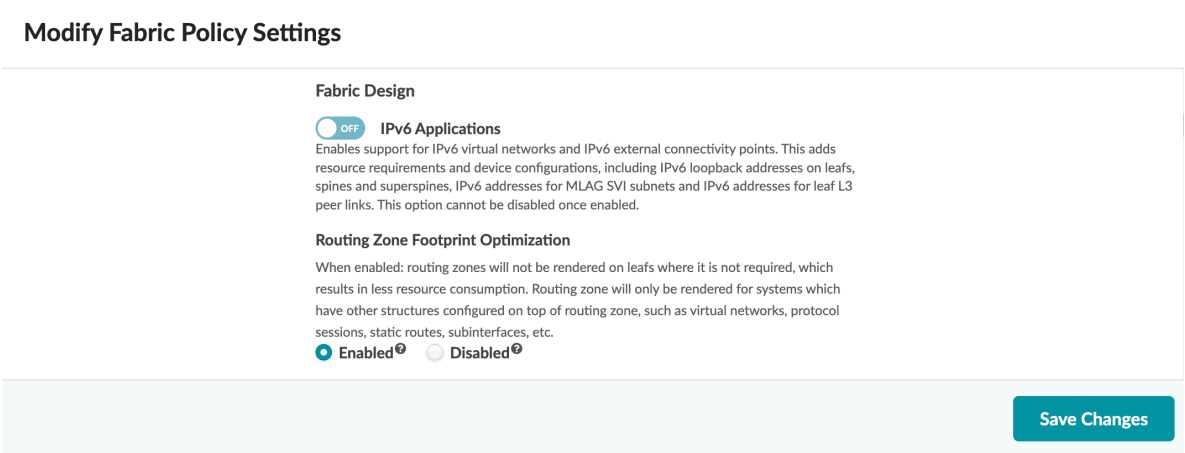
"Assign the required IPv6 IP addresses" on page 58. For more information about IPv6 configuration, see "Virtual Networks" on page 251.

Optimize Routing Zone Resource Usage

In Apstra version 4.2.0 and later, resources used for routing zones are already optimized (enabled) by default. This means VRF configuration is rendered only on leafs where at least one server-endpoint is a member of a virtual network in that routing zone.

In Apstra versions earlier than 4.2.0, all routing zones required resources. When you upgrade an Apstra server from a pre-4.2.0 version to version 4.2.1 or later, optimization is disabled by default. Since enabling optimization is disruptive, you must manually enable it yourself in this case. (Remember, currently you can't upgrade to major releases, such as 4.2.0.)

1. From the blueprint, navigate to **Staged > Fabric Settings > Fabric Policy** and click **Modify Settings**.
2. In the **Modify Fabric Policy Settings** dialog, in the **Fabric Design** section, for **Routing Zone Footprint Optimization**, select **Disable** or **Enable**, as appropriate, then click **Save Changes**.



- **Disabled** - Resources are required for all routing zones (active and inactive).
- **Enabled** - Resources are required only on active routing zones (at least one server-endpoint is a member of a virtual network in that routing zone).

RELATED DOCUMENTATION

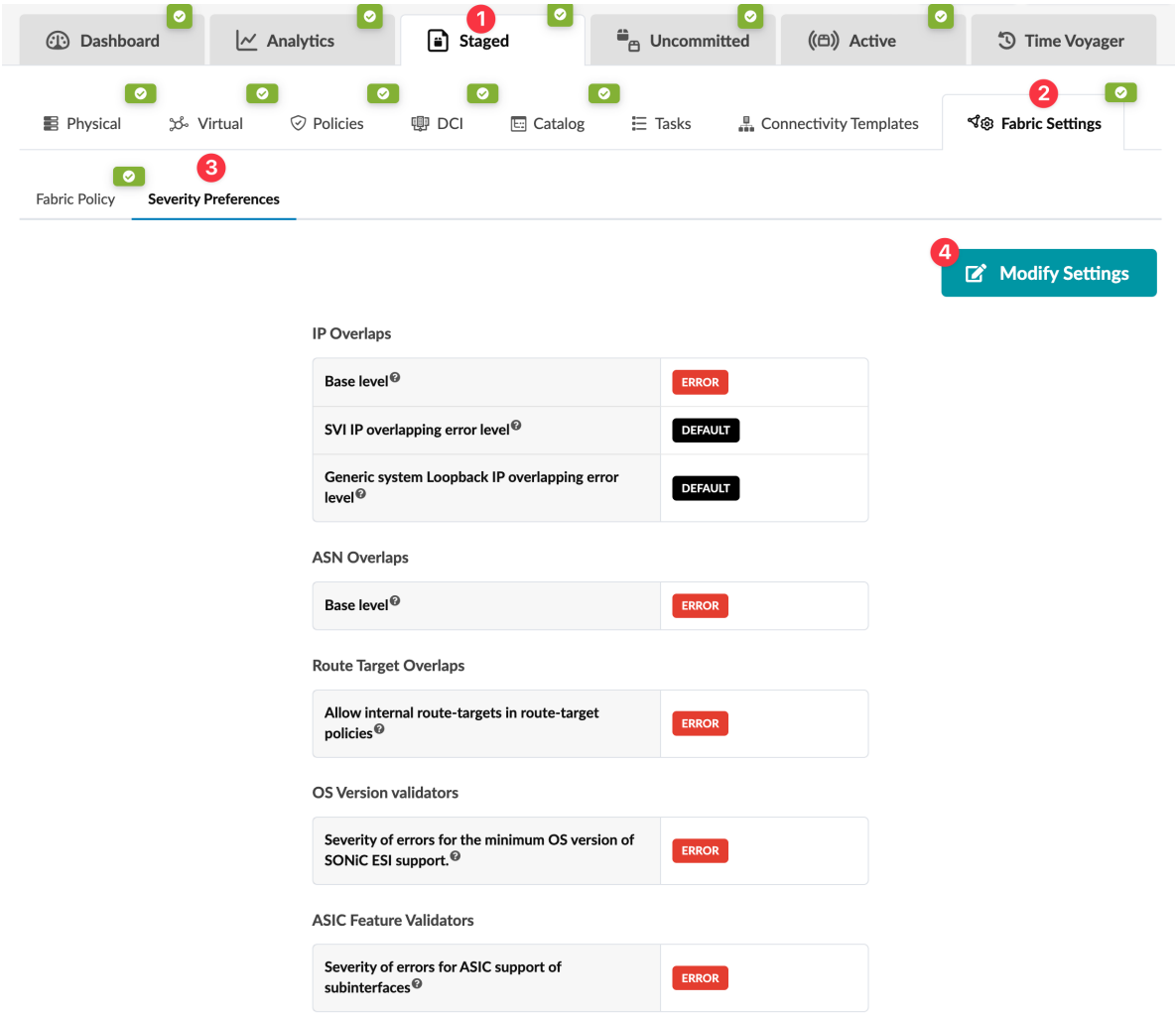
What are Routing Zones | 282

Severity Preferences

IN THIS SECTION

Update Severity Preferences

1. From the blueprint, navigate to **Staged > Fabric Settings > Severity Preferences** and click **Modify Settings**.



The **Modify Severity Preferences Settings** dialog opens.

2. Change settings, as applicable. See below for descriptions.

Modify Severity Preferences Settings

Experimental Features
Modifying the severity level of blueprint validation errors must be done with caution. Relaxation of each setting to levels lower than 'ERROR' carries its own risk. Please contact support for clarifications.

IP Overlaps

Base level
NO WARNING | WARNING | (•) ERROR
Severity of errors raised on overlap of IPs

SVI IP overlapping error level
NO WARNING | WARNING | ERROR | (•) DEFAULT
Defines the severity of the error which is raised when there are duplicated SVI IP addresses within a single virtual network. This setting could cause unexpected traffic flow for routed traffic.

Generic system Loopback IP overlapping error level
NO WARNING | WARNING | ERROR | (•) DEFAULT
Support for uncontrolled generic loopbacks IP overlap errors. This relaxes validation errors and allows IP addresses to be shared between external generic loopbacks and fabric objects both in default and EVPN routing zones. These fabric objects include loopback IPs, physical link IPs, logical link IPs, virtual network subnets and VTEP addresses.

ASN Overlaps

Base level
NO WARNING | WARNING | (•) ERROR
Severity of errors raised on overlap of ASNs

Route Target Overlaps

Allow internal route-targets in route-target policies
NO WARNING | WARNING | (•) ERROR
Severity of errors raised on overlap of a user-defined route-target which overlaps with an internal virtual network or routing zone route-target. This can be used for a form of full table inter-vrf route leaking.

OS Version validators

Severity of errors for the minimum OS version of SONiC ESI support.
NO WARNING | WARNING | (•) ERROR
Severity of errors related to ESI features on SONiC devices. A minimum OS version of 4.2.1 is required to support ESI racks on SONiC devices. This validation can be relaxed in case of an upstream vendor backport, or if the error is incorrectly raised.

ASIC Feature Validators

Severity of errors for ASIC support of subinterfaces
NO WARNING | WARNING | (•) ERROR
Severity of errors related to ASIC support of subinterfaces. This error can be relaxed if an OS vendor provides a software update to add subinterface support on that ASIC.

Save Changes

The parameters available for configuring are as follows:

- **IP Overlaps**
    - - **Base level** - The severity level raised when detecting an overlap of IP addresses. To set a more granular severity level based on the type of IP Overlaps, use the settings below.
        - **SVI IP overlapping error level** - The severity level raised when detecting duplicate SVI IP addresses within a single virtual network. When set to “Default“, the severity level from the IP Oveaqqrlaps “Base Level“ setting is used. Note that duplicate SVI IPs can cause unexpected traffic flow for routed traffic. We recommend leaving the severity level to “Error“ or “Default“ (when the “Error“ level is configured at the “Base Level“ setting).
        - **Generic system Loopback IP overlapping error levels** - The severity level raised when detecting a Generic System Loopback IP overlaps with another external or fabric node IP used in default or EVPN Routing Zones. This can be an IP address used for a loopback, physical link, logical link, virtual network subnet or VTEP interface. When set to “Default“, the severity level from the IP Overlaps “Base Level“ setting is used. Use this setting to relax validation errors and allows these types of overlaps.
- **ASN Overlaps**
    - **Base level** - The severity level raised when detecting an overlap of ASNs.
- **Route Target Overlaps**
    - **Allow internal route-target policies** - Severity of errors raised on overlap of a user-defined route-target which overlaps with an internal virtual network or routing zone route-target. This can be used for a form of full table inter-vrf route leaking.
        - **No Warning** - If validation fails, no warning or error is generated.
        - **Warning** - If validation fails, warnings are raised; you can commit changes.
        - **Error** - If validation fails, errors are raised that must be resolved before you can commit changes.
        - **Default** - Severity level defaults to the base level severity level.
- **OS Version Validators**
    - **Severity of errors for the minimum OS version of SONiC ESI support** - Severity of errors related to ESI features on SONiC devices. A minimum OS version of 4.2.1 is required to support ESI racks on SONiC devices. This validation can be relaxed in case of an upstream vendor backport, or if the error is incorrectly raised.
- **ASIC Feature Validators**

- **Severity of errors for ASIC support of subinterfaces** - Severity of errors related to ASIC support of subinterfaces. This error can be relaxed if an OS vendor provides a software update to add subinterface support on that ASIC.

3. Click **Save Changes** to stage the changes and return to the **Severity Preferences** page.

To deploy changes to the active blueprint, click the **Uncommitted** tab to review and commit (or discard) changes.

PART 5

Staged Freeform Blueprints

Freeform Introduction

IN THIS SECTION

Reference Designs

If your network architecture is comprised of a 3-stage Clos, 5-stage Clos or collapsed fabric, you'll want to take advantage of the abstraction and automation that's included with the **Datacenter** reference design. For all other topologies, you can use the **Freeform** reference design to leverage any feature, protocol, or architecture.

Blueprints created in the Datacenter reference design use a set of design elements to abstract and automate many network activities. Blueprints created in the Freeform reference design consist of *systems* and links that you add and configure yourself, giving you complete control over your architecture. In Freeform we use the term **system** to represent all the types of devices that can be linked in the Apstra environment: switches, routers, Linux hosts and so on.

Device Management

Device management for Freeform blueprints is the same as for Datacenter blueprints. The process of installing agents and acknowledging them to bring them under Apstra management is the same in both reference designs. Only Juniper devices are supported in Freeform blueprints.

Freeform Bluprints and Device Profiles

You can build your Freeform blueprint manually from an empty blueprint, or if you've exported an existing Freeform blueprint, you can use it as a template for a new one. You'll start building your empty blueprint by importing **device profiles** from the design (global) catalog. A device profile represents a device's capabilities without specifying its system ID (serial number). This is what enables you to build your entire network 'offline' before deploying it.

Systems and Links

You'll create **internal systems** and assign device profiles to them. Internal systems are devices that are managed in the Apstra environment. You can bring your devices under Apstra management at any time. If you have them ready, you can assign them as you're creating your internal systems. If they're not ready, that's OK. You can assign them any time before deploying your network.

External systems are the other type of system used in Freeform blueprints. These are systems that are linked to internal systems, and are not under Apstra management.

When you link your systems, you'll select ports and transformations, as applicable. You can also add IP addresses and *tags* as you're creating those links.

Config Templates, Property Sets and Tags

Config templates are text files used to configure internal systems in Freeform. You'll assign a config template to every internal system. You *could* paste configuration directly from your devices into a config template to create a static config template, but then you wouldn't be using the potential of config templates. With some Jinja2 knowledge (and maybe some Python), you can parametrize config templates to do powerful things.

Property sets provide a valuable capability to fully parameterize config templates. Consisting of key-value pairs, they enable you to separate static portions of config templates from variables. You create property sets in the blueprint catalog. (Property sets used in Freeform blueprints are not related to property sets in the design (global) catalog.) You'll include property set names in your config template and then the values in those property sets will be used when configuration is rendered.

You can also create a property set and assign it directly to one system.

Tags are a way for you to assign metadata to Apstra-managed resources. They can help you identify, organize, search for, and filter Apstra systems and links. With tags, you can categorize resources by purpose, owner, environment, or other criteria. Because tags are metadata, they aren't just used for

visual labeling; they are also applied as properties of nodes in the Apstra graph database. This node property (or device property) is then available for you to reference in Jinja config templates for dynamic variables in config generation and the Apstra real-time analytics via Apstra's Live Query technology and Apstra Intent-Based Analytics.

An example of when you might want to use tags is if you have bare metal servers with SRIOV interfaces, and you need to produce specific configuration for those interfaces. You would add the tag `sriov` to the links, then specify in the config template that links with that tag are to be configured a certain way.

Freeform Workflow

1. Access the" Apstra GUI" on page 4.
2. "Bring your devices under Apstra management" on page 651 (same procedure as for Datacenter blueprints). If you don't have your system IDs (serial numbers) yet, that's OK. You can build your entire network 'offline' in the Apstra environment and bring your devices under Apstra management any time before deploying your network.
3. "Create" on page 8 a Freeform blueprint.
   .
4. "Import device profiles" on page 583 for the internal systems you'll create.
5. "Add internal systems" on page 511 for the systems that Apstra will manage.
6. "Add external systems" on page 516 for unmanaged systems, as applicable.
7. "Add links" on page 553 to your systems.
8. "Create config templates" on page 578, and "property sets" on page 585 as needed.
9. "Assign config templates" on page 521 to internal systems with deploy mode set to **Deploy**.
10. If you haven't brought your "devices under Apstra management" on page 651 yet, it's time to do that now.
11. "Assign system IDs" on page 534 (if you haven't already) and set the deploy mode on your systems to **Deploy**.
12. Before deploying your network, you can use the apstra-cli utility to validate config template syntax. For more information, see Juniper Support Knowledge Base article KB69779.
13. Commit changes to deploy blueprint.

RELATED DOCUMENTATION

Export Freeform Blueprint | 503

Commit / Revert Changes to Blueprint | 599

CHAPTER 14

Blueprints

IN THIS CHAPTER

Freeform Blueprints Introduction

IN THIS SECTION

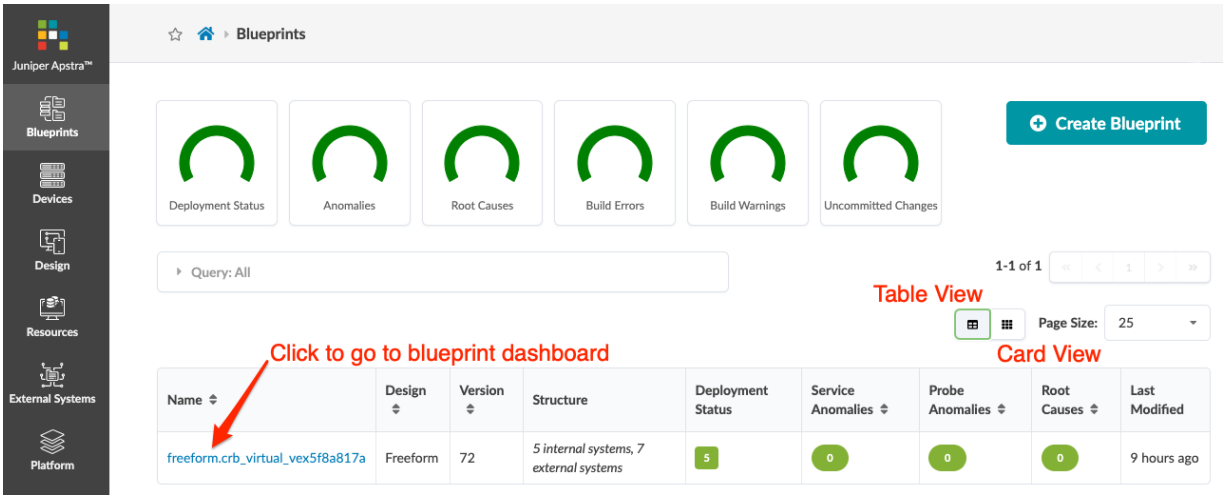
add the follwing related links at the bottom

- Freeform Reference Design Introduction
- Create / Import Freeform Blueprint
- Export Freeform Blueprint
- Delete Freeform Blueprint

Blueprints Summary

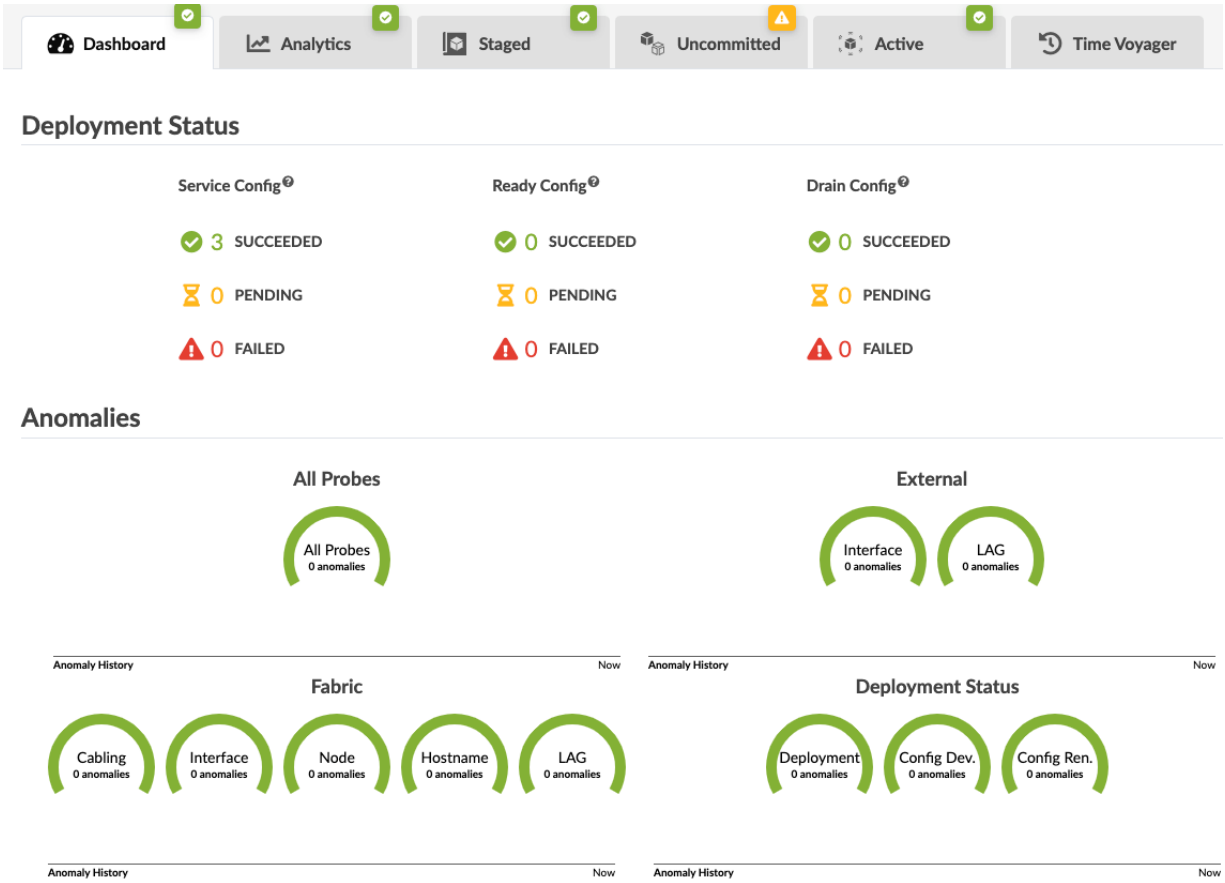
The blueprints summary page shows a summary of all your blueprints. At the top of the page, different status indicators show various statuses across all blueprints (deployment status, anomalies, root causes, build errors and warnings, and uncommitted changes. This is useful to see any issues at a glance when you have many blueprints in your Apstra instance.

From the left navigation menu of the Apstra GUI, click **Blueprints** to go to the blueprints summary page.



Dashboard

From the left navigation menu of the Apstra GUI, click **Blueprints**, then click the name of the blueprint that you want to see. The blueprint dashboard is the default view. It shows the blueprint's overall health and status. You can delete blueprints from here and also export them to be used as templates for other blueprints (by importing them).



RELATED DOCUMENTATION

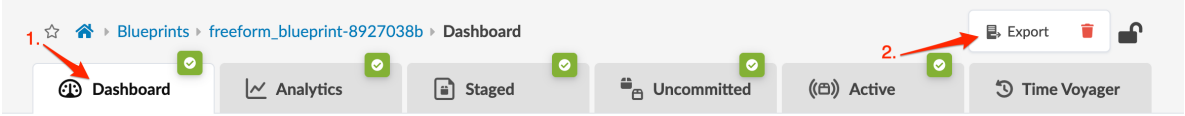
Freeform Introduction | 497

Export Freeform Blueprint

SUMMARY

You can export a Freeform blueprint to use it as a template to create another Freeform blueprint.

1. From the left navigation menu, click **Blueprints**, then click the name of the blueprint to export.
2. From the blueprint dashboard, click **Export** (top-right) to open the export dialog.



3. The exported blueprint includes all content that describes the physical environment (systems, links, device profiles, tags). Additional details are included by default. To exclude any of them from the export file, toggle them off in the dialog.

4. Click **Export** to download the JSON file of the *staged* blueprint contents and return to the blueprint dashboard.

When you create a Freeform blueprint, you'll be able to import this exported Freeform blueprint and use it as a template. You can import the blueprint into the same Apstra instance or into a different one.

RELATED DOCUMENTATION

Freeform Introduction | 497

CHAPTER 15

Physical

IN THIS CHAPTER

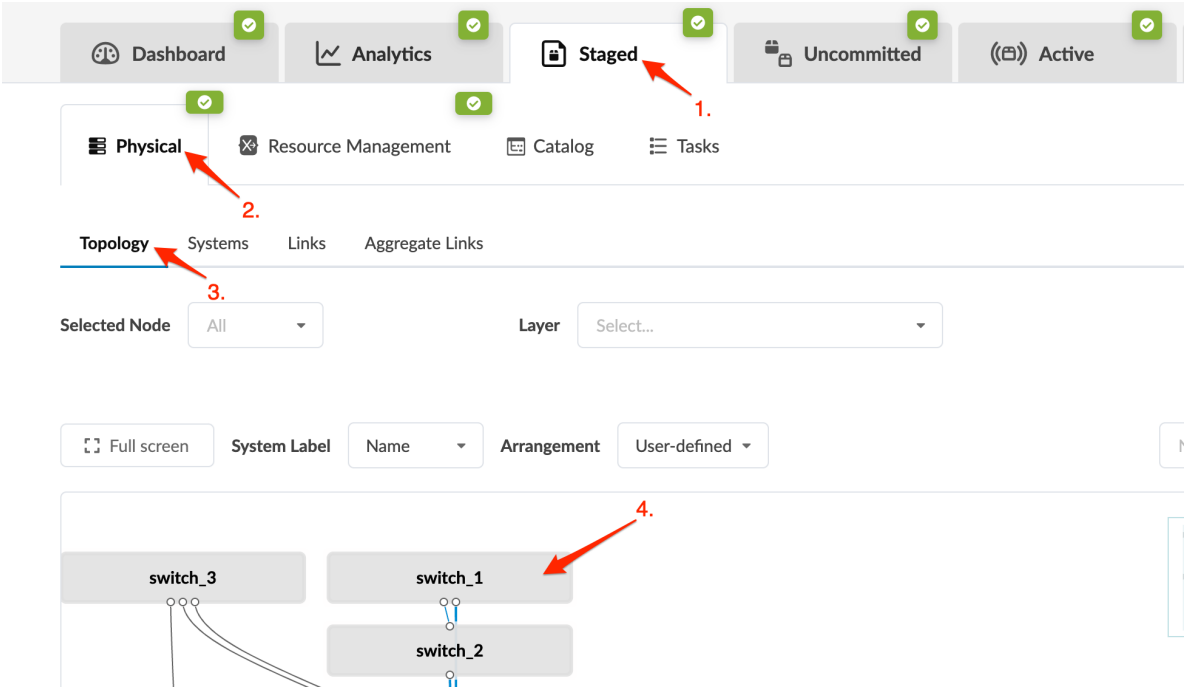
Selection

IN THIS SECTION

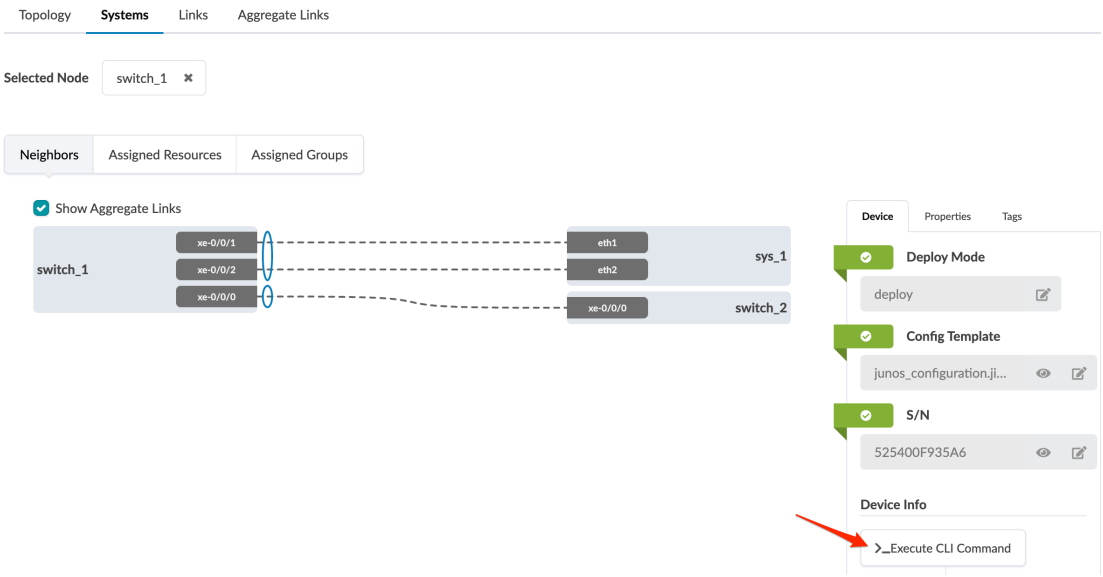
Execute CLI Show Command (Freeform Blueprint)

While in the Apstra environment, you may need device information that's obtained via CLI commands. Traditionally, you need to log in to a machine with access to the device management network, open a terminal, find device IP addresses, SSH to each of them, then run the required CLI commands. You can bypass these steps and run show commands for Juniper devices directly from the Apstra GUI. You can execute CLI commands from within the staged or active blueprint, or from the **Managed Devices** page. The steps below are for Freeform blueprints.

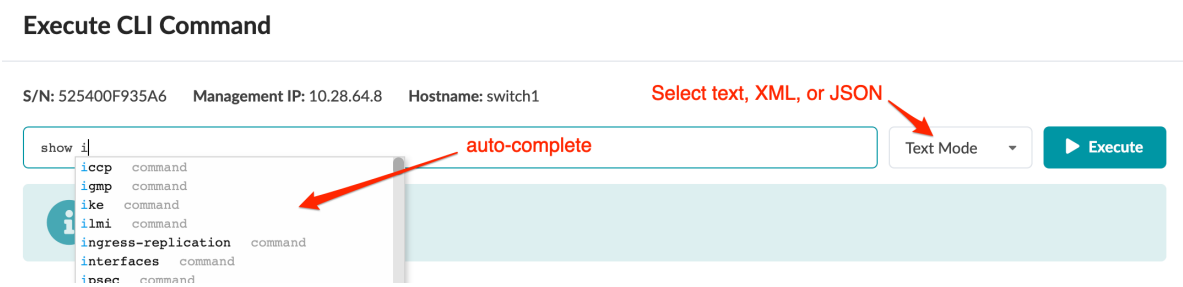
1. From the blueprint, navigate to **Staged > Physical > Topology** (or **Staged > Physical > Systems**) and select a Juniper device node.



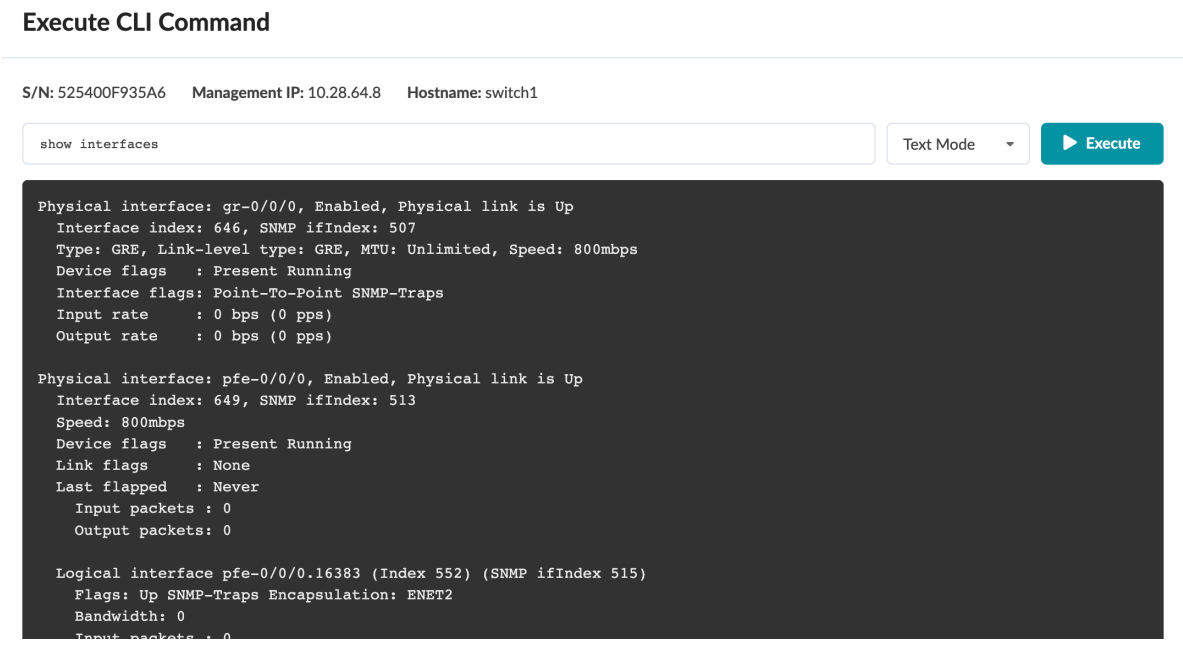
2. In the **Device** tab on the right that appears, click **Execute CLI Command**.



3. In the dialog that opens type `show`, then press the space bar. Available commands appear that you can scroll through to select, or you can start typing the command and it will auto-fill. In our example we're looking for interfaces. We typed `show`, space, then `i`, which filtered the commands to only include those with the letter i. We'll select `interfaces` to complete the command.



4. From the drop-down list, select how you want to view the results: text, XML or JSON.
5. Click **Execute** to return show command results. We used **Text Mode** for our example.



RELATED DOCUMENTATION

Execute CLI Show Command (Devices) | 671

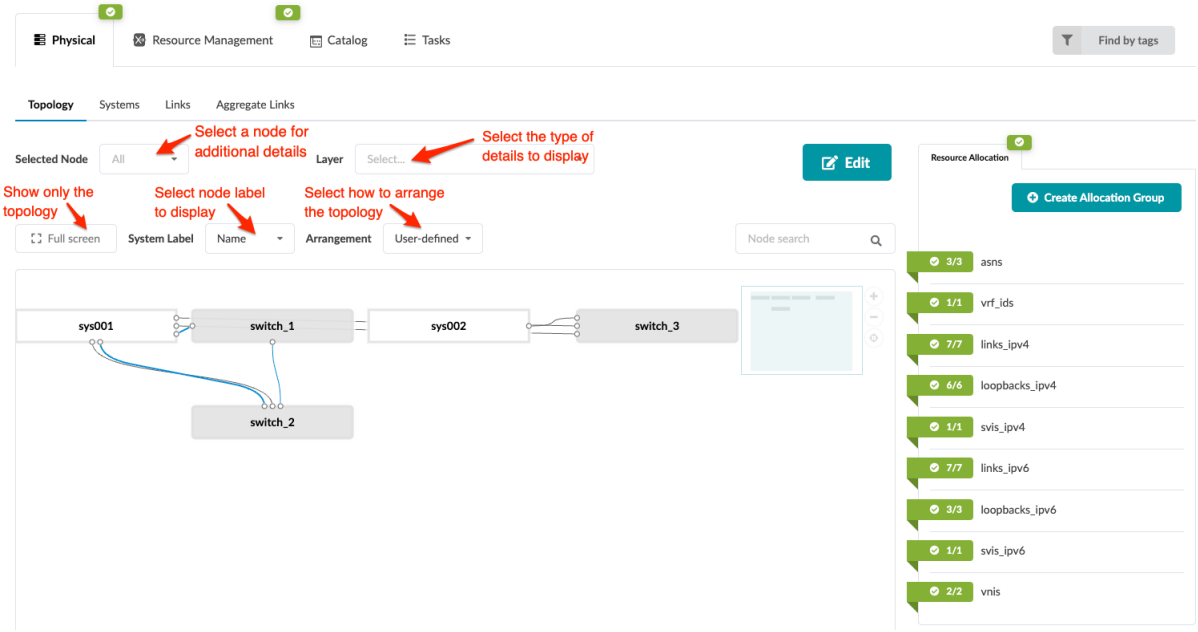
Topology

IN THIS SECTION

Topology (Freeform)

The **Topology** view shows in a graphical way the collection of devices/objects that make up the network and the links that connect devices. It self-documents your intended network state. This is then modeled/created in the Apstra GraphDB for intent-based modeling. You can perform various tasks from the **Topology** view, via the topology editor, as described in later sections.

To go to the topology view from the blueprint, navigate to **Staged > Physical > Topology**.



You can display topology information in various ways:

- To focus on just the topology without showing all the other tabs, view it in full screen mode.
- To select which label to display on system nodes, select it from the **System Label** drop-down list:
  - Name
  - Hostname
  - Serial Number (system ID)
  - IP address
- Select how the elements are arranged by selecting an arrangement from the **Arrangement** drop-down list:
  - User-defined
  - Layered

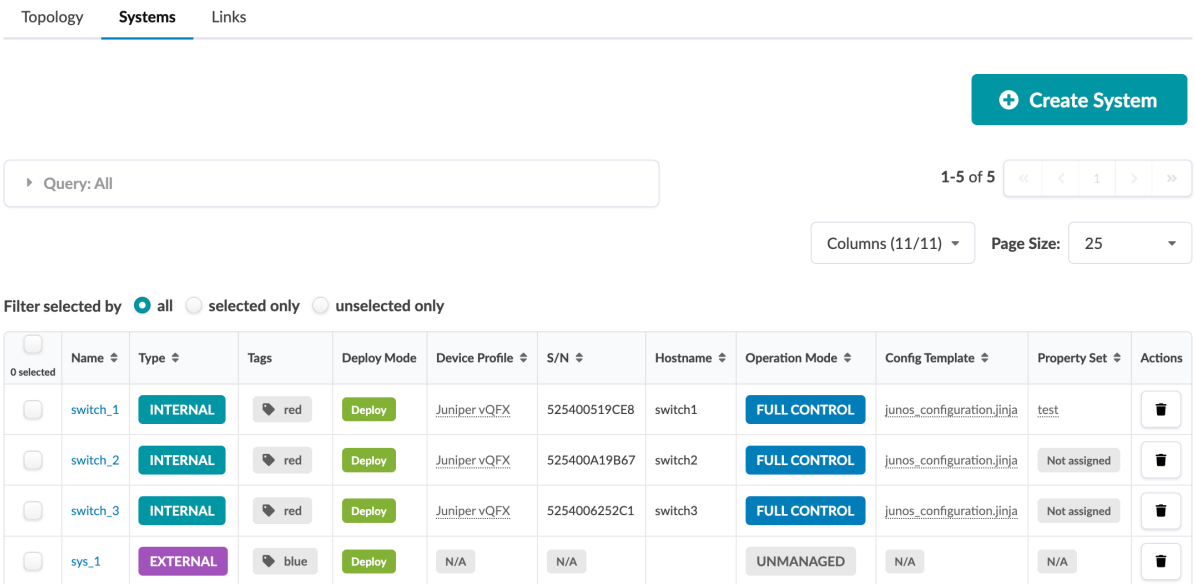
  - Stress
  - Force
  - Compaction
- To display a specific layer, select it from the **Layer** drop-down list:
  - **Config Template Assignments** - Assigned, Not Assigned
  - **Deploy Mode** - Deploy, Ready, Drain, Undeploy
  - **Operation Mode** - Full Control, Telemetry Only
  - **System ID Assignements** - Assigned, Not Assigned
  - **Uncommitted Changes** - Has Uncommitted Changes or not
- Select a node from the **Selected Node** drop-down list to go to the **Systems** detail page.

Systems

IN THIS SECTION

Systems Introduction (Freeform)

The **Systems** view (Staged > Physical > Systems) shows in a table format the collection of devices/objects that make up the network (similar to the Nodes view in Datacenter reference designs). The table includes information about internal and external systems in the blueprint, tags, deploy mode, assigned device profile, assigned system ID, hostname, operation mode (full control), assigned config template, and assigned property set. You can see details at a glance and tell if there are any issues with missing requirements. You can customize what appears in the table by selecting/deselecting elements in the columns drop-down list. You can perform various tasks from the **Systems** view as described in later sections.



Create Internal System (Freeform)

IN THIS SECTION

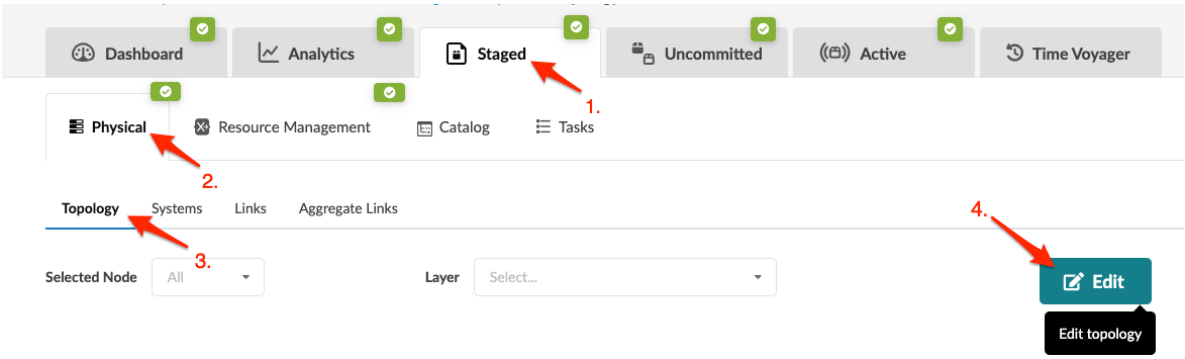
Systems represent switches, routers, Linux hosts and so on. Managed devices that you add to a blueprint are called *internal systems*. You can create systems from scratch, or you can clone systems and

customize them to create new ones. You can create (and clone) from the **Topology** view or from the **Systems** view.

Internal systems must be mapped to device profiles. Before creating systems, make sure you've imported the relevant device profiles into the blueprint catalog.

Create Internal System (from Topology Editor)

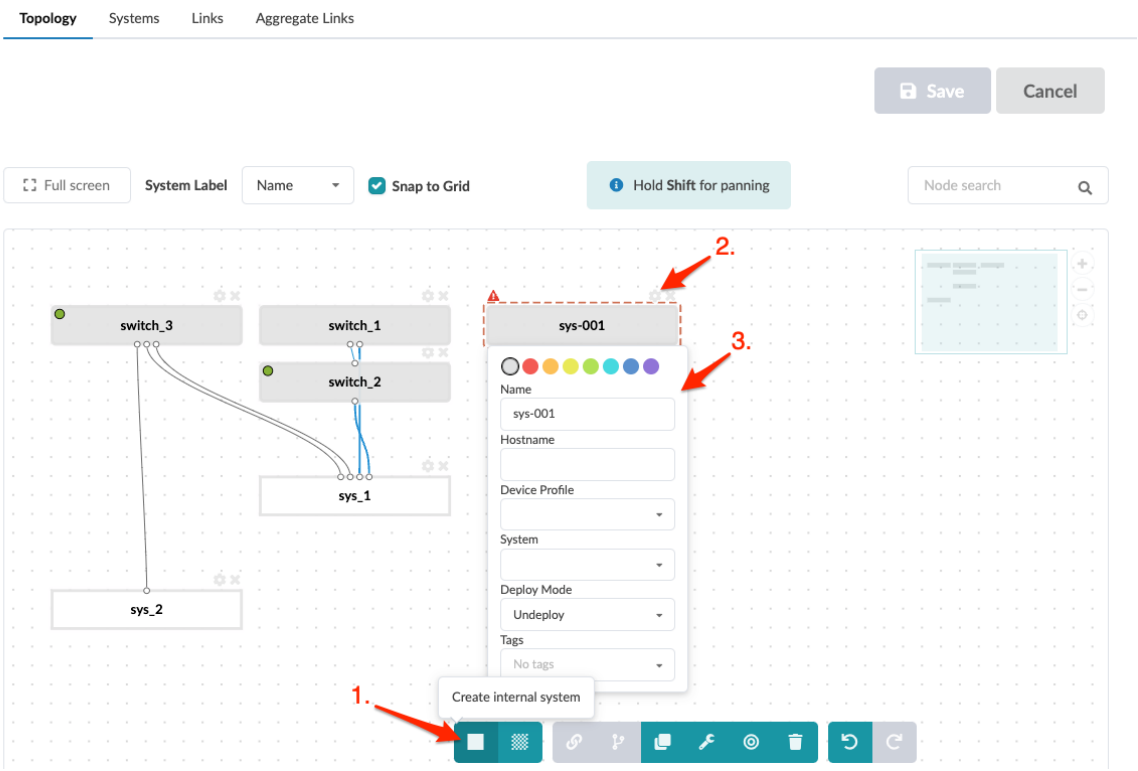
1. From the blueprint, navigate to **Staged > Physical > Topology** and click **Edit**.



> **CAUTION**: Be careful. If you click away from the topology editor without clicking **Save**, your changes are discarded.

2. In the topology editor that opens, click the **Create internal system** button (bottom-left). The system appears as a gray rectangle with a system-generated name. The red triangle indicates that information is needed for required fields. In this case, it's the device profile.

   You can move systems around on the canvas and when you save your changes in the editor and then reopen it, your systems will still be where you moved them.



3. Click the gear to open the parameters dialog.
4. You can change the system color that displays in the topology. This is useful for designating different roles or anything else you'd like to visually differentiate.
5. You can change the system name and hostname to customize them for your environment.
6. Select a device profile from the drop-down list. (Device profiles come from the blueprint catalog. If you don't see the one you need, import it.)
7. You can assign the system ID now or later. To assign it now, select it from the **System** drop-down list. (The list includes managed devices that haven't been assigned yet. If you have your devices ready and they're not appearing in this list, you still need to bring them under Apstra management by adding them to Managed Devices.)
8. You can add tags, then later when you want to find systems you can use the **Find by Tags** feature (upper-right) to find them. You can also include tags in config templates, then systems with those tags will be rendered as specified in the config template.
9. Click **Save** to stage your new system and return to the **Topology** view. (If you leave the page without saving, your changes are discarded.)

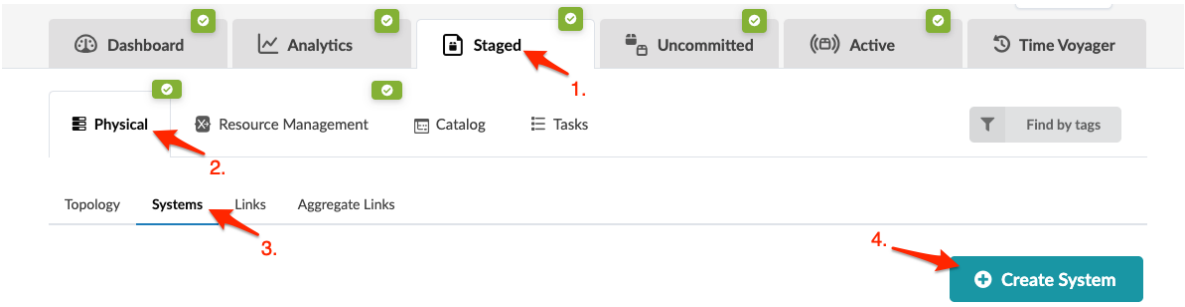
Next Steps:

- Continue to create internal and "external systems" on page 516 until you've added your devices to the topology.
- "Add links" on page 553 to systems.

- "Assign config templates" on page 521 to your internal systems with deploy mode set to **Deploy**.

Create Internal System (from Systems View)

1. From the blueprint, navigate to **Staged > Physical > Systems** and click **Create System**.



2. In the **Create System** dialog, enter a name and select **INTERNAL**.
3. Internal systems are associated with device profiles. You can either assign just the device profile now (and assign the system ID later), or if you've brought your devices under Apstra management, you can select the system ID now.
   - **From Scratch** - select a device profile (that was imported into the blueprint catalog.) (You'll assign the system ID later.)
   - **From Managed Devices** - select a managed device to assign its system ID to the system.
4. Enter a hostname (optional).
5. You can add tags, then later when you want to find systems you can use the **Find by Tags** feature (upper-right) to find them. You can also include tags in config templates, then systems with those tags will be rendered as specified in the config template.
6. Click **Create** to stage your new system and return to the **Systems** view. The newly created system appears in the list.

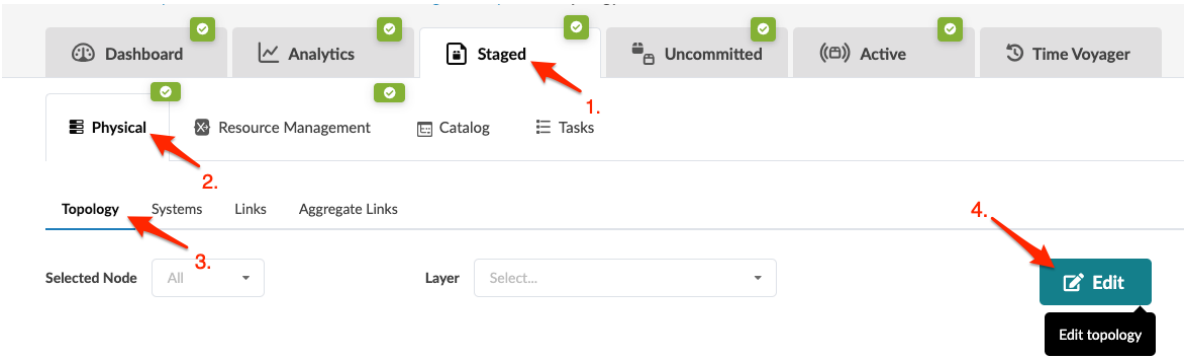
Next Steps:

- Continue to create internal and "external systems" on page 516 until you've added your devices to the topology.
- "Add links" on page 553 to systems.
- "Assign config templates" on page 521 to your internal systems with deploy mode set to **Deploy**.

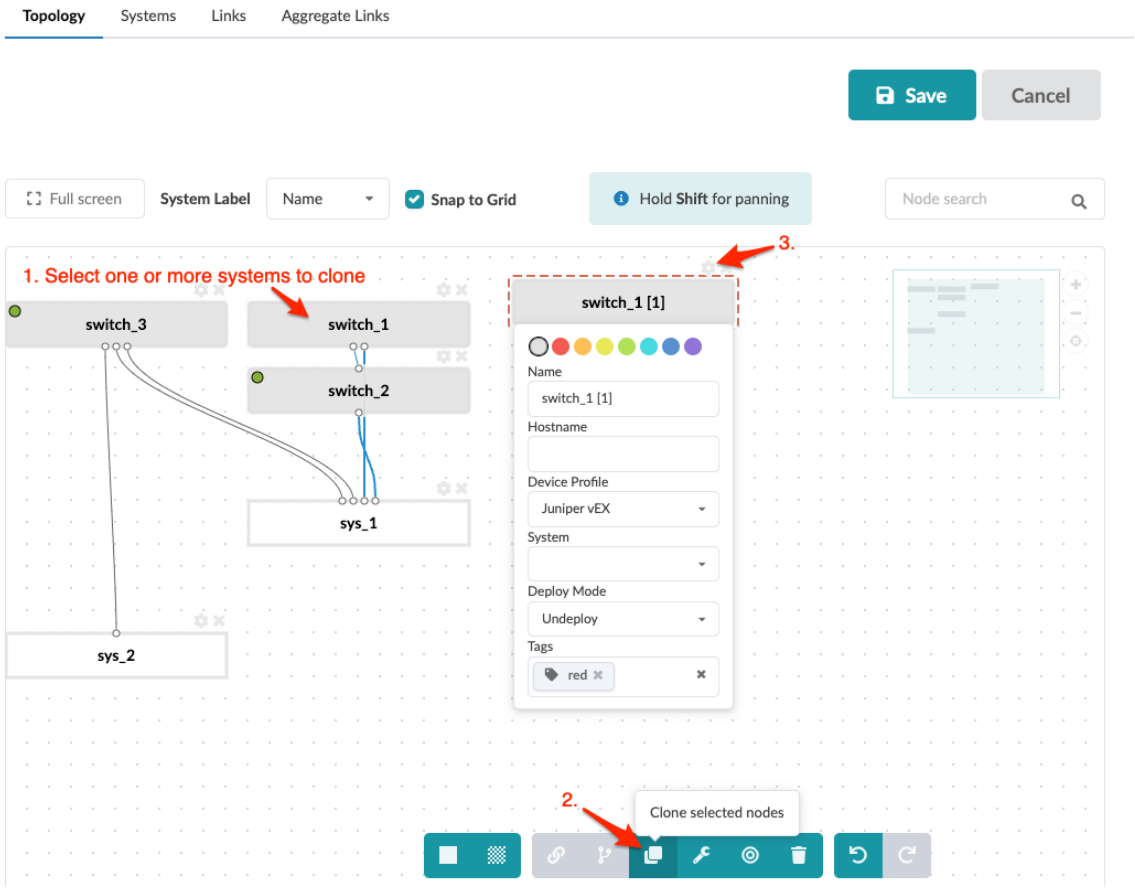
Clone Internal System (from Topology Editor)

You can clone systems and customize them to create new ones from the **Topology** view.

1. From the blueprint, navigate to **Staged > Physical > Topology** and click **Edit**.



2. In the topology editor, select one or more existing internal systems, then click the **Clone selected nodes** button.



3. The new system(s) appear as gray rectangles with system-generated names.

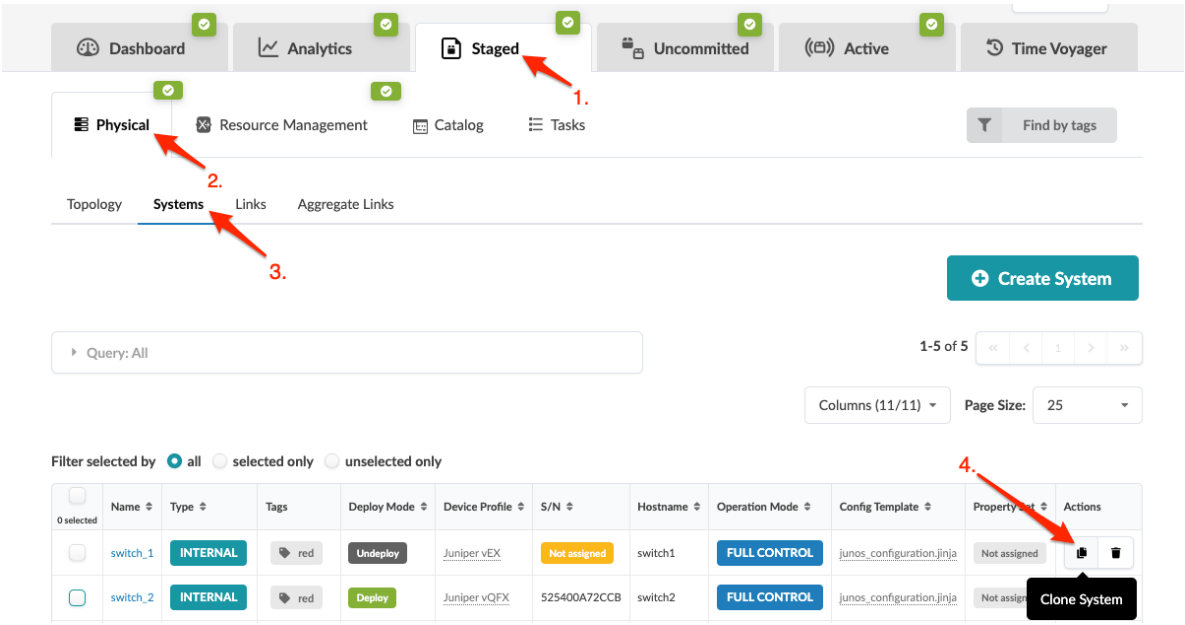
   You can move systems around on the canvas and when you save your changes in the editor and then reopen it, your systems will still be where you moved them.

4. Click the gear to open the parameters dialog, and change details to customize your new system.
5. Click **Save** to stage your new system(s) and return to the **Topology** view. (If you leave the page without saving, your changes are discarded.)

Clone Internal System (from Systems View)

You can clone systems and customize them to create new ones from the **Systems** view.

1. From the blueprint, navigate to **Staged > Physical > Systems** and click **Clone System** for the system you want to clone.



2. Change details to customize your new system.
3. Click **Clone** to stage your new system and return to the **Systems** view.

SEE ALSO

Import Device Profile (Freeform) | 583

Update System ID Assignment (Freeform) | 534

Add Managed Device | 651

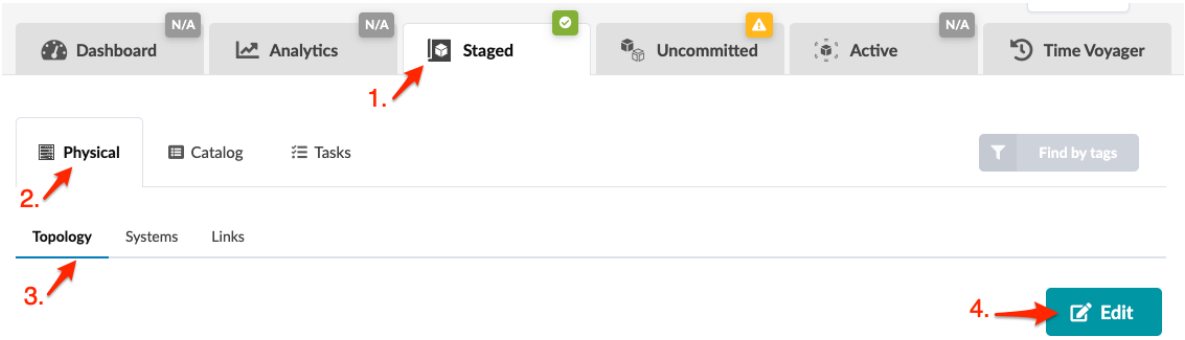
Create External System (Freeform)

IN THIS SECTION

Systems represent switches, routers, Linux hosts and so on. Unmanaged devices that you add to a blueprint are called *external systems*. They link to managed (internal) systems. You can create systems from scratch, or you can clone systems and customize them to create new ones. You can create (and clone) from the **Topology** view or from the **Systems** view.

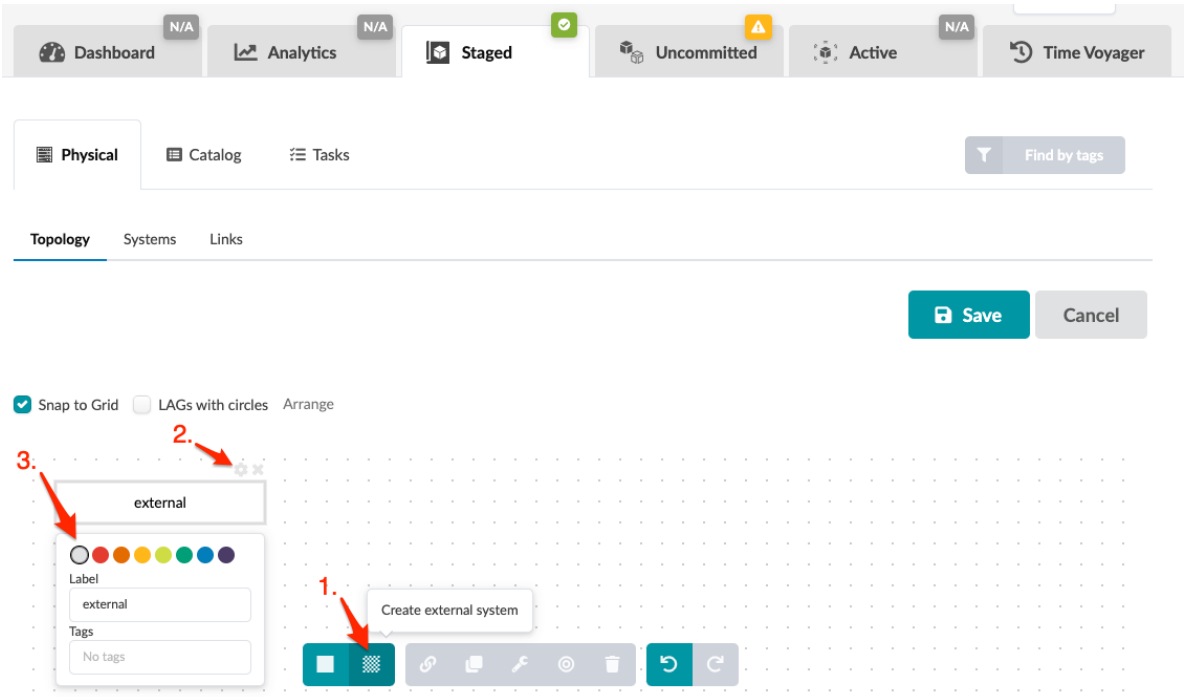
Create External System (from Topology)

1. From the blueprint, navigate to **Staged > Physical > Topology** and click **Edit**. (Images in this section are from Apstra version 4.1.1.)



> **CAUTION**: Be careful. If you click away from the topology editor without clicking **Save**, your changes are discarded.

2. In the topology editor click the **Create external system** button. The system appears as a rectangle with a system-generated name. You can move systems around on the canvas and when you save your changes in the editor and then reopen it, your systems will still be where you moved them to. You can save the system as is since there are no other required fields, or you can open the parameters dialog and configure optional fields.



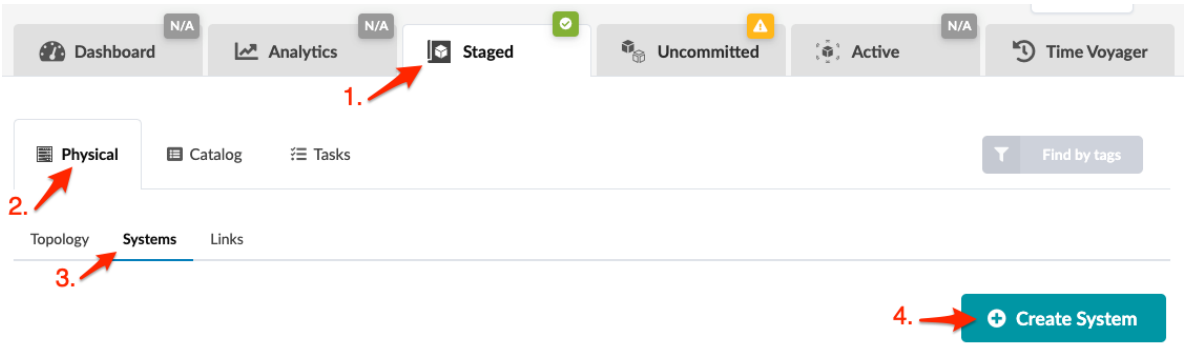
3. Click the gear to open the parameters dialog.
4. You can change the system color that displays in the topology. This is useful for designating different roles or anything else you'd like to visually differentiate.
5. You can change the system label to customize it to your environment.
6. You can add tags, then later when you want to find systems you can use the **Find by Tags** feature (upper-right) to find them.
7. Click **Save** to stage your new system and return to the **Topology** view. (If you leave the page without saving, your changes are discarded.)

Next Steps:

Continue to create external systems and "internal systems" on page 511 until you've added your devices to the topology. Then you can "create links" on page 553 for them.

Create External System (from Systems)

1. From the blueprint, navigate to **Staged > Physical > Systems** and click **Create System**. (The image below is from Apstra version 4.1.1.)



2. Enter a name and select **EXTERNAL**.
3. Enter a hostname (optional) and tags (optional). If you add tags, then later when you want to find systems you can use the **Find by Tags** feature (upper-right) to find them.
4. Click **Create** to stage the change and return to the **Systems** view. The newly created system appears in the list.

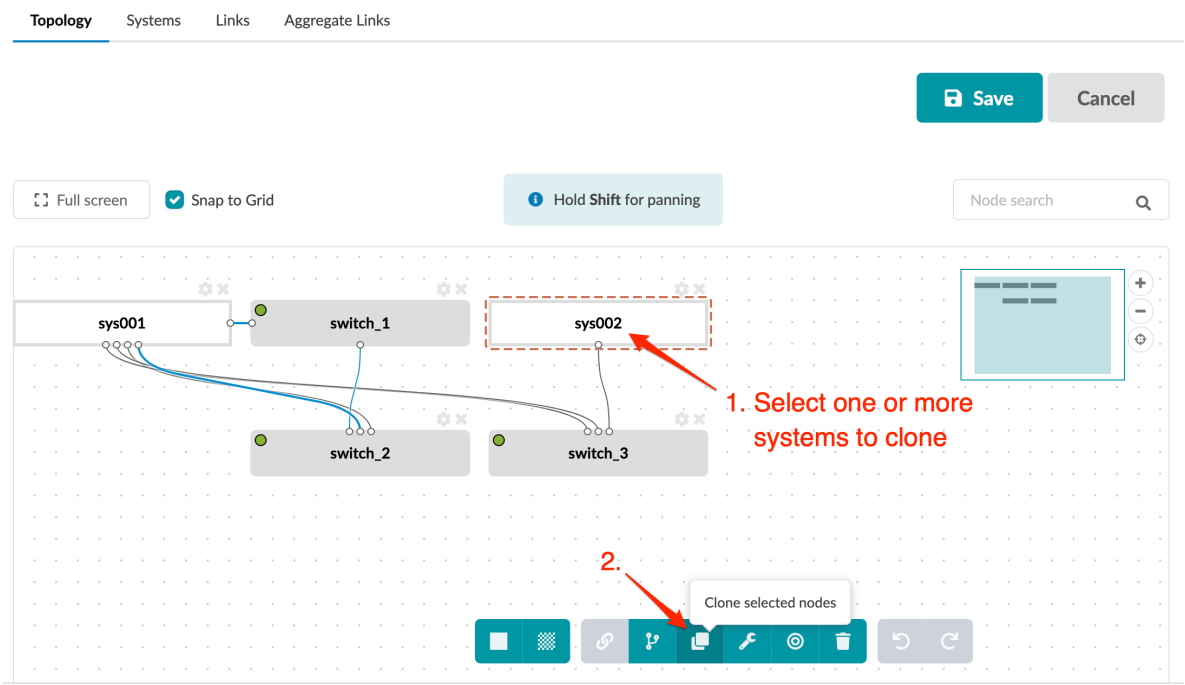
Next Steps:

Continue to create external systems and "internal systems" on page 511 until you've added your devices. Then you can "create links" on page 553 for them.

Clone External System (from Topology)

You can clone systems and customize them to create new ones from the **Topology** view.

1. From the blueprint, navigate to **Staged > Physical > Topology** and click **Edit**.
2. In the topology editor, select one or more existing external systems, then click the **Clone selected nodes** button. (The image below is from Apstra version 4.1.2.)

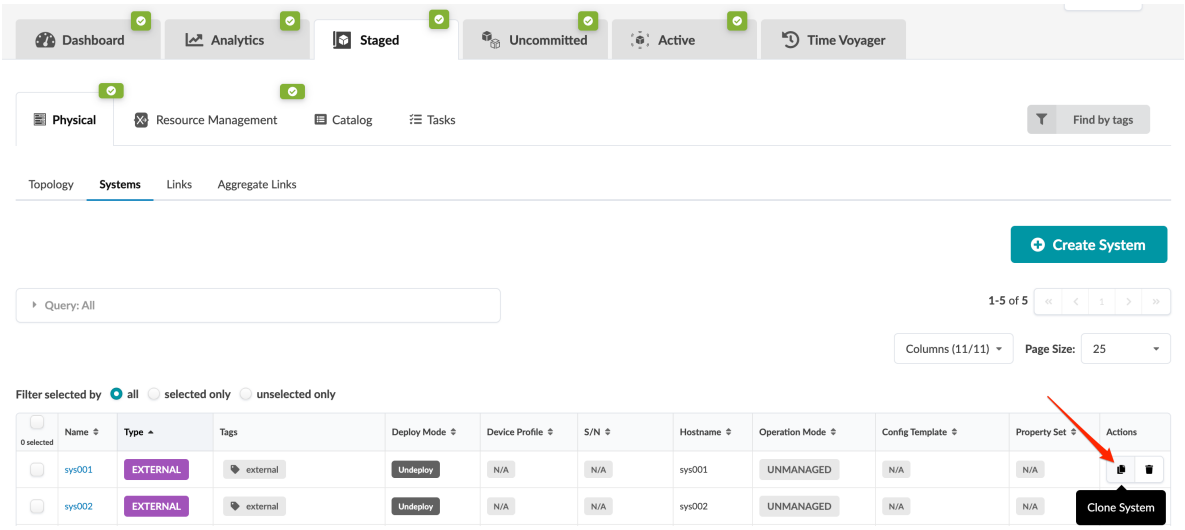


3. The new system(s) appear as gray rectangles with system-generated names. You can move systems around on the canvas and when you save your changes in the editor and then reopen it, your systems will still be where you moved them to.
4. Click the gear to open the parameters dialog, and change details to customize your new system.
5. Click **Save** to stage your new system(s) and return to the **Topology** view. (If you leave the page without saving, your changes are discarded.)

Clone External System (from Systems)

You can clone systems and customize them to create new ones from the **Systems** view, as of Apstra version 4.1.2.

1. From the blueprint, navigate to **Staged > Physical > Systems** and click **Clone System** for the system you want to clone.



2. Change details to customize your new system.
3. Click **Clone** to stage your new system and return to the **Systems** view.

Update Assigned Config Template(Freeform)

SUMMARY

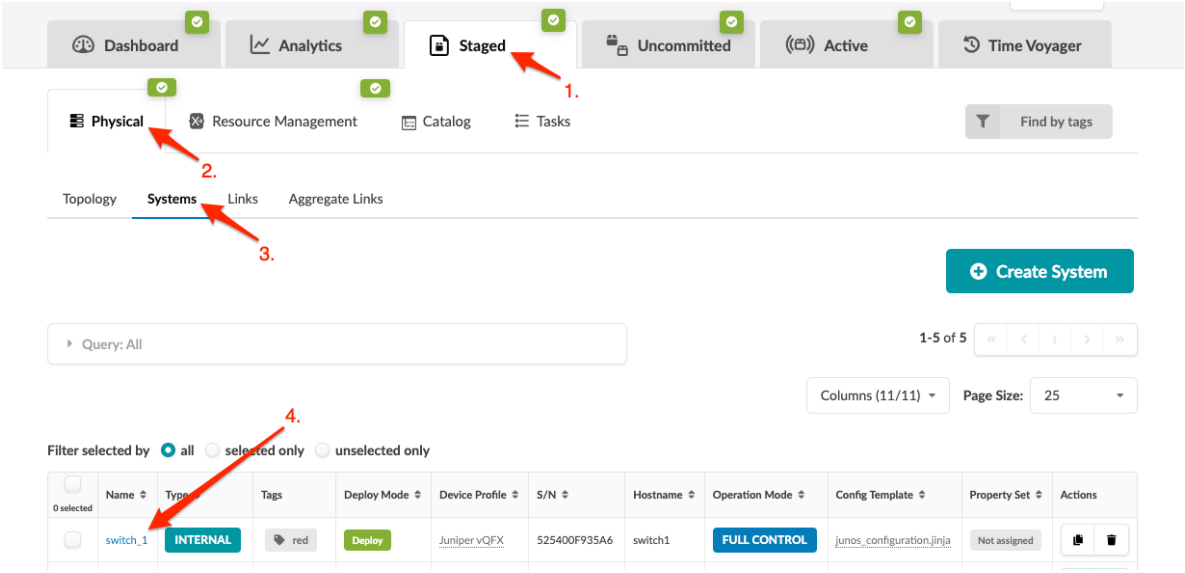
You can update Freeform config template assignments on one or more systems.

IN THIS SECTION

Update Config Template Assignment on One System (from Systems)

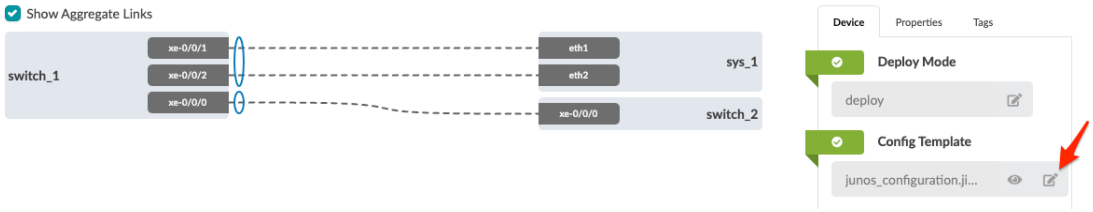
If you haven't created your "config templates" on page 578 yet, do that now.

1. From the blueprint, navigate to **Staged > Physical > Systems** and click the system name to go to details for that system.



NOTE: You can also get to the Systems details page from the **Topology** view. From the blueprint, navigate to **Staged > Physical > Topology** and select the system to update.

2. In the panel on the right, in the **Devices** tab, click the **Edit** button for the **Config Template** field.



3. Select the config template from the drop-down list, then click the **Save** button. (Or, to cancel the change, click the gray discard button. Or, to remove the config template, click the red remove button.) The panel shows the value for the active blueprint and the staged value

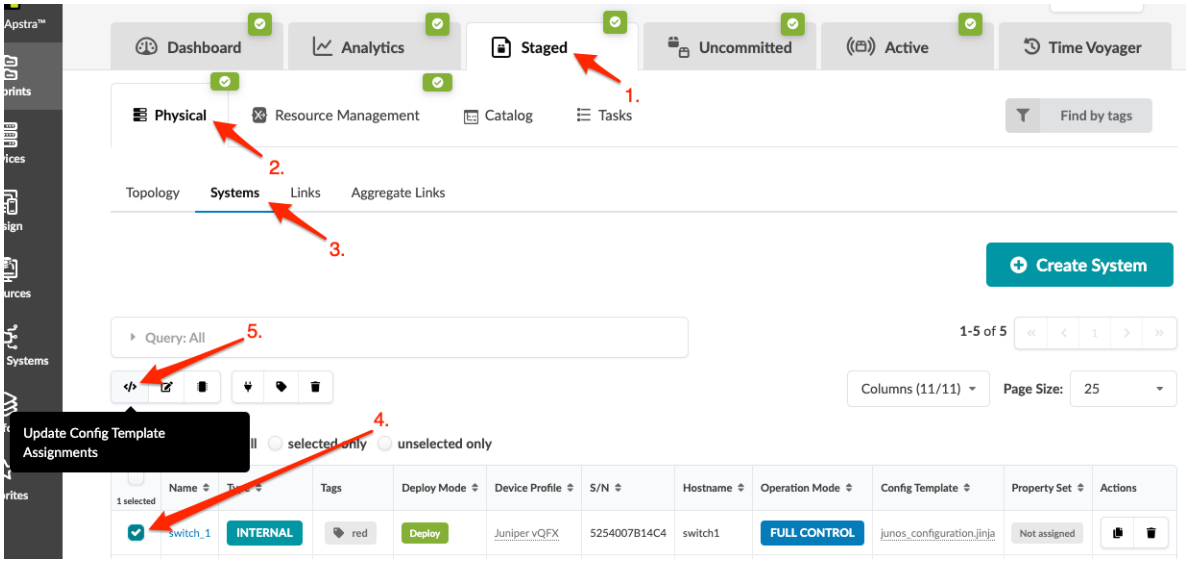
When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Update Config Template Assignment (Multiple Systems)

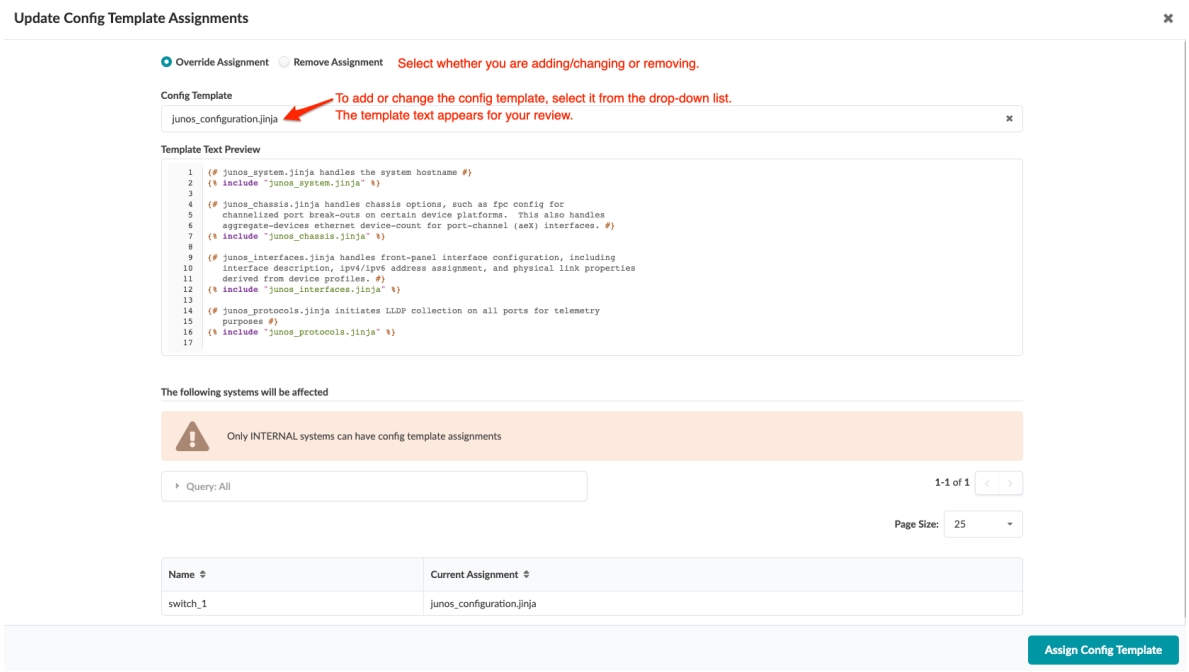
Internal systems with deploy mode set to **Deploy** require an assigned config template.

If you haven't created your "config templates" on page 578 yet, do that now.

1. From the blueprint, navigate to **Staged > Physical > Systems** to go to the **Systems** view.



2. Select one or more check boxes for the system(s) to update. (The same action will be applied to all selected systems. That is, all selected systems will be assigned or unassigned the same config template.)
3. Click the **Update Config Template Assignments** button that appears above the table.
4. To add or replace a config template assignment, leave **Override Assignment** selected and select a config template from the drop-down list. The template text appears for your review. (Each internal system is assigned only one config template, but that config template could nest other config templates within it.)



5. Or, to remove a config template assignment, select **Remove Assignment**.

6. Click **Assign Config Template** (or **Remove Config Template Assignments**, as applicable) to stage the changes and return to the **Systems** view.

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Update System Name (Freeform)

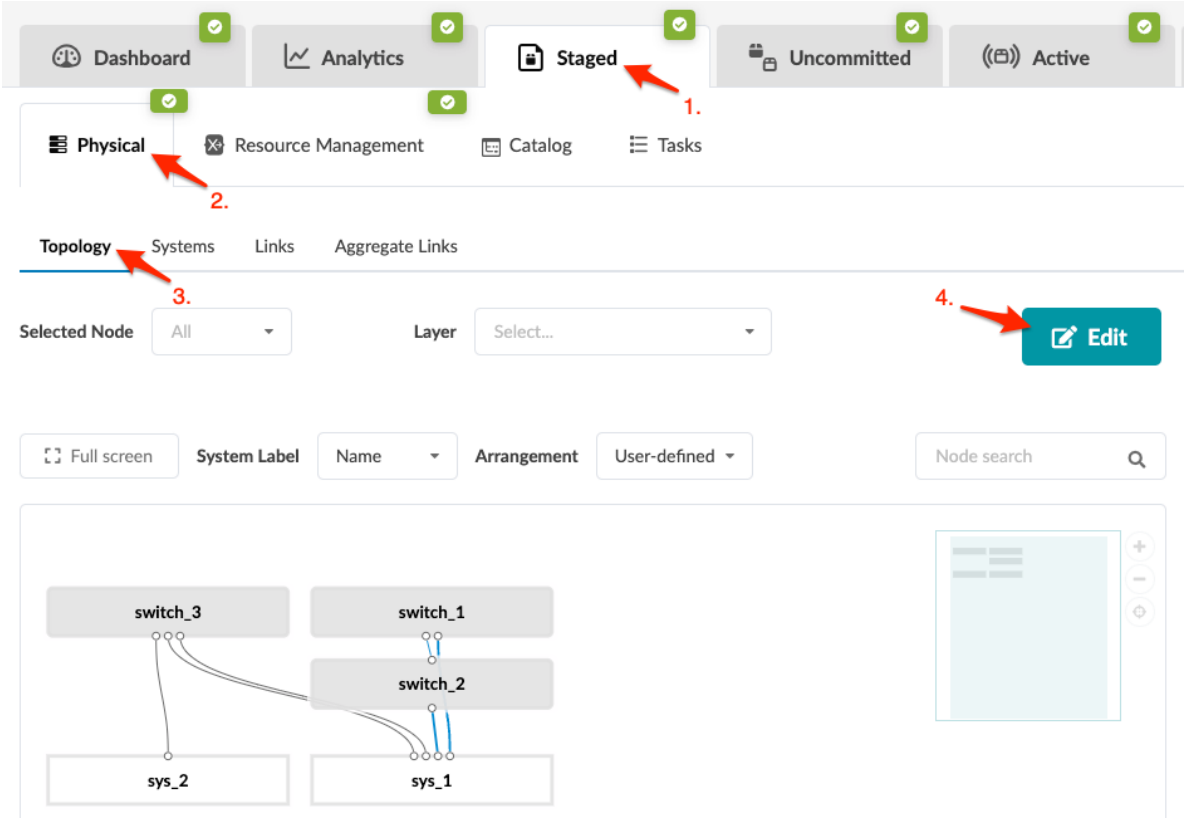
SUMMARY

You can change Freeform system names from the **Topology** or **Systems** view.

IN THIS SECTION

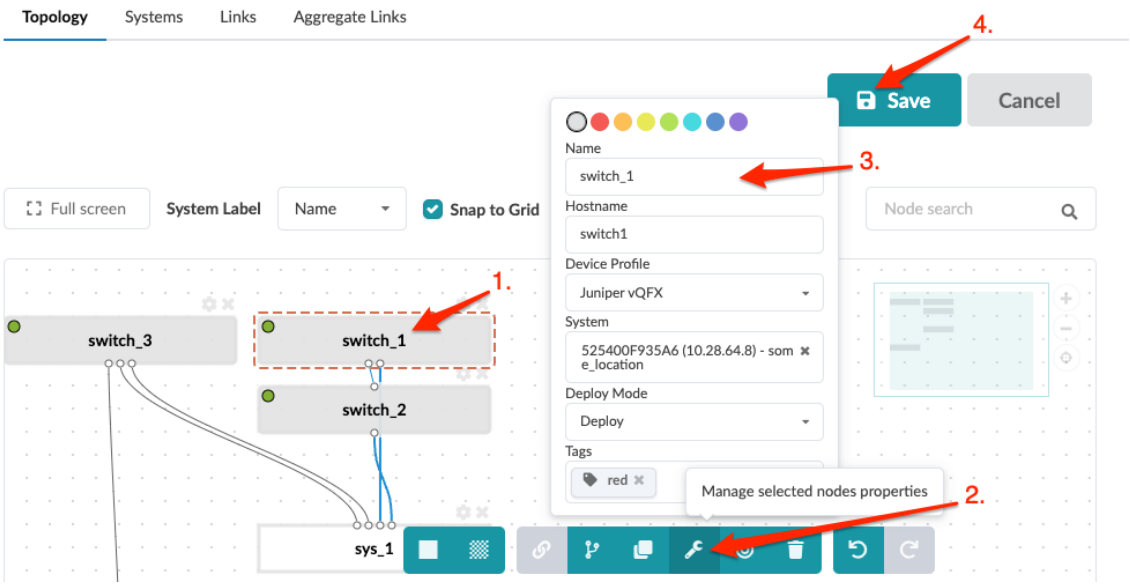
Update System Name (from Topology)

1. From the blueprint, navigate to **Staged > Physical > Topology** and click **Edit** to open the topology editor.



> ⚠ **CAUTION**: Be careful. If you click away from the topology editor after making changes without clicking **Save**, your changes are discarded.

2. In the topology editor, click the system to change, then click the **Manage selected nodes properties** button that becomes available. (You can also open the same dialog by clicking the settings button for the selected system. It's the gear at the top-right of the system.)

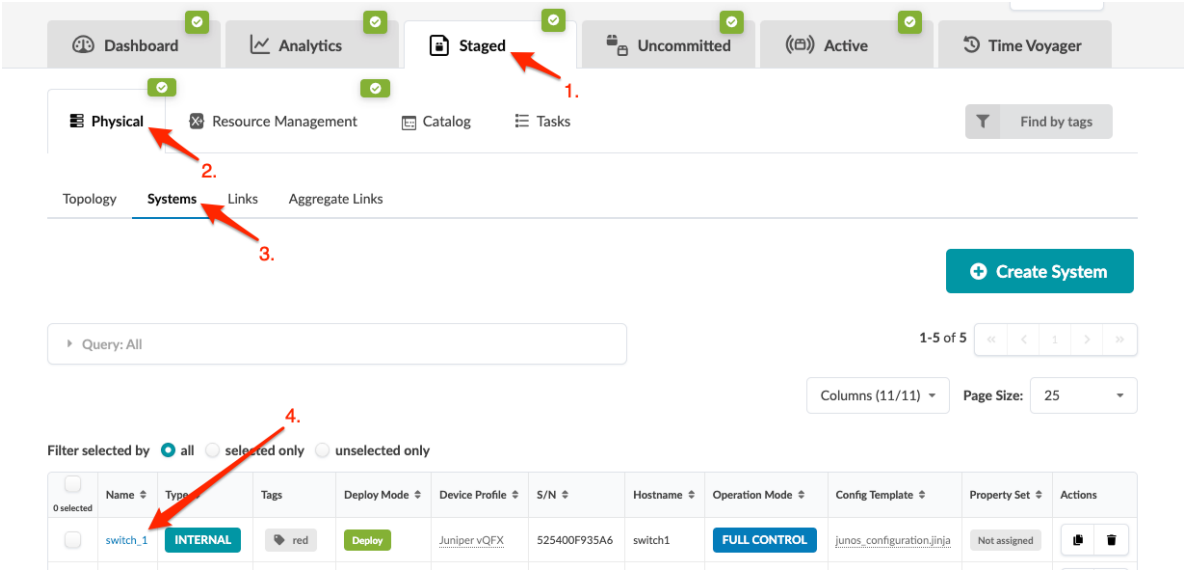


3. Enter the new name in the **Name** field.
4. To close the dialog, click anywhere on the *canvas* outside of the dialog.
5. Click **Save** to stage your changes, exit the topology editor and return to the **Topology** view. (If you leave the page without saving, your changes are discarded.)

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Update System Name (from Systems)

1. From the blueprint, navigate to **Staged > Physical > Systems** and click the system name to go to details for that system.



NOTE: You can also get to the Systems details page from the **Topology** view. From the blueprint, navigate to **Staged > Physical > Topology** and select the system to update.

2. In the panel on the right, click the **Properties** tab, then click the **Edit** button for the **Name** field.

3. Enter the new name and click the **Save** button. (To cancel the change, click the gray discard button.)

The panel shows the values for the active blueprint and the staged blueprint.

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Update Hostname (Freeform)

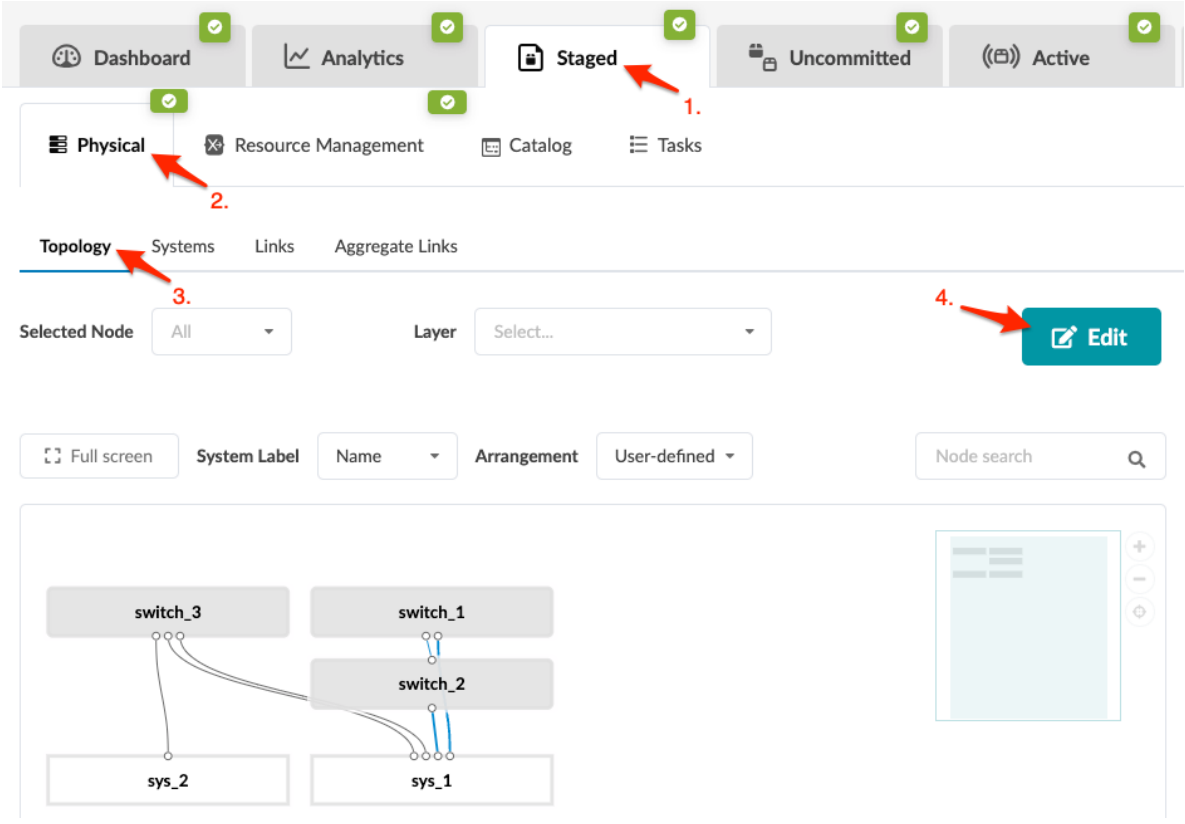
SUMMARY

You can change Freeform system hostnames from the **Topology** or **Systems** view.

IN THIS SECTION

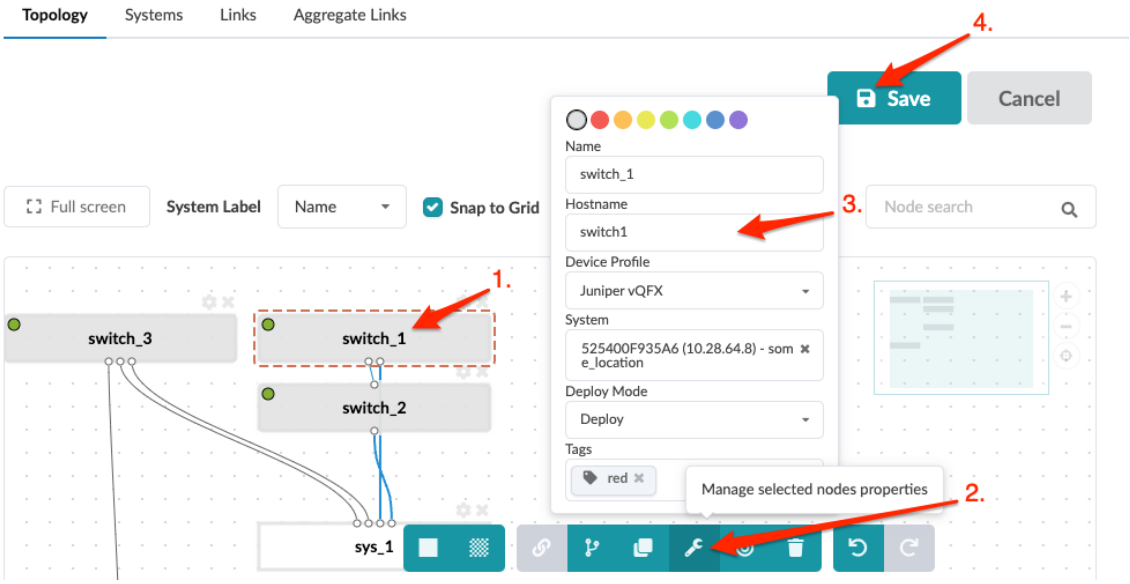
Update System Hostname (from Topology)

1. From the blueprint, navigate to **Staged > Physical > Topology** and click **Edit** to open the topology editor.



> **CAUTION**: Be careful. If you click away from the topology editor after making changes without clicking **Save**, your changes are discarded.

2. In the topology editor, click the system to change, then click the **Manage selected nodes properties** button that becomes available. (You can also open the same dialog by clicking the settings button for the selected system. It's the gear at the top-right of the system.)

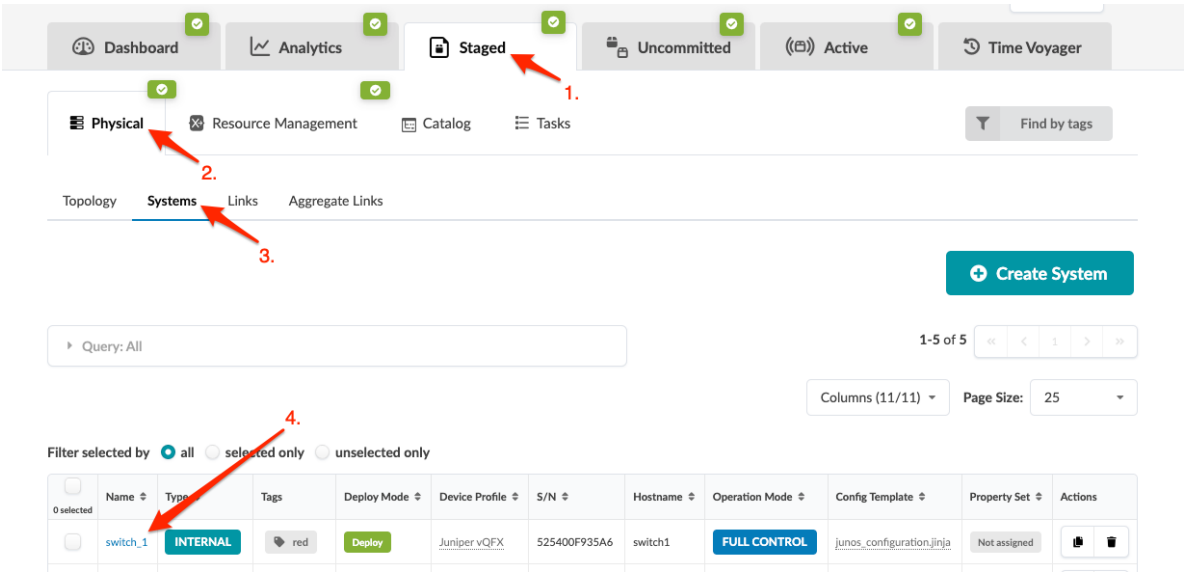


3. Enter the new hostname in the **Hostname** field.
4. To close the dialog, click anywhere on the *canvas* outside of the dialog.
5. Click **Save** to stage your changes, exit the topology editor and return to the **Topology** view. (If you leave the page without saving, your changes are discarded.)

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

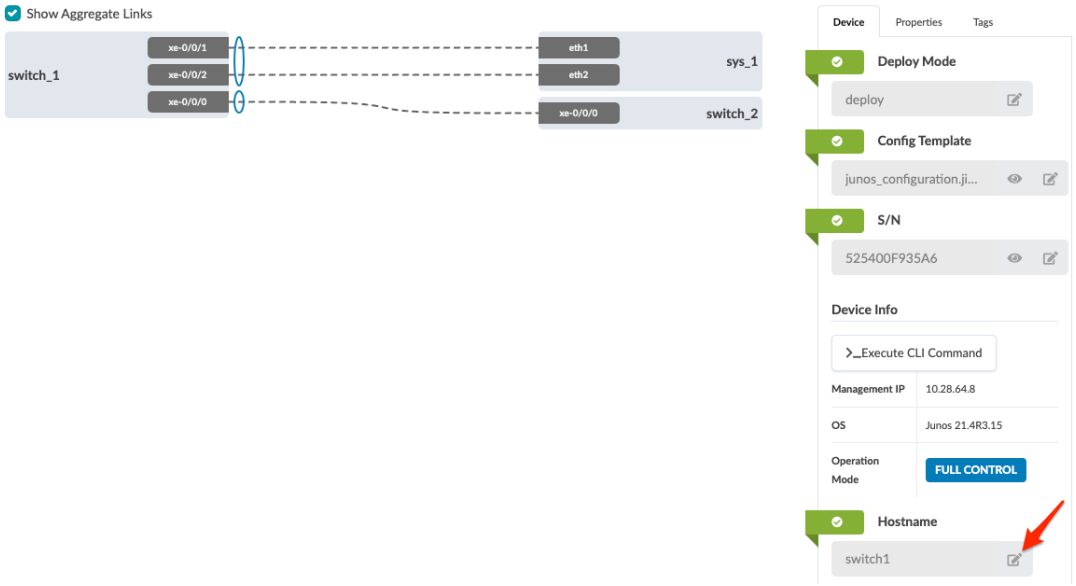
Update System Hostname (from Systems)

1. From the blueprint, navigate to **Staged > Physical > Systems** and click the system name to go to details for that system.



NOTE: You can also get to the Systems details page from the **Topology** view. From the blueprint, navigate to **Staged > Physical > Topology** and select the system to update.

2. In the panel on the right, in the **Devices** tab, click the **Edit** button for the **Hostname** field.



3. Enter the new hostname and click the **Save** button. (To cancel the change, click the gray discard button.)
   The panel shows the values for the active blueprint and the staged blueprint.

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Change Assigned Device Profile (Freeform)

SUMMARY

You can change Freeform device profile assignments from the **Topology** or **Systems** view.

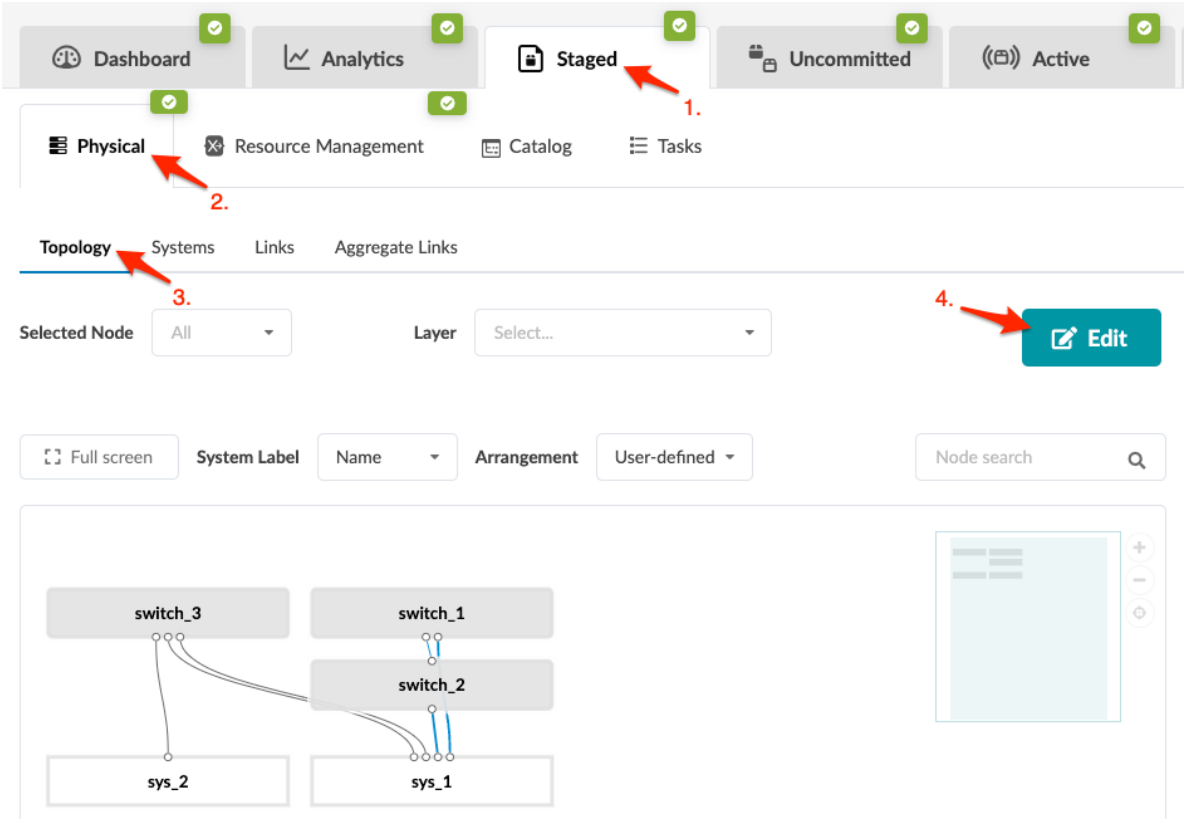
IN THIS SECTION

The device profile may need to change for various reasons, such as the following:

- If you need to update the selector during NOS upgrade
- If you need an RMA
- If Juniper Support provides you with a new device profile to resolve an issue

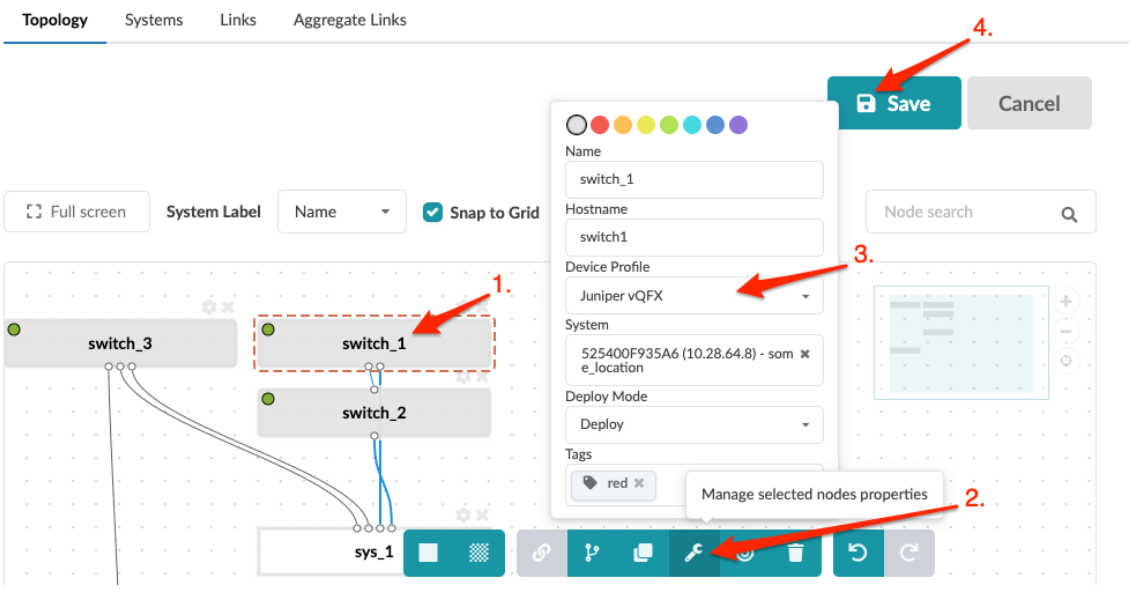
Update Device Profile Assignment (from Topology)

1. From the blueprint, navigate to **Staged > Physical > Topology** and click **Edit** to open the topology editor.



> ⚠ **CAUTION**: Be careful. If you click away from the topology editor after making changes without clicking **Save**, your changes are discarded.

2. In the topology editor, select one or more systems to change to the same device profile, then click the **Manage selected nodes properties** button that becomes available. (You can also open the same dialog by clicking the settings button for a selected system. It's the gear at the top-right of the system.)



3. Select the new device profile from the **Device Profile** drop-down list.

   Device profiles come from the blueprint catalog. If you don't see the one you need, import it.
4. To close the dialog, click anywhere on the *canvas* outside of the dialog.
5. Click **Save** to stage your changes, exit the topology editor and return to the **Topology** view. (If you leave the page without saving, your changes are discarded.)

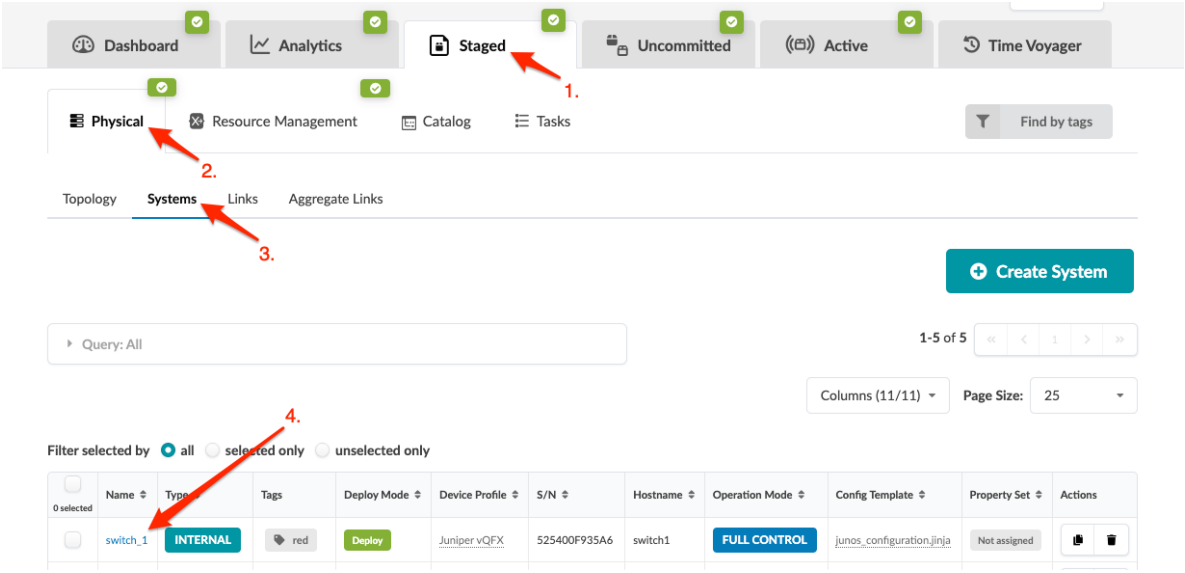
When you're ready to activate your changes, commit them from the **Uncommitted** tab.

SEE ALSO

> Import Device Profile (Freeform) | 583
>
> Upgrade Device NOS | 655

Update Device Profile Assignment (from Systems)

1. From the blueprint, navigate to **Staged > Physical > Systems** and click the system name to go to details for that system.



NOTE: You can also get to the Systems details page from the **Topology** view. From the blueprint, navigate to **Staged > Physical > Topology** and select the system to update.

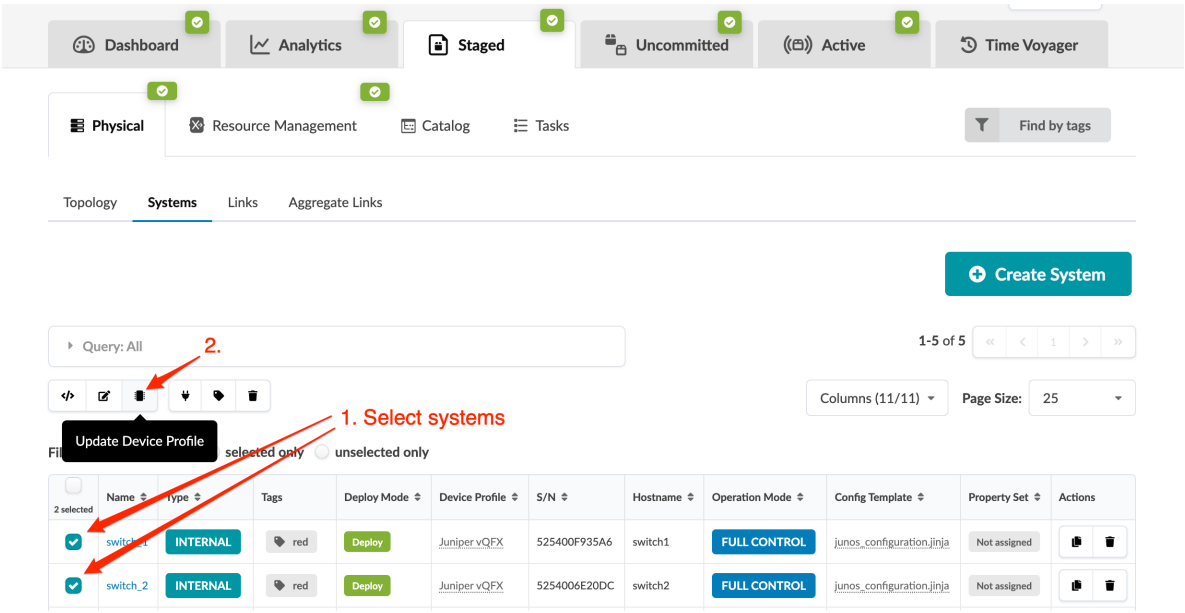
2. In the panel on the right, click the **Properties** tab, then click the **Edit** button for the **Device Profile** field.

3. Select the new device profile from the **Device Profile** drop-down list, then click the **Save** button. (Or, to cancel the change, click the gray discard button.)
   The panel shows the value for the active blueprint and the staged value

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Update One or More Device Profile Assignments (from Systems)

1. From the blueprint, navigate to **Staged > Physical > Systems** and select the check boxes for one or more systems, then click the **Update Device Profile** button that becomes available above the table.



2. In the dialog that opens, select the new device profile from the Device Profile drop-down list.
3. Click **Update** to stage the changes and return to the **Systems** view.

Update System ID Assignment (Freeform)

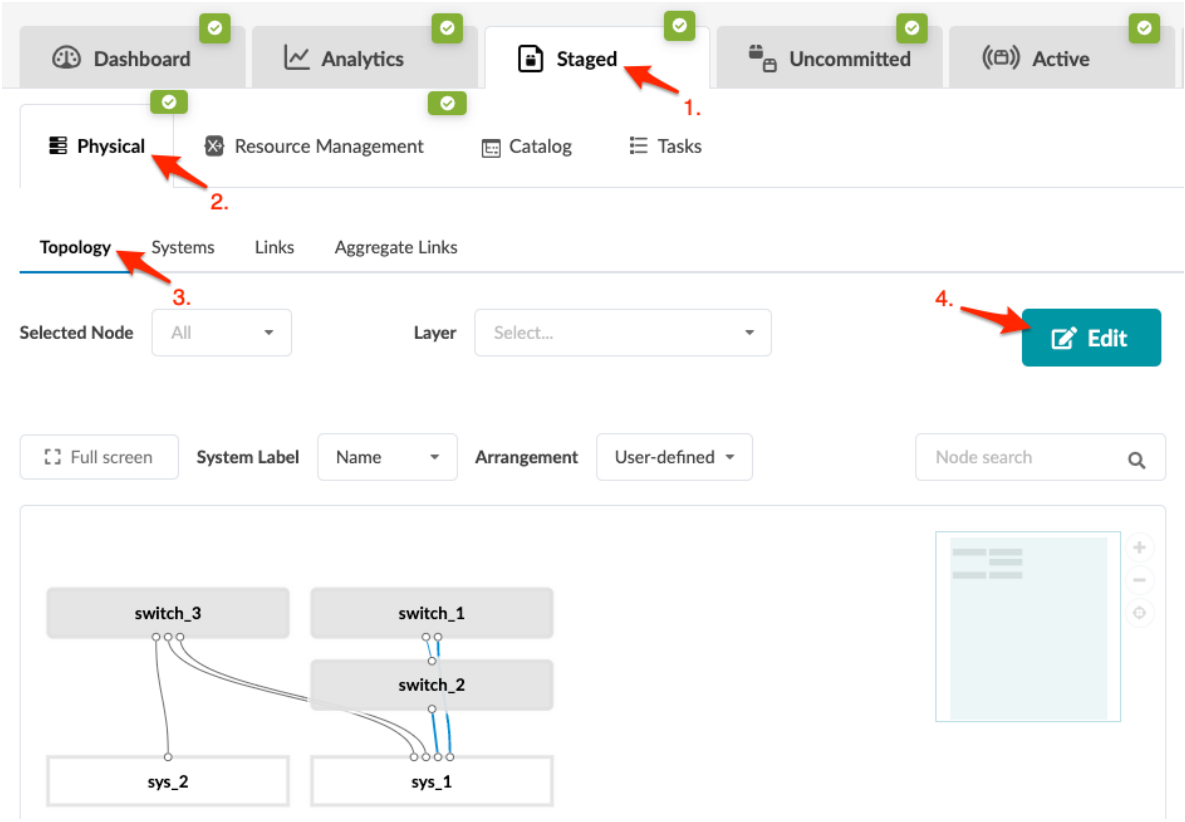
SUMMARY

You can change Freeform system ID (serial number) assignments from the **Topology** or **Systems** view.

IN THIS SECTION

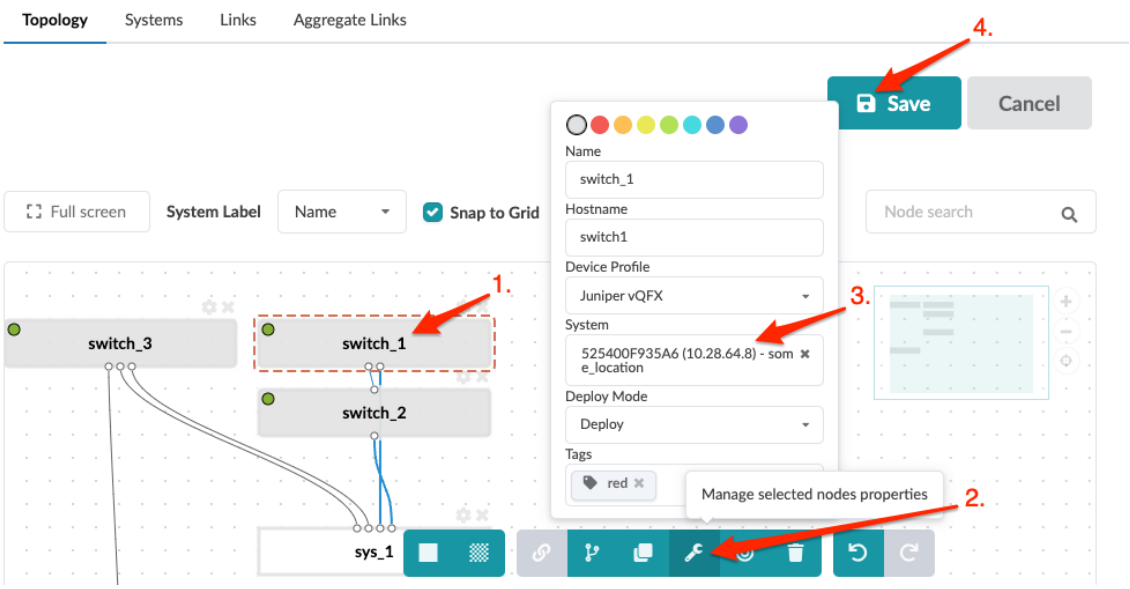
Update One System ID Assignment (from Topology)

1. From the blueprint, navigate to **Staged > Physical > Topology** and click **Edit** to open the topology editor.



> ⚠ **CAUTION**: Be careful. If you click away from the topology editor after making changes without clicking **Save**, your changes are discarded.

2. In the topology editor, click the system to change, then click the **Manage selected nodes properties** button that becomes available. (You can also open the same dialog by clicking the settings button for the selected system. It's the gear at the top-right of the system.)



3. Select the new system ID from the **System** drop-down list, or click **x** to remove an existing assignment.

   The list includes managed devices that haven't been assigned yet. If you have your devices ready and they're not appearing in this list, you still need to bring them under Apstra management by adding them to Managed Devices.

4. To close the dialog, click anywhere on the *canvas* outside of the dialog.

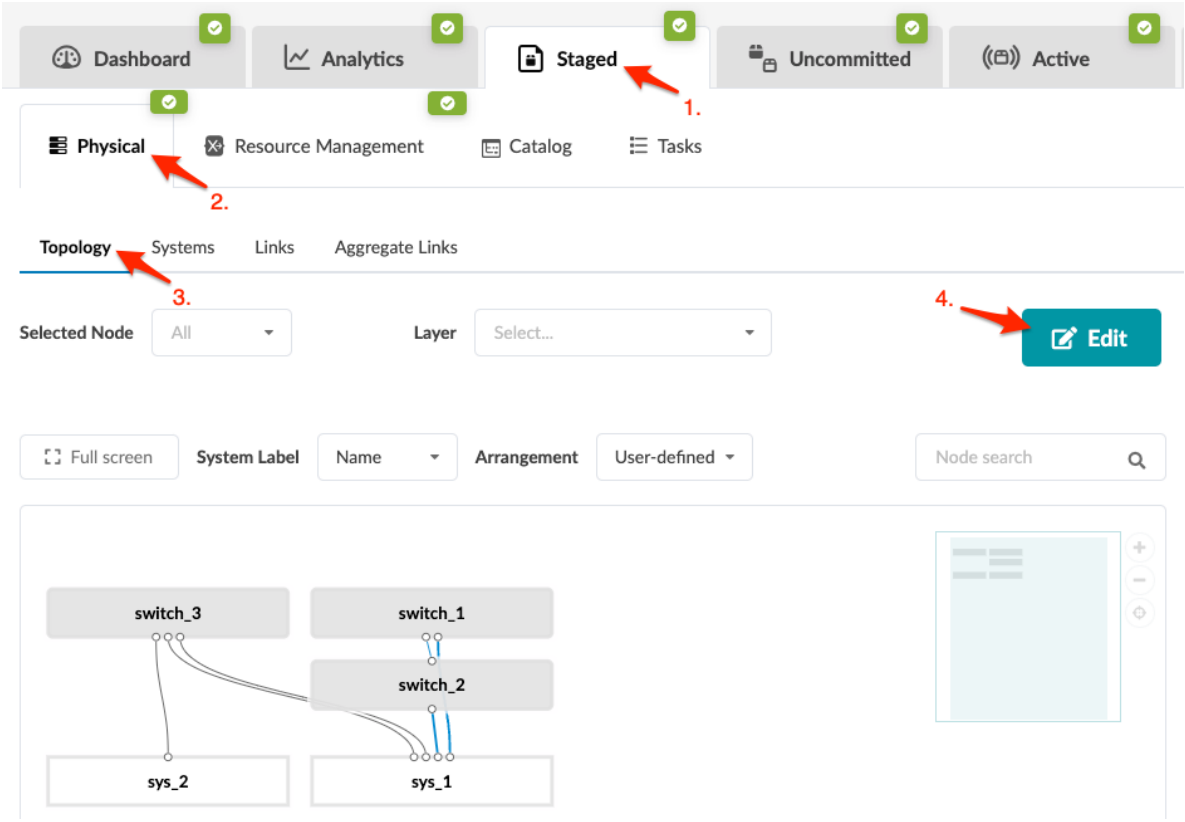
5. Click **Save** to stage your changes, exit the topology editor and return to the **Topology** view. (If you leave the page without saving, your changes are discarded.)

   If you've removed an assignment, the device is still under Apstra management. It's ready and available to be assigned to any blueprint. To remove the device completely from Apstra management, "remove the device from Managed Devices" on page 673.

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

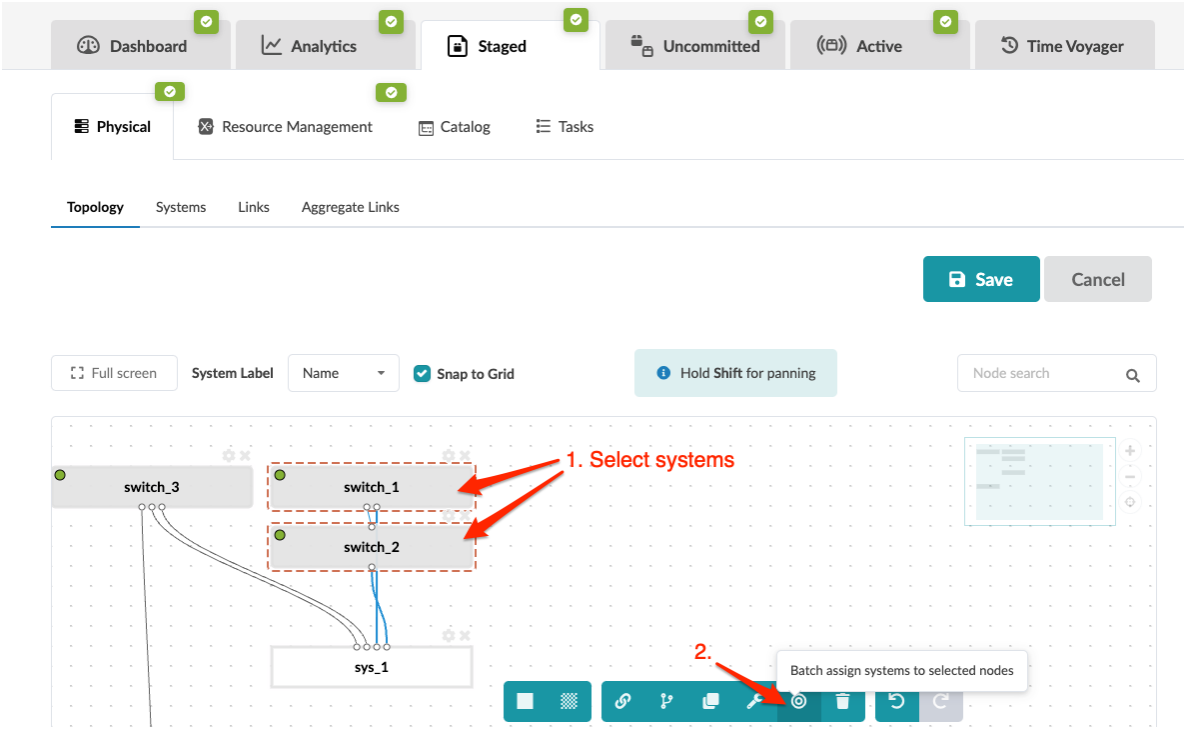
Update One or More System ID Assignments (from Topology)

1. From the blueprint, navigate to **Staged > Physical > Topology** and click **Edit** to open the topology editor.



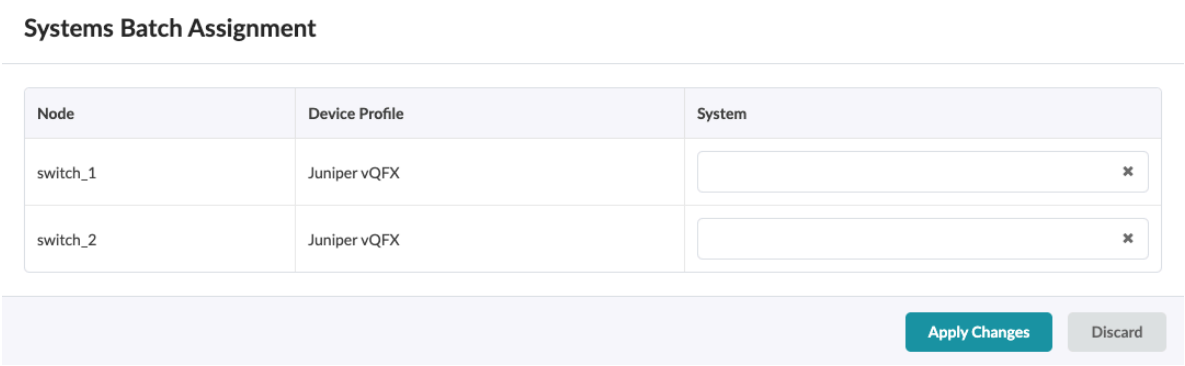
> **CAUTION**: Be careful. If you click away from the topology editor after making changes without clicking **Save**, your changes are discarded.

2. Select the systems to update, then click the **Batch assign systems to selected nodes** button that becomes available.



3. Select system IDs from the **System** drop-down lists, or click the **x** to remove an existing assignment..

   The list includes managed devices that haven't been assigned yet. If you have your devices ready and they're not appearing in this list, you still need to bring them under Apstra management by adding them to Managed Devices.



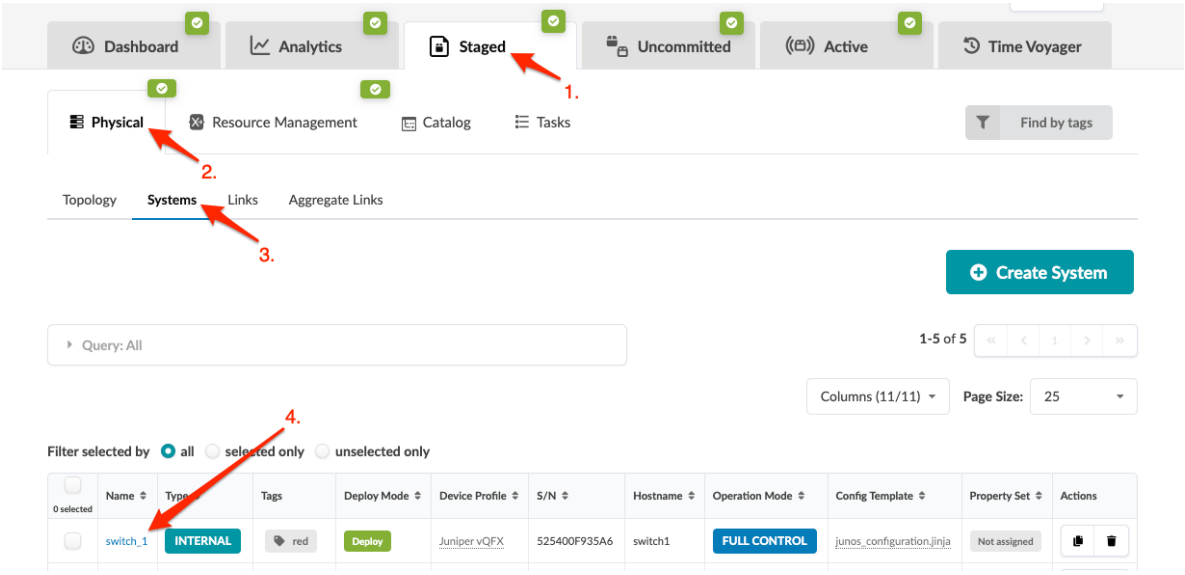
4. Click **Apply Changes** to apply the changes or, if you decide not to keep the changes, click **Discard**.

   If you've removed an assignment, the device is still under Apstra management. It's ready and available to be assigned to any blueprint. To remove the device completely from Apstra management, "remove the device from Managed Devices" on page 673.

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

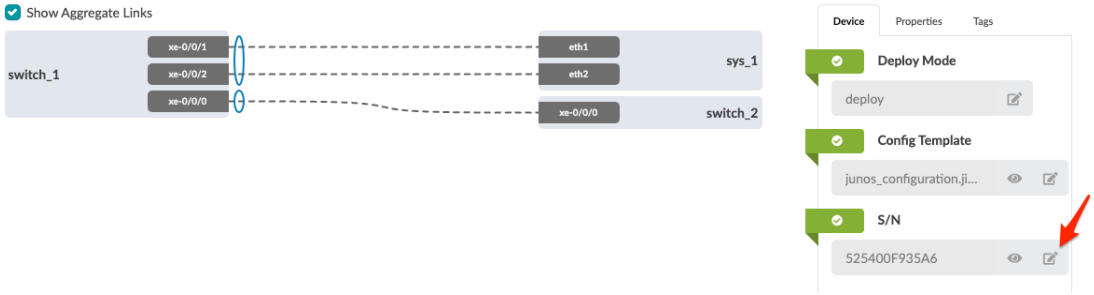
Update One System ID Assignment (from Systems)

1. From the blueprint, navigate to **Staged > Physical > Systems** and click the system name to go to details for that system.



> **NOTE**: You can also get to the Systems details page from the **Topology** view. From the blueprint, navigate to **Staged > Physical > Topology** and select the system to update.

2. In the panel on the right, in the **Devices** tab, click the **Edit** button for the **S/N** field.



3. Select the new system ID from the **System** drop-down list, or click the red box to remove an existing assignment.

   The list includes managed devices that haven't been assigned yet. If you have your devices ready and they're not appearing in this list, you still need to bring them under Apstra management by adding them to Managed Devices.

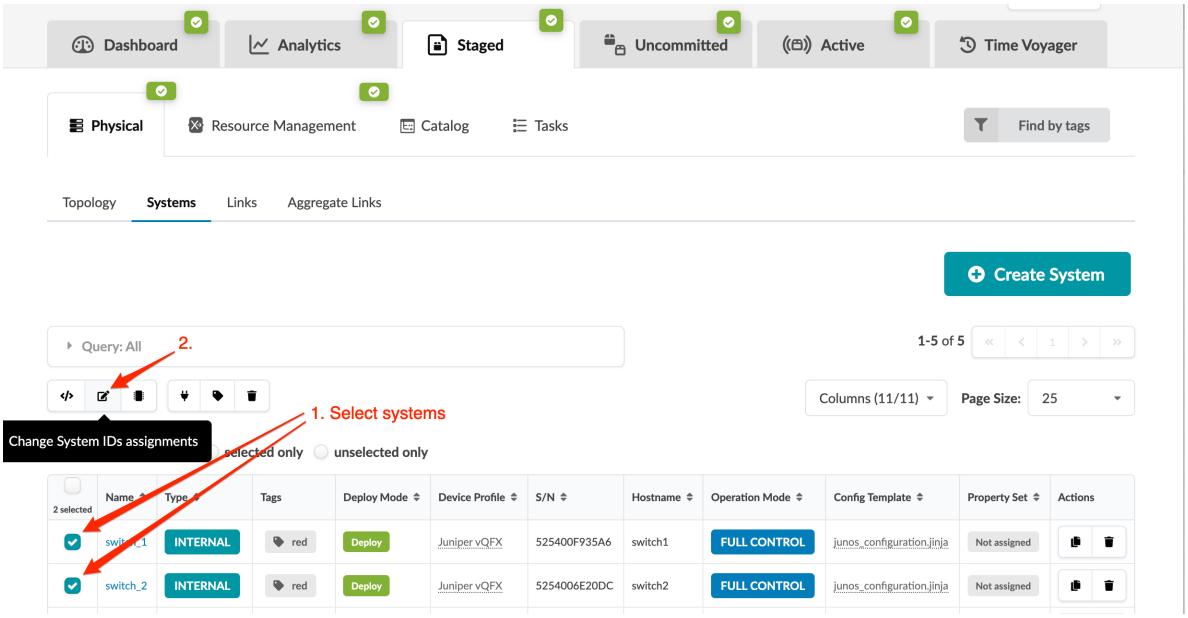
4. If you're going to deploy the device, make sure the deploy mode is set to **Deploy**, then save it. If you're removing an assignment, update the deploy mode to **Undeploy**, then save it.
5. Click the **Save** button to stage your changes.

If you've removed an assignment, the device is still under Apstra management. It's ready and available to be assigned to any blueprint. To remove the device completely from Apstra management, "remove the device from Managed Devices" on page 673.

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

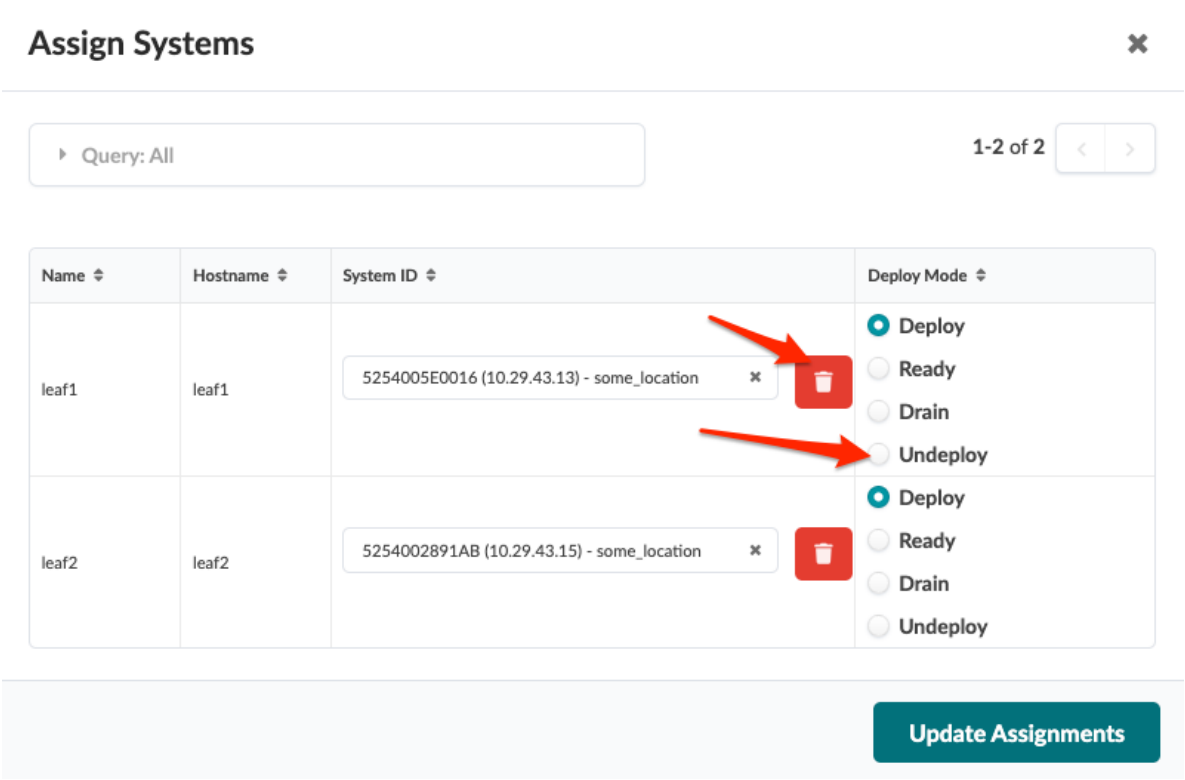
Update One or More System ID Assignments (from Systems)

1. From the blueprint, navigate to **Staged > Physical > Systems** and select the check boxes for one or more systems, then click the **Change System IDs assignments** button.



2. In the dialog that opens, select new system IDs from the **System** drop-down lists, or click the red trash can button to remove an existing assignment. If you're removing an assignment, go ahead and update the deploy mode to **Undeploy** as well.

   The list includes managed devices that haven't been assigned yet. If you have your devices ready and they're not appearing in the lists, you still need to bring them under Apstra management by adding them to Managed Devices.



3. Click **Update Assignments** to stage your changes and return to the **Systems** view.

   If you've removed an assignment, the device is still under Apstra management. It's ready and available to be assigned to any blueprint. To remove the device completely from Apstra management, "remove the device from Managed Devices" on page 673.

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Update Deploy Mode (Freeform)

SUMMARY

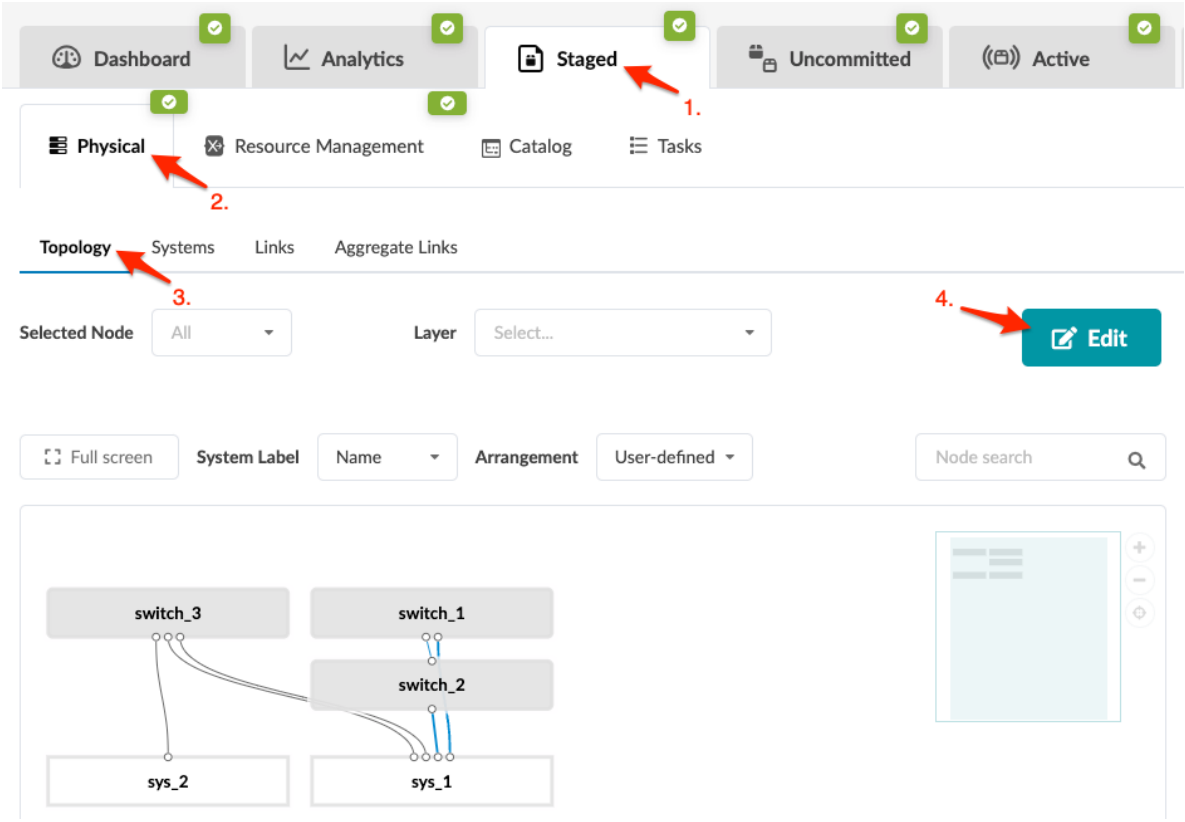
You can update system deploy modes from the **Topology** or **Systems** view.

IN THIS SECTION

NOTE: When you set the deploy mode on a system, it appears in its **Device Context**. But if you haven't added `deploy_mode` (as a Jinja variable) to the config template that's assigned to that system, it has no effect on the rendered configuration.

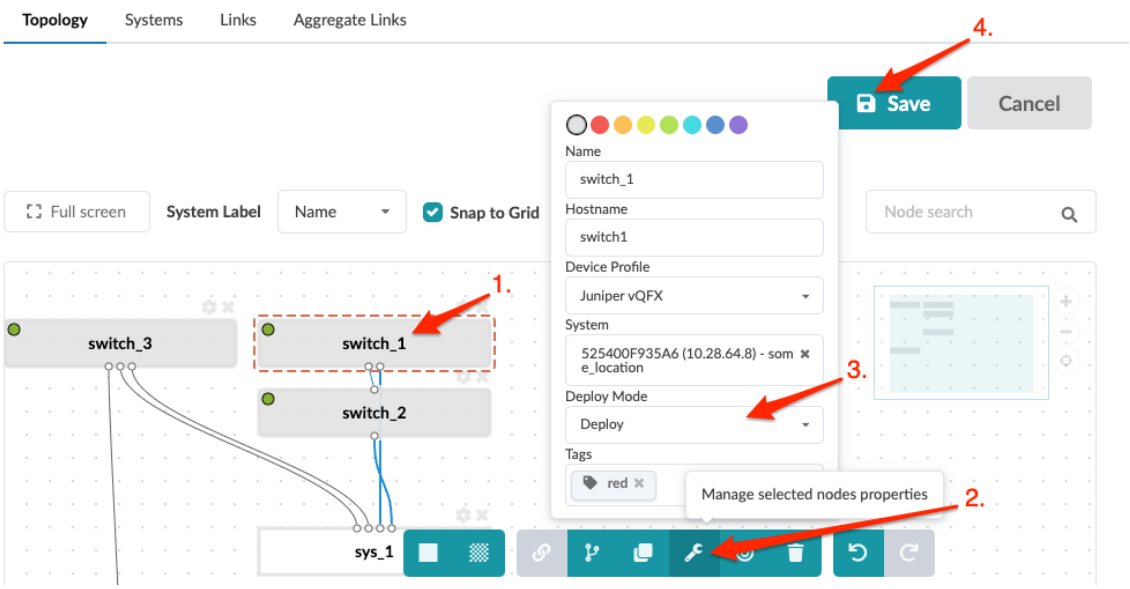
Update Deploy Mode on One or More Systems (from Topology)

1. From the blueprint, navigate to **Staged > Physical > Topology** and click **Edit** to open the topology editor.



CAUTION: Be careful. If you click away from the topology editor after making changes without clicking **Save**, your changes are discarded.

2. In the topology editor, click one or more systems to change, then click the **Manage selected nodes properties** button that becomes available. (You can also open the same dialog by clicking the settings button for the selected system. It's the gear at the top-right of the system.)

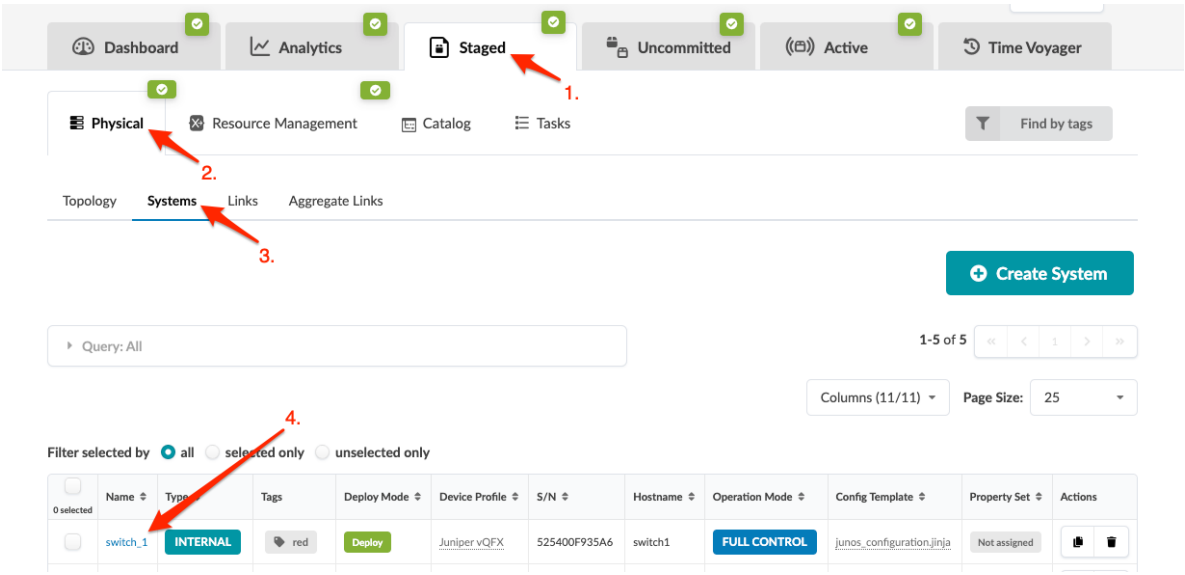


3. Select the new deploy mode from the **Deploy Mode** drop-down list. The changes apply to all selected systems.
4. To close the dialog, click anywhere on the *canvas* outside of the dialog.
5. Click **Save** to stage your changes, exit the topology editor and return to the **Topology** view. (If you leave the page without saving, your changes are discarded.)

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

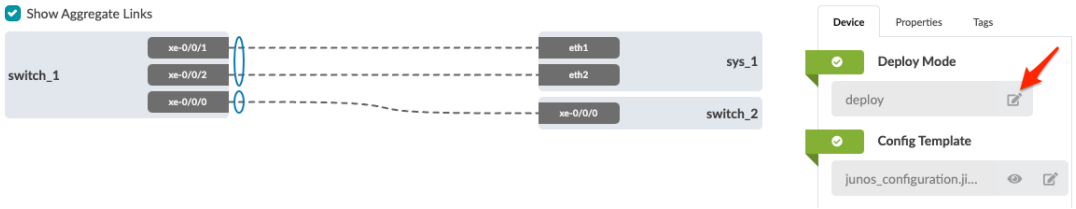
Update Deploy Mode on One System (from Systems)

1. From the blueprint, navigate to **Staged > Physical > Systems** and click the system name to go to details for that system.



NOTE: You can also get to the Systems details page from the **Topology** view. From the blueprint, navigate to **Staged > Physical > Topology** and select the system to update.

2. In the **Devices** panel (on the right side), click the **Edit** button for the **Deploy Mode** field.



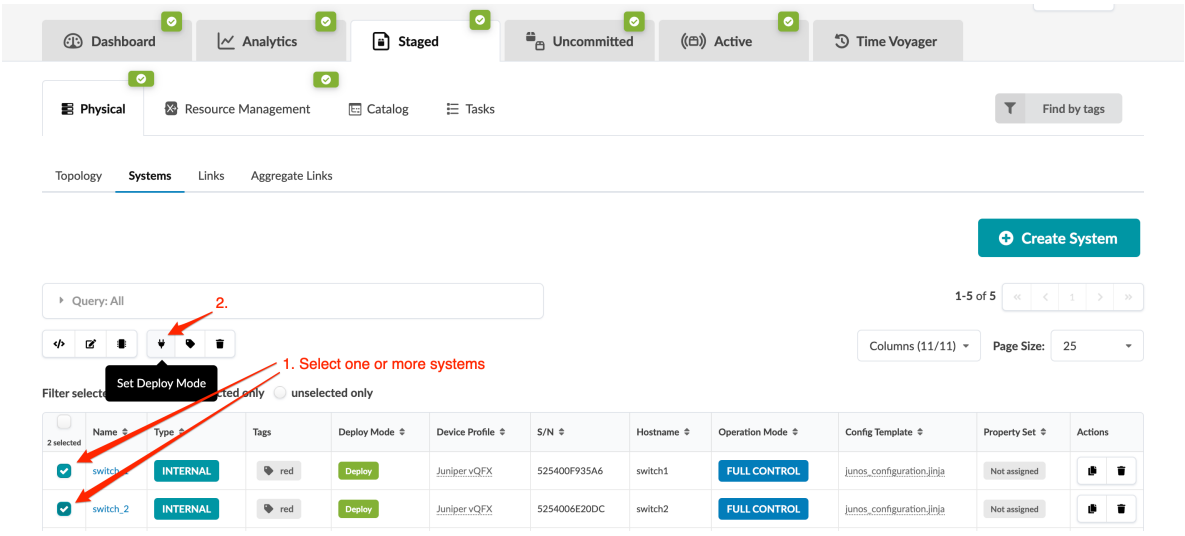
3. Select the deploy mode (deploy, ready, drain, undeploy), then click the **Save** button to stage your changes.

Internal systems with deploy mode set to **Deploy** require an assigned config template. Make sure the config template assigned to the device includes `deploy_mode` or your changes will have no effect on configuration.

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Update Deploy Mode on One or More Systems (from Systems)

1. From the blueprint, navigate to **Staged > Physical > Systems** and select the check boxes for one or more systems, then click the **Set Deploy Mode** button.



2. In the dialog, select the deploy mode (deploy, ready, drain, undeploy) for the selected systems.
3. Click **Set Deploy Mode** to stage the changes and return to the **Systems** view.

Internal systems with deploy mode set to **Deploy** require an assigned config template. Make sure the config template assigned to the device includes `deploy_mode` or your changes will have no effect on configuration.

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Add / Remove Tags on System (Freeform)

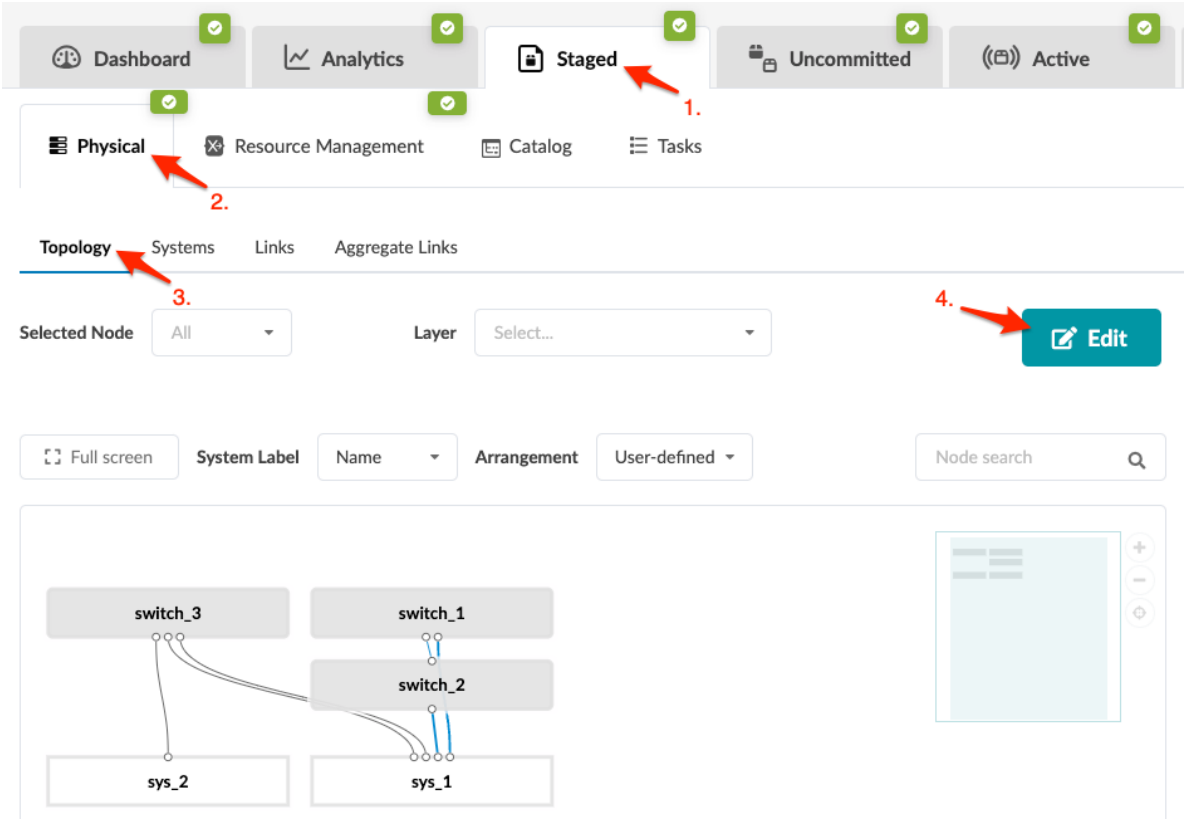
SUMMARY

You can update Freeform system tag assignments from the **Topology** or **Systems** view.

IN THIS SECTION

Update Tags on One or More Systems (from Topology)

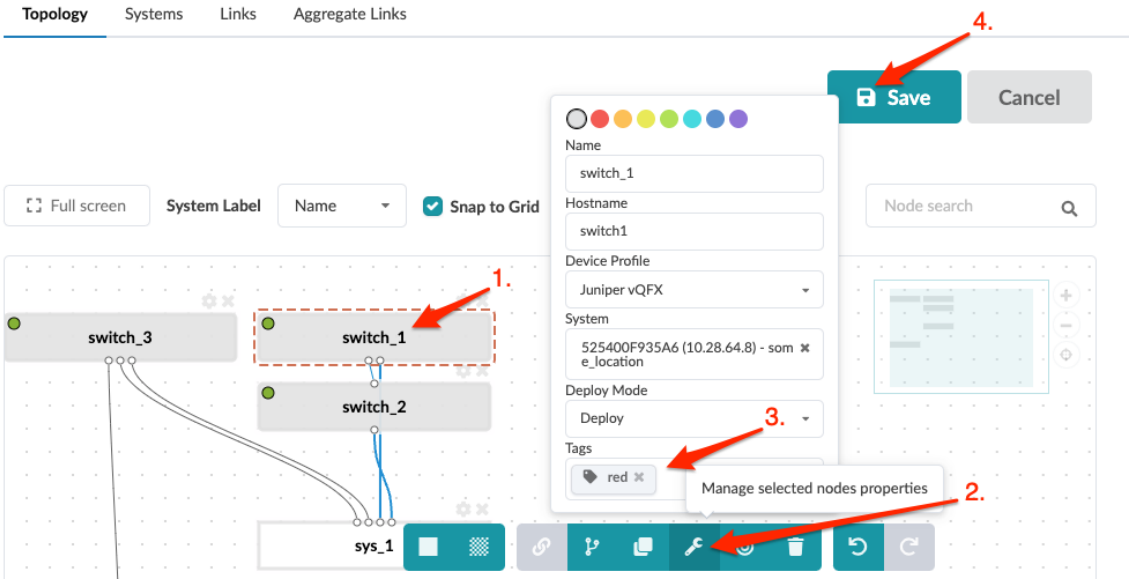
1. From the blueprint, navigate to **Staged > Physical > Topology** and click **Edit** to open the topology editor.



CAUTION: Be careful. If you click away from the topology editor after making changes without clicking **Save**, your changes are discarded.

2. In the topology editor, click one or more systems to change, then click the **Manage selected nodes properties** button that becomes available. (You can also open the same dialog by clicking the settings

button for one selected system. It's the gear at the top-right of the system.)

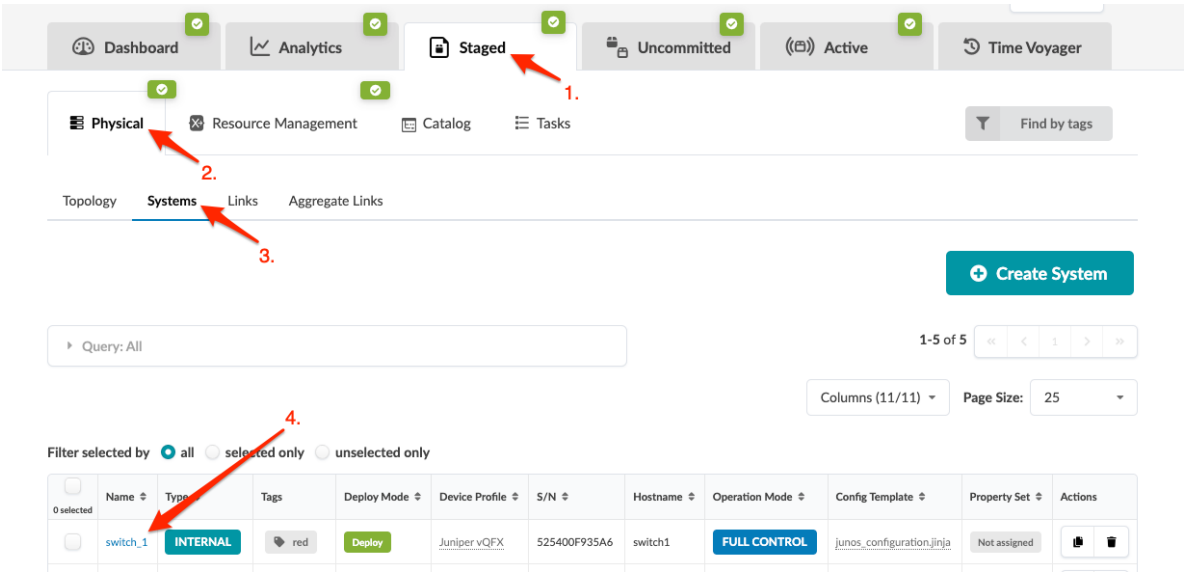


3. Add and remove tags in the **Tag** field, as needed. The changes apply to all selected systems.
4. To close the dialog, click anywhere on the *canvas* outside of the dialog.
5. Click **Save** to stage your changes, exit the topology editor and return to the **Topology** view. (If you leave the page without saving, your changes are discarded.)

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Update Tags on One System (from Systems)

1. From the blueprint, navigate to **Staged > Physical > Systems** and click the system name to go to details for that system.



> **NOTE**: You can also get to the Systems details page from the **Topology** view. From the blueprint, navigate to **Staged > Physical > Topology** and select the system to update.

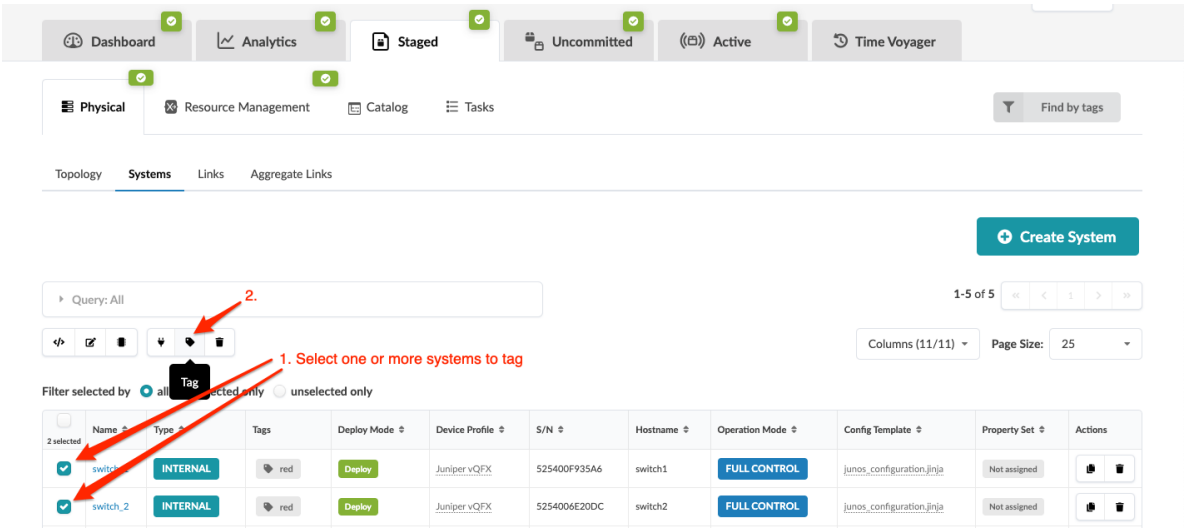
2. Click the **Tags** tab in the right panel, then in the dialog that opens add and/or remove tags, as needed.

3. Click **Update Tags** to update tags for that system and return to the selection view.

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Update Tags on One or More Systems (from Systems)

1. From the blueprint, navigate to **Staged > Physical > Systems** and select the check boxes for one or more systems, then click the **Tag** button that becomes available above the table.



2. In the dialog that opens add and/or remove tags, as needed.
3. Click **Add/Remove Tags** to update tags for the selected systems and return to the table view.

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Delete System (Freeform)

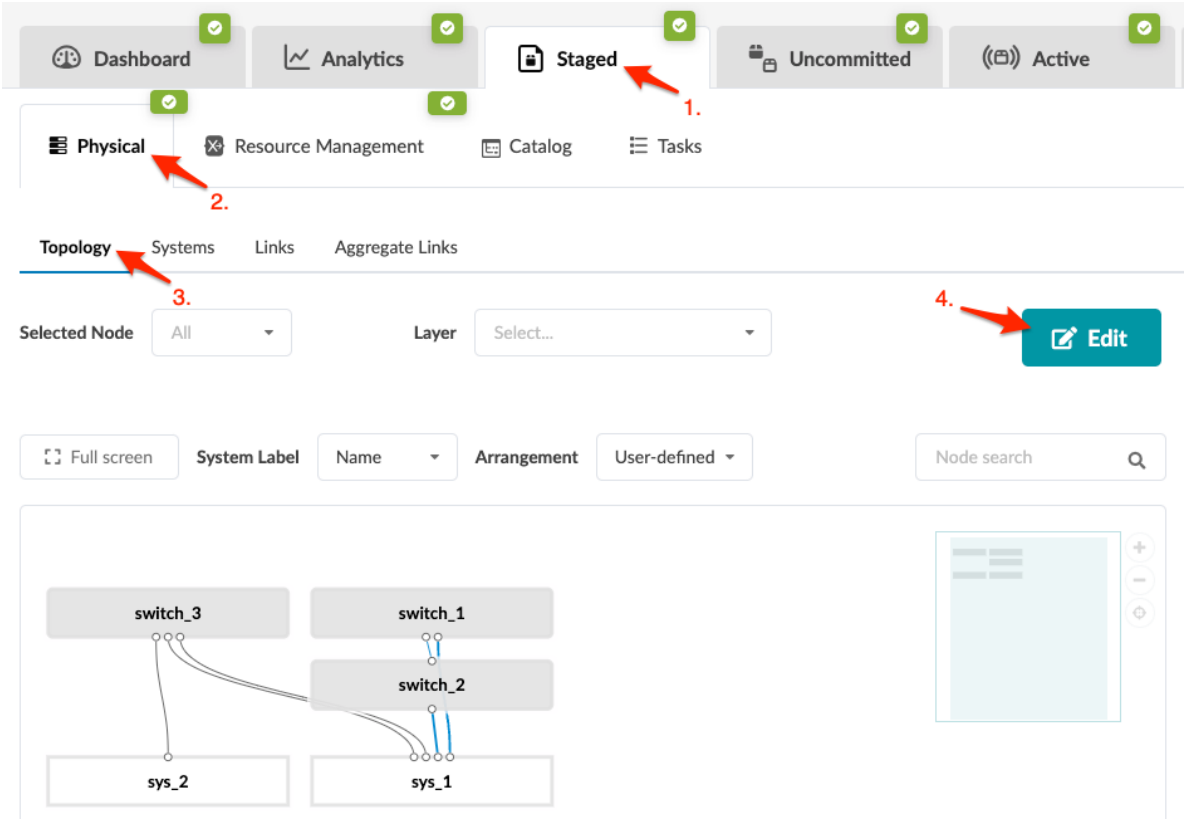
SUMMARY

You can delete Freeform systems from the **Topology** or **Systems** view.

IN THIS SECTION

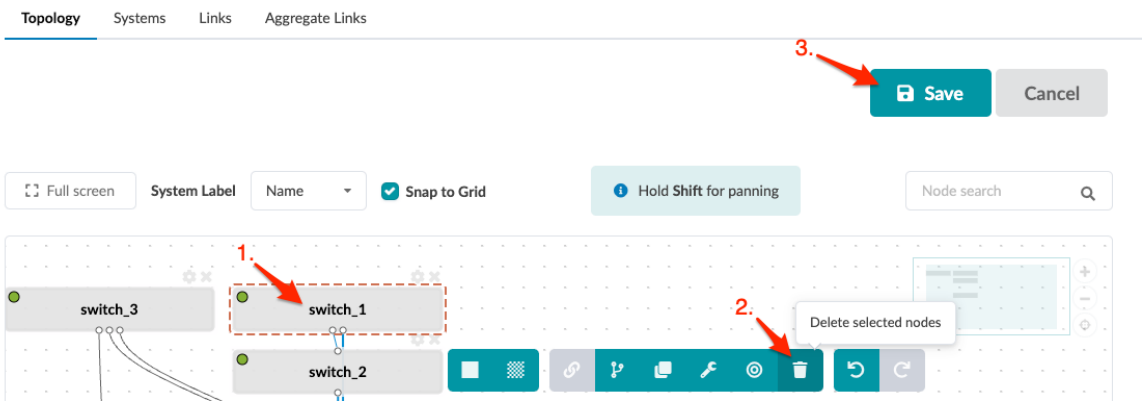
Delete One or More Systems (from Topology)

1. From the blueprint, navigate to **Staged > Physical > Topology** and click **Edit** to open the topology editor.



> ⚠ **CAUTION**: Be careful. If you click away from the topology editor after making changes without clicking **Save**, your changes are discarded.

2. In the topology editor, select one or more systems to delete and click the **Delete selected nodes** button that becomes available.

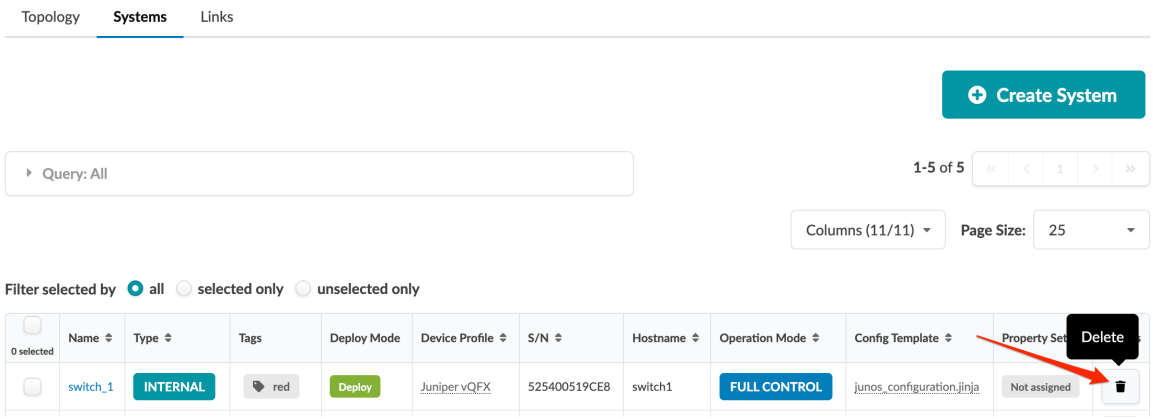


3. Click **Save** to stage your changes, exit the topology editor and return to the **Topology** view.

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Delete One System (from Systems)

1. From the blueprint, navigate to **Staged > Physical > Systems** and click the **Delete** button for the system to delete.

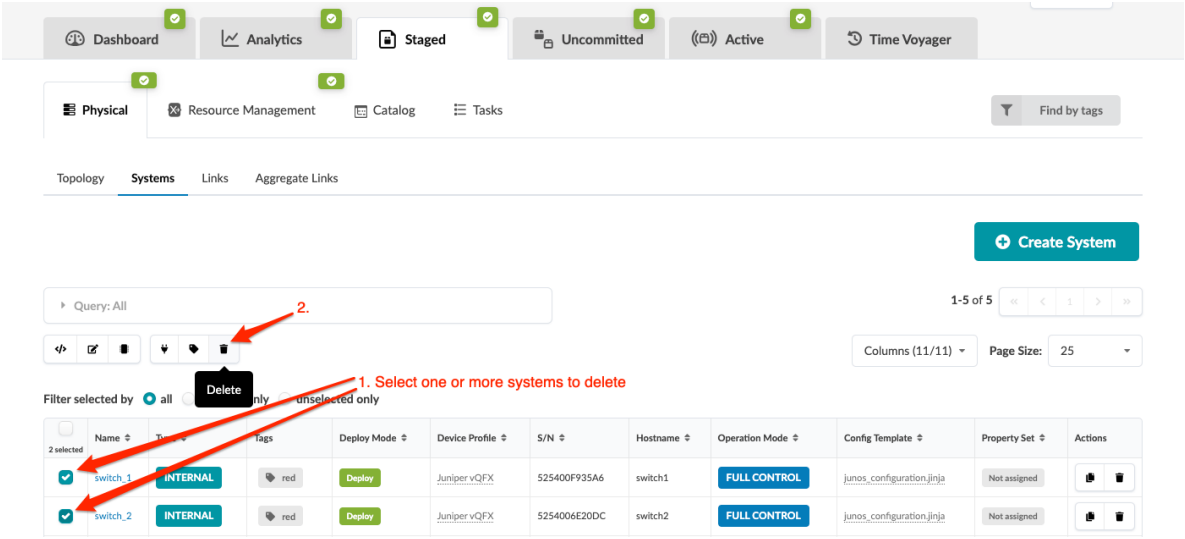


2. Click **Delete** to stage the deletion and return to the **Systems** view.

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Delete One or More Systems (from Systems)

1. From the blueprint, navigate to **Staged > Physical > Systems** and select the check boxes for one or more systems, then click the **Delete** button that becomes available above the table.



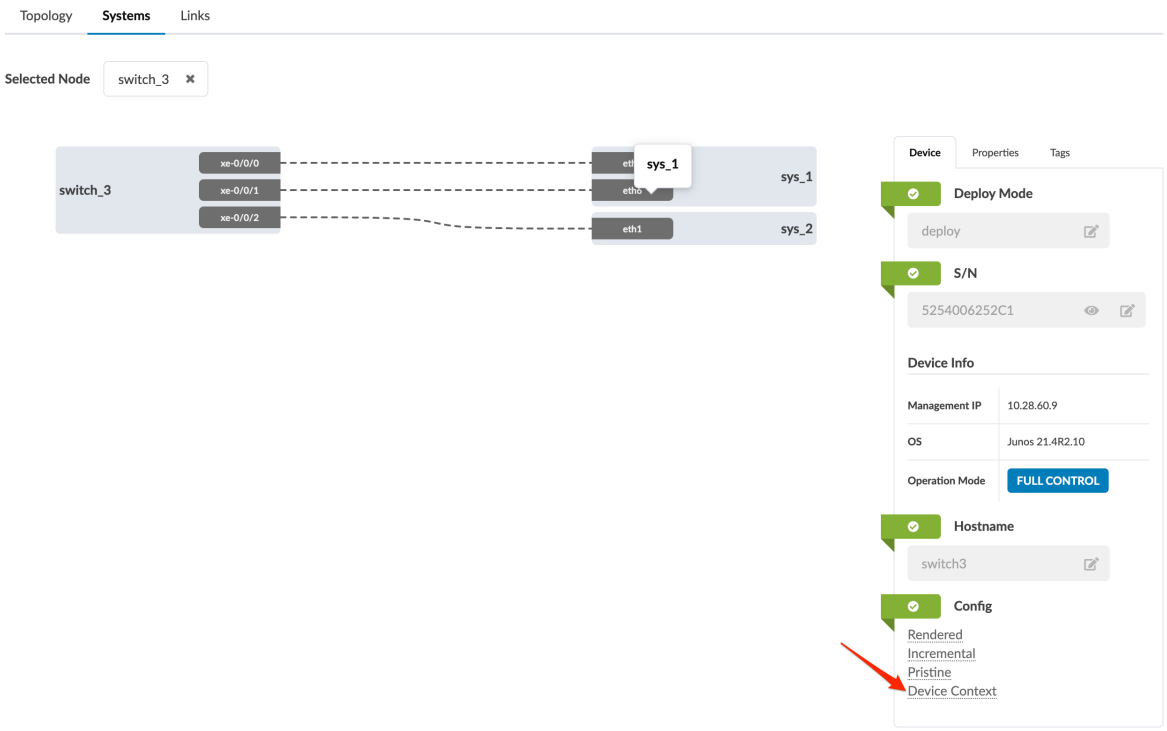
2. Click **Delete** to delete the systems (and any links that are connected to the systems) and return to the **Systems** view.

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

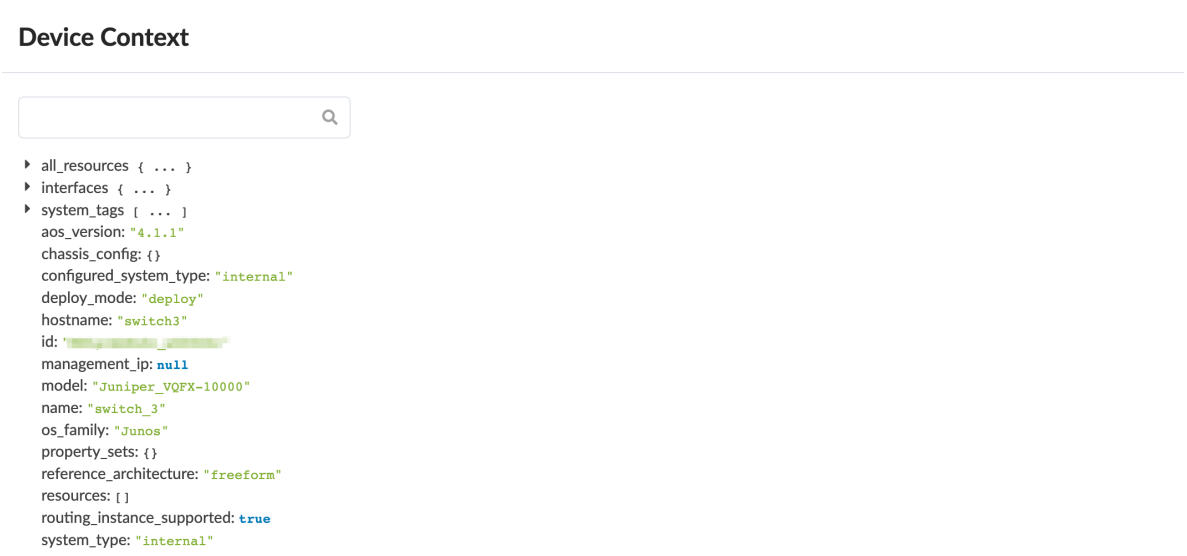
Device Context (Freeform)

The device context includes all the contextual data that you can use when creating dynamic Jinja config templates. It includes such data as interfaces, IP addresses, prefix lengths, name, and state. It also shows you what the neighbor interface is of other devices. You can search for data in a query box to pinpoint the information you're looking for.

1. From the blueprint, either from the **Topology** view or the **Systems** view, click the name of the system to view. Its details appear in the **Systems** view.



2. At the bottom of the device panel on the right, click **Device Context** to go to device context for the device.

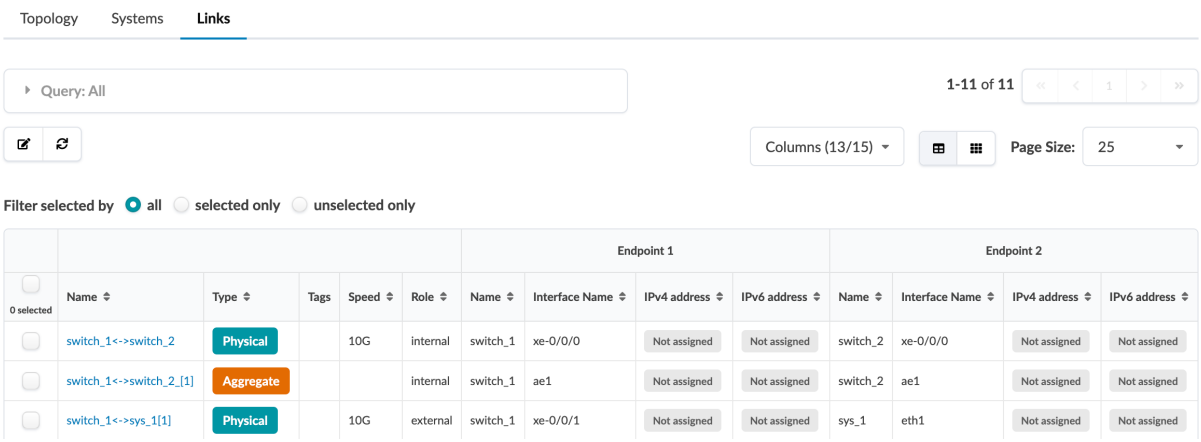


Links

IN THIS SECTION

Links (Freeform)

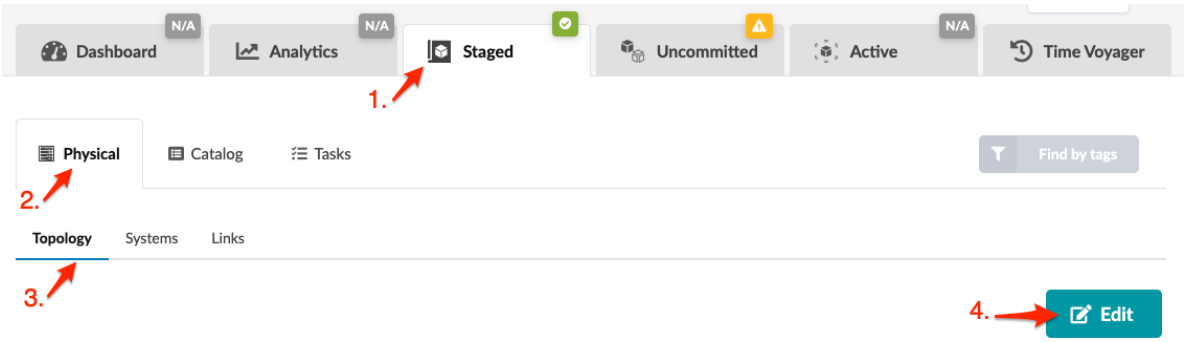
The **Links** view (Staged > Physical > Links) shows all the links that connect your devices together. The table includes information about endpoint names, link type, tags, speed, role, interface names and IP addresses. You can customize what appears in the table by selecting/deselecting elements in the columns drop-down list. You can perform various tasks from the **Links** view as described in later sections.



Add Link (Freeform)

After you've created systems you can link them to each other from the **Topology** view.

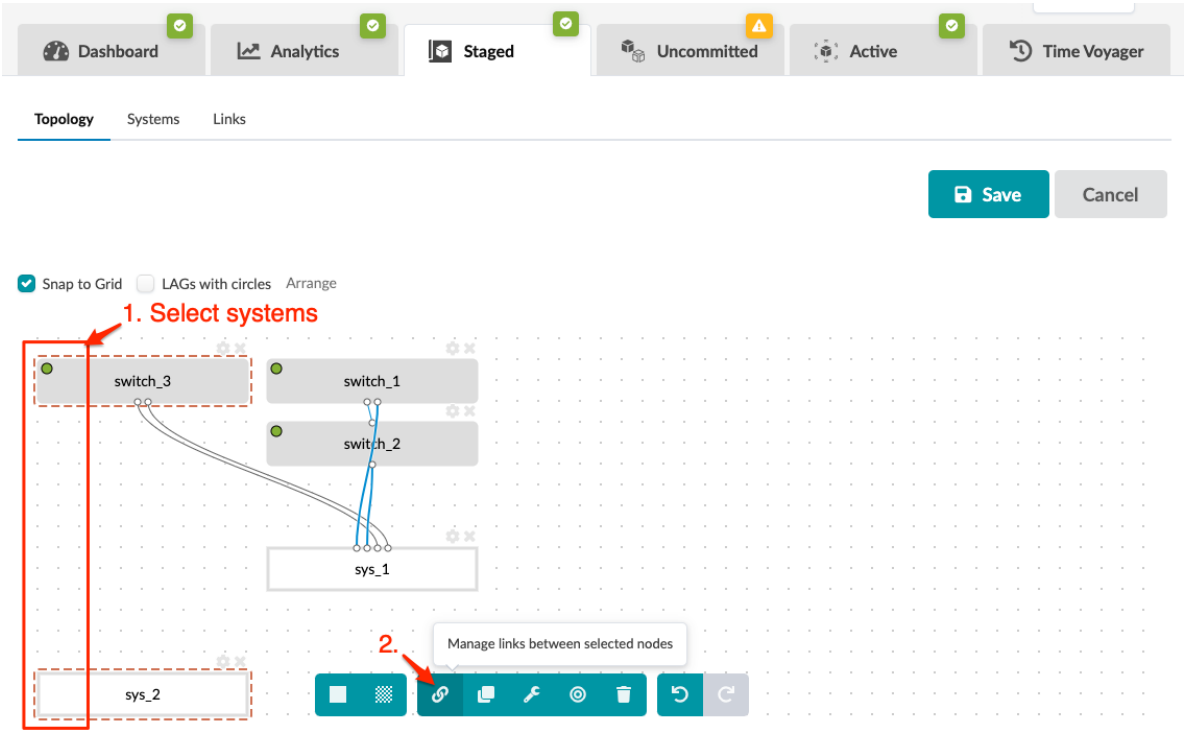
1. From the blueprint, navigate to **Staged > Physical > Topology** and click **Edit**.



> ⚠ **CAUTION**: Be careful. If you click away from the topology editor without clicking **Save**, your changes are discarded.

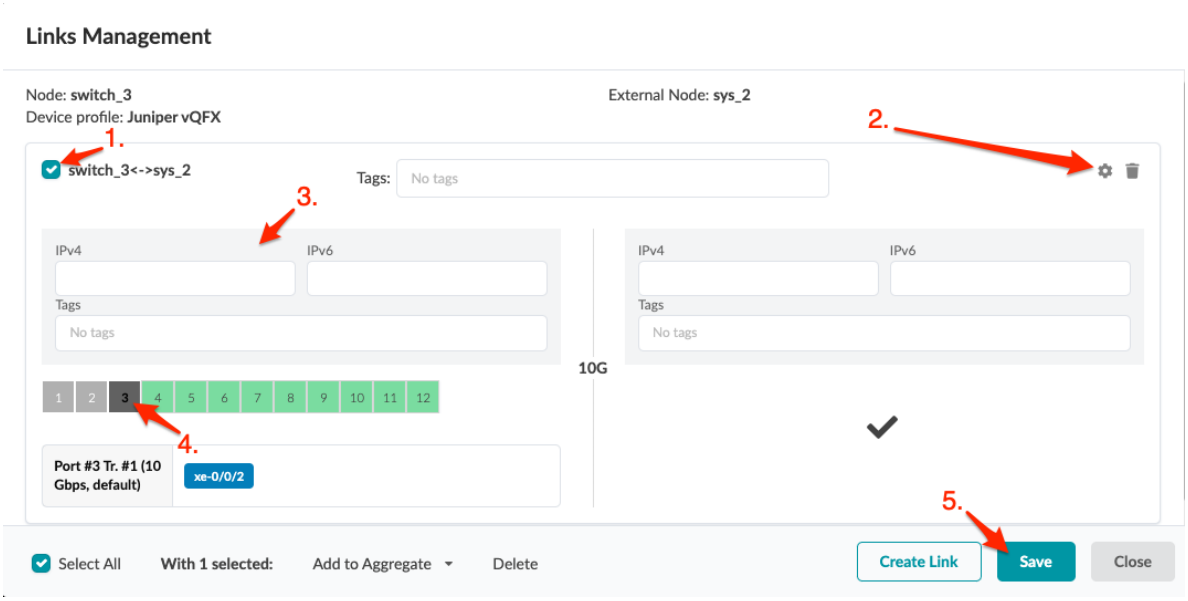
2. In the topology editor select the two systems that you want to link. You can select them in a couple of different ways:

   - Click and drag across the two systems.
   - Hold down the **alt** key (**command** key on a Mac) while clicking the two systems.



When you select two systems additional tasks become available in the context-aware menu at the bottom.

3. Click the **Manage links between selected nodes** button. The **Links Management** dialog opens showing the two node names (and device profiles, as applicable).
4. Click **Create Link**. The port representations appear.



5. Select the check box for the first node.
6. Click the gear (upper-right) to show fields for IPv4, IPv6 and tags.
7. You can enter IP addresses and/or tags to add them to the device model which can be used later when creating config templates.
8. Select ports (and transformations as applicable), then click **Save**. (If you're connecting to an external system as in the example screenshot, you won't select ports.) You're still in the topology editor and if you click away without saving, your changes are discarded.
9. Click **Save** in the topology editor to save your changes and leave the topology editor. (Depending on the size of your topology, you may need to scroll to see the **Save** button.)

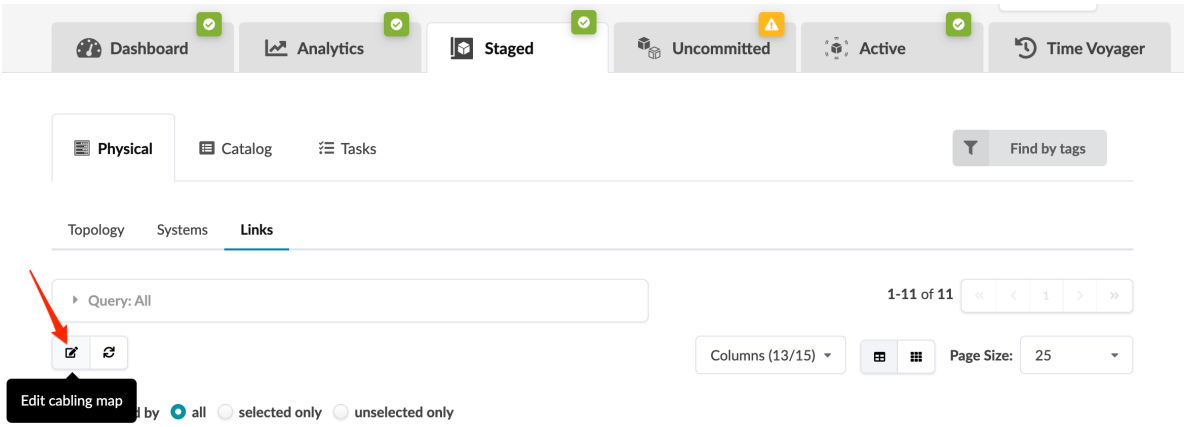
Next Steps:

If you haven't "created config templates" on page 578 yet, create them now. If you have config templates ready for your devices and haven't assigned them yet, "assign" on page 521 them now. When you've assigned all required config templates and all other requirements are met, you can deploy your blueprint from the **Uncommitted** tab.

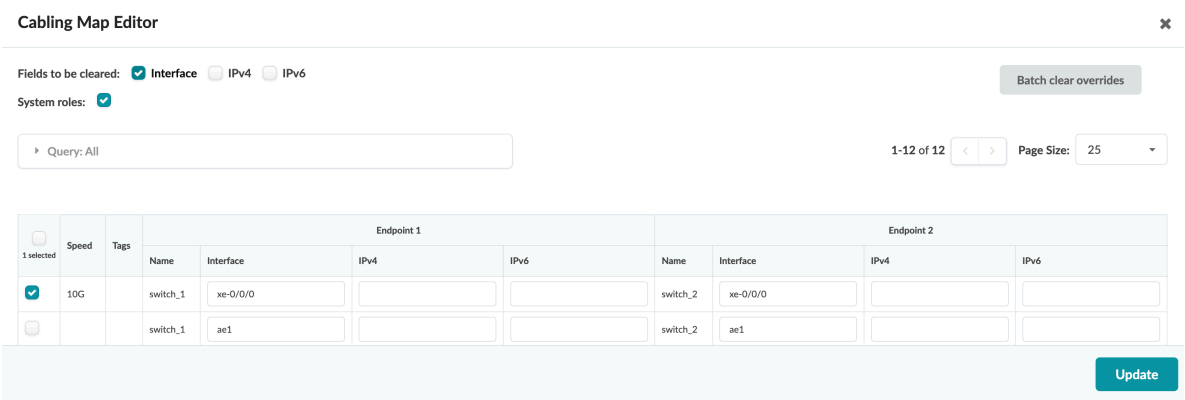
Edit Cabling Map (Freeform)

You can change one or more interfaces and IP addresses in the cabling map editor.

1. From the blueprint, navigate to **Staged > Physical > Links** and click the **Edit cabling map** button.



2. In the cabling map editor, change interface names and/or IP addresses, as applicable.
   - You can use **Batch clear override** to clear all interfaces and IPv4/IPv6 values for selected links.
   - To drop the override for either an interface name or IPv4/IPv6 address, submit an empty value in the corresponding field.



3. Click **Update** to stage your changes and return to the **Links** view.

Next Steps:

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

Fetch LLDP Data (Freeform)

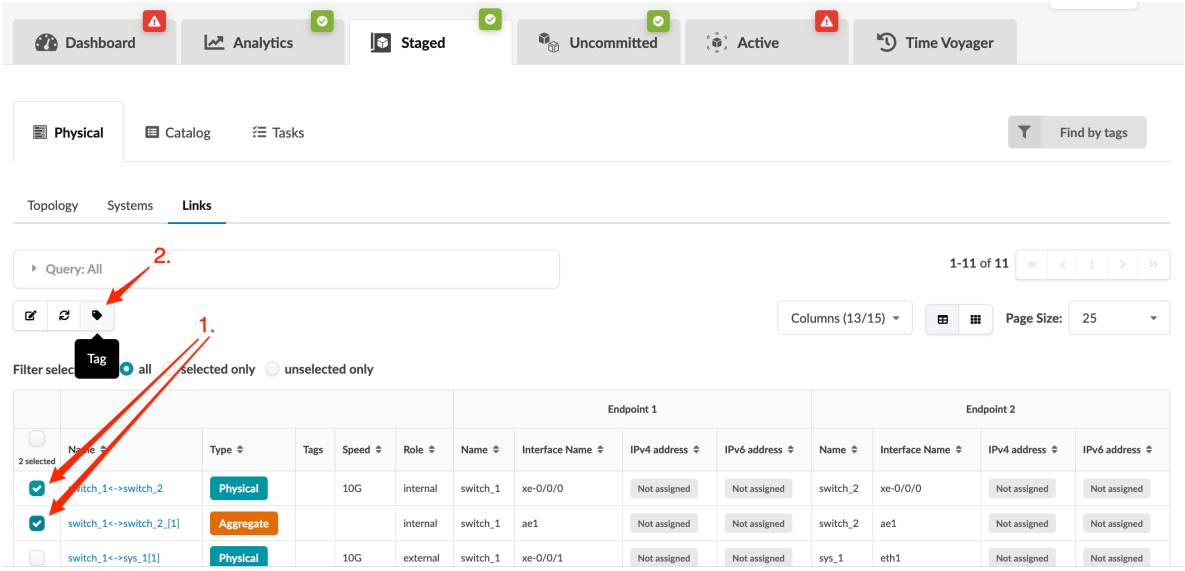
If you've already cabled up your devices, you can have Apstra discover your existing cabling instead of using the cabling map prescribed by Apstra. All system nodes in the blueprint must have system IDs assigned to them.

CAUTION: This is a disruptive operation. All links can potentially be renumbered.

1. From the blueprint, navigate to **Staged > Physical > Links** and click the **Fetch discovered LLDP data** button (second of two buttons above links list).
2. If staged data is *identical* to LLDP discovery results, you will see a message with that statement. Your actual cabling matches the Apstra cabling map. No further action is needed.
3. If staged data is *different* from LLDP discovery results, the message includes the number of links that are different.
4. Scroll to see details of the diffs (in red), or check the **Show only links with LLDP diff?** check box to see only the differences.
5. To accept the changes and update the map to match LLDP data, click **Update Stated Cabling Map from LLDP**.

Manage Link Tags (Freeform)

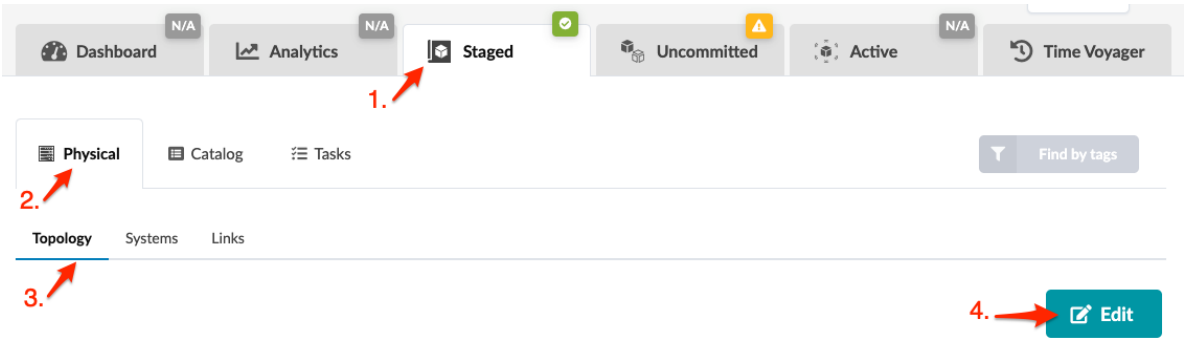
1. From the blueprint, navigate to **Staged > Physical > Links** and select one or more check boxes for the links to manage.



2. Click the **Tag** button that appears above the list after selecting link(s).
3. In the dialog, add and/or remove tags, as needed.
4. Click **Add/Remove Tags** to stage the changes and return to the **Links** view.

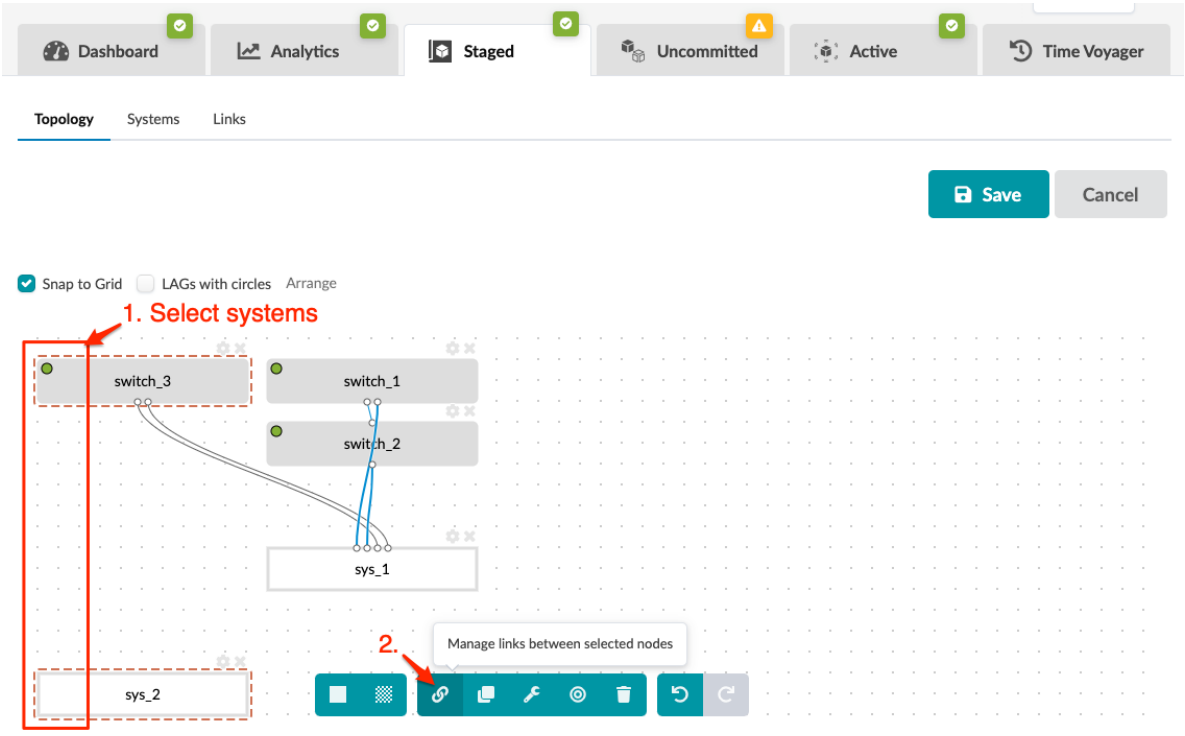
Delete Link (Freeform)

1. From the blueprint, navigate to **Staged > Physical > Topology** and click **Edit**.



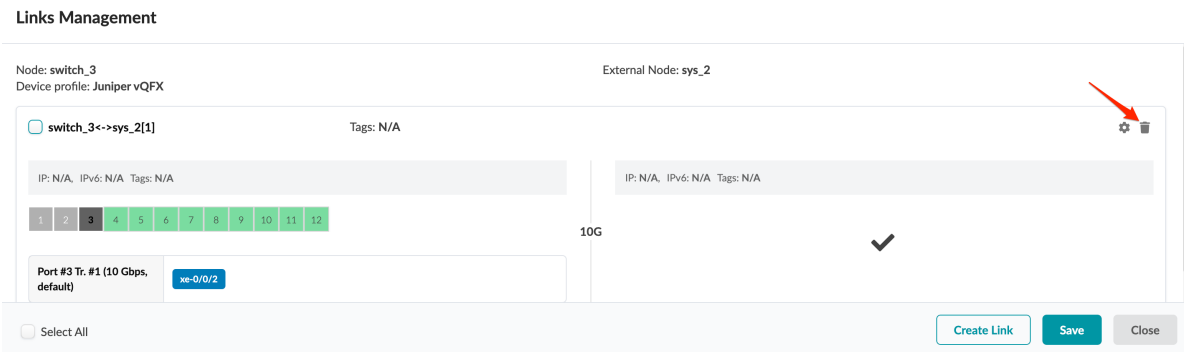
2. In the topology editor select the two systems where the link is that you want to delete. You can select them in a couple of different ways:

   - Click and drag across the two systems.
   - Hold down the **alt** key (**cmd** key on a Mac) while clicking the two systems.



   When you select two systems additional tasks become available in the context-aware menu at the bottom.

3. Click the **Manage links between selected nodes** button. The **Links Management** dialog opens showing the two node names (and device profiles, as applicable).



4. Click the **Delete** button for the link to delete, then click **Save**. You're still in the topology editor and if you click away without saving, your changes are discarded.
5. Click **Save** (right-side) in the topology editor to stage your changes and return to the **Topology** view.

When you're ready to activate your changes, commit them from the **Uncommitted** tab.

CHAPTER 16

Resource Management

IN THIS CHAPTER

Resource Management Introduction (Freeform)

SUMMARY

Manage resources in Freeform blueprints from the **Resource Management** tab. Resources include IPv4 addresses, IPv6 addresses, ASNs, VNIs and integers that are used in VLANs.

IN THIS SECTION

Resource management in Freeform blueprints is similar to that in Datacenter blueprints. The difference is that with Datacenter the mechanism is set up for you, and with Freeform you're responsible for setting it up yourself. You can set it up so resources are assigned and unassigned automatically as needed, just like in the Datacenter reference design.

Resource Types

In Apstra, resources are values that are assigned to various elements of the network. Resources include the following types:

- IPv4 (including Host IPv4)

- IPv6 (including Host IPv6)
- ASN - (autonomous system number)
- VNI (virtual network identifier)
- VLAN (virtual local area network)
- Integer - used for pool type VLAN in local pools

Resource Groupings

Resources for Freeform blueprints are grouped and organized in the following ways:

Resource Pools

- consist of one or more ranges of resource values.
- contain one resource type (ASN, VNI, Integer, IPv4, or IPv6).
- are created in the global **Resources** catalog.
- can be used in one or more blueprints.
- are associated with allocation groups.

Allocation Groups

- consist of mappings to one or more resource pools.
- contain one resource type (ASN, VNI, Integer, IPv4, or IPv6).
- are created in the blueprint.
- are specific to one blueprint.
- provide the mechanism for pulling resources from pools and assigning them.

In the Datacenter reference design, templates determine the initial resource requirements. When you create a Datacenter blueprint (from a template) allocation groups are created automatically. Freefrom reference design doesn't use templates, so resource requirements can't be determined when you create a Freeform blueprint. You'll create them yourself in Freeform blueprints.

Groups (Folders)

- are folders that are organized into a directory.

- contain assigned resources (and resource generators, described below).
- are used to arrange resources in any combination you like.
- can be nested inside other groups.
- can contain more than one resource type per group.
- are created in the blueprint.
- are specific to one blueprint.
- can be created and deleted automatically as needed, using group generators (described below).
- All resources must reside within a group (or group generator) that you create (not directly in the built-in **Root** group).

Local Pools

- consist of one or more ranges of resource values.
- contain only resource type Integer.
- contain only pool type VLAN.
- are created in the blueprint.
- are specific to one blueprint.
- can be created and deleted automatically as needed, with local pool generators (described below).

Generators

Generators automatically create and delete groups, resources, or local pools, as applicable. The graph database returns a set of objects based on a set of conditions that you specify. These conditions define the scope of what is added and/or removed.

Group Generator

You can put all of your resources in one group (folder), but if your design is complex, it's easier to manage resources in multiple groups. You can organize resources in any group combination that makes sense for you. You probably want to have nested groups, and you might want to have a group for every system in your network. Creating groups manually is simple enough; just click the group that you want to put your new group in and give it a name. Then you'd populate the group with your resources, either manually, or automatically with resource generators (described later). But, if you have many systems and

you want a group for every system, creating each group manually is a lot of unnecessary work. You can automate this process with group generators.

To create a group generator, give it a name, then specify a scope based on how you want your groups to be created and managed. Our example of creating one group for every internal system uses the following scope:

```
node('system', system_type='internal', name='target')
```

This scope tells the graph database to find all internal systems and create a group for each one; and assign the applicable system name to each group. The state of the groups keeps in synch with the graph database as the fabric changes. If you subsequently delete a system, the group created for that system is also deleted. All resources in that group are released back to the pool they came from, ready to be re-used. Conversely, if you create a system after this group generator is created, a group for that system is automatically created (and if you created resource generators inside the group generator, resources are also allocated accordingly).

Resource Generator

When it matters what the value is, you can allocate a resource manually, but in most cases you'll want to automate the process with resource generators. Resource generators don't actually generate resources; they pull existing resources from resource pools via allocation groups, based on a specified scope. Before creating a resource generator create any resource pools and allocation groups that you'll need. Creating an allocation group is straightforward; give it a name and select one or more resource pools to include in the group.

Resource Generator in a Group

Resources must be inside a group (or group generator as described below) that you create. To put all resources generated from a resource generator in one group, select the group and create your resource generator from there.

To create a resource generator, give it a name, then specify a resource type, an allocation group, a subnet prefix length for IPv4 only, and a scope. For example, you might want a group to contain link IPs (/31 addresses) for the links between all internal systems (switches) . First, create any resource pools and allocation groups that you'll need. In the resource generator, specify resource type IPv4, an applicable allocation group, the subnet prefix length, and the following scope:

```
node('link', role='internal', name='target')
```

This scope tells the graph database to find all fabric-facing links. The generator specifies to create link IPs for them, and add them to the group. Resources are automatically generated or released as links are added or removed.

Resource Generator in a Group Generator

To put every generated resource in its own group automatically, you can put your resource generator inside a group generator. The resource generator inherits the scope of the group generator.

For example, to create a group for every system and put an ASN in each group, you'd select the group generator already created and create the resource generator from there. The resource generator inherits the scope from the group generator. In our example, the scope is:

```
node('system', system_type='internal', name='target')
```

The graph database finds every internal system, allocates an ASN to each one, then puts each ASN in the applicable group based on internal systems.

Multiple Resource Generators in a Group Generator

You can put multiple resource generators inside a group generator (or group). Let's continue our example that already has a group for every internal system and an ASN in every group. You might also want your internal system groups to include loopback IP addresses. You can create a resource generator for loopback IP addresses in the same group generator as for the ASNs; you'd just select resource type IPv4.

The process is the same as when you added the ASNs. From the same group generator as before create the resource generator.....

Select a group to put the resource in, give it a name, specify the resource type and select an allocation group to pull the resource from. Then you'll have a resource in the specified folder. **You can see the resource in the table and the allocation group** it was pulled from. You an see if it's been assigned yet. Initially, it won't be. (put this in the task doc)

Local Pool Generator

You can create and assign a specific VLAN ID to a specific system (node) in your blueprint. If it doesn't matter what the specific value is, you can create a generator that will dynamically create and delete VLAN IDs based on the conditions you set. Values will be pulled from these pools as needed. These pools are specific to each blueprint.

Config Templates

(Add resources to config templeates.)

Resource Management Workflow

1. Create resource pools ("ASNs" on page 928, "VNIs" on page 930, Integers, "IPv4 addresses" on page 934, "IPv6 addresses" on page 937) in the global **Resources** catalog. This is where you specify ranges of resource values.
2. "Create allocatio groups" on page 572 in the blueprint.

   This is where you specify one or more resource pools to be included in an allocation group. When you're ready to assign resources, you'll select resource pools from one of these allocation groups.
3. Plan how you'd like to organize your resources, then create "groups" on page 566 and "group generators" on page 567 in the blueprint, as applicable.
4. Create "resources" on page 566 and "resource generators" on page 570 in the blueprint, as applicable.
5. Create "local pools" on page 574 and "local pool generators" on page 575 in the blueprint, as applicable.
6. Assign resources. (Assigned Resources and Assigned groups is on the detailed system page). To render the correct configuration using these resources, you have to apply the resources to individual Jinja2 config templates. (Use the resources to render configurations by modifying the Jinja2 config templates from using property sets to resources. Does this still need to be done if I'm not converting property sets to resource management?)

Blueprint Resources

IN THIS SECTION

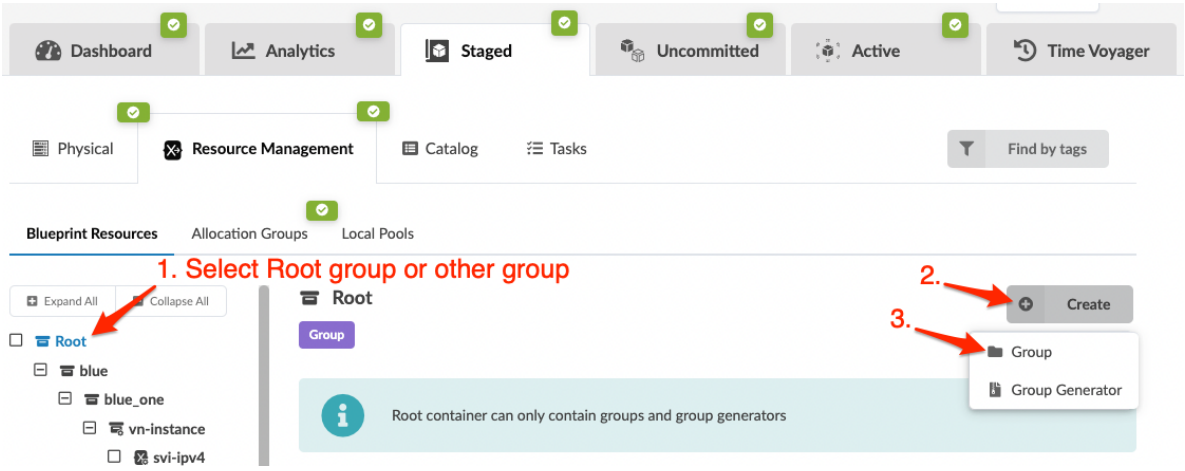
Create Group (Freeform)

SUMMARY

Organize resources in Freeform blueprints with groups (folders).

Groups are folders used to organize resources in Freeform blueprints. You can nest groups inside other groups in as many levels needed to organize your resources. You can add new groups to any existing group. If you haven't created any groups yet, you'll put your new group in the built-in **Root** group. (Instead of, or in addition to, creating groups manually as described here, you can "create group generators" on page 567 that create groups automatically and dynamically based on conditions that you set.)

1. From the blueprint, navigate to **Staged > Resource Management > Blueprint Resources**.



2. Click the group where you want to put the new group, then click **Create** (right-side) and select **Group**.

   The group you selected appears in the immutable **Parent** field.

3. Enter a group name.

4. (The Data field holds metadata that you can associate with the group. It's used to impart information to the object you've created. It may be used for things like a desctiption or perhaps to indicate to others what the object represents.) If you'd like to add context to the group, enter applicable key-value pairs in the **Data** field.

   Example: `{"group_type":"vn"}`

5. Click **Create** to create the group and return to the **Blueprint Resources** view.

When you've created one or more groups you can start putting resources and resource generators into them.

Create Group Generator (Freeform)

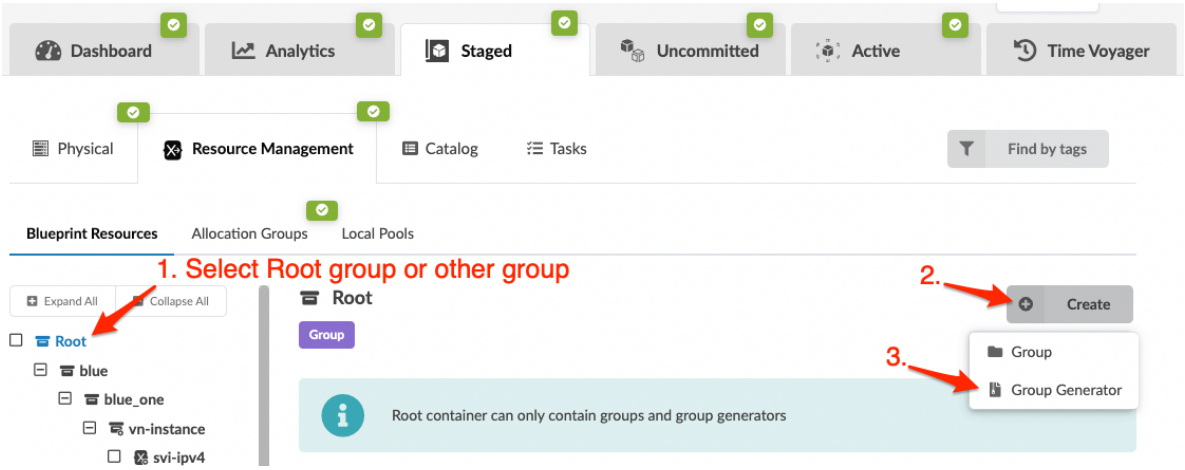
SUMMARY

Groups (folders) in Freeform blueprints organize resources. Group generators (folders with properties) automatically create and delete groups based on specified conditions.

For more explanation, see the "Freeform Resource Management Introduction" on page 560.

1. From the blueprint, navigate to **Staged > Resource Management > Blueprint Resources**.

   The group directory appears on the left. You can nest group generators inside any group.



2. Click the group where you want to put the new group generator, then click **Create** (right-side) and select **Group Generator**.

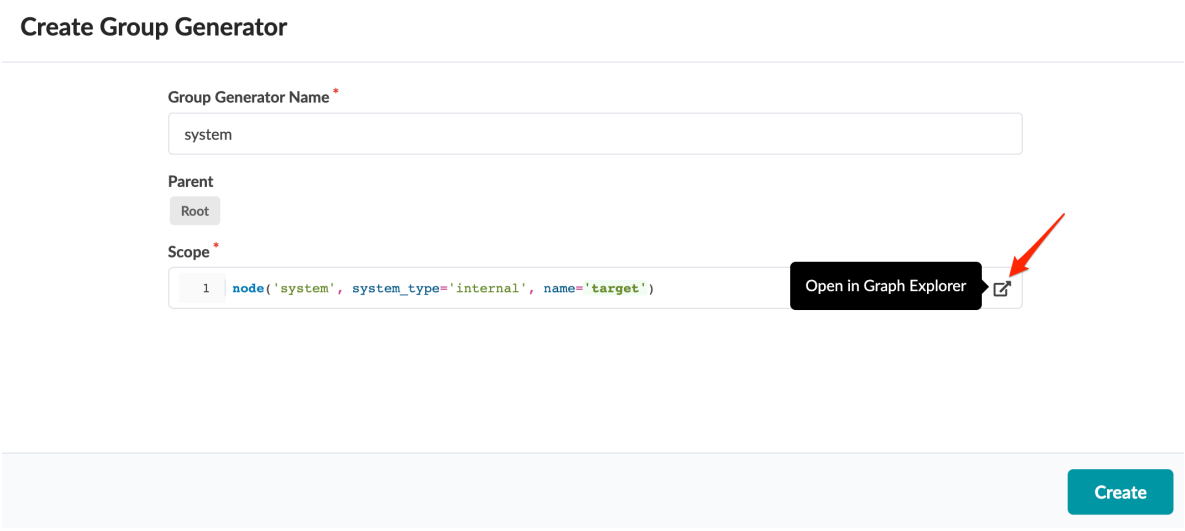
   The name of the group you selected appears in the immutable **Parent** field.

3. Enter a group generator name, then specify a scope based on how you want your groups to be created and managed.

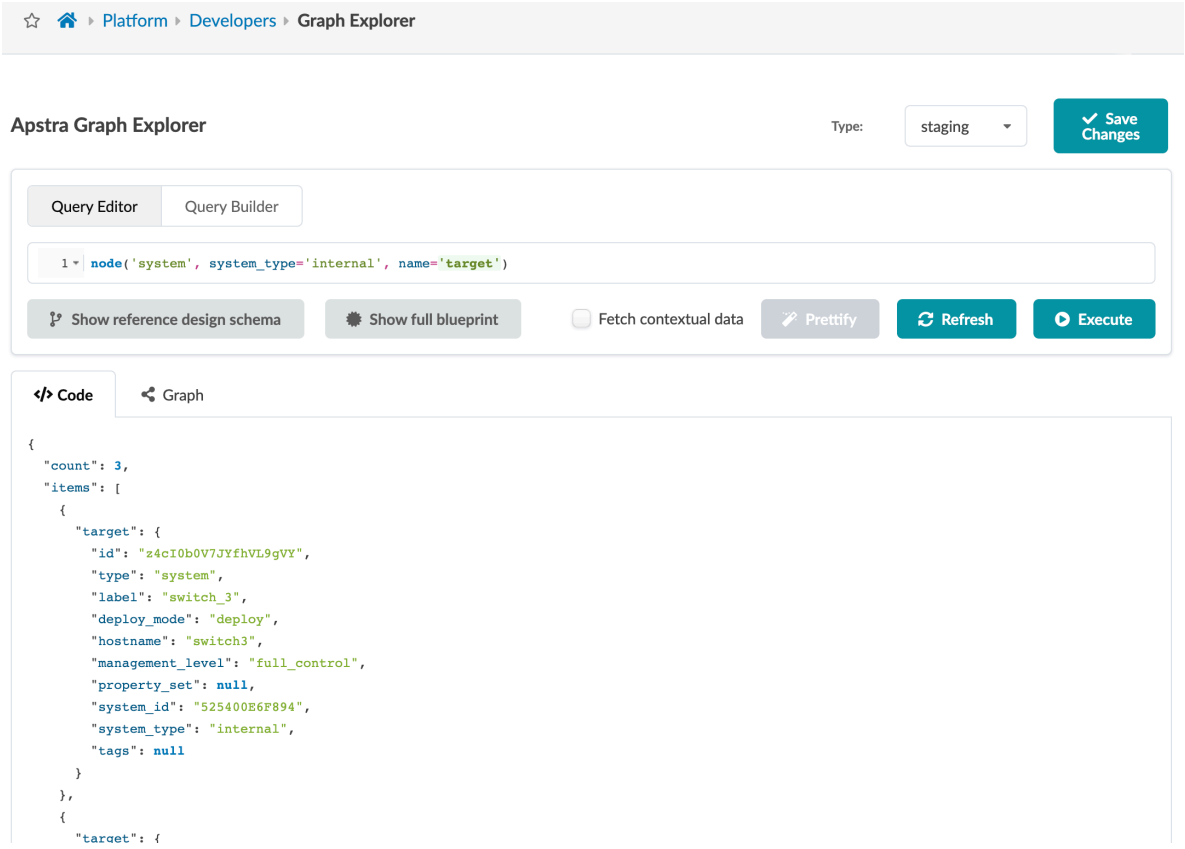
   For example, to create one group for every internal system, use the following scope:

```
node('system', system_type='internal', name='target')
```

   This scope tells the graph database to find all internal systems and create a group for each one of them; then assign the name of the system to each group. If you subsequently delete a system, the group created for that system is also deleted. Conversely, if you create a system after this group generator is created, a group for that system is automatically created.



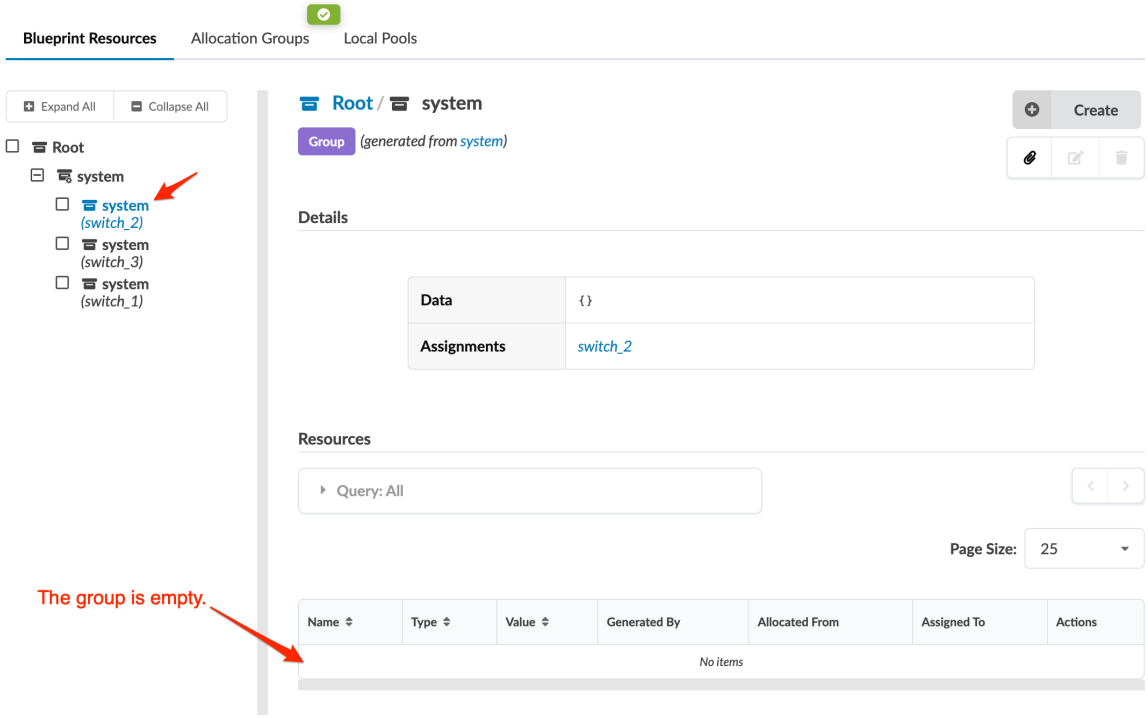
You can click the **Open in Graph Explorer** button to open a new tab that shows the groups that will be created based on the current topology. In our example, the topology includes 3 internal systems, and 3 groups will be created, as expected.



4. Back in the **Create Group Generator** dialog, click **Create** to create the group generator and return to the **Blueprint Resources** view.

   Groups will be created and deleted dynamically based on your specified conditions.

In our example, the group generator named **system** was created inside the **Root** folder, and it automatically created 3 groups, one for each of the systems in the topology. To see the resources in a group, click the name of the group. We haven't put any resources into the group we just created, so the resource table is empty.



Next Steps: "Set up resource generators" on page 570 to automatically add and delete resources in your groups, as needed.

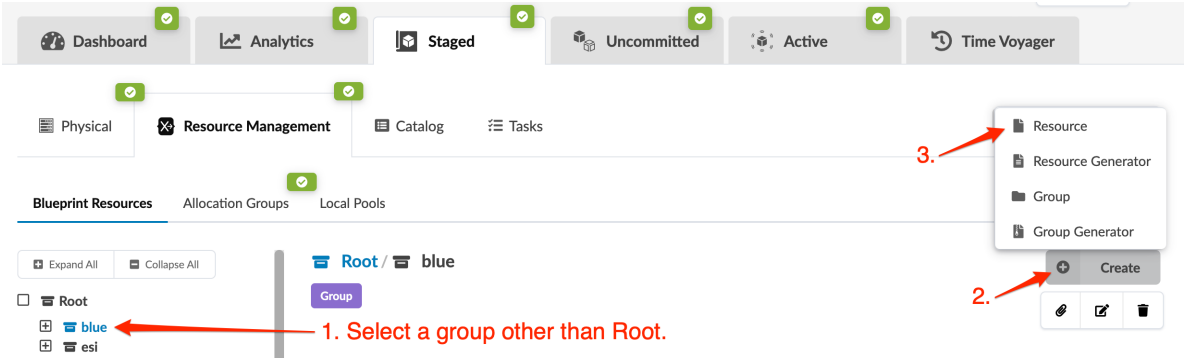
Create Resource (Freeform)

SUMMARY

Resources are values that you assign to systems and links. Resources include IPv4 addresses, IPv6 addresses, ASNs, VNIs, VLANs, and integers.

Resources are located inside groups (folders) that you create. (Resources can't be put directly in the predefined **Root** group). If you haven't "created groups" on page 566 yet, create them before proceeding here.

1. From the blueprint, navigate to **Staged > Resource Management > Blueprint Resources**.



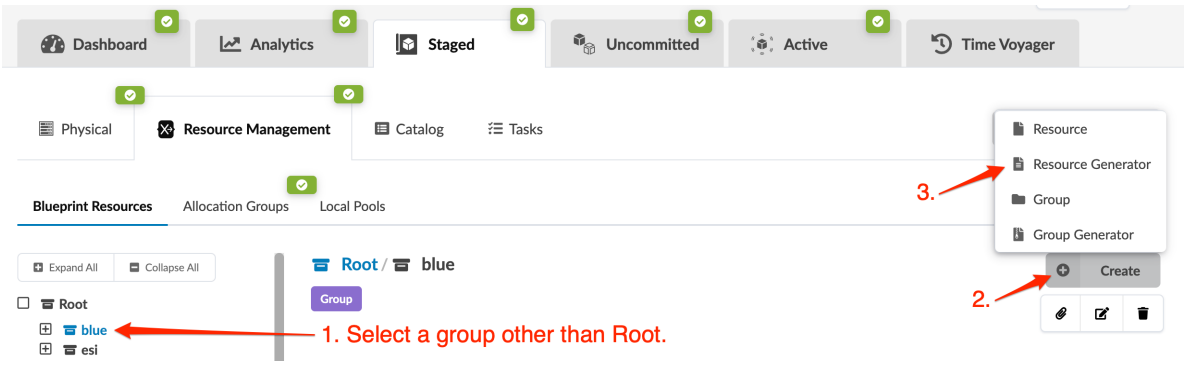
2. Click the group where you want to put the new resource, then click **Create** (right-side) and select **Resource**.

   The group you selected appears in the immutable **Parent** field.
3. Enter a resource name and select a resource type (IPv4, Host IPv4, IPv6, Host IPv6, ASN, VNI, VLAN, Integer).
4. To have Apstra automatically pull resources from pools, select the applicable allocation group from the drop-down list.
5. To manually allocate a resource, enter the value. in the **Value (override)** field.
6. Enter a subnet prefix length, as applicable.
7. Click **Create** to create the resource and return to the **Blueprint Resources** view.

Create Resource Generator (Freeform)

Resource generators are located inside groups (folders) that you create. If you haven't "created groups" on page 566 yet, create them before proceeding. To automate resource allocation you'll also need to confirm that you've created allocation groups and that they map to a sufficent number of resources.

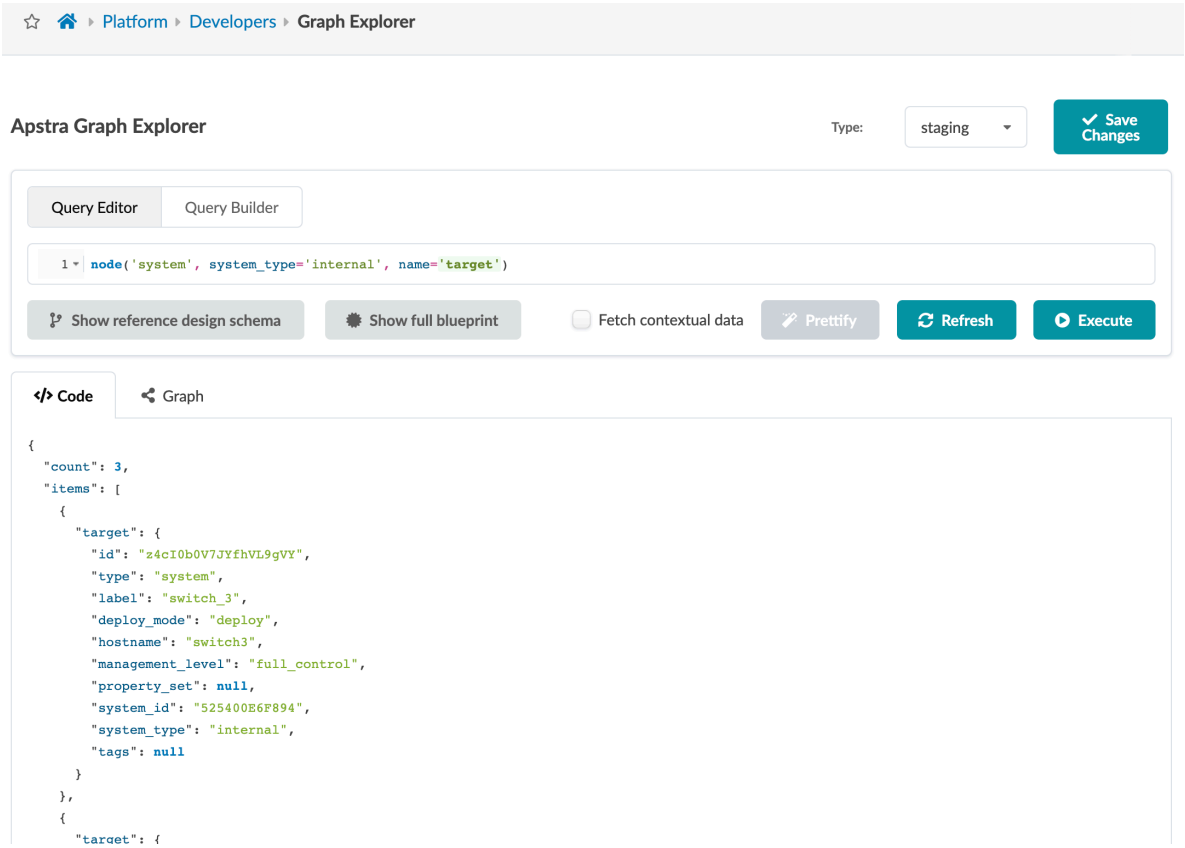
1. From the blueprint, navigate to **Staged > Resource Management > Blueprint Resources**.



2. Select the group where you want to put the new resource generator, then click **Create** (right-side) and select **Resource Generator**.

   The type and name of the container (group) appear in the immutable **Container Type** and **Container** fields, respectively.
3. Enter a resource generator name, then enter the scope for your generator.

To assist with determining scope, you can use the **Graph Explorer**.



4. Click **Create** to create the resource generator and return to the **Blueprint Resources** view.

Resources will be generated and deleted dynamically based on the scope.

Allocation Groups

IN THIS SECTION

Create Allocation Group (Freeform)

SUMMARY

Allocation groups consist of one or more resource pools that you use to assign resources (IPv4, IPv6, ASN, VNI, Integers).

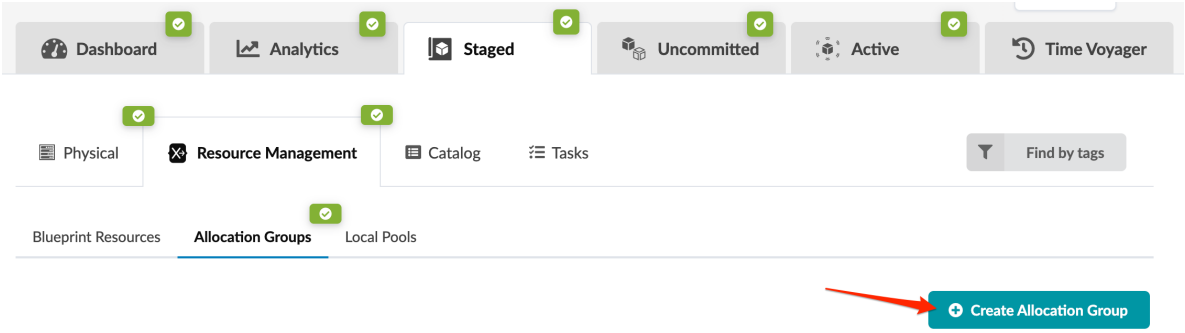
IN THIS SECTION

(You can map additional resource pools to allocation groups at any time. You might get low on resources. If the global resource pools don't have enough resources defined you can create more pools or add a range of values to an existing pool. You can create allocation groups from the resource managment tab or from the topology view. You're just creating a group of already existing resource pools. It's just a way to combing them in one location.)

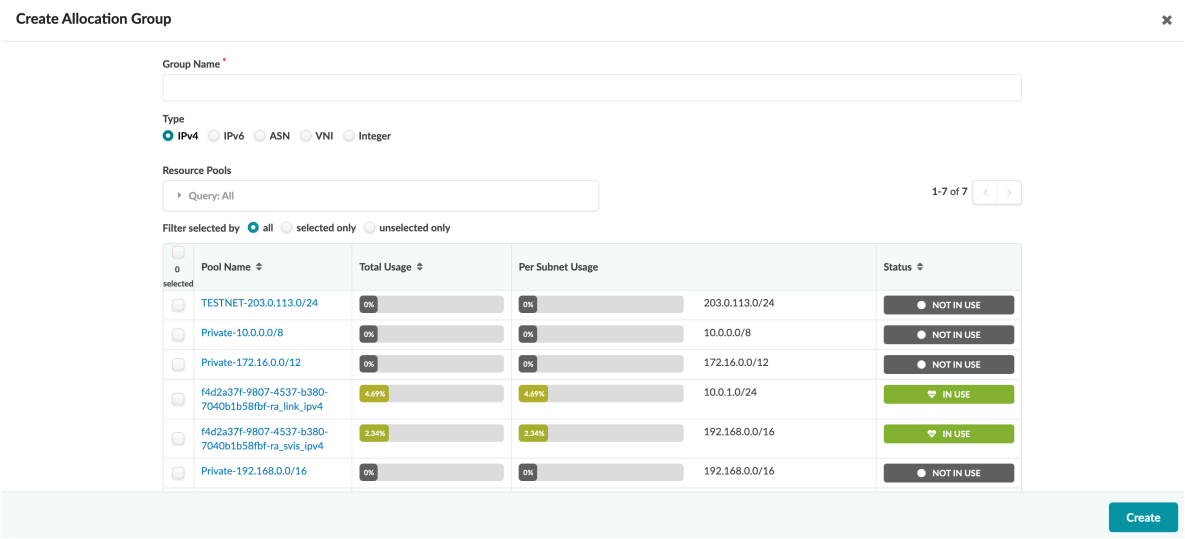
An allocation group consists of one or more "global resource pools" on page 927. You'll assign resources later from one of these allocation groups. If you haven't created the resource pools you need, go do that before proceeding here.

Create Allocation Group (from Resource Management Tab)

1. From the blueprint, navigate to **Staged > Resource Management > Allocation Groups > Create Allocation Group**.



2. Enter an allocation group name and select the resource type (IPv4, IPv6, ASN, VNI, Integer).

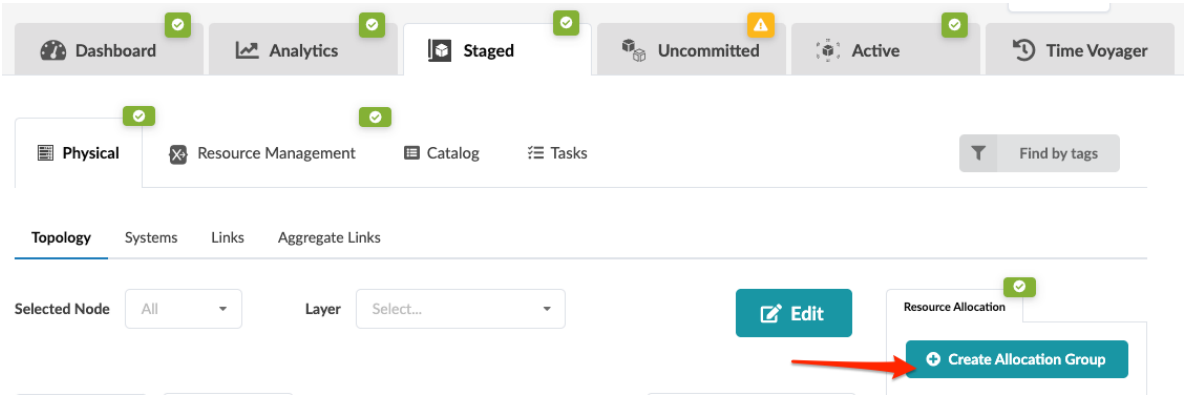


3. Select one or more check boxes for the resource pools to include in the allocation group. (These resource pools are from the global **Resources** catalog in the left navigation menu. You can add resource pools at any time if you need more resources available for your allocation groups.) You can create the group without selecting any resource pools, but of course, you'll need to add at least one before you can assign resources from it.
4. Click **Create** to create the allocation group and return to the table view.

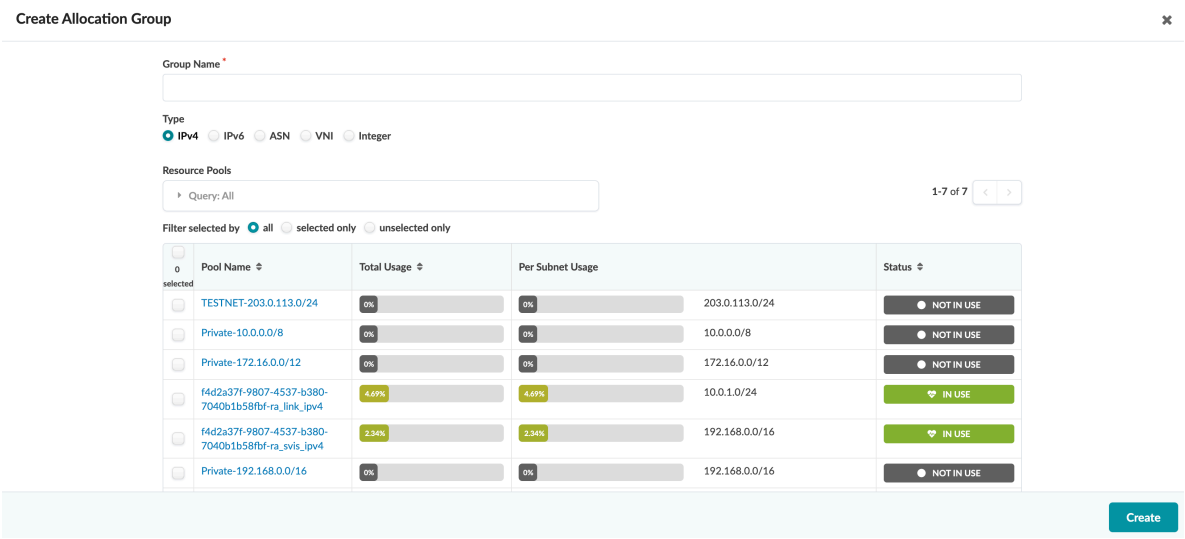
Next Steps: When you assign resources, you'll select an allocation group that you've created; then Apstra will pull resources from the group and assign them, as needed.

Create Allocation Group (from Topology View)

1. From the blueprint, navigate to **Staged > Physical > Topology > Create Allocation Group**.



2. Enter an allocation group name and select a resource type (IPv4, IPv6, ASN, VNI, Integer).



3. Select one or more check boxes for the resource pools to include in the allocation group. (These resource pools are from the global **Resources** catalog in the left navigation menu. You can add resource pools at any time if you need more resources available for your allocation groups.)
4. Click **Create** to create the allocation group and return to the **Topology** view.

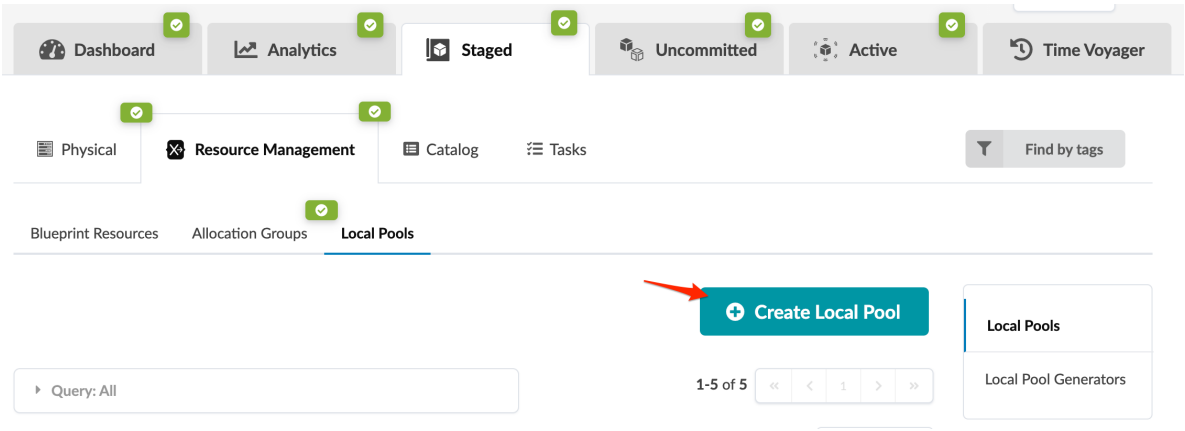
When you assign resources, you'll select an allocation group that you've created; then Apstra will pull resources from the group and assign them, as needed.

Local Pools

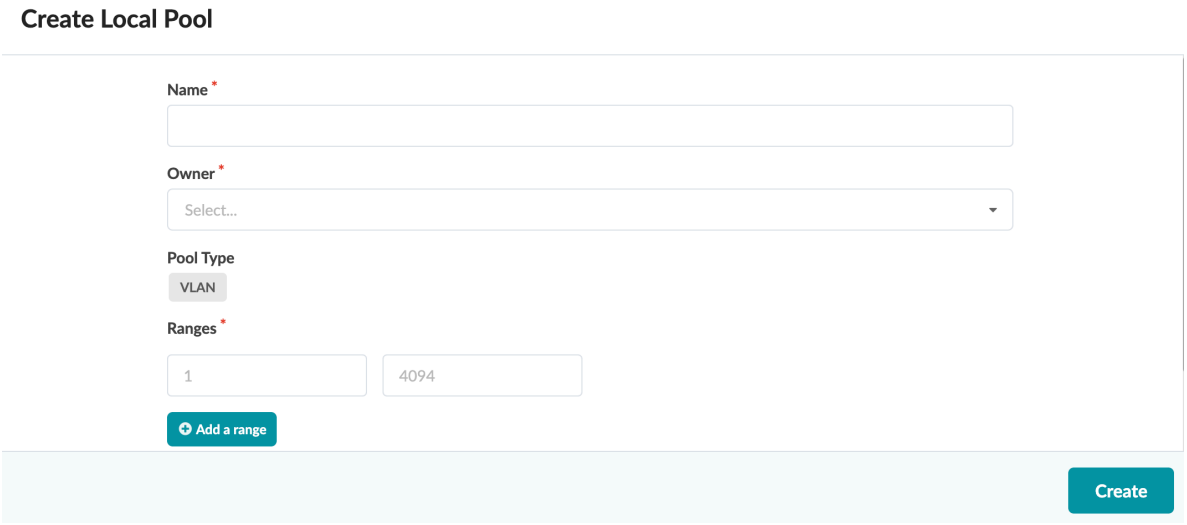
IN THIS SECTION

Create Local Pool (Freeform)

1. From the blueprint, navigate to **Staged > Resource Management > Local Pools > Create Local Pool**.



2. Enter a local pool name.



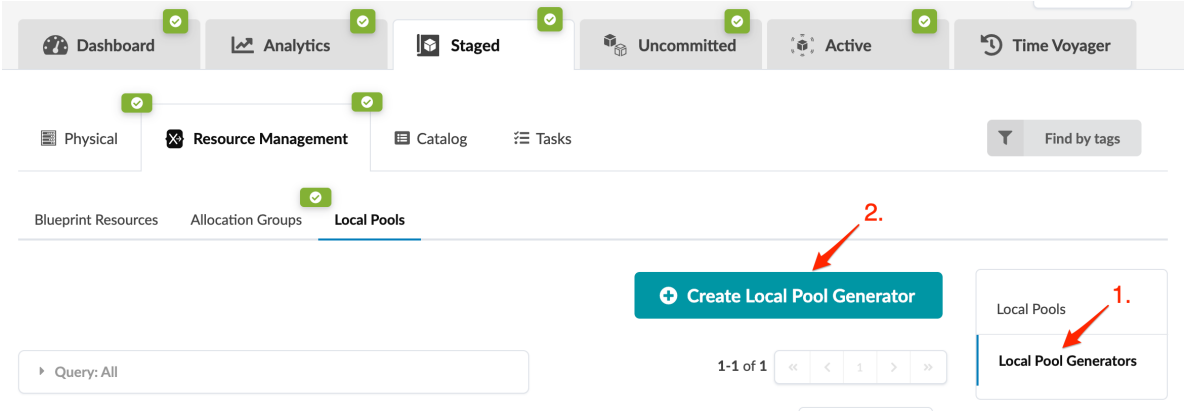
3. You'll be applying the integers to a system. Select the system from the **Owner** drop-down list.
4. Enter the range of integers for the pool.
5. If you want to add another range, click **Add a range** and enter the range.
6. Click **Create** to create the pool and return to the table view.

Create Local Pool Generator (Freeform)

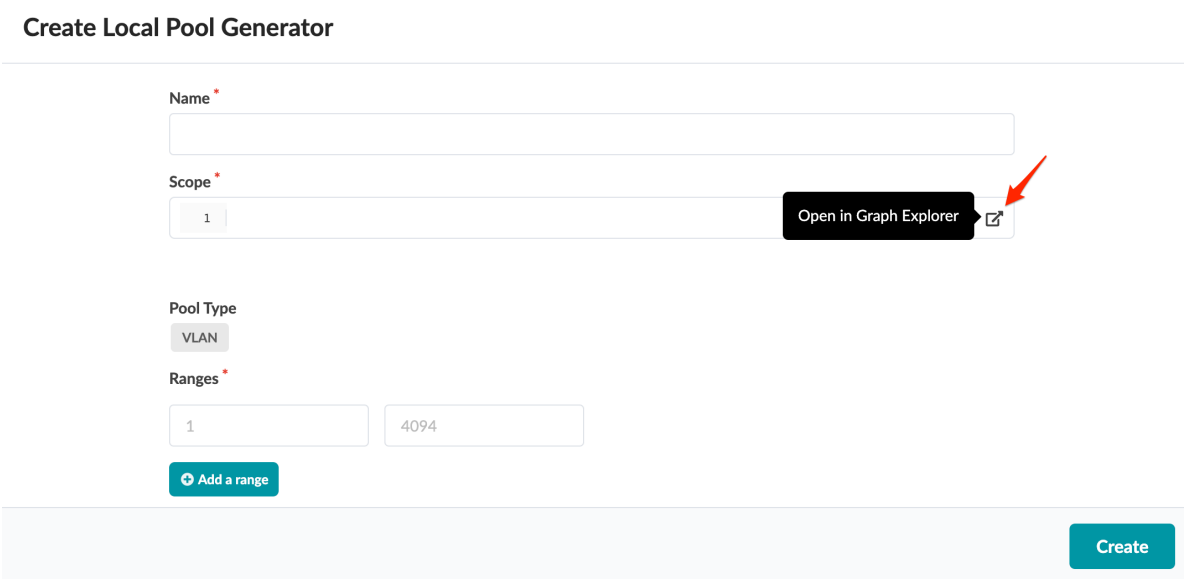
SUMMARY

Use local pools to ?. Create local pool generators to automatically create and delete local pools based on your criteria.

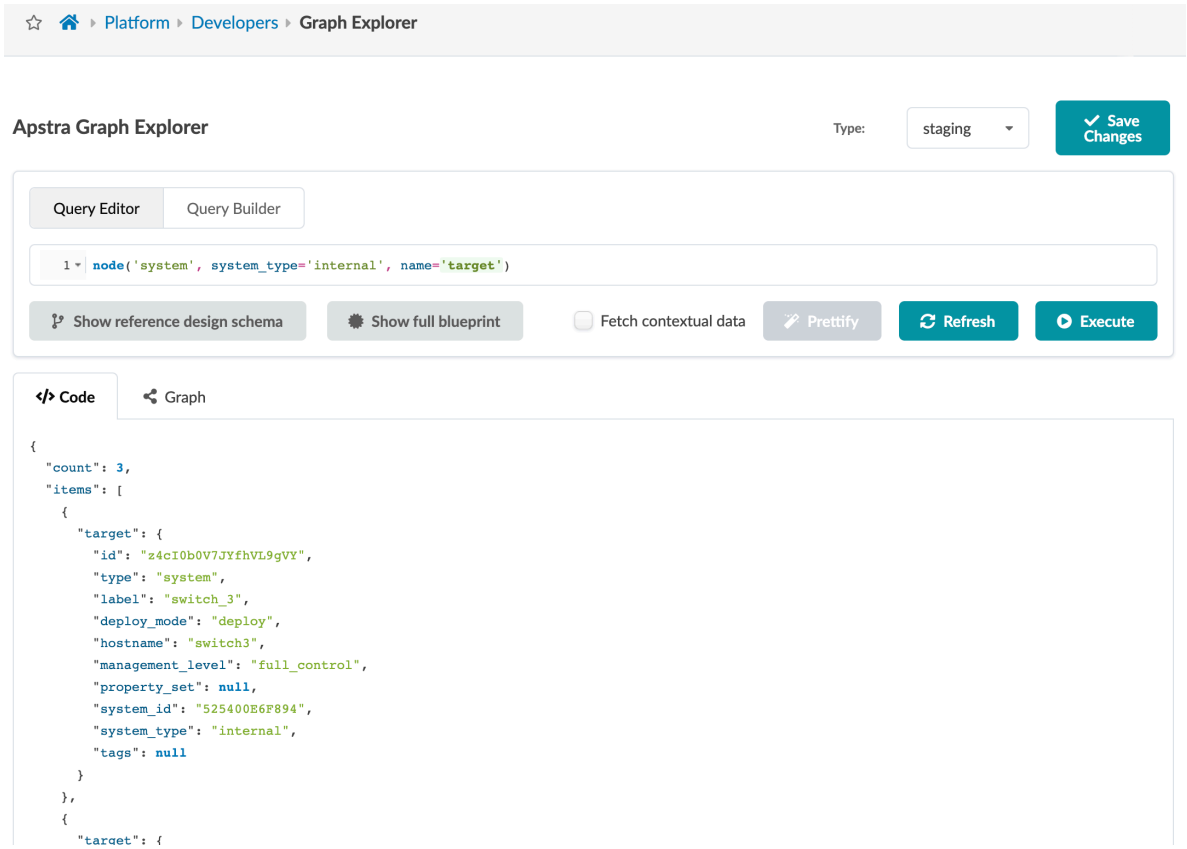
1. From the blueprint, navigate to **Staged > Resource Management > Local Pools > Local Pool Generator > Create Local Pool Generator**.



2. Enter a local pool generator name, then enter the scope for your generator.



To assist with determining scope, you can use the **Graph Explorer**.



3. Click **Create** to create the local pool generator and return to the **Local Pools** table view.

CHAPTER 17

Catalog (Freeform)

IN THIS CHAPTER

Config Templates

IN THIS SECTION

Config Templates (Freeform Blueprint)

IN THIS SECTION

- Config Template and Property Sets | 580

We recommend that you familiarize yourself with the Jinja Template Designer before working with config templates.

Several predefined config templates are included with the Apstra product. To get familiar with the syntax and how config Jinja is used in config templates. check out the sections below.

A Simple Config Template

Let's take a look at the config template `junos_protocols.jinja`, which ships with Apstra software.

```
protocols {
    lldp {
        port-id-subtype interface-name;
        port-description-type interface-description;
        neighbour-port-info-display port-id;
        interface all;
    }
}
```

This straightforward template doesn't include any variables or other conditions. It's nested inside the config template `junos_configuration.jinja`, one of the other predefined config templates. You could create your own config template and nest this basic one in it as well.

Config Template With Variable

Let's look at `junos_system.jinja`, another predefined config template.

```
{% if hostname %}
system {
    host-name {{hostname}};
}
{% endif %}
```

This template includes an if-then statement and the variable `hostname`. When configuration is rendered, if the system device context includes a value for `hostname`, then the rendered configuration includes that value.

Config Template and Property Sets

An example of using property sets is with NTP servers. Configuration for NTP might be consistent across all devices in the enterprise except for time sources or strata per geography. You can build a config template with a variable, named `ntp` for example, in place of the actual IP address. The configuration will be generated with the value of the `ntp` property in a property set. You'd import the same config template into all blueprints, but for blueprints running in the east region you'd import the "EAST" property set, and for blueprint running in the west region you'd import the "WEST" property set. Property sets are global, that is they are blueprint-wide.

The config template could look like this.

```
{% if property_sets.get('ntp') %}
system {
    ntp {
        server {{property_sets['ntp']['ntp_server']}};
    }
}
{% endif %}
```

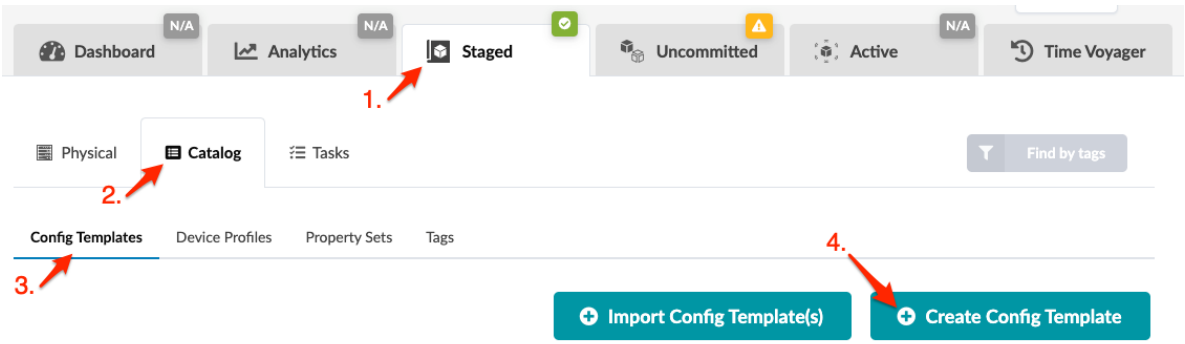
The example below shows the syntax for the property set `ntp` that contains the IP address.

```
ntp_server = '1.2.3.4'
```

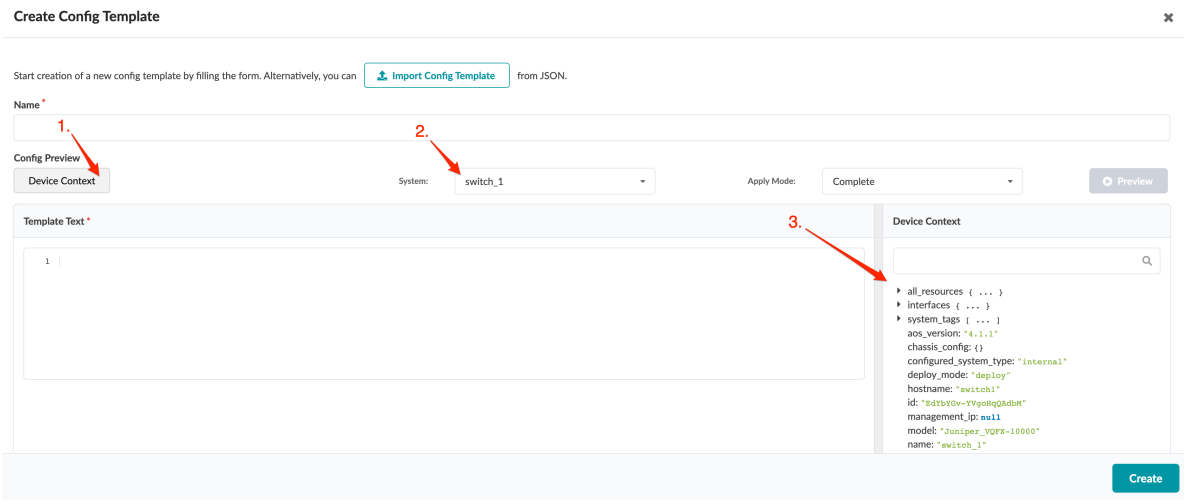
Create Config Template (Freeform Blueprint)

Creating config templates in the blueprint catalog (instead of the design catalog), gives you access to device context for systems that you've already added to your blueprint. Device context groups relevant information into one place, making it easier to get the information you need while creating config templates.

1. From the blueprint, navigate to **Staged > Catalog > Config Templates** and click **Create Config Template**.



2. In the dialog, enter a name for the config template including the `.jinja` extension. (The .jinja extension is required even if you're not using Jinja.)
3. Enter or paste your content into the **Template Text** field. You can also import a config template that you created in the design (global) catalog.
   - To see device context, click **Device Context**.
   - To see device context for a specific system, select it from the **System** drop-down list.
   - Preview and Preview Mode are available only when you're editing a config template.



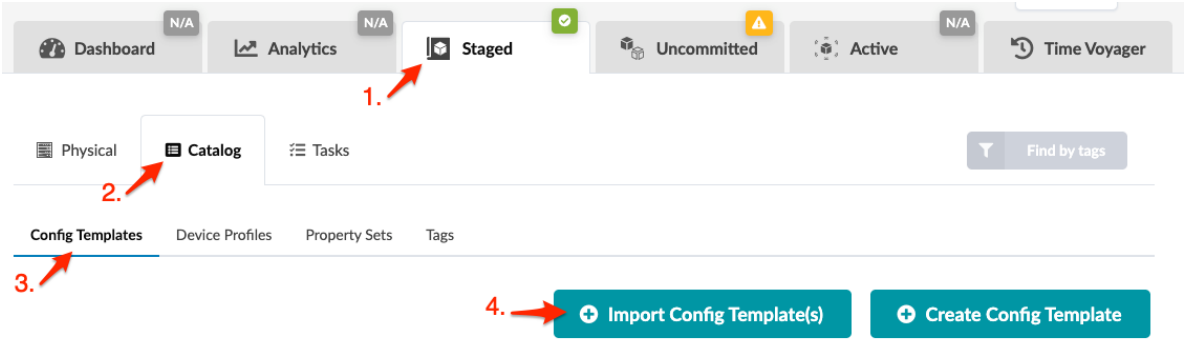
4. Click **Create** to create the config template and return to the config template catalog view.

When you're ready you can "assign config templates" on page 521 to internal systems.

Import Config Template (Freeform)

You can create config templates in the design (global) catalog, then import them into as many blueprints as you want. (You can also create config templates directly in your blueprint, which gives you access to device context making it easier to write config template.)

1. From the blueprint, navigate to **Staged > Catalog > Config Templates** and click **Import Config Template(s)**.

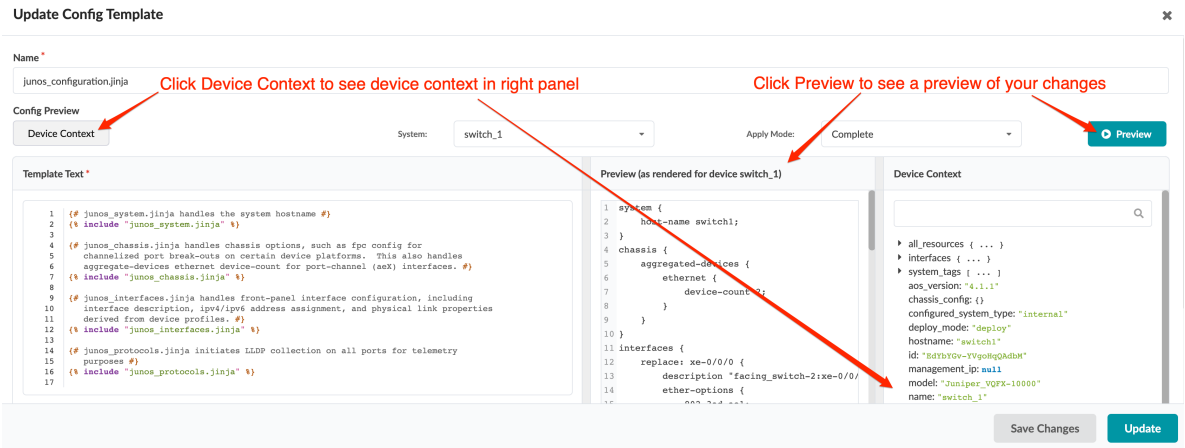


2. Select the check boxes for the config templates to import from the design (global) catalog.

3. Click **Import** to stage the import and return to the table view.

Edit Config Template (Freeform Blueprint)

1. From the blueprint, navigate to **Staged > Catalog > Config Templates** to go to the table view.
2. Either from the table view or the details view click the **Edit** button for the config template to edit.



3. In the dialog, make your changes.
   - To see device context, click **Device Context**.
   - To see device context for a specific system, select it from the **System** drop-down list.
   - To see a preview of your changes, click **Preview**.
     - To see the full configuration, including the changes you're making, select **Complete** from the **Preview Mode** drop-down list.
     - To see only the configuration that you've changed, select **Incremental** from the **Apply Mode** drop-down list.
4. Click **Update** (bottom-right) to update the config template and return to the table view.

Export Config Template (Freeform)

If you create a config template directly in a blueprint, and you want to make it available to other blueprints, you can export it to the design (global) catalog.

1. From the blueprint, navigate to **Staged > Catalog > Config Templates** to go to the table view.
2. Either from the table view or the details view click the **Export config template** button for the config template to export.
3. Click **Copy** to copy the contents, **Export to Global** to export the config template to the design (global) catalog, or click **Save As File** to download the file.
4. When you've copied, exported or downloaded the config template, close the dialog to return to the table view.

Delete Config Template (Freeform Blueprint)

1. From the blueprint, navigate to **Staged > Catalog > Config Templates** to go to the table view.
2. Either from the table view or the details view click the **Delete** button for the config template to delete.
3. Click **Delete** to stage the deletion and return to the table view.

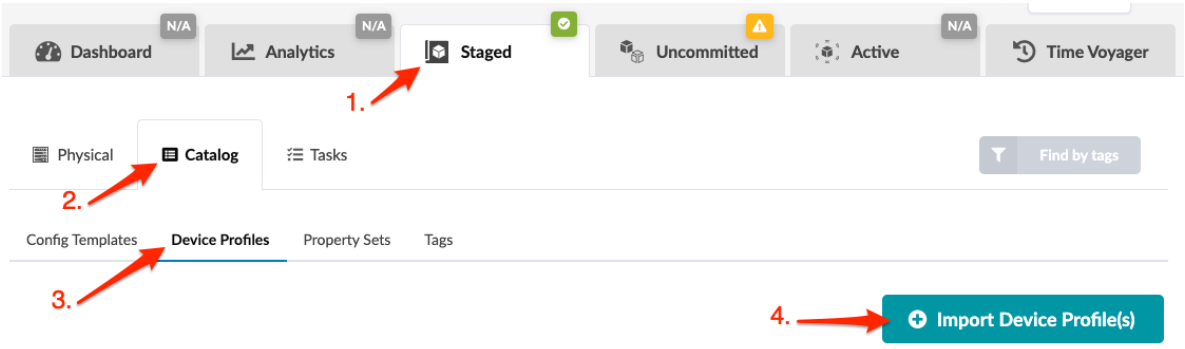
Device Profiles

IN THIS SECTION

Import Device Profile (Freeform)

Device Profiles define the capabilities of supported hardware devices. They interact with devices via system agents. They don't include system IDs (serial numbers) which enables you to build your network in the Apstra environment 'offline' before you have your devices ready. In Freeform blueprints you import device profiles to provide context for configuring systems with config templates.

1. From the blueprint, navigate to **Staged > Catalog > Device Profiles** and click **Import Device profile(s)** (right-side).



2. Select one or more check boxes for the device profile(s) to import into the blueprint. Only supported device profiles in Freeform appear in the list (currently only Juniper devices).
3. Click **Import** to stage the change and return to the table view. The newly imported device profile(s) appear in the list.

Next Steps:

You're ready to "create internal systems" on page 511 and assign your imported device profiles to them.

RELATED DOCUMENTATION

What are Device Profiles | 783

Delete Device Profile (Freeform)

If a device profile is not being used by a system, you can delete it from the Freeform blueprint catalog.

1. From the blueprint, navigate to **Staged > Catalog > Device Profiles** and click the **Delete** button in the **Actions** panel for the device profile to delete.
   The **Delete Device Profile** dialog opens showing the device profile to be deleted.
2. Click **Delete** to stage the deletion and return to the the Device Profile catalog view.

RELATED DOCUMENTATION

What are Device Profiles | 783

Property Sets

IN THIS SECTION

Property Sets (Freeform Blueprints)

Property sets provide a valuable capability to fully parameterize config templates. Consisting of key-value pairs, they enable you to separate static portions of config templates from variables. You create/clone property sets in the blueprint catalog. (Property sets used in Freeform blueprints are not related to

property sets in the design (global) catalog.) You'll include property set names in your config template and then the values in those property sets will be used when configuration is rendered.

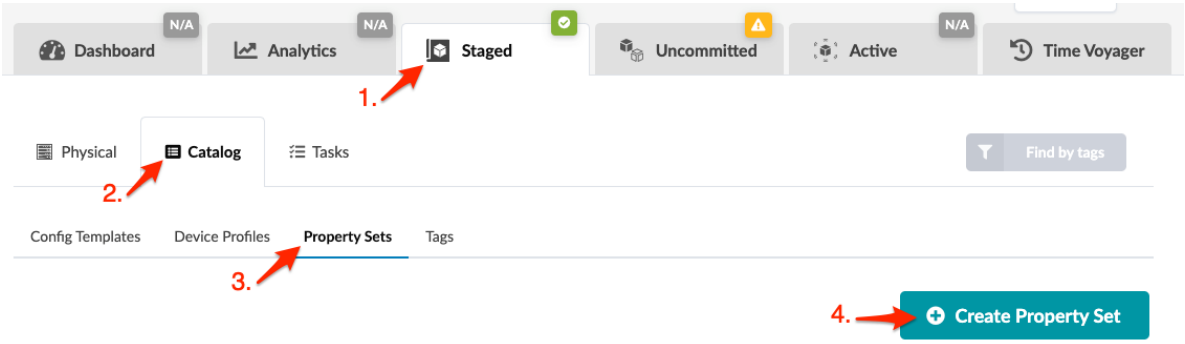
You can also create a property set and assign it directly to one system.

Create Property Set (Freeform Blueprint)

IN THIS SECTION

Create Property Set with Builder

1. From the blueprint, navigate to **Staged > Catalog > Property Sets** and click **Create Property Set**, or to clone an existing property set, click the **Clone** button in the **Actions** panel for the property set to copy. By cloning, you can re-use an existing property set structure with different values for a different node.

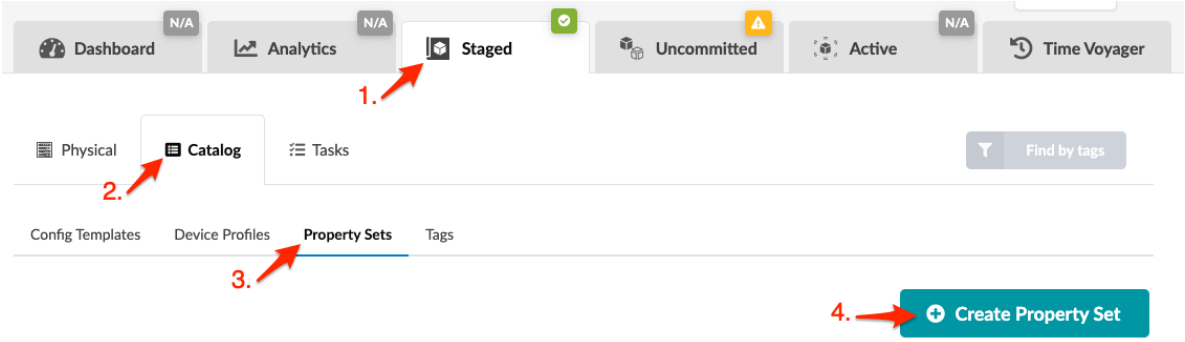


2. Enter a name for the property set.
3. If you want to assign the property set to a specific system, select it from the **System** drop-down list.
4. Select Input Type **Builder**. (YAML is not available when inputting via Builder.)
5. Use the interactive builder to help you create the content for your property set.
6. Click **Create** to stage the new property set and return to the property set catalog. The newly created property set is in the list.

Create Property Set with Editor

1. From the blueprint, navigate to **Staged > Catalog > Property Sets** and click **Create Property Set**, or to clone an existing property set, click the **Clone** button in the **Actions** panel for the property set to

copy. (Cloning is new in Apstra version 4.1.2.) By cloning, you can re-use an existing property set structure with different values for a different node.



2. Enter a name for the property set.
3. If you want to assign the property set to a specific system, select it from the **System** drop-down list.
4. Select input type **Editor**.
5. As of Apstra 4.1.2, you have the option of defining your property set with YAML. Select YAML or JSON, as applicable.
6. Copy and paste your content in the editor or type it in.
7. Click **Create** to stage the new property set and return to the property set catalog. The newly created property set is in the list.

Edit Property Set (Freeform Blueprint)

1. From the blueprint, navigate to **Staged > Catalog > Property Sets** to go to the table view.
2. Either from the table view or the details view, click the **Edit** button for the property set to edit.
3. Make your changes.
4. Click **Update** to stage your changes and return to the table view.

Delete Property Set (Freeform Blueprint)

1. From the blueprint, navigate to **Staged > Catalog > Property Sets** and click the **Delete** button for the property set to delete.
2. Click **Delete** to stage the deletion and return to the table view.

Tags

IN THIS SECTION

Tags (Freeform)

SUMMARY

IN THIS SECTION

Tags Overview

You can **tag** systems, then later when you want to find systems you can use the **Find by Tags** feature to find them.

You can include **Tags** in config templates. Systems/links with those tags will be rendered as specified in the config template. For example, if you have bare metal servers with SRIOV interfaces, and you need to produce specific configuration for those interfaces, you can add the tag `sriov`, then specify that links with that tag to be configured per the config template.

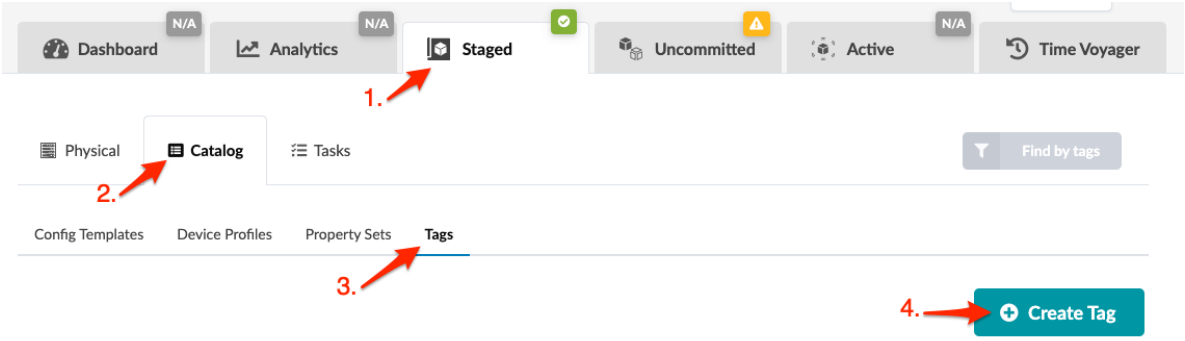
Tags are a way for you to assign metadata to Apstra-managed resources. Tags can help you identify, organize, search for, and filter Apstra systems and links. With tags, you can categorize resources by purpose, owner, environment, or other criteria. Because tags are metadata, they are not just used for visual labeling; they are also applied as properties of nodes in the Apstra graph database. This node property (or device property) is then available for you to reference in Jinja for dynamic variables in config generation and the Apstra real-time analytics via Apstra's Live Qurey technology and Apstra Intent-Based Analytics.

Here is an example of using the tag firewall in a "config template" on page 578 to render a specific description.

```
{% if has_tag(interface.link.neighbor_system.id, 'firewall') %}
 description "this is a firewall facing interface";
{% endif %}
```

Create Tag in Catalog (Freeform)

1. From the blueprint, navigate to **Staged > Catalog > Tags** and click **Create Tag**.



2. Enter a name and (optional) description. Names are case-insensitive.
3. Click **Create** to stage the tag addition and return to the table view. The newly created tag appears in the table.

RELATED DOCUMENTATION

What are Tags | 922

Change Tag Description (Freeform)

1. From the blueprint, navigate to **Staged > Catalog > Tags** and click the **Edit** button for the tag to edit.
2. Change the description.
3. Click **Update** to stage the change and return to the table view.

RELATED DOCUMENTATION

What are Tags | 922

Delete Tag from Catalog (Freeform)

1. From the blueprint, navigate to **Staged > Catalog > Tags** and click the **Delete** button for the tag to delete.
2. Click **Delete** to stage the deletion and return to the table view.

RELATED DOCUMENTATION

What are Tags | 922

CHAPTER 18

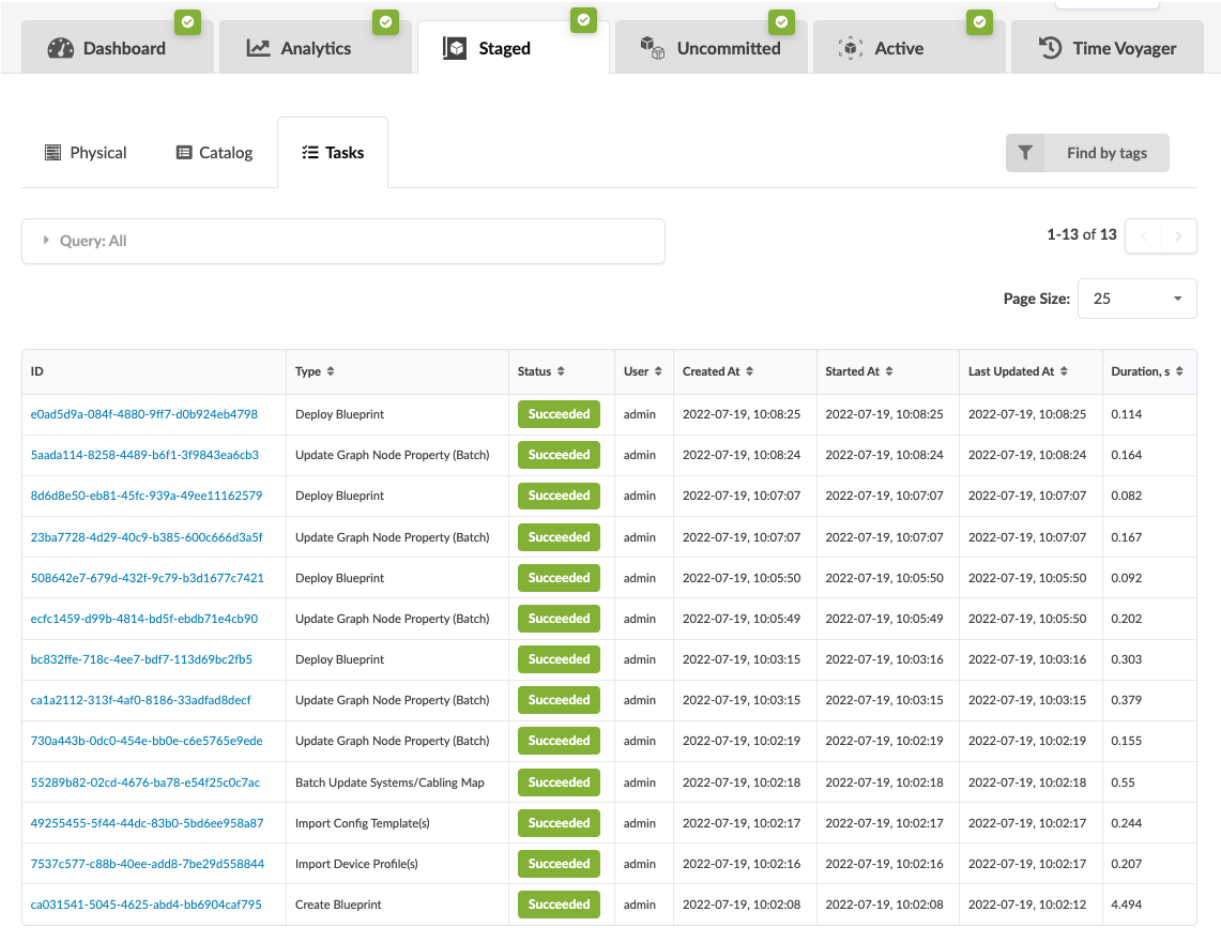
Tasks

IN THIS CHAPTER

Tasks - Staged (Freeform)

Tasks that haven been performed in blueprints appear in the Tasks tab. Blueprint task details include task type, task status (succeeded, failed, in progress), user who performed the task, date/time created/started/updated, and the duration of the task. For any failed tasks, you can click to see error messages.

From the blueprint, navigate to **Staged > Tasks** to go to task history.



ID	Type	Status	User	Created At	Started At	Last Updated At	Duration, s
e0ad5d9a-084f-4880-9ff7-d0b924eb4798	Deploy Blueprint	Succeeded	admin	2022-07-19, 10:08:25	2022-07-19, 10:08:25	2022-07-19, 10:08:25	0.114
5aada114-8258-4489-b6f1-3f9843ea6cb3	Update Graph Node Property (Batch)	Succeeded	admin	2022-07-19, 10:08:24	2022-07-19, 10:08:24	2022-07-19, 10:08:24	0.164
8d6d8e50-eb81-45fc-939a-49ee11162579	Deploy Blueprint	Succeeded	admin	2022-07-19, 10:07:07	2022-07-19, 10:07:07	2022-07-19, 10:07:07	0.082
23ba7728-4d29-40c9-b385-600c666d3a5f	Update Graph Node Property (Batch)	Succeeded	admin	2022-07-19, 10:07:07	2022-07-19, 10:07:07	2022-07-19, 10:07:07	0.167
508642e7-679d-432f-9c79-b3d1677c7421	Deploy Blueprint	Succeeded	admin	2022-07-19, 10:05:50	2022-07-19, 10:05:50	2022-07-19, 10:05:50	0.092
ecfc1459-d99b-4814-bd5f-ebdb71e4cb90	Update Graph Node Property (Batch)	Succeeded	admin	2022-07-19, 10:05:49	2022-07-19, 10:05:49	2022-07-19, 10:05:50	0.202
bc832ffe-718c-4ee7-bdf7-113d69bc2fb5	Deploy Blueprint	Succeeded	admin	2022-07-19, 10:03:15	2022-07-19, 10:03:16	2022-07-19, 10:03:16	0.303
ca1a2112-313f-4af0-8186-33adfad8decf	Update Graph Node Property (Batch)	Succeeded	admin	2022-07-19, 10:03:15	2022-07-19, 10:03:15	2022-07-19, 10:03:15	0.379
730a443b-0dc0-454e-bb0e-c6e5765e9ede	Update Graph Node Property (Batch)	Succeeded	admin	2022-07-19, 10:02:19	2022-07-19, 10:02:19	2022-07-19, 10:02:19	0.155
55289b82-02cd-4676-ba78-e54f25c0c7ac	Batch Update Systems/Cabling Map	Succeeded	admin	2022-07-19, 10:02:18	2022-07-19, 10:02:18	2022-07-19, 10:02:18	0.55
49255455-5f44-44dc-83b0-5bd6ee958a87	Import Config Template(s)	Succeeded	admin	2022-07-19, 10:02:17	2022-07-19, 10:02:17	2022-07-19, 10:02:17	0.244
7537c577-c88b-40ee-add8-7be29d558844	Import Device Profile(s)	Succeeded	admin	2022-07-19, 10:02:16	2022-07-19, 10:02:16	2022-07-19, 10:02:17	0.207
ca031541-5045-4625-abd4-bb6904caf795	Create Blueprint	Succeeded	admin	2022-07-19, 10:02:08	2022-07-19, 10:02:08	2022-07-19, 10:02:12	4.494

6

PART

Uncommitted Blueprints

Uncommitted Introduction

IN THIS SECTION

While you're staging your new blueprint (under the **Staged** tab), the status indicator on the **Uncommitted** tab is red. When you've finished staging the blueprint and resolved any build errors, the indicator turns yellow, or orange if you have warnings, and the **Commit** button turns from gray to black indicating that the blueprint is ready to be committed. When you commit your pending changes you are pushing configuration to the **Active** blueprint. The meaning of the status indicator colors are shown in the table below:

Table 3: Uncommitted Status Indicators

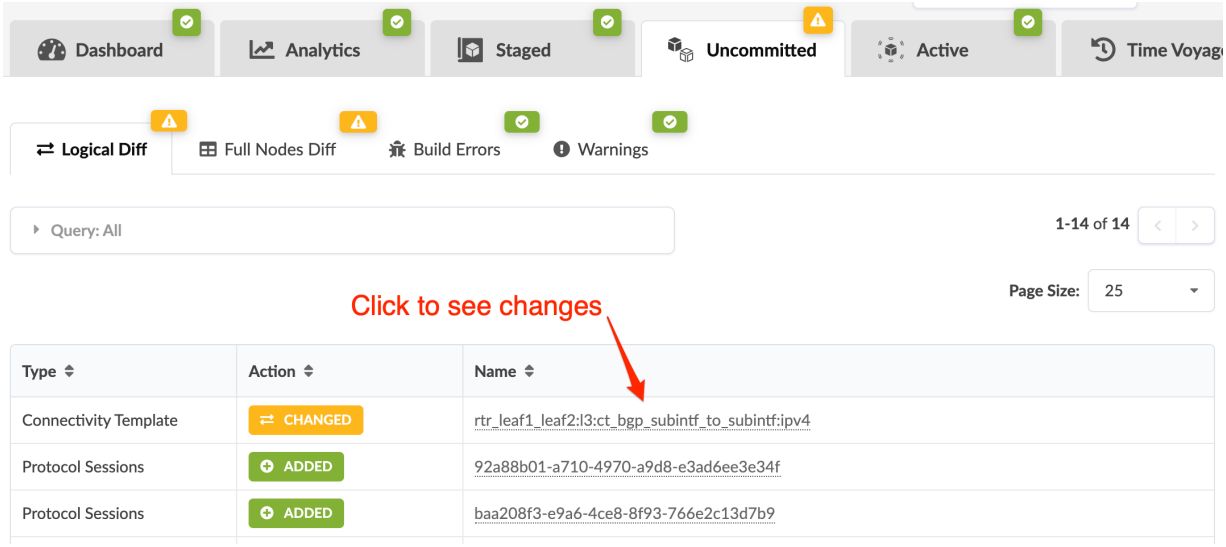
Status Indicator Color	Description
Red	The blueprint needs staging or has **Build Errors** that must be resolved before you can commit.
Orange	The blueprint has **Warnings** to notify you of potential issues. The blueprint may or may not have staged changes. You can commit to a blueprint that has warnings and pending changes.
Yellow	The blueprint has pending changes that you can commit to the blueprint.
Green	The blueprint does not have any pending changes, warning, or errors. The blueprint is active and there is nothing to commit.

Review Staged Changes

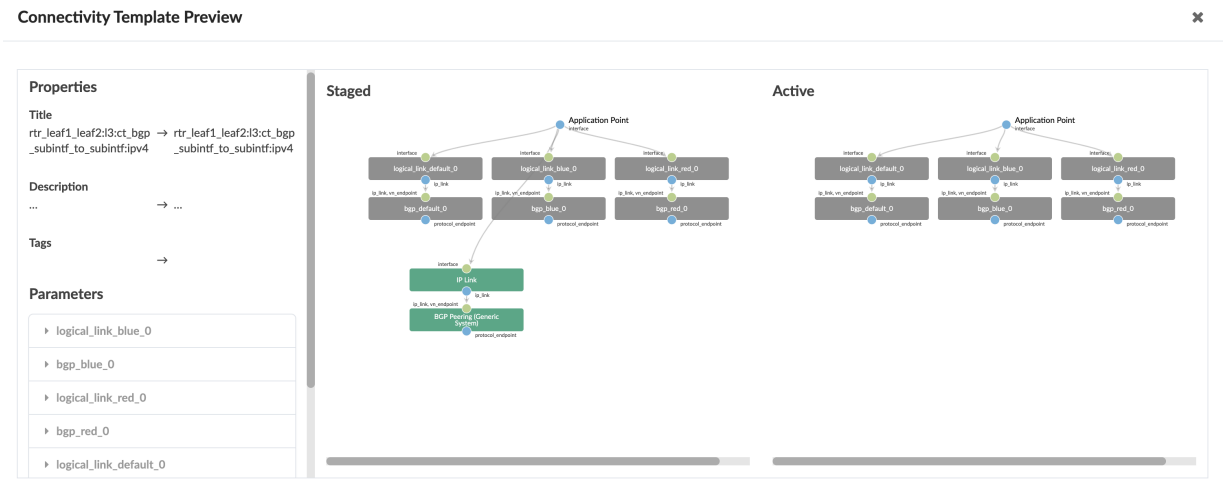
From the blueprint top menu, click **Uncommitted** to go to pending changes. You can review **Logical Diff**, **Full Nodes Diff**, **Build Errors**, **Warnings**, and **Commit Check** for Junos devices. When you're finished reviewing your changes and you've resolved any build errors, proceed to commit your changes to the blueprint or discard them, as applicable. See below for more information about each section.

Logical Diff

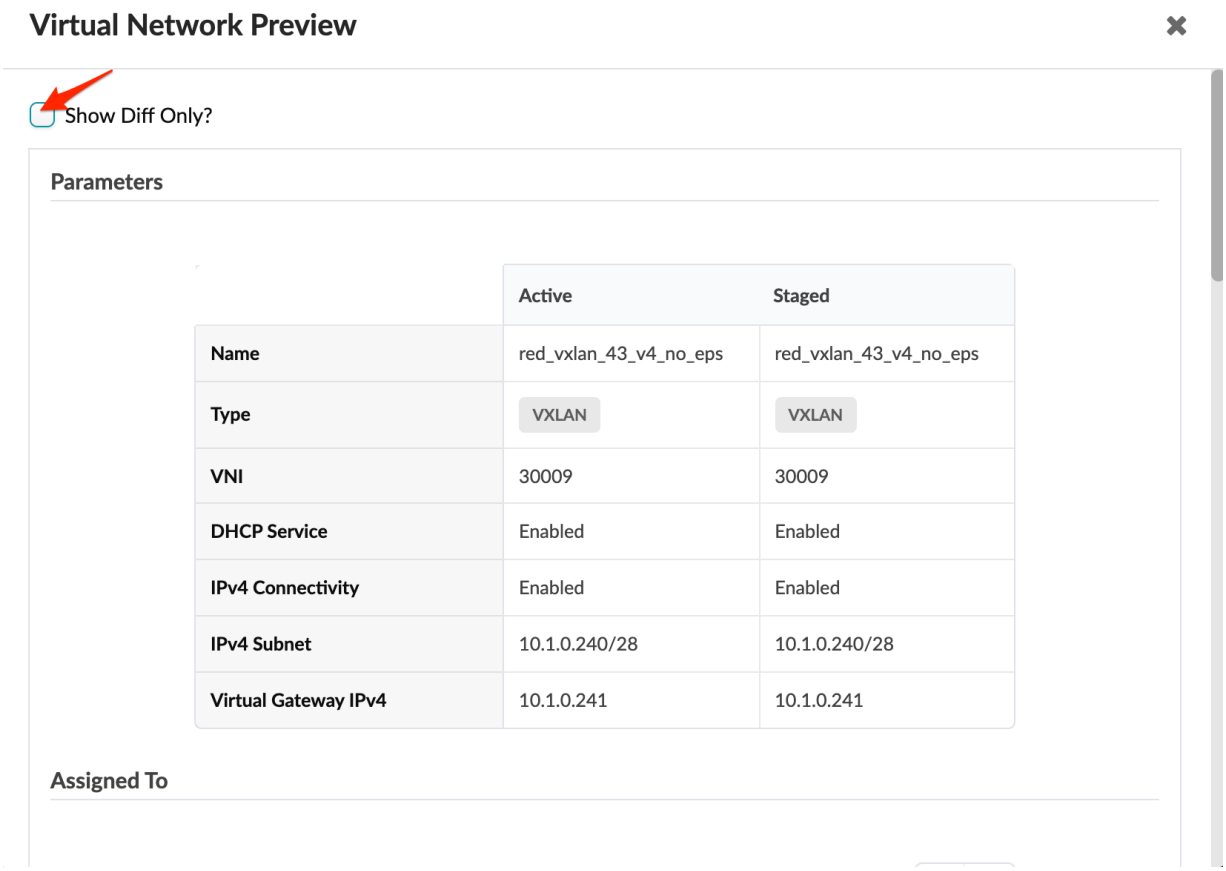
From **Logical Diff**, click a name from the **Name** column to see detailed changes, additions or deletions for that element. (The screenshots below are for a previous Apstra version, which looks slightly different from the current version.)

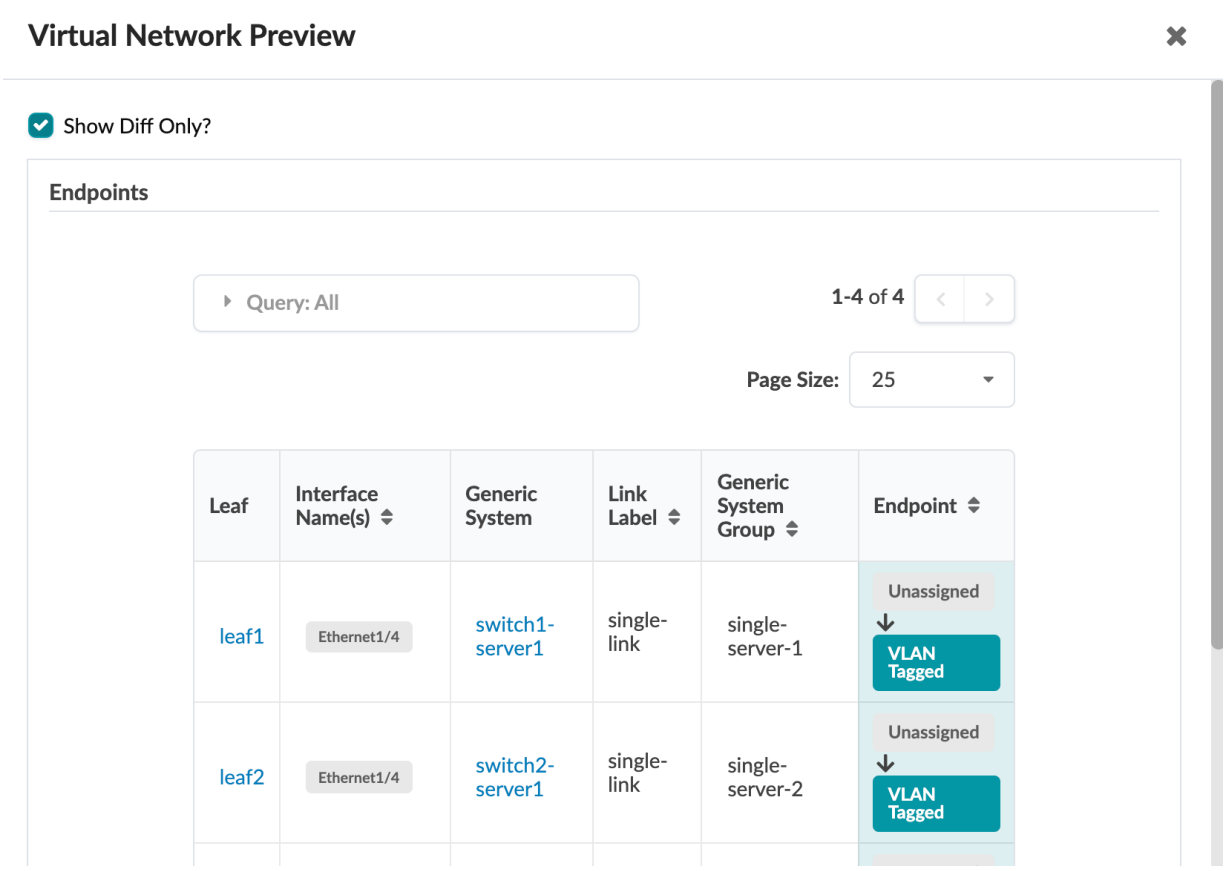


Details for the selected change.



In some cases, you have the option of viewing only the differences, as shown below.





The preview for config template changes is color-coded to easily see the content that has been added (in green) and the content that has been removed (in red).

Full Nodes Diff

Full nodes diff shows all uncommitted changes in one place, organized by node type, change type and raw data. You can sort and search the diffs, then preview the changed element. Full nodes diff requires a fair amount of resources and time to generate.

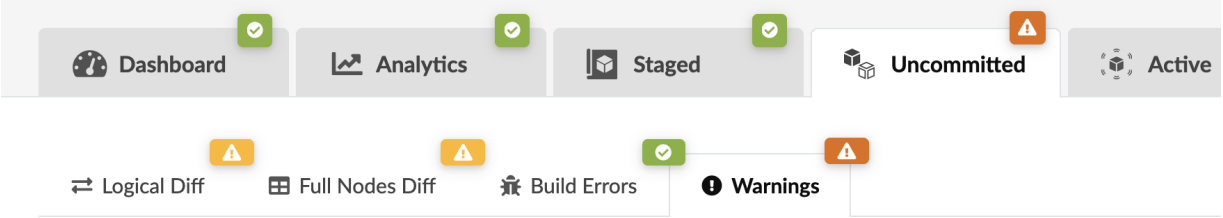
Build Errors

Build errors indicate issues with your blueprint that must be resolved before you can commit to your blueprint. When the issues are resolved, the indicator changes from red to yellow (or orange if you also have warnings), then you can commit to the blueprint.

Warnings

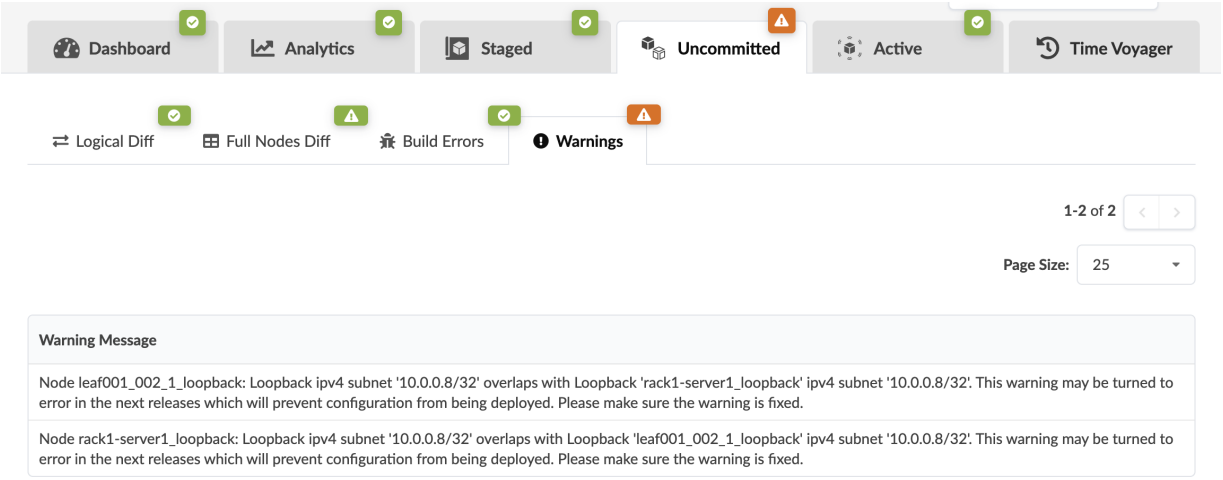
Warnings indicate potential issues with your blueprint. You're not prevented from committing changes to a blueprint with warnings, but it's best to address the issues before proceeding.

The blueprint below has warnings and pending changes. You *can* commit these changes.



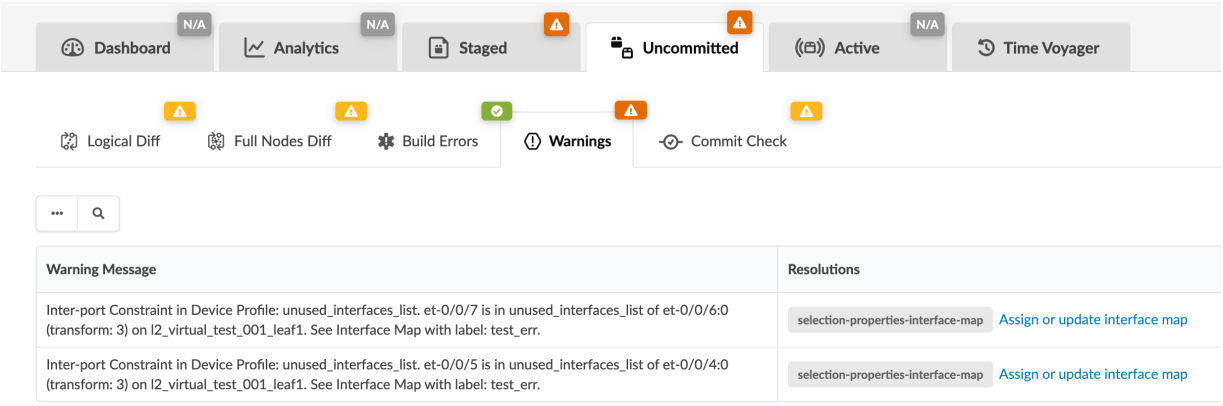
Warning Message
Node leaf001_002_1_loopback: Loopback ipv4 subnet '10.0.0.8/32' overlaps with Loopback 'rack1-server1_loopback' ipv4 subnet '10.0.0 error in the next releases which will prevent configuration from being deployed. Please make sure the warning is fixed.
Node rack1-server1_loopback: Loopback ipv4 subnet '10.0.0.8/32' overlaps with Loopback 'leaf001_002_1_loopback' ipv4 subnet '10.0.0 error in the next releases which will prevent configuration from being deployed. Please make sure the warning is fixed.

The blueprint below has warnings and no pending changes. There is nothing to commit.



Warning Message
Node leaf001_002_1_loopback: Loopback ipv4 subnet '10.0.0.8/32' overlaps with Loopback 'rack1-server1_loopback' ipv4 subnet '10.0.0.8/32'. This warning may be turned to error in the next releases which will prevent configuration from being deployed. Please make sure the warning is fixed.
Node rack1-server1_loopback: Loopback ipv4 subnet '10.0.0.8/32' overlaps with Loopback 'leaf001_002_1_loopback' ipv4 subnet '10.0.0.8/32'. This warning may be turned to error in the next releases which will prevent configuration from being deployed. Please make sure the warning is fixed.

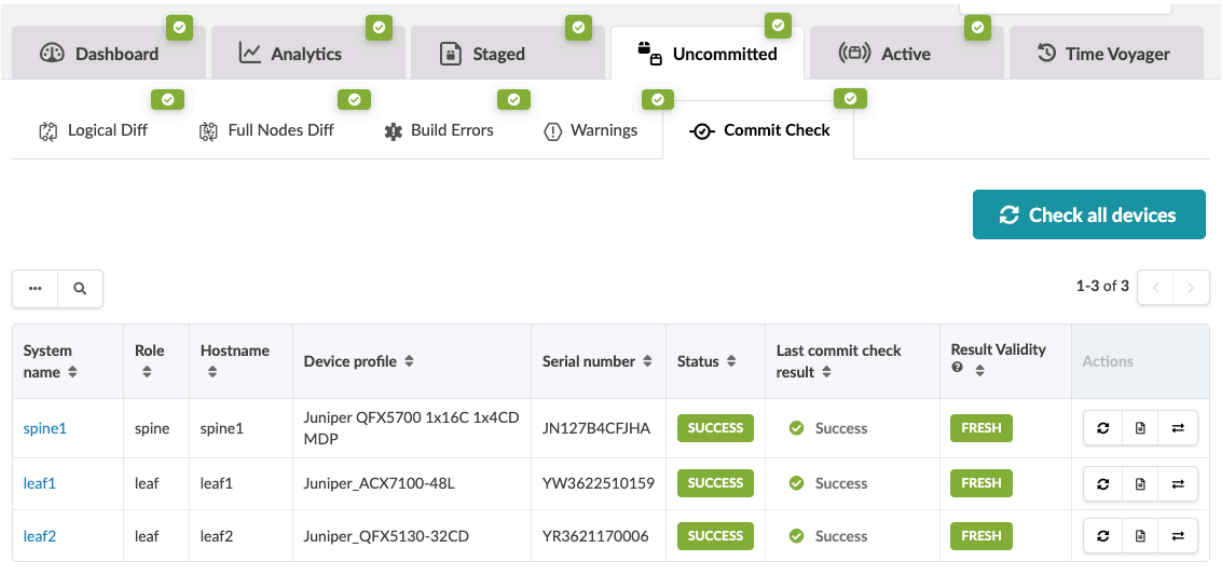
You can validate port assignments on Junos EVO switches only. This enhancement allows Apstra to proactively alert you to potential port issues arising from hardware constraints before pushing configuration. When you attempt to use two or more incompatible ports, the system flags it as a warning. The incompatibility occurs when the speed of one port (transformation) prevents the use of one or more other ports. The screenshot below shows an example.



You can resolve this issue by changing the interface map. Click **Assign or update interface map** in the **Resolutions** column to go to **Physical > Topology**. You can update the interface map in two ways. The best way is to add a new link to the interface that shouldn't be used, then delete the problem interface. Alternatively, you could go into the global catalog, make updates to the interface map, and import it back into the blueprint.

Commit Check

You can check configuration for semantic errors and omissions before deploying Junos OS and Junos Evolved devices, from the **Commit Check** tab.



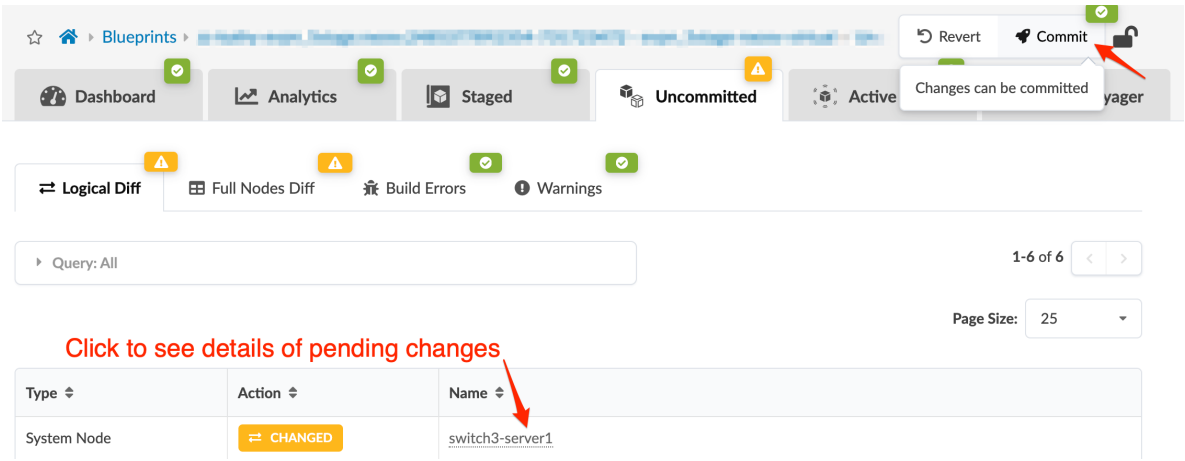
RELATED DOCUMENTATION

Commit / Revert Changes to Blueprint | 599

Commit / Revert Changes to Blueprint

When the **Commit** button on the **Uncommitted** tab becomes clickable, Apstra has validated that requirements are met and you can activate your blueprint changes.

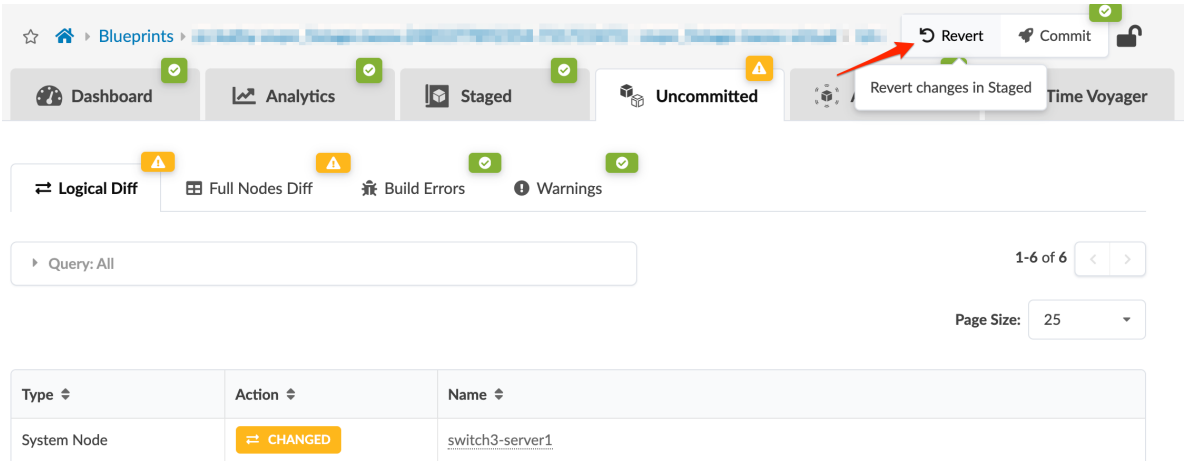
1. From the blueprint top menu, click **Uncommitted**.



2. Review changes, as needed. Click a name to see details.

   [IMAGE OF A CHANGE DETAIL - ACTIVE VS STAGED - SEE topic-map/uncommitted for other images to capture]

3. If you decide to discard the blueprint changes, click **Revert**. In some cases, you might also need to reset resource group overrides. See the note in "Assign ASNs and IP Addresses" on page 58.



4. If you decide to activate the blueprint changes, click **Commit** and add a description. We recommend that you enter the optional revision description to identify changes. These descriptions are displayed in the **Revisions** section of Time Voyager where you can roll back to a previous network state. If you don't add a description now you can always add one later. If you need to roll back to a previous revision, this description helps to determine the appropriate revision. Specific diffs between revisions are not displayed, so the description is the only change information available for that revision.

[IMAGE SHOWING COMMIT BUTTON TOOLTIP]

5. Click **Commit** to push the staged changes to the active blueprint and create a revision. The Apstra engine validates all commits and makes sure everything works as it pushes configuration. Cabling anomalies may appear until validation is complete.

   [IMAGE SHOWING ACTIVE TASKS]

6. While the task is active, you can click **Active Tasks** at the bottom of the screen for information about task progress. (Additional task history is available in the blueprint at **Staged > Tasks**.)

When a blueprint has been committed and devices have been deployed, the network is up and running. However, networks are not static and can require modifications as they evolve. Due to Juniper Apstra's approach of the *network as a single entity* this is extremely easy; all required device configurations are generated and pushed to the devices when you commit the change.

RELATED DOCUMENTATION

Time Voyager Introduction | 627

PART 7

Active Datacenter Blueprints

Active (Datacenter Blueprint)

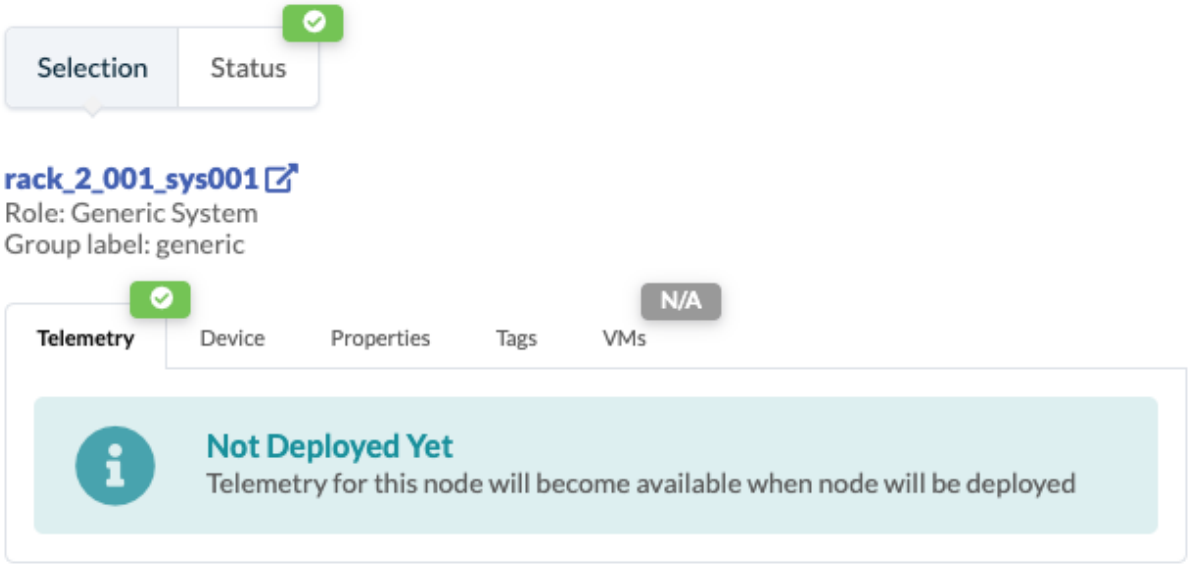
IN THIS SECTION

Active Blueprint Overview

When you deploy your network (by committing the staged blueprint), network status and other details are shown in the **Active** view. From here, you can monitor your network and see any anomalies at-a-glance. You can filter alerts and anomalies by different layers to conduct root cause analysis of problems.

Selection Panel

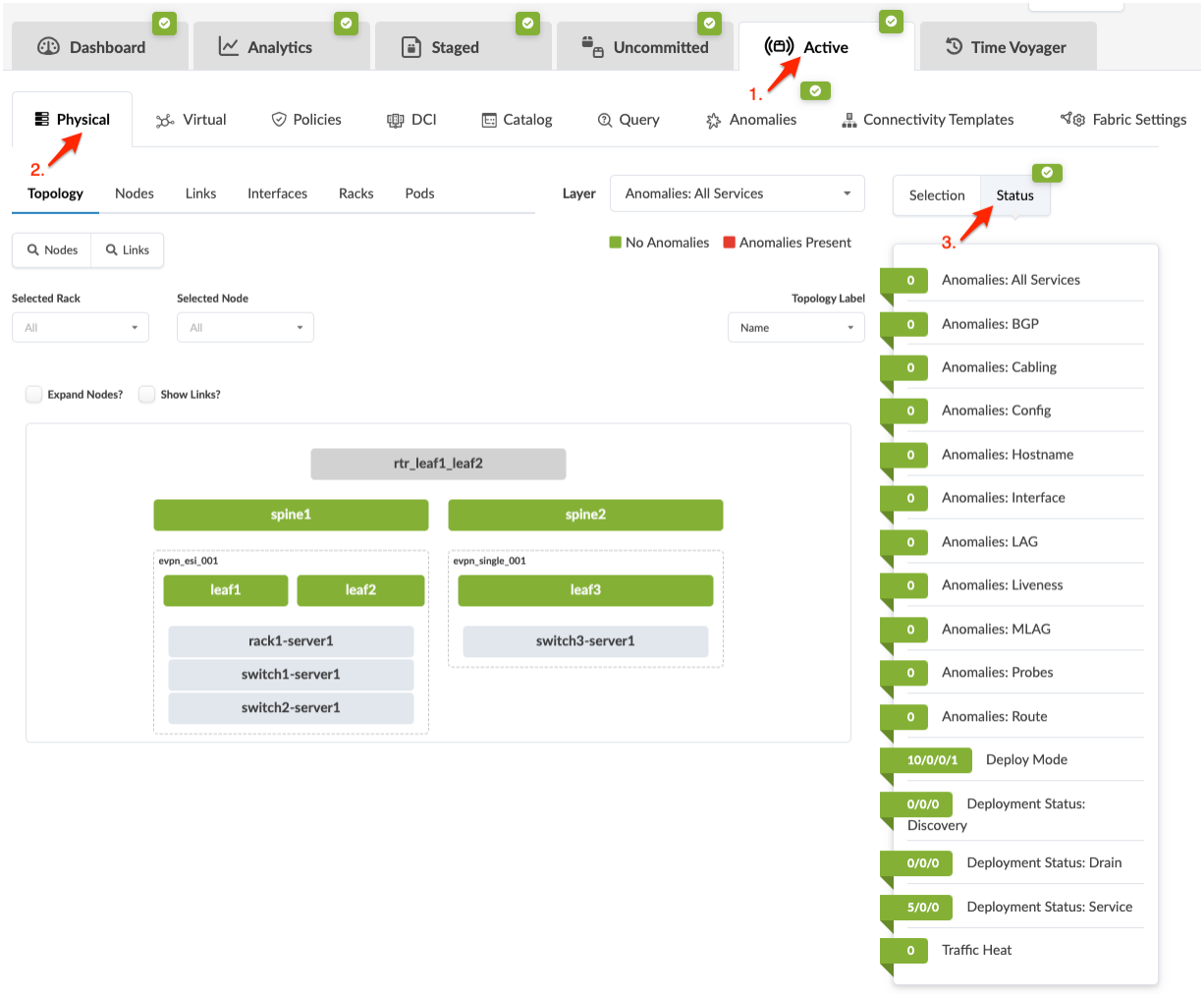
When you select a node in the active **Topology** or **Nodes** view, information about telemetry, device, properties, tags, and VMs for that node are available in the right **Selection** panel.



When you select a link in the active **Topology** or **Links** view, properties and tags information for that link is available in the right **Selection** panel.

Status Panel

From the blueprint, navigate to **Active > Physical** to go to the statuses for services and deploy modes, deployment statuses for discovery, drain and service, as well as traffic heat.



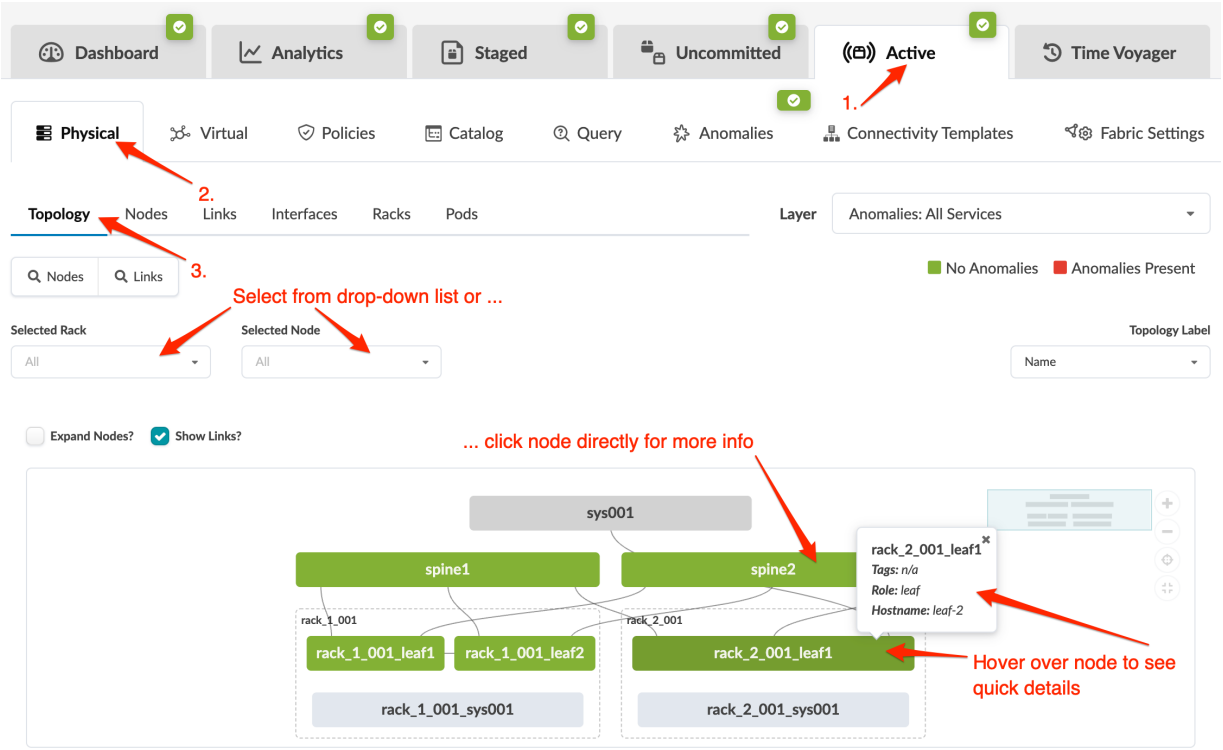
Topology (Active)

IN THIS SECTION

You can look at topologies as 2D views or 3D views. When you select a node from a topology view (by clicking its element in the topology, or by selecting it from the **Selected Nodes** drop-down list), details for the selection are displayed. You can view the selection to show neighbors, links, virtual network endpoints (as of Apstra version 4.0.1), or headroom. Telemetry and other device properties are displayed in the selection panel on the right side of the window.

Topology View (Active)

From the blueprint, navigate to **Active > Physical > Topology**.

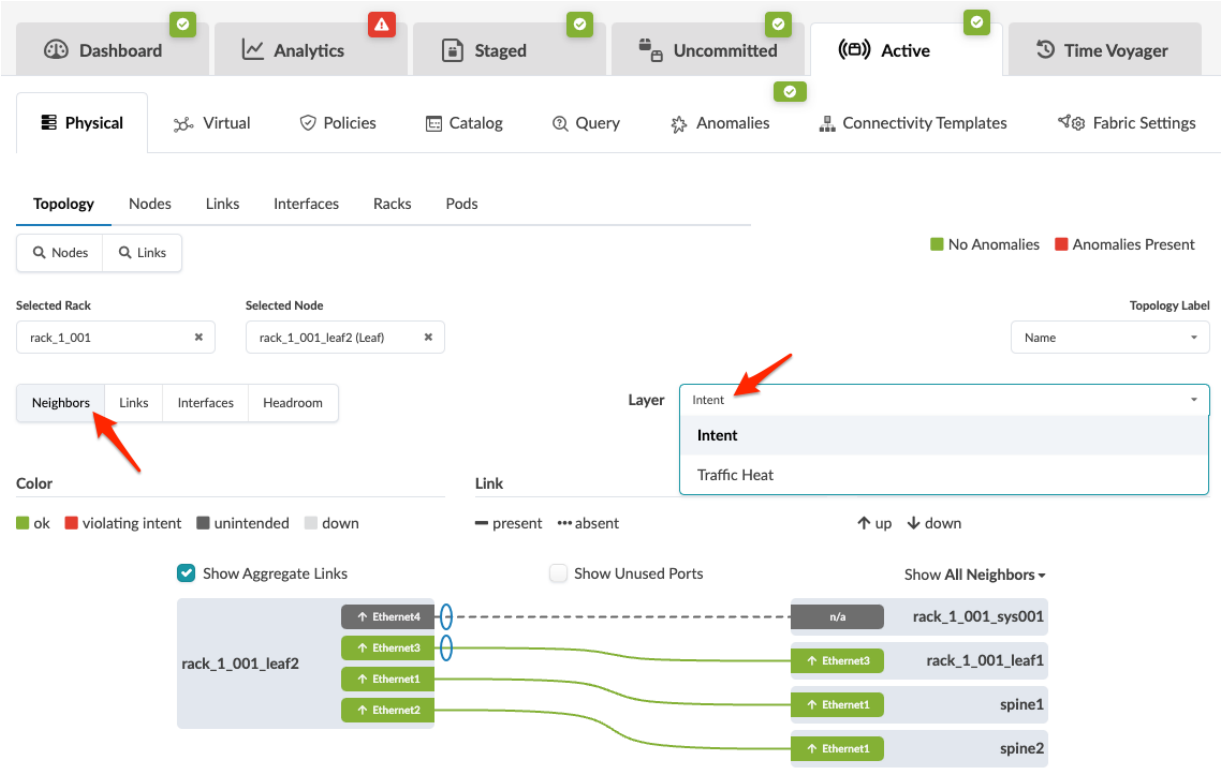


- To make topology elements larger, click the **Expand Nodes** check box.
- To show the links between elements, click the **Show Links** check box.
- To show node name, hostname, role, and tags as applicable, hover over an element.
- To display a different label (name, hostname, S/N), select a different label from the **Topology Label** drop-down list.
- To show rack details, select a rack by either clicking its element or by selecting it from the **Selected Rack** drop-down list.
- To show node details, select the node by either clicking its element in the topology or by selecting it from the **Selected Node** drop-down list.

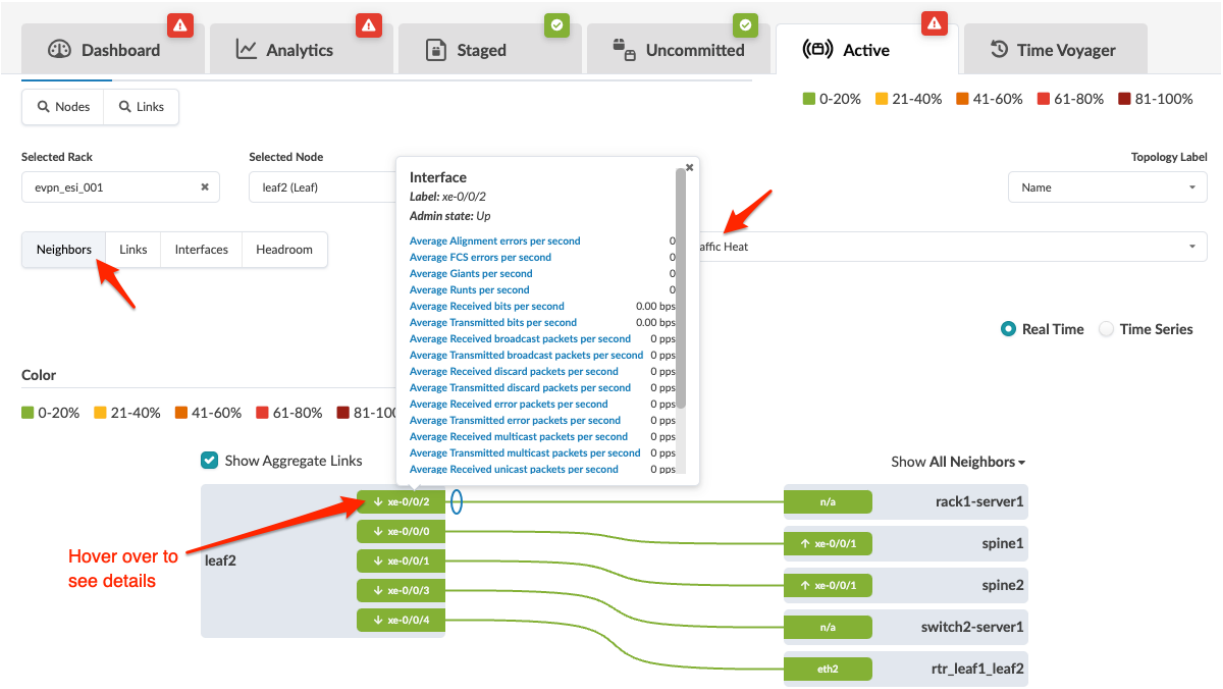
Neighbors View (Active)

- To show aggregate links, click the **Show Aggregate Links** check box.
- To show unused ports, click the **Show Unused Ports** check box.
- To show a different label (name, hostname, S/N), select a different label from the **Topology Label** drop-down list (right side).

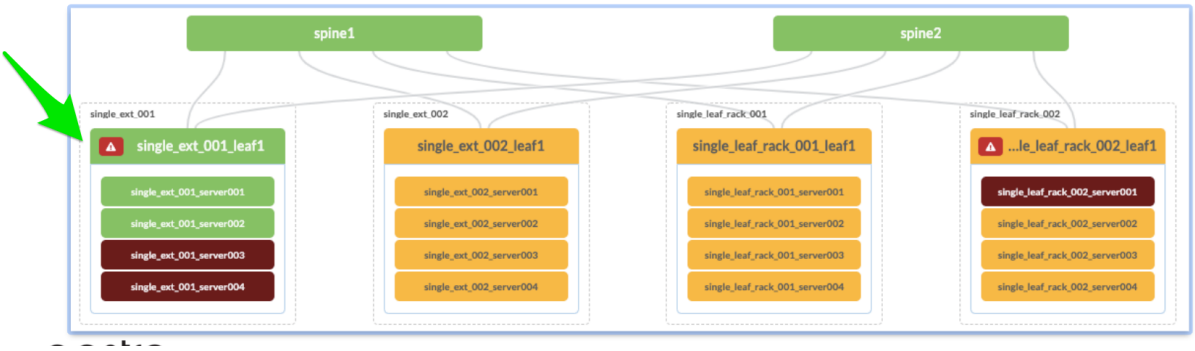
- To show a different layer, select a different layer from the **Layer** drop-down list.
- Choose to show all neighbors or only specific ones (generic, leaf, spine, and so on).



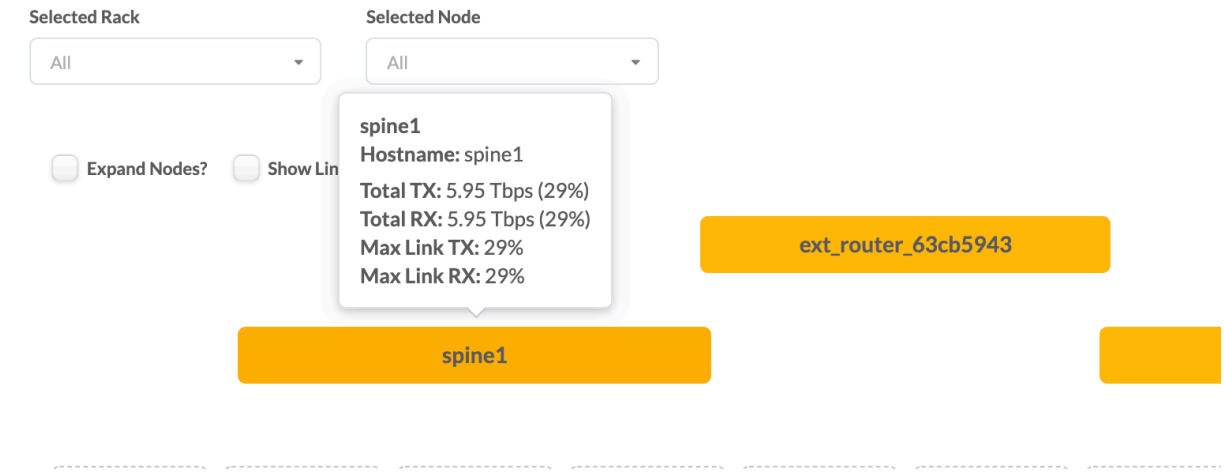
The traffic heat layer is shown below. The colors represent different available/used capacity based on the current system level TX/RX, averaged to 2 minutes, by default. If the aggregated TX or RX across all the device interfaces is < 20% it's **green** . If it's between 21-40%, it's **yellow** and so on. For each 20% difference, capacity is shown with a different color. (Server color is calculated based on the interface counters of the leaf ports facing that server. To see RX/TX per interface for a single node, click the node.



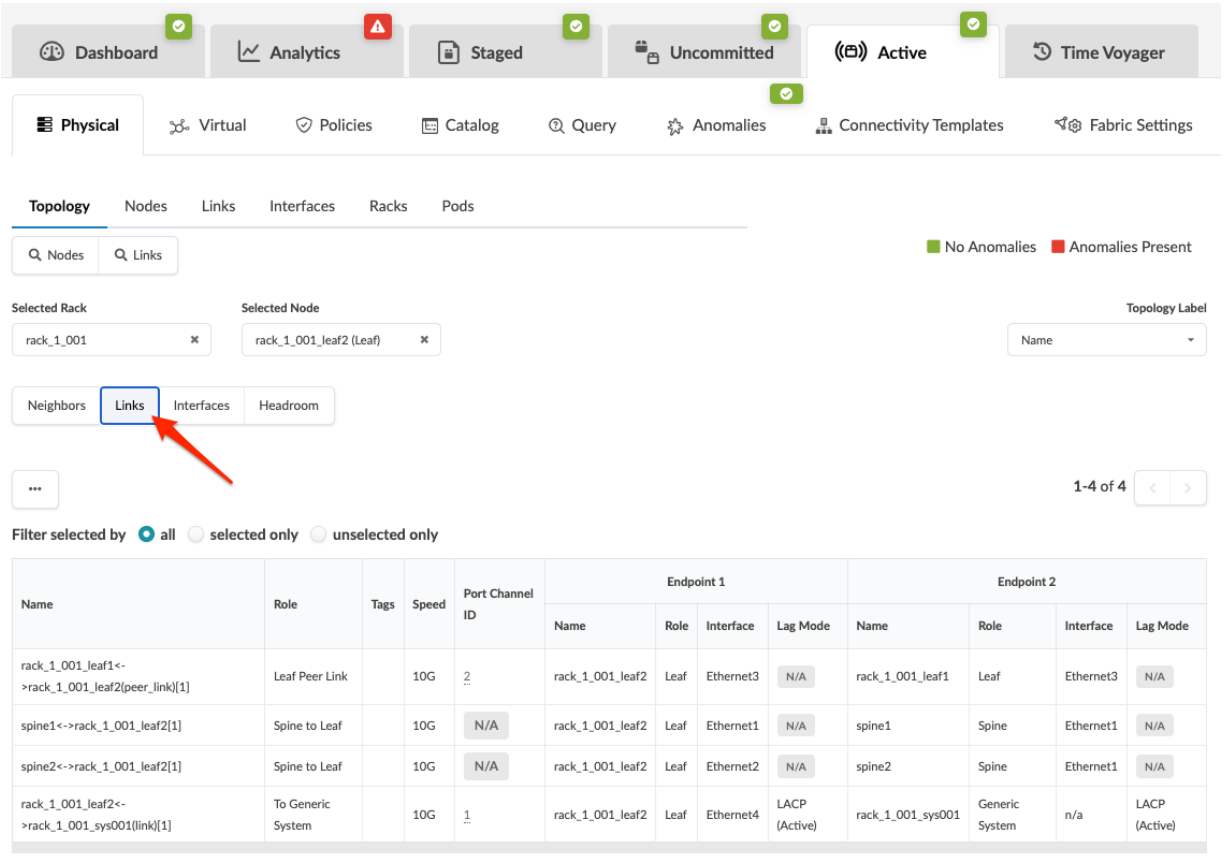
If any of a device's deployed ports are > 81% of its capacity in either RX or TX, a new "Alert" icon is shown on the device. (The screenshot below is for Apstra version 4.2.0, which may be slightly different from newer versions.)



Hovering over a node shows exact aggregated values.

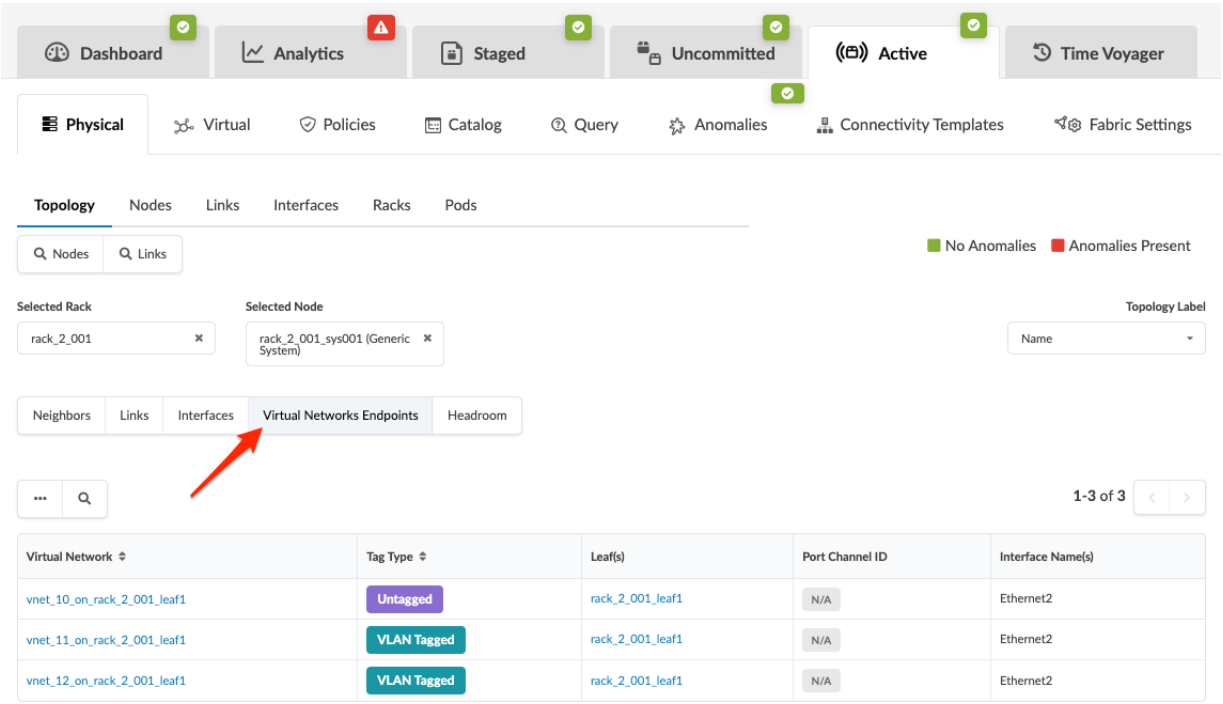


Links View (Active Topology)



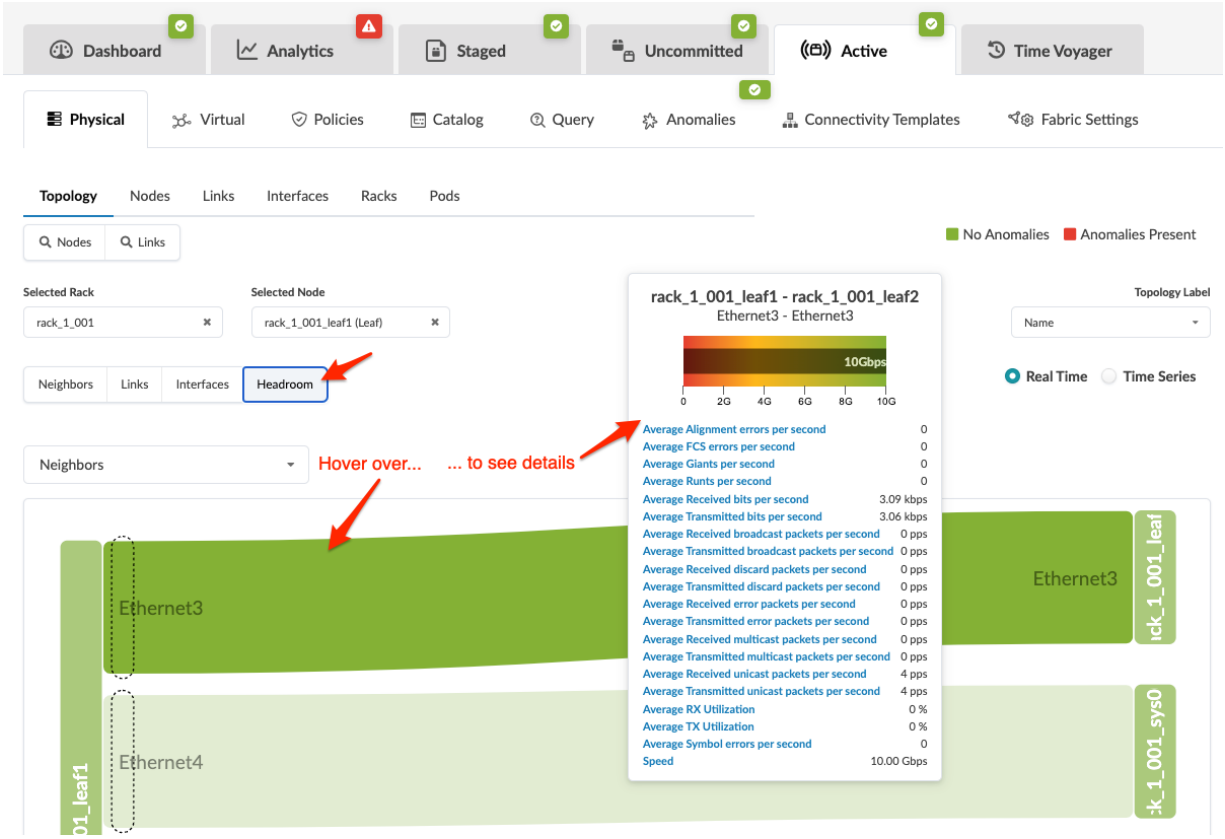
Name	Role	Tags	Speed	Port Channel ID	Endpoint 1				Endpoint 2			
					Name	Role	Interface	Lag Mode	Name	Role	Interface	Lag Mode
rack_1_001_leaf1<->rack_1_001_leaf2(peer_link)[1]	Leaf Peer Link		10G	2	rack_1_001_leaf2	Leaf	Ethernet3	N/A	rack_1_001_leaf1	Leaf	Ethernet3	N/A
spine1<->rack_1_001_leaf2[1]	Spine to Leaf		10G	N/A	rack_1_001_leaf2	Leaf	Ethernet1	N/A	spine1	Spine	Ethernet1	N/A
spine2<->rack_1_001_leaf2[1]	Spine to Leaf		10G	N/A	rack_1_001_leaf2	Leaf	Ethernet2	N/A	spine2	Spine	Ethernet1	N/A
rack_1_001_leaf2<->rack_1_001_sys001(link)[1]	To Generic System		10G	1	rack_1_001_leaf2	Leaf	Ethernet4	LACP (Active)	rack_1_001_sys001	Generic System	n/a	LACP (Active)

Virtual Networks Endpoints (Active)

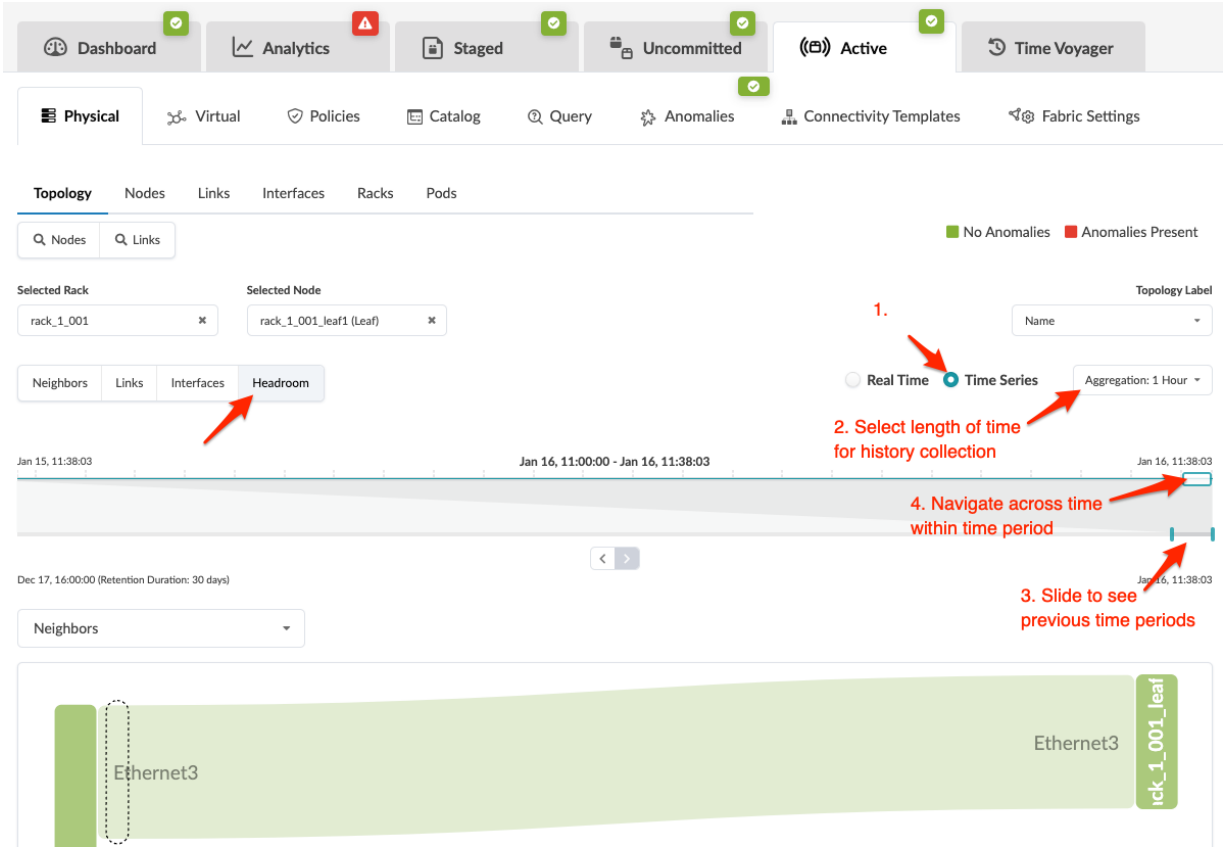


Headroom (Topology)

NOTE: To see the headroom view, the **Device Traffic probe** must be enabled. If you disable or delete the probe, the traffic heat layer in the active topology is not available. For more information, see "Device Traffic probe" on page 1548.



To view traffic history on top of the physical topology from the headroom view, select **Time Series** .



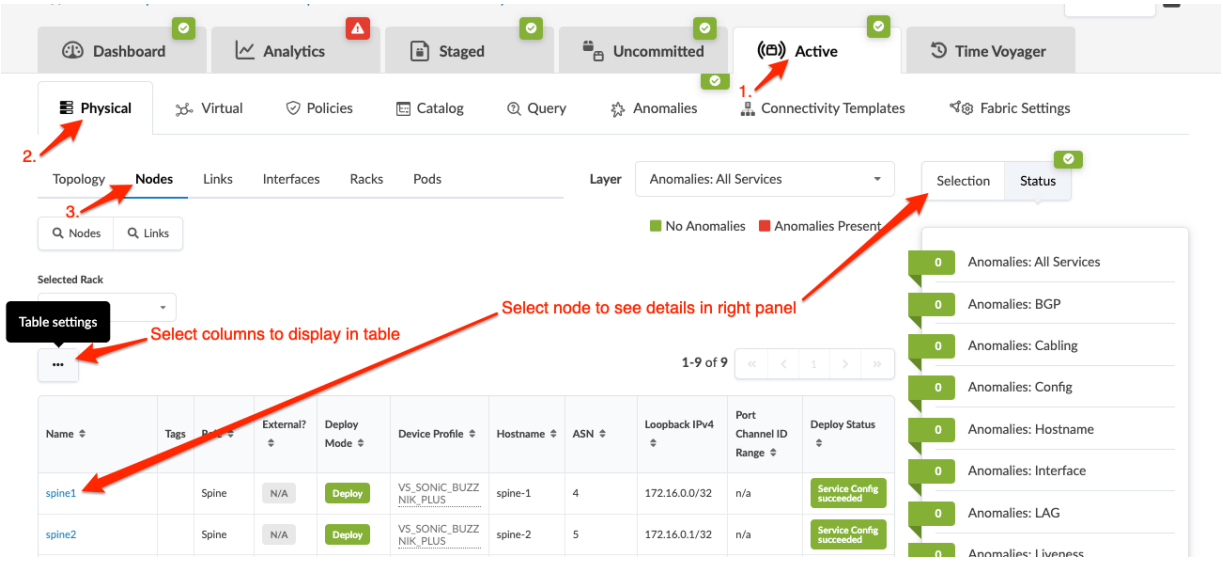
Nodes (Active)

IN THIS SECTION

Active Nodes Overview

From the blueprint, navigate to **Active > Physical > Nodes** to go to nodes in the active topology. You can select which details to display in the table (from table settings). To see additional details (such as

telemetry, properties, tags and virtual) for a specific node, select it, then the right panel displays tabs with more information.



Apply Full Config

> **CAUTION**: Applying a full config is a disruptive operation and results in a temporary loss of service to the device. For information about when to apply a full config, see "Anomalies - Configuration Deviation" on page 616.

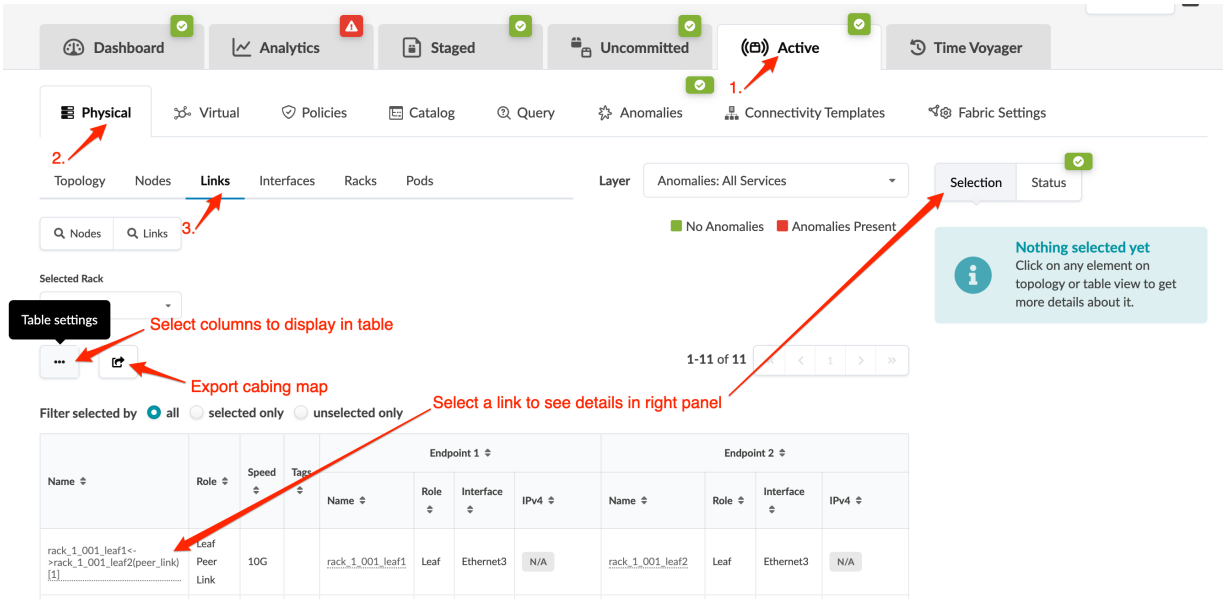
1. From the blueprint, navigate to **Active > Physical > Nodes** and select the device.
2. From the selection panel (right-side) click **Device**, then click **Rendered**, **Incremental**, or **Pristine** to review the different configurations.
3. Click **Apply Full Config**.

Links (Active)

IN THIS SECTION

Active Links Overview

From the blueprint, navigate to **Active > Physical > Links** to go to links in the active topology. To search for specific nodes or links, click its query box, enter your criteria and click **Apply** to go to results. To go to properties of a particular link (in the right panel), click its name.



Export Cabling Map

1. From the blueprint, navigate to **Active > Physical > Links**, click the **Export cabling map** button and select **JSON** or **CSV**.
2. Click **Copy** to copy the contents or click **Save As File** to download the file.
3. When you've copied or downloaded the cabling map, close the dialog to return to the **Links** view.

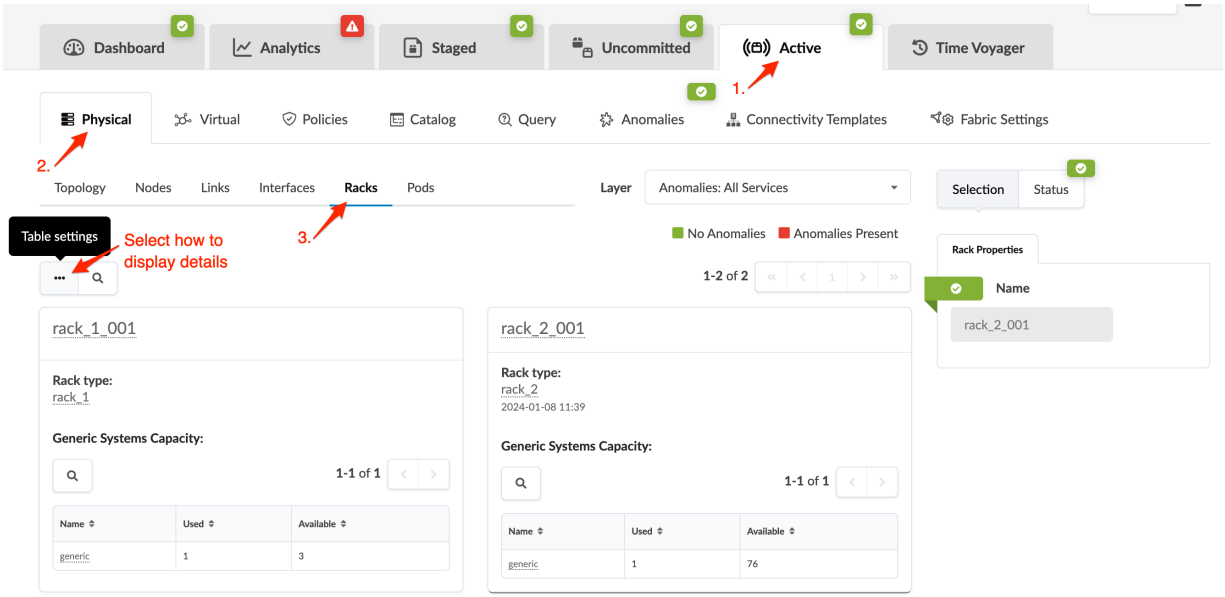
> **NOTE**: Cabling maps can also be exported from the **Staged >Physical >Links** view.

Racks (Active)

IN THIS SECTION

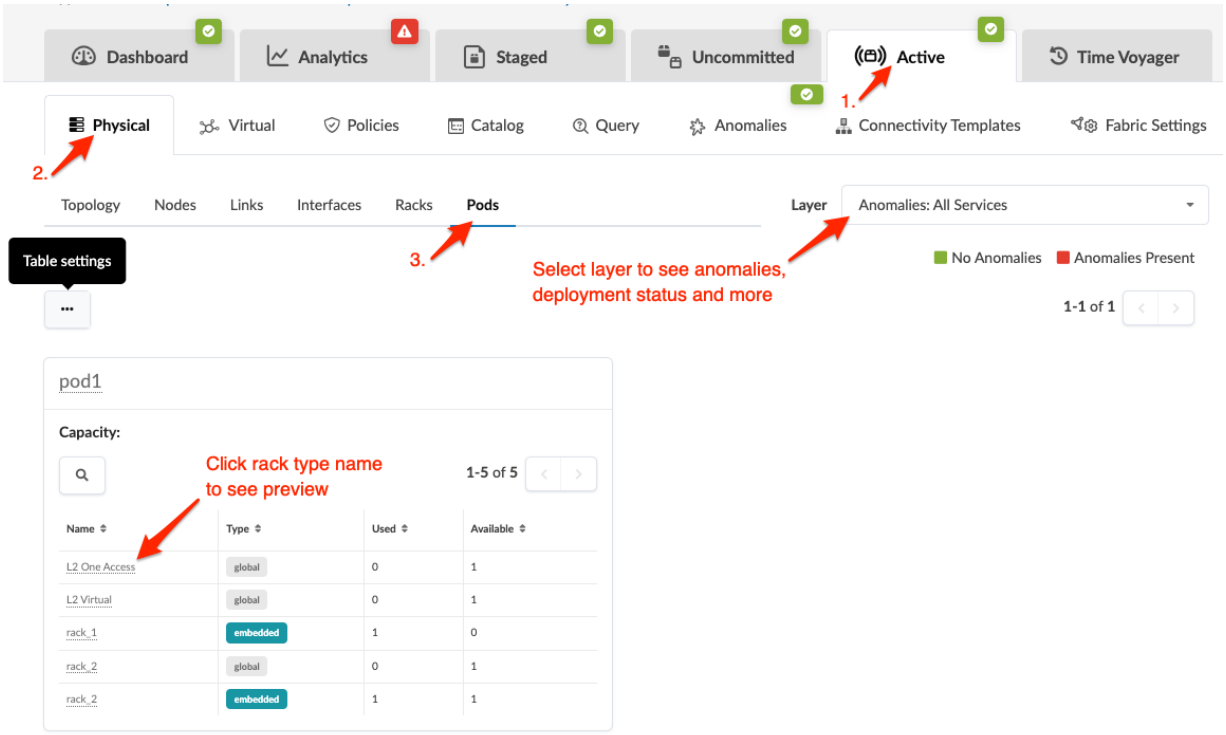
Racks

To go to rack details in the active blueprint, navigate to **Active > Physical > Racks**. You can change the default view from a table to a list in the table settings.



Pods (Active)

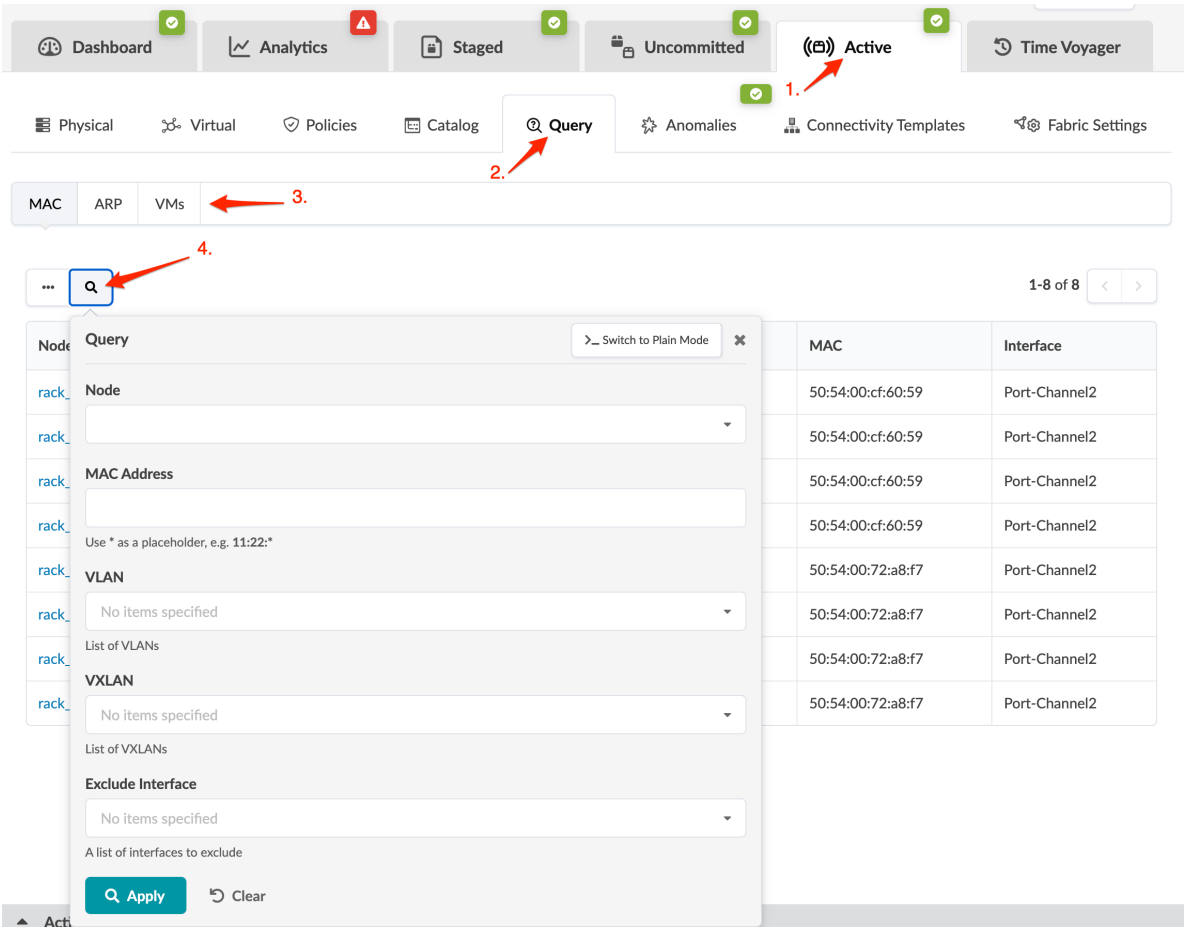
From the blueprint, navigate to **Active > Physical > Pods** to see details about deployed pods. You can search for specific nodes or links and select a layer to see anomalies, deploy modes, deployment status and more. 3-stage topologies have one pod, while 5-stage topologies have two or more pods. Click a rack name in a pod to see its rack type preview.



Query

You can search for MAC addresses, IP addresses and VMs by using the query feature in the active blueprint.

1. From the blueprint, navigate to **Active > Query**.



2. Click **MAC**, **ARP**, or **VMs** depending on your query.
3. Click the query button, enter search criteria, and click **Apply** to see results.

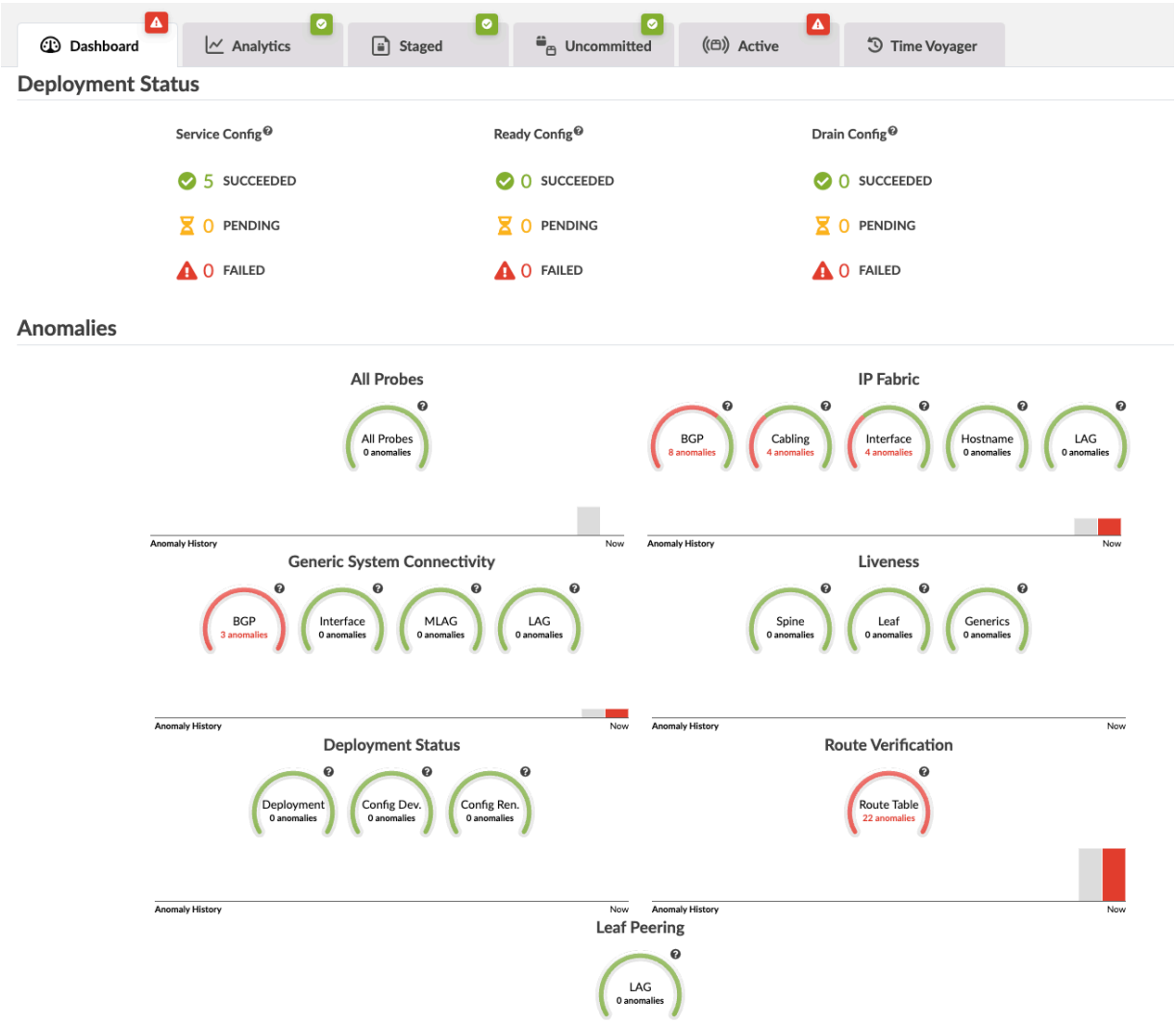
Anomalies (Service)

IN THIS SECTION

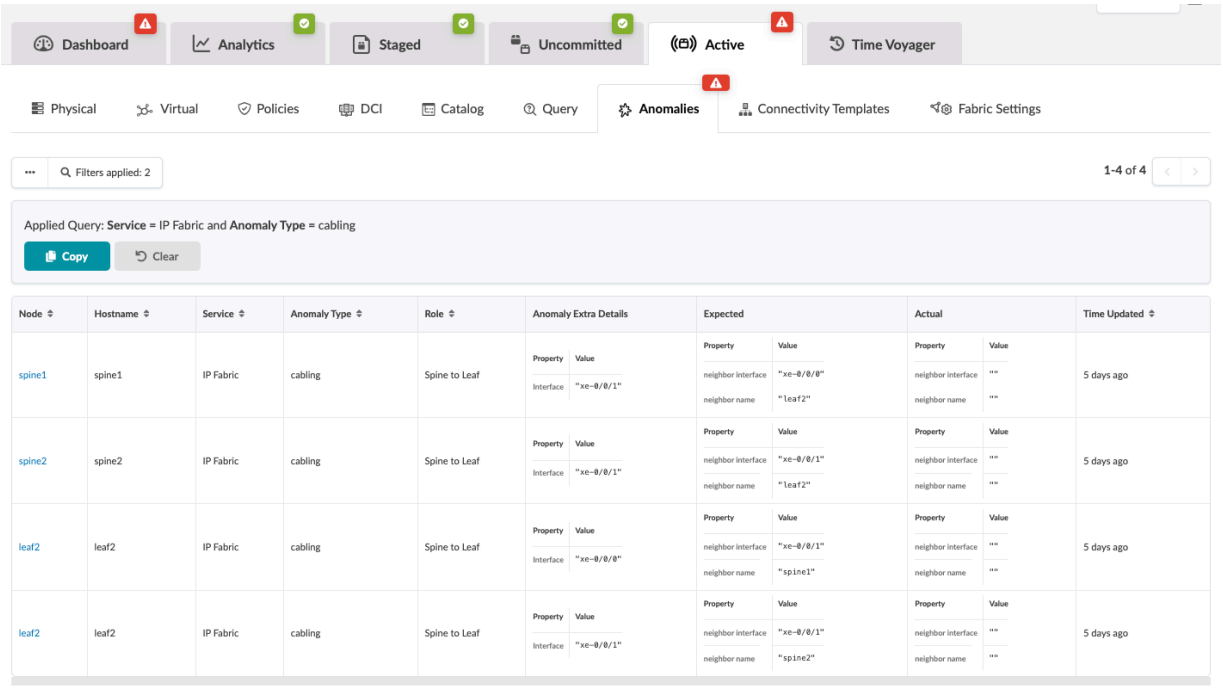
This section covers service anomalies. For analytics anomalies see "IBA Anomalies." on page 27

Discovery Anomalies

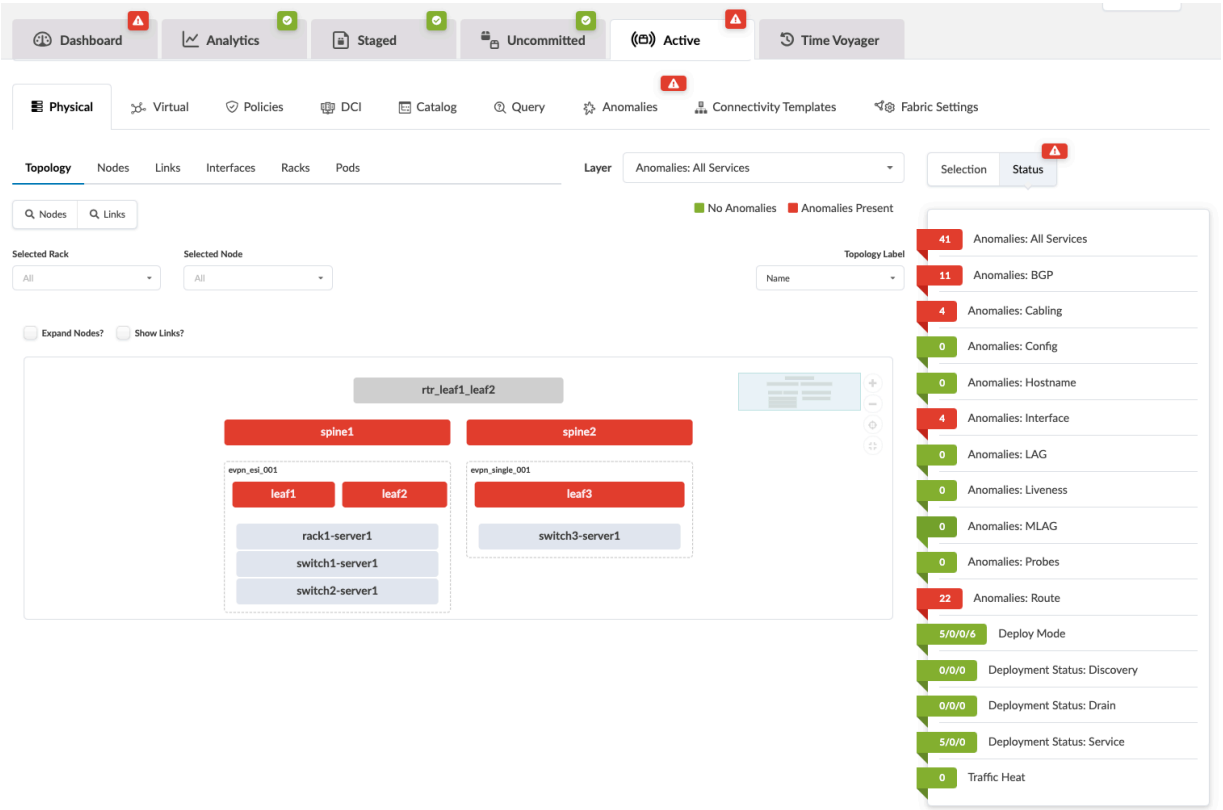
From your blueprint, navigate to **Dashboard** to see a high level view of your network with its different statuses and if/where you have anomalies.



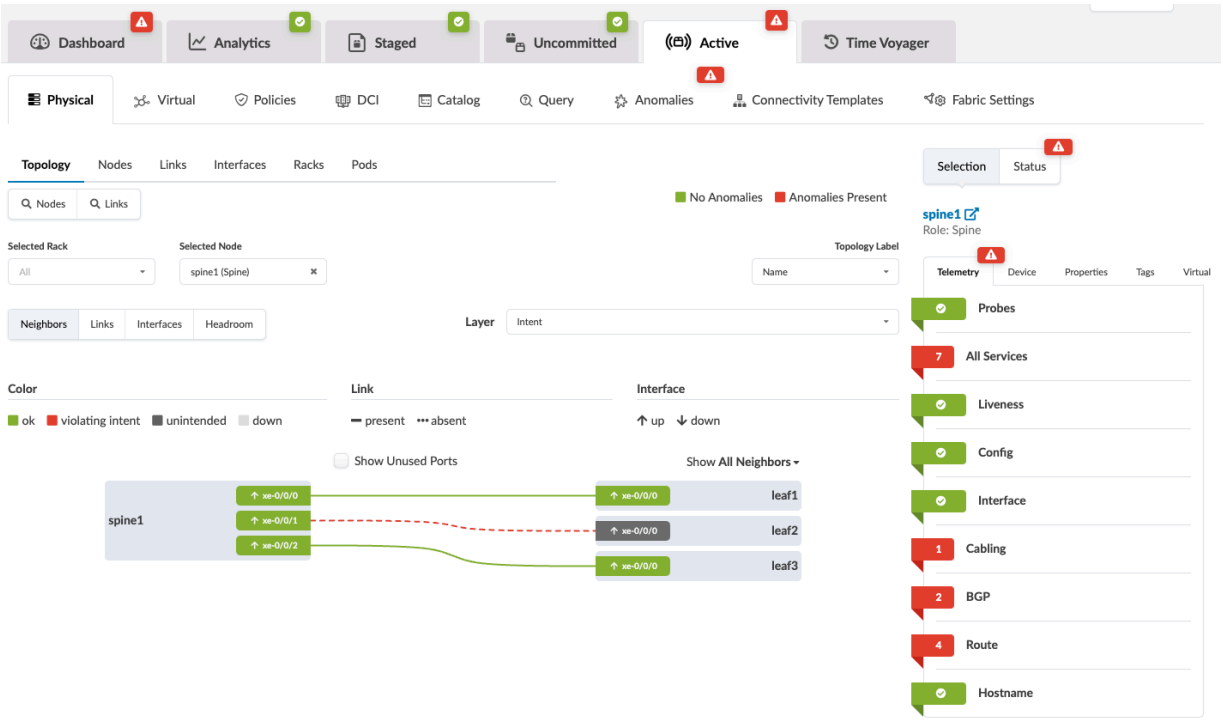
To see anomaly details, click one of the red indicators. The screenshot below is for IP fabric cabling anomalies.



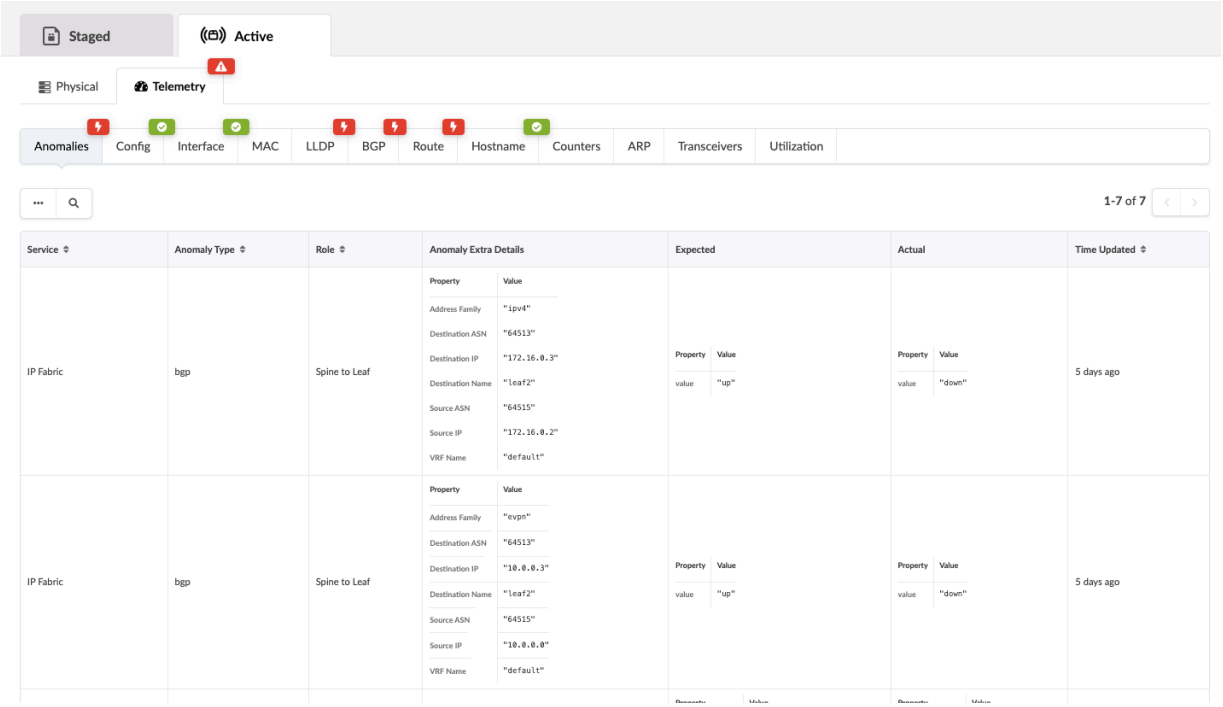
To see the anomalies in the topology view, click **Active**.



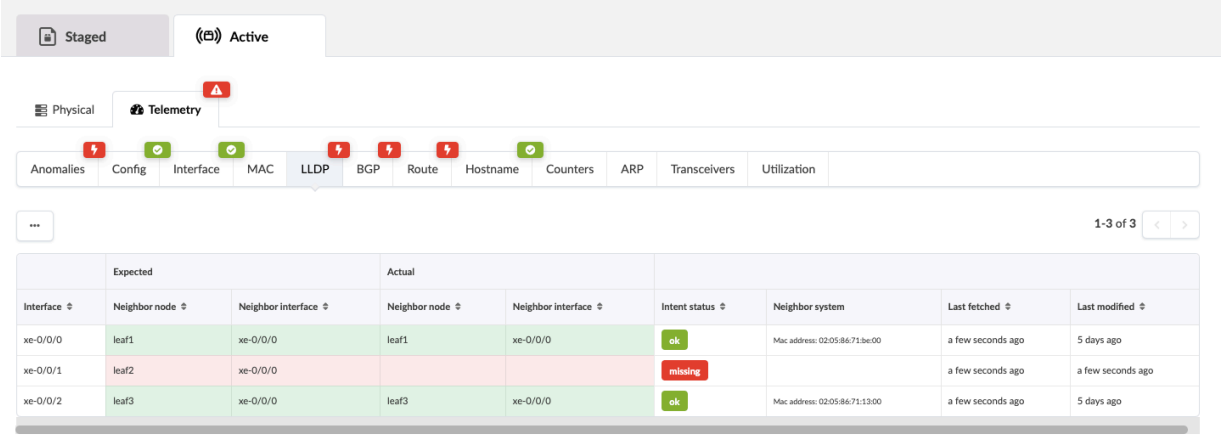
To see the topology view of a selection, click the node in the topology. The screenshot below is for spine1.



To see a comparison of expectations vs. actual, click **All Services** in the right panel. If other anomalies exist in addition to the cabling anomalies (our example), they're shown in this list as well.



To see additional details, click one of the tabs, LLDP for example, click LLDP.



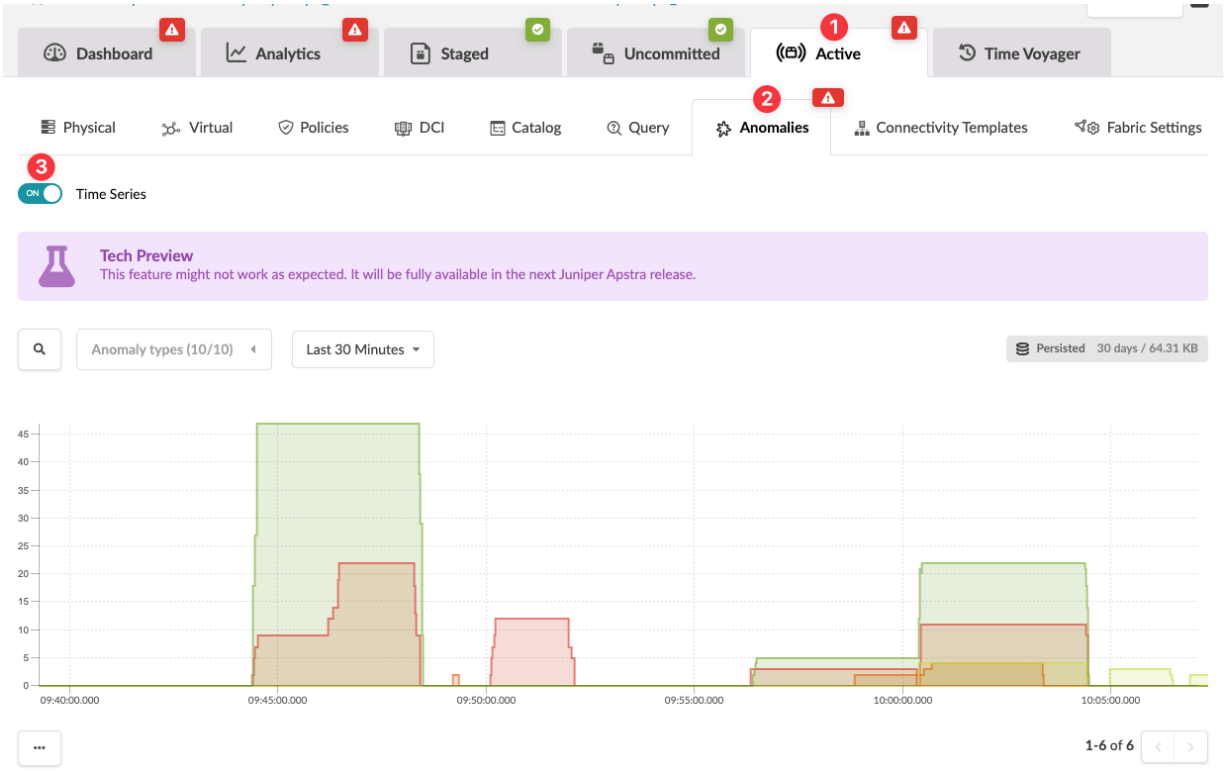
To see how to resolve these cabling issues, see "Fetching Discovered LLDP Data" on page 200.

Blueprint Anomaly History

You can see the history of blueprint anomalies in the **Active** tab of the blueprint (as of Apstra version 5.0.0).

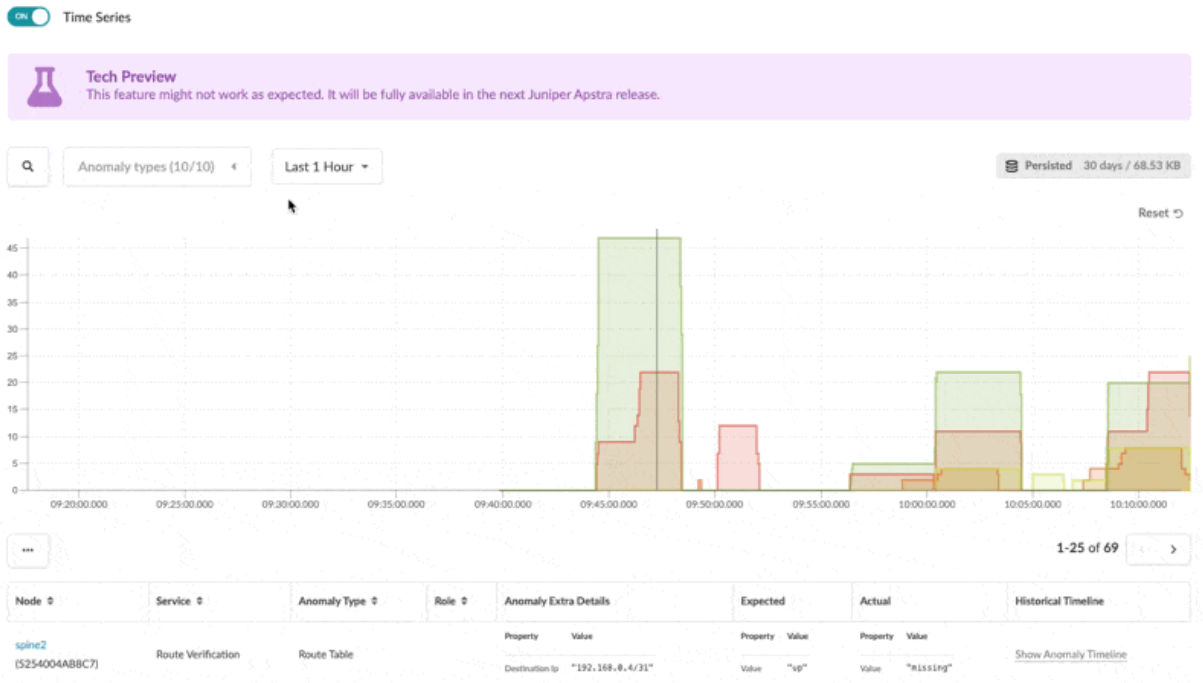
> **NOTE**: This feature is classified as a Juniper Apstra Technology Preview feature. These features are "as is" and voluntary use. Juniper Support will attempt to resolve any issues that customers experience when using these features and create bug reports on behalf of support cases. However, Juniper may not provide comprehensive support services to Tech Preview features. For additional information, refer to the "Juniper Apstra Technology Previews" on page 1781 page or contact "Juniper Support" on page 1374.

From the blueprint, navigate to **Active > Anomalies** and click to toggle ON **Time Series**.



The Anomalies tab now shows a chart of the anomalies count over time, grouped per type of anomaly (BGP, cabling, route and so on). You can zoom in and zoom out to look for a specific time interval in detail.

Selecting an individual anomaly you also can see the historical timeline of the anomaly to show the occurrences of that specific anomaly in the recent history. The default retention period is set to 30 days.



Configuration Deviation

IN THIS SECTION

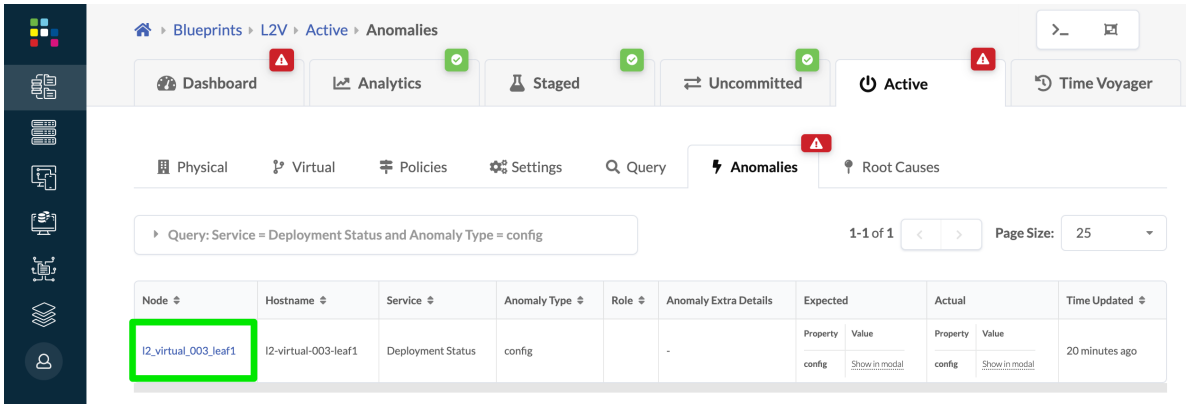
- Config Deviation and Configlets | **625**

Running configurations on devices are continuously compared with the "Golden Config" on page 634. If a config deviation is found, a configuration anomaly is raised. Typically such deviations are seen when changes were made outside of Apstra (from the device CLI), or attempting to deploy configuration on a switch that is not able to take the change. These anomalies remain active until either the anomalous configuration is removed from the device or the anomaly is suppressed. (The screenshots below are from a previous version of Apstra.)

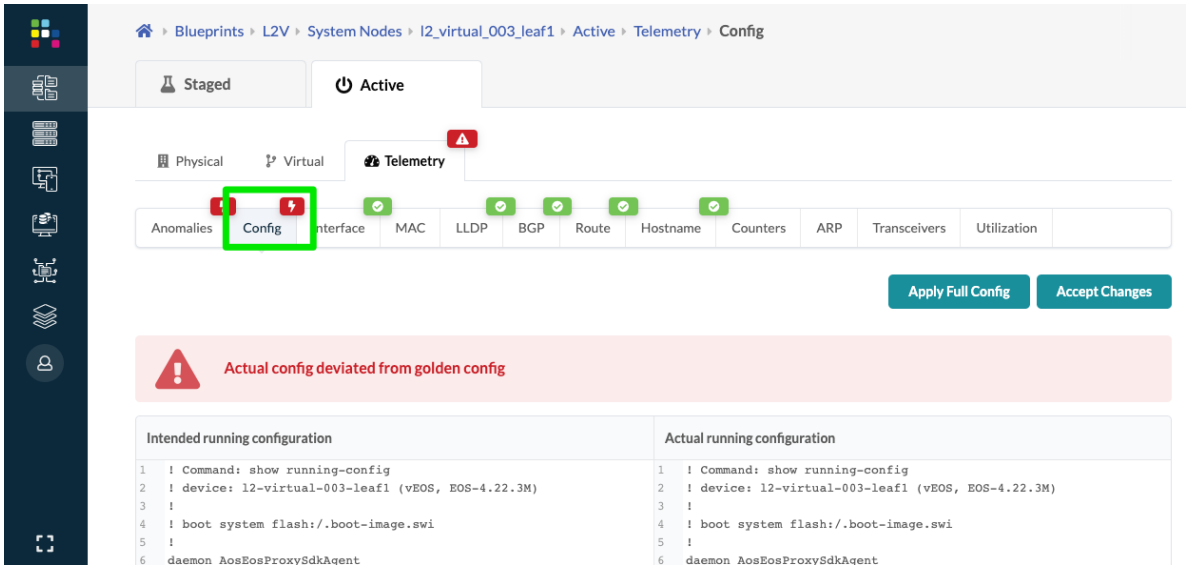
1. From the blueprint dashboard, any configuration deviations are displayed in the **Deployment Status**

section.

2. Click **Config Dev.** to see the list of node(s) with anomalies.



3. Click a node name to see the device telemetry page, then click **Config** to see a side-by-side comparison of the actual config to the golden config. (The difference is not shown in the image below.)



4. To keep the configuration difference, click **Accept Changes**. This **suppresses** the configuration anomaly, and does not affect "Intended" or Apstra-rendered config. the primary purpose of "Accept Changes" is to mitigate *cosmetic* configuration anomalies.

   NOTE: Out-of-band (OOB) changes to the fabric are not supported. Do not **Accept Changes** to attempt to add OOB changes. For custom changes, use "configlets" on page 908.

   CAUTION:

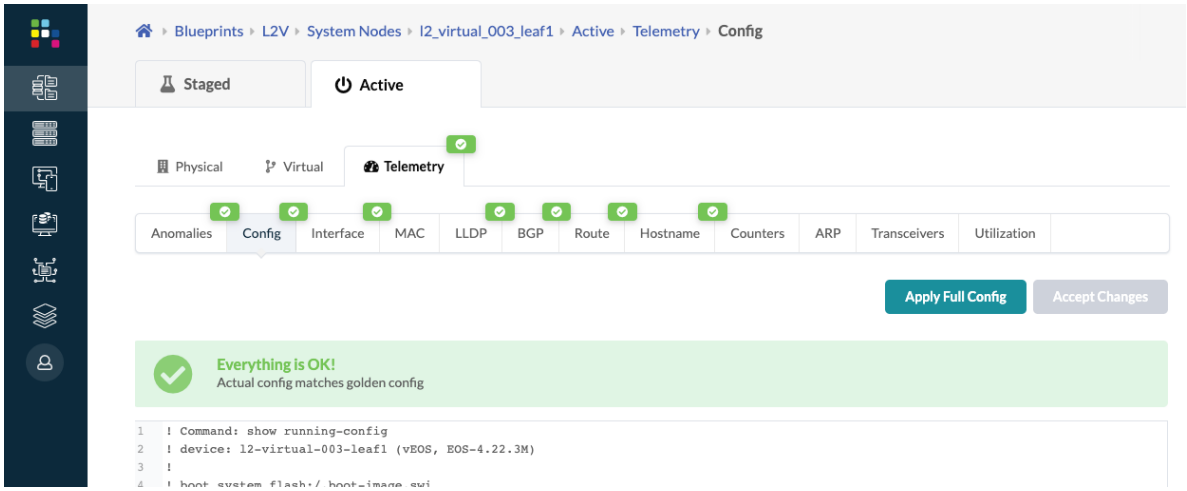
   - Depending on the change, Apstra may overwrite out-of-band changes. There is no way to avoid this. As such, always avoid OOB changes in the Apstra environment.
   - Using *Accept Changes* does **not** make the OOB change persistent. In the event of a full config push or Apstra writing to the same config, all OOB changes are discarded.

5. To make the actual configuration conform to the intended configuration, click **Apply Full Config**, then click **Confirm**. Applying the full config erases the device's current (unintended) configuration before re-applying the complete intended configuration. A full configuration push does not include any OOB changes, and therefore erases them, regardless of their "Accepted" state.

   CAUTION: Applying a full config is a disruptive operation and results in a temporary loss of service to the device.

   CAUTION: Never directly modify any Apstra-rendered config that affects routing and connectivity. Doing so can potentially impact the network's operation. When in doubt, contact "Juniper Support" on page 1374.

6. After resolving the config deviation anomaly (accept changes or apply full config) the actual config matches the golden config and the anomaly is cleared.



Config Deviation and Configlets

If an improperly-configured configlet causes Apstra deployment errors (when the device rejects the command), a **service config deployment** failure occurs. In this case, follow the steps below to resolve the anomaly.

1. From the blueprint, navigate to **Staged > Catalog > Configlets** and delete the configlet.
2. Click **Uncommitted** and commit the change. The configuration deviation remains because the golden config is empty. The golden config is the running config of the device after *successful* deployment of Apstra-rendered config. If deployment fails there is no golden config, thus causing the config deviation.
3. Click **Dashboard**, then click **Config Dev.** (in the **Deployment Status** section).
4. Click the node name, then select **Accept Changes** to notify Apstra that the failure can be ignored.

8

PART

Time Voyager (Blueprints)

Time Voyager Introduction

When you commit a staged blueprint (deploy updates to the network), the result might not be what you expected. Maybe you've committed changes to a blueprint by mistake and you want to undo those changes. Or maybe you've decided to return the network to the state it was in several revisions ago. Depending on the level of complexity, manually staging and committing changes to undo what you've done can be difficult and error-prone. In these cases you'll want to use Time Voyager to quickly restore previous revisions of a blueprint.

You can roll back a blueprint to any retained revision. The 5 most recent blueprint commits are retained, by default. When you commit a sixth time, the first revision is discarded, and the sixth revision becomes the fifth, the second revision becomes the first, and so on as additional blueprint changes are committed. You can change the number of automatically saved revisions to up to 100 revisions. In the **Commit** dialog, a message lets you know that if you've reached your limit and you commit another change, the new revision will replace the oldest auto-saved revision. If you've reached the limit when you want to commit, and you don't want any revisions deleted, you can close the commit dialog without committing, then increase the number of auto-saved revisions in Time Voyager.

You can retain a particular revision indefinitely by *keeping* it, or manually saving it. When you keep a revision it is not included in the 5 revisions that cycle out. You can keep up to 25 revisions, effectively having 30 blueprint revisions to choose from, by default. (If you change the number of automatically saved revisions to the maximum of 100, you could save up to 125 revisions.) Keep in mind that each revision requires storage space. If you decide that you no longer want to keep a revision you can simply delete it.

When committing a blueprint we recommend that you add a revision description to help identify the changes made in that revision. These descriptions are displayed in the revision history section of the blueprint as long as that revision is retained. If you don't add a description when you commit you can always add one later (but you'll need to remember what the changes were). When jumping to a revision (rolling back), this description helps you choose the correct one. Specific differences between revisions are not displayed, so the description is the only change information available for that revision.

When jumping to a revision, any previously staged changes that haven't been committed are discarded. If this is an issue, do not roll back until you've addressed the uncommitted changes.

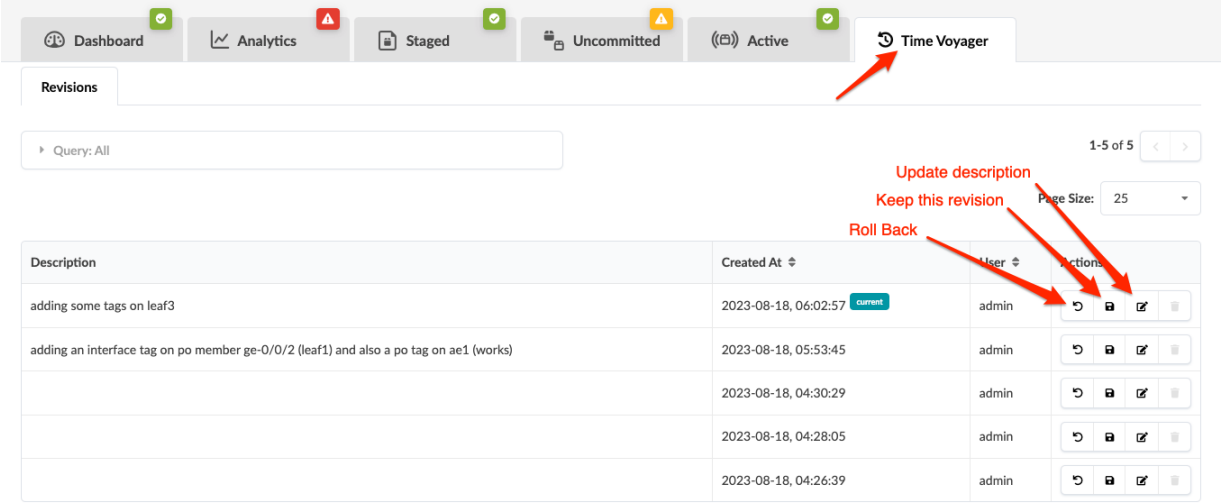
Time Voyager is not just an UNDO function. When using Time Voyager you roll back to a previous commit. This means that anything deleted on the last commit is re-applied when rolling back. There can be many changes in-between revisions, both additions and removals, all of which would be included in the rollback. Before committing a rollback, it's important that you review the pending changes in detail. Time Voyager is better compared with a Revision Control System (for the whole network!) than an UNDO function.

Unsupported Time Voyager Scenarios

- After you've upgraded Apstra server, you can't jump to a blueprint with an older version because the blueprint revision history is discarded on upgrade. If you need to return to a previous Apstra version that was taken prior to upgrading Apstra, refer to "Restore Database" on page 1401. This method could cause issues from a device config standpoint.
- It's not supported when the Pristine config has changed between revisions.
- It's not supported when the NOS versions are different between revisions. You could downgrade the NOS version to the same version using the device manager, then roll back to a previous revision.
- Devices that were allocated in a previous revision that are no longer available result in the build error *system ID does not exist*. (Conversely, *adding* a device and jumping to a previous revision without that device *will* be successful. The added device will be removed.)
- Resources that were assigned in a previous revision that have been reassigned cause the build error *resource already in use*. To resolve the build error, manually assign resources to each member in that group or reset the resource group overrides. (Jumping to a previous revision after a previously assigned global resource pool is modified *may* be successful, but it could cause an intent violation.)
- It's not supported if manual device config changes have been accepted.
- It's not supported in any other cases where the resulting device config state is different.

NOTE: Why not use Apstra server backup/restore to jump to a previous revision? Time Voyager maintains synchronized configuration between the Apstra server and devices (as much as possible); Apstra backup/restore does not. Effectively, the Apstra backup/restore is an out-of-band change from a device configuration standpoint. If a backup is restored, you would need to push a full config to make sure the device configuration reflects what you restored from the database backup. This would most likely be disruptive.

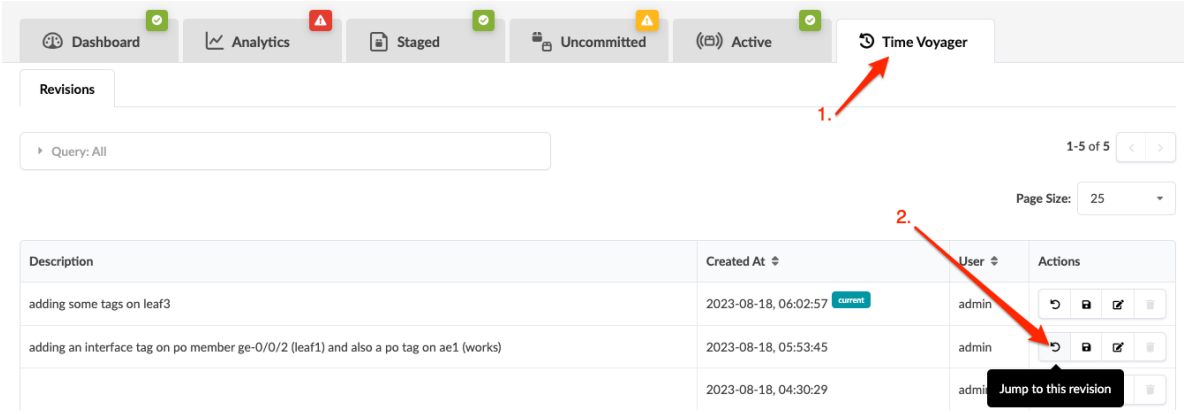
From the blueprint, click **Time Voyager** to go to the retained blueprint revisions. The first revision in the list is the active one. Successive revisions are ordered by date from most recent to oldest.



Roll Back Blueprint Revision

> **NOTE**: When you roll back to a previous revision, any previously staged changes that have not been committed are discarded. If this is an issue, do not jump to a different revision until you've committed the uncommitted changes.

1. From the blueprint, click **Time Voyager**, then click the **Jump to this revision** button for the revision to jump to (first of four buttons in **Actions** section).

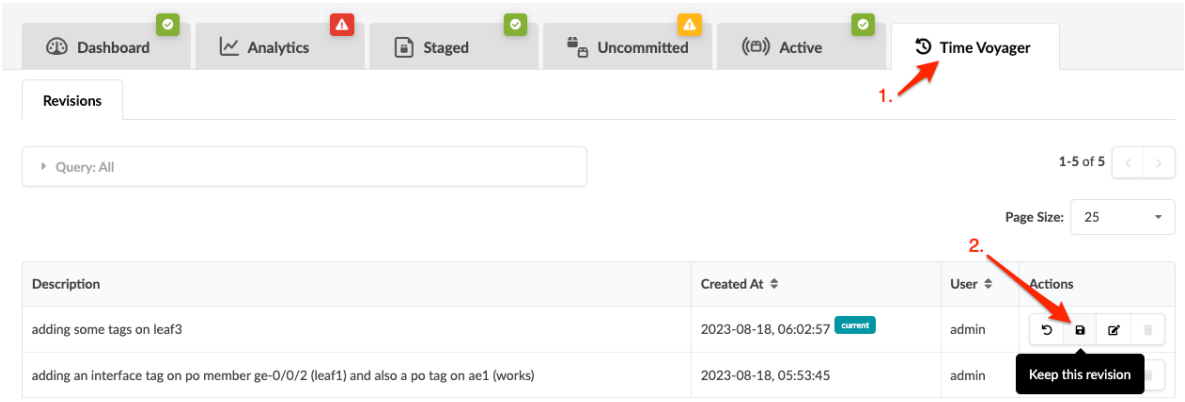


2. Any uncommitted changes in the staged area are discarded. If this is an issue, close the dialog and address the uncommitted changes before proceeding. To proceed, click **Rollback**.

3. You can make additional changes to the blueprint before committing. For example, if you've replaced a device, the device ID (serial number) will change, but the IP won't. You can create the device agent and update the serial number in your blueprint before committing the revision change.
4. Click **Uncommitted**, then click the diff tabs to review the changes.
5. If you decide that you don't want to jump to this revision, click the **Revert** button to discard the changes.
6. To proceed, click the **Commit** button (top-right) to see the dialog for committing changes and creating a revision.
7. We recommend that you enter the optional revision description to identify the changes. Specific differences between revisions are not displayed, so the description is the only change information available for the revision.
8. Click **Commit** to commit your changes to the active blueprint and create a revision.
9. If you click **Time Voyager** you'll see the revision as the current one.

Keep Blueprint Revision

1. From the blueprint, click **Time Voyager**, then click the **Keep this revision** button for the revision to keep (second of four buttons in **Actions** section).



2. Click **Save** to confirm and proceed. The button turns gray indicating that the revision has been saved indefinitely. It won't be deleted until you manually delete it.

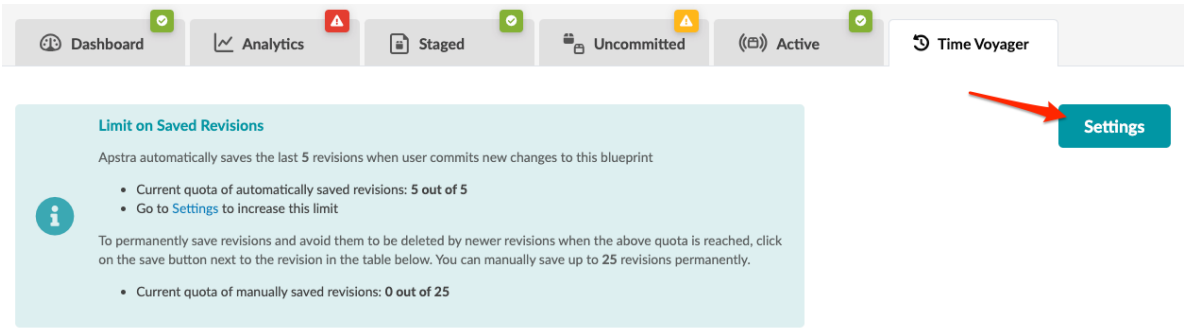
RELATED DOCUMENTATION

Time Voyager Introduction | 627

Change Number of Saved Blueprint Revisions

5 blueprint revisions are saved automatically by default. You can change the setting to save up to 100 revisions. When you commit a revision to the blueprint that exceeds the set number to save, then the oldest revision is automatically deleted.

1. From the blueprint, click **Time Voyager**, then click **Settings** (top-right) to go to the **Update Settings** dialog.



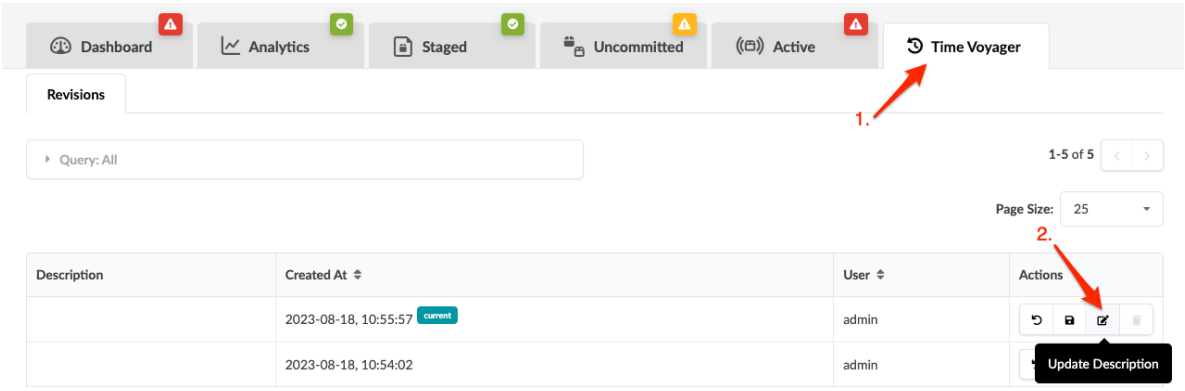
2. Change the maximum number of automatically saved revisions, up to 100. Decreasing the number of automatically saved revisions will delete older revisions, as needed.

RELATED DOCUMENTATION

Time Voyager Introduction | 627

Update Blueprint Revision Description

1. From the blueprint, click **Time Voyager**, then click the **Update description** button for the revision to update (third of four buttons in **Actions** section.)



2. Enter or change the description.
3. Click **Update** to change the description and return to the table view.

RELATED DOCUMENTATION

Time Voyager Introduction | 627

Delete Blueprint Revision

1. From the blueprint, click **Time Voyager**, then click the **Delete** button for the revision to delete (fourth of four buttons in **Actions** section). You can't delete a revision if there are five (5) or fewer of them in the list.
2. Click **Delete** to delete the revision and return to the table view.

RELATED DOCUMENTATION

Time Voyager Introduction | 627

PART 9

Devices

Device Configuration Lifecycle

IN THIS SECTION

CAUTION: A good understanding of the Apstra device configuration lifecycle is essential. Before working with devices in the Apstra environment, we strongly recommend that you fully understand how devices are configured from the moment they are on-boarded to the moment they are decommissioned.

Terminology

Configuration lifecycle stages are as follows:

Configuration Stage	Description
Pristine Config	When you install a device agent, configuration is added to the pre-existing config on the device. Normally, the pristine config doesn't change throughout the device's lifecycle.
Discovery 1 Config	When you *acknowledge* a device, Apstra adds basic configuration, including enabling LLDP on all interfaces.

(Continued)

Configuration Stage	Description
Ready Config (previously known as Discovery 2 Config)	When you assign a device to a blueprint without deploying it (deploy mode: ready), Apstra adds basic configuration, including device hostnames, interface descriptions and port speed / breakout config.
Service Config	When you deploy a device (deploy mode: deploy), Apstra adds configuration that's required in the Apstra environment. *Service Config* consists of Discovery 1 config, Ready (Discovery 2) config and this additional config.
Rendered Config	Complete Apstra-rendered configuration for the device, per the Apstra Reference Design.
Incremental Config	The configuration that will be applied when you commit changes that you've made.
Golden Config	When you commit config changes, Apstra collects a new running configuration called *Golden Config*. Golden config serves as Intent: Apstra continuously compares the running config against the Golden config. When a deployment fails, Apstra unsets the Golden Config.

Configuration Stages: Overview

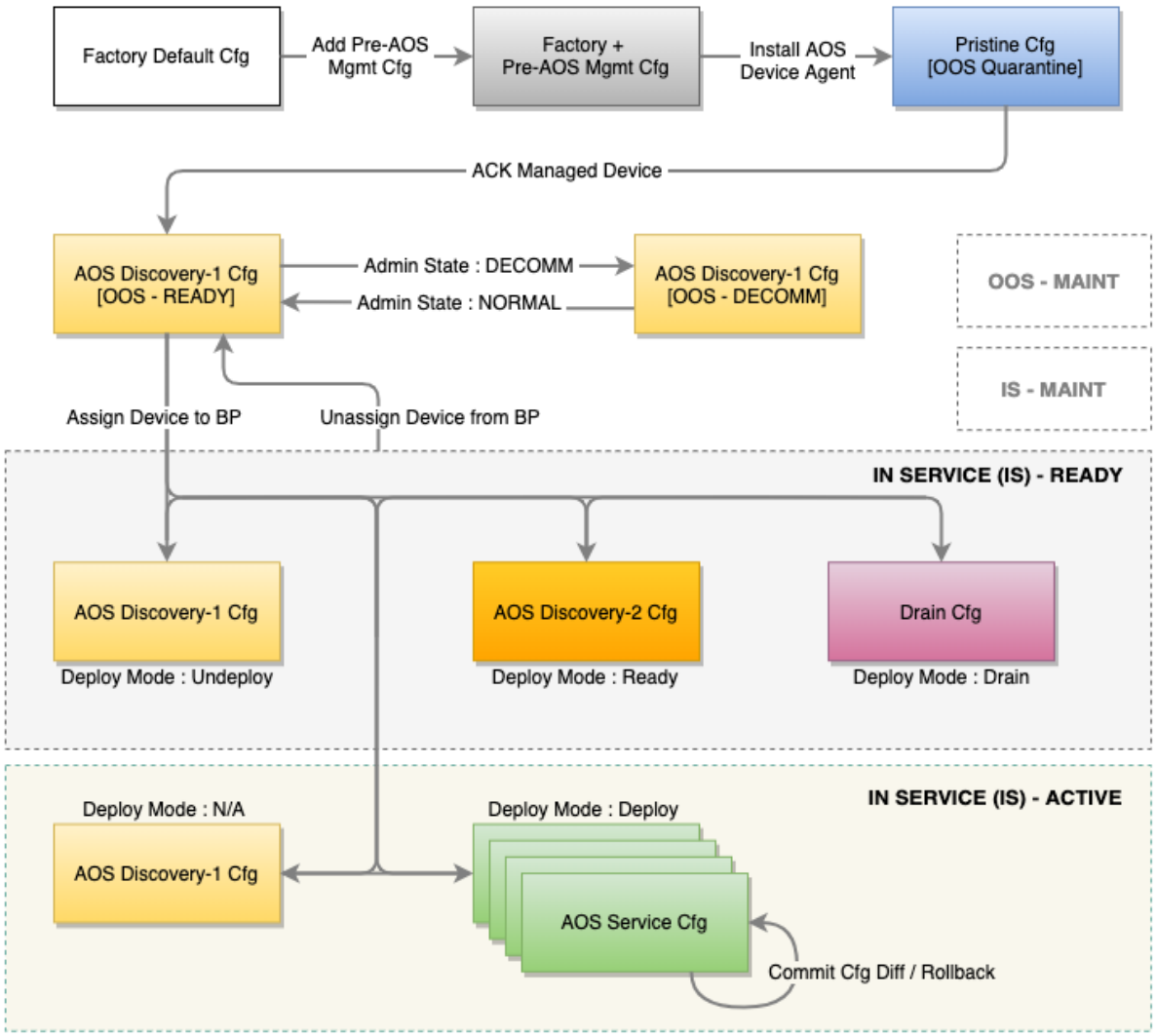
The following table describes the various config events and their resulting device config, Apstra-managed device state, and blueprint deployment mode:

Event	Resulting Device Configuration	Resulting Apstra Managed Device State	Apstra Blueprint Deployment Mode
New device	Factory Default Configuration	N/A	Not Assigned
Add pre-Apstra [mgmt] configuration to device	Factory + Pre-Apstra	N/A	Not Assigned
Install Apstra device system agent	**Pristine Config:** Factory + Pre-Apstra + Agent Install config	OOS-QUARANTINED	Not Assigned
Acknowledge device	**Discovery 1:** Pristine, plus Interfaces Enabled	OOS-READY	Not Assigned

(Continued)

Event	Resulting Device Configuration	Resulting Apstra Managed Device State	Apstra Blueprint Deployment Mode
Assign device to blueprint (no deploy)	**Ready (Discovery 2):** Discovery 1, plus various basic config	IS-READY	Ready
Deploy device	**Service Config:** Ready (Discovery 2) config plus full Apstra-Rendered config	IS-ACTIVE	Deploy
Add/Commit incremental configuration	Delta of resulting config changes from blueprint modifications	IS-ACTIVE	Deploy
Drain device	"Drain" Configuration is added	IS-READY	Drain
Undeploy device	Apstra-rendered config is removed	IS-READY	Undeploy
Unassign device	Discovery 1 config is re-applied	OOS-READY	Not Assigned

AOS Device Lifecycle



Note: This diagram does not include the flows for 'Admin State : MAINT'. When device admin state is set to MAINT, device state will be either 'IN SERVICE (IS) - MAINT' or 'OUT OF SERVICE (OOS) - MAINT' but the device config will not be changed.

CAUTION: When you install an agent on a device, any configuration that was already there becomes part of the Pristine Config, which means it's included in the device's entire configuration lifecycle. Any corrections that you make will be service-impacting.

Configuration Stages: Detail

IN THIS SECTION

New Device (Factory Default)

The lifecycle of a device begins with the **factory default** configuration stage.

Add Pre-Apstra Config (User-required)

Certain minimum base configuration is required for the entire configuration lifecycle. This includes configuration for agent installation and device connectivity. You must configure management IP connectivity between devices and the Apstra server out-of-band (OOB). Configuring it in-band is not supported and could cause connectivity issues when changes are made to the blueprint.

You can bootstrap this **User-required** config with "Apstra ZTP" on page 713, or add it with scripts (or other methods).

CAUTION: Only add configuration that's required for connectivity, for installing the device agent, or that's known to be required **throughout the device lifecycle** (for example Banners or NTP / SNMP / syslog server IP addresses). You can add required configuration that's not rendered by Apstra with "configlets" on page 908.

Install Agent (Pristine)

When you install an onbox agent on a device (or an offbox agent on the server) the device connects and registers with Apstra in the **Quarantined** state. Apstra applies partial configuration to the pre-Apstra

configuration. This configuration is called **Pristine configuration**. Pristine configuration is the basis for all subsequent device configuration.

Acknowledge Device (Discovery 1 / Ready)

When you acknowledge a device, you're putting it in the **Ready** state. This acknowledgment signals your intent to have Apstra manage the device. To the pristine config, Apstra adds minimal base configuration that's essential to Apstra agent operation. This configuration is called **Discovery 1 config**. Discovery 1 applies a *complete* configuration (Full config push), overwriting all existing configuration to ensure config integrity.

- All interfaces are rendered with interface speeds for the assigned device profile.
- All interfaces are `no shutdown` to allow you to view LLDP neighbor information.
- All interfaces are moved to L3 mode (default) to prevent the device from participating in the fabric.

> **NOTE**: Devices that have been acknowledged cannot simply be deleted. Since the device would still have an active agent installed, the devices would re-appear within seconds. To remove a device from Apstra management, see "Remove (Decommission) Device from Managed Devices" on page 673 for the complete workflow.

Assign Device (Ready / Ready)

When you assign a device to a blueprint and set its Deploy Mode to **Ready**, you're putting it in the **Ready (Discovery 2)** state. The device has been staged, but not yet committed (deployed) to the active blueprint. Ready config applies a *complete* configuration (Full config push) to ensure config integrity. Ready configuration brings up network interfaces and configures interface descriptions and validates telemetry, such as LLDP, to ensure it's properly wired and configured. This configuration is non-disruptive to other services in the fabric. Links are up, but they are configured in L3-mode to prevent STP/L2 operations.

- Hostname is configured per blueprint intent.
- All interface descriptions are changed per blueprint intent.
- Interfaces are rendered with blueprint interface speeds.
- No routing or BGP is configured.
- No L3 information is configured on interfaces.
- Fabric MTU is modified for spine devices to 9050 bytes.

Deploy Device (Rendered / Active)

CAUTION: The first time you assign a device and deploy it (set deploy mode to Deploy and commit the blueprint), you're triggering a full configuration push on the device. This action overwrites the complete running configuration with the pristine configuration, then adds the full rendered Apstra configuration. Apstra discards any configuration that's not part of the Apstra-rendered configuration.

When you commit a device, it becomes **Active**, and Apstra deploys the service configuration, moving the device into the **Rendered** configuration stage. Rendered config contents are derived from the pristine config, selected reference design/topology, NOS, and device model. The first rendered config applies a *complete* configuration (removing all existing configuration from the Apstra server per Jinja) to ensure configuration integrity. This is the full end-state of Apstra. A full configuration has been pushed, all interfaces are running, and routing within IP fabric is configured. Full configuration rendering, intent-based telemetry, and standard service operations occur here.

- Hostname is configured per blueprint intent.
- All interface descriptions are changed per blueprint intent.
- Interfaces are rendered with blueprint interface speeds.
- Interface VLANs, LAGS, MLAG, VXLAN, and so on, are managed.
- All L3 information is rendered.
- BGP configuration is fully rendered for all BGP peering information.
- DHCP configuration is configured for any required DHCP relay agents.
- The device is added to the graph database.

After the full configuration is successfully deployed to the device, Apstra takes a snapshot of the device configuration (for example show running-confg) and stores it as the **Golden Configuration**.

CAUTION: If you add configuration at this point, you'll raise configuration deviation anomalies. The deviation is the difference between the current configuration and the stored Golden configuration. Before you can proceed with deployment tasks, you must correct any anomalies.

To see the rendered config file after committing the blueprint, select the device in the **Active** blueprint and click **Config** (right-side).

You can modify a running configuration multiple ways. To modify a config that's not part of the reference design, use "configlets" on page 908.

Stage Device Update (Incremental / Active)

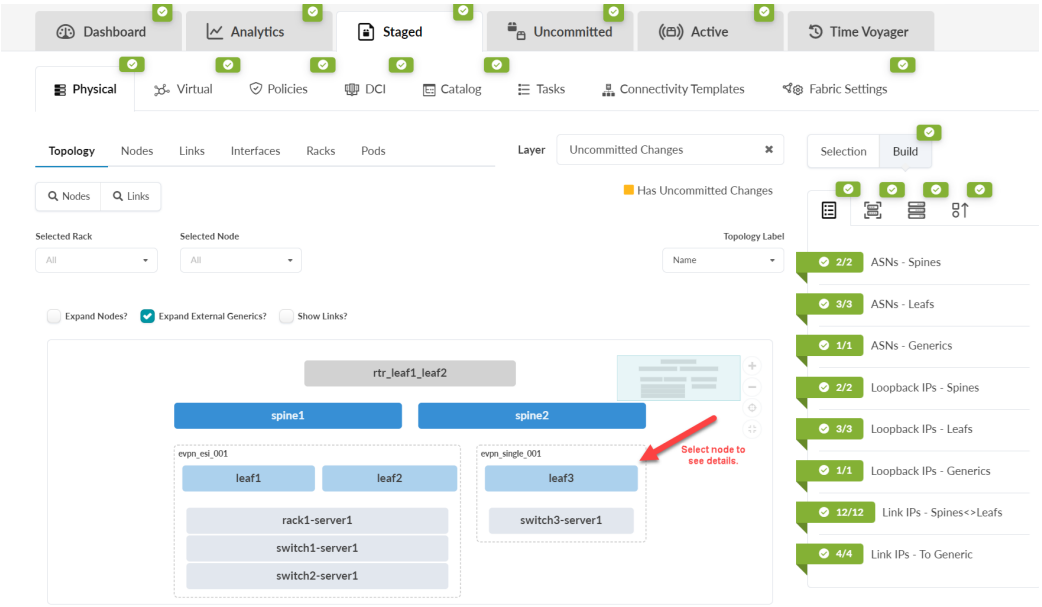
When you stage changes to a running blueprint, you're creating an **Incremental** configuration.

Commit Device Again (Rendered-Updated / Active)

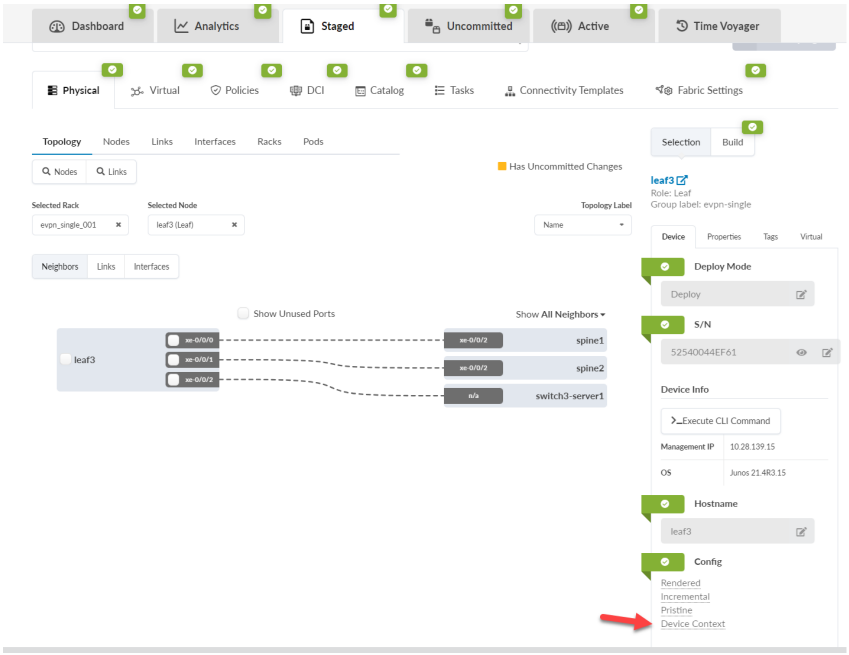
When you commit a change to a blueprint that affects the device's configuration, a partial config updates the rendered config.

View Device Config from Blueprint

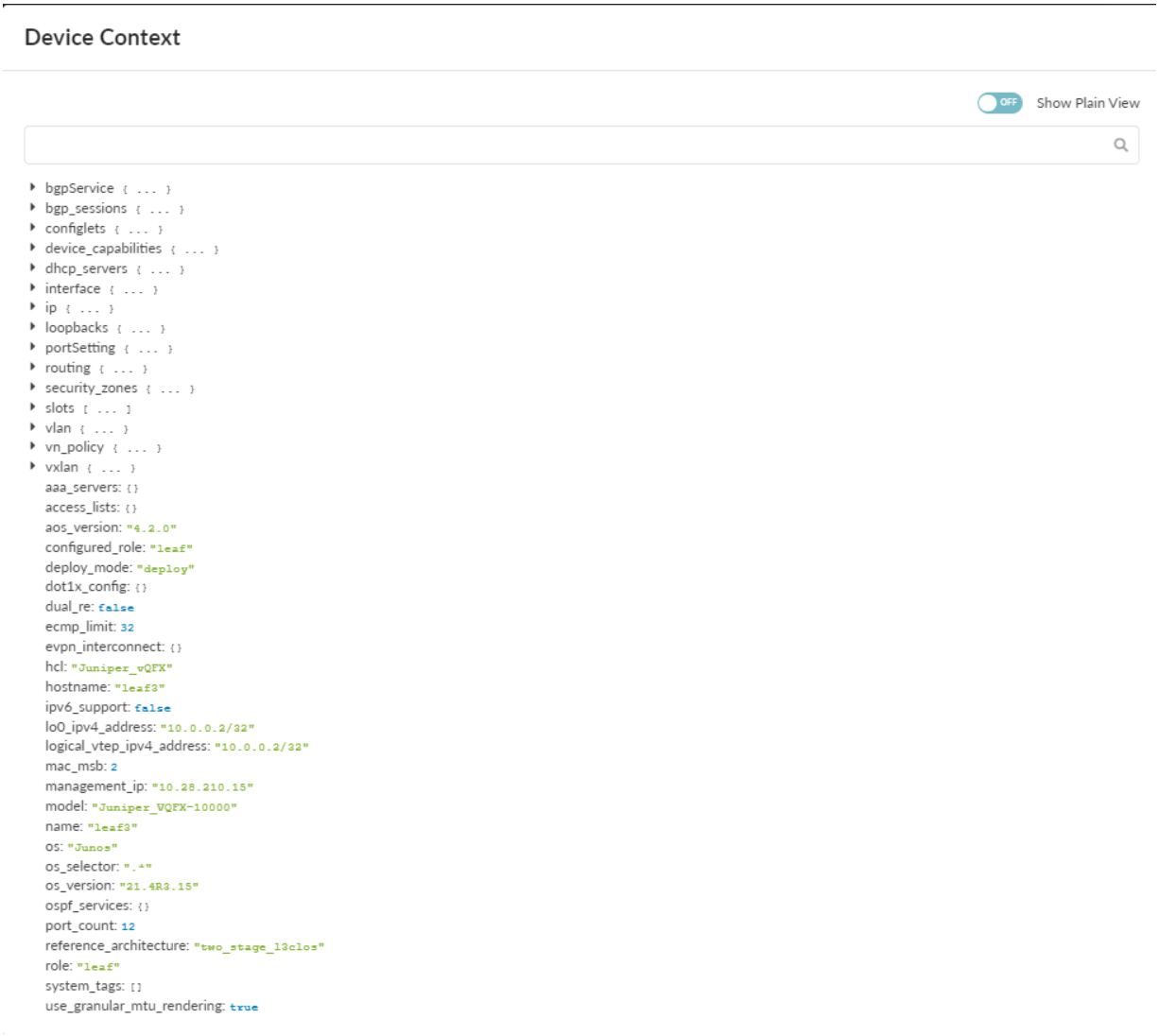
From the blueprint, navigate to **Staged > Physical** to go to the **Topology** view of the physical blueprint.



Click a node in the topology, then from the **Device** tab in the panel on the right, you can click links for rendered, incremental, pristine, or device context in the **Config** section.



The device model is a nested dictionary of variables that you can leverage when creating configlets in Datacenter blueprints or config templates in Freeform blueprints. Device context includes information that's useful when creating configlets in Data Center blueprints. In the interface section you'll find `tags` for interface tags and `intf_tags` for link tags. In the main section you'll find `system_tags`.



The query tab provides dynamic search capabilities to quickly search through keys or values and identify the variables of interest. Syntax is case-sensitive. For example, a search of the keyword **bgp** provides information on the BGP configuration of the switch as well as the BGP sessions (protocol sessions), while a search on the key word **BGP** provides the list of BGP route maps such as "BGP-AOS-Policy". The use of these variables as built-in property-sets inside a configlet must also respect the case-sensitive attribute of the device model.

CAUTION: Device models are an internal data model used in the Apstra environment. They are subject to change without notice or documentation of schema changes.

Configuration Deviations

After each **successful** config deploy the running config is collected and stored internally as the **Golden** configuration. Intent is the cornerstone of the Apstra product. Any difference between the actual running config and this golden config results in a config deviation anomaly on the blueprint's dashboard. The golden config is updated every time config is successfully applied to a device.

Some important points to know:

- Each *successful* configuration deployment results in an updated Golden Config.
- If configuration deployment fails, Golden Config is not set. This means both a config deviation and deployment failure anomaly are raised.
- Running configuration telemetry is continuously collected and matched against the Golden Config. Any difference result in a deviation anomaly.
- Configuration anomalies can be 'suppressed' using the "Accept Changes feature". This does **NOT** mean the change is added to golden config or Intent.

See "Anomalies (Service)" on page 616 for details.

Device Offline (Unavailable)

A managed device (one that has been acknowledged) that is not connected to the Apstra server is in the **unavailable** state. A device could be offline if the device agent interface is offline, if the service is not running, or if a network connectivity error occurs.

Manually Apply Full Config

The **Discovery 1** and **Deploy Device** configuration stages initiate full config pushes. In rare cases, you may need to manually apply a full config push. For example, if the required config is not in place for a blueprint with NX-OS devices that require TCAM carving, the device config will fail. The TCAM config error must be corrected, followed by manually pushing a full config.

> **NOTE**: Perform a full configuration push with the utmost caution, as it is very likely to impact all services running on the box. Exact impact depends on changes being pushed. Also note **all** Out of Band changes are overwritten upon a full push.

Deploy Modes

IN THIS SECTION

Managed devices in blueprints can be in one of several "modes" on page 84:

Not Set

Initial device state. The device is not active in the fabric. When you unassign a system ID from a device in a blueprint, the deploy mode automatically changes to Not Set (as of Apstra version 5.0.0).

Deploy

The device is active in the fabric.

Ready

When you assign a device to a blueprint, it's deploy mode changes to **Ready**; Apstra renders Ready (Discovery 2) configuration (hostnames, interface descriptions, port speed / breakout configuration). The device isn't active in the fabric. Changing from **Deploy** to **Ready** removes Apstra-rendered configuration.

Drain

"Draining a device" on page 652 for physical maintenance enables it to be taken out of service without impacting existing TCP flows. Depending on the device being drained, Apstra uses one of two methods:

For L2 Servers

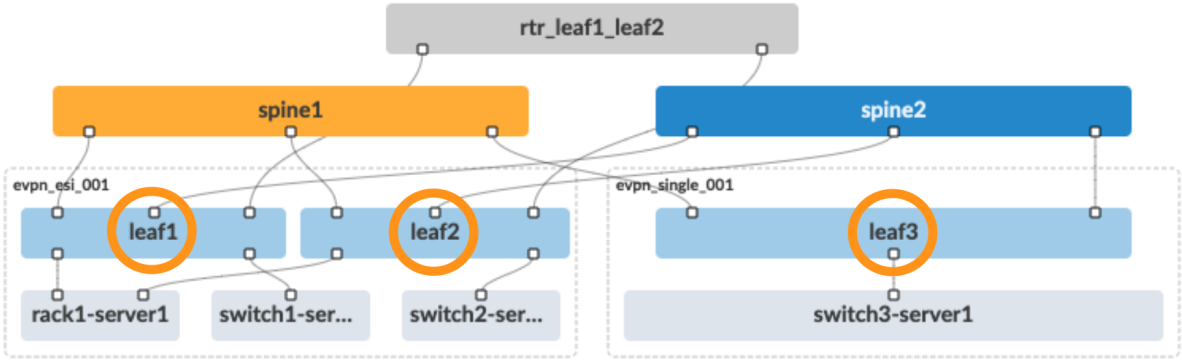
- MLAG peer-links port channels and bond interfaces on any NOS are not changed.
- For Arista EOS and Cisco NX-OS, and Junos OS on Apstra 4.2.1 and above, all interfaces towards L2 servers in the blueprint are `shutdown`.

For Network L3 Switches

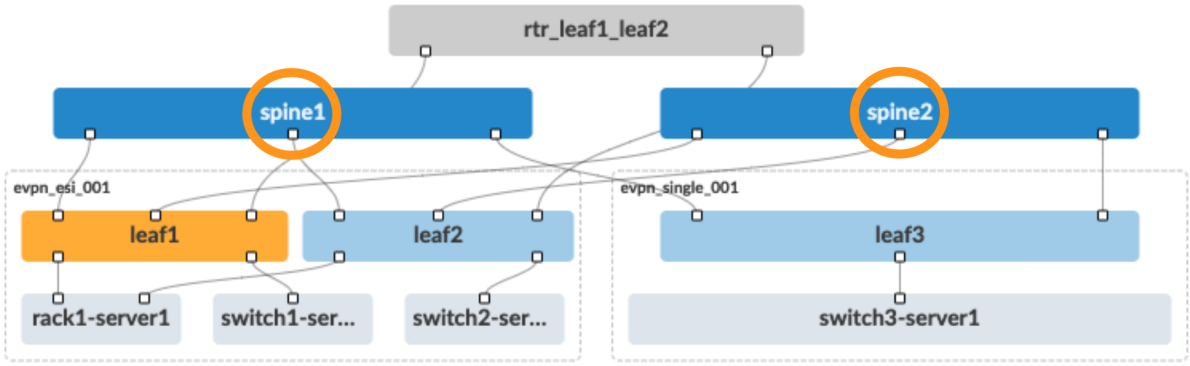
The device uses Inbound/Outbound route-maps 'deny' statements to block any advertisements to 0.0.0.0/0 le 32. This allows existing L3 TCP flows to continue without interruption. After a second or two, the TCP sessions should be re-established by the src/dst devices, or they should negotiate a new TCP port. The new TCP port forces the devices to be hashed onto a new ECMP path from the list of available links. Since no ECMP routes to the destination are available in the presence of a route map, the traffic does not flow through the device that is in **Drain** mode. The device is effectively drained of traffic and can be removed from the fabric (by changing Deploy mode to **Undeploy**).

While TCP sessions drain (which could take some time, especially for EVPN blueprints) BGP anomalies are expected. When configuration deployment is complete, the temporary anomalies are resolved.

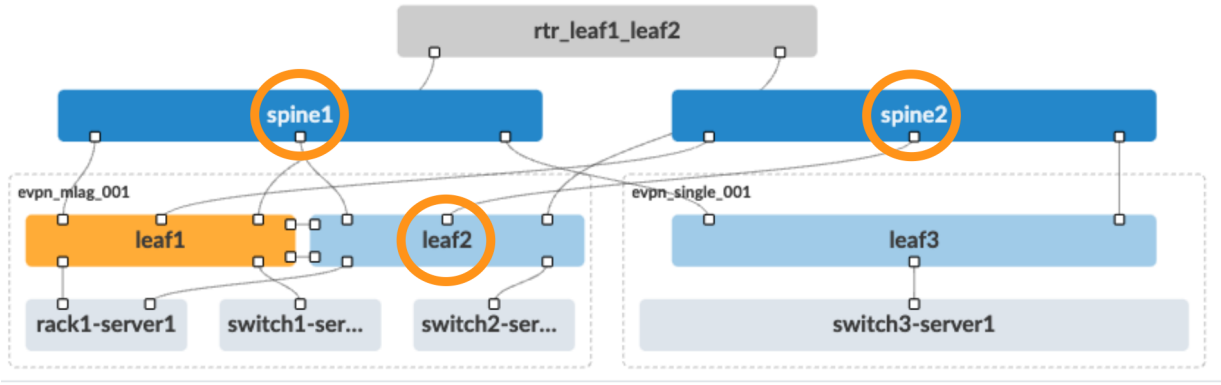
When you change the deploy mode to **Drain** on a device, neighboring device configuration may also be affected, not just the device you're draining. For example, when you drain a spine device, configuration on all connected leaf devices change. Neighboring leaf devices use Inbound/Outbound route filters (route-maps) 'reject (deny)' statements to block any advertisements to 0.0.0.0/0 le 32, for both EVPN (overlay) and FABRIC (underlay).



Similarly, when you drain a leaf device, the configuration on connected spine devices changes. Neighboring spine devices use Inbound/Outbound route filters (route-maps) 'reject (deny)' statements to block any advertisements to 0.0.0.0/0 le 32, for both EVPN (overlay) and FABRIC (underlay).



In the case of an MLAG-based topology, in addition to the configuration on connected spine devices changing, the configuration on the paired leaf device also changes.



Undeploy

Undeploying a device removes the complete service configuration. If a device is carrying traffic it is best to put it in **Drain** mode first (and commit the change) before undeploying the device.

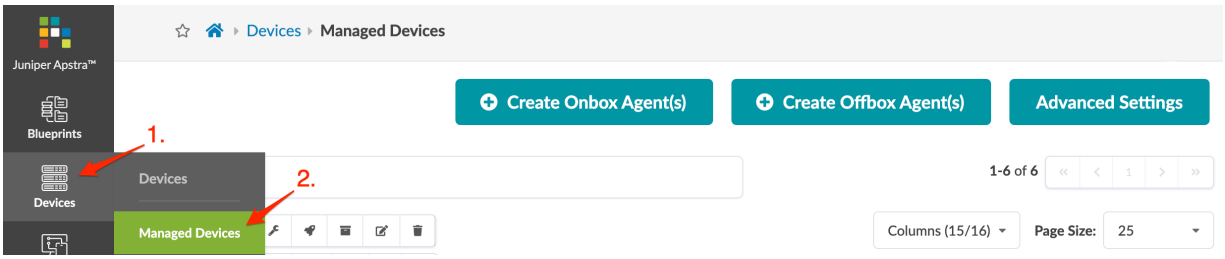
What are Managed Devices

IN THIS SECTION

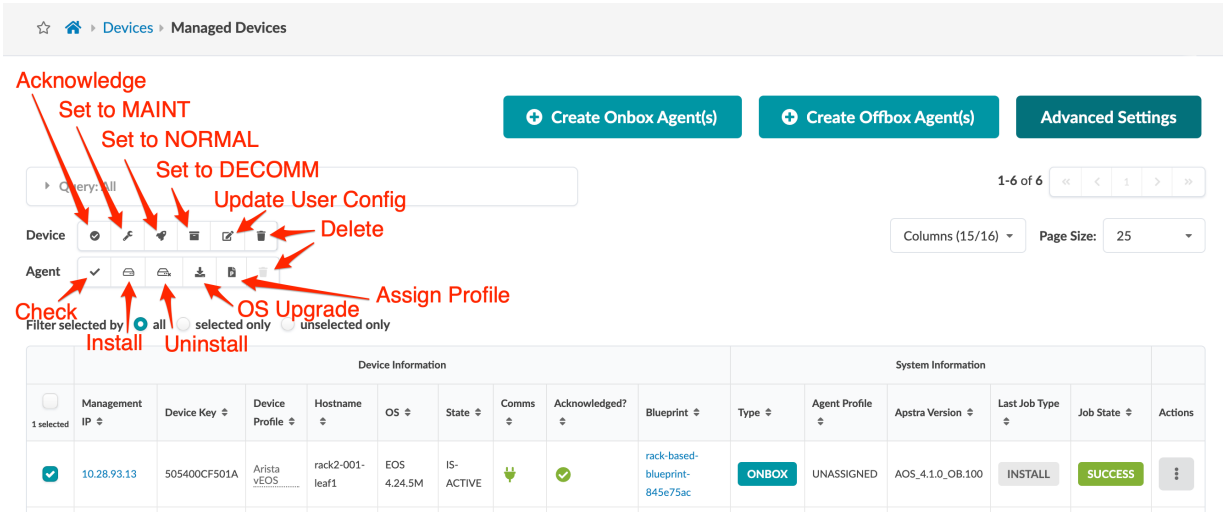
Apstra software uses device system agents to manage devices. These agents manage configuration, device-to-device communication and telemetry collection. You can use "Apstra Zero Touch Provisioning (ZTP)" on page 713 to install agents and bring devices under Apstra management or you can use the device installer.

> **CAUTION**: A good understanding of the "Apstra device configuration lifecycle" on page 634 is essential. Before working with devices in the Apstra environment, we strongly recommend that you fully understand how devices are configured from the moment they are on-boarded to the moment they are decommissioned.

From the left navigation menu in the Apstra GUI, navigate to **Devices > Managed Devices** to go to managed devices.



Devices with installed agents appear in the table. The **Managed Devices** page is the hub for many device-related tasks, which are described in later sections.

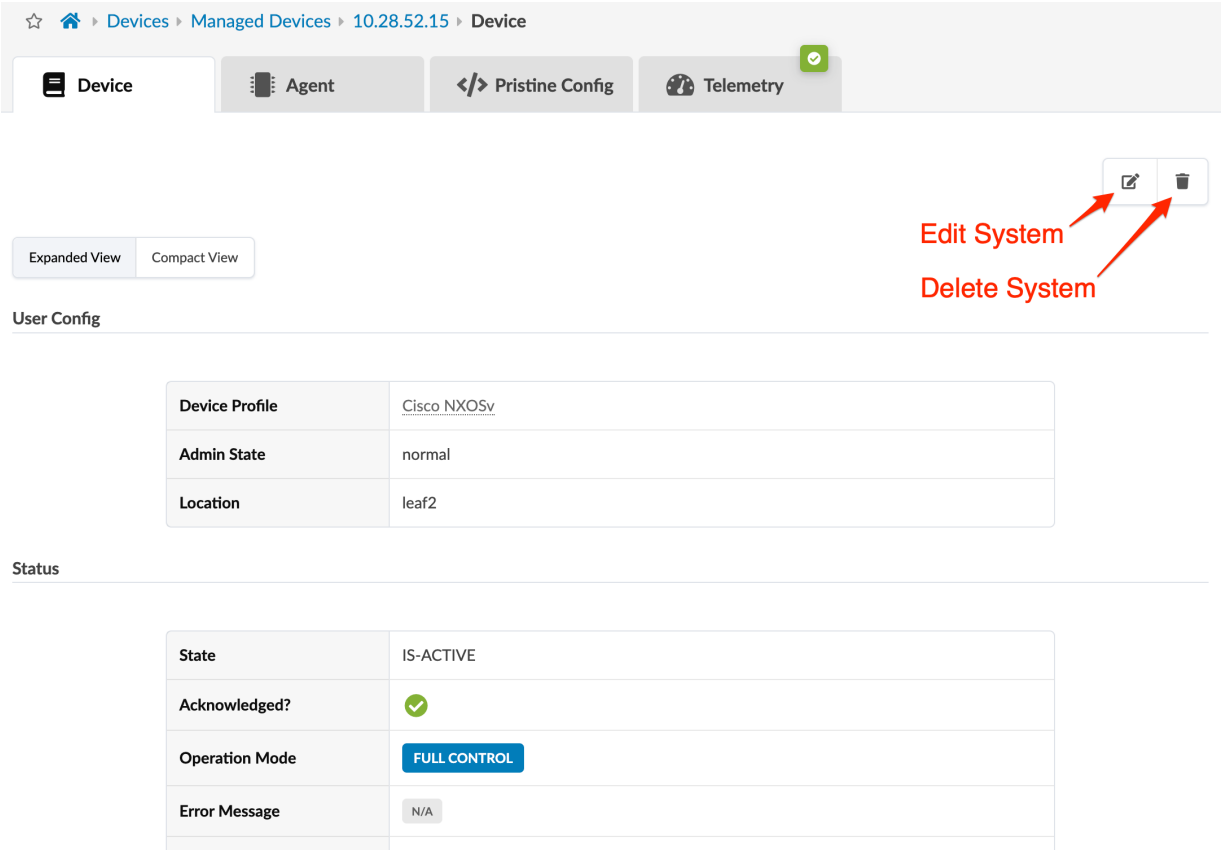


Click a management IP to go to details for its device, agent, pristine config and telemetry as shown below.

Device

The device detail view shows the user config, the device status and other facts about the device. From the device detail page you can edit and delete the device. You can also edit or delete a device from the

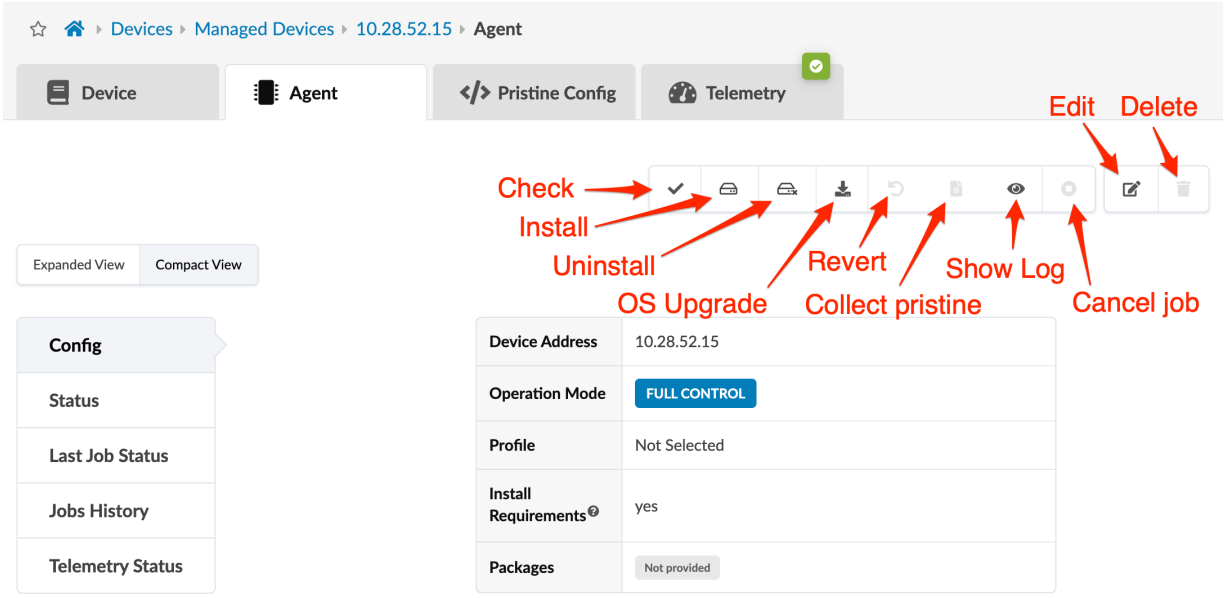
table view or any of the other detail views (Agent, Pristine Config, Telemetry).



Agent

Apstra device system agents handle configuration management, device-to-server communication, and telemetry collection. If you're not using "Apstra ZTP" on page 713 to bootstrap your devices (or if you have a one-off installation) you can use this device installer to automatically install and verify devices. Depending on the device NOS, you can install device agents onbox (agent is installed on the device) or offbox (agent is installed on the Apstra server and communicates with devices via API). For support information, see the **Device Management** section of the "Apstra 5.0.0 Feature Matrix" on page 1481.

The device agent view shows the agent config, agent status, last job status, jobs history and telemetry status. From the agent detail page you can perform various tasks similar to tasks in the table view. For example, you can restore a device's pristine configuration by clicking the **Revert to Pristine Config** button (as of Apstra version 4.0.1) as long as the device is not assigned to a blueprint.

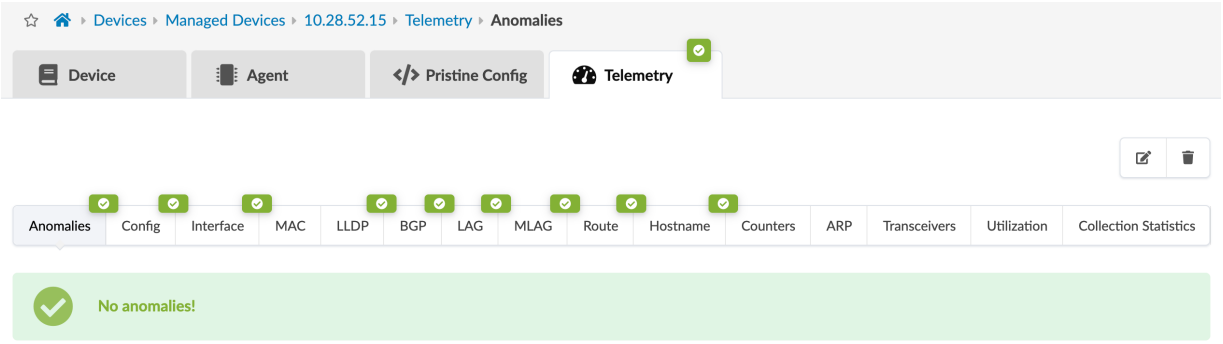


Pristine Config

The pristine config view shows the pre-Apstra configuration on the device. You can edit the pristine config manually or update it directly from the device. You can edit and delete the device. You can also edit or delete the device from the table view or any of the other detail views (Device, Agent, Telemetry).

Telemetry

The telemetry view shows telemetry for the device. For more information, see "Telemetry Services" on page 941.

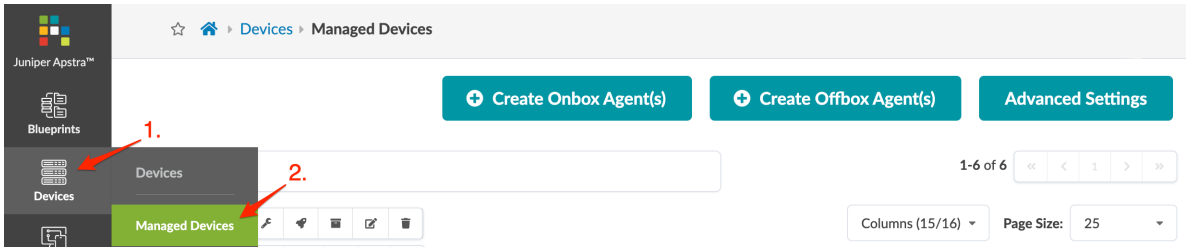


Add Managed Device

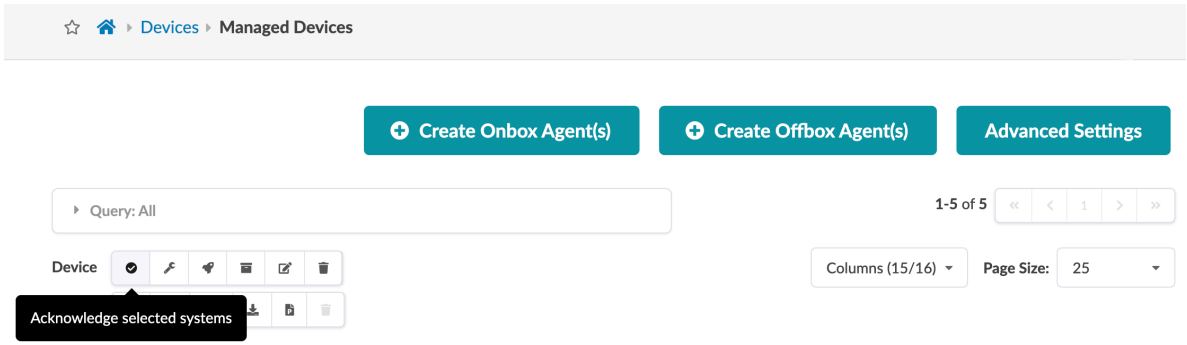
Before working with devices, it's important to have a good understanding of the "device configuration lifecycle" on page 634.

NOTE: Each device is expected to have a unique management IP address. If you're replacing a device (decommissioning for an RMA for example) and you want to use the same management IP address on the replacement device, you must "remove (decommission) the device from Managed Devices" on page 673 before adding the new device.

1. If you're using Juniper offbox agents, "increase the application memory usage" on page 1325.
2. Create and install your "onbox" on page 680 device agent(s) or "offbox" on page 685 device agent(s) for the devices to be managed in the Apstra environment. If you have many of the same devices using the same configuration you might consider creating "agent profiles" on page 696 (Device > Agent Profiles), which can streamline the task of creating many agents.
3. If you're deploying modular devices, you may need to "Change the Assigned Device Profile (Datacenter)" on page 669 that's assigned to your device. (or "Change the Assigned Device Profile (Freeform)" on page 530).
4. Navigate to **Devices > Managed Devices** to see that the device state is **Out of Service Quarantine**. Configuration at this point is called **Pristine Config**.



5. In the left column of the table, select the check box(es) for the device(es) to manage in the Apstra environment.
6. Above where you just clicked, click the **Acknowledge selected systems** button (check mark) in the **Device** action bar.



7. Click **Confirm** to acknowledge the device(s) and return to the table view. The device state changes to **Out of Service Ready**. Configuration at this point is called **Discovery 1 Config** and you can now manage the device(s) from the Apstra environment.

Next Steps:

"Create" on page 8 a blueprint.

You'll assign Datacenter devices to a blueprint during the build phase. For details, see "Assign Device (Datacenter)" on page 64 or "Update System ID Assignment (Freeform)" on page 534, as applicable.

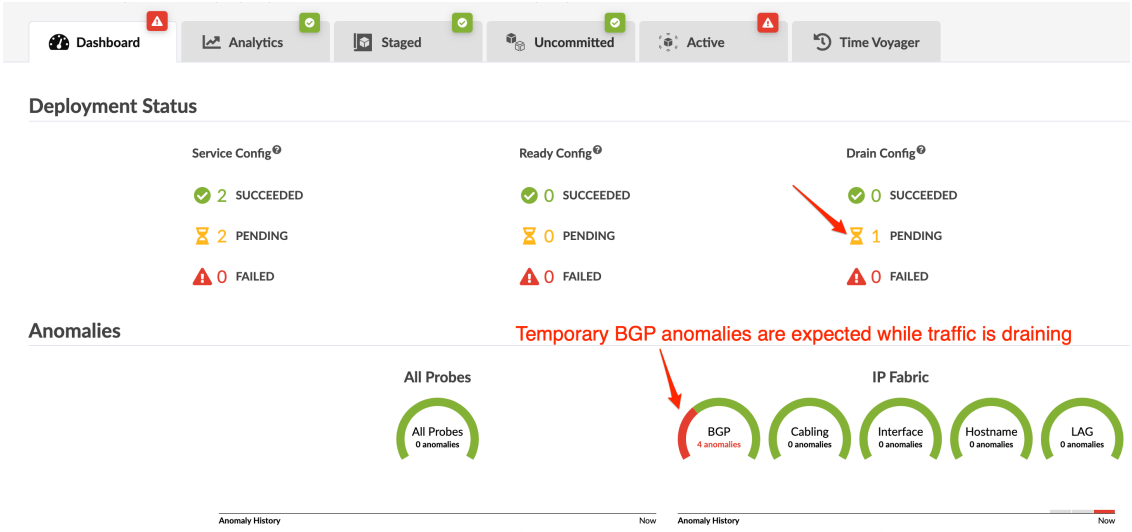
RELATED DOCUMENTATION

What are Logical Devices | 845

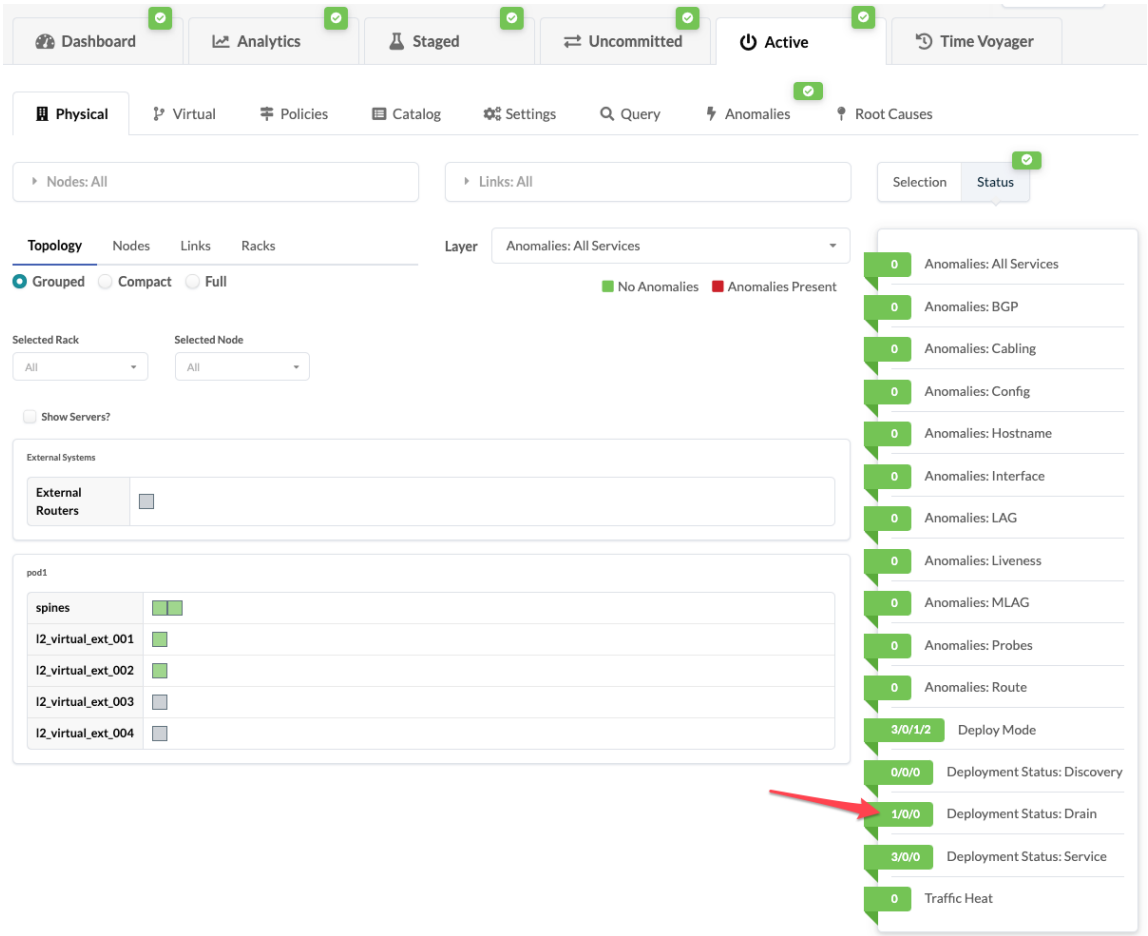
Drain Device Traffic

To take a device out-of-service for maintenance (or decommissioning), set its deploy mode to **Drain**. Draining a device may impact neighboring devices. For details, see "Device Configuration Lifecycle" on page 645.

1. From the blueprint, navigate to **Staged > Physical > Build > Devices** and change the "deploy mode" on page 84 on the device to **Drain**.
2. Click **Uncommitted** to review staged changes. The **Logical Diff** tab shows the changes that will be made to the device, and possibly to its neighbors.
3. Commit staged changes to activate them. While draining is in progress (which could take some time, especially for EVPN blueprints) BGP anomalies are expected. You can monitor draining progress from various locations in the Apstra GUI. When drain configuration is complete, the temporary anomalies are resolved.
   - You can monitor drain status from the **Deployment Status** section of the blueprint dashboard (Drain Config).

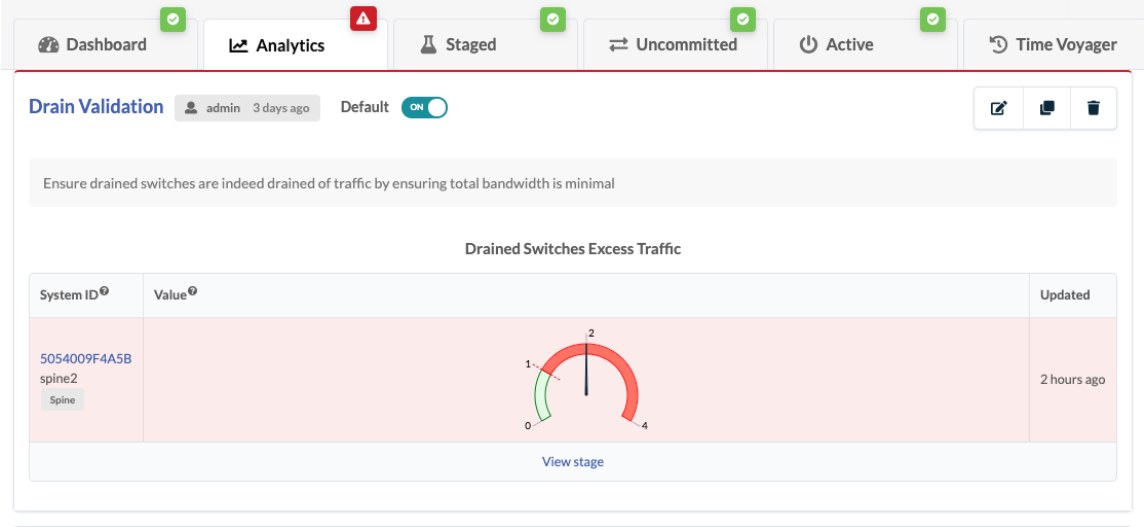


- You can monitor drain status from **Active > Physical** in the **Status** panel (Deployment Status: Drain).



- If you instantiate the predefined **Drain Validation** dashboard, you can monitor drain status from **Analytics > Dashboards**. (If you set the dashboard as default, you can see it on the blueprint dashboard as well as on the analytics dashboard). In the image below, traffic is in the process of

draining.



After performing device maintenance, change the deploy mode back to **Deploy** and commit the change to bring the device back into active service.

RELATED DOCUMENTATION

- What are Blueprint Analytics | 14
- Commit / Revert Changes to Blueprint | 599

Upgrade Device NOS

SUMMARY

Upgrade the network operating systems (NOS) of your Apstra-managed network devices from within the Apstra environment.

IN THIS SECTION

We highly recommend that you become familiar with this procedure before upgrading a device NOS.

NOS Upgrade Overview

You can upgrade a device NOS within the Apstra environment with a few steps. If you've defined your own device profiles, you may need to update them first. You'll register the new OS image that you obtained from the vendor, then click a button to start the upgrade. Apstra takes care of upgrade tasks and other requirements and ensures that pristine config is updated.

NOTE:
can

For information about supported upgrade paths, see "NOS Upgrade Paths" on page 1518 in the References section.

Apstra software ships with predefined device profiles that support specific OS versions. When you upgrade the Apstra server, device profiles with the OS versions that are supported in the new Apstra version are also updated. You can then upgrade the NOS to one of the newly supported versions.

However, device profiles that you've created (cloned) yourself, are not managed in the Apstra environment, so when you upgrade the Apstra server those device profiles aren't automatically updated with newly supported versions. You'll need to follow a few extra steps to add them as described in the next section.

Before beginning the process, make sure of the following:

- Make sure that you understand the "device configuration lifecycle" on page 634 and that you're comfortable with managing deploy modes.
- Make sure that Apstra software is managing the device you're upgrading. Navigate to **Devices > Managed Devices** and confirm that your device is in the table and that it is acknowledged (with a green check mark).
- Before upgrading NOS, delete any device AAA/TACACS+ configlets from the blueprint. After the upgrade is complete, you can reapply them.
- Make sure that the Admin state of the device is set to **normal**. Navigate to **Devices > Managed Devices**, click on the **Management IP** of the device to confirm the admin state. (Do NOT set the Admin state to MAINT/DECOMM or the device could enter an unrecoverable state.)
- Make sure the deploy mode of the device is set to "Drain." on page 652
- Make sure that the Apstra version specified is the same on both the Apstra server and the device. If they are different, you can't upgrade the device. If you attempt to upgrade with different versions, you will not receive a warning; the task status remains in the IN PROGRESS state indefinitely.

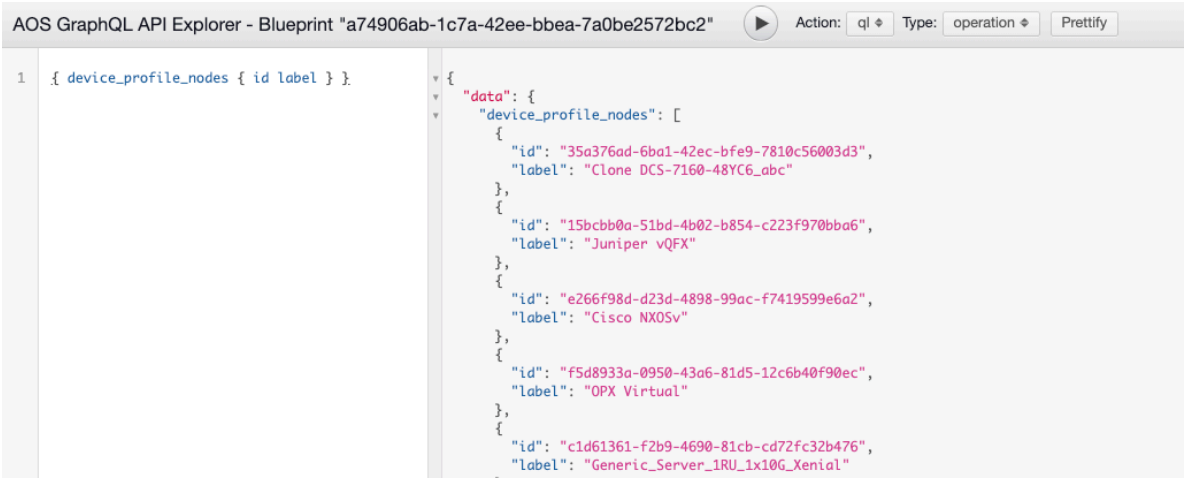
Update User-defined Device Profiles

Make sure that your devices are in the appropriate states for upgrading as described in the overview above.

If you've created (cloned) your own device profiles, you'll need to manually specify OS versions in the device profile and the blueprint that uses that device profile. (If your devices use predefined device profiles, then proceed to the next section to register the new OS image.)

1. From the left navigation menu in the Apstra GUI, navigate to **Devices > Device Profiles**, select your device and update the OS version in the **Selector** section.
2. From the left navigation menu, navigate to **Platform > Developers > Graph Explorer** and find the ID for the device profile. You can find it with the query variables { device_profile_nodes { id label } }

   In this example, the "id" for the label "Clone DCS-7160-48YC6_abc" is "35a376ad-6ba1-42ec-bfe9-7810c56003d3".



3. Use apstra-cli to update the device profile.

   You can use your blueprint ID and the node ID from the previous step, then set the proper model ID ("DCS-7160-48YC6" for example), and execute.

   apstra-cli command format:

```
blueprint set-node-property --blueprint <your blueprint ID> --node_type
device_profile --node <node ID from Step2> --property selector
--value-fn '{"os_version":"4\.(18|20|21|22|23)\..*","model":"<your model>"
,"os": "EOS","manufacturer": "Arista"}'
```

Example:

```
apstra-cli> blueprint set-node-property --blueprint
a74906ab-1c7a-42ee-bbea-7a0be2572bc2 --node_type device_profile
--node 35a376ad-6ba1-42ec-bfe9-7810c56003d3 --property selector
--value-fn '{"os_version":"4\.(18|20|21|22|23)\..*","model":"DCS-7160-48YC6",
"os": "EOS","manufacturer": "Arista"}'
```

4. From the Apstra GUI, navigate to your blueprint, click **Uncommitted** and commit the changes.
5. Proceed to the next section to upgrade the OS in the same manner as for devices using predefined device profiles.

Register / Upload OS Image

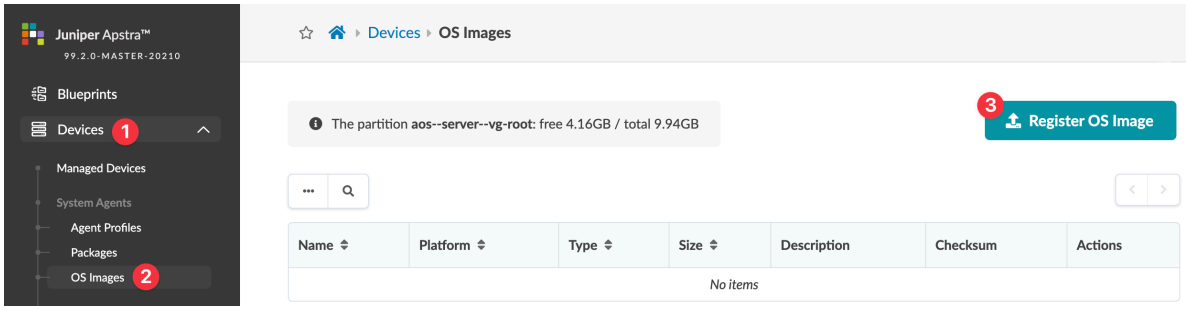
IN THIS SECTION

1. Obtain the OS image from the device vendor.

   CAUTION: Make sure to select a compatible device operating system image for the device that you're upgrading. If you use an incompatible image and the upgrade fails, the deployment lock is not released automatically, even if you recover the device. To release the deployment lock and activate the device again, remove the device assignment from the blueprint, decommission and normalize the device (from Devices > Managed Devices), then reassign the device to the blueprint. For assistance, contact "Juniper Support" on page 1374.

2. From the left navigation menu, navigate to **Devices > System Agents > OS Images** and click **Register OS Image** (top-right).

   You can see how much space is left for uploading new NOS images; if the partition has under 5GB of free space a warning appears in the dialog that opens.

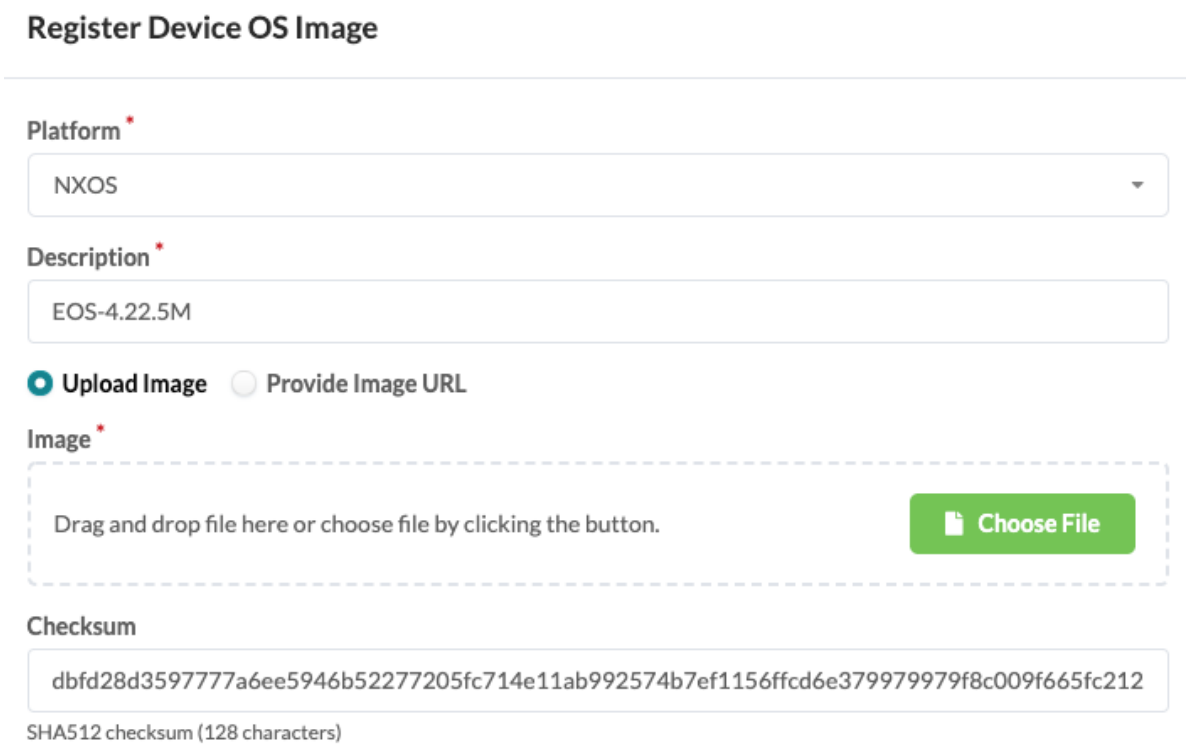


3. Select the platform from the drop-down list (EOS, NXOS, SONIC, JUNOS) and enter a description.
4. Either upload the image directly to the Apstra server or provide a URL download link pointing to an image file on an accessible HTTP server (described in sections below).

Method One: Upload Image

1. Select **Upload Image**, then either click **Choose File** and navigate to the image on your computer, or drag and drop the image from your computer into the dialog window and click **Open**

.



2. Add a checksum (optional) (described in section below).
3. Click **Upload**.

Apstra validates that the software package is supported by the switch OS. If it isn't supported (because the file extension is wrong, for example), then the upload fails immediately, before uploading begins.

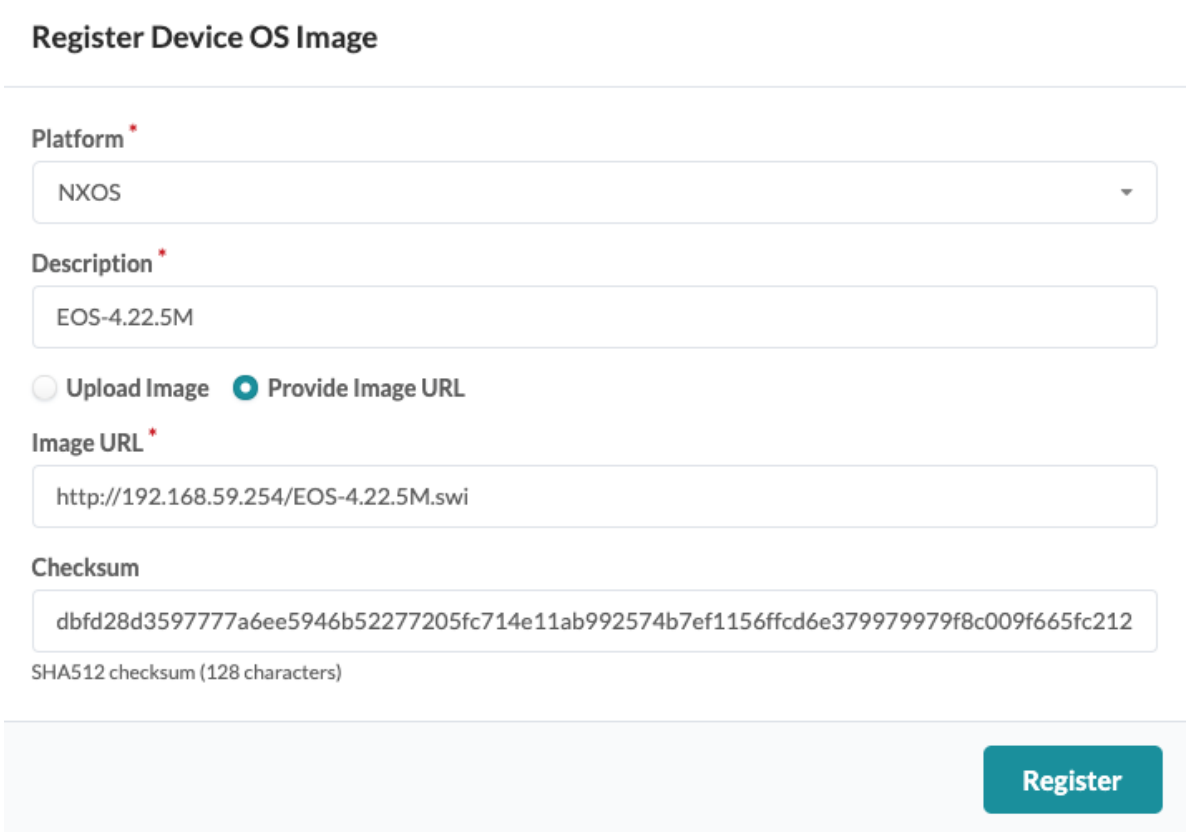
Apstra validates the (optional) checksum. If it can't be verified, the upload process fails immediately, before uploading begins.

If all validation passes, the image is uploaded and appears in the table view.

Method Two: Provide Image URL

If another HTTP server is accessible to the devices being upgraded via their network management port, you can register the OS Image instead of uploading it. HTTP and HTTPS URLs are supported. (FTP, SFTP, SCP and others are not supported.)

1. Select **Provide Image URL**.



2. Enter the URL that points to the image on the other server.
3. Add a checksum (optional) (described in the section below).
4. Click **Register**.

Apstra validates the (optional) checksum. If it can't be verified, the process stops.

If validation passes, the image appears in the table view.

Add Checksum (Optional)

The platform determines the type of checksum that's used:

- Juniper Junos - MD5 (32 characters) or SHA256 (64 characters)
- Enterprise SONiC - MD5 (32 characters)
- Cisco NX-OS - SHA512 (128 characters)
- Arista EOS - SHA512 (128 characters)

If the device vendor provides a checksum file, we recommend that you download the file and copy it to the Checksum field. If a checksum file is not available, you can generate a checksum with the Linux **md5sum** or **shasum** commands, as applicable, or with equivalent programs.

```
$ shasum -a 512 EOS-4.20.11M.swi
dbfd28d3597777a6ee5946b52277205fc714e11ab992574b7ef1156ffcd6e379979979f8c009f665fc21212e4d38d1794
a412d79bab149f859aa72be417c0975  EOS-4.20.11M.swi
$
```

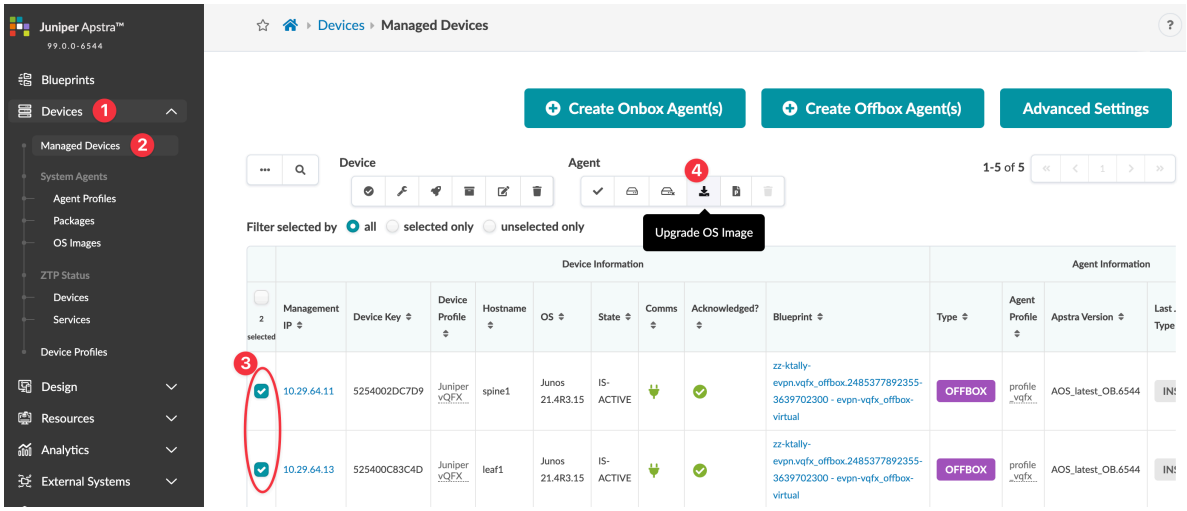
Upgrade OS Image

Make sure that your devices are in the appropriate states for upgrading as described in the overview above, and that if your device profiles are user-defined that you've updated them accordingly.

Before upgrading the device NOS, Apstra rolls back device configuration to its pristine state. It then performs the upgrade and reboots the device, pushing the intended blueprint configuration to devices. Before Apstra pushes this configuration, the device goes live while configured with its pristine configuration, which usually includes interfaces that are up. This means traffic could be blackholed before the intended blueprint configuration is pushed. As of Apstra version 5.0.0, Apstra will automatically shut down interfaces during upgrade. If you want interfaces to remain up during upgrade, then navigate to **Devices > Managed Devices**, click **Advanced Settings**, select the **Skip Shutting Down Interfaces During Upgrade** check box, then click **Update**.

1. From the left navigation menu, navigate to **Devices > Managed Devices**, and select the check box(es) for the device(s) to upgrade. (If you have many devices, use the query function to filter selections.) All selected devices must be of the same type, and they must be upgraded to the same image and

version. To search for specific devices (such as for all EOS devices) click the **Search** button (magnifying glass) and enter a query.



2. Click the **Upgrade OS Image** button (above table in **Agent** section). The dialog lists the available OS images that match the selected devices.
3. Select the appropriate image and click **Upgrade OS Image**. You can monitor the upgrade status from the **Active Jobs** section at the bottom of the page.
4. After the image is uploaded, if a checksum is provided with the OS image, the image checksum is verified. If the MD5/SHA512 checksum is incorrect, or if any other failures occur (such as for insufficient disk space, incorrect remote URL, or when the device NOS version is not changed post upgrade), the job state changes to **FAIL** and the device does not reboot.

> **NOTE**: If an issue arises with the OS image (such as interrupted download or invalid URL) during a NOS upgrade, you are informed before any device configuration is changed. You can then resolve the issue and restart the upgrade process.

5. If the job fails, click the agent to view errors. You can also click the **Show Log** button to view the detailed Ansible job. If an upgrade fails, you must manually resolve the issue causing the failure. For example, with a checksum error, you must either correct the invalid checksum or register a new OS image with a correct checksum, then repeat the upgrade process.
6. If the checksum is correct and no other failures occur, the job state changes to **SUCCESS** and the device reboots.
7. When the device has rebooted with the new image and has reestablished its agent connection with the controller, the upgrade is complete. The **Managed Devices** page displays the new OS version.

Device AAA

IN THIS SECTION

Overview

RADIUS and TACACS+ device AAA (authentication, authorization and accounting) frameworks are supported on Juniper, Cisco and Arista devices. Device AAA is optional and correct implementation is the responsibility of the end user. Minimum requirements for correct Apstra AAA implementations are described below.

> ⚠ **CAUTION**: When using AAA framework we recommend adding a local Apstra user to devices. If AAA authentication or authorization fails when Apstra performs a full configuration push, manual recovery (config push) is required.

You can apply AAA configuration in one of two ways as described below:

Configlets (Recommended)

You add configuration to a configlet, then you import it into a blueprint. Local credentials must be available from the Apstra environment so the device can be added and the configlet can be applied.

> ⚠ **CAUTION**: Before you upgrade the Apstra server, device agent, or NOS, you **must** delete device AAA/TACACS configlets from blueprints. After the upgrade is complete, you can re-apply them.

User-required

Instead of using configlets, you can add configuration before acknowledging a device, so it becomes part of the Pristine Config. For more information, see "Device Configuration Lifecycle" on page 634.

Juniper Junos

CAUTION: Credentials for the Junos offbox system agent user must always be valid and available. When using the AAA framework we recommend that you add a local user to devices and use it for Apstra offbox system agents. Always have “password” be first in Junos config for authentication-order as follows:

```
authentication-order [ password radius ]
```

Cisco NX-OS

CAUTION: A remote user could erratically be removed from NX-OS devices, causing authentication and authorization failures. The user (role 'network-admin') must exist on the device in order to manage the device. If not, Apstra functions such as agent installation, telemetry collection and device configuration may fail. The only known workaround is to use local authentication.

The example NX-OS configuration below has been tested to work correctly with Apstra software. This uses both authentication and authorization:

```
tacacs-server key 7 “<key>“
tacacs-server timeout <timeout>
tacacs-server host <host>
aaa group server tacacs+ <group>
  server <host>
  use-vrf management
  source-interface mgmt0

aaa authentication login default group <group>
aaa accounting default group <group> local
```

```
aaa authentication login error-enable
aaa authentication login ascii-authentication
```

Arista EOS

CAUTION: When TACACS+ AAA is configured on EOS devices, device agent upgrades could fail while files are copied from the Apstra server to the device. This commonly happens if TACACS+ uses a custom password prompt. To prevent this type of failure, temporarily disable all TACACS+ AAA where device authentication uses an admin-level username and password for any device agent operations, including upgrades.

RELATED DOCUMENTATION

Configlets Introduction | **908**

CHAPTER 19

Device

IN THIS CHAPTER

Acknowledge Device

If you're deploying a modular device, you may need to change its device profile before acknowledging it. For details, see "Change the Assigned Device Profile (Datacenter)" on page 669 or "Change the Assigned Device Profile (Freeform)" on page 530.

Acknowledging a device puts it in the **Out of Service Ready** state and signals the intent to have Apstra manage it.

1. From the left navigation menu, navigate to **Devices > Managed Devices** and select the check box(es) for the device(s) to be managed by Apstra.
2. Above where you just clicked, click the check mark to **Acknowledge** the selected device(s).
3. Click **Confirm** to acknowledge the device(s) and return to the list view. The **Acknowledged?** field for the device changes to a green check mark and the device state changes to **OOS-READY**.

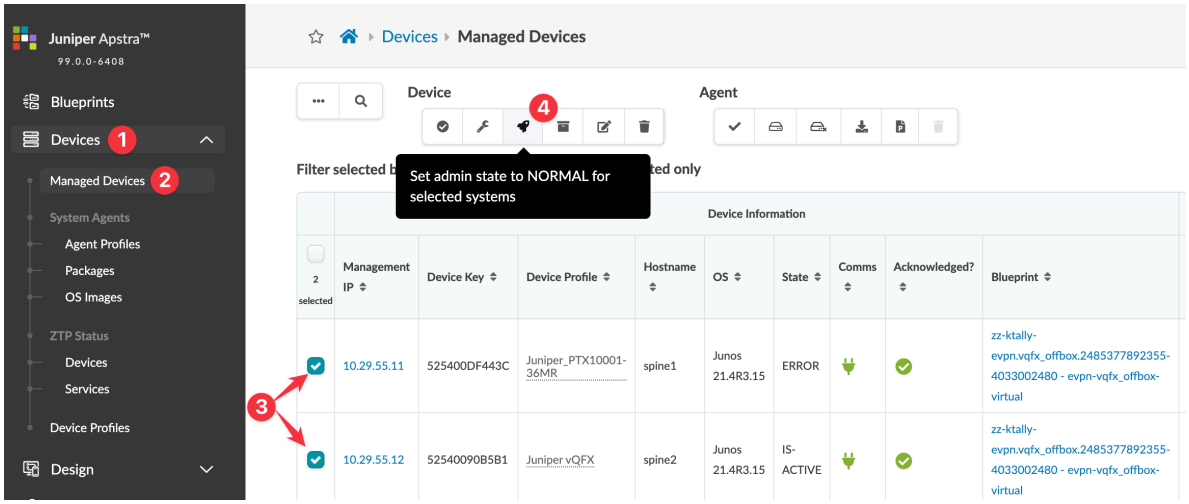
Change Assigned Device Admin States

IN THIS SECTION

You can change the admin state assigned to one or more devices from the **Managed Devices** table, or you can change it for a single device from the **Device** selection page.

Change Admin State from Managed Devices Table

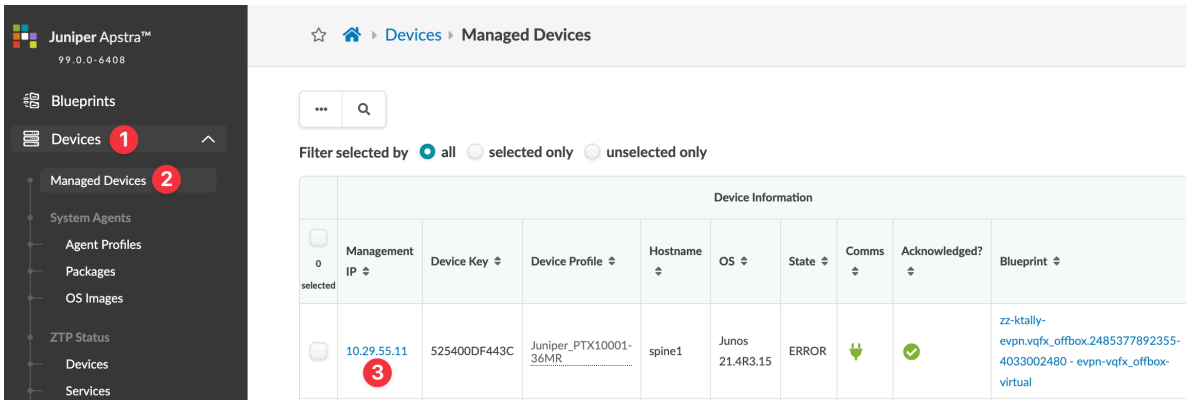
1. From the left navigation menu in the Apstra GUI, navigate to **Devices > Managed Devices** and select the check box(es) for the device(s) with the admin state(s) to change.



2. In the **Device** menu that appears above the table, click the button for the applicable admin state.
   - **Set admin state to NORMAL for selected systems** - If you're "upgrading a device network operating system" on page 655, make sure the admin state is set to **NORMAL** before beginning the process.
   - **Set admin state to DECOMM for selected systems** - If you're decommissioning a device, setting the admin state to **DECOMM** is part of a larger process. See "Remove Device from Managed Devices" on page 673 for the workflow and more details.
   - **Set admin state to MAINT for selected items** - this state is no longer used.
3. Click **Confirm** to change the admin state and return to the **Managed Devices** table.

Change Admin State from Device Selection

1. From the left navigation menu in the Apstra GUI, navigate to **Devices > Managed Devices** and click the management IP for the device with the admin state to change.



2. On the **Device** detail page that opens, click the **Edit** button (upper-right).

3. In the dialog that opens, select the **Admin State**.
   - **Set admin state to NORMAL for selected systems** - If you're "upgrading a device network operating system" on page 655, make sure the admin state is set to **NORMAL** before beginning the process.
   - **Set admin state to DECOMM for selected systems** - If you are decommissioning a device, setting the admin state to **DECOMM** is part of a larger process. See "Remove Device from Managed Devices" on page 673 for the workflow and more details.
   - **Set admin state to MAINT for selected items** - this state is no longer used.
4. Click **Update** to change the admin state and return to the **Device** detail page.

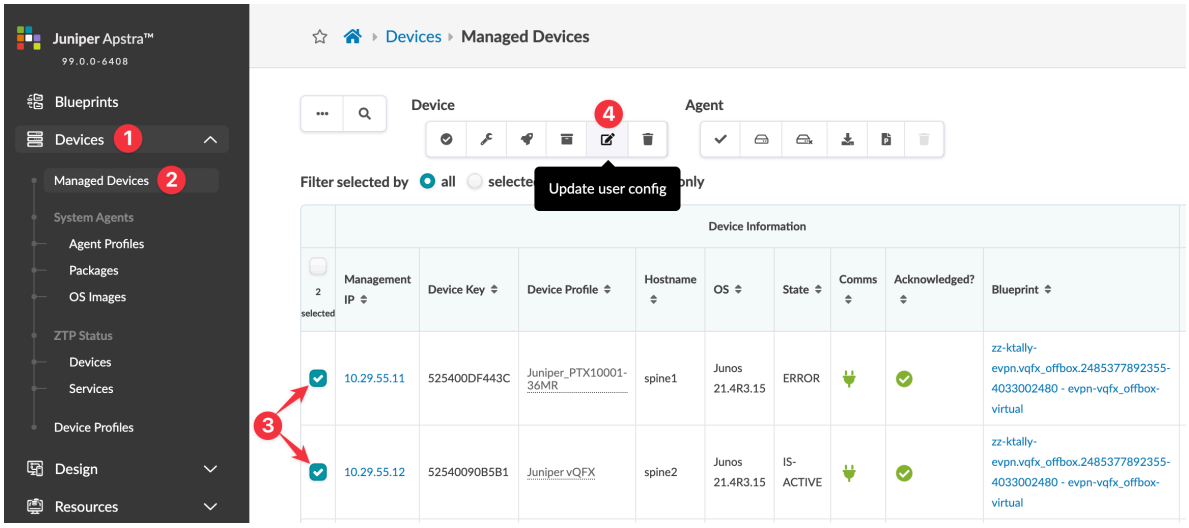
Change Assigned Device Profile (Devices)

IN THIS SECTION

You can change the device profile assigned to one or more devices from the **Managed Devices** table, or you can change it for a single device from the **Device** selection page.

Change Device Profile from Managed Devices Table

1. From the left navigation menu in the Apstra GUI, navigate to **Devices > Managed Devices** and select the check box(es) for the device(s) with the device profile(s) to change.

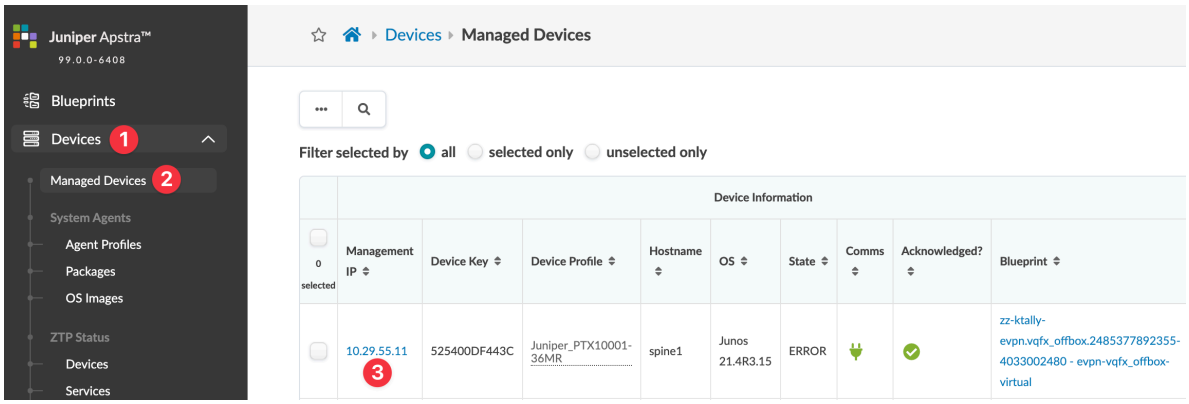


2. In the **Device** menu that appears above the table, click the **Update user config** button (looks like an edit button), select the **Set Device Profile** check box, then select the device profile from the drop-down list.
3. Click **Confirm** to change the device profile and return to the **Managed Devices** table.

If the devices you updated are assigned to a blueprint, you also need to "change the assigned interface map in the blueprint" on page 110.

Change Device Profile from Device Selection

1. From the left navigation menu in the Apstra GUI, navigate to **Devices > Managed Devices** and click the management IP for the device with the device profile to change.



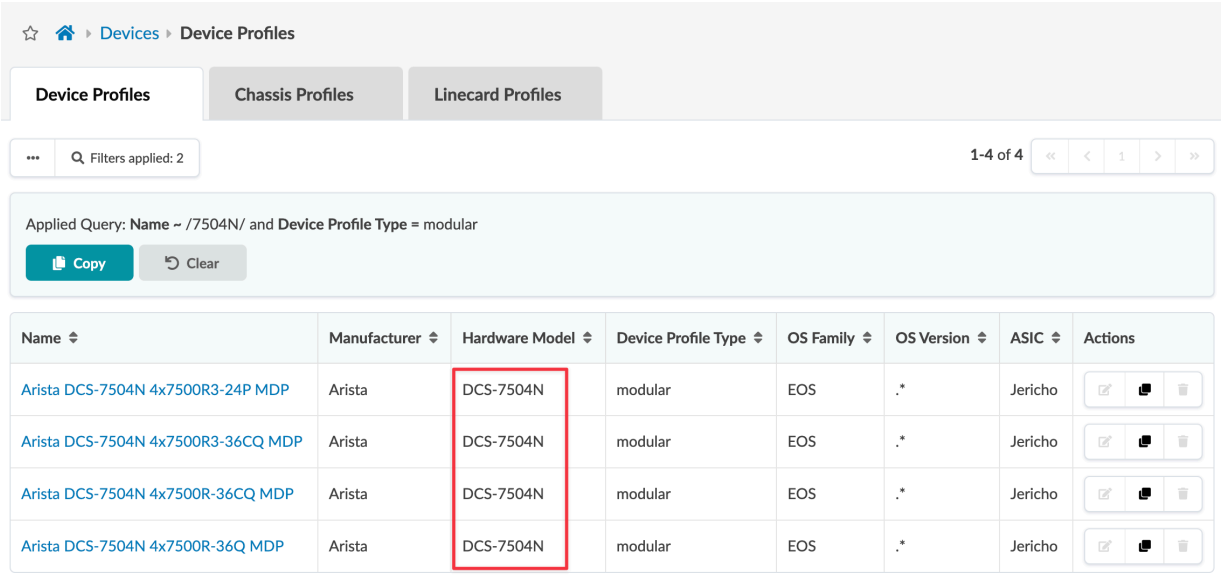
2. On the **Device** detail page that opens, click the **Edit** button (upper-right).

3. In the dialog that opens, select the device profile from the **Device Profile** drop-down list.
4. Click **Update** to change the device profile and return to the **Device** detail page.

If the device you updated is assigned to a blueprint, you also need to "change the assigned interface map in the blueprint" on page 110.

Change Device Profile Example

One example of when you might need to change a device profile is when one device has multiple device profiles associated with it. Device profiles represent different line card configurations.

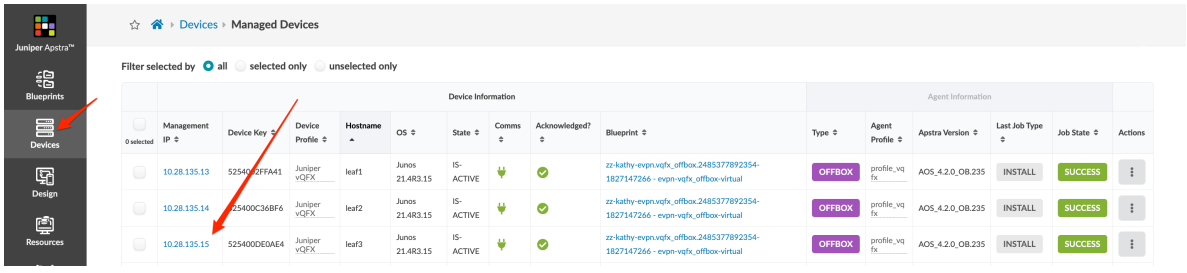


The first device profile that matches the device chassis model (based on the selector model field) is associated with the device (DCS-7504N for example). If you're using a modular device in your network, check that the correct device profile is associated with it. If it's not, change its device profile to the correct one before acknowledging the device and assigning it to a blueprint.

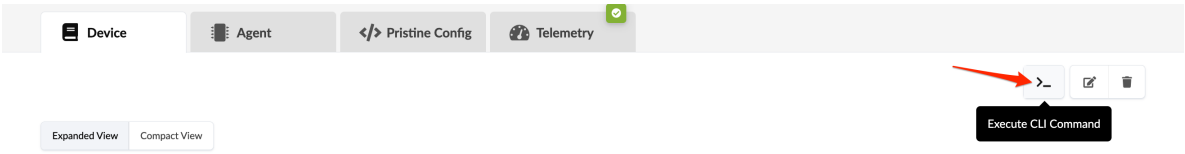
Execute CLI Show Command (Devices)

While in the Apstra environment, you may need device information that's obtained via CLI commands. Traditionally, you need to log in to a machine with access to the device management network, open a terminal, find device IP addresses, SSH to each of them, then run the required CLI commands. You can bypass these steps and run show commands for Juniper devices directly from the Apstra GUI. You can execute CLI commands from within the staged or active blueprint, or from the **Managed Devices** page. The steps below are for the devices page.

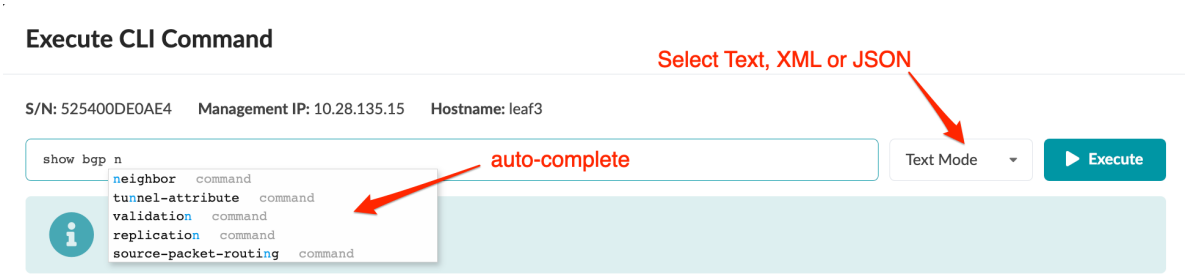
1. From the left navigation menu, navigate to **Devices > Managed Devices** and click the management IP for the device.



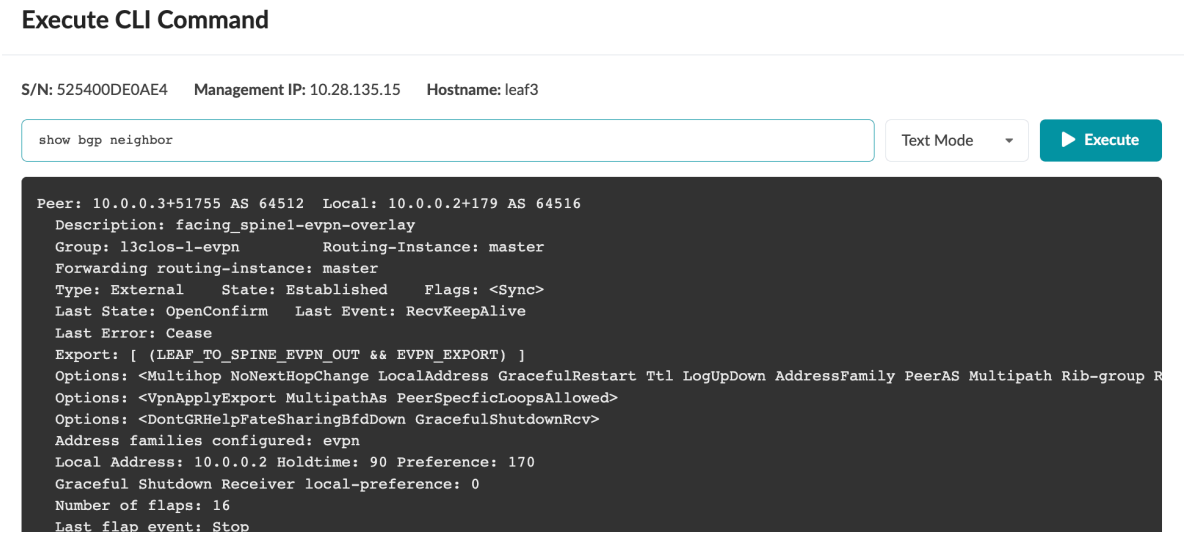
2. On the device detail page that appears, click the **Execute CLI Command** button.



3. In the dialog that opens type `show`, then press the space bar. Available commands appear that you can scroll through to select, or you can start typing the command and it will auto-fill. In our example we're looking for BGP neighbors. We typed `show`, space, then `b`, which filtered the commands to only include those with the letter b. We selected `bgp`, then pressed the space bar to show available arguments for bgp. We typed `n` to show commands including the letter n. We'll select `neighbor` to complete the command.



4. From the drop-down list, select how you want to view the results: text, XML or JSON.
5. Click **Execute** to return show command results. We used **Text Mode** for our example.



RELATED DOCUMENTATION

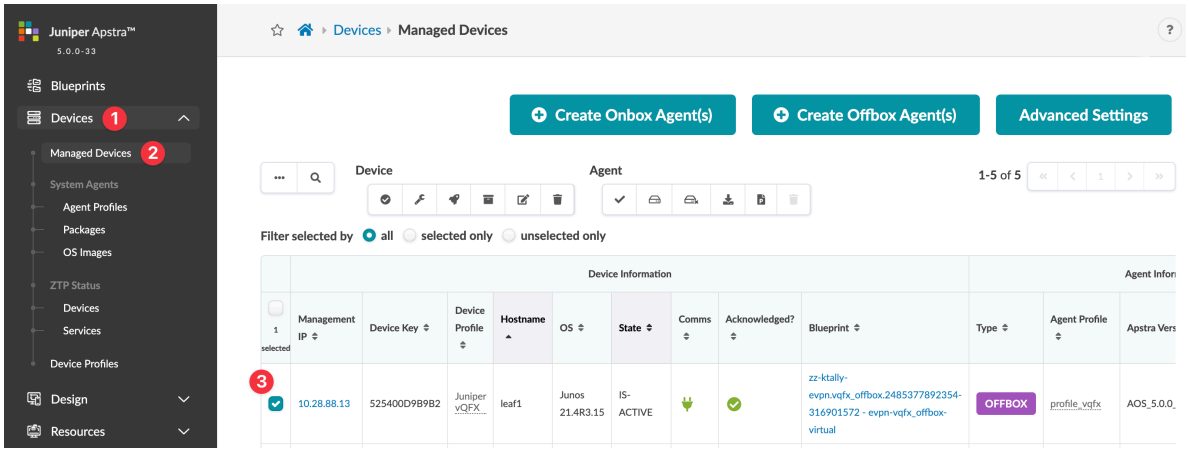
Execute CLI Show Command (Data Center Blueprint) | 89

Execute CLI Show Command (Freeform Blueprint) | 506

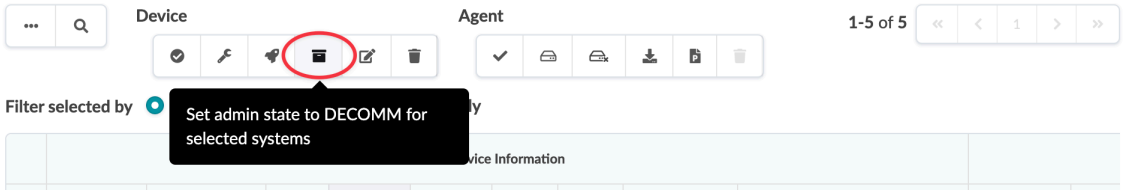
Remove (Decommission) Device from Managed Devices

For successful device removal, it's important to follow these steps in the order specified.

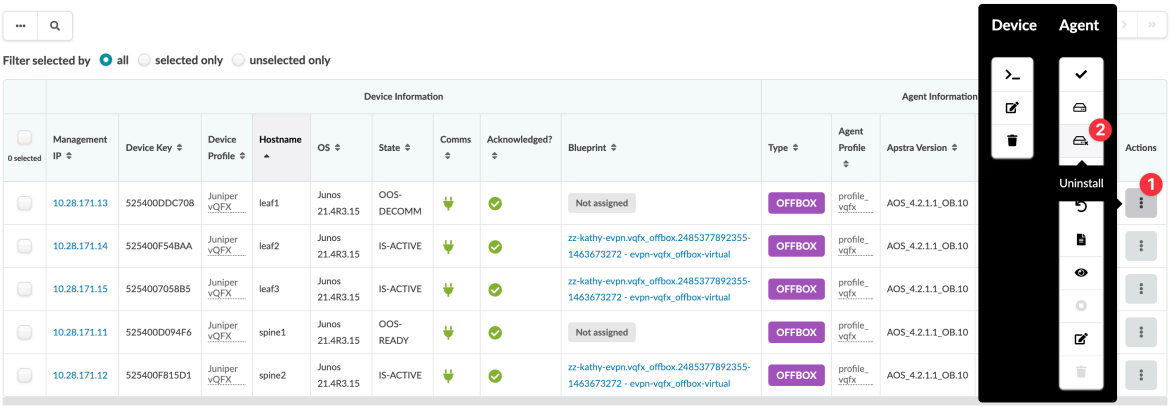
1. If the device is assigned to a blueprint, unassign it from your "datacenter blueprint" on page 85 or "freeform blueprint" on page 534, as applicable.
2. From the left navigation menu, navigate to **Devices > Managed Devices** and check the box for the device to remove from Apstra management.



3. In the **Device** Actions panel that appears above the table, click the **Set admin state to DECOMM for selected systems** button, then click **Confirm** to set the admin state and return to the table. (If the device is assigned to a blueprint, you can't decommission the device.)

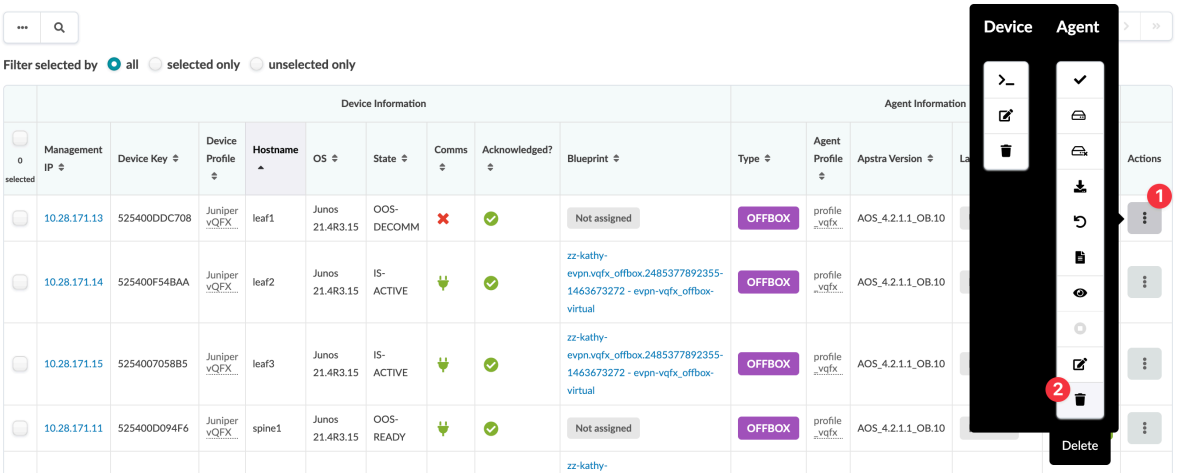


4. Click the three dots in the **Actions** panel for the device to remove, then in the **Agent** Actions panel that opens, click the **Uninstall** button.



The **Uninstall This System Agent?** dialog opens.

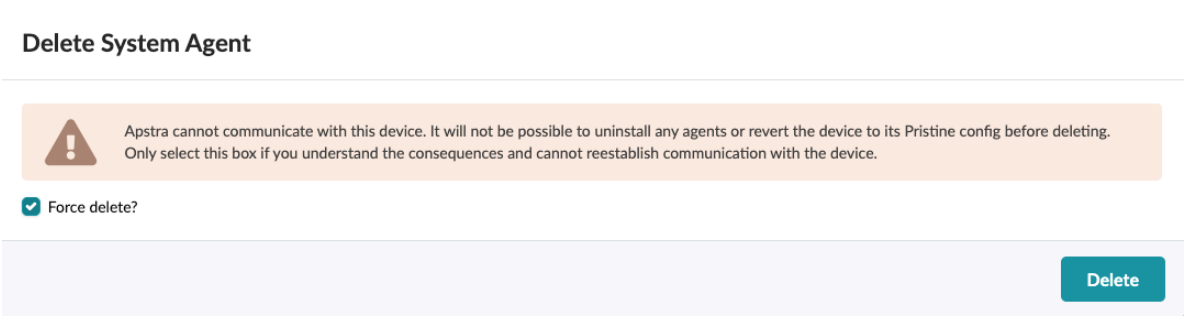
5. Click **Confirm** to uninstall the device and return to the Managed Devices table.
6. Click the three dots in the **Actions** panel again for the device to remove, then in the **Agent** Actions panel that opens, click the **Delete** button.



The **Delete System Agent** dialog opens.

7. Click **Delete** to delete the agent and return to the Managed Devices table.

   If the device is unreachable, the dialog includes a warning with the option to force delete the agent. If you decide to force delete, check the **Force delete** check box, then click **Delete** to force delete the agent and return to the table view.



8. Click the three dots in the **Actions** panel again for the device to remove, then in the **Device** Actions panel that opens, click the **Delete** button (trash can icon).
   The **Delete this resource?** dialog opens.
9. Click the **Delete** button to remove the device from Apstra management and return to the Managed Devices table. (If the device is not in STOCKED or DECOMM stage, you can't delete the device.) Device(s) are disconnected from the Apstra server and removed from the Apstra database.

If you're replacing the device you just removed, follow the steps to "add" on page 651 the replacement device to Managed Devices.

Delete Device

IN THIS SECTION

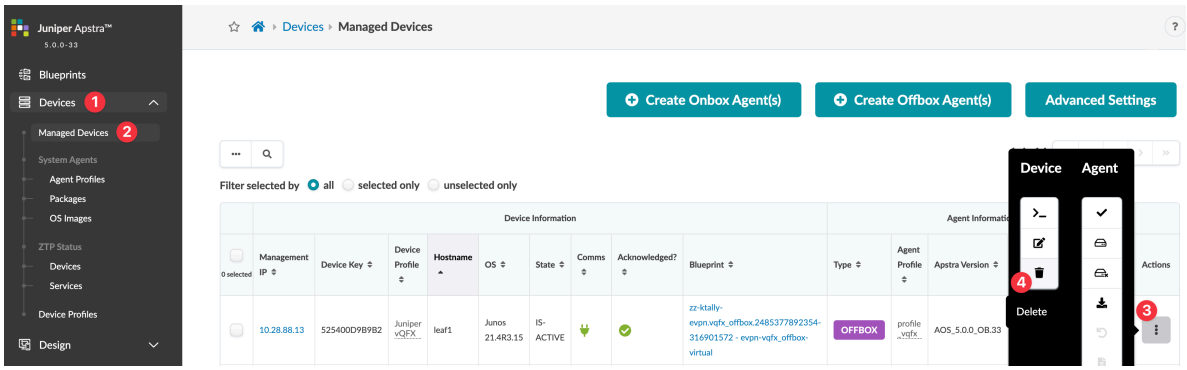
You can delete one or more devices from the **Managed Devices** table, or you can delete a single device from the **Device** selection page.

If you want to remove a device from Apstra management, see "Remove (Decommission) Device from Managed Devices" on page 673 for the complete workflow. There are additional steps before deleting the device.

If the device to be deleted has not been "acknowledged" on page 651, you can delete the device as shown below.

Delete One Device from Managed Device Table

1. From the left navigation menu, navigate to **Devices > Managed Devices** and click the three dots in the **Actions** column (right side) for the device to delete, then click the **Delete** button in the **Device** menu.

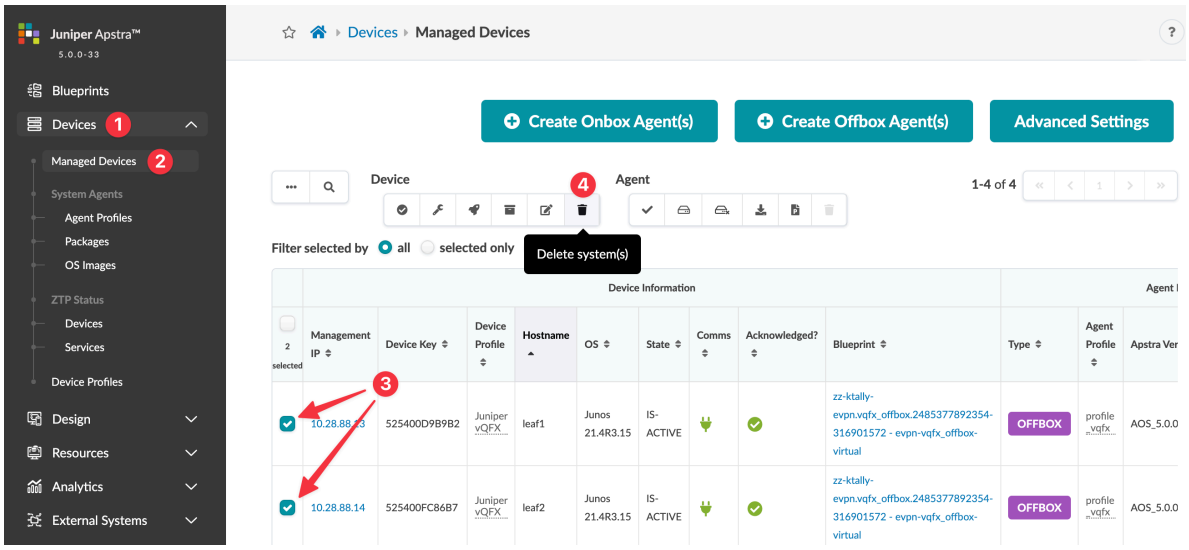


   The **Delete device?** dialog opens.

2. Click **Delete** to delete the device and return to the Managed Devices table.

Delete Multiple Devices from Managed Devices Table

1. From the left navigation menu, navigate to **Devices > Managed Devices** and check the box(es) for the device(s) to delete, then in the **Device** Actions panel that appears above the table, click the **Delete system(s)** button.

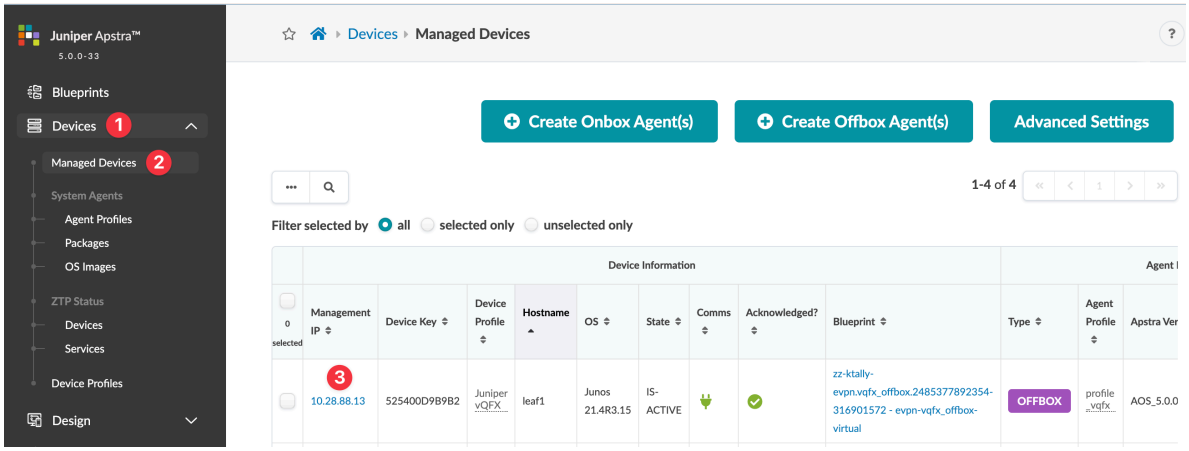


The **Delete systems** dialog opens.

2. Click **Confirm** to delete the device and return to the Managed Devices table. (If the device is not in STOCKED or DECOMM stage, you can't delete the device.) Device(s) are disconnected from the Apstra server and removed from the Apstra database.

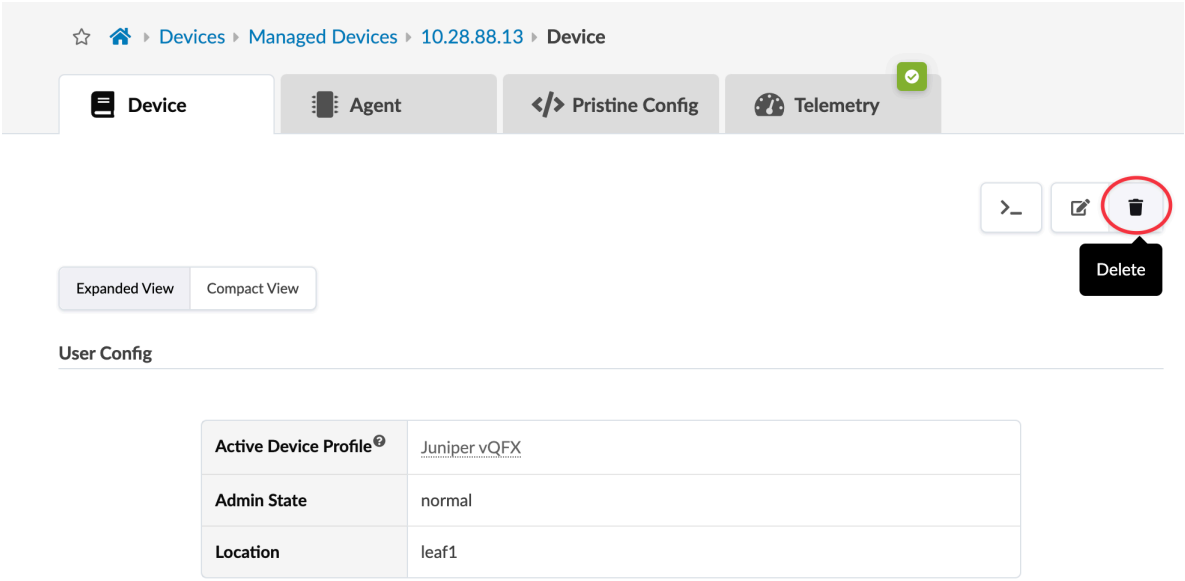
Delete Device from Device Selection

1. From the left navigation menu, navigate to **Devices > Managed Devices** and click the Management IP (not the check box) for the device to delete.



The **Device** page for the selected device opens.

2. Click the **Delete** button (right side).



The **Delete device?** dialog opens.

3. Click **Delete** to delete the device and return to the Managed Devices table.

CHAPTER 20

Agent

IN THIS CHAPTER

What are System Agents

IN THIS SECTION

Apstra device system agents handle configuration management, device-to-server communication, and telemetry collection. If you're not using "Apstra ZTP" on page 713 to bootstrap your devices (or if you have a one-off installation) you can use this device installer to automatically install and verify devices. Depending on the device NOS, you can install device agents on-box (agent is installed on the device) or off-box (agent is installed on the Apstra server and communicates with devices via API). To find out which platforms support onbox and offbox agents, see the **Device Management** section of the "Apstra 5.0.0 Feature Matrix" on page 1481.

For more information about managing devices in the Apstra environment, see "Managed Devices" on page 647.

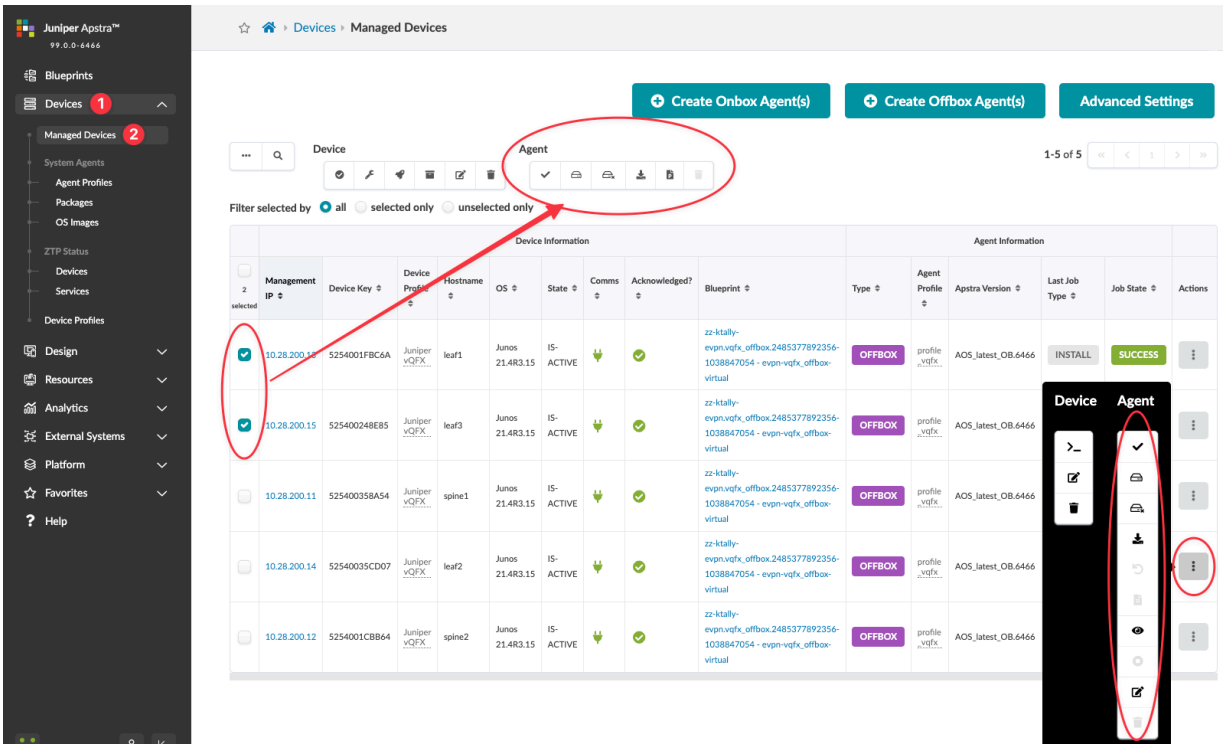
Agents include the following parameters:

Table 4: Device System Agent Parameters

Parameter	Description
Device addresses	Management IP(s) of the device(s)
Platform (off-box only)	For off-box agents only: drop-down list includes supported platforms.
Username / Password	If you're not using an agent profile with credentials, check these boxes and add credentials.
Agent Profile	If you don't want to manually enter credentials and packages, use agent profiles that you previously defined.
Job to run after creation	• Install (default) - installs the agent on the device • Check - creates the agent, but does not install it. It appears in the list view where you can install it later.
Install Requirements (servers only)	For servers only: If servers don't have Internet connectivity, uncheck the box.
Packages	Before creating the agent, install required packages so they are available. Packages associated with selected agent profiles are listed here as well.
Open Options (off-box only)	Passes configured parameters to off-box agents. For example, to use HTTPS as the API connection from off-box agents to devices, use the key-value pair: proto-https - port-443. The following default values can be overridden with open options: • commit_timeout - 60 (integer: seconds) • telemetry_timeout - 100 (integer: seconds) • probe_timeout: 5 (integer: seconds) • log_config_diff - True (boolean)

System Agents in the Apstra GUI

From the left navigation menu, navigate to **Devices > Managed Devices** to go to the managed devices table.



To perform a task related to agents on one or more devices, select their check boxes (first column in table). The **Agent** menu appears above the table with the available tasks (check, install, uninstall, upgrade OS image, assign profile, delete) for the selected agents.

To perform a task related to an agent on a single device, click the **Actions** button for the device. The **Agent** menu appears vertically with the available tasks (check, install, uninstall, OS upgrade, revert, collect pristine config, show log, cancel active job, edit, delete).

See next pages for details on performing tasks related to agents.

Create Onbox Agent

To create onbox agents, you need **full admin / root** privileges. We recommend that you create a dedicated user on the device using "Apstra ZTP" on page 713 or other means. Before installing agents, make sure to do the following:

- Add login credentials for the devices.

- Configure management IP connectivity between devices and the Apstra server. You must do this before installing agents so it's out-of-band (OOB). Configuring management connectivity in-band (through the fabric) is not supported and could cause connectivity issues when you make changes to the blueprint.
- Upload required packages.

1. Configure the minimum configuration on your devices as shown below, as applicable:

Juniper Junos OS Evolved Onbox Agent Minimum Configuration

NOTE: The minimum release version for Junos OS Evolved switches on onbox agents is 22.4R3.

```
system {
    login {
        user aosadmin {
            uid 2000;
            class super-user;
            authentication {
                encrypted-password "xxxxx";
            }
        }
    }
    services {
        ssh;
        netconf {
            ssh;
        }
    }
    management-instance;
}
interfaces {
    em0 {
        unit 0 {
            family inet {
                address <address>/<cidr>;
            }
        }
    }
```

```
}
routing-instances {
    mgmt_junos {
        routing-options {
            static {
                route 0.0.0.0/0 next-hop <management-default-gateway>;
            }
        }
    }
}
```

The minimum release version for Junos OS Evolved switches on onbox agents is 22.4R3.

Cisco NX-OS Onbox Agent Minimum Configuration

```
!
copp profile strict
!
username admin password <admin-password> role network-admin
!
vrf context management
  ip route 0.0.0.0/0 <management-default-gateway>
!
interface mgmt0
  ip address <address>/<cidr>
!
```

Arista EOS Onbox Agent Minimum Configuration

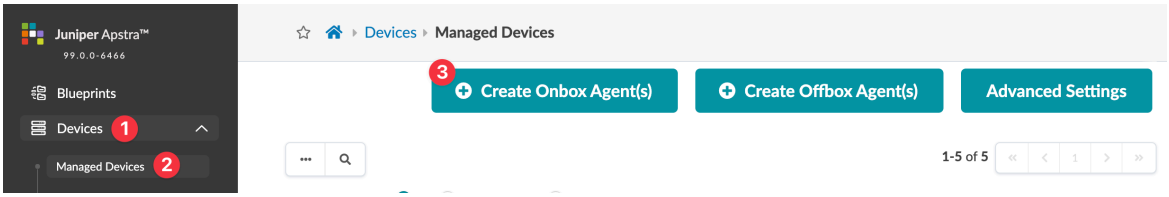
```
!
service routing protocols model multi-agent
!
aaa authorization exec default local
!
username admin privilege 15 role network-admin secret <admin-password>
!
interface Management1
   ip address <address>/<cidr>
!
```

```
ip route vrf management 0.0.0.0/0 <management-default-gateway>
!
```

SONiC

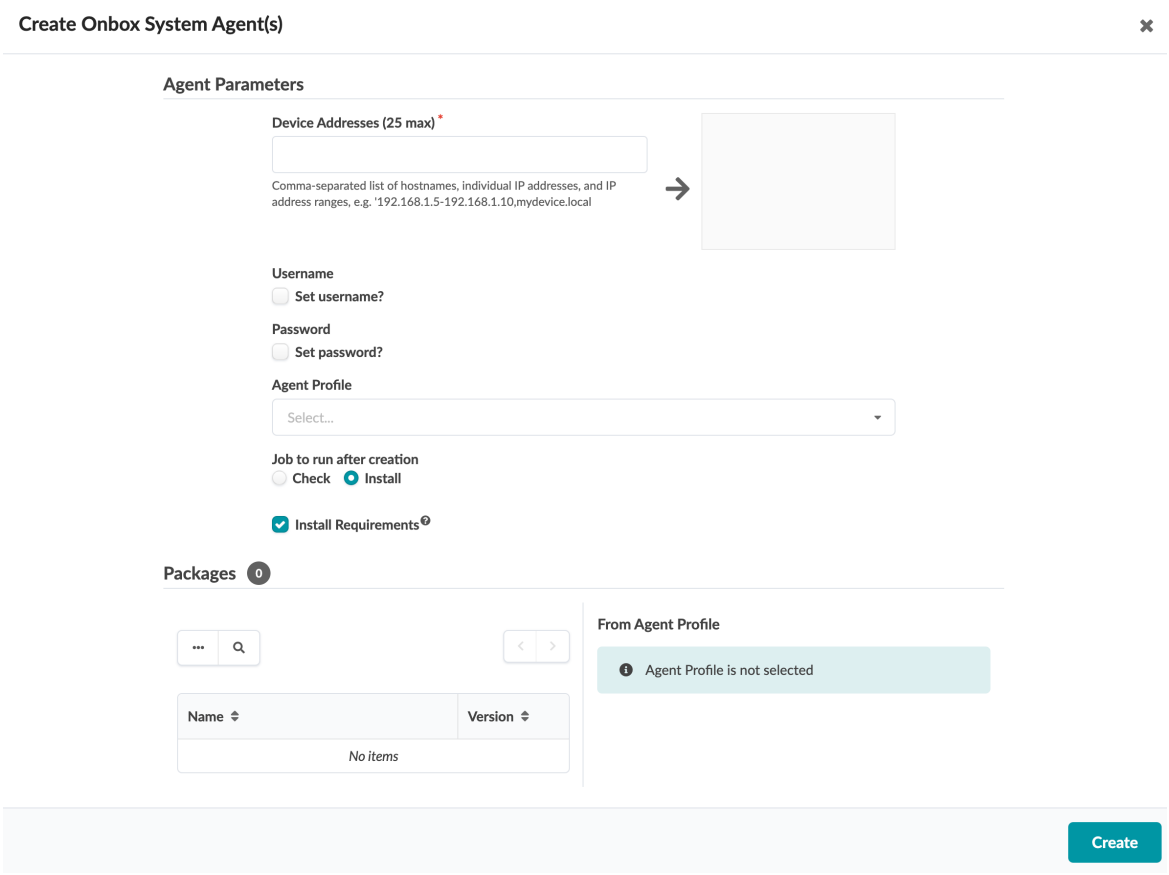
SONiC has no specific configuration requirements other than Management Network and privileged user access.

2. Some configuration could raise validation errors. Make sure the following configuration is *not* on the devices (and any other configuration that would raise validation errors):
   - VLANs other than VLAN 1
   - VRFs other than "management"
   - Interface IP addresses other than "management"
   - Loopback interfaces
   - VLAN interfaces
   - VXLAN interfaces
   - AS-Path access-lists
   - IP prefix-lists
   - Route maps or policies
   - BGP configuration
3. From the left navigation menu, navigate to **Devices > Managed Devices** and click **Create Onbox Agent(s)**.



The **Create Onbox System Agent(s)** dialog opens.

4. Enter up to 25 device IP addresses in the **Device Addresses** field.



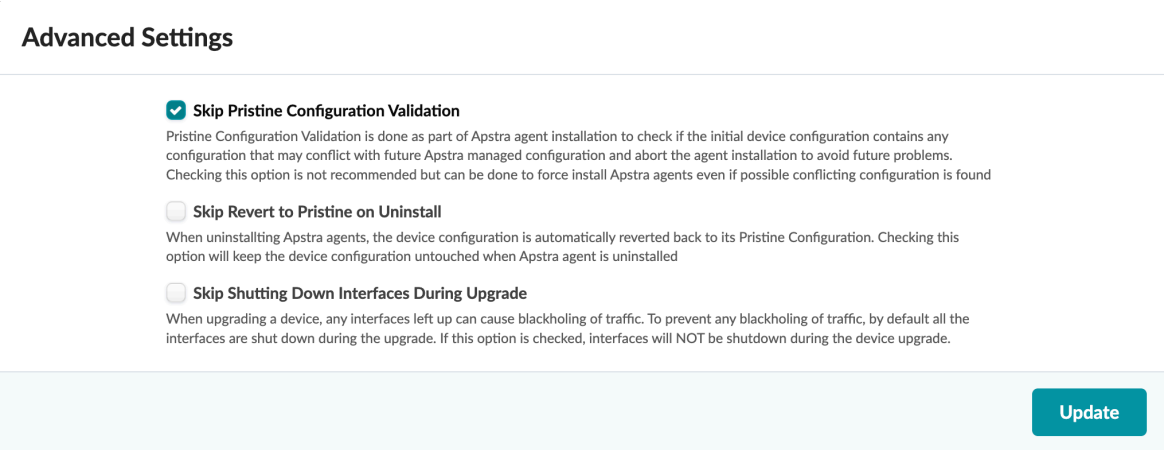
NOTE: If you're creating an agent for Junos EVO that has dual routing engines, create it with a master-only address. If you don't, then when routing engine master switchover occurs, the agent may not work correctly or it may introduce problems.

5. If you're not using an agent profile with credentials, select the check boxes for username and password and add credentials.
6. If you *are* using "agent profiles" on page 696 (that you previously defined), select the agent profile from the **Agent Profile** drop-down list, so you don't have to manually enter credentials and packages.
7. Select the job to run after creation:
   - Install (default) - installs the agent on the device
   - Check - creates the agent, but does not install it. It appears in the table view where you can install it later.
8. **Install Requirements** is for servers. If servers don't have Internet connectivity, deselect the box.
9. "Packages" on page 700 that you've previously installed appear in the **Packages** section. Packages associated with selected agent profiles are listed here as well. Select packages, as required.
10. Click **Create**.

During the agent install process, device configuration is validated; if the device contains configuration that could prevent service configuration from deploying, the agent install process raises an error.

In this case, check the device log for error details (Navigate to **Devices / Managed Devices**, click the three dots in the device's **Actions** panel (right column), then in the **Agent** menu click the **Show Log** button (eyeball).) Manually remove conflicting configuration and start the agent installation process again.

If you must complete the agent installation with configuration validation errors, you can disable pristine configuration validation. To do this, from **Devices > Managed Devices**, click **Advanced Settings** (top-right), select **Skip Pristine Configuration Validation**, then click **Update**.



For information about retaining pre-existing configuration when bringing devices under Apstra management, see "Device Configuration Lifecycle" on page 634.

> **NOTE**: On some platforms (Junos for example) you can configure rate-limiting for management traffic (SSH for example). When the Apstra server interacts directly with devices it can be more bursty than when it interacts with a user. Rate-limiting configurations that are used for hardening security can impact device management, and lead to deployment failures and other agent-related issues.

While the task is active you can view its progress at the bottom of the screen in the **Active Jobs** section. The job status changes from **Initialized** to **In Progress** to **Succeeded**.

Create Offbox Agent

Before installing offbox agents, make sure to do the following:

- Add login credentials for the devices.
- Configure management IP connectivity between devices and the Apstra server. You must do this before installing agents so it's out-of-band (OOB). Configuring management connectivity in-band (through the fabric) is not supported and could cause connectivity issues when changes are made to the blueprint.
- Upload required packages.
- If you're using Juniper offbox agents, "increase the application memory usage" on page 1325.

1. Configure the minimum configuration on your devices as shown below, as applicable:

   Juniper Junos Offbox Agent Minimum Configuration

```
system {
    login {
        user aosadmin {
            uid 2000;
            class super-user;
            authentication {
                encrypted-password "xxxxx";
            }
        }
    }
    services {
        ssh;
        netconf {
            ssh;
        }
    }
    management-instance;
}
interfaces {
    em0 {
        unit 0 {
            family inet {
                address <address>/<cidr>;
            }
        }
    }
}
routing-instances {
```

```
    mgmt_junos {
        routing-options {
            static {
                route 0.0.0.0/0 next-hop <management-default-gateway>;
            }
        }
    }
}
```

Juniper Junos OS Evolved Offbox Agent Minimum Configuration

```
system {
    login {
        user aosadmin {
            uid 2000;
            class super-user;
            authentication {
                encrypted-password "xxxxx";
            }
        }
    }
    services {
        ssh;
        netconf {
            ssh;
        }
    }
    management-instance;
}
interfaces {
    em0 {
        unit 0 {
            family inet {
                address <address>/<cidr>;
            }
        }
    }
}
routing-instances {
    mgmt_junos {
        routing-options {
```

```
            static {
                route 0.0.0.0/0 next-hop <management-default-gateway>;
            }
        }
    }
}
```

The minimum release version for Junos OS Evolved switches on onbox agents is 22.4R3.

Cisco NX-OS Offbox Agent Minimum Configuration

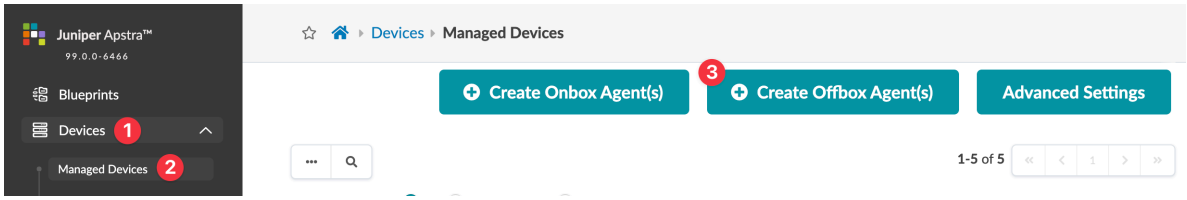
```
!
feature nxapi
feature bash-shell
feature scp-server
feature evmed
copp profile strict
nxapi http port 80
!
username admin password <admin-password> role network-admin
!
vrf context management
  ip route 0.0.0.0/0 <management-default-gateway>
!
nxapi http port 80
!
interface mgmt0
  ip address <address>/<cidr>
!
```

Arista EOS Offbox Agent Minimum Configuration

```
!
service routing protocols model multi-agent
!
aaa authorization exec default local
!
username admin privilege 15 role network-admin secret <admin-password>
!
vrf definition management
```

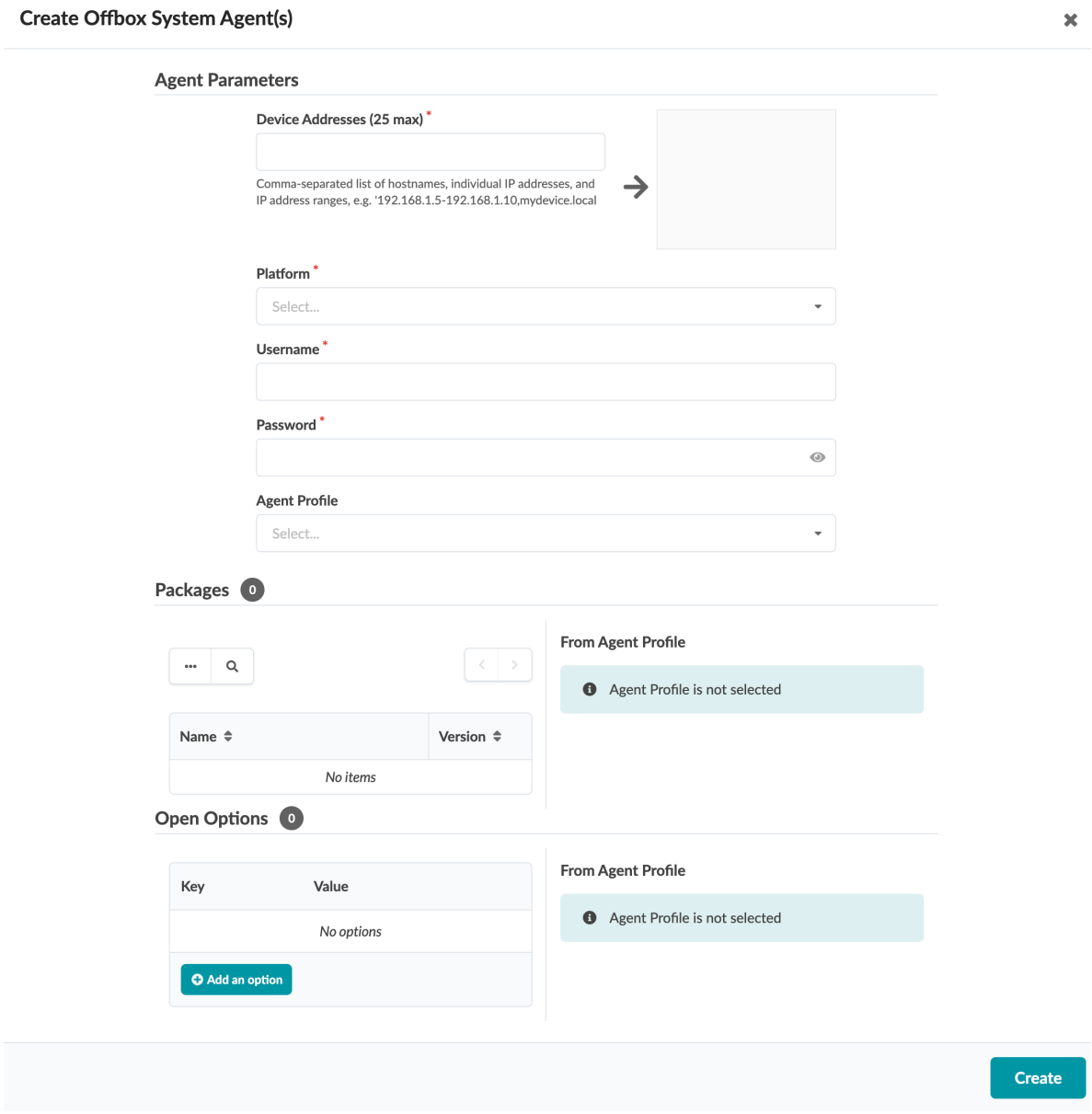
```
   rd 100:100
!
interface Management1
   vrf forwarding management
   ip address <address>/<cidr>
!
ip route vrf management 0.0.0.0/0 <management-default-gateway>
!
management api http-commands
   protocol http
   no shutdown
   !
   vrf management
      no shutdown
!
```

2. Some configuration could raise validation errors. Make sure the following configuration is *not* on the devices (and any other configuration that would raise validation errors):
   - VLANs other than VLAN 1
   - VRFs other than "management"
   - Interface IP addresses other than "management"
   - Loopback interfaces
   - VLAN interfaces
   - VXLAN interfaces
   - AS-Path access-lists
   - IP prefix-lists
   - Route maps or policies
   - BGP configuration
3. From the left navigation menu, navigate to **Devices > Managed Devices** and click **Create Offbox Agent(s)**.



The **Create Offbox System Agent(s)** dialog opens.

4. Enter up to 25 device IP addresses in the **Device Addresses** field.



> **NOTE**: If you're creating an agent for Junos EVO that has dual routing engines, create it with a master-only address. If you don't, then when routing engine master switchover occurs, the agent may not work correctly or it may introduce problems.

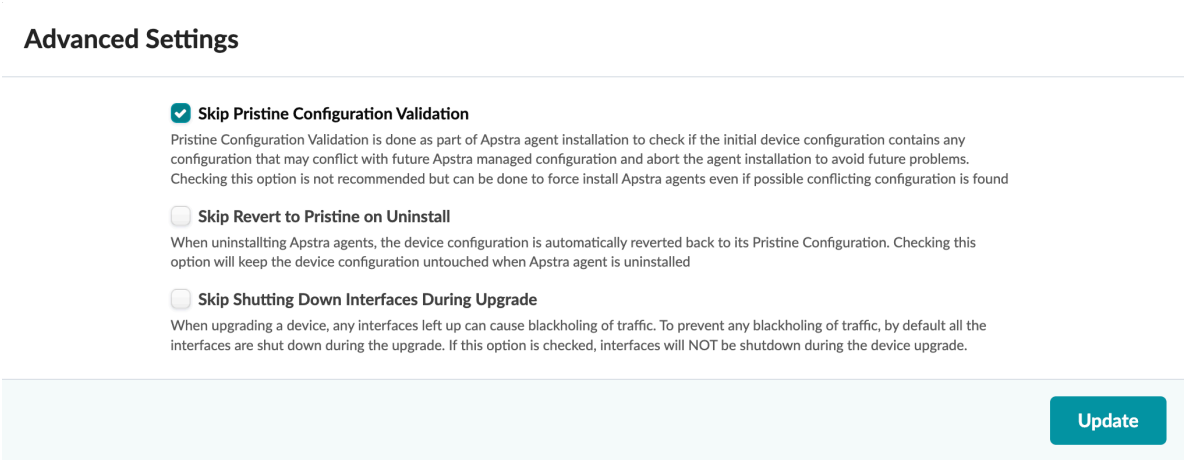
5. Select the relevant platform from the **Platform** drop-down list (EOS, Junos, NX-OS.) (Offbox agents are not supported on SONiC).
6. If you're not using an agent profile with credentials, enter username and password.

7. If you *are* using "agent profiles" on page 696 (that you previously defined), select the agent profile from the **Agent Profile** drop-down list, so you don't have to manually enter credentials and packages.
8. "Packages" on page 700 that you've previously installed appear in the **Packages** section. Packages associated with selected agent profiles are listed here as well. Select packages, as required.
9. Click **Create**.

   During the agent install process, device configuration is validated; if the device contains configuration that could prevent service configuration from deploying, the agent install process raises an error.

   In this case, check the device log for error details (Navigate to **Devices / Managed Devices**, click the three dots in the device's **Actions** panel (right column), then in the **Agent** menu click the **Show Log** button (eyeball).) Manually remove conflicting configuration and start the agent installation process again.

   If you must complete the agent installation with configuration validation errors, you can disable pristine configuration validation. To do this, from **Devices > Managed Devices**, click **Advanced Settings** (top-right), select **Skip Pristine Configuration Validation**, then click **Update**.



For information about retaining pre-existing configuration when bringing devices under Apstra management, see "Device Configuration Lifecycle" on page 634.

> **NOTE**: On some platforms (Junos for example) you can configure rate-limiting for management traffic (SSH for example). When the Apstra server interacts directly with devices it can be more bursty than when it interacts with a user. Rate-limiting configurations that are used for hardening security can impact device management, and lead to deployment failures and other agent-related issues.

While the task is active you can view its progress at the bottom of the screen in the **Active Jobs** section. The job status changes from **Initialized** to **In Progress** to **Succeeded**.

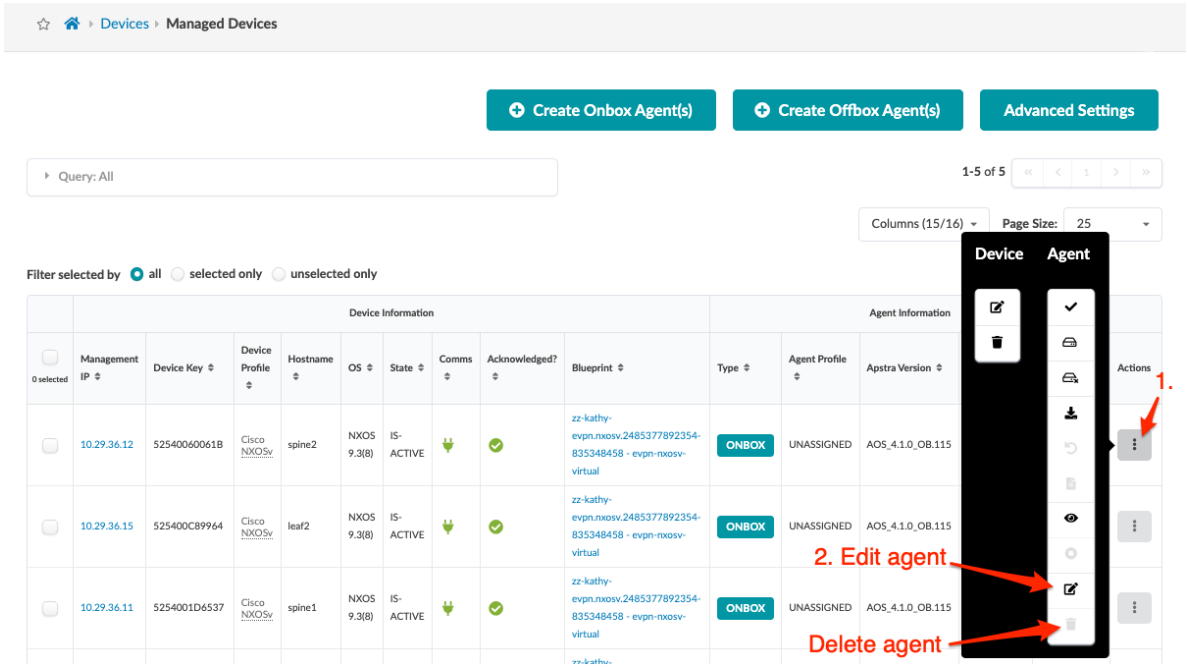
Update Agent

IN THIS SECTION

You can update one agent at a time to change its device address, agent profile, packages, and open-options, or you can update multiple agents simultaneously to assign an agent profile.

Update One Agent

1. From the left navigation menu, navigate to **Devices >Managed Devices** to go to devices and agents.
2. Click the three dots in the **Actions** column (right side) for the device that you want to update, then click the **Edit** button in the **Agent** menu.



The **Edit Onbox (or Offbox) System Agent(s)** dialog opens.

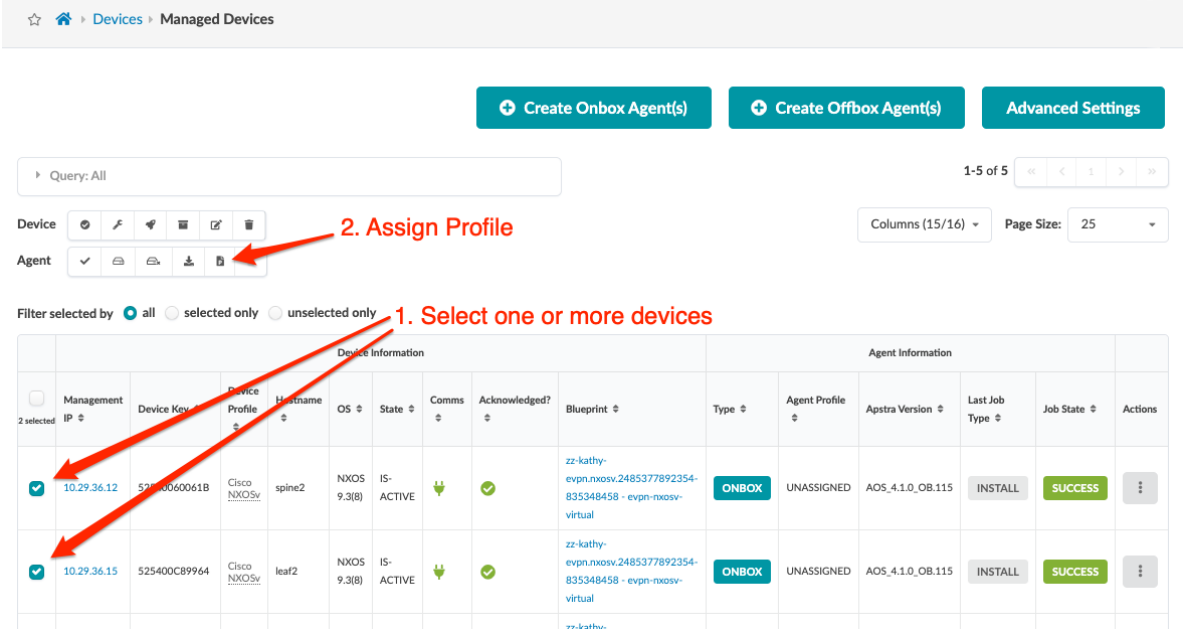
3. Make your changes (device address, platform, username, password, agent profile, packages, open-options, as applicable).

> **CAUTION**: Changing a user requires completely re-onboarding the device. Changing the password involves several steps that are not straightforward (changing the password on the device, device agents, and pristine config). If you need to change a password, we recommend contacting "Juniper Support" on page 1374.

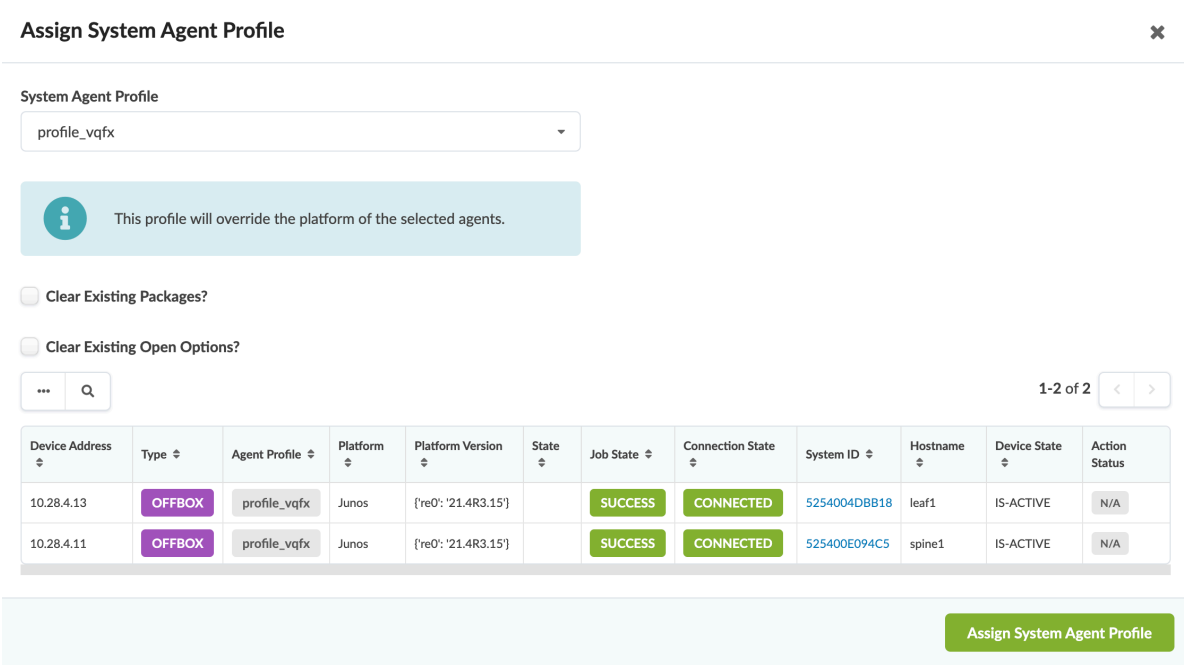
4. Click **Update** to update the agent and return to the table view.

Assign / Change Agent Profile on Multiple Agents

1. From the left navigation menu, navigate to **Devices >Managed Devices** and select one or more check boxes for the device(s) to update.



2. Click the **Assign Profile** button (in the menu that appears above the table after making a selection). The **Assign System Agent Profile** dialog opens.
3. Make your changes (agent profile, clear existing packages, clear open options).



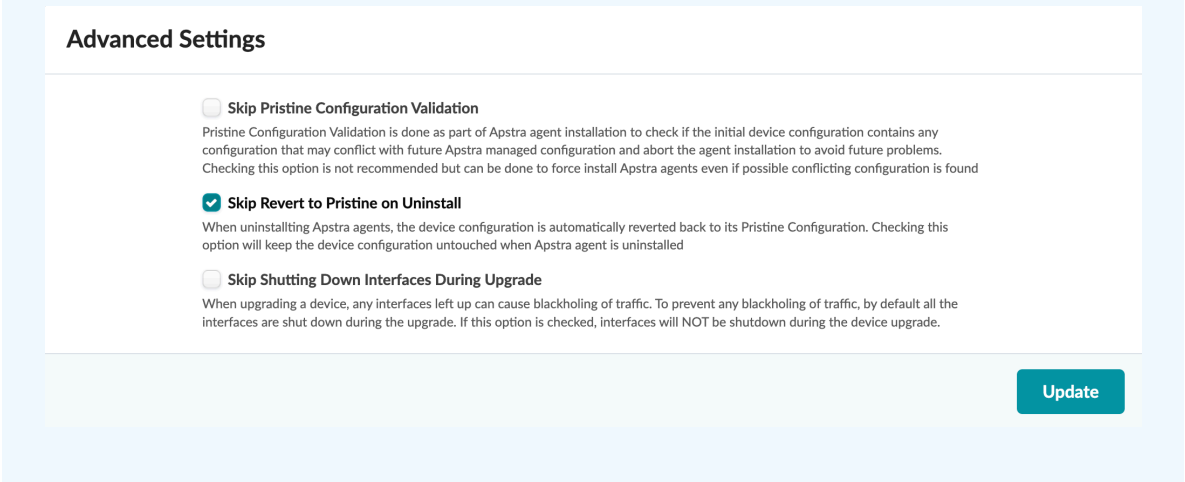
4. Click **Assign System Agent Profile** to save your changes and return to the table view.

Uninstall Agent

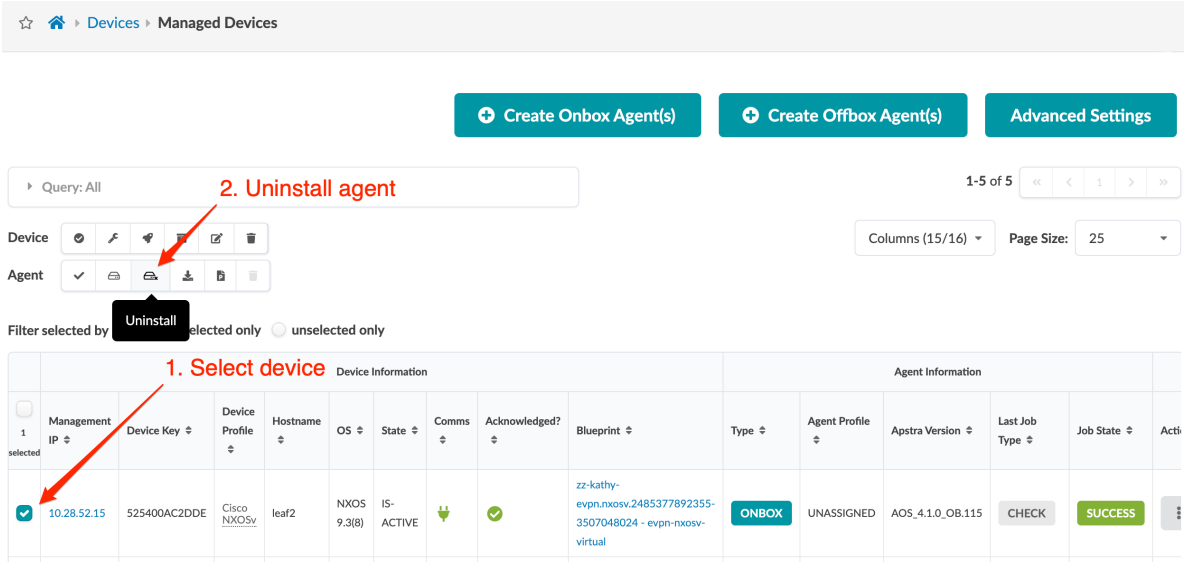
To remove a device from Apstra management, see "Remove (Decommission) Device from Managed Devices" on page 673 for the complete workflow. There are additional steps before and after uninstalling the agent as shown below.

1. From the left navigation menu, navigate to **Devices > Managed Devices** to go to the managed devices table view.

> **NOTE**: When you uninstall a device agent, the pristine configuration is restored on the device by default. If you want to retain existing configuration, click **Advanced Settings** and check the box to **Skip Revert to Pristine on Uninstall**.



2. Check the box(es) for the device(s), then in the **Agent** Actions panel that appears above the table, click the **Uninstall** button.



The **Uninstall selected elements** dialog appears.

3. Click **Uninstall selected elements**, then click **Close** to uninstall the agent and return to the Managed Devices table.

Delete Agent

You can delete an agent only if that agent has been uninstalled and is no longer running on a device.

NOTE: Several steps are involved in removing a device from Apstra management, such as unassigning it from a blueprint, setting the admin state, uninstalling and deleting the agent, and deleting the device from the Managed Devices table. See "Remove Device" on page 673 for details.

1. From the left navigation menu, navigate to **Devices > Managed Devices**, select the device(s) to delete, then click the **Delete** button in the **Agent** section.
2. Click **Delete** to delete the agent(s) and return to the table view.

Agent Profiles

IN THIS SECTION

Agent Profiles Introduction

Agent profiles enable the logical link between device credentials, a device configuration key-value store, and a selection of user-uploaded packages. With agent profiles, you can configure parameters for a certain class of devices that exist in the network and edit their device agent settings as a group. Agent profiles include the following details:

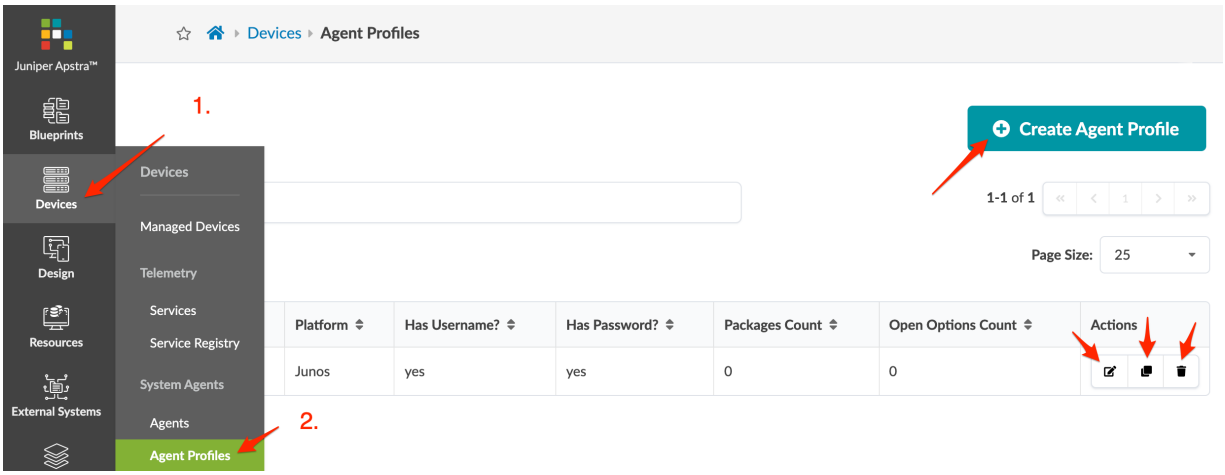
Table 5: Agent Profile Parameters

Name	Description
Name	To identify the device agent profile
Platform	OS family (EOS, Junos, NX-OS)

Table 5: Agent Profile Parameters *(Continued)*

Name	Description
Username / Password	Admin/root username and password on the device
Open Options (offbox only)	Passes configured parameters to offbox agents. For example, to use HTTPS as the API connection from offbox agents to devices, use the key-value pair: proto-https - port-443. You can override the following default values with open options: • commit_timeout - 60 (integer: seconds) • telemetry_timeout - 100 (integer: seconds) • probe_timeout: 5 (integer: seconds) • log_config_diff - True (boolean)
Packages	Admin-provided software packages stored on the Apstra server that you can apply to each device agent that you create using the profile.

From the left navigation menu, navigate to **Devices > System Agents > Agent Profiles** to go to the agent profile table view. You can create, clone, edit, and delete agent profiles.



Create Agent Profile

Before creating an agent profile, upload any "packages" on page 700 that are to be included in the agent profile.

1. From the left navigation menu, navigate to **Devices > System Agents > Agent Profiles** and click **Create Agent Profile**.
2. Enter a unique agent profile name.

3. Select the platform from the drop-down list (optional).
4. Set a username and password (optional).
5. Add open options (optional).
6. Select package(s) (optional).
7. Click **Create** to create the agent profile and return to the table view.

Assign Agent Profile

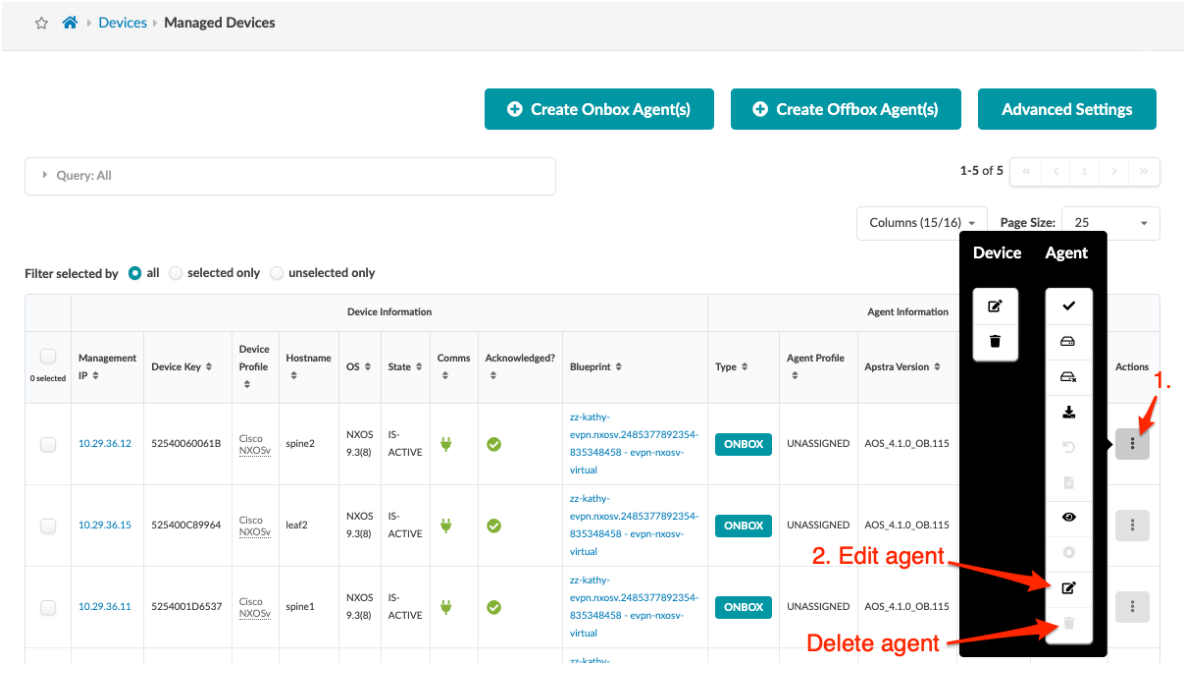
SUMMARY

If you're using agent profiles, you can assign one to an agent when you create the agent, or you can assign it later. If you assign it later, you can assign it to one agent at a time, or to multiple agents simultaneously, as shown below.

IN THIS SECTION

Assign Agent Profile to One Agent

1. From the left navigation menu, navigate to **Devices >Managed Devices** to go to devices and agents information.
2. Click the three dots in the **Actions** column (right side) for the agent you want to update, then click the **Edit** button in the **Agent** menu.

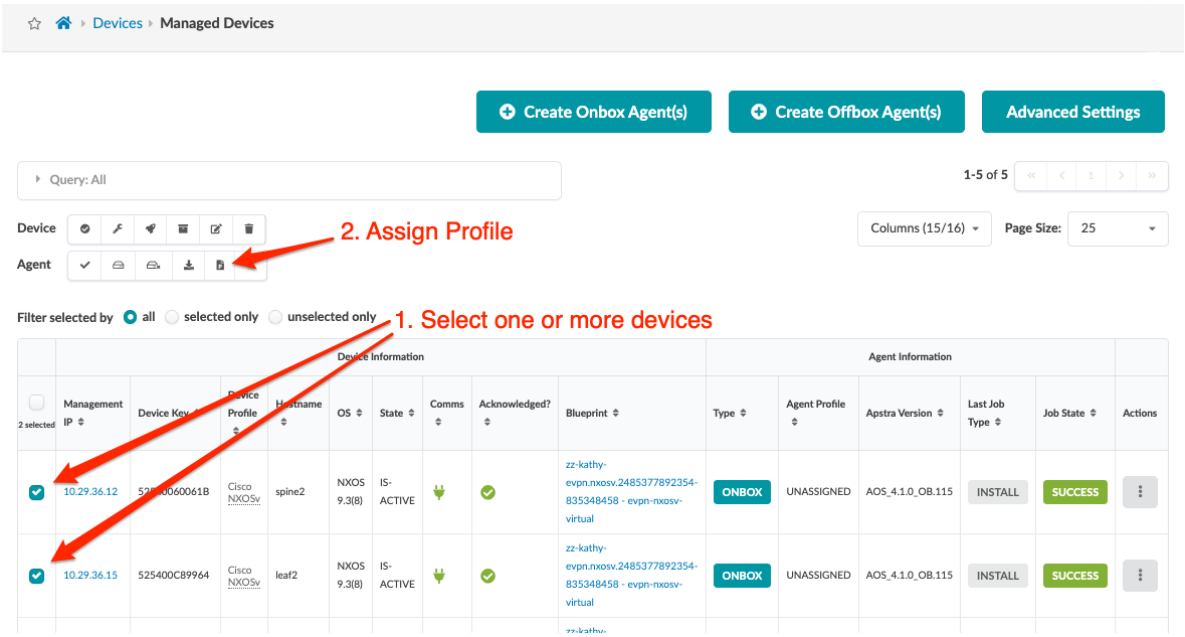


3. Select the agent profile from the drop-down list.

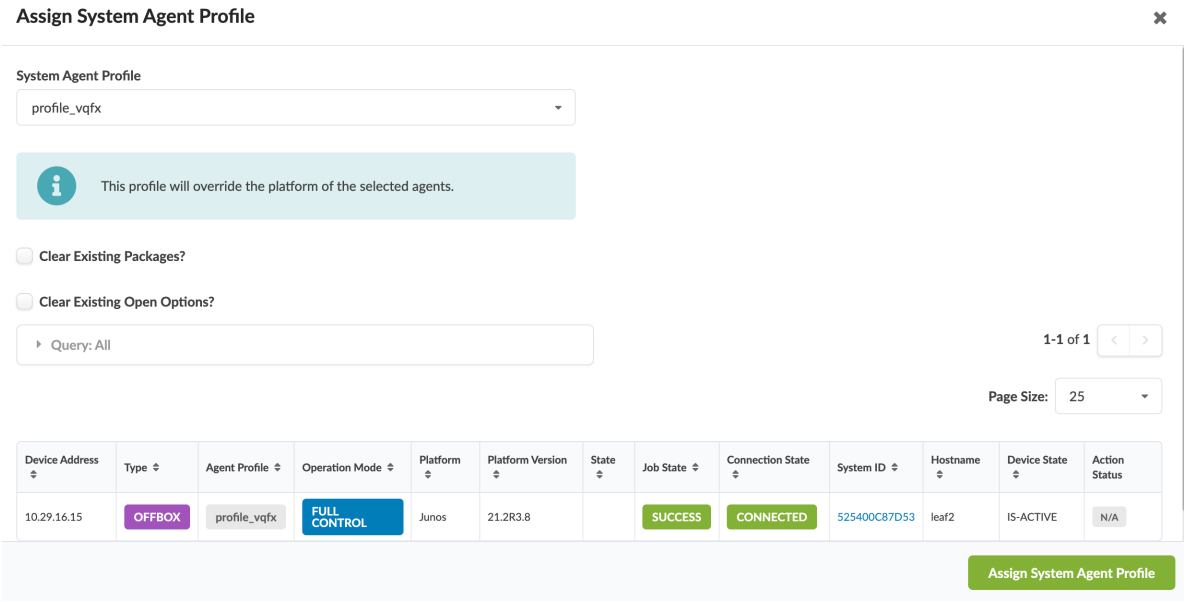
4. Click **Update** to update the agent and return to the table view.

Assign Agent Profile to Multiple Agents

1. From the left navigation menu, navigate to **Devices >Managed Devices** and select one or more check boxes for the device(s) to edit.



2. Click the **Assign Profile** button (in the menu that appears above the table after making a selection).



3. Select a system agent profile from the drop-down list.
4. Click **Assign System Agent Profile** to save your changes and return to the table view.

Edit Agent Profile

1. Either from the table view (Devices > System Agents > Agent Profiles) or the details view, click the **Edit** button for the profile to edit.
2. Make your changes.
3. Click **Update** to update the profile and return to the table view.

Delete Agent Profile

1. Either from the table view (Devices > System Agents > Agent Profiles) or the details view, click the **Delete** button for the profile to delete.
2. Click **Delete** to delete the profile and return to the table view.

Packages (Devices)

IN THIS SECTION

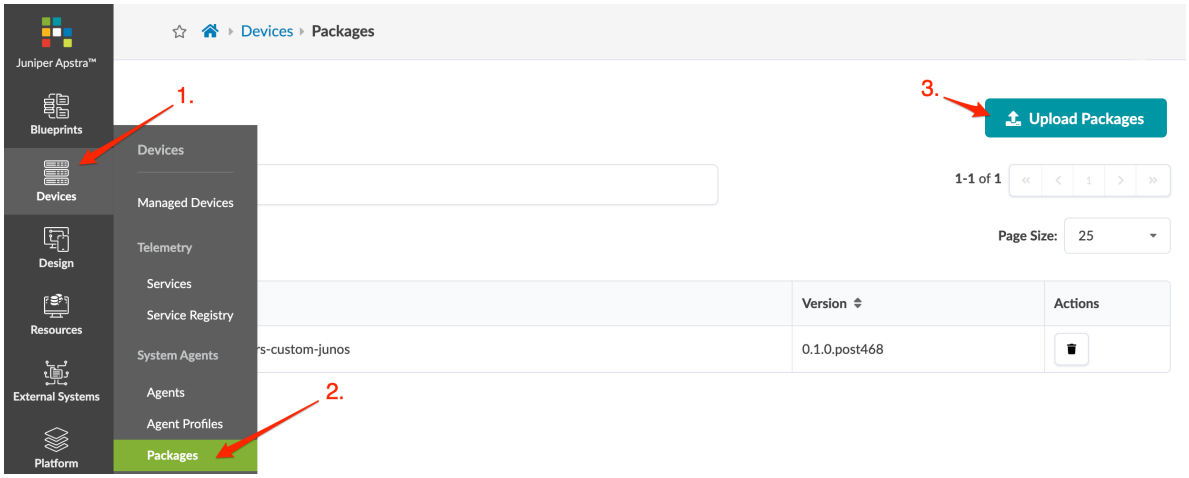
Packages Overview

You can extend Apstra capabilities by adding support for network operating systems (NOS), new telemetry collectors, third party software, and more. You upload packages (sometimes referred to as plugins) to the Apstra server, then include them in device agents and "agent profiles" on page 696. Valid package types include .egg, .whl (Python wheel package) and .gz. One package can include one or more collectors for one or more OS platforms.

Upload Packages

1. Download the required package(s) from Juniper Support Downloads.

2. From the left navigation menu, navigate to **Devices > System Agents > Packages** and click **Upload Packages**.



3. For each package to upload, either click **Choose File** and navigate to the downloaded file, or drag and drop the file into the dialog window.
4. Click **Upload**, then close the dialog to return to the table view.

CHAPTER 21

Pristine Config

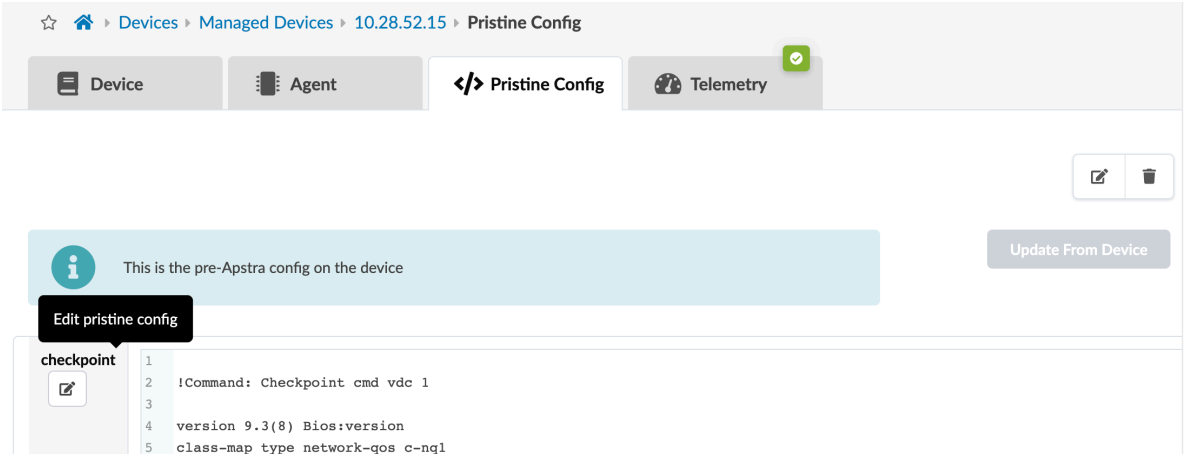
IN THIS CHAPTER

Edit Pristine Config

Modifying pristine config is a local operation, and does not lead to a change to the running device configuration. Changes are applied on the next full config push. If you want to apply persistent changes to a configuration, use "configlets" on page 908.

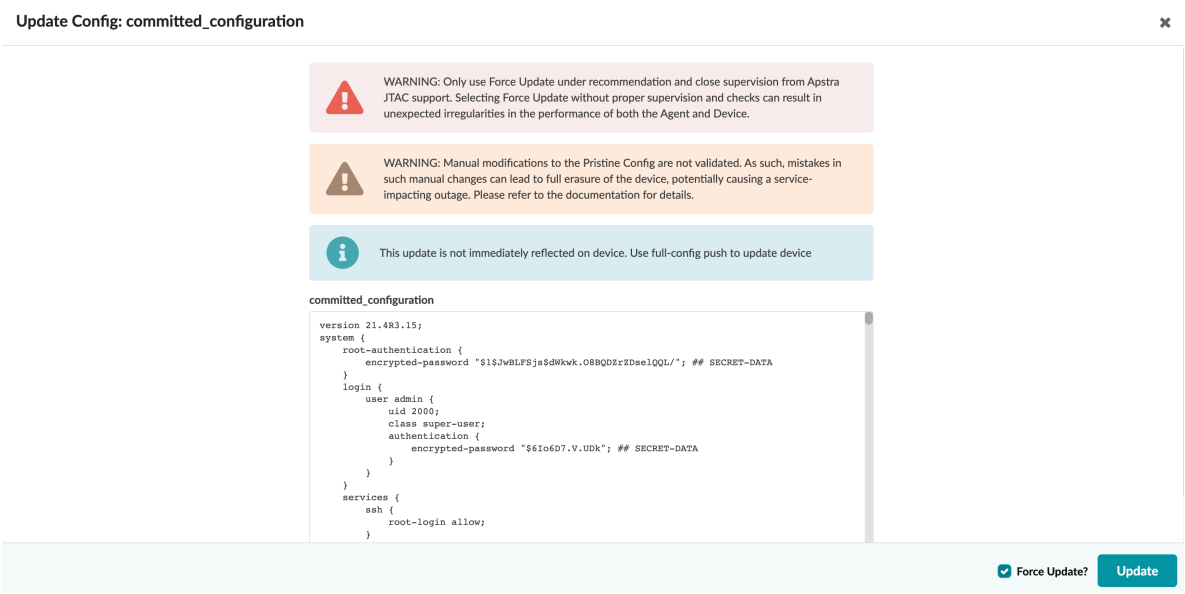
CAUTION: Manual modifications to the Pristine Config are not validated. Mistakes can lead to full erasure of the device, potentially causing a service-impacting outage. Never modify the pristine config directly unless there is no alternative. For assistance, contact "Juniper Support" on page 1374.

1. From the left navigation menu, navigate to **Devices > Managed Devices** and click the **Management IP** of the device to edit.
2. Click the **Pristine Config** tab (top-left), then click the **Edit pristine config** button (under **checkpoint** on the left).



3. Make your changes to the configuration. (For information about what should and should not be included in pristine configurations, see "Create Onbox Agent" on page 680 and "Create Offbox Agent" on page 685, as applicable.)

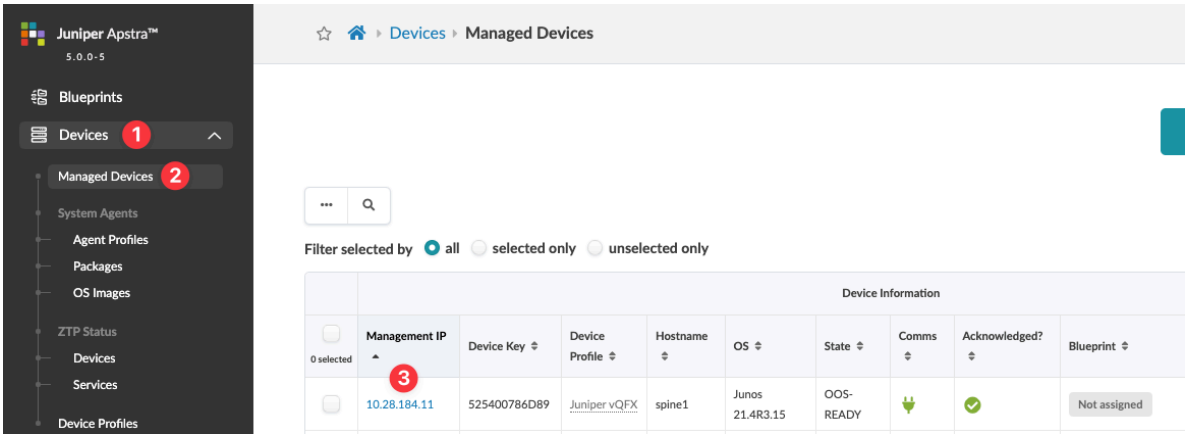
4. If the device is deployed, and you absolutely need to change pristine config, you can force the update without undeploying first. Never use this option unless you are otherwise directed by the Juniper support team. This update has the power to impact your entire network. To proceed, check the **Force Update** check box.



5. Click **Update** to apply the changes.

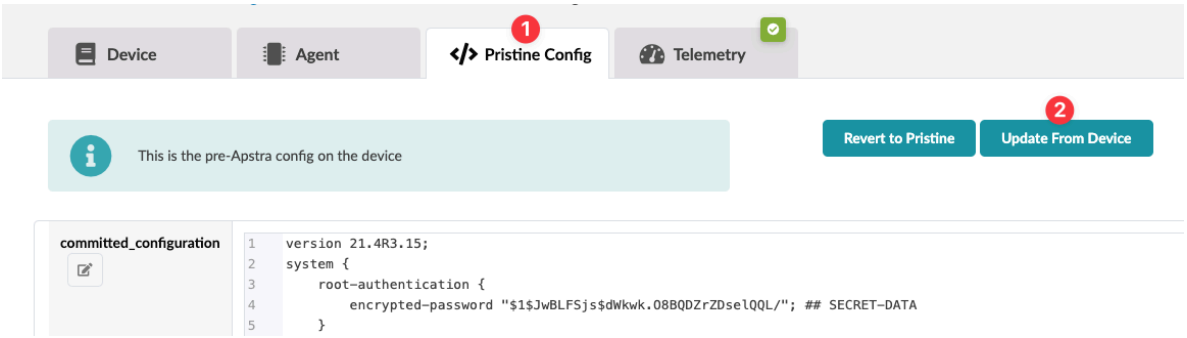
Update Pristine Config from Device

1. From the blueprint, "unassign the device" on page 85 (The deploy mode is automatically changed from Deploy to Not Set, as of Apstra version 5.0.0.) Make sure the device is in the out of service state (OOS-READY or OOS-MAINT).
2. Make any necessary changes to the running device configuration via CLI.
3. From the left navigation menu in the Apstra GUI, navigate to **Devices > Managed Devices** and click the **Management IP** of the device to update.



The **Device** details page for the selection appears.

4. Click the **Pristine Config** tab (top-left), then click **Update From Device** (top-right).



The **Collect Pristine Config from Device** dialog opens.

5. Click **Collect** to update Pristine Config from the device and return to the **Pristine Config** tab.

Verify the Pristine Config. You have copied the running config of the device in the out of service state, which should be **Discovery 1** config. It may include additional configuration such as interface "speed" commands. You can edit Pristine Config again and delete the additional configuration manually. Contact "Juniper Support" on page 1374 for assistance as needed.

CHAPTER 22

Telemetry

IN THIS CHAPTER

Device Telemetry Services

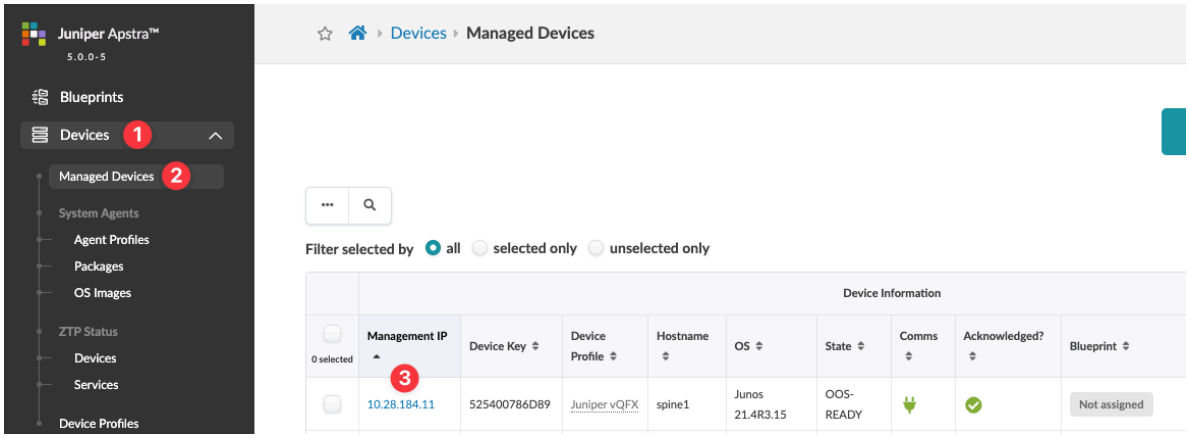
SUMMARY

Apstra ships with a set of built-in telemetry services that are automatically collected when a device is connected. You can also develop your own telemetry services.

IN THIS SECTION

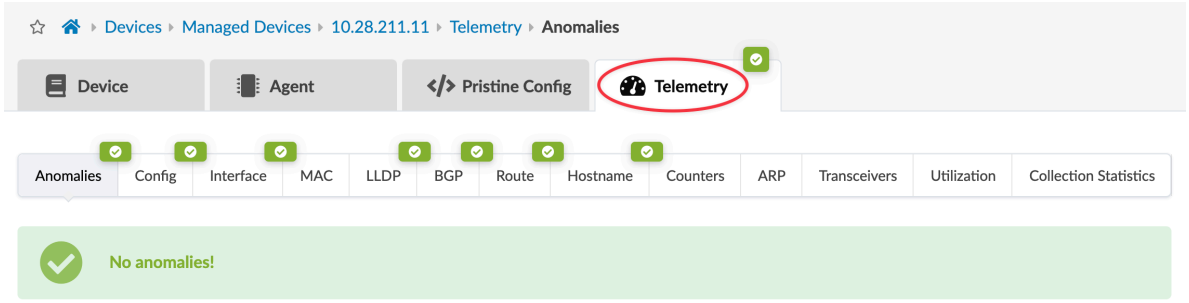
View Device Telemetry

1. From the left navigation menu in the Apstra GUI, navigate to **Devices > Managed Devices** and click the **Management IP** of the device to view.



A page opens that includes information about the device, agent, pristine configuration, and telemetry.

2. Click the **Telemetry** tab to open the **Telemetry** details page.



The **Telemetry** page includes tabs for any telemetry anomalies, the various telemetry services, and collection statistics.

3. Click each tab to see what's included; and check out the sections below for additional details.

Anomalies

If there are any anomalies (indicated in red) from these telemetry services they are aggregated under the **Anomalies** tab. You can look here to see an overview of all telemetry anomalies for the device (or under the individual service tabs for specific service anomalies).

Telemetry Services

Table 6: Device Telemetry Services

Service	Description
ARP	ARP telemetry shows an ARP table. You can query this information via API. Anomalies are not generated.

Table 6: Device Telemetry Services *(Continued)*

Service	Description
BGP	BGP telemetry shows role(s), VRF name, address family, source and destination information, expected and actual states, intent status, last fetched/modified, and BGP peer state.
Config	This is the running config. Devices with deviations between the rendered discovered/service config and the actual config are flagged with a config deviation error. When configuration changes are made outside of Apstra management, alarms are generated immediately. The risk with a configuration deviation is that it is possible for Apstra to overwrite the deviated configuration with a configuration re-write. The correct way to deal with a config deviation alarm is to understand the configuration change being made, and consider setting it up as a "configlet" on page 908 instead.
Counters	Counter telemetry provides information about interface in/out packets, interface errors, statistics, and so on. This feature is consumed by other advanced downstream features like telemetry streaming. No anomalies are generated.
Hostname	When you assign a device with deploy mode **Ready** to a blueprint, the device enters the **Ready** stage (previously known as Discovery 2). Hostname telemetry is collected that validates the device hostname against intent. Mismatches result in anomalies.
Interface	When you assign a device with deploy mode **Ready** to a blueprint, the device enters the **Ready** stage (previously known as Discovery 2). Interface telemetry is collected that compares intent with the up/down state of physical interfaces. It does not include LLDP, LAG or any other attachment information.

Table 6: Device Telemetry Services *(Continued)*

Service	Description
LAG	LAG telemetry shows the health of all the LACP bonds facing servers and between MLAG switches.
LLDP (Cabling)	When you assign a device with deploy mode **Ready** to a blueprint, the device enters the **Ready** stage (previously known as Discovery 2). Every node is part of intent. On each link, there are expected neighbor hostnames, interfaces and connections. Physical cabling and links must match the specified intent. Any deviations result in anomalies that you must correct by either recabling to match the blueprint or by modifying the blueprint to match cabling already in place.
MAC	MAC Address-table telemetry shows which MAC addresses appear on which interfaces, and which VLANs.
MLAG	MLAG telemetry tracks the health status of the MLAG domain itself - the control-protocols required between two leaf switches communicating with each other properly for the MLAG domain state. Implementation detail differences exist between multiple vendors, but the intent is the same -the switches should be healthy among each other. MLAG telemetry is only available for L2 blueprints that have at least one virtual network assigned in an MLAG pair. If an MLAG-attached server is not fully connected, the state changes from 'active_full' to 'active_partial'. **NOTE**: Cisco MLAG (VPC) commands cannot derive the status of the LAG on the VPC peer switch. Accordingly, the state *dual-active* cannot actually gather the command. This is a limitation from Cisco.

Table 6: Device Telemetry Services *(Continued)*

Service	Description
Route	Routing telemetry analyzes the routing table on every managed spine and leaf. Since the entire IP fabric is managed, you can derive and predict full IP table information from the network topology. Deviations in the network routing telemetry (for example, a missing next-hop IP address for a default route) cause an alarm.
Transceivers	Transceiver telemetry gives the network operator statistics on optical interfaces, showing DOM statistics, light levels, lossy interfaces, and other optical statistics. No anomalies are generated.
Utilization (Onbox agents only)	Utilization telemetry allows the network operator to view some vital statistics on the device - CPU and Memory utilization. No anomalies are generated. Utilization telemetry is not available on devices using offbox agents (Junos for example). Therefore, the utilization tab contains the error **Network Device not found**.
You can collect additional telemetry	See the "Extensible Telemetry Guide" on page 1437 for details.

Collection Statistics

Click the **Collection Statistics** tab to see statistics details. See the table below for more information.

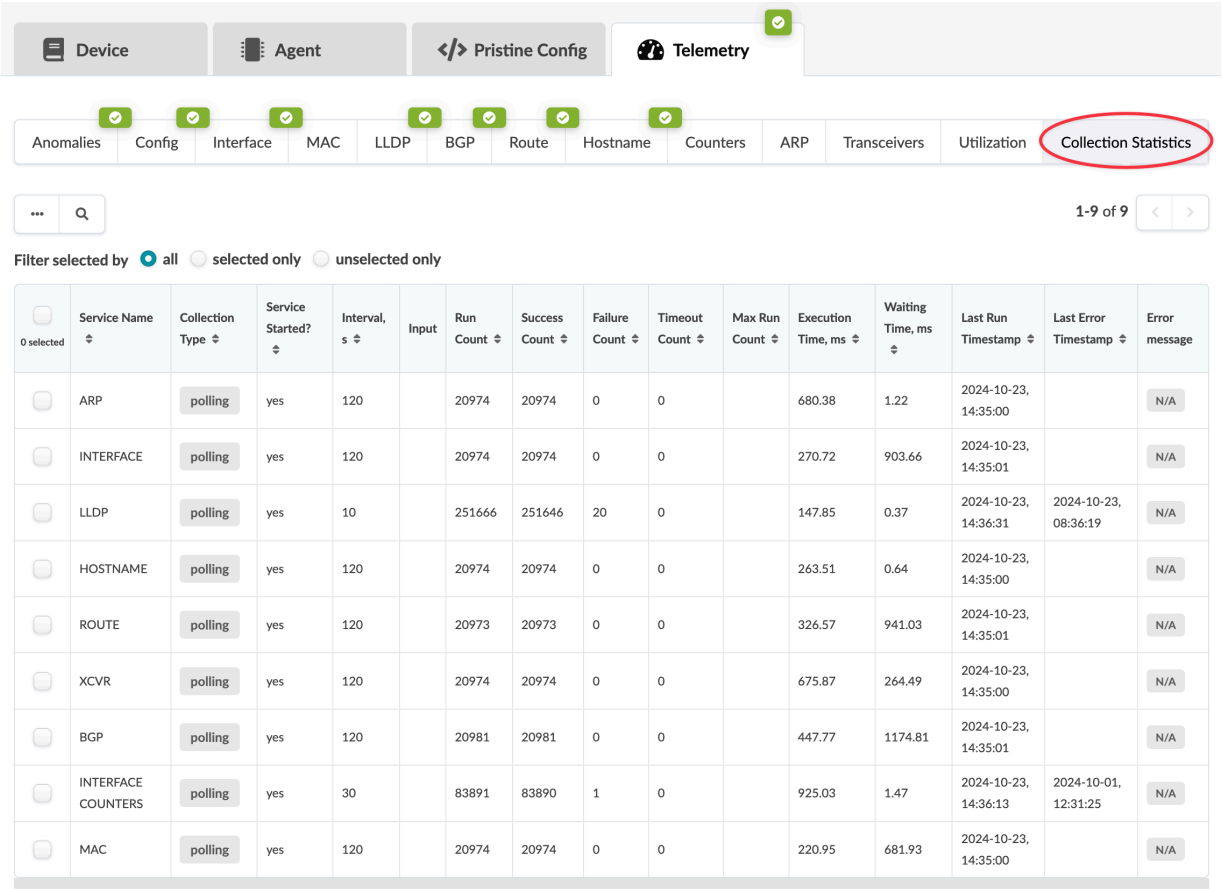


Table 7: Device Collection Statistics

Collection Statistics	Description
Service Name	The name of the service
Collection Type	polling or gRPC
Service Started?	Has the service started?
Interval	How frequently the service is configured to run on the device (in seconds)
Input	The input that is provided to the service for its processing
Run Count	The number of times the collector is scheduled to run
Success Count	The number of times the collector successfully executed

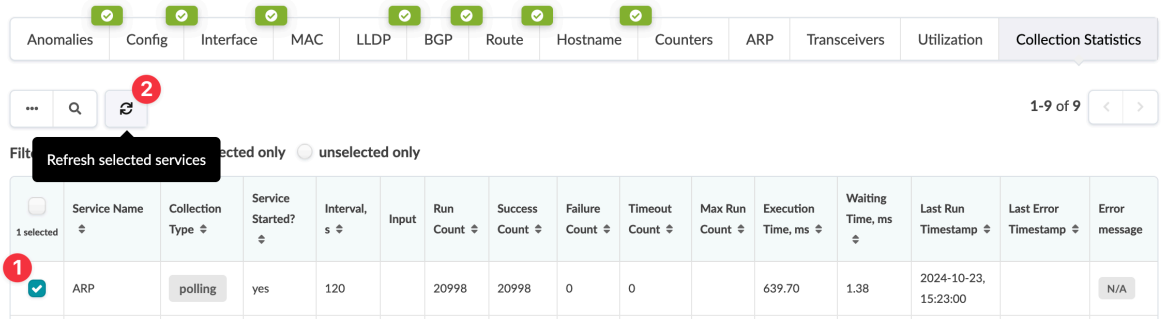
Table 7: Device Collection Statistics *(Continued)*

Collection Statistics	Description
Failure Count	The number of times the collector failed execution
Timeout Count	The number of times the collector timed out
Max Run Count	User-specified maximum number of times for the collector to run
Execution Time	The time it took for collection during the last iteration (in milliseconds)
Waiting Time	A device runs multiple collectors. If some collectors monopolize CPU, other collector executions are deferred. Waiting time is the amount of time that the collector was deferred (in milliseconds).
Last Run Timestamp	Timestamp at which the collector was scheduled to run
Last Error Timestamp	Timestamp at which the collector last reported an error
Error message	Any error message from the last collector iteration.

Refresh Telemetry Service

You can refresh one or more services on a device to get the latest telemetry details.

1. From the **Collection Statistics** tab, select the check box(es) for one or more services, then click the **Refresh selected services** button that appears above the table.



   The **Refresh Telemetry for Selected Services?** dialog opens.

2. Click **Confirm** to refresh telemetry and return to the **Collection Statistics** page.

CHAPTER 23

Apstra ZTP

IN THIS CHAPTER

What is Apstra ZTP

IN THIS SECTION

Overview

Apstra ZTP is a Zero-Touch-Provisioning server for data center infrastructure systems. From an Apstra perspective, it's a process that automatically takes a device from initial boot to a point where it's managed by Apstra. Apstra ZTP takes care of any underlying NOS requirements.

The ZTP process includes the following activities:

1. Generic DHCP (if using DHCP)
   - The device requests an IP address via DHCP.
   - The device receives the assigned IP address and a pointer to the OS installation image.
2. Initialize Device
   - Download the ZTP script, using TFTP.
   - Execute the downloaded script to prepare it to be managed. This includes verifying that the device is running a supported OS; if It's not, it upgrades or downgrades the version, as needed.
   - Set the device admin/root password.
   - Create a device user for the device system agent.
3. Install Device System Agent
   - The ZTP script makes an API call to install a device system agent on the device for onbox agents, or on the Apstra server for offbox agents.

Apstra ZTP runs as an Ubuntu 22.04.3 LTS server running MySQL, DHCP, HTTP, and TFTP servers.

Apstra provides the Apstra ZTP VM image (`.ova`, `.qcow2.gz`, `.vhdx.gz`). You can use the Apstra-provided device provisioning scripts as part of the existing ZTP/DHCP process to automatically install agents on devices as part of the boot process.

The TFTP and nginx HTTP servers don't require configuration. Both servers serve files out of the `/containers_data/tftp` directory.

You'll need to configure the `dhcp.conf` file and the `ztp.json` files during ZTP setup. You can use the Apstra ZTP GUI, and, as of Apstra version 5.0.0, you can also use "AZTP REST API" on page 779 to configure these files, among other tasks.

Apstra ZTP provides a method for automating switch initialization and customization. A useful feature during switch initialization is the ability for our script to make custom configs in the switches prior to their use in a network.

Resource Requirements for Apstra ZTP Server

Table 8: Apstra ZTP Server VM Minimum Resource Requirements

Resource	Setting
Guest OS Type	Ubuntu 22.04.3 LTS 64-bit
Memory	2 GB
CPU	1 vCPU
Disk Storage	64 GB
Network	At least 1 network adapter, initially configured for DHCP

Table 9: Apstra ZTP Network Requirements

Source	Destination	Ports	Role
Device Agents	DHCP server (renewals) and Broadcast (requests)	udp/67 -> udp/68	DHCP Client
Device Agents	Apstra ZTP	any -> tcp/80 (http) any -> tcp/443 (https)	Bootstrap and API scripts
Arista, Cisco, and Juniper Agents	Apstra ZTP	any -> udp/69	TFTP for POAP and ZTP
Apstra ZTP	Apstra server (controller)	any -> tcp/443 (https)	Device System Agent Installer API

Table 9: Apstra ZTP Network Requirements *(Continued)*

Source	Destination	Ports	Role
User	Apstra server (controller)	any -> tcp/443 (https)	Apstra ZTP GUI interface

Apstra Server Required Communication Ports

The Apstra ZTP server and device agents also require connectivity to the Apstra server (controller). For more information, refer to Required Communication Ports in the Juniper Apstra Installation and Upgrade Guide.

Installing and Setting up Apstra ZTP

Follow the links below for detailed Apstra ZTP installation and configuration instructions.

1. "Create a user profile for communicating with Apstra ZTP" on page 717.
2. "Download and deploy the Apstra ZTP server VM" on page 719.

   NOTE: The VM image for Apstra ZTP is a separate VM image from the Apstra server VM image.

3. "Configure the static management IP address for the Apstra ZTP server" on page 720.
4. "Replace the SSL certificate for the Apstra ZTP server GUI" on page 721.
5. "Configure credentials for the Apstra ZTP Server GUI" on page 729.
6. "Configure Apstra Server Details for communicating with Apstra ZTP" on page 730.
7. "Create vendor-specific custom configuration" on page 724, as needed.
8. "Configure Apstra server connection details" on page 730.
9. "Configure the DHCP server for Apstra ZTP" on page 732.
10. "Configure ztp.json" on page 754 for Apstra ZTP. See the "ztp.json Keys" on page 740 page for key details.

Onboarding Devices with Apstra ZTP

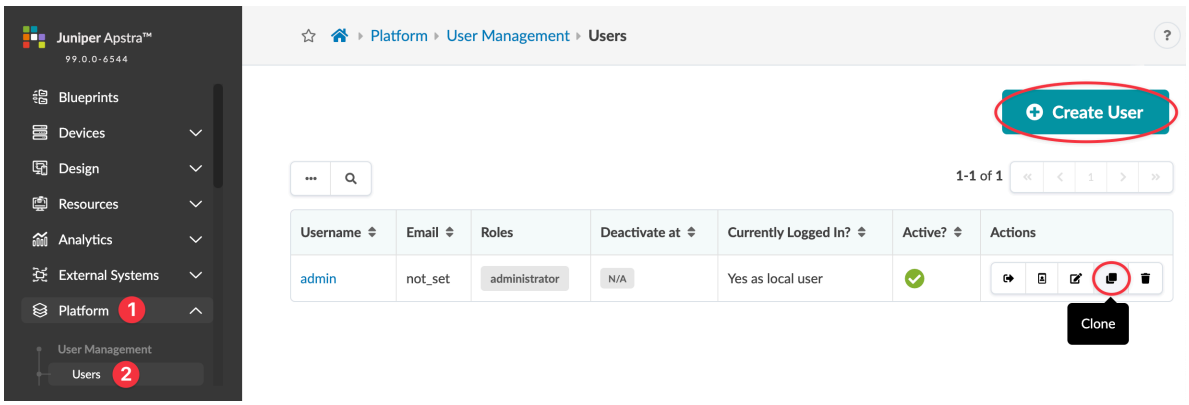
Once Apstra ZTP is set up, you can quickly "onboard devices" on page 770. Make sure device configuration is set to factory default, then boot up your device. Apstra ZTP takes care of the rest up to the point where a device is ready to be *acknowledged*. When you acknowledge a device it's under Apstra management and you can assign it to any blueprint in your Apstra environment.

You can also "check ZTP status" on page 776 of devices and services from the Apstra server GUI.

Create User Profile for Communicating with ZTP Server

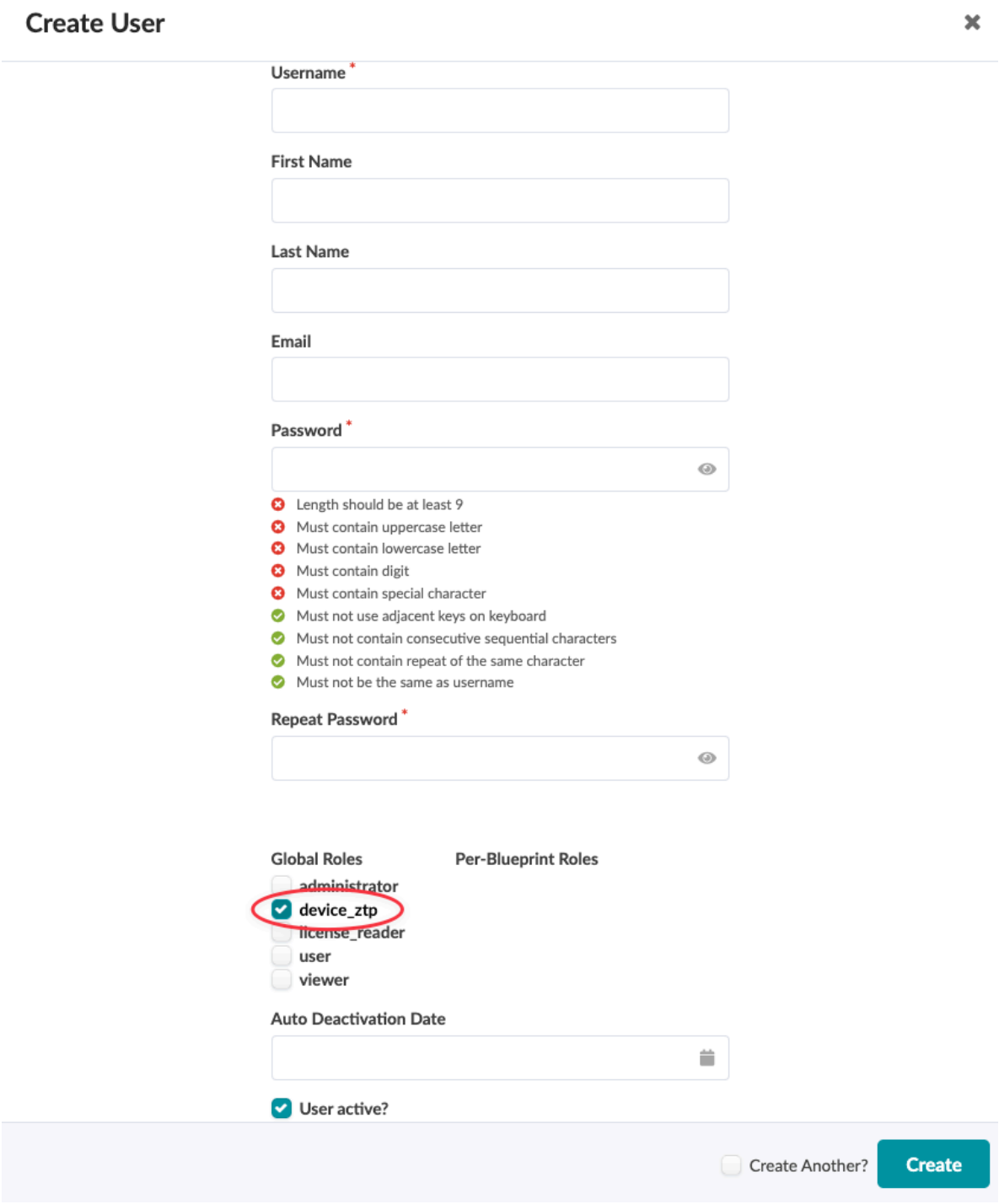
You can use any configured Apstra server GUI user that has API write access (such as admin), but we recommend that you create a designated user that's assigned only the predefined **device_ztp** role. The **device_ztp** role allows users with that role to make API calls to the controller to request device system agent installation.

1. From the left navigation menu, navigate to **Platform > User Management > Users** and click **Create User**.



   The **Create User** dialog opens.

2. Enter a username (such as **ztp**) for the user who will be communicating with the ZTP server, enter a password that meets password complexity requirements, then re-enter the password. (You can "change requirements" on page 1286 from **Platform > Security > Password Complexity Parameters**.)



3. Select the global role **device_ztp**.
4. To deactivate the user automatically on a specific date and time, select the date and time from the **Auto Deactivation Date** calendar. You can also immediately deactivate and activate the user by deselecting or selecting the **User active?** check box. (This feature is new in Apstra version 5.0.0.)
5. Click **Create** to create the user profile and return to the table view.

When you work with Apstra ZTP from the Apstra GUI, log in as this user.

Download and Deploy Apstra ZTP Server VM

SUMMARY

Make sure the server that you'll be using for Apstra ZTP meets minimum resource requirements, then download and deploy the Apstra ZTP server VM.

IN THIS SECTION

Download and Deploy VM

Check that the VM you'll be using for Apstra ZTP meets resource requirements.

1. Apstra ZTP software is delivered as a standalone VM. It's available for each Apstra version and supported hypervisor. As a registered support user, download the appropriate **Apstra ZTP VM** image from Juniper Support Downloads **Application Tools** section.

   Table 10: Apstra ZTP Images

Apstra ZTP Image for VMware ESXi	apstra-ztp-5.0.*-<build-version>.ova (example: apstra-ztp-4.2.0-34.ova)
Apstra ZTP Image for Microsoft Hyper-V	apstra-ztp-5.0.*-<build-version>.vhdx.gz (example: apstra-ztp-4.2.0-34.vhdx.gz)
Apstra ZTP Image for Linux KVM QCOW2	apstra-ztp-5.0.*-<build-version>.qcow2.gz (example: `apstra-ztp-4.2.0-34.qcow2.gz`)

2. Validate the downloaded file against the SHA512/MD5 checksums provided.
3. Deploy the VM with the appropriate resources. When you boot the VM for the first time an interactive bash script runs to assist you with configuration (similar to the one for the Apstra server) (new in Apstra 5.0.0). From there you can change your CLI credentials, configure the static management IP address, and start ZTP services (containers).

NGINX (HTTP), TFTP, Status, DHCPd, and MySQL Docker containers are enabled and run, by default.

```
admin@apstra-ztp:~$ docker ps
CONTAINER ID   IMAGE                    COMMAND                  CREATED        STATUS
PORTS                                                                    NAMES
5f6609074deb   apstra/nginx:4.2.0-34    "sh /init.sh"            29 hours ago   Up 29 hours
0.0.0.0:80->80/tcp, :::80->80/tcp, 0.0.0.0:443->443/tcp, :::443->443/tcp   nginx
3f0dbe2be17b   apstra/tftp:4.2.0-34     "sh /init.sh"            29 hours ago   Up 29 hours
0.0.0.0:69->69/udp, :::69->69/udp                                        tftp
1e05ab10a552   apstra/status:4.2.0-34   "sh /init.sh"            29 hours ago   Up 29 hours
8080/tcp                                                                 status
cd7aa8ad372b   apstra/dhcpd:4.2.0-34    "sh /init.sh"            29 hours ago   Up 28
hours                                                                       dhcpd
12e35bc71b20   mysql:8.0.33             "docker-entrypoint.s…"   29 hours ago   Up 29 hours
3306/tcp, 33060/tcp                                                      db
admin@apstra-ztp:~$
```

4. If you don't want to use the Apstra ZTP DHCP server, stop the dhcpd container and disable it, as shown below.

```
admin@apstra-ztp:~$ docker stop dhcpd
dhcpd
admin@apstra-ztp:~$ docker update --restart=no dhcpd
dhcpd
admin@apstra-ztp:~$
```

Next Step: Configure the Static Management IP Address for the Apstra ZTP server.

Configure Static Management IP Address for Apstra ZTP Server

The Apstra ZTP server attempts to assign an IP address for its eth0 interface via DHCP, by default. If you're using the Apstra ZTP server as a DHCP server, you must set a management IP address. If you didn't set the IP address on first boot using the interactive bash script that automatically runs, then you can follow the steps below to do it now.

1. SSH into the Apstra ZTP server as **admin** (`ssh admin@<apstra-ztp-server-ip>` where `<apstra-ztp-server-ip>` is the IP address of the Apstra ZTP server.)

2. Edit the `/etc/netplan/01-netcfg.yaml` file to configure the static management IP address. See example below. (For more information about using netplan, see https://netplan.io/examples)

```
admin@apstra-ztp:~$ sudo vi /etc/netplan/01-netcfg.yaml
[sudo] password for admin:

# This file describes the network interfaces available on your system
# For more information, see netplan(5).
network:
  version: 2
  renderer: networkd
  ethernets:
    eth0:
      dhcp4: no
      addresses: [192.168.59.4/24]
      routes:
      - to: default
        via: 192.168.59.1
      nameservers:
          search: [example.com, example.net]
          addresses: [69.16.169.11, 69.16.170.11]
```

3. Apply the change with *one* of the following methods:
   - Reboot the Apstra ZTP server with the command `sudo reboot`.
   - Run the command `sudo netplan apply`.

Next Step: Replace the SSL Certificate for the Apstra ZTP Server GUI.

Replace SSL Certificate for Apstra ZTP Server GUI

For security, we recommend that you replace the Apstra ZTP default self-signed SSL certificate with one from your own certificate authority. Web server certificate management is the responsibility of the end user. Juniper support is best effort only.

When you boot up the Apstra ZTP server for the first time, a unique self-signed certificate and key are automatically generated and stored on the Apstra ZTP NGINX container. The certificate is used for encrypting the Apstra ZTP server. We recommend replacing the default SSL certificate.

1. Create a new OpenSSL private key with the built-in openssl command.

```
admin@apstra-ztp:~$ sudo -s
root@apstra-ztp:/home/admin# cd /containers_data/nginx
root@apstra-ztp:/containers_data/nginx# openssl genrsa -out nginx.key 2048
root@apstra-ztp:/containers_data/nginx
```

2. Create a certificate signing request. If you want to create a signed SSL certificate with a Subjective Alternative Name (SAN) for your Apstra ZTP server HTTPS service, you must manually create an OpenSSL template. For details, see Juniper Support Knowledge Base article KB37299.

```
root@apstra-ztp:/containers_data/nginx# openssl req -new -sha256 -key nginx.key -out nginx.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:California
Locality Name (eg, city) []:Sunnyvale
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Juniper Networks
Organizational Unit Name (eg, section) []:Apstra
Common Name (e.g. server FQDN or YOUR name) []:apstra-ztp.apstra.com
Email Address []:support@juniper.net

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
root@apstra-ztp:/containers_data/nginx#
```

3. Submit your Certificate Signing Request (nginx.csr) to your Certificate Authority (CA). The required steps are outside the scope of this document; CA instructions differ per implementation. Any valid SSL certificate will work. The example below is for self-signing the certificate.

```
root@apstra-ztp:/containers_data/nginx# openssl req -x509 -sha256 -days 3650 -key nginx.key -
in nginx.csr -out nginx.crt
```

```
Warning: No -copy_extensions given; ignoring any extensions in the request
root@apstra-ztp:/containers_data/nginx#
```

4. Verify that the SSL certificates match: private key, public key, and CSR.

```
root@apstra-ztp:/containers_data/nginx# openssl rsa -noout -modulus -in nginx.key | openssl
md5
MD5(stdin)= 9246ee21e992d34ce76c5b40b1ef777d
root@apstra-ztp:/containers_data/nginx# openssl req -noout -modulus -in nginx.csr | openssl
md5
MD5(stdin)= 9246ee21e992d34ce76c5b40b1ef777d
root@apstra-ztp:/containers_data/nginx# openssl x509 -noout -modulus -in nginx.crt | openssl
md5
MD5(stdin)= 9246ee21e992d34ce76c5b40b1ef777d
root@apstra-ztp:/containers_data/nginx#
```

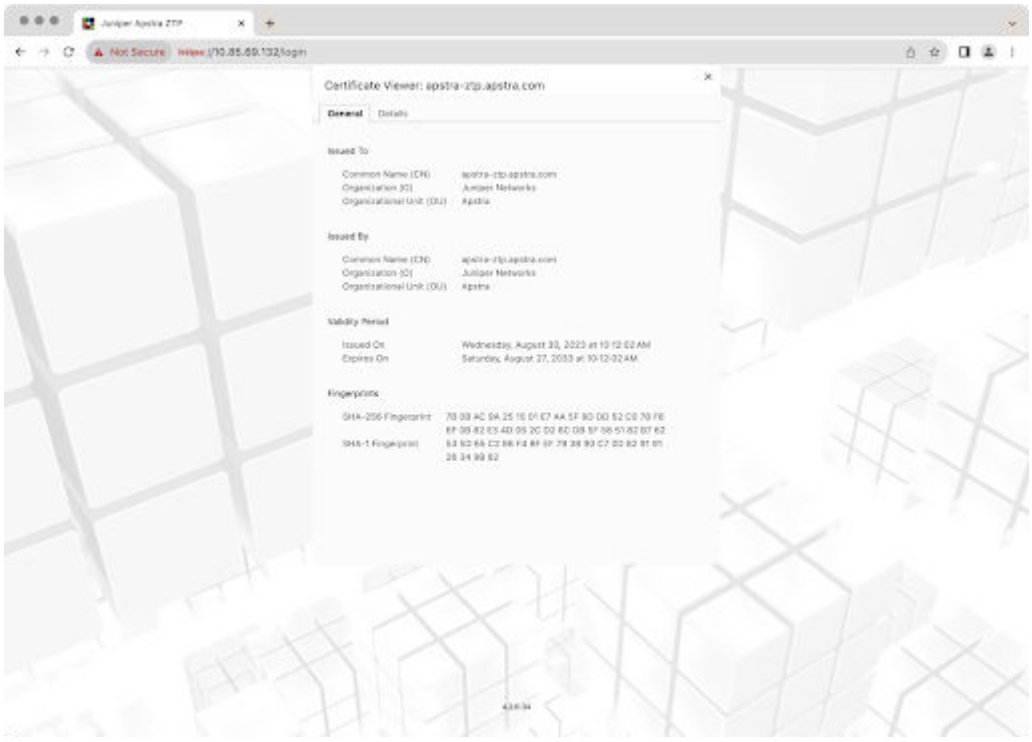
5. Edit the NGINX SSL configuration file `/containers_data/nginx/conf.d/ssl.conf` pointing `ssl_certificate` and `ssl_certificate_key` to the new key and certificate files. Note, the files in the `/containers_data/nginx` are mapped from files in the `/data` directory in the NGINX container.

```
root@apstra-ztp:/containers_data/nginx# nano conf.d/ssl.conf

server {
    listen 443 http2 ssl;
    listen [::]:443 http2 ssl;

    ssl_certificate /data/nginx.crt;
    ssl_certificate_key /data/nginx.key;
[snip]
```

6. To load the new certificate, restart the nginx container.

```
root@apstra-ztp:/containers_data/nginx# docker restart nginx
nginx
root@apstra-ztp:/containers_data/nginx#
```

7. Confirm that the new certificate is in your web browser and that the new certificate common name matches (for example, 'aos-server.apstra.com').



Next Step: Create Vendor-specific Custom Configuration, as needed.

Create Vendor-specific Custom Configuration

SUMMARY

You may need to customize configuration (`custom-config`) based on the device vendor.

IN THIS SECTION

You can use shell scripts to add custom configuration to devices during ZTP. These files are located in the TFTP directory or on a HTTP server that you point with a URL .

When you configure the ztp.json file you'll specify the bash file name in the custom-config field of the platform-specific section.

junos_custom.sh

To customize configuration on Juniper Junos OS and Junos OS Evolved devices, add configuration to `containers_data/tftp/junos_custom.sh`, a bash script file that's executed during the ZTP process.

It can execute Junos configuration commands, such as for Syslog, NTP, and SNMP authentication, before the device system agent is automatically installed.

> **NOTE**: Junos OS and Junos OS Evolved platforms with dual-RE setups require the `set system commit synchronize` command. Without this configuration, the ZTP process fails. We recommend adding the command to the `junos_custom.sh` file.

Refer to the example `junos_custom.sh` file.

```
#!/bin/sh

SOURCE_IP=$(cli -c "show conf interfaces em0.0" | grep address | sed 's/.*address \([0-9.]*\).*/
\1/')

# Syslog
SYSLOG_SERVER="192.168.59.4"
SYSLOG_PORT="514"
# NTP
NTP_SERVER="192.168.59.4"
# SNMP
SNMP_NAME="SAMPLE"
SNMP_SERVER="192.168.59.3"

# Syslog
cli -c "configure; \
set system syslog host $SYSLOG_SERVER any notice ; \
set system syslog host $SYSLOG_SERVER authorization any ; \
set system syslog host $SYSLOG_SERVER port $SYSLOG_PORT ; \
set system syslog host $SYSLOG_SERVER routing-instance mgmt_junos ; \
commit and-quit"
cli -c "configure; \
set system syslog file messages any notice ; \
set system syslog file messages authorization any ; \
commit and-quit"

# NTP
```

```
cli -c "configure; \
set system ntp server $NTP_SERVER routing-instance mgmt_junos ; \
set system ntp source-address $SOURCE_IP routing-instance mgmt_junos ; \
commit and-quit;"

# SNMP
cli -c "configure; \
set snmp name $SNMP_NAME; \
set snmp community public clients $SNMP_SERVER/32 ; \
set snmp community public routing-instance mgmt_junos ; \
set snmp routing-instance-access access-list mgmt_junos ; \
commit and-quit"
```

CAUTION: If you set external AAA authentication (for example `authentication-order`), you need to replicate the device system agent `device-user` and `device-user-password` in the AAA system. Otherwise, the device system agent generates an authentication error.

eos_custom.sh

To customize configuration on Arista EOS devices, add configuration to `containers_data/tftp/eos_custom.sh`, a bash script file that's executed during the ZTP process.

It can execute EOS configuration commands to set the SSH login banner, or any other system configuration that needs to be set before the device system agent is automatically installed.

Refer to the example `eos_custom.sh` file.

```
#!/bin/sh

FastCli -p 15 -c $'conf t\n service routing protocols model multi-agent\n hardware tcam\n system
profile vxlan-routing\n banner login\n
############################################################
UNAUTHORIZED ACCESS TO THIS DEVICE IS PROHIBITED
############################################################\n EOF\n'
```

NOTE: During the ZTP process, the EOS banner login is set to text saying "The device is in Zero Touch Provisioning mode ...". By default, the ZTP script copies this to the permanent configuration.

To prevent this, you **must** configure the `custom-config` pointing to a script (`eos_custom.sh` for example), which configures a different `banner login` or configure `no banner login`.

There must be a space after any `\n`.

nxos_custom.sh (onbox agent)

To customize configuration on Cisco NX-OS devices, add configuration to `containers_data/tftp/nxos_custom.sh`, a bash script file that's executed during the ZTP process.

It can execute NX-OS configuration commands that set system configuration, such as the SSH login banner, or other system configuration that needs to be set before the device system agent is automatically installed.

Refer to the example `nxos_custom.sh` file.

NOTE: You must use the `custom-config` file to add `copp profile strict`.

```
#!/bin/sh

/isan/bin/vsh -c "conf ; copp profile strict ; banner motd ~
########################################################
BANNER BANNER BANNER BANNER BANNER BANNER BANNER BANNER
########################################################
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Donec gravida, arcu vitae tincidunt sagittis, ligula
massa dignissim blah, eu sollicitudin nisl dui at massa.
Aliquam erat volutpat. Vitae pellentesque elit at
pulvinar volutpat. Etiam lacinia derp lacus, non
pellentesque nunc venenatis rhoncus.
########################################################
~"
```

nxos_custom.sh (Offbox Agent)

If you're using Apstra ZTP to prepare a Cisco NX-OS device for use with offbox agents, you must have the `custom-config` file enable the following NX-OS configuration commands.

```
feature nxapi
feature bash-shell
feature scp-server
feature evmed
copp profile strict
nxapi http port 80
```

You can use the following `nxos_custom.sh` to add these along with a banner.

```
#!/bin/sh

/isan/bin/vsh -c "conf ; feature nxapi ; nxapi http port 443 ; feature bash-shell ; feature scp-
server ; feature evmed ; copp profile strict ; banner motd ~
##########################################################
BANNER BANNER BANNER BANNER BANNER BANNER BANNER BANNER
##########################################################
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Donec gravida, arcu vitae tincidunt sagittis, ligula
massa dignissim blah, eu sollicitudin nisl dui at massa.
Aliquam erat volutpat. Vitae pellentesque elit at
pulvinar volutpat. Etiam lacinia derp lacus, non
pellentesque nunc venenatis rhoncus.
##########################################################
~"
```

sonic_custom.sh

To customize configuration on Enterprise SONiC devices, add configuration to `containers_data/tftp/sonic_custom.sh`, a bash script file that's executed during the ZTP process.

It can execute EOS configuration commands, such as for settting Radius authentication, before the device system agent is automatically installed.

Refer to the example `sonic_custom.sh` file.

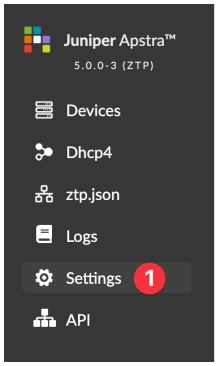
```
#!/bin/bash

sed -i s/"#Banner.*"/"Banner \/etc\/issue.net"/ /etc/ssh/sshd_config

cat >& /etc/issue.net << EOF
Provisioned by AOS
Date: $(date)
EOF

service ssh restart
```

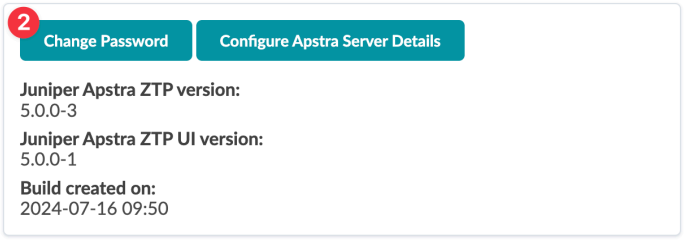
Configure Credentials for Apstra ZTP Server GUI

You'll need to create a user for accessing the Apstra ZTP server GUI. (Different credentials are used for the Apstra server GUI and the Apstra *ZTP* server GUI.) You 'll use these credentials to connect to Apstra's API.

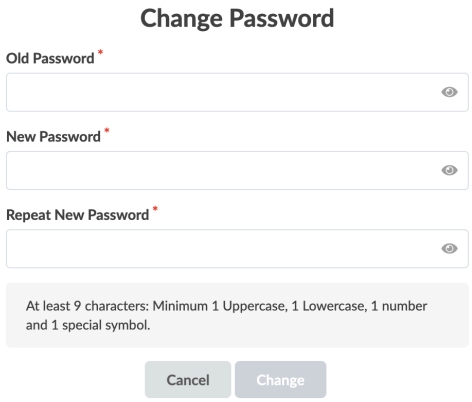
1. From the latest web browser version of Google Chrome or Mozilla FireFox, enter the URL `https://<apstra_ztp_server_ip>` where `<apstra_ztp_server_ip>` is the IP address of the Apstra ZTP server (or a DNS name that resolves to the IP address of the Apstra ZTP server).
2. If a security warning appears, click **Advanced** and **Proceed to** the site. The warning occurs because the SSL certificate that was generated during installation is self-signed, and you didn't replace it with a signed one when you installed the software. We recommend, for security reasons, that you "replace the SSL certificate" on page 721.
3. If this is the first time you're logging in, enter username **admin** and password **admin** (the default password). You must update the default Apstra ZTP GUI password. (The only user that's supported in the Apstra ZTP GUI is **admin**.) You can also change the password any time after this initial update.

   If this is not the first time you're logging in to the ZTP GUI, log in, then in the left navigation menu, click **Settings**, then click **Change Password**, as shown below.





4. In the **Change Password** dialog that opens, change the password to one that meets complexity requirements, then click **Change**.



You're automatically logged out.

Next Step: Configure Apstra server connection details.

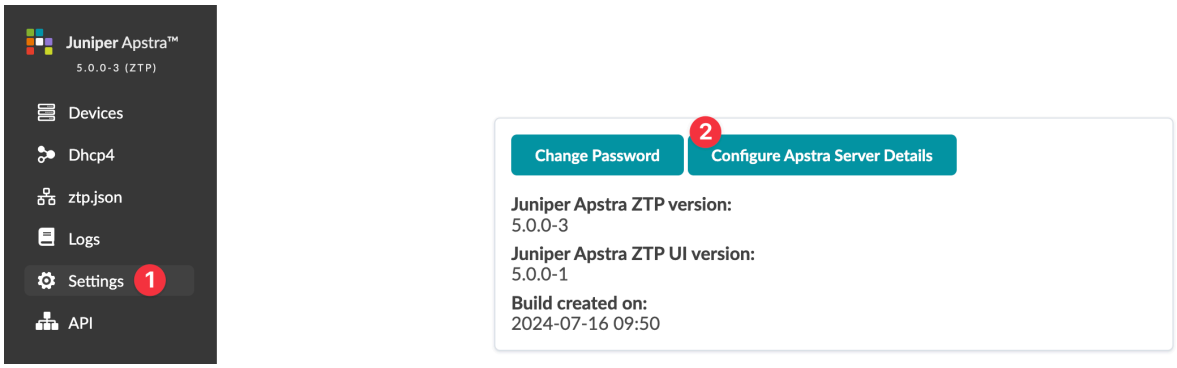
Configure Apstra Server Connection Details

Before configuring Apstra server details, we recommend that you create a user profile on the Apstra server for the sole purpose of interacting with Apstra ZTP.

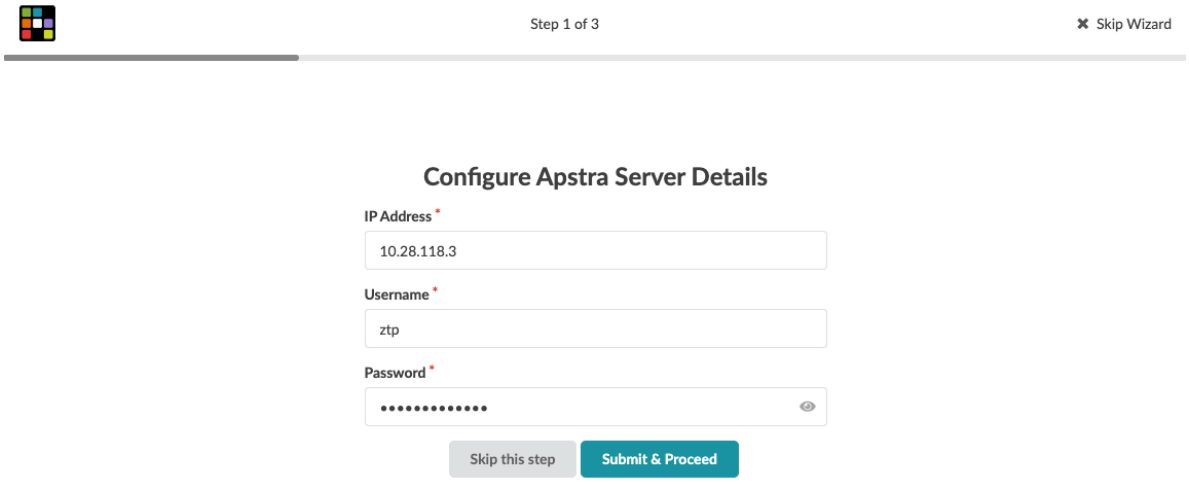
You'll need to specify the Apstra server IP address that communicates with ZTP devices and the Apstra ZTP server. The Apstra ZTP server and the devices to be onboarded with ZTP connect to the Apstra server API, for logging and device system agent creation.

1. Log in to the Apstra ZTP server GUI.
2. If this is the first time you're logging in after changing the default password, you're presented with a 3-step wizard. Step 1 of the wizard is to configure Apstra server details. The wizard runs only the first time you log in with the new password; It won't appear again. (You can skip this step and come back later to complete this configuration.)

If you're not using the wizard, then from the left navigation menu of the Apstra ZTP GUI, click **Settings**, then click **Configure Apstra Server Details**, as shown below.



3. Enter the Apstra server IP address that will communicate with ZTP devices and the Apstra ZTP server.



4. Enter the username and password for the ZTP user profile you previously created, then click **Submit & Proceed** (or **Save** if you're not using the wizard). The Apstra server credentials are verified; incorrect credentials result in an error.

Next Step: Configure DHCP Server for Apstra ZTP.

RELATED DOCUMENTATION

Create User Profile for Communicating with ZTP Server | 717

Configure DHCP Server for Apstra ZTP

IN THIS SECTION

When a device connects to a network, Dynamic Host Configuration Protocol (DHCP) automatically assigns an available IP address to it. Apstra, by default, uses a Kea DHCP server for device management (as of Apstra version 5.0.0, previously it was ISC). If you set up a different DHCP server, you're responsible for configuring the same details as described here.

DHCP Parameters

The DHCP file includes the following details:

Table 11: DHCP Basic Setup Parameters

Field Name	Description
Domain Search (`domain-search`)	Option to configure multiple search domains (Example entry: `dc1.yourdatacenter.com`) To specify more than one value, separate them with a comma (,). For the parsing of the file to succeed, each line must end with a semicolon (;)
Domain Name (`domain-name`)	Option to indicate the domain name to use (Example entry: `dc1.yourdatacenter.com`)
Domain Name Servers (`domain-name-servers`)	Option to list all of the DNS IP addresses. To specify more than one value, separate them with a comma (,). For the parsing of the file to succeed, each line must end with a semicolon (;)
TFTP Server IP (`tftp-server-name`)	IP address of the TFTP server

Table 11: DHCP Basic Setup Parameters *(Continued)*

Field Name	Description
Subnet / subnet `subnet`	CIDR notation of IPv4 subnet (Example entry: `10.28.180.4/24`)
Subnet / IP Pools / IP Range Start	The first address of the IP subnet available for dynamic assignment
Subnet / IP Pools / IP Range End	The last address of the IP subnet available for dynamic assignment
Subnet / Router	The default gateway IP address
Reservation Mode	Defines the reservation modes to use for this subnet
Host Reservations / Hardware Address	The host MAC address
Host Reservations / Fixed IP Address	The fixed IP address allocted for the host
Reservation Mode Default	All, Global, Out of Pool, or Disabled
Global Host Reservations / Hardware Address	The host MAC address
Global Host Reservations / Fixed IP Address	The fixed IP address allocted for the host

Table 12: Configurator Parameters

Field Name	Description
valid-lifetime (`valid-lifetime`)	Defines the default IP lease validity duration
max-valid-lifetime (`max-valid-lifetime`)	Defines the maximum IP lease duration
match-client-id (`match-client-id`)	To indicate if server will use client id for lease lookups

Table 12: Configurator Parameters *(Continued)*

Field Name	Description
interfaces-config / interfaces (`interfaces`)	The interfaces on which the DHCP server would listen for DHCP messages
lease-database / type (`type`)	Defines the lease-database type
lease-database / name (`name`)	The file path inside DHCPD container where lease information would be stored
lease-database / lfc-interval (`lfc-interval`)	The intervals at which DHCP will initiate lease file cleanup
subnet / id (id)	
subnet (`subnet`)	CIDR notation of IPv4 subnet
option-date / name	The standard DHCPv4 option name
option-date / code	The standard DHCPv4 option code
option-date / space	space
option-date / data	The value passed to the specified option
option-def /name (`name`)	The standard DHCPv4 option name
option-def / code (`code`)	The standard DHCPv4 option code
option-def / space (`space`)	space
option-def / type (`type`)	The value passed to the specified option

Table 12: Configurator Parameters *(Continued)*

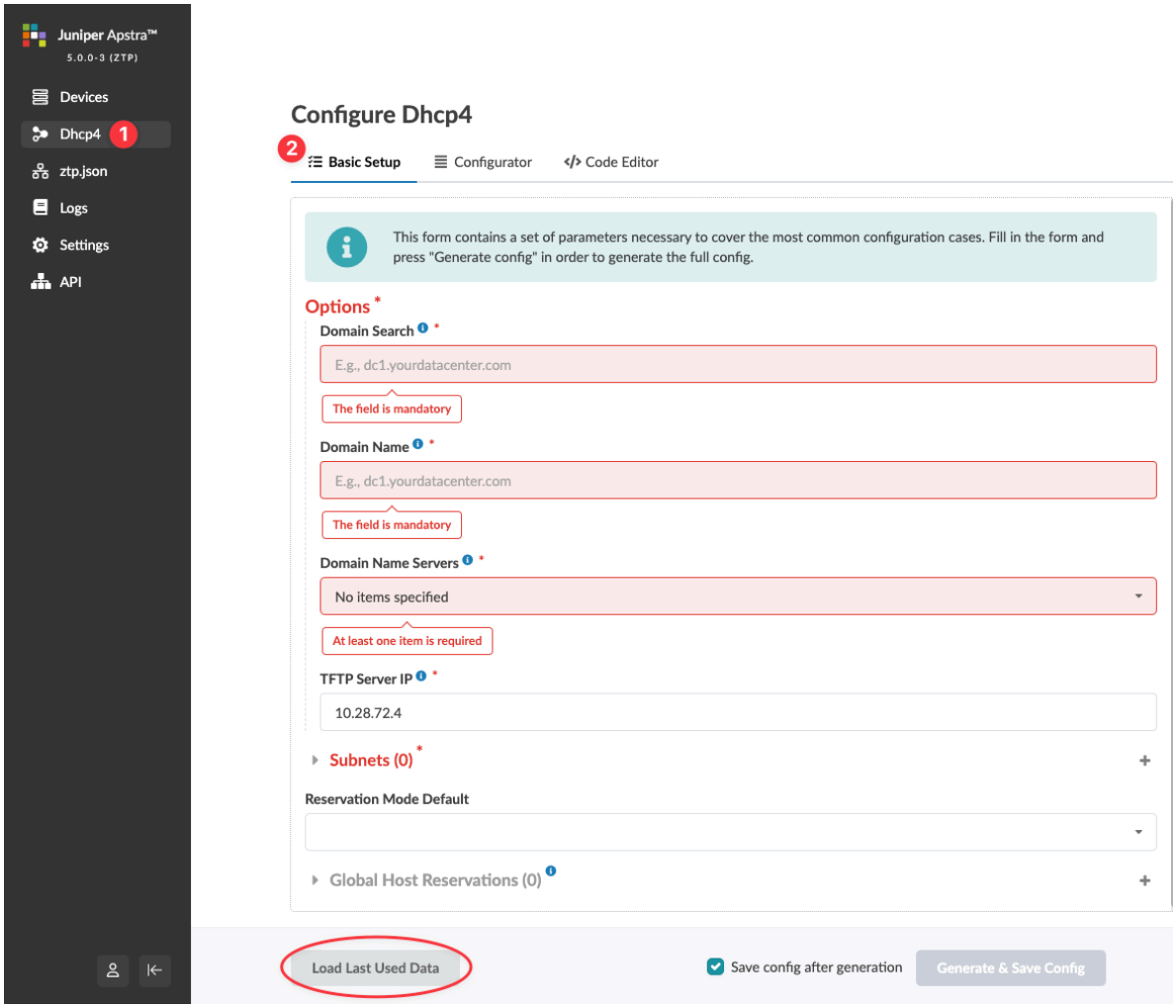
Field Name	Description
client-classes / name (`name`)	"arista", "cisco", "sonic"
client-classes / test (`test`)	test
control-socket / socket-name (`socket-name`)	The location of UNIX socket file used to communicate with control agent
control-socket / socket-type (`socket-type`)	The DHCP socket type
loggers / name (`name`)	name
loggers / severity (`severity`)	Defines the logging severity level
loggers / debuglevel (`debuglevel`)	Specifies the level of debug message when severity is set to debug
loggers / output	output
reservation-mode (`reservation-mode`)	Types of reservations allowed (none, all, global, out of pool, disabled)
reservations-global (`reservations-global`)	Indicates if the server should look up global reservations
reservations-in-subnet (`reservations-in-subnet`)	Indicate if the server should look up in-subnet reservations
reservations-out-of-pool (`reservations-out-of-pool`)	Specify if the server can assume that all reserved addresses are out-of-pool. It can be ignored if "reservation-in-subnet" is false
reservations / Hardware Address	Used to list global reservations
reservations / Fixed IP Address	The fixed IP address allocated for the host

Use GUI Configurator to Configure DHCP

You can set up optional parameters that the DHCP services will send to every device that asks for DHCP services. You have the option of configuring these parameters at a later time via configlets. (If you define these parameters in the DHCP service, don't try to set them up again via Apstra; the device OS may return an error.)

If in doubt, don't enter random parameters; it may result in timeouts as a service tries to resolve IP addresses.

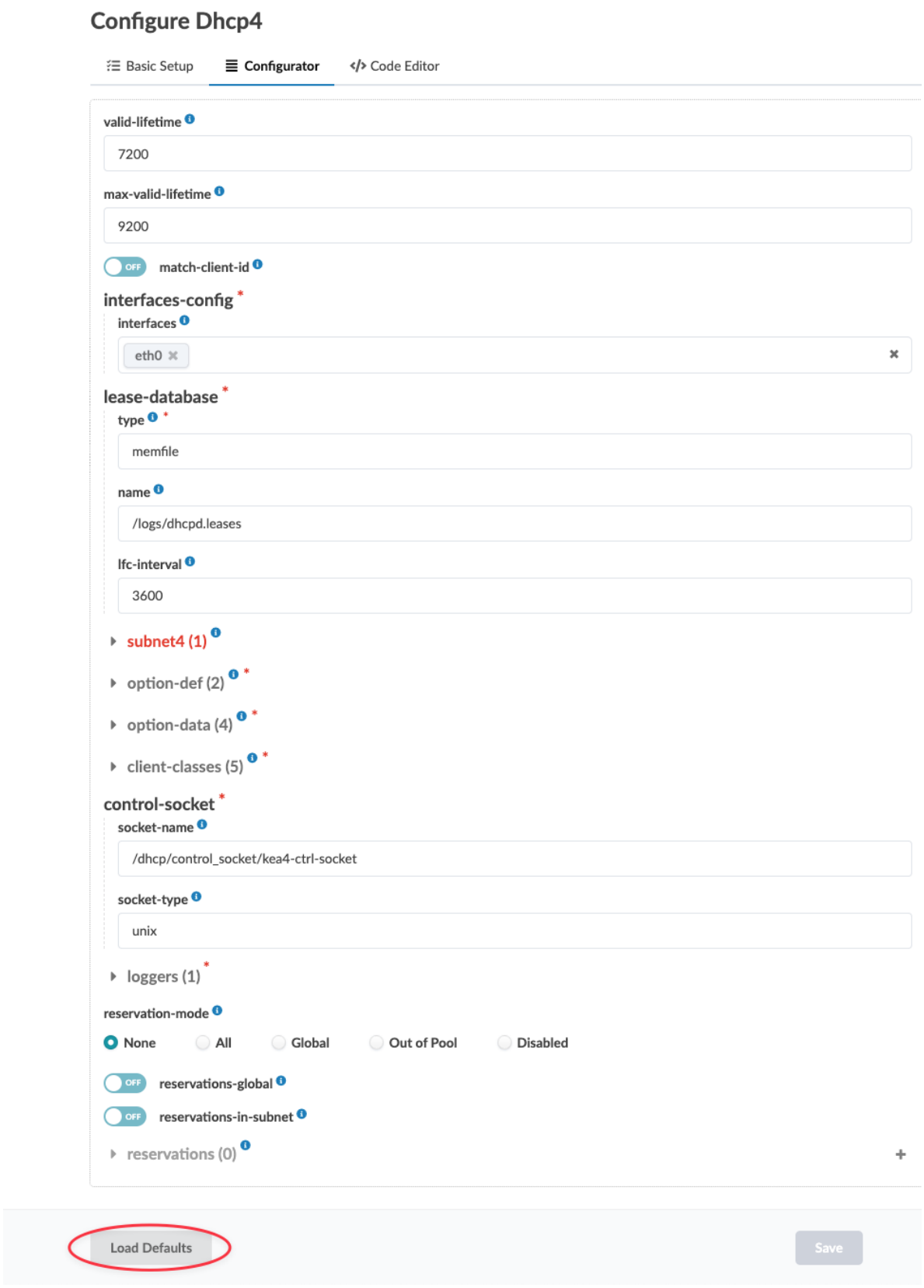
1. If you're using the wizard, step 2 is to configure DHCP. (You can skip this step for now and come back later to complete this configuration.) If you're not using the wizard, then from the left navigation menu of the Apstra ZTP GUI, click **Dhcp4**, then click **Basic Setup**.



   The **Basic Setup** screen appears.
2. Enter basic setup details, as applicable. See **DHCP Parameters** above for details. To cancel your entries, you can click **Load Last Used Data** at the bottom of the screen (new in Apstra version 5.0.0).
3. Click **Configurator** (to the right of the Basic Setup tab).

The **Configurator** screen appears.

4. Enter details, as applicable. If you've entered details and want to start over, you can click **Load Defaults** (bottom left of screen) (new in Apstra version 5.0.0).



5. Click **Save** to save the new configuration.

Use GUI Code Editor to Configure DHCP

1. From the left navigation menu of the Apstra ZTP GUI, click **Dhcp4**, then click **Code Editor**.

   The **Code Editor** screen opens.

2. Update the file, as appropriate. If you've entered details and want to start over, you can click **Load Defaults** (bottom left of screen) (new in Apstra version 5.0.0).
3. Click **Save** to save the new configuration.

Use Text Editor to Configure DHCP

We recommend that you use the "Apstra ZTP GUI Configurator" on page 736 to configure DHCP, but you have the option of configuring the file directly with a text editor, such as vi or nano.

> **NOTE**: All configuration files are owned by `root`. You must use sudo to run commands as `root` using the `sudo` command or after becoming `root` with the `sudo -s` command.

1. Open a terminal and SSH into the Apstra ZTP server.

   `ssh admin@<apstra-ztp-server-ip>` where `<apstra-ztp-server-ip>` is the IP address of the Apstra ZTP server.

2. To see DHCP files on the Apstra ZTP VM, navigate to the `/containers_data/dhcp` directory.

   `admin@apstra-ztp:~$ sudo ls -l /containers_data/dhcp`

```
admin@apstra-ztp:~$ sudo ls -l /containers_data/dhcp
total 36
drwxrwxrwx 2 root  root  4096 May  9 18:54 control_socket
-rwxrwxrwx 1 root  root  5838 May  9 18:53 dhcpd.conf
-rwxrwxrwx 1 admin admin 5836 May  7 17:38 dhcpd.conf.template
-rw-r--r-- 1 root  root     0 May  9 23:22 dhcpd.cong
-rwxrwxrwx 1 admin admin  198 May  7 17:38 Dockerfile
```

```
-rwxrwxrwx 1 admin admin 3074 May  7 17:38 init.sh
-rwxrwxrwx 1 admin admin 1830 May  7 17:38 README
-rwxrwxrwx 1 admin admin 1821 May  7 17:38 rsyslog.conf
admin@apstra-ztp:~$
```

3. Open the `dhcpd.conf` file in a text editor, such as vi or nano and update the file, as appropriate.

   ```
   admin@apstra-ztp:~$ sudo vi /containers_data/dhcp/dhcpd.conf
   ```

4. After modifying any DHCP configuration, restart the Apstra ZTP DHCP process with the `sudo docker restart dhcpd` command. This forces it to read the new file values.

   ```
   admin@apstra-ztp:~$ docker restart dhcpd
   dhcpd
   admin@apstra-ztp:~$
   ```

ztp.json Keys

IN THIS SECTION

The Apstra ZTP configuration file (`ztp.json`) includes all configuration for the Apstra ZTP script (`ztp.py`). Never modify `ztp.py` directly. You can use several methods for configuring `ztp.json`. First, familiarize yourself with the keys in the file, as described below, then configure the file, as needed for your devices (as described in later pages).

Key Categories

Contents of the `ztp.json` file are organized from more general sections to more specific sections as follows:

- **defaults** - Configured values in the `defaults` section are used for all devices unless more specific sections use the same keys. The more specific section values take precedence. Here's an example of what you might put in the defaults section:

```
"defaults": {
  "device-root-password": "root-password-123",
  "device-user": "admin",
  "device-user-password": "admin-password-123",
  "system-agent-params": {
    "agent_type": "onbox",
    "install_requirements": false
  }
}
```

- **platform-specific** - Configured values in each of the vendor platform sections (`junos`, `junos-evo`, `eos`, `nxos`, `sonic`, `linux`) are used for all devices from that vendor unless the same keys in more specific sections are defined. Here's an example for devices using SONiC OS:

```
"sonic": {
  "sonic-versions": ["SONiC-OS-3.4.0-Enterprise_Advanced"],
  "sonic-image": "http://10.85.24.52/sonic/3.4.0/sonic-3.4.0-GA-adv-bcm.bin",
  "device-root-password": "admin",
  "device-user": "admin",
  "device-user-password": "admin",
  "custom-config": "sonic_custom.sh",
  "system-agent-params": {
    "agent_type": "onbox",
    "job_on_create": "install"
  }
}
```

- **model-specific** - Configured values in a specific model number section are used for all devices of that model, unless, of course, the same keys are used in a serial number section. Then the value in the serial number section would be used. Here's an example for a specific Juniper device:

```
"QFX10002-36Q": {
  "junos-versions": ["21.2R1-S2.2"],
  "junos-image": "http://10.85.24.52/juniper/21.2R1-S2.2/jinstall-host-qfx-10-f-x86-64-21.2R1-
```

```
S2.2-secure-signed.tgz"
 }
```

- **serial number-specific** - Configured values under a specific serial number (system ID) are used for that one device. For example:

```
"TH0TFD6TCET0015G0015": {
   "sonic-versions": ["SONIC-OS-4.0.5-Enterprise_Advanced"],
   "sonic-image": "http://10.85.24.52/sonic/4.0.5/sonic-broadcom-enterprise-advanced-4.0.5-
GA.bin"
}
```

Keys List

See descriptions and examples below for all keys in the `ztp.json` file. All keys are included in each category, whether or not they apply specifically to that category. For example, you may notice the `junos` section includes the key `nxos-version`. This is to allow all keys to be included in the `defaults` category. You can ignore or remove keys that don't apply to your devices.

nxos-versions

The `nxos-versions` parameter includes valid OS versions for Cisco NX-OS devices.

Example: `"nxos-versions": [ "10.2(5)", "9.3(11)" ]`

nxos-image(-location)

In the Configurator is called `nxos-image-location` and in the code editor and CLI it's called `nxos-image`.

If the version running on the device doesn't match a version in `nxos-versions`, then the Cisco NX-OS image location specified in the `nxos-image` field is uploaded.

By default, the image is loaded via TFTP. You can also load the image via HTTP by specifying the HTTP/HTTPS server URL with the IP address. You can also upload NX-OS images to Devices / OS images on the Apstra server.

TFTP Example (default): "nxos-image": ["nxos.10.2.5.bin"]

HTTP Server Example: "nxos-image": "http://192.168.59.4/nxos.10.2.5.bin"

Devices / OS Images Example: "nxos-image": "https://192.168.59.3/dos_images/nxos.10.2.5.bin"

eos-versions

The `eos-versions` parameter includes valid OS versions for Arista EOS devices.

Example: `"eos-versions": [ "4.27.6M" ]`

eos-image(-location)

In the Configurator is called `eos-image-location` and in the code editor and CLI it's called `eos-image`.

If the version running on the device doesn't match a version in `eos-versions`, then the Arista EOS SWI image location specified in the `eos-image` field is uploaded.

By default, the image is loaded via TFTP. You can also load the image via HTTP by specifying the HTTP/HTTPS server URL with the IP address. You can also upload NX-OS images to Devices / OS images on the Apstra server.

TFTP Example (default): "eos-image": ["EOS-4.24.5M.swi"]

HTTP Server Example: "eos-image": "http://192.168.59.3/dos_images/EOS-4.21.51F.swi"

Devices / OS Images Example: "nxos-image": "https://192.168.59.3/dos_images/nxos.10.2.5.bin"

To use any HTTP server for image transfer, enter a valid HTTP or HTTPS URL with IP address. For example: `"eos-image": "http://192.168.59.3/dos_images/EOS-4.27.6M.swi"`

This example uses HTTP from the Apstra ZTP server (192.168.59.4) to transfer the Arista EOS image from the Apstra ZTP `/container_data/tftp/` directory.

You can also upload Arista EOS images to the Apstra controller Devices / OS Images. For example:

`"nxos-image": "https://192.168.59.3/dos_images/EOS-4.27.6M.swi"`

junos-versions

The `junos-versions` parameter includes valid OS versions for Juniper Junos OS devices.

Example: `"junos-versions": [ "22.4R2" ]`

junos-image(-location)

In the Configurator is called `junos-image-location` and in the code editor and CLI it's called `junos-image`.

If the running Junos OS version doesn't match a version in the `junos-versions` list, then the image in the `junos-image` field is uploaded.

By default, the image is loaded from the ZTP server's `/container_data/tftp/` directory via TFTP. To use any HTTP server for transferring the image, enter a valid HTTP URL with IP address. For example:

"junos-image": "http://192.168.59.4/jinstall-host-qfx-5-18.4R3-S4.2-signed.tgz"

To use any HTTP server for image transfer, enter a valid HTTP or HTTPS URL with IP address. For example:

```
"junos-image": "http://192.168.59.4/jinstall-host-qfx-5e-x86-64-21.4R3-S4.13-secure-signed.tgz"
```

This example uses HTTP from the Apstra ZTP server (192.168.59.4) to transfer the Juniper Junos image from the Apstra ZTP `/container_data/tftp/` directory.

You can also upload Juniper Junos images to the Apstra controller Devices / OS Images. For example:

```
"junos-image": "https://192.168.59.3/dos_images/jinstall-host-qfx-5e-x86-64-21.4R3-S4.13-secure-signed.tgz"
```

junos-evo-versions

The `junos-evo-versions` parameter includes valid OS versions for Juniper Junos OS Evolved devices.

Example: `"junos-versions": [ "22.4R2-EVO" ]`

junos-evo-image(-location)

sonic-versions

The `sonic-versions` parameter includes valid OS versions for Enterprise SONiC devices.

Example: `"sonic-versions": [ "SONiC-OS-4.0.5-Enterprise_Advanced" ]`

sonic-image(-location)

In the Configurator is called `sonic-image-location` and in the code editor and CLI it's called `sonic-image`.

This is the filename of the SONiC ONIE BIN image to load if the running version does not match a version in the `sonic-versions` list.

To use any HTTP server for image transfer, enter a valid HTTP or HTTPS URL with IP address. For example:

```
"sonic-image":
         "http://192.168.59.3/sonic-broadcom-enterprise-advanced-4.0.5-GA.bin"
```

This example uses HTTP from the Apstra ZTP server (192.168.59.4) to transfer the SONiC image from the Apstra ZTP `/container_data/tftp/` directory.

You can also upload SONiC images to the Apstra controller Devices / OS Images. For example:

```
"sonic-image":
        "https://192.168.59.3/dos_images/sonic-broadcom-enterprise-advanced-4.0.5-GA.bin"
```

device-root-password

The ZTP process sets the device root password to this value. For Arista EOS and Cisco NX-OS devices, the `device-root-password` is used to set the password for the system `admin` password.

Example: `"device-root-password": "root-admin-password"`

device-user

Username for the device system agent. Also, if necessary, the ZTP process creates a user on the device with this username and and the `device-user-password`.

Example:

```
"device-user": "aosadmin",
   "device-user-password": "aosadmin-password"
```

device-user-password

Password for the device system agent. Also, if necessary, the ZTP process creates a user on the device with the `device-user` and this password.

Example:

```
"device-user": "aosadmin",
   "device-user-password": "aosadmin-password"
```

custom-config

This is the filename of the custom configuration shell script in the TFTP directory or a URL pointing to the file on a HTTP server. This shell script runs during the ZTP process allowing you to add custom configuration to the device. **See Platform Specific Information** for more information.

Example: `"custom-config": "junos_custom.sh"`

dual-routing-engine (check box)

system-agent-params

System agent parameters are used to create new users and device system agents on each device. (For all available `system-agent-params` options, see the REST API documentation for `/api/system-agents`.)

```
"system-agent-params": {
  "id": "",
  "agent_type": "",
  "platform": "",
  "job_on_create": "",
  "operation_mode": "",
  "profile": "",
  "packages": [],
  "force_package_install": false,
  "install_requirements": false,
  "enable_monitor": false
```

The individual system agent parameters are described below.

agent_type (system-agent-params)

Agent type is either onbox or offbox

Example: `"agent_type": "onbox"`

platform (system-agent-params)

This field is used only for offbox agents only. Set it to the device platform ("eos", "nxos", "junos"). Lowercase only

Example: `"platform": "junos"`

job_on_create (system-agent-params)

To have the onbox agent installed on the device, `set job_on_create` to `install`

Example: `"job_on_create": "install"`

operation_mode (system-agent-params)

profile (system-agent-params)

The device agent profile as defined in Apstra to use during agent creation. The value must be the ID of the agent profile, not the agent profile name.

Example: “profile”: “8d68d1ec-c168-4ef3-8ffd-09389c17a3e4”

Requirements for Juniper on Apstra version 4.2.0: If you need to provide “profile” parameters, you must use UUID instead of the profile name/label.

The parameters `force_package_install`, `install_requirements`, and `enable_monitor` are always visible in the ZTP server. These will cause agent creation failure during the ZTP process. You must remove these parameters from `system-agent-params` for Juniper agent creation to work. However, due to a bug, when you remove these parameters from the ztp.json file via the Apstra GUI, they aren't actually removed. The configurator adds them back in. To prevent this from happening, use the CLI instead of the Apstra ZTP GUI, and log in via an SSH connection to the ZTP server. Then restart the tftp container.

packages (system-agent-params)

Set to configure the additional SDK or extended telemetry packages to upload to the system agent.

Example:

```
"packages": [
  "aos-deployment-helper-nxos",
  "aosstdcollectors-builtin-nxos",
```

```
  "aosstdcollectors-custom-nxos"
]
```

force_package_install (system-agent-params)

For Juniper devices used in Apstra 4.2.0, you must remove this parameter via CLI. If you use the Apstra ZTP GUI, the parameter will be added back in.

install_requirements (system-agent-params)

Always set to false. Not currently needed for any supported Network Operating System.

Example: `"install_requirements": false`

For Juniper devices used in Apstra 4.2.0, you must remove this parameter via CLI. If you use the Apstra ZTP GUI, the parameter will be added back in.

enable_monitor (system-agent-params)

For Juniper devices used in Apstra 4.2.0, you must remove this parameter via CLI. If you use the Apstra ZTP GUI, the parameter will be added back in.

open_options (is this still relevant?)

Example:

```
"open_options": {
  "proto": "https",
  "port": "443"
}
```

Offbox agents only. Set to enable HTTPS between offbox agent to device API interface. If open_options isn't defined, the connection defaults to HTTP.

Examples

See the sections below for some examples of `ztp.json` values.

Defaults

An example of values you might want to include as defaults include the following:

```
"defaults": {
   "nxos-image": "http://buildfiles.dc1.apstra.com/apstrktr/switch_images/cisco/nxos.10.2.5.bin",
   "eos-image": "http://buildfiles.dc1.apstra.com/apstrktr/switch_images/arista/EOS-4.27.6M.swi",
   "junos-image": "http://10.24.128.10/apstrktr/switch_images/juniper/junos-5e-22.2R3.15.tgz",
   "junos-evo-image": "http://10.24.128.10/apstrktr/switch_images/juniper/junos-evo-install-qfx-
ms-fixed-x86-64-22.2R3.13-EVO.iso",
   "sonic-image": "http://buildfiles.dc1.apstra.com/apstrktr/switch_images/sonic/sonic-4.0.5-GA-
adv-bcm.bin",
   "device-root-password": "strongrootpassword",
   "device-user": "admin",
   "device-user-password": "stronguserpassword"
   "system-agent-params": {
       "agent_type": "onbox",
       "install_requirements": false
}
```

All devices would use these values unless values for more specific categories are configured, then those would take precedence. More specific categories include platform-specific, such as `"junos"`, `"junos-evo"`, `"eos"`, `"nxos"`, and `"sonic"`; model-specific, such as `"QFX10002-60C"`; and serial number-specific, such as `"TH0TFD6TCET0015G0015"`.

Cisco Onbox Agent on Apstra ZTP 4.2.0 Example

```
{
"nxos": {
   "nxos-versions": [ "10.2(5)" ],
   "nxos-image": "http://192.168.59.4/nxos.10.2.5.bin",
   "device-root-password": "strongrootpassword",
   "custom-config": "nxos_custom.sh",
   "device-user": "admin",
   "device-user-password": "stronguserpassword",
   "system-agent-params": {
      "agent_type": "onbox",
      "job_on_create": "install"
      }
```

```
   }
}
```

Cisco Offbox Agent with HTTP on Apstra ZTP 4.2.0 Example

```
{
"nxos": {
   "nxos-versions": [ "10.2(5)" ],
   "nxos-image": "http://192.168.59.4/nxos.10.2.5.bin",
   "custom-config": "nxos_custom.sh",
      ]"device-user": "admin",
      "device-user-password": "admin-password",
   "system-agent-params": {
      "username": "admin",
      "password": "admin",
      "agent_type": "offbox",
      "platform": "nxos",
      "open_options": {
         "proto": "https",
         "port": "443"
      },
      "packages": [
         "aos-deployment-helper-nxos",
         "aosstdcollectors-builtin-nxos",
         "aosstdcollectors-custom-nxos"
         ]
      }
   }
}
```

This configuration enables secure offbox agent HTTPS (port 443) between the offbox agent on the server and the device API.

NOTE: open_options" can't be configured from the Apstra ZTP UI configurator. You must use the GUI code editor or edit the ztp.json file from the CLI.

Arista Onbox Agent on Apstra ZTP 4.2.0 Example

```
{
  "eos": {
    "eos-versions": [ "4.27.6M" ],
    "eos-image": "http://192.168.59.3/EOS-4.27.6M.swi",
    "custom-config": "eos_custom.sh",
    "device-root-password": "admin-password",
    "device-user": "admin",
    "device-user-password": "admin-password",
    "system-agent-params": {
      "agent_type": "onbox",
      "job_on_create": "install"
    }
  }
}
```

Junos Offbox Agent on Apstra ZTP 4.2.0 Example

```
{
  "junos": {
    "junos-versions": [ "21.4R3-S4.13" ],
    "junos-image": "http://192.168.59.4/jinstall-host-qfx-5e-x86-64-21.4R3-S4.13-secure-
signed.tgz",
    "device-root-password": "root-password",
    "device-user": "admin",
    "device-user-password": "admin-password",
    "custom-config": "junos_custom.sh",
    "system-agent-params": {
      "platform": "junos",
      "agent_type": "offbox",
      "job_on_create": "install"
    }
  }
}
```

```
{
        "junos": {
                "junos-versions": ["21.2R1-S2.2"],
```

```
                "junos-image": "http://10.85.24.52/juniper/21.2R1-S2.2/jinstall-host-qfx-5e-
x86-64-21.2R1-S2.2-secure-signed.tgz",
                "device-root-password": "root123",
                "device-user": "admin",
                "device-user-password": "admin",
                "system-agent-params": {
                        "platform": "junos",
                        "agent_type": "offbox",
                        "job_on_create": "install"
                }
        },
        "QFX10002-36Q": {
               "junos-versions": ["21.2R1-S2.2"],
                "junos-image": "http://10.85.24.52/juniper/21.2R1-S2.2/jinstall-host-qfx-10-f-
x86-64-21.2R1-S2.2-secure-signed.tgz"
        },
        "JNP10002-60C [QFX10002-60C]": {
                "junos-versions": ["21.2R1-S1.3"],
                "junos-image": "http://10.85.24.52/juniper/21.2R1-S1.3/junos-vmhost-install-qfx-
x86-64-21.2R1-S1.3.tgz"
        }
}
```

Junos Evolved Onbox Agent on Apstra ZTP 4.2.0 Example

```
{
  "junos-evo": {
    "junos-evo-versions": [ "22.4R2.11-EVO" ],
    "junos-evo-image": "http://192.168.59.4/junos-evo-install-qfx-ms-x86-64-22.4R2.11-EVO.iso",
    "device-root-password": "root-password",
    "device-user": "admin",
    "device-user-password": "admin-password",
    "custom-config": "junos_custom.sh",
    "system-agent-params": {
      "agent_type": "onbox",
      "job_on_create": "install"
    }
  }
}
```

You can use the following additional fields for dual RE platforms, such as PTX10004.

```
"dual-routing-engine": true,
"management-ip": "10.161.37.7",
"management-gw-ip": "10.161.39.254",
"management-subnet-prefixlen": "21",
"management-master-ip": "10.161.37.8",
"management-backup-ip": "10.161.37.9",
```

Juniper OS Evolved

Enterprise SONiC Onbox Agent on Apstra ZTP 4.2.0 Example

```
{
"sonic": {
   "sonic-versions": [ "SONiC-OS-4.0.5-Enterprise_Advanced" ],
   "sonic-image": "http://192.168.59.4/sonic-broadcom-enterprise-advanced-4.0.5-GA.bin",
   "device-root-password": "root-password",
   "device-user": "admin",
   "device-user-password": "admin-password",
   "custom-config": "sonic_custom.sh",
   "system-agent-params": {
      "agent_type": "onbox",
      "job_on_create": "install"
      }
   }
}
```

NOTE: If you use another `device-user` besides `admin` (`aosadmin` for example) Apstra ZTP creates this new user, but it doesn't change the password for the default SONiC `admin` user (password set to `YourPaSsWoRd` by default).

Model-specific Example

```
"JNP10002-60C [QFX10002-60C]": {
    "junos-versions": [ "21.2R1-S1.3" ],
```

```
    "junos-image": "http://10.85.24.52/juniper/21.2R1-S1.3/junos-vmhost-install-qfx-
x86-64-21.2R1-S1.3.tgz",
```

Configure ztp.json with Configurator

SUMMARY

The Apstra ZTP Configurator is a GUI for configuring the ztp.json file.

IN THIS SECTION

The preferred method for configuring ztp.json is with the Apstra ZTP Configurator. Using the Configurator reduces the chances of human error. The steps below show you how to access the Configurator. Check out the platform-based examples, then refer to the ztp.json Keys page for details about all the keys.

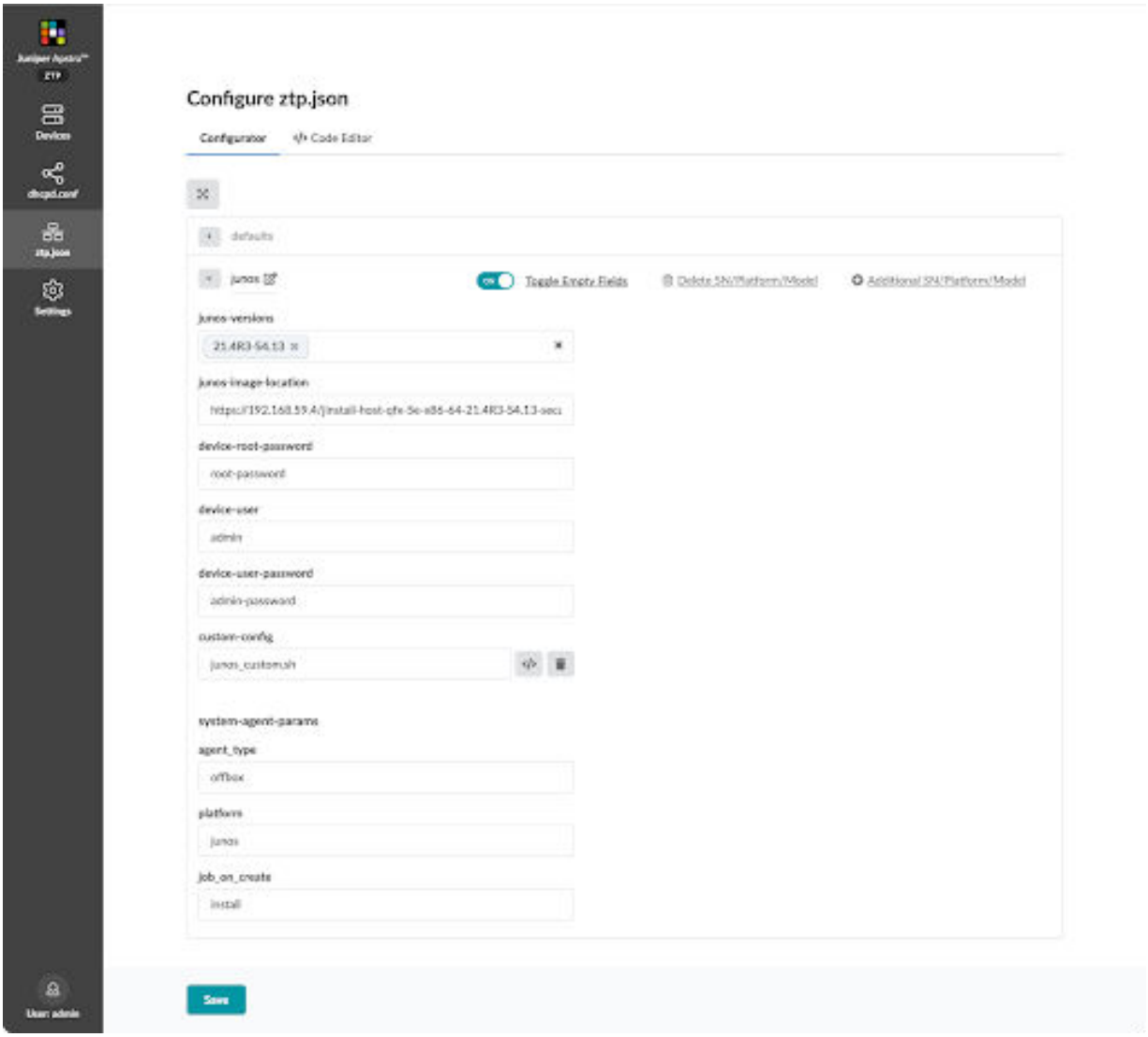
Access the ztp.json Configurator

1. If you're using the wizard, step 3 is to configure the `ztp.json` file. (You can skip this step for now and come back later to complete this configuration.) If you're not using the wizard, then from the left navigation menu of the Apstra ZTP GUI, click **ztp.json**

   The default screen goes to the **Code Editor** page.
2. Click **Configurator** to go to the Configurator page, which has sections for defaults and the various vendor platforms.
3. To add a section for a platform, model, or serial number, click **Additional SN/Platform/Model**. To delete a section for a platform, model, or serial number, click **Delete SN/Platform/Model**. All sections (the ones that come preloaded and the ones you add) start with all of the possible keys for the `ztp.json` file. To improve visibility and readability, you can toggle to show only the fields that have been configured and hide the ones without data.
4. Enter the relevant values for defaults, platforms, models, and serial numbers, as applicable, then click **Save**. See examples for various vendor platforms in the following sections.

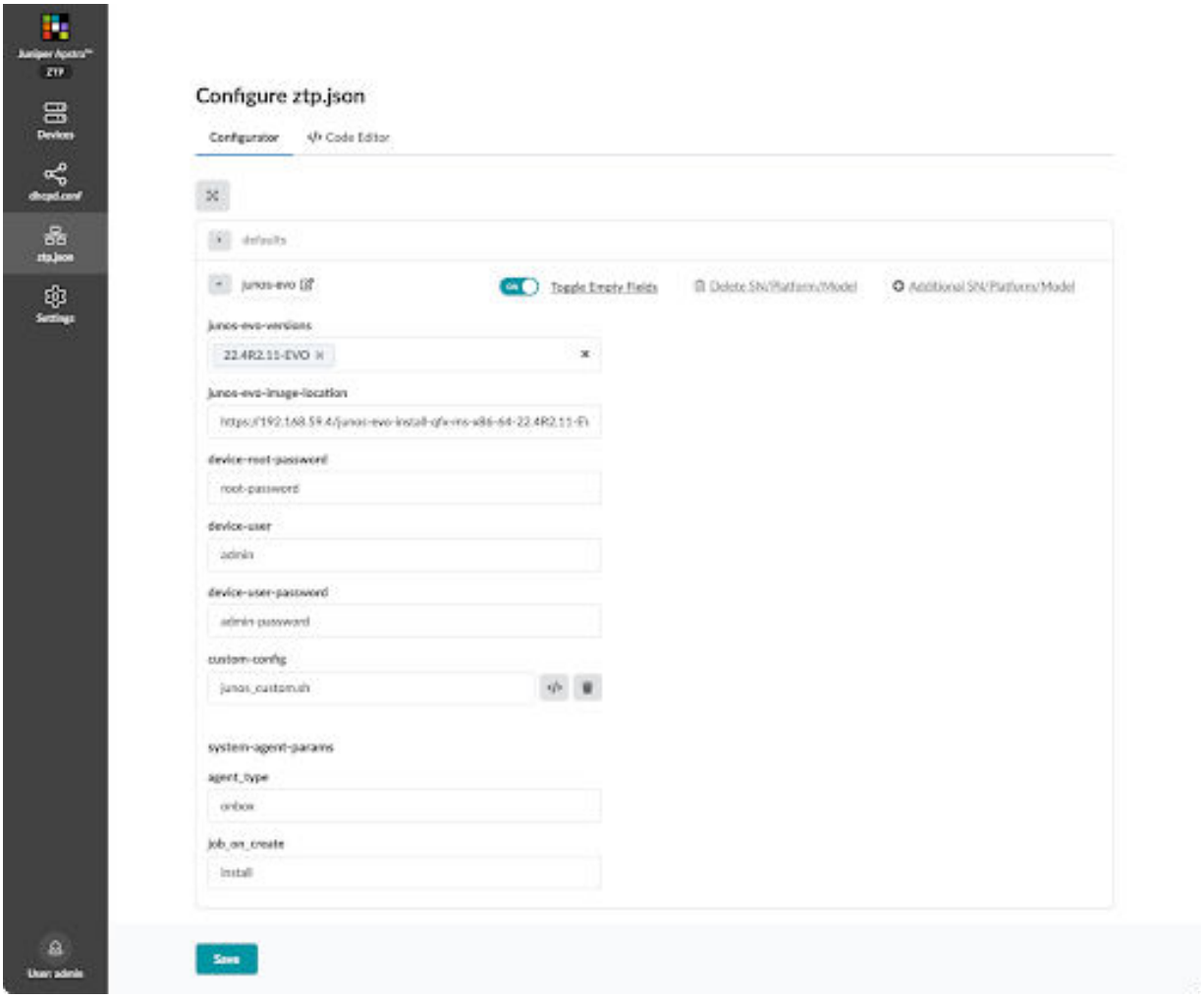
Juniper Junos Example

Offbox Agent / Apstra ZTP 4.2.0



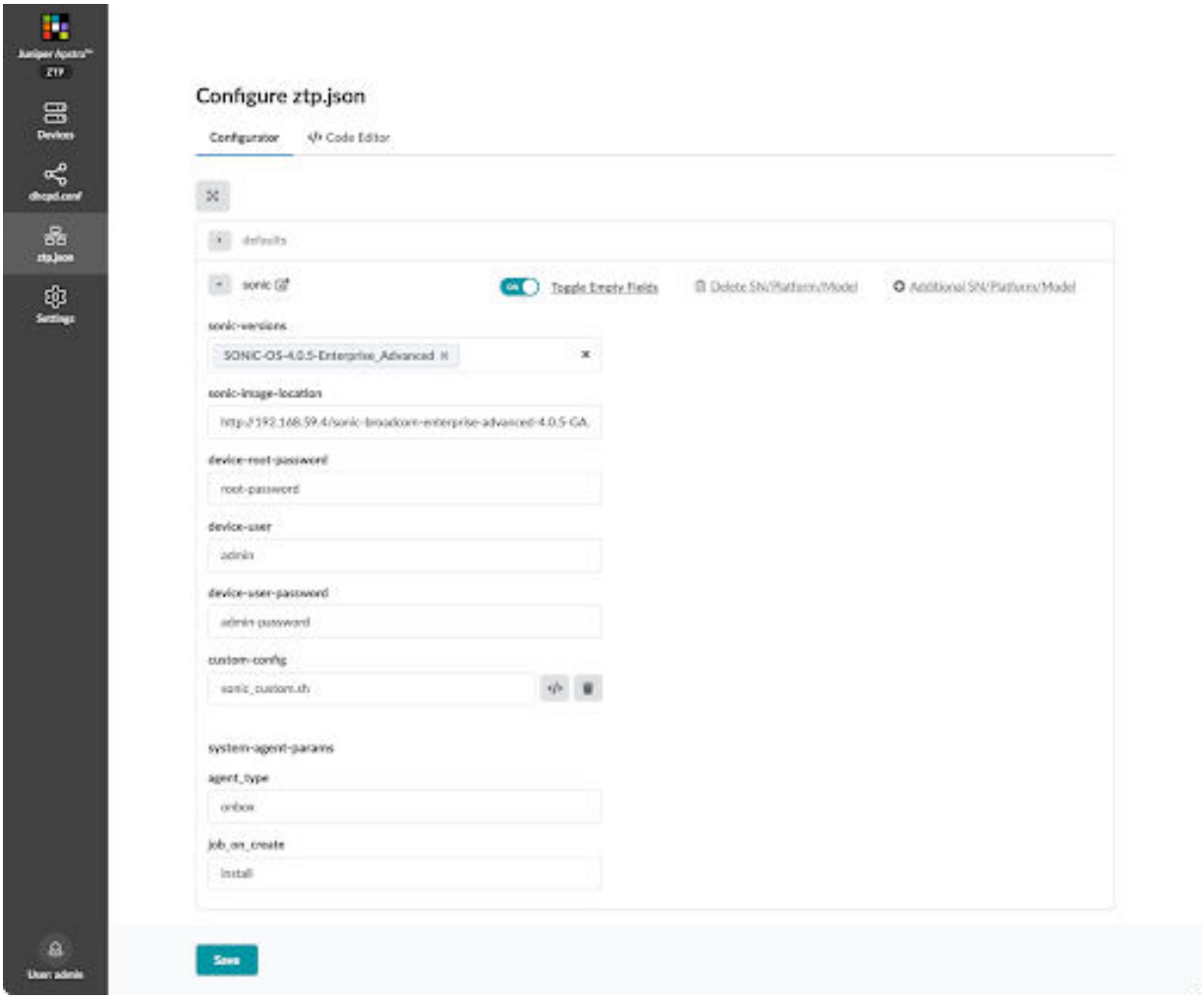
Juniper Junos Evolved Example

Onbox Agent / Apstra ZTP 4.2.0



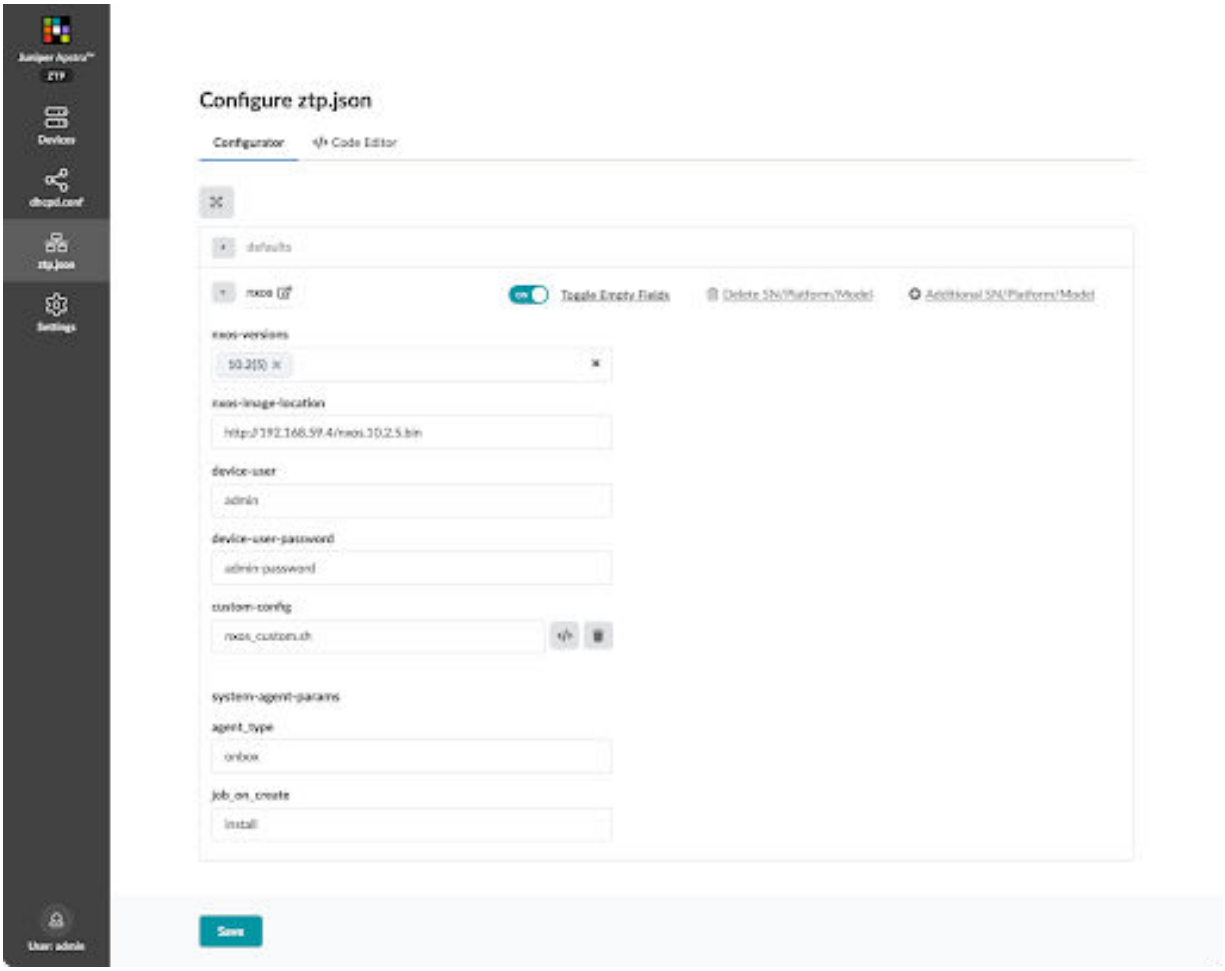
Enterprise SONiC Example

Onbox Agent / Apstra ZTP 4.2.0



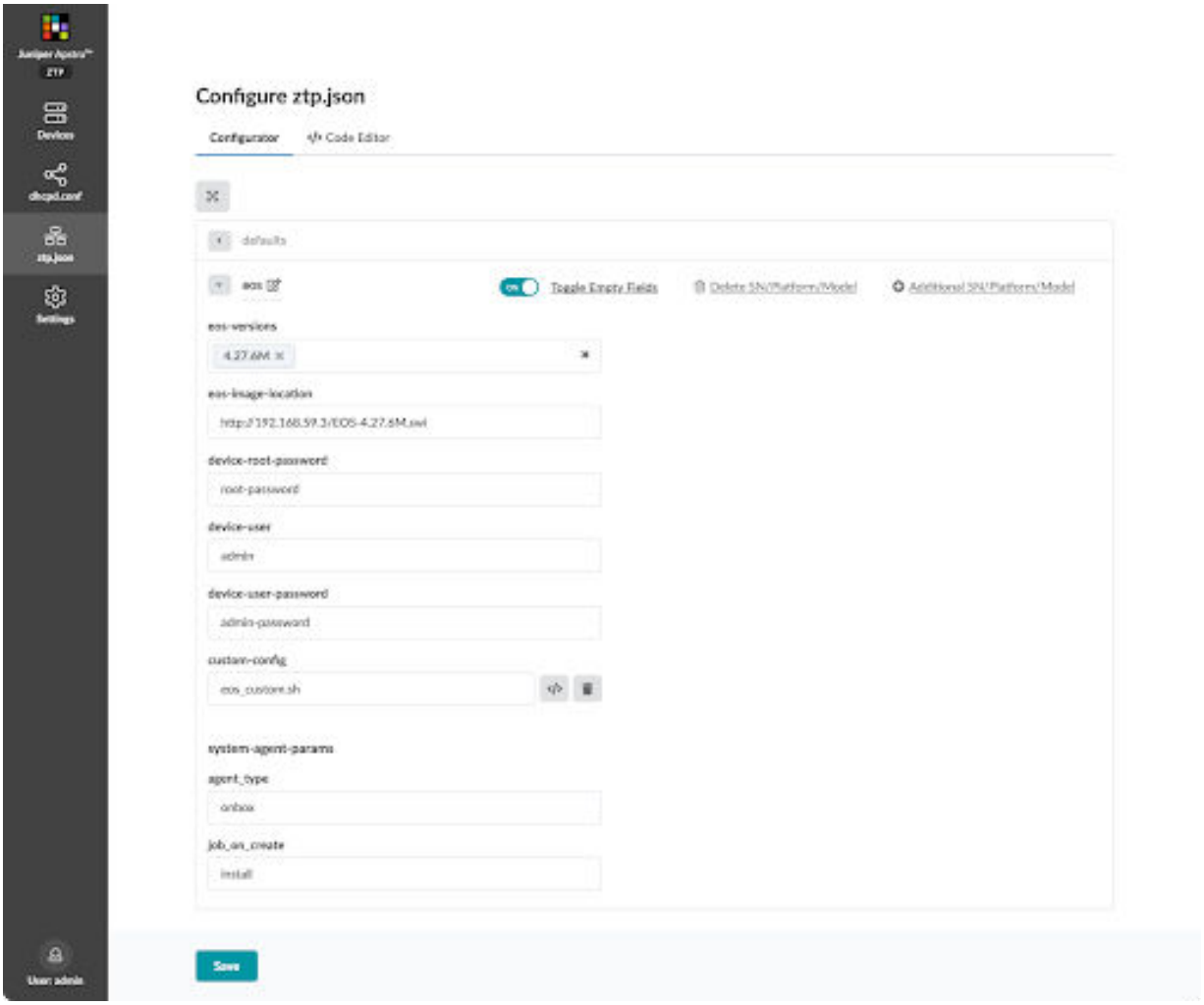
Cisco NX-OS Example

Onbox Agent / Apstra ZTP 4.2.0



Arista EOS Example

Onbox Agent / Apstra ZTP 4.2.0



RELATED DOCUMENTATION

- ztp.json Keys | 740

Configure ztp.json with CLI

SUMMARY

SSH in to the Apstra ZTP server and configure the ztp.json file.

IN THIS SECTION

- Configure ztp.json with CLI | 760

To lessen the chance of errors, we recommend using the Apstra ZTP GUI Configurator to configure `ztp.json`, but you have the option of using CLI instead, as described below. You can configure ZTP configuration directly with text editors such as vi or nano.

Configure ztp.json with CLI

1. Open a terminal and SSH into the Apstra ZTP server.

   `ssh admin@<apstra-ztp-server-ip>` where `<apstra-ztp-server-ip>` is the IP address of the Apstra ZTP server.

2. The ZTP configuration file is on the Apstra ZTP VM in the `/containers_data/tftp` directory.

```
admin@apstra-ztp:~$ sudo ls -l /containers_data/tftp
total 336
-rwxrwxrwx 1 admin admin   2448 Aug 28 16:39 config_verifier.py
-rwxrwxrwx 1 admin admin    742 Aug 28 16:39 container_init.sh
-rwxr-xr-x 1 admin admin    292 Sep 11 15:17 cumulus_slicer_custom.sh
-rwxrwxrwx 1 admin admin    178 Aug 28 16:39 Dockerfile
-rwxrwxrwx 1 admin admin    107 Aug 28 16:39 eos_custom.sh
-rwxr-xr-x 1 admin admin     75 Sep 11 15:17 eos_slicer_custom.sh
-rwxrwxrwx 1 admin admin   5735 Aug 28 16:39 junos_apstra_ztp_bootstrap.sh
-rwxrwxrwx 1 admin admin   1799 Aug 28 16:39 junos_custom.sh
-rwxr-xr-x 1 admin admin    564 Sep 11 15:17 junos_evo_slicer_custom.sh
-rwxr-xr-x 1 admin admin    517 Sep 11 15:17 junos_slicer_custom.sh
-rwxr-xr-x 1 admin admin    214 Sep 11 15:17 linux_slicer_custom.sh
-rwxrwxrwx 1 admin admin     86 Aug 28 16:39 nxos_custom.sh
-rwxr-xr-x 1 admin admin     78 Sep 11 15:17 nxos_slicer_custom.sh
-rwxrwxrwx 1 admin admin    205 Aug 28 16:39 poap-md5sum
-rw-rw-r-- 1 admin admin  26720 Sep 11 15:17 pxelinux.0
-rwxrwxrwx 1 admin admin   1843 Aug 28 16:39 rsyslog.conf
-rwxrwxrwx 1 admin admin    170 Aug 28 16:39 sonic_custom.sh
-rwxr-xr-x 1 admin admin     88 Sep 11 15:17 sonic_slicer_custom.sh
-rwxrwxrwx 1 admin admin   2640 Sep 11 15:17 ztp.json
-rwxrwxrwx 1 admin admin 115549 Aug 28 16:58 ztp.py
-rwxrwxrwx 1 root  root  115506 Aug 28 16:58 ztp.py.md5
```

3. Open the `ztp.json` file with a text editor, such as vi or nano.

```
admin@apstra-ztp:~$ sudo nano /containers_data/tftp/ztp.json
```

4. The default section includes default key-value pairs. The values defined here are applied to all devices, unless the same key is defined in more specific sections, such as for specific platforms,

models, or serial numbers. More specific keys take precedence over less specific keys. See "ztp.json Keys" on page 740 for key details.

```
{
   "defaults": {
      "nxos-image": "http://buildfiles.dc1.apstra.com/apstrktr/switch_images/cisco/
nxos.10.2.5.bin",
      "eos-image": "http://buildfiles.dc1.apstra.com/apstrktr/switch_images/arista/
EOS-4.27.6M.swi",
      "cumulus-image": "http://buildfiles.dc1.apstra.com/apstrktr/switch_images/cumulus/
cl-4.2.1-bcm-amd64.bin",
      "junos-image": "http://10.24.128.10/apstrktr/switch_images/juniper/
junos-5e-22.2R3.15.tgz",
      "junos-evo-image": "http://10.24.128.10/apstrktr/switch_images/juniper/junos-evo-
install-qfx-ms-fixed-x86-64>
      "sonic-image": "http://buildfiles.dc1.apstra.com/apstrktr/switch_images/sonic/
sonic-4.0.5-GA-adv-bcm.bin",
      "license": "http://onie.dc1.apstra.com/lic/cumulus.lic",
      "device-root-password": "admin",
      "device-user": "admin",
      "device-user-password": "admin"
   },
```

or this one?

```
{
  "defaults": {
    "nxos-versions": [
      "nxos-version1"
    ],
    "nxos-image": "aos_nxos_image.bin",
    "eos-versions": [
      "eos-version1",
      "eos-version2"
    ],
    "eos-image": "aos_eos_image.bin",
    "junos-versions": [
      "junos-version1",
      "junos-version2"
    ],
    "junos-image": "http://10.85.24.52/juniper/21.2R1-S2.2/jinstall-host-qfx-5e-x86-64-21.2R1-
```

```
S2.2-secure-signed.tgz",
    "junos-evo-versions": [
      "junos-evo-version1"
    ],
    "junos-evo-image": "http://server_address/path/to/install_package_name.tgz",
    "sonic-versions": [
      "sonic-version1",
      "sonic-version2"
    ],
    "sonic-image": "http://server_address/path/to/sonic_package_name.bin",
    "device-root-password": "admin",
    "device-user": "aosadmin",
    "device-user-password": "aosadmin",
    "custom-config": "",
    "management-subnet-prefixlen": "",
    "management-master-ip": "",
    "management-backup-ip": "",
    "management-ip": "",
    "management-gw-ip": "",
    "dual-routing-engine": false,
    "system-agent-params": {
      "platform": "",
      "agent_type": "onbox",
      "operation_mode": "",
      "profile": "",
      "install_requirements": false,
      "job_on_create": "",
      "force_package_install": false,
      "enable_monitor": false,
      "packages": [],
      "id": ""
    }
  },
  "junos": {
    "nxos-versions": [],
    "nxos-image": "",
    "eos-versions": [],
    "eos-image": "",
    "junos-versions": [
      "21.2R1-S2.2"
    ],
    "junos-image": "http://10.85.24.52/juniper/21.2R1-S2.2/jinstall-host-qfx-5e-x86-64-21.2R1-
S2.2-secure-signed.tgz",
```

```
    "junos-evo-versions": [],
    "junos-evo-image": "",
    "sonic-versions": [],
    "sonic-image": "",
    "device-root-password": "root123",
    "device-user": "aosadmin",
    "device-user-password": "aosadmin123",
    "custom-config": "",
    "management-subnet-prefixlen": "",
    "management-master-ip": "",
    "management-backup-ip": "",
    "management-ip": "",
    "management-gw-ip": "",
    "dual-routing-engine": false,
    "system-agent-params": {
      "platform": "junos",
      "agent_type": "offbox",
      "install_requirements": false,
      "job_on_create": "install",
      "force_package_install": false,
      "packages": []
    }
  },
  "junos-evo": {
    "nxos-versions": [],
    "nxos-image": "",
    "eos-versions": [],
    "eos-image": "",
    "junos-versions": [],
    "junos-image": "",
"junos-evo-versions": [],
    "junos-evo-image": "",
    "sonic-versions": [],
    "sonic-image": "",
    "device-root-password": "root123",
    "device-user": "",
    "device-user-password": "aosadmin123",
    "custom-config": "",
    "management-subnet-prefixlen": "",
    "management-master-ip": "",
    "management-backup-ip": "",
    "management-ip": "",
    "management-gw-ip": "",
```

```
    "dual-routing-engine": false,
    "system-agent-params": {
      "platform": "junos",
      "agent_type": "offbox",
      "operation_mode": "",
      "profile": "",
      "install_requirements": false,
      "job_on_create": "install",
      "force_package_install": false,
      "enable_monitor": false,
      "packages": [],
      "id": ""
    }
  },
  "eos": {
    "nxos-versions": [],
    "nxos-image": "",
    "eos-versions": [],
    "eos-image": "",
    "junos-versions": [],
    "junos-image": "",
    "junos-evo-versions": [],
    "junos-evo-image": "",
    "sonic-versions": [],
    "sonic-image": "",
    "device-root-password": "",
    "device-user": "",
    "device-user-password": "",
    "custom-config": "eos_custom.sh",
    "management-subnet-prefixlen": "",
    "management-master-ip": "",
    "management-backup-ip": "",
    "management-ip": "",
    "management-gw-ip": "",
    "dual-routing-engine": false,
    "system-agent-params": {
      "platform": "",
      "agent_type": "",
      "operation_mode": "",
      "profile": "",
      "install_requirements": false,
      "job_on_create": "",
      "force_package_install": false,
```

```
      "enable_monitor": false,
      "packages": [],
      "id": ""
    }
  },
  "nxos": {
    "nxos-versions": [],
    "nxos-image": "",
    "eos-versions": [],
    "eos-image": "",
    "junos-versions": [],
    "junos-image": "",
    "junos-evo-versions": [],
    "junos-evo-image": "",
    "sonic-versions": [],
    "sonic-image": "",
    "device-root-password": "admin123",
    "device-user": "",
    "device-user-password": "",
    "custom-config": "",
    "management-subnet-prefixlen": "",
    "management-master-ip": "",
    "management-backup-ip": "",
    "management-ip": "",
    "management-gw-ip": "",
    "dual-routing-engine": false,
    "system-agent-params": {
      "platform": "",
      "agent_type": "onbox",
      "operation_mode": "",
      "profile": "",
      "install_requirements": false,
      "job_on_create": "",
      "force_package_install": false,
      "enable_monitor": false,
      "packages": [],
      "id": ""
    }
  },
  "JNP10002-60C [QFX10002-60C]": {
    "nxos-versions": [],
    "nxos-image": "",
    "eos-versions": [],
```

```
    "eos-image": "",
    "junos-versions": [
      "21.2R1-S1.3"
    ],
    "junos-image": "http://10.85.24.52/juniper/21.2R1-S1.3/junos-vmhost-install-qfx-
x86-64-21.2R1-S1.3.tgz",
    "junos-evo-versions": [],
    "junos-evo-image": "",
    "sonic-versions": [],
    "sonic-image": "",
    "device-root-password": "",
    "device-user": "",
    "device-user-password": "",
    "custom-config": "",
    "management-subnet-prefixlen": "",
    "management-master-ip": "",
    "management-backup-ip": "",
    "management-ip": "",
    "management-gw-ip": "",
    "dual-routing-engine": false,
    "system-agent-params": {
      "platform": "",
      "agent_type": "",
      "operation_mode": "",
      "profile": "",
      "install_requirements": false,
      "job_on_create": "",
      "force_package_install": false,
      "enable_monitor": false,
      "packages": [],
      "id": ""
    }
  },
  "QFX10002-36Q": {
    "nxos-versions": [],
    "nxos-image": "",
    "eos-versions": [],
    "eos-image": "",
    "junos-versions": [
      "21.2R1-S2.2"
    ],
    "junos-image": "http://10.85.24.52/juniper/21.2R1-S2.2/jinstall-host-qfx-10-f-
x86-64-21.2R1-S2.2-secure-signed.tgz",
```

```
    "junos-evo-versions": [],
    "junos-evo-image": "",
    "sonic-versions": [],
    "sonic-image": "",
    "device-root-password": "",
    "device-user": "",
    "device-user-password": "",
    "custom-config": "",
    "management-subnet-prefixlen": "",
    "management-master-ip": "",
    "management-backup-ip": "",
    "management-ip": "",
    "management-gw-ip": "",
    "dual-routing-engine": false,
    "system-agent-params": {
      "platform": "",
      "agent_type": "",
      "operation_mode": "",
      "profile": "",
      "install_requirements": false,
      "job_on_create": "",
      "force_package_install": false,
      "enable_monitor": false,
      "packages": [],
      "id": ""
    }
  }
}
```

5. The platform-specific section includes configuration that's applied to each device based on its platform. For information about custom-config files, see "Create Vendor-specific Custom Configuration " on page 724

```
  "junos": {
     "device-root-password": "root123",
     "custom-config": "junos_slicer_custom.sh"
  },
  "junos-evo": {
     "device-root-password": "root123",
     "custom-config": "junos_evo_slicer_custom.sh"
  },
  "eos": {
```

```
      "custom-config": "eos_slicer_custom.sh"
   },
   "nxos": {
      "custom-config": "nxos_slicer_custom.sh"
   },
   "cumulus": {
      "custom-config": "cumulus_slicer_custom.sh"
   },
   "sonic": {
      "custom-config": "sonic_slicer_custom.sh"
   },
   "linux": {
      "custom-config": "linux_slicer_custom.sh"
   }
}
```

6. Model-specific parameters include the following:
7. Serial number parameters include the following:

```
   "5254003765C9": {
      "device-root-password": "root123",
      "junos-versions": [
         "22.2R3.15"
      ],
      "junos-image": "",
      "system-agent-params": null
   },
   "525400EB0A3C": {
      "device-root-password": "root123",
      "junos-versions": [
         "22.2R3.15"
      ],
      "junos-image": "",
      "system-agent-params": null
   },
   "5254002FFA41": {
      "device-root-password": "root123",
      "junos-versions": [
         "22.2R3.15"
      ],
      "junos-image": "",
      "system-agent-params": null
```

```
    },
    "525400C36BF6": {
       "device-root-password": "root123",
       "junos-versions": [
          "22.2R3.15"
       ],
       "junos-image": "",
       "system-agent-params": null
    },
    "525400DE0AE4": {
       "device-root-password": "root123",
       "junos-versions": [
          "22.2R3.15"
       ],
       "junos-image": "",
       "system-agent-params": null
    },
    "525400B813C8": {
       "system-agent-params": null
    },
    "5254007F0AF8": {
       "system-agent-params": null
    },
    "5254008DDCCF": {
       "system-agent-params": null
    },
    "525400602A79": {
       "system-agent-params": null
    },
    "5254004D62B3": {
       "system-agent-params": null
    },
```


RELATED DOCUMENTATION

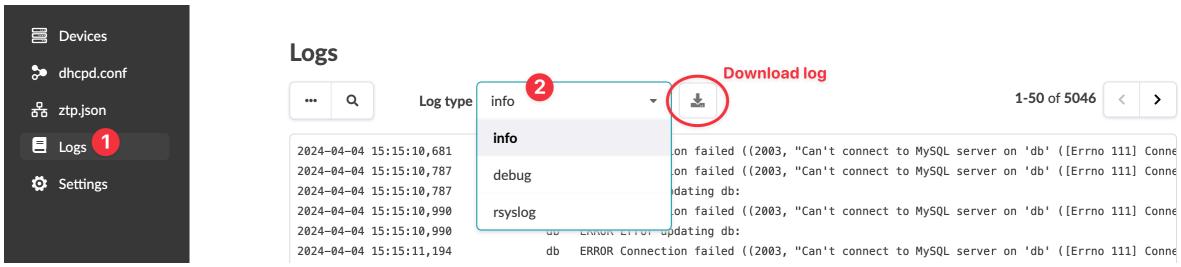
Configure ztp.json with Configurator | 754

ztp.json Keys | 740

Show Apstra ZTP Logs

To assist with troubleshooting, you can access various ztp logs directly from the Apstra ZTP GUI (as of Apstra version 5.0.0).

1. From the left navigation menu of the Apstra ZTP GUI, click **Logs**.



2. From the **Log type** drop-down list, select the log type to show: info, debug, rsyslog.
3. If you want to download the log, click the **Download log** button.

Onboard Devices with Apstra ZTP

IN THIS SECTION

Apstra ZTP manages the bootstrap and lifecycle of devices managed by Apstra.

Before onboarding devices, make sure that your devices are set to factory default. Different vendors have different methods for setting their devices back to factory default after having added configuration.

NOTE: To prevent being locked out of a device when there is a problem during the ZTP process, ZTP uses default, hard-coded credentials. These credentials are:

- root / admin
- aosadmin / aosadmin

Juniper Junos

EX switches require Junos OS version 21.2 or higher. EX switches using Junos OS versions below 21.1 are missing the Python module that's required for ZTP.

Juniper Devices Minimum Resource Requirements

Apstra ZTP uses a custom script to create offbox agents, create local users and set other system configuration. The ZTP process copies a new OS image to the switch. Before installing Apstra ZTP, ensure that the switch has sufficient disk space for the OS image.

```
root@leaf001-001-2> show system storage
Filesystem         Size  Used  Avail  Capacity   Mounted on
/dev/gpt/junos     6.0G  1.0G   4.5G       18%  /.mount
<...>
```

Juniper Junos Bootstrap File

Apstra ZTP uses a Python script to provision the device during ZTP. To allow the Python script (`ztp.py`) to run on a device that is not Junos OS Evolved, additional configuration is required. Use the `junos_apstra_ztp_bootstrap.sh` script to bootstrap Apstra ZTP on Junos. It downloads and runs the ZTP script.

Junos OS Evolved devices don't require this bootstrap; they run the Apstra ZTP python script (ztp.py) directly.

Restart Juniper Junos ZTP

To erase (zeroize) the device and restart Juniper Junos ZTP process:

```
root@leaf3> request system zeroize
```

Troubleshoot Juniper Junos ZTP

When in ZTP mode, the Juniper switch downloads the `ztp.py` and `ztp.json` files to the `/var/preserve/apstra` directory. For diagnostics, take note of the `/var/preserve/apstra/aosztp.log` file.

You can find additional useful messages in `/var/log/messages` (search for 'ztp').

Requirements for 4.2.0 [DOCS-1013]

1. In ztp.json , system-agent-params, If you need to provide “profile” parameters, you must use UUID instead of the profile name/label.
2. In ztp.json , system-agent-params, the following additional params are always visible in the ZTP server, however these will cause agent creation failure during the ZTP process.
   a. The parameters are force_package_install, install_requirements, enable_monitor
   b. These must be removed from the system-agent-params for agent creation to work via ZTP, howver due to a bug when these parameters are removed from the UI ztp.json file, they are no removed and configurator add them again. The only solution is to manually modify the ztp.json file by logging via SSH connection to ZTP server, and then restarting the tftp container.

Enterprise SONiC

Enterprise SONiC Devices Minimum Resource Requirements

NOTE: Apstra ZTP 4.2 used with Apstra version 4.2 has support for SONiC Enterprise Distribution devices. SONiC devices with earlier versions of Apstra ZTP, or the software, are not supported.

Apstra ZTP uses a custom script to create onbox agents, create local users and set other system configuration.

As part of the ZTP process a new OS image is copied to the switch. Before installing Apstra ZTP ensure that the switch has sufficient disk space for the OS image.

NOTE: If you're using ONIE to install Enterprise SONiC on a device, you must copy the image to the `/containers_data/tftp` directory and rename it to `onie-installer` or another ONIE download name (`onie-installer-x86_64-dell_z9100_c2538-r0` for example). When rebooting in ONIE, the device

searches for this file on the HTTP then TFTP server. If it doesn't find the file, then ZTP fails. Once ONIE SONiC installation successfully completes, the SONiC device starts ZTP automatically.

To restart the SONiC ZTP process, use the `sudo ztp enable` and `sudo ztp run` commands.

```
admin@sonic:~$ sudo ztp enable
admin@sonic:~$ sudo ztp run
ZTP will be restarted. You may lose switch data and connectivity, continue?[yes/NO] yes
admin@sonic:~$
```

Cisco NX-OS

Cisco NX-OS Devices Minimum Resource Requirements

Ensure that sufficient disk space is available on the switch. As part of the ZTP process a new OS image is copied to the switch. Before installing Apstra ZTP ensure that the switch has sufficient disk space for the OS image.

```
switch1# dir bootflash: | include free|total
1296171008 bytes free
3537219584 bytes total
```

Restart Cisco NX-OS ZTP

NOTE: If an agent is already installed on the device, before you restart the device ZTP process remove the agent either via the UI device agent installer or manually via the device CLI.

```
C9K-172-20-65-5# guestshell destroy

Remove remaining AOS data from system
Removing the guest-shell deletes most of the data left by AOS.  Some files are
still on the bootflash:/.aos folder.

C9K-172-20-65-5# delete bootflash:.aos no-prompt
```

To restart Cisco NX-OS ZTP process:

```
switch# write erase
switch# reload
```

Arista EOS

Arista EOS Devices Minimum Resource Requirements

NOTE: Apstra ZTP has limited support and known issues for virtual Arista EOS (vEOS) devices.

- ZTP EOS upgrades are not supported on vEOS devices. EOS versions for vEOS device must match `eos-versions` set in `ztp.json` file.
- ZTP Logging to the controller does not work for vEOS devices due to the lack of a device serial number. This will be addressed in a future version.

As part of the ZTP process, a new OS image is copied to the switch. Before installing Apstra ZTP ensure that the switch has sufficient disk space for the OS image.

```
switch1#dir flash:
Directory of flash:/

<...>

3957878784 bytes total (3074723840 bytes free)
```

Restart Arista EOS ZTP

CAUTION: If an agent is already installed on the device, before you restart the device ZTP process, remove the agent extension either via the UI Device Agent Installer or manually via the device CLI.

```
l2-virtual-001-leaf1#sho extensions
Name                                    Version/Release    Status    Extension
--------------------------------------- ------------------ --------- ---------
```

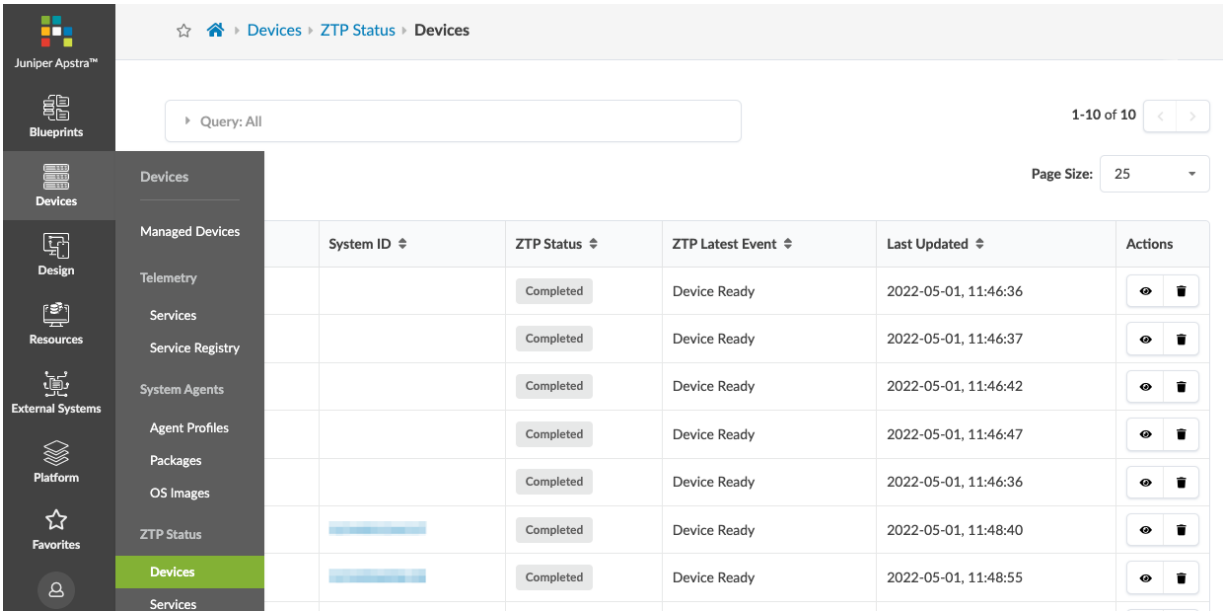
```
        aos-device-agent-3.1.0-0.1.205.i386.rpm   3.1.0/0.1.205      A, I      1


        A: available | NA: not available | I: installed | NI: not installed | F: forced
        l2-virtual-001-leaf1#delete extension:aos-device-agent-3.1.0-0.1.205.i386.rpm
        l2-virtual-001-leaf1#no extension aos-device-agent-3.1.0-0.1.205.i386.rpm
        l2-virtual-001-leaf1#copy installed-extensions boot-extensions
        Copy completed successfully.
        l2-virtual-001-leaf1#delete /recursive flash:aos*
        l2-virtual-001-leaf1#
```

To restart Arista EOS ZTP process:

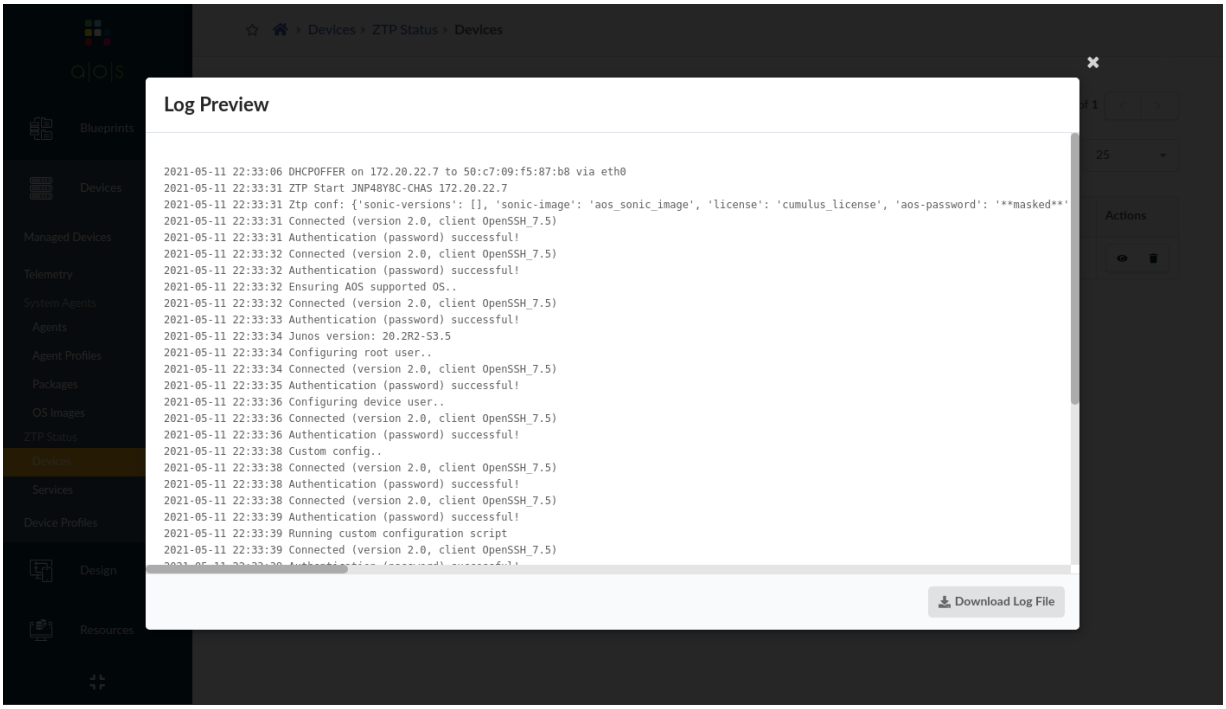
```
localhost# delete flash:zerotouch-config
localhost# write erase
Proceed with erasing startup configuration? [confirm]y
localhost# reload
```

Monitor Onboarding Status

When executed, the ZTP script sends logs to the Apstra server via API. You can monitor the ZTP process from the Apstra GUI. From the left navigation menu, navigate to **Devices > ZTP Status > Devices**.



Each device that's interacting with DHCP and ZTP is listed here along with its System ID (serial number) if known, ZTP Status, ZTP Latest Event and the date and time the device status was last updated. To see the full DHCP and ZTP log for the device, click the "Show Log" button (the eye in the **Actions** panel).



You can download the log file. If you don't need the logs for a device anymore, click the **Delete** button. Log files for all processes are retained in the `/containers_data/logs` directory.

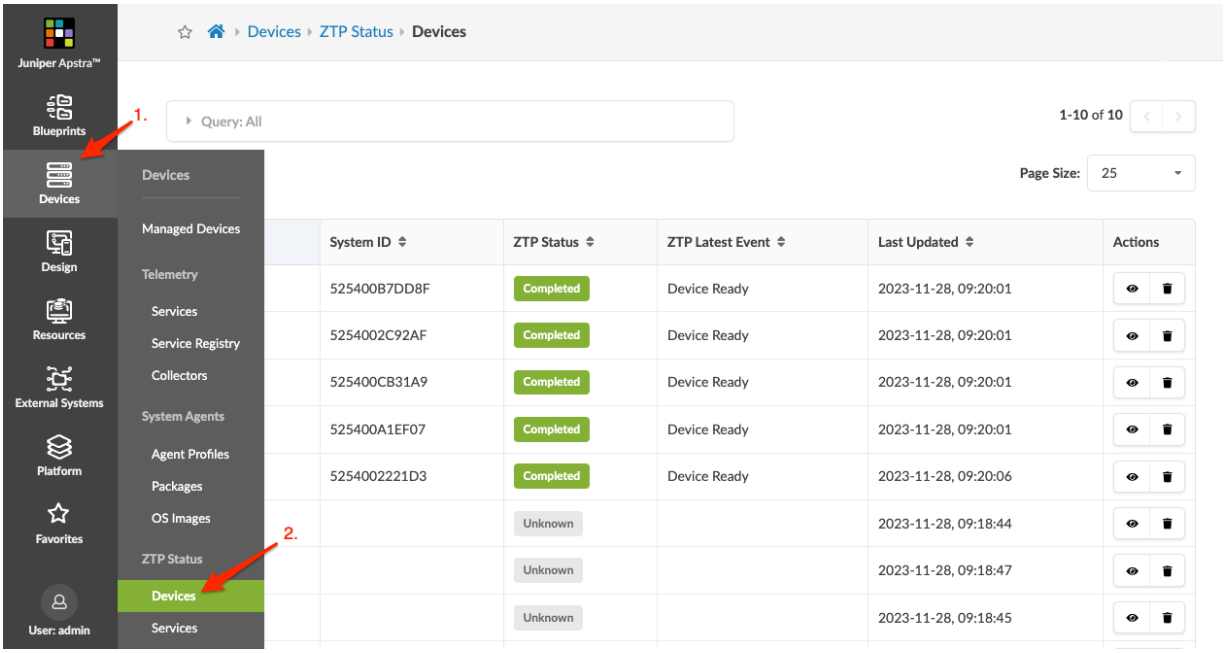
When the ZTP process successfully onboards a device it's included in the Managed Devices page, ready to be acknowledged and assigned to a blueprint. Navigate to **Devices > Managed Devices** to see available devices.

Check ZTP Status of Devices and Services

IN THIS SECTION

Devices

To check the ZTP status of devices, from the left navigation menu of the Apstra GUI, navigate to **Devices > ZTP Status > Devices**.



Each device interacting with DHCP and ZTP is listed along with its system ID (serial number), if known, ZTP status (where it is in the onboarding process), the latest ZTP event, and when the device status was last updated.

To see the full DHCP and ZTP log file for a device, click the **Show Log** button (eye icon) in the **Actions** panel.

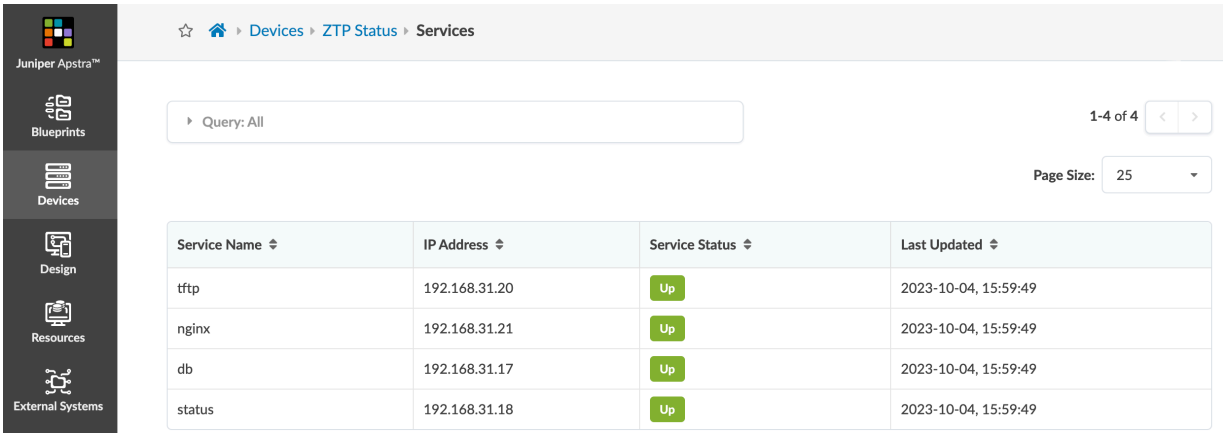
If you don't need to see the logs for a device anymore in the Apstra GUI, click the **Delete** button in the **Actions** panel. Log files for all processes are retained in the `/containers_data/logs` directory.

```
root@apstra-ztp:/containers_data/logs# ls -l
total 7132
-rw-r--r-- 1 root root 6351759 Oct 28 17:47 debug.log
drwxr-xr-x 2 root root    4096 Oct 27 19:20 devices
-rw------- 1 root root       0 Oct 23 20:02 dhcpd.leases
-rw-r--r-- 1 root root  926980 Oct 28 17:39 info.log
-rw------- 1 root root      58 Oct 23 20:02 README
-rw------- 1 root root     469 Oct 27 02:13 rsyslog.log
root@apstra-ztp:/containers_data/logs# tail info.log
2020-10-28 17:16:38,786     root.status    INFO Incoming: dhcpd dhcpd[18]: DHCPACK on
```

```
192.168.59.9 to 04:f8:f8:6b:36:91 via eth0
2020-10-28 17:18:04,299     root.status    INFO Incoming: dhcpd dhcpd[18]: DHCPREQUEST for
192.168.59.9 from 04:f8:f8:6b:36:91 via eth0
2020-10-28 17:18:04,300     root.status    INFO Incoming: dhcpd dhcpd[18]: DHCPACK on
192.168.59.9 to 04:f8:f8:6b:36:91 via eth0
2020-10-28 17:19:29,250     root.status    INFO Incoming: dhcpd : -- MARK --
2020-10-28 17:19:29,442     root.status   ERROR Failed to update status of all
containers: /api/ztp/service 404 b'{"errors":"Resource not found"}'
2020-10-28 17:33:29,353     root.status    INFO Incoming: tftp : -- MARK --
2020-10-28 17:33:29,538     root.status   ERROR Failed to update status of all
containers: /api/ztp/service 404 b'{"errors":"Resource not found"}'
2020-10-28 17:33:34,768     root.status    INFO Incoming: status : -- MARK --
2020-10-28 17:39:29,349     root.status    INFO Incoming: dhcpd : -- MARK --
2020-10-28 17:39:29,539     root.status   ERROR Failed to update status of all
containers: /api/ztp/service 404 b'{"errors":"Resource not found"}'
root@apstra-ztp:/containers_data/logs#
```

Services

To check the ZTP status of services, from the left navigation menu of the Apstra GUI, navigate to **Devices > ZTP Status > Services**.



Each service name includes its Docker IP address, service status and when the service status was last updated.

Reset Apstra ZTP GUI Admin Password

When you reset the Apstra ZTP GUI password, the configured GUI admin password and the Apstra server configurations are erased; the dhcp.conf and ztp.json files do not change.

1. SSH into the Apstra ZTP server as user **admin** (ssh admin@<apstra-ztp-server-ip> where <apstra-ztp-server-ip> is the IP address of the Apstra ZTP server.)
2. Run the command `docker exec -it status /scripts/reset_ztp_user.py` as shown in the example below:

```
admin@apstra-ztp:~$ docker exec -it status /scripts/reset_ztp_user.py
SUCCESS: User table is reset to init state
admin@apstra-ztp:~$
```

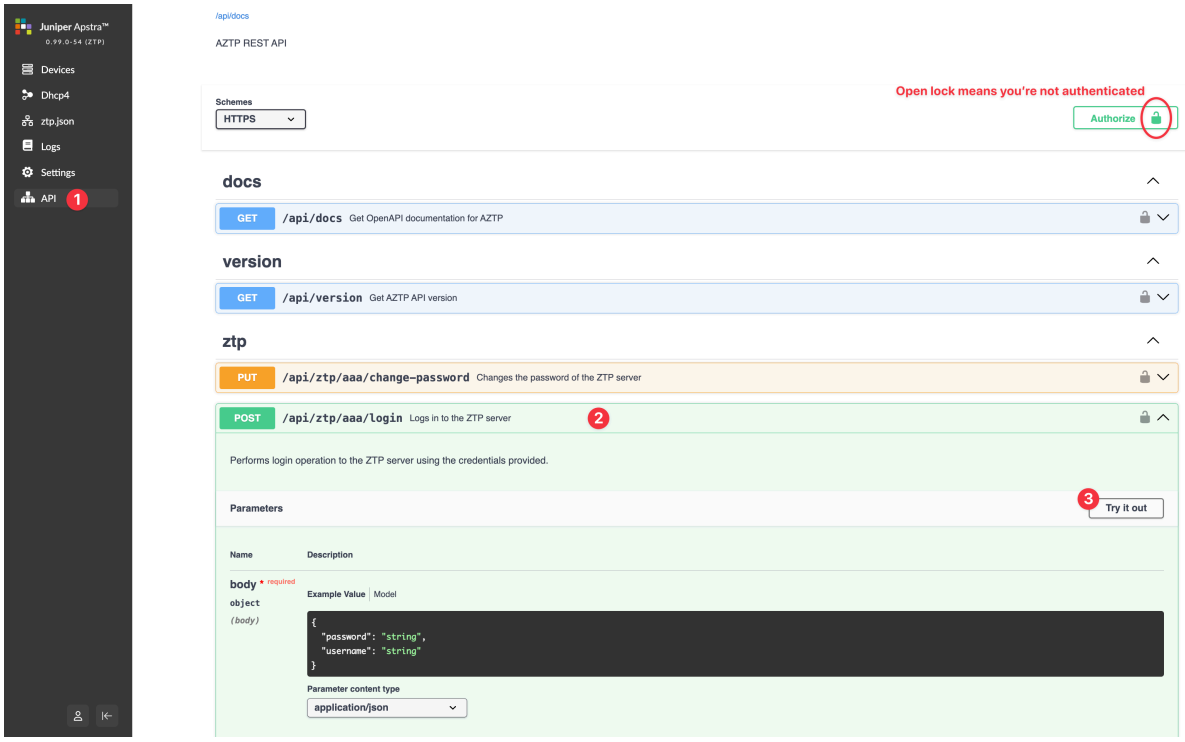
3. Log in to the Apstra ZTP GUI (default password: **admin**).
   You're immediately asked to change the default password.
4. Enter the default password, enter a secure password that meets complexity requirements, then re-enter the new password.
5. Click **Change** to change the password.

Next time you log in, you're presented with the 3-step wizard as if it's the first time you're logging in. If you've previously completed configuration, just click **Skip Wizard** (upper-right of screen).

Authenticate User (AZTP REST API)

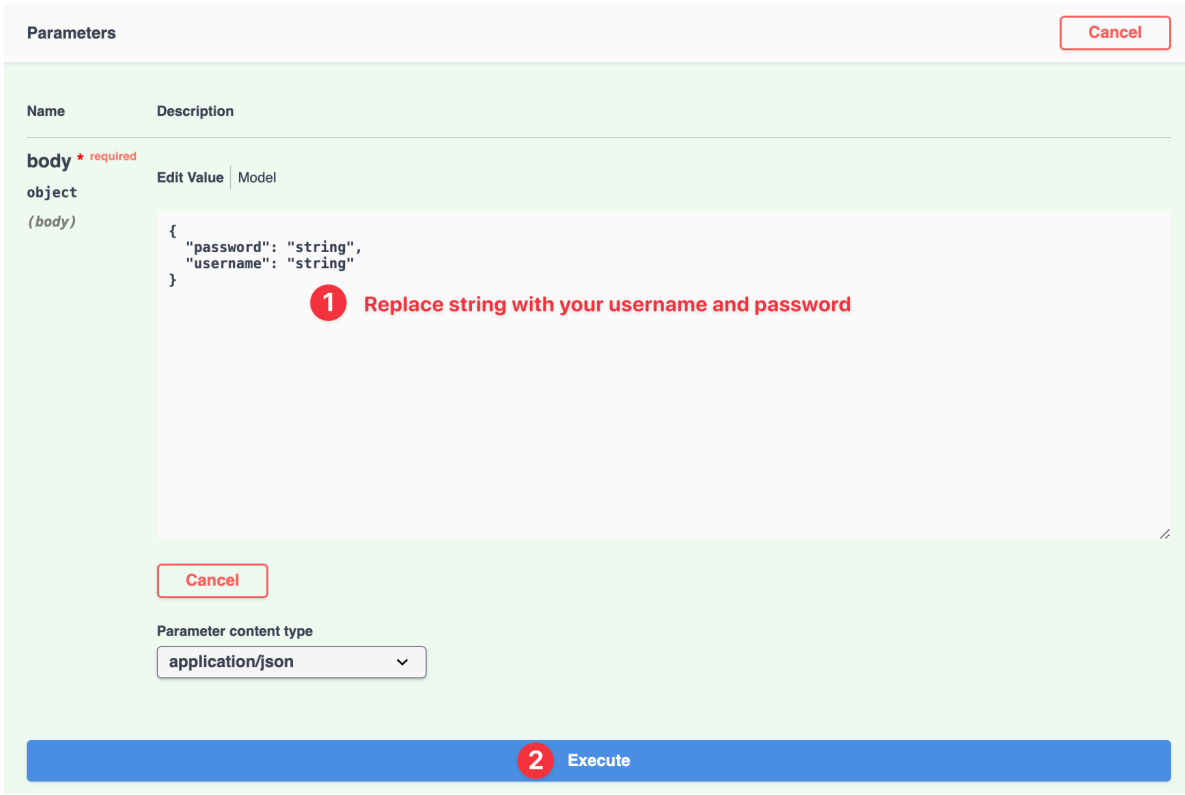
Let's get you authenticated so you can make API calls to the Apstra ZTP server from the Apstra ZTP GUI (new in Apstra version 5.0.0).

1. From the left navigation menu of the Apstra ZTP GUI, click **API**, click to expand **POST /api/ztp/aaa/login** (in the **ztp** section), then click **Try it out**.



The parameters become editable.

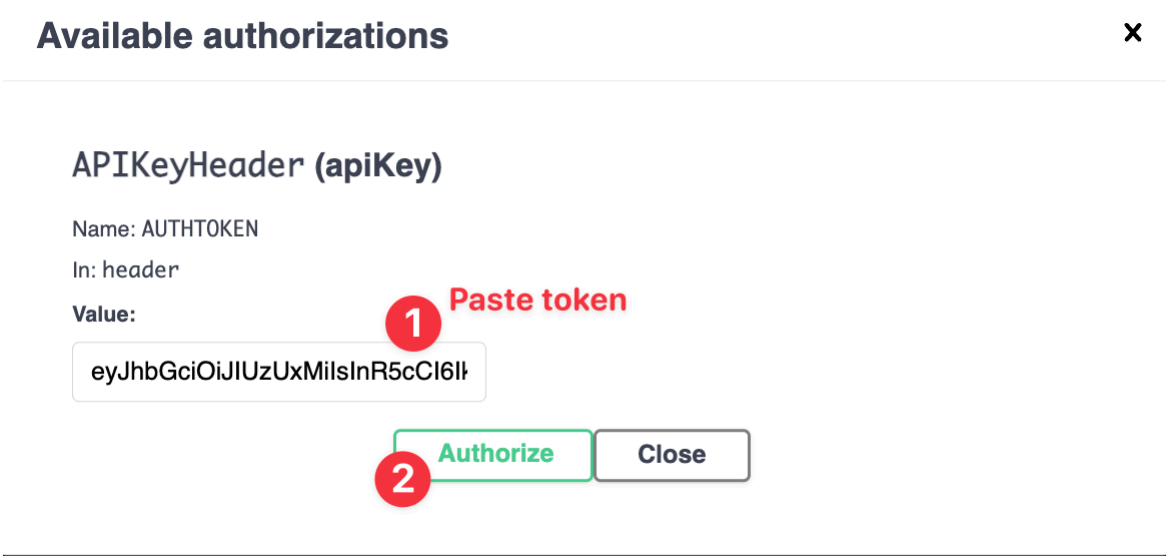
2. Replace the username and password strings with your own, then click **Execute**.



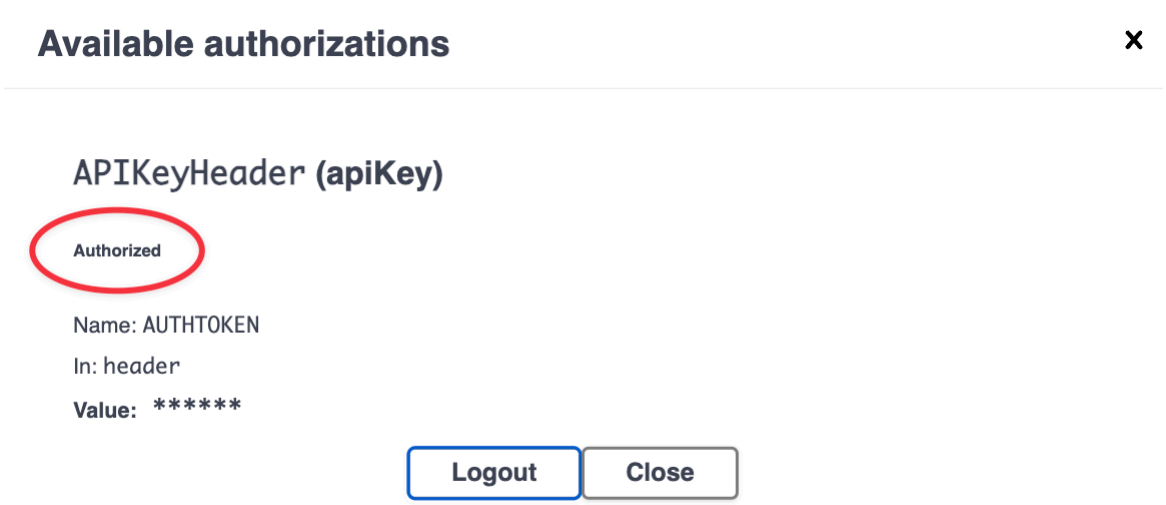
A token is added to the **Response body** of the **Server response** section.

3. Copy the token (without quotation marks).

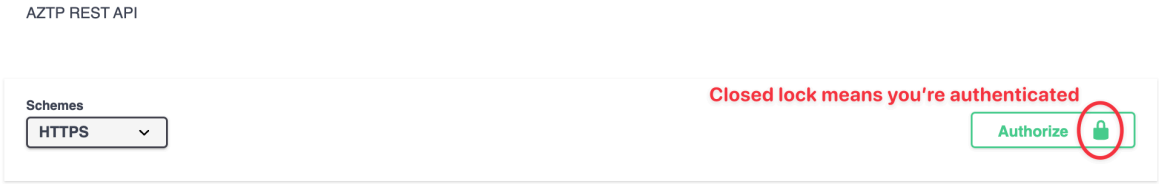
4. Scroll to the top of the page and click **Authorize** (top-right, shown in the first step).
   The **Authorize** dialog appears.
5. Paste the token and click **Authorize**.



   The dialog shows that you've been authorized.
6. Click the **X** in the dialog (top-right) or click **Close** to close the dialog.



The lock on the **Authorize** button changes to a closed lock, indicating that you're authenticated.



You're ready to make API calls. See the **docs** section for the AZTP OpenAPI documentation and the **ztp** section for available API calls.

CHAPTER 24

Device Profiles

IN THIS CHAPTER

What are Device Profiles

IN THIS SECTION

Device profiles define capabilities of supported hardware devices. Some feature capabilities have different behaviors across NOS versions and thus, capabilities are expressed per NOS version. By default, the version matches all supported versions. As additional hardware models are qualified, they are added to the "list of qualified devices" on page 1503.

Device profiles (vendor-specific details of devices) are associated with logical devices (abstractions of physical details of devices) to create "interface maps" on page 853. When you build your blueprint you'll assign these interface maps to devices in your topology.

Device profiles are either monolithic or modular. The following sections describe device profile parameters. For additional information about device profiles, see Adding Device Profiles Using the Apstra UI.

Monolithic Device Profiles

Summary

Table 13: Monolithic Device Profile Summary

Summary Section	Description
Type	Monolithic
Physical Device	Yes or No
Name	Name of device profile. 64 characters or fewer.
Ref Design Capabilities	Datacenter and/or Freeform
Chassis Profile	For modular device profiles only

Selector

The Selector section contains device-specific information to match the hardware device to the device profile as described below:

Table 14: Device Profile Selector

Selector Section	Description
Manufacturer	Selected from drop-down list
Model	Determines whether a device profile can be applied to specific hardware. Selected from drop-down list or entered as a regular expression (regex).
OS family	Defines how configuration is generated, how telemetry commands are rendered, and how configuration is deployed on a device. Selected from drop-down list.

Table 14: Device Profile Selector *(Continued)*

Selector Section	Description
Version	Determines whether a device profile can be applied to specific hardware. Selected from drop-down list or entered as regex.

Capabilities

You can leverage the hardware and software capabilities defined in this section in other parts of the Apstra environment to adapt the generated configuration, or to prevent an incompatible situation. With the exception of ECMP, hardware capabilities modify configuration rendering or deployment. Capabilities include the following details:

Table 15: Device Profile Capabilities

Capabilities Section	Description
CPU (cpu:string)	Describes the CPU architecture of the device. For example: "x86"
Userland (bits) (userland:integer)	Type of userland (application binary/kernel) the device supports. For example: "32" or "64".
RAM (GB) (ram:integer)	Amount of memory on the device. For example: "16"
ECMP limit (ecmp_limit:integer)	Maximum number of Equal Cost Multi Path routes. For example: "64". This field changes BGP configuration on the device (ecmp max-paths).
Form factor (form_factor:string)	Number of rack units (RU)s on the device. For example: "1RU", "2RU", "6RU", "7RU", "11RU","13RU"
ASIC (asic:string)	The switch chipset ASIC. For example: "T2", "T2(3)", "T2(6)", "Arad(3)", "Alta", "TH", "Spectrum", "XPliant XP80", "ASE2", "Jericho". Used to assist telemetry, configuration rendering and VXLAN routing semantics
LXC (lxc_support: boolean)	Selected if the device supports LXC containers.

Table 15: Device Profile Capabilities *(Continued)*

Capabilities Section	Description
ONIE (onie: boolean)	Selected if the device supports ONIE.

Supported Features (Cisco Only)

COPP - When Control Plane Policing is enabled (COPP), strict CoPP profile config is rendered for the specified NX-OS version resulting in the following configuration rendering:

```
terminal dont-ask
copp profile strict
```

This terminal dont-ask config is needed only when enabling the CoPP profile strict config, since we do not want NX-OS to wait for confirmation:

```
switch(config)# copp profile strict
This operation can cause disruption of control traffic. Proceed (y/n)? [no] ^C
switch(config)#
switch(config)# terminal dont-ask
switch(config)# copp profile strict
switch(config)#
```

CoPP is enabled by default, except for Cisco 3172PQ NXOS. You can specify multiple versions.

Breakout - Enable breakout to indicate that ports on specified modules can be broken out to lower speed split ports.

Apstra software first un-breakouts all ports that are breakout-capable, and then applies the proper breakout commands according to intent. This is based on the assumption that the global negation command `no interface breakout module<module_number>` can always be applied successfully to a module with breakout capable ports. (This is idempotent when applied on ports that are not broken out.) However, we recognize that this assumption may be broken in future versions of NX-OS, or with a certain combination of cables / transceivers inserted into breakout-capable ports.

The example below is for the negation command for a module (1) that is set to True:

```
no interface breakout module 1
!
```

Since the negation command is always applicable per module, each module is specified individually. The advantages of this include:

- In modular systems, not all line cards have breakout capable ports.
- In non-modular systems, the breakout capable ports may not always be in module 1.

Breakout is enabled by default except for the following devices with modules incapable of breaking out ports: 3172PQ NXOS, 9372TX NXOS, C9372PX NXOS, C9396PX NXOS, NXOSv.

Historical Context - With a particular version of NX-OS the POAP stage would apply breakout config on those ports which are breakout capable. POAP behavior, introduced in 7.0(3)I4(1) POAP, determines which breakout map (for example, 10gx4, 50gx2, 25gx4, or 10gx2) brings up the link connected to the DHCP server. If breakout is not supported on any of the ports, POAP skips the dynamic breakout process. After the breakout loop completes, POAP proceeds with the DHCP discovery phase as normal. Apstra reverts any such breakout config that might have been rendered during the POAP stage to ensure that the ports are put back to default speed by applying the negation command.

Sequence Numbers Support - Applicable to autonomous system (AS) path. Enable when the device supports sequence numbers. Apstra sequences into the entry list to resequence and generate config as follows:

```
ip as-path access-list MyASN seq 5 permit ^$
ip as-path access-list Rtr seq 5 permit ^3
ip as-path access-list Srvr seq 15 permit _103$
```

The numbers 5 and 15 are sequence numbers applicable to devices that support AS sequencing.

Sequence numbers support is enabled for all Cisco device profiles by default (except Cisco 3172PQ NXOS, which does not support sequence numbers). For platforms that do not support sequence numbers, disabling this feature ensures that the AS sequence numbers are removed from the device model dictionary to avoid addition and negation in the event that something is resequenced. This scenario has no requirement to render anything on these platforms, because the entry can't be sequenced.

Other supported features - not available from the Apstra GUI include "vxlan", "bfd", "vrf_limit", "vtep_limit", "floodlist_limit", "max_l2_mtu", and "max_l3-mtu". They can be included in the backend using the following format:

key : value :: feature : feature_properties Example: 32 vtep_limit: 32

Ports

The ports section defines the types of available ports, their capabilities and how they are organized.

Every port contains a collection of supported speed transformations. Each transformation represents the breakout capability (such as 1-40GBe port breaking out to 4-10GBe ports), and hence contains a collection of interfaces.

Example: If port 1 is a QSFP28 100->4x10, 100->1x40 breakout capable port, then port 1 has a collection of three transformations, one each for 4x10, 1x40 and 1x100 breakouts. The transformation element in the collection which represents the 4x10 has a collection of 4 interfaces, 1x40 and 1x100 has a collection of 1 interface.

Ports parameters include the following details:

Table 16: Device Profile Ports

Ports Section	Description
Port Index (port_id: integer)	Indicates a unique port in the collection of ports in the Device Profile.
Row Index (row_id: integer)	Represents the top-to-bottom dimensions of the port panel. Shows where the port is placed in the device's panel. For instance, in a panel with two rows and many columns the row index is either "1" or "2".
Column Index (column_id: integer)	Represents the left-to-right dimensions of the port panel. Shows where the port is placed in the device's panel. For instance, in a panel with thirty-two ports and two rows, the column index is in the range of "1" through "16".
Panel Index (panel_id: integer)	Indicates the panel that the port belongs to given the physical layout of ports in the device specification
Slot ID (slot_id: integer)	Represents the module that the port belongs to. A modular switch has more than one slot. In fixed function network function devices, Slot ID is usually "0".
Failure Domain (failure_domain_id: integer)	Indicates if multiple panels are relying on the same hardware components. Used when creating the cabling plan to ensure that two uplinks are not attached to the same failure domain.
Connector Type (connector_type: string)	Port transceiver type. Speed capabilities of the port are directly related to the connector type, given that certain connector types can run in certain speeds. For instance, "sfp", "sfp28", "qsfp", "qsfp28".

Table 16: Device Profile Ports *(Continued)*

Ports Section	Description
Transformations (transformations: list)	Possible breakouts for the port. Every entry is a specific supported speed. Each transformation has a collection of interfaces.
Number of interfaces (interfaces:list)	Dependent on the breakout capability of the port. For a transformation representing a certain breakout speed, the interfaces contain information about the interface names and interface settings with which the device intends to be configured. The "setting" information is crucial for configuring the interfaces correctly on the device.

Based on the OS information entered in the device profile's selector field, the Apstra GUI displays the applicable settings fields. The fields vary with the vendor OS (as found in examples below). When a device profile is created or edited, the "setting" is validated from the vendor-specific schema as listed below.:

```
eos_port_setting = Dict({
    'interface': Dict({
        'speed': Enum([
            '', '1000full', '10000full', '25gfull', '40gfull',
            '50gfull', '100gfull',
        ])}),
    'global': Dict({
        'port_group': Integer(),
        'select': String()
        })
    })

nxos_port_setting = Dict({
    'interface': Dict({
        'speed': Enum([
            '', '1000', '10000', '25000', '40000', '50000',
            '100000',
        ])}),
    'global': Dict({
        "port_index": Integer(),
        "speed": String(),
        "module_index": Integer()
        })
```

```
    })

junos_port_setting = Dict({
    'interface': Dict({
        'speed': Enum([
            '', 'disabled', '1g', '10g', '25g', '40g', '50g', '100g'
        ])}),
    'global': Dict({
        'speed': Enum([
            '', '1g', '10g', '25g', '40g', '50g', '100g'
        ]),
        "port_index": Optional(Integer()),
        "fpc": Optional(Integer()),
        "pic": Optional(Integer())
        })
    })

sonic_port_setting = Dict({
    'interface': Dict({
        "command": Optional(String()),
        "speed": String(),
        "lane_map": Optional(String())
        })
    })
})
```

Apstra does not necessarily use all the information above for modeling. It's made available to other Apstra API orchestration tools for collection and use.

Modular Device Profiles

Summary

Table 17: Modular Device Profile Summary

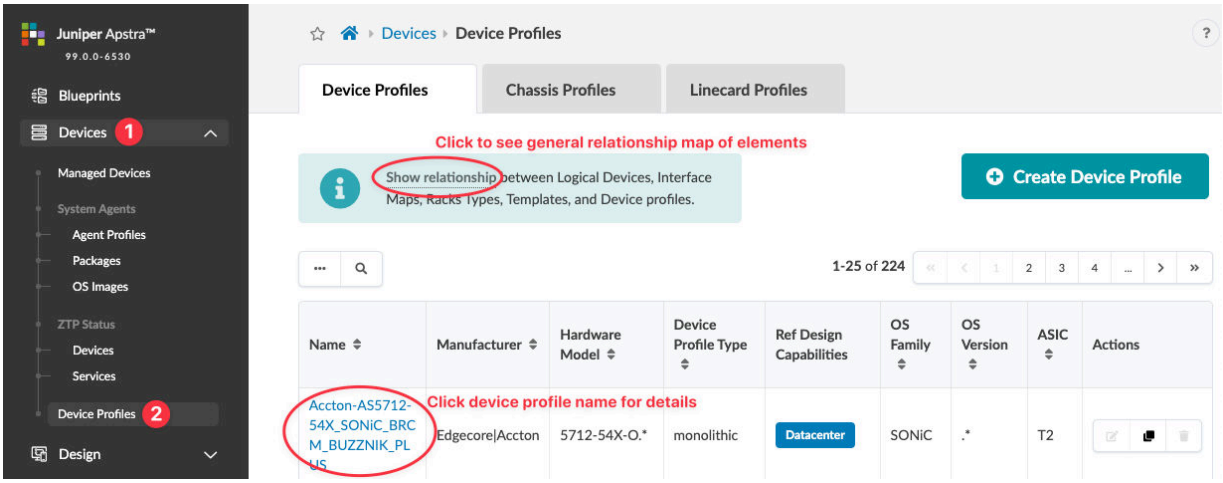
Type	Modular
Name	Name of device profile. 64 characters or fewer.

Table 17: Modular Device Profile Summary *(Continued)*

Chassis Profile	The chassis profile to associate with the DP

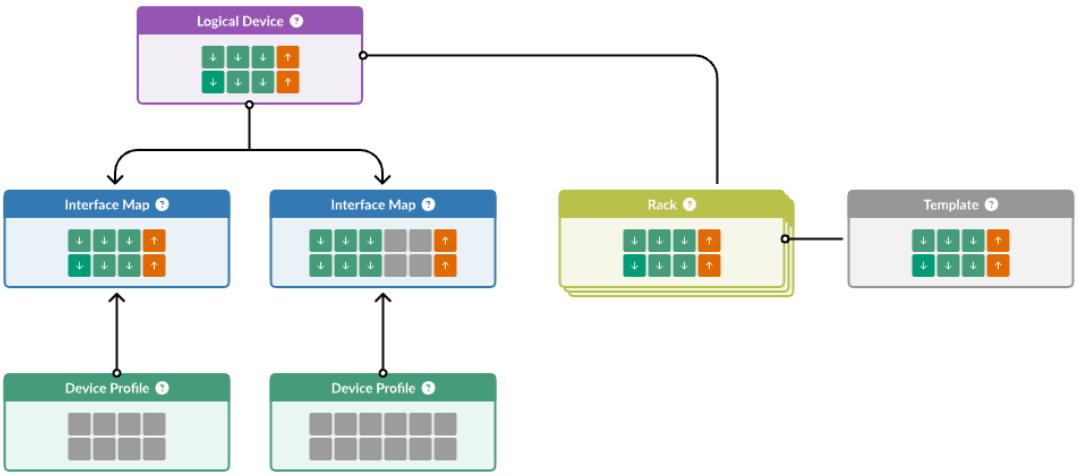
Device Profiles in the Apstra GUI

From the left navigation menu in the Apstra GUI, navigate to **Devices > Device Profiles** to go to the device profile table view.



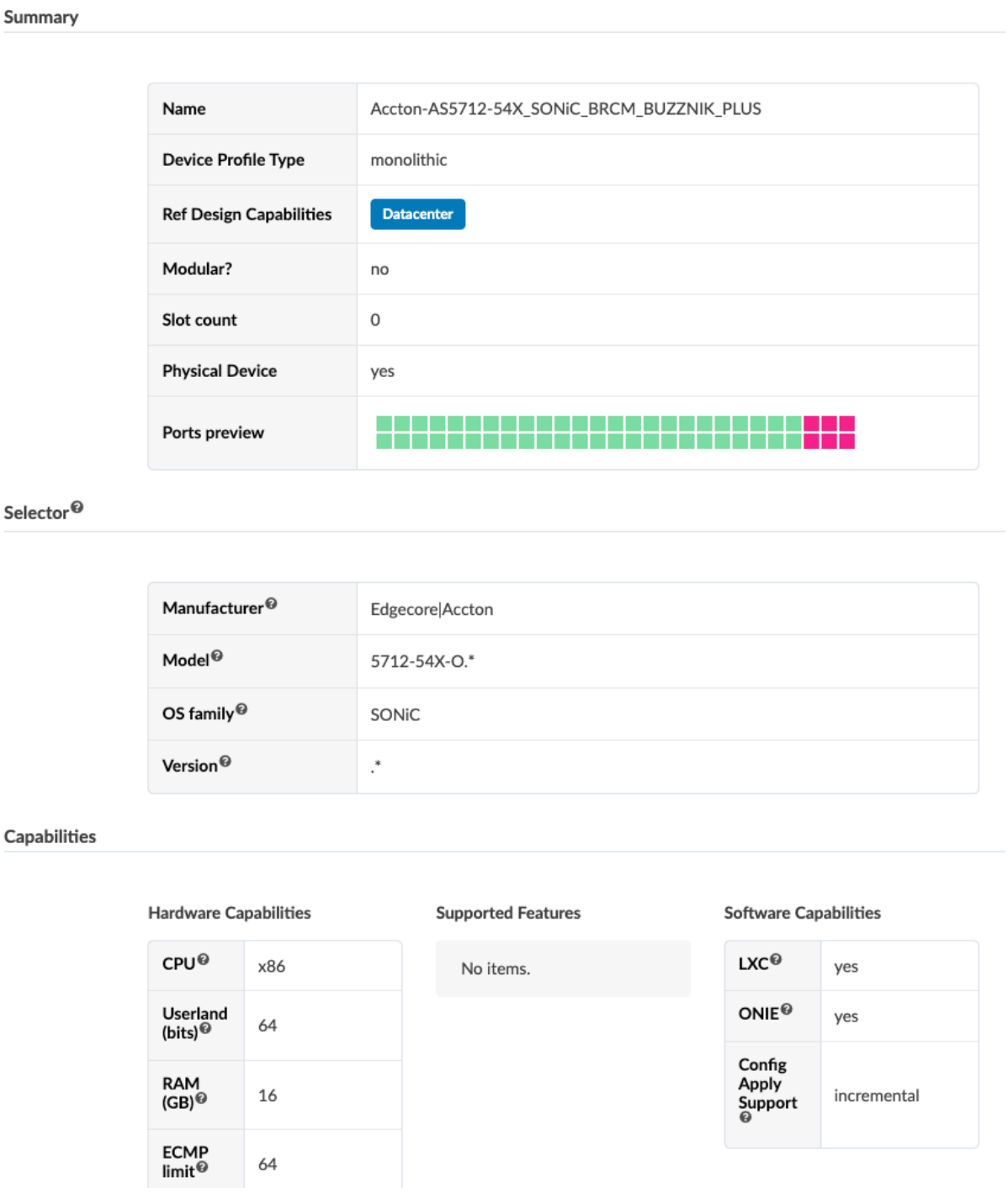
To see how Apstra device profiles and design elements are related to each other, click **Show relationship** (new in Apstra version 5.0.0). This is helpful if you're new to the Apstra environment.

Static Conceptual Graphic for Design Elements



Many device profiles are predefined for you. To search for a device profile, click the **Search** button (magnifying glass) and enter your criteria.

Click a device profile name to go to its details.

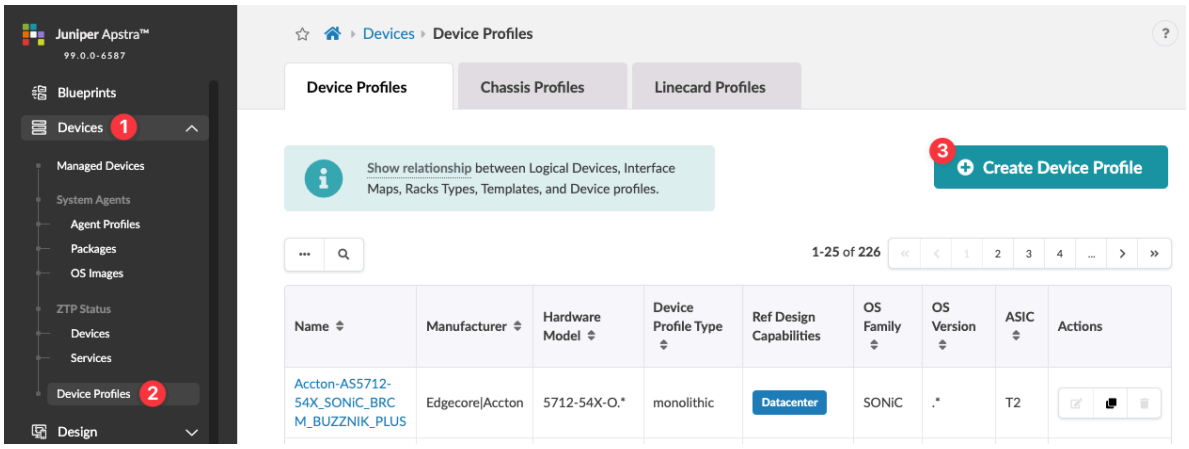


You can create, edit, and delete device profiles.

Create Monolithic Device Profile

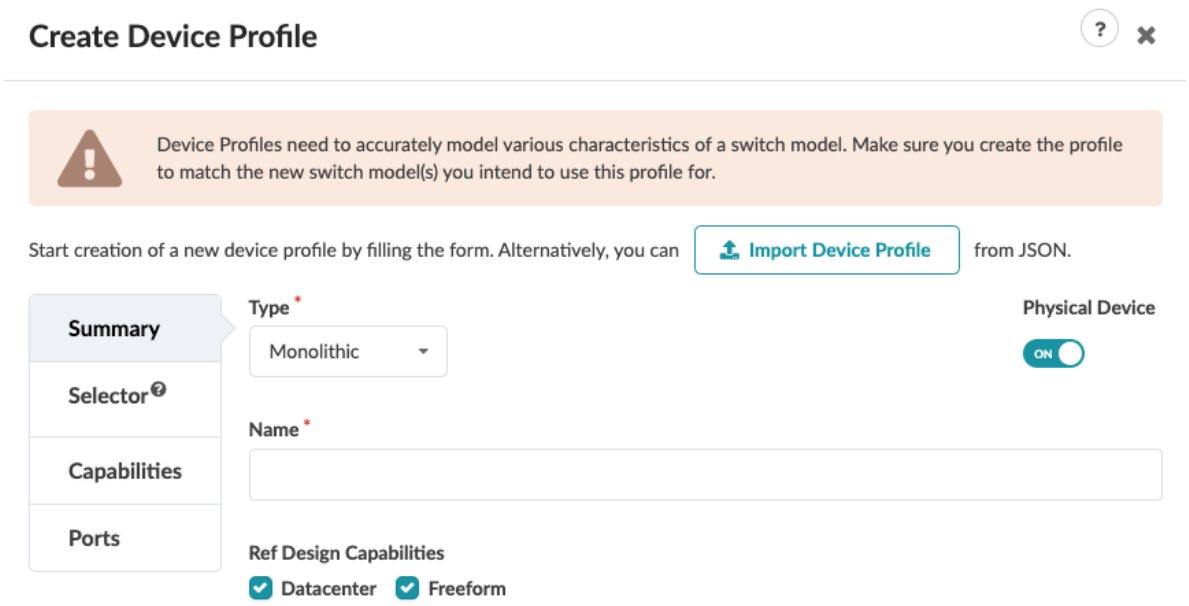
Device profiles contain extensive hardware model details. Make sure the profile accurately describes all hardware characteristics. For assistance, contact "Juniper Support" on page 1374.

1. From the left navigation menu, navigate to **Devices > Device Profiles** and click **Create Device Profile**.



The **Create Device Profile** dialog opens on the **Summary** section.

2. Enter the relevant information in the **Summary** section.



a. If you've created a JSON payload, click **Import Device Profile** and select the file to import it. Otherwise, continue to the next step.

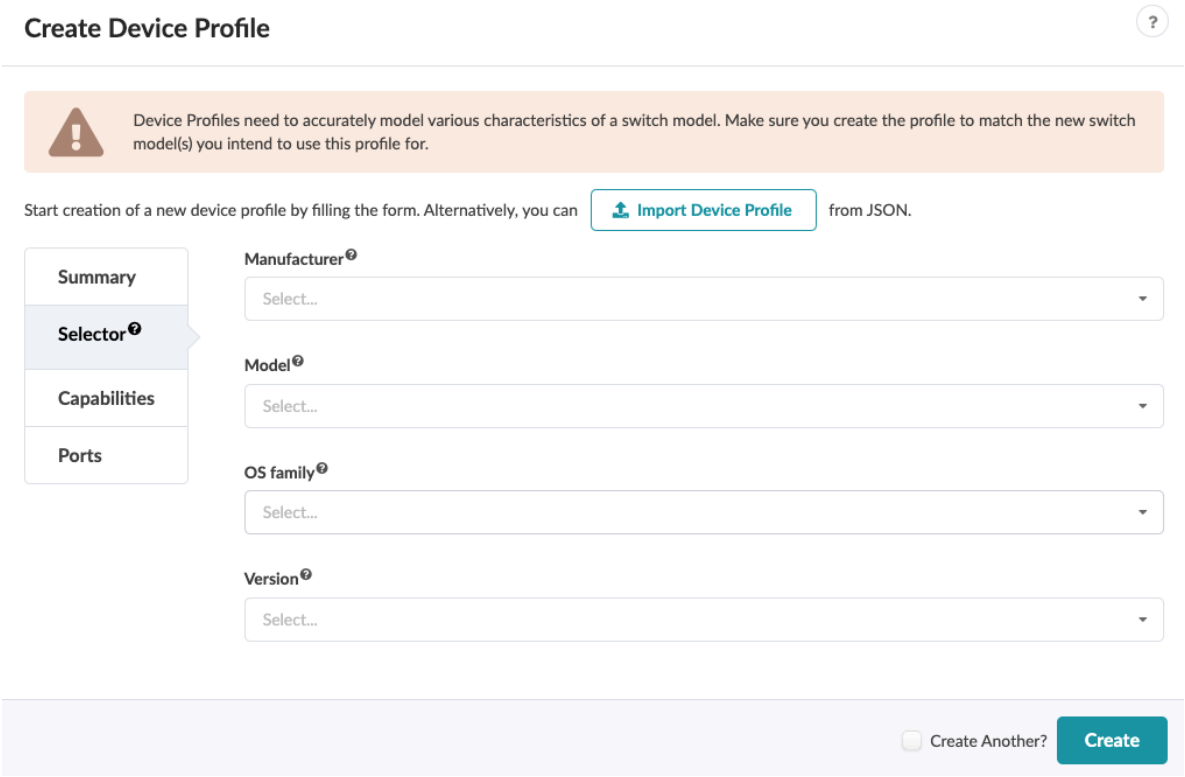
b. Leave the **Type** as Monolithic.

c. If you're not using a physical device, toggle off **Physical Device**.

d. Enter a unique device profile name.

e. Select the relevant check box(es) for **Ref Design Capabilities** (new in Apstra version 5.0.0).

3. Click **Selector** in the left panel.
   The **Selector** section of the **Create Device Profile** dialog opens.
4. Enter the relevant information in the **Selector** section.



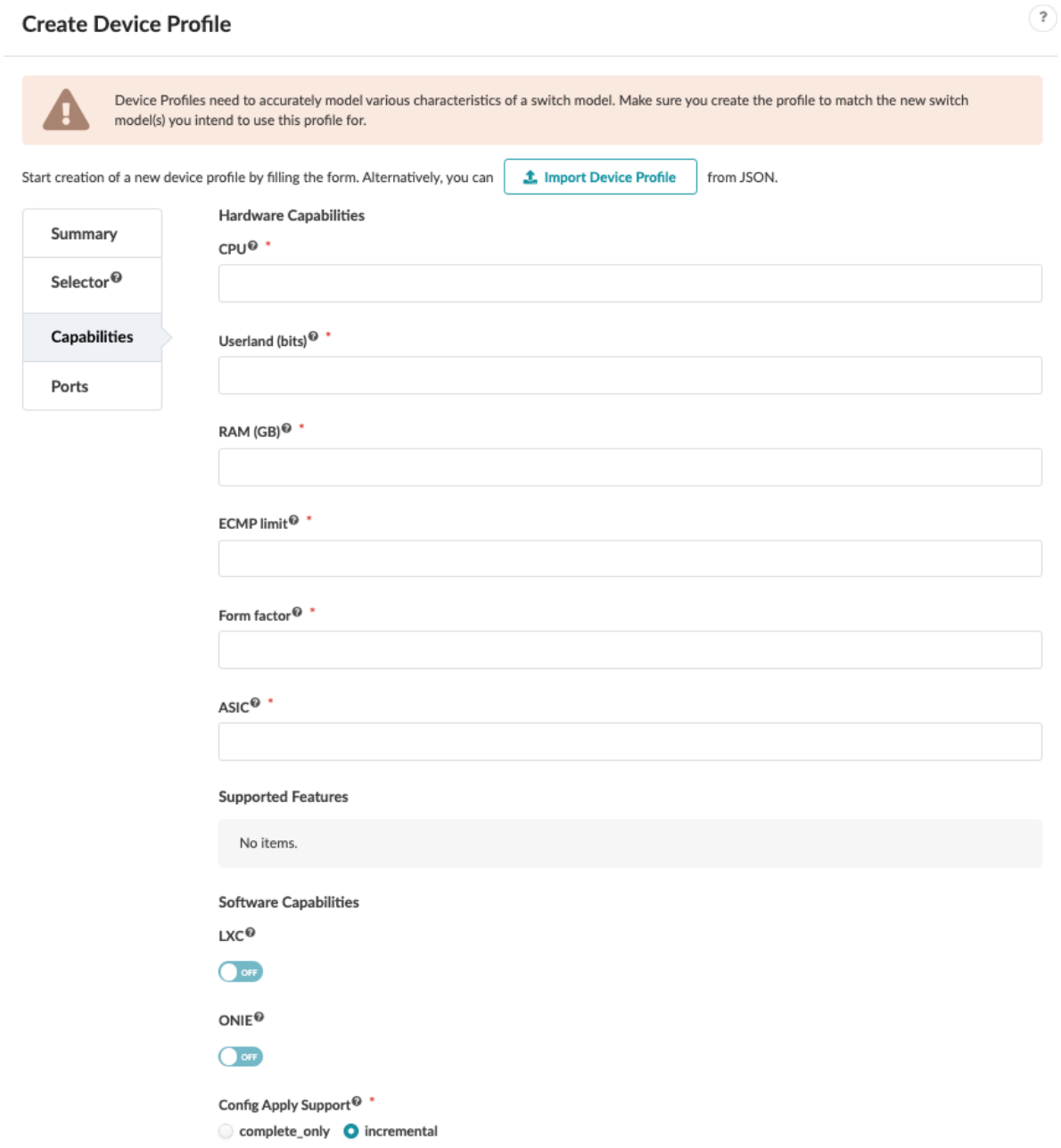
a. Select the manufacturer from the **Manufacturer** drop-down list.

b. Select the model number from the **Model** drop-down list.

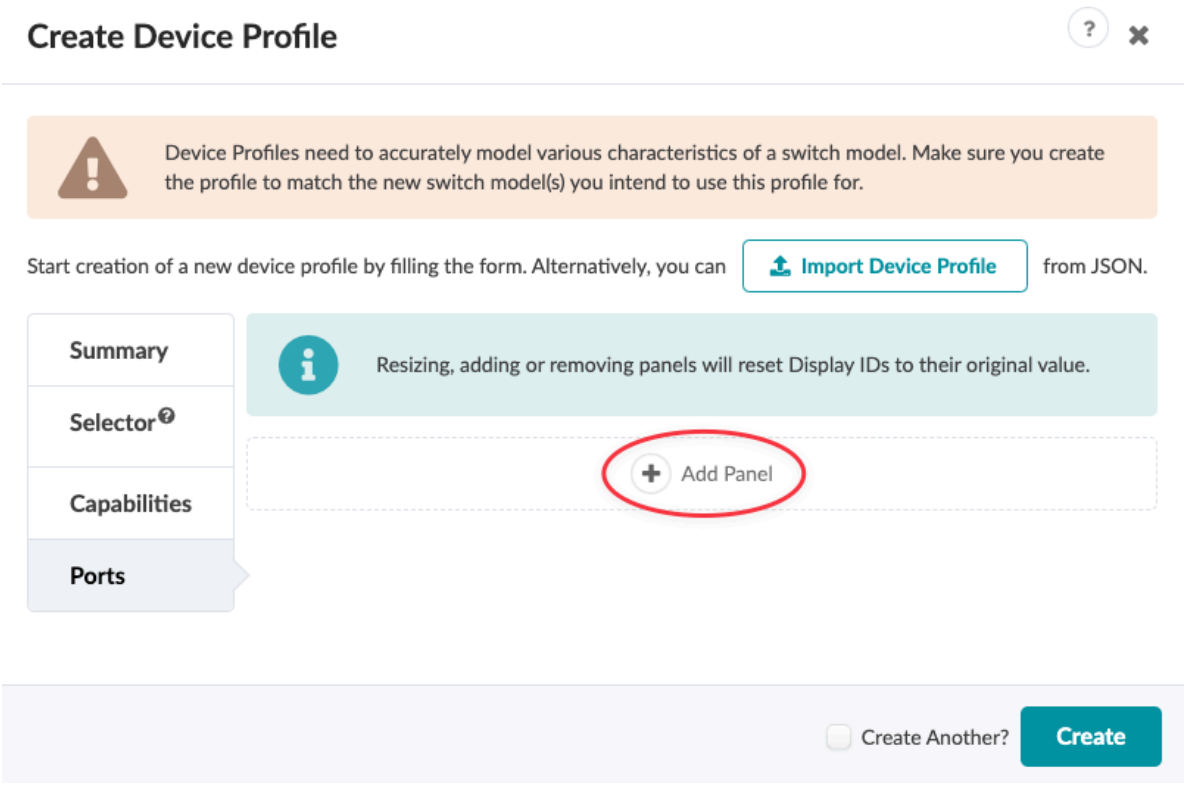
c. Select the OS family from the **OS family** drop-down list.

d. Select the OS version from the **Version** drop-down list.

5. Click **Capabilities** in the left panel.
   The **Capabilities** section of the **Create Device Profile** dialog opens.
6. Enter the relevant information in the **Capabilities** section. For details, see the "Capabilities" on page 785 section of **What are Device Profiles**.



7. Click **Ports** in the left panel.
   The **Ports** section of the **Create Device Profile** dialog opens.
8. Click **Add Panel**, then enter the relevant information in the **Ports** section. For details, see the "Ports" on page 787 section of **What are Device Profiles**.



9. Click **Create** to create the device profile and return to the **Device Profiles** table view.

Update Device Profile

If a device profile is used in an "interface map" on page 853, you may not be able to update it if it would adversely affect that interface map. You can't change predefined profiles, since your changes would be discarded when you upgrade the Apstra server. (You *could* clone and update a predefined device profile instead.)

> ⚠ **CAUTION**: Updating a device profile can lead to a mismatch between the profile's stated abilities and the device's actual capabilities, potentially leading to unexpected results.

1. Either from the table view (Devices > Device Profiles) or the details view, click the **Edit** button for the device profile to update.
2. Make your changes.
3. Click **Update** (bottom-right) to update the device profile and return to the table view.

Delete Device Profile (Devices)

Predefined device profiles can't be deleted. Device profiles used in interface maps can't be deleted.

1. Either from the table view (Devices > Device Profiles) or the details view, click the **Delete** button for the device profile to delete.
2. Click **Delete** to delete the device profile and return to the table view.

   You can also use REST API to manage device profiles. Navigate to **Platform > Developers** for **REST API Documentation** and tools.

Juniper Device Profiles

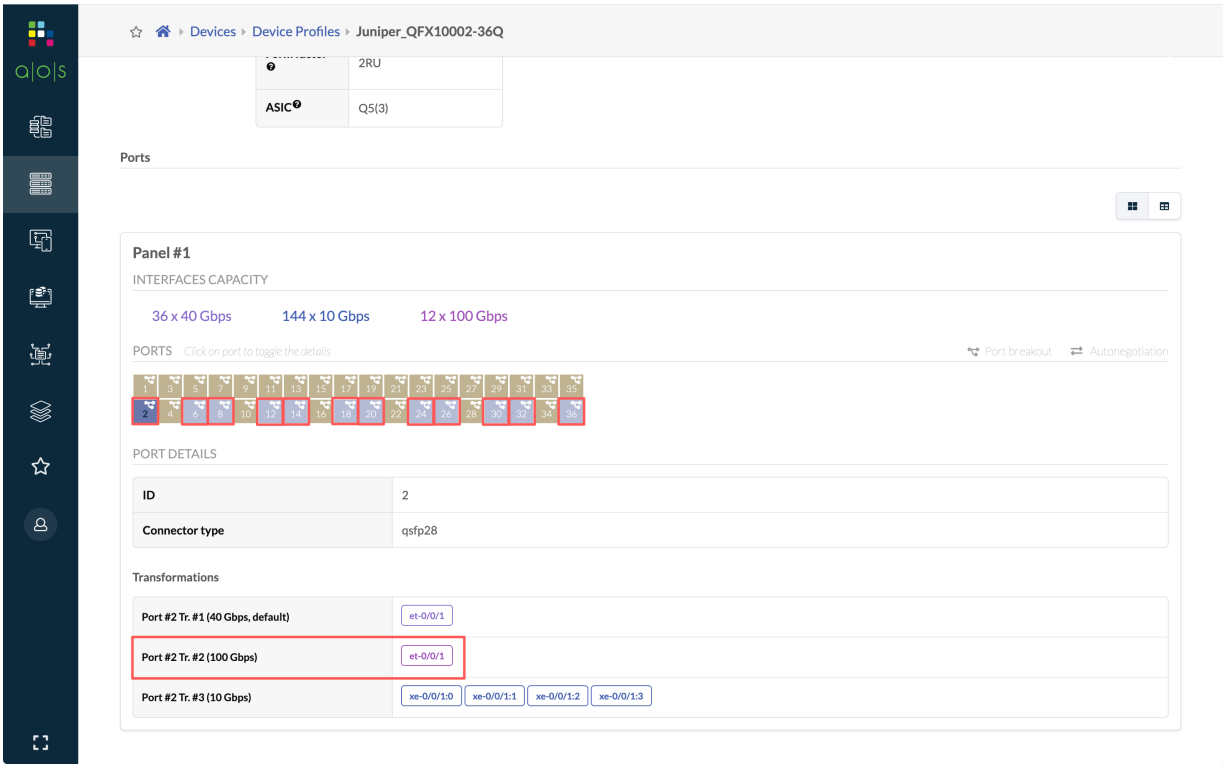
IN THIS SECTION

Overview

Predefined device profiles for most qualified Juniper devices ship with Apstra software. For a complete list of qualified and recommended Juniper device series and Junos versions, see "Qualified Device and NOS Versions" on page 1503. Juniper device profile constraints are specified below.

Juniper QFX10002

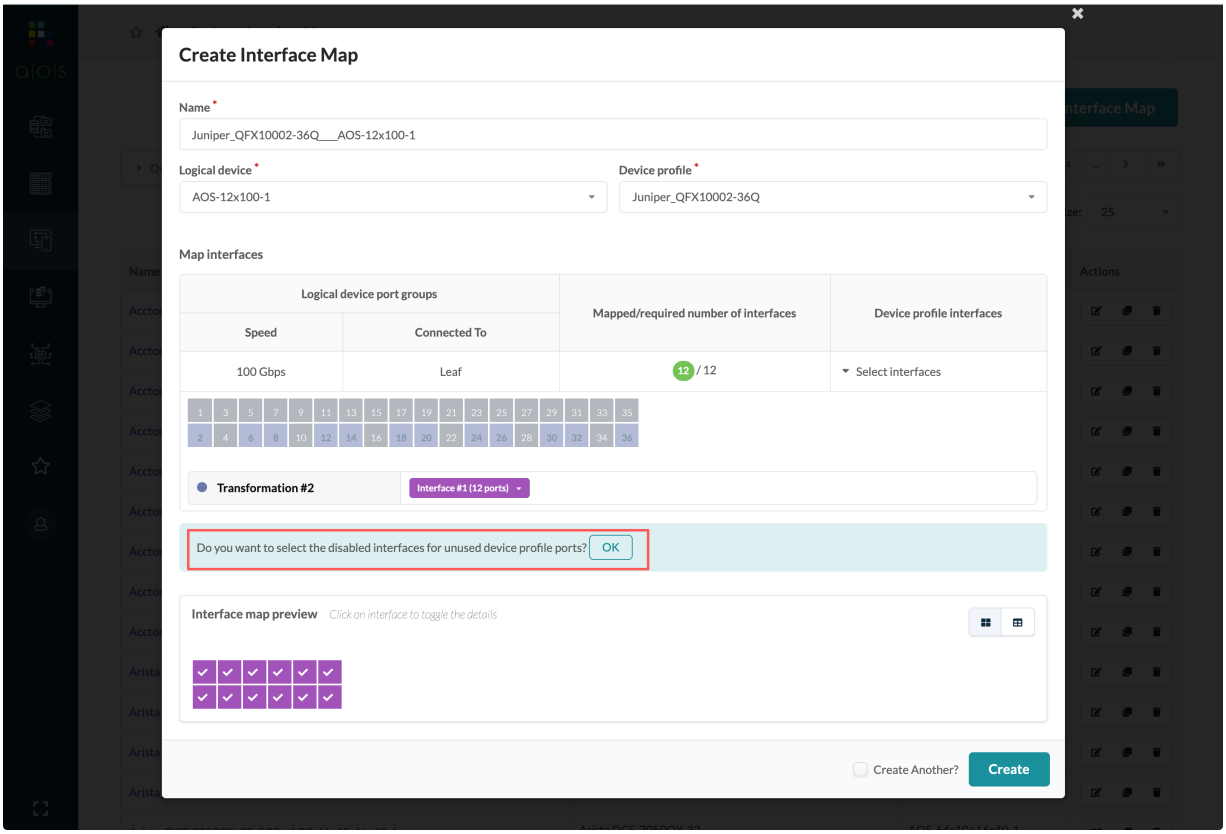
The 36-port Juniper QFX10002-36Q and 72-port QFX10002-72Q are qualified devices. Both of these models have a port constraint where only certain ports can be used with QSFP28 100G transceivers.



If these ports are used as 100G, then the adjacent QSFP 40G ports can't be used. The device profile can't automatically disable the adjacent QSFP 40G ports. You must create an interface map with these ports unused and disabled.

When you select the 100G ports while you're creating the interface map for QFX10002, you are asked if you want to select the disabled interfaces for unused device profile ports. For 100G ports on the

QFX10002, click **OK** so the unused QSFP ports are disabled and can't be used.



SONiC Device Profile

IN THIS SECTION

Background

Devices are recognized in the Apstra environment with device profiles. They capture device-specific semantics, which are required for the Apstra software to discover them and to run network configs that work well for the datapath once inside the blueprint.

Device profiles are REST entities, which enable you to create, edit, delete, and list during the design phase. Device profiles are used to create interface maps, which get directly used inside the Apstra config rendering engine when blueprints are deployed.

This document covers the knowledge required to create (and edit) a semantically correct Sonic DP, so that not only does it pass the validations in place in Apstra which ensure the right DP is created in the database, but also honors the vendor semantic requirement applicable to the device so that it does not result in deploy failure when the generated configuration is pushed to the network device.

Problem Statement

Device profiles are vendor semantics-aware data structures. To create a device profile, you need the device specification from the vendor. To create a valid and config-friendly JSON, you'll need to translate these specifications into the Apstra device profile data model.

Solution

The high level data model is the same for all DPs. The same keys are used for every device profile. The way we get the values might differ, or might be loaded with a vendor constraint. The document enlists the following:

- The schema of the DP and the nested elements inside the DP.
- The meaning of each key value pair in the schema.
- The vendor specific recipe the values are populated.
- List any constraints, corner cases to consider, especially for port configurations for certain (group of) models.
- Any lessons learnt along the way creating those DPs already in production useful in creating future ones.

User Interface

When you create device profiles from the Apstra GUI, some of your entries are semantically validated. It's not completely capable of ensuring deep vendor-specific constraints and requirements though. With the exact vendor specification, the GUI assists you with creating a semantically valid DP which becomes part of the Apstra database data model.

Alternatively, you can write your own Python code that contains the vendor specifications, normalize it as per Apstra DP data model and generate the json to then import with the GUI.

Selector information

Entering the correct information in all four of the selector fields is critical for the device to get matched to the device profile.

Selector Field	Value	Command to get the information on device
model	0x21	show platform syseeprom
manufacturer	If 0x2D in syseeprom, 0x2D else 0x2B	show platform syseeprom
OS family	SONiC	Show version
version	.*	Show version

Capabilities

If you have the device specification, you can obtain its hardware and software capabilities for entry into the device profile.

The table below contains commonly found values in SONiC devices (based on qualified devices).

Selector Field	Value	Command to get the information on device
userland	64 (int)	Does not affect config
form_factor	'1RU' (string)	Does not affect config
ecmp_limit	64 (int)	Does not affect config
asic	'T2' (string)	Does not affect config
cpu	'x86' (string)	Does not affect config
ram	16 (int) (Note, the unit is in GB)	Does not affect config
onie	True (bool) (default)	Does not affect config
lxc	True (bool) (default)	Does not affect config

Interface naming conventions

Sonic follows the naming conventions per the sonic port name file as found Azure SONIC on the github master. https://github.com/Azure/SONiC/blob/master/doc/sonic-port-name.md

To create a SONIC device profile, you must read through the device specific port_config.ini (for example, sonic-buildimage/device/mellanox/x86_64-mlnx_msn2100-r0/ACS-MSN2100/port_config.ini) file and follow the instructions in the above link to come up with the right interface names.

The port_congi.ini specifies interface names that SONiC uses. The device profile must match interface names which will generate the PORT configs in the configuration file (config_db.json) . For this document purposes, port_config.ini and config_db.json should have the same interface naming standard. Use those interface names in your DP along with the lane numbers provided in the port_cfg.ini file. Once a device profile has been generated based on the aforementioned steps, Apstra will use that along with the LD to generate the Interface Map (IM). Apstra as part of its validation will make sure that the IM (which describes the port and its speeds) are indeed available and supported under "/usr/share/sonic/device/x86_64-mlnx_msn2100-r0/ACS-MSN2100/port_config.ini" . This validation is performed to make sure SONiC NOS stack does not fail due to unsupported port configuration (in config_db.json) getting wrongly generated in Apstra due to wrong DP. So it is important that the end user makes sure the DP that is generated for a SONiC platform has the correct interface names and lane maps as reflected in port_config.ini file for that particular platform. A platform may have a few different port_config.ini files part of different HWSKUs for that platform. Apstra will try to validate the generated port configs with any of the available options for that platform. Apstra currently does not use the Dynamic Port breakout feature which is on-going in the SONiC project.

Troubleshooting

Device mismatch usually occurs at the beginning of a device's lifecycle. If the device is not selecting the device profile, check the four selector fields in the device profile.

If ports are configured with incorrect speeds or if the OS-specific port constraints were not handled in the device profile or interface map, then deploy errors could be raised.

A possible flow for root cause would be:

- Check the DP for obvious port capabilities errors. Is the port really capable of the speeds the DP has configured. The device specific port_config.ini Sonic open source project is a good resource to parse for ERROR messages.
- Check if the DP has configured autoneg or disabled interfaces correctly. Autoneg and disabled can both be expressed in the interface setting field.
- When debugging the interface names and lane mapping, please take a look at the corresponding port_config.ini. As an example for AS5712-54X edgecore/accton box we can get the port_config.ini file that has the details like lane/name/alias at https://github.com/Azure/sonic-buildimage/tree/master/device/accton/x86_64-accton_as5712_54x-r0/Accton-AS5712-54X

- You can find the naming constraints in the official SONiC documentation. For example if you want to generate the interface names for Accton 5712 54X running SONIC, the port_config.ini is the authority. https://github.com/Azure/sonic-buildimage/blob/master/device/accton/x86_64-accton_as5712_54x-r0/Accton-AS5712-54X/port_config.ini Sometimes the device might have inter-port constraints. For SONiC, it's generally laid out in the port_config.ini file. A specific platform could have multiple port_config.ini files, and a specific manufacturer with each port_config.ini file residing in their on HWSKU folders in the sonic image (like the one referenced above). The ability to try out different port speeds on (outside of what is listed in the port_config.ini) will need knowledge of the chipset and also the physical switch manufacturer to see what can be achieved. This information may not be available in any white papers unless requested of vendors.

Example: DP and port_config.ini

Port_config.ini from sonic-buildimage is below for Dell_Z9100 (x86_64-dell_z9100_c2538-r0/Force10-Z9100-C32

```
# name          lanes                alias              index
Ethernet0       49,50,51,52          hundredGigE1/1     1
Ethernet4       53,54,55,56          hundredGigE1/2     2
Ethernet8       57,58,59,60          hundredGigE1/3     3
Ethernet12      61,62,63,64          hundredGigE1/4     4
Ethernet16      65,66,67,68          hundredGigE1/5     5
Ethernet20      69,70,71,72          hundredGigE1/6     6
Ethernet24      73,74,75,76          hundredGigE1/7     7
Ethernet28      77,78,79,80          hundredGigE1/8     8
Ethernet32      37,38,39,40          hundredGigE1/9     9
Ethernet36      33,34,35,36          hundredGigE1/10    10
Ethernet40      45,46,47,48          hundredGigE1/11    11
Ethernet44      41,42,43,44          hundredGigE1/12    12
Ethernet48      81,82,83,84          hundredGigE1/13    13
Ethernet52      85,86,87,88          hundredGigE1/14    14
Ethernet56      89,90,91,92          hundredGigE1/15    15
Ethernet60      93,94,95,96          hundredGigE1/16    16
Ethernet64      97,98,99,100         hundredGigE1/17    17
Ethernet68      101,102,103,104      hundredGigE1/18    18
Ethernet72      105,106,107,108      hundredGigE1/19    19
Ethernet76      109,110,111,112      hundredGigE1/20    20
Ethernet80      21,22,23,24          hundredGigE1/21    21
Ethernet84      17,18,19,20          hundredGigE1/22    22
Ethernet88      29,30,31,32          hundredGigE1/23    23
Ethernet92      25,26,27,28          hundredGigE1/24    24
Ethernet96      117,118,119,120      hundredGigE1/25    25
Ethernet100     113,114,115,116      hundredGigE1/26    26
```

```
Ethernet104     125,126,127,128       hundredGigE1/27    27
Ethernet108     121,122,123,124       hundredGigE1/28    28
Ethernet112     5,6,7,8               hundredGigE1/29    29
Ethernet116     1,2,3,4               hundredGigE1/30    30
Ethernet120     13,14,15,16           hundredGigE1/31    31
Ethernet124     9,10,11,12            hundredGigE1/32    32
```

Translate port_config to a port-to-lane_map data structure using parse.py script:

```
Parse.py
=========
#!/usr/bin/python
# Copyright (c) 2017 Apstrktr, Inc.  All rights reserved.
# Apstrktr, Inc. Confidential and Proprietary.
#
# This source code is licensed under End User License Agreement found in the
# LICENSE file at http://apstra.com/eula

# pylint: disable=line-too-long

import sys
from pprint import pprint

# Run the program as ./parse.py <path_to_sonic_platform_port_config.ini>
# ex: ./parse.py sonic-buildimage/device/mellanox/x86_64-mlnx_msn2100-r0/ACS-MSN2100/
port_config.ini
def get_lanemap(buf):
    if not buf:
        return None
    d = {}
    interface_indices = []
    for line in buf.split('\n'):
        if line.startswith('#'):
            continue
        words = line.split(' ')
        words = [word for word in words if len(word)]
        if not len(words):
            continue
        intf = words[0][8:]
        lane = words[1].split(',')
        interface_indices.append(intf)
        if len(lane) > 1:
```

```
            one = 'Ethernet' + str(intf)
            two = 'Ethernet' + str(int(intf)+1)
            three = 'Ethernet' + str(int(intf)+2)
            four = 'Ethernet' + str(int(intf)+3)
            d.update({one:lane[0]})
            d.update({two:lane[1]})
            d.update({three:lane[2]})
            d.update({four:lane[3]})
        else:
            d.update({words[0]:words[1]})
    return {'interface_names' : interface_indices, 'lane_mapping' : d}

def parse_portconfig(f):
    buf = ''
    with open(f, 'r') as stream:
        buf = stream.read()
    return {'<Platform>': get_lanemap(buf)}


if __name__ == '__main__':
    assert len(sys.argv) > 1, "Missing port_config.ini in cmdline"
    print "Collecting lane information from ", sys.argv[1]
    pprint(parse_portconfig(sys.argv[1]))
    print
"================================================================================================"
    print "              Substitute <Platform> with an identifier for the platform"
    print "     Append the dump into sdk/device-profile/sonic.py's sonic_device_info dictionary"
    print
"================================================================================================"


To run parse.py

parse.py  <Path to the port_config.ini file from sonic_buildimage>

Example:

parse.py sonic-buildimage/device/dell/x86_64-dell_z9100_c2538-r0/Force10-Z9100-C32/
port_config.ini

Collecting lane information from  sonic-buildimage/device/dell/x86_64-dell_z9100_c2538-r0/
Force10-Z9100-C32/port_config.ini
{'<Platform>': {'interface_names': ['0',
```

```
                                '4',
                                '8',
                                '12',
                                '16',
                                '20',
                                '24',
                                '28',
                                '32',
                                '36',
                                '40',
                                '44',
                                '48',
                                '52',
                                '56',
                                '60',
                                '64',
                                '68',
                                '72',
                                '76',
                                '80',
                                '84',
                                '88',
                                '92',
                                '96',
                                '100',
                                '104',
                                '108',
                                '112',
                                '116',
                                '120',
                                '124'],
                'lane_mapping': {'Ethernet0': '49',
                                 'Ethernet1': '50',
                                 'Ethernet10': '59',
                                 'Ethernet100': '113',
                                 'Ethernet101': '114',
                                 'Ethernet102': '115',
                                 'Ethernet103': '116',
                                 'Ethernet104': '125',
                                 'Ethernet105': '126',
                                 'Ethernet106': '127',
                                 'Ethernet107': '128',
                                 'Ethernet108': '121',
```

```
                                'Ethernet109': '122',
                                'Ethernet11': '60',
                                'Ethernet110': '123',
                                'Ethernet111': '124',
                                'Ethernet112': '5',
                                'Ethernet113': '6',
                                'Ethernet114': '7',
                                'Ethernet115': '8',
                                'Ethernet116': '1',
                                'Ethernet117': '2',
                                'Ethernet118': '3',
                                'Ethernet119': '4',
                                'Ethernet12': '61',
                                'Ethernet120': '13',
                                'Ethernet121': '14',
                                'Ethernet122': '15',
                                'Ethernet123': '16',
                                'Ethernet124': '9',
                                'Ethernet125': '10',
                                'Ethernet126': '11',
                                'Ethernet127': '12',
                                'Ethernet13': '62',
                                'Ethernet14': '63',
                                'Ethernet15': '64',
                                'Ethernet16': '65',
                                'Ethernet17': '66',
                                'Ethernet18': '67',
                                'Ethernet19': '68',
                                'Ethernet2': '51',
                                'Ethernet20': '69',
                                'Ethernet21': '70',
                                'Ethernet22': '71',
                                'Ethernet23': '72',
                                'Ethernet24': '73',
                                'Ethernet25': '74',
                                'Ethernet26': '75',
                                'Ethernet27': '76',
                                'Ethernet28': '77',
                                'Ethernet29': '78',
                                'Ethernet3': '52',
                                'Ethernet30': '79',
                                'Ethernet31': '80',
                                'Ethernet32': '37',
```

```
                                'Ethernet33': '38',
                                'Ethernet34': '39',
                                'Ethernet35': '40',
                                'Ethernet36': '33',
                                'Ethernet37': '34',
                                'Ethernet38': '35',
                                'Ethernet39': '36',
                                'Ethernet4': '53',
                                'Ethernet40': '45',
                                'Ethernet41': '46',
                                'Ethernet42': '47',
                                'Ethernet43': '48',
                                'Ethernet44': '41',
                                'Ethernet45': '42',
                                'Ethernet46': '43',
                                'Ethernet47': '44',
                                'Ethernet48': '81',
                                'Ethernet49': '82',
                                'Ethernet5': '54',
                                'Ethernet50': '83',
                                'Ethernet51': '84',
                                'Ethernet52': '85',
                                'Ethernet53': '86',
                                'Ethernet54': '87',
                                'Ethernet55': '88',
                                'Ethernet56': '89',
                                'Ethernet57': '90',
                                'Ethernet58': '91',
                                'Ethernet59': '92',
                                'Ethernet6': '55',
                                'Ethernet60': '93',
                                'Ethernet61': '94',
                                'Ethernet62': '95',
                                'Ethernet63': '96',
                                'Ethernet64': '97',
                                'Ethernet65': '98',
                                'Ethernet66': '99',
                                'Ethernet67': '100',
                                'Ethernet68': '101',
                                'Ethernet69': '102',
                                'Ethernet7': '56',
                                'Ethernet70': '103',
                                'Ethernet71': '104',
```

```
                                'Ethernet72': '105',
                                'Ethernet73': '106',
                                'Ethernet74': '107',
                                'Ethernet75': '108',
                                'Ethernet76': '109',
                                'Ethernet77': '110',
                                'Ethernet78': '111',
                                'Ethernet79': '112',
                                'Ethernet8': '57',
                                'Ethernet80': '21',
                                'Ethernet81': '22',
                                'Ethernet82': '23',
                                'Ethernet83': '24',
                                'Ethernet84': '17',
                                'Ethernet85': '18',
                                'Ethernet86': '19',
                                'Ethernet87': '20',
                                'Ethernet88': '29',
                                'Ethernet89': '30',
                                'Ethernet9': '58',
                                'Ethernet90': '31',
                                'Ethernet91': '32',
                                'Ethernet92': '25',
                                'Ethernet93': '26',
                                'Ethernet94': '27',
                                'Ethernet95': '28',
                                'Ethernet96': '117',
                                'Ethernet97': '118',
                                'Ethernet98': '119',
                                'Ethernet99': '120'}}}
================================================================================
              Substitute <Platform> with an identifier for the platform
     Append the dump into sdk/device-profile/sonic.py's sonic_device_info dictionary
================================================================================
```

The output from above will become a dictionary entry in sonic_device_info in the sonic device_profile generator python file.

Corresponding Device Profile generated in Apstra:

```
{
```

```
  "hardware_capabilities": {
    "asic": "TH",
    "cpu": "x86",
    "ecmp_limit": 64,
    "form_factor": "1RU",
    "ram": 16,
    "userland": 64
  },
  "id": "Force10-Z9100_SONiC",
  "label": "Dell Force10-Z9100_SONiC",
  "ports": [
    {
      "column_id": 1,
      "connector_type": "qsfp28",
      "failure_domain_id": 1,
      "panel_id": 1,
      "port_id": 0,
      "row_id": 1,
      "slot_id": 0,
      "transformations": [
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet0",
              "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\":
\"49,50,51,52\"}}",
              "speed": {
                "unit": "G",
                "value": 100
              },
              "state": "active"
            }
          ],
          "is_default": true,
          "transformation_id": 1
        },
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet0",
              "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\":
```

```
\"49,50,51,52\"}}",
              "speed": {
                "unit": "G",
                "value": 40
              },
              "state": "active"
            }
          ],
          "is_default": false,
          "transformation_id": 2
        }
      ]
    },
    {
      "column_id": 1,
      "connector_type": "qsfp28",
      "failure_domain_id": 1,
      "panel_id": 1,
      "port_id": 1,
      "row_id": 2,
      "slot_id": 0,
      "transformations": [
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet4",
              "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\":
\"53,54,55,56\"}}",
              "speed": {
                "unit": "G",
                "value": 100
              },
              "state": "active"
            }
          ],
          "is_default": true,
          "transformation_id": 1
        },
        {
          "interfaces": [
            {
              "interface_id": 1,
```

```
              "name": "Ethernet4",
              "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\":
\"53,54,55,56\"}}",
              "speed": {
                "unit": "G",
                "value": 40
              },
              "state": "active"
            }
          ],
          "is_default": false,
          "transformation_id": 2
        }
      ]
    },
    {
      "column_id": 2,
      "connector_type": "qsfp28",
      "failure_domain_id": 1,
      "panel_id": 1,
      "port_id": 2,
      "row_id": 1,
      "slot_id": 0,
      "transformations": [
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet8",
              "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\":
\"57,58,59,60\"}}",
              "speed": {
                "unit": "G",
                "value": 100
              },
              "state": "active"
            }
          ],
          "is_default": true,
          "transformation_id": 1
        },
        {
          "interfaces": [
```

```
            {
              "interface_id": 1,
              "name": "Ethernet8",
              "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\":
\"57,58,59,60\"}}",
              "speed": {
                "unit": "G",
                "value": 40
              },
              "state": "active"
            }
          ],
          "is_default": false,
          "transformation_id": 2
        }
      ]
    },
    {
      "column_id": 2,
      "connector_type": "qsfp28",
      "failure_domain_id": 1,
      "panel_id": 1,
      "port_id": 3,
      "row_id": 2,
      "slot_id": 0,
      "transformations": [
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet12",
              "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\":
\"61,62,63,64\"}}",
              "speed": {
                "unit": "G",
                "value": 100
              },
              "state": "active"
            }
          ],
          "is_default": true,
          "transformation_id": 1
        },
```

```
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet12",
              "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\":
\"61,62,63,64\"}}",
              "speed": {
                "unit": "G",
                "value": 40
              },
              "state": "active"
            }
          ],
          "is_default": false,
          "transformation_id": 2
        }
      ]
    },
    {
      "column_id": 3,
      "connector_type": "qsfp28",
      "failure_domain_id": 1,
      "panel_id": 1,
      "port_id": 4,
      "row_id": 1,
      "slot_id": 0,
      "transformations": [
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet16",
              "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\":
\"65,66,67,68\"}}",
              "speed": {
                "unit": "G",
                "value": 100
              },
              "state": "active"
            }
          ],
          "is_default": true,
```

```
          "transformation_id": 1
        },
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet16",
              "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\":
\"65,66,67,68\"}}",
              "speed": {
                "unit": "G",
                "value": 40
              },
              "state": "active"
            }
          ],
          "is_default": false,
          "transformation_id": 2
        }
      ]
    },
    {
      "column_id": 3,
      "connector_type": "qsfp28",
      "failure_domain_id": 1,
      "panel_id": 1,
      "port_id": 5,
      "row_id": 2,
      "slot_id": 0,
      "transformations": [
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet20",
              "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\":
\"69,70,71,72\"}}",
              "speed": {
                "unit": "G",
                "value": 100
              },
              "state": "active"
            }
```

```
          ],
          "is_default": true,
          "transformation_id": 1
        },
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet20",
              "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\":
\"69,70,71,72\"}}",
              "speed": {
                "unit": "G",
                "value": 40
              },
              "state": "active"
            }
          ],
          "is_default": false,
          "transformation_id": 2
        }
      ]
    },
    {
      "column_id": 4,
      "connector_type": "qsfp28",
      "failure_domain_id": 1,
      "panel_id": 1,
      "port_id": 6,
      "row_id": 1,
      "slot_id": 0,
      "transformations": [
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet24",
              "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\":
\"73,74,75,76\"}}",
              "speed": {
                "unit": "G",
                "value": 100
              },
```

```
              "state": "active"
            }
          ],
          "is_default": true,
          "transformation_id": 1
        },
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet24",
              "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\":
\"73,74,75,76\"}}",
              "speed": {
                "unit": "G",
                "value": 40
              },
              "state": "active"
            }
          ],
          "is_default": false,
          "transformation_id": 2
        }
      ]
    },
    {
      "column_id": 4,
      "connector_type": "qsfp28",
      "failure_domain_id": 1,
      "panel_id": 1,
      "port_id": 7,
      "row_id": 2,
      "slot_id": 0,
      "transformations": [
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet28",
              "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\":
\"77,78,79,80\"}}",
              "speed": {
                "unit": "G",
```

```
                "value": 100
              },
              "state": "active"
            }
          ],
          "is_default": true,
          "transformation_id": 1
        },
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet28",
              "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\":
\"77,78,79,80\"}}",
              "speed": {
                "unit": "G",
                "value": 40
              },
              "state": "active"
            }
          ],
          "is_default": false,
          "transformation_id": 2
        }
      ]
    },
    {
      "column_id": 5,
      "connector_type": "qsfp28",
      "failure_domain_id": 1,
      "panel_id": 1,
      "port_id": 8,
      "row_id": 1,
      "slot_id": 0,
      "transformations": [
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet32",
              "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\":
\"37,38,39,40\"}}",
```

```
              "speed": {
                "unit": "G",
                "value": 100
              },
              "state": "active"
            }
          ],
          "is_default": true,
          "transformation_id": 1
        },
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet32",
              "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\":
\"37,38,39,40\"}}",
              "speed": {
                "unit": "G",
                "value": 40
              },
              "state": "active"
            }
          ],
          "is_default": false,
          "transformation_id": 2
        }
      ]
    },
    {
      "column_id": 5,
      "connector_type": "qsfp28",
      "failure_domain_id": 1,
      "panel_id": 1,
      "port_id": 9,
      "row_id": 2,
      "slot_id": 0,
      "transformations": [
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet36",
```

```
              "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\":
\"33,34,35,36\"}}",
              "speed": {
                "unit": "G",
                "value": 100
              },
              "state": "active"
            }
          ],
          "is_default": true,
          "transformation_id": 1
        },
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet36",
              "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\":
\"33,34,35,36\"}}",
              "speed": {
                "unit": "G",
                "value": 40
              },
              "state": "active"
            }
          ],
          "is_default": false,
          "transformation_id": 2
        }
      ]
    },
    {
      "column_id": 6,
      "connector_type": "qsfp28",
      "failure_domain_id": 1,
      "panel_id": 1,
      "port_id": 10,
      "row_id": 1,
      "slot_id": 0,
      "transformations": [
        {
          "interfaces": [
            {
```

```
              "interface_id": 1,
              "name": "Ethernet40",
              "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\":
\"45,46,47,48\"}}",
              "speed": {
                "unit": "G",
                "value": 100
              },
              "state": "active"
            }
          ],
          "is_default": true,
          "transformation_id": 1
        },
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet40",
              "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\":
\"45,46,47,48\"}}",
              "speed": {
                "unit": "G",
                "value": 40
              },
              "state": "active"
            }
          ],
          "is_default": false,
          "transformation_id": 2
        }
      ]
    },
    {
      "column_id": 6,
      "connector_type": "qsfp28",
      "failure_domain_id": 1,
      "panel_id": 1,
      "port_id": 11,
      "row_id": 2,
      "slot_id": 0,
      "transformations": [
        {
```

```
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet44",
              "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\":
\"41,42,43,44\"}}",
              "speed": {
                "unit": "G",
                "value": 100
              },
              "state": "active"
            }
          ],
          "is_default": true,
          "transformation_id": 1
        },
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet44",
              "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\":
\"41,42,43,44\"}}",
              "speed": {
                "unit": "G",
                "value": 40
              },
              "state": "active"
            }
          ],
          "is_default": false,
          "transformation_id": 2
        }
      ]
    },
    {
      "column_id": 7,
      "connector_type": "qsfp28",
      "failure_domain_id": 1,
      "panel_id": 1,
      "port_id": 12,
      "row_id": 1,
      "slot_id": 0,
```

```
      "transformations": [
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet48",
              "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\":
\"81,82,83,84\"}}",
              "speed": {
                "unit": "G",
                "value": 100
              },
              "state": "active"
            }
          ],
          "is_default": true,
          "transformation_id": 1
        },
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet48",
              "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\":
\"81,82,83,84\"}}",
              "speed": {
                "unit": "G",
                "value": 40
              },
              "state": "active"
            }
          ],
          "is_default": false,
          "transformation_id": 2
        }
      ]
    },
    {
      "column_id": 7,
      "connector_type": "qsfp28",
      "failure_domain_id": 1,
      "panel_id": 1,
      "port_id": 13,
```

```
      "row_id": 2,
      "slot_id": 0,
      "transformations": [
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet52",
              "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\":
\"85,86,87,88\"}}",
              "speed": {
                "unit": "G",
                "value": 100
              },
              "state": "active"
            }
          ],
          "is_default": true,
          "transformation_id": 1
        },
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet52",
              "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\":
\"85,86,87,88\"}}",
              "speed": {
                "unit": "G",
                "value": 40
              },
              "state": "active"
            }
          ],
          "is_default": false,
          "transformation_id": 2
        }
      ]
    },
    {
      "column_id": 8,
      "connector_type": "qsfp28",
      "failure_domain_id": 1,
```

```
      "panel_id": 1,
      "port_id": 14,
      "row_id": 1,
      "slot_id": 0,
      "transformations": [
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet56",
              "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\":
\"89,90,91,92\"}}",
              "speed": {
                "unit": "G",
                "value": 100
              },
              "state": "active"
            }
          ],
          "is_default": true,
          "transformation_id": 1
        },
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet56",
              "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\":
\"89,90,91,92\"}}",
              "speed": {
                "unit": "G",
                "value": 40
              },
              "state": "active"
            }
          ],
          "is_default": false,
          "transformation_id": 2
        }
      ]
    },
    {
      "column_id": 8,
```

```
      "connector_type": "qsfp28",
      "failure_domain_id": 1,
      "panel_id": 1,
      "port_id": 15,
      "row_id": 2,
      "slot_id": 0,
      "transformations": [
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet60",
              "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\":
\"93,94,95,96\"}}",
              "speed": {
                "unit": "G",
                "value": 100
              },
              "state": "active"
            }
          ],
          "is_default": true,
          "transformation_id": 1
        },
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet60",
              "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\":
\"93,94,95,96\"}}",
              "speed": {
                "unit": "G",
                "value": 40
              },
              "state": "active"
            }
          ],
          "is_default": false,
          "transformation_id": 2
        }
      ]
    },
```

```
    {
      "column_id": 9,
      "connector_type": "qsfp28",
      "failure_domain_id": 1,
      "panel_id": 1,
      "port_id": 16,
      "row_id": 1,
      "slot_id": 0,
      "transformations": [
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet64",
              "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\":
\"97,98,99,100\"}}",
              "speed": {
                "unit": "G",
                "value": 100
              },
              "state": "active"
            }
          ],
          "is_default": true,
          "transformation_id": 1
        },
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet64",
              "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\":
\"97,98,99,100\"}}",
              "speed": {
                "unit": "G",
                "value": 40
              },
              "state": "active"
            }
          ],
          "is_default": false,
          "transformation_id": 2
        }
```

```
      ]
    },
    {
      "column_id": 9,
      "connector_type": "qsfp28",
      "failure_domain_id": 1,
      "panel_id": 1,
      "port_id": 17,
      "row_id": 2,
      "slot_id": 0,
      "transformations": [
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet68",
              "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\":
\"101,102,103,104\"}}",
              "speed": {
                "unit": "G",
                "value": 100
              },
              "state": "active"
            }
          ],
          "is_default": true,
          "transformation_id": 1
        },
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet68",
              "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\":
\"101,102,103,104\"}}",
              "speed": {
                "unit": "G",
                "value": 40
              },
              "state": "active"
            }
          ],
          "is_default": false,
```

```
          "transformation_id": 2
        }
      ]
    },
    {
      "column_id": 10,
      "connector_type": "qsfp28",
      "failure_domain_id": 1,
      "panel_id": 1,
      "port_id": 18,
      "row_id": 1,
      "slot_id": 0,
      "transformations": [
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet72",
              "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\":
\"105,106,107,108\"}}",
              "speed": {
                "unit": "G",
                "value": 100
              },
              "state": "active"
            }
          ],
          "is_default": true,
          "transformation_id": 1
        },
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet72",
              "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\":
\"105,106,107,108\"}}",
              "speed": {
                "unit": "G",
                "value": 40
              },
              "state": "active"
            }
```

```
          ],
          "is_default": false,
          "transformation_id": 2
        }
      ]
    },
    {
      "column_id": 10,
      "connector_type": "qsfp28",
      "failure_domain_id": 1,
      "panel_id": 1,
      "port_id": 19,
      "row_id": 2,
      "slot_id": 0,
      "transformations": [
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet76",
              "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\":
\"109,110,111,112\"}}",
              "speed": {
                "unit": "G",
                "value": 100
              },
              "state": "active"
            }
          ],
          "is_default": true,
          "transformation_id": 1
        },
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet76",
              "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\":
\"109,110,111,112\"}}",
              "speed": {
                "unit": "G",
                "value": 40
              },
```

```
              "state": "active"
            }
          ],
          "is_default": false,
          "transformation_id": 2
        }
      ]
    },
    {
      "column_id": 11,
      "connector_type": "qsfp28",
      "failure_domain_id": 1,
      "panel_id": 1,
      "port_id": 20,
      "row_id": 1,
      "slot_id": 0,
      "transformations": [
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet80",
              "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\":
\"21,22,23,24\"}}",
              "speed": {
                "unit": "G",
                "value": 100
              },
              "state": "active"
            }
          ],
          "is_default": true,
          "transformation_id": 1
        },
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet80",
              "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\":
\"21,22,23,24\"}}",
              "speed": {
                "unit": "G",
```

```
                "value": 40
              },
              "state": "active"
            }
          ],
          "is_default": false,
          "transformation_id": 2
        }
      ]
    },
    {
      "column_id": 11,
      "connector_type": "qsfp28",
      "failure_domain_id": 1,
      "panel_id": 1,
      "port_id": 21,
      "row_id": 2,
      "slot_id": 0,
      "transformations": [
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet84",
              "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\":
\"17,18,19,20\"}}",
              "speed": {
                "unit": "G",
                "value": 100
              },
              "state": "active"
            }
          ],
          "is_default": true,
          "transformation_id": 1
        },
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet84",
              "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\":
\"17,18,19,20\"}}",
```

```
              "speed": {
                "unit": "G",
                "value": 40
              },
              "state": "active"
            }
          ],
          "is_default": false,
          "transformation_id": 2
        }
      ]
    },
    {
      "column_id": 12,
      "connector_type": "qsfp28",
      "failure_domain_id": 1,
      "panel_id": 1,
      "port_id": 22,
      "row_id": 1,
      "slot_id": 0,
      "transformations": [
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet88",
              "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\":
\"29,30,31,32\"}}",
              "speed": {
                "unit": "G",
                "value": 100
              },
              "state": "active"
            }
          ],
          "is_default": true,
          "transformation_id": 1
        },
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet88",
```

```
              "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\":
\"29,30,31,32\"}}",
              "speed": {
                "unit": "G",
                "value": 40
              },
              "state": "active"
            }
          ],
          "is_default": false,
          "transformation_id": 2
        }
      ]
    },
    {
      "column_id": 12,
      "connector_type": "qsfp28",
      "failure_domain_id": 1,
      "panel_id": 1,
      "port_id": 23,
      "row_id": 2,
      "slot_id": 0,
      "transformations": [
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet92",
              "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\":
\"25,26,27,28\"}}",
              "speed": {
                "unit": "G",
                "value": 100
              },
              "state": "active"
            }
          ],
          "is_default": true,
          "transformation_id": 1
        },
        {
          "interfaces": [
            {
```

```
              "interface_id": 1,
              "name": "Ethernet92",
              "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\":
\"25,26,27,28\"}}",
              "speed": {
                "unit": "G",
                "value": 40
              },
              "state": "active"
            }
          ],
          "is_default": false,
          "transformation_id": 2
        }
      ]
    },
    {
      "column_id": 13,
      "connector_type": "qsfp28",
      "failure_domain_id": 1,
      "panel_id": 1,
      "port_id": 24,
      "row_id": 1,
      "slot_id": 0,
      "transformations": [
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet96",
              "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\":
\"117,118,119,120\"}}",
              "speed": {
                "unit": "G",
                "value": 100
              },
              "state": "active"
            }
          ],
          "is_default": true,
          "transformation_id": 1
        },
        {
```

```
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet96",
              "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\":
\"117,118,119,120\"}}",
              "speed": {
                "unit": "G",
                "value": 40
              },
              "state": "active"
            }
          ],
          "is_default": false,
          "transformation_id": 2
        }
      ]
    },
    {
      "column_id": 13,
      "connector_type": "qsfp28",
      "failure_domain_id": 1,
      "panel_id": 1,
      "port_id": 25,
      "row_id": 2,
      "slot_id": 0,
      "transformations": [
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet100",
              "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\":
\"113,114,115,116\"}}",
              "speed": {
                "unit": "G",
                "value": 100
              },
              "state": "active"
            }
          ],
          "is_default": true,
          "transformation_id": 1
```

```
        },
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet100",
              "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\":
\"113,114,115,116\"}}",
              "speed": {
                "unit": "G",
                "value": 40
              },
              "state": "active"
            }
          ],
          "is_default": false,
          "transformation_id": 2
        }
      ]
    },
    {
      "column_id": 14,
      "connector_type": "qsfp28",
      "failure_domain_id": 1,
      "panel_id": 1,
      "port_id": 26,
      "row_id": 1,
      "slot_id": 0,
      "transformations": [
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet104",
              "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\":
\"125,126,127,128\"}}",
              "speed": {
                "unit": "G",
                "value": 100
              },
              "state": "active"
            }
          ],
```

```
          "is_default": true,
          "transformation_id": 1
        },
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet104",
              "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\":
\"125,126,127,128\"}}",
              "speed": {
                "unit": "G",
                "value": 40
              },
              "state": "active"
            }
          ],
          "is_default": false,
          "transformation_id": 2
        }
      ]
    },
    {
      "column_id": 14,
      "connector_type": "qsfp28",
      "failure_domain_id": 1,
      "panel_id": 1,
      "port_id": 27,
      "row_id": 2,
      "slot_id": 0,
      "transformations": [
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet108",
              "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\":
\"121,122,123,124\"}}",
              "speed": {
                "unit": "G",
                "value": 100
              },
              "state": "active"
```

```
            }
          ],
          "is_default": true,
          "transformation_id": 1
        },
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet108",
              "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\":
\"121,122,123,124\"}}",
              "speed": {
                "unit": "G",
                "value": 40
              },
              "state": "active"
            }
          ],
          "is_default": false,
          "transformation_id": 2
        }
      ]
    },
    {
      "column_id": 15,
      "connector_type": "qsfp28",
      "failure_domain_id": 1,
      "panel_id": 1,
      "port_id": 28,
      "row_id": 1,
      "slot_id": 0,
      "transformations": [
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet112",
              "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\": \"5,6,7,8\"}}",
              "speed": {
                "unit": "G",
                "value": 100
              },
```

```
            "state": "active"
          }
        ],
        "is_default": true,
        "transformation_id": 1
      },
      {
        "interfaces": [
          {
            "interface_id": 1,
            "name": "Ethernet112",
            "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\": \"5,6,7,8\"}}",
            "speed": {
              "unit": "G",
              "value": 40
            },
            "state": "active"
          }
        ],
        "is_default": false,
        "transformation_id": 2
      }
    ]
  },
  {
    "column_id": 15,
    "connector_type": "qsfp28",
    "failure_domain_id": 1,
    "panel_id": 1,
    "port_id": 29,
    "row_id": 2,
    "slot_id": 0,
    "transformations": [
      {
        "interfaces": [
          {
            "interface_id": 1,
            "name": "Ethernet116",
            "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\": \"1,2,3,4\"}}",
            "speed": {
              "unit": "G",
              "value": 100
            },
```

```
              "state": "active"
            }
          ],
          "is_default": true,
          "transformation_id": 1
        },
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet116",
              "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\": \"1,2,3,4\"}}",
              "speed": {
                "unit": "G",
                "value": 40
              },
              "state": "active"
            }
          ],
          "is_default": false,
          "transformation_id": 2
        }
      ]
    },
    {
      "column_id": 16,
      "connector_type": "qsfp28",
      "failure_domain_id": 1,
      "panel_id": 1,
      "port_id": 30,
      "row_id": 1,
      "slot_id": 0,
      "transformations": [
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet120",
              "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\":
\"13,14,15,16\"}}",
              "speed": {
                "unit": "G",
                "value": 100
```

```
              },
              "state": "active"
            }
          ],
          "is_default": true,
          "transformation_id": 1
        },
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet120",
              "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\":
\"13,14,15,16\"}}",
              "speed": {
                "unit": "G",
                "value": 40
              },
              "state": "active"
            }
          ],
          "is_default": false,
          "transformation_id": 2
        }
      ]
    },
    {
      "column_id": 16,
      "connector_type": "qsfp28",
      "failure_domain_id": 1,
      "panel_id": 1,
      "port_id": 31,
      "row_id": 2,
      "slot_id": 0,
      "transformations": [
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet124",
              "setting": "{\"interface\": {\"speed\": \"100000\", \"lane_map\":
\"9,10,11,12\"}}",
              "speed": {
```

```
                "unit": "G",
                "value": 100
              },
              "state": "active"
            }
          ],
          "is_default": true,
          "transformation_id": 1
        },
        {
          "interfaces": [
            {
              "interface_id": 1,
              "name": "Ethernet124",
              "setting": "{\"interface\": {\"speed\": \"40000\", \"lane_map\": \"9,10,11,12\"}}",
              "speed": {
                "unit": "G",
                "value": 40
              },
              "state": "active"
            }
          ],
          "is_default": false,
          "transformation_id": 2
        }
      ]
    }
  ],
  "selector": {
    "manufacturer": "Dell|DELL",
    "model": "Z9100-ON",
    "os": "SONiC",
    "os_version": ".*"
  },
  "slot_count": 0,
  "software_capabilities": {
    "lxc_support": false,
    "onie": true
  }
}
```

PART 10

Design

CHAPTER 25

Logical Devices

IN THIS CHAPTER

What are Logical Devices

IN THIS SECTION

Logical devices are abstractions of physical devices. They define device capabilities without defining any vendor-specific information. This enables you to design your network fabric based on a common set of form factors (ports, speeds and roles) before selecting underlying hardware. Some applications of logical devices include:

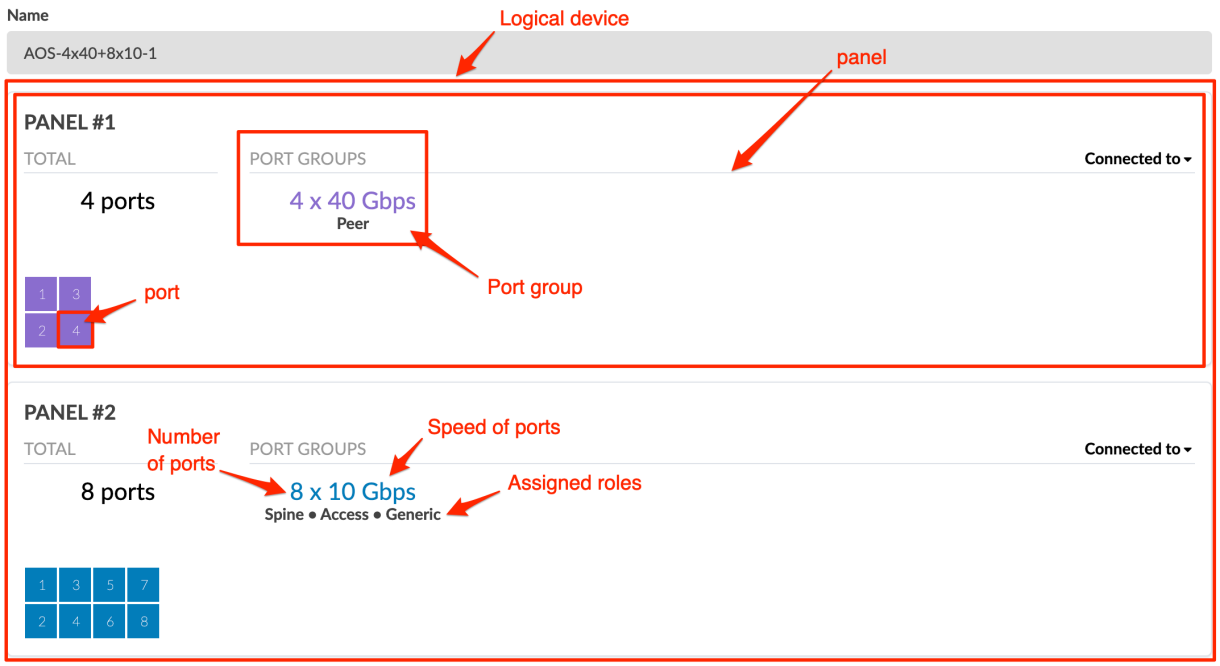
- Specifying speed and roles for specific ports (For example, the 48th port is always a leaf, or the speed of the 10th port is always 1 Gbps).
- Preparing for port speed transformations (For example, transforming one - 40 GbE port into four - 10 GbE ports).
- Using non-standard port speeds (For example, for a 1 GbE SFP in a 10 GbE port, the underlying hardware is automatically configured correctly.)

- Solving for automatic cable map generation that takes into account failure domains on modular systems (for example, a line card).

Logical Device Details

Logical devices include the following details:

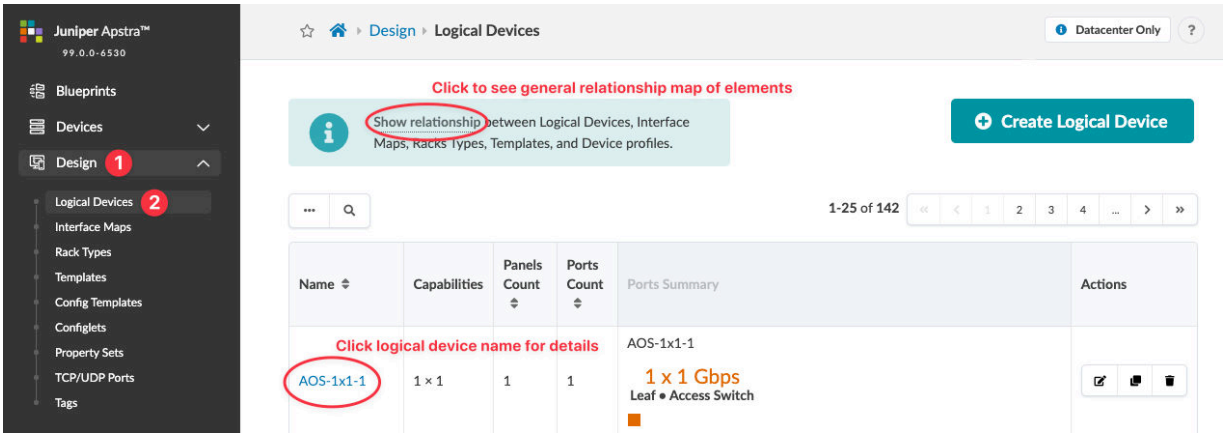
Name	The name is a unique identifier for the logical device.
Panel	Logical devices include one or more panels. A panel is the port layout based on IP fabric, forwarding engine, line card (slot) or physical layout. A panel contains one or more port groups.
Port Group	A port group is a collection of ports with the same speed and role(s).
Number of ports	Port groups specify the number of ports in that group.
Speed	Port groups specify the speed of the ports in that group.
Roles	• Superspine - For superspines facing spines on 5-stage Clos data center fabric. • Spine - For spines facing leaf devices, or for spines facing superspines on 5-stage Clos data center fabric. • Leaf - For leaf devices facing spine or generic systems. • Access (Junos only) - Port is configured to face an access device. To learn more about this feature and its limitations, contact "Juniper Support" on page 1374 . • Peer (link between two leaf devices) - Used for MLAG domains to provide a trunk between two leaf switches. • Unused - Configuration is not rendered and ports are not allocated (use to specify a dead port, for example). • Generic - Certain roles are not specified in logical devices (for example, a firewall, external router, bare metal server, or load balancer).



Logical devices are used in interface maps, rack types and templates in Day 0 operations. They are also used in Day 2 operations.

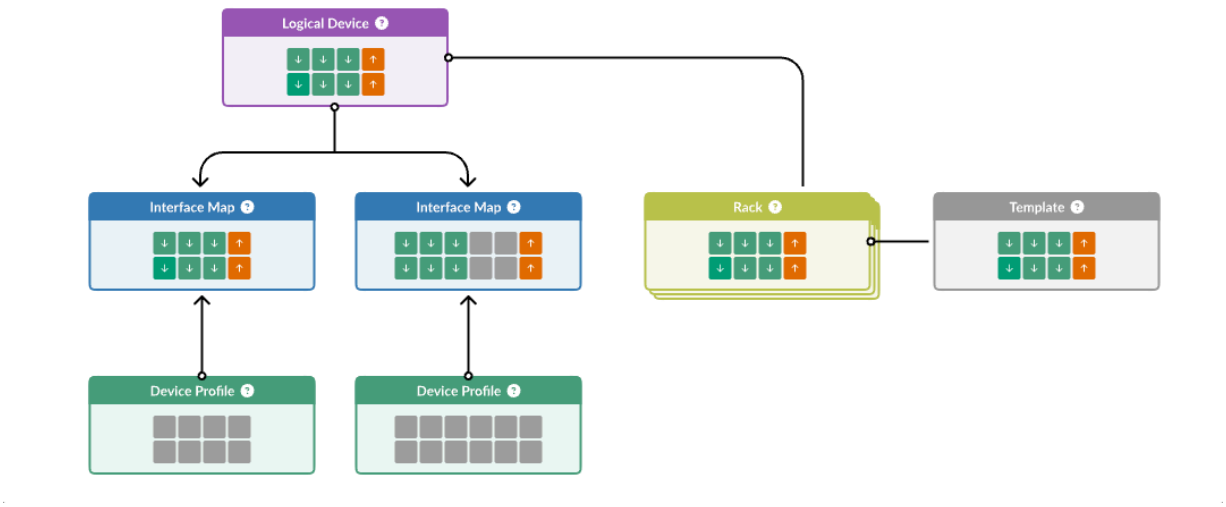
Logical Devices in the Apstra GUI

From the left navigation menu in the Apstra GUI, navigate to **Design > Logical Devices** to go to the logical devices table in the design (global) catalog.



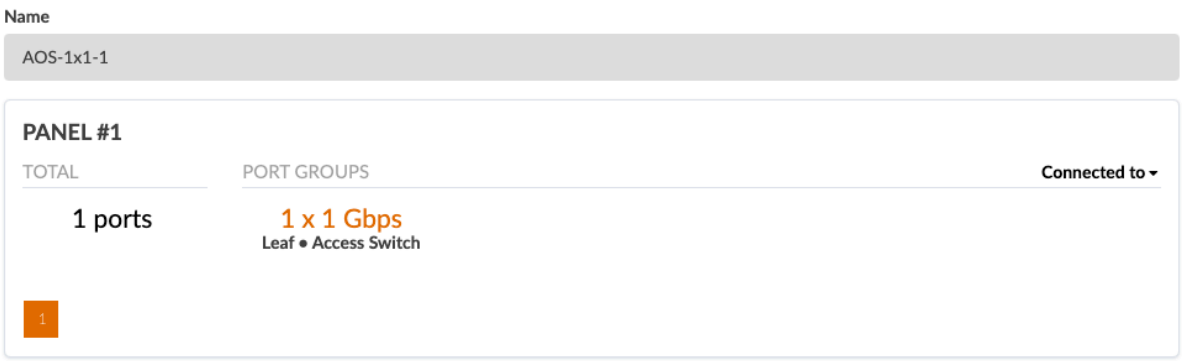
To see how design elements and device profiles are related to each other, click **Show relationship** (new in Apstra version 5.0.0). This is helpful if you're new to the Apstra environment.

Static Conceptual Graphic for Design Elements



Many logical devices are predefined for you. To search for a logical device by its name and/or capabilities, click the **Search** button (magnifying glass) and enter your criteria.

Click a logical device name to go to its details.

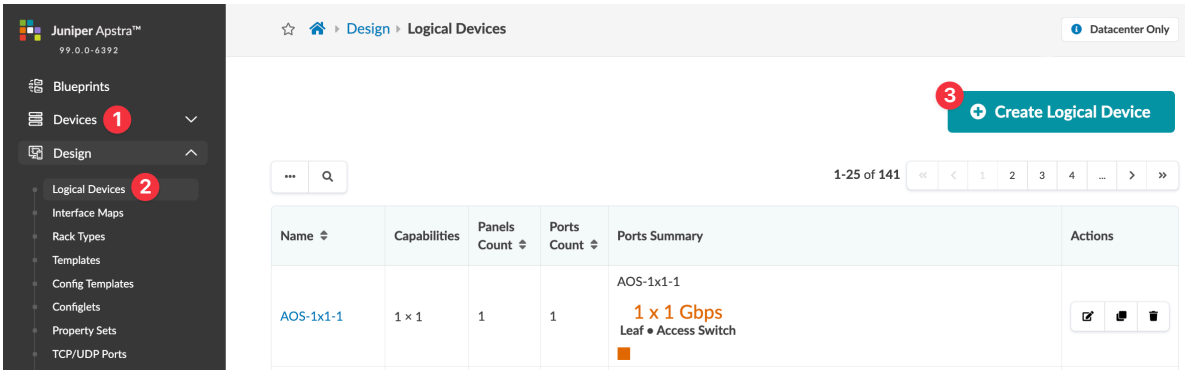


You can create, edit and delete logical devices.

Create Logical Device

If none of the predefined logical devices meet the capability requirements of your qualified device, you can create a custom one.

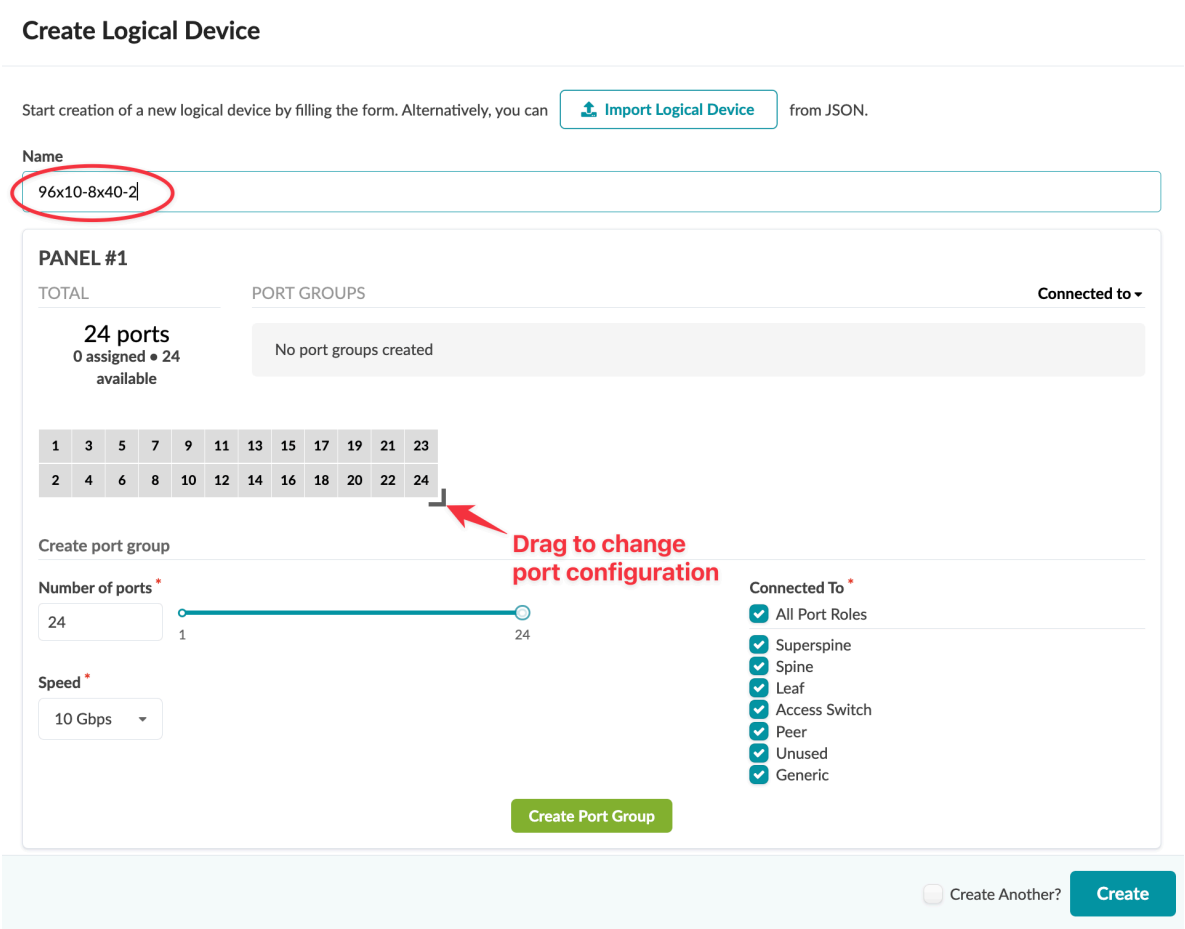
1. From the left navigation menu, navigate to **Design > Logical Devices** and click **Create Logical Device**.



2. In the dialog that opens, enter a unique name for the logical device (64 characters or fewer).

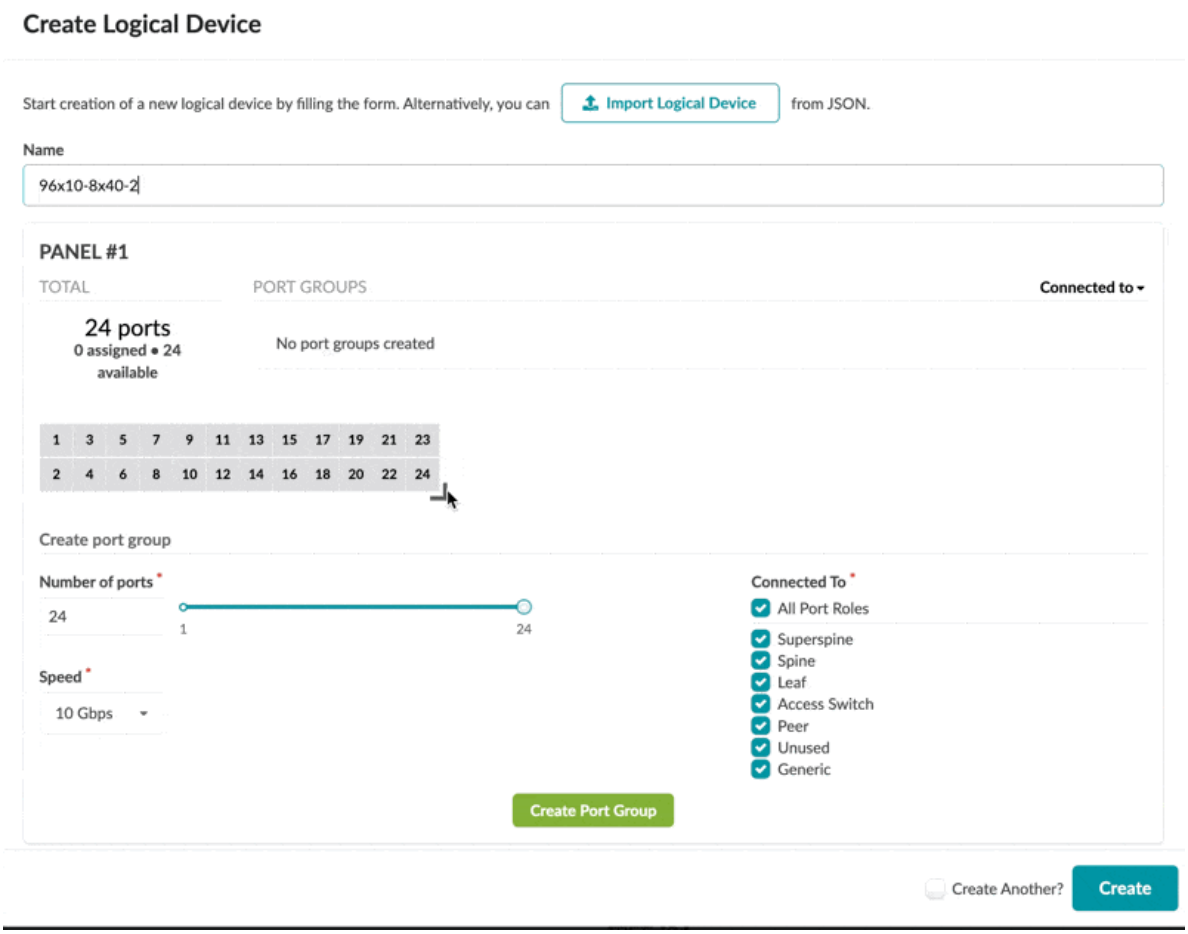
   Using descriptive names makes it easier for you to find later. In our example, we're naming the logical device **96x10-8x40-2**, which represents the following configuration:

   - **96x10** - a panel with one port group consisting of 96 ports at 10 Gbps
   - **8x40** - a panel with one port group consisting of 8 ports at 40 Gbps
   - **2** - number of panels (rack units)



3. Specify the number and arrangement of ports by dragging from the bottom-right corner of the port layout.

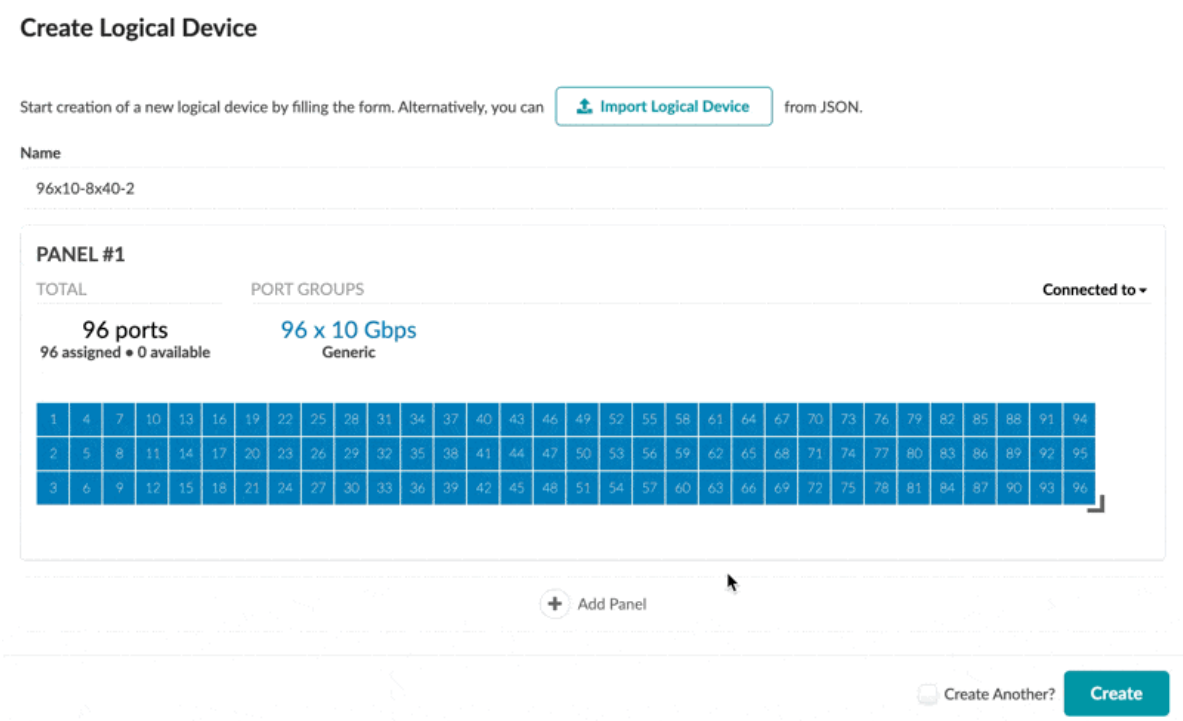
   The default panel layout consists of 24 ports, 2 rows of 12 ports each. In our example, the first panel has 96 ports, 3 rows of 32 ports each.



4. To create a port group, specify the **Number of ports** and their **Speed**; select the role(s) for these ports and click **Create Port Group**.
5. If your panel design has multiple port speeds and roles, create additional port groups for them in the same manner. All ports must be assigned to a port group. For any ports that won't be used, create a port group and assign it the *Unused* role.

   In our example, the first panel has 96 ports, all with the same speed and role, so we're using all the ports in one port group.
6. To add a panel, click **Add Panel** (bottom-middle) and repeat the steps as for the first panel.

   Our example has a second panel with 8 ports at 40 Gbps with superspine, spine, and generic roles.



7. Click **Create** (bottom-right) to create the logical device and return to the table view.

Logical devices are used in interface maps, rack types and templates in Day 0 operations. They are also used in Day 2 operations.

RELATED DOCUMENTATION

What are Logical Devices | **845**

Edit Logical Device

If you're editing a logical device that was previously used (embedded) in a rack type or template, that rack type or template is unaffected. This prevents them from being changed unintentionally. If your intent *is* for a rack type or template to use a modified logical device, you can "update the rack type in the template" on page 904.

You can edit custom logical devices. If you edit one of the predefined logical devices (the ones that ship with Apstra), instead of changing it Apstra automatically creates a clone of it, preserving the predefined logical device as is.

1. Either from the table view (Design > Logical Device) or the details view, click the **Edit** button for the logical device to edit.

2. In the dialog that opens, edit the logical device as needed.
   - To change port group details, access the dialog by clicking its description.
   - To add or remove ports from a port group, drag from the bottom-right corner of the port group layout to resize it. If you're adding ports, enter the port speed and role(s).
   - To remove a port group, click the delete button (upper-right).
   - To add a panel, click **Add Panel** and enter relevant port group details.
3. Click **Update** (bottom-right) to update the logical device in the design (global) catalog and return to the table view. If you edit one of the predefined logical devices, the system doesn't change the predefined one; it automatically clones it and adds 'copy' to the end of the existing name.

RELATED DOCUMENTATION

What are Logical Devices | 845

Delete Logical Device

You can delete a logical device if it's not linked to an interface map in the design (global) catalog.

1. Either from the table view (Design > Logical Devices) or the details view, click the **Delete** button for the logical device to delete.
2. In the dialog that opens, click **Delete** to delete the logical device from the design (global) catalog and return to the table view.

RELATED DOCUMENTATION

What are Logical Devices | 845

CHAPTER 26

Interface Maps

IN THIS CHAPTER

What are Interface Maps

IN THIS SECTION

Apstra separates device capabilities from device models. This abstraction enables you to design your network based on capabilities *before* having to select vendors. Interface maps bring the two together by mapping interfaces in logical devices to interfaces in device profiles. When you build your blueprint, you'll assign these interface maps to managed devices in your blueprint.

Some characteristics and capabilities of interface maps include:

- Precisely select device ports, transformations and interfaces.
- Select interfaces in a contiguous or noncontiguous manner.
- Provision QSFP+ breakout ports to transform ports, such 40GbE ports to 10GbE, 100GbE ports to 25GbE, and so on.
- Port breakouts and available speeds affect possible values of the mapping fields.

- The logical device enables you to plan port and panel mappings accordingly. For example, you can assign a network policy that ensures that spine uplink ports on a leaf switch are always the furthest right ports on a panel.
- If a smaller logical device is mapped to a larger physical device, the unmapped ports in the device profile are marked as **Unused** in the interface map.

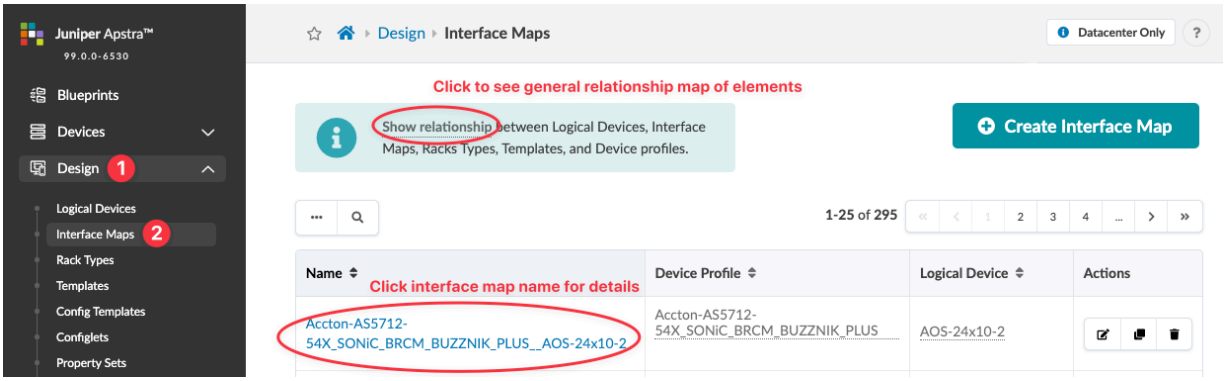
Interface Map Details

Interface maps include the following details:

Logical Device	Abstraction of the physical device.
Device Profile	Physical device characteristics.
Interfaces	Mapping between logical devices and physical devices (device profile)

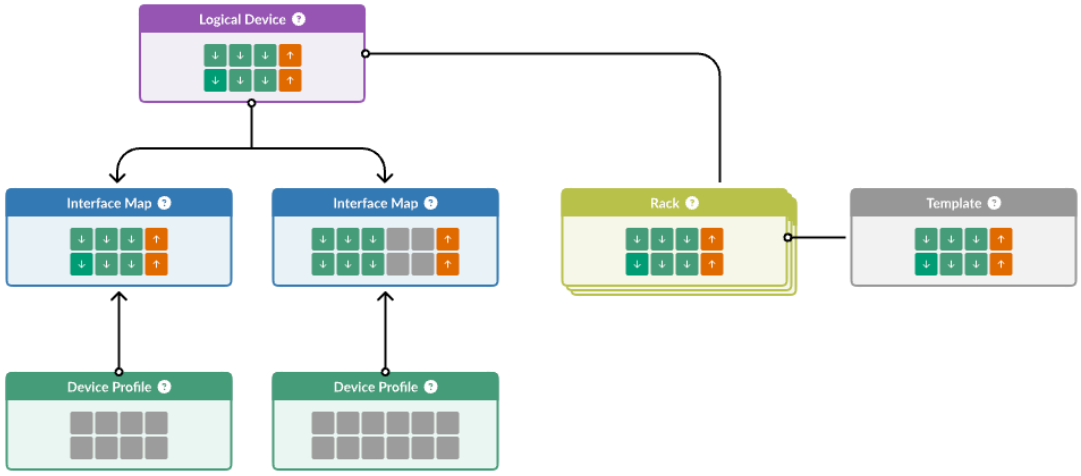
Interface Maps in the Apstra GUI

From the left navigation menu in the Apstra GUI, navigate to **Design > Interface Maps** to go to the interface maps table in the design (global) catalog.



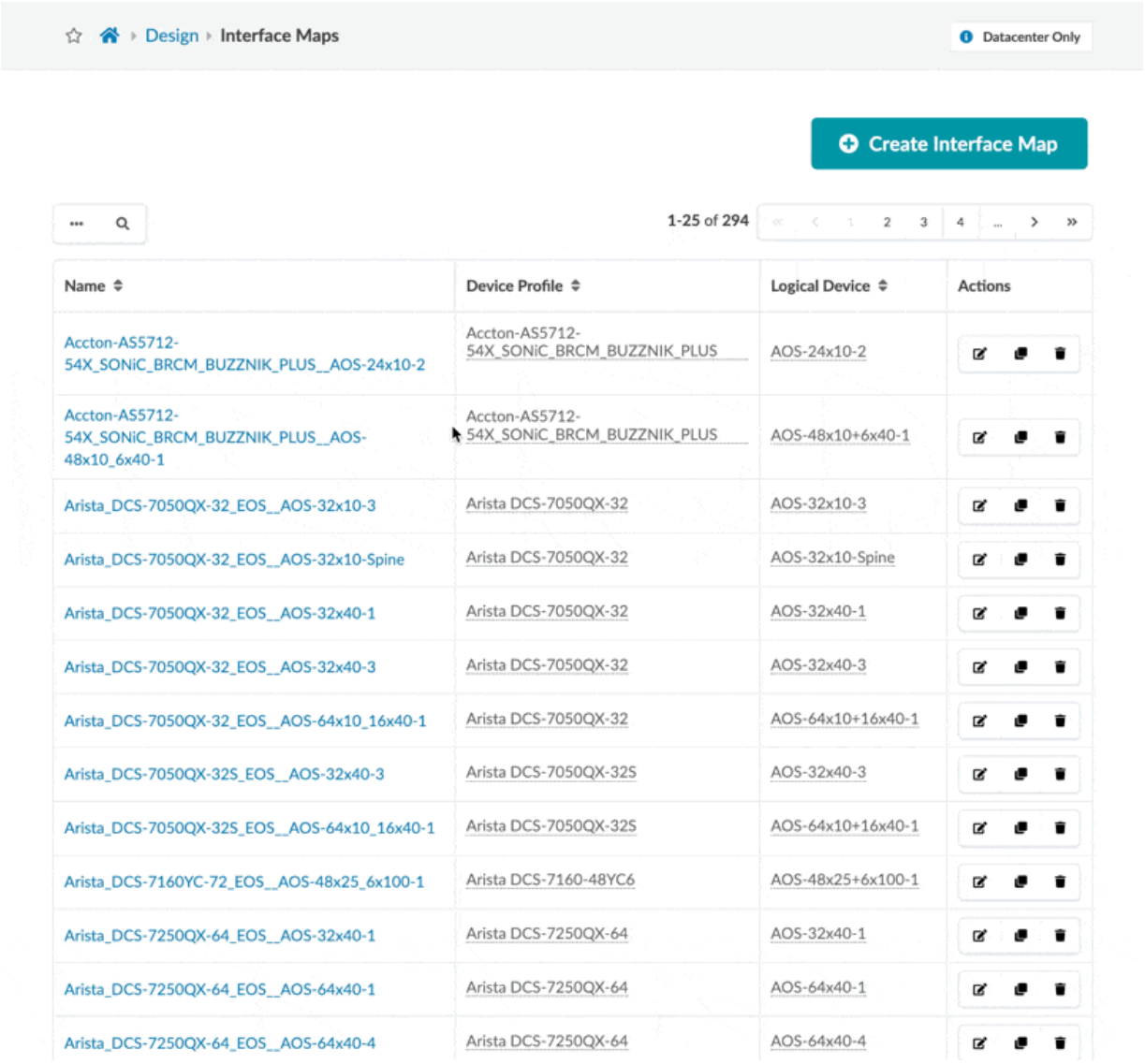
To see how design elements and device profiles are related to each other, click **Show relationship** (new in Apstra version 5.0.0). This is helpful if you're new to the Apstra environment.

Static Conceptual Graphic for Design Elements



Many interface maps are predefined for you. To search for an interface map by its name, device profile and/or logical device, click the **Search** button (magnifying glass) and enter your criteria.

Click an interface map name to go to its details.



You can create, edit and delete interface maps.

RELATED DOCUMENTATION

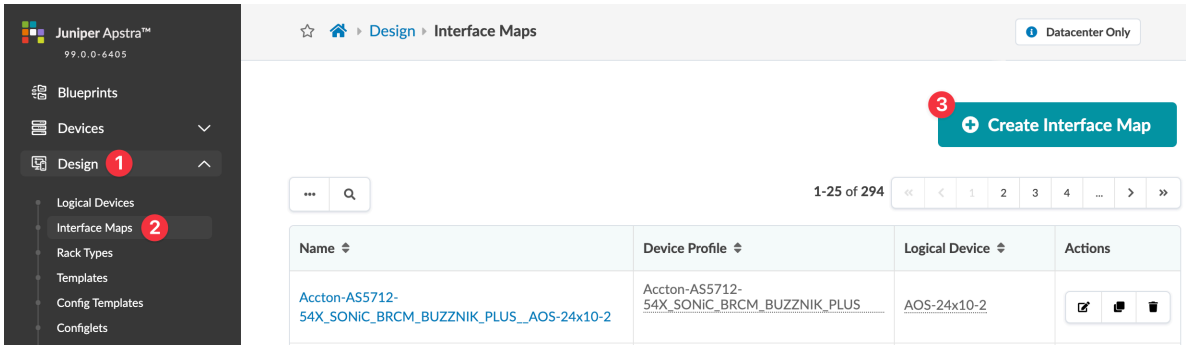
Assign Interface Maps (Device Profiles) | 62

Create Interface Map

Interface maps bridge the capabilities in logical devices with the vendor specifications in device profiles.

Before creating interface maps, make sure the logical device you need is in the "logical device design (global) catalog" on page 845 (Design > Logical Devices) and that the device profile you need is in the "device profiles table" on page 647 (Devices > Device Profiles). Create them, if needed.

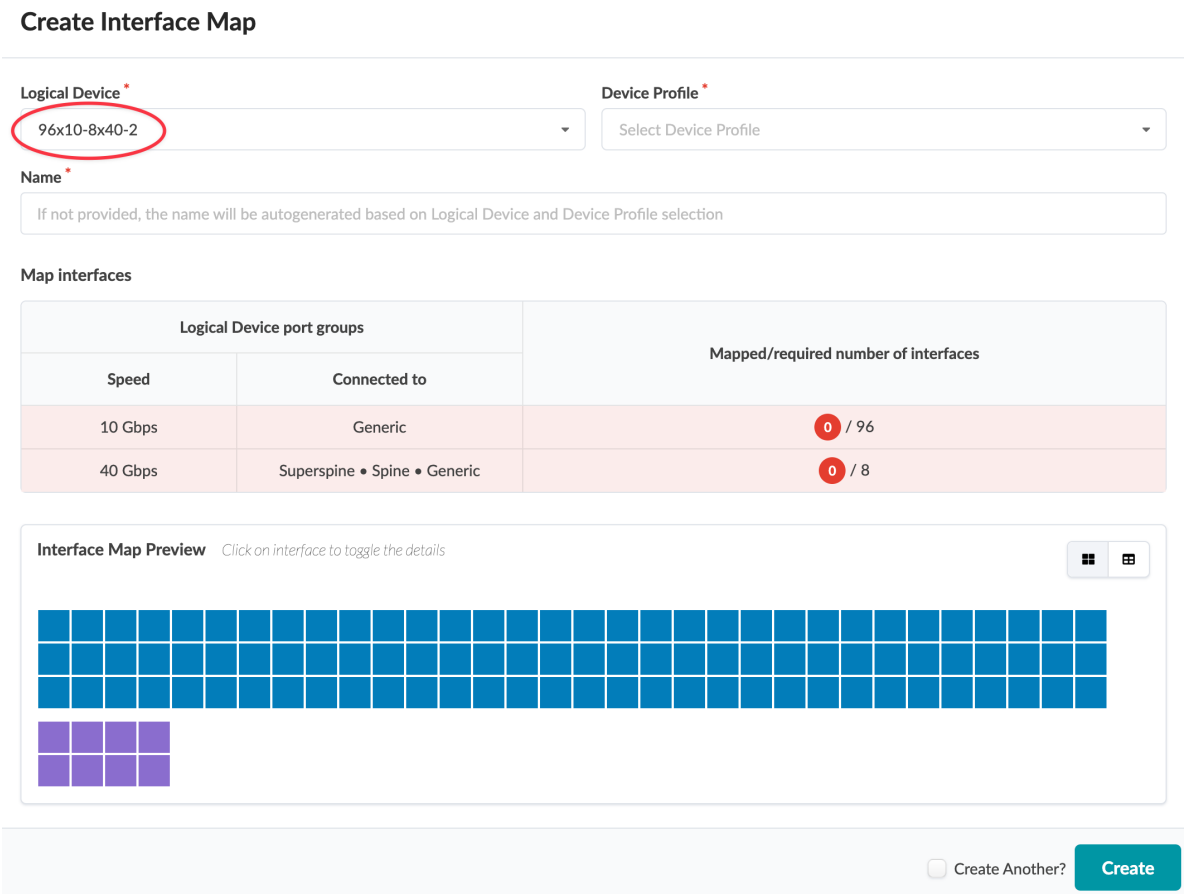
1. From the left navigation menu, navigate to **Design > Interface Maps** and click **Create Interface Map**.



2. In the dialog that opens, select a logical device from the **Logical Device** drop-down list.

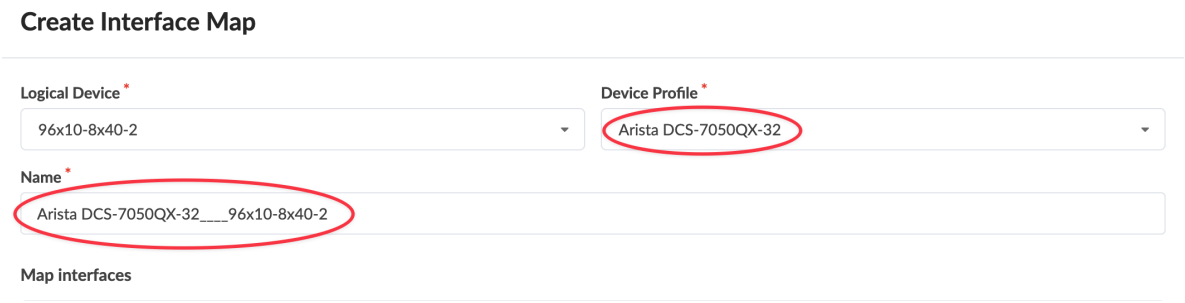
   We're using a custom logical device, **96x10-8x40-2** for our example. If you'd like to follow along, you can "create the logical device" on page 848 before continuing. This logical device has 96 - 10 GbE ports for generic systems (servers) and 8 - 40 GbE ports for uplinks to superspines, spine switches or generic systems (external routers).

   The **Map interfaces** table shows each port group, their speed and roles. The red color indicates that the interfaces haven't been mapped yet.



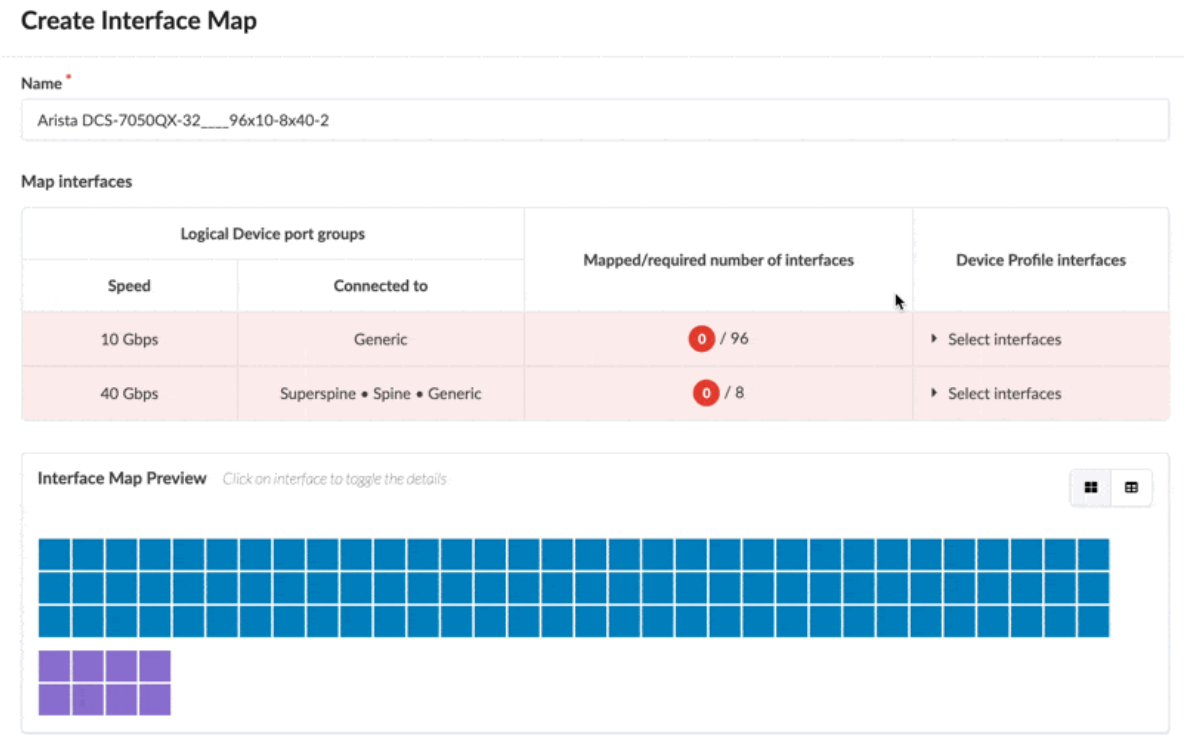
3. Select a device profile that matches the logical device capabilities from the **Device Profile** drop-down list.

   In our example, we're creating an interface map to create dense server connectivity. We'll select **Arista DCS-7050QX-32**. This device has 24 - 40 GbE QSFP+ ports that are transformable (4x10 GbE or 1x40 GbE) and 8 - 40 GbE QSFP+ ports that are not transformable. Our plan is to break out the 24 - 40 GbE transformable ports to 96 - 10 GbE ports.

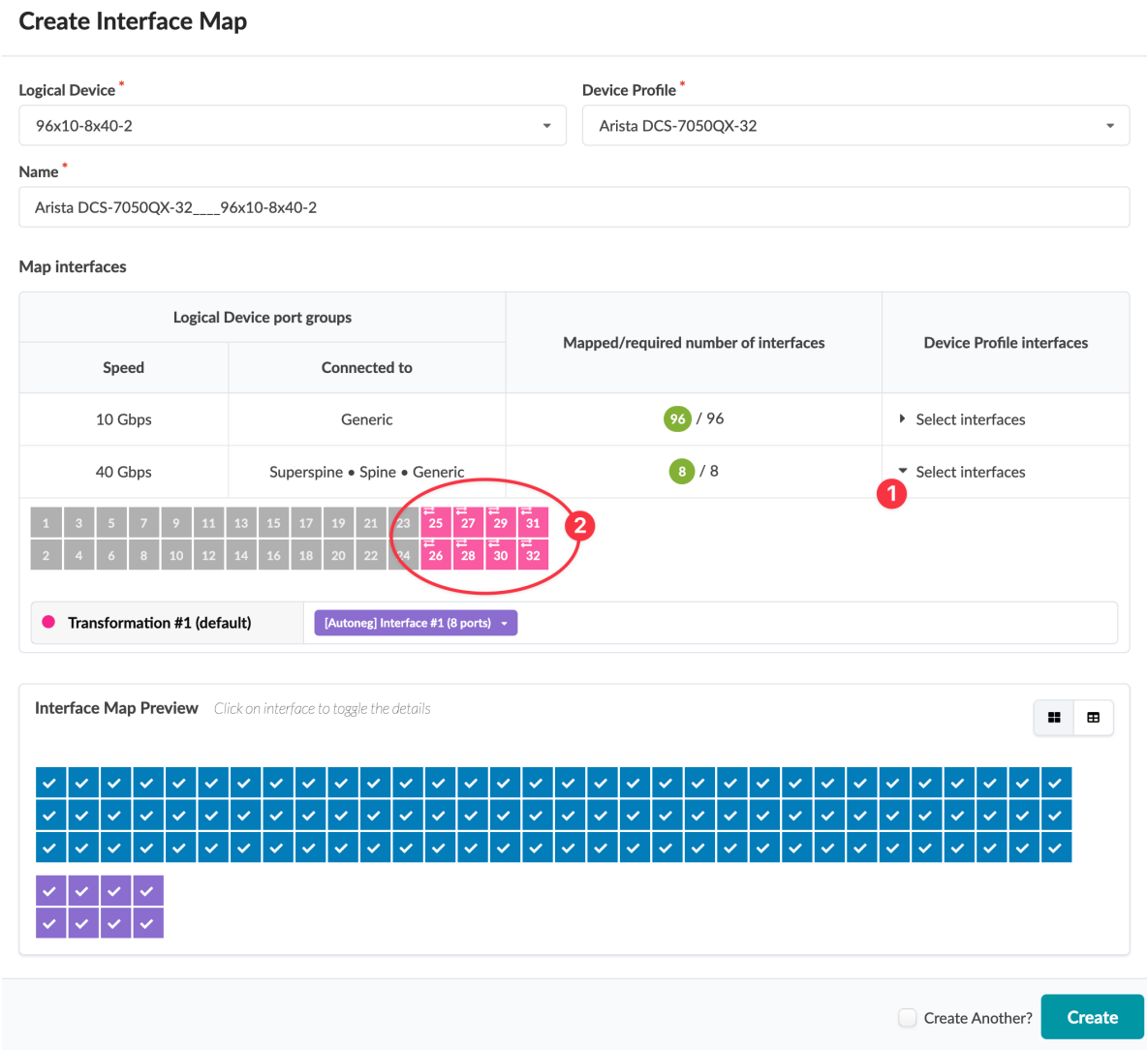


4. As soon as both the logical device and device profile are selected the interface map name is automatically populated. The auto-generated name consists of the concatenation of the names of the selected logical device and device profile. If you prefer, you can enter your own unique name (64 characters or fewer).

5. Now let's map the interfaces. In the first port group row, in the **Device Profile Interfaces** column, click **Select interfaces**, then in the port layout that appears, drag from the first port to the 24th port to select them. As you select ports the white numbers turn gray. After you've selected them, check marks appear on the applicable ports in the **Interface Map Preview**. When all interfaces are selected the red circle turns green.



6. For the second port group, click **Select interfaces**, then in the port layout that appears, drag to select the remaining 8 ports .



7. Click **Create** to create the interface map and return to the table view.

Edit Interface Map

If you're editing an interface map that was previously imported (embedded) into a blueprint, that blueprint is unaffected by the interface map change. This prevents the blueprint from being changed unintentionally.

CAUTION: Any changes made to predefined interface maps (the ones that ship with Apstra software) are discarded when Apstra is upgraded. To retain a customized

interface map through Apstra upgrades, clone the predefined interface map, give it a unique name, and customize it instead of changing the predefined one directly.

1. Either from the table view (Design > Interface Maps) or the details view, click the **Edit** button for the interface map to edit.
2. In the dialog that opens, edit the interface map as needed.
3. Click **Update** (bottom-right) to update the interface map and return to the table view.

RELATED DOCUMENTATION

What are Interface Maps | 853

Delete an Interface Map (Design)

1. Either from the table view (Design > Interface Maps) or the details view, click the **Delete** button for the interface map to delete.
2. In the dialog that opens, click **Delete** to delete the interface map from the design (global) catalog and return to the table view.

RELATED DOCUMENTATION

What are Interface Maps | 853

CHAPTER 27

Rack Types

IN THIS CHAPTER

What are Rack Types

IN THIS SECTION

Rack types are abstractions of racks that define the *capabilities* of racks without defining vendor-specific information. By not including any vendor details you can design your racks in the Apstra environment *before* selecting vendor hardware. Rack types specify a number of logical devices and link details. Logical devices are abstractions of physical devices that specify the number of ports on a device, their speed, and their assigned roles. Logical devices can have various roles; the ones applicable to rack types are leaf, access and generic system. The access role applies to devices using Junos OS only. Generic systems are devices that attach compute/storage.

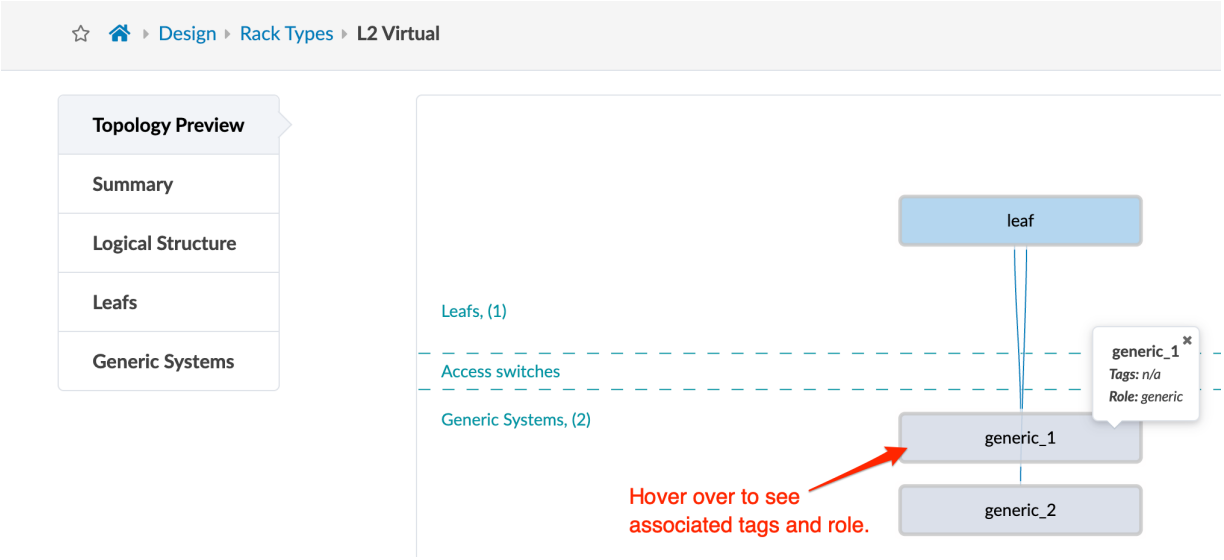
When the majority of racks in your data center use the same leaf hardware with the same link speeds to hosts, uplink speeds to spines and so on, instead of designing and building every single rack in a data center, you can take advantage of the efficiency of designing one rack type and applying it in multiple places.

Rack Type Details

Rack type details are divided into the following sections:

Topology Preview

The **Topology Preview** section shows a visual representation of the logical elements in a rack type. The screenshot below shows the topology preview for the predefined rack type named **L2 Virtual**. It consists of one leaf device and 2 generic systems. (Generic systems attach compute/storage; they connect to a single rack in the topology.)



Summary

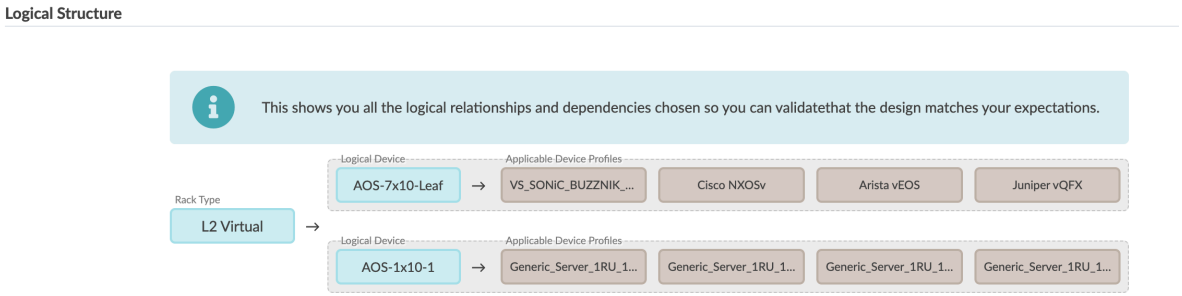
The **Summary** section includes the rack type name, a description (optional), and a designation of whether it's used in a Clos fabric or a full mesh fabric.

Summary	Description
Name (and optional description)	A unique name to identify the rack type, 17 characters or fewer
Fabric connectivity design	• **L3 Clos** - used in 3-stage and 5-stage fabric templates with spine devices. The spine level connects leaf devices to each other. • **L3 Collapsed** - used in collapsed (spineless) templates. Leaf devices are connected directly to each other (full mesh).

Summary	Description
	See "Apstra 5.0.0 Feature Matrix" on page 1481 for supported vendors.

Logical Structure

The **Logical Structure** section (new in Apstra version 5.0.0) is a visual representation of the logical relationships and dependencies in the rack type so you can validate the design against your expectations. The screenshot below is for the predefined rack type named **L2 Virtual**. It includes two different types of logical devices: one for the leaf with seven 10Gbps ports and the other for the generic systems with one 10Gbps port. Each logical device can use one of several device profiles as shown in the logical structure. Device profiles (hardware models) and logical devices (capabilities of devices) are associated with each other via interface maps. If you created a custom interface map that used one of the logical devices below, it would be added as one of the applicable device profiles below.)



Leaf Devices

The **Leaf Devices** section includes the following details:

Leaf Devices	Description
Leaf Name	64 characters or fewer
Leaf Logical Device	Used as ToR leaf switch network device(s)
Links per spine, and Link speed (L3 Clos Only)	Number of leaf-spine links and their speed.

Leaf Devices	Description

Redundancy Protocol	**CAUTION**: Make sure that the intended platform supports the chosen redundancy protocol. For example, L3 MLAG peers are not supported on SONiC, and ESI is supported only on Junos and (as of Apstra version 5.0.0) SONiC.
	• **None** - For single-homed connections
	• **MLAG** - For dual-homed connections. Both switches use the same logical device.
	  • **MLAG Keepalive VLAN ID** - If left blank during rack type creation, 2999 is assigned to the peer link during the build phase. If 2999 conflicts with vendors' reserved ranges, enter a different ID.
	  NOTE: Network device vendors have varying requirements for "reserved" VLAN ID ranges. For example, Cisco NX-OS reserves the VLAN ID range from 3968 to 4094. Arista, by default, uses a VLAN ID range from 1006 to 4094 for internal VLANs for routed ports.
	  • **Peer Links, and Link speed** - Number of links between the MLAG devices, and their speed
	  • **Peer Link Port Channel ID**
	  • **L3 peer links, and Link speed** -Used mainly for BGP peering between border MLAG leaf devices in non-default routing zones. Mainly used for routed L3 traffic to solve EVPN blackhole issues or if upstream routers go down. L3 peer-links act as backup paths for the north-south traffic. Other than border leaf it can be used on any other ToR leaf devices as well as for avoiding blackholing traffic for a VRF.

Leaf Devices	Description
	• **L3 Peer Link Port Channel ID** • **ESI** (Junos and SONiC) - Ethernet Segment ID assigned to the bundled links. Specifying device platforms other then Juniper Junos or SONiC (such as Cisco, Arista) results in blueprint build errors. For information about Juniper ESI, see "Juniper EVPN Support" on page 1429, and "Update ESI MAC msb" on page 435.
Tags	User-specified. Select tags from drop-down list generated from global catalog or create tags on-the-fly (which then become part of the global catalog). Tags used in rack types are embedded, so any subsequent changes to tags in the global catalog do not affect the rack type.

Access Switches

ESI support at the access layer is supported. You can dual-home generic systems (servers) to access switches. We're leveraging EVPN at the access layer to enable ESI-LAG towards the generic system while keeping the L2 only nature of the access switch role.

Supported/Unsupported Topologies for ESI Access:

- Each member of an access switch pair dual-attached to the leaf pair is supported.
- Each member of an access switch pair single-attached to the leaf pair is supported.
- One member of an access switch pair dual-attached to the leaf pair and the other member of an access switch pair single-attached to the leaf pair is not supported.

This is supported on 3-Stage, 5-Stage, and collapsed fabric blueprints. Day 2 topology changes are available through Add/Edit/Remove Racks.

Requirements for the switch model acting as access switch are:

- EVPN-VxLAN with VTEP support is required on the Access Switches.
- L2 VxLAN only is required, L3 VxLAN (RIOT) is not required, and will continue to be available only at the leaf layer.

When creating and managing access switches, follow the general workflow for building a network while taking into account the following options and design considerations.

1. When creating logical devices, on leaf switches facing an access switch, select the port role **access**, and configure ports in the access switch logical device.
2. Create an interface map per standard procedure.
3. Create a rack type with configured access switches.
4. Create a template that uses rack types with access switches.
5. Create a blueprint and build it following the general "workflow" on page 3. You can perform the same tasks as for other blueprints.

The **Access Switches** section includes the following details:

Access Switches	Description
Access Switch Name	64 characters or fewer
Access Switch count	Number of access switches. These switches share the same logical link group.
Logical Device	The logical device that's applied to this access switch.
Redundancy Protocol	• None - For single-homed connections • ESI (Junos only,and SONiC * as a Tech Preview) - Ethernet Segment ID assigned to the bundled links. Specifying device platforms other than Juniper Junos (such as Cisco, Arista) results in blueprint build errors. **NOTE**: ESI as a redundancy protocol on SONiC has been classified as a Juniper Apstra Technology Preview feature. These features are "as is" and voluntary use. Juniper Support will attempt to resolve any issues that customers experience when using these features and create bug reports on behalf of support cases. However, Juniper may not provide comprehensive support services to Tech Preview features. For additional information, refer to the "Juniper Apstra Technology Previews" on page 1781 page or contact "Juniper Support" on page 1374.

Access Switches	Description
	For information about Juniper ESI support, see "Juniper EVPN Support" on page 1429 and for information about ESI, see "Update ESI MAC msb" on page 435. • L3 Peer Links - Number of L3 peer links between both access switches. • Link Speed - Link speed on the peer link interfaces.
Tags	User-specified. Select tags from drop-down list generated from global catalog or create tags on-the-fly (which then become part of the global catalog). Tags used in rack types are embedded, so any subsequent changes to tags in the global catalog do not affect the rack type.
Logical Link	• Name - 64 characters or fewer • Leaf - Leaf configured in Leafs section • Physical link count per individual switch • Link speed • Tags - User-specified. Select tags from drop-down list generated from global catalog or create tags on-the-fly (which then become part of the global catalog). Tags used in rack types are embedded, so any subsequent changes to tags in the global catalog do not affect the rack type.

Generic Systems

The **Generic Systems** section includes the following details:

Generic Systems	Description
Name	64 characters or fewer
Generic system count	Number of systems in the set
Port Channel ID Min, and Max	Port channel IDs are used when rendering leaf device port-channel configuration towards generic systems. default: 1-4096. You can customize this field. All non-default port channel numbers must be unique per *system*, not per *blueprint*.

Generic Systems	Description
Logical Device	The generic system network device
Tags	User-specified. Select tags from drop-down list generated from global catalog or create tags on-the-fly (which then become part of the global catalog). Useful for specifying generic systems as servers or external routers on nodes and links. Tags used in rack types are embedded, so any subsequent changes to tags in the global catalog do not affect the rack type.

Logical Link	• **Name** - 64 characters or fewer
	• **Switch** - Leaf configured in **Leafs** section
	• **LAG Mode**
	  • **LACP (Active)** - Link Aggregation Control Group (LACP) in active mode - This mode actively advertises LACP BPDU even when the neighbor does not.
	  • **LACP (Passive)** - Link Aggregation Control Group (LACP) in passive mode - This mode doesn't generate LACP BPDU until it sees one from a neighbor.
	  • **Static LAG (no LACP)** - Static LAGs don't participate in LACP and will unconditionally operate in forwarding mode.
	  • **No LAG** - This link is not part of a LAG.
	• **Physical link count per individual leaf, and Link speed)** - Number of links from each generic system to each leaf and their speed. If using dual leaf switches, this number should be half of the total links attached to the generic system.
	• **Tags** - User-specified. Select tags from drop-down list generated from global catalog or create tags on-the-fly (which then become part of the global catalog). Useful for specifying generic systems as servers or external routers on nodes and links. Tags used in rack types are embedded, so any subsequent changes to tags in the global catalog do not affect the rack type.

NOTE: You can also add generic systems to blueprints as a Day 2 operation. For more information, see "Add Generic System" on page 127.

Predefined Rack Types

The tables below show details of the predefined rack types that are included with Apstra.

Table 18: Predefined L3 Clos Rack Types without Access Switches

Rack Type Name	Number and Type of Leafs	Leaf Details	Number of Generic Systems	Generic System Details
evpn-mlag	1 MLAG pair	MLAG pair	3	
evpn-single	1 single leaf	single leaf		
L2 Compute	1 single leaf	One panel with forty-eight 10 Gbps ports • Roles: Access / Peer / Generic One panel with six 40 Gbps ports • Roles: Spine / Generic	40	One 10 Gbps link single-homed at **leaf** LAG Mode: No LAG Roles: Leaf / Access
L2 ESI 2x Links	1 ESI group	ESI group	1	
L2 HPC	1 single leaf	single leaf	16	
L2 MLAG 2x Links	1 MLAG pair	MLAG pair	1	
L2 MLAG Leaf	1 MLAG pair			
L2 One Leaf	1 single leaf	single leaf	48	

Table 18: Predefined L3 Clos Rack Types without Access Switches *(Continued)*

Rack Type Name	Number and Type of Leafs	Leaf Details	Number of Generic Systems	Generic System Details
L2 Virtual	1 single leaf	Seven 10 Gbps ports: • 2 spine/leaf • 2 peer • 2 access/generic • 1 generic	2	One 10 Gbps leaf/ access port /// 10 Gbps link single-homed at leaf
L2 Virtual 2xDual	2 single leafs	single leafs	1	
L2 Virtual 2xMLAG	2 MLAG pairs	MLAG pairs	1	
L2 Virtual Dual	2 single leafs	single leafs	2	
L2 Virtual MLAG	1 MLAG pair	MLAG pair	2	
MLAG Compute	1 MLAG pair	MLAG pair	40	

Table 19: Predefined L3 Clos Rack Types with Access Switches

Rack Type Name	Number and Type of Leafs	Leaf Details	Number of Access Switches	Access Switch Details	Number of Generic Systems	Generic System Details
L2 Access 4x	1 single leaf	1 x 40 Gbps links per spine	4 single switches	2 x 10 Gbps leaf_link single-homed at leaf. LAG Mode: LACP (Active)	4	
L2 ESI Acs Dual	1 ESI group		1 ESI group		3	

Table 19: Predefined L3 Clos Rack Types with Access Switches *(Continued)*

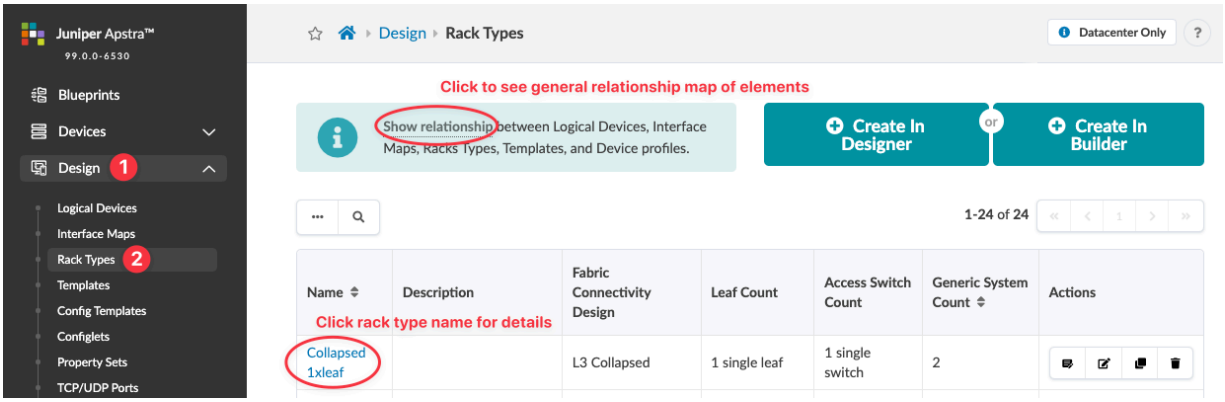
Rack Type Name	Number and Type of Leafs	Leaf Details	Number of Access Switches	Access Switch Details	Number of Generic Systems	Generic System Details
L2 ESI Acs Single	1 ESI group		1 ESI group		2	
L2 MLAG 1x access	1 MLAG pair		1 single switch		2	
L2 MLAG 2acs+1lef	1 MLAG pair, 1 single leaf		3 single switches		4	
L2 MLAG 2x access	1 MLAG pair		2 single switches		2	
L2 One Access	1 single leaf		1 single switch		4	

Table 20: Predefined Collapsed Fabric Rack Types

Rack Type Name	Number and Type of Leaf Devices	Leaf Details	Number of Access Switches	Access Switch Details	Number of Generic Systems	Generic System Details
Collapsed 1xleaf	1 single leaf device	2 x 10 Gbps mesh links Logical Device Roles: • 2 ports: spine / leaf • 2 ports: peer • 2 ports: access / generic • 1 port: generic	1 single switch	1 x 10 Gbps **leaf link** single-homed at **leaf** LAG Mode: LACP (Active) Roles: • 8 ports: leaf / access / peer / generic	2	One 10 Gbps **link** single-homed at **access** LAG Mode: No LAG Roles: • 1 port: Leaf / Access
Collapsed 2xleafs	1 ESI group	Two 10 Gbps mesh links Roles: • 2 ports: spine / leaf • 2 ports: peer • 2 ports: access / generic • 1 port: generic	None	N/A	2	One 10 Gbps link dual-homed at **ESI leaf** LAG Mode: LACP (Active) Roles: • 2 ports: Leaf / Access

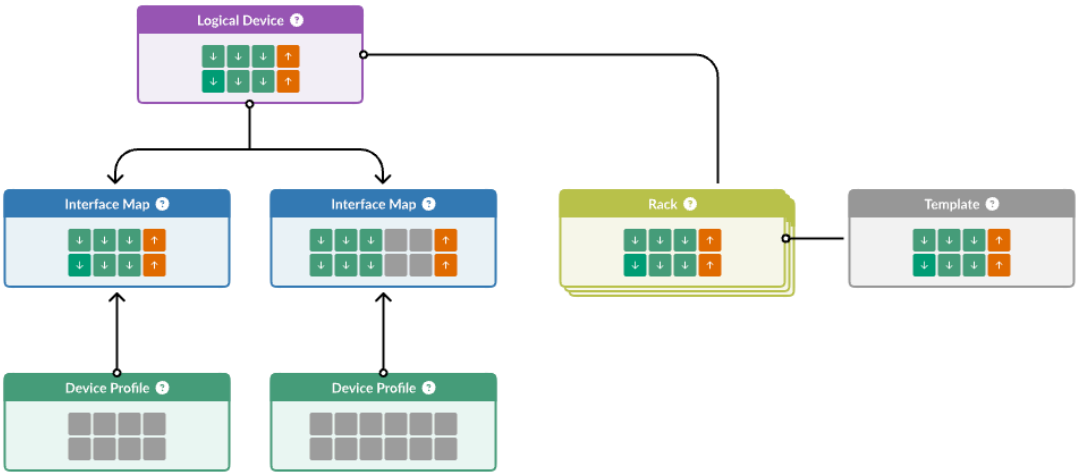
Rack Types in the Apstra GUI

From the left navigation menu of the Apstra GUI, navigate to **Design > Rack Types** to go to the rack types table in the design (global) catalog.



To see how design elements and device profiles are related to each other, click **Show relationship** (new in Apstra version 5.0.0). This is helpful if you're new to the Apstra environment.

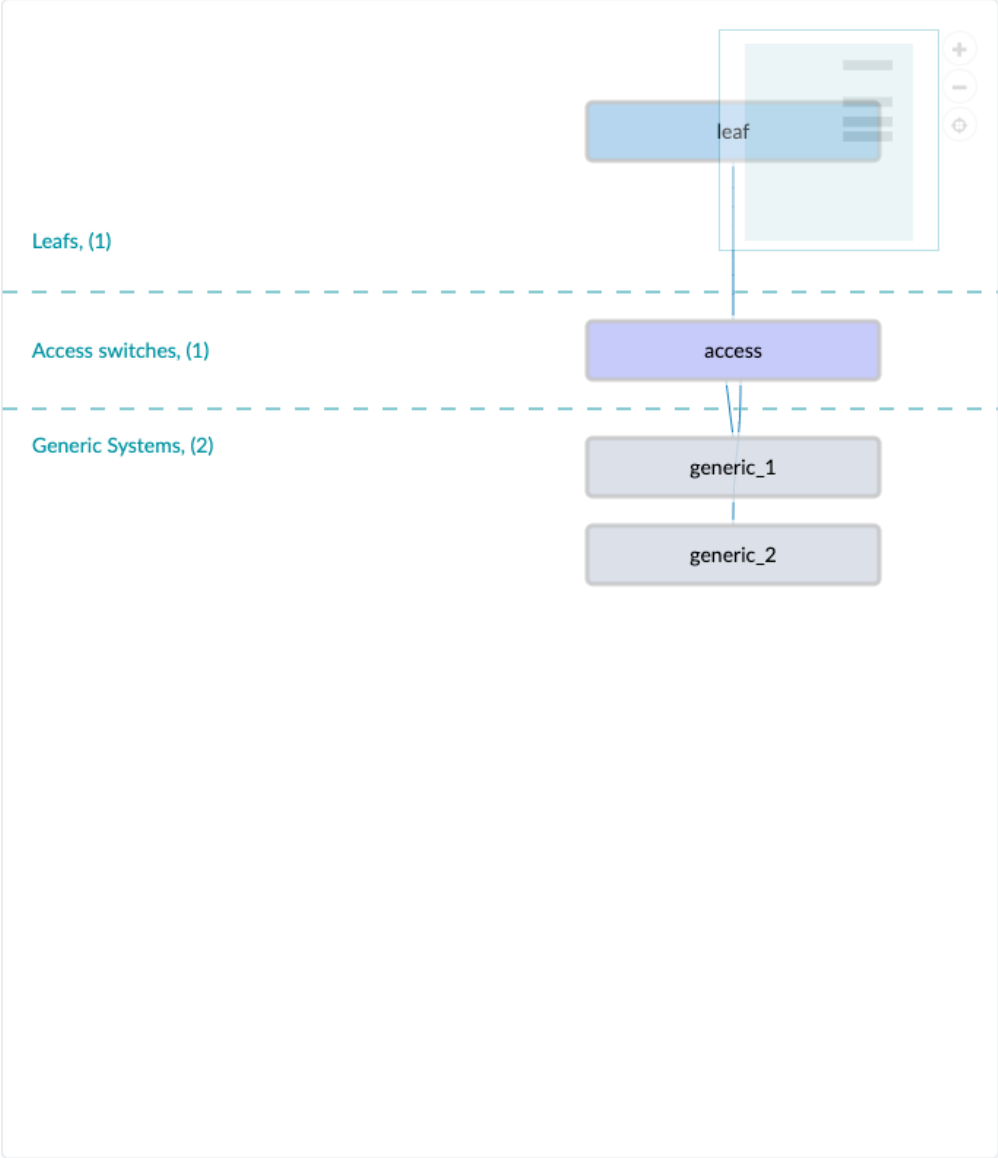
Static Conceptual Graphic for Design Elements



Many rack types are predefined for you. To search for a rack type by its name, click the **Search** button (magnifying glass) and enter your criteria.

Click a rack type name to go to its details.

Topology Preview

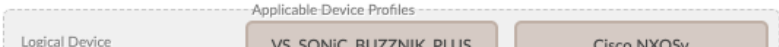


Summary

Display Name	Collapsed 1xleaf
Description	
Fabric Connectivity Design	L3 Collapsed

Logical Structure

This shows you all the logical relationships and dependencies chosen so you can validate that the design matches your expectations.



You can create, edit, and delete rack types.

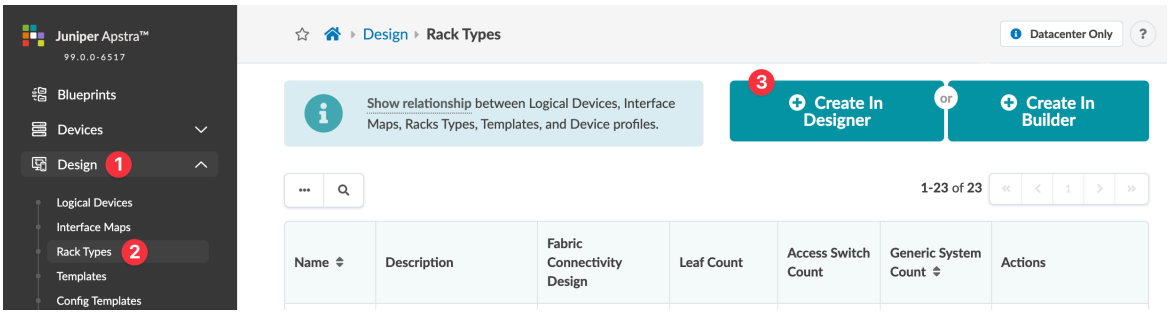
RELATED DOCUMENTATION

- What are Interface Maps | 853
- What are Templates | 889

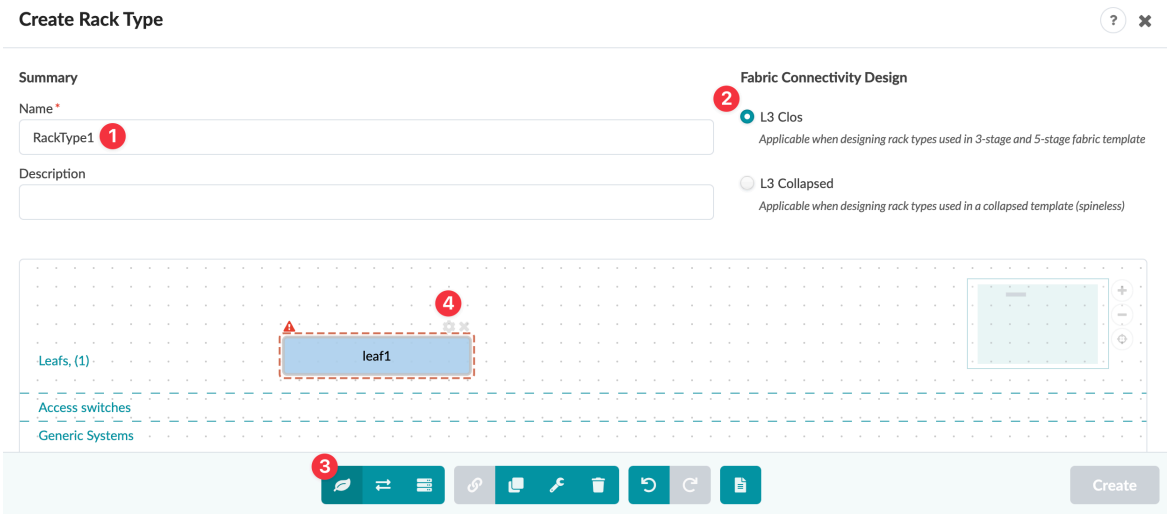
Create Rack Type in Designer

To demonstrate the process for creating rack types with the **Builder** we'll create a rack type for a dual-connected L2 rack with two leaf switches that have 4-100GbE spine links and 40 dual-connected 10GbE "generic systems" on page 126.

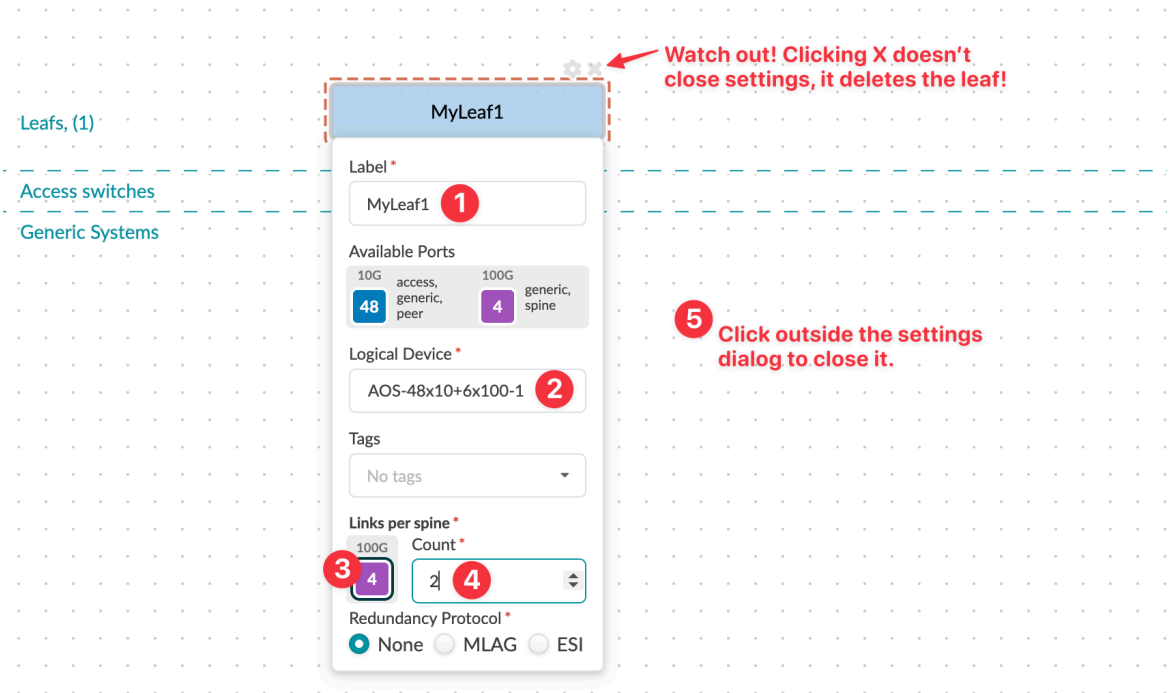
1. Rack types are assigned "logical devices" on page 847. For *your* application, check that the global catalog (Design > Logical Devices) includes the logical devices that you'll need for your rack type. "Create" on page 848 them, if necessary. Our example uses predefined logical devices, so we're ready to create our rack type.
2. From the left navigation menu, navigate to **Design > Rack Types** and click **Create in Designer**.



   The Designer **Create Rack Type** dialog opens.
3. In the **Summary** section, enter a unique rack type name (17 characters or fewer) and an (optional) description, then (for this example) select **L3 Clos** fabric connectivity design for this example.
4. Click the **Add Leaf** button (first selection in bottom row), then in the upper-right of the leaf element that appears, click the gear to open the Settings dialog. (The red triangle indicates that required fields need information.)



5. If you want to use a name other than the default name, enter it in the **Label** field. Leaf devices need to be associated with logical devices. If you selected L3 Clos (which we did for this example), you also need to set the links per spine. Enter a leaf name and select **AOS-48x10+6x100-1** from the **Logical Device** drop-down list (for our example). When you select a logical device, the available ports and roles appear so you can confirm that the logical device has roles you need so you can create links. (If you're building a rack for L3 Clos, only logical devices with a spine role are included in the list. If you're building a rack for L3 Collapsed, all logical devices from the design (global) catalog are included in the list.) Click the link speed box, change the **Links per spine** to **2**), then click outside the leaf dialog to close it.

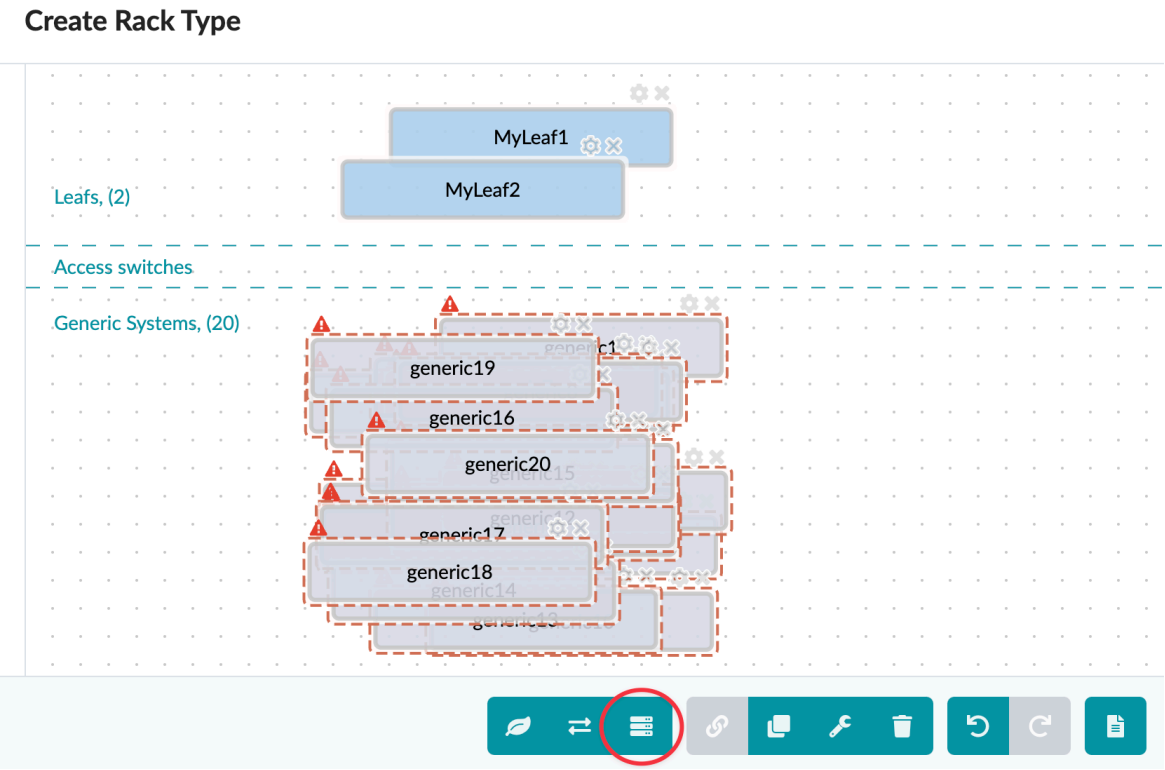


6. Click the **Add Generic System** button (third selection in bottom row), then in the upper-right of the leaf element that appears, click the gear to open the Settings dialog.

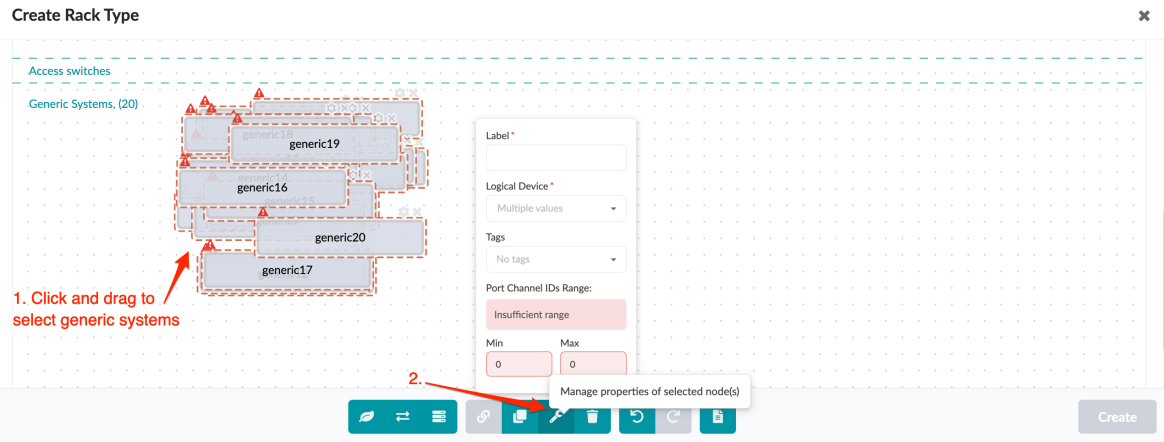
7. If you want to use a name other than the default name, enter it in the **Label** field. Generic systems need to be associated with logical devices. Select **AOS-2x10-1** from the **Logical Device** drop-down list (for our example), then click outside the leaf dialog to close it.
8. Click and drag to select the leaf and the generic system, then click the **Add link** button (fourth selection in bottom row).
9. In the settings dialog that opens, select 2 for Count, then click **Create**.
10. If you want to use a link name other than the default, enter it in the **Label** field.
11. Select LAG Mode LACP Active (if it's not already selected).
12.
13. Enter a name second leaf, select **AOS-48x10+6x100-1** from the **Logical Device** drop-down list, click the link speed box, change the **Links per spine** to **2**), then click outside the leaf dialog to close it.
14. Now for the generic systems. You can create many generic systems, then assign a logical device to them all at the same time. Click the **Add Generic** button (third selection in bottom row) 20 times, (for 20 generic systems), then click and drag across all generic systems to select them.
15.
16. Click the **Add Leaf** button again, enter a name for the second leaf, select **AOS-48x10+6x100-1** from the **Logical Device** drop-down list, click the link speed box, change the **Links per spine** to **2**), then click outside the leaf dialog to close it. (Since the two leaf devices in our example are identical, you could instead click the **Clone** button to duplicate the existing one.)

    NOTE: You can move leafs around on the canvas and when you save your changes in the editor and then reopen it, your leafs will still be where you moved them.

17. Now for the generic systems. You can create many generic systems, then assign a logical device to them all at the same time. Click the **Add Generic** button (third selection in bottom row) 20 times, (for 20 generic systems), then click and drag across all generic systems to select them.



18. Click the **Manage properties of selected node(s)** button in the bottom row, select **AOS-2x10-1** from the Logical Device from the drop-down list (for our example). (The available selections include all logical devices in the design (global) catalog that have a generic role.), then click outside the generic system settings dialog to close it.



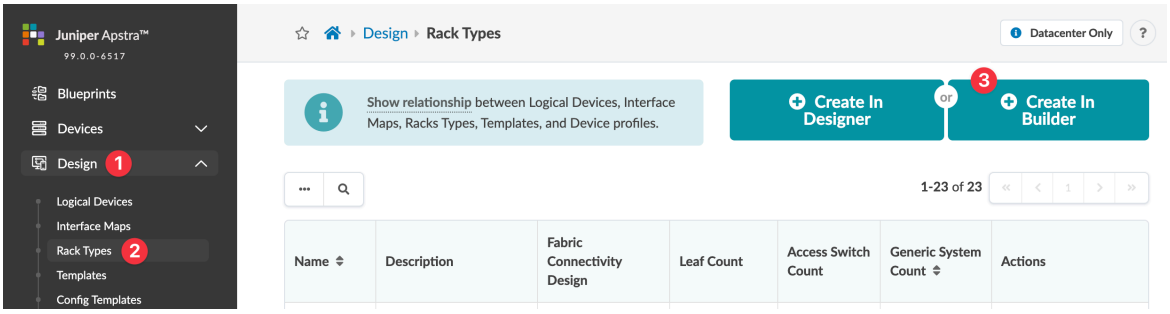
19.

20. Click **Create** to create the rack type in the global catalog and return to the **Rack Types** table view.

Create Rack Type in Builder

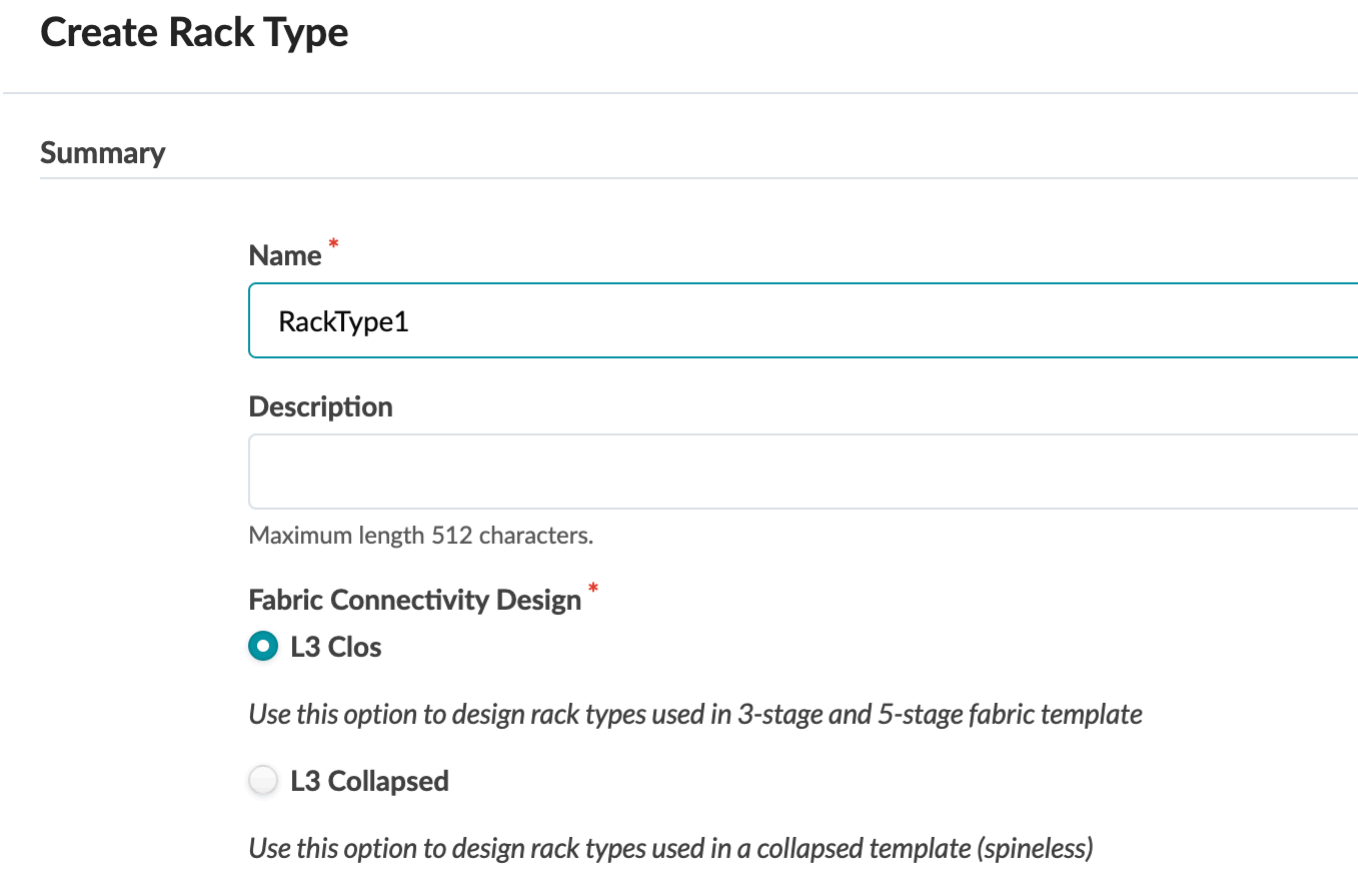
To demonstrate the process for creating rack types with the **Builder** we'll create a rack type for a dual-connected L2 rack with two leaf switches that have 4-100GbE spine links and 40 dual-connected 10GbE "generic systems" on page 126.

1. Rack types are assigned "logical devices" on page 847. For *your* application, check that the global catalog (Design > Logical Devices) includes the logical devices that you'll need for your rack type. "Create" on page 848 them, if necessary. Our example uses predefined logical devices, so we're ready to create our rack type.
2. From the left navigation menu, navigate to **Design > Rack Types** and click **Create in Builder**.

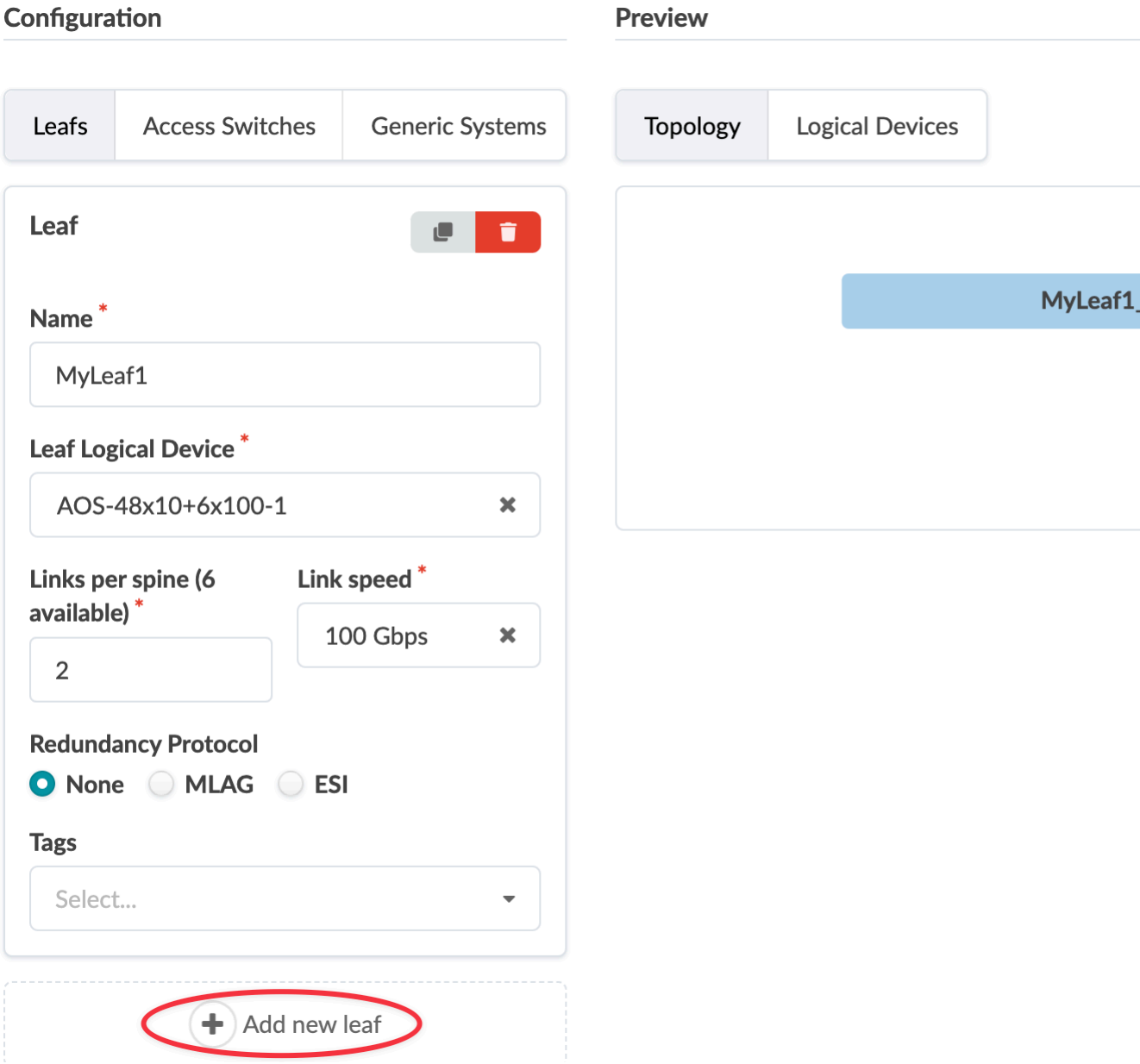


   The Builder **Create Rack Type** dialog opens.
3. In the **Summary** section, enter a unique rack type name (17 characters or fewer) (RackType1 in our example) and an (optional) description, then (for this example) select the **L3 Clos** fabric connectivity design.

> **NOTE**: For rack type parameter details, see "What are Rack Types " on page 862.



4. In the **Configuration** section, select **Leafs**, enter a leaf name (MyLeaf1), select **AOS-48x10+6x100-1** from the **Leaf Logical Device** drop-down list, and change the **Links per spine** to **2**. This logical device represents a device with 48-10Gbps ports that can connect to access switches, peers or generic systems, and 6-100Gbps ports that can connect to spines or generic systems. Notice the **Topology** preview on the right side shows the first leaf.



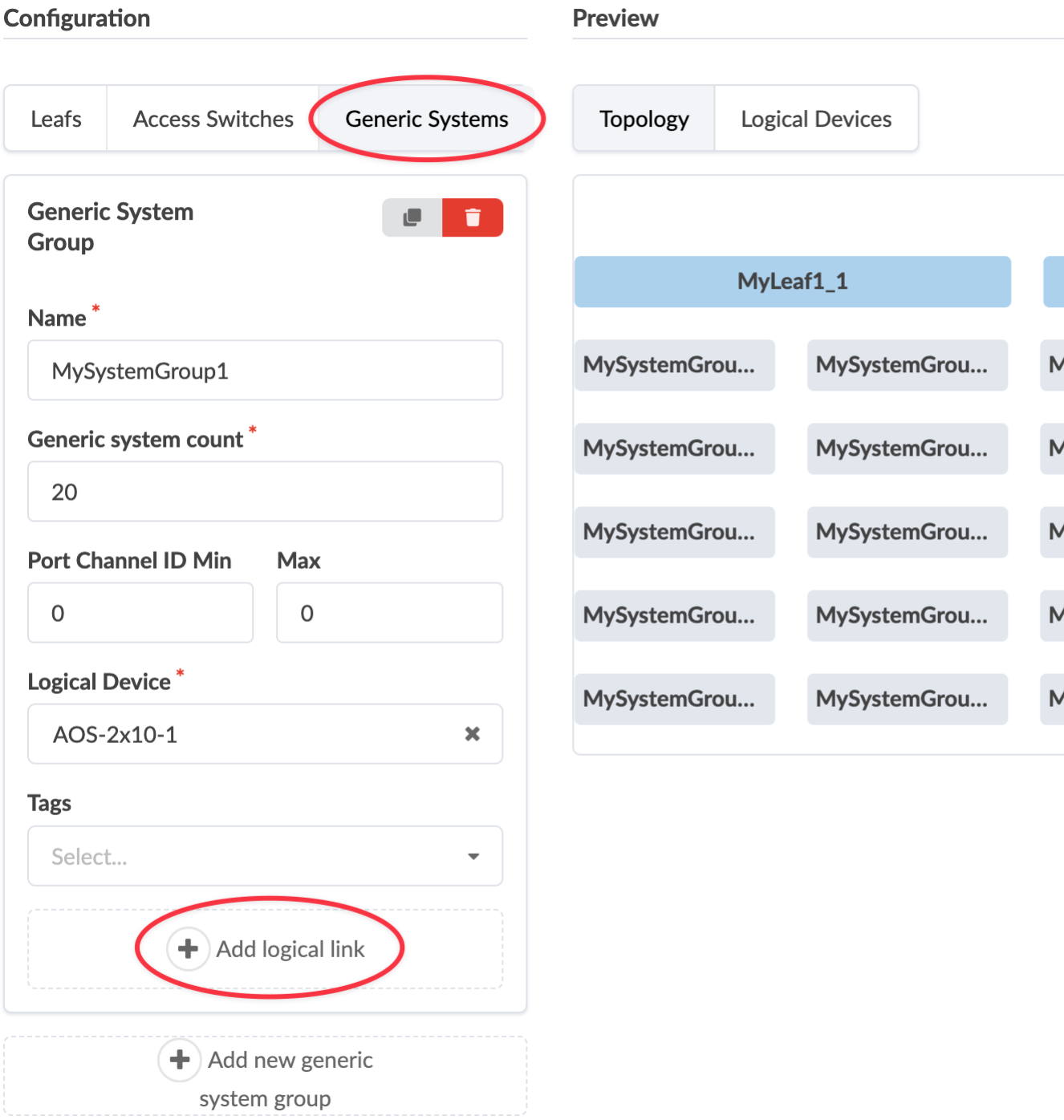
NOTE: Instead of scrolling through the list in the **Leaf Logical Device** drop-down list you can start typing in the field to filter the list based on your input.

5. Click **Add new leaf**, enter a name for the second leaf (MyLeaf2), select **AOS-48x10+6x100-1** from the **Leaf Logical Device** drop-down list, and change the **Links per spine** to **2**. Notice the **Topology** preview on the right side now shows both leaf devices.

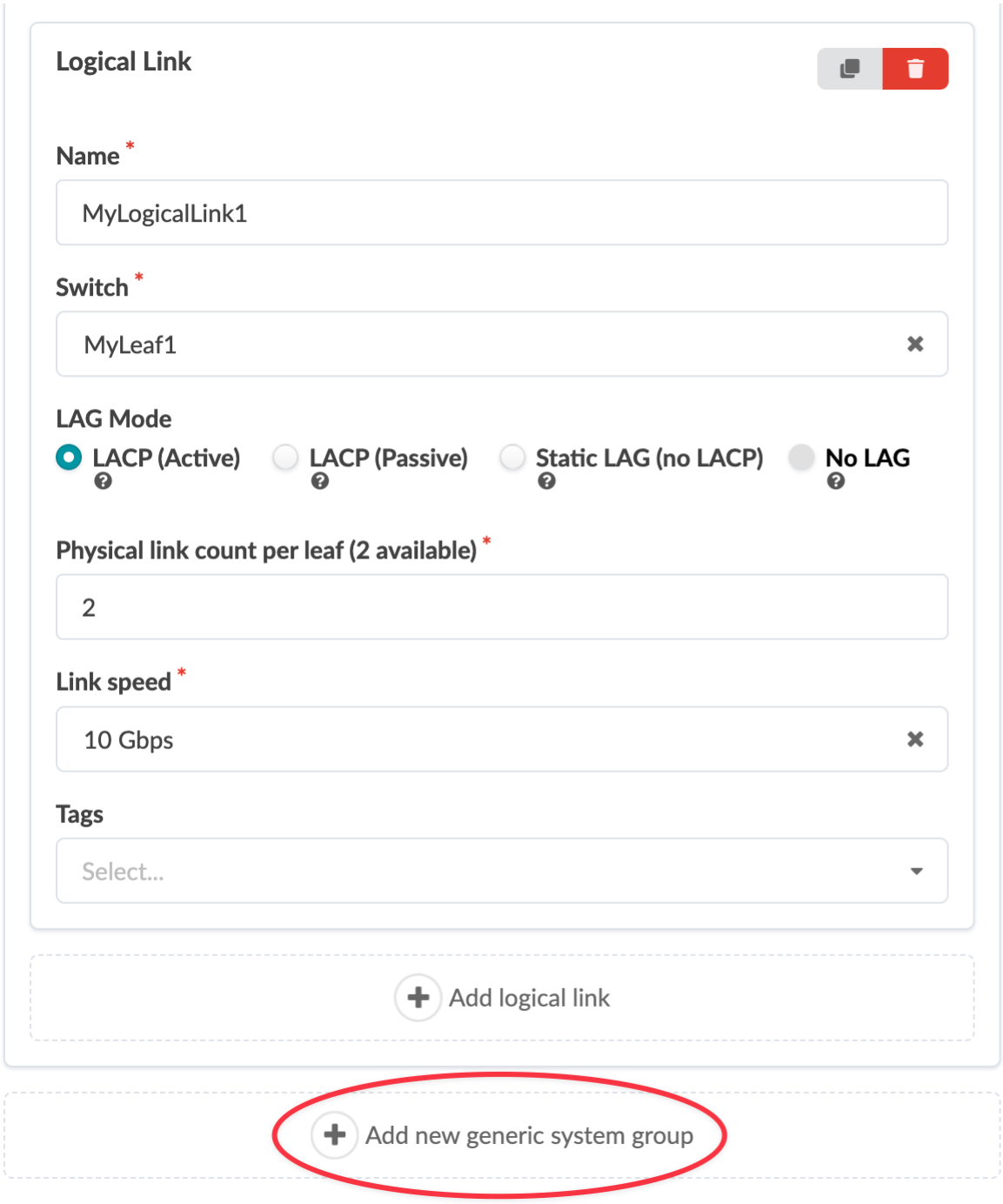
> **NOTE**: If the second leaf is identical to the first one, you could save time re-entering details by clicking the **Clone** button (next to the trash can button) in the first leaf, then changing the name of the second leaf, accordingly.

6. Click **Generic Systems**, click **Add new generic system group**, enter a generic system name (MySystemGroup1), change the **Generic system count** to **20**, then select **AOS-2x10-1** from the **Logical Device** drop-down list. This logical device represents a device with 2-10Gbps ports that can connect to access switches or leaf devices. Notice that the **Topology** preview changes as you configure the rack type.

> **NOTE**: The logical device drop-down list doesn't include logical devices with multiple panels. To specify multiple port groups (each port group has a different speed), create a logical device with a single panel that has multiple port groups. It will then be available in the drop-down list.
>
> The drop-down list also doesn't include logical devices with generic port role (for example, generic-to-generic is not allowed) because of the current built-in validation logic.



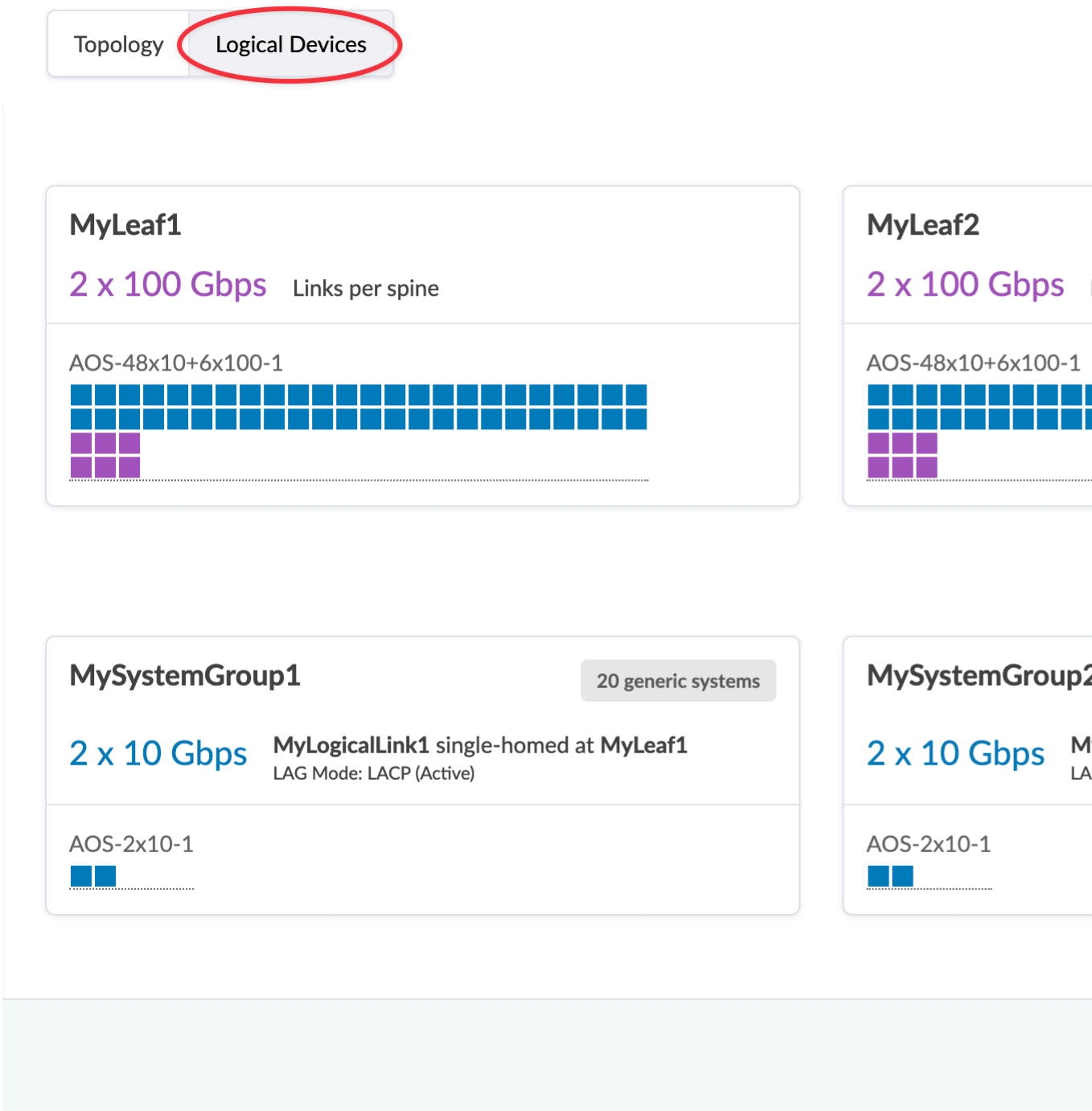
7. Now let's add links between the first leaf device and the generic systems. Click **Add logical link**, enter a logical link name (MyLogicalLink1), select the name of the first leaf you created from the **Switch** drop-down list (MyLeaf1) select **LACP (Active)** for **LAG Mode**, then change **Physical link count per leaf** to **2**.



8. Let's add another group of generic systems, for the second leaf. Click **Add new generic system group**, and create the second group in the same manner. Enter a name (MySystemGroup2), change the **Generic system count** to **20**, then select **AOS-2x10-1** for the logical device.
9. Click **Add logical link**, enter a name (MyLogicalLink2), select the name of the second leaf you created from the **Switch** drop-down list (MyLeaf2), select **LACP (Active)** for **LAG Mode** then change **Physical link count per leaf** to **2**.

NOTE: Since we're using the same generic systems and links for the second leaf, you *could* clone the generic system group, then change the names of the cloned generic systems and logical links, and change the switch to the second leaf.

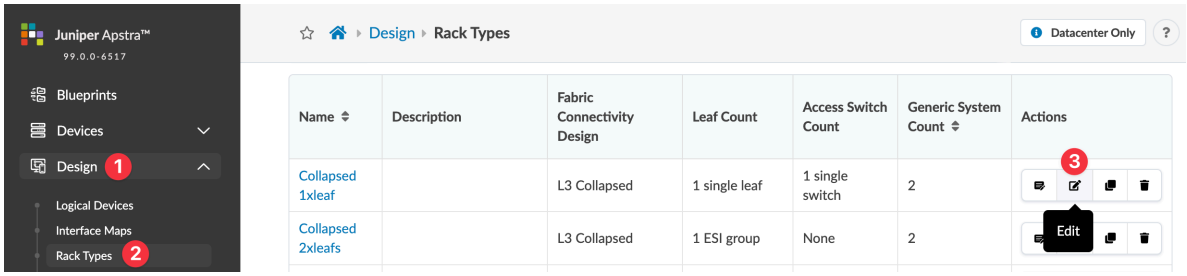
10. To review the logical device details that you've configured in the rack type, click **Logical Devices** in the **Preview** section.



11. Click **Create** to create the rack type in the global catalog and return to the **Rack Types** table view.

Update Rack Type

1. From the left navigation menu, navigate to **Design > Rack Type** and click the **Edit** button for the rack type to update.

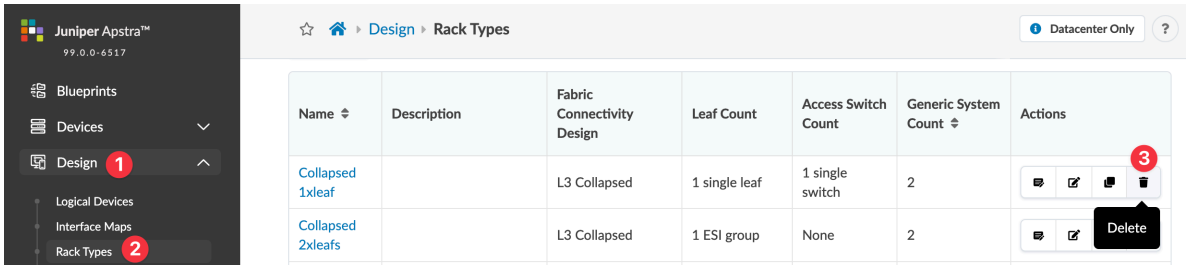


2. Make your changes.
3. Click **Update** (bottom-right) to update the rack type in the global catalog and return to the **Rack Types** table view.

When you change a rack type in the global catalog, it doesn't affect rack types that have already been embedded into templates (or blueprints that were created from those templates). If your intent *is* for a template to use a modified rack type, then after editing the rack type in the global catalog you must edit the template to use it. To change the rack type used in a blueprint, you would edit the rack to replace the rack type with the modified one.

Delete Rack Type

1. From the left navigation menu, navigate to **Design > Rack Type** and click the **Delete** button for the rack type to delete.



2. Click **Delete** to delete the rack type from the global catalog and return to the **Rack Types** table view.

Deleting a rack type from the global catalog does not affect templates and blueprints that previously embedded that rack type.

For information about deleting racks from blueprints, see "Delete Rack" on page 235 (Blueprints > Staged > Physical > Racks).

CHAPTER 28

Templates

IN THIS CHAPTER

What are Templates

IN THIS SECTION

Templates are abstractions of network designs that define the capabilities of a network, its structure and policy intent, without defining any vendor-specific information. They're defined with other elements of abstraction (rack-types, logical devices), as well as other details, as described below. When you're ready to build your network, you'll use a template to create a blueprint. Templates can be rack-based, pod-based, or spineless (collapsed). See the sections below for details on each type of template.

Rack types are used in templates. Apstra provides many predefined templates, but if you do need to create a custom template, you'll use one of the many predefined rack types or one that you've custom-built yourself. When the majority of racks in your data center use the same leaf hardware with the same

link speeds to hosts, uplink speeds to spines and so on, instead of designing and building every single rack in a data center, you can take advantage of the efficiency of using one rack type.

Rack-based Template Details

Rack-based templates define the type and number of racks to connect as top-of-rack (ToR) switches (or pairs of ToR switches). Rack-based templates are divided into the following sections:

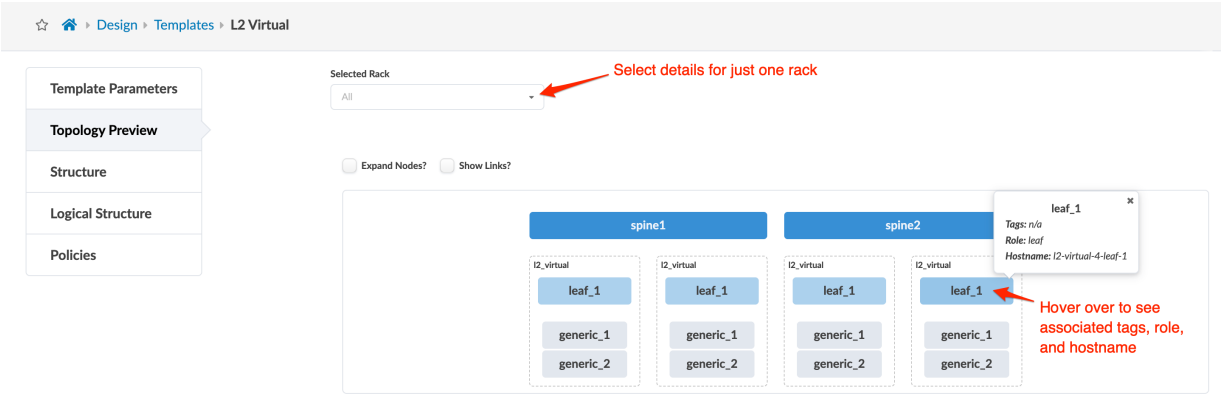
Template Parameters

The **Template Parameters** section includes the template name and type:

- **Name** - A unique name to identify the template. 17 characters or fewer
- **Type** - Rack-based: based on the type and number of racks to connect

Topology Preview

The **Topology Preview** section shows a visual representation of the elements included in the template. The screenshot below is for the predefined rack-based rack type named **L2 Virtual**.



Structure

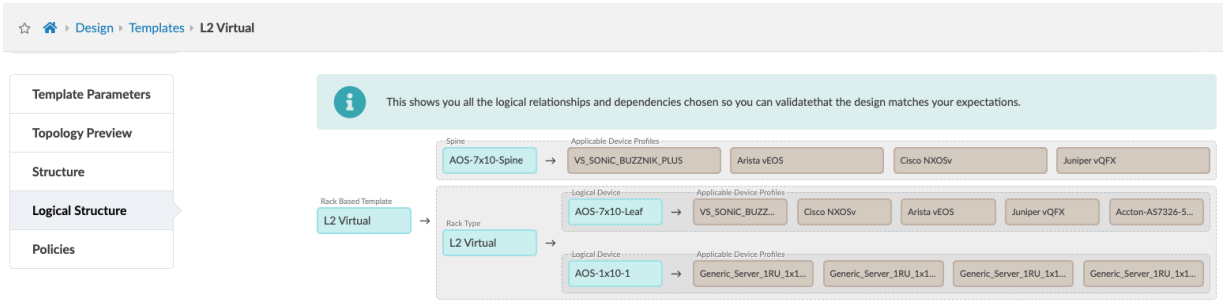
The **Structure** section includes details about the components that make up the template, as shown in the table below:

Structure	Options
Spines	• **Spine Logical Device and Count** - Type and number of spine logical devices

Structure	Options
	• **Links per Superspine Count and Speed** - Number and speed of links to any superspine devices
Tags	User-specified. Select tags from drop-down list generated from global catalog or create tags on-the-fly (which then become part of the global catalog). Useful for specifying external routers. Tags used in templates are embedded, so any subsequent changes to tags in the global catalog do not affect templates.
Rack Types	• Type of rack • Number of each selected rack type ESI-based rack types in rack-based templates without EVPN are invalid.

Logical Structure

The **Logical Structure** section (new in Apstra version 5.0.0) consists of a visual representation of the logical elements in the template. It shows the logical relationships and dependencies so you can validate that the design matches your expectations. The screenshot below is for the predefined rack-based rack type named **L2 Virtual**. In the screenshot, you can see that for the AOS-7x10-Leaf logical device, you could use one of 5 different devices. These associations are the interface maps that use that leaf logical device. If you created a new interface map that included the AOS-7x10-Leaf logical device and a different device profile, it would also appear in this logical structure graphic.



Policies

The **Policies** section includes policies that you can configure in rack-based templates are shown in the table below.

Policy	Options
ASN Allocation Scheme (spine)	• **Unique** - applies to 3-stage designs. Each spine is assigned a different ASN. • **Single** - applies to 5-stage designs. All spine devices in each pod are assigned the same ASN, and all superspine devices are assigned another ASN.
Overlay Control Protocol	• Defines the inter-rack virtual network overlay protocol in the fabric. Overlay control protocol on *deployed* blueprints can't be changed. • **Static VXLAN** (renamed to Pure IP Fabric in Apstra version 4.2.1) - uses static VXLAN routing the Head End Replication (HER) flooding to distribute Layer 2 virtual network traffic between racks. • **MP-EBGP EVPN** - uses EVPN family eBGP sessions between device loopbacks to exchange EVPN routes for hosts (Type 2) and networks (Type 5). Only homogeneous, single-vendor EVPN fabrics are supported. EVPN-VXLAN capabilities for inter-rack virtual networks are dependent on the make and model of network devices used. See "Virtual Networks" on page 251 for more information. External systems must be connected to racks (not spine devices).
Spine to Leaf Links Underlay Type	• **IPv4** - uses addresses from "IPv4 resource pools" on page 934. • **IPv6 RFC-5549** - uses addresses from "IPv6 resource pools" on page 937. Not supported when overlay control protocol is MP-EBGP EVPN.

Policy	Options
	• **IPv6-IPv6 Dual Stack**

Pod-based Template Details

Pod-based templates are used to create large, 5-stage Clos networks, essentially combining multiple rack-based templates using an additional layer of superspine devices. See "5-Stage Clos Architecture" on page 1425 for more information.

Pod-based templates are divided into the following sections:

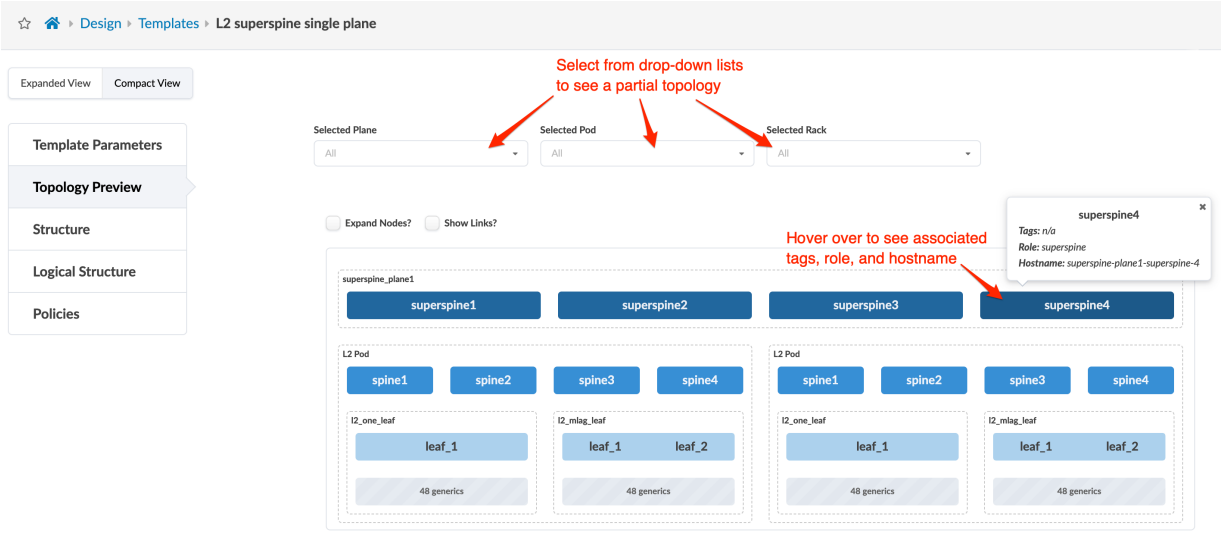
Template Parameters

The **Template Parameters** section includes the template name and the type of template:

- **Name** - A unique name to identify the template. 17 characters or fewer
- **Type** - Pod-based: based on the type and number of rack-based templates to connect

Topology Preview

The **Topology Preview** section shows a visual representation of the elements included in the template. The screenshot below is for the predefined pod-based rack type named **L2 superspine single plane**.



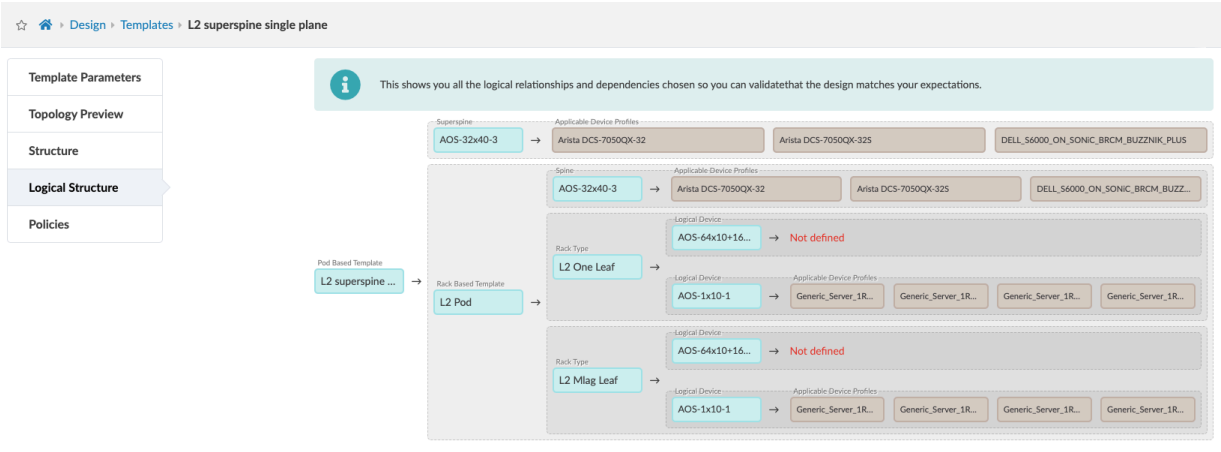
Structure

The **Structure** section includes details about the components that make up the template, as shown in the table below:

Structure	Options
Superspines	• **Superspine Logical Device and Count** - Type and number of superspine logical devices • **Plane Count and Per Plane Count** - Number of planes and number of superspine devices per plane
Tags	User-specified. Select tags from drop-down list generated from global catalog or create tags on-the-fly (which then become part of the global catalog). Useful for specif.ying external routers. Tags used in templates are embedded, so any subsequent changes to tags in the global catalog do not affect templates.
Pods	Type of rack-based template and number of each selected template

Logical Structure

The **Logical Structure** section (new in Apstra version 5.0.0) consists of a visual representation of the logical elements in the template. It shows the logical relationships and dependencies so you can validate that the design matches your expectations. The screenshot below is for the predefined pod-based rack type named **L2 superspine single plane.**



Policies

The **Policies** section includes policies that you can configure in pod-based templates are shown in the table below

Policy	Options
Spine to Superspine Links	• **IPv4** - uses addresses from "IPv4 resource pools" on page 934. • **IPv6 RFC-5549** - uses addresses from "IPv6 resource pools" on page 937. Not supported when overlay control protocol is MP-EBGP EVPN. • **IPv4-IPv6 Dual Stack**
Overlay Control Protocol	• Defines inter-rack virtual network overlay protocol used in the fabric. Overlay control protocol on *deployed* blueprints can't be changed. • **Static VXLAN** (renamed to Pure IP Fabric in Apstra version 4.2.1) - uses static VXLAN routing the Head End Replication (HER) flooding to distribute Layer 2 virtual network traffic between racks. • **MP-EBGP EVPN** - uses EVPN family eBGP sessions between device loopbacks to exchange

Policy	Options
	EVPN routes for hosts (Type 2) and networks (Type 5). Only homogeneous, single-vendor EVPN fabrics are supported. EVPN-VXLAN capabilities for inter-rack virtual networks are dependent on the make and model of network devices used. See "Virtual Networks" on page 251 for more information. External systems must be connected to racks (not spine devices).

Collapsed Template Details

Collapsed templates allow you to consolidate leaf, border leaf and spine functions into a single pair of devices. A full mesh topology is created at the leaf level instead of at leaf-spine connections. This spineless template uses L3 collapsed rack types. Collapsed templates have the following limitations:

- No support for upgrading collapsed L3 templates to L3 templates with spine devices (To achieve the same result you could move devices from the collapsed L3 blueprint to an L3 Clos blueprint.)
- Collapsed L3 templates can't be used as pods in 5-stage templates.
- You can't mix vendors inside redundant leaf devices - the two leaf devices must be from the same vendor and model.
- Leaf-to-leaf links can't be added, edited, or deleted.
- Inter-leaf connections are limited to full mesh.
- IPv6 is not supported.

Collapsed templates are divided into the following sections:

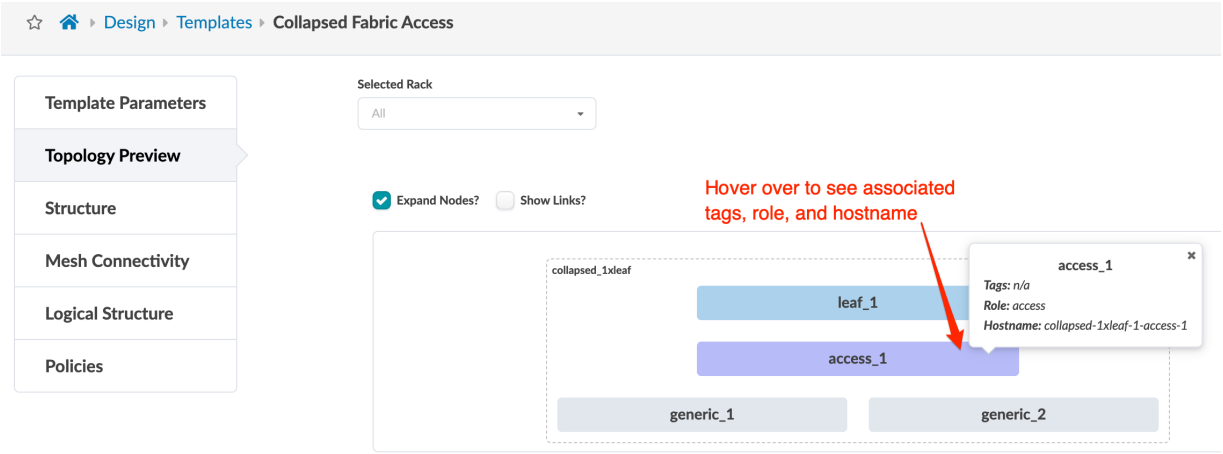
Template Parameters

The **Template Parameters** section includes the template name and the type of template:

- **Name** - A unique name to identify the template. 17 characters or fewer
- **Type** - Collapsed: a spineless template using L3 collapsed rack types

Topology Preview

The **Topology Preview** section shows a visual representation of the elements included in the template. The screenshot below is for the predefined collapsed rack type named **Collapsed Fabric Access**.



Structure

The **Structure** section includes details about the components that make up the template, as shown in the table below:

Structure	Options
Rack Type	Type of L3 collapsed rack and number of each selected rack type

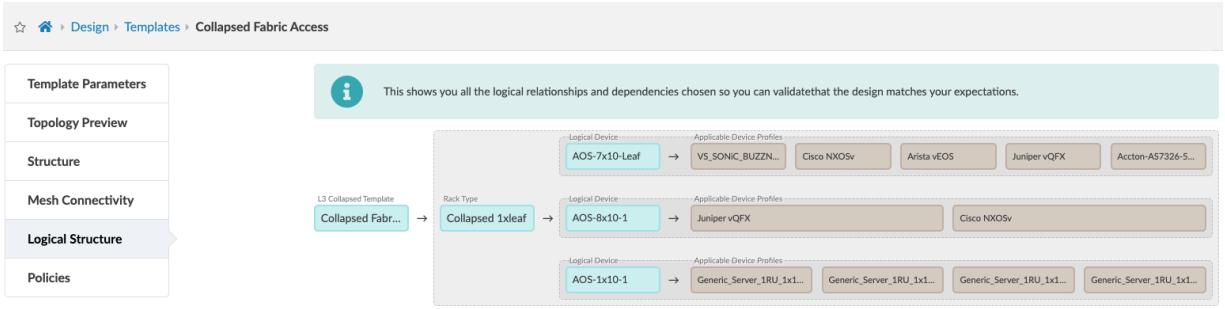
Mesh Connectivity

The **Mesh Connectivity** section includes details about the number of mesh links and their speeds, as shown in the table below:

Mesh Connectivity	Options
Mesh Links Count and Speed	Defines the link set created between every pair of physical devices, including devices in redundancy groups (MLAG / ESI). These links are always physical L3. No logical links are needed at the mesh level.

Logical Structure

The **Logical Structure** section (new in Apstra version 5.0.0) consists of a visual representation of the logical elements in the template. It shows the logical relationships and dependencies so you can validate that the design matches your expectations. The screenshot below is for the predefined collapsed rack type named **Collapsed Fabric Access.**



Policies

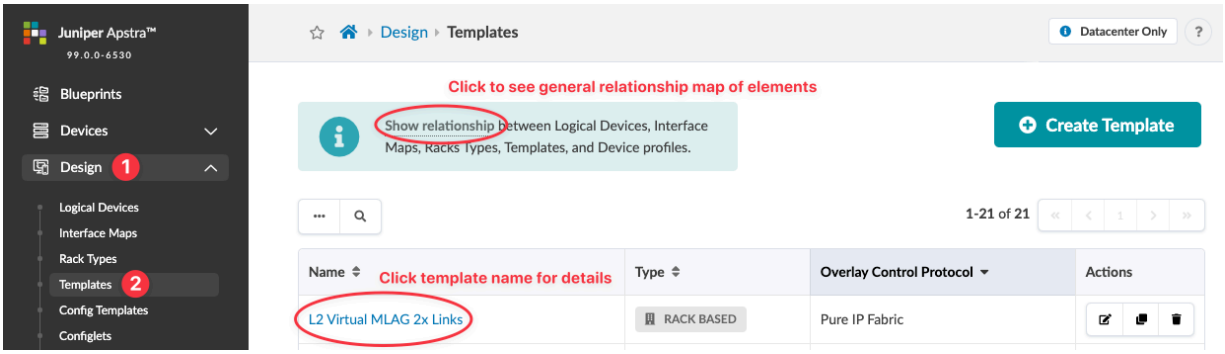
The **Policies** section includes policies that you can configure in collapsed templates are shown in the table below

Policy	OPtions
Overlay Control Protocolegp	• Defines the inter-rack virtual network overlay protocol in the fabric. Overlay control protocol on *deployed* blueprints can't be changed. • **Static VXLAN** (renamed to Pure IP Fabric in Apstra version 4.2.1) - uses static VXLAN routing the Head End Replication (HER) flooding to distribute Layer 2 virtual network traffic between racks. • **MP-EBGP EVPN** - uses EVPN family eBGP sessions between device loopbacks to exchange EVPN routes for hosts (Type 2) and networks (Type 5). Only homogeneous, single-vendor EVPN fabrics are supported. EVPN-VXLAN capabilities for inter-rack virtual networks are dependent on the make and model of network devices used. See "Virtual Networks" on page

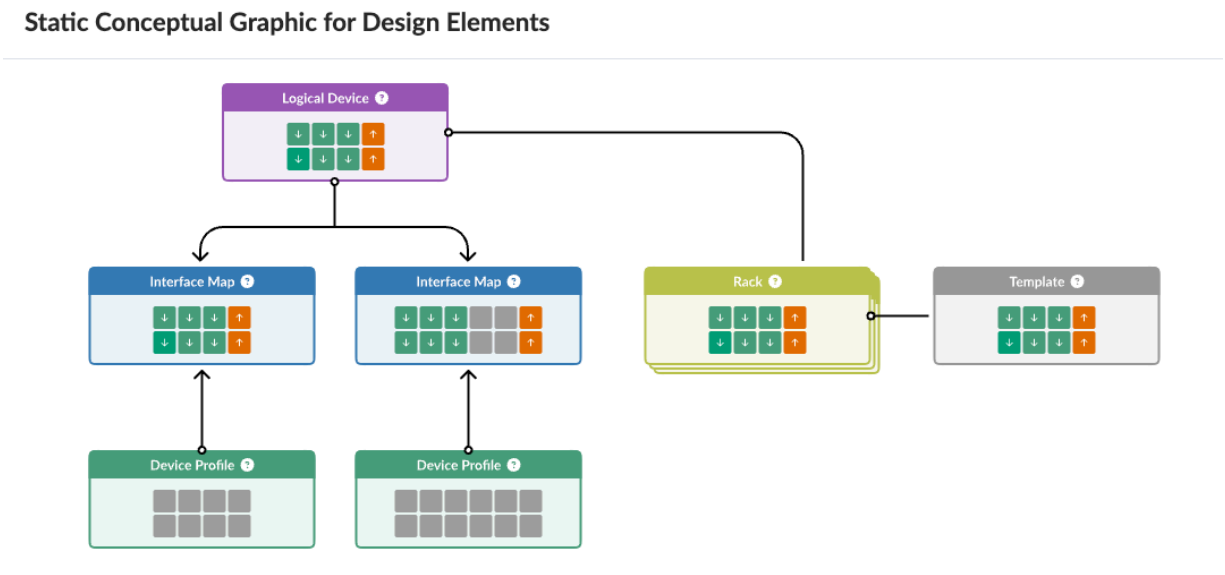
Policy	OPtions
	251 for more information. External systems must be connected to racks (not spine devices).

Templates in the Apstra GUI

From the left navigation menu, navigate to **Design > Templates** to go to the templates table in the design (global) catalog.



To see how design elements and device profiles are related to each other, click **Show relationship** (new in Apstra version 5.0.0). This is helpful if you're new to the Apstra environment.



Many templates are predefined for you. To search for a template by its name, type of template and/or overlay control protocol, click the **Search** button (magnifying glass) and enter your criteria.

Click a template name to got its details.

Template Parameters

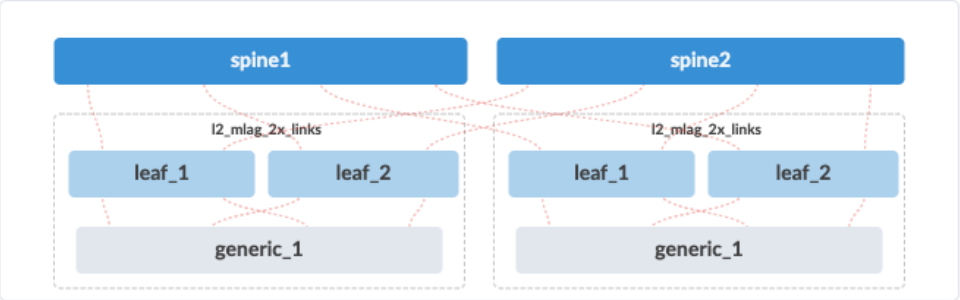
Name	L2 Virtual MLAG 2x Links
Type	RACK BASED

Topology Preview

Selected Rack

All

Expand Nodes? Show Links?

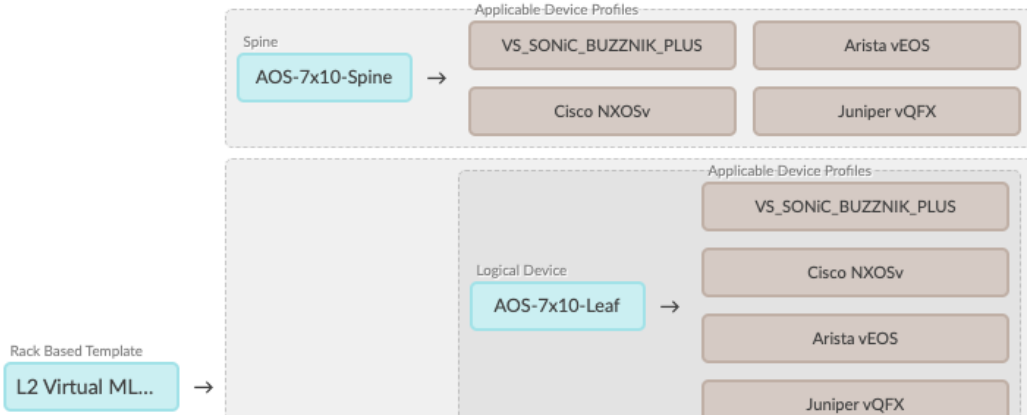


Structure

Spines	2 of AOS-7x10-Spine
Tags	
Rack Types	2 of L2 MLAG 2x Links (1 generic system)

Logical Structure

This shows you all the logical relationships and dependencies chosen so you can validate that the design matches your expectations.



You can create, edit, and delete templates.

RELATED DOCUMENTATION

- What are Rack Types | 862
- What are Tags | 922

Create Rack-based Template

You can build a multi-rack environment by selecting multiple rack types, but you can't mix Layer 2 and Layer 3 racks in the same template.

1. If your design requires rack types and/or logical devices that are not in the global catalog, create them before proceeding.
2. From the left navigation menu, navigate to **Design > Templates** and click **Create Template**.
3. Enter a unique name (64 characters or fewer).
4. Select **RACK BASED**.
5. Select applicable policies.
6. Select a rack type from the drop-down list and select the number of that type to include in the template. Notice that as you enter information, the topology preview on the right changes accordingly.
   - To add another rack, click **Add racks**.
7. Select the **Spine Logical Device** from the drop-down list, then select the number of them to include in the template. Make sure to select one that provides a sufficient number of spine ports for your design. For 5-stage designs, make sure to select a logical device that includes the **Superspine** role.
8. For 5-stage designs, enter the number and connection speed of links for **Superspine Connectivity**.
9. Select tags, as applicable (to specify external routers for example), from the drop-down list or create them on-the-fly.
10. Click **Create** to create the template and return to the table view.

Next Steps: Create a blueprint from the template.

RELATED DOCUMENTATION

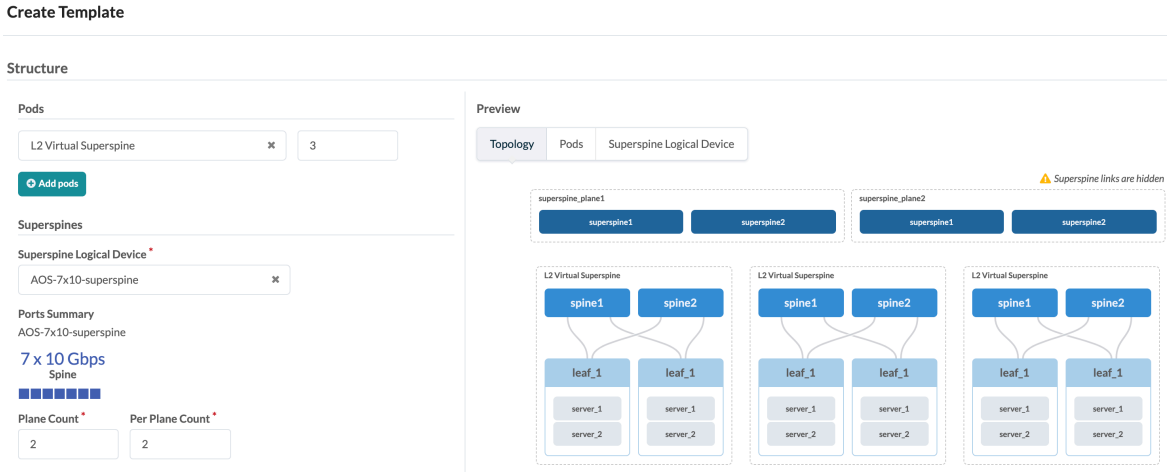
- What are Rack Types | 862
- What are Templates | 889

Create Pod-based Template

A pod-based template consists of multiple rack-based templates; it's essentially a "template of templates" used to build 5-stage Clos networks.

1. If your design requires templates, rack types and/or logical devices that are not in the global catalog, create them before proceeding.
2. From the left navigation menu, navigate to **Design > Templates** and click **Create Template**.
3. Enter a unique name (64 characters or fewer).
4. Select **POD BASED**.
5. Select applicable policies.
6. Select a pod from the drop-down list and select the number of that type of pod. Notice that as you enter information, the topology preview on the right changes accordingly.
   - To add another type of pod, click **Add pods** and select another pod from the drop-down list.
7. Select a **Superspine Logical Device** from the drop-down list.
8. Select the number of planes and the number of superspine devices per plane.
9. Select tags, as applicable (to specify external routers for example), from the drop-down list or create them on-the-fly.
10. Click **Create** to create the template.

    The example below shows a pod-based template with three pods and two planes, each containing two superspine devices:



Next Steps: Create a blueprint from the template.

RELATED DOCUMENTATION

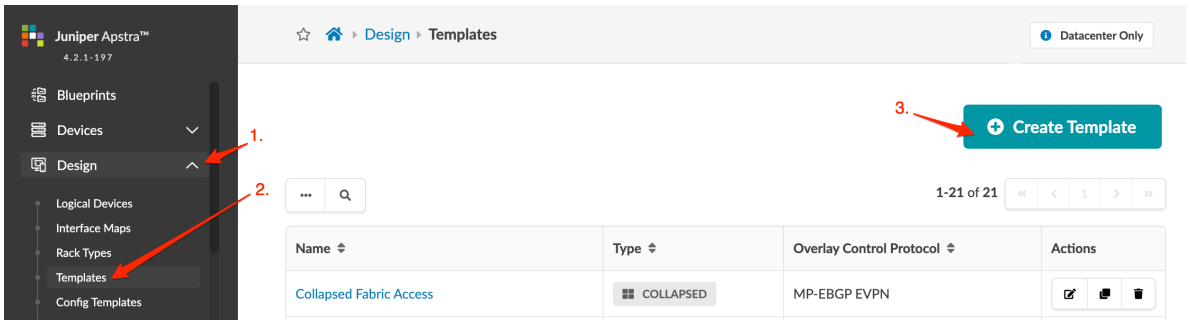
5-Stage Clos Architecture | 1425

What are Rack Types | 862

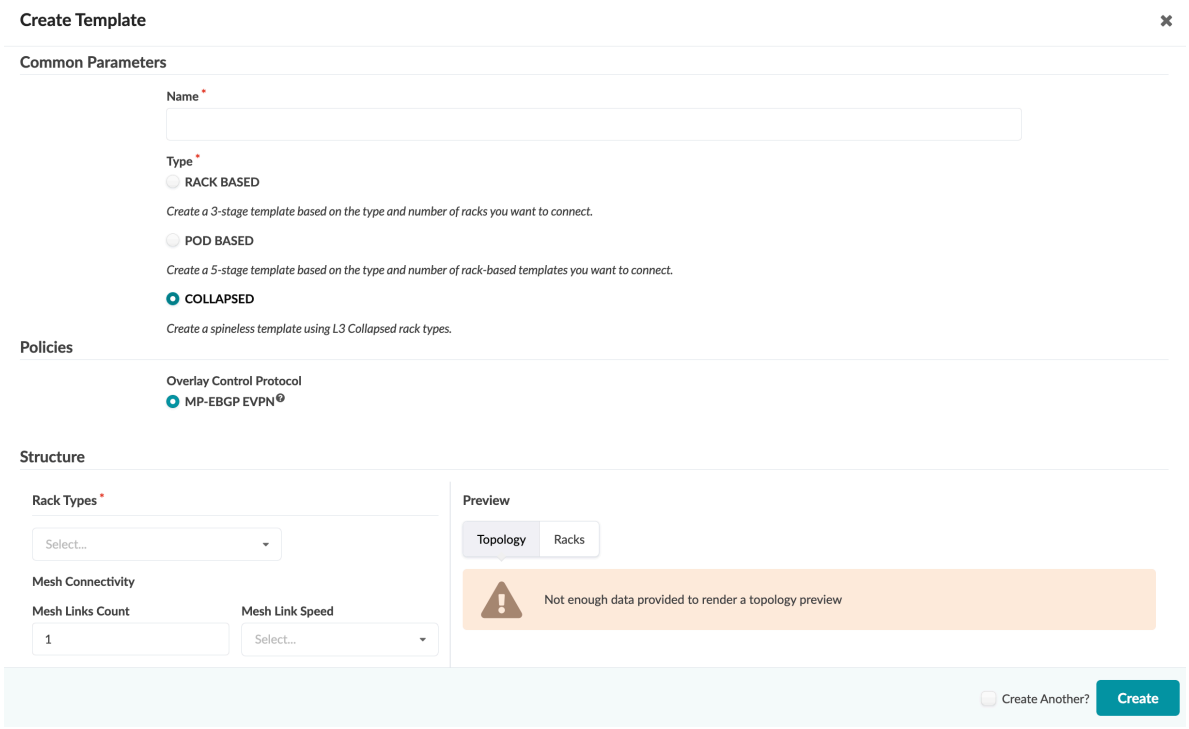
What are Templates | 889

Create Collapsed Template

1. From the left navigation menu, navigate to **Design > Templates** and click **Create Template**.



2. Enter a unique name (64 characters or fewer).



3. Select **COLLAPSED**.
4. Select applicable policies.
5. Select a rack type from the drop-down list (only L3 collapsed rack types are available for selecting), then select the number of mesh links and their speeds. Notice that as you enter information, the topology preview on the right changes accordingly.

6. Click **Create** to create the template and return to the table view.

Next Steps: Create a blueprint from the template.

RELATED DOCUMENTATION

What are Rack Types | 862

What are Templates | 889

Edit Template

1. From the left navigation menu, navigate to **Design > Templates** and click the **Edit** button (top-right) for the template to update.
2. Make your changes.
   - To update a rack type in a rack-based template, first update the rack type in the global catalog, then delete the original rack type from the template (click **X** to the right of the template). *Before* clicking **Update**, select the same (modified) rack type from the drop-down list.
3. Click **Update** (bottom-right) to update the template and return to the table view.

Changes made to a template in the global catalog don't affect blueprints that were previously created with that template, thereby preventing potentially unintended changes to those blueprints.

RELATED DOCUMENTATION

Update Rack Type | 887

What are Templates | 889

Delete Template

Do not delete a template if it's referenced by a blueprint.

1. From the left navigation menu, navigate to **Design > Templates** and click the **Delete** button for the template to delete.
2. Click **Delete** to delete the template and return to the table view.

RELATED DOCUMENTATION

What are Templates | 889

CHAPTER 29

Config Templates (Freeform)

IN THIS CHAPTER

Config Templates (Freeform Design)

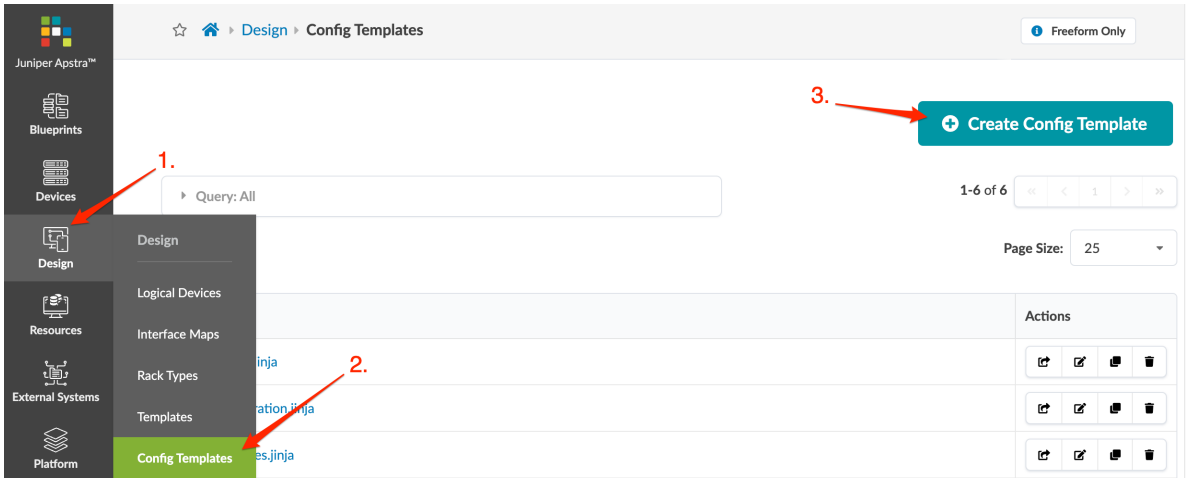
IN THIS SECTION

Config templates are text files used to configure internal systems in Freeform. You'll assign a config template to every internal system. You could paste configuration directly from your devices into a config template to create a static config template, but then you wouldn't be using the potential of config templates. With some Jinja2 knowledge (and maybe some Python), you can parametrize config templates to do powerful things.

For more information about config templates, see "Config Templates (Freeform Blueprint)" on page 578.

Create Config Template

1. From the left navigation menu of the Apstra GUI, navigate to **Design > Config Templates** and click **Create Config Template**.



2. Enter a unique name for the config template including the `.jinja` extension. (The .jinja extension is required even if you're not using Jinja.)
3. Enter or paste your content into the **Template Text** field.
4. Click **Create** to create the config template and return to the config template table view. Your newly created config template is available to be imported into any blueprint catalog.

> **NOTE**: You can also create config templates directly in the blueprint catalog. If you've already created your internal systems in your blueprint, you'll have access to its Device Context all in one place which makes it easier to get device information that you need for config templates.

Edit Config Template

1. From the left navigation menu of the Apstra GUI, navigate to **Design > Config Templates** to go to the table view.
2. Either from the table view or the details view, click the **Edit** button for the config template to edit.
3. Make your changes.
4. Click **Update** to update the config template and return to the table view.

Delete Config Template

1. From the left navigation menu of the Apstra GUI, navigate to **Design > Config Templates** to go to the table view.
2. Either from the table view or the details view, click the **Delete** button for the config template to delete.
3. Click **Delete** to stage the deletion and return to the table view.

CHAPTER 30

Configlets (Datacenter)

IN THIS CHAPTER

Configlets Introduction

IN THIS SECTION

Configlets are configuration templates that augment Apstra's reference design with non-native device configuration. They consist of one or more generators. Each generator specifies a NOS type (config style), when to render the configuration, and CLI commands (and file name as applicable). The section that you select when creating the configlet determines when the configuration is rendered.

When you want to use a configlet, you import it from the global catalog into a blueprint catalog and assign it to one or more roles and/or deployed devices. You can edit the roles and/or devices in a blueprint configlet, but if you want to change the configlet itself, you must export it to the global catalog, modify it, and re-import it into the blueprint.

You can use the same configlets across the entire enterprise, but we recommend creating and applying regionally-specific "property sets" on page 916 instead.

NOTE: Improperly configured configlets may not raise warnings or restrictions. Testing and validating configlets for correctness is the responsibility of the end user. We recommend that you test configlets on a separate dedicated service to ensure that the configlet performs exactly as intended.

Passwords and other secret keys are not encrypted in configlets.

Configlet Applications

Some applications for configlets include the following:

- Syslog
- SNMP access policy
- TACACS / RADIUS
- Management ACLs
- Control plane policing
- NTP
- Username / password

When Not to Use Configlets

CAUTION: Using configlets to add non-native configuration is not always appropriate or possible. Configlets are powerful, but if used improperly they pose risks to deployment stability and reference design feature interactions. Testing and validating configlets for correctness is the responsibility of the end user.

Don't use configlets to replace reference design configuration, such as for routing or connectivity. If you change interface configuration, the Apstra-intended interface configuration could be overwritten. For example, if a configlet creates a network span port, you must apply the configlet to an **Unused** port, or it might inadvertently overwrite one that is already in use.

On Cisco NX-OS and Arista EOS devices, do not use configlets to configure multi-line banners (such as banner motd) because of a problematic extra non-ASCII character that cannot be entered. Instead, configure multi-line banners with Cisco POAP (Power-on Auto Provisioning) or ZTP (Arista Zero Touch

Provisioning) before installing the device agent. The banner configuration becomes part of the device's pristine configuration and persists throughout the Apstra configuration. Another option is to manually configure multi-line banners on the device. This method causes a *configuration deviation* anomaly that you can clear by accepting the new configuration as the golden config. For more information, see "Configuration Deviation" on page 616.

Configlet Parameters

Configlets include the following details. The selected config style (NOS type) and section determine whether template text, negation template text and filename are required:

Table 21: Configlet Parameters

Name	Description
Configlet Name	A unique name to identify the configlet, 64 characters or fewer
Config Style (NOS Type)	Junos, NX-OS, EOS, SONiC
• Section: **System** (NX-OS, EOS, SONiC) • Section: Top-Level: Hierarchical (previously called System) (Junos)	• Runs commands as root user. Improper changes could break the functionality of the reference design and **take down a network**. • When a device is unassigned from a node, the negation template text removes configuration. For example, if the template text is `username example privilege 15 secret 0 MyPassword`, the negation template text might be `no username example` • Can use in conjunction with File configlets to restart processes or perform administrative tasks after File configlets render. • System configlets can nest other configuration. • For NX-OS and EOS, the appropriate configure terminal context is applied. It doesn't need to be part of the configlet.
Section: **Top-Level: Set / Delete** (Junos)	Author configlets using Juniper "Set" style rather than structured JSON

Table 21: Configlet Parameters *(Continued)*

Name	Description
• Section: **Interface** (NX-OS, EOS) • Section: **Interface-Level: Hierarchical** (Junos)	• For physical devices only. • You specify the interface when you "import" on page 443 the configlet into a blueprint (scope).
Section: **Interface-Level: Set** (Junos)	Author configlets using Juniper "Set" command rather than structured JSON. Text is validated to begin with 'set'.
Section: **Interface-Level: Delete** (Junos)	Author configlets using Juniper "Delete" command rather than structured JSON. Text is validated to begin with 'delete'.
Section: **File** (SONiC)	• The entire contents of the file must be present within the configlet because the entire file is overwritten; there is no versioning or storing of the original file contents, soyou can't restore it to its original content. **Improper use can take down a network**. Do not use on config files of critical processes (such as `/etc/frr/frr.conf` or `/etc/network/interfaces/`). • Contents are written, as root user, to the `/etc` directory file (because of Apstra's Docker container host mount). To write to a file outside of `/etc` (`/usr` for example) build the File configlet, then use a System configlet to move the file afterwards.
Section: System Top (NX-OS, EOS)	Ensures that you can overwrite a setting to implement programmed intent. When the reference design is applied, any needed features that were "turned off" in this configlet are reenabled.
Section: FRR (SONiC)	• Configlet configuration is appended to the end of the Apstra-generated `/etc/frr/frr.conf` file and becomes part of FRR intent. Configuration is incrementally included in frr-reload. • **Template text is not validated**. Errors are likely to cause deployment errors, unintended configuration and device impact.
Template Text	CLI commands to add configuration to devices. Issued directly to devices without validation.
Negation Template Text	CLI commands to disable configlet functionality (when a device is unassigned). Issued directly to devices without validation.

Table 21: Configlet Parameters *(Continued)*

Name	Description
Filename	For File configlets

Configuration Rendering Order

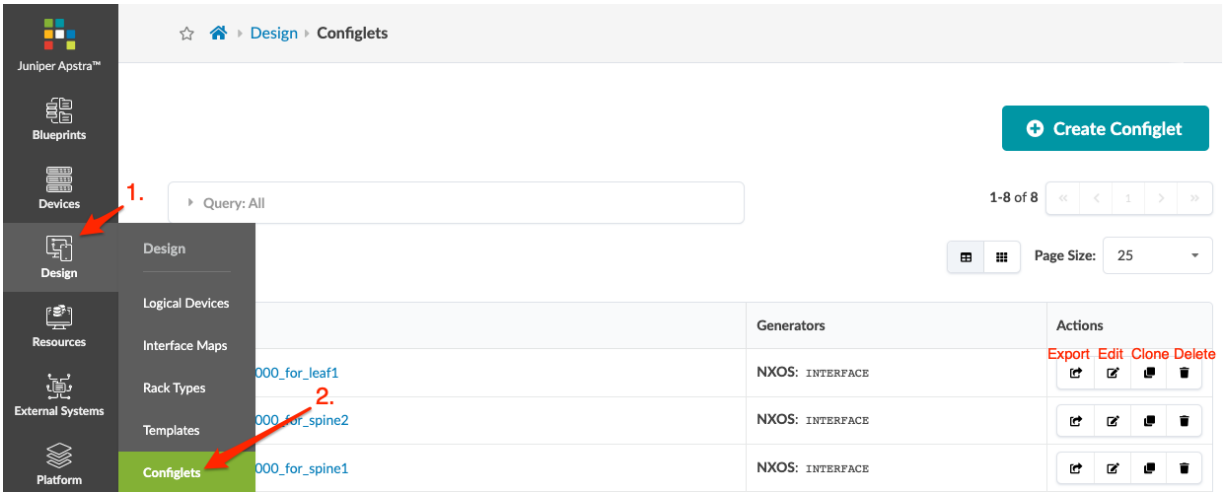
Configuration rendering order is as follows:

1. System Top: negation template text (NX-OS, EOS)
2. System Top: template text (NX-OS, EOS)
3. Apstra reference design
4. Interface: negation template text (Junos, NX-OS, EOS)
5. System: negation template text (Junos, NX-OS, EOS, SONiC)
6. File (SONiC)
7. System: template text (Junos, NX-OS, EOS, SONiC)
8. Interface: template text (Junos, NX-OS, EOS)

To control the order of operations within a section, create configlets with numeric names. For example, `01_syslog` renders before `02_ntp`. Configlets are then ordered based on the condition of the configlet (for example the spine or leaf role), and then by the Node ID of the configlet.

View Configlets (Design)

From the left navigation menu, navigate to **Design > Configlets** to go to configlets in the design (global) catalog. You can create, clone, import, export, edit and delete configlets.



RELATED DOCUMENTATION

What are Property Sets (Datacenter Design) | 916

Create Configlet (Design)

To learn how you can access a dictionary of variables (device model) that you can use when you create configlets, see the "Device Configuration Lifecycle" on page 634.

1. From the left navigation menu, navigate to **Design > Configlets** and click **Create Configlet**.
2. If you've created a JSON payload, click **Import Configlet** and select the file to import it. Otherwise, continue to the next step.
3. Enter a unique configlet name.
4. Select a NOS type (config style).
5. Select the section where you want to render the configlet. Available choices depend on the selected config style. (OSPF for external routers is no longer supported. While OSPF configlets still appear in the Apstra GUI, they should not be used.)
6. In the **Template Text** and **Negation Template Text** fields (as applicable), enter CLI commands. For Interface-Level Set or Delete configlets, do not include `set` or `delete` in the text. See Configlet examples in the Reference section. Avoid using shortened versions of commands. Jinja syntax is highlighted with color coding to improve readability, especially for complex configlets with multiple

property set variables or when Jinja control structures (such as loops and conditionals) are used. Jinja syntax is validated. If Jinja syntax is incorrect, a validation error is raised.

> ⚠ **CAUTION**: Using a raw text editor (OSX TextEdit, Windows Notepad++) is critical. Hidden characters can cause unforeseen issues when the configlet is deployed.

> **NOTE**: Instead of hard-coding data into a configlet, you can refer to a "property set" on page 916 (key-value pairs). For an example, see the "Arista NTP example" on page 1737 in the References section.

7. If **Negation Template Text** is required, enter the CLI commands to remove the configuration.
8. For File configlets, enter the filename in the **Filename** field.
9. To add another generator, click **Add a style** and enter details. (Tip: Configlets can contain syntax for multiple vendors. Create one single-purpose configlet with a generator for each vendor NOS type to include its own syntax.)
10. Click **Create** to add the configlet to the global catalog.

When you're ready to use the configlet in a blueprint, "import" on page 443 it into the blueprint's catalog.

Export Configlet (Design)

Exporting configlets makes it easier for SEs to deliver predefined configlets to customers and makes it easier to copy configlets across Apstra instances.

1. From the table view (Design > Configlets) or the details view, click the **Export configlet** button for the configlet to export. Configlet details are displayed.
2. To copy the contents, click **Copy**, then paste it.
3. To download the JSON file to your local computer, click **Save as File**.
4. When you've copied and/or downloaded the file, click the **X** to close the dialog.

Edit Configlet (Design)

Changing configlets in the design (global) catalog doesn't affect configlets in blueprint catalogs. If your intent is for a blueprint to use a modified configlet, see "Edit Configlet (Blueprint)" on page 446 for the workflow.

1. From the table view (Design > Configlets) or the details view, click the **Edit** button for the configlet to edit.
2. Make your changes (name, config style, section, template text, negation template text, filename, as applicable).
3. Click **Update** (bottom-right) to update the configlet in the global catalog and return to the table view.

Delete Configlet (Design)

Deleting configlets in the design (global) catalog doesn't affect configlets in blueprint catalogs.

1. Either from the table view (Design > Configlets) or the details view, click the **Delete** button for the configlet to delete.
2. Click **Delete** to delete the configlet from the global catalog and return to the table view.

CHAPTER 31

Property Sets (Datacenter)

IN THIS CHAPTER

What are Property Sets (Datacenter Design)

Property sets are data sets that define device properties. They work in conjunction with configlets and Analytics probes. (Config templates in Freeform blueprints also use property sets, but they're not related to property sets in the Design catalog, as discussed here.) Instead of embedding data directly into configlets or probes, you can store variable values in a property set, then associate that property set with the configlet or probe. This gives you flexibility in case you want to change values later. After you create your blueprint, you'll import your configlets and property sets from the Design (global) catalog into the blueprint catalog.

First, you need to write the property set (and the configlet or probe that'll use it). You can write it in JSON or YAML. You can use key-value pairs, lists, dictionaries, and any combination of these data structures by nesting them.

Below is an example of a property set and configlet that uses it to change the SNMP location field based on a provided list of system_name to location mapping.

Property Set

```
{
     "created_at": "2022-08-26T13:20:04.488463+0000",
     "updated_at": "2022-08-28t18:57:41.169692+0000",
     "values_yaml": "PS_SNMP_Locations:\n  leaf1: DC-Room1-Rack32\n  leaf2: DC1-room1-Rack34\n
leaf3: DC1-Room1-Rack33\n spine1: DC1-Room1-Rack30\n  spine2: DC1-Room1-Rack31\n",
     "values": {
```

```
        "PS_SNMP_Locations": {
            "spine1": "DC1-Room1-Rack30",
            "spine2": "DC1-Room1-Rack31",
            "leaf1": "DC1-Room1-Rack32",
            "leaf3": "DC1-Room1-Rack33",
            "leaf2": "DC1-Room1-Rack34"
        }
    },
    "label": "PS_SNMP_Locations",
    "id": "c4006bb8-f8f4-4aa7-82c3-8da5dfc03c43"
}
```

NOTE: You can enter property set details in any order, but when you open a property set after creating it, it will have been automatically sorted alphabetically (per the Python dictionary function). For example, if you create the property set above, it would be sorted as shown below.

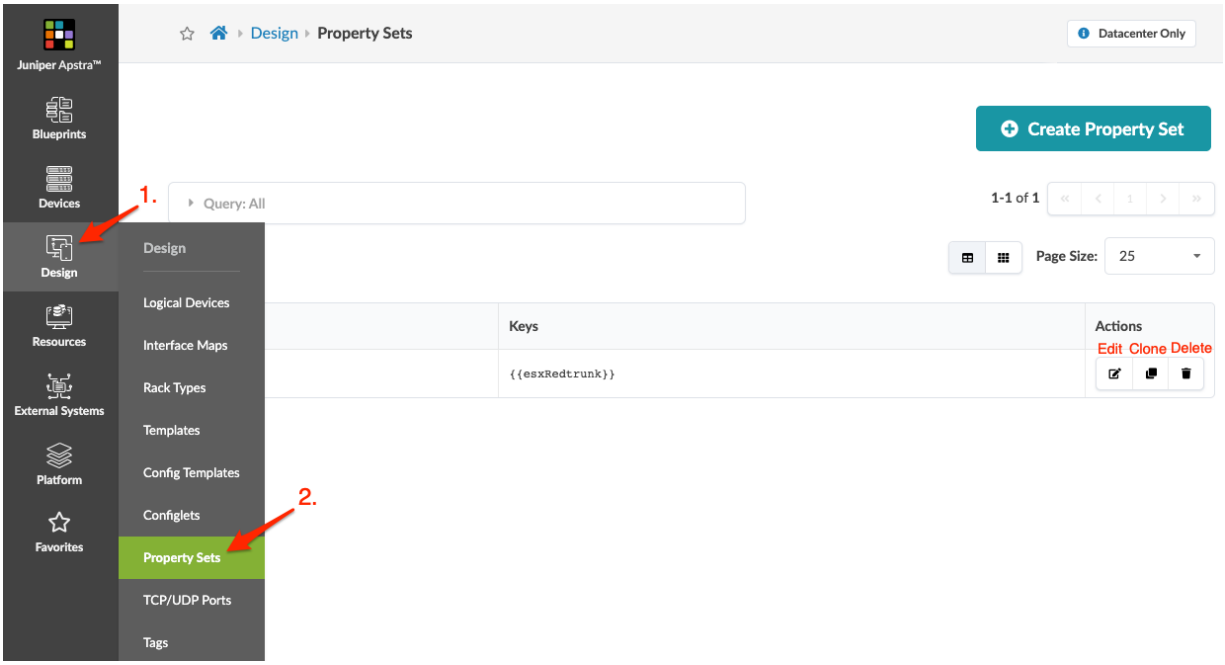
```
{
  "created_at": "2023-08-26T13:20:04.488463+0000",
  "id": "c4006bb8-f8f4-4aa7-82c3-8da5dfc03c43",
  "label": "PS_SNMP_Locations",
  "updated_at": "2023-08-28t18:57:41.169692+0000",
  "values": {
    "PS_SNMP_Locations": {
      "leaf1": "DC1-Room1-Rack32",
      "leaf2": "DC1-Room1-Rack34",
      "leaf3": "DC1-Room1-Rack33",
      "spine1": "DC1-Room1-Rack30",
      "spine2": "DC1-Room1-Rack31"
    }
  },
  "values_yaml": "PS_SNMP_Locations:\n  leaf1: DC-Room1-Rack32\n  leaf2: DC1-room1-Rack34\n
leaf3: DC1-Room1-Rack33\n spine1: DC1-Room1-Rack30\n  spine2: DC1-Room1-Rack31\n"
}
```

Configlet

```
{
  "ref_archs": [
    "two_stage_l3clos"
  ],
```

```
  "generators": [
    {
      "config_style": "junos",
      "section": "system",
      "template_text": "{% if PS_SNMP_Locations[hostname] is defined %}\nsnmp {\n     location
\"{{PS_SNMP_Locations[hostname]}}\";\n}\n{5 endif %}\n",
      "negation_template_text": ::,
      "filename": ""
    }
   ],
   "created_at": "2022-08-26T13:23:57.2720142",
   "id": "b2739659-897d-4fa2-a8e9-2060ae1c045f",
   "last_modified_at": "2022-08-26T13:29:40.1924382",
   "display_name": "SNMP_location"
}
```


From the left navigation menu, navigate to **Design > Property Sets** to go to property sets in the Design catalog. You can create, clone, edit and delete property sets.



Create Property Set (Datacenter Design)

1. From the left navigation menu, navigate to **Design > Property Sets** and click **Create Property Set**.
2. Enter a unique property set name.

3. You can define property sets with YAML or JSON. (With the YAML option, if you were previously using Ansible, you can now import your Ansible variables defined in `host_vars` and `group_vars`.) Enter your content directly via the Editor, or for YAML you can also use the builder.
4. To add another property, click **Add a Property**.
5. Click **Create** to create the property set and return to the table view.

Edit Property Set (Datacenter Design)

To prevent potentially unintended changes to existing blueprints, changes to property sets in the global catalog do not affect property sets in the blueprint catalog. If your intent is for a blueprint to use a modified property set, then you must re-import the revised property set into the blueprint.

1. From the left navigation menu, navigate to **Design > Property Sets** and click the name of the property set to edit.
2. Click the **Edit** button (top-right) and make your changes.
3. Click **Update** (bottom-right) to update the Property Set.

Delete Property Set (Design)

If a property set is assigned to a "configlet" on page 908, it can't be deleted.

1. Either from the table view (Design > Property Sets) or the details view, click the **Delete** button for the property set to delete.
2. Click **Delete** to delete the property set from the global catalog and return to the table view.

CHAPTER 32

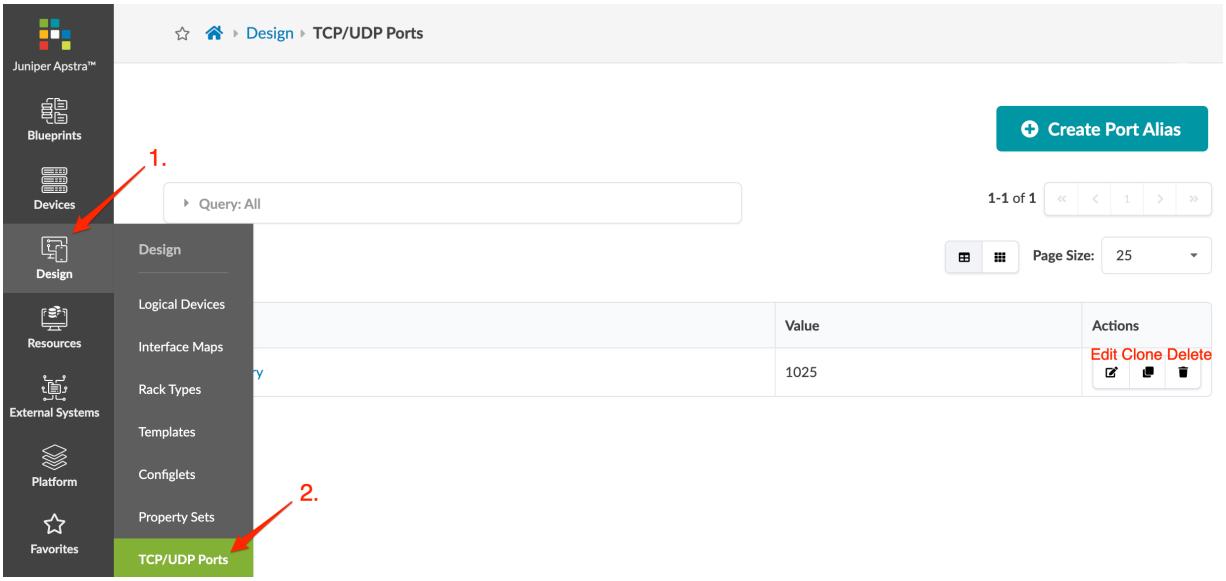
TCP/UDP Ports

IN THIS CHAPTER

TCP/UDP Port Alias Introduction

When you create a security policy and add rules for TCP or UDP protocols, a source port and destination port are specified. You can enter port numbers or you can create aliases ahead of time that can be entered instead of the port numbers. For example, you could create an alias with name *SSH* and a value of *22*.

From the left navigation menu, navigate to **Design > TCP/UDP Ports** to go to TCP/UDP ports. You can create, clone, edit and delete port aliases.



Create TCP/UDP Port Alias

1. From the left navigation menu, navigate to **Design > TCP/UDP Ports** and click **Create Port Alias**.
2. Enter a unique alias name.
3. Enter one or more values.
4. Click **Create** to create the alias and return to the table view. When you add a rule for TCP or UDP protocols to a security policy, the TCP/UDP port alias appears in the drop-down list.

RELATED DOCUMENTATION

TCP/UDP Port Alias Introduction | 920

Edit TCP/UDP Port Alias

1. From the left navigation menu, navigate to **Design > TCP/UDP Ports** and click the **Edit** button for the port alias to edit.
2. Make your changes.
3. Click **Update** to update the TCP/UDP port alias and return to the table view.

RELATED DOCUMENTATION

TCP/UDP Port Alias Introduction | 920

Delete TCP/UDP Port Alias

1. From the left navigation menu, navigate to **Design > TCP/UDP Ports** and click the **Delete** button for the port alias to delete.
2. Click **Delete** to delete the TCP/UDP port alias from the system and return to the table view.

RELATED DOCUMENTATION

TCP/UDP Port Alias Introduction | 920

CHAPTER 33

Tags

IN THIS CHAPTER

What are Tags

Tags add user-defined information to nodes and links. You can add tags to the following elements:

- Leafs, switches, and generic systems in rack types (Design)
- Spines in templates (Design)
- Connectivity Templates (Blueprints)
- Intent-Based Analytics (IBA) Probes (Blueprints)
  - ECMP Imbalance (External Interfaces) probe
  - Total East/West Traffic probe
  - Critical Services: Utilization, Trending, Alerting probe
  - Leafs Hosting Critical Services: Utilization, Trending, Alerting probe

For example, you assign servers and external routers the **generic** port role in logical devices, and then tag them with their specific roles when you design rack types and templates. When you create a blueprint, tags from the relevant design elements are embedded into the tag section of the blueprint catalog.

Changes you may subsequently make to tags in the design elements don't affect the blueprint that had previously used those tags. If you want a blueprint to use revised tags from a design element, you can "import " on page 451them.

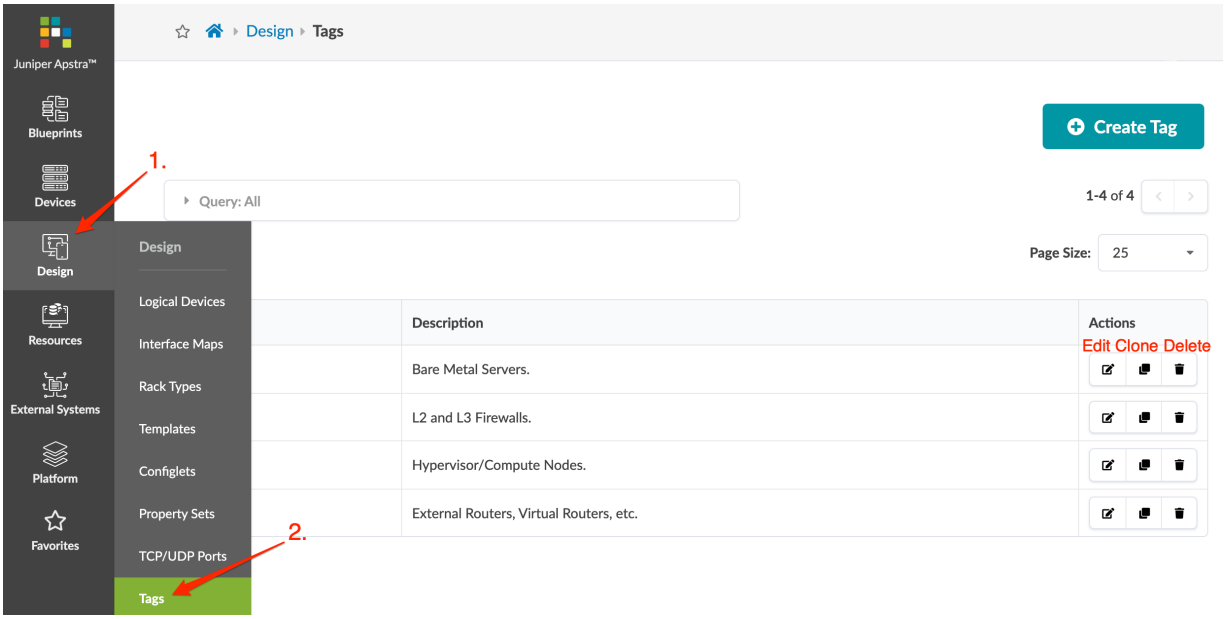
You can "export" on page 451 tags that you created in a blueprint to the global catalog (as long as they have a unique name) where they can be used in subsequent design elements.

Tags are part of the graph. They appear in the device context, so it's easier to find them to use them programmatically.

Tags include the following details:

- **Name** - Case-insensitive. They must be unique across all tags defined in the design.
- **Description** - Optional field to add any details (for example, server roles, external router roles or customer name).

From the left navigation menu, navigate to **Design > Tags** to go to tags in the global catalog. Four tags (Bare Metal, Firewall, Hypervisor, Router) are predefined for you. You can create, clone, edit and delete tags in the global catalog.



Create Tag in Catalog (Design)

1. From the left navigation menu, navigate to **Design > Tags** and click **Create Tag**.
2. Enter a unique tag name.
3. Enter a description (optional).
4. Click **Create** to create the tag and return to the table view.

RELATED DOCUMENTATION

What are Tags | 922

Change Tag Description (Design)

You cannot change tag names directly; You can only change tag descriptions.

NOTE: You can change a tag name indirectly by creating a tag with the preferred name, applying the tag to the rack type or template, then deleting the tag with the original name from the rack type or template (then deleting the original tag).

To change a tag name indirectly:

1. Create a tag with the preferred name.
2. Apply the tag to the rack type or template.
3. Delete the tag with the original name from the rack type or template.
4. Delete the original tag.

1. Either from the table view (Design > Tags) or the details view, click the **Edit** button for the tag to change.
2. Change the description.
3. Click **Update** to update the tag description and return to the table view.

RELATED DOCUMENTATION

What are Tags | 922

Delete Tag from Catalog (Design)

Deleting a tag from the design (global) catalog does not affect rack types and templates that have previously been assigned the tag.

1. Either from the table view (Design > Tags) or the details view, click the **Delete** button for the tag to delete.
2. Click **Delete** to delete the tag and return to the table view.

RELATED DOCUMENTATION

What are Tags | 922

PART 11

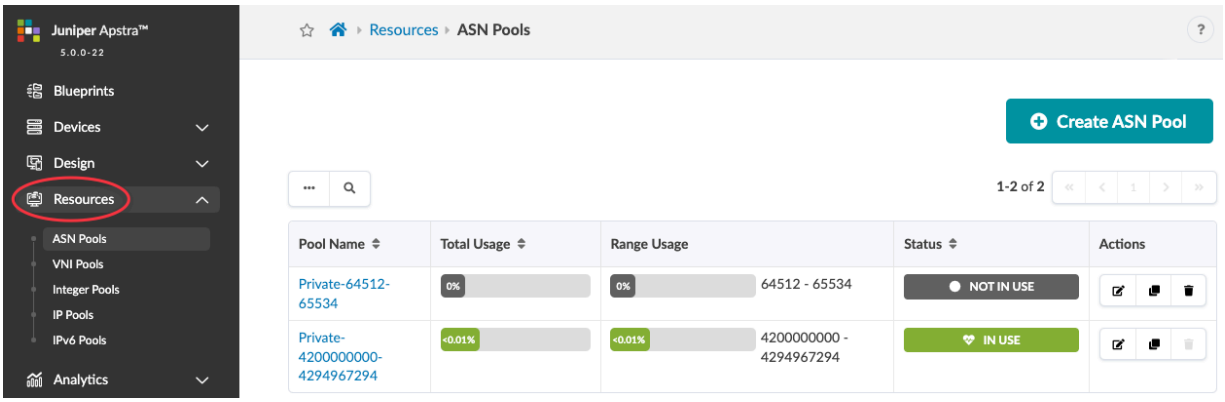
Resources

What are Resources

Resources in the Apstra environment are identifiers of the following types:

- Autonomous system numbers (ASNs)
- Virtual network identifiers (VNIs)
- Integers (optional resource type for Freeform blueprints only)
- IPv4 addresses
- IPv6 addresses

Usually, it doesn't matter what the actual values of these resources are; you just care that some value has been assigned in your blueprint. These values come from resource pools. A resource pool contains one or more ranges of values of a specific resource type. In the Apstra GUI, you'll find resource pools by clicking **Resources** in the left navigation menu, then selecting a resource pool type. For more details about resource pools, see the individual resource pool pages ("ASNs" on page 928, "VNIs" on page 930, "Integers" on page 932, "IPv4 Addresses" on page 934, "IPv6 Addresses" on page 937).



When you're ready to assign resources in your blueprint, just tell Apstra which resource pool(s) to use. The required resources are automatically pulled and assigned. This means you can save time by assigning many resources in one quick transaction. If you find yourself short on resources while you're assigning them, you can create additional resource pools from your blueprint. For details about assigning resource in blueprints, see "Assign ASNs and IP Addresses (Datacenter)" on page 58, "Update Virtual Network Resource Assignments" on page 263, "Assign Routing Zone Resources" on page 290, or "Resource Management (Freeform)" on page 560, as applicable.

For those rare times when you *do* care what the actual value is, Apstra's got you covered. For details, see "Change Assigned ASN (Datacenter)" on page 113, "Change Assigned Loopback IP Address (Datacenter)" on page 115, and "Change Assigned Link IP Addresses (Datacenter)" on page 197, or "Change Routing Zone Details" on page 293.

ASNs

IN THIS SECTION

ASNs in Apstra

IN THIS SECTION

Autonomous system numbers (ASNs) are used to support BGP in the underlay. In Apstra, you create ASN pools in the global **Resources** catalog. When you're building your blueprint you'll "assign those pools" on page 58, then Apstra will automatically pull resources as needed. If you need to assign a specific ASN, you can "assign it manually" on page 113.

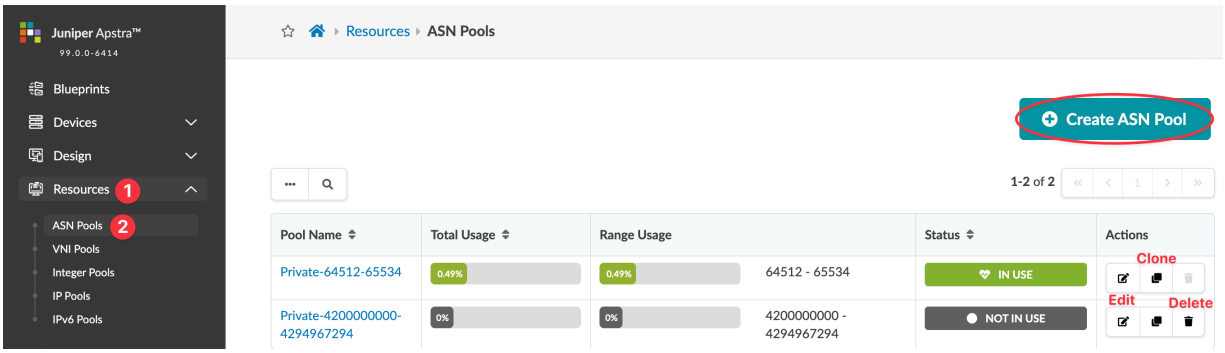
ASN Pool Details

ASN pools include the following details:

Pool Name	A unique name to identify the resource pool
Total Usage	Percentage of ASNs in use for all ranges in the resource pool. (Hover over the status bar to see the number of ASNs in use and the total number of ASNs in the pool.)
Range Usage	The ASNs included in the range and the percentage that are in use. (Hover over the status bar to see the number of ASNs in use and the total number of ASNs in that range.)
Status	Indicates if the pool is in use

ASN Pools in the Apstra GUI

From the left navigation menu in the Apstra GUI, navigate to **Resources > ASN Pools** to go to the global catalog where you can create, edit and delete ASN pools.



Create ASN Pool

1. From the left navigation menu, navigate to **Resources > ASN Pools** and click **Create ASN Pool** (or to copy an existing pool and customize it, click the **Clone** button for the pool to copy).
2. Enter a unique name and range. To add another range, click **Add a range** and enter the range.
3. Click **Create** to create the pool and return to the table view.

Update ASN Pool

You can add, change and delete ranges, but if any ASNs are in use, they can't be removed from a range.

1. From the left navigation menu, navigate to **Resources > ASN Pools** and click the **Edit** button for the pool to update.
2. Add, change and/or delete ASN ranges, as required.
3. Click **Update** to update the pool and return to the table view.

Delete ASN Pool

You can delete ASN pools as long as none of the ASNs within the pool are in use.

1. From the left navigation menu, navigate to **Resources > ASN Pools** and click the **Delete** button for the pool to delete.
2. Click **Delete** to delete the pool and return to the table view.

VNIs

IN THIS SECTION

VNIs in Apstra

IN THIS SECTION

Virtual network identifiers (VNIs) are used in VXLAN encapsulation to provide Layer 2 separation for the overlay traffic in your data center fabric. (For more information about VNI usage, see "Virtual Networks" on page 251.

In Apstra, you create VNI pools in the global **Resources** catalog. When you create "virtual networks" on page 258 and "routing zones" on page 286 you'll assign those pools, then Apstra will automatically pull resources as needed. If you need to assign a specific VNI, you can assign it manually

VNI Pool Details

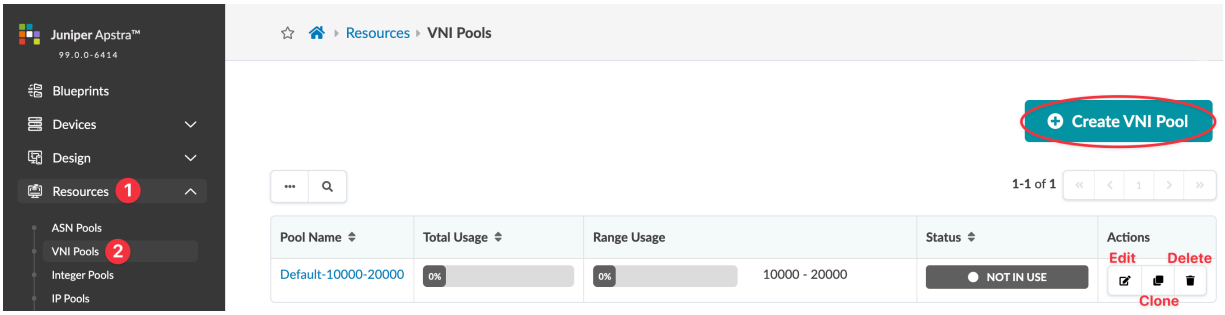
VNI pools include the following details:

Pool Name	A unique name to identify the resource pool
Total Usage	Percentage of VNIs in use for all ranges in the resource pool. (Hover over the status bar to see the number of VNIs in use and the total number of VNIs in the pool.)
Range Usage	The VNIs included in the range and the percentage that are in use. (Hover over the status bar to see the number of VNIs in use and the total number of VNIs in that range.)

Status Indicates if the pool is in use

VNI Pools in the Apstra GUI

From the left navigation menu in the Apstra GUI, navigate to **Resources > VNI Pools** to go to the global catalog where you can create, edit and delete VNI pools.



Create VNI Pool

1. From the left navigation menu, navigate to **Resources > VNI Pools** and click **Create VNI Pool** (or to copy an existing pool and customize it, click the **Clone** button for the pool to copy).
2. Enter a unique name and a valid range (4096 through 16777214). To add another range, click **Add a range** and enter the range.
3. Click **Create** to create the pool and return to the table view.

Update VNI Pool

You can add and change VNI ranges; you can delete a VNI range as long as none of the VNIs within the range are in use.

1. From the left navigation menu, navigate to **Resources > VNI Pools** and click the **Edit** button for the pool to update.
2. Add, change, and/or delete ranges, as required.
3. Click **Update** to update the pool and return to the table view.

Delete VNI Pool

You can delete VNI pools as long as none of the VNIs within the pool are in use.

1. From the left navigation menu, navigate to **Resources > VNI Pools** and click the **Delete** button for the pool to delete.
2. Click **Delete** to delete the pool and return to the table view.

Integers

IN THIS SECTION

Integers in Apstra

IN THIS SECTION

Integers are optional resource types for Freeform blueprints only. In Apstra, you create Integer pools in the global **Resources** catalog. When you're building your blueprint, you can assign an integer pool, then Apstra will automatically pull resources as needed. If you need to assign a specific integer, you can assign it manually.

Integer Pool Details

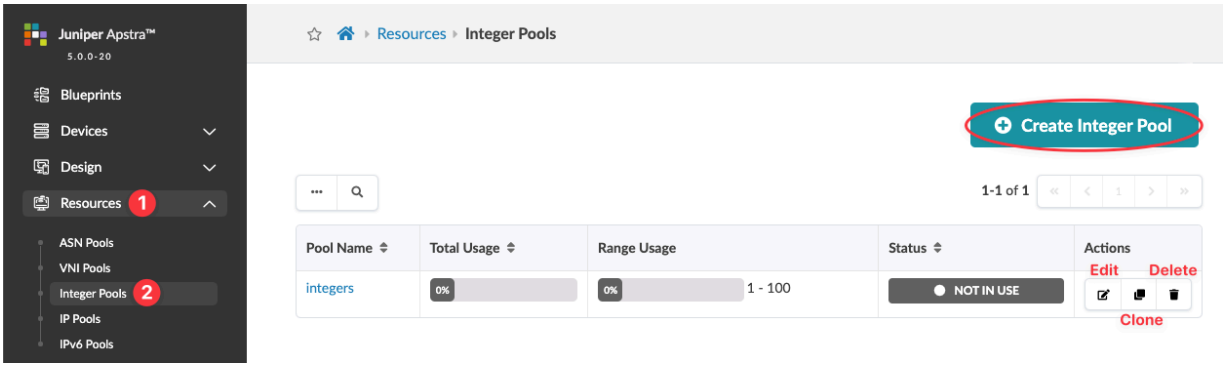
Integer pools include the following details:

Name A unique name to identify the resource pool

Ranges The beginning and ending numbers of the range.

Integer Pools in the Apstra GUI

From the left navigation menu in the Apstra GUI, navigate to **Resources > Integer Pools** to go to the global catalog where you can create, edit and delete integer pools.



Create Integer Pool

1. From the left navigation menu, navigate to **Resources > Integer Pools** and click **Create Integer Pool**.
2. Enter a unique name and range. To add another range, click **Add a range** and enter the range.
3. Click **Create** to create the pool and return to the table view.

Update Integer Pool

1. From the left navigation menu, navigate to **Resources > Integer Pools** and click the **Edit** button for the pool to update.
2. Make your changes. You can add, change and delete ranges, but you cannot remove integers that are in use.
3. Click **Update** to update the pool and return to the table view.

Delete Integer Pool

1. From the left navigation menu, navigate to **Resources > Integer Pools** and click the **Delete** button for the pool to delete.

2. Click **Delete** to delete the pool and return to the table view.

IPv4 Addresses

IN THIS SECTION

IPv4 Addresses in Apstra

IN THIS SECTION

IP addresses are used in the following situations:

Loopback IPs - Spines/Leafs/Generics - the loopback IP is used as the BGP router ID.

SVI Subnets - MLAG Domain - A Switch Virtual Interfaces (SVI) subnet for an MLAG domain is used to allocate an IP address between MLAG leaf switches.

Link IPs - Spines <-> Leafs - Link IPs are used between spine devices and leaf devices to build the L3-Clos fabric. These IPs are necessary for BGP peering between spine devices and leaf devices, and represent the 'fabric' of the network.

Link IPs - Generics - IP addresses facing generic systems are used to statically-route the generic system loopback and route across that link.

In Apstra, you create IP pools in the global **Resources** catalog. When you're building your blueprint you'll "assign" on page 58 those pools, then Apstra will automatically pull resources as needed. If you need to assign a specific "Loopback IP address" on page 115 or "Link IP address" on page 197, you can assign it manually.

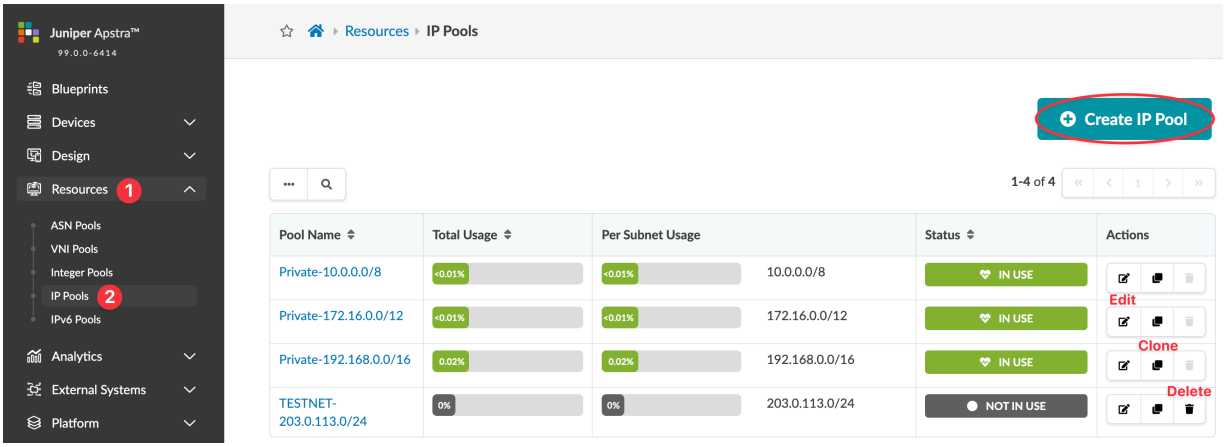
IP Pool Details

IP pools include the following details:

Pool Name	A unique name to identify the resource pool
Total Usage	Percentage of IP addresses in use for all subnets in the resource pool. (Hover over the status bar to see the number of IP addresses in use and the total number of IP addresses in the pool.)
Per Subnet Usage	The IP addresses included in the subnet and the percentage that are in use. (Hover over the status bar to see the number of IP addresses in use and the total number of IP addresses in that subnet.)
Status	Indicates if the pool is in use

IP Pools in the Apstra GUI

From the left navigation menu in the Apstra GUI, navigate to **Resources > IP Pools** to go to the global catalog where you can create, edit and delete IPv4 pools.



Subnet-based access control for Apstra GUI access (whitelisting) is part of platform security enhancements. You can configure "Access Control List (ACL) rules" on page 1284 for IPv4 networks. (IPv6 is not supported on the Apstra web framework.)

Create IPv4 Pool

> ⚠ **CAUTION**: IP address ranges are not validated. It is your responsibility to specify valid IP addresses. If you configure a switch with an invalid IP block you may receive an **error** during the deploy phase. For example, specifying the erroneous multicast subnet 224.0.0.0/4 would be accepted, but it would result in an unsuccessful deployment. If you assign the same range (or overlapping range) of IP addresses to a blueprint, the duplicate assignment is detected and you'll receive a **warning** in the blueprint. You can commit changes to blueprints with warnings without resolving the issues.

1. From the left navigation menu, navigate to **Resources > IP Pools** and click **Create IP Pool** (or to copy an existing pool and customize it, click the **Clone** button for the pool to copy).
2. Enter a unique name and valid subnet. To add another subnet, click **Add a Subnet** and enter a subnet.
3. Click **Create** to create the pool and return to the table view.

Update IPv4 Pool

You can add and change subnets in IP pools; you can delete a subnet as long as none of the IP addresses in the subnet are in use.

1. From the left navigation menu, navigate to **Resources > IP Pools** and click the **Edit** button for the pool to update.
2. Add, change and/or delete subnets, as required.
3. Click **Update** to update the pool and return to the table view.

Delete IPv4 Pool

You can delete IP pools as long as none of the IP addresses within the pool are in use.

1. From the left navigation menu, navigate to **Resources > IP Pools** and click the **Delete** button for the pool to delete.
2. Click **Delete** to delete the pool and return to the table view.

IPv6 Addresses

IN THIS SECTION

IPv6 Addresses in Apstra

IN THIS SECTION

IPv6 is supported on EVPN L2 deployments and L3 deployments. Full feature parity for IPv6 across vendors is not available. Refer to the Apstra Feature Matrix for details.

To use IPv6 addressing, you must "enable IPv6" on page 488 in the blueprint (Staged > Policies > Fabric Addressing Policy).

In Apstra, you create IP pools in the global **Resources** catalog. When you're building your blueprint you'll "assign" on page 58 those pools, then Apstra will automatically pull resources as needed. If you need to assign a specific "Loopback IP address" on page 115 or "Link IP address" on page 197, you can assign it manually.

IPv6 Pool Details

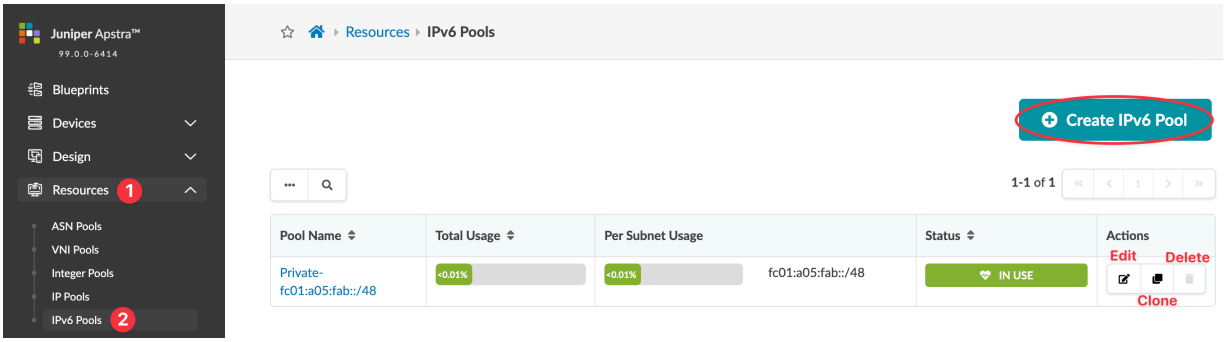
IPv6 pools include the following details:

Pool Name A unique name to identify the resource pool

Total Usage	Percentage of IPv6 addresses in use for all subnets in the resource pool. (Hover over the status bar to see the number of IPv6 addresses in use and the total number of IPv6 addresses in the pool.)
Per Subnet Usage	The IPv6 addresses included in the subnet and the percentage that are in use. (Hover over the status bar to see the number of IPv6 addresses in use and the total number of IPv6 addresses in that subnet.)
Status	Indicates if the pool is in use

IPv6 Pools in the Apstra GUI

From the left navigation menu in the Apstra GUI, navigate to **Resources > IPv6 Pools** to go to the global catalog where you can create, edit and delete IPv6 pools. The pool fc01:a05:fab::/48 is predefined.



Create IPv6 Pool

1. From the left navigation menu, navigate to **Resources > IPv6 Pools** and click **Create IPv6 Pool** (or to copy an existing pool and customize it, click the **Clone** button for the pool to copy).
2. Enter a unique name and valid subnet. To add another subnet, click **Add a Subnet** and enter the subnet.
3. Click **Create** to create the pool and return to the table view.

Update IPv6 Pool

You can add and change subnets in IPv6 pools; you can delete a subnet as long as none of the IP addresses in the subnet are in use.

1. From the left navigation menu, navigate to **Resources > IPv6 Pools** and click the **Edit** button for the pool to update.
2. Add, change and/or delete subnets, as required.

3. Click **Update** to update the pool and return to the table view.

Delete IPv6 Pool

You can delete IP pools as long as none of the IP addresses within the pool are in use.

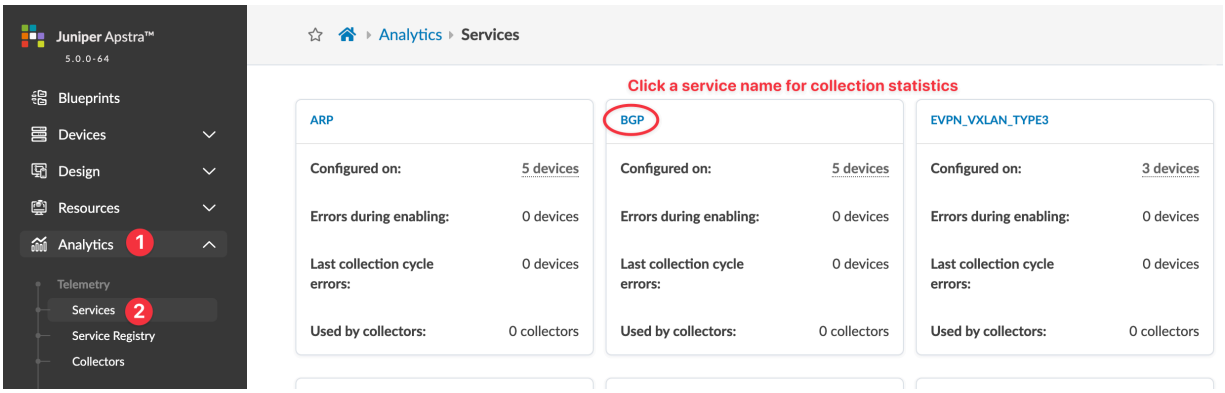
1. From the left navigation menu, navigate to **Resources > IPv6 Pools** and click the **Delete** button for the pool to delete.
2. Click **Delete** to delete the pool and return to the table view.

PART 12

Analytics - Telemetry

Telemetry Services

From the left navigation menu, navigate to **Devices > Telemetry > Services** to go to a summary of telemetry services.



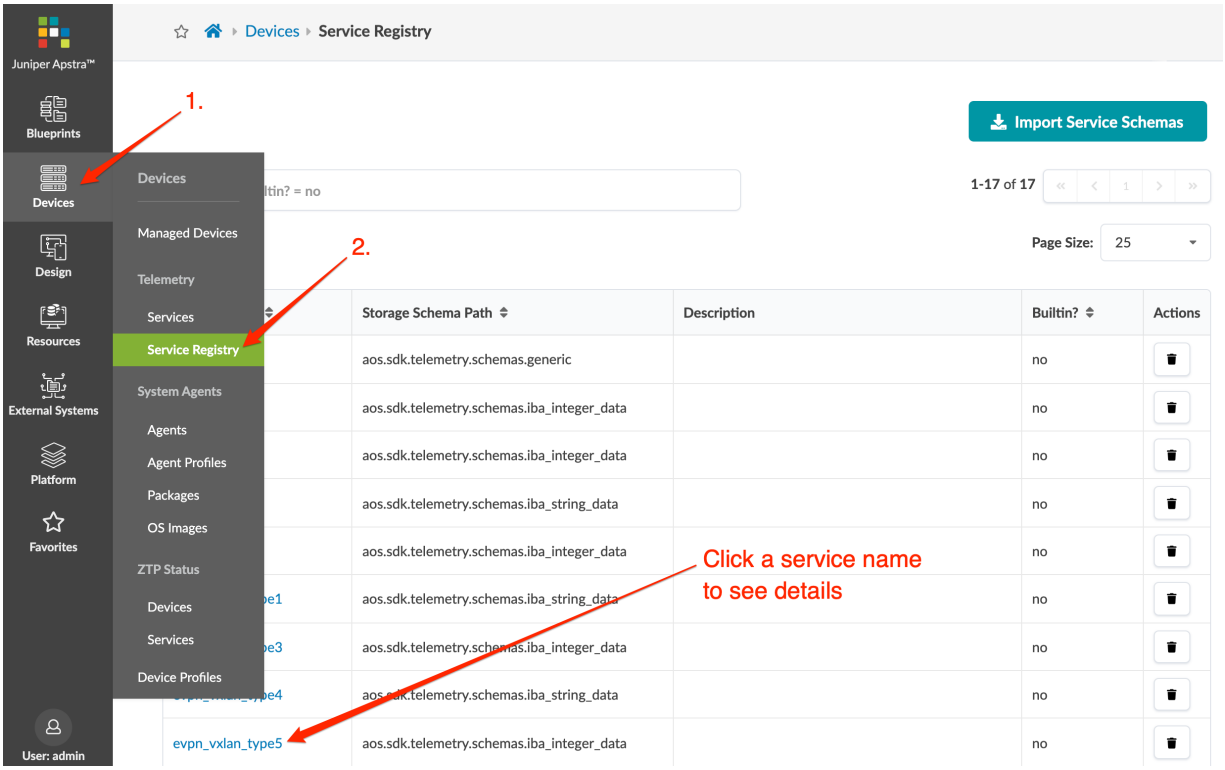
For service details, see "Device Telemetry Services" on page 941.

Service Registry

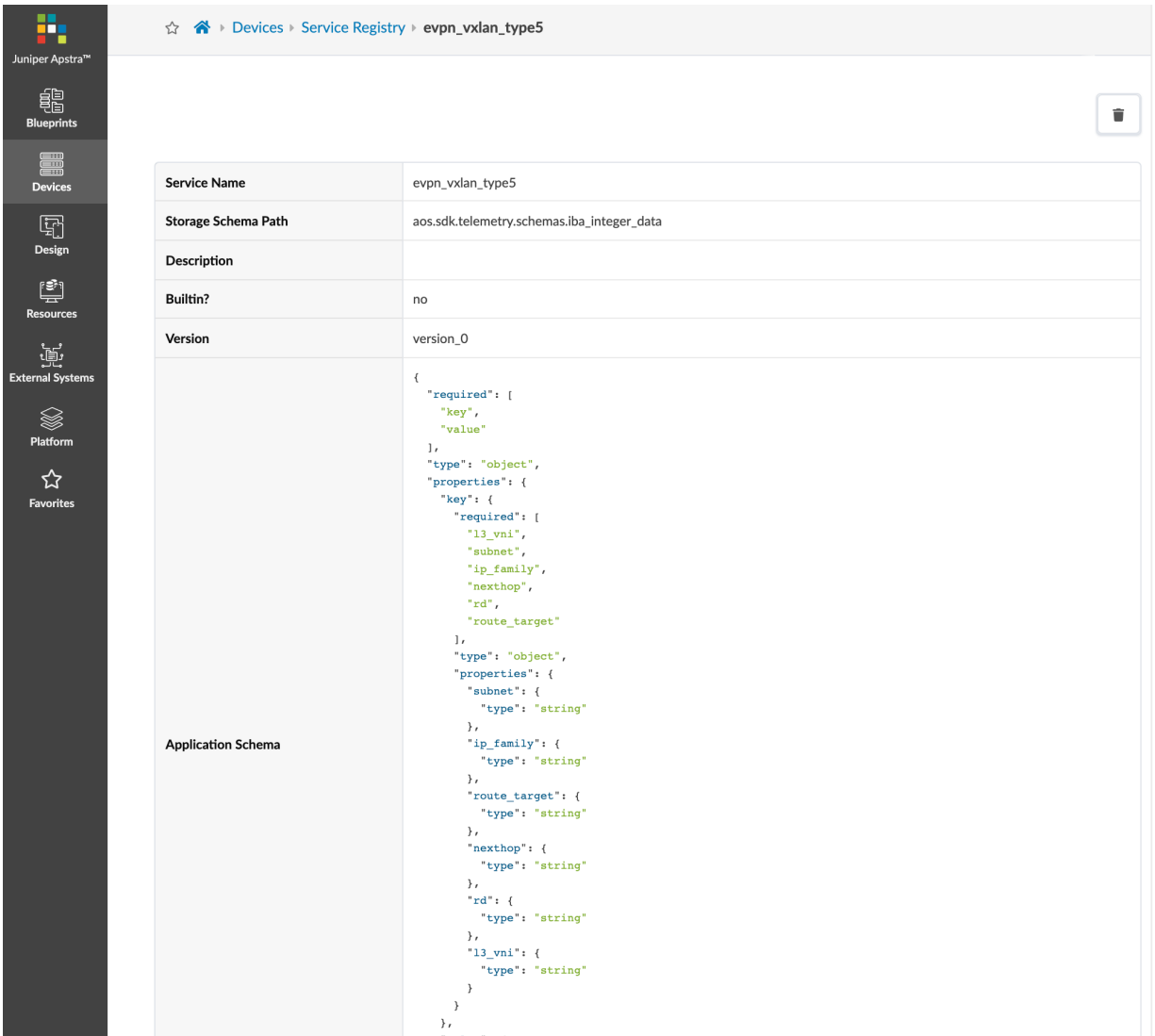
IN THIS SECTION

Service Registry Overview

From the left navigation menu, navigate to **Devices > Service Registry** to go to the service registry. You can view, import and delete telemetry service schemas via the GUI.



To see service registry details, click a service name.



Import Service Schemas

1. From the left navigation menu, navigate to **Devices > Service Registry** and click **Import Service Schemas**.
2. Either click **Choose File** and navigate to the file on your computer, or drag and drop the file from your computer into the dialog window and click **Import**.

Delete Service Registry

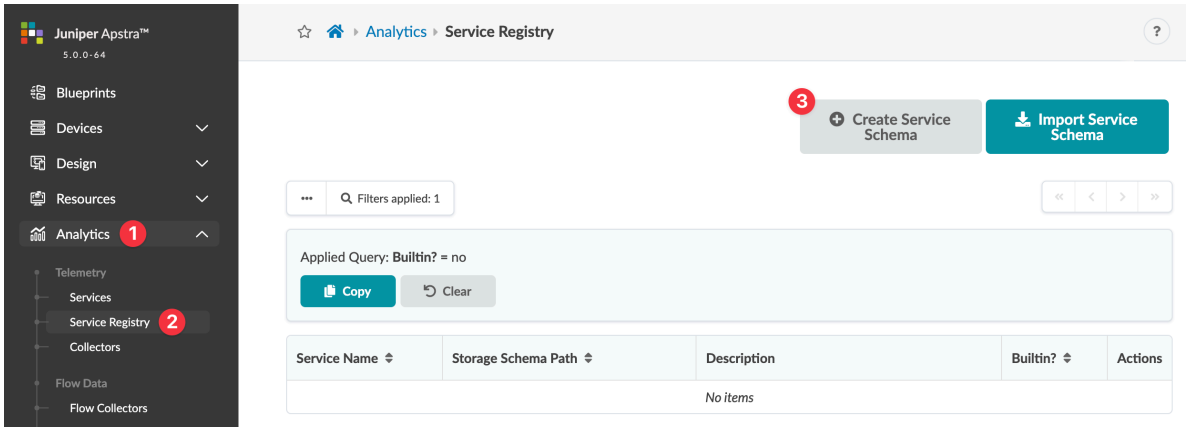
1. Either from the table view (Devices > Service Registry) or the details view, click the **Delete** button for the service to delete.
2. Click **Delete Service Schema** to remove the schema from the system and return to the service registry screen.

Create Telemetry Service Schema

SUMMARY

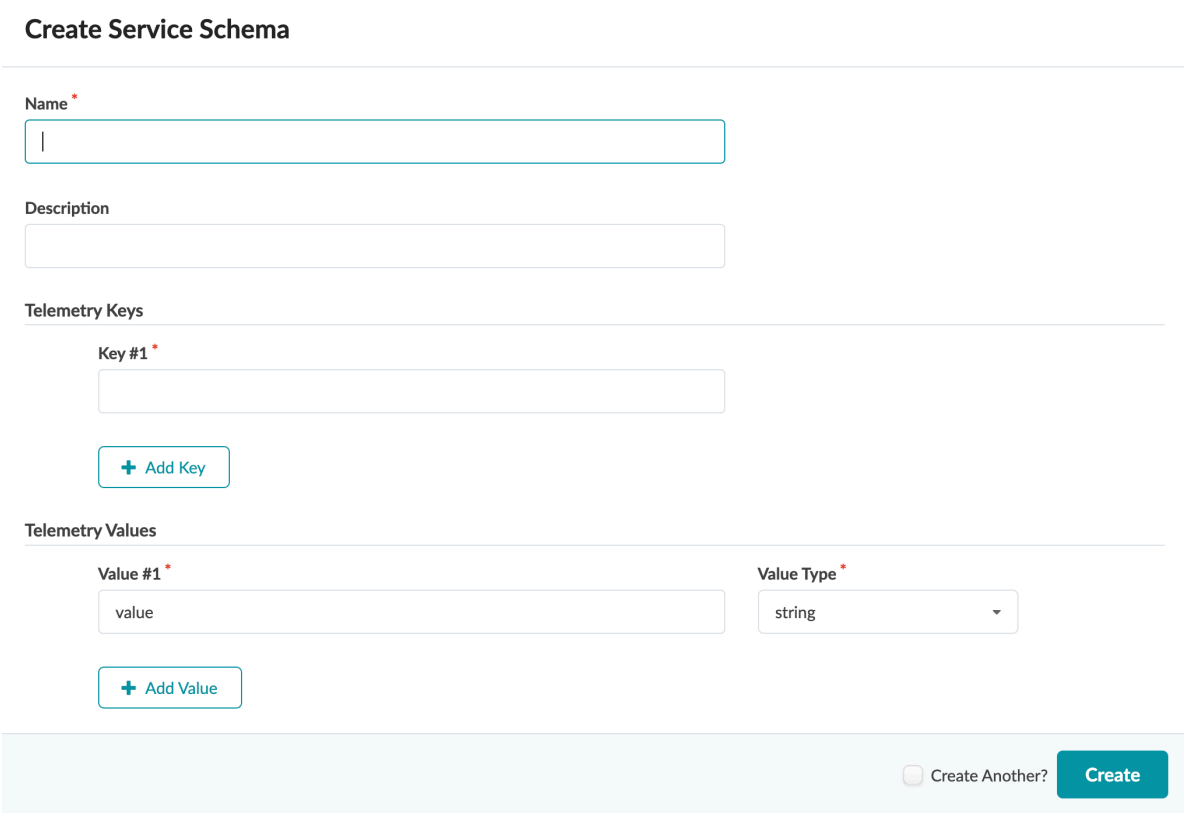
Create a custom telemetry service schema, which is used when creating custom telemetry collectors.

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



The **Create Service Schema** dialog opens.

2. Enter a schema name, (optional) description, and one or more telemetry keys and values (of type string or integer).



3. Click **Create** to create the schema and return to the **Service Registry** page.

When you create a telemetry collector, you'll select a custom telemetry service schema.

Telemetry Collection Statistics

To go to collection statistics for devices using a specific service, click a service name. Telemetry collection statistics include the following details:

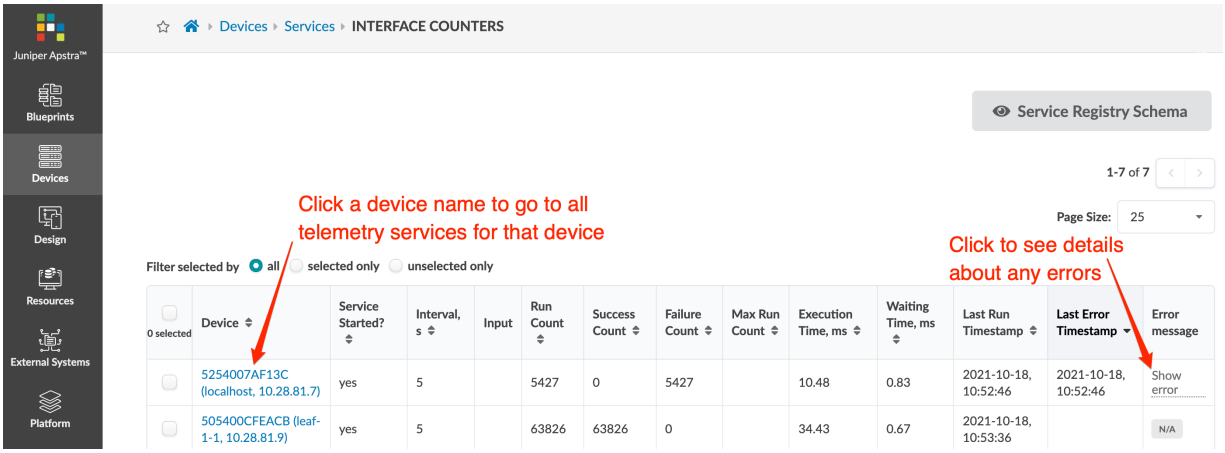
Collection Statistics	Description
Device	The device key
Service Started?	Has the service started?

(Continued)

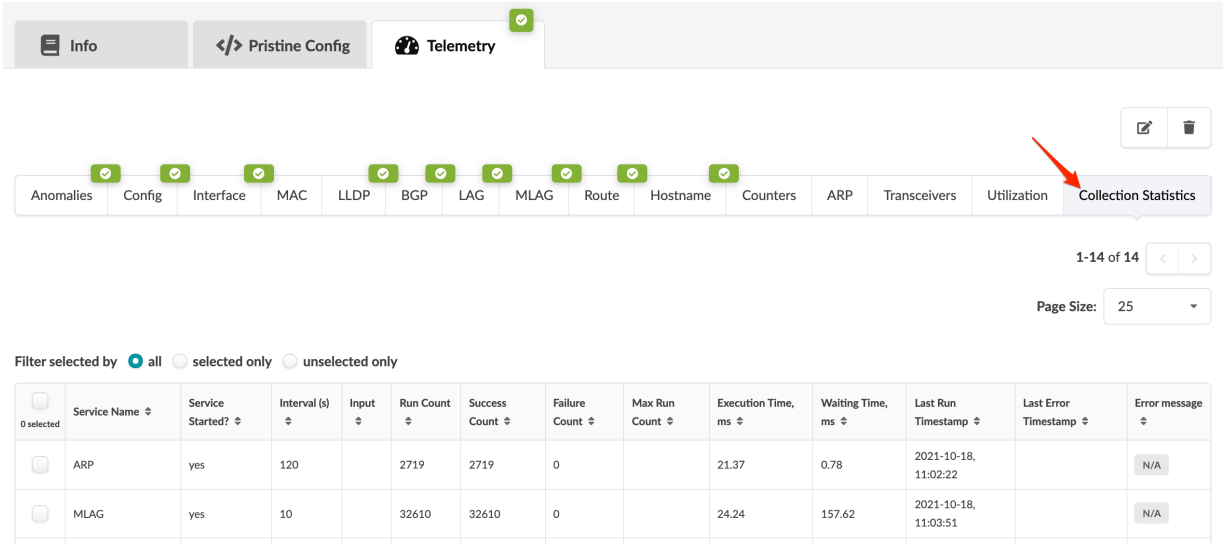
Collection Statistics	Description
Interval	How frequently the service is configured to run on the device (in seconds)
Input	The input that is provided to the service for its processing
Run Count	The number of times the collector is scheduled to run
Success Count	The number of times the collector successfully executed
Failure Count	The number of times the collector failed execution
Max Run Count	User-specified maximum number of times for the collector to run
Execution Time	The time it took for collection during the last iteration (in milliseconds)
Waiting Time	A device runs multiple collectors. If some collectors monopolize CPU, other collector executions are deferred. Waiting time is the amount of time that the collector was deferred (in milliseconds).
Last Run Timestamp	Timestamp at which the collector was scheduled to run
Last Error Timestamp	Timestamp at which the collector last reported an error
Error message	Error message from the last collector iteration.

From the collection statistics screen, you can see if there are any service errors that were generated during the telemetry collection process (in the **Error message** column). Click the **Show error** link to see its details.

From this screen you can also go to to all telemetry services for a specific device by clicking the device name.



To go to collection statistics for all services on a specific device, click **Collection Statistics**.



Telemetry Streaming

The Apstra server transmits the following content to user-defined end-hosts for further processing of data and for use within your own internal systems:

Data Type	Description
Counter Data	Performance Monitoring (PM) data is time-series numerical values such as interface counters, CPU memory utilization, and CPU usage. This information is typically stored and graphed for visual analysis. Typical tools used for this purpose include Graphite and Cacti.
Event Data	Event data is a collection of status information that you may need to refer back for troubleshooting your network. syslog is the best reference for example event. You need a general amount of event history so that you can perform troubleshooting activities over a period of time. While this is an undefined amount of time, you generally want as much time as possible, because you don't get to troubleshoot a problem the instant that it occurs.
Alert Data	Alert data is a collection of information that requires your attention to resolve an issue. In the best cases, alerts tell you what is wrong relative to the network service, and provide the necessary data to allow you to identify root-cause and resolve the issue as fast as possible.

Data streams are implemented with Google Protocol Buffers (GPB). GPBs define and implement the format of data streams. GPBs allow software developers to use a language-agnostic definition of events and data types.

GPB offers support for C++, Python, Go, and possibly more languages in the future. Example Python code named "AOSOM Streaming" on page 1466 is available for GPBs . The AOSOM Streaming demo software is open source and you can download it from github: https://github.com/Apstra/aosom-streaming.

Developers have various language options : C++, Python, Go. This means it integrates nicely with our C++ infrastructure. And then Infrastructure Engineers can use Python or Go for the client.

Route Anomalies for a Host - Example

HTTP GET https://aos-server/api/blueprints/\{blueprint_id}/anomalies (output has been truncated to only show example of one missing route. Actual GET response will return entire routing table)


```
{
  "items": [
    {
      "actual": {
        "value": "missing"
      },
      "anomaly_type": "route",
      "expected": {
        "value": "up"
      },
      "id": "547bcbc9-963f-4477-904b-712482aa6428",
      "identity": {
        "anomaly_type": "route",
        "destination_ip": "0.0.0.0/0",
        "system_id": "000C29202526"
      },
      "last_modified_at": "2017-06-09T17:28:13.773324Z",
      "role": "unknown",
      "severity": "critical"
    },
    {
      "actual": {
        "value": "partial"
      },
      "anomaly_type": "route",
      "expected": {
        "value": "up"
      },
      "id": "92a6804a-42ff-4cbd-a52b-5c6acadc1d23",
      "identity": {
        "anomaly_type": "route",
        "destination_ip": "0.0.0.0/0",
        "system_id": "000C29EA59A7"
      },
      "last_modified_at": "2017-06-09T17:28:44.787604Z",
```

```
      "role": "unknown",
      "severity": "critical"
    },
    {
      "actual": {
        "value": "partial"
      },
      "anomaly_type": "route",
      "expected": {
        "value": "up"
      },
      "id": "25886eb7-e629-4f56-9479-686fe1e53c64",
      "identity": {
        "anomaly_type": "route",
        "destination_ip": "0.0.0.0/0",
        "system_id": "000C29E808A1"
      },
      "last_modified_at": "2017-06-09T17:28:13.773423Z",
      "role": "unknown",
      "severity": "critical"
    },
    {
      "actual": {
        "value": "partial"
      },
      "anomaly_type": "route",
      "expected": {
        "value": "up"
      },
      "id": "2b7a77ac-fd12-41fe-acfc-a53678b177ed",
      "identity": {
        "anomaly_type": "route",
        "destination_ip": "0.0.0.0/0",
        "system_id": "000C2982786A"
      },
      "last_modified_at": "2017-06-09T17:28:13.773389Z",
      "role": "unknown",
      "severity": "critical"
    },
    {
      "actual": {
        "value": "partial"
      },
```

```
      "anomaly_type": "route",
      "expected": {
        "value": "up"
      },
      "id": "50a1e0d6-e483-4bc4-bed8-cbc5666569f8",
      "identity": {
        "anomaly_type": "route",
        "destination_ip": "0.0.0.0/0",
        "system_id": "000C2998C7E7"
      },
      "last_modified_at": "2017-06-09T17:28:13.773453Z",
      "role": "unknown",
      "severity": "critical"
    },
    {
      "actual": {
        "value": "down"
      },
      "anomaly_type": "bgp",
      "expected": {
        "value": "up"
      },
      "id": "ab9f4273-e86f-456c-8cc7-7115f3aafa45",
      "identity": {
        "anomaly_type": "bgp",
        "destination_asn": "1",
        "destination_ip": "10.1.1.1",
        "source_asn": "65417",
        "source_ip": "10.0.0.5",
        "system_id": "000C29202526"
      },
      "last_modified_at": "2017-06-09T17:28:13.727949Z",
      "role": "to_external_router",
      "severity": "critical"
    }
  ],
  "count": 6
}
```

Juniper Telemetry Commands

This section assists network administrators in understanding why telemetry alarms exist, and how they are generated. This is a partial list of interface commands.

Apstra uses CLI to retrieve telemetry from Junos OS and Junos OS Evolved devices.

Table 22: Juniper Telemetry Commands

Service	Command
Interface Counters	`show interfaces extensive`
Interface Error Counters	`show interfaces extensive`
Interface Status	`show interfaces terse`
LLDP neighbors	`show lldp neighbors`
BGP sessions	`show bgp neighbor`
Hostname	`show system information`
ARP	`show arp no-resolve` Provides the ARP information. This is combined with `show configuration routing-instances` which provides the VRF membership for interfaces.
MAC Table	Apstra has two collectors for retrieving MAC telemetry: `show ethernet-switching table extensive` is used with CLIs gRPC collectors use Xpaths. `/network-instances/network-instance/mac-table/entries/entry`
Routing Table	`show route table inet`

Table 22: Juniper Telemetry Commands *(Continued)*

Service	Command
Port Channel	`show lacp interfaces`

Cisco Telemetry Commands

This section assists network administrators in understanding why telemetry alarms exist, and how they are generated. This is a partial list of interface commands.

Cisco telemetry is derived from the NX-API with 'show' commands and embedded event manager applets that provide context data to the device agent while it is running. Most commands are run as their CLI version wrapped into JSON output.

Table 23: Cisco Telemetry Commands

Service	Command
Interface counters	`show interface counters | json`
Interface error counters	`show interface counters errors | json`
Interface status	`show interface status | json`
LLDP neighbors	`show lldp neighbors detail | json`
BGP Sessions	`show bgp session | json`
Hostname	`show hostname | json` and `show hosts | json`
ARP	`show ip arp vrf default | json`
MAC Table	`show mac address-table | json`
Routing table	`show ip route | json`

Table 23: Cisco Telemetry Commands *(Continued)*

Service	Command
Port-channel	`show port-channel summary | json`
MLAG	`show vpc | json`

Arista Telemetry Commands

This section assists network administrators in understanding why telemetry alarms exist, and how they are generated. This is not an exhaustive list of interface commands.

Arista EOS uses a few techniques from the EOS SDK API to directly subscribe to event notifications from the switch, for example 'interface down' or 'new route' notifications. When using an event-based notification, you do not have to continually render 'show' commands every few seconds. The EOS SDK gives you the information immediately as soon as the switch has the status.

CAUTION: Event-based subscription requires the EOSProxySDK agent.

When the Arista API does not provide information (LLDP statistics), Apstra runs CLI commands at a regular interval to derive telemetry expectations.

Table 24: Arista Telemetry Commands

Service	Command
Interface counters	`show interface counters`
Interface error counters	`show interfaces counters errors`
Interface status	`show interfaces status`
LLDP neighbors	`show lldp neighbors detail`
BGP Sessions	`show ip bgp summary`

Table 24: Arista Telemetry Commands *(Continued)*

Service	Command
Hostname	`show hostname`
ARP	ARP collection is done using an event-monitor for performance. `show event-monitor arp` and `show ip arp`
MAC Table	MAC address collection is done using an event-monitor for performance. `show event-monitor mac` and `show mac address-table`
Routing table	`show ip route`
Port-channel	`show port-channel summary`
MLAG	`show mlag` and `show mlag interfaces`

Linux Server Telemetry Command

Linux Servers use simple CLI commands and standard Linux sockets for most telemetry collection.

Table 25: Linux Server Telemetry Commands

Service	Command
Interface counters	`ethtool -m`
Interface error counters	`ethtool -m`
Interface status	Interface status is collected using the netlink api (`AF_INET`)
LLDP neighbors	`lldpctl -f xml`

Table 25: Linux Server Telemetry Commands *(Continued)*

Service	Command
BGP Sessions	`vtysh -c 'show ip bgp summary json'`
Hostname	`hostname`
ARP	`ip -4 neigh`
MAC Table	`brctl showmacs`
Routing table	`show ip route` and the `AF_INET` linux socket
Port-channel	`netshow bondmems --json`
MLAG	`clagctl -j`

Debugging Telemetry

Enable trace options to debug telemetry output. On the Device Agent, in `/etc/aos.conf` (usually), set these options and restart the agent.

```
[DeviceTelemetryAgent]
log_config = aos.infra.core.entity_util:DEBUG,aos.device.DeviceTelemetryAgent:DEBUG
trace_config = MountFacility/0-8,DHT,AgentHeartbeat,TelemetryProxy
```

Log files containing trace information for telemetry agents will then be viewable in `/var/log/aos/DeviceTelemetryAgent.<pid>.<timestamp>.log`. These log files are verbose, but they may point to various rendering and parsing issues in the environment. When you finish troubleshooting, be sure to disable logging.

PART 13

Analytics - Flow

CHAPTER 34

Apstra Flow Overview

IN THIS CHAPTER

Apstra Flow Introduction

Juniper Apstra Flow is a robust solution for collecting and analyzing data center network flow traffic. This feature streamlines the gathering of network traffic flows and telemetry by offering a seamless integration with your organization-specific information. Apstra Flow delivers unmatched visibility and insights into your network traffic.

The Apstra Flow user-friendly dashboards, with enriched data and analytics, gives you a holistic understanding of the network for various critical use cases, including:

- **Performance and availability**

  Provides granular information about network traffic flows, congestion, high latency, and packet loss.

- **Capacity planning and cost control**

  Enables you to implement strategies to optimize the flow of network traffic, ensuring the most efficient use of available resources.

- **Security**

  Improves security by detecting and responding to threats more effectively while maintaining compliance with regulatory requirements.

With Apstra Flow, you gain a powerful toolset to optimize and fortify your network infrastructure.

NOTE: Apstra Flow is a feature in the Apstra Premium tier licensing plan. This feature is available only if you are an Apstra premium customer. For Apstra licensing information, see the Juniper Licensing User Guide.

System Requirements

IN THIS SECTION

Network Connectivity

IN THIS SECTION

Depending on the configured options, Apstra Flow requires various TCP and UDP ports to receive flow records, retrieve data for enrichment, and store data in your chosen data platform. To allow communication on TCP and UDP ports, you must configure any host or network firewalls to allow traffic to pass through as described in the following sections:

Listening for Apstra Flow

You can configure the Apstra Flow collector to listen for incoming flow record packets on one or more UDP ports. Table 1 on page 959 shows the collector's UDP default ports.

Table 26: Apstra Flow Collector Default Ports

Protocol	Port	Direction	Description
UDP	9995	in	Apstra Flow default port
UDP	2055	in	Netflow standard port
UDP	4739	in	IPFIX standard port
UDP	6343	in	sFlow standard port

While a variety of ports can be used to listen for flow record packets, the specific ports which must allowed are those for which the collector is configured using `EF_FLOW_SERVER_UDP_PORT`.

Accessing Enrichment Data

The Apstra Flow collector can enrich flow records with additional information. The following sections show the various enrichment options and corresponding allowed ports.

DNS

Required when `EF_PROCESSOR_ENRICH_IPADDR_DNS_ENABLE` is `true`.

Table 27: DNS Allowed Port

Protocol	Port	Direction	Description
UDP	53	out	DNS

SNMP

Required when `EF_PROCESSOR_ENRICH_NETIF_SNMP_ENABLE` is `true`.

Table 28: SNMP Allowed Port

Protocol	Port	Direction	Description
UDP	161	out	Network interface attributes through SNMP.

OpenSearch

Specify one of the TCP allowed ports when `EF_OUTPUT_OPENSEARCH_ENABLE` is `true`.

Table 29: OpenSearch Allowed Ports

Protocol	Port	Direction	Description
TCP	9200	out	OpenSearch REST API.
TCP	5601	out	OpenSearch dashboards, GUI and API.

VM Sizing

We conducted tests for VM sizing using Apstra Flow OVA on ESXi 8.0 (Table 5 on page 960). VM sizes and storage results are listed in Table 6 on page 961.

NOTE: Other workloads were not active on the system during testing. "Noisy neighbors" or other resource contention could negatively impact the results in production environments.

Table 30: Components Tested for VM Sizing

Component	Description
CPU	AMD EPYC 7702 (64-core Zen2), locked at 2.9GHz to avoid thermal throttling.

Table 30: Components Tested for VM Sizing *(Continued)*

Component	Description
Memory	256GB DDR4 3200MT/s, all 8 memory channels populated for maximum memory bandwidth.
Storage (Direct-SSD)	4TB SATA SSD
Storage (NFS)	8x HDD (RAID10) with NVMe read/write cache through 10Gbe

Table 31: VM Sizing and Storage Results

VM Size	CPU and Memory Sizing	Ingest Capacity
Default (Medium) VM	• CPU: 16vCPUs • Memory: 64 GB	Direct-SSD: • Recommended ingest: 12,000 records per second. • Burst ingest: up to 40,000 flow records per second. NFS • Recommended ingest: 10,000 records per second. • Burst ingest: up to 40,000 flow records per second.

Table 31: VM Sizing and Storage Results *(Continued)*

VM Size	CPU and Memory Sizing	Ingest Capacity
Small VM	• CPU: 8vCPUs • Memory: 32 GB	Direct-SSD: • Recommended ingest: 6,500 records per second. • Burst ingest: Up to 21,500 flow records per second. NFS • Recommended ingest: 5,500 records per second. • Burst ingest: up to 21,500 flow records per second.
Large VM	• CPU: 24vCPUs • Memory: 64 GB	Direct-SSD: • Recommended ingest: 16,000 records per second. • Burst ingest: up to 53,000 flow records per second. NFS • Recommended ingest: 13,000 records per second. • Burst ingest: up to 53,000 flow records per second.
X-Large VM (custom)	Contact your Juniper sales representative for guidance on creating a cluster for a custom deployment.	Greater than 15,000 FPS.

Licensing

The Apstra Flow collector operates with the integration of a Juniper Apstra license. Premium license holders benefit from enhanced features and an elevated flow rate capacity. In contrast, users with the

basic license have a constraint of up to 500 flows per second. For information about activating your license, see the Juniper Licensing User Guide.

CHAPTER 35

Dashboards

IN THIS CHAPTER

Apstra Flow Dashboards

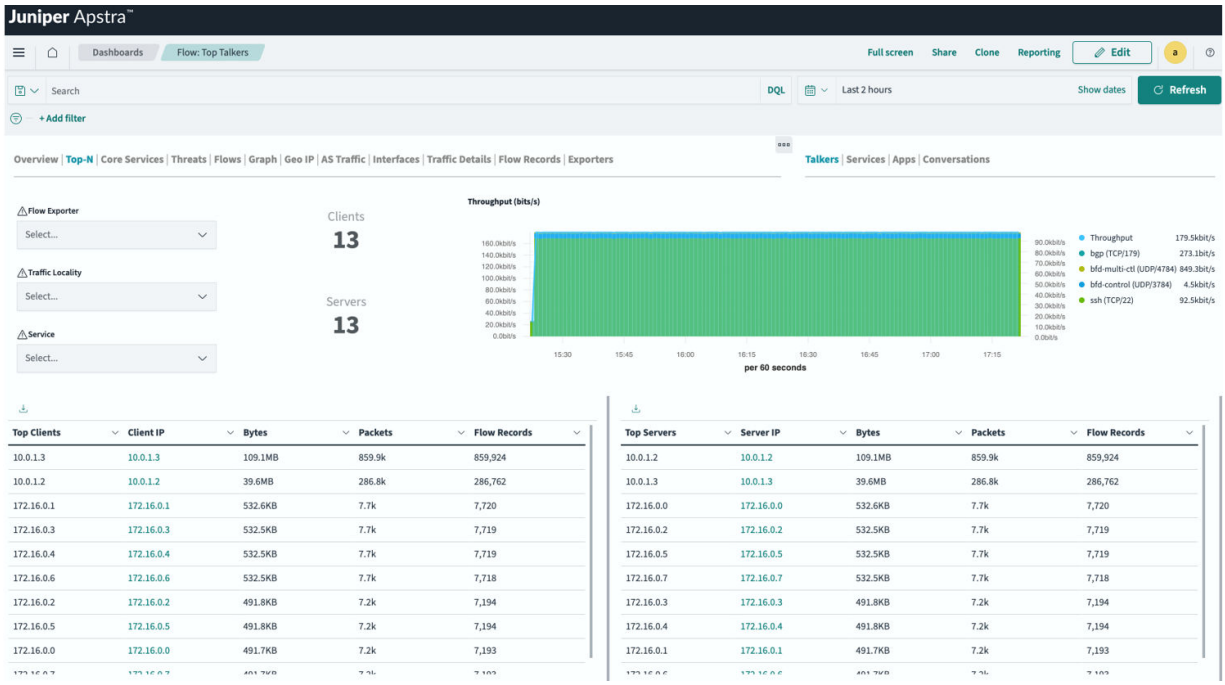
IN THIS SECTION

The following sections show examples of use cases using the Apstra Flow dashboards used to manage your network. See the Juniper Apstra Flow Installation Guide for information about the different ways to launch the dashboards.

Top-N Dashboard

Understanding and optimizing network performance is critical for any network operator. Apstra Flow lets you continuously learn about what is traversing your network at any given time and helps you continually improve network functionality. The Top-N Dashboard (Figure 1 on page 965) shows an example of which hosts are most active on your network. You can filter this traffic by talkers (traffic source and destination) and services such as SSH and HTTPS, Apps, and Conversations. You can also add filters to further enrich the data shown. For example, you might want to narrow in on a particular source or destination, service, or TCP flags to only show the connection status.

Figure 7: Top-N Dashboard



Threats Dashboard

Apstra Flow can perform basic threat detection for flows. The Threats dashboard shows any DDoS, port scans, and brute force attempts on your network. The following example (Figure 2 on page 966) shows the number of repeated SSH sessions that were sent between hosts in the network. Apstra Flow displays these sessions as brute-force attempts.

NOTE: You can configure the threats dashboard to specify the type of flow information you want to monitor, either IPFIX or sFlow v5 (Figure 3 on page 966).

Figure 8: Threats Dashboard

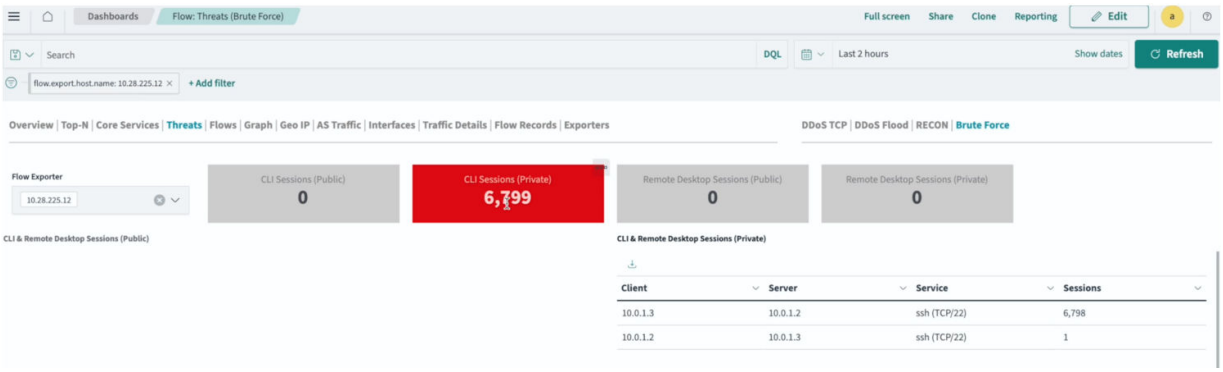
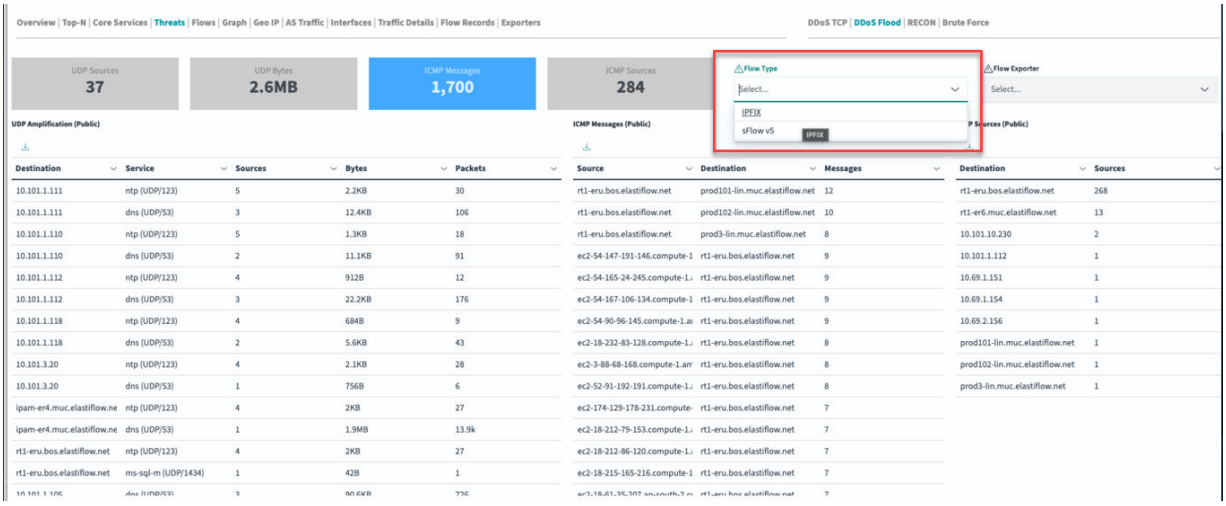


Figure 9: Flow Types



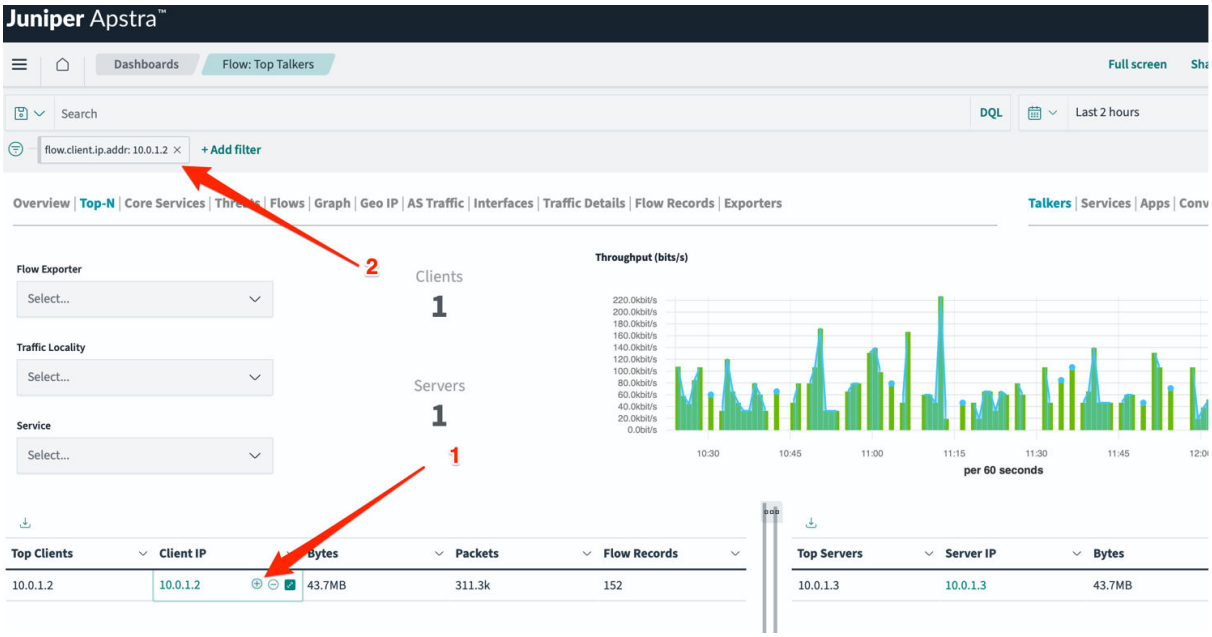
Apstra Flow allows you to trigger the flow results yourself by using the open-source Hping3 network scanning tool. This tool can be used to send different packet types for security vulnerability testing.

Performance and Planning Dashboards

Gaining inside knowledge into your network can help you rebalance your applications and capacity planning, but to do that, you need to see how the flows are impacting individual interfaces. To see a particular traffic flow in the Apstra GUI, you can create a filter that persists across the top-level tabs in the Apstra Flow dashboard.

For example, from the **Flow: Top Talkers** tab (Figure 4 on page 967), we chose the most talkative source IP (src) address (indicated by arrow 1). By hovering over that IP and clicking the + sign, we created a filter (indicated by arrow 2).

Figure 10: Example of Top-N Talkers

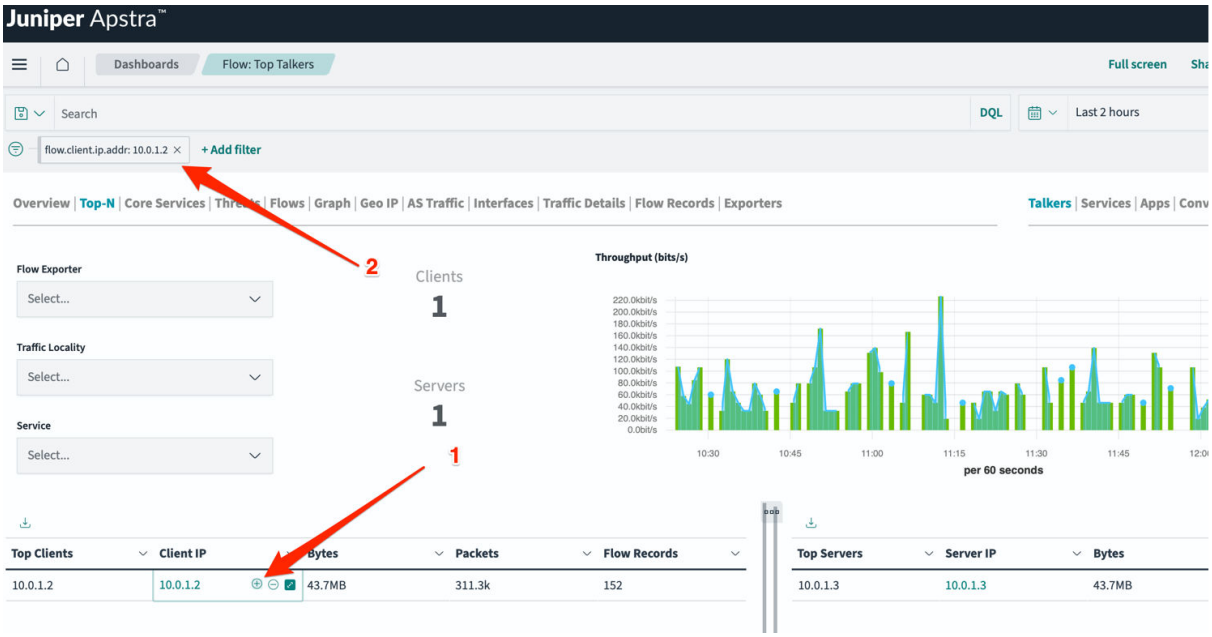


This filter also applies to other tabs, such as the Interface tab shown in Figure 5 on page 968. This tab shows the interfaces on which your chosen IP address communicates.

From the Interfaces dashboard, you can:

- Identify link saturation: See which interfaces are experiencing high traffic volume, helping you determine where to rebalance applications or add capacity.
- Drill down further: Select individual flow exporters (switches), interface types (ingress/egress), and specific interfaces for even more granular analysis.

Figure 11: Interfaces Dashboard



This use case shows you where you are having issues with link saturation in your network. This is useful for capacity planning exercises, or troubleshooting cloud-native applications in a data center.

CHAPTER 36

Supported Flow Records

IN THIS CHAPTER

Supported Information Elements

Apstra Flow receives and enriches network flow records and telemetry sent from network devices, cloud services and applications using IPFIX, NetFlow, and sFlow information elements (IEs). These IEs contain attribute values that are related to the observed network traffic. The Apstra Flow collector supports IEs from many vendors and multiple networking technologies.

NOTE: For the complete set of IEs supported by each license tier, see the record type specific lists of IEs below:

RELATED DOCUMENTATION

- IPFIX IEs | 970
- NetFlow IEs | 1035
- sFlow IEs (Flow Samples) | 1087
- sFlow IEs (Counter Samples) | 1113

IPFIX IEs

IN THIS SECTION

Apstra Flow supports the following standards-based IPFIX information elements (IEs):

Standards-based IPFIX IEs (PEN: 0)

ID	Name
1	octetDeltaCount
2	packetDeltaCount
3	deltaFlowCount
4	protocolIdentifier
5	ipClassOfService
6	tcpControlBits
7	sourceTransportPort
8	sourceIPv4Address
9	sourceIPv4PrefixLength
10	ingressInterface

(Continued)

ID	Name
11	destinationTransportPort
12	destinationIPv4Address
13	destinationIPv4PrefixLength
14	egressInterface
15	ipNextHopIPv4Address
16	bgpSourceAsNumber
17	bgpDestinationAsNumber
18	bgpNextHopIPv4Address
19	postMCastPacketDeltaCount
20	postMCastOctetDeltaCount
21	flowEndSysUpTime
22	flowStartSysUpTime
23	postOctetDeltaCount
24	postPacketDeltaCount
25	minimumIpTotalLength
26	maximumIpTotalLength
27	sourceIPv6Address
28	destinationIPv6Address
29	sourceIPv6PrefixLength

(Continued)

ID	Name
30	destinationIPv6PrefixLength
31	flowLabelIPv6
32	icmpTypeCodeIPv4
33	igmpType
34	samplingInterval
35	samplingAlgorithm
36	flowActiveTimeout
37	flowIdleTimeout
38	engineType
39	engineId
40	exportedOctetTotalCount
41	exportedMessageTotalCount
42	exportedFlowRecordTotalCount
44	sourceIPv4Prefix
45	destinationIPv4Prefix
46	mplsTopLabelType
47	mplsTopLabelIPv4Address
48	samplerId
49	samplerMode

(Continued)

ID	Name
50	samplerRandomInterval
51	classId
52	minimumTTL
53	maximumTTL
54	fragmentIdentification
55	postIpClassOfService
56	sourceMacAddress
57	postDestinationMacAddress
58	vlanId
59	postVlanId
60	ipVersion
61	flowDirection
62	ipNextHopIPv6Address
63	bgpNextHopIPv6Address
64	ipv6ExtensionHeaders
65	transportPacketLoss `Cisco Legacy`
66	transportUnreachability `Cisco Legacy`
67	transportLatency `Cisco Legacy`
68	dataPoints `Cisco Legacy`

(Continued)

ID	Name
69	variance `Cisco Legacy`
70	mplsTopLabelStackSection
71	mplsLabelStackSection2
72	mplsLabelStackSection3
73	mplsLabelStackSection4
74	mplsLabelStackSection5
75	mplsLabelStackSection6
76	mplsLabelStackSection7
77	mplsLabelStackSection8
78	mplsLabelStackSection9
79	mplsLabelStackSection10
80	destinationMacAddress
81	postSourceMacAddress
82	interfaceName
83	interfaceDescription
84	samplerName
85	octetTotalCount
86	packetTotalCount
87	flagsAndSamplerId

(Continued)

ID	Name
88	fragmentOffset
89	forwardingStatus
90	mplsVpnRouteDistinguisher
91	mplsTopLabelPrefixLength
92	srcTrafficIndex
93	dstTrafficIndex
94	applicationDescription
95	applicationId
96	applicationName
97	subApplicationTag `Cisco Legacy`
98	postIpDiffServCodePoint
99	multicastReplicationFactor
100	className
101	classificationEngineId
102	layer2packetSectionOffset
103	layer2packetSectionSize
104	layer2packetSectionData
105	applicationVersion `Cisco Legacy`
106	applicationVersionName `Cisco Legacy`

(Continued)

ID	Name
107	applicationVendor `Cisco Legacy`
109	subApplicationName `Cisco Legacy`
110	subApplicationDescription `Cisco Legacy`
111	templateParameterRangeEnd `Cisco Legacy`
128	bgpNextAdjacentAsNumber
129	bgpPrevAdjacentAsNumber
130	exporterIPv4Address
131	exporterIPv6Address
132	droppedOctetDeltaCount
133	droppedPacketDeltaCount
134	droppedOctetTotalCount
135	droppedPacketTotalCount
136	flowEndReason
137	commonPropertiesId
138	observationPointId
139	icmpTypeCodeIPv6
140	mplsTopLabelIPv6Address
141	lineCardId
142	portId

(Continued)

ID	Name
143	meteringProcessId
144	exportingProcessId
145	templateId
146	wlanChannelId
147	wlanSSID
148	flowId
149	observationDomainId
150	flowStartSeconds
151	flowEndSeconds
152	flowStartMilliseconds
153	flowEndMilliseconds
154	flowStartMicroseconds
155	flowEndMicroseconds
156	flowStartNanoseconds
157	flowEndNanoseconds
158	flowStartDeltaMicroseconds
159	flowEndDeltaMicroseconds
160	systemInitTimeMilliseconds
161	flowDurationMilliseconds

(Continued)

ID	Name
162	flowDurationMicroseconds
163	observedFlowTotalCount
164	ignoredPacketTotalCount
165	ignoredOctetTotalCount
166	notSentFlowTotalCount
167	notSentPacketTotalCount
168	notSentOctetTotalCount
169	destinationIPv6Prefix
170	sourceIPv6Prefix
171	postOctetTotalCount
172	postPacketTotalCount
173	flowKeyIndicator
174	postMCastPacketTotalCount
175	postMCastOctetTotalCount
176	icmpTypeIPv4
177	icmpCodeIPv4
178	icmpTypeIPv6
179	icmpCodeIPv6
180	udpSourcePort

(Continued)

ID	Name
181	udpDestinationPort
182	tcpSourcePort
183	tcpDestinationPort
184	tcpSequenceNumber
185	tcpAcknowledgementNumber
186	tcpWindowSize
187	tcpUrgentPointer
188	tcpHeaderLength
189	ipHeaderLength
190	totalLengthIPv4
191	payloadLengthIPv6
192	ipTTL
193	nextHeaderIPv6
194	mplsPayloadLength
195	ipDiffServCodePoint
196	ipPrecedence
197	fragmentFlags
198	octetDeltaSumOfSquares
199	octetTotalSumOfSquares

(Continued)

ID	Name
200	mplsTopLabelTTL
201	mplsLabelStackLength
202	mplsLabelStackDepth
203	mplsTopLabelExp
204	ipPayloadLength
205	udpMessageLength
206	isMulticast
207	ipv4IHL
208	ipv4Options
209	tcpOptions
210	paddingOctets
211	collectorIPv4Address
212	collectorIPv6Address
213	exportInterface
214	exportProtocolVersion
215	exportTransportProtocol
216	collectorTransportPort
217	exporterTransportPort
218	tcpSynTotalCount

(Continued)

ID	Name
219	tcpFinTotalCount
220	tcpRstTotalCount
221	tcpPshTotalCount
222	tcpAckTotalCount
223	tcpUrgTotalCount
224	ipTotalLength
225	postNATSourceIPv4Address
226	postNATDestinationIPv4Address
227	postNAPTSourceTransportPort
228	postNAPTDestinationTransportPort
229	natOriginatingAddressRealm
230	natEvent
231	initiatorOctets
232	responderOctets
233	firewallEvent
234	ingressVRFID
235	egressVRFID
236	VRFname
237	postMplsTopLabelExp

(Continued)

ID	Name
238	tcpWindowScale
239	biflowDirection
240	ethernetHeaderLength
241	ethernetPayloadLength
242	ethernetTotalLength
243	dot1qVlanId
244	dot1qPriority
245	dot1qCustomerVlanId
246	dot1qCustomerPriority
247	metroEvcId
248	metroEvcType
249	pseudoWireId
250	pseudoWireType
251	pseudoWireControlWord
252	ingressPhysicalInterface
253	egressPhysicalInterface
254	postDot1qVlanId
255	postDot1qCustomerVlanId
256	ethernetType

(Continued)

ID	Name
257	postIpPrecedence
258	collectionTimeMilliseconds
259	exportSctpStreamId
260	maxExportSeconds
261	maxFlowEndSeconds
262	messageMD5Checksum
263	messageScope
264	minExportSeconds
265	minFlowStartSeconds
266	opaqueOctets
267	sessionScope
268	maxFlowEndMicroseconds
269	maxFlowEndMilliseconds
270	maxFlowEndNanoseconds
271	minFlowStartMicroseconds
272	minFlowStartMilliseconds
273	minFlowStartNanoseconds
274	collectorCertificate
275	exporterCertificate

(Continued)

ID	Name
276	dataRecordsReliability
277	observationPointType
278	newConnectionDeltaCount
279	connectionSumDurationSeconds
280	connectionTransactionId
281	postNATSourceIPv6Address
282	postNATDestinationIPv6Address
283	natPoolId
284	natPoolName
285	anonymizationFlags
286	anonymizationTechnique
287	informationElementIndex
288	p2pTechnology
289	tunnelTechnology
290	encryptedTechnology
294	bgpValidityState
295	IPSecSPI
296	greKey
297	natType

(Continued)

ID	Name
298	initiatorPackets
299	responderPackets
300	observationDomainName
301	selectionSequenceId
302	selectorId
303	informationElementId
304	selectorAlgorithm
305	samplingPacketInterval
306	samplingPacketSpace
307	samplingTimeInterval
308	samplingTimeSpace
309	samplingSize
310	samplingPopulation
311	samplingProbability
312	dataLinkFrameSize
313	ipHeaderPacketSection
314	ipPayloadPacketSection
315	dataLinkFrameSection
316	mplsLabelStackSection

(Continued)

ID	Name
317	mplsPayloadPacketSection
318	selectorIdTotalPktsObserved
319	selectorIdTotalPktsSelected
320	absoluteError
321	relativeError
322	observationTimeSeconds
323	observationTimeMilliseconds
324	observationTimeMicroseconds
325	observationTimeNanoseconds
326	digestHashValue
327	hashIPPayloadOffset
328	hashIPPayloadSize
329	hashOutputRangeMin
330	hashOutputRangeMax
331	hashSelectedRangeMin
332	hashSelectedRangeMax
333	hashDigestOutput
334	hashInitialiserValue
335	selectorName

(Continued)

ID	Name
336	upperCILimit
337	lowerCILimit
338	confidenceLevel
339	informationElementDataType
340	informationElementDescription
341	informationElementName
342	informationElementRangeBegin
343	informationElementRangeEnd
344	informationElementSemantics
345	informationElementUnits
346	privateEnterpriseNumber
347	virtualStationInterfaceId
348	virtualStationInterfaceName
349	virtualStationUUID
350	virtualStationName
351	layer2SegmentId
352	layer2OctetDeltaCount
353	layer2OctetTotalCount
354	ingressUnicastPacketTotalCount

(Continued)

ID	Name
355	ingressMulticastPacketTotalCount
356	ingressBroadcastPacketTotalCount
357	egressUnicastPacketTotalCount
358	egressBroadcastPacketTotalCount
359	monitoringIntervalStartMilliSeconds
360	monitoringIntervalEndMilliSeconds
361	portRangeStart
362	portRangeEnd
363	portRangeStepSize
364	portRangeNumPorts
365	staMacAddress
366	staIPv4Address
367	wtpMacAddress
368	ingressInterfaceType
369	egressInterfaceType
370	rtpSequenceNumber
371	userName
372	applicationCategoryName
373	applicationSubCategoryName

(Continued)

ID	Name
374	applicationGroupName
375	originalFlowsPresent
376	originalFlowsInitiated
377	originalFlowsCompleted
378	distinctCountOfSourceIPAddress
379	distinctCountOfDestinationIPAddress
380	distinctCountOfSourceIPv4Address
381	distinctCountOfDestinationIPv4Address
382	distinctCountOfSourceIPv6Address
383	distinctCountOfDestinationIPv6Address
384	valueDistributionMethod
385	rfc3550JitterMilliseconds
386	rfc3550JitterMicroseconds
387	rfc3550JitterNanoseconds
388	dot1qDEI
389	dot1qCustomerDEI
390	flowSelectorAlgorithm
391	flowSelectedOctetDeltaCount
392	flowSelectedPacketDeltaCount

(Continued)

ID	Name
393	flowSelectedFlowDeltaCount
394	selectorIDTotalFlowsObserved
395	selectorIDTotalFlowsSelected
396	samplingFlowInterval
397	samplingFlowSpacing
398	flowSamplingTimeInterval
399	flowSamplingTimeSpacing
400	hashFlowDomain
401	transportOctetDeltaCount
402	transportPacketDeltaCount
403	originalExporterIPv4Address
404	originalExporterIPv6Address
405	originalObservationDomainId
406	intermediateProcessId
407	ignoredDataRecordTotalCount
408	dataLinkFrameType
409	sectionOffset
410	sectionExportedOctets
411	dot1qServiceInstanceTag

(Continued)

ID	Name
412	dot1qServiceInstanceId
413	dot1qServiceInstancePriority
414	dot1qCustomerSourceMacAddress
415	dot1qCustomerDestinationMacAddress
416	layer2OctetDeltaCount
417	postLayer2OctetDeltaCount
418	postMCastLayer2OctetDeltaCount
419	layer2OctetTotalCount
420	postLayer2OctetTotalCount
421	postMCastLayer2OctetTotalCount
422	minimumLayer2TotalLength
423	maximumLayer2TotalLength
424	droppedLayer2OctetDeltaCount
425	droppedLayer2OctetTotalCount
426	ignoredLayer2OctetTotalCount
427	notSentLayer2OctetTotalCount
428	layer2OctetDeltaSumOfSquares
429	layer2OctetTotalSumOfSquares
430	layer2FrameDeltaCount

(Continued)

ID	Name
431	layer2FrameTotalCount
432	pseudoWireDestinationIPv4Address
433	ignoredLayer2FrameTotalCount
434	mibObjectValueInteger
435	mibObjectValueOctetString
436	mibObjectValueOID
437	mibObjectValueBits
438	mibObjectValueIPAddress
439	mibObjectValueCounter
440	mibObjectValueGauge
441	mibObjectValueTimeTicks
442	mibObjectValueUnsigned
445	mibObjectIdentifier
446	mibSubIdentifier
447	mibIndexIndicator
448	mibCaptureTimeSemantics
449	mibContextEngineID
450	mibContextName
451	mibObjectName

(Continued)

ID	Name
452	mibObjectDescription
453	mibObjectSyntax
454	mibModuleName
455	mobileIMSI
456	mobileMSISDN
457	httpStatusCode
458	sourceTransportPortsLimit
459	httpRequestMethod
460	httpRequestHost
461	httpRequestTarget
462	httpMessageVersion
463	natInstanceID
464	internalAddressRealm
465	externalAddressRealm
466	natQuotaExceededEvent
467	natThresholdEvent
468	httpUserAgent
469	httpContentType
470	httpReasonPhrase

(Continued)

ID	Name
471	maxSessionEntries
472	maxBIBEntries
473	maxEntriesPerUser
474	maxSubscribers
475	maxFragmentsPendingReassembly
476	addressPoolHighThreshold
477	addressPoolLowThreshold
478	addressPortMappingHighThreshold
479	addressPortMappingLowThreshold
480	addressPortMappingPerUserHighThreshold
481	globalAddressMappingHighThreshold
482	vpnIdentifier
483	bgpCommunity
486	bgpExtendedCommunity
489	bgpLargeCommunity

Cisco (PEN: 9)

- cTag
- scTrafficProcessorId
- scSourceIpSample
- scDestinationIpSample

- scFlowContextId
- scSubscriberId
- scPackageId
- scServiceId
- scProtocolId
- scSkipppedSessions
- scInitiatingSide
- scReportTime
- scTransactionDurationMillisec
- scTimeFrame
- scSessionUpstreamVolume
- scSessionDownstreamVolume
- scProtocolSignature
- scZoneId
- scFlavorId
- scFlowCloseMode
- scAccessString
- scInfoString
- scClientPort
- scServerPort
- scSubscriberCounterId
- scServiceUsageCounterId
- scBreachState
- scReason
- scConfiguredDuration
- scDuration

- scEndTime
- scUpstreamVolume
- scDownstreamVolume
- scSessions
- scSeconds
- scPackageCounterId
- scGeneratorId
- scServiceGlobalCounterId
- scConcurrentSessions
- scActiveSubscribers
- scTotalActiveSubscribers
- scLinkId
- scAttackId
- scAttackIp
- scAttackOtherIp
- scAttackPortNumber
- scAttackType
- scAttackSide
- scAttackIpProtocol
- scAttacks
- scAttackMaliciousSessions
- INGRESS_ACL_ID
- EGRESS_ACL_ID
- FW_EXT_EVENT
- FW_EVENT_LEVEL
- FW_EVENT_LEVEL_ID

- FW_CONFIGURED_VALUE
- FW_ERM_EXT_EVENT
- FW_ERM_EXT_EVENT_DESC
- audio rtp packets lost
- audio rtp packets expected
- audio rtp fwd out-of-sequence sum
- audio rtp seconds ok
- audio rtp seconds concealed
- audio rtp seconds concealed severe
- audio rtp jitter ticks
- audio g107 impairment
- audio g107 lossRate
- audio g107 codec baseline
- audio g107 codec baseline bpl
- audio g107 impairment one-way-delay
- audio concealment ratio now
- audio concealment ratio minimum
- audio concealment ratio maximum
- audio concealment time
- audio speech time
- audio packets ok
- audio packets cs
- audio packets scs
- audio packets rtp
- audio packets silence
- audio duration receive

- audio duration receive voice
- audio duration early packet
- audio duration clock adjust
- audio duration playout increase
- audio duration playout decrease
- audio duration late discard
- audio frame size
- audio frames-per-packet
- audio frame arriving times difference
- audio frame arriving times difference vari
- audio noise level current
- audio noise level average
- audio noise level minimum
- audio noise level maximum
- audio noise level configured
- audio snr current
- audio snr average
- audio snr minimum
- audio snr maximum
- audio snr configured
- vxlan sgt
- vxlan flags
- vxlan vtep input
- vxlan vtep output
- SGT_SOURCE_TAG
- SGT_DESTINATION_TAG

- SGT_SOURCE_NAME
- SGT_DESTINATION_NAME
- flow cts switch derived-sgt
- FW_EXT_EVENT
- FW_BLACKOUT_SECS
- FW_HALFOPEN_HIGH
- FW_HALFOPEN_RATE
- FW_ZONEPAIR_ID
- FW_MAX_SESSIONS
- FW_ZONEPAIR_NAME
- FW_EXT_EVENT_DESC
- FW_SUMMARY_PKT_CNT
- FW_HALFOPEN_CNT
- waas dre input
- waas dre output
- waas lz input
- waas lz output
- waas original bytes
- waas optimised bytes
- waas application
- waas class
- waas connection mode
- waas bytes input
- waas bytes output
- PACKETS_DROPPED
- counter packets dropped permanent

- PACKET_RATE
- BYTE_RATE
- application media bytes counter
- application media bytes counter permanent
- application media bytes rate
- application media packets counter
- application media packets counter permanen
- application media packets rate
- application media packets rate variation
- application media event
- monitor event
- timestamp interval
- transport packets expected counter
- transport packets expected counter permane
- transport round-trip-time
- transport event packet-loss counter
- transport event packet-loss counter perman
- transport packets lost counter
- transport packets lost counter permanent
- transport packets lost rate
- transport rtp ssrc
- transport rtp jitter mean
- transport rtp jitter minimum
- transport rtp jitter maximum
- misc unsupported
- counter bytes rate per-flow

- counter bytes rate per-flow min
- counter bytes rate per-flow max
- counter packets rate per-flow
- counter packets rate per-flow min
- counter packets rate per-flow max
- application media bytes rate per-flow
- application media bytes rate per-flow min
- application media bytes rate per-flow max
- application media packets rate variation m
- application media packets rate variation m
- transport rtp flow count
- transport rtp payload-type
- transport packets lost counter min
- transport packets lost counter max
- transport event packet-loss counter min
- transport event packet-loss counter max
- transport packets lost rate min
- transport packets lost rate max
- transport tcp flow count
- transport round-trip-time sum
- transport round-trip-time samples
- transport round-trip-time min
- transport round-trip-time max
- metadata global-session-id
- metadata multi-party-session-id
- metadata clock-rate

- server response time average
- refused sessions
- client network delay average
- server network delay average
- NETWORK_DELAY_AVG {network delay average
- application delay average
- session time minimum
- session time maximum
- session time average
- transaction time average
- closed sessions
- retransmitted packets
- transport bytes out-of-order
- client throughput average
- unresponsive sessions
- transport packets out-of-order
- IPv4 source observation node
- IPv4 destination observation node
- IPv6 source observation node
- IPv6 destination observation node
- pfr one-way-delay sum
- pfr one-way-delay samples
- pfr one-way-delay
- packet arrival timestamp
- transport tcp window-size minimum
- transport tcp window-size maximum

- transport tcp window-size average
- transport tcp maximum-segment-size
- transport tcp window-size sum
- tcpWindowSizeSum
- transport rtp jitter mean sum
- application media packets rate variation
- transport tcp window-size average sum
- transport rtp jitter inter arrival sum
- transport rtp jitter inter arrival samples
- transport rtp jitter inter arrival mean
- pfr site source id ipv4
- pfr site destination id ipv4
- transport bytes lost
- transport bytes expected
- transport bytes lost rate
- network delay sum
- network delay sample
- pfr counter event error traffic-class miti
- pfr counter event error traffic-class miti
- pfr counter event error traffic-class miti
- pfr site source prefix ipv4
- pfr site destination prefix ipv4
- pfr site source prefix ipv6
- pfr site destination prefix ipv6
- pfr site source prefix mask ipv4
- pfr site destination prefix mask ipv4

- pfr site source prefix mask ipv6
- pfr site destination prefix mask ipv6
- pfr service provider tag identifier
- pfr label identifier
- application voice number called
- application voice number calling
- application voice setup time
- application voice call duration
- application voice rx bad-packet
- application voice rx out-of-sequence
- application voice codec id
- application voice play delay current
- application voice play delay minimum
- application voice play delay maximum
- application voice sip call-id
- application voice router global-call-id
- application voice delay round-trip
- application voice delay end-point
- application voice r-factor 1
- application voice r-factor 2
- application voice mos conversation
- application voice mos listening
- application voice concealment-ratio averag
- application voice jitter configured type
- application voice jitter configured minimu
- application voice jitter configured maximu

- application voice jitter configured initia
- application voice rx early-packet count
- application voice rx late-packet count
- application voice jitter buffer-overrun
- application voice packet conceal-count
- bandwidth used
- bandwidth used percentage
- application video resolution width last
- application video resolution height last
- application video frame rate
- application video payload bitrate average
- application video payload bitrate fluctuat
- application video frame I counter frames
- application video frame I counter packets
- application video frame I counter bytes
- application video frame STR counter frames
- application video frame STR counter packet
- application video frame STR counter bytes
- application video frame LTR counter frames
- application video frame LTR counter packet
- application video frame LTR counter bytes
- application video frame super-P counter fr
- application video frame super-P counter pa
- application video frame super-P counter by
- application video frame NR counter frames
- application video frame NR counter packets

- application video frame NR counter bytes
- application video frame I slice-quantizati
- application video frame STR slice-quantiza
- application video frame LTR slice-quantiza
- application video frame super-P slice-quan
- application video frame NR slice-quantizat
- application video eMOS compression bitstre
- application video eMOS compression network
- application video frame I counter packets
- application video frame STR counter packet
- application video frame LTR counter packet
- application video frame super-P counter pa
- application video frame NR counter packets
- application video frame percentage damaged
- application video eMOS packet-loss bitstre
- application video eMOS packet-loss network
- application video scene-complexity
- application video level-of-motion
- transport rtp sequence-number
- transport rtp sequence-number last
- iOAM my node-id
- iOAM my node name
- start timestamp
- end timestamp
- IOAM packet counter
- IOAM byte count

- IOAM cs0 packet counter
- IOAM cs0 byte count
- IOAM cs1 packet counter
- IOAM cs1 byte count
- IOAM cs2 packet counter
- IOAM cs2 byte count
- IOAM cs3 packet counter
- IOAM cs3 byte count
- IOAM cs4 packet counter
- IOAM cs4 byte count
- IOAM cs5 packet counter
- IOAM cs5 byte count
- IOAM cs6 packet counter
- IOAM cs6 byte count
- IOAM cs7 packet counter
- IOAM cs7 byte count
- IOAM lost packet counter
- IOAM duplicate packet counter
- IOAM reordered packet counter
- IOAM highest PPC sequence number
- iOAM node-id
- ipv6 protocol filed
- iOAM E2E Header
- iOAM Path Map
- iOAM number of nodes
- iOAM node1 id

- iOAM node1 in if id
- iOAM node1 eif id
- iOAM node2 id
- iOAM node2 in if id
- iOAM node2 eif id
- iOAM node3 id
- iOAM node3 in if id
- iOAM node3 eif id
- iOAM node4 id
- iOAM node4 in if id
- iOAM node4 eif id
- iOAM Application metadata
- iOAM sfc-id
- iOAM sfc validated count
- iOAM sfc invalidated count
- pfr br ipv4 address
- pfr status
- reason id
- threshold
- pfr priority
- long-term round-trip-time
- mos below
- rsvp bw pool
- flow left time
- bw percentage
- bw fee

- transport source-port min
- transport source-port max
- transport destination-port min
- transport destination-port max
- capacity
- ingress bw
- max ingress bw
- egress bw
- max egress bw
- ingress rollup bw
- egress rollup bw
- kth rollup bw
- link group name
- bgp community
- bgp prepend
- entrance downgrade
- discard rollup count
- services pfr class-tag-id
- services pfr mc-id
- sip header from uri host ip addr
- sip header from uri userinfo user
- sip header to uri host ip addr
- sip header to uri userinfo user
- sip sess duration
- sip sess end_reason
- sip sess_dialed

- sip sess_connected
- sip sess_failed
- AAA_USERNAME
- XLATE_SRC_ADDR_IPV4
- XLATE_DST_ADDR_IPV4
- XLATE_SRC_PORT
- XLATE_DST_PORT
- FW_EVENT
- artClientNetworkTimeLongLivedMaximum
- artClientNetworkTimeLongLivedMinimum
- artServerNetworkTimeLongLivedMaximum
- artServerNetworkTimeLongLivedMinimum
- policy qos classification hierarchy
- c3pl class cce-id
- c3pl class name
- c3pl class type
- c3pl policy cce-id
- c3pl policy name
- c3pl policy type
- interface input fex-node-id
- interface output fex-node-id
- interface power
- monitor device-type
- connection server counter bytes network
- connection client counter bytes network
- wireless afd drop packets

- wireless afd accept packets
- wireless afd drop bytes
- wireless afd accept bytes
- connection concurrent-connections
- application transaction counter new
- services waas segment
- services waas passthrough-reason
- connection delay network long-lived to-ser
- connection delay network long-lived to-cli
- connection delay network long-lived client
- connection delay network client-to-server
- connection delay network to-server num-sam
- connection delay network to-client num-sam
- artClientpackets
- artServerpackets
- connection client counter bytes retransmit
- connection client counter packets retransm
- connection server counter bytes retransmit
- connection server counter packets retransm
- connection transaction counter complete
- connection transaction duration sum
- connection transaction duration max
- connection transaction duration min
- art count new connections
- art count responses
- art count responses histogram bucket1

- art count responses histogram bucket2
- art count responses histogram bucket3
- art count responses histogram bucket4
- art count responses histogram bucket5
- art count responses histogram bucket6
- art count responses histogram bucket7
- connection delay response to-server histog
- connection delay response to-server sum
- connection delay response to-server max
- connection delay response to-server min
- connection delay application sum
- connection delay application max
- connection delay application min
- connection delay response client-to-server
- connection delay response client-to-server
- connection delay response client-to-server
- connection delay network client-to-server
- connection delay network client-to-server
- connection delay network client-to-server
- onnection delay network to-client sum
- connection delay network to-client max
- connection delay network to-client min
- connection delay network to-server sum
- connection delay network to-server max
- connection delay network to-server min
- mos worst 100

- mos quality
- mos total count
- application http uri statistics
- policy qos queue index
- policy qos queue drops
- datalink event
- datalink event extended
- l4r server ipv4 address
- l4r server transport port
- l4r server ipv6 address
- l4r event
- l4r event timestamp
- pbhk mapped ipv4 address
- pbhk mapped transport port
- pbhk event
- pbhk event timestamp
- ETTA_INITIAL_DATA_PACKET
- ETTA_SEQUENCE_OF_PACKET_LENGTHS_AND_TIMES
- ETTA_SEQUENCE_OF_APPLICATION_LENGTHS_AND_TIMES
- ETAByteDistribution
- ETTA_TLS_RECORDS
- ETTA_TLS_CIPHER_SUITES
- ETTA_TLS_EXTENSIONS
- ETTA_TLS_VERSION
- ETTA_TLS_KEY_LENGTH
- ETTA_TLS_SESSION_ID

- ETTA_TLS_RANDOM
- ETTA_TLS_EXTENSION_LENGTHS
- ETTA_TLS_EXTENSION_TYPES
- application family name
- application set name
- application category name
- application sub category name
- application group name
- AVCSubApplicationValue
- connection client ipv4 address
- connection server ipv4 address
- connection client ipv6 address
- connection server ipv6 address
- connection client transport port
- connection server transport port
- connection id
- application traffic-class
- application business-relevance
- nvzFlowUDID
- nvzFlowLoggedInUser
- nvzFlowOSName
- nvzFlowOSVersion
- nvzFlowSystemManufacturer
- nvzFlowSystemType
- nvzFlowProcessAccount
- nvzFlowParentProcessAccount

- nvzFlowProcessName
- nvzFlowProcessHash
- nvzFlowParentProcessName
- nvzFlowParentProcessHash
- nvzFlowDNSSuffix
- nvzFlowDestinationHostname
- nvzFlowL4ByteCountIn
- nvzFlowL4ByteCountOut
- nvzFlowOSEdition
- nvzFlowModuleNameList
- nvzFlowModuleHashList
- nvzFlowCoordinatesList
- nvzFlowInterfaceInfoUID
- nvzFlowInterfaceIndex
- nvzFlowInterfaceType
- nvzFlowInterfaceName
- nvzFlowInterfaceDetailsList
- nvzFlowInterfaceMac
- nvzFlowUserAccountType
- nvzFlowProcessAccountType
- nvzFlowParentProcessAccountType
- overlay session id input
- overlay session id output
- routing vrf service
- tloc table overlay session id
- tloc local system ip address

- tloc local color
- tloc remote system ip address
- tloc remote color
- tloc tunnel protocol
- connection id long
- drop cause id
- counter bytes sdwan dropped long
- sdwan sla-not-met
- sdwan preferred-color-not-met
- sdwan qos-queue-id
- drop cause name
- counter packets appqoe fec-d-pkts
- counter packets appqoe fec-r-pkts
- counter packets appqoe pkt-dup-d-pkts-orig
- counter packets appqoe pkt-dup-d-pkts-dup
- counter packets appqoe pkt-dup-r-pkts
- counter packets sdwan pkt-cxp-d-pkts
- counter bytes appqoe ssl-read
- counter bytes appqoe ssl-written
- counter bytes appqoe ssl-en-read
- counter bytes appqoe ssl-en-written
- counter bytes appqoe ssl-de-read
- counter bytes appqoe ssl-de-written
- appqoe ssl service type
- appqoe ssl traffic type
- appqoe ssl policy action

- ETAInitialDataPacketOld
- ETASequenceofPktLengthsandTimes
- wlan_id
- timestampAbsoluteMonitoring-intervalStart
- timestampAbsoluteMonitoring-intervalEnd

Juniper Networks (PEN: 2636)

- ifa_headers
- ifa_metadata
- ifa_sampled_packet
- Packet Loss Priority
- Forwarding Class Name

LANcope, now Cisco (PEN: 8712)

- FlowSensorInitiator
- FlowSensorTCPSYNACKTotalCount
- FlowSensorTCPSRSTotalCount
- FlowSensorRoundTripTime
- FlowSensorServerResponseTime
- FlowSensorRetransmits
- FlowSensorTCPBadTotalCount
- FlowSensorTCPFragTotalCount
- FlowSensorSourceEmailIn
- FlowSensorSourceEmailOut
- FlowSensorSourceEmailInMessages
- FlowSensorSourceEmailOutMessages
- FlowSensorSourceEmailInTrys

- FlowSensorSourceEmailOutTrys
- FlowSensorDestinationEmailIn
- FlowSensorDestinationEmailOut
- FlowSensorDestinationEmailInMessages
- FlowSensorDestinationEmailOutMessages
- FlowSensorDestinationEmailInTrys
- FlowSensorDestinationEmailOutTrys
- FlowSensorTraces
- FlowSensorEmbeddedICMPProtocol
- FlowSensorEmbeddedICMPType
- FlowSensorEmbeddedICMPCode
- FlowSensorApplicationIdentifier
- FlowSensorBadFlagXmas
- FlowSensorBadFlagSYNFIN
- FlowSensorBadFlagBadRST
- FlowSensorBadFlagNoACK
- FlowSensorBadFlagURG
- FlowSensorBadFlagNoFlag
- FlowSensorShortFragAttack
- FlowSensorFragPacketTooShort
- FlowSensorFragPacketTooLong
- FlowSensorFragPacketDifferentSizes
- FlowSensorApplicationDetails
- FlowSensorTrustsecSourceIdentifier
- EndpointFlowProcessAccount
- EndpointFlowProcessName

- EndpointFlowProcessHash
- EndpointFlowParentProcessAccount
- EndpointFlowParentProcessName
- EndpointFlowParentProcessHash

IPFIX Reverse Information Element Private Enterprise (PEN: 29305)

- octetDeltaCount
- packetDeltaCount
- deltaFlowCount
- protocolIdentifier
- ipClassOfService
- tcpControlBits
- sourceTransportPort
- sourceIPv4Address
- sourceIPv4PrefixLength
- ingressInterface
- destinationTransportPort
- destinationIPv4Address
- destinationIPv4PrefixLength
- egressInterface
- ipNextHopIPv4Address
- bgpSourceAsNumber
- bgpDestinationAsNumber
- bgpNextHopIPv4Address
- postMCastPacketDeltaCount
- postMCastOctetDeltaCount

- flowEndSysUpTime
- flowStartSysUpTime
- postOctetDeltaCount
- postPacketDeltaCount
- minimumIpTotalLength
- maximumIpTotalLength
- sourceIPv6Address
- destinationIPv6Address
- sourceIPv6PrefixLength
- destinationIPv6PrefixLength
- flowLabelIPv6
- icmpTypeCodeIPv4
- igmpType
- samplingInterval
- samplingAlgorithm
- flowActiveTimeout
- flowIdleTimeout
- engineType
- engineId
- ipv4RouterSc
- sourceIPv4Prefix
- destinationIPv4Prefix
- mplsTopLabelType
- mplsTopLabelIPv4Address
- samplerId
- samplerMode

- samplerRandomInterval
- classId
- minimumTTL
- maximumTTL
- fragmentIdentification
- postIpClassOfService
- sourceMacAddress
- postDestinationMacAddress
- vlanId
- postVlanId
- ipVersion
- flowDirection
- ipNextHopIPv6Address
- bgpNextHopIPv6Address
- ipv6ExtensionHeaders
- mplsTopLabelStackSection
- mplsLabelStackSection2
- mplsLabelStackSection3
- mplsLabelStackSection4
- mplsLabelStackSection5
- mplsLabelStackSection6
- mplsLabelStackSection7
- mplsLabelStackSection8
- mplsLabelStackSection9
- mplsLabelStackSection10
- destinationMacAddress

- postSourceMacAddress
- interfaceName
- interfaceDescription
- samplerName
- octetTotalCount
- packetTotalCount
- flagsAndSamplerId
- fragmentOffset
- forwardingStatus
- mplsVpnRouteDistinguisher
- mplsTopLabelPrefixLength
- srcTrafficIndex
- dstTrafficIndex
- applicationDescription
- applicationId
- applicationName
- postIpDiffServCodePoint
- multicastReplicationFactor
- className
- classificationEngineId
- layer2packetSectionOffset
- layer2packetSectionSize
- layer2packetSectionData
- bgpNextAdjacentAsNumber
- bgpPrevAdjacentAsNumber
- droppedOctetDeltaCount

- droppedPacketDeltaCount
- droppedOctetTotalCount
- droppedPacketTotalCount
- flowEndReason
- observationPointId
- icmpTypeCodeIPv6
- mplsTopLabelIPv6Address
- lineCardId
- portId
- meteringProcessId
- exportingProcessId
- wlanChannelId
- wlanSSID
- flowStartSeconds
- flowEndSeconds
- flowStartMilliseconds
- flowEndMilliseconds
- flowStartMicroseconds
- flowEndMicroseconds
- flowStartNanoseconds
- flowEndNanoseconds
- flowStartDeltaMicroseconds
- flowEndDeltaMicroseconds
- systemInitTimeMilliseconds
- flowDurationMilliseconds
- flowDurationMicroseconds

- destinationIPv6Prefix
- sourceIPv6Prefix
- postOctetTotalCount
- postPacketTotalCount
- postMCastPacketTotalCount
- postMCastOctetTotalCount
- icmpTypeIPv4
- icmpCodeIPv4
- icmpTypeIPv6
- icmpCodeIPv6
- udpSourcePort
- udpDestinationPort
- tcpSourcePort
- tcpDestinationPort
- tcpSequenceNumber
- tcpAcknowledgementNumber
- tcpWindowSize
- tcpUrgentPointer
- tcpHeaderLength
- ipHeaderLength
- totalLengthIPv4
- payloadLengthIPv6
- ipTTL
- nextHeaderIPv6
- mplsPayloadLength
- ipDiffServCodePoint

- ipPrecedence
- fragmentFlags
- octetDeltaSumOfSquares
- octetTotalSumOfSquares
- mplsTopLabelTTL
- mplsLabelStackLength
- mplsLabelStackDepth
- mplsTopLabelExp
- ipPayloadLength
- udpMessageLength
- isMulticast
- ipv4IHL
- ipv4Options
- tcpOptions
- tcpSynTotalCount
- tcpFinTotalCount
- tcpRstTotalCount
- tcpPshTotalCount
- tcpAckTotalCount
- tcpUrgTotalCount
- ipTotalLength
- postNATSourceIPv4Address
- postNATDestinationIPv4Address
- postNAPTSourceTransportPort
- postNAPTDestinationTransportPort
- natOriginatingAddressRealm

- natEvent
- initiatorOctets
- responderOctets
- firewallEvent
- ingressVRFID
- egressVRFID
- VRFname
- postMplsTopLabelExp
- tcpWindowScale
- ethernetHeaderLength
- ethernetPayloadLength
- ethernetTotalLength
- dot1qVlanId
- dot1qPriority
- dot1qCustomerVlanId
- dot1qCustomerPriority
- metroEvcId
- metroEvcType
- pseudoWireId
- pseudoWireType
- pseudoWireControlWord
- ingressPhysicalInterface
- egressPhysicalInterface
- postDot1qVlanId
- postDot1qCustomerVlanId
- ethernetType

- postIpPrecedence
- collectionTimeMilliseconds
- exportSctpStreamId
- maxExportSeconds
- maxFlowEndSeconds
- messageMD5Checksum
- messageScope
- minExportSeconds
- minFlowStartSeconds
- opaqueOctets
- sessionScope
- maxFlowEndMicroseconds
- maxFlowEndMilliseconds
- maxFlowEndNanoseconds
- minFlowStartMicroseconds
- minFlowStartMilliseconds
- minFlowStartNanoseconds
- collectorCertificate
- exporterCertificate
- dataRecordsReliability
- observationPointType
- newConnectionDeltaCount
- connectionSumDurationSeconds
- connectionTransactionId
- postNATSourceIPv6Address
- postNATDestinationIPv6Address

- natPoolId
- natPoolName
- anonymizationFlags
- anonymizationTechnique
- informationElementIndex
- p2pTechnology
- tunnelTechnology
- encryptedTechnology
- basicList
- subTemplateList
- subTemplateMultiList
- bgpValidityState
- IPSecSPI
- greKey
- natType
- initiatorPackets
- responderPackets
- observationDomainName
- selectionSequenceId
- selectorId
- informationElementId
- selectorAlgorithm
- samplingPacketInterval
- samplingPacketSpace
- samplingTimeInterval
- samplingTimeSpace

- samplingSize
- samplingPopulation
- samplingProbability
- dataLinkFrameSize
- ipHeaderPacketSection
- ipPayloadPacketSection
- dataLinkFrameSection
- mplsLabelStackSection
- mplsPayloadPacketSection
- selectorIdTotalPktsObserved
- selectorIdTotalPktsSelected
- absoluteError
- relativeError
- observationTimeSeconds
- observationTimeMilliseconds
- observationTimeMicroseconds
- observationTimeNanoseconds
- digestHashValue
- hashIPPayloadOffset
- hashIPPayloadSize
- hashOutputRangeMin
- hashOutputRangeMax
- hashSelectedRangeMin
- hashSelectedRangeMax
- hashDigestOutput
- hashInitialiserValue

- selectorName
- upperCILimit
- lowerCILimit
- confidenceLevel
- informationElementDataType
- informationElementDescription
- informationElementName
- informationElementRangeBegin
- informationElementRangeEnd
- informationElementSemantics
- informationElementUnits
- privateEnterpriseNumber
- virtualStationInterfaceId
- virtualStationInterfaceName
- virtualStationUUID
- virtualStationName
- layer2SegmentId
- layer2OctetDeltaCount
- layer2OctetTotalCount
- ingressUnicastPacketTotalCount
- ingressMulticastPacketTotalCount
- ingressBroadcastPacketTotalCount
- egressUnicastPacketTotalCount
- egressBroadcastPacketTotalCount
- monitoringIntervalStartMilliSeconds
- monitoringIntervalEndMilliSeconds

- portRangeStart
- portRangeEnd
- portRangeStepSize
- portRangeNumPorts
- staMacAddress
- staIPv4Address
- wtpMacAddress
- ingressInterfaceType
- egressInterfaceType
- rtpSequenceNumber
- userName
- applicationCategoryName
- applicationSubCategoryName
- applicationGroupName
- originalFlowsPresent
- originalFlowsInitiated
- originalFlowsCompleted
- distinctCountOfSourceIPAddress
- distinctCountOfDestinationIPAddress
- distinctCountOfSourceIPv4Address
- distinctCountOfDestinationIPv4Address
- distinctCountOfSourceIPv6Address
- distinctCountOfDestinationIPv6Address
- valueDistributionMethod
- rfc3550JitterMilliseconds
- rfc3550JitterMicroseconds

- rfc3550JitterNanoseconds
- dot1qDEI
- dot1qCustomerDEI
- flowSelectorAlgorithm
- flowSelectedOctetDeltaCount
- flowSelectedPacketDeltaCount
- flowSelectedFlowDeltaCount
- selectorIDTotalFlowsObserved
- selectorIDTotalFlowsSelected
- samplingFlowInterval
- samplingFlowSpacing
- flowSamplingTimeInterval
- flowSamplingTimeSpacing
- hashFlowDomain
- transportOctetDeltaCount
- transportPacketDeltaCount
- originalExporterIPv4Address
- originalExporterIPv6Address
- originalObservationDomainId
- intermediateProcessId
- ignoredDataRecordTotalCount
- dataLinkFrameType
- sectionOffset
- sectionExportedOctets
- dot1qServiceInstanceTag
- dot1qServiceInstanceId

- dot1qServiceInstancePriority
- dot1qCustomerSourceMacAddress
- dot1qCustomerDestinationMacAddress
- postLayer2OctetDeltaCount
- postMCastLayer2OctetDeltaCount
- layer2OctetTotalCount
- postLayer2OctetTotalCount
- postMCastLayer2OctetTotalCount
- minimumLayer2TotalLength
- maximumLayer2TotalLength
- droppedLayer2OctetDeltaCount
- droppedLayer2OctetTotalCount
- ignoredLayer2OctetTotalCount
- notSentLayer2OctetTotalCount
- layer2OctetDeltaSumOfSquares
- layer2OctetTotalSumOfSquares
- layer2FrameDeltaCount
- layer2FrameTotalCount
- pseudoWireDestinationIPv4Address
- ignoredLayer2FrameTotalCount
- mobileIMSI
- mobileMSISDN
- httpStatusCode
- sourceTransportPortsLimit
- httpRequestMethod
- httpRequestHost

- httpRequestTarget
- httpMessageVersion
- natInstanceID
- internalAddressRealm
- externalAddressRealm
- natQuotaExceededEvent
- natThresholdEvent
- httpUserAgent
- httpContentType
- httpReasonPhrase
- maxSessionEntries
- maxBIBEntries
- maxEntriesPerUser
- maxSubscribers
- maxFragmentsPendingReassembly
- addressPoolHighThreshold
- addressPoolLowThreshold
- addressPortMappingHighThreshold
- addressPortMappingLowThreshold
- addressPortMappingPerUserHighThreshold
- globalAddressMappingHighThreshold
- vpnIdentifier
- bgpCommunity
- bgpExtendedCommunity
- stand
- bgpLargeCommunity

RELATED DOCUMENTATION

NetFlow IEs | 1035

sFlow IEs (Flow Samples) | 1087

sFlow IEs (Counter Samples) | 1113

NetFlow IEs

IN THIS SECTION

The Apstra Flow collector supports Netflow v1, v5, v6, v7 and v9 flow records.

NetFlow v1

The collector supports the following Netflow v1 information elements (IEs):

- srcaddr
- dstaddr
- nexthop
- input
- output
- dPkts
- dOctets

- First
- Last
- srcport
- dstport
- prot
- tos
- tcp_flags

NetFlow v5

The collector supports the following Netflow v5 information elements (IE):

- srcaddr
- dstaddr
- nexthop
- input
- output
- dPkts
- dOctets
- First
- Last
- srcport
- dstport
- tcp_flags
- prot
- tos
- src_as
- dst_as

- src_mask
- dst_mask

NetFlow v6

The collector supports the following Netflow v6 information elements (IE):

- srcaddr
- dstaddr
- nexthop
- input
- output
- dPkts
- dOctets
- First
- Last
- srcport
- dstport
- tcp_flags
- prot
- tos
- src_as
- dst_as
- src_mask
- dst_mask

NetFlow v7

The collector supports the following Netflow v7 information elements (IE):

- srcaddr

- dstaddr
- nexthop
- input
- output
- dPkts
- dOctets
- First
- Last
- srcport
- dstport
- tcp_flags
- prot
- tos
- src_as
- dst_as
- src_mask
- dst_mask
- ipv4RouterSc

NetFlow v9

NetFlow v9 supports the following standards-based IEs (PEN: 0):

Table 32: NetFlow v9 Standards-based IEs (PEN: 0)

ID	Name
1	octetDeltaCount
2	packetDeltaCount

Table 32: NetFlow v9 Standards-based IEs (PEN: 0) *(Continued)*

ID	Name
3	deltaFlowCount
4	protocolIdentifier
5	ipClassOfService
6	tcpControlBits
7	sourceTransportPort
8	sourceIPv4Address
9	sourceIPv4PrefixLength
10	ingressInterface
11	destinationTransportPort
12	destinationIPv4Address
13	destinationIPv4PrefixLength
14	egressInterface
15	ipNextHopIPv4Address
16	bgpSourceAsNumber
17	bgpDestinationAsNumber
18	bgpNextHopIPv4Address
19	postMCastPacketDeltaCount
20	postMCastOctetDeltaCount
21	flowEndSysUpTime

Table 32: NetFlow v9 Standards-based IEs (PEN: 0) ***(Continued)***

ID	Name
22	flowStartSysUpTime
23	postOctetDeltaCount
24	postPacketDeltaCount
25	minimumIpTotalLength
26	maximumIpTotalLength
27	sourceIPv6Address
28	destinationIPv6Address
29	sourceIPv6PrefixLength
30	destinationIPv6PrefixLength
31	flowLabelIPv6
32	icmpTypeCodeIPv4
33	igmpType
34	samplingInterval
35	samplingAlgorithm
36	flowActiveTimeout
37	flowIdleTimeout
38	engineType
39	engineId
40	exportedOctetTotalCount

Table 32: NetFlow v9 Standards-based IEs (PEN: 0) *(Continued)*

ID	Name
41	exportedMessageTotalCount
42	exportedFlowRecordTotalCount
44	sourceIPv4Prefix
45	destinationIPv4Prefix
46	mplsTopLabelType
47	mplsTopLabelIPv4Address
48	samplerId
49	samplerMode
50	samplerRandomInterval
51	classId
52	minimumTTL
53	maximumTTL
54	fragmentIdentification
55	postIpClassOfService
56	sourceMacAddress
57	postDestinationMacAddress
58	vlanId
59	postVlanId
60	ipVersion

Table 32: NetFlow v9 Standards-based IEs (PEN: 0) *(Continued)*

ID	Name
61	flowDirection
62	ipNextHopIPv6Address
63	bgpNextHopIPv6Address
64	ipv6ExtensionHeaders
65	transportPacketLoss `Cisco Legacy`
66	transportUnreachability `Cisco Legacy`
67	transportLatency `Cisco Legacy`
68	dataPoints `Cisco Legacy`
69	variance `Cisco Legacy`
70	mplsTopLabelStackSection
71	mplsLabelStackSection2
72	mplsLabelStackSection3
73	mplsLabelStackSection4
74	mplsLabelStackSection5
75	mplsLabelStackSection6
76	mplsLabelStackSection7
77	mplsLabelStackSection8
78	mplsLabelStackSection9
79	mplsLabelStackSection10

Table 32: NetFlow v9 Standards-based IEs (PEN: 0) *(Continued)*

ID	Name
80	destinationMacAddress
81	postSourceMacAddress
82	interfaceName
83	interfaceDescription
84	samplerName
85	octetTotalCount
86	packetTotalCount
87	flagsAndSamplerId
88	fragmentOffset
89	forwardingStatus
90	mplsVpnRouteDistinguisher
91	mplsTopLabelPrefixLength
92	srcTrafficIndex
93	dstTrafficIndex
94	applicationDescription
95	applicationId
96	applicationName
97	subApplicationTag `Cisco Legacy`
98	postIpDiffServCodePoint

Table 32: NetFlow v9 Standards-based IEs (PEN: 0) *(Continued)*

ID	Name
99	multicastReplicationFactor
100	className
101	classificationEngineId
102	layer2packetSectionOffset
103	layer2packetSectionSize
104	layer2packetSectionData
105	applicationVersion `Cisco Legacy`
106	applicationVersionName `Cisco Legacy`
107	applicationVendor `Cisco Legacy`
109	subApplicationName `Cisco Legacy`
110	subApplicationDescription `Cisco Legacy`
111	templateParameterRangeEnd `Cisco Legacy`
128	bgpNextAdjacentAsNumber
129	bgpPrevAdjacentAsNumber
130	exporterIPv4Address
131	exporterIPv6Address
132	droppedOctetDeltaCount
133	droppedPacketDeltaCount
134	droppedOctetTotalCount

Table 32: NetFlow v9 Standards-based IEs (PEN: 0) *(Continued)*

ID	Name
135	droppedPacketTotalCount
136	flowEndReason
137	commonPropertiesId
138	observationPointId
139	icmpTypeCodeIPv6
140	mplsTopLabelIPv6Address
141	lineCardId
142	portId
143	meteringProcessId
144	exportingProcessId
145	templateId
146	wlanChannelId
147	wlanSSID
148	flowId
149	observationDomainId
150	flowStartSeconds
151	flowEndSeconds
152	flowStartMilliseconds
153	flowEndMilliseconds

Table 32: NetFlow v9 Standards-based IEs (PEN: 0) *(Continued)*

ID	Name
154	flowStartMicroseconds
155	flowEndMicroseconds
156	flowStartNanoseconds
157	flowEndNanoseconds
158	flowStartDeltaMicroseconds
159	flowEndDeltaMicroseconds
160	systemInitTimeMilliseconds
161	flowDurationMilliseconds
162	flowDurationMicroseconds
163	observedFlowTotalCount
164	ignoredPacketTotalCount
165	ignoredOctetTotalCount
166	notSentFlowTotalCount
167	notSentPacketTotalCount
168	notSentOctetTotalCount
169	destinationIPv6Prefix
170	sourceIPv6Prefix
171	postOctetTotalCount
172	postPacketTotalCount

Table 32: NetFlow v9 Standards-based IEs (PEN: 0) *(Continued)*

ID	Name
173	flowKeyIndicator
174	postMCastPacketTotalCount
175	postMCastOctetTotalCount
176	icmpTypeIPv4
177	icmpCodeIPv4
178	icmpTypeIPv6
179	icmpCodeIPv6
180	udpSourcePort
181	udpDestinationPort
182	tcpSourcePort
183	tcpDestinationPort
184	tcpSequenceNumber
185	tcpAcknowledgementNumber
186	tcpWindowSize
187	tcpUrgentPointer
188	tcpHeaderLength
189	ipHeaderLength
190	totalLengthIPv4
191	payloadLengthIPv6

Table 32: NetFlow v9 Standards-based IEs (PEN: 0) *(Continued)*

ID	Name
192	ipTTL
193	nextHeaderIPv6
194	mplsPayloadLength
195	ipDiffServCodePoint
196	ipPrecedence
197	fragmentFlags
198	octetDeltaSumOfSquares
199	octetTotalSumOfSquares
200	mplsTopLabelTTL
201	mplsLabelStackLength
202	mplsLabelStackDepth
203	mplsTopLabelExp
204	ipPayloadLength
205	udpMessageLength
206	isMulticast
207	ipv4IHL
208	ipv4Options
209	tcpOptions
210	paddingOctets

Table 32: NetFlow v9 Standards-based IEs (PEN: 0) *(Continued)*

ID	Name
211	collectorIPv4Address
212	collectorIPv6Address
213	exportInterface
214	exportProtocolVersion
215	exportTransportProtocol
216	collectorTransportPort
217	exporterTransportPort
218	tcpSynTotalCount
219	tcpFinTotalCount
220	tcpRstTotalCount
221	tcpPshTotalCount
222	tcpAckTotalCount
223	tcpUrgTotalCount
224	ipTotalLength
225	postNATSourceIPv4Address
226	postNATDestinationIPv4Address
227	postNAPTSourceTransportPort
228	postNAPTDestinationTransportPort
229	natOriginatingAddressRealm

Table 32: NetFlow v9 Standards-based IEs (PEN: 0) *(Continued)*

ID	Name
230	natEvent
231	initiatorOctets
232	responderOctets
233	firewallEvent
234	ingressVRFID
235	egressVRFID
236	VRFname
237	postMplsTopLabelExp
238	tcpWindowScale
239	biflowDirection
240	ethernetHeaderLength
241	ethernetPayloadLength
242	ethernetTotalLength
243	dot1qVlanId
244	dot1qPriority
245	dot1qCustomerVlanId
246	dot1qCustomerPriority
247	metroEvcId
248	metroEvcType

Table 32: NetFlow v9 Standards-based IEs (PEN: 0) *(Continued)*

ID	Name
249	pseudoWireId
250	pseudoWireType
251	pseudoWireControlWord
252	ingressPhysicalInterface
253	egressPhysicalInterface
254	postDot1qVlanId
255	postDot1qCustomerVlanId
256	ethernetType
257	postIpPrecedence
258	collectionTimeMilliseconds
259	exportSctpStreamId
260	maxExportSeconds
261	maxFlowEndSeconds
262	messageMD5Checksum
263	messageScope
264	minExportSeconds
265	minFlowStartSeconds
266	opaqueOctets
267	sessionScope

Table 32: NetFlow v9 Standards-based IEs (PEN: 0) *(Continued)*

ID	Name
268	maxFlowEndMicroseconds
269	maxFlowEndMilliseconds
270	maxFlowEndNanoseconds
271	minFlowStartMicroseconds
272	minFlowStartMilliseconds
273	minFlowStartNanoseconds
274	collectorCertificate
275	exporterCertificate
276	dataRecordsReliability
277	observationPointType
278	newConnectionDeltaCount
279	connectionSumDurationSeconds
280	connectionTransactionId
281	postNATSourceIPv6Address
282	postNATDestinationIPv6Address
283	natPoolId
284	natPoolName
285	anonymizationFlags
286	anonymizationTechnique

Table 32: NetFlow v9 Standards-based IEs (PEN: 0) *(Continued)*

ID	Name
287	informationElementIndex
288	p2pTechnology
289	tunnelTechnology
290	encryptedTechnology
294	bgpValidityState
295	IPSecSPI
296	greKey
297	natType
298	initiatorPackets
299	responderPackets
300	observationDomainName
301	selectionSequenceId
302	selectorId
303	informationElementId
304	selectorAlgorithm
305	samplingPacketInterval
306	samplingPacketSpace
307	samplingTimeInterval
308	samplingTimeSpace

Table 32: NetFlow v9 Standards-based IEs (PEN: 0) *(Continued)*

ID	Name
309	samplingSize
310	samplingPopulation
311	samplingProbability
312	dataLinkFrameSize
313	ipHeaderPacketSection
314	ipPayloadPacketSection
315	dataLinkFrameSection
316	mplsLabelStackSection
317	mplsPayloadPacketSection
318	selectorIdTotalPktsObserved
319	selectorIdTotalPktsSelected
320	absoluteError
321	relativeError
322	observationTimeSeconds
323	observationTimeMilliseconds
324	observationTimeMicroseconds
325	observationTimeNanoseconds
326	digestHashValue
327	hashIPPayloadOffset

Table 32: NetFlow v9 Standards-based IEs (PEN: 0) *(Continued)*

ID	Name
328	hashIPPayloadSize
329	hashOutputRangeMin
330	hashOutputRangeMax
331	hashSelectedRangeMin
332	hashSelectedRangeMax
333	hashDigestOutput
334	hashInitialiserValue
335	selectorName
336	upperCILimit
337	lowerCILimit
338	confidenceLevel
339	informationElementDataType
340	informationElementDescription
341	informationElementName
342	informationElementRangeBegin
343	informationElementRangeEnd
344	informationElementSemantics
345	informationElementUnits
346	privateEnterpriseNumber

Table 32: NetFlow v9 Standards-based IEs (PEN: 0) *(Continued)*

ID	Name
347	virtualStationInterfaceId
348	virtualStationInterfaceName
349	virtualStationUUID
350	virtualStationName
351	layer2SegmentId
352	layer2OctetDeltaCount
353	layer2OctetTotalCount
354	ingressUnicastPacketTotalCount
355	ingressMulticastPacketTotalCount
356	ingressBroadcastPacketTotalCount
357	egressUnicastPacketTotalCount
358	egressBroadcastPacketTotalCount
359	monitoringIntervalStartMilliSeconds
360	monitoringIntervalEndMilliSeconds
361	portRangeStart
362	portRangeEnd
363	portRangeStepSize
364	portRangeNumPorts
365	staMacAddress

Table 32: NetFlow v9 Standards-based IEs (PEN: 0) *(Continued)*

ID	Name
366	staIPv4Address
367	wtpMacAddress
368	ingressInterfaceType
369	egressInterfaceType
370	rtpSequenceNumber
371	userName
372	applicationCategoryName
373	applicationSubCategoryName
374	applicationGroupName
375	originalFlowsPresent
376	originalFlowsInitiated
377	originalFlowsCompleted
378	distinctCountOfSourceIPAddress
379	distinctCountOfDestinationIPAddress
380	distinctCountOfSourceIPv4Address
381	distinctCountOfDestinationIPv4Address
382	distinctCountOfSourceIPv6Address
383	distinctCountOfDestinationIPv6Address
384	valueDistributionMethod

Table 32: NetFlow v9 Standards-based IEs (PEN: 0) *(Continued)*

ID	Name
385	rfc3550JitterMilliseconds
386	rfc3550JitterMicroseconds
387	rfc3550JitterNanoseconds
388	dot1qDEI
389	dot1qCustomerDEI
390	flowSelectorAlgorithm
391	flowSelectedOctetDeltaCount
392	flowSelectedPacketDeltaCount
393	flowSelectedFlowDeltaCount
394	selectorIDTotalFlowsObserved
395	selectorIDTotalFlowsSelected
396	samplingFlowInterval
397	samplingFlowSpacing
398	flowSamplingTimeInterval
399	flowSamplingTimeSpacing
400	hashFlowDomain
401	transportOctetDeltaCount
402	transportPacketDeltaCount
403	originalExporterIPv4Address

Table 32: NetFlow v9 Standards-based IEs (PEN: 0) *(Continued)*

ID	Name
404	originalExporterIPv6Address
405	originalObservationDomainId
406	intermediateProcessId
407	ignoredDataRecordTotalCount
408	dataLinkFrameType
409	sectionOffset
410	sectionExportedOctets
411	dot1qServiceInstanceTag
412	dot1qServiceInstanceId
413	dot1qServiceInstancePriority
414	dot1qCustomerSourceMacAddress
415	dot1qCustomerDestinationMacAddress
416	layer2OctetDeltaCount
417	postLayer2OctetDeltaCount
418	postMCastLayer2OctetDeltaCount
419	layer2OctetTotalCount
420	postLayer2OctetTotalCount
421	postMCastLayer2OctetTotalCount
422	minimumLayer2TotalLength

Table 32: NetFlow v9 Standards-based IEs (PEN: 0) *(Continued)*

ID	Name
423	maximumLayer2TotalLength
424	droppedLayer2OctetDeltaCount
425	droppedLayer2OctetTotalCount
426	ignoredLayer2OctetTotalCount
427	notSentLayer2OctetTotalCount
428	layer2OctetDeltaSumOfSquares
429	layer2OctetTotalSumOfSquares
430	layer2FrameDeltaCount
431	layer2FrameTotalCount
432	pseudoWireDestinationIPv4Address
433	ignoredLayer2FrameTotalCount
434	mibObjectValueInteger
435	mibObjectValueOctetString
436	mibObjectValueOID
437	mibObjectValueBits
438	mibObjectValueIPAddress
439	mibObjectValueCounter
440	mibObjectValueGauge
441	mibObjectValueTimeTicks

Table 32: NetFlow v9 Standards-based IEs (PEN: 0) *(Continued)*

ID	Name
442	mibObjectValueUnsigned
445	mibObjectIdentifier
446	mibSubIdentifier
447	mibIndexIndicator
448	mibCaptureTimeSemantics
449	mibContextEngineID
450	mibContextName
451	mibObjectName
452	mibObjectDescription
453	mibObjectSyntax
454	mibModuleName
455	mobileIMSI
456	mobileMSISDN
457	httpStatusCode
458	sourceTransportPortsLimit
459	httpRequestMethod
460	httpRequestHost
461	httpRequestTarget
462	httpMessageVersion

Table 32: NetFlow v9 Standards-based IEs (PEN: 0) *(Continued)*

ID	Name
463	natInstanceID
464	internalAddressRealm
465	externalAddressRealm
466	natQuotaExceededEvent
467	natThresholdEvent
468	httpUserAgent
469	httpContentType
470	httpReasonPhrase
471	maxSessionEntries
472	maxBIBEntries
473	maxEntriesPerUser
474	maxSubscribers
475	maxFragmentsPendingReassembly
476	addressPoolHighThreshold
477	addressPoolLowThreshold
478	addressPortMappingHighThreshold
479	addressPortMappingLowThreshold
480	addressPortMappingPerUserHighThreshold
481	globalAddressMappingHighThreshold

Table 32: NetFlow v9 Standards-based IEs (PEN: 0) ***(Continued)***

ID	Name
482	vpnIdentifier
483	bgpCommunity
486	bgpExtendedCommunity
489	bgpLargeCommunity

Cisco (Pen: 9)

- scTag
- scTrafficProcessorId
- scSourceIpSample
- scDestinationIpSample
- scFlowContextId
- scSubscriberId
- scPackageId
- scServiceId
- scProtocolId
- scSkipppedSessions
- scInitiatingSide
- scReportTime
- scTransactionDurationMillisec
- scTimeFrame
- scSessionUpstreamVolume
- scSessionDownstreamVolume
- scProtocolSignature

- scZoneId
- scFlavorId
- scFlowCloseMode
- scAccessString
- scInfoString
- scClientPort
- scServerPort
- scSubscriberCounterId
- scServiceUsageCounterId
- scBreachState
- scReason
- scConfiguredDuration
- scDuration
- scEndTime
- scUpstreamVolume
- scDownstreamVolume
- scSessions
- scSeconds
- scPackageCounterId
- scGeneratorId
- scServiceGlobalCounterId
- scConcurrentSessions
- scActiveSubscribers
- scTotalActiveSubscribers
- scLinkId
- scAttackId

- scAttackIp
- scAttackOtherIp
- scAttackPortNumber
- scAttackType
- scAttackSide
- scAttackIpProtocol
- scAttacks
- scAttackMaliciousSessions
- INGRESS_ACL_ID
- EGRESS_ACL_ID
- FW_EXT_EVENT
- FW_EVENT_LEVEL
- FW_EVENT_LEVEL_ID
- FW_CONFIGURED_VALUE
- FW_ERM_EXT_EVENT
- FW_ERM_EXT_EVENT_DESC
- audio rtp packets lost
- audio rtp packets expected
- audio rtp fwd out-of-sequence sum
- audio rtp seconds ok
- audio rtp seconds concealed
- audio rtp seconds concealed severe
- audio rtp jitter ticks
- audio g107 impairment
- audio g107 lossRate
- audio g107 codec baseline

- audio g107 codec baseline bpl
- audio g107 impairment one-way-delay
- audio concealment ratio now
- audio concealment ratio minimum
- audio concealment ratio maximum
- audio concealment time
- audio speech time
- audio packets ok
- audio packets cs
- audio packets scs
- audio packets rtp
- audio packets silence
- audio duration receive
- audio duration receive voice
- audio duration early packet
- audio duration clock adjust
- audio duration playout increase
- audio duration playout decrease
- audio duration late discard
- audio frame size
- audio frames-per-packet
- audio frame arriving times difference
- audio frame arriving times difference vari
- audio noise level current
- audio noise level average
- audio noise level minimum

- audio noise level maximum
- audio noise level configured
- audio snr current
- audio snr average
- audio snr minimum
- audio snr maximum
- audio snr configured
- vxlan sgt
- vxlan flags
- vxlan vtep input
- vxlan vtep output
- SGT_SOURCE_TAG
- SGT_DESTINATION_TAG
- SGT_SOURCE_NAME
- SGT_DESTINATION_NAME
- flow cts switch derived-sgt
- FW_EXT_EVENT
- FW_BLACKOUT_SECS
- FW_HALFOPEN_HIGH
- FW_HALFOPEN_RATE
- FW_ZONEPAIR_ID
- FW_MAX_SESSIONS
- FW_ZONEPAIR_NAME
- FW_EXT_EVENT_DESC
- FW_SUMMARY_PKT_CNT
- FW_HALFOPEN_CNT

- waas dre input
- waas dre output
- waas lz input
- waas lz output
- waas original bytes
- waas optimised bytes
- waas application
- waas class
- waas connection mode
- waas bytes input
- waas bytes output
- PACKETS_DROPPED
- counter packets dropped permanent
- PACKET_RATE
- BYTE_RATE
- application media bytes counter
- application media bytes counter permanent
- application media bytes rate
- application media packets counter
- application media packets counter permanen
- application media packets rate
- application media packets rate variation
- application media event
- monitor event
- timestamp interval
- transport packets expected counter

- transport packets expected counter permane
- transport round-trip-time
- transport event packet-loss counter
- transport event packet-loss counter perman
- transport packets lost counter
- transport packets lost counter permanent
- transport packets lost rate
- transport rtp ssrc
- transport rtp jitter mean
- transport rtp jitter minimum
- transport rtp jitter maximum
- misc unsupported
- counter bytes rate per-flow
- counter bytes rate per-flow min
- counter bytes rate per-flow max
- counter packets rate per-flow
- counter packets rate per-flow min
- counter packets rate per-flow max
- application media bytes rate per-flow
- application media bytes rate per-flow min
- application media bytes rate per-flow max
- application media packets rate variation m
- application media packets rate variation m
- transport rtp flow count
- transport rtp payload-type
- transport packets lost counter min

- transport packets lost counter max
- transport event packet-loss counter min
- transport event packet-loss counter max
- transport packets lost rate min
- transport packets lost rate max
- transport tcp flow count
- transport round-trip-time sum
- transport round-trip-time samples
- transport round-trip-time min
- transport round-trip-time max
- metadata global-session-id
- metadata multi-party-session-id
- metadata clock-rate
- server response time average
- refused sessions
- client network delay average
- server network delay average
- network delay average
- application delay average
- session time minimum
- session time maximum
- session time average
- transaction time average
- closed sessions
- retransmitted packets
- transport bytes out-of-order

- client throughput average
- unresponsive sessions
- transport packets out-of-order
- IPv4 source observation node
- IPv4 destination observation node
- IPv6 source observation node
- IPv6 destination observation node
- pfr one-way-delay sum
- pfr one-way-delay samples
- ONE_WAY_DELAY {pfr one-way-delay
- packet arrival timestamp
- transport tcp window-size minimum
- transport tcp window-size maximum
- transport tcp window-size average
- transport tcp maximum-segment-size
- transport tcp window-size sum
- tcpWindowSizeSum
- transport rtp jitter mean sum
- application media packets rate variation s
- transport tcp window-size average sum
- transport rtp jitter inter arrival sum
- transport rtp jitter inter arrival samples
- transport rtp jitter inter arrival mean
- pfr site source id ipv4
- pfr site destination id ipv4
- transport bytes lost

- transport bytes expected
- transport bytes lost rate
- network delay sum
- network delay sample
- pfr counter event error traffic-class miti
- pfr counter event error traffic-class miti
- pfr counter event error traffic-class miti
- pfr site source prefix ipv4
- pfr site destination prefix ipv4
- pfr site source prefix ipv6
- pfr site destination prefix ipv6
- pfr site source prefix mask ipv4
- pfr site destination prefix mask ipv4
- pfr site source prefix mask ipv6
- pfr site destination prefix mask ipv6
- pfr service provider tag identifier
- pfr label identifier
- application voice number called
- application voice number calling
- application voice setup time
- application voice call duration
- application voice rx bad-packet
- application voice rx out-of-sequence
- application voice codec id
- application voice play delay current
- application voice play delay minimum

- application voice play delay maximum
- application voice sip call-id
- application voice router global-call-id
- application voice delay round-trip
- application voice delay end-point
- application voice r-factor 1
- application voice r-factor 2
- application voice mos conversation
- application voice mos listening
- application voice concealment-ratio averag
- application voice jitter configured type
- application voice jitter configured minimu
- application voice jitter configured maximu
- application voice jitter configured initia
- application voice rx early-packet count
- application voice rx late-packet count
- application voice jitter buffer-overrun
- application voice packet conceal-count
- bandwidth used
- bandwidth used percentage
- application video resolution width last
- application video resolution height last
- application video frame rate
- application video payload bitrate average
- application video payload bitrate fluctuat
- application video frame I counter frames

- application video frame I counter packets
- application video frame I counter bytes
- application video frame STR counter frames
- application video frame STR counter packet
- application video frame STR counter bytes
- application video frame LTR counter frames
- application video frame LTR counter packet
- application video frame LTR counter bytes
- application video frame super-P counter fr
- application video frame super-P counter pa
- application video frame super-P counter by
- application video frame NR counter frames
- application video frame NR counter packets
- application video frame NR counter bytes
- application video frame I slice-quantizati
- application video frame STR slice-quantiza
- application video frame LTR slice-quantiza
- application video frame super-P slice-quan
- application video frame NR slice-quantizat
- application video eMOS compression bitstre
- application video eMOS compression network
- application video frame I counter packets
- application video frame STR counter packet
- application video frame LTR counter packet
- application video frame super-P counter pa
- application video frame NR counter packets

- application video frame percentage damaged
- application video eMOS packet-loss bitstre
- application video eMOS packet-loss network
- application video scene-complexity
- application video level-of-motion
- transport rtp sequence-number
- transport rtp sequence-number last
- iOAM my node-id
- iOAM my node name
- start timestamp
- end timestamp
- IOAM packet counter
- IOAM byte count
- IOAM cs0 packet counter
- IOAM cs0 byte count
- IOAM cs1 packet counter
- IOAM cs1 byte count
- IOAM cs2 packet counter
- IOAM cs2 byte count
- IOAM cs3 packet counter
- IOAM cs3 byte count
- IOAM cs4 packet counter
- IOAM cs4 byte count
- IOAM cs5 packet counter
- IOAM cs5 byte count
- IOAM cs6 packet counter

- IOAM cs6 byte count
- IOAM cs7 packet counter
- IOAM cs7 byte count
- IOAM lost packet counter
- IOAM duplicate packet counter
- IOAM reordered packet counter
- IOAM highest PPC sequence number
- iOAM node-id
- ipv6 protocol filed
- iOAM E2E Header
- iOAM Path Map
- iOAM number of nodes
- iOAM node1 id
- iOAM node1 in if id
- iOAM node1 eif id
- iOAM node2 id
- iOAM node2 in if id
- iOAM node2 eif id
- iOAM node3 id
- iOAM node3 in if id
- iOAM node3 eif id
- iOAM node4 id
- iOAM node4 in if id
- iOAM node4 eif id
- iOAM Application metadata
- iOAM sfc-id

- iOAM sfc validated count
- iOAM sfc invalidated count
- pfr br ipv4 address
- pfr status
- reason id
- threshold
- pfr priority
- long-term round-trip-time
- mos below
- rsvp bw pool
- flow left time
- bw percentage
- bw fee
- transport source-port min
- transport source-port max
- transport destination-port min
- transport destination-port max
- capacity
- ingress bw
- max ingress bw
- egress bw
- max egress bw
- ingress rollup bw
- egress rollup bw
- kth rollup bw
- link group name

- bgp community
- bgp prepend
- entrance downgrade
- discard rollup count
- services pfr class-tag-id
- services pfr mc-id
- sip header from uri host ip addr
- sip header from uri userinfo user
- sip header to uri host ip addr
- sip header to uri userinfo user
- sip sess duration
- sip sess end_reason
- sip sess_dialed
- sip sess_connected
- sip sess_failed
- AAA_USERNAME
- XLATE_SRC_ADDR_IPV4
- XLATE_DST_ADDR_IPV4
- XLATE_SRC_PORT
- XLATE_DST_PORT
- FW_EVENT
- artClientNetworkTimeLongLivedMaximum
- artClientNetworkTimeLongLivedMinimum
- artServerNetworkTimeLongLivedMaximum
- artServerNetworkTimeLongLivedMinimum
- policy qos classification hierarchy

- c3pl class cce-id
- c3pl class name
- c3pl class type
- c3pl policy cce-id
- c3pl policy name
- c3pl policy type
- interface input fex-node-id
- interface output fex-node-id
- interface power
- monitor device-type
- connection server counter bytes network
- connection client counter bytes network
- wireless afd drop packets
- wireless afd accept packets
- wireless afd drop bytes
- wireless afd accept bytes
- connection concurrent-connections
- application transaction counter new
- services waas segment
- services waas passthrough-reason
- connection delay network long-lived to-ser
- connection delay network long-lived to-cli
- connection delay network long-lived client
- connection delay network client-to-server
- connection delay network to-server num-sam
- connection delay network to-client num-sam

- artClientpackets
- artServerpackets
- connection client counter bytes retransmit
- connection client counter packets retransm
- connection server counter bytes retransmit
- connection server counter packets retransm
- connection transaction counter complete
- connection transaction duration sum
- connection transaction duration max
- connection transaction duration min
- art count new connections
- art count responses
- art count responses histogram bucket1
- art count responses histogram bucket2
- art count responses histogram bucket3
- art count responses histogram bucket4
- art count responses histogram bucket5
- art count responses histogram bucket6
- art count responses histogram bucket7
- connection delay response to-server histog
- connection delay response to-server sum
- connection delay response to-server max
- connection delay response to-server min
- connection delay application sum
- connection delay application max
- connection delay application min

- connection delay response client-to-server
- connection delay response client-to-server
- connection delay response client-to-server
- connection delay network client-to-server
- connection delay network client-to-server
- connection delay network client-to-server
- onnection delay network to-client sum
- connection delay network to-client max
- connection delay network to-client min
- connection delay network to-server sum
- connection delay network to-server max
- connection delay network to-server min
- mos worst 100
- mos quality
- mos total count
- application http uri statistics
- policy qos queue index
- policy qos queue drops
- datalink event
- datalink event extended
- l4r server ipv4 address
- l4r server transport port
- l4r server ipv6 address
- l4r event
- l4r event timestamp
- pbhk mapped ipv4 address

- pbhk mapped transport port
- pbhk event
- pbhk event timestamp
- ETTA_INITIAL_DATA_PACKET
- ETTA_SEQUENCE_OF_PACKET_LENGTHS_AND_TIMES
- ETTA_SEQUENCE_OF_APPLICATION_LENGTHS_AND_TIMES
- ETAByteDistribution
- ETTA_TLS_RECORDS
- ETTA_TLS_CIPHER_SUITES
- ETTA_TLS_EXTENSIONS
- ETTA_TLS_VERSION
- ETTA_TLS_KEY_LENGTH
- ETTA_TLS_SESSION_ID
- ETTA_TLS_RANDOM
- ETTA_TLS_EXTENSION_LENGTHS
- ETTA_TLS_EXTENSION_TYPES
- application family name
- application set name
- application category name
- application sub category name
- application group name
- AVCSubApplicationValue
- connection client ipv4 address
- connection server ipv4 address
- connection client ipv6 address
- connection server ipv6 address

- connection client transport port
- connection server transport port
- connection id
- application traffic-class
- application business-relevance
- nvzFlowUDID
- nvzFlowLoggedInUser
- nvzFlowOSName
- nvzFlowOSVersion
- nvzFlowSystemManufacturer
- nvzFlowSystemType
- nvzFlowProcessAccount
- nvzFlowParentProcessAccount
- nvzFlowProcessName
- nvzFlowProcessHash
- nvzFlowParentProcessName
- nvzFlowParentProcessHash
- nvzFlowDNSSuffix
- nvzFlowDestinationHostname
- nvzFlowL4ByteCountIn
- nvzFlowL4ByteCountOut
- nvzFlowOSEdition
- nvzFlowModuleNameList
- nvzFlowModuleHashList
- nvzFlowCoordinatesList
- nvzFlowInterfaceInfoUID

- nvzFlowInterfaceIndex
- nvzFlowInterfaceType
- nvzFlowInterfaceName
- nvzFlowInterfaceDetailsList
- nvzFlowInterfaceMac
- nvzFlowUserAccountType
- nvzFlowProcessAccountType
- nvzFlowParentProcessAccountType
- overlay session id input
- overlay session id output
- routing vrf service
- tloc table overlay session id
- tloc local system ip address
- tloc local color
- tloc remote system ip address
- tloc remote color
- tloc tunnel protocol
- connection id long
- drop cause id
- counter bytes sdwan dropped long
- sdwan sla-not-met
- sdwan preferred-color-not-met
- sdwan qos-queue-id
- drop cause name
- counter packets appqoe fec-d-pkts
- counter packets appqoe fec-r-pkts

- counter packets appqoe pkt-dup-d-pkts-orig
- counter packets appqoe pkt-dup-d-pkts-dup
- counter packets appqoe pkt-dup-r-pkts
- counter packets sdwan pkt-cxp-d-pkts
- counter bytes appqoe ssl-read
- counter bytes appqoe ssl-written
- counter bytes appqoe ssl-en-read
- counter bytes appqoe ssl-en-written
- counter bytes appqoe ssl-de-read
- counter bytes appqoe ssl-de-written
- appqoe ssl service type
- appqoe ssl traffic type
- appqoe ssl policy action
- ETAInitialDataPacketOld
- ETASequenceofPktLengthsandTimes
- wlan_id
- timestampAbsoluteMonitoring-intervalStart
- timestampAbsoluteMonitoring-intervalEnd

LANcope, now Cisco (PEN: 8712)

- FlowSensorInitiator
- FlowSensorTCPSYNACKTotalCount
- FlowSensorTCPSRSTotalCount
- FlowSensorRoundTripTime
- FlowSensorServerResponseTime
- FlowSensorRetransmits

- FlowSensorTCPBadTotalCount
- FlowSensorTCPFragTotalCount
- FlowSensorSourceEmailIn
- FlowSensorSourceEmailOut
- FlowSensorSourceEmailInMessages
- FlowSensorSourceEmailOutMessages
- FlowSensorSourceEmailInTrys
- FlowSensorSourceEmailOutTrys
- FlowSensorDestinationEmailIn
- FlowSensorDestinationEmailOut
- FlowSensorDestinationEmailInMessages
- FlowSensorDestinationEmailOutMessages
- FlowSensorDestinationEmailInTrys
- FlowSensorDestinationEmailOutTrys
- FlowSensorTraces
- FlowSensorEmbeddedICMPProtocol
- FlowSensorEmbeddedICMPType
- FlowSensorEmbeddedICMPCode
- FlowSensorApplicationIdentifier
- FlowSensorBadFlagXmas
- FlowSensorBadFlagSYNFIN
- FlowSensorBadFlagBadRST
- FlowSensorBadFlagNoACK
- FlowSensorBadFlagURG
- FlowSensorBadFlagNoFlag
- FlowSensorShortFragAttack

- FlowSensorFragPacketTooShort
- FlowSensorFragPacketTooLong
- FlowSensorFragPacketDifferentSizes
- FlowSensorApplicationDetails
- FlowSensorTrustsecSourceIdentifier
- EndpointFlowProcessAccount
- EndpointFlowProcessName
- EndpointFlowProcessHash
- EndpointFlowParentProcessAccount
- EndpointFlowParentProcessName
- EndpointFlowParentProcessHash

RELATED DOCUMENTATION

IPFIX IEs | **970**

sFlow IEs (Flow Samples) | **1087**

sFlow IEs (Counter Samples) | **1113**

sFlow IEs (Flow Samples)

IN THIS SECTION

- extended_mpls_FTN (enterprise = 0, format = 1010) | **1101**
- extended_mpls_LDP_FEC (enterprise = 0, format = 1011) | **1101**
- extended_vlantunnel (enterprise = 0, format = 1012) | **1101**
- extended_80211_payload (enterprise = 0, format = 1013) | **1101**
- extended_80211_rx (enterprise = 0, format = 1014) | **1102**
- extended_80211_tx (enterprise = 0, format = 1015) | **1102**
- extended_openflow_v1 (enterprise = 0, format = 1017) | **1102**
- extended_fc (enterprise = 0, format = 1018) | **1103**
- extended_queue_length (enterprise = 0, format = 1019) | **1103**
- extended_nat_port (enterprise = 0, format = 1020 | **1103**
- extended_L2_tunnel_egress (enterprise = 0, format = 1021 | **1103**
- extended_L2_tunnel_ingresss (enterprise = 0, format = 1022 | **1103**
- extended_ipv4_tunnel_egress (enterprise = 0, format = 1023) | **1103**
- extended_ipv4_tunnel_ingress (enterprise = 0, format = 1024) | **1104**
- extended_ipv6_tunnel_egress (enterprise = 0, format = 1025) | **1104**
- extended_ipv6_tunnel_ingress (enterprise = 0, format = 1026) | **1104**
- extended_decapsulate_egress (enterprise = 0, format = 1027) | **1105**
- extended_decapsulate_ingress (enterprise = 0, format = 1028) | **1105**
- extended_vni_egress (enterprise = 0, format = 1029) | **1105**
- extended_vni_ingress (enterprise = 0, format = 1030) | **1105**
- extended_ib_lrh enterprise = 0, format = 1031) | **1105**
- extended_ib_grh (enterprise = 0, format = 1032) | **1106**
- extended_ib_brh (enterprise = 0, format = 1033) | **1106**
- extended_vlanin (enterprise = 0, format = 1034) | **1106**
- extended_vlanout (enterprise = 0, format = 1035) | **1106**
- extended_egress_queue (enterprise = 0, format = 1036) | **1106**
- extended_acl (enterprise = 0, format = 1037) | **1106**
- extended_function (enterprise = 0, format = 1038) | **1107**
- extended_transit (enterprise = 0, format = 1039) | **1107**
- extended_queue (enterprise = 0, format = 1040) | **1107**
- transaction (enterprise = 0, format = 2000) | **1107**
- extended_nfs_storage_transaction (enterprise = 0, format = 2001) | **1107**

The Apstra Flow collector supports the following sFlow IEs (information elements):

flow_sample

- sample_sequence
- source_id_type
- source_id
- sampling_rate
- sample_pool
- drops
- interface_format_input

- interface_input
- interface_format_output
- interface_output

sampled_header (enterprise = 0, format = 1)

- header_protocol
- frame_length
- stripped
- length
- header

Ethernet

- destination_mac_addr
- source_mac_addr

VLAN C-Tag "inner tag"

- c_vlan_pcp
- c_vlan_dei
- c_vlan_id

VLAN S-Tag "outer tag"

- s_vlan_pcp
- s_vlan_dei
- s_vlan_id

Internet Protocol version 4 (IPv4) (ether_type: 0x0800)

- version
- dscp

- ecn
- total_length
- flags
- ttl
- protocol
- source_ipaddr
- destination_ipaddr
- options

Internet Protocol version 6 (IPv6) (ether_type: 0x86dd)

- version
- dscp
- ecn
- flow_label
- payload_length
- next_header
- hop_limit
- source_ipaddr
- destination_ipaddr
- ext_route_type
- ext_route_hops

Point-to-Point Protocol over Ethernet (PPPoE) Discovery (ether_type: 0x8863)

- code
- session_id
- ddl_proto
- type

- version

Point-to-Point Protocol over Ethernet (PPPoE) Session (ether_type: 0x8864)

- code
- session_id
- ddl_proto
- type
- version

Transmission Control Protocol (TCP)

- header_size
- source port
- destination port
- flags
- seq
- ack
- window
- urgent_pointer
- options

User Datagram Protocol (UDP)

- source_port
- destination_port
- pdu_length

Authentication Header (AH)

- icv
- spi

Address Resolution Protocol (ARP)

- dst_hw_addr
- dst_proto_addr
- hw_type
- op_code
- proto_type
- src_hw_addr
- src_proto_addr

Border Gateway Protocol (BGP)

- error_code
- error_subcode
- hold_time
- msg_type
- route_afi
- route_safi
- router_as
- router_ip
- version

Encapsulating Security Payload (ESP)

- spi

Generic Routing Encapsulation (GRE)

- flow_id
- key
- pptp_call_id

- pptp_payload_size
- version
- vsid

Internet Control Message Protocol (ICMP)

- v4_code
- v4_type
- v6_code
- v6_type
- conv_error_pointer
- dns_names
- dns_ttl
- echo_ext_req_flags
- echo_ext_resp_flags
- echo_ext_state
- id
- mobile_subtype
- param_error_pointer
- photuris_pointer
- redirect_next_hop
- router_advert_addrs
- router_advert_size
- router_advert_lifetime
- seq_num
- subnet_mask
- timestamp_origin

- timestamp_rx
- timestamp_tx
- traceroute_id
- traceroute_hops_out
- traceroute_hops_in
- traceroute_bandwidth_out
- traceroute_mtu_out

Inband Flow Analyzer (IFA)

- flags
- gns
- metadata_action
- metadata_frag_id
- metadata_frag_last
- metadata_frag_packet_id
- metadata_size_max
- metadata_req
- metadata_size
- metadata_ttl

Internet Group Management Protocol (IGMP)

- group
- max_resp_time
- type

Logical Link Control (LLC)

- dsap

- dsap_u
- dsap_ig
- lpdu_frame_type
- lpdu_info_seq_curr
- lpdu_info_seq_next
- lpdu_pf
- lpdu_super_frame_type
- lpdu_super_seq_next
- lpdu_unnum_frame_type
- ssap
- ssap_u
- ssap_cr

Multi-Protocol Label Switching (MPLS)

- mpls_label
- mpls_tc
- mpls_ttl

Network Service Header (NSH)

- flag_oam_bit
- metadata_type
- sfp_si
- sfp_spi
- ttl
- version
- opt_metadata_class
- opt_metadata_type

- opt_payload

Open Shortest Path First (OSPF)

- version
- router_ip
- area
- auth_code
- msg_type
- inst_id

Point-to-Point Protocol (PPP)

- addr
- cntrl_value
- dll_proto

Pseudowire

- pwe3_seq

Subnetwork Access Protocol (SNAP)

- oui

Virtual Extensible LAN (VXLAN)

- flags
- vni

sampled_ethernet (enterprise = 0, format = 2)

- source_mac_addr
- destination_mac_addr

- eth_type

sampled_ipv4 (enterprise = 0, format = 3)

- ip_protocol
- src_ip
- dst_ip
- src_port
- dst_port
- tcp_flags
- tos

sampled_ipv6 (enterprise = 0, format = 4)

- ip_protocol
- src_ip
- dst_ip
- src_port
- dst_port
- tcp_flags
- ip_priority

extended_switch (enterprise = 0, format = 1001)

- src_vlan
- src_priority
- dst_vlan
- dst_priority

extended_router (enterprise = 0, format = 1002)

- next_hop

- src_mask_len
- dst_mask_len

extended_gateway (enterprise = 0, format = 1003)

- next_hop
- router_as
- source_as
- source_peer_as
- destination_count
- destinations
- community_count
- communities
- localpref

extended_user (enterprise = 0, format = 1004)

- src_charset
- src_user
- dst_charset
- dst_user

extended_url (enterprise = 0, format = 1005)

- direction
- url
- host

extended_mpls (enterprise = 0, format = 1006

- nexthop
- in_stack

- out_stack

extended_nat (enterprise = 0, format = 1007)

- src_address
- dst_address

extended_mpls_tunnel (enterprise = 0, format = 1008)

- tunnel_lsp_name
- tunnel_id
- tunnel_cos

extended_mpls_vc (enterprise = 0, format = 1009)

- vc_instance_name
- vll_vc_id
- vc_label_cos

extended_mpls_FTN (enterprise = 0, format = 1010)

- mplsFTNDescr
- mplsFTNMask

extended_mpls_LDP_FEC (enterprise = 0, format = 1011)

- mplsFecAddrPrefixLength

extended_vlantunnel (enterprise = 0, format = 1012)

- stack

extended_80211_payload (enterprise = 0, format = 1013)

- ciphersuite
- data

extended_80211_rx (enterprise = 0, format = 1014)

- ssid
- bssid
- version
- channel
- speed
- rsni
- rcpi
- packet_duration

extended_80211_tx (enterprise = 0, format = 1015)

- ssid
- bssid
- version
- transmissions
- packet_duration
- retrans_duration
- channel
- speed
- power

extended_openflow_v1 (enterprise = 0, format = 1017)

- flow_cookie
- flow_match
- flow_actions

extended_fc (enterprise = 0, format = 1018)

- t11FcRouteSrcMask
- t11FcRouteDestMask
- next_hop_t11FcRouteDomainId
- t11FcRouteType
- t11FcRouteProto
- t11FcRouteMetric

extended_queue_length (enterprise = 0, format = 1019)

- queueIndex
- queueLength

extended_nat_port (enterprise = 0, format = 1020

- src_port
- dst_port

extended_L2_tunnel_egress (enterprise = 0, format = 1021

- source_mac_addr
- destination_mac_addr
- eth_type

extended_L2_tunnel_ingresss (enterprise = 0, format = 1022

- source_mac_addr
- destination_mac_addr
- eth_type

extended_ipv4_tunnel_egress (enterprise = 0, format = 1023)

- ip_protocol

- src_ip
- dst_ip
- src_port
- dst_port
- tcp_flags
- tos

extended_ipv4_tunnel_ingress (enterprise = 0, format = 1024)

- ip_protocol
- src_ip
- dst_ip
- src_port
- dst_port
- tcp_flags
- tos

extended_ipv6_tunnel_egress (enterprise = 0, format = 1025)

- ip_protocol
- src_ip
- dst_ip
- src_port
- dst_port
- tcp_flags
- ip_priority

extended_ipv6_tunnel_ingress (enterprise = 0, format = 1026)

- ip_protocol

- src_ip
- dst_ip
- src_port
- dst_port
- tcp_flags
- ip_priority

extended_decapsulate_egress (enterprise = 0, format = 1027)

- inner_header_offset

extended_decapsulate_ingress (enterprise = 0, format = 1028)

- inner_header_offset

extended_vni_egress (enterprise = 0, format = 1029)

- vni

extended_vni_ingress (enterprise = 0, format = 1030)

- vni

extended_ib_lrh enterprise = 0, format = 1031)

- src_vl
- src_sl
- src_dlid
- src_slid
- src_lnh
- dst_vl
- dst_sl
- dst_dlid

- dst_slid
- dest_lnh

extended_ib_grh (enterprise = 0, format = 1032)

- flow_label
- tc
- s_gid
- d_gid

extended_ib_brh (enterprise = 0, format = 1033)

- pky
- dst_qp

extended_vlanin (enterprise = 0, format = 1034)

- tpid
- tci

extended_vlanout (enterprise = 0, format = 1035)

- tpid
- tci

extended_egress_queue (enterprise = 0, format = 1036)

- queue

extended_acl (enterprise = 0, format = 1037)

- acl_number
- acl_name
- direction

extended_function (enterprise = 0, format = 1038)

- symbol

extended_transit (enterprise = 0, format = 1039)

- delay

extended_queue (enterprise = 0, format = 1040)

- depth

transaction (enterprise = 0, format = 2000)

- direction
- wait
- duration
- status
- bytes_received
- bytes_send

extended_nfs_storage_transaction (enterprise = 0, format = 2001)

- path
- operation
- status

extended_scsi_storage_transaction (enterprise = 0, format = 2002)

- lun
- operation
- status

extended_http_transaction (enterprise = 0, format = 2003)

- url
- host
- referer
- useragent
- user
- status

extended_socket_ipv4 (enterprise = 0, format = 2100)

- protocol
- local_ip
- remote_ip
- local_port
- remote_port

extended_socket_ipv6 (enterprise = 0, format = 2101)

- protocol
- local_ip
- remote_ip
- local_port
- remote_port

extended_proxy_socket_ipv4 (enterprise = 0, format = 2102)

- extended_socket_ipv4

extended_proxy_socket_ipv6 (enterprise = 0, format = 2103)

- extended_socket_ipv6

memcached_operation (enterprise = 0, format = 2200)

- protocol
- cmd
- key
- nkeys
- value_bytes
- uS
- status

http_request (enterprise = 0, format = 2201)

- method
- uri
- host
- referer
- useragent
- authuser
- bytes
- duration
- status

app_operation (enterprise = 0, format = 2202)

- application
- operation
- attributes
- status_descr
- req_bytes

- resp_bytes
- duration
- status

app_parent_context (enterprise = 0, format = 2203)

- application
- operation
- attributes

app_initiator (enterprise = 0, format = 2204)

- actor

app_target (enterprise = 0, format = 2205)

- actor

http_request (enterprise = 0, format = 2206)

- method
- protocol
- uri
- host
- referer
- useragent
- xff
- authuser
- mime-type
- req_bytes
- resp_bytes
- duration

- status

extended_proxy_request (enterprise = 0, format = 2207)

- uri

extended_nav_timing (enterprise = 0, format = 2208)

- type
- redirectCount
- navigationStart
- unloadEventStart
- unloadEventEnd
- redirectStart
- redirectEnd
- fetchStart
- domainLookupStart
- domainLookupEnd
- connectStart
- connectEnd
- secureConnectionStart
- requestStart
- responseStart
- responseEnd
- domLoading
- domInteractive
- domContentLoadedEventStart
- domContentLoadedEventEnd
- domComplete

- loadEventStart
- loadEventEnd

extended_tcp_info (enterprise = 0, format = 2209)

- direction
- snd_mss
- rcv_mss
- unacked
- lost
- retrans
- pmtu
- rtt
- rttvar
- snd_cwnd
- reordering
- min_rtt

extended_entities (enterprise = 0, format = 2210)

- src_ds
- dst_ds

bst_egress_queue (enterprise = 4413, format = 1)

- queue

RELATED DOCUMENTATION

IPFIX IEs | **970**

NetFlow IEs | **1035**

sFlow IEs (Counter Samples) | **1113**

sFlow IEs (Counter Samples)

IN THIS SECTION

The Apstra Flow collector supports the following sFlow IEs (information elements):

if_counters (enterprise = 0, format = 1)

- ifIndex
- ifType
- ifSpeed
- ifDirection
- ifStatus
- ifInOctets

- ifInUcastPkts
- ifInMulticastPkts
- ifInBroadcastPkts
- ifInDiscards
- ifInErrors
- ifInUnknownProtos
- ifOutOctets
- ifOutUcastPkts
- ifOutMulticastPkts
- ifOutBroadcastPkts
- ifOutDiscards
- ifOutErrors
- ifPromiscuousMode

ethernet_counters (enterprise = 0, format =2)

- dot3StatsAlignmentErrors
- dot3StatsFCSErrors
- dot3StatsSingleCollisionFrames
- dot3StatsMultipleCollisionFrames
- dot3StatsSQETestErrors
- dot3StatsDeferredTransmissions
- dot3StatsLateCollisions
- dot3StatsExcessiveCollisions
- dot3StatsInternalMacTransmitErrors
- dot3StatsCarrierSenseErrors
- dot3StatsFrameTooLongs

- dot3StatsInternalMacReceiveErrors
- dot3StatsSymbolErrors

tokenring_counters (enterprise = 0, format = 3)

- dot5StatsLineErrors
- dot5StatsBurstErrors
- dot5StatsACErrors
- dot5StatsAbortTransErrors
- dot5StatsInternalErrors
- dot5StatsLostFrameErrors
- dot5StatsReceiveCongestions
- dot5StatsFrameCopiedErrors
- dot5StatsTokenErrors
- dot5StatsSoftErrors
- dot5StatsHardErrors
- dot5StatsSignalLoss
- dot5StatsTransmitBeacons
- dot5StatsRecoverys
- dot5StatsLobeWires
- dot5StatsRemoves
- dot5StatsSingles
- dot5StatsFreqErrors

vg_counters (enterprise = 0, format = 4)

- dot12InHighPriorityFrames
- dot12InHighPriorityOctets
- dot12InNormPriorityFrames

- dot12InNormPriorityOctets
- dot12InIPMErrors
- dot12InOversizeFrameErrors
- dot12InDataErrors
- dot12InNullAddressedFrames
- dot12OutHighPriorityFrames
- dot12OutHighPriorityOctets
- dot12TransitionIntoTrainings
- dot12HCInHighPriorityOctets
- dot12HCInNormPriorityOctets
- dot12HCOutHighPriorityOctets

vlan_counters (enterprise = 0, format = 5)

- vlan_id
- octets
- ucastPkts
- multicastPkts
- broadcastPkts
- discards

ieee80211_counters (enterprise = 0, format = 6)

- dot11TransmittedFragmentCount
- dot11GroupTransmittedFrameCount
- dot11FailedCount
- dot11RetryCount
- dot11MultipleRetryCount
- dot11FrameDuplicateCount

- dot11RTSSuccessCount
- dot11RTSFailureCount
- dot11AckFailureCount
- dot11ReceivedFragmentCount
- dot11GroupReceivedFrameCount
- dot11FCSErrorCount
- dot11TransmittedFrameCount
- dot11WEPUndecryptableCount
- dot11QosDiscardedFragmentCount
- dot11AssociatedStationCount
- dot11QosCFPollsReceivedCount
- dot11QosCFPollsUnusedCount
- dot11QosCFPollsUnusableCount
- dot11QosCFPollsLostCount

lag_port_stats (enterprise = 0, format = 7)

- dot3adAggPortActorSystemID
- dot3adAggPortPartnerOperSystemID
- dot3adAggPortAttachedAggID
- dot3adAggPortState
- dot3adAggPortStatsLACPDUsRx
- dot3adAggPortStatsMarkerPDUsRx
- dot3adAggPortStatsMarkerResponsePDUsRx
- dot3adAggPortStatsUnknownRx
- dot3adAggPortStatsIllegalRx
- dot3adAggPortStatsLACPDUsTx

- dot3adAggPortStatsMarkerPDUsTx
- dot3adAggPortStatsMarkerResponsePDUsTx

slow_path_counts (enterprise = 0, format = 8)

- unknown
- other
- cam_miss
- cam_full
- no_hw_support
- cntrl

ib_counters (enterprise = 0, format = 9)

- PortXmitData
- PortRcvData
- PortXmitPkts
- PortRcvPkts
- SymbolErrorCounter
- LinkErrorRecoveryCounter
- LinkDownedCounter
- PortRcvErrors
- PortRcvRemotePhysicalErrors
- PortRcvSwitchRelayErrors
- PortXmitDiscards
- PortXmitConstraintErrors
- PortRcvConstraintErrors
- LocalLinkIntegrityErrors
- ExcessiveBufferOverrunErrors

- VL15Dropped

sfp (enterprise = 0, format = 10)

- module_id
- module_num_lanes
- module_supply_voltage
- module_temperature
- index
- tx_bias_current
- tx_power
- tx_power_min
- tx_power_max
- tx_wavelength
- rx_power
- rx_power_min
- rx_power_max
- rx_wavelength

processor (enterprise = 0, format = 1001)

- 5s_cpu
- 1m_cpu
- 5m_cpu
- total_memory
- free_memory

radio_utilization (enterprise = 0, format = 1002)

- elapsed_time

- on_channel_time
- on_channel_busy_time

queue_length (enterprise = 0, format = 1003)

- queueIndex
- segmentSize
- queueSegments
- queueLength0
- queueLength1
- queueLength2
- queueLength4
- queueLength8
- queueLength32
- queueLength128
- queueLength1024
- queueLengthMore
- dropped

of_port (enterprise = 0, format = 1004)

- datapath_id
- port_no

port_name (enterprise = 0, format = 1005)

- name

host_descr (enterprise = 0, format = 2000)

- hostname
- uuid

- machine_type
- os_name
- os_release

host_adapters (enterprise = 0, format = 2001)

- ifIndex
- mac_address

host_parent (enterprise = 0, format = 2002)

- container_type
- container_index

host_cpu (enterprise = 0, format = 2003)

- load_one
- load_five
- load_fifteen
- proc_run
- proc_total
- cpu_num
- cpu_speed
- uptime
- cpu_user
- cpu_nice
- cpu_system
- cpu_idle
- cpu_wio
- cpu_intr

- cpu_sintr
- interrupts
- contexts

host_memory (enterprise = 0, format = 2004)

- mem_total
- mem_free
- mem_shared
- mem_buffers
- mem_cached
- swap_total
- swap_free
- page_in
- page_out
- swap_in
- swap_out

host_disk_io (enterprise = 0, format = 2005)

- hyper disk_total
- disk_free
- part_max_used
- reads
- bytes_read
- read_time
- writes
- bytes_written
- write_time

host_net_io (enterprise = 0, format = 2006)

- hyper bytes_in
- pkts_in
- errs_in
- drops_in
- bytes_out
- packets_out
- errs_out
- drops_out

mib2_ip_group (enterprise = 0, format = 2007)

- ipForwarding
- ipDefaultTTL
- ipInReceives
- ipInHdrErrors
- ipInAddrErrors
- ipForwDatagrams
- ipInUnknownProtos
- ipInDiscards
- ipInDelivers
- ipOutRequests
- ipOutDiscards
- ipOutNoRoutes
- ipReasmTimeout
- ipReasmReqds
- ipReasmOKs

- ipReasmFails
- ipFragOKs
- ipFragFails
- ipFragCreates

mib2_icmp_group (enterprise = 0, format = 2008)

- icmpInMsgs
- icmpInErrors
- icmpInDestUnreachs
- icmpInTimeExcds
- icmpInParamProbs
- icmpInSrcQuenchs
- icmpInRedirects
- icmpInEchos
- icmpInEchoReps
- icmpInTimestamps
- icmpInAddrMasks
- icmpInAddrMaskReps
- icmpOutMsgs
- icmpOutErrors
- icmpOutDestUnreachs
- icmpOutTimeExcds
- icmpOutParamProbs
- icmpOutSrcQuenchs
- icmpOutRedirects
- icmpOutEchos

- icmpOutEchoReps
- icmpOutTimestamps
- icmpOutTimestampReps
- icmpOutAddrMasks
- icmpOutAddrMaskReps

mib2_tcp_group (enterprise = 0, format = 2009

- tcpRtoAlgorithm
- tcpRtoMin
- tcpRtoMax
- tcpMaxConn
- tcpActiveOpens
- tcpPassiveOpens
- tcpAttemptFails
- tcpEstabResets
- tcpCurrEstab
- tcpInSegs
- tcpOutSegs
- tcpRetransSegs
- tcpInErrs
- tcpOutRsts
- tcpInCsumErrs

mib2_udp_group (enterprise = 0, format = 2010)

- udpInDatagrams
- udpNoPorts
- udpInErrors

- udpOutDatagrams
- udpRcvbufErrors
- udpSndbufErrors
- udpInCsumErrors

virt_node (enterprise = 0, format = 2100)

- mhz
- cpus
- memory
- memory_free
- num_domains

virt_cpu (enterprise = 0, format = 2101)

- state
- cpuTime
- nrVirtCpu

virt_memory (enterprise = 0, format = 2102)

- memory
- maxMemory

virt_disk_io (enterprise = 0, format = 2103)

- capacity
- allocation
- available
- rd_req
- rd_bytes
- wr_req

- wr_bytes
- errs

virt_net_io (enterprise = 0, format = 2104)

- rx_bytes
- rx_packets
- rx_errs
- rx_drop
- tx_bytes
- tx_packets
- tx_errs
- tx_drop

jmx_runtime (enterprise = 0, format = 2105)

- vm_name
- vm_vendor
- vm_version

jmx_statistics (enterprise = 0, format = 2106)

- heap_initial
- heap_used
- heap_committed
- heap_max
- non_heap_initial
- non_heap_used
- non_heap_committed
- non_heap_max

- gc_count
- gc_time
- classes_loaded
- classes_total
- classes_unloaded
- compilation_time
- thread_num_live
- thread_num_daemon
- thread_num_started
- fd_open_count
- fd_max_count

memcached_counters (enterprise = 0, format = 2200)

- uptime
- rusage_user
- rusage_system
- curr_connections
- total_connections
- connection_structures
- cmd_get
- cmd_set
- cmd_flush
- get_hits
- get_misses
- delete_hits
- delete_misses

- incr_hits
- incr_misses
- decr_hits
- decr_misses
- cas_misses
- cas_hits
- cas_badval
- auth_cmds
- auth_errors
- bytes_read
- bytes_written
- limit_maxbytes
- conn_accepts
- listen_disabled_num
- threads
- conn_yields
- bytes
- curr_items
- total_items
- evictions

http_counters (enterprise = 0, format = 2201)

- method_option_count
- method_get_count
- method_head_count
- method_post_count

- method_put_count
- method_delete_count
- method_trace_count
- method_connect_count
- method_other_count
- status_1XX_count
- status_2XX_count
- status_3XX_count
- status_4XX_count
- status_5XX_count
- status_other_count

app_operations (enterprise = 0, format = 2202)

- success
- other
- timeout
- internal_error
- bad_request
- forbidden
- too_large
- not_implemented
- not_found
- unavailable
- unauthorized

app_resources (enterprise = 0, format = 2203)

- user_time

- system_time
- mem_used
- mem_max
- fd_open
- fd_max
- conn_open
- conn_max

memcache_counters (enterprise = 0, format = 2204)

- cmd_set
- cmd_touch
- cmd_flush
- get_hits
- get_misses
- delete_hits
- delete_misses
- incr_hits
- incr_misses
- decr_hits
- decr_misses
- cas_hits
- cas_misses
- cas_badval
- auth_cmds
- auth_errors
- threads

- conn_yields
- listen_disabled_num
- curr_connections
- ejected_connections
- total_connections
- connection_structures
- evictions
- reclaimed
- curr_items
- total_items
- bytes_read
- bytes_written
- bytes
- limit_maxbytes

app_workers (enterprise = 0, format = 2206)

- workers_active
- workers_idle
- workers_max
- req_delayed
- req_dropped

ovs_dp_stats (enterprise = 0, format = 2207)

- hits
- misses
- lost
- mask_hits

- flows
- masks

energy (enterprise = 0, format = 3000)

- voltage
- current
- real_power
- power_factor
- energy
- errors

temperature (enterprise = 0, format = 3001)

- minimum
- maximum
- errors

humidity (enterprise = 0, format = 3002)

- relative humidity

fans (enterprise = 0, format = 3003)

- total
- failed
- speed

bst_device_buffers (enterprise = 4413, format = 1)

- unicast buffers percentage utilization
- multicast buffers percentage utilization

bst_port_buffers (enterprise = 4413, format = 2)

- ingress unicast buffers utilization
- ingress multicast buffers utilization
- egress unicast buffers utilization
- egress multicast buffers utilization
- per egress queue unicast buffers utilization
- per egress queue multicast buffers utilization

hw_tables (enterprise = 4413, format = 3)

- broadcom_hw_host_entries
- broadcom_hw_host_entries_max
- broadcom_hw_ipv4_entries
- broadcom_hw_ipv4_entries_max
- broadcom_hw_ipv6_entries
- broadcom_hw_ipv6_entries_max
- broadcom_hw_ipv4_ipv6_entries
- broadcom_hw_ipv6_ipv6_entries_max
- broadcom_hw_long_ipv6_entries
- broadcom_hw_long_ipv6_entries_max
- broadcom_hw_total_routes
- broadcom_hw_total_routes_max
- broadcom_hw_ecmp_nexthops
- broadcom_hw_ecmp_nexthops_max
- broadcom_hw_mac_entries
- broadcom_hw_mac_entries_max
- broadcom_hw_ipv4_neighbors

- broadcom_hw_ipv6_neighbors
- broadcom_hw_ipv4_routes
- broadcom_hw_ipv6_routes
- broadcom_hw_acl_ingress_entries
- broadcom_hw_acl_ingress_entries_max
- broadcom_hw_acl_ingress_counters
- broadcom_hw_acl_ingress_counters_max
- broadcom_hw_acl_ingress_meters
- broadcom_hw_acl_ingress_meters_max
- broadcom_hw_acl_ingress_slices
- broadcom_hw_acl_ingress_slices_max
- broadcom_hw_acl_egress_entries
- broadcom_hw_acl_egress_entries_max
- broadcom_hw_acl_egress_counters
- broadcom_hw_acl_egress_counters_max
- broadcom_hw_acl_egress_meters
- broadcom_hw_acl_egress_meters_max
- broadcom_hw_acl_egress_slices
- broadcom_hw_acl_egress_slices_max

nvidia_gpu (enterprise = 5703, format = 1)

- nvidia_gpu_devices
- nvidia_gpu_processes
- nvidia_gpu_gpu_time
- nvidia_gpu_mem_time
- nvidia_gpu_mem_total

- nvidia_gpu_mem_free
- nvidia_gpu_ecc_errors
- nvidia_gpu_energy
- nvidia_gpu_temperature
- nvidia_gpu_fan_speed

RELATED DOCUMENTATION

IPFIX IEs | **970**

NetFlow IEs | **1035**

sFlow IEs (Counter Samples) | **1113**

CHAPTER 37

Flow Enrichment

IN THIS CHAPTER

Maxmind GeoIP2 and GeoLite2

The Apstra Flow collector can determine attributes associated with the autonomous system (AS) and geolocation to which a public IP address belongs using GeoLite2 databases. See the Maxmind website to sign up and download the databases. You can then make these databases available to the Flow Data collector for enrichment of public IP addresses.

RELATED DOCUMENTATION

User-Defined Metadata | 1138

Network Interfaces | 1146

User-Defined Metadata

IN THIS SECTION

User-Defined Metadata Enrichment

IN THIS SECTION

IP Address Enrichment Module

The IP address enrichment module provides supplemental information for IP addresses, such as hostname, autonomous system, geolocation, reputation and user-defined metadata. This information is cached for improved performance and flow record throughput. For more control of when enrichment is applied, you can include or exclude IP addresses from various enrichers by CIDR, IP range or individual IP address.

For example:

```
# Specify whether the IP/CIDR/Range is considered to be "internal".
192.0.2.0/24:
  internal: true

# Additional options are name, vlan, tags and metadata.
192.0.2.192/26:
  name: atlanta_guest_wifi
  vlan: 1001
  tags:
    - wifi
    - dhcp
  metadata:
    dhcp.pool.name: atlanta_guest_wifi
    .site.id: atlanta

# Metadata fields beginning with a . will be organized under the object containing the IP
address.
192.0.2.194-192.0.2.198:
  metadata:
    .site.bldg.id: hq
    .site.floor.id: 2
```

```
    .site.rack.id: 1

# An individual IP address.
192.0.2.194:
  metadata:
    device.type.name: wifi_ap
```

Metadata Types (IP Addresses)

The user-defined metadata enricher supports a combination of predefined metadata types and enables you to provide custom data as key-value pairs. Table 1 on page 1148 describes the metadata types you can use for IP addresses.

Table 33: IP Address Metadata Types

Attribute	Data Type	Field Populated	Description
`internal`	boolean	`<object>.isInternal`	Specifies whether or not the IP belongs to a network considered to be *internal*.
`name`	string	`<object>.ip.subnet.name`	Name given to this subnet.
`vlan`	number (0-4094)	`<object>.vlan.tag.id`	A VLAN ID.
`tags`	array of strings	`<object>.ip.subnet.tags`	Tags that describe attributes of the subnet or IP address.
`metadata`	sequence of attributes	`<object><attribute>` or `<attribute>`	Key-value pairs added at the IP object or record levels.

Detailed Attribute Descriptions

- `internal`: Boolean attribute used to specify whether the CIDR, range or IP address is *internal* or *external*. This differs from whether the IP address is within a private or public IP range.

  Some private IP addresses are considered *external*, such as IPs used within a DMZ. Similarly some public IPs are still considered *internal* if the IPs are assigned to resources operated by the organization and to which access is generally restricted.

- `name`: string attribute used to provide a user-friendly name to a subnet relevant to the user or organization.

  > **NOTE**: Only a single `name` value is returned for a given IP address. Make sure that there are no conflicting names among overlapping CIDRs, ranges and IP addresses. If you must assign multiple values, add these values to the `tags` attribute.

- `vlan`: Enables you to specify a VLAN tag for a CIDR, range or IP address. This tag is typically assigned to source and destination and client and server related fields.
  - This tag does not conflict with VLAN tags provided in the flow records from network devices.
  - Devices report on the VLAN tags observed on their own interfaces, *not* the flow endpoints.
  - The VLAN tags reported by devices are typically assigned to the *in* and *out* related fields.
- `tags`: Array of string values for attributes that further describe the CIDR, range or IP address.
- `metadata`: List of key-value pairs added as fields to the record. These fields can be *custom* fields specific or existing fields from the Apstra Flow CODEX schema. When you specify CODEX fields, the configured metadata value overrides any values that exist in the record.

You can specify key names with or without a leading "."

- If specified *with* a leading "." the field is placed within the parent object containing the network interface.
- If specified *without* a leading "." the field is placed at the root of the record.

Consider the IP address from `flow.src.ip.addr`:

- If the metadata key is defined as `.site.name`, the value is assigned to `flow.src.site.name`.
- If the metadata key is defined as `site.name`, the value is assigned directly to `site.name`.

Merging Values from Multiple Definitions

You can merge attribute values for an IP address that matches multiple CIDR, range or IP address entries into a single result set.

For example:

```
192.168.0.0/16:
  metadata:
    .geo.loc.coord: 48.167106,11.486918
```

```
    .geo.city.name: Munich
    .geo.country.code: DE
    .geo.country.name: Germany
    .geo.tz.name: Europe/Berlin

192.168.1.0/24:
  name: munich_hq
  tags:
    - campus
  metadata:
    sec.zone.name: campus

192.168.1.151-192.168.1.200:
  tags:
    - guest_wifi
    - dhcp
  metadata:
    .host.name: guest_wifi
    .ip.addr: 192.168.1.0
```

The above example includes:

- A Class C private network `192.168.0.0/16` that includes location metadata.
- A `192.168.0.0/24` subnet tagged as the campus network and the firewall zone to which it belongs.
- A range of IP address that belong to the guest WiFi and are provided by DHCP.

Because the value `flow.src.ip.addr` (`192.168.1.152`), matches all three entries in the above configuration, the resulting enrichment fields added to the record will be:

```
flow.src.ip.subnet.name: munich_hq
flow.src.ip.subnet.tags: [campus guest_wifi dhcp]
flow.src.geo.loc.coord: 48.167106,11.486918
flow.src.geo.city.name: Munich
flow.src.geo.country.code: DE
flow.src.geo.country.name: Germany
flow.src.geo.tz.name: Europe/Berlin
sec.zone.name: campus
flow.src.host.name: guest_wifi
flow.src.ip.addr: 192.168.1.0
```

NOTE: In the above use case, the `host.name` and `ip.addr` were overridden to generic static values anonymizing the individual guest WiFi users. This enables the traffic to be collected and analyzed without tracking each guest individually. This also allows network or security operations to investigate suspect traffic they might want to block, while preserving individual guests' privacy.

Scoping Enrichment with Include/Exclude

IN THIS SECTION

You can include or exclude the hostname, DNS, and Maxmind GeoIP enrichment features to a subset of IP addresses by specifying ASs or CIDRs. You can specify the Include/exclude definitions in the provided YAML files to update and refresh without the need to restart the Apstra Flow collector. See `/etc/juniper/hostname/incl_excl.yml` and `/etc/juniper/hostname/user_defined.yml`.

The following output shows an example of include/exclude definitions:

```
include:
  asn:
    - 14168
  cidr:
    - 10.0.0.0/8
    - 192.168.0.0/16
exclude:
  #asn:
  #  -
  cidr:
    - 192.168.100.0/24
```

Evaluation of Include/Exclude Definitions

It is important to understand how include/exclude definitions are evaluated to ensure your configuration provides the desired outcome.

The following rules apply:

- If no specific include values are defined, *everything* is included.
- Exclude values are evaluated within the scope of included values.

Examples of Include/Exclude Definitions

NOTE: While the following examples use only CIDRs, the same logic applies to ASN values.

no include/exclude definitions

```
# no path provided or an empty file
```

If no include/excludes are defined, *everything* is included.

Table 34: no include/exclude IP Addresses

IP Address	Included
192.168.0.1	yes
10.0.0.1	yes
10.111.0.1	yes

only include is defined

```
include:
  cidr:
    - 10.0.0.0/8
```

Only those IP addresses within a defined AS or CIDR are included. In this example, only IP addresses within the CIDR `10.0.0.0/8` are included.

Table 35: only include IP Addresses

IP Address	Included
192.168.0.1	no
10.0.0.1	yes
10.111.0.1	yes

only exclude is defined

```
exclude:
  cidr:
    - 10.111.0.0/16
```

All IP address *not* specifically excluded by the defined AS or CIDR are included. In this example, all IP addresses *except* those within the CIDR `10.111.0.0/16` are included.

Table 36: only exclude IP Addresses

IP Address	Included
192.168.0.1	yes
10.0.0.1	yes
10.111.0.1	no

both include/exclude are defined

```
include:
  cidr:
    - 10.0.0.0/8
```

```
exclude:
  cidr:
    - 10.111.0.0/16
```

Only those IP addresses within a specified AS or CIDR are included, *except* those within an excluded AS or CIDR.

Table 37: include/exclude IP Addresses

IP Address	Included
192.168.0.1	no
10.0.0.1	yes
10.111.0.1	no

- `192.168.0.1` is *not* included because it is not within an included AS or CIDR.
- `10.0.0.1` is included because it is within an included AS or CIDR.
- `10.111.0.1` is *not* included.

SEE ALSO

Maxmind GeoIP2 and GeoLite2 | **1138**

Network Interfaces | **1146**

Network Interfaces

IN THIS SECTION

Network Interface Enrichment Module

The network interface enrichment module provides supplemental information for network interfaces, such as name (`ifName`), description (`ifDescr`), alias (`ifAlias`), type (`ifType`), bandwidth (`ifSpeed/ifHighSpeed`), committed information rate (CIR), user-defined tags, and additional user-defined metadata. These values are cached for improved performance and flow record throughput.

The following is a example of the network interface enrichment module:

```
10.0.0.1:
  1:
    ifName: lo
    ifDescr: lo
    ifAlias: lo
    ifType: 24
    ifSpeed: 10000000
    tags:
      - router_mgmt
    metadata:
      sec.zone.name: network
  3:
    internal: false
    ifName: eth0
    ifDescr: eth0
    ifAlias: internet
    ifType: 6
    ifSpeed: 1000000000
    cirIn: 200000000
    cirOut: 12000000
    tags:
      - verizon
    metadata:
      sec.zone.name: internet

10.0.0.2:
  501:
    ifName: vlan
    ifDescr: vlan
    ifSpeed: 1000000000
  502:
    ifName: ge-0/0/0
```

```
    ifDescr: ge-0/0/0
    ifSpeed: 1000000000
```

Metadata Types (Network Interfaces)

The user-defined metadata enricher supports a combination of predefined metadata types as well as the ability to provide custom data as key-value pairs. Table 1 on page 1148 describes the types of metadata you can use for network interfaces.

Table 38: Network Interfaces Metadata Types

Attribute	Data Type	Field Populated	Description
`ifName`	string	`<object>.netif.name`	The textual name of the interface. The value of this object must match name of the network interface as assigned by the device.
`ifAlias`	string	`<object>.netif.alias`	An administratively defined "alias" name for the interface.
`ifType`	unsigned	`<object>.netif.type.id`, `<object>.netif.type.name`	The type of interface as specified in IF-MIB (RFC 2233). Additional values for `ifType` are assigned by IANA through updates to the `IANAifType` textual convention.
`ifSpeed`	unsigned	`<object>.netif.bandwidth.bw`	The interface bandwidth in bps (bits per second).
`cirIn`	unsigned	`<object>.netif.bandwidth.prov.in`	The interface ingress provisioned maximum bandwidth in bps.
`internal`	bool	`<object>.isInternal`	Specifies whether or not the network interface is connected to a network considered to be *internal*.

Table 38: Network Interfaces Metadata Types *(Continued)*

Attribute	Data Type	Field Populated	Description
`tags`	array of strings	`<object>.netif.tags`	Tags that describe attributes of the network interface.
`metadata`	sequence of attributes	`<object><attribute>` or `<attribute>`	Key-value pairs added at the network interface object or record levels. These fields can be either custom fields specific to the needs of the user, or existing fields from the Apstra Flow CODEX schema. If you specify CODEX fields, the configured metadata value overrides any values that exist in the record.

You can specify key names with or without a leading "."

- If specified *with* a leading "." the field is placed within the parent object containing the network interface.
- If specified *without* a leading "." the field is placed at the root of the record.

Consider the network interface from `flow.src.ip.addr`:

- If the metadata key is defined as `.circuit.name`, the value is assigned to `flow.in.netif.circuit.name`.
- If the metadata key is defined as `circuit.name`, the value is assigned directly to `circuit.name`.

SEE ALSO

Maxmind GeoIP2 and GeoLite2 | **1138**

User-Defined Metadata | **1138**

CHAPTER 38

Monitor Apstra Flow

IN THIS CHAPTER

Metrics

IN THIS SECTION

The Apstra Flow collector exposes the `/metrics` endpoint to provide Prometheus-compatible statistics related to its performance and the resources it uses. The endpoint returns data in a Prometheus text-based exposition format. See the Prometheus documentation to learn more.

This topic describes the type of statistics you can retrieve from the `/metrics` endpoint in Apstra Flow.

app_info

The `app_info` statistic shows the application details. For example:

```
app_info{arch="arm64",cpus="8",env="native",hostname="M1-MacBook-
Pro.local",os="darwin",run_id="b1214e11-198f-43e7-81f1-c9986e9b3ff7"} 1
```

This record contains the following labels:

Table 39: Application Labels

Label	Description
`arch`	The environment running the application.
`cpus`	The number of available CPUs.
`env`	Native install or Docker install.
`hostname`	The name of machine.
`os`	The operating system running the application.
`run_id`	The application ID.

License Units

The `license_units` statistic shows details on your Apstra license. For example:

```
license_units{account_id="",expiration="0",level="0",riskiq_disabled="false"} 1
```

This record contains the following labels:

Table 40: License Units Labels

Label	Description
`account_id`	The license account ID.
`expiration`	The license expiration date.
`level`	The license level.
`riskiq_disabled`	riskiq is disabled for the license.

Flow UDP Server

IN THIS SECTION

The following statistics show the options for UDP server input.

udp_server_packet_queue_util

The `udp_server_packet_queue_util` statistic shows the utilization of the packet queue that stores received packets waiting to be processed. For example:

```
udp_server_packet_queue_util{application="flowcoll"} 0
```

This record contains the following label:

Table 41: udp_server_packet_queue_util Label

Label	Description
`application`	The name of the application.

udp_server_packets_received_total

The `udp_server_packets_received_total` statistic shows the total count of packets received by the UDP server. For example:

```
udp_server_packets_received_total{application="flowcoll",port="9995"} 0
```

This record contains the following labels:

Table 42: udp_server_packets_received_total Labels

Label	Description
application	The name of the application.
port	The port on which the UDP server listens.

udp_server_bytes_received_total

The udp_server_bytes_received_total statistic provides the total count of bytes received by the UDP server.

```
udp_server_bytes_received_total{application="flowcoll",port="9995"} 0
```

This record contains the following labels:

Table 43: udp_server_bytes_received_total Labels

Label	Description
application	Name of the application.
port	Port on which the UDP server listens.

Processor

IN THIS SECTION

- record_queue_util | 1154

record_queue_util

The `record_queue_util` statistic shows the ratio of the record queue size divided by its capacity. For example:

```
record_queue_util{application="flowcoll"} 0
```

This record contains the following label:

Table 44: record_queue_util Label

Label	Description
`application`	The name of the application.

OpenSearch Output

IN THIS SECTION

outputs_records_received_total

The `outputs_records_received_total` statistic shows the total number of records received by the output. For example:

```
outputs_records_received_total{application="flowcoll",namespace="default",output="opensearch"} 0
```

This record contains the following labels:

Table 45: outputs_records_received_total Labels

Label	Description
`application`	The name of the application.
`namespace`	The name of the namespace.
`output`	The name of the output.

outputs_records_sent_total

The `outputs_records_sent_total` statistic shows the total number of records sent by the output. For example:

```
outputs_records_sent_total{application="flowcoll",namespace="default",output="opensearch"} 0
```

This record contains the following labels:

Table 46: outputs_records_sent_total Labels

Label	Description
`application`	The name of the application.
`namespace`	The name of the namespace.
`output`	The name of the output.

outputs_bulk_requests_total

The `outputs_bulk_requests_total` statistic shows the total count of bulk requests sent by the output. For example:

```
outputs_bulk_requests_total{application="flowcoll",namespace="default",output="opensearch"} 0
```

This record contains the following labels:

Table 47: outputs_bulk_request_total Labels

Label	Description
`application`	The name of the application.
`namespace`	The name of the namespace.
`output`	The name of the output.

outputs_bulk_requests_errored_total

The `outputs_bulk_requests_errored_total` statistic shows the total count of errored bulk requests. For example:

```
outputs_bulk_requests_errored_total{application="flowcoll",namespace="default",output="opensearch
"} 0The outputs_bulk_requests_errored_total record provides the following labels:
```

This record contains the following labels:

Table 48: outputs_bulk_request_errored_total Labels

Label	Description
`application`	The name of the application.
`namespace`	The name of the namespace.
`output`	The name of the output.

outputs_records_errored_total

The `outputs_records_errored_total` statistic shows the total count of errored records. For example:

```
outputs_records_errored_total{application="flowcoll",namespace="default",output="opensearch"} 0
```

This record contains the following labels:

Table 49: outputs_records_errored_total Labels

Label	Description
`application`	The name of the application.
`namespace`	The name of the namespace.
`output`	The name of the output.

CHAPTER 39

Configuration Reference

IN THIS CHAPTER

YAML Configuration Files

Apstra Flow supports YAML configuration files for all binaries.

To use a YAML file for configuration, place the file in the default location `/etc/juniper/<binary_name>.yml` or specify a custom location with the `--config` or `-c` flag when running the binary. For example:

```
./flow-collector --config /etc/juniper/flowcoll.yml <this is default location>
```

Or

```
./flow-collector -c /etc/flowdata/flowcoll.yml\ <this is an example of a custom location>
```

NOTE: You can also use environment variables to configure the Apstra Flow collector to override the YAML defined values.

To configure log settings for the flow-collector binary, follow these steps:

1. Create a YAML file called `flowcoll.yml`.

2. In the YAML file, go to the log settings section. From here, you can customize the log settings and create additional YAML files for other binaries, if needed.

```
EF_LOGGER_LEVEL: 'info'
EF_LOGGER_ENCODING: 'json'
EF_LOGGER_FILE_LOG_ENABLE: true
EF_LOGGER_FILE_LOG_FILENAME: '/var/log/juniper/flowcoll/flowcoll.log'
EF_LOGGER_FILE_LOG_MAX_SIZE: 100
EF_LOGGER_FILE_LOG_MAX_AGE: 7
EF_LOGGER_FILE_LOG_MAX_BACKUPS: 4
EF_LOGGER_FILE_LOG_COMPRESS: false
```

3. Run the flow-collector binary with the `--config` flag. Specify the path to your `flowcoll.yml` file as follows:

```
./flow-collector --config=/path/to/flowcoll.yml
```

RELATED DOCUMENTATION

Common Options | 1159

Apstra Flow Collector | 1187

Common Options

SUMMARY

This topic describes the common configuration options for Apstra Flow.

IN THIS SECTION

Licensing

SUMMARY

The following sections describe the licensing API configuration options for Apstra Flow.

IN THIS SECTION

EF_JUNIPER_APSTRA_API_HOSTNAME

Use this setting to define the hostname or IP address where the Apstra server provides its API services. This setting is the same IP address or hostname you use to access the Apstra GUI. Note that this value must start with `http://` or `https://`.

- Example: `http://localhost`
- Default value: `''`

NOTE: Use the EF_JUNIPER_APSTRA_API_ADDRESS and EF_JUNIPER_APSTRA_API_TLS_ENABLE environment variables over EF_JUNIPER_APSTRA_API_HOSTNAME to create the URI needed to connect to the Apstra license server.

EF_JUNIPER_APSTRA_API_PORT

Use this setting to specify the port number on which the Apstra server exposes its API services. The most commonly used ports are port 80 and port 443.

- Example: `80`
- Default value: `''`

EF_JUNIPER_APSTRA_API_TLS_SKIP_VERIFICATION

Set this value to `true` to bypass TLS verification, only if necessary.

> **NOTE**: While this action might be necessary under certain testing conditions, it also carries inherent security risks.

- Valid values: `true`, `false`
- Default value: `false` (uses TLS verification)

EF_JUNIPER_APSTRA_API_USERNAME

Use this setting to input the username associated with your Apstra server. This setting is the same username you use to access the Apstra GUI.

- Default value: `''`

EF_JUNIPER_APSTRA_API_PASSWORD

Use this setting to enter the password corresponding to your Apstra server. This password is the same password you use to access the Apstra GUI.

- Default value: `''`

Logging

SUMMARY

The following sections describe the logging configuration options for Apstra Flow.

IN THIS SECTION

EF_LOGGER_LEVEL

Use this setting to specify the output level for logging.

- Valid values: `debug`, `info`, `warn`, `error`, `panic`, `fatal`
- Default value: `info`

EF_LOGGER_ENCODING

Use this setting to specify the output format of the produced logs.

- Valid values: `console`, `json`
- Default: `json`

EF_LOGGER_FILE_LOG_ENABLE

Set to `true` to enable writing logs to a file.

- Valid values: `true`, `false`
- Default value: `false`

EF_LOGGER_FILE_LOG_FILENAME

Use this setting to specify the path to the file where the logs are written. When you enable file logging, `EF_LOGGER_FILE_LOG_ENABLE` is set to `true`.

- Default path: `/var/log/flowdata/flowcoll/flowcoll.log`

EF_LOGGER_FILE_LOG_MAX_SIZE

Use this setting to specify the maximum size (MB) of the log file before it is rotated.

- Valid values: Any integer greater than `1`.
- Minimum value: `1`
- Default value: `100` megabytes

EF_LOGGER_FILE_LOG_MAX_AGE

Use this setting to specify the maximum number of days to retain old log files based on the timestamp encoded in the filenames. Because a day is defined as 24 hours, this value might not correspond to calendar days due to daylight savings, leap seconds, and so on.

- Valid values: Any integer greater than or equal to `0`.
- Default: `''` (Does not remove old log files based on age).

EF_LOGGER_FILE_LOG_MAX_BACKUPS

Use this setting to specify the maximum number of old log files to retain. The default is to retain 4 old log files.

> **NOTE**: You can remove log files due to age (see `EF_LOGGER_FILE_LOG_MAX_AGE`) even if the maximum number of backups is not reached.

- Valid values: Any integer greater than or equal to `0`.
- Default value: `4`

EF_LOGGER_FILE_LOG_COMPRESS

Use this setting to enable compression of log files. Set the value to `true` to enable compression.

- Valid values: `true`, `false`
- Default: `false`

API

SUMMARY

The following sections describe the API configuration options for Apstra Flow.

IN THIS SECTION

The Apstra Flow collector exposes an API that includes a Prometheus-compatible metrics endpoint and various endpoints for administrative tasks. These endpoints are described in the following sections:

EF_INSTANCE_NAME

Use this setting to configure the name of the collector instance.

- Default name: `default`

EF_API_IP

Use this setting to define the IP address on which the collector listens for API requests.

- Default IP address: `0.0.0.0`

EF_API_PORT

Use this setting to define the port the Apstra Flow collector listens for API requests.

- Default port number: `8080`

EF_API_TLS_ENABLE

Use this setting to enable or disable TLS connections to the API endpoint.

- Valid values: `true`, `false`
- Default value: `false`

EF_API_TLS_CERT_FILEPATH

Use this setting to specify the path to the certificate to use for TLS connections to the API endpoint.

- Default: `''`

EF_API_TLS_KEY_FILEPATH

Use this setting to specify the path to the key to use for TLS connections to the API endpoint.

- Default: `''`

EF_API_BASIC_AUTH_ENABLE

Use this setting to enable or disable basic authentication protection of API endpoints.

- Default: `false`

EF_API_BASIC_AUTH_USERNAME

Use this setting to specify the username to use to connect to basic authentication protection of API endpoints.

- Default: `''`

EF_API_BASIC_AUTH_PASSWORD

Use this setting to specify the password to use to connect to basic authentication protection of API endpoints.

- Default: `''`

Processor

SUMMARY

The following sections describe the processor configuration options for Apstra Flow.

IN THIS SECTION

EF_PROCESSOR_POOL_SIZE

Use this setting to specify the number of record processors to start. You will need at least one processor for every 2000 records/second. Increasing the number of processors enables the collector to better handle a high volume of high latency enrichment tasks such as DNS lookup for IP addresses.

NOTE: While increasing the number of processors can be beneficial, you might see diminishing returns at higher processor counts. Especially when the number of processors exceeds the number of available CPU threads (real cores + SMT threads) or vCPUs. If you require more than 64 processors, and have an Apstra standard or premium License, it might be more beneficial to use multiple collector instances.

- Default: `4 * the number of license units`

EF_PROCESSOR_DECODE_IPFIX_ENABLE

Set to `true` to enable decoding of IPFIX records.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_DECODE_NETFLOW1_ENABLE

Set to `true` to enable decoding of Netflow v1 records.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_DECODE_NETFLOW5_ENABLE

Set to `true` to enable decoding of Netflow v5 records.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_DECODE_NETFLOW6_ENABLE

Set to `true` to enable decoding of Netflow v6 records.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_DECODE_NETFLOW7_ENABLE

Set to `true` to enable decoding of Netflow v7 records.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_DECODE_NETFLOW9_ENABLE

Set to `true` to enable decoding of Netflow v9 records.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_DECODE_SFLOW5_ENABLE

Set to `true` to enable decoding of sFlow v5 records.

- Valid values: `true`, `false`

- Default value: `true`

EF_PROCESSOR_DECODE_SFLOW_FLOWS_ENABLE

Set to `true` to enable decoding of sFlow `flow_sample` and `flow_sample_expanded` records.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_DECODE_SFLOW_FLOWS_KEEP_SAMPLES

When set to `true`, the packet data from an sFlow `sampled_header` record is stored in `l2.section.sample` as a hex-encoded string.

- Valid values: `true`, `false`
- Default value: `false`

EF_PROCESSOR_DECODE_SFLOW_COUNTERS_ENABLE

Set to `true` to enable decoding of sFlow `counters_sample` and `counters_sample_expanded` records.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_DECODE_MAX_RECORDS_PER_PACKET

Corrupt packets can cause issues decoding records. To prevent this, you can limit the number of records to be decoded from a packet. When the network between the device and collector has an MTU larger than `1500`, the default value can be exceeded by normal packets. This configuration option enables you to increase the threshold when necessary.

- Default value: `64`

EF_PROCESSOR_TRANSLATE_KEEP_IDS

Use this setting to specify which ID values to be included in the final dataset.

- Valid values:
  - `none`: All identifiers are removed from the final dataset.
  - `default`: Most identifiers are removed from the final dataset. Note that some identifiers that are required for common use-cases, such as raw protocol port values, are included.

  - `all`: All identifiers are included in the final dataset.
- Default value: `default`

EF_PROCESSOR_DURATION_PRECISION

- Valid values:
  - `sec`: seconds
  - `ds`: deciseconds
  - `cs`: centiseconds
  - `ms`: millseconds
  - `us`: microseconds
  - `ns` : nanoseconds
- Default value: `ms`

> **NOTE**: For most data sources, this value is specified in milliseconds (`ms`).

EF_PROCESSOR_TIMESTAMP_PRECISION

Use this setting to specify the desired precision of timestamp values. Values received at a different precision than specified are converted to the desired precision.

- Valid values:
  - `sec`: seconds
  - `ds`: deciseconds
  - `cs`: centiseconds
  - `ms`: millseconds
  - `us`: microseconds
  - ns : nanoseconds
- Default value: `ms`

EF_PROCESSOR_PERCENT_NORM

The desired representation of percentages. Values received with a different representation than specified are converted to the desired representation.

- Valid values:
  - `1`: values are based on a scale of 0 to 1.
  - `100`: values are based on a scale of 0 to 100.
- Default value: `100`

EF_PROCESSOR_KEEP_CPU_TICKS

For telemetry sources that provide CPU usage, such as timeticks, utilization percentages are calculated. When this setting is set to `false` (default value), the timetick values are removed from the final dataset. If this setting is set to `true`, both the timetick values and utilization values are kept.

- Valid values: `true`, `false`
- Default value: `false`

EF_PROCESSOR_DROP_FIELDS

Use this setting to remove a comma-separated list of fields from all records.

> **NOTE**: The conversion from the default CODEX schema to alternate schemas happens within the respective outputs as fields are dropped before the outputs. You must use CODEX field names to configure this option.

- Valid values:
  - any CODEX-schema field names, comma-separated. For example: `flow.export.sysuptime,flow.export.version.ver,flow.start.sysuptime,flow.end.sysuptime,flow.seq_num`
- Default value: `''`

EF_PROCESSOR_ENRICH_ASN_PREF

If enrichment with AS attributes is enabled, but the AS is referenced directly in the flow record data, use this setting to specify which source is preferred. If the preferred source is not available for a given record, the decoder will fall-back to the alternate option.

- Valid values:

  - `lookup`: The AS determined by lookup.
  - `flow`: The AS is indicated directly in the flow record data.
- Default value: `lookup`

EF_PROCESSOR_ENRICH_JOIN_ASN

Some features require that related values from separate fields are stored as an array in a single field. A join attribute of AS related fields is enabled when this setting is set to `true`.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_ENRICH_JOIN_GEOIP

Some features require that related values from separate fields are stored as an array in a single field. A join attribute of IP subnetwork related fields is enabled of GeoIP related fields is enabled when this setting is set to `true`.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_ENRICH_JOIN_NETATTR

Some features require that related values from separate fields are stored as an array in a single field. A join attribute of network attribute related fields is enabled when this setting is `true`.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_ENRICH_JOIN_SUBNETATTR

Some features require that related values from separate fields are stored as an array in a single field. A join attribute of IP subnetwork related fields is enabled when this setting is set to `true`.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_ENRICH_JOIN_SEC

Some features require that related values from separate fields are stored as an array in a single field. A join attribute of security attribute related fields is enabled when this setting is set to `true`.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_EXPAND_CLISRV

The Apstra Flow collector infers the client/server relationship of two source/destination endpoints. Use this setting to enable or disable inference. The default value is `true`.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_EXPAND_CLISRV_NO_L4_PORTS

For flow records related to protocols that include no layer-4 ports, the collector infers the client/server relationship of the two source/destination endpoints using the order of the IP addresses. Use this setting to enable or disable inference. The default value is `true`.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_IFA_ENABLE

- Valid values: `true`, `false`
- Default value: `false`

EF_PROCESSOR_IFA_WORKER_SIZE

Use this setting to specify the number of IFA Hop record processors to start.

- Default value: `4 * the number of license units`

STDOUT Output

IN THIS SECTION

stdout

The `stdout` output is used to output JSON-formatted records to a standard output. This output is useful during the initial installation or when troubleshooting issues to see Apstra Flow collector output directly in the terminal or logs.

NOTE: The `stdout` output is used primarily for manual testing. This is because (at more than a few flow records per second), the data scrolls too fast to be useful.

EF_OUTPUT_STDOUT_ENABLE

Use this setting to enable or disable the `stdout`. The default value is `false`.

- Valid values: `true`, `false`
- Default value: `false`

EF_OUTPUT_STDOUT_FORMAT

Use this setting to specify how JSON documents are formatted. The default value is `json_pretty`.

- Valid values:
  - `json`: Outputs a single JSON-formatted record per line.
  - `json_pretty`: Outputs each record as a "pretty" formatted JSON document ("pretty" refers to whitespace added to the document for easier human-readability).
- Default value: `json_pretty`

Generic HTTP Output

SUMMARY

Use the Generic HTTP output option to send records to an HTTP endpoint.

IN THIS SECTION

EF_OUTPUT_GENERIC_HTTP_ENABLE

Use this setting to specify whether Generic HTTP output is enabled or disabled.

- Valid values: `true`, `false`
- Default value: `false`

EF_OUTPUT_GENERIC_HTTP_ECS_ENABLE

Use this setting to specify whether the data is sent using Elastic Common Schema (ECS).

- Valid values: `true`, `false`
- Default value: `false`

EF_OUTPUT_GENERIC_HTTP_BATCH_DEADLINE

Use this setting to specify the maximum waiting time (ms) for a batch of records to fill before being sent to the HTTP Endpoint.

- Default value: `2000`

EF_OUTPUT_GENERIC_HTTP_BATCH_MAX_BYTES

Use this setting to specify the maximum size (in bytes) for a batch of records being sent to the HTTP Endpoint.

- Default value: `8388608`

EF_OUTPUT_GENERIC_HTTP_TIMESTAMP_SOURCE

Use this setting to determine the timestamp source used to set the `@timestamp` field. Typically, `end` is the recommended setting. However, in the case of poorly behaving or misconfigured devices, `collect` might be the better option. For this reason the default value is `collect` because it handles a variety of scenarios.

Valid values:

- `start`: The flow start time indicated in the flow. Use the timestamp from `flow.start.timestamp`:
- `end`: The flow end time (or last reported time). Use the timestamp from `flow.end.timestamp`.
- `export`: The time from the flow record header. Use the timestamp from `flow.export.timestamp`.
- `collect`: The time that the collector processed the flow records. Use the timestamp from `flow.collect.timestamp`.
- Default value: `collect`

EF_OUTPUT_GENERIC_HTTP_ADDRESSES

Specifies the HTTP servers to which the output connects. It is a comma-separated list of HTTP servers, including port number.

IMPORTANT: Do not include `http://` or `https://` in the provided value. You enable of disable TLS communications by using using `EF_OUTPUT_GENERIC_HTTP_TLS_ENABLE`.

- • Default value: ``

Default value: 127.0.0.1: 8888

EF_OUTPUT_GENERIC_HTTP_USERNAME

Use this setting to specify the username used to connect to the HTTP endpoint.

- Default value: ``

EF_OUTPUT_GENERIC_HTTP_PASSWORD

Use this setting to specify the password used to connect to the HTTP endpoint.

- Default value: ``

EF_OUTPUT_GENERIC_HTTP_TLS_ENABLE

Use this setting to enable or disable TLS connections to the HTTP server.

- Valid values: `true`, `false`
- Default value: `false`

EF_OUTPUT_GENERIC_HTTP_TLS_SKIP_VERIFICATION

Use this setting to enable or disable TLS verification of the HTTP server to which the output is trying to connect to.

- Valid values: `true`, `false`
- Default value: `false`

EF_OUTPUT_GENERIC_HTTP_TLS_CA_CERT_FILEPATH

Use this setting to specify the path to the CA certificate used to verify the HTTP server to which the output is attempting to connect to.

- Default value: `''`

EF_OUTPUT_GENERIC_HTTP_DROP_FIELDS

Use this setting to specify a comma-separated list of fields you want to remove from all records.

NOTE: Fields are dropped after any output specific fields are added and after any schema conversion. This means that you must use the field names shown in the Apstra Flow UI.

- Valid values: any field names that are related to the enabled schema, comma-separated. For example:
  `flow.export.sysuptime,flow.export.version.ver,flow.start.sysuptime,flow.end.sysuptime,flow.seq_num`
- Default value: `''`

Monitor

SUMMARY

The following sections describe the monitor output configuration options for Apstra Flow.

IN THIS SECTION

EF_OUTPUT_MONITOR_ENABLE

The monitor output generates a log message containing the rate of records received and decoded by the Apstra Flow collector over the past interval (see `EF_OUTPUT_MONITOR_INTERVAL`). This output is useful for sizing or troubleshooting. To enable this option, set `EF_OUTPUT_MONITOR_ENABLE` to `true`.

- Valid values: `true`, `false`
- Default value: `false`

EF_OUTPUT_MONITOR_INTERVAL

Use this setting to specify the interval, in seconds, at which the rate of records is calculated and logged.

- Default value: `300` (5 minutes)

OpenSearch

IN THIS SECTION

The following sections describe the OpenSearch output configuration options.

NOTE: You can use the OpenSearch output to send records to OpenSearch, Open Distro for OpenSearch and Amazon OpenSearch Service.

EF_OUTPUT_OPENSEARCH_ENABLE

Use this setting to enable or disable OpenSearch output. The default value is `false`.

- Valid values: `true`, `false`
- Default value: `false`

EF_OUTPUT_OPENSEARCH_BATCH_DEADLINE

Use this setting to specify the maximum time (in ms) to wait for a batch of records to fill up before the records are sent to the OpenSearch bulk API.

- Default value: `2000` ms.

EF_OUTPUT_OPENSEARCH_BATCH_MAX_BYTES

Use this setting to specify the maximum size of batch of records that can be sent to the OpenSearch bulk API.

- Default value: `8388608` bytes.

EF_OUTPUT_OPENSEARCH_TIMESTAMP_SOURCE

Use this setting to specify the timestamp source used to set the `@timestamp` field. The recommended setting is `end`. If your device is behaving poorly or is misconfigured, we suggest you use the `collect` option instead.

- Valid timestamp values:
  - `start`: The `flow.start.timestamp` indicates the flow start time.
  - `end`: The `flow.end.timestamp` is the last reported flow end time.
  - `export`: The `flow.export.timestamp` indicates time received from the flow record header.
  - `collect`: The `flow.collect.timestamp` indicates the time the Apstra Flow collector processes the flow record.
- Default timestamp value: `collect`

EF_OUTPUT_OPENSEARCH_INDEX_PERIOD

Use this setting to specify how often new indexes are created (daily, weekly, monthly) and how to create and delete indexes.

- Valid values:
  - `daily` : Indices are created each day. Specify this time period suffix as: `-yyyy.MM.dd`.
  - `weekly`: Indices are created each week. Specify this time period suffix as: `-yyyy.'w'ww`.
  - `monthly`: Indices are created each month. Specify this time period suffix as: `-yyyy.MM`.
  - `ilm` (Index Lifecycle Management): Use to create and delete indices.
- Default value: `daily`

EF_OUTPUT_OPENSEARCH_INDEX_SUFFIX

Use this setting to specify a suffix to the indexes. This setting is useful if you have separate indexes for different environments, locations or other organizational units.

- Default value: `''`

EF_OUTPUT_OPENSEARCH_INDEX_TEMPLATE_ENABLE

Use this setting to specify the output attempts to add the required index template to OpenSearch.

- Valid values: `true`, `false`
- Default value: `true`

EF_OUTPUT_OPENSEARCH_INDEX_TEMPLATE_OVERWRITE

Use this setting to determine if the index template should be overwritten or if it exists. If the output is configured to add the index template to OpenSearch, set `EF_OUTPUT_OPENSEARCH_INDEX_TEMPLATE_ENABLE` to `true`.

- Valid values: `true`, `false`
- Default value: `false`

EF_OUTPUT_OPENSEARCH_INDEX_TEMPLATE_SHARDS

Use this setting to indicate the number of shards in which the index is created. As a general rule, additional shards increases ingest performance, assuming there are sufficient data nodes in which the shards can be distributed.

- Recommended number of shards: equal to the number of OpenSearch data nodes to which data to which the data is indexed.
- Default number of shards: `3`

> **NOTE**: This setting configures the index template that is sent to OpenSearch. It does *not* change any existing indexes.

EF_OUTPUT_OPENSEARCH_INDEX_TEMPLATE_REPLICAS

Use this setting to specify the number of replicas created for each shard.

In general, additional replicas increases query performance assuming sufficient data nodes exist across which the replicas can be distributed.

If you are using a multinode cluster and data redundancy is desired, this value must be at least `1`.

- Recommended number of replicas:
  - Use `1` if indexing data to a multi-node cluster.
  - Use `0` for a single-node.
- Default value: `1`

> **NOTE**: This setting configures the index template sent to OpenSearch. It does *not* change any existing indexes.

EF_OUTPUT_OPENSEARCH_INDEX_TEMPLATE_REFRESH_INTERVAL

Use this setting to specify the period for the refresh interval. This setting indicates the time that newly ingested documents are added to a segment, before the segment is added to the indexes. Only after the refresh interval ends and the segment is added to the indexes, do the documents become searchable.

- Recommended refresh intervals:
  - `5s`: Use this value for the data to become available for queries more quickly. Note that shorter refresh intervals might negatively impact ingest performance.
  - `30s` (or longer): Use this value if maximizing ingest performance is your highest priority. Note that longer refresh intervals negatively impact the real-time accessibility of new records.
  - `10s` or `15s`: Use these values for most network traffic analytic use-cases. These interval numbers are a reasonable compromise between ingest performance and data accessibility.

- Default value: `10s`

> **NOTE**: This setting configures the indexes template that is sent to OpenSearch. It does *not* change any existing indexes.

EF_OUTPUT_OPENSEARCH_INDEX_TEMPLATE_CODEC

Use this setting to determine the level of compression used for stored values.

- Valid values:
  - `default`: stored values are compressed using LZ4.
  - `best_compression`: stored values are compressed using the `DEFLATE` value. This value reduces disk capacity requirements with the trade-off of slightly higher CPU utilization.
- Default value: `best_compression`

> **NOTE**: This setting configures the indexes template that is sent to OpenSearch. It does *not* change any existing indexes.

EF_OUTPUT_OPENSEARCH_INDEX_TEMPLATE_ISM_POLICY

If data is being stored to an Open Distro for an OpenSearch cluster, this setting specifies the Index State Management (ISM) policy ID that is applied to the indexes The default value is `''`.

> **NOTE**: You must configure the ISM policy separately in OpenSearch.

- Default value: `''`

EF_OUTPUT_OPENSEARCH_INDEX_TEMPLATE_PIPELINE_DEFAULT

Use this setting to specify the name of the OpenSearch default pipeline or to process the OpenSearch ingest pipeline before the pipeline is indexed.

- Default name: `_none`

EF_OUTPUT_OPENSEARCH_INDEX_TEMPLATE_PIPELINE_FINAL

Use this setting to specify the name of the OpenSearch final pipeline or to process the OpenSearch ingest pipeline before the pipeline is indexed.

- Default value: `_none`

EF_OUTPUT_OPENSEARCH_ADDRESSES

Use this setting to specify the OpenSearch servers to which the output should connect. This value is a comma-separated list of OpenSearch nodes, including port number. Do *not* include `http://` or `https://` in the value.

- Default value: `127.0.0.1:9200`

> **NOTE**: You can enable or disable TLS communications using the `EF_OUTPUT_OPENSEARCH_TLS_ENABLE` option.

EF_OUTPUT_OPENSEARCH_USERNAME

Use this setting to specify the username to connect to the OpenSearch server.

- Default value: `admin`

EF_OUTPUT_OPENSEARCH_PASSWORD

Use this setting to specify the password to connect to the OpenSearch server.

- Default value: `admin`

EF_OUTPUT_OPENSEARCH_CLIENT_CA_CERT_FILEPATH

Use this setting to specify the path to the CA certificate used for client PKI authentication.

- Default value: `''`

EF_OUTPUT_OPENSEARCH_CLIENT_CERT_FILEPATH

Use this setting to specify the path to the client certificate used for client PKI authentication.

- Default value: `''`

EF_OUTPUT_OPENSEARCH_CLIENT_KEY_FILEPATH

Use this setting to specify the path to the client key used for client PKI authentication.

- Default value: `''`

EF_OUTPUT_OPENSEARCH_TLS_ENABLE

Use this setting to enable or disable TLS connections to the OpenSearch server. The default value is `false`.

- Valid values: `true`, `false`
- Default value: `false`

EF_OUTPUT_OPENSEARCH_TLS_SKIP_VERIFICATION

Use this setting to enable or disable TLS verification of the OpenSearch server. The default value is `false`.

- Valid values: `true`, `false`
- Default value: `false`

EF_OUTPUT_OPENSEARCH_TLS_CA_CERT_FILEPATH

Use this setting to specify the path to the CA certificate used tp verify the OpenSearch server connection.

- Default value: `''`

EF_OUTPUT_OPENSEARCH_RETRY_ENABLE

Use this setting to specify whether to retry connecting to the OpenSearch server after a connection has failed.

- Valid values: `true`, `false`
- Default: `true`

EF_OUTPUT_OPENSEARCH_RETRY_ON_TIMEOUT_ENABLE

Use this setting to specify whether to retry bulk indexing requests that timed-out.

- Valid values: `true`, `false`
- Default: `true`

EF_OUTPUT_OPENSEARCH_MAX_RETRIES

Use this setting to specify the number of times to retry bulk indexing requests which have timed-out.

- Default value: `3` times

EF_OUTPUT_OPENSEARCH_RETRY_BACKOFF

Use this setting to specify the number of milliseconds (ms) you want the output to *backoff* before retrying a failed bulk request.

- Default value: `1000` ms

EF_OUTPUT_OPENSEARCH_DROP_FIELDS

Use this setting to create a comma-separated list of fields to be removed from all records.

> **NOTE**: Fields are dropped if you add any output specific fields and dropped after any schema conversion. Make sure you use the same field names as the names that appear in the Apstra GUI.

- Valid values: Any field names related to the enabled schema, comma-separated. For example: `flow.export.sysuptime,flow.export.version.ver,flow.start.sysuptime,flow.end.sysuptime,flow.seq_num`
- Default value: `''`

EF_OUTPUT_OPENSEARCH_ALLOWED_RECORD_TYPES

Use this setting to create a comma-separated list of record types. This list is particularly useful when used with multiple namespaced outputs, such as sending flow records to one datastore and telemetry to another.

- Valid values: `as_path_hop`, `flow_option`, `flow` , `telemetry`, `ifa_hop`
- Default values: `'as_path_hop,flow_option,flow,telemetry,ifa_hop '`

SEE ALSO

YAML Configuration Files | **1158**

Apstra Flow Collector | **1187**

Apstra Flow Collector

IN THIS SECTION

Inputs

IN THIS SECTION

EF_FLOW_SERVER_UDP_IP

The Apstra Flow collector receives network flow records over UDP. Use this setting to specify the interface IP address that the collector will listen on.

- Valid values: `0.0.0.0` or any valid IP address to which the UDP socket can be bound.
- Default IP address: `0.0.0.0` (listens on all interfaces)

EF_FLOW_SERVER_UDP_PORT

Use this setting to specify the UDP port on which the collector creates a socket to receive incoming packets. You can specify multiple ports, separated by a comma. For example: `2055,6343,4739`.

Valid values: Any valid port number.

Common values include:

- `2055`: Netflow standard port
- `4739`: IPFIX standard port
- `6343`: sFlow standard port
- `9995-9998`: Commonly use port numbers

EF_FLOW_SERVER_UDP_READ_BUFFER_MAX_SIZE

The size (in bytes) of the UDP receive buffer that the UDP server requests, is created by the operating system kernel when the socket is created. If this value exceeds the maximum allowed buffer size (`net.core.rmem_max` on Linux), the maximum allowed size is used.

- Default: `33554432`

EF_FLOW_PACKET_STREAM_MAX_SIZE

- Default: `16384` bytes

Decoder/Processor

IN THIS SECTION

EF_PROCESSOR_DECODE_IPFIX_ENABLE

Set to `true` to enable decoding of IPFIX records.

- Valid values: `true`, `false`
- Default: `true`

EF_PROCESSOR_DECODE_NETFLOW1_ENABLE

Set to `true` to enable decoding of Netflow v1 records.

- Valid values: `true`, `false`
- Default: `true`

EF_PROCESSOR_DECODE_NETFLOW5_ENABLE

Set to `true` to enable decoding of Netflow v5 records.

- Valid values: `true`, `false`
- Default: `true`

EF_PROCESSOR_DECODE_NETFLOW6_ENABLE

Set to `true` to enable decoding of Netflow v6 records.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_DECODE_NETFLOW7_ENABLE

Set to `true` to enable decoding of Netflow v7 records.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_DECODE_NETFLOW9_ENABLE

Set to `true` to enable decoding of Netflow v9 records.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_DECODE_SFLOW5_ENABLE

Set to `true` to enable decoding of sFlow v5 records.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_DECODE_SFLOW_FLOWS_ENABLE

Set to `true` to enable decoding of sFlow `flow_sample` and `flow_sample_expanded` records.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_DECODE_SFLOW_FLOWS_KEEP_SAMPLES

When set to `true`, the packet data from an sFlow `sampled_header` record is stored in `l2.section.sample` as a hex-encoded string.

- Valid values: `true`, `false`
- Default: `false`

EF_PROCESSOR_DECODE_SFLOW_COUNTERS_ENABLE

Set to `true` to enable decoding of sFlow `counters_sample` and `counters_sample_expanded` records.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_DECODE_MAX_RECORDS_PER_PACKET

Corrupt packets can cause issues decoding records. To prevent this, you can use this setting to limit the number of records that will be decoded from a packet. When the network between the device and collector has an MTU larger than `1500`, the default value might be exceeded by normal packets. The `EF_PROCESSOR_DECODE_MAX_RECORDS_PER_PACKET` setting allows you to increase the threshold, when necessary.

- Default value: `64`

EF_PROCESSOR_TRANSLATE_KEEP_IDS

Use this setting to specify the identifier values to be included in the final dataset.

Valid values:

- `none`: All identifiers are removed from the final dataset.
- `default`: Most identifiers are removed from the final dataset. Note that some identifiers required for common use-cases (such as raw protocol port values) are included in the final dataset.
- `all`: All identifiers are included in the final dataset.
- Default value: `default`

EF_PROCESSOR_ENRICH_ASN_PREF

If you enable enrichment with autonomous system (AS) attributes, and if AS is already indicated directly in the flow record data, you can use the `EF_PROCESSOR_ENRICH_ASN_PREF` setting to specify which source is preferred. If the preferred source is not available for a given record, the decoder fall backs to the alternate option.

- Valid values:
  - `lookup`: The AS is determined by lookup.
  - `flow`: The AS is indicated directly in the flow record data.
- Default value: `lookup`

EF_PROCESSOR_ENRICH_JOIN_ASN

Some features require that related values from separate fields are stored as an array in a single field. A *join* of AS related fields is enabled when `EF_PROCESSOR_ENRICH_JOIN_ASN` is set to `true`.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_ENRICH_JOIN_GEOIP

Some features require that related values from separate fields are stored as an array in a single field. A *join* of GeoIP related fields is enabled when `EF_PROCESSOR_ENRICH_JOIN_GEOIP` is set to `true`.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_ENRICH_JOIN_NETATTR

Some features require that related values from separate fields are stored as an array in a single field. A *join* of network attribute related fields is enabled when `EF_PROCESSOR_ENRICH_JOIN_NETATTR` is set to `true`.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_ENRICH_JOIN_SUBNETATTR

Some features require that related values from separate fields are stored as an array in a single field. A *join* of IP subnetwork attribute related fields is enabled when `EF_PROCESSOR_ENRICH_JOIN_SUBNETATTR` is set to `true`.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_ENRICH_JOIN_SEC

Some features require that related values from separate fields are stored as an array in a single field. A *join* of security attribute related fields is enabled when `EF_PROCESSOR_ENRICH_JOIN_SEC` is set to `true`.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_EXPAND_CLISRV

The collector infers the client/server relationship of two source/destination endpoints. The `EF_PROCESSOR_EXPAND_CLISRV` setting determines if inference is enabled or disabled.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_EXPAND_CLISRV_NO_L4_PORTS

For flow records related to protocols that include "no layer-4 ports", the collector infers the client/server relationship of the two source/destination endpoints by using the order of the IP addresses. Use this `EF_PROCESSOR_EXPAND_CLISRV_NO_L4_PORTS` setting to enable or disable inference. The default setting is `true`.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_IFA_ENABLE

- Valid values: `true`, `false`
- Default value: `false`

EF_PROCESSOR_IFA_WORKER_SIZE

Use to specify the the number of IFA Hop record processors to start.

- Default number: `4 * the number of license units`

Sampling Rates

IN THIS SECTION

Devices can sample packets to reduce the overall volume of traffic metered for flow accounting, The various sampling rate configuration options are described as follows:

fEF_PROCESSOR_ENRICH_SAMPLERATE_CACHE_SIZE

The Apstra Flow collector adjusts the calculation of bytes and packets based on the sampling rate used. Usually devices inform the collector of the sampling rate either within the flow record or as option data sent periodically by the device. Use the `EF_PROCESSOR_ENRICH_SAMPLERATE_CACHE_SIZE` setting to specify the size of the cache to be used to hold sample rate information learned from option data.

- Default value: `32768`

EF_PROCESSOR_ENRICH_SAMPLERATE_USERDEF_ENABLE

Sometimes a device might not transmit information about the sampling rate for which it is configured. Use the `EF_PROCESSOR_ENRICH_SAMPLERATE_USERDEF_ENABLE` setting to statically define the sampling rate in the file provided to the collector.

- Valid values: `true`, `false`
- Default value: `false`

EF_PROCESSOR_ENRICH_SAMPLERATE_USERDEF_PATH

If static sample rates are configured for devices in a file, the `EF_PROCESSOR_ENRICH_SAMPLERATE_USERDEF_PATH` setting specifies the path from where that file can be loaded.

For example:

```
'192.0.2.1': 1024
'192.0.2.2': 512
```

The default path is: `/etc/flowdata/settings/sample_rate.yml`

EF_PROCESSOR_ENRICH_SAMPLERATE_USERDEF_OVERRIDE

In some use cases, you might want to use a user-defined sample rate rather than the rate provided by the device. Set the `PROCESSOR_ENRICH_SAMPLERATE_USERDEF_OVERRIDE` setting to `true` to check for a user-defined rate even if the device has already provided a rate.

- Valid values: `true`, `false`
- Default value: `false`

General Settings

IN THIS SECTION

EF_PROCESSOR_ENRICH_TOTALS_IF_NO_DELTAS

Most flow exporters provide byte and packet quantities as *delta* values. Delta values refer to the byte and packet quantities since the last flow record was reported. However, some exporters, such as the Juniper MX-Series router sending IPFIX, provide these quantities only as *total* values. Total values refers to the quantity over the entire lifetime of the flow.

In cases where the exporter sends *only* totals, you might want to use these values to populate the `flow.bytes` and `flow.packets`. When `EF_PROCESSOR_ENRICH_TOTALS_IF_NO_DELTAS` is set to `true`, the *total* quantities are used.

NOTE: Total quantities can be problematic for many datastores. A simple sum of *total* values across multiple records within a time window will not produce an accurate quantity, as is it does with *delta* values. As a result, long-lived flows can over-report bytes and packets values if *total* values are used.

- Valid values: `true`, `false`
- Default value: `true`

Applications

IN THIS SECTION

The Apstra Flow collector caches application attributes learned from option data. The collector allows you to define application attributes by any combination of IP/CIDR/IP range and port/port range.

EF_PROCESSOR_ENRICH_APP_ID_ENABLE

- Valid values: `true`, `false`
- Default: `false`

EF_PROCESSOR_ENRICH_APP_ID_PATH

If the vendor-defined AppID to application attribute mappings is enabled (`EF_PROCESSOR_ENRICH_APP_ID_ENABLE` is `true`) this setting specifies the path to the file.

The default path is: `/etc/flowdata/app/appid.yml`

EF_PROCESSOR_ENRICH_APP_ID_TTL

Use this setting to specify the length of time the application attributes are cached after they are initially fetched.

NOTE: Changes to the underlying files are not made (even after the files were re-loaded at the refresh interval) until the AppID has expired from the cache.

- Default value: `7200`

EF_PROCESSOR_ENRICH_APP_IPPORT_ENABLE

Various flow record sources send the mapping of application IDs to applications names as option data. In cases where no application identity technology is available, you can specify applications by IP address and port number.

- Valid values: `true`, `false`

- Default value: `false`

EF_PROCESSOR_ENRICH_APP_IPPORT_PATH

When user-defined IP/port to application mappings is enabled, the (`EF_PROCESSOR_ENRICH_APP_IPPORT_ENABLE` is `true`) setting specifies the path to this file.

For example:

```
192.168.1.0/24:
  8090:
    name: "Synergy-cidr-port"
    category: "category-cidr-port"
    subcategory: "subcategory-cidr-port"
    metadata:
      ".location": "austin-cidr-port"
      "business.unit": "finance-cidr-port"
      "dev.unit": "dev-cidr-port"
      "app.count": 27

192.168.1.1-192.168.1.20:
  8090:
    name: "Synergy-iprange-port"
    category: "category-iprange-port"
    subcategory: "subcategory-iprange-port"
    metadata:
      .location: "austin-iprange-port"

  8090-9000:
    name: "Synergy-iprange-portrange"
    category: "category-iprange-portrange"
    subcategory: "subcategory-iprange-portrange"
    metadata:
      .location: "austin-iprange-portrange"
      business.unit: "finance-iprange-portrange"
      qa.unit: "qa-iprange-portrange"
      finace.unit: "finance-iprange-portrange"

192.168.1.1:
  8090:
    name: "Synergy-ip-port"
    category: "category-ip-port"
```

```
    subcategory: "subcategory-ip-port"
    metadata:
      .location: "austin-ip-port"
      business.unit: "finance-ip-port"
```

- Default path: `/etc/flowdata/app/ipport.yml`

EF_PROCESSOR_ENRICH_APP_IPPORT_TTL

Use this setting to specify the length of time application attributes are cached after they are initially fetched.

> **NOTE**: Changes to the underlying files are not made, even after the files have been reloaded at the refresh interval, until the IP/Port has expired from the cache.

- Default value: `7200`

EF_PROCESSOR_ENRICH_APP_IPPORT_PRIVATE

If user-defined application attributes are enabled (`EF_PROCESSOR_ENRICH_APP_IPPORT_ENABLE` is `true`) this setting specifies whether application names are checked for private IP addresses.

- Valid values: `true`, `false`
- Default: `true`

EF_PROCESSOR_ENRICH_APP_IPPORT_PUBLIC

If user-defined application attributes are enabled (`EF_PROCESSOR_ENRICH_APP_IPPORT_ENABLE` is `true`) this setting specifies whether application names are checked for public IP addresses.

- Valid values: `true`, `false`
- Default value: `false`

EF_PROCESSOR_ENRICH_APP_REFRESH_RATE

Files defined for application attribute enrichment can be loaded automatically to refresh values without restarting the collector. Use this setting to specifies the refresh interval, in minutes, that the file will be reloaded.

- Default value: `15` (`0` value disables this setting)

IP Addresses

IN THIS SECTION

Name Resolution

You can configure the collector to resolve IP addresses to hostnames. The following settings allow this feature to be tuned to the needs of your environment.

EF_PROCESSOR_ENRICH_IPADDR_DNS_ENABLE

Use this setting to enables DNS reverse lookups of IP addresses found in the received flow records.

- Valid values: `true`, `false`
- Default value: `false`

EF_PROCESSOR_ENRICH_IPADDR_DNS_NAMESERVER_IP

The collector uses the operating system's configured name resolution to resolve IP addresses to hostnames. This is the default behavior. Optionally, you can specify a nameserver to use instead.

NOTE: If configured, this setting *must* contain a valid IP address.

- Default: `empty`

EF_PROCESSOR_ENRICH_IPADDR_DNS_NAMESERVER_TIMEOUT

If `EF_PROCESSOR_ENRICH_IPADDR_DNS_NAMESERVER_IP` contains a valid IP address, this setting contains the timeout period, in milliseconds, for queries to the name server.

- Default: `3000`

EF_PROCESSOR_ENRICH_IPADDR_DNS_RESOLVE_PRIVATE

When DNS resolution is enabled (`EF_PROCESSOR_ENRICH_IPADDR_DNS_ENABLE` is `true`), this setting specifies whether private IP addresses will be resolved to hostnames.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_ENRICH_IPADDR_DNS_RESOLVE_PUBLIC

If DNS resolution is enabled (`EF_PROCESSOR_ENRICH_IPADDR_DNS_ENABLE` set to `true`), this setting specifies whether public IP addresses will be resolved to hostnames.

- Valid values: `true`, `false`
- Default: `true`

EF_PROCESSOR_ENRICH_IPADDR_DNS_USERDEF_PATH

The `EF_PROCESSOR_ENRICH_IPADDR_DNS_USERDEF_PATH` setting specifies the path to the file containing user-defined hostname mappings. This feature is enabled only if a path is configured, otherwise it is disabled.

```
'192.0.2.1': 'host1'
'192.0.2.2': 'host2'
```

- Default setting: `''`
- Recommended path: `/etc/flowdata/hostname/user_defined.yml`

EF_PROCESSOR_ENRICH_IPADDR_DNS_USERDEF_REFRESH_RATE

Use this setting to automatically load refresh values without restarting the collector. The value you specify indicates the refresh interval time, in minutes, that the file will take to reload.

- Default value: `15` (if set to `0`, refresh values are disabled)

EF_PROCESSOR_ENRICH_IPADDR_DNS_INCLEXCL_PATH

For more control of when enrichment is applied, you can include or exclude IP addresses from hostname enrichment by AS or CIDR. Use this setting to specify the path to the `inclu_excl.yml` file. For more information about the include/exclude functionality, see "Scoping Enrichment with Include/Exclude" on page 1143.

- Default setting: `''`
- Recommended path: `/etc/flowdata/hostname/incl_excl.yml`

EF_PROCESSOR_ENRICH_IPADDR_DNS_INCLEXCL_REFRESH_RATE

Use this setting to automatically refresh values without restarting the collector. The value you specify indicates the refresh interval, in minutes, that the file will take to reload.

- Default value: `15` (if set to `0`, refresh values are disabled)

Maxmind

EF_PROCESSOR_ENRICH_IPADDR_MAXMIND_ASN_ENABLE

Use this setting (`EF_PROCESSOR_ENRICH_IPADDR_MAXMIND_ASN_ENABLE` is `true`) to allow the collector to determine attributes associated with the ASs to which a public IP address belongs.

- Valid values: `true`, `false`
- Default value: `false`

EF_PROCESSOR_ENRICH_IPADDR_MAXMIND_ASN_PATH

Use this setting to specify the path to the Maxmind database. Enrichment with AS attributes is enabled using lookups in a Maxmind database when `EF_PROCESSOR_ENRICH_IPADDR_MAXMIND_ASN_ENABLE` is `true`.

- Default path: `/etc/flowdata/maxmind/GeoLite2-ASN.mmdb`

EF_PROCESSOR_ENRICH_IPADDR_MAXMIND_GEOIP_ENABLE

Set `EF_PROCESSOR_ENRICH_IPADDR_MAXMIND_GEOIP_ENABLE` to `true` to allow the collector to determine GeoIP attributes associated with a public IP address.

- Valid values: `true`, `false`
- Default value: `false`

EF_PROCESSOR_ENRICH_IPADDR_MAXMIND_GEOIP_PATH

If enrichment with GeoIP attributes is enabled using lookups in a Maxmind database ((`EF_PROCESSOR_ENRICH_IPADDR_MAXMIND_GEOIP_ENABLE` is `true`), this specifies the path to the Maxmind database.

- Default path: `/etc/flowdata/maxmind/GeoLite2-City.mmdb`

EF_PROCESSOR_ENRICH_IPADDR_MAXMIND_GEOIP_VALUES

If enrichment with GeoIP attributes is enabled using lookups in a Maxmind database (`EF_PROCESSOR_ENRICH_IPADDR_MAXMIND_GEOIP_ENABLE` is `true`), this setting specifies the GeoIP attributes from the Maxmind database to be included in the resulting record.

- Valid values:
  - `city`, `continent`, `continent_code`, `country`, `country_code`, `location`, `timezone`
- Default values: `city,country,country_code,location,timezone`

EF_PROCESSOR_ENRICH_IPADDR_MAXMIND_GEOIP_LANG

If enrichment with GeoIP attributes is enabled using lookups in a Maxmind database (`EF_PROCESSOR_ENRICH_IPADDR_MAXMIND_GEOIP_ENABLE` is `true`), this setting to specifies the language to be used for any language-specific values.

- Valid values
  - `de`: German
  - `en`: English
  - `es`: Spanish
  - `fr`: French
  - `ja`: Japanese
  - `pt-BR`: Brazilian Portuguese
  - `ru`: Russian
  - `zh-CN`: Simplified Chinese
- Default value: `en`

EF_PROCESSOR_ENRICH_IPADDR_MAXMIND_GEOIP_INCLEXCL_PATH

For more control of when enrichment is applied, you can include or exclude IP addresses from GeoIP enrichment by ASs or CIDRs. The `EF_PROCESSOR_ENRICH_IPADDR_MAXMIND_GEOIP_INCLEXCL_PATH` setting specifies the path to the `incl_excl.yml` file.

- Default setting: `''`
- Recommended path: `/etc/flowdata/hostname/incl_excl.yml`

For more details on the include/exclude functionality see "Scoping Enrichment with Include/Exclude" on page 1143.

EF_PROCESSOR_ENRICH_IPADDR_MAXMIND_GEOIP_INCLEXCL_REFRESH_RATE

The file specified in `EF_PROCESSOR_ENRICH_IPADDR_MAXMIND_GEOIP_INCLEXCL_PATH` can be loaded automatically to refresh values without restarting the collector. Use this setting to specify the refresh interval, in minutes, the file will take to reload.

- Default value: `15` (Note: when set to `0`, the refresh interval is not used).

EF_PROCESSOR_ENRICH_IPADDR_MAXMIND_GEOIP_INCLEXCL_REFRESH_RATE

The file specified in `EF_PROCESSOR_ENRICH_IPADDR_MAXMIND_GEOIP_INCLEXCL_PATH` can be loaded automatically to refresh values without restarting the collector. Use this setting to specify the refresh interval, in minutes, the file will take to reload.

- Default value: `15` (Note: when this value is set to `0`, the refresh interval is not used).

User-Defined Metadata

User-defined metadata adds additional information to a record for a given IP address. It can also be used to override existing fields. You can specify metadata for CIDR blocks, IP ranges or individual IP addresses.

EF_PROCESSOR_ENRICH_IPADDR_METADATA_ENABLE

Use this setting to enable or disable user-defined metadata enrichment. The default is `true`.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_ENRICH_IPADDR_METADATA_USERDEF_PATH

If the user-defined metadata enrichment is enabled (`EF_PROCESSOR_ENRICH_IPADDR_METADATA_ENABLE` is `true`), this setting specifies the path to the metadata file. If this value is undefined or empty, metadata enrichment is disabled.

For more information on user-defined metadata functionality, see: "User-Defined Metadata Enrichment" on page 1139.

- Default value: `''`

- Recommended path: `/etc/flowdata/metadata/ipaddrs.yml`

EF_PROCESSOR_ENRICH_IPADDR_METADATA_REFRESH_RATE

The file specified in `EF_PROCESSOR_ENRICH_IPADDR_METADATA_USERDEF_PATH` can be loaded automatically to refresh values without restarting the collector. This value specifies the refresh interval, in minutes, that the file will be reloaded. The value of `0` disables refreshing of the values.

- Default value: `15`

Network Interfaces

IN THIS SECTION

Option Records

The Apstra Flow collector will attempt to determine network interface attributes learned from Netflow v9 or IPFIX option records.

EF_PROCESSOR_ENRICH_NETIF_FLOW_OPTIONS_ENABLE

Setting this value to `false` will disable the enrichment of records with interface attributes learned from NetFlow or IPFIX options records.

- Valid values: `true`, `false`
- Default value: `true`

SNMP

Flow records generally include the indexes of ingress and egress interfaces by which the network traffic traversed the exporting device. The collector will attempt to determine the names, and attributes of these interfaces, as learned by polling the exporting device using SNMP.

EF_PROCESSOR_ENRICH_NETIF_SNMP_ENABLE

Use this setting to specify if SNMP polls are to be used to gather the network interface attributes.

- Valid values: `true`, `false`
- Default value: `false`

EF_PROCESSOR_ENRICH_NETIF_SNMP_PORT

If SNMP polling of attributes is enabled (`EF_PROCESSOR_ENRICH_NETIF_SNMP_ENABLE` is `true`), this setting specifies the UDP port that is used for such polls.

- Default UDP port: `161` (the default SNMP port number)

EF_PROCESSOR_ENRICH_NETIF_SNMP_VERSION

If SNMP polling of attributes is enabled (`EF_PROCESSOR_ENRICH_NETIF_SNMP_ENABLE` is `true`), this setting specifies the SNMP version that is used for such polls.

> **NOTE**: All network devices that are polled *must* support this version of SNMP.

Valid values:

- `1`: SNMPv1
- `2`: SNMPv2c
- `3`: SNMPv3

EF_PROCESSOR_ENRICH_NETIF_SNMP_COMMUNITIES

If SNMP polling of attributes is enabled (`EF_PROCESSOR_ENRICH_NETIF_SNMP_ENABLE` is `true`), this setting specifies the SNMP community strings that may be used for such polls. If a comma-separated list is specified, the collector will try each community in the order specified. Once a community returns a successful response, the collector remembers the community for future polls of the device.

> **NOTE**: All network devices polled *must* be configured to all visibility of collected attributes using this community. It may be necessary to specify a *view* associated with this community. See the documentation for your devices for help in determining the correct configuration steps.

- Example: `public,private,whatever`
- Default setting: `public`

EF_PROCESSOR_ENRICH_NETIF_SNMP_V3_USERNAME

Use this setting to specify the username used to authenticate the device using SNMPv3.

- Default setting: `''`

EF_PROCESSOR_ENRICH_NETIF_SNMP_V3_AUTHENTICATION_PROTOCOL

Use this setting to specify the authentication protocol used to authenticate the username with the device using SNMPv3.

Valid values:

- `noauth`, `md5`, `sha`, `sha224`, `sha256`, `sha384`, `sha512`
- Default value: `noauth`

EF_PROCESSOR_ENRICH_NETIF_SNMP_V3_AUTHENTICATION_PASSPHRASE

Use this setting to specify the authentication passphrase used to authenticate the username with the device using SNMPv3.

- Default passphrase: `''`

EF_PROCESSOR_ENRICH_NETIF_SNMP_V3_PRIVACY_PROTOCOL

Use this setting to specify the privacy protocol used to encrypt SNMPv3 traffic between the SNMP input and the device.

Valid values:

- `nopriv`, `des`, `aes`, `aes192`, `aes256`, `aes192c`, `aes256c`
- Default value: `nopriv`

EF_PROCESSOR_ENRICH_NETIF_SNMP_V3_PRIVACY_PASSPHRASE

Use this setting to specify the privacy passphrase used to encrypt SNMPv3 traffic between the SNMP input and the device.

- Default passphrase: `''`

EF_PROCESSOR_ENRICH_NETIF_SNMP_TIMEOUT

If SNMP polling of attributes is enabled (`EF_PROCESSOR_ENRICH_NETIF_SNMP_ENABLE` set `true`), this setting specifies the number of seconds to wait for the polled device to respond.

- Default value: `2`

EF_PROCESSOR_ENRICH_NETIF_SNMP_RETRIES

If SNMP polling of attributes is enabled (`EF_PROCESSOR_ENRICH_NETIF_SNMP_ENABLE` is `true`), this setting specifies the number of retries to attempt after the initial poll has timed out or otherwise fails. The timeout period is doubled for each retry.

- Default value: `1`

User-Defined Metadata

User-defined metadata allows you to add additional information to a record for a given network interface or to override existing fields.

EF_PROCESSOR_ENRICH_NETIF_METADATA_ENABLE

Use this setting to enable or disable user-defined metadata enrichment. The default value is `true`.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_ENRICH_NETIF_METADATA_USERDEF_PATH

If user-defined metadata enrichment is enabled (`EF_PROCESSOR_ENRICH_NETIF_METADATA_ENABLE` is `true`) this setting specifies the path to the metadata file. If this value is undefined or empty, metadata enrichment is disabled.

For more details on user-defined metadata, see "User-Defined Metadata" on page 1138.

- Default setting: `''`
- Recommended path: `/etc/flowdata/metadata/netifs.yml`

EF_PROCESSOR_ENRICH_NETIF_METADATA_REFRESH_RATE

The file specified in `EF_PROCESSOR_ENRICH_NETIF_METADATA_USERDEF_PATH` can be loaded automatically to refresh values without restarting the collector. This value specifies the refresh interval, in minutes, that the file will be reloaded.

- Default value: `15` (The value of `0` disables refreshing of the values).

Community/Conversation IDs

EF_PROCESSOR_ENRICH_COMMUNITYID_ENABLE

Use this setting to specify if flow records should be enriched with a Community ID value.

> **NOTE**: For more information about community IDs see the community-id-spec.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_ENRICH_COMMUNITYID_SEED

This setting is a 16-bit value used as the seed for determining the Community ID of a flow record.

- Default value: `0`

EF_PROCESSOR_ENRICH_CONVERSATIONID_ENABLE

Use this setting to enable or disable flow records enriched with a Conversation ID value. This value is similar to a community ID, however rather than being based on the SRC/DST relationship of two endpoints, this value is based on the client/server perspective. Although multiple unique sessions (such as a unique client-side port for each session) have their own Community ID, they share the same Conversation ID. This setting allows for greater flexibility when exploring a complex flow dataset.

- Valid values: `true`, `false`
- Default value: `true`

EF_PROCESSOR_ENRICH_CONVERSATIONID_SEED

This setting is a 16-bit value used as the seed for determining the conversation ID of a flow record.

- Default value: `0`

SEE ALSO

YAML Configuration Files | **1158**

Common Options | **1159**

CHAPTER 40

API

IN THIS CHAPTER

API Endpoints Options

An API endpoint is a specific location within an API that accepts requests and sends back responses. This is a way for different systems and applications to communicate with each other, by sending and receiving information and instructions through the endpoint.

See "Metrics" on page 1150 for a list of API endpoint common the Apstra Flow collector.

CHAPTER 41

Additional Documentation

IN THIS CHAPTER

Configure sFlow and NetFlow on Junos OS Devices

IN THIS SECTION

This topic describes how to configure sFlow and NetFlow on Juniper switches.

Configure sFlow on a Juniper EX or QFX Switch

To configure sFlow on a Juniper EX or QFX series switch, follow these steps:

1. Access the switch CLI.

   Connect to your Juniper EX or QFX switch through SSH or a console cable. If you are connecting through SSH, use a tool like PuTTY or the built-in SSH client in your terminal. Then enter the switch's IP address, username, and password to log in.

2. Enter configuration mode.

```
configure
```

3. Configure the sFlow settings.

```
set protocols sflow agent-id AGENT_IP_ADDRESS
set protocols sflow collector x.x.x.x udp-port yyyy
set protocols sflow polling-interval POLLING_INTERVAL
set protocols sflow sample-rate SAMPLE_RATE
set protocols sflow interfaces INTERFACE_NAME
```

   Specify the sampling rate, polling interval, and IP address and port of the remote flow collector. For example:

   - `AGENT_IP_ADDRESS`: IP address of the sFlow agent (typically the switch's management IP address).
   - `x.x.x.x`: Apstra Flow collector's IP address.
   - `yyyy`: Apstra Flow collector's port number.
   - `POLLING_INTERVAL`: Enter the desired polling interval in seconds (e.g 30 sec.) and desired `SAMPLE_RATE` (for example, 1024 for 1 in 1024 packets).
   - `INTERFACE_NAME`: Name of the interface you want to monitor (for example, `ge-0/0/0`). You can configure multiple interfaces.

4. Commit and save your changes.

   ```
   commit save
   ```

5. Exit configuration mode.

   Type `exit` to leave configuration mode and return to the Juniper EX or QFX switch CLI.

6. Verify your configuration by entering the following commend:

   ```
   show sflow
   ```

   This command displays the sFlow settings you just configured.

Your Juniper EX or QFX series switch will now start exportinging Apstra Flow to the Apstra Flow collector.

Configure Flow Sampling on Juniper Routers

You can configure Juniper routers to export flow records using Netflow v9. The NetFlow version 9 flow template enables you to define a flow record template suitable for IPv4 traffic, IPv6 traffic, MPLS traffic, a combination of IPv4 and MPLS traffic, or peer AS billing traffic.

NOTE: We recommend using Netflow v9, rather than IPFIX, for flow export from Juniper devices. IPFIX records from Juniper include only *total* counters for bytes and packets, rather than

the defacto standard *delta* counters. Most flow collection solutions work better with delta values, which are provided by Juniper devices using Netflow v9.

You can enable both `input` (ingress) and `output` (egress) directions.

To configure flow sampling on a Juniper router:

1. Create an instance, as shown in the following example.

```
user@router# set chassis fpc 0 sampling-instance FLOWDATA
```

2. Configure the size of the flow table.

   Starting with Junos OS Release 15.1F2, by default, the software allocates one 1K IPv4 flow table. If desired, you can allocate up to 15 256K IPv4 flow tables using the following command:

```
user@router# set chassis fpc inline-services flow-table-size ipv4-flow-table-size 15
```

   The maximum supported flow table size for a combination of both IPv4 and IPv6 is 15. For example, you can set the flow table size for IPv4 to 10 and set the size for IPv6 to 5.

```
user@router# set chassis fpc 0 inline-services flow-table-size ipv4-flow-table-size 10
user@router# set chassis fpc 0 inline-services flow-table-size ipv6-flow-table-size 5
```

   NOTE: The flow table size recommended by Juniper is 4 (4 x 256K flows), which equates to 1 million flows. You can configure a larger size, however the system will issue a warning message.

   To simplify the sizing of flow tables, the MX series supports a `flex-flow-sizing` option that does not require a manual sizing between IPv4 tables and IPv6 tables. Rather than using the `flow-table-size` command, specify the following configuration:

```
user@router# set chassis fpc 0 inline-services flex-flow-sizing
```

You can run the following command to determine if flows are being dropped, and to determine if any adjustments to the flow table sizes are required:

```
user@router# show services accounting errors inline-jflow fpc-slot 0 | match "Flow Creation
Failures"
    Flow Creation Failures: 1146233714
    IPv4 Flow Creation Failures: 1111175982
    IPv6 Flow Creation Failures: 35057732

user@router# show services accounting errors inline-jflow fpc-slot 0 | match "Flow Creation
Failures"
    Flow Creation Failures: 1146234132
    IPv4 Flow Creation Failures: 1111176365
    IPv6 Flow Creation Failures: 35057767
```

3. Configure the service to extended flow memory. This service provides more scale in flows for inline services sampling.

```
user@router# set chassis fpc 0 inline-services use-extended-flow-memory
```

4. Add the template configuration for both IPv4 (`ipv4-template`) and IPv6 (`ipv6-template`).

```
user@router# set services flow-monitoring version9 template ipv4 ipv4-template
user@router# set services flow-monitoring version9 template ipv6 ipv6-template
```

   a. Set the `flow-active-timeout` and `flow-inactive-timeout` determine how frequently flow records will be sent for metered flows.

```
user@router# set services flow-monitoring version9 template ipv4 flow-active-timeout 60
user@router# set services flow-monitoring version9 template ipv4 flow-inactive-timeout 60
user@router# set services flow-monitoring version9 template ipv6 flow-active-timeout 60
user@router# set services flow-monitoring version9 template ipv6 flow-inactive-timeout 60
```

   b. Add the `vlan-id` to the `flow-key` to include VLAN IDs in both the ingress and egress directions.

```
user@router# set services flow-monitoring version9 template ipv4 flow-key vlan-id
user@router# set services flow-monitoring version9 template ipv6 flow-key vlan-id
```

5. Set the rate at which packets will be sampled.

```
user@router# set forwarding-options sampling instance FLOWDATA input rate 128
```

6. Specify where the flow records should be sent for both IPv4 and IPv6 templates.

   You must specify both the IP address and port number on which the Apstra Flow collector is listening, as well as the flow record version.

```
ser@router# set forwarding-options sampling instance FLOWDATA family inet output flow-server
192.0.2.11 port 9995
user@router# set forwarding-options sampling instance FLOWDATA family inet output flow-server
192.0.2.11 version9 template ipv4
user@router# set forwarding-options sampling instance FLOWDATA family inet6 output flow-
server 192.0.2.11 port 9995
user@router# set forwarding-options sampling instance FLOWDATA family inet6 output flow-
server 192.0.2.11 version9 template ipv6
```

7. Specify the IP address from which the device will send the packets containing the flow records.

```
user@router# set forwarding-options sampling instance FLOWDATA family inet output inline-
jflow source-address 192.0.2.222
user@router# set forwarding-options sampling instance FLOWDATA family inet6 output inline-
jflow source-address 192.0.2.222
```

8. Enable sampling for each interface for which traffic should be observed. You can enable both `input` and `output` (ingress and egress) directions.

```
user@router# set interfaces xe-0/1/1 unit 110 family inet sampling input
user@router# set interfaces xe-0/1/1 unit 110 family inet sampling output
```

9. Commit your configuration.

```
user@router# commit
commit complete
```

   The Apstra Flow collector must first receive the template records from the Juniper device, after which it will decode and process the version 9 records. After a few minutes, you'll see data in the data platform to which the collector is configured to send it.

Configure the hsflowd sFlow Agent

This topic describes how to configure `hsflowd`. `hsflowd` is an open-source host sFlow agent designed to monitor servers, VMs, and containers. `hsflowd` provides resource usage statistics, performance metrics, and network traffic data by leveraging the sFlow standard.

To configure `hsflowd`:

1. Install the `hsflowd` package.

   The installation process varies depending on your OS. For example, on a Debian-based system like Ubuntu, you can use the following commands:

   ```
   sudo apt-get update
   sudo apt-get install hsflowd
   ```

   For other systems, you might need to download and compile the source code from the GitHub Repository.

2. After you complete the installation, locate the `hsflowd` configuration file at: `/etc/hsflowd.conf`. Then, open the file with a text editor such as `nano` or `vi`.

3. Configure the sFlow settings.

   Replace the `AGENT_IP_ADDRESS` with the IP address of the sFlow agent (typically the host's IP address) and `x.x.x.x` with the IP address of your Apstra Flow collector. You can adjust the sampling, polling, header, and datagram values as needed.

   ```
   <sFlow> <sFlowSettings> <sampling>400</sampling> <polling>20</polling> <header>128</header> <datagram>1400</
   datagram> <agent>AGENT_IP_ADDRESS</agent> </sFlowSettings> <collectors> <collector> <ip>x.x.x.x</ip>
   <udpport>6343</udpport> </collector> </collectors> </sFlow>
   ```

4. Save and exit the configuration file.

5. To apply the changes, restart the `hsflowd` service. The command you use varies depending on your OS. For a Debian-based system, such as Ubuntu, run the following command:

   ```
   sudo systemctl restart hsflowd
   ```

6. Verify the configuration.

   To verify that hsflowd is running and exporting Apstra Flow, enter the following command:

   ```
   sudo systemctl status hsflowd
   ```

   This command displays the status of the hsflowd service, indicating that the service is running and active.

You server will now start exporting sFlow data to the specified flow collector.

Generate a Support Bundle

IN THIS SECTION

The support bundle allows you to generate a compressed TAR file containing relevant data (logs, configs, and metric data) for troubleshooting or analysis. If needed, you can send these bundled files to your Apstra support team to help diagnose issues with the collector.

To use the support bundle command line tool, run the `flowcoll` command with the `--support-bundle` or `-s` flag as shown in the following examples:

Basic Example

```
sudo /usr/share/juniper/bin/flowcoll -s
```

Advanced Example

```
sudo /usr/share/juniper/bin/flowcoll -s -sc /my/config/dir -sl /my/log/dir -st 3 -si 3000
```

When the command runs successfully, the tool generates a compressed TAR file in the directory `/home` named similarly to `ef_support_bundle-20230831T164759.tar.gz`.

Command Line Options

If you do not specify an option, its specified default value is used as shown in the following table:

Table 50: CLI Default Options

Option	Shorthand	Default Value	Description
`support-bundle`	`-s`	`false`	Enables support bundle mode.

Table 50: CLI Default Options *(Continued)*

Option	Shorthand	Default Value	Description
support-bundle-config-dir	-sc	/etc/flowcoll	The path to the collector's configuration directory.
--support-bundle-logs-dir	-sl	/var/log/flowcoll	The path to the collector's log directory.
--support-bundle-metrics-interval	-si	1000	The interval (in ms) that metrics are collected.
--support-bundle-metrics-times	-st	1	The numbers of times metrics are collected.
--support-bundle-output	-so	<Working directory>	The path where the output file is written to.

NOTE: Adjusting the collection interval (-si) and times (st), makes it easier to track and spot trends in metrics over time.

Generate Support Bundle by Endpoint

You can generate a support bundle by using the following methods:

- HTTP Method: POST
- URL: /support-bundle

Request Body

All fields in the request boy are optional. The defaults are used if none are specified.

- **logDirPath** (string): Directory path of the log files. The default path is: /var/log/juniper/flowcoll
- **configDirPath** (string): Directory path of the configuration file. The default path is: /etc/juniper

Query Parameters

- **Interval** (integer): Interval at which the metrics are fetched (in ms).The default interval is: 1000 ms.

- **Times** (integer): The number of times the metrics endpoint is fetched. The default is `1`.

Authentication: This endpoint supports basic authentication *only* if the collector is specifically configured for it. For configuration details, see the API options under the "Common Options" on page 1159 section.

Examples of Support Bundles by Endpoint

Basic Example

Using the default settings, the following example shows a basic example without any query parameters or request body.

```
curl -X POST \
  -H "Content-Type: application/json" \
  -O -J \
  http://localhost:8080/support-bundle
```

Advanced Example

The following example shows an advanced `curl` request that includes the request body and query parameters.

```
curl -X POST \
  -H "Content-Type: application/json" \
  -d '{
    "logDirPath": "/path/to/log/dir",
    "configDirPath": "/path/to/log/dir"
  }' \
  -O -J \
  http://localhost:8080/support-bundle?interval=2000&times=2
```

Responses

Table 2 on page 1220 shows the codes and responses you might see when you generate a support bundle by endpoint.

Table 51: Support Bundle Code and Responses

Code	Reason	Description
200	OK	A successful response returns the support bundle file for download. The file has the following naming convention: `ef_support_bundle-YYYYMMDDTHHmmss.tar.gz`, where `YYYYMMDDTHHmmss` is a timestamp that indicates the time the bundle was created.
400	Bad request	Indicates that the query parameters are invalid.
500	Internal Server Error	An internal server error occurred while processing the request.

CHAPTER 42

Knowledge Base

IN THIS CHAPTER

Installation

Problem: .deb Upgrade Fails File Overwrite

Upgrading the Apstra Flow collector to the latest version on Debian-based distributions, such as Ubuntu, reports an error when trying to overwrite a file.

You can safely overwrite package files. Other files, that contain user-defined data or parameters, require you to restore your backup files after the upgrade.

NOTE: In some cases, the `/etc/juniper/app/appid.yml` file cannot be overwritten.

Solution

1. Backup your `/etc/juniper` directory.
2. Use the command `--force-overwrite` when running the install command.

Configuration

SUMMARY

This topic lists the configuration errors you might see when configuring Apstra Flow.

IN THIS SECTION

CA Certificate Path Incorrect

The collector's log indicates that the certificate file path for an output is incorrect.

- **Symptom**: The collector's log indicates a message similar to the following:

  ```
  {"level":"panic","ts":"2023-08-25T11:34:48.953Z","logger":"flowcoll","caller":"opensearch/
  instance_registration.go:33","msg":"failed to instantiate config","code":"opensearch/conf-
  error","reason":"ENV: 'EF_OUTPUT_OPENSEARCH_TLS_CA_CERT_FILEPATH' Value: '/root/http_ca.crt'
  Error: failed 'file_if_set' validation"
  ```

  Note the message: `Error: failed 'file_if_set' validation`.

- **Problem**: The collector cannot find a file at the path that is specified for the output. For OpenSearch, the output is: `EF_OUTPUT_OPENSEARCH_TLS_CA_CERT_FILEPATH`. If this setting is not blank, you must set it to a valid certificate file or the collector will not run.

- **Solution**:

  Set

  ```
  EF_OUTPUT_OPENSEARCH_TLS_CA_CERT_FILEPATH
  ```

  or

  ```
  EF_OUTPUT_OPENSEARCH_TLS_SKIP_VERIFICATION
  ```

  to the full path of a valid certificate file.

OpenSearch Authentication Failure

The collector's log indicates `failed to bootstrap opensearch` and `unable to authenticate user [<username>] for REST request`.

- **Symptom**: The collector's log indicates a message similar to the following:

```
2023-09-23T18:05:19.604Z        error   bootstrapper[opensearch]     opensearch/
bootstrap.go:147  failed to bootstrap opensearch. retrying...  {"code": "opensearch/bootstrap-
failure", "reason": "error while creating default ilm policy - GET ism policy error for ism
policies 'network'- status code 401 not expected - {\"error\":{\"header\":{\"WWW-Authenticate
\":[\"Basic realm=\\\"security\\\" charset=\\\"UTF-8\\\"\",\"Bearer realm=\\\"security\\
\"\",\"ApiKey\"]},\"reason\":\"unable to authenticate user [xxxxZZopen] for REST request [/
_plugins/_ism/policies/network]\",\"root_cause\":[{\"header\":{\"WWW-Authenticate\":[\"Basic
realm=\\\"security\\\" charset=\\\"UTF-8\\\"\",\"Bearer realm=\\\"security\\\"\",\"ApiKey
\"]},\"reason\":\"unable to authenticate user [xxxxZZopen] for REST request [/_plugins/_ism/
policies/network]\",\"type\":\"security_exception\"}],\"type\":\"security_exception
\"},\"status\":401}"}
github.com/juniper/flowcoll/pkg/outputs/opensearch.(*Bootstrap).Run
        /tmp/flowcoll/pkg/outputs/opensearch/bootstrap.go:147
github.com/juniper/flowcoll/pkg/outputs/opensearch.NewCreateInstanceFunc.func1
        /tmp/flowcoll/pkg/outputs/opensearch/instance_registration.go:155
github.com/juniper/flowcoll/pkg/instantiator.(*Instantiator).Run
        /tmp/flowcoll/pkg/instantiator/instantiator.go:79
```

- **Problem**: The collector's OpenSearch output is unable to authenticate with the OpenSearch host(s) specified in `EF_OUTPUT_OPENSEARCH_ADDRESSES`.
- **Solution**: Verify that you entered your username (`EF_OUTPUT_OPENSEARCH_USERNAME`) and password (`EF_OUTPUT_OPENSEARCH_PASSWORD`) correctly. You can test the username and password manually using `curl` command. For example:

```
curl -XGET https://127.0.0.1:9200/_cat/indices -u username:password --insecure
```

Operation

IN THIS SECTION

Flow Collector Queues 90 Percent Full

The Apstra Flow collector's log reports the errors, `processor to output writer` or `UDP Server to Flow Decoder` are 90 percent full. For example:

```
{"level":"info","ts":"2023-08-07T08:08:14.301Z","logger":"flowcoll","caller":"flowprocessor/
metrics.go:118","msg":"flow processor to output writer is 90% full. This is normal when the
collector is starting. If it persists for hours, it may indicate that you are at your license
threshold or your system is under-resourced."}
```

```
{"level":"info","ts":"2023-08-07T08:08:34.264Z","logger":"flowcoll","caller":"server/
metrics.go:125","msg":"UDP Server to Flow Decoder is 90% full. This is normal when the collector
is starting. If it persists for hours, it may indicate that you are at your license threshold or
your system is under-resourced."}
```

You might see these logs accompanied by `throttle` logs.

```
2023-06-28T21:20:21.821Z        warn    throttle/restricted_throttle.go:105     [throttler]:
start burst
2023-06-28T21:20:41.822Z        warn    throttle/restricted_throttle.go:111     [throttler]:
stop burst
2023-06-28T21:20:41.822Z        warn    throttle/restricted_throttle.go:117     [throttler]:
start recovery
2023-06-28T21:50:42.142Z        warn    throttle/restricted_throttle.go:123     [throttler]:
stop recovery
```

Problem

These messages usually occur when the collector first starts, as various internal processes might not be fully initialized. However, if the messages persist after the first few minutes, one of the following issues might exist:

- ONLY `flow processor to output writer`: Indicates that the system, which data is being output, lacks sufficient performance to ingest records at the rate being sent by the Apstra Flow collector. This can caused by insufficient CPU, memory, disk space, or excessive disk latency. Insufficient network bandwidth between the collector and target system can also cause the problem.
- BOTH `UDP Server to Flow Decoder` and `flow processor to output writer`: This is a further progression of the previous condition. The resulting back pressure from the slow downstream system is now likely causing data to be lost.
- ONLY `UDP Server to Flow Decoder`: The internal decoder/processor workers cannot keep up with the rate of records being received. This issue can be caused by one of the following conditions:
  - More records are being received than are allowed by the license. If so, `throttler` messages will also appear in the log.
  - The collector has insufficient resources, primarily CPU cores, to process the rate of records being received.
  - The caches, for IP addresses, interfaces, and so on, have yet to be "warmed up" so the related high latency enrichment tasks are limiting throughput.

> **NOTE**: Increasing the output pool size manually, through `EF_PROCESSOR_POOL_SIZE` often solves this issue.

Solution

The solution varies depending on the issue, as described in the problem section above.

- - ONLY `flow processor to output writer`: Increases the performance of the system to which records are being sent.
  - BOTH `UDP Server to Flow Decoder` and `flow processor to output writer`: Increases the performance of the system to which records are being sent.
  - ONLY `UDP Server to Flow Decoder`:
    - Increase the CPU cores available to the collector.
    - If the collector has sufficient CPU resources try increasing the processor pool size by setting the `EF_PROCESSOR_POOL_SIZE`. This allows great concurrency of high latency enrichment tasks.

> **NOTE**: If `throttler` messages appear in the log, contact your Juniper sales representative about subscription options that allows you to collector additional flow records.

Dashboard Updates

Question:

When upgrading the OpenSearch dashboards, is it necessary to also update the Apstra Flow dashboards?

Answer:

The Apstra Flow dashboards should continue to work after upgrading the OpenSearch dashboards. If However, if you want to use the latest dashboards, you can usually overwrite the existing saved objects. You can also delete the existing objects that were saved before importing the latest.

> **NOTE**: When overwriting or deleting your existing saved objects, any changes you made previously are lost. We recommended that you first export any customized visualizations, saved searches, or dashboards. You can then re-import these items as necessary.

Change the Apstra Flow Indexes Names

Question:

When using the OpenSearch outputs, can the names of the indexes be changed?

Answer:

No, changing the names of Apstra Flow-related indexes is not supported. The various components of the Apstra Flow solutions are designed to work together in an integrated manner. Changing the indexes names can potentially break dashboards, index state management (ISM) policies, machine-learning jobs and alerts. Although you can use an ingest pipeline to change the indexes names as records are ingested,Apstra Flow does not support this type of environment.

Often the reason for changing indexes names is to achieve multi-tenancy, separate indexes for different environments, locations or other organizational units. To facilitate this use-case, use the `EF_OUTPUT_OPENSEARCH_INDEX_SUFFIX` option to add a suffix to the index name.

Consider the index name `juniper-flow-codex-2.2-2023.01.01`.

For example, if you set the suffix value to `staging`, the resulting index name will be called: `juniper-flow-codex-2.2-staging-2023.01.01`. You can then control access to the `staging` indexes by setting permissions for the `*-staging-*` index naming pattern.

Network Flows

SUMMARY

This

IN THIS SECTION

Configure the UDP Input

Question: Can the Apstra Flow collector's port be changed?

Answer: The collector receives packets containing flow records using UDP. Table 1 on page 1227 lists the three configurable parameters:

Table 52: UDP Input Parameters

UDP Parameters	Description
"EF_FLOW_SERVER_UDP_PORT" on page 1187	The UDP port that the Apstra Flow collector listens to for NetFlow/IPFIX/sFlow packets.
"EF_FLOW_SERVER_UDP_IP" on page 1187	The IP addresses that the UDP socket is bound to on the Apstra Flow collector.
"EF_FLOW_SERVER_UDP_READ_BUFFER_MAX_SIZE" on page 1188	The UDP receive buffer for the system. If this value exceeds the maximum allowed buffer size (`net.core.rmem_max`) on Linux, the maximum allowed size is used.

Flow Records Not Received

Problem

Flow exporters are configured to output IPFIX, sFlow, or NetFlow, but one or more flow exporters' data does not appear in the Apstra Flow dashboards.

There can be several reasons for this:

- The packets carrying the expected flow records are not arriving at the system running the flow collector.
- The packets carrying the expected flow records are not arriving on a UDP port on which the collector is not listening.
- The Linux firewall is blocking the packets from reaching the collector.

Solution

Verify the packets are arriving.

Use `tcpdump` to verify that the packets carrying the expected flow records are arriving at the interface where the collector is listening. For example, if the collector is listening on UDP port `2055`(`EF_FLOW_SERVER_UDP_PORT`) the following `tcpdump` command shows the incoming packets to this port:

```
sudo tcpdump "udp port 2055
```

To see packets from a specific exporter, you can also specify the exporter's IP address. For example, if packets are expected from `192.0.2.11`, use the following command:

```
sudo tcpdump "src 192.0.2.11 and udp port 2055"
```

NOTE: You might need to specify the interface on which `tcpdump` observes the incoming packet. You can do this by specifying the `-i` option in the `tcpdump` command. For example:

```
sudo tcpdump -i eth0 "src 192.0.2.11 and udp port 2055"
```

If you do not receive any packets, this can mean that:

- The device is not sending the packets.
- The packets are being sent to the wrong place.
- The packets are being blocked along the way, e.g. by a firewall

You will need to troubleshoot and fix this issue before proceeding.

Verify the collector is receiving the packets.

Verify that the collector is receiving the packets from the operating system, by running the collector with `debug` (`EF_LOGGER_LEVEL`) set to `debug`. A message should appear indicating when the packets were received and from which IP addresses the packets were sent.

If you have verified that the packets are arriving at the system, but you do not see any messages in the collector's logs, the packets are likely being blocked by the Linux firewall. You can temporarily disable the Linux firewall to confirm this. If after doing this the logs indicate that packets are received, you will need to reconfigure the Linux firewall to allow the traffic to reach the collector.

Unsupported sFlow Structure

The log indicates that the Apstra Flow collector cannot process an sFlow record because it has enterprise-specific information that is not supported. For example:

```
{"level":"error","ts":"2023-06-09T02:50:20.427Z","logger":"flow_processor","caller":"flowprocesso
r/flow.go:75","msg":"failed to process record","code":"processor/process-record-
error","reason":"sFlow v5: could not decode samples: flow struct not supported - enterprise:
25506, format: 1003","stacktrace":"g
```

Problem: The collector received an sFlow structure it does not recognize. This is usually due to a vendor sending its own enterprise-specific structure.

Solution: To add support for a specific sFlow Structure, contact Juniper support. You will need to supply a PCAP of the records that contain the structure and documentation from the vendor about the contents of the structure.

Netflow v9/IPFIX Template Not Received

Problem: The Apstra Flow collector's log displays the error: `Could not decode flowsets: template not yet received`. This issue applies to both NetFlow v9 and IPFIX templates.

The Apstra Flow collector indicates a message similar to the following:

```
error netflow9/netflow9.go:59        netflow v9: could not decode flowsets: template not yet
received from 10.1.1.1 for session: 27856, observation domain: 33312, template ID 260
```

Solution:

In most cases, waiting allows the issue to resolve itself. You'll usually see these messages when starting the collector, however these messages should stop after the needed templates are received. Devices usually send templates every few minutes, although some may take 15-30 minutes. This interval is usually configurable, but may vary by vendor and model.

If waiting does not solve the problem, contact your Juniper sales representative. To investigate your issues, we'll need a PCAP of the incoming records from the device in question. The PCAP will need to be long enough to include templates.

In the following example, `tcpdump` is configured to capture incoming packets to port `2055` from `192.0.2.11` and write them to a file named `netflow.pcap`.

```
sudo tcpdump "src 192.0.2.11 and udp port 2055" -w netflow.pcap -vvv
```

Bidirectional Flow Support

Question: How does Apstra Flow handle a flow exporter that supports bidirectional flow records (RFC 5103), where two directions of traffic are expressed in a single record?

Answer: The collector produces two unidirectional records, one for each direction. This allows the bidirectional flow records to be processed and analyzed in the same manner as unidirectional flows.

14

PART

Analytics - Exploratory Analytics

QBA Exporatory Interface

SUMMARY

Juniper Apstra release 5.0 introduces an exploratory analytics interface that enables you to explore and visualize data sets using the Query-Based Analytics (QBA) language. This feature enables you to analyze large sets of data and discover correlations or previously unknown data patterns about key aspects of your network infrastructure.

IN THIS SECTION

NOTE: This feature has been classified as a Juniper Apstra Technology Preview feature. These features are "as is" and voluntary use. Juniper Support will attempt to resolve any issues that customers experience when using these features and create bug reports on behalf of support cases. However, Juniper may not provide comprehensive support services to Tech Preview features.

For additional information, refer to the "Juniper Apstra Technology Previews" on page 1781 page or contact "Juniper Support" on page 1374.

Query-Based Analytics Overview

Juniper Apstra release 5.0 enhances IBA with the introduction of a Query-Based Analytics (QBA) interface. QBA is a unified-query interface for interacting with various data sources within Apstra. Apstra generates various kinds of data about the Datacenter (DC). This data is stored in different databases, either in-memory or on-disk. The interface lets you query and associate data from different sources for troubleshooting or data analytics.

The QBA Exploratory Interface is a GUI tool that lets you perform your own data analysis with powerful queries and data visualization options using QBA. You can use the exploratory interface to systematically build queries, then choose an intuitive visualization option. After Apstra visualizes the data, you can experiment with various transformations (statistical operators) on the data to shape and manipulate the data as you see fit.

For example, let's say you're interested in the average TX utilization of a switch. Using the interface, you can visualize this metric in many different aggregation intervals, from a maximum of 30 days all the way down to the minute. You can limit your queries to certain interfaces and system IDs (system_id). After

you submit the query, the query visualization tab on the right-hand side of the page populates. You can select from the following visualization options:

- Line Chart: Visualizes changes in numeric values over intervals of time. Data points are connected with straight lines.
- Area Chart: Visualizes changes in numeric values over intervals of time. Similar to a line chart, but the space between the x-axis and the plotted line is filled in.
- Stacked Area Chart: Visualizes changes in groups of data over intervals of time. Similar to an area chart, except the data for each group is displayed, or stacked, on top of each other.
- Scatterplot: Visualizes the relationship between two numeric values. The x-axis represents the value of the first data point, and the y-axis represents the value of the second data point.
- Pie Chart: Visualizes percentages (portions) of data in relation to a whole value (100%).
- Donut Chart: Visualizes the value of different portions of data in relation to a whole.

You can also view the data with the following output options:

- Table
- Time-Series Table
- JSON

After selecting a visualization method, you can manipulate the chart by selecting system ID's of devices you are interested in. You can also choose different x-axis and y-axis options. Selecting the Show Trend Line option plots the points of the chart with a line for better trend visualization.

Query Sources

IN THIS SECTION

With the exploratory interface, you can query different types of data from different sources across your managed devices.

MetricDb

MetricDb is a Juniper Apstra proprietary time-series database. Apstra components such as IBA, Audit, Anomaly, and System-Resource-Monitoring use MetricDb to store time-series/historic data about various aspects of the DC. This data includes network-traffic features (i.e. interface-counters, ecmp-imbalance etc.), device-health features (i.e. CPU usage), memory usage, temperature, power, transceiver stats, Controller features (i.e. audit, anomalies etc). When querying MetricDb, the 'locator' attribute specifies which of the previously mentioned graph sources to use. We recommend that you use 'operational' graph for QBA queries.

Graph

Apstra Graph database (Blueprint) is a knowledge-base of the running-state of the DC. Graph represents a single source of truth regarding infrastructure, policies, constraints etc. Apstra maintains the following copies of the graph:

- Config: Captures user intent in the form of security policies, preferred routes, network requirements etc.
- Staging: Represents the config graph plus the translation of the intent into automated administrative tasks across the DC.
- Operational: Captures the current running state of the DC.

The following is an example graph query:

```
graph_query = {
    "name": "graph",
    "from": {
        "source": "persisted_sysdb_files",
        "locator": ["{}/{}".format(base_dir, sysdb_stem),
                    blueprint_id,
                    "operation"]
    },
    "filter": "node('system', name='system', role=is_in(['leaf', 'spine']),
deploy_mode='deploy').out('part_of_pod').node('pod', name='pod')",
    "select": {
        "keys": {
            "system_id": ["system", "system_id"],
        },
        "data": {
            "label": ["system", "label"],
            "role": ["system", "role"],
```

```
            "deploy_mode": ["system", "deploy_mode"],
            "pod_label": ["pod", "label"],
        }
    }
}
```

Note that the 'locator' parameter generally refers to "operational" graph. For more information about graph, see Graph.

As an example, let's build a query that fetches data from these two data sources. Let's say we want to improve the current DC inventory of optical transcievers, and gain insight into the most efficient maintenance windows for optical transceivers. We suspect that by visualizing temperature trends of the transceivers, we may be able to learn more and leverage this knowledge to anticipate failures or compare vendors.

We don't necessarily have a clear idea of what we're looking for, but we want to explore the data. We know that the Apstra IBA "Optical-transceiver" probe monitors various stats like temperature, power etc. from all optical transceivers in the DC. Within the stage of this probe, metricDb is enabled and collecting historical data. We can use the '2min-avg' stage metric to collect 'model', 'vendor', and 'temperature' data. Additionally, we can customize this data fo rour needs using 'hostname', 'role', 'label', and 'ipv4_addr' information about the system from Graph.

By combining these two data sources, we can classify the data per (system, interface) pair and plot a the trend of temperature variables over the last week, month, or year, based on data availability. The Exploratory interface tool is the GUI-based builder for queries like this. The example below shows this query in JSON format.

Advanced users might choose to bypass the Exploratory interface and make direct updates to the query instead. In such cases, you can run queries manually using '/api/qba/query' rest-api endpoint. Note that this query comprises three subqueries:

- Subquery for MetricDb metric 'Optical probe, 2min-avg' stage
- Graph subquery
- Final subquery that combines the prior two subquery results to form the final, aggregated results

This query output can be plotted as a line graph to visualize trends. After the data is visualized, you can explore the information as you see fit. You might compare the data based on vendor, or any number of optical transceiver stats.

```
query = [
    # metric db query
    {
        # sub query can be referenced by name
```

```
        "name": "xcvr_metricdb",
        "from": {
            "source": "metricdb",
            "locator": ["iba", "sf-pod", "xcvr_probe", "2min-avg"],
            "between": {
                "begin": "2022-08-20T05:00:00",
                "end": "2022-08-27T05:00:00",
                "aggregation_interval": 3600
            }
        },
        "select" : {
            "keys": {
                "system": "system",
                "interface": "interface",
            },
            "data": {
                "model": "model",
                "vendor": "vendor",
                "temp": "temperature"
            }
        },
        "filter": [
            "or",
            ["=", "vendor", "juniper"],
            ["=", "vendor", "arista"]
        ],
    },

    # graph query
    {
        # sub query can be referenced by name
        "name": "graph",
        "from": {
            "source": "graph",
            "locator": ["sf-pod", "operational"]
        },
        "select" : {
            "keys": {
                "system_id": ["@", "system", "system_id"],
                "ifc_name": ["@", "interface", "if_name"]
            },
            "data": {
                "hostname": ["@", "system", "hostanme"],
```

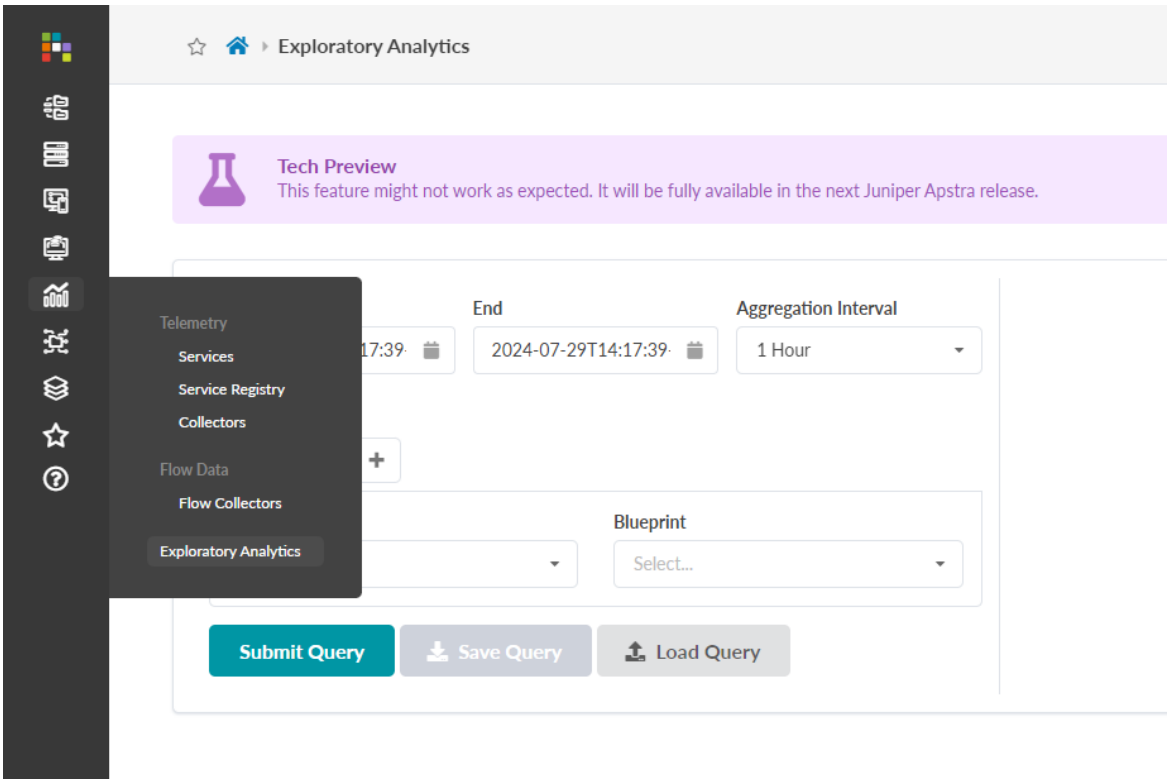
```
                "role": ["@", "system", "role"],
                "label": ["@", "interface", "label"],
                "ipv4_addr": ["@", "interface", "ipv4_addr"]
            }
        },
        "filter": "node('system', name='system', system_id=not_none())."
              "out('hosted_interfaces')."
              "node('interface', name='interface')",
        "ordered": [
            {"system_id": "ascending"},
            {"ifc_name": "ascending"}
        ]
    },

    # To combine the above two query results, and show how
    # each interface's temperature is trending using simple
    # linear regression. And sort the result in descending
    # order, i.e. the one with faster rising temperature
    # is placed in the front.
    {
        "from": [
            {   # result of xcvr metricdb probe query
                "source": "query",
                "name": "xcvr_metricdb"
            },
            {   # result of graph query
                "source": "query",
                "name": "graph",
                "join": {
                    "system_id": "system",
                    "ifc_name": "interface"
                }
            }
        ],
        "select" : {
            "keys": {
                "system_id": ["@", "xcvr_metricdb", "system"],
                "ifc_name": ["@", "xcvr_metricdb", "interface"]
            },
            "data": {
                "temp_trend": ["slr", ["@", "xcvr_metricdb", "temp"]],
                "Hostname": ["@", "graph", "hostname"],
                "Ipv4_addr": ["@", "graph", "ipv4_addr"]
```

```
            }
        },
        "ordered": [{"temp_trend|coef": "descending"}]
    }
]
```

Use the Exploratory Interface

Use the following steps to build and visualize a query with the QBA Exploratory Interface. The interface is the GUI tool that builds the query step-by-step. At each step, it lets you to build a subquery, run it, and visualize the intermediate data. You can create a chain of subqueries that transform the data shape and size to arrive at final subquery that summarizes the final data visualization. In the following steps, we use a 'Traffic data trend analysis' use-case to demonstrate GUI components.

1. From the Apstra GUI, navigate to the interface under **Analytics > Exploratory Analytics**.



2. Select a **Begin** date, **End** date, and **Aggregation Interval** for your query.
3. Select a **Blueprint** from the dropdown.
4. Select one of the following **Probes**:

   Probes are data analytics pipelines that consist of mutiple processors.

- Device Traffic
- Device System Health

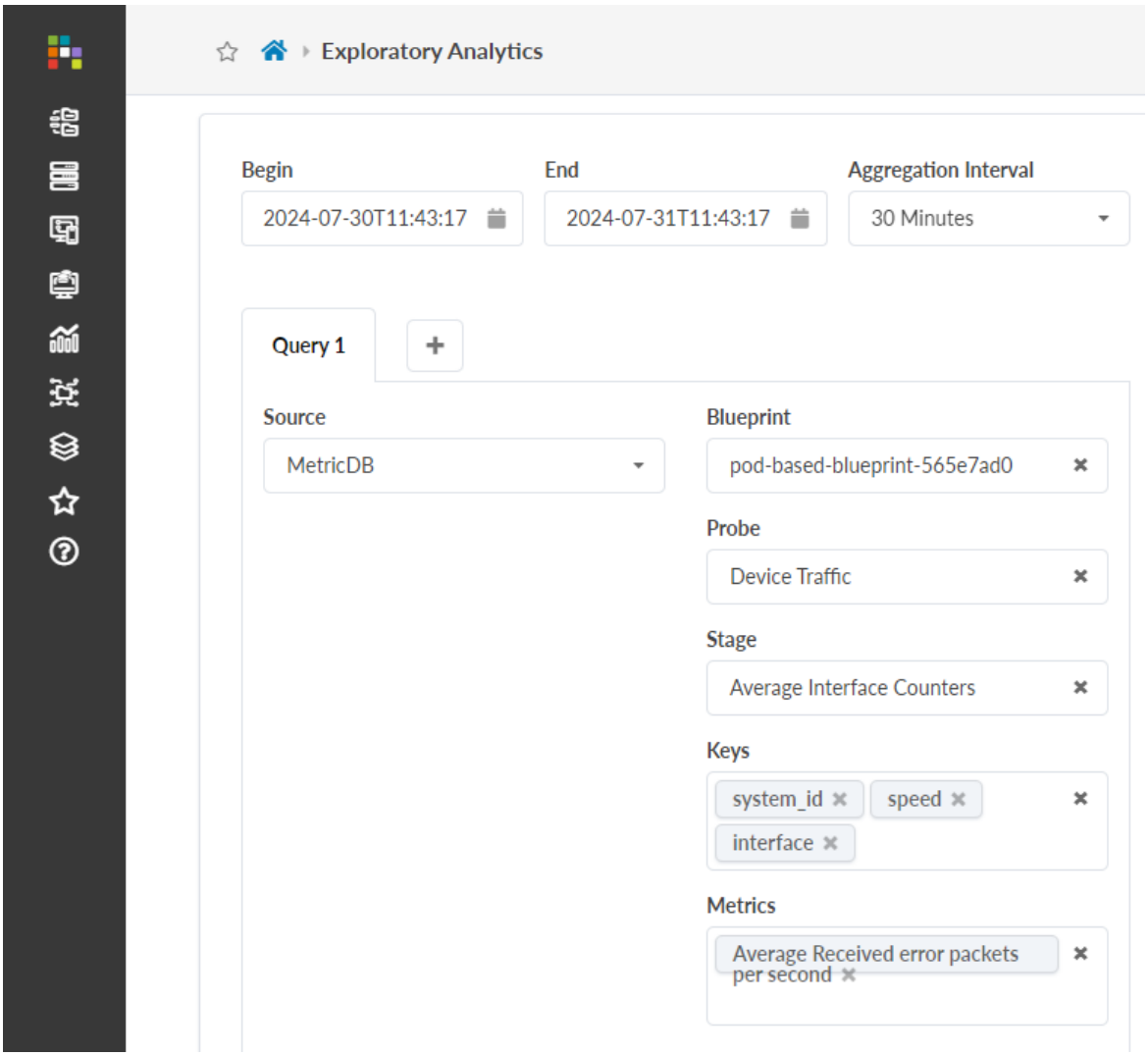
5. Select a **Stage** from the dropdown.

   A stage is a processor output of a probe's collected telemetry data. The output of a stage can be fed into the input of another stage.

   The available stages are:

   - System Interface Counters: Displays interface data grouped by system, such as Aggregated TX, Aggregated RX...etc.
   - Average Interface Counters: Displays average interface counter data grouped by system and interface, such as Average TX utilization, Average RX utilization..etc.

6. (Optional) Select one or more **Keys** from the dropdown. The `system_id` key is selected by default.
7. Select one or more **Metrics** to visualize from the dropdown.

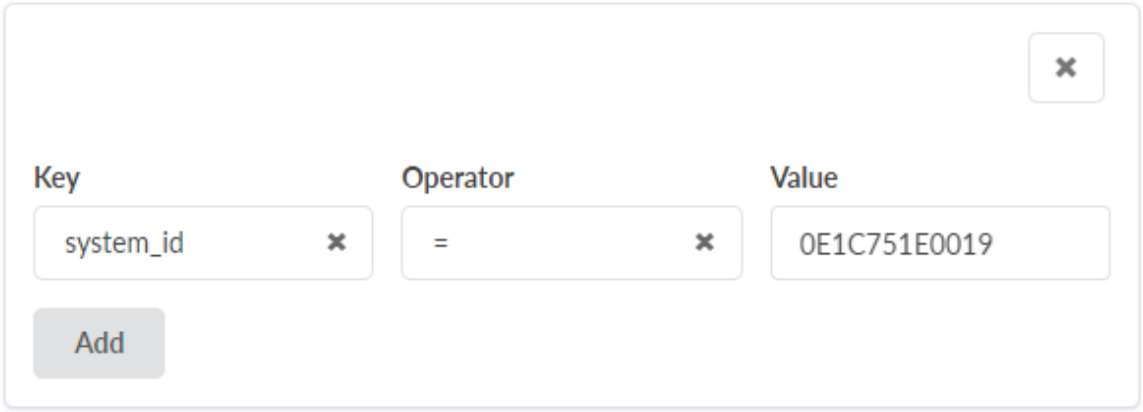


8. (Optional) Add a **Filter** to narrow down the query results.

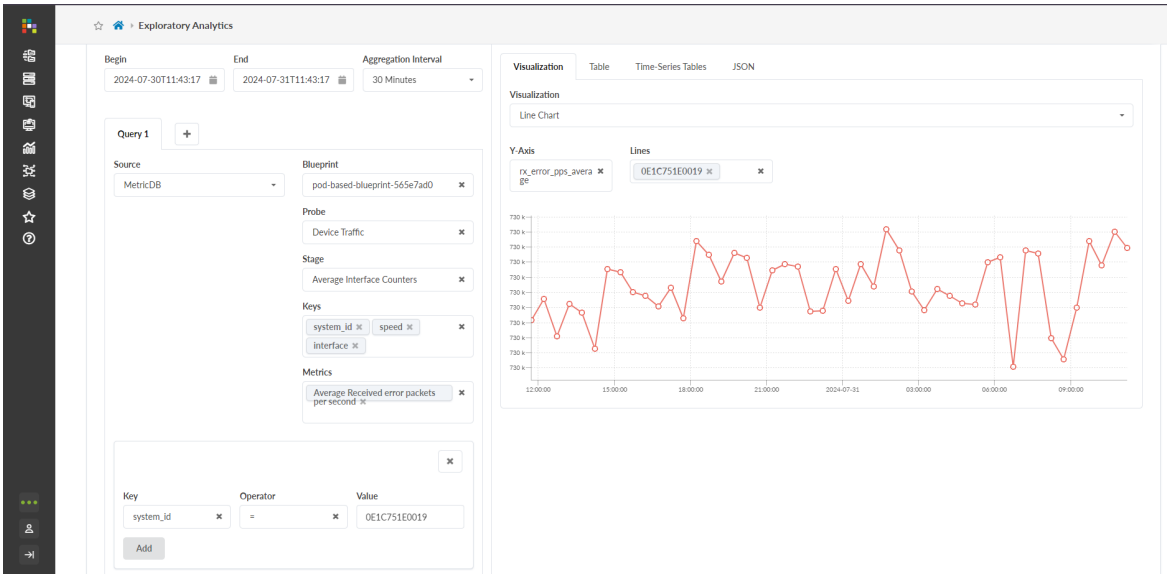
The available options are:

- Key: An attribute that identifies a row in the resulting chart.
- Operator: A connector that defines two conditions of a query.
- Value: A user-defined string.

For example, the Filter `system_id = 0E1C751E0019` limits the query output to the device associated with that `system_id` value.



9. Click **Submit Query**, and select a **Visualization** option from the dropdown on the visualization pane.

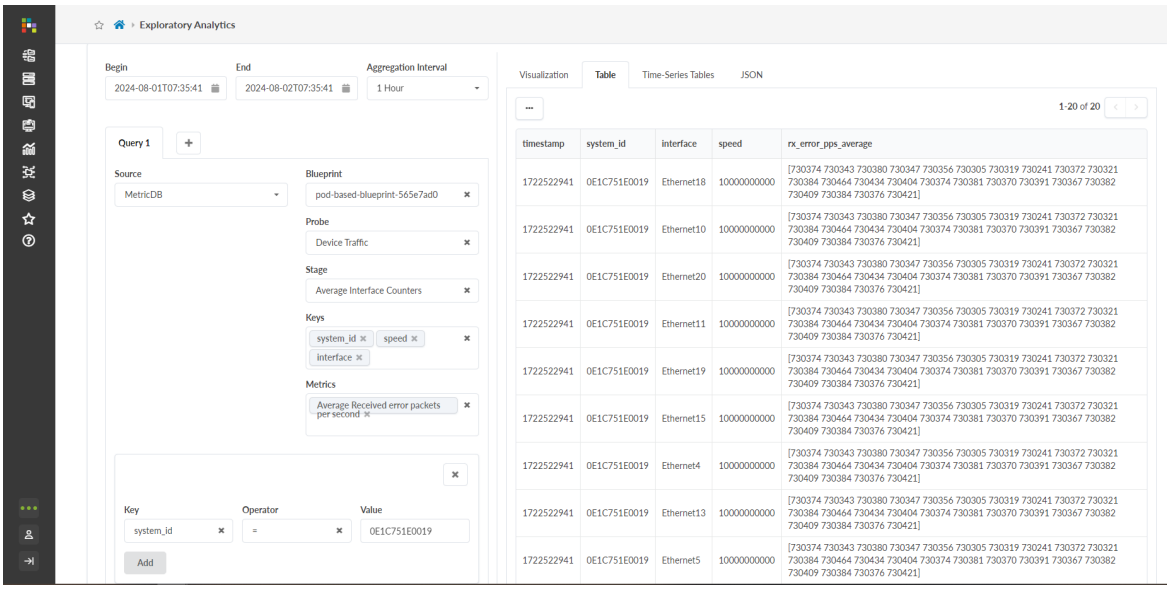


10. (Optional) Add a **Transformation** to the query output.

    A transformation is an operation performed in the middle of a data processing pipeline. In QBA, transformations perform grouping or aggregation operations on an input table (the initial query output). The newly-aggregated data is shown in an output table.

    Let's use the query we've previously built above, shown in Table view, as an input table. Without any applied transformations, the input table has five columns with many rows. These rows indicate

the "Average Received error packets per second (rx_error_pps_average)" for each interface associated with our specified `system_id`, at a given timestamp.



Transformations condense and filter the cells in the output table, based on data from the example input table above.

- Simple Linear Regression (SLR): Models the linear relationship between two variables. Use SLR to predict the value of an independent variable based on the value of a dependent variable. In this example, the dependent variable (x-axis) is the average received error packets per second. The independent variable (y-axis) is the interface type. Applying an SLR transformation results in the following output:

Table | JSON

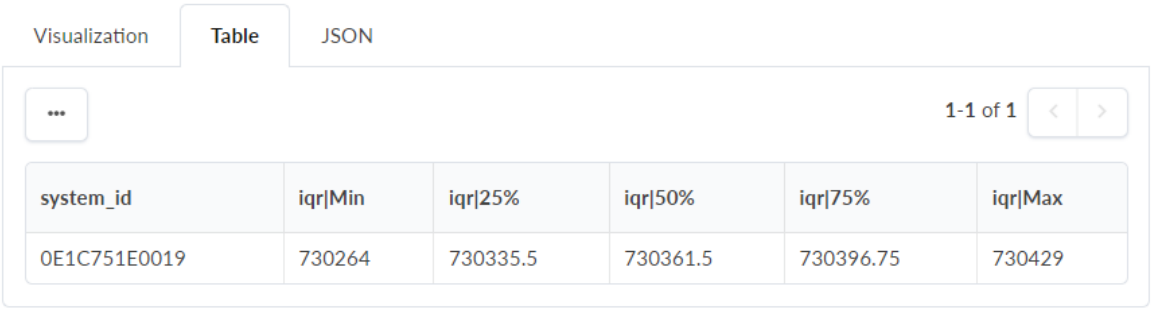
1-1 of 1

system_id	slr|gradient	slr|intercept	slr|gradient_variance
0E1C751E0019	2.787391304347826	730338.7366666666	1.496481245346125

Note the table elements:

- slr|gradient: The gradient of the regression line. Represents the rate of change of the average received error packets per second for a one-unit change in interface type.
- slr|intercept: Where the regression line intercepts the y-axis. The value of the dependent variable when the independent variable is 0.
- slr|gradient_variance: The variance of the gradient, or the uncertainty in the gradient estimate.

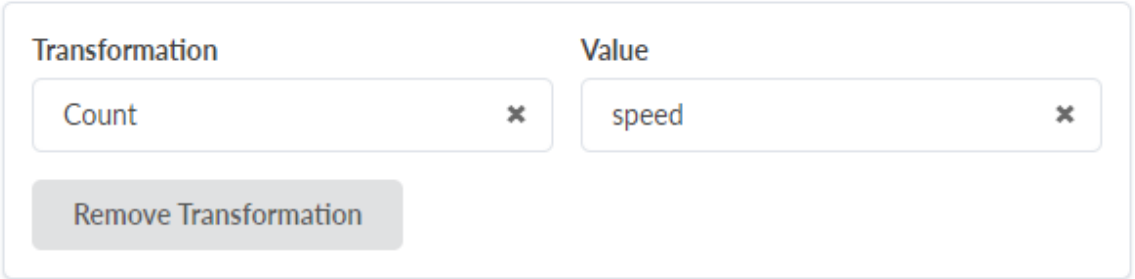
- Interquartile Range (IQR): The amount of spread in the middle half (50%) of a distribution of data. Calculates the IQR on the specified column (Value) and returns five numeric columns:
  - min
  - 25%
  - 50%
  - 75%
  - max

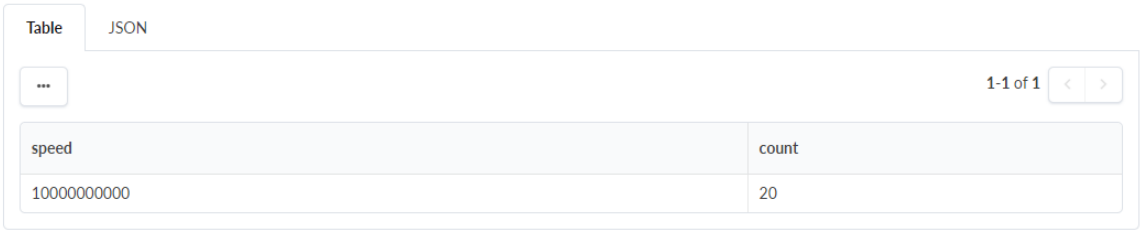


system_id	iqr|Min	iqr|25%	iqr|50%	iqr|75%	iqr|Max
0E1C751E0019	730264	730335.5	730361.5	730396.75	730429

- Most Recent: Returns the last value in the "rx_error_pps_average" column.
- Statistical Measures: Calculates the statistics (“min”, “max”, “mean”, “median”, “mode”, standard deviation (“stddev”)) for the "rx_error_pps_average" column for each row in the input table.
- Covariance: Calculates the covariance between the two specified columns and returns a single numeric column.
- Count: Counts the number of rows with similar keys and stores the final count as a single variable.

If we add a Count transformation and select a **Value** of the **speed** column, the output table is vertically condensed to display a single row with an aggregated value of interfaces with a given speed.





SEE ALSO

- Probes Introduction
- Probes

PART 15

External Systems (RBAC Providers)

CHAPTER 43

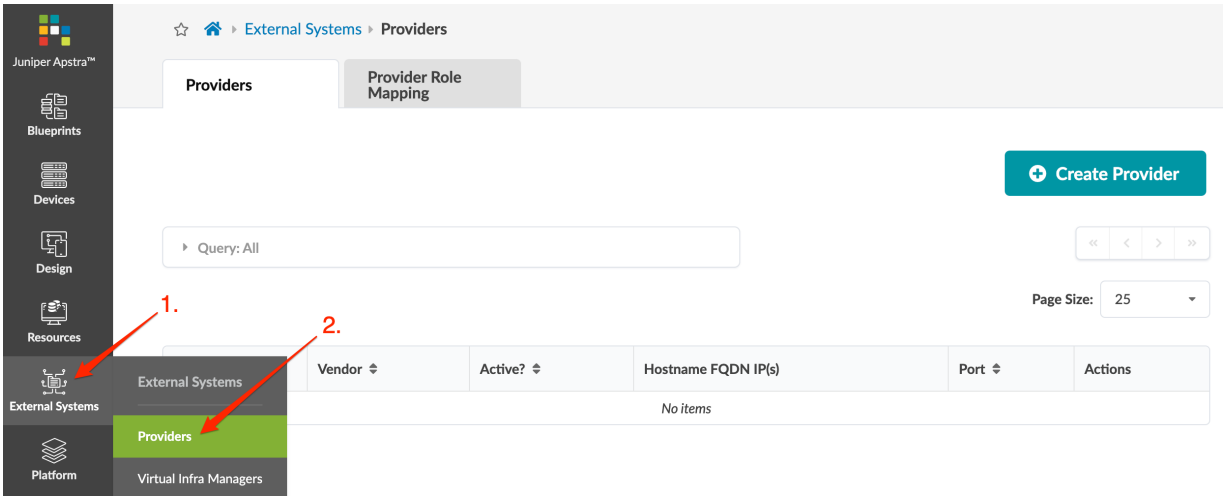
Providers

IN THIS CHAPTER

Providers (External Systems)

You can use Role-Based Access Control (RBAC) for specifying access permissions. RBAC servers are remote network servers that authenticate and authorize network access based on roles assigned to individual users within an enterprise (The accounting part of AAA is not included). If a user's group in the RBAC server is not specified, or if the provider group is not mapped to any user roles, that user cannot log in. This restriction avoids security issues by ignoring users without mapped groups. You can use the following protocols to authenticate and authorize users: LDAP, Active Directory, TACACS+, and RADIUS. Only Active Directory is supported as an external authentication server. No other versions are supported as external authentication servers, including RedHat IdM and Open LDAP. See the individual protocol sections for more information.

From the left navigation menu, navigate to **External Systems > Providers** to go to providers. You can create, clone, edit and delete providers.



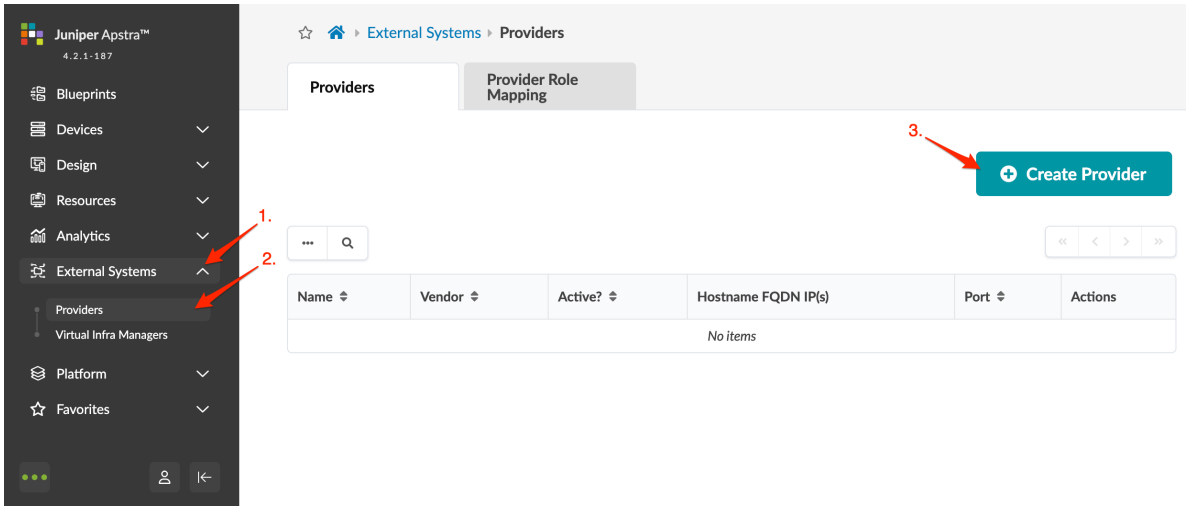
LDAP Provider

IN THIS SECTION

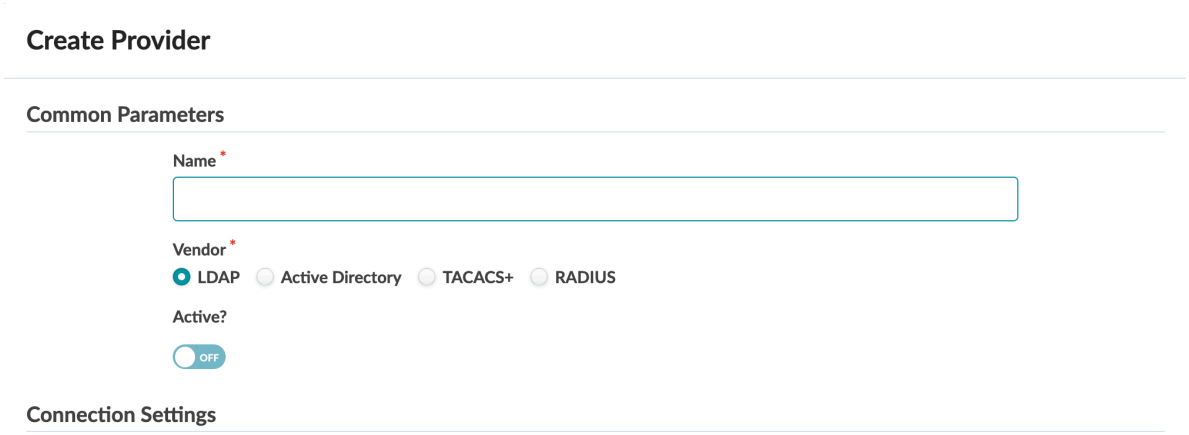
Create LDAP Provider

Lightweight Directory Access Protocol (LDAP)

1. From the left navigation menu, navigate to **External Systems > Providers** and click **Create Provider**.



2. Enter a **Name** (64 characters or fewer), select **LDAP**, and if you want LDAP to be the active provider, toggle on **Active?**.



3. For **Connection Settings**, enter/select the following:
   - **Port** - The TCP port - LDAP: **389**, LDAPS: **636**
   - **Hostname FQDN IP(s)** - The fully qualified domain name (FQDN) or IP address of the LDAP server. For high availability (HA) environments, specify multiple LDAP servers using the same settings. If the first server cannot be reached, connections to succeeding ones are attempted in order.
4. For **Provider-specific Parameters** enter/select the following, as appropriate:
   - **Groups Search DN** - The LDAP Distinguished Name (DN) path for the RBAC Groups Organizational Unit (OU)
   - **Users Search DN** - The LDAP Distinguished Name (DN) path for the RBAC Users Organization Unit (OU)
   - **Bind DN** - The LDAP Distinguished Name (DN) path for the active server user that the Apstra server will connect as

- **Password** - The LDAP server user password for the Apstra server to connect as
- **Encryption** - None, SSL/TLS or STARTTLS
- **Advanced Config**
  - **Timeout** (seconds) - Increasing timeout above the default 30 seconds may impact API responsiveness for all users. If you need a longer timeout for MFA support, you may increase the timeout up to 60 seconds. If you require a timeout above 60 seconds, contact "Juniper Technical Support" on page 1374.
  - **Username Attribute Name** - The LDAP attribute from the user entry that Apstra Server uses for authentication. (usually cn or uid)
  - **User Search Attribute Name**
  - **User First Name Attribute Name**
  - **User Last Name Attribute Name**
  - **User Email Attribute Name**
  - **User Object Class Attribute Name**
  - **User Member Attribute Name**
  - **Group Name Attribute Name**
  - **Group DN Attribute Name**
  - **Group Search Attribute Name**
  - **Group Member Attribute Name**
  - **Group Member Mapping Attribute Name**
  - **Group Object Class Attribute Name**

5. You can **Check provider parameters** and **Check login** (to verify authentication with the remote user credentials) before creating the provider.
6. Click **Create** to create the provider and return to the table view.

Configure LDAP Provider

To authorize Apstra users via a LDAP provider, the LDAP server must be configured to properly return a provider group attribute. This attribute must be mapped to a defined Apstra Role. The example configuration below is for the open-source OpenLDAP server.

```
dn: ou=People,dc=example,dc=com
objectClass: organizationalUnit
ou: People

dn: ou=Groups,dc=example,dc=com
objectClass: organizationalUnit
ou: Groups

dn: cn=user,ou=Groups,dc=example,dc=com
gidNumber: 5000
cn: user
objectClass: posixGroup
memberUid: USER1

dn: cn=USER1,ou=People,dc=example,dc=com
cn: USER1
givenName: USER1
loginShell: /bin/sh
objectClass: inetOrgPerson
objectClass: posixAccount
uid: USER1
userPassword: USER1
uidNumber: 10000
gidNumber: 5000
sn: USER1
homeDirectory: /home/users/USER1
mail: USER1@example.com
```

The **user** group must be mapped to a defined Apstra Role.

After configuring and activating a provider, you must "map" on page 1259 that provider to one or more user roles to give access permissions to users with those roles.

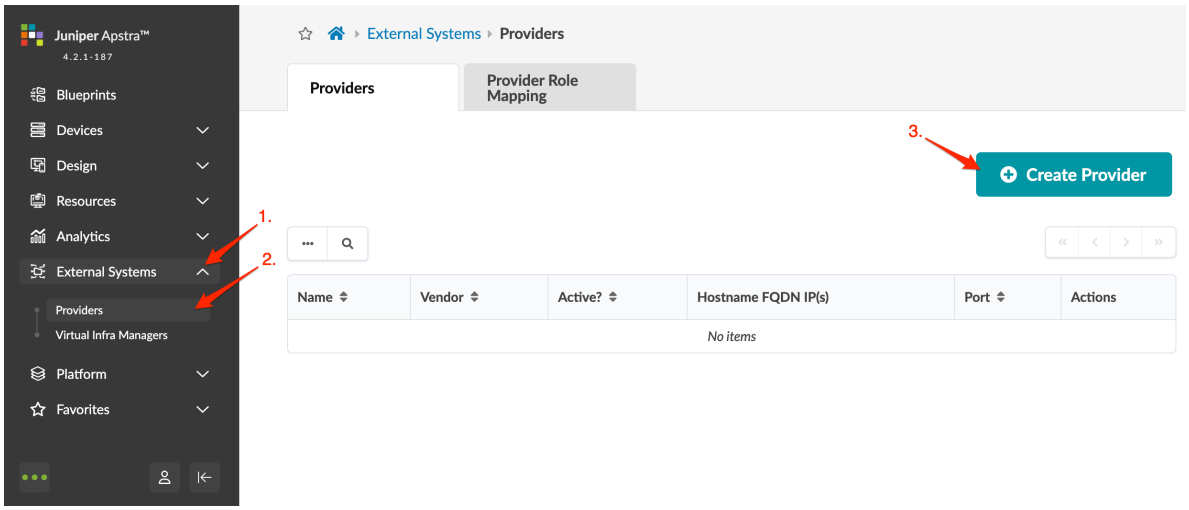
Active Directory Provider

IN THIS SECTION

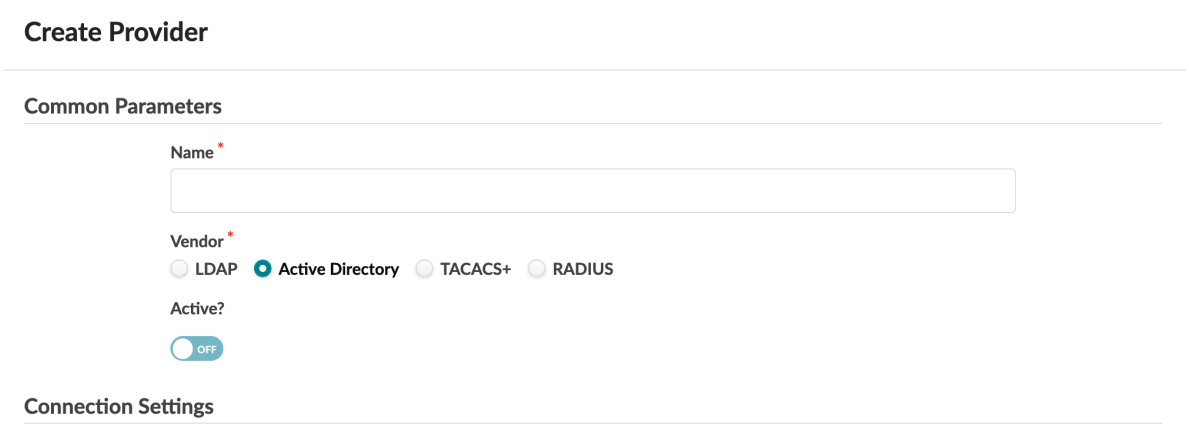
Active Directory (AD) is a database-based system that provides authentication, directory, policy, and other services in a Windows environment.

Create Active Directory Provider

1. From the left navigation menu, navigate to **External Systems > Providers** and click **Create Provider**.



2. Enter a **Name** (64 characters or fewer), select **Active Directory**, and if you want Active Directory to be the active provider, toggle on **Active?**.



3. For **Connection Settings**, enter/select the following:
   - **Port** - The TCP port used by the server
   - **Hostname FQDN IP(s)** - The fully qualified domain name (FQDN) or IP address of the AD server. For high availability (HA) environments, specify multiple AD servers using the same settings. If the first server cannot be reached, connections to succeeding ones are attempted in order.
4. For **Provider-specific Parameters** enter/select the following, as appropriate:
   - **Groups Search DN** - The AD Distinguished Name (DN) path for the RBAC Groups Organizational Unit (OU)
   - **Users Search DN** - The AD Distinguished Name (DN) path for the RBAC Users Organization Unit (OU)
   - **Bind DN** - The AD Distinguished Name (DN) path for the active server user that the Apstra server will connect as
   - **Password** - The AD server user password for Apstra server to connect as
   - **Encryption** - None, SSL/TLS or STARTTLS
   - **Advanced Config**
     - **Timeout** (seconds) - Increasing timeout above the default 30 seconds may impact API responsiveness for all users. If you need a longer timeout for MFA support, you may increase the timeout up to 60 seconds. If you require a timeout above 60 seconds, contact "Juniper Technical Support" on page 1374.
     - **Username Attribute Name** - The AD attribute from the user entry that the Apstra server uses for authentication. (usually **cn** or **uid**)
     - **User Search Attribute Name**
     - **User First Name Attribute Name**
     - **User Last Name Attribute Name**
     - **User Email Attribute Name**
     - **User Object Class Attribute Name**
     - **User Member Attribute Name**
     - **Group Name Attribute Name**
     - **Group DN Attribute Name**
     - **Group Search Attribute Name**

- **Group Member Attribute Name**
- **Group Member Mapping Attribute Name**
- **Group Object Class Attribute Name**

5. You can **Check provider parameters** and **Check login** (to verify authentication with the remote user credentials) before creating the provider.
6. Click **Create** to create the provider and return to the table view.

After configuring and activating a provider, you must "map" on page 1259 that provider to one or more user roles to give access permissions to users with those roles.

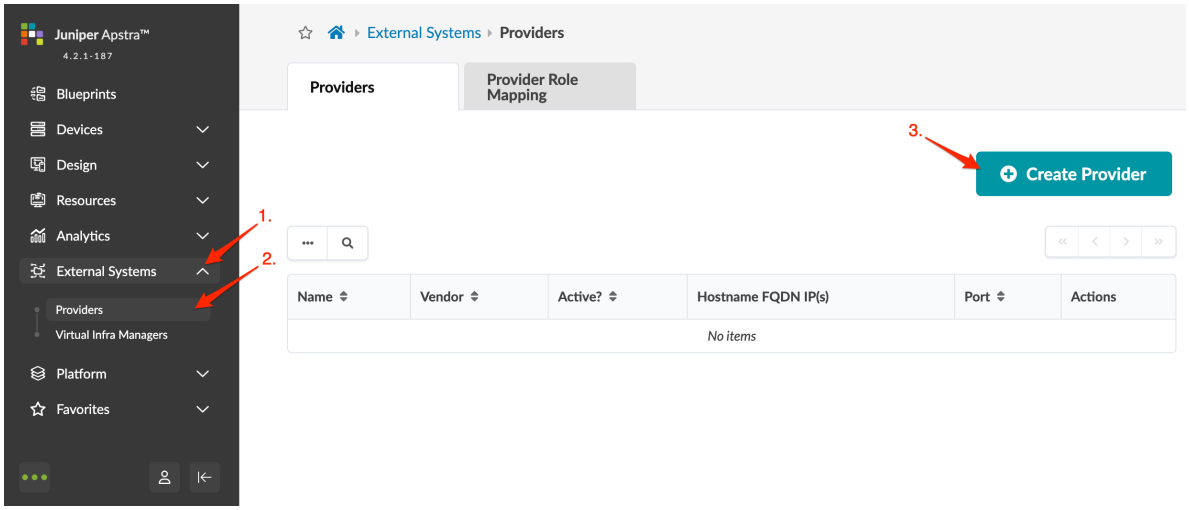
TACACS+ Provider

IN THIS SECTION

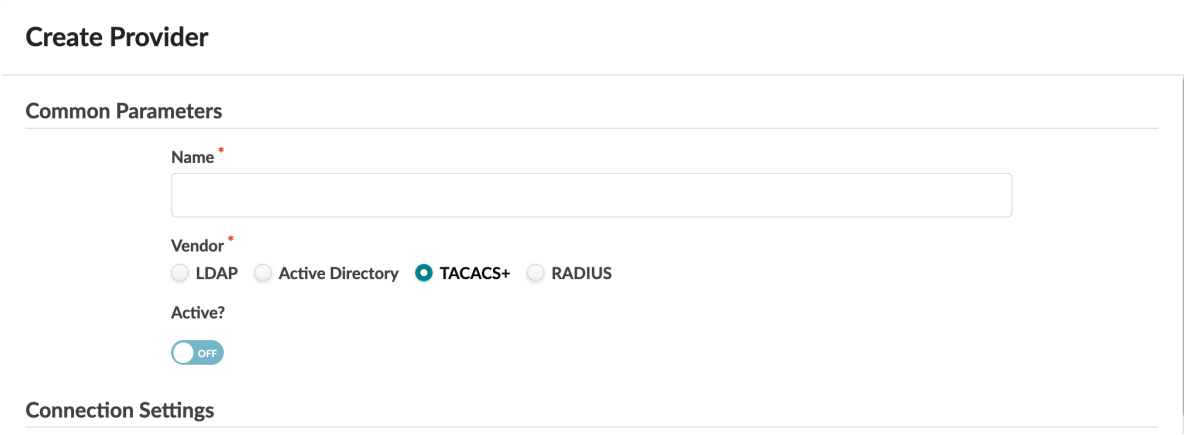
Terminal Access Controller Access-Control Systems (TACACS+)

Create TACACS+ Provider

1. From the left navigation menu, navigate to **External Systems > Providers** and click **Create Provider**.



2. Enter a **Name** (64 characters or fewer), select **TACACS+**, and if you want TACACS+ to be the active provider, toggle on **Active?**.



3. For **Connection Settings**, enter/select the following:
   - **Port** - The TCP port used by the server, usually **49**
   - **Hostname FQDN IP(s)** - The fully qualified domain name (FQDN) or IP address of the TACACS+ server. For high availability (HA) environments, specify multiple TACACS+ servers using the same settings. If the first server cannot be reached, connections to succeeding ones are attempted in order.
4. For **Provider-specific Parameters** enter/select the following, as appropriate:
   - **Shared Key** - shared key configured on the server

     Caution

     Shared key is not displayed when editing a configured TACACS+ provider. If you do not change it, the previously configured shared key is retained. If you test the provider and you have not re-entered the shared key, a null shared key is used for the test and may not work.
   - **Auth Mode** - Authentication mode - ASCII (clear-text), PAP (Password Authentication Protocol), or CHAP (Challenge-Handshake Authentication Protocol)
   - **Advanced Configuration**
     - **Timeout** (seconds) - Increasing timeout above the default 30 seconds may impact API responsiveness for all users. If you need a longer timeout for MFA support, you may increase the timeout up to 60 seconds. If you require a timeout above 60 seconds, contact "Juniper Technical Support" on page 1374.
5. You can **Check provider parameters** and **Check login** (to verify authentication with the remote user credentials) before creating the provider.
6. Click **Create** to create the provider and return to the table view.

Configure TACACS+ Provider

To authorize Apstra users via a TACACS+ provider, the TACACS+ server must be configured to properly return an **aos-group** attribute. This attribute must be mapped to a defined Apstra Role. The example configuration below is for the open-source tac_plus TACACS+ server.

```
user = jdoe {
    default service = permit
    name = "John Doe"
    member = admin
    login = des LQqpIWvpxDXDw
}

group = admin {
    service = exec {
        priv-lvl = 15
    }
    cmd=show {
        permit .*
    }
    service = aos-exec {
        default attribute = permit
        priv-lvl = 15
        aos-group = apstra-admins
    }
}
```

The **apstra-admins** group must be mapped to a defined Apstra Role.

After configuring and activating a provider, you must "map" on page 1259 that provider to one or more user roles to give access permissions to users with those roles.

RADIUS Provider

IN THIS SECTION

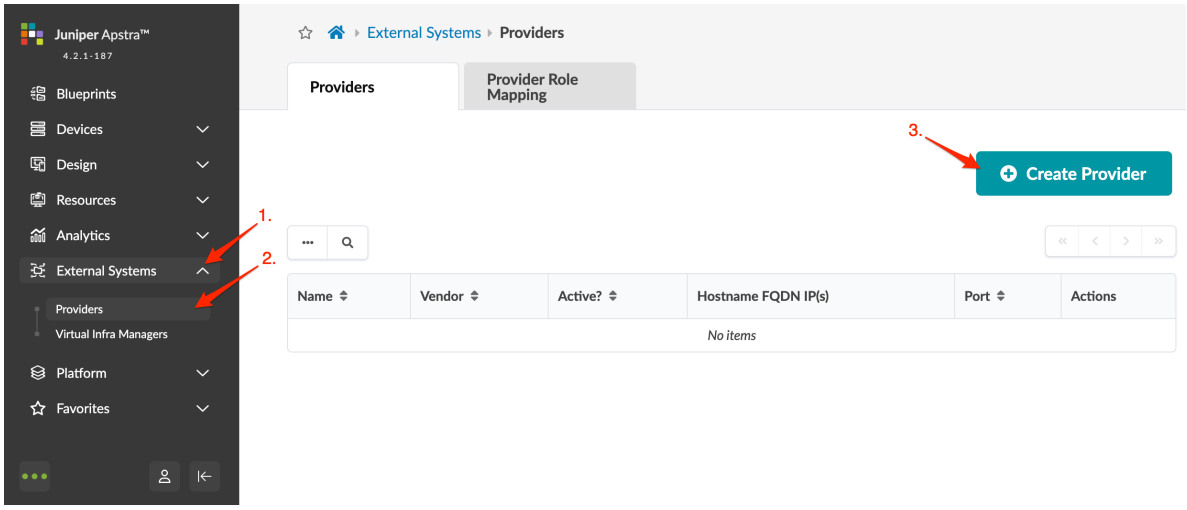
Remote Authentication Dial-In User Service (RADIUS). See below for limitations.

RADIUS Limitations

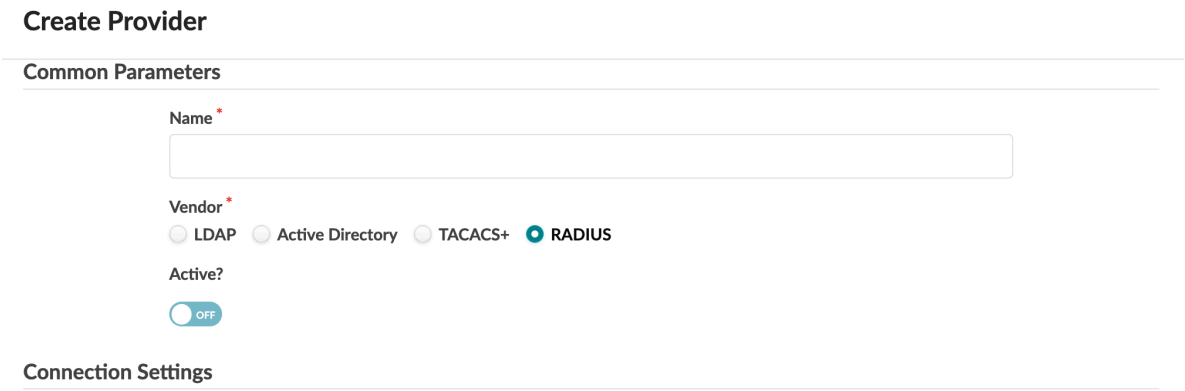
- All password changes should be done on the RADIUS server. You cannot change the user's password from Apstra once you enable the External Provider.
- RADIUS authentication does not control Linux user login via SSH.
- If the user group is changed on the RADIUS server, you will need to change role-mapping in Apstra accordingly.
- Nested groups are not allowed. You must explicitly assign each group to a role.
- When a user logs in, only username and password are required for authenticating against the remote RADIUS server. Log in credentials are not cached. Therefore, when a user logs in, a connection between Apstra and the remote RADIUS server is required.

Create RADIUS Provider

1. From the left navigation menu, navigate to **External Systems > Providers** and click **Create Provider**.



2. Enter a **Name** (64 characters or fewer), select **RADIUS**, and if you want RADIUS to be the active provider, toggle on **Active?**.



3. For **Connection Settings**, enter/select the following:
   - **Port** - The TCP port used by the server, default is **1812** as specified in RFC 2865.
   - **Hostname FQDN IP(s)** - The fully qualified domain name (FQDN) or IP address of the RADIUS server. For high availability (HA) environments, specify multiple RADIUS servers using the same settings. If the first server cannot be reached, connections to succeeding ones are attempted in order.
4. For **Provider-specific Parameters** enter/select the following, as appropriate:
   - **Shared Key** (64 characters or fewer) - shared key configured on the server

     CAUTION: Shared key is not displayed when editing a configured RADIUS provider. If you do not change it, the previously configured shared key is retained. If you test the provider and you have not re-entered the shared key, a null shared key is used for the test and may not work.

     An example of a pre-shared key configuration that tests successfully with Apstra software is from Ubuntu FreeRADIUS (an open source RADIUS server). The Shared Key as given in the RADIUS server configuration must be provided in Apstra.

     ```
     home_server localhost {
     ipaddr = 127.0.0.1
     port = 1812
     type = "auth"
     secret = "testing123"
     response_window = 20
     max_outstanding = 65536
     ```

   - Advanced Config

- **Group Name Attribute Name** - To specify a role that a user belongs to, the RADIUS server must specify the users' group. The user group information must be specified with **Framed-Filter-ID** as the attribute. It is used to assign users to different RADIUS groups.

  For example, the FreeRADIUS config below specifies the **Framed-Filter-ID** attribute to be **freerad**. In this case, when mapping later, you would enter **freerad** for the Provider Group.

```
/etc/freeradius/users
       freerad Cleartext-Password := "testing123"
       Framed-Filter-Id = "freerad"
```

  So that the user can be mapped to an existing group in the Apstra environment, the RADIUS server must return the Apstra group name as part of the authentication response.

  CAUTION: If the group is unmapped, users cannot log in.

- **Timeout** (seconds) - Increasing timeout above the default 30 seconds may impact API responsiveness for all users. If you need a longer timeout for MFA support, you may increase the timeout up to 60 seconds. If you require a timeout above 60 seconds, contact "Juniper Technical Support" on page 1374.

After configuring and activating a provider, you must "map" on page 1259 that provider to one or more user roles to give permissions to users with those roles.

Edit RBAC Provider

CAUTION: Any users who are logged into Apstra software when a setting is changed in an active RBAC provider, are immediately logged out without notification. To continue, the user must log back into the Apstra server. This does not affect users who are defined locally on the Apstra server (for example, **admin**).

1. Either from the table view (External Systems > Providers) or the details view, click the **Edit** button for the provider to edit.
2. Make your changes.
3. Click **Update** (bottom-right) to edit the provider and return to the table view.

Delete RBAC Provider

1. Either from the table view (External Systems > Providers) or the details view, click the **Delete** button for the provider to delete.
2. Click **Delete** to delete the provider and return to the table view.

CHAPTER 44

Provider Role Mapping

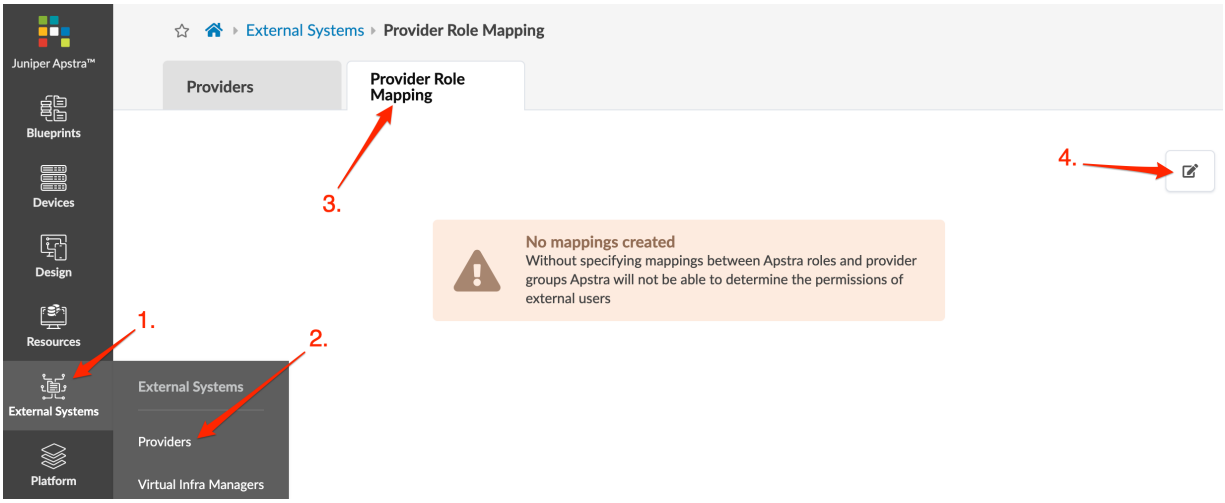
IN THIS CHAPTER

Provider Role Map Overview

After configuring an RBAC provider, you must map the provider to one or more user roles to give access permissions to users with those roles. You can create, edit and delete provider role mappings, as needed. Other details to be aware of include the following:

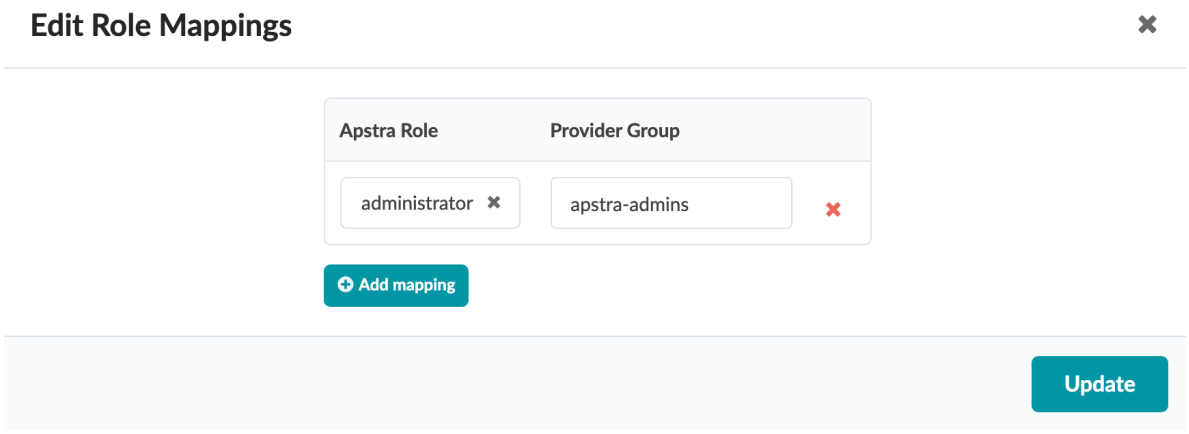
- Only one provider can be active at a time.
- You can map more than one Apstra role to the same provider group (new in version 4.0).
- When the same username exists both locally and in the RBAC provider, the local user is used to authenticate login attempts.
- Changing users with the web-based RBAC feature does not modify accounts on the Apstra server VM. To change these credentials, use standard Linux CLI commands: "useradd", "usermod", "userdel", "passwd".

From the left navigation menu, navigate to **External Systems > Providers > Provider Role Mapping** to go to provider role mapping.



Create Provider Role Map

1. From the left navigation menu, navigate to **External Systems > Providers > Provider Role Mapping** and click the **Edit** button (top-right).
2. Click **Add mapping**, select a role from the drop-down list, then enter a provider group. The following is an example for mapping the **apstra-admins** group that was configured in TACACS+ configuration.



> **TIP**: To see user role details, navigate to **Platform > User Management > Roles**. From there, you can also create new roles, as needed.

3. To add another role mapping, click **Add mapping** and select an **Apstra Role** and **Provider Group**. You can have more than one role associated with the same provider group.

4. Click **Update** to create the role map. If the provider that you mapped is the active provider, then users with the mapped roles can log in with their usernames and passwords defined in the RBAC server.

Edit RBAC Provider Role Map

CAUTION: Changing role mappings for an active provider causes all remotely logged in users to be logged out (because the session tokens are cleared when changes are made). Users will need to log back into the system. This includes user **admin**, if **admin** is not logged in locally.

1. From the left navigation menu, navigate to **External Systems > Providers > Provider Role Mapping** and click the **Edit** button (top-right).
2. Edit role mapping as needed.
3. Click **Update** to update the role map.

Delete RBAC Provider Role Map

1. From the left navigation menu, navigate to **External Systems > Providers > Provider Role Mapping**, click the **Edit** button (top-right), then click the **X** next to the mapping to delete.
2. Click **Update** to update the role map.

PART 16

Platform

CHAPTER 45

User Management

IN THIS CHAPTER

User / Role Management Introduction

IN THIS SECTION

Overview

You need a user (profile) to work in the Apstra GUI environment. The areas in the environment that you can access and/or change are determined by the roles assigned to you as a user. Apstra ships with one predefined user called **admin** that's assigned the **administrator** role. The **administrator** role is one of the five predefined roles as shown in the table below:

Table 53: Predefined User Roles

Role	Permissions
administrator	Includes all permissions.
device_ztp	Includes one permission, to edit ZTP. For setting up Apstra ZTP server, We recommend creating a dedicated user and assigning only this role.
License_reader	Includes one permission, to read Juniper Apstra Licenses.
user	Includes permission to view and edit various elements including permission to create users.
viewer	Includes permission to only view various elements.

You can't modify predefined roles, but you can create custom roles if you have **Write/Read Roles** permission. The **admin** user has this permission; you can also create a user and assign it with the **Write/Read Roles** permission.

You can't modify the predefined **admin** user, but you can create custom users if you have **Read/Write Users** permission. The **admin** and **user** users have this permission; you can also create a user and assign it with the **Read/Write Users** permission

Each role applies to one of three different permission types, as shown in the table below.

Table 54: User Role Permission Types

Permission Type	Permissions
Global	Includes general blueprint permissions (not blueprint-specific) as well as areas outside of blueprints, such as for devices, design, resources, AAA, Analytics, External Systems, Platform and others.
Granular	Includes blueprint-related permissions for all blueprints or for selected blueprints.
Tenant	Includes permissions based on routing zones. (new in Apstra version 5.0.0)

See sections below for more details about each permission type.

Global Permissions

Blueprints

Includes permissions for the following:

- Allow overriding other users staged changes (write only)
- Blueprints (read, write, commit, delete)
- Connectivity Templates (read only)
- Show information about user who locked blueprint (read only)

Devices

Includes permissions for the following:

- Agents (read, write)
- Chassis Profiles (read, write)
- Device Profiles (read, write)
- Devices (read, write)
- Linecard Profiles (read, write)
-
- ZTP (read, write)

Design

Includes permissions for the following:

- Config Templates (read, write)
- Configlets (read, write)
- Interface Maps (read, write)
- Logical Devices (read, write)
- Port Aliases (read, write)

- Property Sets (read, write)
- Rack Types (read, write)
- Tags (read, write)
- Templates (read, write)

Resources

Includes permissions for the following:

- ASN Pools (read, write)
- Integer Pools (read, write)
- IP Pools (read, write)
- IPv6 Pools (read, write)
- VNI Pools (read, write)

AAA

Includes permissions for the following:

- Audit Config (read, write)
- Audit Events (read only)
- Roles (read, write)
- Security Config (read, write)
- Users (read, write)

Analytics

Includes permissions for the following:

- Flow Data Collectors (read only)
- Juniper Apstra Query Based Analytics (read only)
- Telemetry Service Registry (read only)

External Systems

Includes permissions for the following:

- AAA Providers (read, write)
- Virtual Infra Manager (read, write)

Platform

Includes permissions for the following:

- Exempt Juniper Apstra Cluster Management read-only mode (write only)
- Juniper Apstra Cluster Management (read, write)
- Juniper Apstra Licenses (read, write)
- Juniper Apstra Metric Logs (read only)
- Streaming (read, write)
- Sysdb Data (read, write)

Other

Includes permissions for the following:

- Connector Types (read only)
- Graph Queries (read, write)
- Port Setting Schema (read only)
- Telemetry RPC Schema Registry (read only)
- Tenants (read only)

Granular Permissions

You can apply granular permissions to all blueprints or to selected blueprints.

Common Permissions

Includes permissions for the following:

- Read blueprint

- Make any change to staging blueprint
- Allow overriding other users staged changes
- Commit changes
- Show information about user who locked blueprint

Datacenter-specific Permissions

Includes permissions for the following:

- Manage tenants
- Manage racks and links
- Manage resource groups
- Manage routing zones
- Manage generic systems
- Manage virtual networks (includes managing VN endpoints)
- Manage virtual network endpoints

Freeform-specific Permissions

Includes permissions for the following:

- Manage property sets
- Manage resources

Tenant Permissions

You can apply tenants permissions to tenants.

Tenant-specific Permissions

Includes permissions for the following:

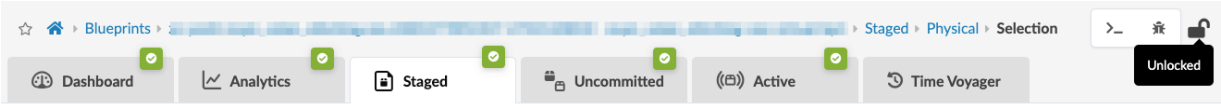
- Manage resource groups
- Manage routing zones
- Manage virtual networks

- Manage virtual network endpoints

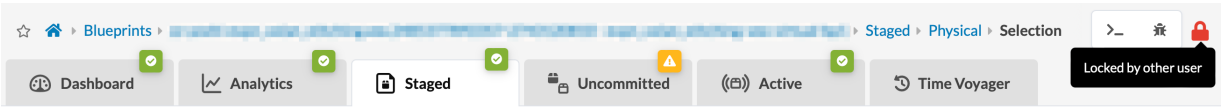
Blueprint Locking Feature

The blueprint locking feature prevents restricted users (based on their roles) from making changes that effectively are not permitted. In particular, a restricted user should not be able to commit changes made by another user.

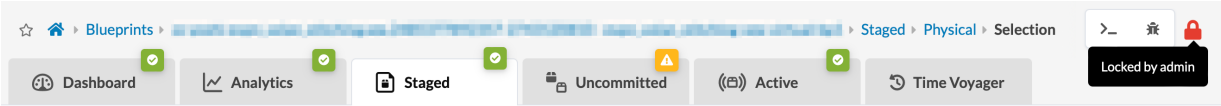
If a blueprint has no changes to commit, it is unlocked.



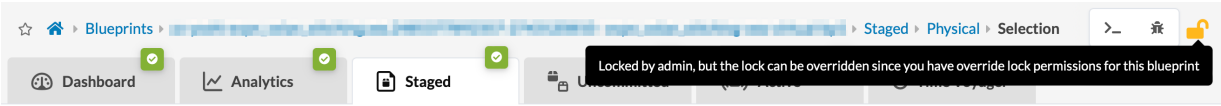
If you have permission (based on the your assigned roles) to create/update/delete virtual networks, for example, and another user has made uncommitted changes to the blueprint, the blueprint is locked. You can't create/update/delete virtual networks until the changes are committed or reverted by the locking user who made the uncommitted changes, unless you are the one who made the changes.



If you have permission (based on your assigned roles) to see the name of the user who created the pending changes, the name is displayed.



A user with "Allow overriding other users staged changes" permission can make any changes to, apply changes for, and revert changes for any blueprint.



Role-Based Access Control (RBAC)

You can "map roles" on page 1259 to external groups used by authentication providers such as LDAP, Active Directory, TACACS+, and RADIUS.

With Enhanced Role Based Access Control, you can create blueprint-specific roles with specific privileges allowing limited control to associated users. This allows you to create more hierarchical roles and protect against accidental changes to the network.

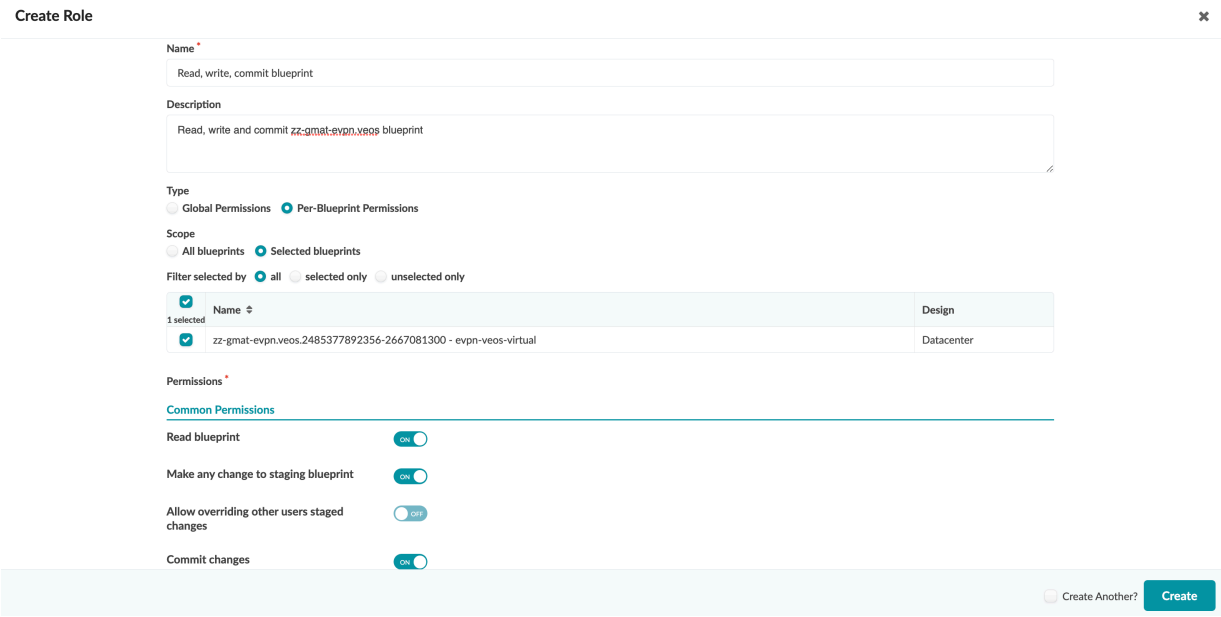
For example, a user assigned the role **Manage generic systems** can add generic systems, copy existing generics, add links to generic systems, add links to leaf devices, and update node tags. A user assigned the role **Manage racks and link**s can perform all those operations plus they can change rack speeds and delete links. A user with the **Manage racks and links** role essentially has permissions for all FE/FFE operations. If you want to restrict a user to physical server operations only, assign them the **Manage generic systems** role, and not the **Manage racks and links** role.

Use Cases

These use cases are meant to give you an idea of how to work with roles and users. Specific steps for creating roles and users are described in later sections.

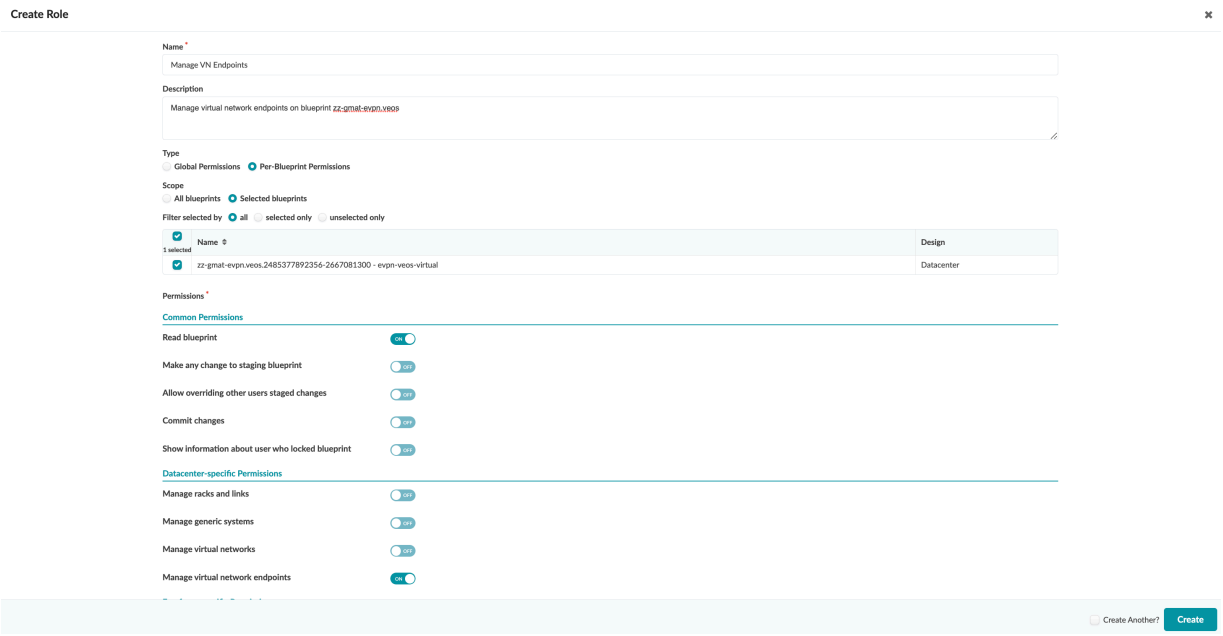
Read, Write and Commit Specific Blueprints

To allow a user to read, write and commit specific blueprints, create a per-blueprint permissions role for the specified blueprint(s). Toggle on **Read blueprint**, **Make any change to staging blueprint**, and **Commit changes**. These permissions include **Manage virtual networks** and **Manage virtual network endpoints** even though those permissions may or may not be toggled on. Assign the role to the user.



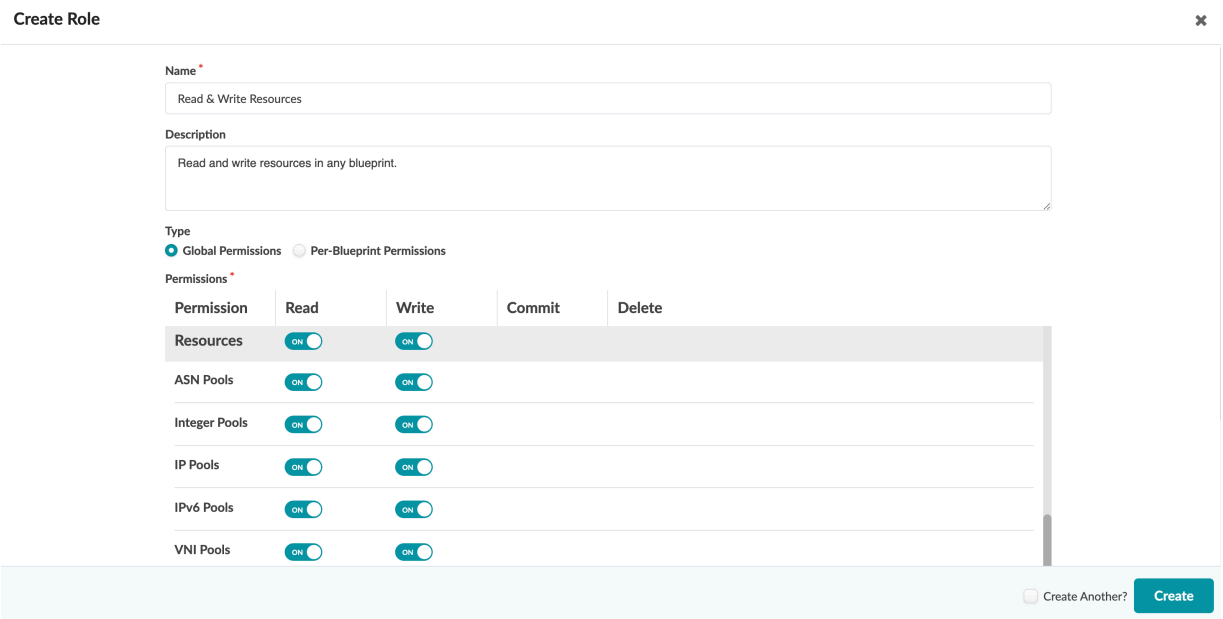
Manage VN Endpoints on Specific Blueprints

To allow a user to only manage virtual network endpoints on specific blueprints, select **Per-Blueprint Permissions**, select one or more blueprint IDs (or **All** for all blueprints), then toggle on **Manage virtual network endpoints**. Assign the role to the user.



Read and Write Resources on all Blueprints

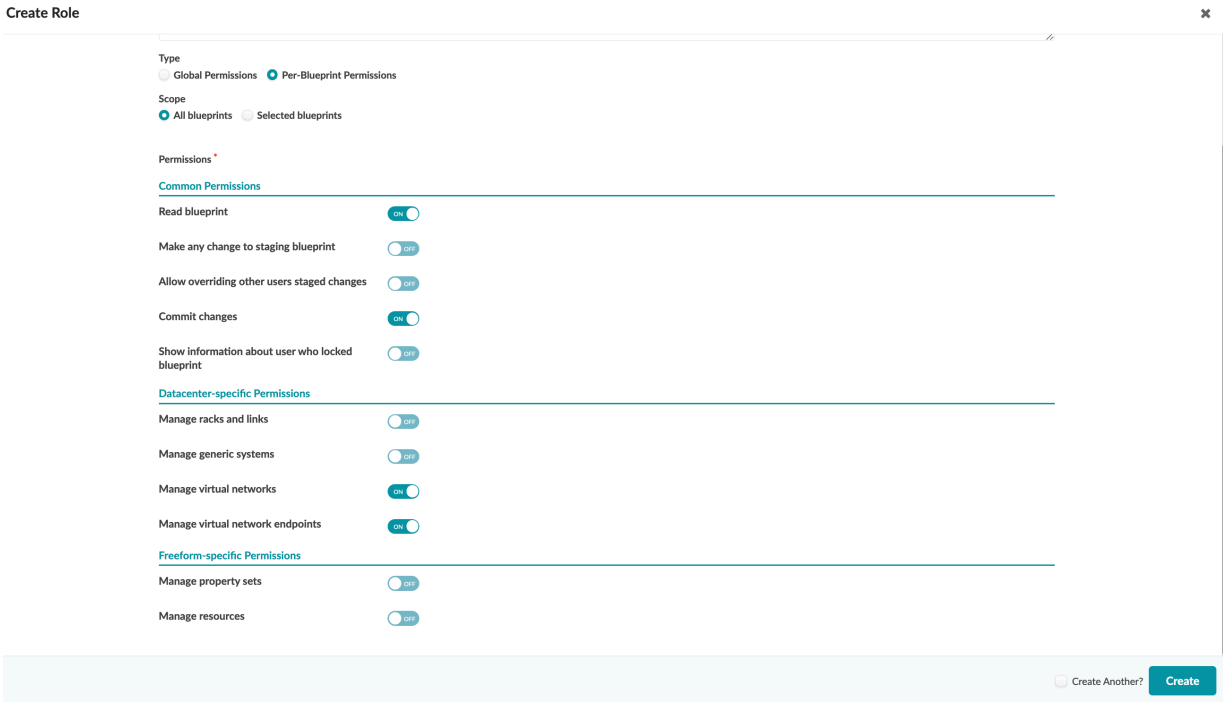
To allow a user to read and write resources on any blueprint, create a global permissions role. Toggle on **Resources** for **Read** and **Write** to toggle on all resources at once. Assign the role to the user.



Create Virtual Networks only (not Including Allocating Resources)

To limit a user's role to only create virtual networks and look at blueprint details, create a role for **Per-Blueprint Permissions**, and either select specific blueprints or all blueprints. Then toggle on **Read**

Blueprint, **Commit changes**, **Manage virtual networks**, and **Manage virtual network endpoints**. By not selecting **Make any change to staging blueprint** you are limiting the changes that can be made to virtual networks only. Assign the role to the user.



Create Virtual Networks and Allocate Resources

To be able to create virtual networks and allocate resources to them, you can assign several roles as follows:

- Read and Write Resources on all Blueprints (described in previous section)
- Create Virtual Networks Only (not Including Allocating Resources) (described in previous section) with the addition of toggling on **Make any change to staging blueprint**. This also permits a user with this role to make other changes besides virtual network changes.

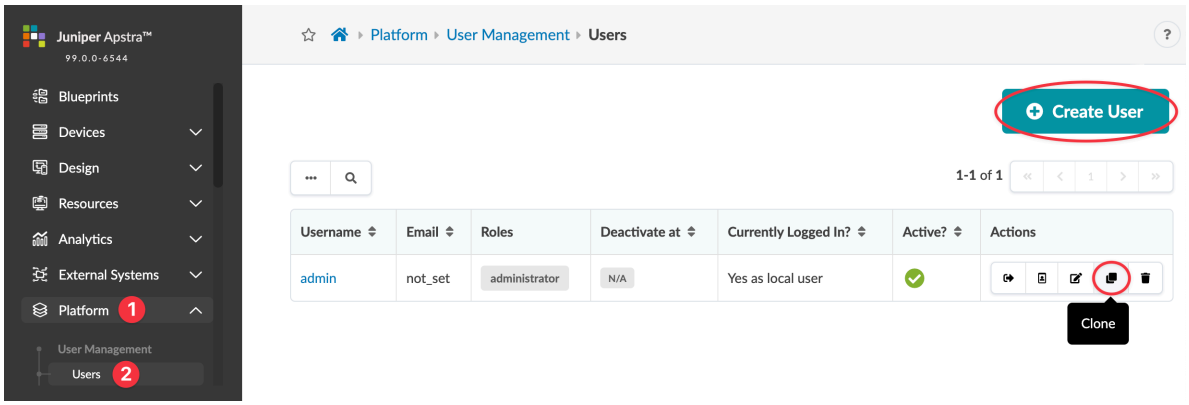
Users

IN THIS SECTION

Create User Profile

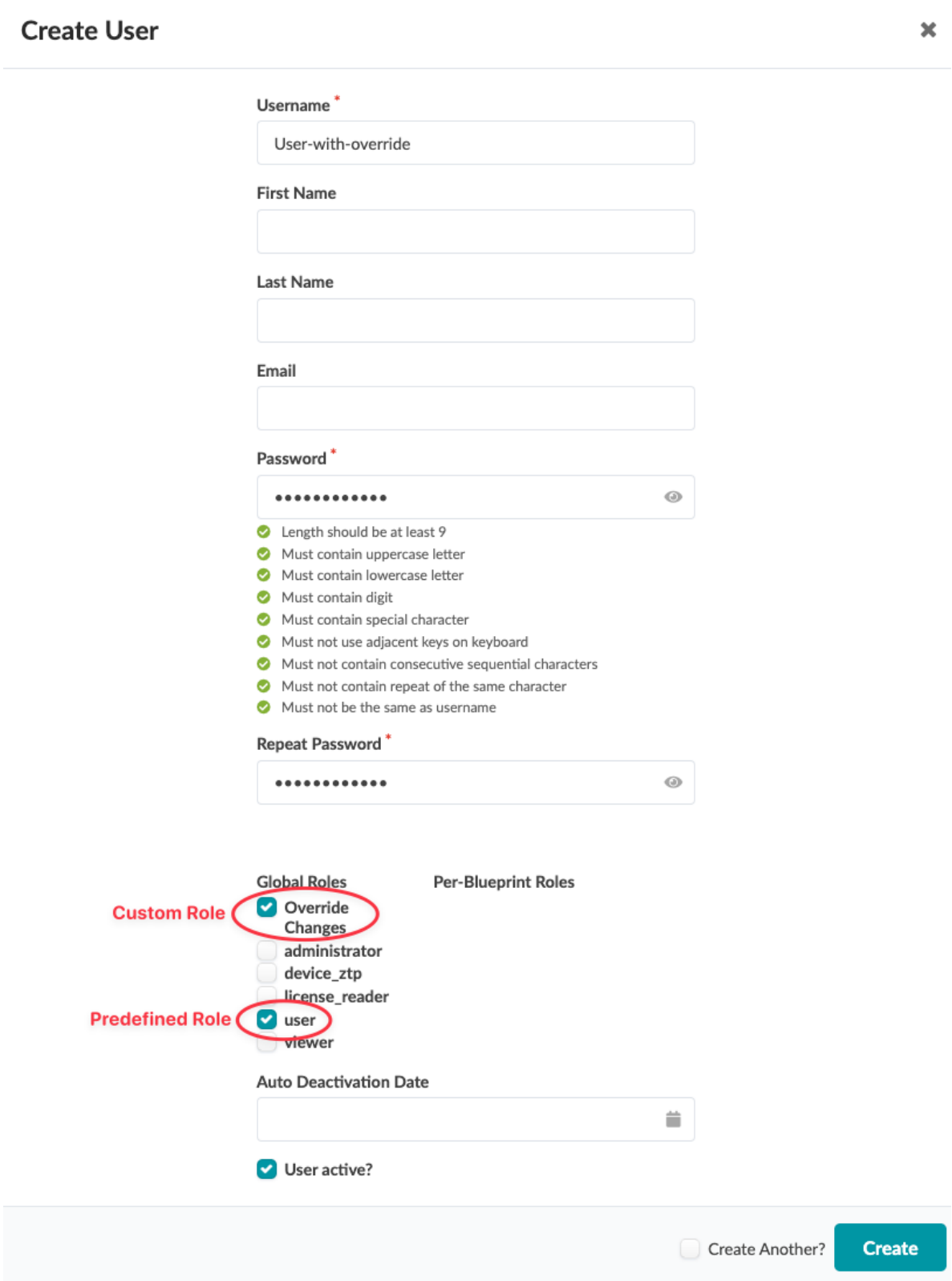
Creating a user profile enables a user to access the Apstra platform via its GUI. (To enable a user to access the Apstra platform via SSH, create a local Linux system user.)

1. From the left navigation menu, navigate to **Platform > User Management > Users** and click **Create User** (or to copy an existing user profile and customize it, click the **Clone** button for the user profile to copy).



   The **Create User** dialog opens (or the **Clone User** dialog opens, as applicable).

2. Enter a username, enter a password that meets password complexity requirements, then re-enter the password. (You can "change requirements" on page 1286 from **Platform > Security > Password Complexity Parameters**.)

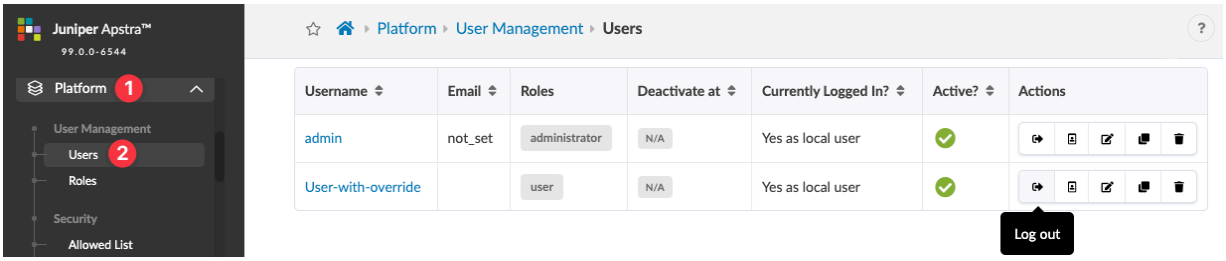

Create User
Username *
User-with-override
First Name
Last Name
Email
Password *
Length should be at least 9
Must contain uppercase letter
Must contain lowercase letter
Must contain digit
Must contain special character
Must not use adjacent keys on keyboard
Must not contain consecutive sequential characters
Must not contain repeat of the same character
Must not be the same as username
Repeat Password *
Global Roles
Per-Blueprint Roles
Custom Role
Override Changes
administrator
device_ztp
license_reader
Predefined Role
user
viewer
Auto Deactivation Date
User active?
Create Another?
Create

3. Select the check boxes for one or more roles for the new user. (To see permissions for each of the global roles, navigate to Platform > User Management > Roles.) If custom roles have been created, they appear as options along with the predefined roles.

   For example, you may want a user to have the default roles for **user**, and also be able to see who has staged any blueprint changes and be able to override those changes. First, "create a custom role" on page 1277 with the override permission. Then, create a user with two roles: the **user** role and this custom role.
4. To deactivate the user automatically on a specific data and time, select the date and time from the **Auto Deactivation Date** calendar. You can also immediately deactivate and activate the user by deselecting or selecting the **User active?** check box. (This feature is new in Apstra version 5.0.0.)
5. Click **Create** to create the user profile and return to the table view.

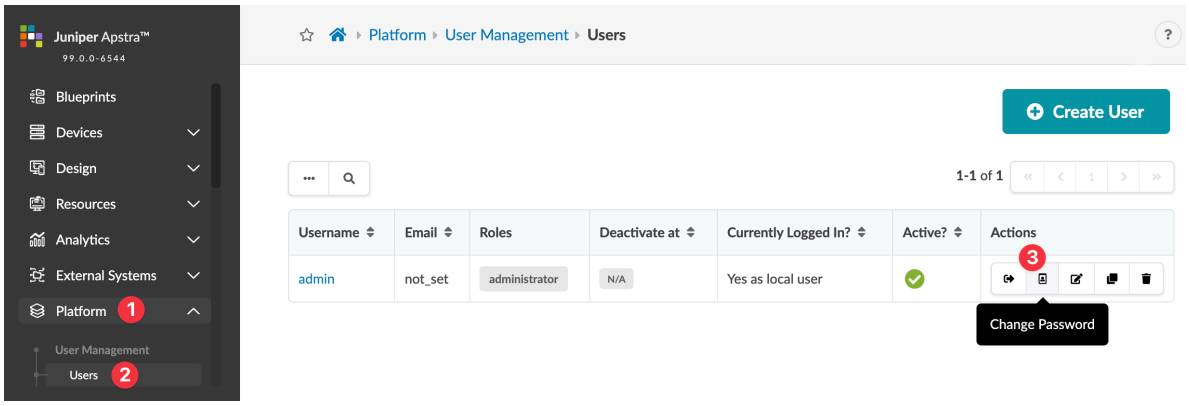
Log Out User

From the left navigation menu, navigate to **Platform > User Management > Users** and click the **Log Out** button in the **Actions** panel, for the user. The user is logged out of the Apstra environment.



Change Apstra GUI User Password

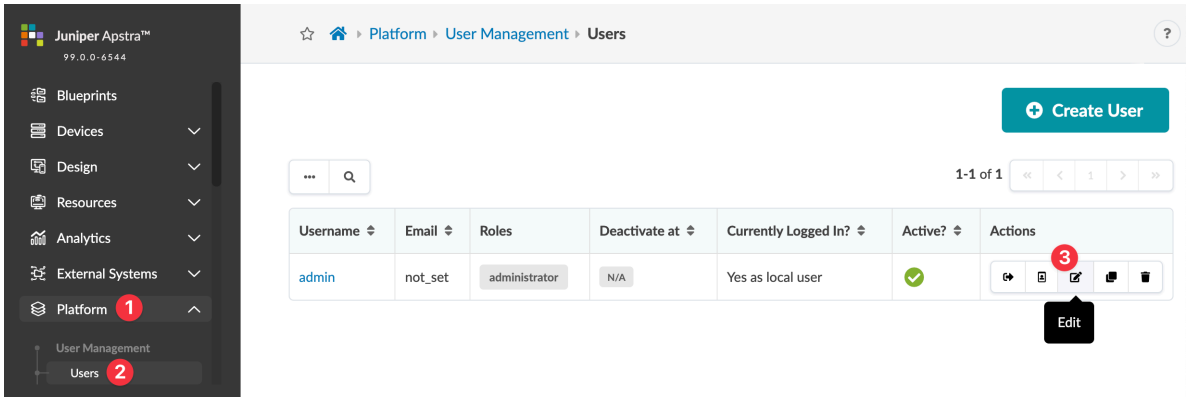
1. From the left navigation menu, navigate to **Platform > User Management > Users** and click the **Change Password** button in the **Actions** panel, for the user.



2. Enter a new password that meets password complexity requirements, then re-enter the new password. (You can "change password requirements" on page 1286 from **Platform > Security > Password Complexity Parameters**.)
3. Click **Change Password** to update the password.

Update User Profile

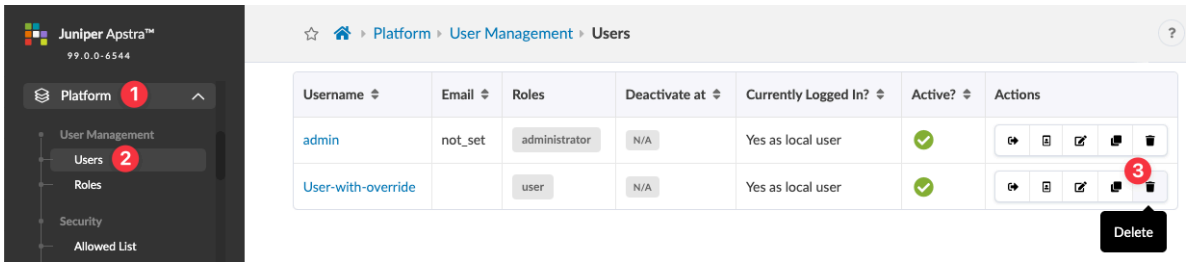
1. From the left navigation menu, navigate to **Platform > User Management > Users** and click the **Edit** button in the **Actions** panel, for the user profile to update.



2. Change roles and/or other details, as needed.
3. Click **Update** to update the user profile and return to the table view.

Delete User Profile

1. From the left navigation menu, navigate to **Platform > User Management > Users** and click the **Delete** button in the **Actions** panel, for the user profile to delete. (User **admin** can't be deleted.)



2. Click **Delete** to delete the user profile and return to the table view.

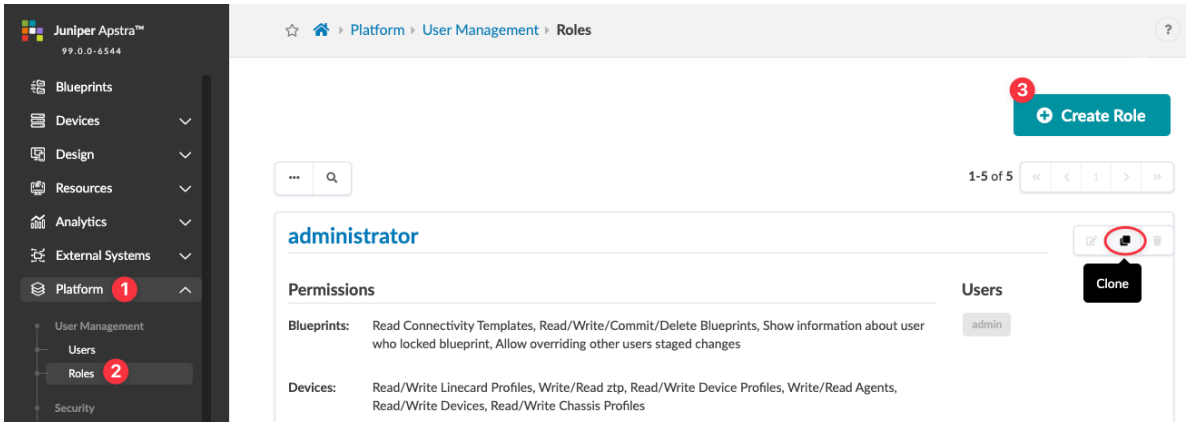
Roles

IN THIS SECTION

Create User Role

User roles specify permissions for working in the different areas of the Apstra environment. They can be specific to blueprints, parts of blueprints (as of Apstra version 5.0.0), or more general in nature. To customize a user's access and edit capability you'll assign roles to user profiles. Start by creating roles based on the permissions you want to control.

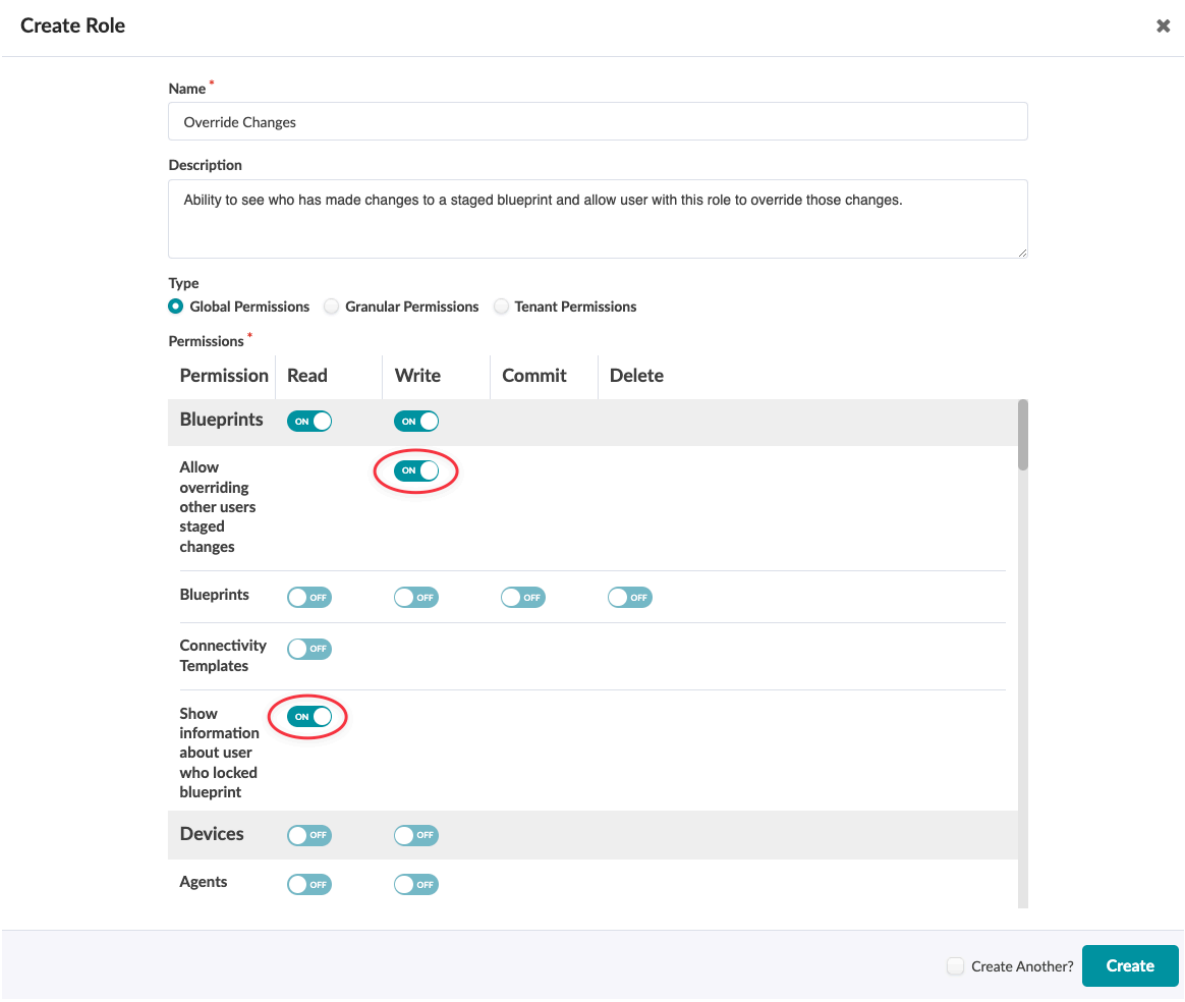
1. From the left navigation menu of the Apstra GUI, navigate to **Platform > User Management > Roles** and click **Create Role** (or to copy an existing user role and customize it, click the **Clone** button for the user role to copy).



   The **Create Role** dialog opens (or the **Clone Role** dialog opens, as applicable).

2. Enter a unique role name and (optional) description, then select permissions, as applicable.

> **NOTE**: Roles are either global, granular (per-blueprint) or tenant (routing zone and their inherited elements). Be careful. If you select permissions in one type, then click the radio button for another type, you'll lose the permissions you already set.

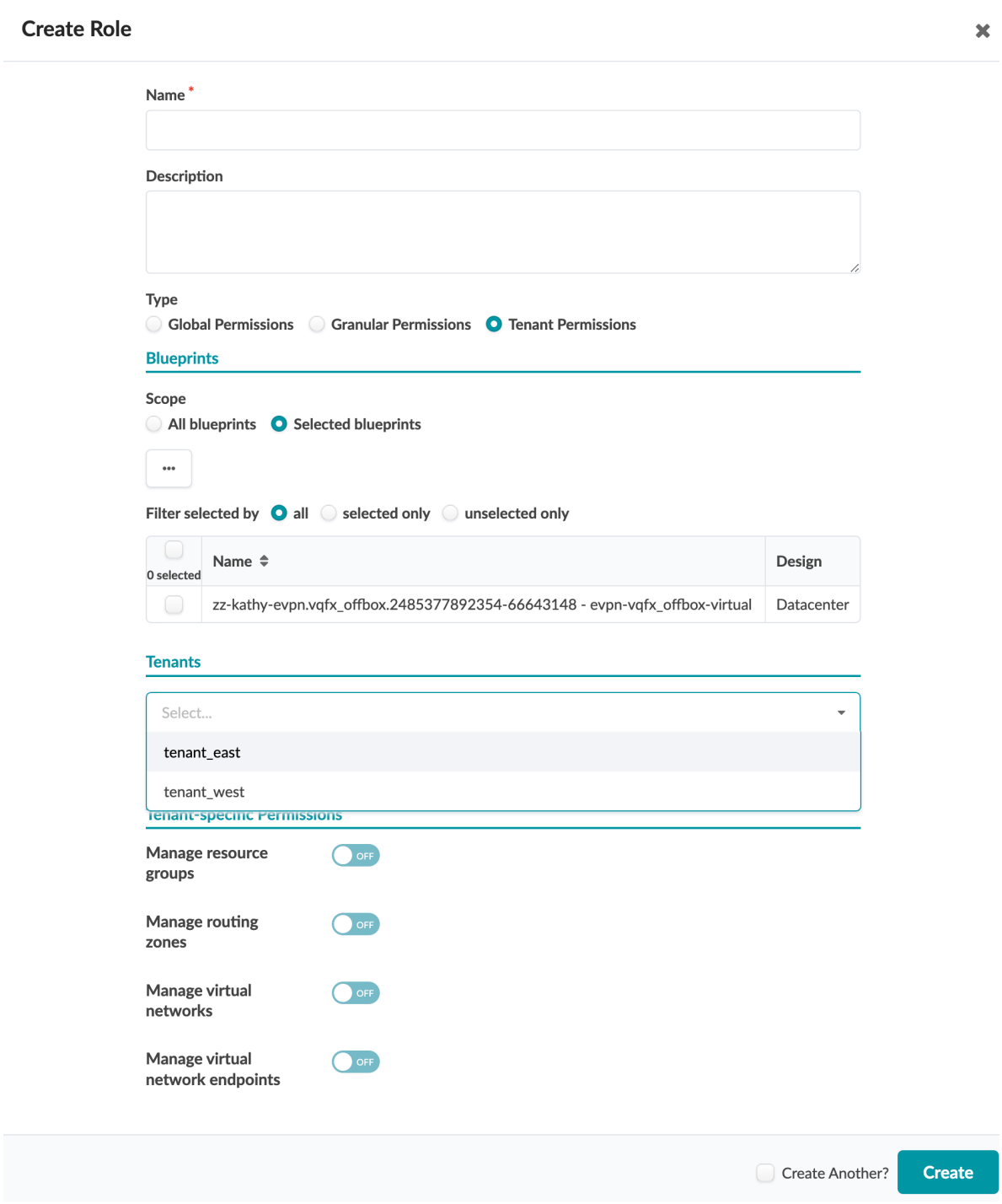


Global Permissions pertain to Apstra details *other* than blueprint details. They include general blueprint read, write, commit and delete permissions as well as permissions for platform, external systems, resources, design, devices, and more. To add global permissions, select **Global Permissions** and toggle on/off one or more permissions.

For example (circled in the image above), if another user has staged changes in a blueprint, that blueprint is locked for additional changes until that (unidentified) user commits or reverts the changes. You can create and assign a role that allows a user to see who made the changes and/or allow them to override those changes. (The **admin** role already has these permissions by default.)

To grant permissions based on blueprints *instead*, select **Granular Permissions**, select either specific blueprints or **All blueprints**, then select one or more permissions that are datacenter-specific, freeform-specific or common to all blueprints.

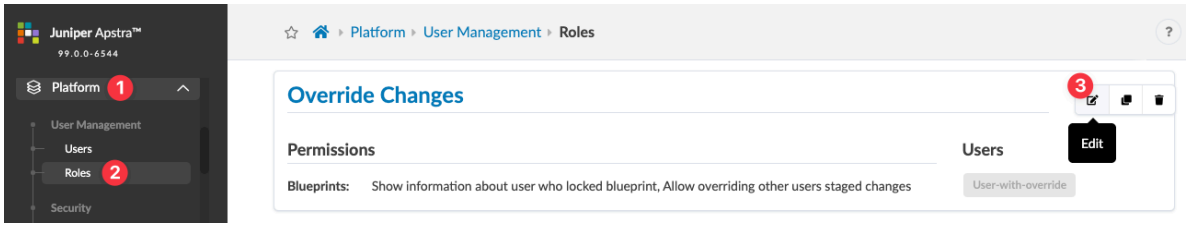
To grant permissions at the routing zone level *instead*, select **Tenant Permissions**, select either specific blueprints or **All blueprints**, select one or more tenants, then toggle on/off one or more permissions.



3. Click **Create** to create the role and return to the **Roles** view.

Update User Role

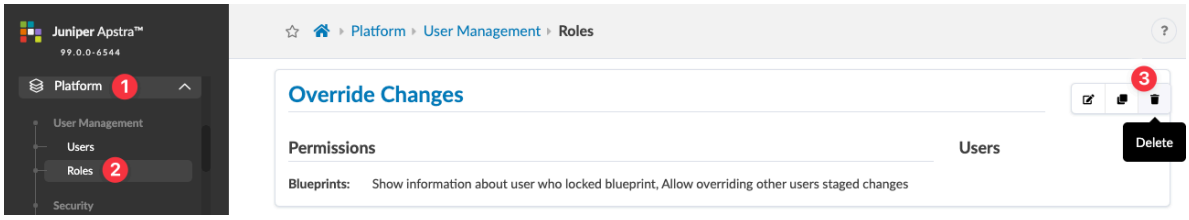
1. From the left navigation menu, navigate to **Platform > User Management > Roles** and click the **Edit** button for the user role to update. The four predefined user roles (administrator, device_ztp, user, viewer) can't be modified.



2. Change permissions, as applicable.
3. Click **Update** to update the role and return to the table view.

Delete User Role

1. From the left navigation menu, navigate to **Platform > User Management > Roles** and click the **Delete** button for the user role to delete. You can't delete a role if it's assigned to a user. The four predefined user roles (administrator, device_ztp, user, viewer) can't be deleted.



2. Click **Delete** to delete the role and return to the table view.

CHAPTER 46

Security

IN THIS CHAPTER

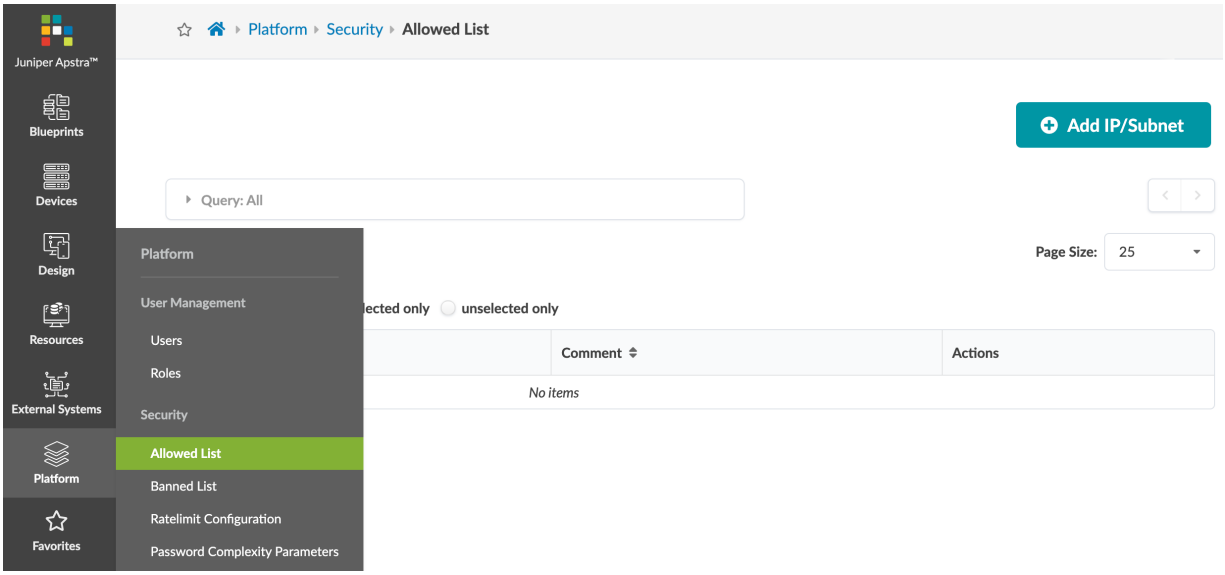
Allowed List

IN THIS SECTION

Allowed List Overview

You can add trusted IP/subnets to the allowed list so they are never locked out, even if they violate rate limit rules. You can add and change comments about those IP/subnets. Changes to the allowed list are recorded in the event log (Platform > Event Log).

From the left navigation menu, navigate to **Platform > Security > Allowed List**. You can search and sort the list. You can add, edit, and delete IP/subnets.



Add IP/Subnet to Allowed List

1. From the left navigation menu, navigate to **Platform > Security > Allowed List** and click **Add IP/Subnet**.
2. Enter an IP address or subnet, and a comment.
3. To keep the dialog open to add another IP/subnet, check the **Create Another** check box.
4. Click **Create** to add the IP/subnet and return to the table view (or, if you checked **Create Another**, return to the dialog to enter another IP/subnet).

Edit IP/Subnet to Allowed List

1. From the left navigation menu, navigate to **Platform > Security > Allowed List** and click the **Edit** button for the IP/subnet to edit.
2. Change the comment.
3. Click **Update** to complete the change and return to the table view.

Delete IP/Subnet from Allowed List

1. From the left navigation menu, navigate to **Platform > Security > Allowed List**.
2. Select the IP/subnet(s) to delete.
   - To delete a single IP/subnet, click the **Delete** button for the IP/subnet (right-side).
   - To delete one or more IP/subnets, click the checkbox (left-side) for one or more IP/subnets and click the **Delete** button above the list.
3. Click **Update** to complete the deletion and return to the table view.

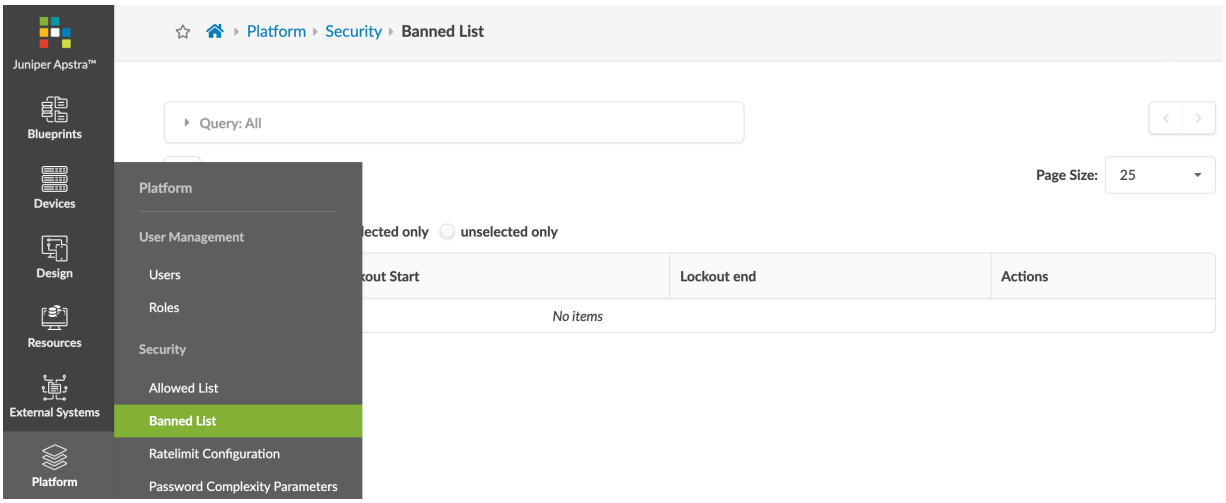
Banned List

IN THIS SECTION

Banned List Overview

IP/subnets that violate rate limit rules are automatically added to the banned list and are locked out for the configured lockout period, or until an admin removes them from the banned list. The banned list has a lower precedence than the allowed list, so an IP/subnet on the banned list may actually not be banned. Changes to the banned list are recorded in the event log (Platform > Event Log).

From the left navigation menu, navigate to **Platform > Security > Banned List** to go to IP/subnets on the banned list. You can search and sort the list. You can remove IP/subnets from the banned list.



Delete IP/Subnet from Banned List

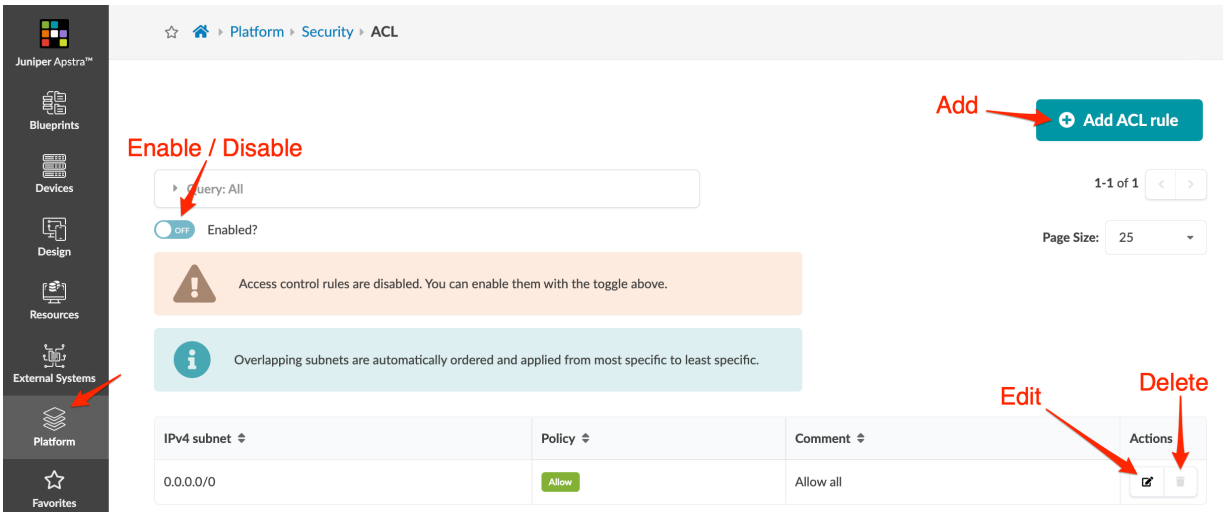
1. From the left navigation menu, navigate to **Platform > Security > Banned List** and click the **Delete** button to the right of the IP/subnet(s) to delete.
2. Click **Delete** to remove the IP/subnet from the banned list and immediately allow logins from that IP/subnet.

ACL Rules

IN THIS SECTION

Overview

Subnet-based access control for Apstra GUI access (whitelisting) is part of platform security enhancements. You can configure Access Control List (ACL) rules for IPv4 networks. (IPv6 is not supported on the Apstra web framework.) When you create and enable rules, the rules are automatically sorted from more specific to less specific, and IP addresses are checked against them in that order. If the rule allows access to a subnet, any IP address within that subnet is allowed access. If the rule denies access to a subnet, any IP address within that subnet is denied access.



Enable / Disable ACL Rules

Access control list rules are disabled by default.

If you enable ACL rules, make sure you always add a rule to allow access to a subnet that your IP address is a part of, so you don't lock yourself out.

If you enable ACL rules, and the default rule (0.0.0.0/0) is set to deny, the Apstra UI and system agents can't make necessary REST API calls to the Apstra controller unless you add a rule to allow access from loopback (127.0.0.0/8) and docker (172.17.0.0/16) networks.

1. From the left navigation menu, navigate to **Platform > Security > ACL** to go to the table view.
2. Click the toggle to enable or disable the rules, as applicable.

Add ACL Rule

1. From the left navigation menu, navigate to **Platform > Security > ACL** and click **Add ACL rule**.
2. Enter an IP subnet and select whether to allow or deny access to IP addresses within that subnet. You also have the option of adding a comment.
3. Click **Create** to create the rule and return to the table view.

Edit ACL Rule

1. From the left navigation menu, navigate to **Platform > Security > ACL** and click the **Edit** button for the rule to edit.
2. Change the policy, as applicable. You also have the option of adding/editing/deleting a comment.
3. Click **Update** to change the rule and return to the table view.

Delete ACL Rule

So that an IP address eventually matches to a subnet, 0.0.0.0/0 can't be deleted..

1. From the left navigation menu, navigate to **Platform > Security > ACL** and click the **Delete** button for the rule to delete.
2. Click **Delete** to delete the rule and return to the table view.

Rate Limit Configuration

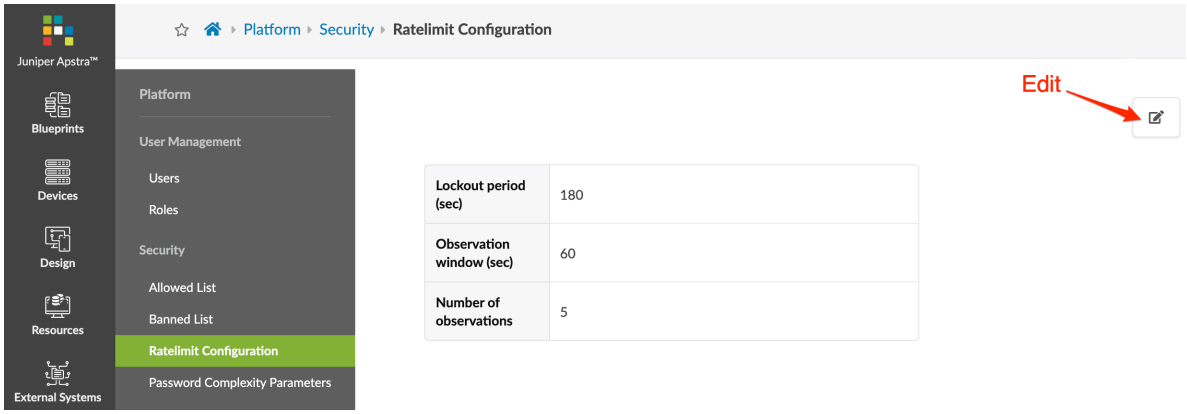
IN THIS SECTION

Rate Limit Configuration Overview

Default settings allow 5 login attempts within 60 seconds. After the fifth failed attempt, the IP/subnet is blocked and added to the banned list for 3 minutes (found at **Platform > Security > Banned List**), or until an admin removes it from the list. When you change rate limit configuration, any banned IP/subnets are immediately affected. For example, if you change the lockout period from 3 minutes to 5 minutes, an IP/subnet that's already on the banned list would remain on the banned list for an additional 2 minutes.

Edit Rate Limit Configuration

1. From the left navigation menu, navigate to **Platform > Security > Ratelimit Configuration** and click the **Edit** button (top-right).

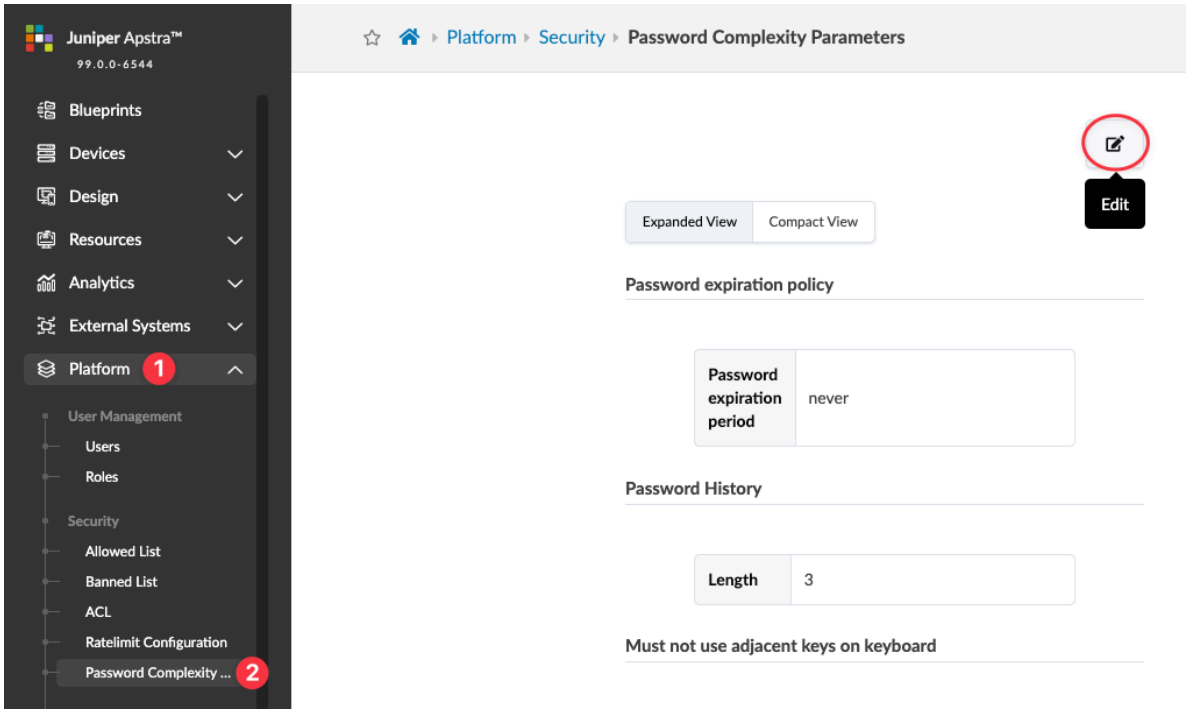


2. Change parameter values (lockout period, time period, number of attempts).
3. Click **Update** to complete the change and return to the Rate Limit Configuration page.

Update Password Complexity Requirements

When you update password complexity requirements, existing passwords are not affected. The updated requirements apply to passwords that are created or changed going forward.

1. From the left navigation menu, navigate to **Platform > Security > Password Complexity Parameters** and click the **Edit** button (top-right).



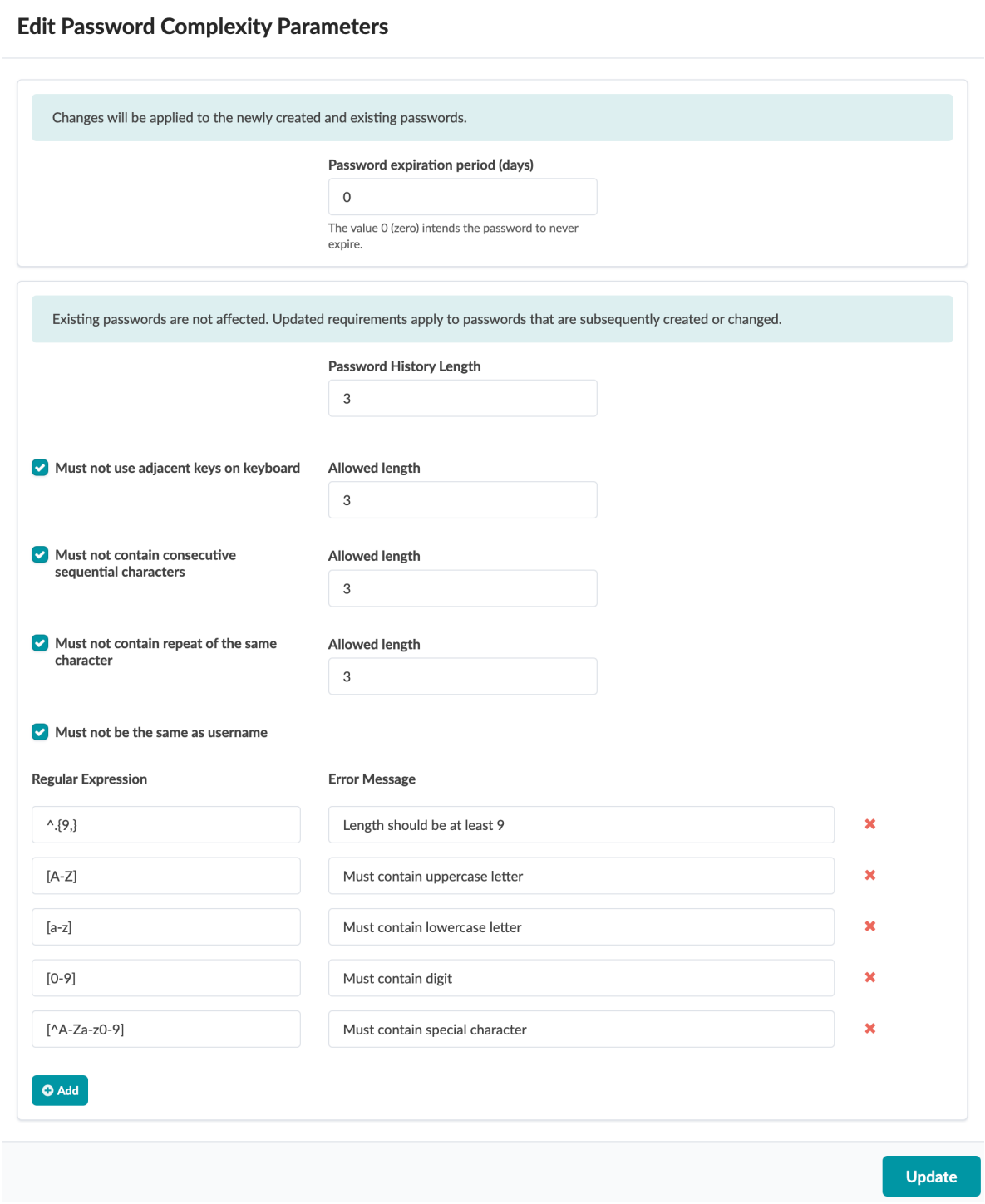
The **Edit Password Complexity Parameters** dialog opens.

2. Add, change and/or delete requirements, as applicable.

- **Password expiration policy** - To set the password to expire in a specific number of days enter the number of days (new in Apstra version 5.0.0).
- **Password History Length** - User is not allowed to re-use a certain number of previous passwords (including the current one). For example, if you don't want the user to use their previous two passwords, you would enter 3 in this field.
- Must not use adjacent keys on keyboard
- Must not contain consecutive sequential characters
- Must not contain repeat of the same character
- Must not be the same as username
- Length should be at least 9 (default)
- Must contain uppercase letter
- Must contain lowercase letter
- Must contain digit
- Must contain special character

For regular expressions:

- To add a rule, click **Add** and enter a regular expression and error message.
- To change a rule, change values as appropriate and update the error message.
- To delete a rule, click the red **X** to the right of the rule to delete.



3. Click **Update** to complete the change and close the dialog. When you create or update passwords going forward, the new requirements will take effect.

CHAPTER 47

External Services

IN THIS CHAPTER

Syslog Configuration (Platform)

IN THIS SECTION

Syslog Overview

System Log (syslog) is a running list of everything that's going on in your system. You can use these logs to audit events or review anomalies. You can configure syslog to send messages for specific types of systems (facilities) to external syslog servers. (You can also "export event logs to a CSV file" on page 1306.)

Syslog configuration includes the following details:

Name	Description
IP Address	The remote syslog server IP address or hostname
Port	The remote syslog server port

(Continued)

Name	Description
Protocol	UDP or TCP

(Continued)

Name	Description
Facility	The type of system that's logging the messages Facilities are mapped to Apstra syslogs as follows: • 0 - kern - kernal messages • 1 - user - user-level messages • 2 - mail - mail system • 3 - daemon - system daemons • 4 - auth - security/authentication messages • 5 - syslog - messages generated internally by syslogd • 6 - lpr - line printer subsystem • 7 - news - network news subsystem • 8 - uucp - UUCP subsystem • 10 - authpriv - security/authentication messages • 11 - ftp - FTP daemon • 15 - cron - Cron subsystem • 16 - local0 - locally used facilities • 17 - local1 - locally used facilities • 18 - local2 - locally used facilities • 19 - local3 - locally used facilities • 20 - local4 - locally used facilities • 21 - local5 - locally used facilities • 22 - local6 - locally used facilities • 23 - local7 - locally used facilities

(Continued)

Name	Description
Time Zone	The syslog message time zone. If you have proper time zone translation, you won't need to synch the system time zone (or Docker time zone) with your external syslog server. Rather than assuming the message time is in Zulu/UTC-0, the time zone translation needs to append the correct time zone information to the timestamp. Then, you can better correlate Apstra events in your external message systems.

Syslog messages follow Common Event Format (CEF) conventions as shown below:

NOTE: {host} is the the Apstra server hostname. If you want to change the hostname, you must use the procedure on the "Change Apstra Server Hostname" on page 1415 page. If you change the hostname with any other method, the new hostname won't be included in syslog entries.

```
AOS Log Format:

'{timestamp} {host}'
'CEF:{version}|{device_vendor}|{device_product}|{device_version}|'
'{device_event_class_id}|{name}|{severity}|{extension}

Where:

  {version}        : CEF version, currently always "0"
  {device_vendor}  : always "Apstra"
  {device_product} : always "AOS"
  {device_version} : current AOS version
  {device_event_class_id} : "100" for audit logs, "101" for anomaly logs
  {name}           : "Audit event" for audit logs, "Alert" for anomaly logs
  {severity}       : "5" for audit logs, "10" for anomaly logs

And where {extension} is either :

  For anomaly logs : msg=<json payload>
  For audit logs   : cat=<activity> src=<src_IP> suser=<username> act=<activity result>
cs1Label=<field1_type> cs1=<field1_value> cs2Label=<field2_type> cs2=<field2_value>
cs3Label=<field3_type> cs3=<field3_value>
```

```
Anomaly Log JSON Format

  blueprint_label : Name of the blueprint the anomaly was raised in.
  timestamp       : Unix timestamp when the Anomaly was raised.
  origin_name     : Serial Number of the device the anomaly affects.
  alert           : The value is a JSON Payload with the actual anomaly (see Alert JSON Payload
below)
  origin_hostname : Hostname of the device the anomaly affects. It can be AOSHOST, an empty
string if the hostname could not be determined or a valid value.
  device_hostname : Hostname of the device the anomaly affects or <device hostname unknown> if a
hostname could not be determined
  origin_role     : Role of the device the anomaly affects.

Alert JSON Payload:
<ALERT TYPE>_alert: Contains a JSON payload with key-value pair of information pertaining to the
alert. Here <ALERT TYPE>_alert can be valid anomaly/alert names such as hostname_alert,
probe_alert, liveness_alert etc.
  id           : UUID of the anomaly.
  first_seen      : Unix timestamp when the Anomaly was raised for the first time.
  raised          : True when anomaly is present, False when it is cleared.
  severity        : The severity level of the anomaly. Set to 3 for critical, 2 for high, 1 for
medium and 0 for low.

Audit Log Format:

  cat       : Activity performed. Valid values: "Login",
"Logout","BlueprintCommit","BlueprintRevert","BlueprintRollback",
"BlueprintDelete","DeviceConfigChange",

"OperationModeChangeToMaintenance","OperationModeChangeToNormal","OperationModeChangeToReadOnly",
"RatelimitExceptionAdd","RatelimitExceptionDelete",

"RatelimitClear","SystemChangeApiOperationModeToMaintenance","SystemChangeApiOperationModeToNorma
l","UserCrete","UserUpdate","UserDelete",

"SyslogCreate","SyslogUpdate","SyslogDelete","AuthAclEnable","AuthAclDisable","AuthAclRuleAdd","A
uthAclRuleUpdate" and "AuthAclRuleDelete".
  src       : Source IP of the client making HTTP requests to perform the activity.
  suser     : Who performed the activity.
  act       : Outcome of the activity - free-form string. In the case when the activity was
performed successfully, the value stored is “Success“. In case of error, include error string.
Ex: Unauthorized
  cs1Label  : The string “Blueprint Name”. Only exists if activity is associated with a
```

```
blueprint (optional)
  cs1       : Name of the blueprint on which action was taken. Only exists if activity is
associated with a blueprint (optional)
  cs2Label  : The string “Blueprint ID”. Only exists if activity is associated with a blueprint
(optional)
  cs2       : Id of the blueprint on which action was taken. Only exists if activity is
associated with a blueprint (optional)
  cs3Label  : The string “Commit Message”. Only exists if user has added a commit message
(optional)
  cs3       : Commit Message. Only exists if user has added a commit message (optional)
  deviceExternalId : Id (typically serial number) of the managed device on which action was
taken. Only exists if activity is associated with a device such as for “DeviceConfigChange”
(optional)
  deviceConfig  : Config that is pushed and applied on the device where “#012” is used to
indicate a line break to log collectors and parsers. Only exists if activity is associated with
a device such as for “DeviceConfigChange” (optional)
```

Example of Audit Syslog Message:

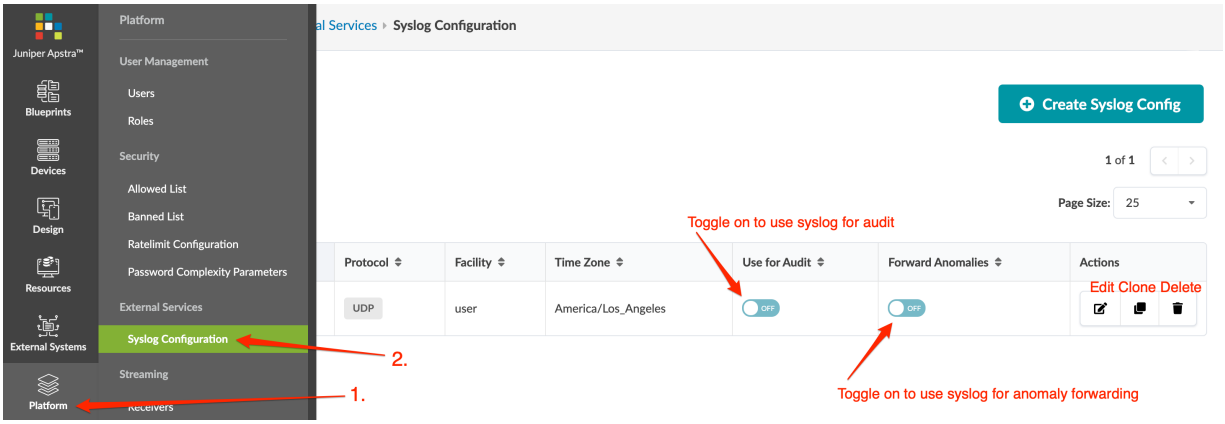
```
Jan 31 03:11:01 aos-server - 2023-01-31T03:11:01.699190+0000 aos-server
CEF:0|Apstra|AOS|4.1.2-269|100|Audit event|5|cat=Logout src=172.24.212.62 suser=admin act=Success

Jan 31 03:11:01 aos-server - 2023-01-31T03:11:01.699190+0000 aos-server
CEF:0|Apstra|AOS|4.1.2-269|100|Audit event|5|cat=BlueprintCommit src=172.24.212.62 suser=admin
act=Success cs1Label=Blueprint Name
cs1=rack-based-blueprint-33ded50f cs2Label=Blueprint ID cs2=rack-based-blueprint-33ded50f
```

Example of Anomaly Syslog Message:

```
Jan 31 03:11:01 aos-server - 2023-01-31T03:11:01.699190+0000 aos-server
CEF:0|Apstra|AOS|4.1.2-269|101|Alert|10|msg={u'blueprint_label': u'rack-based-
blueprint-33ded50f', u'timestamp': 1679002758562407, u'origin_name':
u'time_series', u'alert': {u'probe_alert': {u'expected_int_max': 99, u'stage_name':
u'leaf_match_perc_range', u'probe_label': u'leaf_to_spine_interface_statuses',
u'actual_int': 83, u'probe_id': u'60b03bb0-0e22-4a6d-b32d-e15085149b7b', u'key_value_pairs': [],
u'item_id': u'1', u'expected_int': -9223372036854775808},
u'first_seen': 1679002758562121, u'raised': False, u'severity': 3, u'id': u'02a17b60-cc3e-4afb-
baba-733a8c654df6'}, u'origin_hostname': u'AOSHOST',
'device_hostname': '<device hostname unknown>', u'origin_role': u''}

Jan 31 03:11:01 aos-server - 2023-01-31T03:11:01.699190+0000 aos-server
```

```
CEF:0|Apstra|AOS|4.1.2-269|101|Alert|10|msg={u'blueprint_label': u'rack-based-
blueprint-33ded50f', u'timestamp': 1679002754682990, u'origin_name':
u'50540015FA9D', u'alert': {u'first_seen': 1679002749600167, u'raised': False, u'severity': 3,
u'hostname_alert': {u'expected_hostname': u'leaf-3',
u'actual_hostname': u''}, u'id': u'0457a759-7d3a-4bf8-97e8-e13e518cf267'}, u'origin_hostname':
u'', 'device_hostname': '<device hostname unknown>', u'origin_role': u'leaf'}
```

From the left navigation menu, navigate to **Platform > External Services > Syslog Configuration** to see configurations. You can create, clone, edit and delete syslog configurations.



Create Syslog Config

1. From the left navigation menu, navigate to **Platform > External Services > Syslog Configuration** and click **Create Syslog Config** (top-right).
2. Configure the Syslog server. (See overview above for details.)
3. Click **Create** to save the configuration and return to the table view.
4. To configure another Syslog server, repeat the steps above.
5. To enable messages to be sent to a configured server, toggle on **Use for Audit** and/or **Forward Anomalies**, as appropriate.

Edit Syslog Config

1. From the left navigation menu, navigate to **Platform > External Services > Syslog Configuration** and click the **Edit** button for the Syslog configuration to edit.
2. Make your changes.
3. Click **Update** to update the Syslog configuration and return to the table view.

Delete Syslog Config

1. From the left navigation menu, navigate to **Platform > External Services > Syslog Configuration** and click the **Delete** button for the Syslog configuration to delete.
2. Click **Delete Syslog Config** to delete the Syslog configuration and return to the table view.

CHAPTER 48

Streaming

IN THIS CHAPTER

Receivers (Platform)

IN THIS SECTION

Streaming Receivers Overview

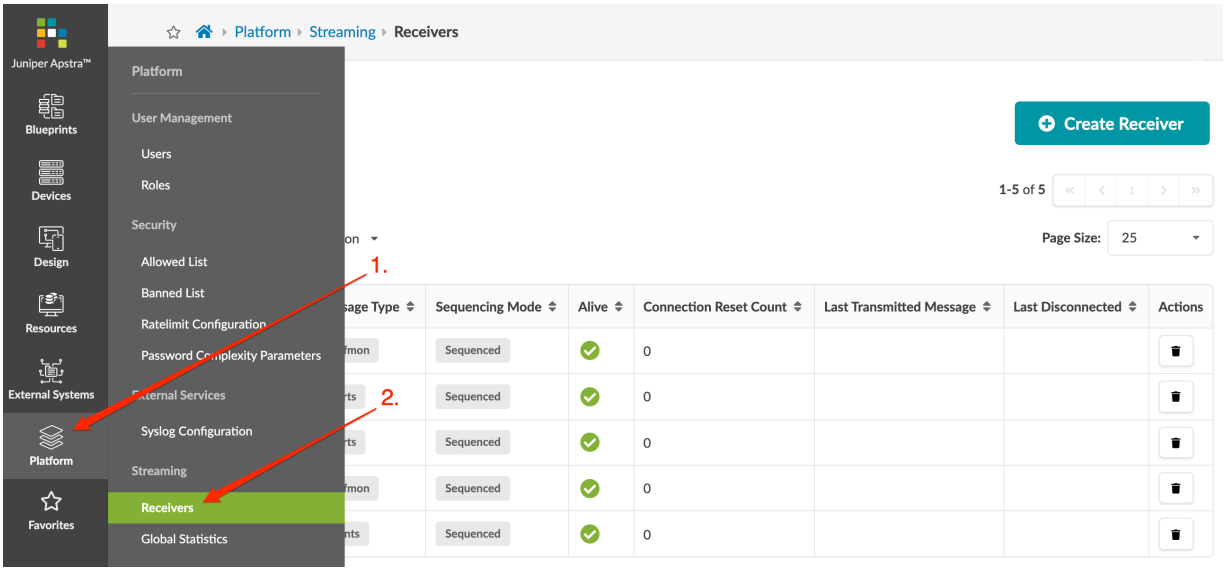
You can configure the Apstra server to stream alerts, events and perfmon, or any combination thereof. Each data type is sent to a streaming receiver over its own TCP socket. Even if all three data types are configured for the same streaming receiver, three (3) connections are created between the Apstra server and the streaming receiver. This also allows for all three types to be sent to three different streaming receivers.

Receivers include the following details:

- **Hostname** - Hostname
- **Port** - default: 4444
- **Message Type** - alerts, events, perfmon

- **Sequencing Mode** - unsequenced, sequenced

From the left navigation menu, navigate to **Platform > Streaming > Receivers** to go to receivers. You can create and delete receivers.



Create Receiver

1. From the left navigation menu of the Apstra GUI, navigate to **Platform > Streaming > Receivers** and click **Create Receiver**.
2. Enter/select required values.
3. Click **Create** to create the receiver and return to the table view.

Delete Receiver

1. From the left navigation menu of the Apstra GUI, navigate to **Platform > Streaming > Receivers** and click the delete button for the receiver to delete.
2. Click **Delete** to delete the receiver from the system and return to the table view.

Configure Receivers Using Telegraf Plugin

You can use the Apstra Telegraf input plugin to receive streaming telemetery from Apstra. Telegraf is an agent for collecting, processing, aggregating, and writing metrics. This is the component of AOSOM-Streaming that handles the reception of the protobuf messages from the Apstra environment. For more information, see the "AOSOM Streaming Guide" on page 1466. The Telegraf platform consists of input and output plugins that you can choose from for aggregating and storing metrics to different backend databases. The Apstra input plugin for Telegraf deserializes the protobuf stream and creates metrics that can then be sent to a particular backend database, such as Prometheus, InfluxDB, or Elasticsearch.

The configuration described here assumes you are using the Apstra Telegraf input plugin. You can configure streaming receivers in Apstra with the Telegraf plugin by providing it Apstra credentials. We

recommend that you use a separate Apstra account with only the streaming credentials. If you configure through the GUI, then there is no need to supply credentials in the Telegraf config file.

The easiest way to run the Telegraf receiver is in a docker container. The `docker-compose.yml` snippet below shows the configuration for the Telegraf container. This pulls the latest Apstra supported Telegraf container from Docker Hub.

```
# Telegraf container config
  telegraf-prom:
    image: apstra/telegraf:latest
    command: telegraf
   volumes:
      - ./config/telegraf-prom.toml:/etc/telegraf/telegraf.conf
    ports:
      - '9999:9999'
```

The Telegraf configuration file - `./config/telegraf-prom.toml` - is mapped to `/etc/telegraf/telegraf.conf` on the container. It includes the following parameters:

- **address** - specifies the IP address of the streaming receiver
- **port** - specifies the port that the streaming receiver will be listening on
- **streaming_type** - specifies the type of data to be streamed from Apstra to this receiver

The remaining parameters are only necessary if you want the Apstra Telegraf plugin to configure the streaming receivers in Apstra via the API.

- **aos_server** - specifies the IP address of the Apstra server
- **aos_port** - should always be 443
- **aos_login** - Apstrs username
- **aos_password** - Apstra password

The input and output plugin configurations are shown in the snippet below. The output plugin is configured for the Prometheus client and listens on port 9126. The input plugin is configured for Apstra.

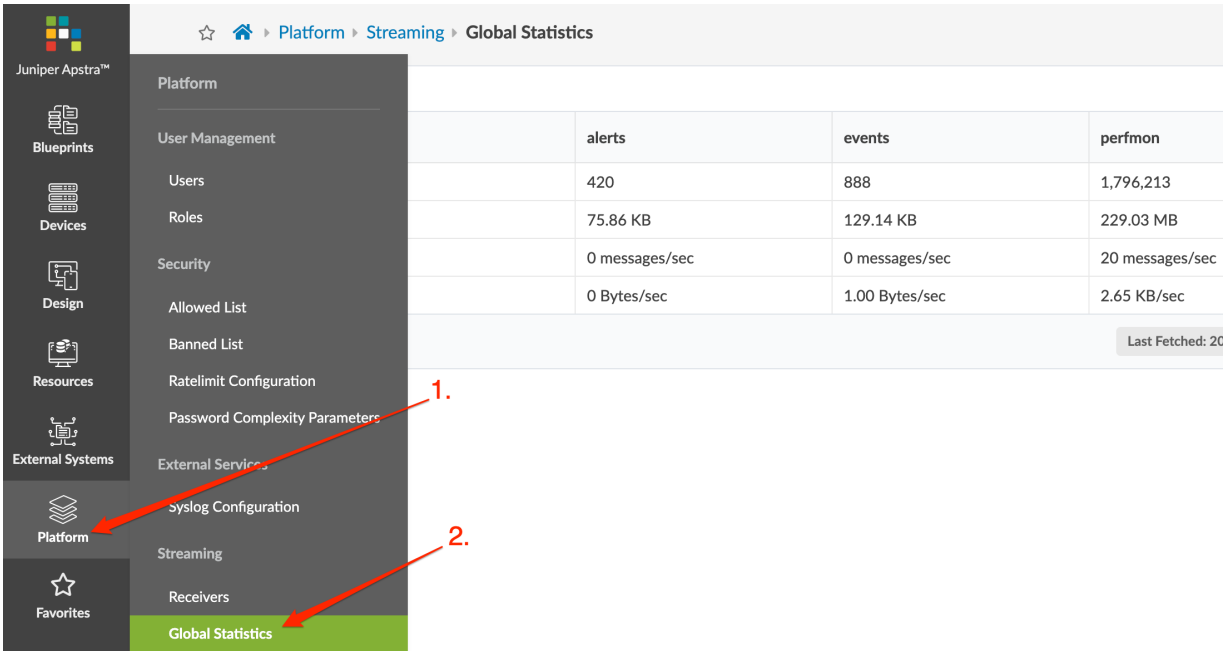
```
# Configuration for Prometheus server to expose metrics
[[outputs.prometheus_client]]
  listen = ":9126"
  expiration_interval = "0"

[[inputs.aos]]
```

```
address = "10.1.1.200"
port = 9999
streaming_type = [ "perfmon", "alerts", "events" ]
aos_server = "$AOS_SERVER"
aos_port = $AOS_PORT
aos_login = "$AOS_LOGIN"
aos_password = "$AOS_PASSWORD"
```

Global Statistics (Platform)

Global statistics include information that is unrelated to any specific receiver. These statistics provide crucial information required for better planning of receivers. Whenever you reset the Apstra server, these global statistics are reset.

From the left navigation menu, navigate to **Platform > Streaming > Global Statistics** to see global statistics.



Event Log (Audit Log)

IN THIS SECTION

Event Log Introduction

IN THIS SECTION

As users work in the Apstra environment their actions are logged. These event logs are useful when investigating general usage, network outages, and possible suspicious activity. See below for the information that's collected.

Types of Events that are Logged

Events for the following event types are logged:

Table 55: Event Types

Event	Description
Login	A user logged in (success and failure).

Table 55: Event Types *(Continued)*

Event	Description
Logout	A user logged out.
BlueprintCommit	Changes were applied from the staged blueprint to the active blueprint.
BlueprintRevert	Changes in the staged blueprint were discarded.
BlueprintRollback	The staged blueprint was rolled back to a previous version.
BlueprintDelete	The entire blueprint was deleted.
DeviceConfigChange	The configuration of a device was changed. This includes any configuration change that Apstra pushes to any managed device (including Time Voyager). The event is attributed to the logged-in user making the change.
MigrationCheckpoint	When you upgrade the Apstra server on a new VM (VM-VM) to versions 5.0.0 and higher, a MigrationCheckpoint event is logged. This event includes upgrade details, such as the Apstra version of the old Apstra server (source), the Apstra version of the new Apstra server (target), and a timestamp from during the upgrade process. The upgrade is followed up by a Metricdb data migration and upon a successful migration on the new VM, the Metricdb data before the timestamp is from the old VM and data after the timestamp is from the new VM.
OperationModeChangeToMaintenance	The blueprint operation mode was changed to Maintenance by a user.
OperatonModeChangeToNormal	The blueprint operation mode was changed to Normal by a user, or by the system when disk usage and memory are under the utilization threshold (the operation is in read/write mode).
OperationModeChangeToReadOnly	The blueprint operation mode was changed to Read Only by the user, or by the system when the utilization threshold is surpassed (the operation is in read only mode).
RatelimitExceptionAdd	A ratelimit exception was added.

Table 55: Event Types *(Continued)*

Event	Description
RatelimitExceptionDelete	A ratelimit exception was deleted.
RatelimitClear	A ratelimit was cleared.
SyslogCreate	Syslog was created.
SyslogUpdate	Syslog was updated.
SyslogDelete	Syslog was deleted.
UserCreate	A user profile was created (by creating or cloning).
UserUpdate	A user profile was updated.
UserDelete	A user profile was deleted.
AuthAclEnable	Access control rules were enabled.
AuthAClDisable	Access control rules were disabled.
AuthAclRuleAdd	An access control rule was added.
AuthAclRuleUpdate	An access control rule was updated.
AuthAclRuleDelete	An access control rule was deleted.

Types of Event Details that are Collected

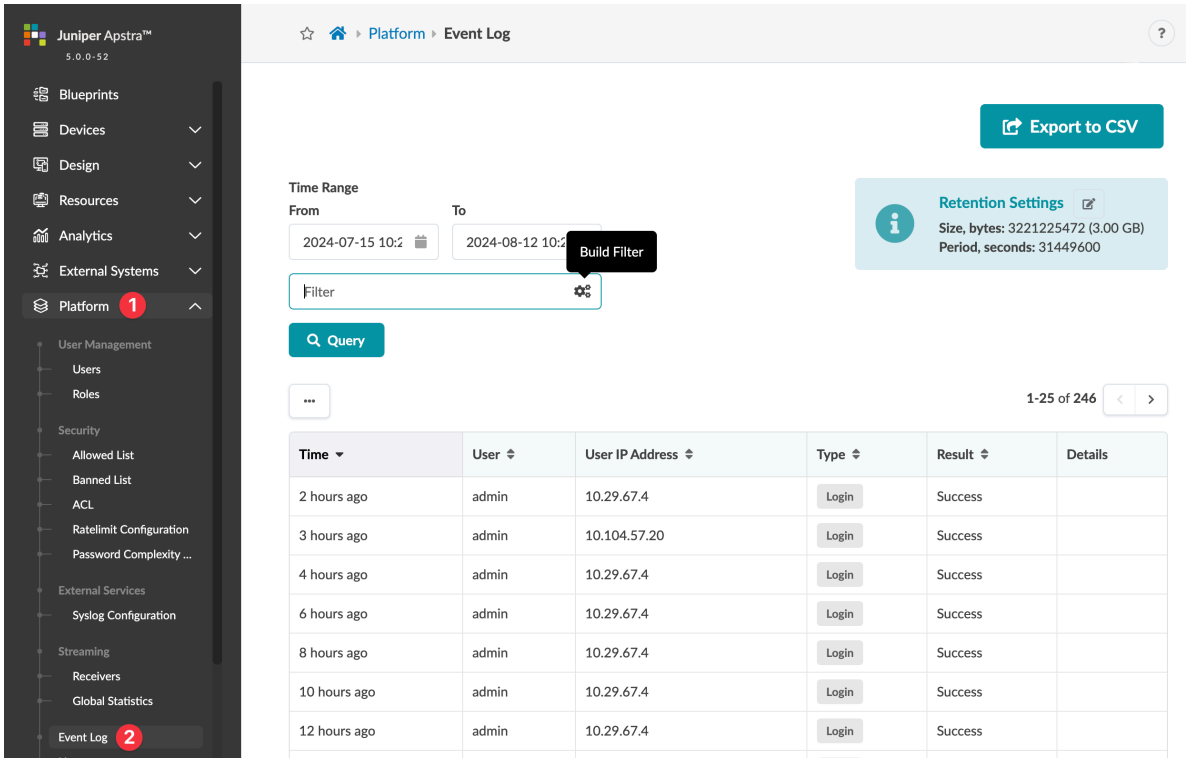
The following details are logged for each event (as applicable):

Table 56: Event Details

Property	Description
Time Range	The timeframe of when the event occurred (hover over time field to see date and time).
User	The user who performed the activity, the **system** or a username such as **admin**.
User IP Address	The IP address associated with the user who made the change.
Type (of Event)	The type of event (listed in table above).
Blueprint Name (Blueprint ID)	The ID of the blueprint where the change was made.
Blueprint Commit Message	The description of the changes that were committed to the blueprint, if provided.
Device Key (Device ID)	Typically, the serial number of the managed device where the change was made.
Device Configuration	The configuration that's pushed and applied to the device.
Result	The outcome of the activity. Success means operation is accepted by the system. In the case of an error, the error string is included (unauthorized, for example).

Search Event Logs

1. From the left navigation menu, navigate to **Platform > Event Log** to go to the table of logged events.



2. The table displays the 25 most recent events, by default. To change the number of events that are displayed, click the **Table settings** button (three dots below the **Query** button) and select a number from the drop-down list.
3. Click the **Query Builder**, enter/select your query from the available search fields. You can enter multiple values in some fields; events that match criteria for all fields are returned. Click **Apply** for results.

NOTE: The event log is based on the number of events. Up to 10,000 events are retained. They're written to log-rotated files as a second repository. You can configure logrotate parameters in the Apstra server configuration file (`/etc/aos/aos.conf`).

4. Click **Confirm** for results.

In addition to using the Apstra GUI to search for events as described above, you can also use API (/api/audit/events).

Export Event Log to CSV File

1. From the left navigation menu, navigate to **Platform > Event Log** and click **Export to CSV** (top-right).
2. To filter the data to export, enter your query.
3. Click **Save as CSV File** to download the CSV file.

Send Event Log to External Syslog Server

For details about sending the event log to an external system with the Syslog protocol, see "Syslog Configuration" on page 1290.

Parse Apstra Logs

IN THIS SECTION

Apstra uses Common Event Format (CEF), a standard for the interoperability of event or log-generating devices and applications. The standard defines a syntax for log records. It comprises a standard prefix and a variable extension formatted as key-value pairs.

Apstra Log Format

```
'{timestamp} {host} '
       'CEF:{version}|{device_vendor}|{device_product}|{device_version}|'
       '{device_event_class_id}|{name}|{severity}|{extension}
```

Where:

- `version` is always “0”
- `device_vendor` is always “Apstra”
- `device_product` is always “Apstra”
- `device_version` is the current Apstra version
- `device_event_class_id` is “100” for audit logs and “101” for anomaly logs
- `name` is always “Audit even” for audit logs and “Alert” for anomaly logs
- `severity` is always “medium” for audit logs and “Very-High” for anomaly logs

And where:

- {extension} is either:
  - For anomaly logs: msg=<json payload>
  - For audit logs: cat=<activity> src=<src_IP> suser=<username> act=<activity result> cs1Label=<field1_type> cs1=<field1_value>cs2Label=<field2_type> cs2=<field2_value> cs3Label=<field2_type> cs2=<field2_value>

Audit Log Fields

Table 57: Audit Log Fields

Field	Description	Applies to
cat	Activity performed. Valid values: “Login”, “Logout”, “BlueprintCommit”, “DeviceConfigChange”, “BlueprintDelete”.	All messages
src	Source IP of the client making HTTP requests	All messages

suser	Who performed the activity	All messages
act	Outcome of the activity - free-form string. “Success” means operation is accepted by system. In case of error, include error string. Ex: Unauthorized	All messages
cs1Label	The string “Blueprint Name”	Cat = “BlueprintCommit” or “BlueprintDelete”
cs1	Name of the blueprint on which action was taken.	Cat = “BlueprintCommit” or “BlueprintDelete”
cs2Label	The string “Blueprint ID”	Cat = “BlueprintCommit” or “BlueprintDelete”
cs2	Id of the blueprint on which action was taken.	Cat = “BlueprintCommit” or “BlueprintDelete”
cs3Label	The string “Commit Message”. Only exists if user has added a commit message (optional)	Cat = “BlueprintCommit” or “BlueprintDelete”
cs3	Commit Message. Only exists if user has added a commit message (optional)	Cat = “BlueprintCommit”
deviceExternalId	Id (typically serial number) of the managed device on which action was taken.	Cat = “DeviceConfigChange”
deviceConfig	Config that is pushed and applied on the device where “#012” is used to indicate a line break to log collectors and parsers.	Cat = “DeviceConfigChange”

Anomalies JSON Fields

Table 58: Anomalies JSON Fields Table

Field	Description	Applies to
u'blueprint_label'	String. Name of the blueprint the anomaly was raised in.	All messages
u'timestamp'	String. Name of the blueprint the anomaly was raised in.	All messages

u'origin_name'	String. Name of the blueprint the anomaly was raised in.	All messages
u'alert'	The value is a JSON Payload with the actual anomaly (see next table)	
u'origin_hostname'	String. Hostname of the device the anomaly affects.	All messages
u'device_hostname'	String. Hostname of the device the anomaly affects.	All messages
u'origin_role	String. Hostname of the device the anomaly affects.	All messages

Table 59: Main Msg Format

Field	Description	Applies to
u'first_seen'	String. Unix timestamp when the Anomaly was raised for the first time.	All messages
u'raised'	Always True	All messages
u'severity	The severity level of the anomaly. In Apstra today, all anomalies are raised with severity level 3.	All messages

Anomaly Log Examples

IBA Anomaly “MLAG Anomaly”

The device_event_class_id = 101 for all anomalies

```
06 04 2020 08:42:50 10.23.59.188 <SLOG:INFO> 1 2020-06-04T13:26:54.195385Z aos-server - - -
2020-06-04T13:26:54.194168+0000 aos-server CEF:0|Apstra|Apstra|3.2.2-12|101|Alert|Very-High|
msg={u'blueprint_label': u'LAB', u'timestamp': 1591277214194168, u'origin_name': u'FDO21260P7L',
u'alert': {u'first_seen': 1591277214194141, u'raised': True, u'severity': 3, u'mlag_alert':
{u'peer_link_status': u'down', u'actual_domain_state': 1, u'mlag_id': 0, u'expected_intf_state':
0, u'hostname': u'USDAL1-LAB93108-LF1', u'peer_link': u'port-channel3',
u'expected_peer_link_status': u'up', u'actual_intf_state': 0, u'expected_domain_state': 4,
u'ifname': u'', u'domain_id': u'1'}, u'id': u'6656a961-3139-4825-b89d-93f071271891'},
u'origin_hostname': u'LAB1_HOST1', 'device_hostname': 'LAB1_HOST1', u'origin_role': u'leaf'}
```

IBA Anomaly “Unexpected Hostname”

```
Jun  8 21:35:25 aos-server - 2020-06-08T21:35:25.757009+0000 aos-server CEF:0|Apstra|Apstra|
3.3.0-299|101|Alert|Very-High|msg={u'blueprint_label': u'test', u'timestamp': 1591652125757009,
u'origin_name': u'505400C5CAAA', u'alert': {u'first_seen': 1591652125757001, u'raised': True,
u'severity': 3, u'hostname_alert': {u'expected_hostname': u'spine1', u'actual_hostname':
u'localhost'}, u'id': u'7f693f1d-2aeb-44a4-93f1-656400cfff7e'}, u'origin_hostname':
u'localhost', 'device_hostname': 'localhost', u'origin_role': u''}
```

User Logout and Logging

The device_event_class_id = 100 for all events

```
Jun  8 19:43:33 aos-server - 2020-06-08T19:43:33.392984+0000 aos-server CEF:0|Apstra|Apstra|
3.3.0-299|100|Audit event|medium|cat=Logout src=10.1.253.6 suser=admin act=Success
Jun  8 19:43:39 aos-server - 2020-06-08T19:43:39.267262+0000 aos-server CEF:0|Apstra|Apstra|
3.3.0-299|100|Audit event|medium|cat=Login src=10.1.253.6 suser=admin act=Success
```

Blueprint Delete

```
Jun  8 21:23:41 aos-server - 2020-06-08T21:23:41.426107+0000 aos-server CEF:0|Apstra|Apstra|
3.3.0-299|100|Audit event|medium|cat=BlueprintDelete src=10.1.253.6 suser=admin act=Success
cs1Label=Blueprint Name cs1=test cs2Label=Blueprint ID cs2=2bd8f38f-9242-461c-855e-8146a4f68bb9
```

Blueprint Commit

```
Jun  8 21:42:19 aos-server - 2020-06-08T21:42:19.550216+0000 aos-server CEF:0|Apstra|Apstra|
3.3.0-299|100|Audit event|medium|cat=BlueprintCommit src=10.1.253.6 suser=admin act=Success
cs1Label=Blueprint Name cs1=test cs2Label=Blueprint ID cs2=5ba55c14-6c01-4537-9dd7-d32c8c41616b
cs3Label=Commit Message cs3=New_Virtual_Network
```

Device Config Change

Revert a full day-0 BP deployment

```
Jun  8 21:35:27 aos-server - 2020-06-08T21:35:27.132831+0000 aos-server CEF:0|Apstra|Apstra|
3.3.0-299|100|Audit event|medium|cat=DeviceConfigChange src=10.1.253.6 suser=admin act=Success
deviceExternalId=505400C5CAAA deviceConfig=
...
<Device Config, see next table>
...
```

Device Config

Note that “#012” is used to indicate a line break

```
service interface inactive expose#012
!#012
spanning-tree mode none#012
!#012
hostname spine1#012
interface Ethernet1#012
  description facing_l2-virtual-ext-001-leaf1:Ethernet1#012
  no switchport#012
  ip address 203.0.113.4/31#012
  no shutdown#012
  description facing_l2-virtual-ext-002-leaf1:Ethernet1/1#012
  no switchport#012
  ip address 203.0.113.6/31#012
  no shutdown#012
  exit#012
!#012
interface Ethernet3#012
  description facing_l2-virtual-ext-003-leaf1:Ethernet1/1#012
  no switchport#012
  ip address 203.0.113.8/31#012
  no shutdown#012
  exit#012!#012
interface Ethernet4#012
  description facing_l2-virtual-ext-004-leaf1:Ethernet1#012
  no switchport#012
  ip address 203.0.113.10/31#012
```

```
  no shutdown#012
  exit#012!#012
interface Ethernet5#012
  no switchport#012
  no shutdown#012
  exit#012
!#012
interface Ethernet6#012
  no switchport#012
  no shutdown#012
  exit#012
!#012
interface Ethernet7#012
  no switchport#012
  no shutdown#012
  exit#012
!#012
ip routing#012!#012
service routing protocols model multi-agent#012
interface loopback 0#012
  ip address 203.0.113.20/32#012
  exit#012
!#012
ip prefix-list AllPodNetworks seq 5 permit 0.0.0.0/0 le 32#012
ip as-path access-list MyASN permit ^$#012
route-map AllPodNetworks permit 10#012
  match ip address prefix-list AllPodNetworks#012
  exit#012
!#012
route-map EVPN permit 10#012
  set ip next-hop unchanged#012
  exit#012
!#012
router bgp 4200000000#012
  router-id 203.0.113.20#012
  no bgp default ipv4-unicast#012
  bgp log-neighbor-changes#012
  bgp bestpath as-path multipath-relax#012
  redistribute connected route-map AllPodNetworks#012!#012
  neighbor l3clos-s peer-group#012
  neighbor l3clos-s timers 1 3#012
  neighbor l3clos-s soft-reconfiguration inbound#012
  neighbor l3clos-s maximum-routes 0 warning-limit 90 percent#012
```

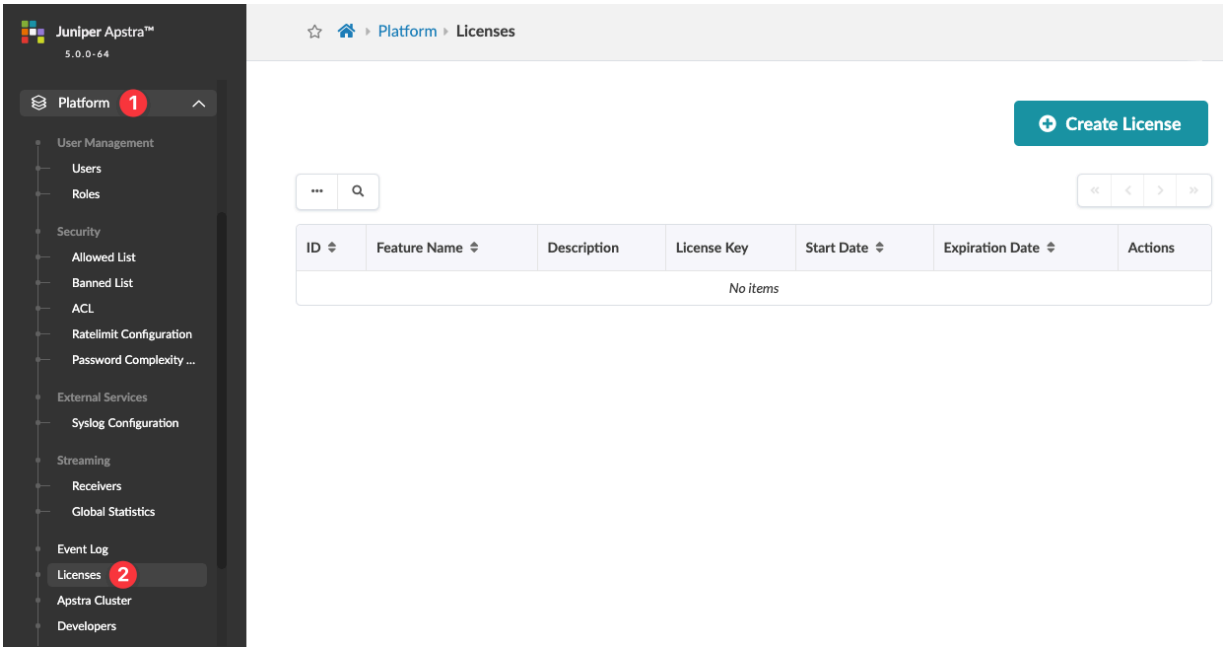
```
  neighbor l3clos-s-evpn peer-group#012
  neighbor l3clos-s-evpn ebgp-multihop 2#012
  neighbor l3clos-s-evpn timers 1 3#012
  neighbor l3clos-s-evpn send-community extended#012
  neighbor l3clos-s-evpn soft-reconfiguration inbound#012
  neighbor l3clos-s-evpn update-source loopback0#012
  neighbor l3clos-s-evpn maximum-routes 0 warning-limit 90 percent#
!#012
!#012
  neighbor 203.0.113.0 remote-as 64512#012
  neighbor 203.0.113.0 peer-group l3clos-s-evpn#012
  neighbor 203.0.113.0 description facing_l2-virtual-ext-001-leaf1-evpn-overlay#012
  neighbor 203.0.113.5 remote-as 64512#012
  neighbor 203.0.113.5 peer-group l3clos-s#012
  neighbor 203.0.113.5 description facing_l2-virtual-ext-001-leaf1#012
  neighbor 203.0.113.1 remote-as 64513#012
  neighbor 203.0.113.1 peer-group l3clos-s-evpn#012
  neighbor 203.0.113.1 description facing_l2-virtual-ext-002-leaf1-evpn-overlay#012
  neighbor 203.0.113.7 remote-as 64513#012
  neighbor 203.0.113.7 peer-group l3clos-s#012
  neighbor 203.0.113.7 description facing_l2-virtual-ext-002-leaf1#012
  neighbor 203.0.113.2 remote-as 64514#012
  neighbor 203.0.113.2 peer-group l3clos-s-evpn#012
  neighbor 203.0.113.2 description facing_l2-virtual-ext-003-leaf1-evpn-overlay#012
  neighbor 203.0.113.9 remote-as 64514#012
  neighbor 203.0.113.9 peer-group l3clos-s#012
  neighbor 203.0.113.9 description facing_l2-virtual-ext-003-leaf1#012
  neighbor 203.0.113.3 remote-as 64515#012
  neighbor 203.0.113.3 peer-group l3clos-s-evpn#012
  neighbor 203.0.113.3 description facing_l2-virtual-ext-004-leaf1-evpn-overlay#012
  neighbor 203.0.113.11 remote-as 64515#012
  neighbor 203.0.113.11 peer-group l3clos-s#012
  neighbor 203.0.113.11 description facing_l2-virtual-ext-004-leaf1#012
  address-family evpn#012
    neighbor l3clos-s-evpn route-map EVPN out#012
    neighbor 203.0.113.0 activate#012
    neighbor 203.0.113.1 activate#012
    neighbor 203.0.113.2 activate#012
    neighbor 203.0.113.3 activate#012
    exit#012
  address-family ipv4#012
    neighbor 203.0.113.11 activate#012
    neighbor 203.0.113.5 activate#012
```

```
    neighbor 203.0.113.7 activate#012
    neighbor 203.0.113.9 activate#012
    exit#012
  maximum-paths 32#012
  exit#012
```

Licenses

Entering a license key is required when you're using Juniper Apstra Flow.

To go to the licenses page, from the left navigation menu, navigate to **Platform > Licenses**.



For more information, see the Juniper Apstra Flow Installation and Upgrade Guide.

CHAPTER 49

Apstra VM Clusters

IN THIS CHAPTER

Apstra VM Clusters

You can monitor and manage different aspects of the Apstra environment, such as its configuration, usage, and containers. If your network includes many devices with offbox agents, or if you are taking advantage of Apstra's Intent Based Analytics feature, you might need more resources than can be provided from just one virtual machine (VM). To increase resource capacity, you can add worker node VMs to create a cluster with the Apstra controller node VM.

Apstra Cluster Nodes

IN THIS SECTION

Nodes Overview

The Apstra controller acts as the cluster manager. When you add a worker VM to the main Apstra controller VM, it registers with the Apstra server VM through sysDB. It collects facts about the VM (such as core/memory/disk configuration and usage), and launches a local VM container. The Apstra controller VM reacts to REST API requests, configures the worker VM for joining or leaving the cluster, and keeps track of cluster-wide runtime information. It also reacts to container configuration entities and schedules them to the worker VM.

Apstra VM nodes include the following details:

Table 60: Apstra VM Nodes Parameters

Name	Description
Address	IP address or Fully-Qualified Domain Name (FQDN) of the VM
Name	Apstra VM name, such as **controller** (the main Apstra controller node) or **worker - iba** (a worker node)
State	ACTIVE, MISSING, or FAILED
Roles	Controller or worker
Tags	The controller node and any worker nodes that you add are tagged with `iba` and `offbox`, by default. If you delete one or both of these tags or delete a worker node with one or both of these tags, any IBA and/or offbox containers in that node automatically move to a VM with those tags. Make sure there is another node with the tag(s) you're deleting or the containers will be deleted when you delete the tag or node.

Table 60: Apstra VM Nodes Parameters *(Continued)*

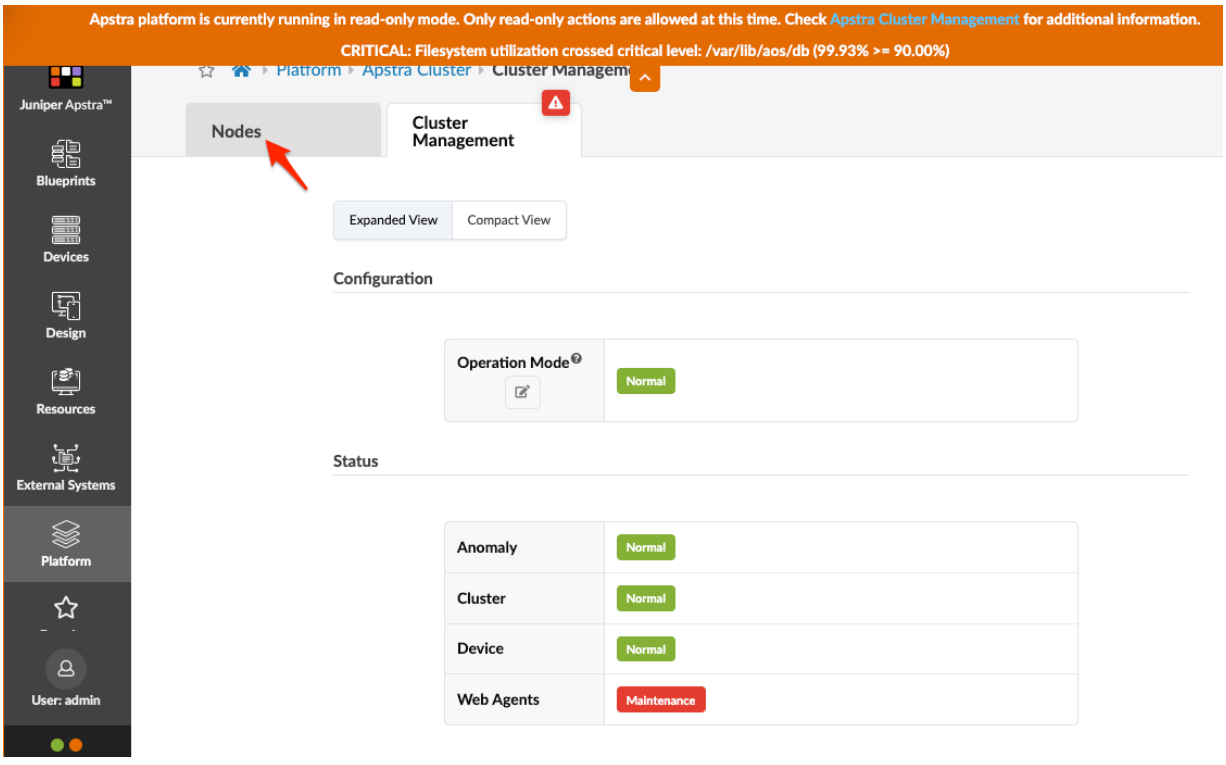
Name	Description
Capacity Score	Apstra uses the capacity score for load balancing new containers across the cluster of available nodes. It's calculated in relation to the configured application weight of each container based on allocated memory. Example calculation - 64GB of memory allocated for the VM and an application weight of 500MB configured for offbox agents: • Each offbox agent has a capacity score cost of 5 • (64GB / 500MB) * 5 capacity score of each offbox agent = 640 total capacity score • Controller nodes have half the capacity score available due to overhead (640 / 2 = 320 in above example) but worker nodes have the full capacity score available (640 in above example) The capacity score changes only if the memory allocated to the VM is changed, or if the application weight is changed.
Containers Count	Number of containers
CPU	Number of CPUs
Errors	As applicable. An example of an error is when an agent process has restarted because an agent has crashed.
Usage*	• Memory Usage (percentage) • CPU Usage (percentage) • Disk Usage - Current VM disk usage per logical volume (GB and percentage) • Container Service Usage - derived from the required resources and the size of the container. For example, if an offbox agent that needs 250 MB is running in a 500MB *worker* node, the container service usage is 50%. (An IBA container may require 1GB.) A *controller* node begins at 50% usage because it includes its own processing agents that perform controller-specific processing logic.
Containers	The containers running on the node and the resources that each container uses

Table 60: Apstra VM Nodes Parameters *(Continued)*

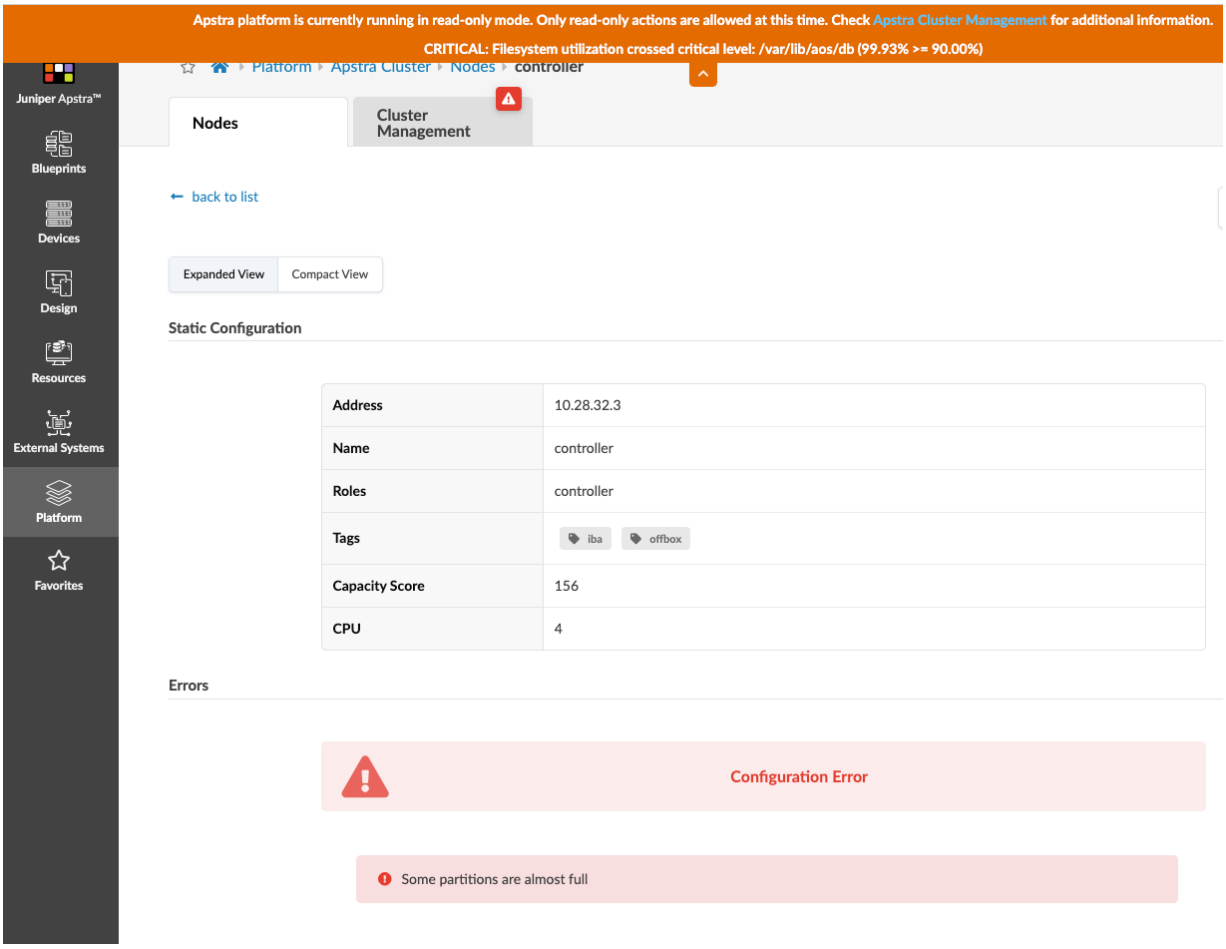
Name	Description
Username/ Password	Apstra Server VM SSH username/password login credentials

* If memory utilization exceeds 80%, a warning message appears at the top of all GUI pages. This lets you know that you need to free up or add disk space and/or memory soon, to avoid a critical resource shortage.

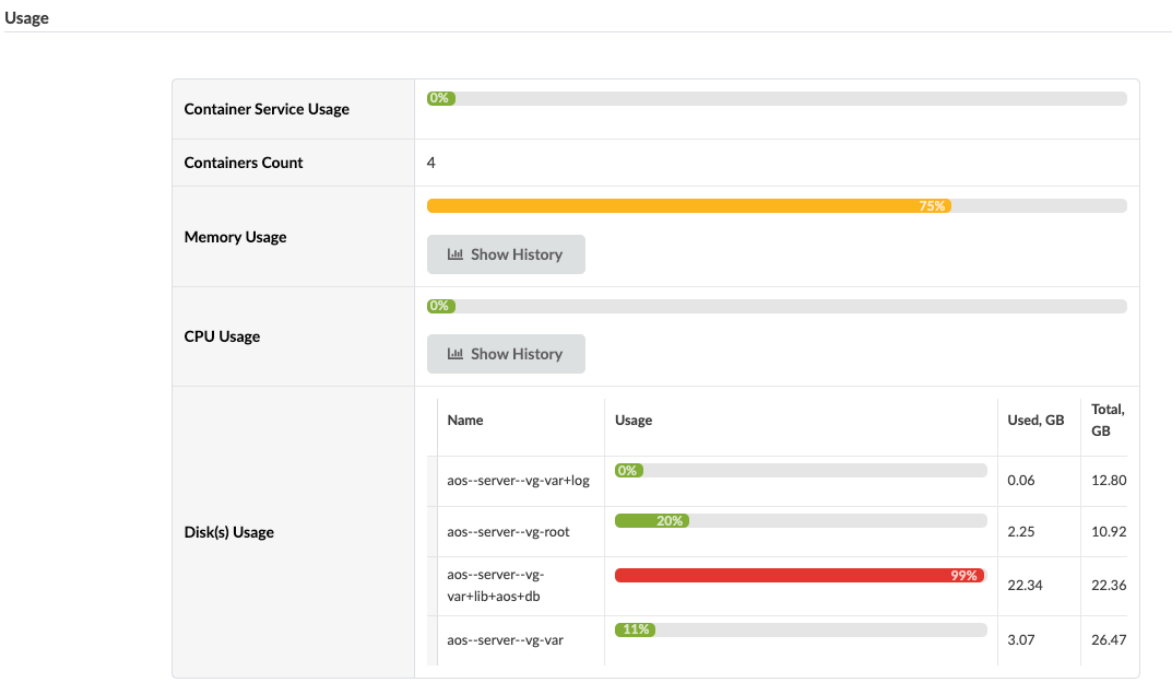
If memory utilization exceeds 90%, a critical message appears at the top of all GUI pages. Before you can make any more changes to the fabric, you must address the shortage by adding disk space to the problematic filesystem(s) or by adding memory, as needed. You can click the link to go to **Apstra Cluster Management** for more information.



Click the **Nodes** tab, then click the IP address of the controller for details.



Scroll down to see usage.

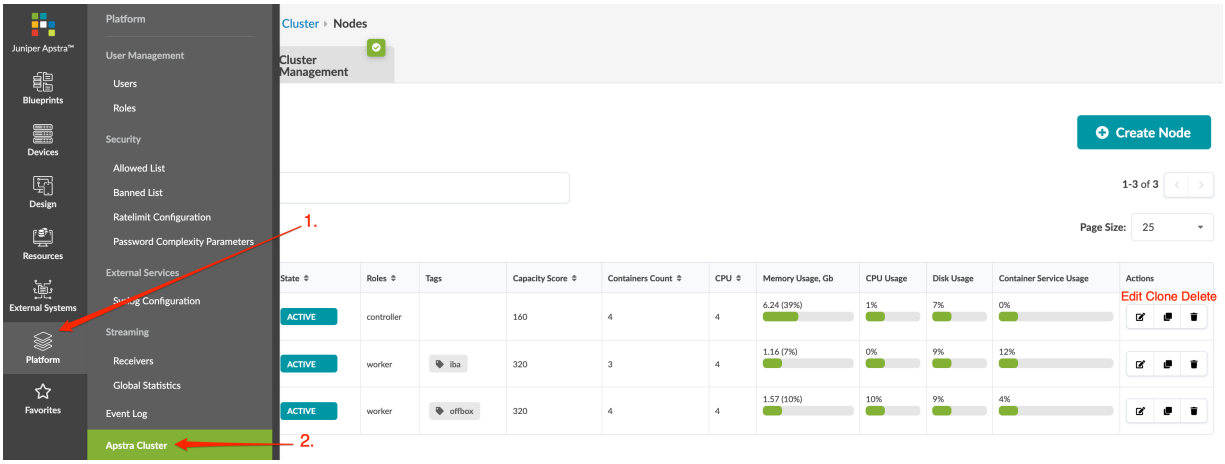


Some suggestions for recovering resources are as follows:

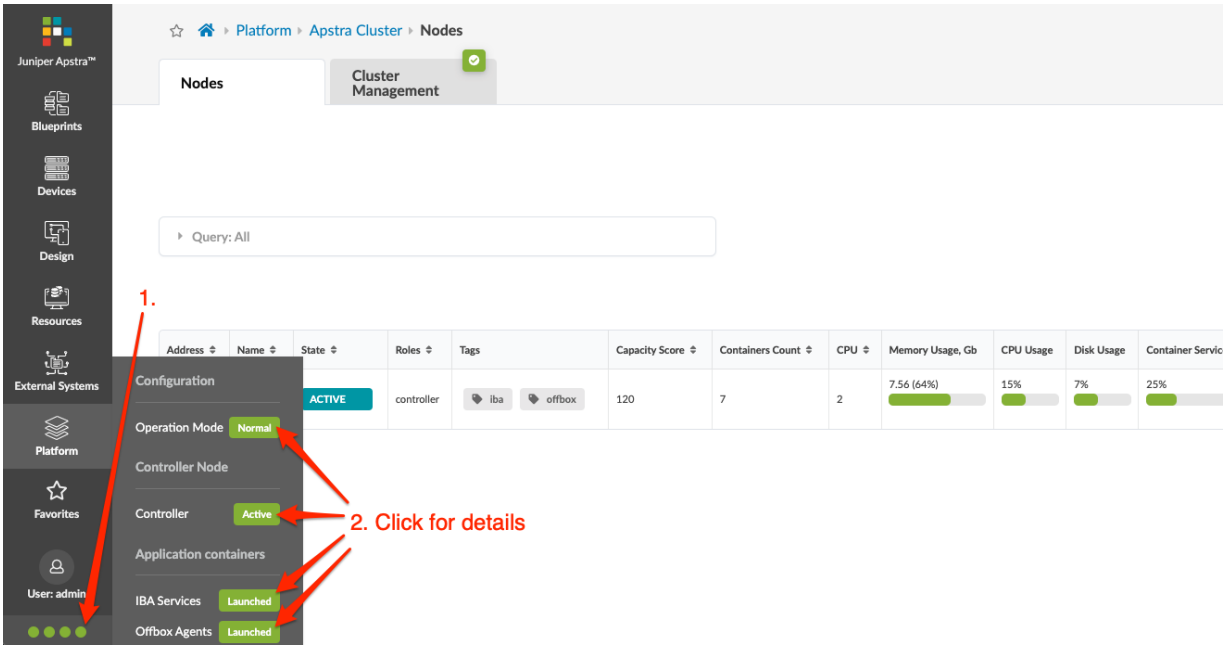
- Remove the **iba** tag from the controller VM so that IBA units are rescheduled to worker nodes, thus reducing both memory and disk space usage.
- Create worker nodes to spread out the load for IBA units and/or offbox device agents.

You can change the default thresholds that trigger warnings and critical messages. In the "Apstra server configuration file" on page 1752 (`/etc/aos/aos.conf`) change the options for `system_operation_filesystem_thresholds` and/or `system_operation_memory_thresholds`. Then, send SIGHUP to the ClusterManager Agent. You can set disk space utilization thresholds on a per-filesystem basis. For example, you might want to be more conservative with `/var/lib/aos/db` which contains MainSysdb's persistence files and Time Voyager revisions, so crossing a lower usage threshold (such as 85%) triggers the read-only mode.

To access Apstra VMs, from the left navigation menu, navigate to **Platform > Apstra Cluster**. Click a node address to see its details. You can create, clone, edit and delete Apstra nodes.



At the bottom left section of every page, you have continuous visibility of platform health. Green indicates the active state. Red indicates an issue, such as missing agent, the disk being in read only mode, or an agent rebooting (after the agent has rebooted, the status returns to active). If **IBA Services** or **Offbox Agents** is green, all containers are launched. If one of them is red, at least one container has failed. From any page, click one of the dots, then click a section for details. Clicking **Controller**, **IBA Services**, and **Offbox Agents** all take you to **Nodes** details.



Create Apstra Node

The controller node and worker nodes must use the same Apstra version (5.0.0, for example).

1. Install Apstra software on the VMs to cluster.
2. From the left navigation menu, navigate to **Platform > Apstra Cluster** and click **Add Node**.

3. Enter a name, tags (optional), address (IP or FQDN), and Apstra Server VM SSH username/password login credentials. (`iba` and `offbox` tags are added by default.)
4. Click **Create**. As the main Apstra controller connects to the new Apstra VM worker node, the state of the new Apstra VM changes from **INIT** to **ACTIVE**.

Edit Apstra Node

1. Either from the table view (Platform > Apstra Cluster) or the details view, click the **Edit** button for the VM to edit.
2. Make your changes. If you delete `iba` and/or `offbox` tags from the node, the IBA and/or offbox containers (as applicable) are moved to another node with those tags. Make sure the cluster has another node with those tags, or the containers will be deleted instead of moved.

> ⚠ **CAUTION**: To prevent containers from being deleted, don't delete tags unless another node in the cluster has the same tags.

3. Click **Update** to update the Apstra VM worker node.

Delete Apstra Node

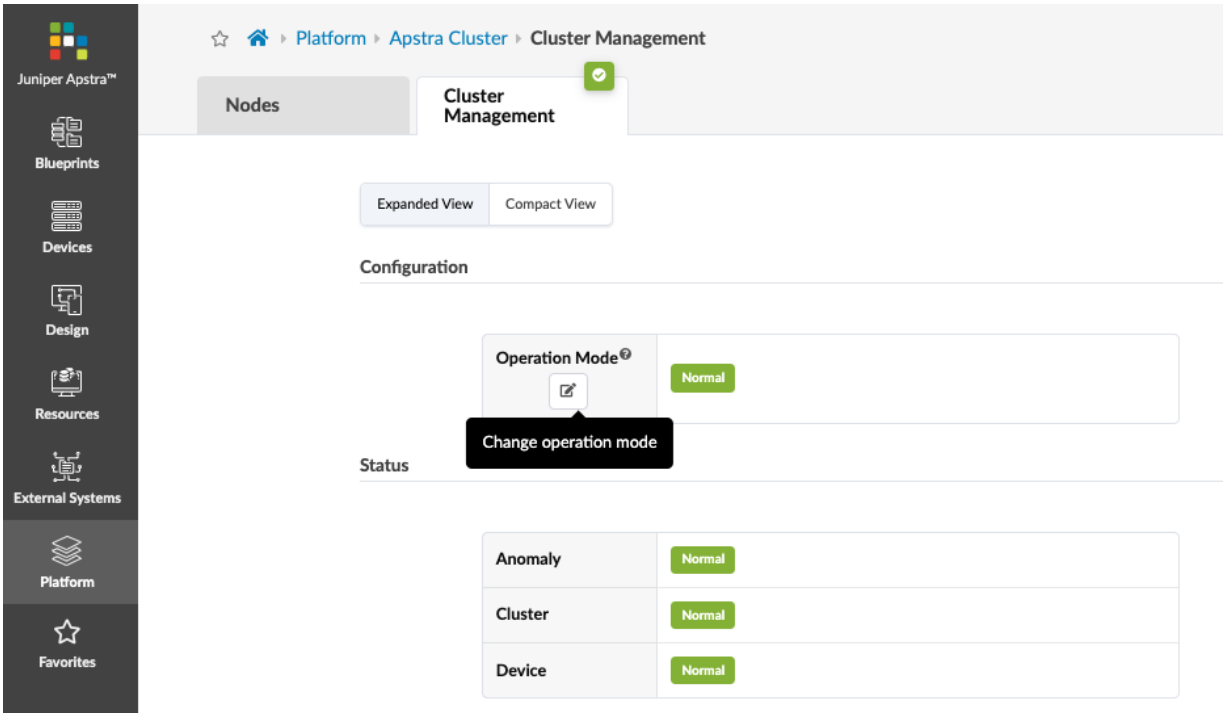
When you delete a node that includes `iba` and/or `offbox` tags, the IBA and/or offbox containers (as applicable) are moved to another node with those tags. Make sure the cluster has another node with those tags, or the containers will be deleted instead of moved.

> ⚠ **CAUTION**: To prevent containers from being deleted, don't delete nodes with `iba` and/or `offbox` tags unless another node in the cluster has the same tags.

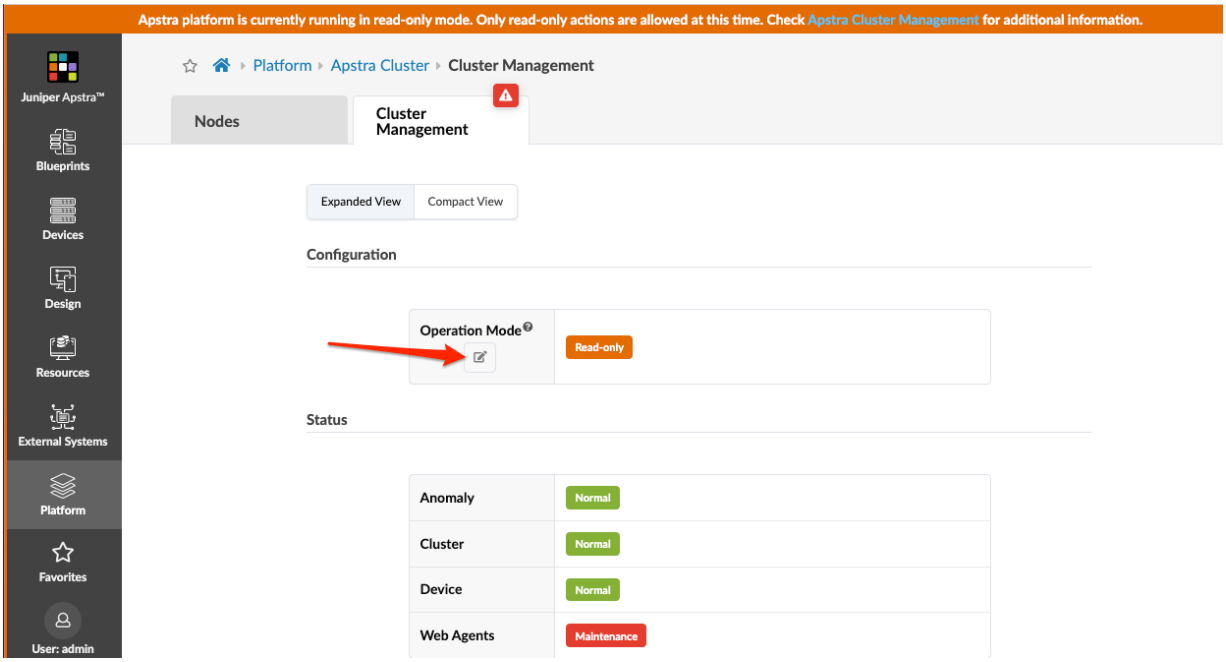
1. Either from the table view (Platform > Apstra Cluster) or the details view, click the **Delete** button for the Apstra VM to delete.
2. Click **Delete** to delete the Apstra VM.

Apstra Cluster Management

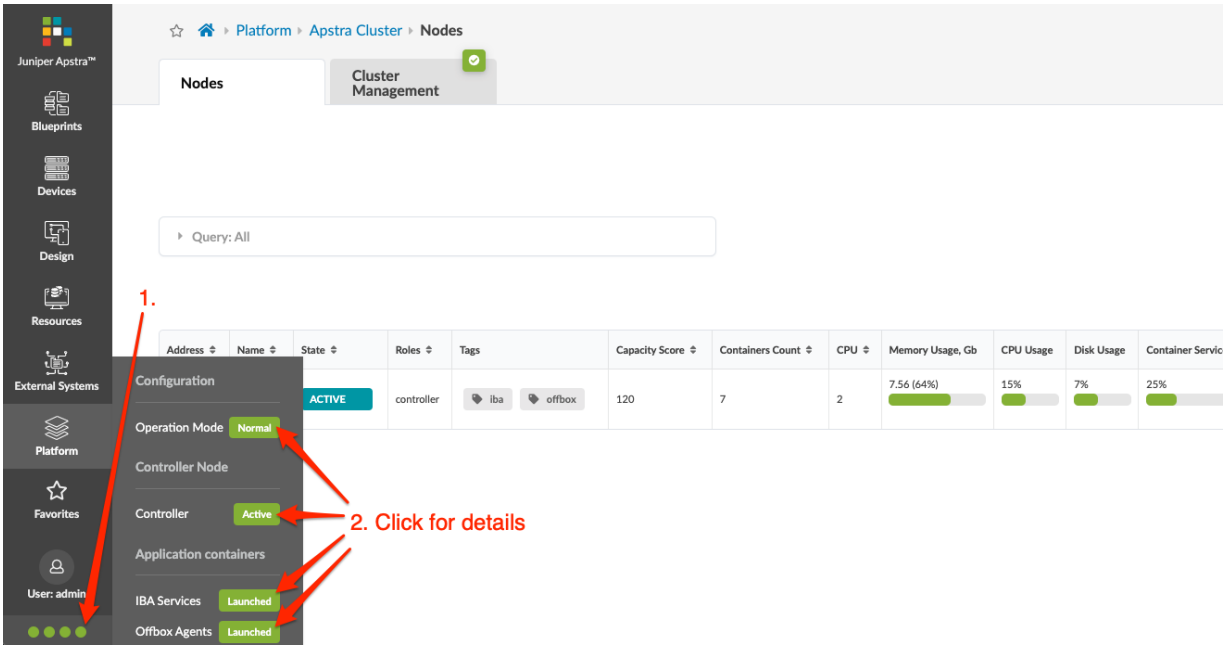
From the left navigation menu, navigate to **Platform > Apstra Cluster > Cluster Management** to go to Apstra cluster configuration and status.



Apstra admins may want to temporarily block all users (including themselves) from performing design and blueprint changes in the Apstra environment because they're troubleshooting something, or want to perform some maintenance operations on the Apstra server (backups, VM migration, VM OS updates and so on). Admins can change the **operation mode** from **Normal** to **Read-only** to block users from API and WebUI (PUT/POST). By default, only admins have permission to enable/disable the read-only mode.



At the bottom left section of every page, you have continuous visibility of platform health. Green indicates the active state. Red indicates some kind of issue, such as a missing agent, the disk being in read only mode, or an agent rebooting (after the agent has rebooted, the status returns to active). From any page, click one of the dots, then click the section that you want details for. Clicking **Operation Mode** takes you to cluster management details.



Change Cluster Application Memory Usage (API)

You can change cluster application memory usage for offbox agents and Intent Based Analytics (IBA) via API. If you're using Juniper offbox agents, increase memory allocation to 500 MB (from the 250 MB default). A single API call applies to all offbox agents.

1. From the left navigation menu in the Apstra GUI, navigate to **Platform > Developers** and click **REST API Documentation**.

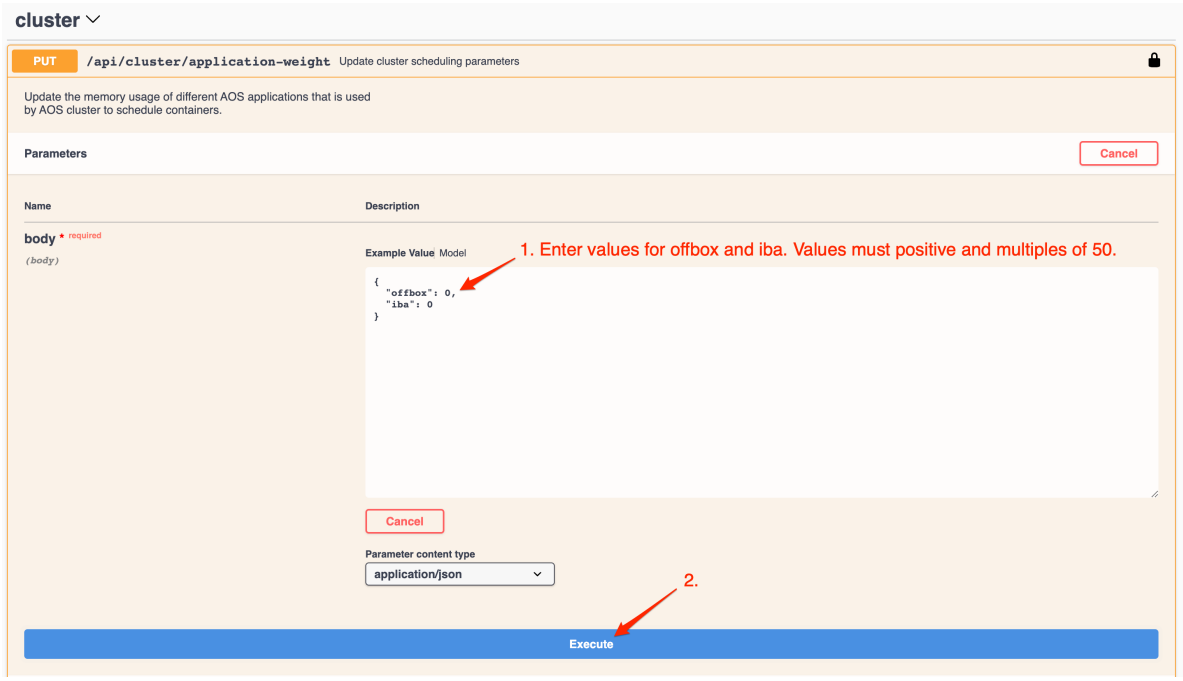
   The Swagger API developer tool for the Apstra environment appears.

2. Click **cluster**, click **GET /api/cluster/application-weight**, then click **Execute**.

   The currrent values for **offbox** and **iba** appear in the response body.

3. Click **PUT / api/cluster/application-weight**, then click **Try it out**.

   The parameters become editable.



4. Enter values for both **offbox** and **iba**, then click **Execute**. (The values must be positive and multiples of 50.) Juniper offbox agents require 500 MB.
5. To confirm your changes, click **cluster**, click **GET /api/cluster/application-weight**, then click **Execute**.
6. You can close the window at any time to leave the tool.

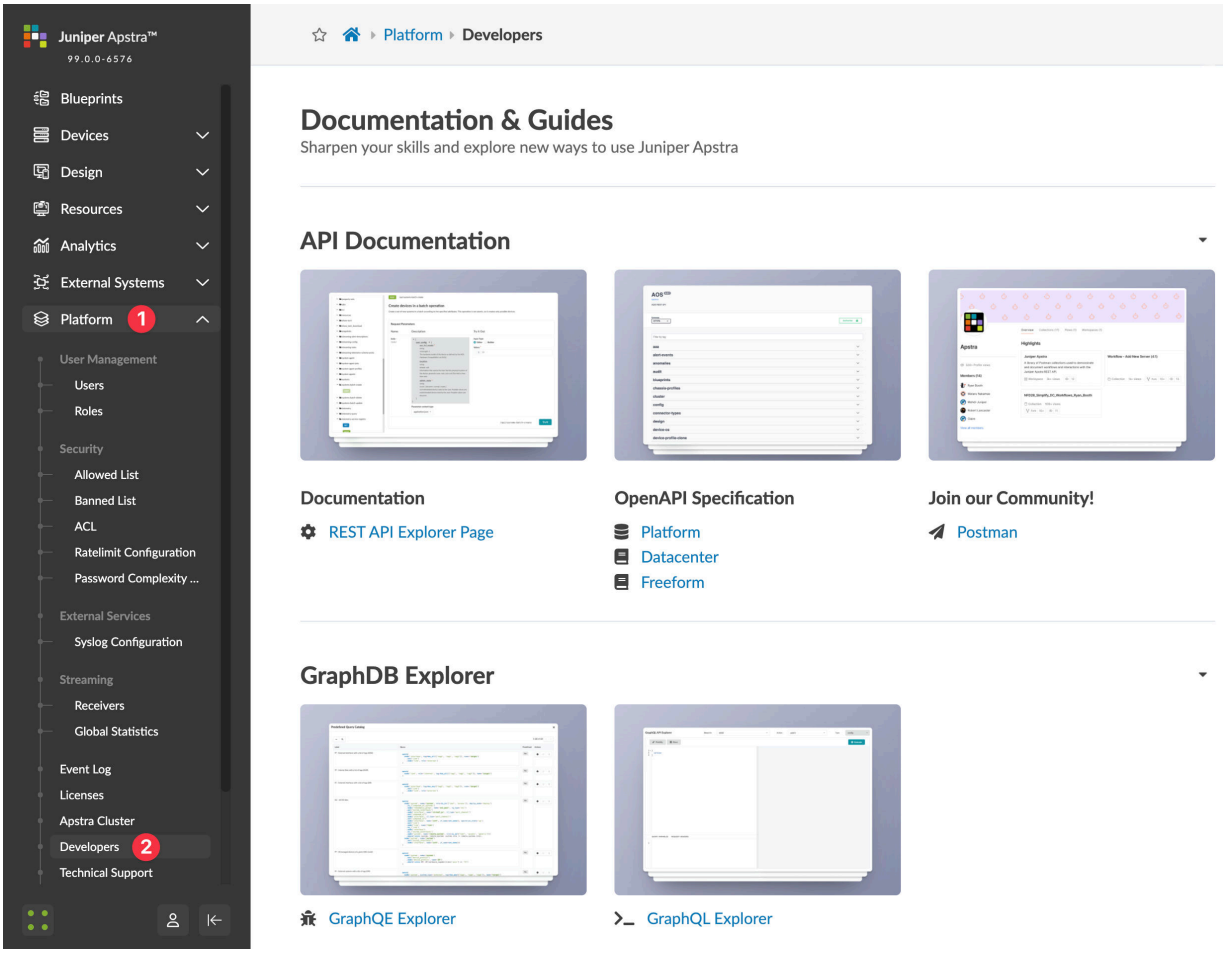
CHAPTER 50

Developers

IN THIS CHAPTER

Developers (Platform)

From the left navigation menu of the Apstra GUI, navigate to **Platform > Developers** to go to Developer Documentation and Guides.

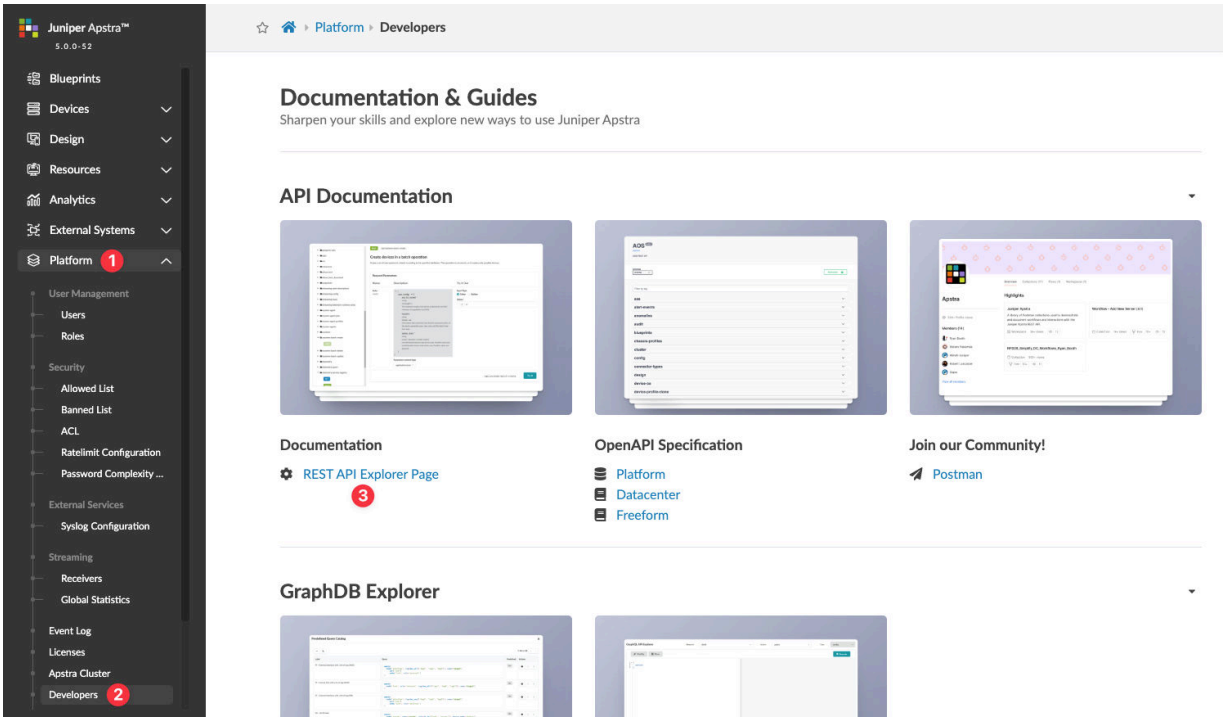


From here you can access API documentation, GraphDB Explorer, SDK, and other tools like Jinja function reference, custom telemetry collection expressions, and Webcons.

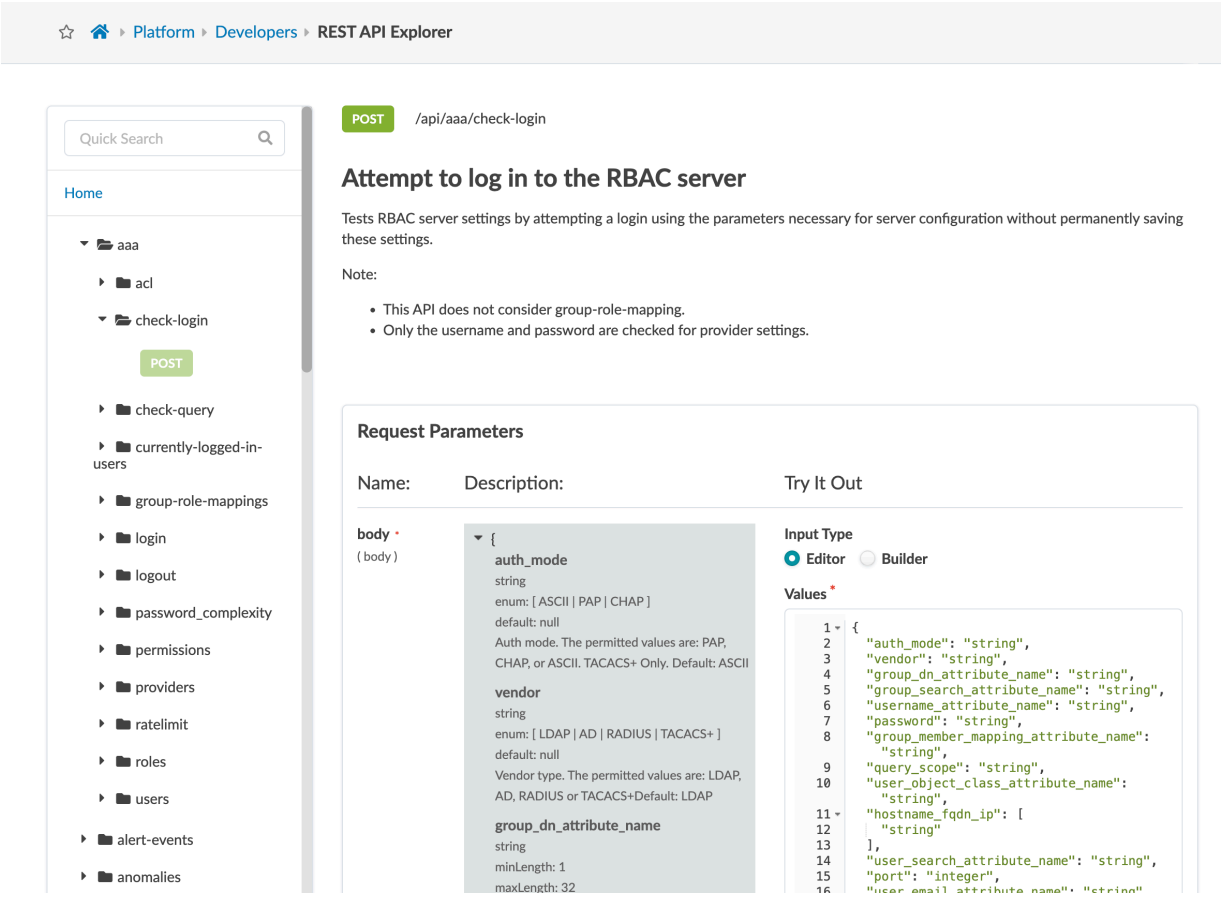
REST API Explorer

With Apstra's REST API explorer, you can browse and search for specific REST API endpoints.

From the left navigation menu, navigate to **Platform > Developers** and click **Rest API Explorer Page**



The left column contains a list of API categories from which you can browse. You can also search for a specific endpoint by entering a query in the **Quick Search** field. The details view of an endpoint includes information about the URL, method, summary, parameters and responses. The example below shows the model for checking provider settings by login with username and password.

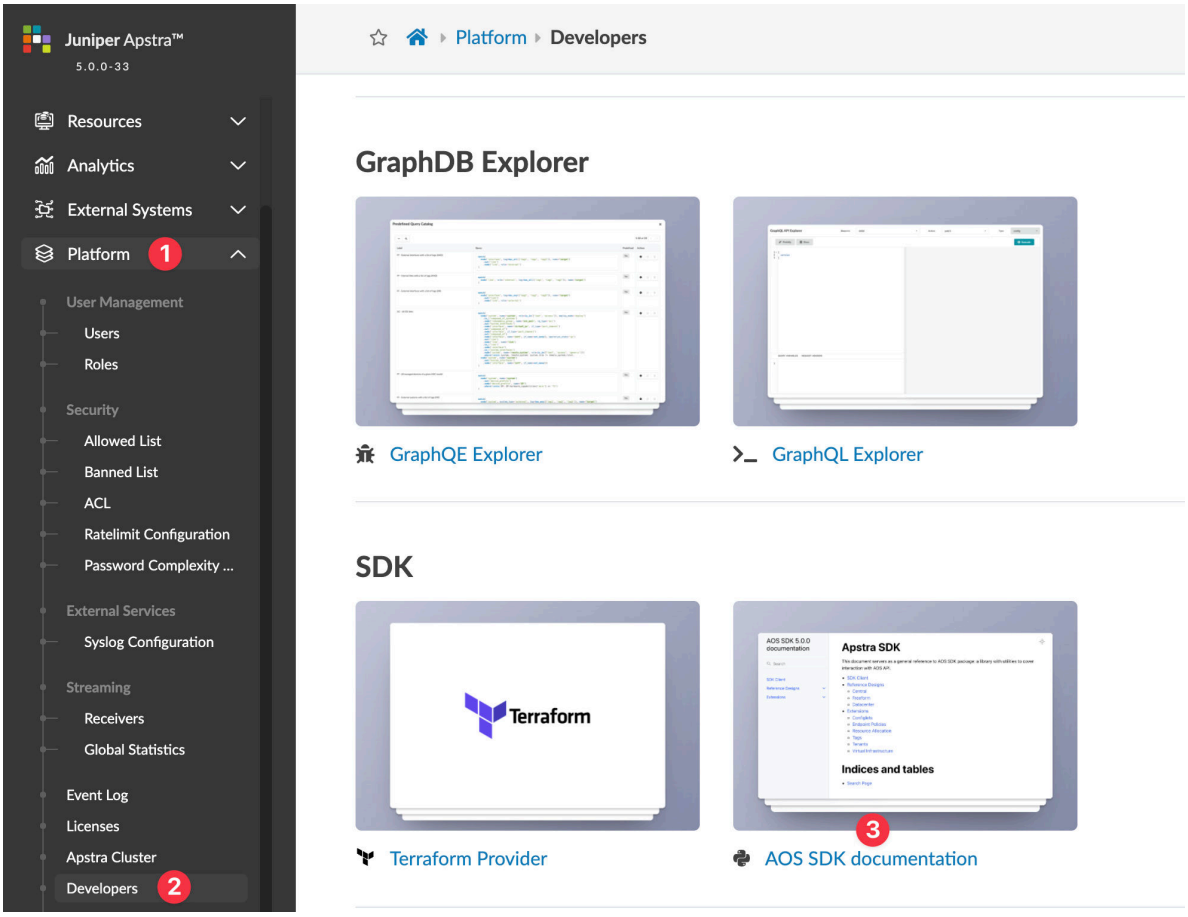


AOS SDK Documentation

SUMMARY

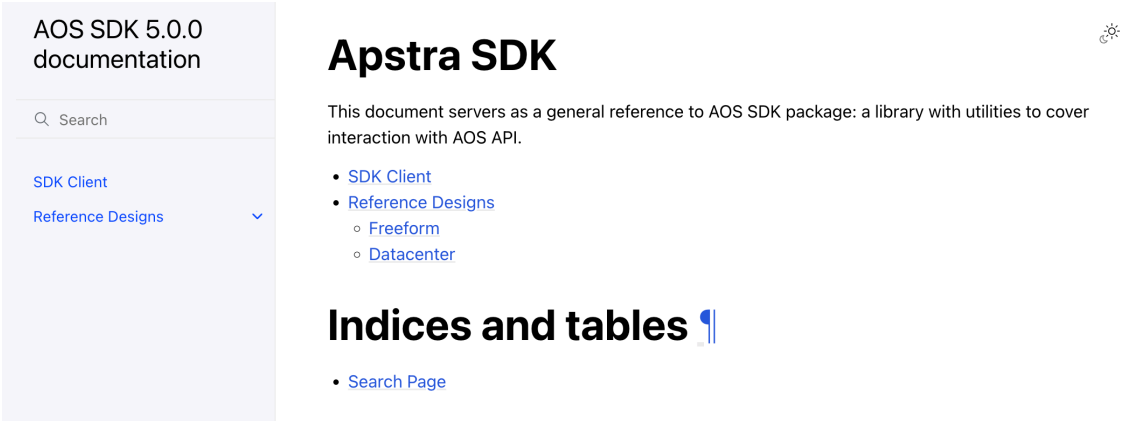
You can build additional automation code with Apstra's supported and tested Python SDK. The SDK provides a layer of abstraction on top of the APIs.

1. From the left navigation menu of the Apstra GUI, navigate to **Platform > Developers**, scroll to the SDK section and click **AOS SDK Documentation**.



The **AOS SDK 5.0.0 documentation** page opens in a new window.

2. From here you can access the SDK Client and Reference Design information. Detailed documentation and examples are included.



Resource Pools (API)

IN THIS SECTION

This reference demonstrates the resource group API usage with parity to the UI. For full API documentation, view the REST Platform API reference under the Apstra GUI.

To list resource group slots in a blueprint, perform an authenticated HTTP GET to https://aos-server/api/blueprints/<blueprint_id>/resource_groups

Both **ASN pools** and **IP pools** must be assigned in order for a blueprint to complete the build phase.

API - ASN Pools

Create ASN Pool

An example payload for creating an ASN Pool:

If an ID is not specified, one will be created and returned in the HTTP response.

```
{
    "id": "RFC6996-Private",
    "display_name": "RFC6996-Private",
    "tags": [ "default" ],
    "ranges": [
        {
            "last": 65534,
            "first": 64512
        }
    ]
}
```

To create an ASN pool perform an HTTP POST to https://aos-server/api/resources/asn-pools with a JSON payload.

```
curl 'https://192.168.25.250/api/resources/asn-pools?comment=create'
-H 'AuthToken: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6
ImFkbWluIiwiY3JlYXRlZF9hdCI6IjIwMTctMDUtMzFUMDA6MjI6MDcuNTIwMTgzWiIsIn
Nlc3Npb24iOiJjOTliOGVlOS05Y2NjLTRjZTAtYTY5NS0wODI3N2ZkYjA0ZDYifQ.FnJMR3
crPoD0-lQRXnpPOJ8TCsRG9Wr-DaddnAIj6ko' - --data-binary '{"display_name"
:"Example","ranges":[{"first":100,"last":200}],"tags":[]}' --compressed
--insecure
```

List ASN Pools

```
curl 'https://192.168.25.250/api/resources/asn-pools' -H 'AuthToken:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImFkbWluIiwiY3JlY
XRlZF9hdCI6IjIwMTctMDUtMzFUMDA6MjI6MDcuNTIwMTgzWiIsInNlc3Npb24iOiJjOTliO
GVlOS05Y2NjLTRjZTAtYTY5NS0wODI3N2ZkYjA0ZDYifQ.FnJMR3crPoD0-lQRXnpPOJ8TCsR
G9Wr-DaddnAIj6ko' --compressed --insecure
```


```
{
    "items": [
        {
            "created_at": "2017-05-30T12:56:07.293082Z",
            "display_name": "Private ASN",
            "id": "c23ea447-8f37-419a-9b1c-c48cc55d5b9c",
            "last_modified_at": "2017-05-30T12:56:07.293082Z",
            "ranges": [
                {
                    "first": 65412,
                    "last": 65534,
                    "status": "pool_element_in_use"
                }
            ],
            "status": "in_use",
            "tags": []
        }
    ]
}
```

Delete ASN Pool

To delete an ASN Pool perform an HTTP DELETE to https://aos-server/resources/asn-pools/{pool_id}

A successful DELETE returns HTTP 200 OK.

```
curl
'https://192.168.25.250/api/resources/asn-pools/d0312b4a-017e-4478-8b8d-df0417ce8d3b'
-X DELETE -H 'AuthToken: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2Vybm
FtZSI6ImFkbWluIiwiY3JlYXRlZF9hdCI6IjIwMTctMDUtMzFUMDA6MjI6MDcuNTIwMTgzW
iIsInNlc3Npb24iOiJjOTliOGVlOS05Y2NjLTRjZTAtYTY5NS0wODI3N2ZkYjA0ZDYifQ.FnJ
MR3crPoD0-lQRXnpPOJ8TCsRG9Wr-DaddnAIj6ko' --compressed --insecure
```

Assign ASN to Blueprint

To assign an IP pool to the blueprint perform an HTTP PUT to https://aos-server/blueprints/<blueprint_id>/resource_groups/ip/<pool_name>

For instance, to post a resource pool to **spine_loopback_ips**, first obtain the ID of the resource pool, and append it to a list for slot assignation. When updating the IP Pool resource group, specify all pools in the payload at the same time. We cannot add single pools, so PUT them all at once.

Payload:

```
{"pool_ids": ["pool_id1", "pool_id2", "pool_id3"] }
```

```
curl
'https://192.168.25.250/api/blueprints/4c1e69c6-97bd-4c99-9504-7818f138b17f/resource_groups/asn/
spine_asns'
-X PUT -H 'AuthToken: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2Vyb
mFtZSI6ImFkbWluIiwiY3JlYXRlZF9hdCI6IjIwMTctMDUtMzFUMDA6MjI6MDcuNTI
wMTgzWiIsInNlc3Npb24iOiJjOTliOGVlOS05Y2NjLTRjZTAtYTY5NS0wODI3N2ZkYj
A0ZDYifQ.FnJMR3crPoD0-lQRXnpPOJ8TCsRG9Wr-DaddnAIj6ko'  --data-binary
'{"pool_ids":["c23ea447-8f37-419a-9b1c-c48cc55d5b9c"]}' --compressed --insecure
```

A successful ASSIGNMENT returns HTTP 200 OK.

Unassign ASN from Blueprint

When removing IP pools from a blueprint, PUT an empty pool_id list to the blueprint with the payload `[]`:

PUT to the HTTP endpoint https://aos-server/api/blueprints/<blueprint_id> /resource_groups/asn/<pool_name>

With the payload:

```
{ "pool_ids": [] }
```

```
curl
'https://192.168.25.250/api/blueprints/4c1e69c6-97bd-4c99-9504-7818f138b17f/resource_groups/asn/
spine_asns'
-X PUT -H 'AuthToken: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFt
ZSI6ImFkbWluIiwiY3JlYXRlZF9hdCI6IjIwMTctMDUtMzFUMDA6MjI6MDcuNTIwMTgzWi
IsInNlc3Npb24iOiJjOTliOGVlOS05Y2NjLTRjZTAtYTY5NS0wODI3N2ZkYjA0ZDYifQ.FnJ
MR3crPoD0-lQRXnpPOJ8TCsRG9Wr-DaddnAIj6ko'  --data-binary '{"pool_ids":[]}'
--compressed --insecure
```

If the request is successful there will be no response.

List ASN assigned to Blueprint

Available ASN Pool resource groups for assignment can be shown with an HTTP GET to https://aos-server/api/blueprints/<blueprint_id>/resource_groups

```
curl
'https://192.168.25.250/api/blueprints/4c1e69c6-97bd-4c99-9504-7818f138b17f/resource_groups'
-H 'AuthToken: eyJhbGciOiJIUzI1NwMTctMDUtMzFUMDA6MjI6MDcuNTIwMTgz
WiIsInNlc3Npb24iOiJjOTliOGVlOS05Y2NjLTRjZTAtYTY5NS0wODI3N2ZkYjA0ZD
YifQ.FnJMR3crPoD0-lQRXnpPOJ8TCsRG9Wr-DaddnAIj6ko' --compressed --insecure
| python -m json.tool
```

```
{
    "items": [
        {
            "name": "leaf_asns",
```

```
            "pool_ids": [
                "c23ea447-8f37-419a-9b1c-c48cc55d5b9c"
            ],
            "type": "asn"
        },
        {
            "name": "spine_asns",
            "pool_ids": [
                "c23ea447-8f37-419a-9b1c-c48cc55d5b9c"
            ],
            "type": "asn"
        },
        {
            "name": "leaf_loopback_ips",
            "pool_ids": [
                "56e8e0dc-babd-4652-92a5-fc37294a7b26"
            ],
            "type": "ip"
        },
        {
            "name": "mlag_domain_svi_subnets",
            "pool_ids": [
                "ed7d8830-c703-4ac0-8252-77e0f272a677"
            ],
            "type": "ip"
        },
        {
            "name": "spine_leaf_link_ips",
            "pool_ids": [
                "ed7d8830-c703-4ac0-8252-77e0f272a677"
            ],
            "type": "ip"
        },
        {
            "name": "spine_loopback_ips",
            "pool_ids": [
                "56e8e0dc-babd-4652-92a5-fc37294a7b26"
            ],
            "type": "ip"
        }
    ]
}
```

API - IP Pools

Create IP Pool

JSON Payload for creating an IP Pool:

```
{
    "id": "example_ip_pool",
    "display_name": "example_ip_pool",
    "tags": ["default"],
    "subnets": [
        {"network": "10.0.0.0/8"}
    ]
}
```

The **subnets** section requires a list of dictionaries with keyword **network** and value matching a CIDR mask. The subnets cannot overlap with each other in the same pool. That is to say, 192.168.10.0/24 and 192.168.0.0/16 cannot be configured in the same pool.

Tags are optional and are not currently used in Apstra. If ID is specified, it will be saved, otherwise an ID will be returned in the HTTP Response after creating the pool.

An HTTP POST to https://aos-server/api/resources/ip-pools with JSON payload will reply with the ID of the new IP pool.

```
curl 'https://192.168.25.250/api/resources/ip-pools' -X
POST -H 'AuthToken: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmF
tZSI6ImFkbWluIiwiY3JlYXRlZF9hdCI6IjIwMTctMDUtMzFUMDA6MjI6MDcuNTIwMTgzWi
IsInNlc3Npb24iOiJjOTliOGVlOS05Y2NjLTRjZTAtYTY5NS0wODI3N2ZkYjA0ZDYifQ.Fn
JMR3crPoD0-lQRXnpPOJ8TCsRG9Wr-DaddnAIj6ko' --data-binary '{"display_name":
"example_ip_pool","subnets":[{"network":"10.0.0.0/8"},{"network":
"192.168.0.0/16"}],"tags":[]}' --compressed --insecure
```

```
{"id": "d0312b4a-017e-4478-8b8d-df0417ce8d3b"}
```

List IP Pools

Perform an HTTP GET to https://aos-server/api/resources/ip-pools -

```
jp@ApstraVM ~ $ curl 'https://192.168.25.250/api/resources/ip-pools' -H
'AuthToken: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImFkbW
luIiwiY3JlYXRlZF9hdCI6IjIwMTctMDUtMzFUMDA6MjI6MDcuNTIwMTgzWiIsInNlc3Npb24
iOiJjOTliOGVlOS05Y2NjLTRjZTAtYTY5NS0wODI3N2ZkYjA0ZDYifQ.FnJMR3crPoD0-lQRXnpP
OJ8TCsRG9Wr-DaddnAIj6ko' --compressed --insecure | python -m json.tool
```


```
{
    "items": [
        {
            "created_at": "2017-05-31T03:48:38.562331Z",
            "display_name": "example_ip_pool",
            "id": "d5046aa6-eab2-4990-9816-0a519ce1a8db",
            "last_modified_at": "2017-05-31T03:48:38.562331Z",
            "status": "not_in_use",
            "subnets": [
                {
                    "network": "10.0.0.0/8",
                    "status": "pool_element_available"
                },
                {
                    "network": "192.168.0.0/16",
                    "status": "pool_element_available"
                }
            ],
            "tags": []
        },
        {
            "created_at": "2017-05-30T12:56:50.576598Z",
            "display_name": "L3-CLOS",
            "id": "ed7d8830-c703-4ac0-8252-77e0f272a677",
            "last_modified_at": "2017-05-30T12:56:50.576598Z",
            "status": "in_use",
            "subnets": [
                {
                    "network": "10.16.0.0/16",
                    "status": "pool_element_in_use"
                }
```

```
            ],
            "tags": []
        },
        {
            "created_at": "2017-05-30T12:56:24.222906Z",
            "display_name": "Loopbacks",
            "id": "56e8e0dc-babd-4652-92a5-fc37294a7b26",
            "last_modified_at": "2017-05-30T12:56:24.222906Z",
            "status": "in_use",
            "subnets": [
                {
                    "network": "10.254.0.0/16",
                    "status": "pool_element_in_use"
                }
            ],
            "tags": []
        },
        {
            "created_at": "2017-05-31T03:49:15.485164Z",
            "display_name": "example_ip_pool",
            "id": "d0312b4a-017e-4478-8b8d-df0417ce8d3b",
            "last_modified_at": "2017-05-31T03:49:15.485164Z",
            "status": "not_in_use",
            "subnets": [
                {
                    "network": "10.0.0.0/8",
                    "status": "pool_element_available"
                },
                {
                    "network": "192.168.0.0/16",
                    "status": "pool_element_available"
                }
            ],
            "tags": []
        }
    ]
}
```


Delete IP pool

To delete an IP Pool perform an HTTP DELETE to https://aos-server/resources/ip-pools/{pool_id}

A successful DELETE returns HTTP 200 OK and an empty JSON response {}

```
curl
'https://192.168.25.250/api/resources/ip-pools/d0312b4a-017e-4478-8b8d-df0417ce8d3b'
-X DELETE -H 'AuthToken: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZS
I6ImFkbWluIiwiY3JlYXRlZF9hdCI6IjIwMTctMDUtMzFUMDA6MjI6MDcuNTIwMTgzWiIsInNl
c3Npb24iOiJjOTliOGVlOS05Y2NjLTRjZTAtYTY5NS0wODI3N2ZkYjA0ZDYifQ.FnJMR3crPoD0
-lQRXnpPOJ8TCsRG9Wr-DaddnAIj6ko' --compressed --insecure
```

Assign IP to Blueprint

To assign an IP pool to the blueprint perform an HTTP PUT to https://aos-server/blueprints/<blueprint_id>/resource_groups/ip/<group_name>

For instance, to associate a resource pool **spine_loopback_ips** with a blueprint first obtain the ID of the resource pool, and append it to a list for slot assignation. When updating the IP Pool resource group, specify all pools in the payload at the same time. We cannot add single pools, so PUT them all at once. Instruct Apstra to associate IP pool with ID 'ed7d8830-c703-4ac0-8252-77e0f272a677'to the blueprint. You may have to GET existing pool IDs prior to adding a new one to avoid deleting existing pools.

Payload:

```
{"pool_ids": ["pool_id1", "pool_id2", "pool_id3"] }
```

```
curl
'https://192.168.25.250/api/blueprints/4c1e69c6-97bd-4c99-9504-7818f138b17f/resource_groups/ip/
spine_loopback_ips'
-X PUT -H 'AuthToken: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImF
kbWluIiwiY3JlYXRlZF9hdCI6IjIwMTctMDUtMzFUMDA6MjI6MDcuNTIwMTgzWiIsInNlc3Npb2
4iOiJjOTliOGVlOS05Y2NjLTRjZTAtYTY5NS0wODI3N2ZkYjA0ZDYifQ.FnJMR3crPoD0-lQRXnp
POJ8TCsRG9Wr-DaddnAIj6ko'  --data-binary '{"pool_ids":["ed7d8830-c703-4ac0-825
2-77e0f272a677"]}' --compressed --insecure
```

A successful ASSIGNMENT returns an HTTP 200 OK.

Remove IP from Blueprint

To remove IP pools from the blueprint PUT an empty `pool_id` list to the blueprint with the payload `[]`:

PUT to the HTTP endpoint https://aos-server/api/blueprints/<blueprint_id>/ resource_groups/ip/ <allocation_group_name>

With the payload:

```
{ "pool_ids": [] }
```

CURL Example

```
curl
'https://192.168.25.250/api/blueprints/4c1e69c6-97bd-4c99-9504-7818f138b17f/resource_groups/ip/
spine_loopback_ips'
-X PUT -H 'AuthToken: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZ
SI6ImFkbWluIiwiY3JlYXRlZF9hdCI6IjIwMTctMDUtMzFUMDA6MjI6MDcuNTIwMTgzWiIsI
nNlc3Npb24iOiJjOTliOGVlOS05Y2NjLTRjZTAtYTY5NS0wODI3N2ZkYjA0ZDYifQ.FnJMR3cr
PoD0-lQRXnpPOJ8TCsRG9Wr-DaddnAIj6ko'  --data-binary '{"pool_ids":[]}'
--compressed --insecure
```

A successful REMOVAL returns an empty response: {}

List IPs Assigned to Blueprint

```
curl
'https://192.168.25.250/api/blueprints/4c1e69c6-97bd-4c99-9504-7818f138b17f/resource_groups'
-H 'AuthToken: eyJhbGciOiJIUzI1NwMTctMDUtMzFUMDA6MjI6MDcuNTIwMTgzWiIsInNlc3
Npb24iOiJjOTliOGVlOS05Y2NjLTRjZTAtYTY5NS0wODI3N2ZkYjA0ZDYifQ.FnJMR3crPoD
0-lQRXnpPOJ8TCsRG9Wr-DaddnAIj6ko' --compressed --insecure | python -m json.tool
```

```
{
    "items": [
        {
            "name": "leaf_asns",
            "pool_ids": [
                "c23ea447-8f37-419a-9b1c-c48cc55d5b9c"
            ],
            "type": "asn"
        },
        {
            "name": "spine_asns",
```

```
            "pool_ids": [
                "c23ea447-8f37-419a-9b1c-c48cc55d5b9c"
            ],
            "type": "asn"
        },
        {
            "name": "leaf_loopback_ips",
            "pool_ids": [
                "56e8e0dc-babd-4652-92a5-fc37294a7b26"
            ],
            "type": "ip"
        },
        {
            "name": "mlag_domain_svi_subnets",
            "pool_ids": [
                "ed7d8830-c703-4ac0-8252-77e0f272a677"
            ],
            "type": "ip"
        },
        {
            "name": "spine_leaf_link_ips",
            "pool_ids": [
                "ed7d8830-c703-4ac0-8252-77e0f272a677"
            ],
            "type": "ip"
        },
        {
            "name": "spine_loopback_ips",
            "pool_ids": [
                "56e8e0dc-babd-4652-92a5-fc37294a7b26"
            ],
            "type": "ip"
        }
    ]
}
```

Configlets (API)

IN THIS SECTION

For full API documentation, view the Platform API reference from the web interface. This is a targeted section to demonstrate configlet API similarly to the UI. The main difference between the Web UI and REST API is that the Apstra API does not make any use of the configlets stored under api/design/configlets when working with a blueprint. Design-configlets are meant for consumption under the UI. When working with configlets on the API, work directly with the blueprint.

Configlets live in http://aos-server/api/design/configlets and are referenced by ID.

```
{
  "ref_archs": [
    "two_stage_l3clos"
  ],
  "created_at": "string",
  "last_modified_at": "string",
  "id": "string",
  "generators": [
    {
      "config_style": "string",
      "template_text": "string",
      "negation_template_text": "string"
    }
  ],
  "display_name": "string",
  "section": "string"
}
```

API - Create Configlet

To create a configlet, POST to https://aos-server/api/design/configlets with a valid JSON structure representing the configlet. You can assign this configlet from the Apstra GUI. This method is not required for the REST API to assign to a blueprint. See the assigning a configlet section for more details.

A POST will create a new configlet. A PUT will overwrite an existing configlet. PUT requires the URL of the configlet. https://aos-server/api/design/configlets/{id}

```
curl -H "AuthToken: EXAMPLE" -d '{"display_name":"DNS","ref_archs":
["two_stage_l3clos"],"section":"system","generators":[{"config_style":"eos","template_text":"ip
name-server 192.168.1.1","negation_template_text":"no ip name-server 192.168.1.1"}]}' -X POST
"http://aos-server/api/design/configlets"
```

The response will contain the ID of the newly created configlet `{"id": "995446c7-de7d-46bb-a88a-786839556064"}`

API - Delete Configlet

Deleting a configlet requires an HTTP DELETE to the configlet by URL http://aos-server/api/design/configlets/{id}

```
curl -H "AuthToken: EXAMPLE" -X DELETE "http://aos-server/api/design/configlets/995446c7-
de7d-46bb-a88a-786839556064"
```

A successful DELETE has an empty response {}

API - Assign Configlet

Assigning a configlet to a blueprint requires assignation of device conditions as well as embedding the configlet details. When assigning a configlet to a blueprint, the configlets available as design resources aren't necessary. These are only used for UI purposes.

The assigned configlet lives in https://aos-server/api/blueprints/blueprint_id/configlets

JSON Syntax for putting a configlet to a blueprint. Basically, this is just an 'items' dictionary element containing a list of configlet schemas.

```
{
  "items": [
    {
```

```
      "template_params": [
        "string"
      ],
      "configlet": {
        "generators": [
          {
            "config_style": "string",
            "template_text": "string",
            "negation_template_text": "string"
          }
        ],
        "section": "string",
        "display_name": "string"
      },
      "condition": "string"
    }
  ]
}
```


CURL Example - HTTP PUT


```
curl "http://aos-server/api/blueprints/e4068e99-813c-4290-b7cc-e145d85a98a8/configlets" -X PUT -
H "AuthToken: EXAMPLE" -H "Content-Type: application/json; charset=utf-8" --data
"[{""configlet"":{""generators"":[{""config_style"":""eos"",""template_text"":""ip name-server
192.168.1.1"",""negation_template_text"":""no ip name-server
192.168.1.1""}],""section"":""system"",""display_name"":""DNS""},""condition"":""role==spine""},
{""configlet"":{""generators"":[{""config_style"":""eos"",""template_text"":""ip name-server
192.168.1.1"",""negation_template_text"":""no ip name-server
192.168.1.1""}],""section"":""system"",""display_name"":""DNS""},""condition"":""role==leaf""}]"
```


Response


```
{"items": [{"configlet": {"generators": [{"config_style": "eos", "template_text": "ip name-
server 192.168.1.1", "negation_template_text": "no ip name-server 192.168.1.1"}], "section":
"system", "display_name": "DNS"}, "condition": "role==spine"}, {"configlet": {"generators":
[{"config_style": "eos", "template_text": "ip name-server 192.168.1.1",
"negation_template_text": "no ip name-server 192.168.1.1"}], "section": "system",
"display_name": "DNS"}, "condition": "role==leaf"}]}
```

API - Unassign Configlet

To unassign a configlet, remove it from the items list by PUT with an empty json post.

```
curl "http://aos-server/api/blueprints/e4068e99-813c-4290-b7cc-e145d85a98a8/configlets" -X PUT -
H "AuthToken: EXAMPLE" -H "Content-Type: application/json; charset=utf-8" --data ""
```

The response is an empty json set once the configlet is deleted: {"items": []}

Property Sets (API)

IN THIS SECTION

For full API documentation, view the Platform API reference from the web interface. This is a targeted section to demonstrate property sets API similarly to the web interface.

Property sets live in http://aos-server:8888/api/property-sets and are referenced by ID.

```
{
  "items": [
    {
      "label": "string",
      "values": {
        "additionalProp1": "string",
        "additionalProp2": "string",
        "additionalProp3": "string"
      },
      "id": "string"
    }
```

```
      ]
    }
```

API - Create Property Set

To create a property set, POST to https://aos-server/api/property-sets with a valid JSON structure representing the property set. Creating a property set this way only allows it to be available for assignation in the web interface - it is not required in this method for the REST API to assign to a blueprint. See the assigning a property set section for more details.

A POST will create a new property set. A PUT will overwrite an existing property set. PUT requires the URL of the property set. https://aos-server:8888/api/design/property-sets/{id}

```
curl -H "AuthToken: EXAMPLE" -d '{"values": {"NTP_SRV1": "192.168.1.1", "NTP_SRV1":
"192.168.1.1"}, "label": "NTP-servers"}' -X POST "http://aos-server:8888/api/design/property-
sets"
```

The response will contain the ID of the newly created property-set `{"id": "73223e81-a451-4e7f-91fb-fb476f4b9fc8"}`

API - Delete Property Set

Deleting a property set requires an HTTP DELETE to the property set by URL http://aos-server:8888/api/design/property-sets/{id}

```
curl -H "AuthToken: EXAMPLE" -X DELETE "http://aos-server:8888/api/design/property-sets/73223e81-
a451-4e7f-91fb-fb476f4b9fc8"
```

A successful DELETE has an empty response {}

API - Assign Property Set

Assigning a property set to a blueprint requires an HTTP POST to the blueprint by URL http://aos-server:8888/api/blueprints/{blueprint_ID}/property-sets

```
{
     "id": "73223e81-a451-4e7f-91fb-fb476f4b9fc8"
}
```

The response will contain the ID of the assigned property-sets `{"id": "73223e81-a451-4e7f-91fb-fb476f4b9fc8"}`

CURL Example - API HTTP PUT

```
curl "http://aos-server:8888/api/blueprints/e4068e99-813c-4290-b7cc-e145d85a98a8/property-sets/
73223e81-a451-4e7f-91fb-fb476f4b9fc8" -X DELETE -H "AuthToken: EXAMPLE"
```

Response

```
{"id": "73223e81-a451-4e7f-91fb-fb476f4b9fc8"}
```

API - Unassign Property Set

Deleting a property set requires an HTTP DELETE to the blueprint property set by URL http://aos-server:8888/api/blueprints/{blueprint_ID}/property-sets{id}

```
curl "http://aos-server:8888/api/blueprints/e4068e99-813c-4290-b7cc-e145d85a98a8/property-sets/
73223e81-a451-4e7f-91fb-fb476f4b9fc8" -X DELETE -H "AuthToken: EXAMPLE"
```

A successful DELETE has an empty response {}

Interface Descriptions (API)

IN THIS SECTION

Besides main parameters of network interfaces like name, speed and port mode, Apstra also configures a description for physical interfaces and aggregated logical interfaces (so called port channels). Interface description is automatically generated if the following conditions are met:

1. The interface is connected to a peer.

2. The interface belongs to leaf, spine or generic system.
3. The peer interface belongs to leaf, spine, or generic system with virtual network endpoint on this server.

The generated description has the form `<facing_|to.><peer-device-label>[:peer-interface-name]`. Examples:

- facing_spine2:Ethernet1/2
- to.server1:eth0
- to.server2

The prefix of the name is `facing_` if the peer is leaf, spine or external router. The prefix is `to.` in case peer device is an L2 or L3 server. The peer interface name part is present only when the peer device is controlled by Apstra.

Apstra REST API - Interface descriptions

The Apstra API is able to change the auto-generated interface description. However, there is no such functionality in the Apstra UI.

The interface description may contain ASCII characters with codes 33-126 and spaces, except "?", which is interpreted as a command-completion. The description length is limited to 240 characters, which is the longest possible length across supported switch models.

Interfaces are stored internally as graph nodes with certain set of properties. Description is one of these properties. To modify the description, use the generic API to interact with graph nodes.

API - Obtain interface configuration

To obtain interface configuration, send GET request to https://aos-server/api/blueprints/{blueprint-id}/nodes/{interface-node-id}.

Request:

```
{
    "description": "facing_dkl-2-leaf:Ethernet1/2",
    "mlag_id": null,
    "tags": null,
    "if_name": "swp2",
    "label": null,
    "port_channel_id": null,
    "ipv4_addr": "203.0.113.10/31",
    "mode": null,
```

```
    "if_type": "ip",
    "type": "interface",
    "id": "interface-id-1",
    "protocols": "ebgp"
}
```

API - Create or modify interface description

To create or modify interface description, send PATCH request to https://aos-server/api/blueprints/{blueprint-id}/nodes/{interface-node-id} with a valid JSON. The JSON should contain the "description" field with a valid data.

```
curl -X PATCH -H "AuthToken: EXAMPLE" \
     -d '{"description": "New description I want!"}'
     http://aos-server:8888/api/blueprints/id-1/nodes/interface-id-1
```

Response:

```
{
    "description": "New description I want!",
    "mlag_id": null,
    "tags": null,
    "if_name": null,
    "label": null,
    "port_channel_id": null,
    "ipv4_addr": null,
    "mode": null,
    "if_type": "ip",
    "type": "interface",
    "id": "interface-id-1",
    "protocols": "ebgp"
}
```

API - Delete interface description

To delete custom interface description and get back to automatic description generation, set the description to empty value.

Request:

```
curl -X PATCH -H "AuthToken: EXAMPLE" \
     -d '{"description": ""}'
     http://aos-server:8888/api/blueprints/id-1/nodes/interface-id-1
```

Response:

```
{
    "description": "",
    "mlag_id": null,
    "tags": null,
    "if_name": null,
    "label": null,
    "port_channel_id": null,
    "ipv4_addr": null,
    "mode": null,
    "if_type": "ip",
    "type": "interface",
    "id": "interface-id-1",
    "protocols": "ebgp"
}
```

Subsequent GET request will show that the description was automatically generated.

Request:

```
curl -H "AuthToken: EXAMPLE" \
     http://aos-server:8888/api/blueprints/id-1/nodes/interface-id-1
```

Response:

```
{
    "description": "facing_dkl-2-leaf:Ethernet1/2",
    "mlag_id": null,
    "tags": null,
    "if_name": "swp2",
    "label": null,
    "port_channel_id": null,
    "ipv4_addr": "203.0.113.10/31",
```

```
    "mode": null,
    "if_type": "ip",
    "type": "interface",
    "id": "interface-id-1",
    "protocols": "ebgp"
}
```

Probes (API)

IN THIS SECTION

Generic Probe REST API

The information below describes as much of the API as necessary to understand how to use IBA for someone already familiar with Apstra API conventions. Formal API documenation is reserved for the API documentation itself.

We will walk through the API as it's used for the example workflow described in the introduction, demonstrating its general capability by specific example.

Create Probe

To create a probe, the operator POSTs to `/api/blueprints/<blueprint_id>/probes` with the following form:

```
{
    "label": "server_tx_bytes",
    "description": "Server traffic imbalance",
```

```
    "tags": ["server", "imbalance"],
    "disabled": false,
    "processors": [
        {
            "name": "server_tx_bytes",
            "outputs": {
                "out": "server_tx_bytes_output"
            },
            "properties": {
                "counter_type": "tx_bytes",
                "graph_query": "node('system',
name='sys').out('hosted_interfaces').node('interface', name='intf').out('link').node('link',
link_type='ethernet', speed=not_none()).in_('link').node('interface',
name='dst_intf').in_('hosted_interfaces').node('system', name='dst_node',
role='server').ensure_different('intf', 'dst_intf')",
                "interface": "intf.if_name",
                "system_id": "sys.system_id"
            },
            "type": "if_counter"
        },
        {
            "inputs": {
                "in": "server_tx_bytes_output"
            },
            "name": "std",
            "outputs": {
                "out": "std_dev_output"
            },
            "properties": {
                "ddof": 0,
                "group_by": []
            },
            "type": "std_dev"
        },
        {
            "inputs": {
                "in": "std_dev_output"
            },
            "name": "server_imbalance",
            "outputs": {
                "out": "std_dev_output_in_range"
            },
            "properties": {
```

```
                "range": {
                    "max": 100
                }
            },
            "type": "range_check"
        },
        {
            "inputs": {
                "in": "std_dev_output_in_range"
            },
            "name": "server_imbalance_anomaly",
            "outputs": {
                "out": "server_traffic_imbalanced"
            },
            "type": "anomaly"
        }
    ],
    "stages": [
        {
            "name": "server_tx_bytes_output",
            "description": "Collect server tx_bytes",
            "tags": ["traffic counter"],
            "units": "Bps"
        }
    ]
}
```

As seen above, the endpoint is given an input of probe metadata, a processor instance list, and output stage list.

Probe metadata is composed of the following fields:

label human-readable probe label; required,

description optional description of the probe,

tags list of strings with the probe tags; optional,

disabled optional boolean that tells whether probe should be disabled. Disabled probes don't provide any data and don't consume any resources. The probe is not disabled by default.

Each processor instance contains an instance name (defined by user), processor type (a selection from a catalog defined by the platform and the reference design), and `inputs` and/or `outputs`. All additional fields in each processor are specific to that type of processor, are specified in the `properties` sub-field, and can

be learned by introspection via our introspection API at `/api/blueprints/<blueprint_id>/telemetry/processors`; we will go over this API later.

Matching our working example, we will go through each entry we have in the processor list in the above example.

In the first entry, we have a processor instance of type `if_counter` that we name `server_tx_bytes`. It takes as input a query called `graph_query` which is a graph query. It then has two other fields named `interface` and `system_id`. These three fields together indicate that we want to collect a (first time-derivative of) counter for every server-facing port in the system. For every match of the query specified by `graph_query`, we extract a system_id by taking the `system_id` field of the `sys` node in the resulting path (as specified in the `system_id` processor field) and an interface name by taking the `if_name` field of the `intf` node in the resulting path (as specified in the `interface` processor field). The combination of system ID and interface is used to identify an interface in the network, and its tx_bytes counter (as specified by `counter_type`) is put into the output of this processor. The output of this processor is of type "Number Set" (NS); stage types are discussed exhaustively later. This processor has no inputs, so we do not supply an `input` field. It has one output, labeled `out` (as defined by the if_counter processor type); we map that output to a stage labeled `server_tx_bytes_output`.

The second processor is of type `std_dev` and takes as input the stage we created before called `server_tx_bytes_output`; see the processor-specific documentation for the meaning of the `ddof` field. Also, see the processor-specific documentation for the full meaning of the `group_by` field. It will suffice to say for now that in this case `group_by` tells us to construct a single output "Number" (N) from the input NS; that is, this processor outputs a single number-the standard deviation taken across each of the many input numbers. This output is named "std_dev_output".

The third processor is of type `range_check` and takes as input `std_dev_output`. It checks that the input is out of the expected range specified by `range` - in this case if the input is ever greater-than 100 (we have chosen this arbitrary value to indicate when the server-directed traffic is unbalanced). This processor has a single output we choose to label `std_dev_output_in_range`. This output (as defined by the range_check processor type) is of type DS (Discrete State) and can take values either `true` or `false`, indicating whether or not a value is out of the range.

Our final processor is of type `anomaly` and takes as input `std_dev_output_in_range`. It raises an Apstra anomaly when the input is in the `true` state. This processor has a single output we choose to label `server_traffic_imbalanced`. This output (as defined by the anomaly processor type) is of type DS (Discrete State) and can take values either `true` or `false`, indicating whether or not an anomaly is raised. We do not do any further processing with this anomalous state data in this example, but that does not preclude its general possibility.

Finally, we have a `stages` field. This is a list of a subset of output stages, with each stage indicated by the `name` field which refers to the stage label. This list is meant to add metadata to each output stage that cannot be inferred from the DAG itself. Currently, supported fields are:

description string with a stage description,

tags list of strings that make a set of tags for stage,

units string that is meant to describe the units of the stage data.

All these fields are optional.

This stage metadata is returned when fetching data from that stage via the REST API and used by the GUI in visualization.

HTTP POST can be sent to `/api/blueprints/<blueprint_id>/probes`. Here, we POST probe configuration, as exemplified in the "POST for Probe Creation" figure to create a new probe. POSTing to this endpoint will return a UUID, as most of the other creation endpoints in Apstra, which can be used for further operations.

Changed in version 2.3: To get a predictable probe id instead of a UUID described above, one could specify it by adding an "id" property to the request body.

```
{
    "id": "my_tx_bytes_probe",
    "label": "server_tx_bytes",
    "processors": [],
    "rest_of_the": "request_body"
}
```

Changed in version 2.3: Previously, stage definitions were inlined into processor definitions like this:

```
{
    "label": "test probe",
    "processors": [
        {
            "name": "testproc",
            "outputs": {"out": "test_stage"},
            "stages": [{"name": "out", "units": "pps"}]
        }
    ]
}
```

This no longer works, and stage name should refer to the stage label instead of the internal stage name. So the example above should look this way:

```
{
    "stages": [{"name": "test_stage", "units": "pps"}]
}
```

Additional note: it's recommended not to inline stage definitions into processor definitions, and place that as a stand-alone element like in POST example above.

HTTP DELETE can be sent to `/api/blueprints/<blueprint_id>/probes/<probe_id>` where to delete the probe specified by its `probe_id`.

HTTP GET can be sent to `/api/blueprints/<blueprint_id>/probes/<probe_id>` to retrieve the configuration of the probe as it was POSTed. It will contain more fields than it was specified at probe creation:

id with id of the probe (or UUID if it was not specified at creation time),

state with actual state of the probe; possible values are "created" for a probe being configured, "operational" for a successfully configured probe, and "error" if probe configuration has failed.

last_error contains detailed error description for the most-recent error for probes in the "error" state. It has the following sub-fields:

- level: a message level, such as "error" or "info".
- message: text with error details.
- timestamp: when the message was registered.

The complete list of probe messages could be obtained by issuing HTTP GET request to `/api/blueprints/<blueprint_id>/probes/<probe_id>/messages`.

Messages are sorted by the 'timestamp' field, oldest come first.

Additionally, HTTP GET can be sent to `/api/blueprints/<blueprint_id>/probes` to retrieve all the probes for blueprint `<blueprint_id>`.

2.3

HTTP PATCH and PUT methods for probes are available since Apstra version 2.3.

HTTP PATCH can be sent to `/api/blueprints/<blueprint_id>/probes/<probe_id>` to update the probe metadata or disable or enable the probe.

```
{
    "label": "new server_tx_bytes",
    "description": "some better probe description",
    "tags": ["production"],
    "stages": [
        {
            "name": "server_tx_bytes",
            "description": "updated stage description",
            "tags": ["server traffic"],
            "units": "bps"
        }
    ]
}
```

This example updates probe metadata for the probe that was created with the POST request listed above. All fields here are optional, values that were not specified remain unchanged.

Every stage instance is also optional, that is, only specified stages will be updated, and not specified stages remain unchanged.

Tags collection is updated entirely, i.e. if it was `tags: ["a", "b"]` and the PATCH payload specified `tags: ["c"]`, then the resulting collection will look like `tags: ["c"]` (NOT `tags: ["a", "b", "c"]`).

With PATCH it's not possible to change probe's set of processor and stages. Please read further for PUT description which allows to do that.

HTTP PUT can be sent to `/api/blueprints/<blueprint_id>/probes/<probe_id>` to replace a probe.

This is very similar to POST, with the difference being that it replaces the old configuration for probe `<probe_id>` with the new one specified in the payload. Payload format for this request is the same as for POST, but `id` is not allowed.

Inspect Probe

Stages are implicitly created by being named in the input and output of various processors. You can inspect the various stages of a probe. The API for reading a particular stage is `/api/blueprints/<blueprint_id>/probes/<probe_id>/stages/<stage_name>`

NOTE: Each stage has a type. This is a function of the generating processor and the input stage(s) to that processor. The types are: Number (N); Number Time Series (NTS), Number Set (NS); Number Set Time Series (NSTS); Text (T); Text Time Series (TTS); Text Set (TS); Text Set Time Series (TSTS); Discrete State (DS); Discrete State Time Series (DSTS); Discrete State Set (DSS); Discrete Set Time Series (DSSTS)

A NS is exactly that: a set of numbers.

Similarly, a DSS is a set of discrete-state variables. Part of the specification of a DSS (and DSSTS) stage is the possible values the discrete-state variable can take.

A text set is a set of strings.

A NSTS is a set of time-series with numbers as values. For example, a member of this set would be: (time=0 seconds, value=3), (time=3 seconds, value=5), (time=6 seconds, value=23), and so-on.

An DSTS is the same as an NSTS except values are discrete-state.

An TSTS is the same as an NSTS except values are strings.

Number (N), Discrete-State (DS), and Text (T) are simply Number Sets, Discrete State Sets, and Text Sets guaranteed to be of length one.

NTS, DSTS, and TS are the same as above, but are time-series instead of single values.

Let's consider the first stage - "server_tx_bytes". This stage contains the tx_bytes counter for every server-facing port in the system. We can get it from the url `/api/blueprints/<blueprint_id>/probes/<probe_id>/stages/server_tx_bytes_output`

The response we get would be of the same form as the following:


```
{
    "properties": [
        "interface",
        "system_id"
    ],
    "type": "ns",
    "units": "bytes_per_second",
    "values": [
        {
            "properties": {
                "interface": "intf1",
                "system_id": "spine1"
            },
```

```
            "value": 22
        },
        {
            "properties": {
                "interface": "intf2",
                "system_id": "spine1"
            },
            "value": 23
        },
        {
            "properties": {
                "interface": "intf1",
                "system_id": "spine3"
            },
            "value": 24
        }
    ]
}
```


As we know from our running example, the "server_tx_bytes" stage contains the tx_bytes value for every server-facing interface in the network. Looking at the above example, we can see that this stage is of type "ns", indicating NS or Number-Set. As mentioned before, data in stages is associated with context. This means that every element in the set of a stage is associated with a group of key-value pairs. Per every stage, the keys are the same for every piece of data (or, equivalently, item in the set). These keys are listed in the "properties" field of a given stage, and are generally a function of the generating processor. Each of the items in "values" assigns a value to each of the properties of the stage and provides a value (the "Number" in the "Number Set"). The meaning of this data in this stage is that tx_bytes on intf1 of spine1 is 22, on intf2 of spine1 is 23, and on intf1 of spine3 is 24 bytes per second.

Notice that "units" is set for this stage as specified in the running example.

To query the second stage in our probe, send an HTTP GET to the std endpoint `/api/blueprints/<blueprint_id>/probes/<probe_id>/stages/std_dev_output`.


```
{
    "type": "n",
    "units": "",
    "value": 1
}
```


This stage is a number. It has no context, only a single value. In our example, this is the standard deviation across all spines.

The penultimate stage in our probe can be queried at the endpoint `/api/blueprints/<blueprint_id>/probes/<probe_id>/stages/server_traffic_imbalanced`.

```
{
    "possible_values": [
        "true",
        "false"
    ],
    "type": "ds",
    "units": "",
    "value": false
}
```

As shown, this stage indicates whether server traffic is imbalanced ("true") or not ("false") by indicating if the standard deviation across of tx_bytes across all server-facing ports is greater-than 100. Note the "possible_values" field describes all values that the discrete-state "value" can take.

All processors of a probe can also be queried via `/api/blueprints/<blueprint_id>/probes/<probe_id>/processors/<processor_name>`. By doing such a query, you can discover the configuration used for creation of said processor.

Query Probe Anomalies

The final stage of our example processor raises an Apstra Anomaly (and sets its output to "true"), when the standard deviation of tx_bytes across server-facing interfaces is greater-than 100.

You can query probe anomalies via the standard anomaly API at `/api/blueprints/<bluprint_id>/anomalies?type=probe`.

Following is the JSON form of an anomaly that would be raised by our example probe (with ellipses for data we don't care about for this example):

```
{
    "actual": {
        "value_int": 101
    },
    "anomaly_type": "probe",
    "expected": {
        "value_int": 100
    },
    "id": "...",
    "identity": {
```

```
        "anomaly_type": "probe",
        "probe_id": "efb2bf7f-d8cc-4a55-8e9b-9381e4dba61f",
        "properties": {},
        "stage_id": "server_traffic_imbalanced"
    },
    "last_modified_at": "...",
    "severity": "critical"
}
```

As seen in the above example, the identity contains the probe_id and the name of the stage on which the anomaly was raised and which requires further inspection by the operator. Within a given stage, if the type of the stage were a set-based type, the "properties" field of the anomaly would be filled with the properties of the specific item in the set that caused the anomaly. This brings up the important point that multiple anomalies can be raised on a single stage, as long as each is on a different item in the set. In our example, since the stage in question is of type NS, the "properties" field is not set.

Introspect Processors

The set of processors available to the operator is a function of the platform and the reference design. Apstra provides an API for the operator to list all available processors, learn what parameters they take, and learn what inputs they require and outputs they yield.

The API in question is found at `/api/blueprints/<blueprint_id>/telemetry/processors`.

It yields a list of processor descriptions. In the following example, we show the description for the std_dev processor.

```
{
    "description": "Standard Deviation Processor.\n\n Groups as described by group_by, then
calculates std deviation and\n outputs one standard deviation for each group. Output is NS.\n
Input is an NS or NSTS.\n ",
    "inputs": {
        "in": {
            "required": true,
            "types": [
                {
                    "keys": [],
                    "possible_values": null,
                    "type": "ns"
                },
                {
                    "keys": [],
```

```
                    "possible_values": null,
                    "type": "nsts"
                }
            ]
        }
    },
    "outputs": {
        "out": {
            "required": true,
            "types": [
                {
                    "keys": [],
                    "possible_values": null,
                    "type": "ns"
                }
            ]
        }
    },
    "label": "Standard Deviation",
    "name": "std_dev",
    "schema": {
        "additionalProperties": false,
        "properties": {
            "ddof": {
                "default": 0,
                "description": "Standard deviation correction value, is used to correct divisor
(N - ddof) in calculations, e.g. ddof=0 - uncorrected sample standard deviation, ddof=1 -
corrected sample standard deviation.",
                "title": "ddof",
                "type": "integer"
            },
            "enable_streaming": {
                "default": false,
                "type": "boolean"
            },
            "group_by": {
                "default": [
                    "system_id"
                ],
                "items": {
                    "type": "string"
                },
                "type": "array"
```

```
            }
        },
        "type": "object"
    }
}
```

As seen above, there is a string-based description, the name of type processor type (as supplied to the REST API in probe configuration). The set of parameters specific to a given probe is described in the "schema".

Special notice must be paid to "inputs" and "outputs". Even though these are in the "schema" section, they are present on every type of processor. Each processor can take zero-or-more more input stages and must output one-or-more stages. Optional stages have "required" set to false. The names of the stages (relative to a particular instance of a processor) they take are described in these variables. We can see that the "std_dev" processor takes a single input named "in" and a single output named "out". This is reflected in our usage of it in the previous example.

There's one special input name: `*`. For example:

```
"inputs": {
    "*": {
        "required": true,
        "types": [
            {
                "keys": [],
                "possible_values": null,
                "type": "ns"
            },
            {
                "keys": [],
                "possible_values": [],
                "type": "dss"
            },
            {
                "keys": [],
                "possible_values": null,
                "type": "ts"
            }
        ]
    }
}
```

It means the processor accepts one or more inputs of the specified types with arbitrary names.

Changed in 3.0: Previously, inputs and outputs section didn't specify whether specific inputs or outputs were required, so the format was changed from the following:

This syntax is deprecated and invalid.

```
 "inputs": {
     "in": [
         {
             "data_type": "ns",
             "keys": [
                 "system_id"
             ],
             "value_map": null,
             "value_type": "int64"
         }
         ...
     ]
}
```

Stream Data

Any processor instance in any probe can be configured to have its output stages streamed in the "perfmon" channel of Apstra streaming output. If the property "enable_streaming" is set to "true" in the configuration for any processor, its output stages will have all their data streamed.

For Non-Time-Series-based stages, each will generate a message whenever their value changes. For Time-Series based stages, each will generate a message whenever a new entry is made into the time-series. For Set-based stages, each item in the set will generate a message according to the two prior rules.

Each message that is generated has a value, a timestamp, and a set of key-value pairs. The value is self-explanatory. The timestamp is the time at which the value changed for Non Time-series-based stages and the timestamp of the new entry for Time-series based stages. The key-value pairs correspond to the "properties" field we observed earlier in the "values" section of stages, thus providing context.

Below we have the format for messages from IBA which is encapsulated in a PerfMon message (and that in-turn in an AosMessage). The key-value pairs of context are put into the "property" repeated field (with "name" as the key and "value" as the value) while the value is put into the "value" field. "probe_id" and

"stage_name" are as they appear. The blueprint_id is put into the "origin_name" of the encapsulated AosMessage. Similarly the timestamp is put into the generic "timestamp" field.

```
message ProbeProperty {
  required string name = 5;
  required string value = 6;
}
message ProbeMessage {
  repeated ProbeProperty property = 1;
  oneof value {
    int64 int64_value = 2;
    float float_value = 3;
    string string_value = 4;
  }
required string probe_id = 5;
required string stage_name = 6;
}
```

RCI Fault Model (API)

IN THIS SECTION

You can access complete Apstra API documentation from the web interface in the **Platform > Developers** section.

- A blueprint is associated with zero or more Root Cause Identification instances.
- Root Cause Identification instances are enabled (created) / disabled (deleted) via CRUD API for Root Cause Identification sub-resource under the blueprint.

- The instances that can be created depends on the reference design of the blueprint. In this first phase of Root Cause Identification, only two_stage_l3clos has Root Cause Identification support, and right now it only allows one Root Cause Identification instance per blueprint.

Create Root Cause Identification Instance

```
POST /api/blueprints/<blueprint_id>/arca
  Request Payload schema
  {
  "model_name":  s.String() # Name of ARCA instance's system fault model (ref
  design specific)
  "trigger_period": s.Float(min=10.0) # ARCA instance runs every <trigger_period>
  seconds.
  }
```

Example for blueprints for ref design two_stage_l3clos:

```
{
    "model_name": "default",
    "trigger_period": 10.0
}

Return values:
201 - Successfully created the RCI instance. Response payload:

{"id": <RCI  instance ID>}

The ID is used in GET, PUT, DELETE

404 - Blueprint does not exist or is not deployed
422 - Validation error. Response payload:

{"error": <message>}

Possible error messages:
Model name is not found for the reference design
An ARCA instance already exists for given model name
trigger_period is too small
```

Update Root Cause Identification Instance

Using the PUT API, you can tweak the execution frequency of the Root Cause Identification instance.

```
PUT /api/blueprints/<blueprint_id>/arca/<arca_id>
    Request Payload schema
    {
    "trigger_period": s.Float(min=10.0)
    }

Return values:
200 - Update succeeded.
404 - ARCA instance not found.
422 - Validation error. Response payload:

{"error": <message>}

Possible error messages:
trigger_period is too small
```

Delete Root Cause Identification Instance

Using the GET API, you can obtain the current status (set of root causes) of the Root Cause Identification instance.

```
GET /api/blueprints/<blueprint_id>/arca/<arca_id>

Return values:
200 - see response schema below
404 - ARCA instance not found
```

```
Response payload schema
{
    "id": String,    # ARCA instance ID
    "model_name": String,    # see POST payload
    "trigger_period": Float,   # see POST payload
    "state":  Enum("created", "operational"),
    "config_updated_at": Timestamp # of last update to instance via POST/PUT
    "status_updated_at": Timestamp # of last update to ARCA results
```

```
    "root_cause_count": Integer(min=0)   # Number of root causes identified
    "root_causes": List(ROOT_CAUSE_OBJ)  # Actual root causes
}
```

Timestamps are in ISO8601 format in UTC timezone, e.g. “2018-10-16T22:12:34+0000” If state == “created”, then Status_updated_at == UNIX epoch root_cause_count == 0 “root_causes” key is not returned

Each ROOT_CAUSE_OBJ has the following schema:

```
{
    "id": String,   # Unique ID for the root cause in the ARCA instance
    "context": String,   # Encoded context such as references to graph nodes
    "description": String,   # Human-readable text, e.g. "link <blah> broken"
    "timestamp":  Timestamp,   # of when RC is detected (ISO8601 format)
    "symptoms": List(SYMPTOM_OBJ),   # List of symptoms; always non-empty
}
```

Notes on root cause detection and IDs: A root cause may be detected multiple times over the blueprint’s lifetime. For instance, a root cause is defined for broken cable between spine1 and leaf1. This root cause can appear at any time, and it may disappear once the problem is fixed. A root cause has a unique ID scoped in the ARCA instance. This means that the ID may appear and disappear corresponding to whether the problem occurs or gets fixed, e.g. cable gets broken or reconnected What to expected as root cause ID: In two_stage_l3clos the root cause ID is a composition of graph node and relationship IDs, and some immutable but readable name of the root cause. Example: <graph link node id>/broken.

Each SYMPTOM_OBJ has the following schema:

```
{
    "id": String,   # Unique ID for the symptom in the ARCA instance
    "context": String,  # Encoded context such as system ID, service name
    "description": String, # Readable, e.g. "interface swp1 on leaf1 is down"
}
```

Given the same ARCA system fault model, the set of symptom IDs are always the same for given root cause. However, the context may be different. For instance, the symptom “interface swp1 on leaf1 is down” is the same, while context of different instances of this symptom may have different system IDs depending on which system ID is assigned to leaf1 when the root cause for this symptom is detected. Example symptom ID: <graph interface node id>/down

List Root Cause Identification Instances

```
GET /api/blueprints/<blueprint_id>/arca

Return values
200 - see response schema below
404 - blueprint not found or blueprint not deployed
```

Response schema:

```
{
    "items": List(ARCA_INSTANCE_DIGEST),  # list may be empty
}
```

ARCA_INSTANCE_DIGEST has the same schema as the response payload of GET individual ARCA instance, except that it does not contain the “root_causes” key.

In this phase, for two_stage_l3clos blueprints, there is at most 1 element in the list, because only 1 ARCA instance is allowed per blueprint.

Health Check Apstra VMs (API)

NOTE: You can also check the health of Apstra VMs from the Apstra GUI.

From the left navigation menu of the Apstra GUI, navigate to **Platform > Developers** to access REST API documentation. From there you can access cluster APIs.

```
/api/cluster/nodes/{node_id}  .. Get AOS slave node status.
/api/cluster/nodes/{node_id}/errors  .. Retrieve error for an AOS cluster node.
```

Here is an example of REST API with curl command:

```
curl -X GET "https://172.20.159.3/api/cluster/nodes/AosController/errors" -H "accept:
application/json"
```

If no error occurs, the output is as follows:

```
{
  "state": "active",
  "errors": []
}
```

If the agent process has rebooted, the error is shown as follows:

```
{
  "state": "active",
  "errors": [
    "agentReboot"
  ]
}
```

API From Python

IN THIS SECTION

Following are examples of Python 3 code using the Apstra API.

API User Login

```
import requests, sys
```

```
# IP of Cloudlabs AOS Server
aos_server = '172.16.90.3'
username = 'admin'
password = 'aos aos'

# authenticate and get a auth token
url = 'https://' + aos_server + '/api/user/login'
headers = { 'Content-Type':"application/json", 'Cache-Control':"no-cache" }
data = '{ \"username\":\"' + username + '\", \"password\":\"' + password + '\" }'
response = requests.request("POST", url, data=data, headers=headers, verify=False)
print('POST',url,response.status_code)
if response.status_code != 201:
    sys.exit('error: authentication failed')
auth_token = response.json()['token']
print(auth_token)
headers = { 'AuthToken':auth_token, 'Content-Type':"application/json", 'Cache-Control':"no-
cache" }
```

API - Blueprints

```
# get blueprint ID ... assuming there is only one
url = 'https://' + aos_server + '/api/blueprints'
response = requests.request('GET', url, headers=headers, verify=False)
print('GET', url, response.status_code)
blueprint_id = response.json()['items'][0]['id']
blueprint_name = response.json()['items'][0]['label']
print(blueprint_name, blueprint_id)
```

API - Blueprint Racks

```
# get a list of racks
bound_to = ''
url = 'https://' + aos_server + '/api/blueprints/' + blueprint_id + '/racks'
response = requests.request('GET', url, headers=headers, verify=False)
print('GET', url, response.status_code)
for item in response.json()['items']:
        bound_to += '{\"system_id\":\"' + item['leafs'][0]['id'] + '\"},'
bound_to = bound_to[:-1]
print(bound_to)
```

API - Blueprint Routing Zones (Security Zones)

```
# get routing zone ID ... assuming there is only one
url = 'https://' + aos_server + '/api/blueprints/' + blueprint_id + '/security-zones'
response = requests.request('GET', url, headers=headers, verify=False)
print('GET', url, response.status_code)
for item in response.json()['items']:
    if(response.json()['items'][item]['vrf_name'] != 'default'):
        security_zone_name = response.json()['items'][item]['vrf_name']
        security_zone_id = item
        break
print(security_zone_name, security_zone_id)
```

API - Blueprint Virtual Networks

```
# create a virtual network
vn_name = "My-VN"
url = 'https://' + aos_server + '/api/blueprints/' + blueprint_id + '/virtual-networks'
data = '{\"label\":\"' + vn_name + '\",\"vn_type\":\"vxlan\",\"bound_to\":[' + bound_to +
'],\"security_zone_id\":\"' + security_zone_id + '\"}'
print(data)
response = requests.request('POST', url, data=data, headers=headers, verify=False)
print('POST', url, response.status_code)
```

Run Python

The proceeding Python3 code can be run on the Cloudlabs AOS Server. Use the `python3.6` command to run the Python script.

```
admin@aos-server:~$ python3.6 test.py
POST https://192.168.3.3/api/user/login 201
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImFkbWluIiwiY3JlYXRlZF9hdCI6IjIwMjAtMDItMjFU
MTc6NDM6NTkuNTU1MDQzIiwidXNlcl9zZXNzaW9uIjoiMmU0Y2YwODktNzZmYS00NDg4LTlhNmItYWViYjc3MmQyNDE2IiwiZ
XhwIjoxNTgyMzkzNDM5fQ.GWsy292pfpPVpisbQNKc3EHrDlxh1OUmpQaQ-dF-mwY
GET https://192.168.3.3/api/blueprints 200
neil-blueprint cbfe7a43-4da7-4b2c-90a2-ea0bae4ed79a
GET https://192.168.3.3/api/blueprints/cbfe7a43-4da7-4b2c-90a2-ea0bae4ed79a/racks 200
{"system_id":"2cbb0fc0-5f87-4671-8d8b-e909cbf84fdd"},{"system_id":"98002bb9-
d0a9-484c-86e7-2aac2b926bf7"},{"system_id":"73bd231c-f78e-499f-bf98-fa80c1102a4a"},
```

```
{"system_id":"19fb6155-e9eb-4ae7-b5b3-933416f0e3cd"}
GET https://192.168.3.3/api/blueprints/cbfe7a43-4da7-4b2c-90a2-ea0bae4ed79a/security-zones 200
Finance 4aaa4499-3194-4904-a1ae-daabbe3ed329
{"label":"My-VN","vn_type":"vxlan","bound_to":[{"system_id":"2cbb0fc0-5f87-4671-8d8b-
e909cbf84fdd"},{"system_id":"98002bb9-d0a9-484c-86e7-2aac2b926bf7"},{"system_id":"73bd231c-
f78e-499f-bf98-fa80c1102a4a"},{"system_id":"19fb6155-e9eb-4ae7-
b5b3-933416f0e3cd"}],"security_zone_id":"4aaa4499-3194-4904-a1ae-daabbe3ed329"}
POST https://192.168.3.3/api/blueprints/cbfe7a43-4da7-4b2c-90a2-ea0bae4ed79a/virtual-networks 201
admin@aos-server:~$
```

CHAPTER 51

Technical Support

IN THIS CHAPTER

Juniper Technical Support

Technical Support is available to all customers with a valid and current license for Juniper Apstra software. This includes customers who have purchased a license directly or via a partner or reseller. This also includes customers who have obtained an evaluation license. If your purchased or evaluation license is expired, Juniper Support may not be able to offer support and will refer you to the appropriate sales team to purchase a current license. For more information about working with Juniper Support, refer to Guidelines & Policies.

If you require assistance with registration or with opening a technical support case via phone, call Juniper Customer Care at +1-888-314-5822 (toll free, US & Canada). If you are outside the US or Canada, call +1-408-745-9500 or a country number listed on the Contact Support page.

To aid the support process, we ask that you provide Juniper Support with diagnostic information from the Apstra environment. Separate *show tech* files are needed from the Apstra controller and from each of the affected device agents. You can obtain show tech files, from the GUI (recommended) or the CLI, as described in the next sections. You may also be asked for a "backup" on page 1399 of your Apstra database.

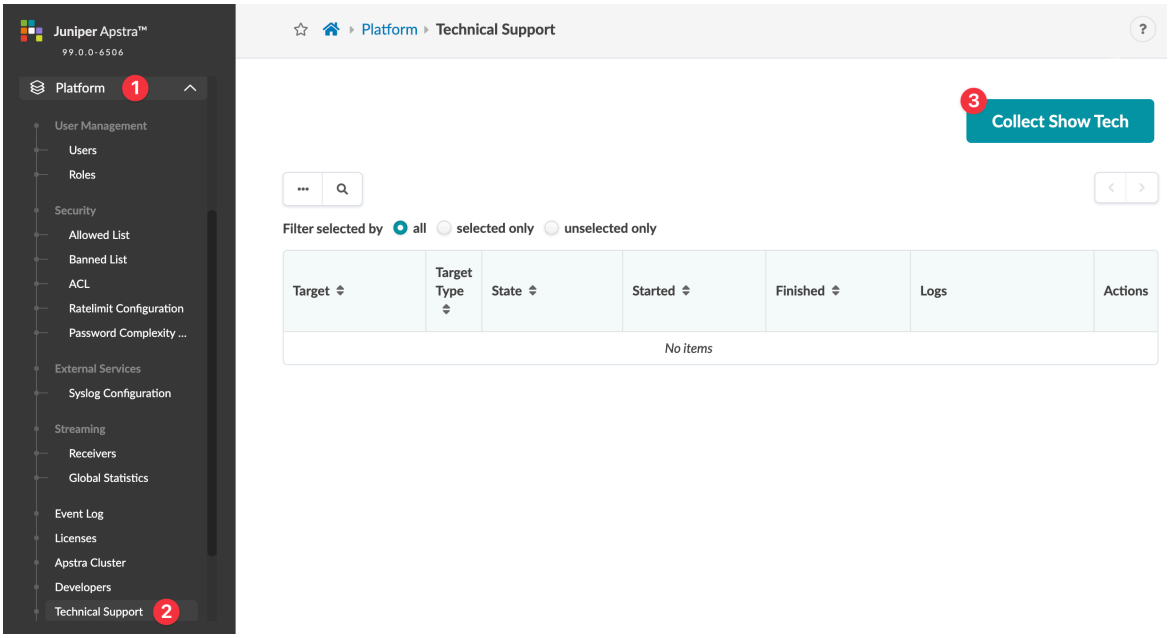
Collect Show Tech (GUI)

You can collect the following show tech files from the Apstra GUI:

- Apstra controller
- Apstra workers (new in Apstra version 5.0.0)
- Connected device agents (onbox and offbox)
- Flow Data

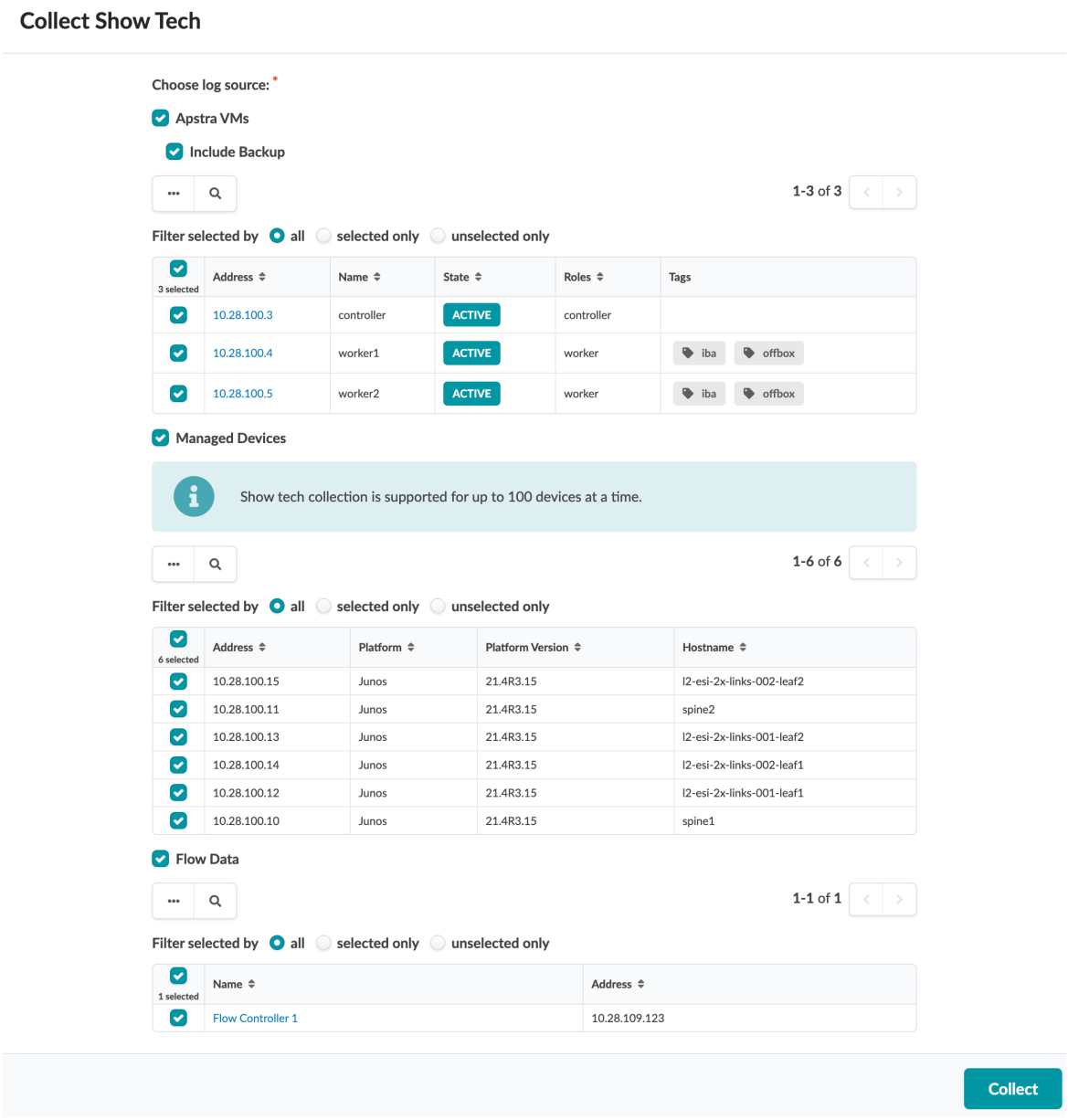
If you haven't configured local credentials for the Apstra controller, from the left navigation menu, navigate to **Platform > Apstra Cluster** and edit the controller to configure credentials. (These are the credentials you use for the VM console or SSH.)

1. From the left navigation menu, navigate to **Platform > Technical Support** and click **Collect Show Tech**.



   The **Collect Show Tech** dialog opens.

2. Select the check box for **Apstra VMs** to see the table of available Apstra VMs (the controller and any workers).



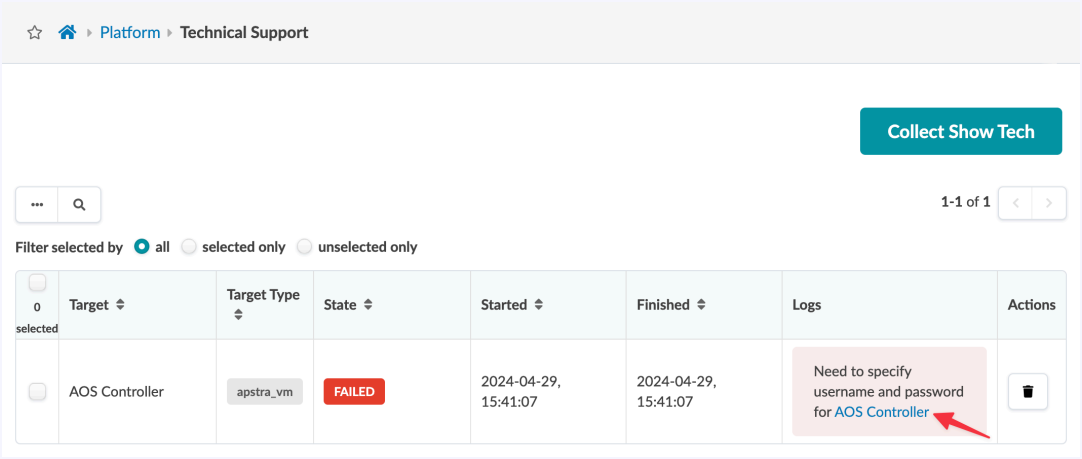
3. Select the check boxes for the VMs that need show tech collected.
4. You can collect show_tech from the Apstra GUI that includes a copy of the backup. If Juniper Support requests a backup, select the **Include Backup** check box. This backup provides information for Support and Engineering. It doesn't include credentials, so it's not suitable for restoring your production environment. (Use backups from the "Back Up Apstra Database" on page 1399 procedure instead.)
5. Select the check box for **Managed Devices** to see the list of managed devices (devices with agents that have been acknowledged).
6. Select the check boxes for the devices that need show tech collected.

NOTE: When device show tech is collected, the configured device system agent username and password authentication are used. If you've configured the device to use a different authentication (AAA) method with a different username and password (such as RADIUS and TACACS) you can't collect show tech from the Apstra GUI. You must "collect show tech with CLI" on page 1381.

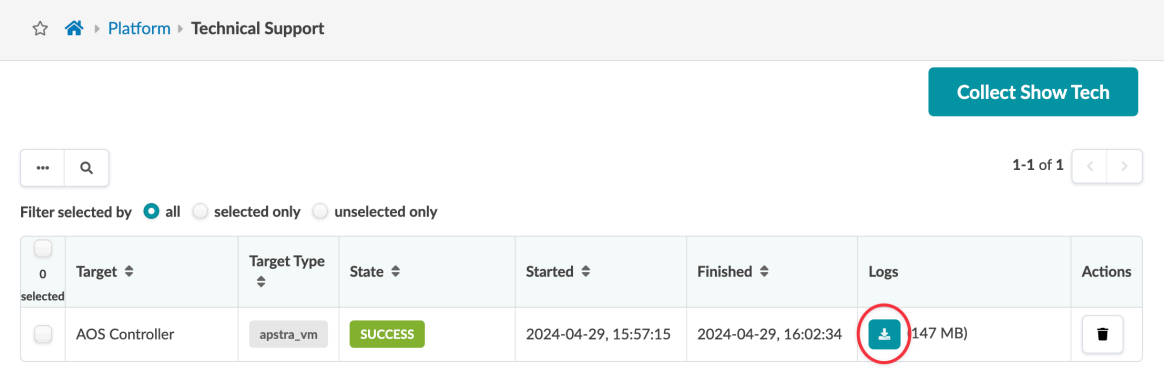
7. If you're using Flow Data and would like to collect show tech for it, check the **Flow Data** check box, then select check boxes for the flow controllers that need show tech collected.
8. Click **Collect** to start the collection process.

NOTE: For Apstra server controllers with large databases, the operation may timeout. If this happens, you must "collect show tech using the CLI" on page 1380.

TIP: If the image below appears, you still need to configure local credentials on the node. Click the link to go to the controller node screen, click the **Edit** button (right side), then enter the username and password you use for the VM console or SSH.



9. After the jobs are complete and marked **SUCCESS**, click the download button for *each* of the files (under **Logs**).



TIP: After the files have been downloaded, you can free up disk space by deleting jobs.

10. From a computer with the ability to upload, upload the show tech files to your customer case.

Collect Show Tech: Offbox Agents (CLI)

We recommend that you use the Apstra GUI to obtain show tech files, but you have the option of using CLI instead, as described below. You'll need the device management IP address(es) and a valid device SSH username and password.

NOTE: If your offbox agents are for infra, you'll collect show tech with a different method. Refer to "Show-Tech: Infra Offbox Agents (CLI)" on page 1379 for details.

1. SSH into the Apstra server that the offbox agent is running on. (`ssh admin@<apstra-server-ip>` where `<apstra-server-ip>` is the IP address of the Apstra server.)
2. To copy the show tech file(s) to your user directory, run the `aos_offbox_show_tech_collector` command with the following arguments:
   - `--ips` <ip address of one or more devices> (for example: 11.29.53.7 11.29.53.8 11.29.53.9)
   - `--aos-ip` <ip address of the Apstra server> (for example: 11.29.53.3)
   - `--os-type` <vendor OS type> (for example: junos)
   - `--user` <admin user name> (for example: admin)
   - `--password` <admin password> (for example: xu8&j3d'j1=dHnr)

Example for 3 Devices:

```
admin@aos-server:~$ sudo aos_offbox_show_tech_collector --ips 11.29.53.7 11.29.53.8
11.29.53.9 --aos-ip 11.29.53.3 --os-type junos --user admin
[sudo] password for admin:
SSH password for remote device:
2022-11-15 22:24:09,947 invoking DI container to collect 11.29.53.9 show tech
2022-11-15 22:25:32,778 AOS offbox show tech generated at /home/admin
2022-11-15 22:25:32,805 invoking DI container to collect 11.29.53.8 show tech
2022-11-15 22:26:45,773 AOS offbox show tech generated at /home/admin
2022-11-15 22:26:45,799 invoking DI container to collect 11.29.53.7 show tech
2022-11-15 22:27:55,811 AOS offbox show tech generated at /home/admin

admin@aos-server:~$ ls -l
total 217440
-rw-r--r-- 1 root  root      75958 Nov 15 22:27 11.29.53.7-5254009E6B20-junos-show-tech.tar.gz
-rw-r--r-- 1 root  root      76180 Nov 15 22:26 11.29.53.8-52540039A6F3-junos-show-tech.tar.gz
-rw-r--r-- 1 root  root     107620 Nov 15 22:25 11.29.53.9-5254001A5CEB-junos-show-tech.tar.gz
-rw------- 1 root  root       8737 Nov 15 22:27 aos_di_11.29.53.7_show_tech_run.log
-rw------- 1 root  root       8614 Nov 15 22:26 aos_di_11.29.53.8_show_tech_run.log
-rw------- 1 root  root       8491 Nov 15 22:25 aos_di_11.29.53.9_show_tech_run.log

admin@aos-server:~$
```

3. Copy the show tech file(s) to a local computer with the ability to upload.
4. Upload the show tech file to your customer case.

Collect Show Tech: Infra Offbox Agents (CLI)

The instructions below are for collecting show tech files for infra offbox agents. If your offbox agents are not for infra, refer to "Show Tech: Apstra Controller and Device Agents (GUI)" on page 1375 or "Show Tech: Apstra Offbox Agents (CLI)" on page 1378.

1. SSH into the Apstra server that the offbox agent is running on. (`ssh admin@<apstra-server-ip>` where `<apstra-server-ip>` is the IP address of the Apstra server.)
2. Run `docker ps` to get the name of the container (in the NAMES column).

3. Run the `docker exec -ti <offbox_container_name> aos_show_tech` command where `<offbox_container_name>` is the name you retrieved when you ran `docker ps`. For example:

```
admin@aos-server:~$ docker exec -ti aos-offbox-172_20_47_6-f aos_show_tech
AOS show tech generated at /tmp/aos_show_tech_20200401_181128.tar.gz
admin@aos-server:~$
```

4. Using SCP, run the `docker cp` command to copy the show tech file from the offbox agent Docker container to the `/tmp` directory of the Apstra server. For example:

```
admin@aos-server:~$ docker cp aos-offbox-172_20_47_6-f:/tmp/
aos_show_tech_20200401_181128.tar.gz .
admin@aos-server:~$ ls
aos_show_tech_20200401_181128.tar.gz  docker.service.log
admin@aos-server:~$
```

5. Locate the file archive in the `/tmp` directory and copy it to a local computer with the ability to upload. Then upload the show tech file to your customer case.

Collect Show Tech: Apstra Controller (CLI)

We recommend using the "Apstra GUI" on page 1375 to obtain Apstra server show tech files, but you have the option of using the Apstra server Linux CLI instead, as described below.

1. SSH into the Apstra server. (`ssh admin@<apstra-server-ip>` where `<apstra-server-ip>` is the IP address of the Apstra server.)
2. Run the `sudo aos_show_tech` command to generate and copy the show tech file to the current working directory of the Apstra server. For example:

```
admin@aos-server:~$ sudo aos_show_tech
[sudo] password for admin:
Generating technical support data under directory /tmp/tmp.YmjuJDhatJ
--- collecting sysinfo/cpuinfo from /proc/cpuinfo ---
--- collecting network/etc_hosts from /etc/hosts ---
--- collecting aos/aos.conf from /etc/aos/aos.conf ---
--- collecting sysinfo/meminfo from /proc/meminfo ---
--- collecting sysinfo/vmstat from /proc/vmstat ---
--- collecting network/etc_hostname from /etc/hostname ---
--- collecting network/interfaces_config from /etc/network/interfaces ---
--- collecting network/resolv.conf from /etc/resolv.conf ---
```

```
--- collecting logs/kern_log from /var/log/kern.log* ---
--- collecting logs/syslog from /var/log/syslog* ---
--- collecting filesystem/aos_cachaca_db_usage with command: du -a /var/lib/aos/cachaca ---
--- collecting sysinfo/uptime with command: uptime ---
--- collecting filesystem/aos_db_usage with command: du -a /var/lib/aos/db ---
--- collecting filesystem/disk_free with command: df -h ---
[snip]
Remaining dump took 8.477 ms
2020-04-01 03:35:39,010 131:INFO:aos.infra.core.entity_util:Create partition mount factory
for partition Anomaly
Dumping entity (anomaly_sysdb_dump/Tac) took 0.389 ms
Dumping entity (anomaly_sysdb_dump/alert_aggregation) took 3.986 ms
Dumping entity (anomaly_sysdb_dump/streaming) took 0.173 ms
Dumping entity (anomaly_sysdb_dump/alerts) took 4.174 ms
Dumping entity (anomaly_sysdb_dump/counters) took 0.160 ms
Dumping entity (anomaly_sysdb_dump/telemetry_adaptor) took 0.156 ms
Dumping entity (anomaly_sysdb_dump/deployment) took 0.214 ms
Dumping entity (anomaly_sysdb_dump/device) took 0.675 ms
Dumping entity (anomaly_sysdb_dump/cachaca) took 0.144 ms
Dumping entity (anomaly_sysdb_dump/var) took 0.201 ms
Skipping SysDB dump
Archiving show tech data into aos_show_tech_20200401_033431.tar.gz
Removing working directory /tmp/tmp.YmjuJDhatJ
All done.
admin@aos-server:~$
```

3. Locate the file in the working directory and copy the file to a local computer via SCP.
4. Upload the show tech file to your customer case.

Collect Show Tech: Onbox Agents (CLI)

We recommend using the "Apstra GUI" on page 1375 to obtain onbox agent show tech files, but you have the option of using the Apstra server Linux CLI instead, as described below.

1. SSH to the device.
2. For Arista devices, run `bash` to go the Arista Networks EOS shell, then run the command `sudo python3 /usr/bin/aos_show_tech --platform eos` as shown below.

```
l2-virtual-003-leaf1#bash
[admin@l2-virtual-003-leaf1 ~]$ sudo python3 /usr/bin/aos_show_tech --platform eos
```

```
AOS show tech generated at /tmp/aos_show_tech_20240723_182219.tar.gz
[admin@l2-virtual-003-leaf1 ~]$
```

3. For Cisco devices, run `guestshell`, then run the command `sudo python3 /usr/bin/aos_show_tech --platform nxos` as shown below.

```
l2-virtual-002-leaf1# guestshell
[admin@guestshell ~]$ sudo python3 /usr/bin/aos_show_tech --platform nxos
AOS show tech generated at /tmp/aos_show_tech_20240723_182059.tar.gz
[admin@guestshell ~]$
```

4. For SONiC devices, run the command `sudo python3 /usr/bin/aos_show_tech --platform sonic` as shown below

```
[SONiC]

admin@sonic:~$ sudo python3 /usr/bin/aos_show_tech --platform sonic
AOS show tech generated at /tmp/aos_show_tech_20240723_181532.tar.gz
admin@sonic:~$
```

5. Locate the file archive in the `/tmp` directory (for example, `aos_show_tech_20240723_181532.tar.gz`) and copy it, via SCP, to a local computer with the ability to upload.
6. Upload the show tech file to your customer case.

Collect Show Tech: Apstra ZTP (CLI)

To aid the support process for Apstra ZTP, we ask that you provide Juniper Support with diagnostic information from the Apstra ZTP environment. You can obtain show tech files from the Apstra ZTP CLI as described below.

1. SSH into the Apstra ZTP server. (`ssh admin@<apstra-ztp-server-ip>` where `<apstra-ztp-server-ip>` is the IP address of the Apstra ZTP server.)
2. Run the `sudo ztp_show_tech` command to generate and copy the show tech file to the current working directory of the Apstra ZTP server. For example:

```
admin@apstra-ztp:~$ sudo ztp_show_tech
2023-09-05_20:14:23 Generating technical support data under directory /tmp/tmp.0CymRu9K2f
2023-09-05_20:14:23 --- collecting ztp_config/dhcpd.conf from /containers_data/dhcp/
dhcpd.conf ---
```

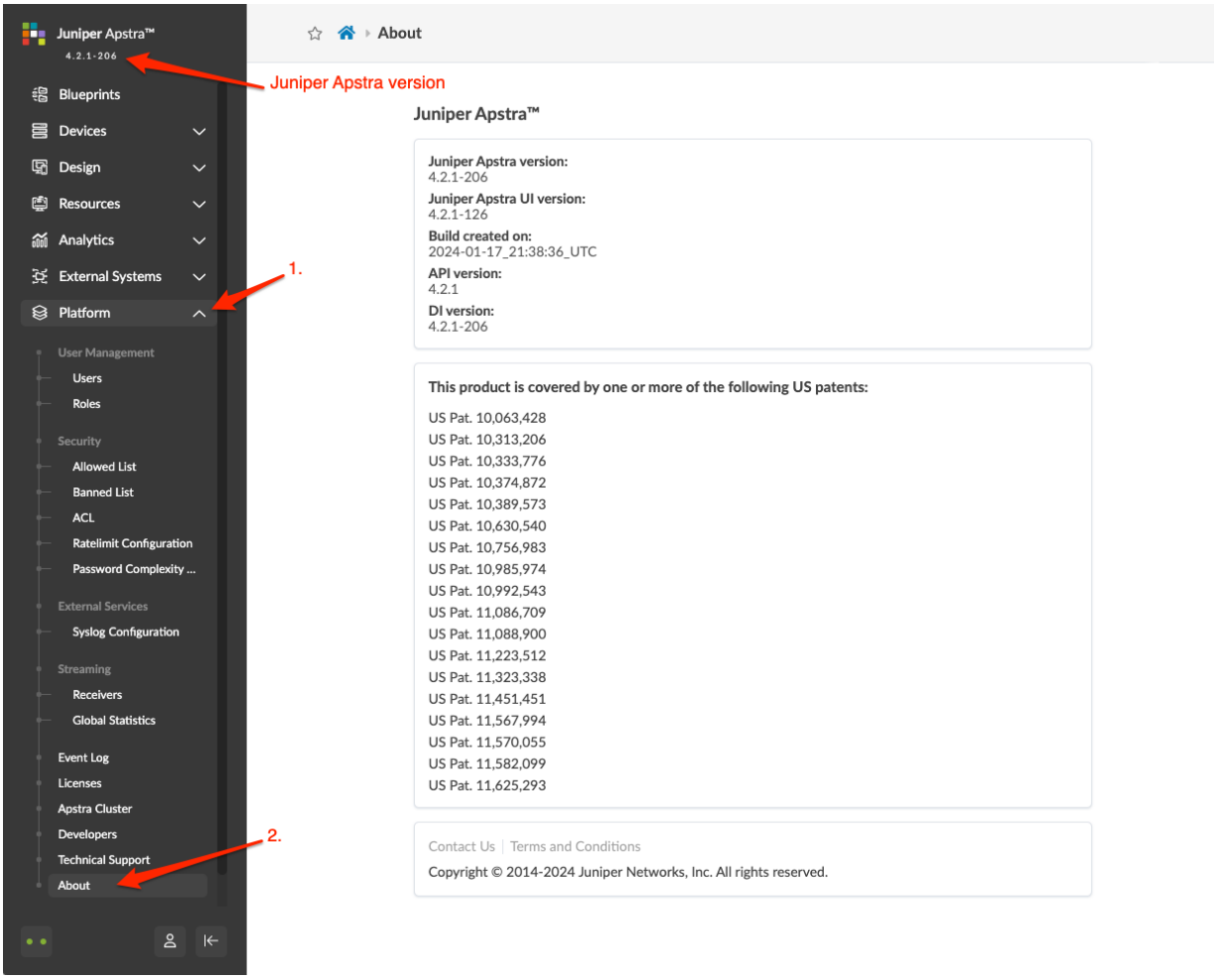
```
2023-09-05_20:14:23 --- collecting system_info/vmstat from /proc/vmstat ---
2023-09-05_20:14:23 --- collecting ztp_config/ztp_version from /etc/apstra_ztp/version ---
2023-09-05_20:14:23 --- collecting system_info/meminfo from /proc/meminfo ---
2023-09-05_20:14:23 --- collecting system_info/syslog from /var/log/syslog ---
2023-09-05_20:14:23 --- collecting system_info/cpuinfo from /proc/cpuinfo ---
2023-09-05_20:14:23 --- collecting ztp_config/docker-compose.yml from /etc/apstra_ztp/docker-
compose.yml ---
2023-09-05_20:14:23 --- collecting logs/ztp from /containers_data/logs/ ---
2023-09-05_20:14:25 --- collecting files from tftp directory ---
'/containers_data/tftp/ztp.py' -> 'ztp_config/tftp/ztp.py'
'/containers_data/tftp/eos_custom.sh' -> 'ztp_config/tftp/eos_custom.sh'
'/containers_data/tftp/sonic_custom.sh' -> 'ztp_config/tftp/sonic_custom.sh'
'/containers_data/tftp/junos_custom.sh' -> 'ztp_config/tftp/junos_custom.sh'
'/containers_data/tftp/junos_apstra_ztp_bootstrap.sh' -> 'ztp_config/tftp/
junos_apstra_ztp_bootstrap.sh'
'/containers_data/tftp/ztp.py.md5' -> 'ztp_config/tftp/ztp.py.md5'
'/containers_data/tftp/nxos_custom.sh' -> 'ztp_config/tftp/nxos_custom.sh'
'/containers_data/tftp/config_verifier.py' -> 'ztp_config/tftp/config_verifier.py'
'/containers_data/tftp/Dockerfile' -> 'ztp_config/tftp/Dockerfile'
'/containers_data/tftp/container_init.sh' -> 'ztp_config/tftp/container_init.sh'
'/containers_data/tftp/rsyslog.conf' -> 'ztp_config/tftp/rsyslog.conf'
'/containers_data/tftp/ztp.json' -> 'ztp_config/tftp/ztp.json'
'/containers_data/tftp/poap-md5sum' -> 'ztp_config/tftp/poap-md5sum'
2023-09-05_20:14:25 --- collecting docker/docker_version with command: docker --version ---
2023-09-05_20:14:25 --- collecting ztp_config/mysql_dump.sql with command: timeout -k 5 30
docker exec db /usr/bin/mysqldump -uadmin -padmin ui; case $? in 124) echo Timeout exception:
command was CANCELED;; 137) echo Timeout exception: command was KILLED;; esac ---
2023-09-05_20:14:27 --- collecting docker/networks with command: timeout -k 5 30 docker
network ls; case $? in 124) echo Timeout exception: command was CANCELED;; 137) echo Timeout
exception: command was KILLED;; esac ---
2023-09-05_20:14:28 --- collecting docker/containers with command: timeout -k 5 30 docker ps -
a; case $? in 124) echo Timeout exception: command was CANCELED;; 137) echo Timeout
exception: command was KILLED;; esac ---
2023-09-05_20:14:28 --- collecting system_info/disk_free with command: df -h ---
2023-09-05_20:14:28 --- collecting system_info/ubuntu_version with command: lsb_release -a ---
2023-09-05_20:14:28 --- collecting docker/docker_compose_version with command: docker-compose
--version ---
2023-09-05_20:14:30 --- collecting docker/daemon.log with command: journalctl -u
docker.service ---
2023-09-05_20:14:30 --- collecting docker/logs/tftp with command: timeout -k 5 30 docker logs
-t tftp; case $? in 124) echo Timeout exception: command was CANCELED;; 137) echo Timeout
exception: command was KILLED;; esac ---
2023-09-05_20:14:46 --- collecting docker/logs/db with command: timeout -k 5 30 docker logs
```

```
db; case $? in 124) echo Timeout exception: command was CANCELED;; 137) echo Timeout
exception: command was KILLED;; esac ---
2023-09-05_20:14:46 --- collecting docker/logs/nginx with command: timeout -k 5 30 docker
logs -t nginx; case $? in 124) echo Timeout exception: command was CANCELED;; 137) echo
Timeout exception: command was KILLED;; esac ---
2023-09-05_20:14:46 --- collecting logs/nginx/error.log with command: docker cp
nginx:/var/log/nginx/error.log logs/nginx/error.log ---
2023-09-05_20:14:48 --- collecting docker/logs/status with command: timeout -k 5 30 docker
logs -t status; case $? in 124) echo Timeout exception: command was CANCELED;; 137) echo
Timeout exception: command was KILLED;; esac ---
2023-09-05_20:14:48 --- collecting logs/nginx/access.log with command: docker cp
nginx:/var/log/nginx/access.log logs/nginx/access.log ---
2023-09-05_20:14:49 --- collecting system_info/memory with command: free -m ---
2023-09-05_20:14:49 --- collecting system_info/containers_data_disk_usage with command: du /
containers_data ---
2023-09-05_20:14:49 --- collecting docker/logs/dhcpd with command: timeout -k 5 30 docker
logs -t dhcpd; case $? in 124) echo Timeout exception: command was CANCELED;; 137) echo
Timeout exception: command was KILLED;; esac ---
2023-09-05_20:15:02 Archiving show tech data into ztp_show_tech_20230905_201423.tar.gz
2023-09-05_20:15:05 Removing working directory /tmp/tmp.0CymRu9K2f
2023-09-05_20:15:05 All done.
admin@apstra-ztp:~$
```

3. Locate the file archive in the current directory (for example, `ztp_show_tech_20230905_201423.tar.gz`), and via SCP, copy the file to a local computer with the ability to upload.
4. Upload the show tech file to your customer case.

Check Apstra Versions and Patent Numbers

From the Apstra GUI, from the left navigation menu, navigate to **Platform > About** to see the Juniper Apstra versions. (You can also find the Juniper Apstra version at the top of the left navigation menu under the Juniper Apstra logo.) This page also includes the U.S. patent numbers that apply to the Juniper Apstra product.


Juniper Apstra™
4.2.1-206
Juniper Apstra version
Blueprints
Devices
Design
Resources
Analytics
External Systems
Platform
1.
User Management
Users
Roles
Security
Allowed List
Banned List
ACL
Ratelimit Configuration
Password Complexity ...
External Services
Syslog Configuration
Streaming
Receivers
Global Statistics
Event Log
Licenses
Apstra Cluster
Developers
Technical Support
About
2.
About
Juniper Apstra™
Juniper Apstra version:
4.2.1-206
Juniper Apstra UI version:
4.2.1-126
Build created on:
2024-01-17_21:38:36_UTC
API version:
4.2.1
DI version:
4.2.1-206
This product is covered by one or more of the following US patents:
US Pat. 10,063,428
US Pat. 10,313,206
US Pat. 10,333,776
US Pat. 10,374,872
US Pat. 10,389,573
US Pat. 10,630,540
US Pat. 10,756,983
US Pat. 10,985,974
US Pat. 10,992,543
US Pat. 11,086,709
US Pat. 11,088,900
US Pat. 11,223,512
US Pat. 11,323,338
US Pat. 11,451,451
US Pat. 11,567,994
US Pat. 11,570,055
US Pat. 11,582,099
US Pat. 11,625,293
Contact Us | Terms and Conditions
Copyright © 2014-2024 Juniper Networks, Inc. All rights reserved.

17

PART

Favorites & User

Favorites & User

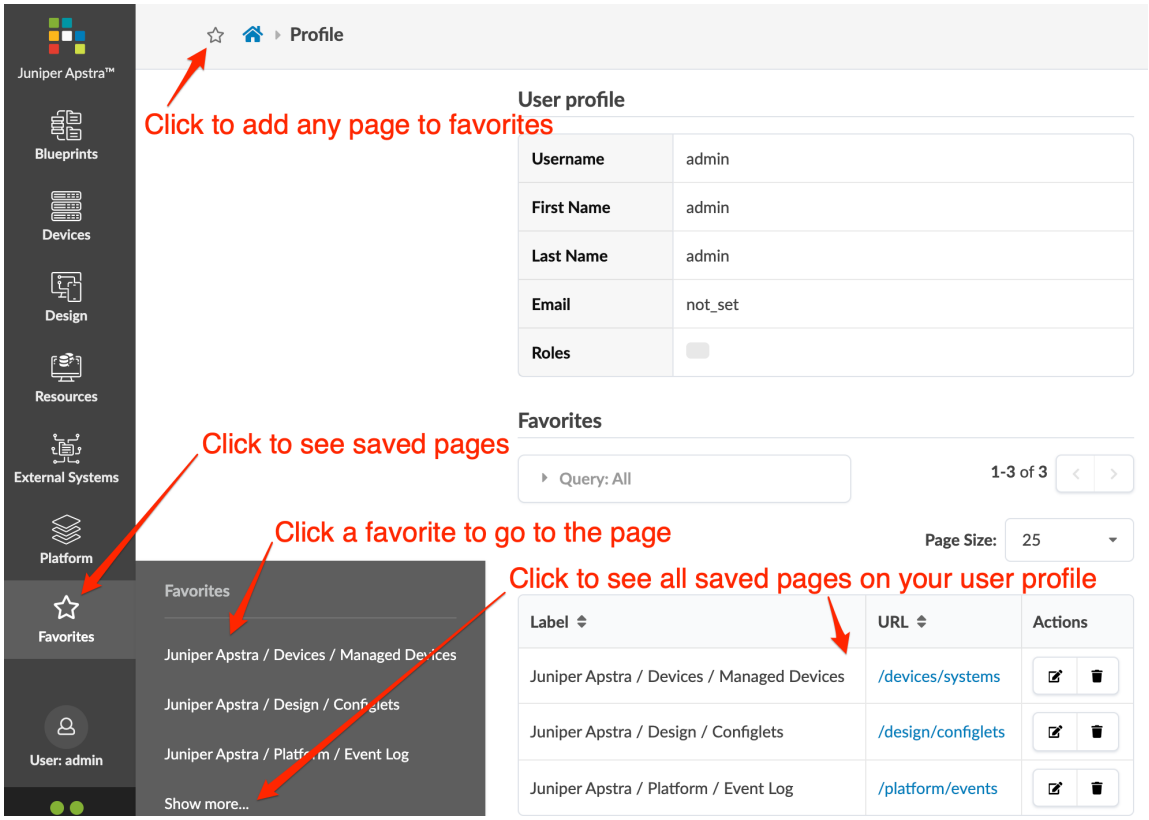
IN THIS SECTION

You can return quickly to frequently visited pages by saving them as favorites. From your user profile page, you can manage favorites, change your password, username and email; and log out of the Apstra software.

Manage Favorites

- To add a favorite - click the star in the upper-left corner of the page to save. Leave the default name or rename it, then click **Add**. The outlined star becomes a shaded star to indicate that it is saved as a favorite.
- To remove a favorite - click the shaded star on the saved page. The star becomes an outline.
- To go to your list of favorites from anywhere in the Apstra GUI, click **Favorites** in the left navigation menu.
  - To go to a favorite page from the **Favorites** menu - click its name. Up to five saved pages appear in the drop-down list.
  - To go to to your list of favorites from the **Favorites** menu - click **Show more** to go to your profile page where you can link to all favorite pages and change their names.
- To go to your profile page to see all your favorites, click your user name in the left navigation menu (bottom), then click **Profile**.
  - To go to a favorite page from your profile page - click its link.
  - To change the name of a link from your profile page - click the **Edit label** button, change the name, then click **Update**.

- To remove a favorite page from your profile page - click the **Remove** button (trash can) and click **Delete**.



Change Your User Password

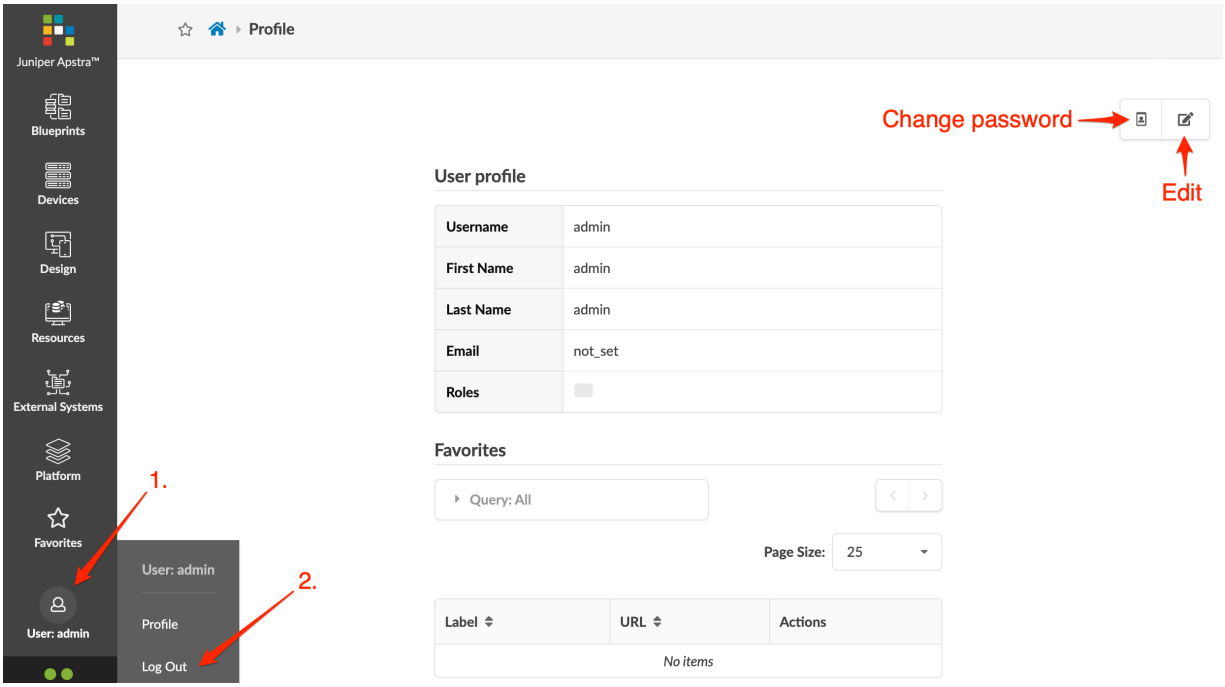
1. From any page, click your username in the left navigation menu (bottom) and click **Profile** to see your profile page.
2. Click the **Change Password** button (top-right), enter your current password, then enter your new password that meets password complexity requirements, twice.
3. Click **Change Password** to update your password and return to your profile.

Change Your User Name/Email

1. From any page, click your username in the left navigation menu (bottom) and click **Profile** to go to your profile page.
2. Click the **Edit** button (top-right), then change your name and/or email, as applicable.
3. Click **Save** to update your details and return to your profile.

Log Out

From any page, click your username in the left navigation menu (bottom) and click **Log Out**. Your viewing preferences (visible fields, show links) are saved so when you log in again, you'll have the same customized views.



PART 18

Apstra Server Management

Apstra Server Introduction

IN THIS SECTION

The information in this section is about *managing* the Apstra server. For information about *installing and upgrading* the Apstra server, see the Juniper Apstra Installation and Upgrade Guide.

Apstra Server Hardening

The Apstra server base OS uses Ubuntu 22.04 LTS to pick up the latest Linux OS improvements. The Apstra server backend uses Python 3.

Monitor Apstra Server via CLI

1. To check general status from the Apstra server CLI, run the command `sudo service aos status`.

```
admin@aos-server:/$ sudo service aos status
[sudo] password for admin:
● aos.service - LSB: Start AOS management system
     Loaded: loaded (/etc/init.d/aos; generated)
     Active: active (exited) since Tue 2023-11-28 17:13:52 UTC; 3 weeks 0 days ago
       Docs: man:systemd-sysv-generator(8)
        CPU: 991ms

Nov 28 17:13:51 aos-server aos[1402]: Container aos_sysdb_1  Starting
Nov 28 17:13:51 aos-server aos[1402]: Container aos_metadb_1  Starting
Nov 28 17:13:51 aos-server aos[1402]: Container aos_nginx_1  Starting
Nov 28 17:13:51 aos-server aos[1402]: Container aos_controller_1  Starting
Nov 28 17:13:52 aos-server aos[1402]: Container aos_nginx_1  Started
Nov 28 17:13:52 aos-server aos[1402]: Container aos_auth_1  Started
```

```
Nov 28 17:13:52 aos-server aos[1402]: Container aos_sysdb_1  Started
Nov 28 17:13:52 aos-server aos[1402]: Container aos_metadb_1  Started
Nov 28 17:13:52 aos-server aos[1402]: Container aos_controller_1  Started
Nov 28 17:13:52 aos-server systemd[1]: Started LSB: Start AOS management system.
admin@aos-server:/$
```

2. To troubleshoot, run the `aos_controller_health_check` script. It searches for known error signatures in the Apstra server logs (such as agent crashes) and returns the output. If no errors are found, no output is returned. See below for sample command.

```
admin@aos-server:~$ docker exec aos_controller_1 aos_controller_health_check
admin@aos-server:~$
```

Restart Apstra Server

To restart the Apstra server you can reboot the VM or run the following commands.

1. Run the command `sudo service aos stop`.

   When the Apstra server is down, device agents may temporarily log "liveness" telemetry alarms.
2. Run the command `sudo service aos start`.

```
admin@aos-server:~$ sudo service aos stop
admin@aos-server:~$ sudo service aos start
admin@aos-server:~$
```

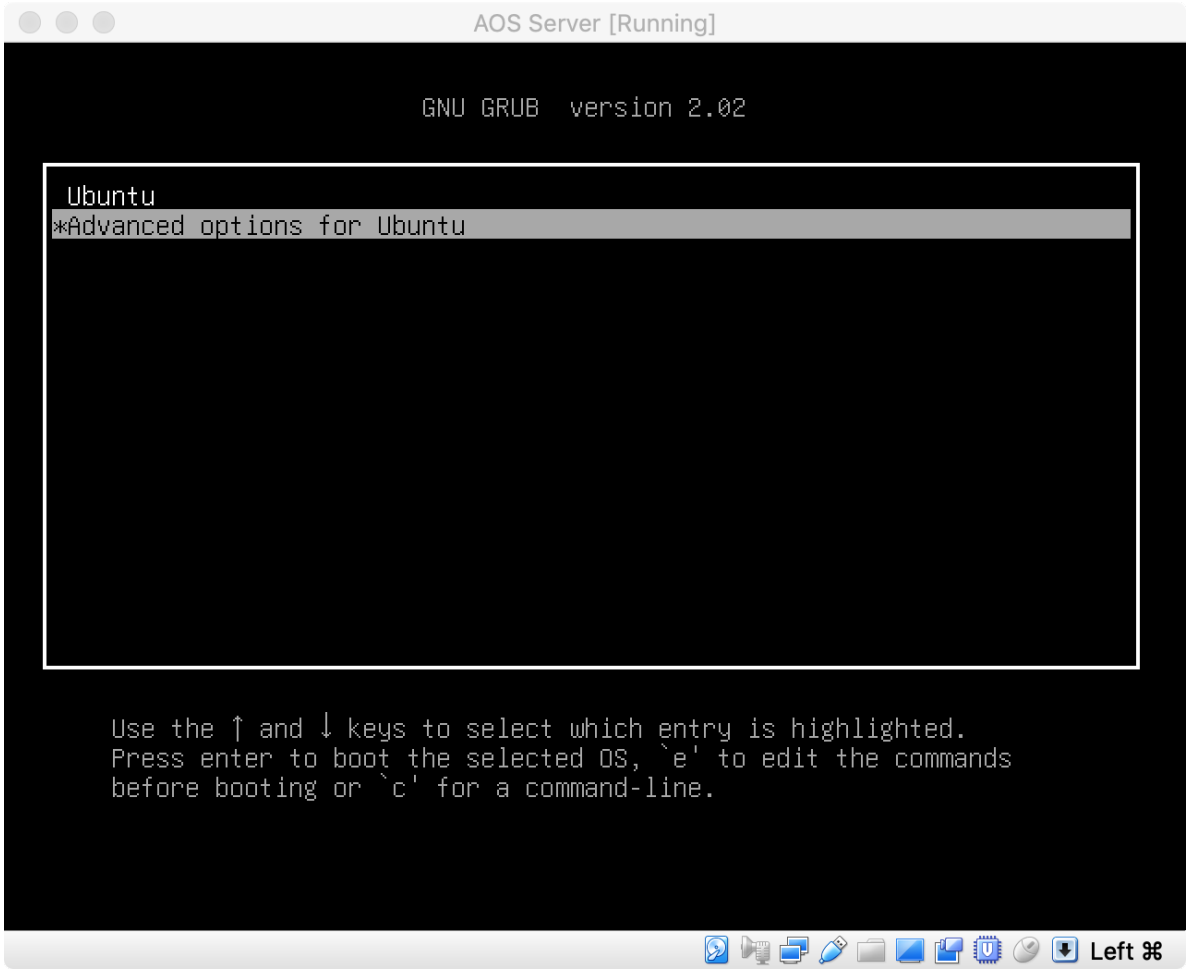
After services are restored (in a minute or two) the "liveness" telemetry alarm resets.

Reset Apstra Server VM Password

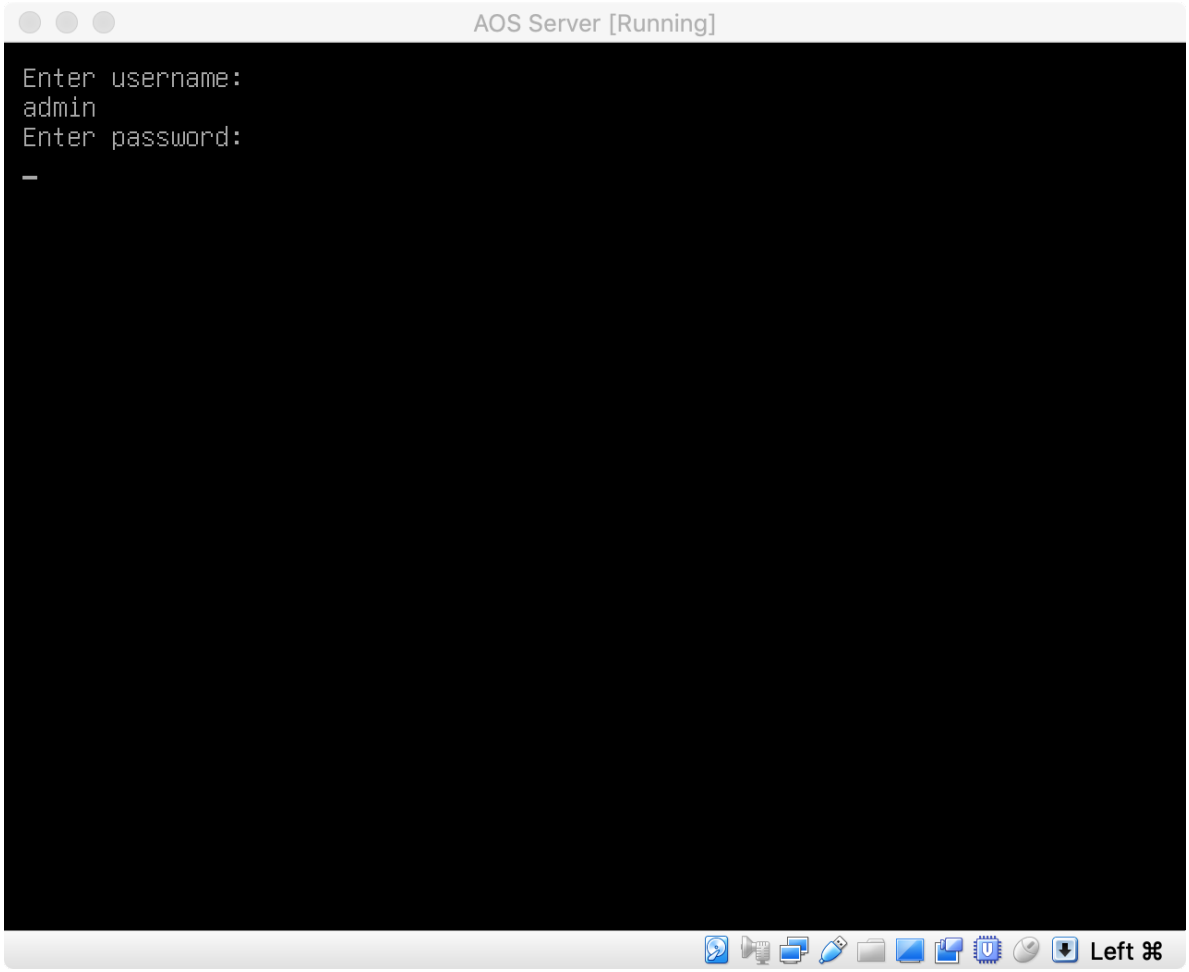
If you lose your **admin** password for the Apstra server VM, and *you still have console access to the Apstra server VM*, you can reset your password.

1. Attach to the Apstra server console and send a "reset" signal to the VM. To access the GRUB menu, immediately press the **esc** or **shift** key in the console on reboot.

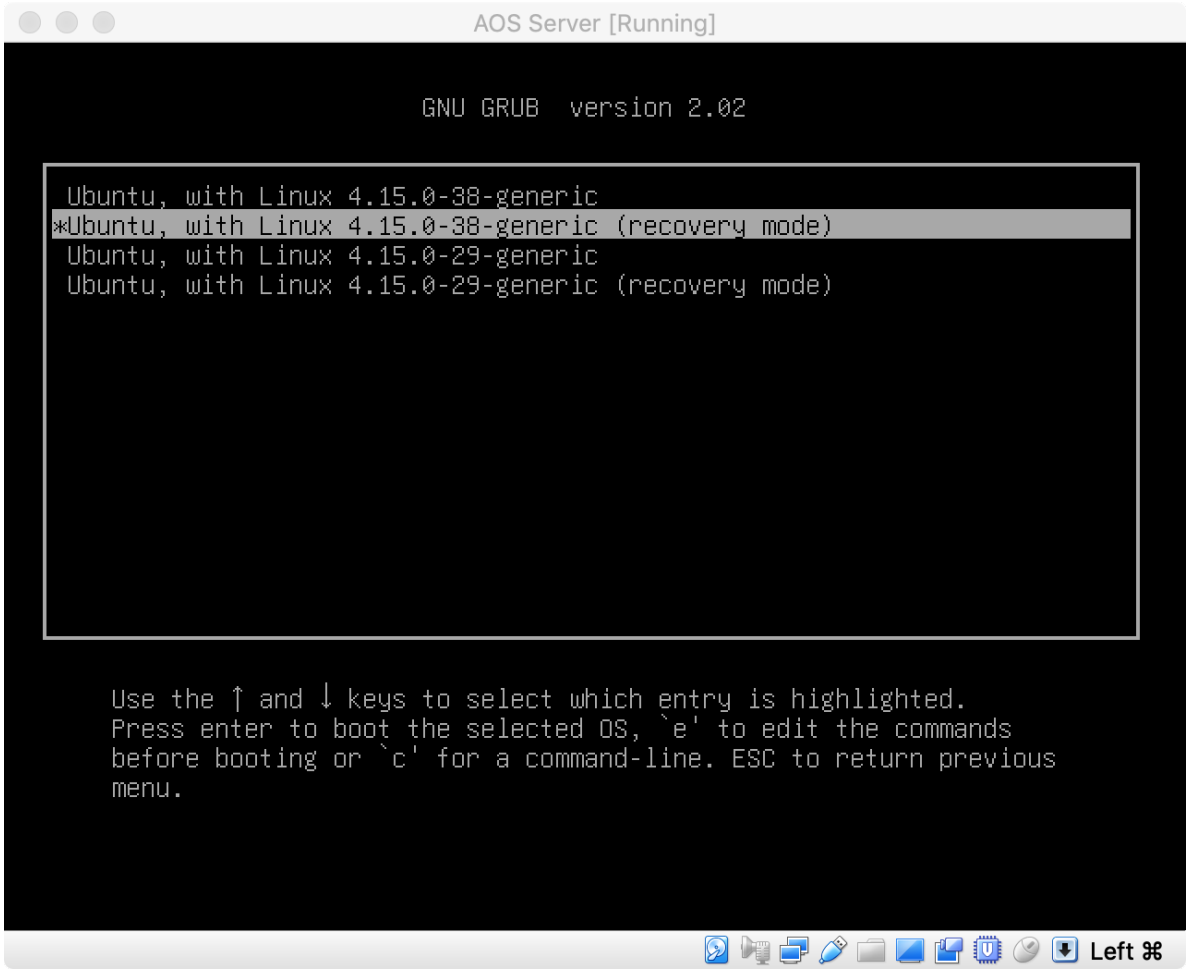
2. Select **Advanced options for Ubuntu**.



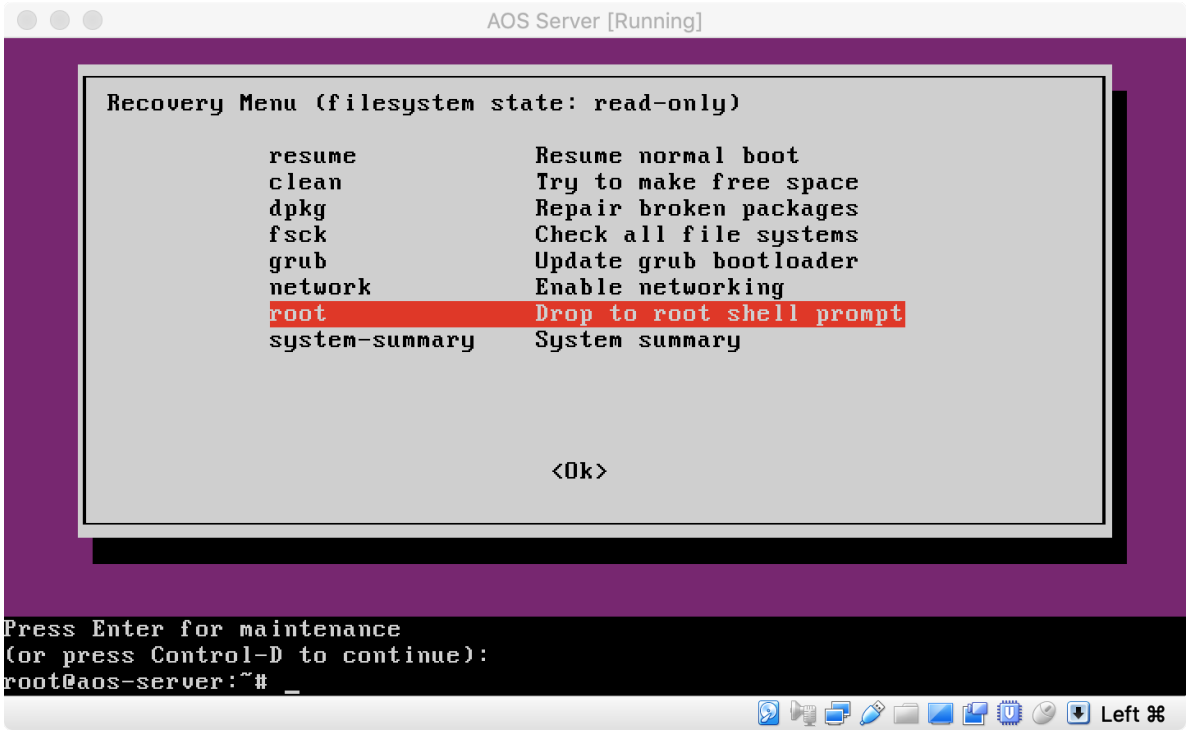
3. Enter username **admin** and password **apstra**.



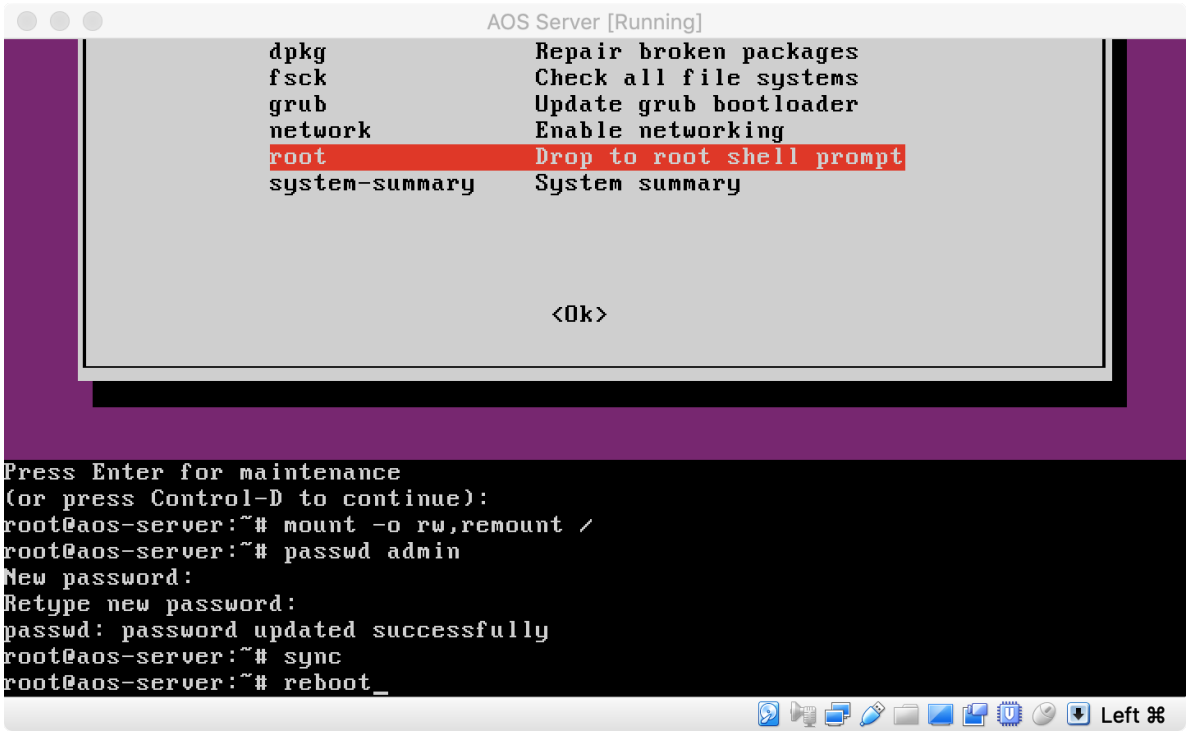
4. At the next GRUB menu, select the first **(recovery mode)** option.



5. From the **Recovery Menu**, select **root**, then press **Enter** to enter a root shell prompt.



6. At the root shell prompt run the command `mount -o rw,remount /`.
7. Run the command `passwd admin` to reset the default CLI password for **admin**.
8. Run the command `sync`.
9. Run the command `reboot` to reboot the Apstra server VM. (Your deployed fabric is not affected.)

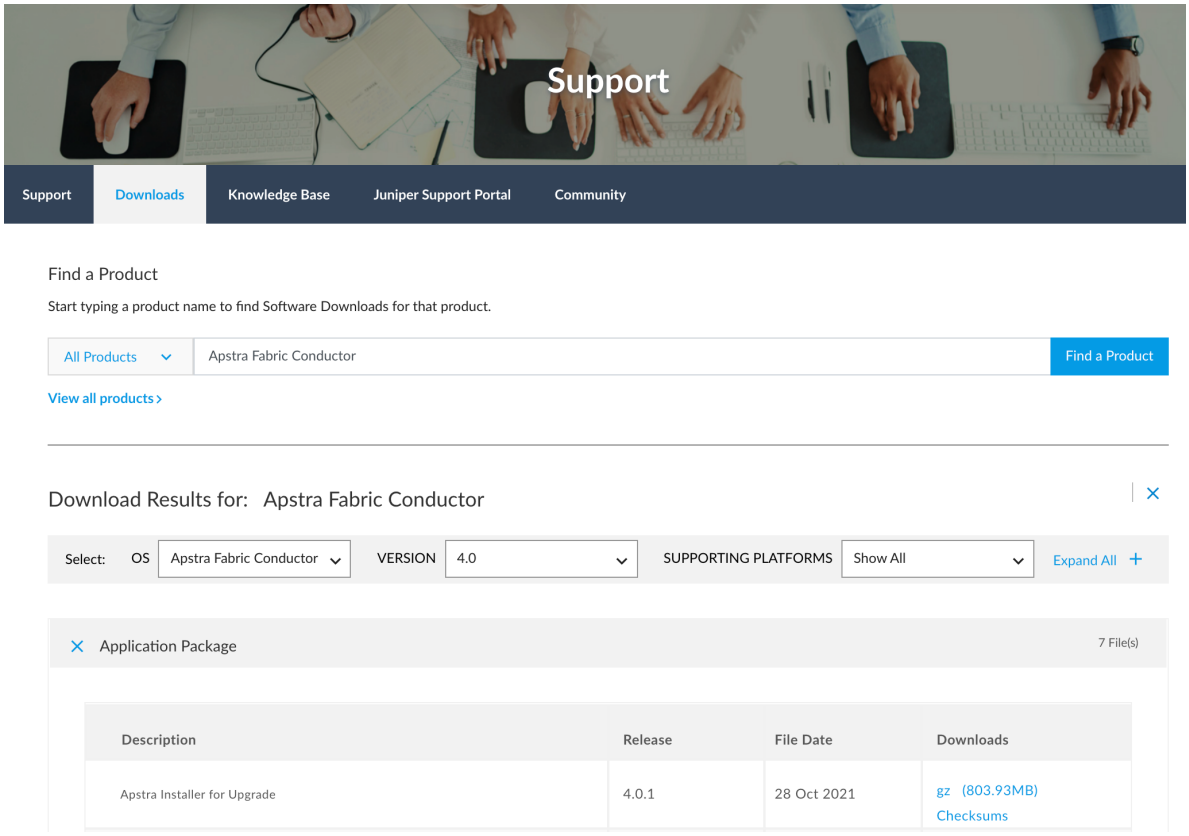


After reboot, you can log in to the Apstra server VM Linux CLI as user **admin** with the new password.

Reinstall Apstra Server

> **CAUTION**: Reinstalling the Apstra server removes ALL Apstra data from the Apstra server VM and reinstalls a fresh version. Use with care. This is mostly helpful for *proof of concepts* or demo installs. If you have problems that require you to reinstall the software, contact "Juniper Technical Support" on page 1374.

1. If you want to retain the Apstra database, "back it up" on page 1399 now.
2. Download the "Installer" .run file from Juniper Support Downloads.



3. Run the command `service aos stop` to stop Apstra service, if possible.

```
admin@aos-server:~$ sudo service aos stop
admin@aos-server:~$
```

4. Delete the Apstra server database.

```
admin@aos-server:~$ sudo rm -rf /var/lib/aos/db/*
admin@aos-server:~$
```

5. Remove the aos-compose package.

```
admin@aos-server:~$ sudo dpkg -r aos-compose
(Reading database ... 110457 files and directories currently installed.)
Removing aos-compose (3.3.0-660) ...
Processing triggers for ureadahead (0.100.0-21) ...
Processing triggers for systemd (237-3ubuntu10.41) ...
admin@aos-server:~$
```

6. Reinstall the Apstra software from the .run file.

```
admin@aos-server:~$ sudo bash aos_3.3.0-662.run
Verifying archive integrity... All good.
Uncompressing AOS installer  100%
610bd1ae69b7: Loading layer [==================================================>]  52.44MB/
52.44MB
87db235c4ff8: Loading layer [==================================================>]  211.3MB/
211.3MB
668b88b6cd3d: Loading layer [==================================================>]  117.3MB/
117.3MB
b1dd55ca7fd9: Loading layer [==================================================>]  20.63MB/
20.63MB
3f8ebc7f1fae: Loading layer [==================================================>]  4.608kB/
4.608kB
Loaded image: aos:3.3.0-662
AOS[2020-07-28_02:58:36]: Installing AOS 3.3.0-662 package
admin@aos-server:~$
```

You can now "restore" on page 1401 a database backup or build a new blueprint.

Apstra Database Overview

The Apstra server and related databases run in Docker containers. The database is stored in a single folder in the Apstra server at `/var/lib/aos/db`. You can copy the database between Apstra servers.

Source and Target database versions must be the same version. If versions are different, contact "Juniper Technical Support" on page 1374 for assistance before proceeding.

To ensure that device agents can 'call home' properly after database restoration, Source and Target must have the same IP address when starting the Apstra server, You can restore the software to a different IP address, but then you must reconfigure each device agent (`/mnt/flash/aos-config`, `/etc/aos/aos.conf`) to point to the new Apstra server IP address.

CAUTION: Any changes you make *within* the Apstra server are *not* stored in the backup.

Back up Apstra Database

You can back up the database while the Apstra server is running. Device/OS image information is not included in backups. When restoring a database, any device/OS image information is discarded.

Before backing up your database, disable any active IBA probes and wait until any database "write" tasks have completed.

1. Run the command `aos_backup` to back up the database. Backups are saved as dated snapshots (`/var/lib/aos/snapshot/<date>/aos.data.tar.gz`) in the Apstra server.

   If all IBA probes have been disabled and all "write" tasks have completed, the following message appears.

```
admin@aos-server:~$ sudo aos_backup
======================================================================
  Backup operation completed successfully.
======================================================================
New AOS snapshot: 2023-06-29_20-56-26
admin@aos-server:~$
```

If many IBA probes are enabled or if any other DB "write" tasks are in progress, they may not be included in the backup, and the following message appears.

```
admin@aos-server:~$ sudo aos_backup
Including secret keys from the backup
Include all sysdb files
========================================================================
Warning:
 Backup operation has been completed successfully. However AOS state
has been changed while this script was running, which means some
changes might not have been captured in the snapshot created in this
backup. You may choose to invoke aos_backup script again if you wish
to capture these changes right now instead of waiting for the next
backup operation.
========================================================================
New AOS snapshot: 2023-06-29_16-15-57
admin@aos-server:~$
```

If this message appears, disable your IBA probes and run the `aos_backup` command again.

2. Backups are stored on the Apstra server itself. If the server needs to be restored or if its disk image becomes corrupt, any backups/restores are lost along with the Apstra server. We recommend that you periodically move backups/restores off of the Apstra server to a secure location. Also, if you've scheduled cron jobs to periodically backup the database, make sure to rotate those files off of the Apstra server to keep the Apstra server VM disk from becoming full. Copy the contents of the snapshot directory to your backup infrastructure.

```
admin@aos-server:~$ sudo ls -lah /var/lib/aos/snapshot/
total 20K
drwx------ 5 root root 4.0K Jun 29 20:58 .
drwxr-xr-x 7 root root 4.0K Jun 29 02:43 ..
drwx------ 2 root root 4.0K Jun 29 02:43 2023-06-29_02-43-12
drwx------ 2 root root 4.0K Jun 29 20:56 2023-06-29_20-56-26
drwx------ 2 root root 4.0K Jun 29 20:58 2023-06-29_20-58-54
admin@aos-server:~$
```

Restore Apstra Database

CAUTION: Always restore a database from a new "backup" on page 1399, never from older backups or from the backup included in a show_tech.

When you restore a database, the worker VMs will go into a failed state. This problem also occurs when you restore a backup to another worker VM with the same IP address. To fix this issue, add the worker VMs again.

If you make changes after you back up the database, those changes aren't included in the restore. This could create differences between device configs and the Apstra environment. If this happens, you must perform a full config push, which is service-impacting.

Don't restore a database using the backup included in a show_tech. Juniper Support and Engineering use it for analysis. It doesn't include credentials, so it's not suitable for restoring your production environment.

NOTE: If you're restoring a backup to a new Apstra server that uses a different network interface for access (eth1 vs eth0 for example), you must update the `metadb` variable in the `[controller]` section of the `/etc/aos/aos.conf` configuration file, then restart the Apstra server.

1. Backups are saved in dated snapshot directories. Verify that you have a fresh backup in the `/var/lib/aos/snapshots/` directory.

```
admin@aos-server:~$ sudo ls -lah /var/lib/aos/snapshot/
total 12K
drwx------  3 root root 4.0K Dec 19 21:24 .
drwxr-xr-x 13 root root 4.0K Dec 19 21:24 ..
drwx------  3 root root 4.0K Dec 19 21:24 2023-12-19_21-24-10
```

2. The file name must be `aos.data.tar.gz`. Verify the file name, and correct it, if needed.

```
admin@aos-server:~$ sudo ls -lah /var/lib/aos/snapshot/2023-12-19_21-24-10
total 125M
drwx------ 3 root root 4.0K Dec 19 21:24 .
drwx------ 3 root root 4.0K Dec 19 21:24 ..
-rw------- 1 root root 125M Dec 19 21:24 aos.data.tar.gz
-rwxr-xr-x 1 root root 2.6K Dec 19 21:24 aos_restore
```

```
-rw------- 1 root root    1 Dec 19 21:24 comment.txt
drwx------ 2 root root 4.0K Dec 19 21:24 metadata
admin@aos-server:~$
```

3. Run the `aos_restore` command as illustrated below. The restore process first backs up the current database.

```
admin@aos-server:~$ sudo bash /var/lib/aos/snapshot/2023-12-19_21-24-10/aos_restore
Including secret keys from the backup
Include all sysdb files
New AOS snapshot: 2023-12-19_21-49-08
[+] Running 5/5
   Container aos_controller_1
Stopped                                                                  11.1s
   Container aos_auth_1
Stopped                                                                  11.0s
   Container aos_metadb_1
Stopped                                                                  11.0s
   Container aos_sysdb_1
Stopped                                                                  11.0s
   Container aos_nginx_1
Stopped                                                                   0.7s
(Reading database ... 83704 files and directories currently installed.)
Removing aos-compose (4.2.0-236) ...
tar: Removing leading `/' from member names
/var/lib/aos/db/
/var/lib/aos/db/_Main-00000000656e68c2-000bc9e4-log
/var/lib/aos/db/_Auth-00000000656e68be-000eacab-log-valid
/var/lib/aos/db/_Auth-000000006553e3a7-0000be2f-log-valid
/var/lib/aos/db/_Central-00000000656e68b4-0002ce01-checkpoint
/var/lib/aos/db/_AosController-00000000656e68b9-000bbbf0-log
/var/lib/aos/db/_Central-000000006553e3a5-00064668-log-valid
/var/lib/aos/db/_Main-000000006553e3aa-00052829-log
/var/lib/aos/db/_Auth-000000006553e3a7-0000be2f-checkpoint
/var/lib/aos/db/_Main-00000000656e68c2-000bc9e4-checkpoint
/var/lib/aos/db/_Central-00000000656e68b4-0002ce01-log
/var/lib/aos/db/_AosSysdb-00000000656e68aa-0000ee5d-log
/var/lib/aos/db/_Auth-00000000656e68be-000eacab-log
/var/lib/aos/db/_Main-000000006553e3aa-00052829-checkpoint-valid
/var/lib/aos/db/_AosController-00000000656e68b9-000bbbf0-checkpoint
/var/lib/aos/db/.devpi/
/var/lib/aos/db/.devpi/server/
```

```
/var/lib/aos/db/.devpi/server/.event_serial
/var/lib/aos/db/.devpi/server/.serverversion
/var/lib/aos/db/.devpi/server/.sqlite
/var/lib/aos/db/.devpi/server/.nodeinfo
/var/lib/aos/db/_AosSysdb-00000000656e68aa-0000ee5d-log-valid
/var/lib/aos/db/_Central-00000000656e68b4-0002ce01-log-valid
/var/lib/aos/db/_Central-000000006553e3a5-00064668-checkpoint-valid
/var/lib/aos/db/_Main-000000006553e3aa-00052829-checkpoint
/var/lib/aos/db/_AosSysdb-00000000656e68aa-0000ee5d-checkpoint
/var/lib/aos/db/_Metadb-00000000656e68a9-000c719b-log
/var/lib/aos/db/_Metadb-00000000656e68a9-000c719b-log-valid
/var/lib/aos/db/_AosAuth-00000000656e68a9-0007cb45-log-valid
/var/lib/aos/db/_Auth-000000006553e3a7-0000be2f-log
/var/lib/aos/db/_Main-000000006553e3aa-00052829-log-valid
/var/lib/aos/db/_AosAuth-00000000656e68a9-0007cb45-checkpoint
/var/lib/aos/db/_Central-00000000656e68b4-0002ce01-checkpoint-valid
/var/lib/aos/db/_Central-000000006553e3a5-00064668-log
/var/lib/aos/db/_Auth-000000006553e3a7-0000be2f-checkpoint-valid
/var/lib/aos/db/_Metadb-00000000656e68a9-000c719b-checkpoint
/var/lib/aos/db/_Main-00000000656e68c2-000bc9e4-checkpoint-valid
/var/lib/aos/db/_AosAuth-00000000656e68a9-0007cb45-log
/var/lib/aos/db/_AosController-00000000656e68b9-000bbbf0-checkpoint-valid
/var/lib/aos/db/_Auth-00000000656e68be-000eacab-checkpoint
/var/lib/aos/db/_Metadb-00000000656e68a9-000c719b-checkpoint-valid
/var/lib/aos/db/_Auth-00000000656e68be-000eacab-checkpoint-valid
/var/lib/aos/db/_AosController-00000000656e68b9-000bbbf0-log-valid
/var/lib/aos/db/_AosSysdb-00000000656e68aa-0000ee5d-checkpoint-valid
/var/lib/aos/db/_AosAuth-00000000656e68a9-0007cb45-checkpoint-valid
/var/lib/aos/db/_Central-000000006553e3a5-00064668-checkpoint
/var/lib/aos/db/_Main-00000000656e68c2-000bc9e4-log-valid
/var/lib/aos/anomaly/
/var/lib/aos/anomaly/_Anomaly-00000000650916f3-000e3d9b-checkpoint
/var/lib/aos/anomaly/_Anomaly-00000000656e68bf-0006052e-checkpoint
/var/lib/aos/anomaly/_Anomaly-00000000650916f3-000e3d9b-checkpoint-valid
/var/lib/aos/anomaly/_Anomaly-000000006553e3a7-0004794b-checkpoint
/var/lib/aos/anomaly/_Anomaly-000000006553e3a7-0004794b-log-valid
/var/lib/aos/anomaly/_Anomaly-00000000656e68bf-0006052e-checkpoint-valid
/var/lib/aos/anomaly/_Anomaly-00000000650916f3-000e3d9b-log
/var/lib/aos/anomaly/_Anomaly-00000000656e68bf-0006052e-log-valid
/var/lib/aos/anomaly/_Anomaly-000000006553e3a7-0004794b-checkpoint-valid
/var/lib/aos/anomaly/_Anomaly-00000000656e68bf-0006052e-log
/var/lib/aos/anomaly/_Anomaly-000000006553e3a7-0004794b-log
/var/lib/aos/anomaly/_Anomaly-00000000650916f3-000e3d9b-log-valid
```

```
/etc/aos/aos.conf
/etc/aos-img-chksum/
/etc/aos-img-chksum/checksums
/etc/aos-img-chksum/key.pub
/etc/aos-img-chksum/checksums.signed
/opt/aos/aos-compose.deb
/opt/aos/frontend_images/
/opt/aos/frontend_images/jinja_docs.zip
/opt/aos/frontend_images/aos-web-ui.zip
/opt/aos/frontend_images/sdt_docs.zip
/etc/aos/version
/etc/aos-auth/secret_key
/etc/aos-credential/secret_key
Selecting previously unselected package aos-compose.
(Reading database ... 83670 files and directories currently installed.)
Preparing to unpack /opt/aos/aos-compose.deb ...
Unpacking aos-compose (4.2.0-236) ...
Setting up aos-compose (4.2.0-236) ...
Verifying checksums for docker images...
Signature Verified Successfully
Verified.
[+] Running 5/5
   Container aos_auth_1
Started                                                    0.5s
   Container aos_metadb_1
Started                                                    0.7s
   Container aos_sysdb_1
Started                                                    0.4s
   Container aos_controller_1
Started                                                    0.5s
   Container aos_nginx_1
Started                                                    0.4s
admin@aos-server:~$
```

4. When the database has been restored and migrated to a new server, the entire system state has been copied from the backed up installation to the new target. Run the command `service aos status` to validate the restoration.

```
admin@aos-server:~$ sudo service aos status
● aos.service - LSB: Start AOS management system
     Loaded: loaded (/etc/init.d/aos; generated)
     Active: active (exited) since Tue 2023-12-05 00:02:46 UTC; 2 weeks 0 days ago
```

```
       Docs: man:systemd-sysv-generator(8)
        CPU: 541ms

Dec 05 00:02:45 aos-server aos[1112]: Container aos_nginx_1  Starting
Dec 05 00:02:45 aos-server aos[1112]: Container aos_metadb_1  Starting
Dec 05 00:02:45 aos-server aos[1112]: Container aos_auth_1  Starting
Dec 05 00:02:45 aos-server aos[1112]: Container aos_sysdb_1  Starting
Dec 05 00:02:46 aos-server aos[1112]: Container aos_auth_1  Started
Dec 05 00:02:46 aos-server aos[1112]: Container aos_sysdb_1  Started
Dec 05 00:02:46 aos-server aos[1112]: Container aos_metadb_1  Started
Dec 05 00:02:46 aos-server aos[1112]: Container aos_controller_1  Started
Dec 05 00:02:46 aos-server aos[1112]: Container aos_nginx_1  Started
Dec 05 00:02:46 aos-server systemd[1]: Started LSB: Start AOS management system.
admin@aos-server:~$
```

5. The database is stored on the Apstra server itself. If the server needs to be restored or if its disk image becomes corrupt, any backups/restores are lost along with the Apstra server. We recommend that you periodically move backups/restores off of the Apstra server to a secure location. Also, if you've scheduled cron jobs to periodically backup the database, make sure to rotate those files off of the Apstra server to keep the Apstra server VM disk from becoming full. Copy the contents of the snapshot directory to your backup infrastructure.

```
admin@aos-server:~$ sudo ls -lah /var/lib/aos/snapshot/
total 32K
drwx------  8 root root 4.0K Jun 29 19:31 .
drwxr-xr-x 13 root root 4.0K Jun 29 19:32 ..
drwx------  3 root root 4.0K Jun 29 15:44 2023-12-19_21-24-10
drwx------  3 root root 4.0K Jun 29 15:45 2023-12-19_15-45-37
drwx------  3 root root 4.0K Jun 29 16:21 2023-12-19_16-21-36
drwx------  3 root root 4.0K Jun 29 18:11 2023-12-19_18-11-34
drwx------  3 root root 4.0K Jun 29 18:40 2023-12-19_18-40-03
drwx------  3 root root 4.0K Jun 29 19:31 2023-12-19_19-31-43
admin@aos-server:~$
```

RELATED DOCUMENTATION

Back up Apstra Database | 1399

Reset Apstra Database

The commands below delete *all* data on the Apstra server to a fresh state.

1. Run the command `service aos stop`.
2. Run the command `rm -rf /var/lib/aos/db/*`.
3. Run the command `service aos start`.

```
admin@aos-server:~$ sudo service aos stop
admin@aos-server:~$ sudo rm -rf /var/lib/aos/db/*
admin@aos-server:~$ sudo service aos start
admin@aos-server:~$
```

Migrate Apstra Database

CAUTION: If you bring up a new Apstra server with the same IP address as your old Apstra server without any configuration, when the device agents re-register with the new Apstra server they will revert to an unconfigured "Quarantined" state. You must isolate the new Apstra server from the network while you change its IP address, restore the database and restart the Apstra server.

If you want to **maintain the same IP address** on the new Apstra server, then bring up a new Apstra server VM (with the same version as the original Apstra server) with a temporary IP address. After migrating an `aos_backup` to the new Apstra server, the original Apstra server is shut down and the IP address is changed to the original IP address on the new server. We recommend this process if you're using onbox device system agents.

If you want to **use a new IP address** on the new Apstra server, you must manually reconfigure the `aos.conf` file for each onbox device system agent. This is not required for offbox device system agents.

To migrate an active instance from one server to another:

1. Run the command `sudo aos_backup` to back up the original Apstra server.

```
admin@aos-server:~$ sudo aos_backup
======================================================================
  Backup operation completed successfully.
```

```
=============================================================
New AOS snapshot: 2023-07-27_22-49-34
admin@aos-server:~$
```

2. Copy the snapshot to the new server using a temporary IP address on the new Apstra server.
3. Compress and move the snapshot directory to the new Apstra server. This example uses the `scp` command to copy the file to the new Apstra server using a different IP address.

```
admin@aos-server:~$ sudo tar zcvf aos_backup.tar.gz /var/lib/aos/snapshot/2023-07-27_22-49-3
2023-07-27_22-49-34/
2023-07-27_22-49-34/comment.txt
2023-07-27_22-49-34/aos_restore
2023-07-27_22-49-34/aos.data.tar.gz
admin@aos-server:~$ sudo chown admin:admin aos_backup.tar.gz
admin@aos-server:~$ scp aos_backup.tar.gz admin@172.20.203.4:
Apstra Operating System (AOS) Virtual Appliance

Password:
aos_backup.tar.gz                                   100%   20MB 140.9MB/s   00:00
admin@aos-server:~$
```

4. After the snapshot has been removed from the old Apstra server, stop service (or completely shut down the Apstra server VM) to disconnect the old Apstra server.

```
admin@aos-server:~$ sudo service aos stop
admin@aos-server:~$
```

5. If you want to use the same IP address, you must manually reconfigure the eth0 interface on the new Apstra server to the IP address of the old Apstra server. For more information, see the Configuration section of the Juniper Apstra Installation and Upgrade guide.
6. On the new Apstra server, uncompress the tar.gz file.

```
admin@aos-server:~$ tar zxvf aos_backup.tar.gz
2023-07-27_22-49-34/
2023-07-27_22-49-34/comment.txt
2023-07-27_22-49-34/aos_restore
2023-07-27_22-49-34/aos.data.tar.gz
admin@aos-server:~$
```

7. Run the command `aos_restore` to restore the database on the new Apstra server. This command automatically starts the service after restoring the database.

```
admin@aos-server:~$ cd 2023-07-27_22-49-34
admin@aos-server:~/2023-07-27_22-49-34$ sudo bash aos_restore
[sudo] password for admin:
======================================================================
  Backup operation completed successfully.
======================================================================
New AOS snapshot: 2023-07-27_23-07-13
Stopping aos_sysdb_1      ... done
Stopping aos_auth_1       ... done
Stopping aos_controller_1 ... done
Stopping aos_nginx_1      ... done
Stopping aos_metadb_1     ... done
(Reading database ... 110457 files and directories currently installed.)
Removing aos-compose (3.3.0-658) ...
Processing triggers for ureadahead (0.100.0-21) ...
Processing triggers for systemd (237-3ubuntu10.41) ...
tar: Removing leading `/' from member names
/etc/aos/aos.conf
/etc/aos-credential/secret_key
/var/lib/aos/db/
/var/lib/aos/db/_AosController-000000005f1f376f-0003998b-checkpoint
/var/lib/aos/db/_AosSysdb-000000005f1f376d-000a90ba-log-valid
/var/lib/aos/db/_Main-000000005f1f376f-000569a8-checkpoint
/var/lib/aos/db/_Central-000000005f1f376e-000da3de-checkpoint-valid
/var/lib/aos/db/_Central-000000005f1f376e-000da3de-log
/var/lib/aos/db/_Main-000000005f1f376f-000569a8-log-valid
/var/lib/aos/db/_AosAuth-000000005f1f376d-000a40ff-log
/var/lib/aos/db/_Auth-000000005f1f376e-000f2d35-log-valid
/var/lib/aos/db/_Auth-000000005f1f376e-000f2d35-checkpoint-valid
/var/lib/aos/db/_Metadb-000000005f1f376d-000cb9a9-checkpoint-valid
/var/lib/aos/db/_Central-000000005f1f376e-000da3de-checkpoint
/var/lib/aos/db/_Metadb-000000005f1f376d-000cb9a9-log
/var/lib/aos/db/_Credential-000000005f1f376e-000d740e-log-valid
/var/lib/aos/db/_AosAuth-000000005f1f376d-000a40ff-checkpoint-valid
/var/lib/aos/db/_Metadb-000000005f1f376d-000cb9a9-checkpoint
/var/lib/aos/db/_Main-000000005f1f376f-000569a8-log
/var/lib/aos/db/_AosSysdb-000000005f1f376d-000a90ba-checkpoint-valid
/var/lib/aos/db/_AosController-000000005f1f376f-0003998b-log-valid
/var/lib/aos/db/_Auth-000000005f1f376e-000f2d35-checkpoint
```

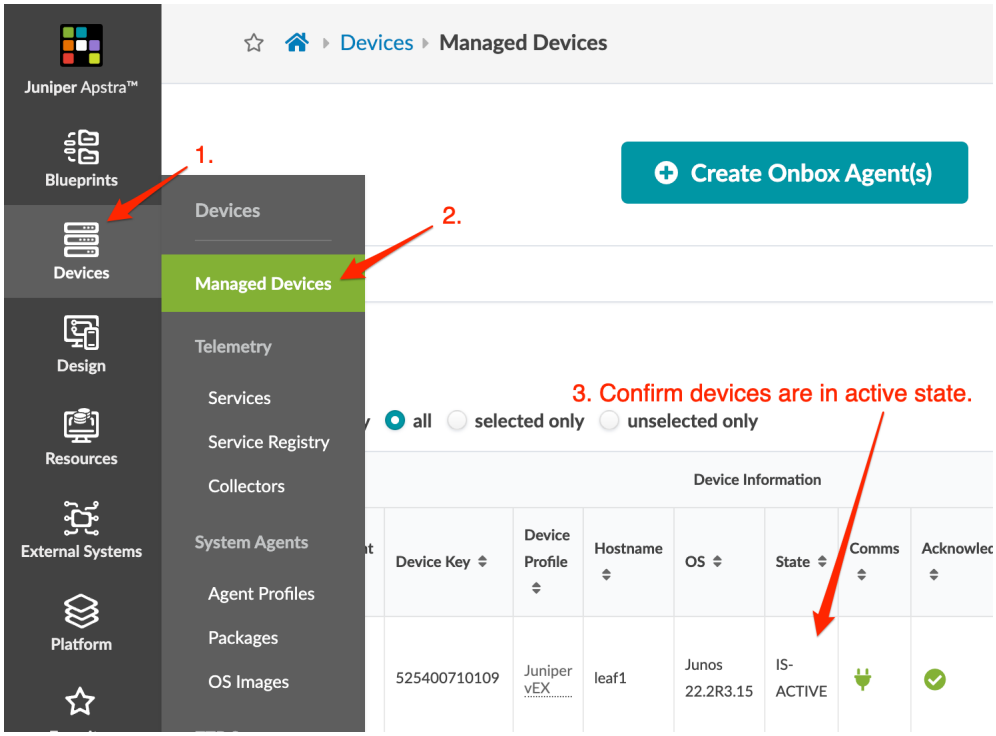
```
/var/lib/aos/db/_AosSysdb-000000005f1f376d-000a90ba-log
/var/lib/aos/db/_AosSysdb-000000005f1f376d-000a90ba-checkpoint
/var/lib/aos/db/_AosAuth-000000005f1f376d-000a40ff-log-valid
/var/lib/aos/db/blueprint_backups/
/var/lib/aos/db/blueprint_backups/6b90ccfd-a1e0-4473-83e7-d62bce24635f/
/var/lib/aos/db/blueprint_backups/6b90ccfd-a1e0-4473-83e7-d62bce24635f/47/
/var/lib/aos/db/blueprint_backups/6b90ccfd-a1e0-4473-83e7-d62bce24635f/47/graph.json.zip
/var/lib/aos/db/blueprint_backups/6b90ccfd-a1e0-4473-83e7-d62bce24635f/47/graph.md5sum
/var/lib/aos/db/_Central-000000005f1f376e-000da3de-log-valid
/var/lib/aos/db/_Auth-000000005f1f376e-000f2d35-log
/var/lib/aos/db/_Credential-000000005f1f376e-000d740e-log
/var/lib/aos/db/_Credential-000000005f1f376e-000d740e-checkpoint
/var/lib/aos/db/_Credential-000000005f1f376e-000d740e-checkpoint-valid
/var/lib/aos/db/.devpi/
/var/lib/aos/db/.devpi/server/
/var/lib/aos/db/.devpi/server/.nodeinfo
/var/lib/aos/db/.devpi/server/.secret
/var/lib/aos/db/.devpi/server/.sqlite
/var/lib/aos/db/.devpi/server/.serverversion
/var/lib/aos/db/.devpi/server/.event_serial
/var/lib/aos/db/_AosController-000000005f1f376f-0003998b-log
/var/lib/aos/db/_Main-000000005f1f376f-000569a8-checkpoint-valid
/var/lib/aos/db/_Metadb-000000005f1f376d-000cb9a9-log-valid
/var/lib/aos/db/_AosAuth-000000005f1f376d-000a40ff-checkpoint
/var/lib/aos/db/_AosController-000000005f1f376f-0003998b-checkpoint-valid
/var/lib/aos/anomaly/
/var/lib/aos/anomaly/_Anomaly-000000005f1f36a4-000aaa68-checkpoint-valid
/var/lib/aos/anomaly/_Anomaly-000000005f1f331b-0000e8eb-checkpoint
/var/lib/aos/anomaly/_Anomaly-000000005f1f376f-00002176-checkpoint
/var/lib/aos/anomaly/_Anomaly-000000005f1f376f-00002176-log
/var/lib/aos/anomaly/_Anomaly-000000005f1f331b-0000e8eb-log
/var/lib/aos/anomaly/_Anomaly-000000005f1f2abc-0000a867-log
/var/lib/aos/anomaly/_Anomaly-000000005f1f331b-0000e8eb-checkpoint-valid
/var/lib/aos/anomaly/_Anomaly-000000005f1f2abc-0000a867-checkpoint
/var/lib/aos/anomaly/_Anomaly-000000005f1f36a4-000aaa68-checkpoint
/var/lib/aos/anomaly/_Anomaly-000000005f1f376f-00002176-log-valid
/var/lib/aos/anomaly/_Anomaly-000000005f1f36a4-000aaa68-log
/var/lib/aos/anomaly/_Anomaly-000000005f1f331b-0000e8eb-log-valid
/var/lib/aos/anomaly/_Anomaly-000000005f1f2abc-0000a867-checkpoint-valid
/var/lib/aos/anomaly/_Anomaly-000000005f1f2abc-0000a867-log-valid
/var/lib/aos/anomaly/_Anomaly-000000005f1f36a4-000aaa68-log-valid
/var/lib/aos/anomaly/_Anomaly-000000005f1f376f-00002176-checkpoint-valid
/opt/aos/aos-compose.deb
```

```
/opt/aos/frontend_images/
/opt/aos/frontend_images/aos-web-ui.zip
Selecting previously unselected package aos-compose.
(Reading database ... 110440 files and directories currently installed.)
Preparing to unpack /opt/aos/aos-compose.deb ...
Unpacking aos-compose (3.3.0-658) ...
Setting up aos-compose (3.3.0-658) ...
Processing triggers for ureadahead (0.100.0-21) ...
Processing triggers for systemd (237-3ubuntu10.41) ...
Starting aos_nginx_1      ... done
Starting aos_sysdb_1      ... done
Starting aos_controller_1 ... done
Starting aos_metadb_1     ... done
Starting aos_auth_1       ... done
admin@aos-server:~/2023-07-27_22-49-34$
```

8. Run the command `service aos status` and verify that the Apstra server is running.

```
admin@aos-server:~/2023-07-27_22-49-34$ service aos status
* aos.service - LSB: Start AOS management system
   Loaded: loaded (/etc/init.d/aos; generated)
   Active: active (exited) since Thu 2023-07-27 20:23:09 UTC; 2h 45min ago
     Docs: man:systemd-sysv-generator(8)
    Tasks: 0 (limit: 4915)
   CGroup: /aos.service
admin@aos-server:~/2023-07-27_22-49-34$
```

9. From the Apstra GUI, from the left navigation menu, navigate to **Devices > Managed Devices** to verify that your devices are online in the "Active" state.



Replace SSL Certificate on Apstra Server with Signed One

When you boot up the Apstra server for the first time, a unique self-signed certificate is automatically generated and stored on the Apstra server at `/etc/aos/nginx.conf.d` (`nginx.crt` is the public key for the webserver and `nginx.key` is the private key.) The certificate is used for encrypting the Apstra server and REST API. It's not for any internal device-server connectivity. Since the HTTPS certificate is not retained when you back up the system, you must manually back up the `etc/aos` folder. We recommend replacing the default SSL certificate. Web server certificate management is the responsibility of the end user. Juniper support is best effort only.

1. Back up the existing OpenSSL keys.

```
admin@aos-server:/$ sudo -s
[sudo] password for admin:

root@aos-server:/# cd /etc/aos/nginx.conf.d
```

```
root@aos-server:/etc/aos/nginx.conf.d# cp nginx.crt nginx.crt.old
root@aos-server:/etc/aos/nginx.conf.d# cp nginx.key nginx.key.old
```

2. Create a new OpenSSL private key with the built-in openssl command.

```
root@aos-server:/etc/aos/nginx.conf.d# openssl genrsa -out nginx.key 2048
Generating RSA private key, 2048 bit long modulus
.............+++
......+++
e is 65537 (0x10001)
```

> **CAUTION**: Don't modify `nginx.crt` or `nginx.key` filenames. They're referred to in `nginx.conf`. As part of subsequent service upgrades, these files could be replaced, so the filenames must be predictable.
>
> Also, don't change configuration in `nginx.conf`, as this file may be replaced during Apstra server upgrade, and any changes you make would be discarded.

3. Create a certificate signing request. If you want to create a signed SSL certificate with a Subjective Alternative Name (SAN) for your Apstra server HTTPS service, you must manually create an OpenSSL template. For details, see Juniper Support Knowledge Base article KB37299.

> **CAUTION**: If you have created custom OpenSSL configuration files for advanced certificate requests, don't leave them in the Nginx configuration folder. On startup, Nginx will attempt to load them (*.conf), causing a service failure.

```
root@aos-server:/etc/aos/nginx.conf.d# openssl req -new -sha256 -key nginx.key -out nginx.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:California
Locality Name (eg, city) []:Menlo Park
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Apstra, Inc
Organizational Unit Name (eg, section) []:
```

```
Common Name (e.g. server FQDN or YOUR name) []:aos-server.apstra.com
Email Address []:support@apstra.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

4. Submit your Certificate Signing Request (nginx.csr) to your Certificate Authority. The required steps are outside the scope of this document; CA instructions differ per implementation. Any valid SSL certificate will work. The example below is for self-signing the certificate.

```
root@aos-server:/etc/aos/nginx.conf.d# openssl req -x509 -sha256 -days 3650 -key nginx.key -
in nginx.csr -out nginx.crt
root@aos-server:/etc/aos/nginx.conf.d#
```

5. Verify that the SSL certificates match: private key, public key, and CSR.

```
root@aos-server:/etc/aos/nginx.conf.d# openssl rsa -noout -modulus -in nginx.key | openssl md5
(stdin)= 60ac4532a708c98d70fee0dbcaab1e75

root@aos-server:/etc/aos/nginx.conf.d# openssl req -noout -modulus -in nginx.csr | openssl md5
(stdin)= 60ac4532a708c98d70fee0dbcaab1e75

root@aos-server:/etc/aos/nginx.conf.d# openssl x509 -noout -modulus -in nginx.crt | openssl
md5
(stdin)= 60ac4532a708c98d70fee0dbcaab1e75
```

6. To load the new certificate, restart the nginx container.

```
root@aos-server:/etc/aos/nginx.conf.d# docker restart aos_nginx_1
aos_nginx_1
root@aos-server:/etc/aos/nginx.conf.d
```

7. Confirm that the new certificate is in your web browser and that the new certificate common name matches 'aos-server.apstra.com'.

Replace SSL Certificate on Apstra Server with Self-Signed One

When you boot up the Apstra server for the first time, a unique self-signed certificate is automatically generated and stored on the Apstra server at `/etc/aos/nginx.conf.d` (`nginx.crt` is the public key for the webserver and `nginx.key` is the private key.) The certificate is used for encrypting the Apstra server and REST API. It's not for any internal device-server connectivity. Since the HTTPS certificate is not retained when you back up the system, you must manually back up the `etc/aos` folder. We support and recommend replacing the default SSL certificate.

1. Back up the existing OpenSSL keys.

```
admin@aos-server:/$ sudo -s
[sudo] password for admin:

root@aos-server:/# cd /etc/aos/nginx.conf.d
root@aos-server:/etc/aos/nginx.conf.d# cp nginx.crt nginx.crt.old
root@aos-server:/etc/aos/nginx.conf.d# cp nginx.key nginx.key.old
```

2. If a Random Number Generator seed file .rnd doesn't exist in `/home/admin`, create one.

```
root@aos-server:~# touch /home/admin/.rnd
root@aos-server:~#
```

3. Generate a new OpenSSL private key and self-signed certificate.

```
root@aos-server:/etc/aos/nginx.conf.d# openssl req -newkey rsa:2048 -nodes -keyout nginx.key -
x509 -days 824 -out nginx.crt -addext extendedKeyUsage=serverAuth -addext
subjectAltName=DNS:apstra.com
Generating a RSA private key
.....................................+++++
.....................................................................................+++++
writing new private key to 'nginx.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
```

```
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:California
Locality Name (eg, city) []:Menlo Park
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Apstra, Inc
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:aos-server.apstra.com
Email Address []:support@apstra.com
root@aos-server:/etc/aos/nginx.conf.d#
```

4. To load the new certificate, restart the nginx container.

```
root@aos-server:/etc/aos/nginx.conf.d# docker restart aos_nginx_1
aos_nginx_1
root@aos-server:/etc/aos/nginx.conf.d
```

Change Apstra Server Hostname

You have the option of changing the default Apstra server hostname (aos-server).

1. SSH into the Apstra server as user **admin** (ssh admin@<apstra-server-ip> where <apstra-server-ip> is the IP address of the Apstra server.)
2. As root user, run the command `aos_hostname <hostname>` where `<hostname>` is the new hostname.

```
admin@aos-server:~$ sudo aos_hostname new-aos-server
[sudo] password for admin:
admin@aos-server:~$
```

The new hostname will display the next time you log in.

NOTE: Do not use `/etc/hostname` to change the Apstra server hostname. With this method, if you configure syslog to be forwarded to an external server, the default hostname will be entered into the log instead of the new one.

FIPS 14-2 Support

SUMMARY

FIPS 14-2 Level 1 ensures that Apstra uses approved cryptographic algorithms, providing basic security without requiring advanced physical protections.

Overview of FIPS 140-2

FIPS 140-2 is a U.S. government standard that specifies security requirements for cryptographic modules. Implementing Level 1 ensures that Apstra uses approved cryptographic algorithms, providing basic security without requiring advanced physical protections.

Why FIPS 140-2 is Needed

Apstra handles sensitive data that must comply with regulatory standards for some users. Implementing FIPS 140-2 Level 1 ensures that our cryptographic operations meet these security requirements, giving users confidence in our data protection.

Default FIPS Mode

By default, Apstra operates with FIPS mode disabled to minimize disruption. You can manually enable FIPS mode using the CLI.

Enabling and Managing FIPS 140-2

You can manage FIPS mode with the following commands:

- `aos_fips enable` – Enables FIPS mode on the Apstra VM.
- `aos_fips disable` – Disables FIPS mode on the Apstra VM.
- `aos_fips status` – Reports the status of FIPS mode, checking configurations such as Apstra config, SSH configuration, NGINX config, Docker containers, and OpenSSL settings.

Example Command:

To check the FIPS status, run:

```
aos_fips status
```

Sample output indicates whether FIPS mode is activated and which components are FIPS-enabled.

```
admin@aos-server:~$ aos_fips status
Checking aos.config...
 FIPS mode is activated
Checking Docker Compose settings...
 effective config is FIPS enabled
Checking OpenSSL effective config on host...
 effective config is FIPS enabled
Checking Docker containers...
 [5109bea58085][aos-uninstall-onbox-30e49774-10.28.212.15-j2] FIPS activated
 [2295b59992c6][aos-uninstall-onbox-54b69f7d-10.28.212.13-j2] FIPS activated
 [f6e3b4993c33][          aos_nginx_1] FIPS activated
 [104caaddaf9c][      aos_controller_1] FIPS activated
 [ebf916a27b15][          aos_sysdb_1] FIPS activated
 [b11a6723bc26][          aos_auth_1] FIPS activated
 [b893a2fb8c54][        aos_license_1] FIPS activated
 [95c281aca3ee][         aos_metadb_1] FIPS activated
Checking SSH server settings...
 FIPS config found
Checking SSH client settings...
 FIPS config found
Host smoke test...
 FIPS is activated
Checking NGINX config...
 effective config is FIPS enabled
Overall status: ENABLED
```

Cluster Setup:

For clustered environments, FIPS mode must be enabled or disabled on all Apstra VMs in the cluster using the `aos_fips enable` or `aos_fips disable` command. The order of execution across VMs doesn't matter.

Upgrade Process

During an upgrade, the FIPS setting is preserved. The `aos_fips enable` command automatically runs post-upgrade to maintain FIPS mode, ensuring ongoing compliance.

FIPS 140-2 Implementations

Apstra has implemented FIPS 140-2 compliance across several components:

- **ZTP Server VM:** Ensures secure cryptographic operations during device initialization in the Zero Touch Provisioning process.

- **Apstra Controller VM:** Manages network orchestration, policy management, and device configuration securely, adhering to FIPS 140-2 standards for all cryptographic functions.
- **Apstra Worker VM:** Handles secure data processing and communication tasks, ensuring all cryptographic operations meet FIPS 140-2 requirements.
- **Off-box Device Agents:** Manages external devices with FIPS-compliant cryptographic communication between the agent and both the devices and the Controller VM.
- **On-box Device Agents:**Operates directly on network devices, securing configurations and communications. Note that this compliance applies only to the agent; Apstra doesn't enable FIPS on the network operating system itself.

NOTE: The underlying host operating system for these VMs and agents is not FIPS-140 enabled. This means the specific cryptographic modules within Apstra components are compliant, but the overall system security depends on the host environment.

19

PART

Apstra CLI Utility

Install Apstra CLI Utility

SUMMARY

Augment Juniper Apstra GUI functionality with Apstra CLI, Apstra's command line utility. You can use Apstra CLI on any system that's running a compatible version of Docker.

IN THIS SECTION

Install Apstra-CLI

1. Download the **Apstra CLI Utility** for your Apstra version from the **Application Tools** section of Juniper Support Downloads.
2. Copy the Apstra CLI Docker container `tar.gz` file to the Apstra server. (The file name is something like, `apstracli-release_4.2.0.11.tar.gz`.) For example:

```
scp apstracli-release_4.2.0.11.tar.gz admin@10.28.65.3/home/admin
```

3. Load the provided Docker image into Docker with the `docker image load` command. For example:

```
admin@aos-server:~$ docker image load -i apstracli-release_4.2.0.11.tar.gz
beee9f30bc1f: Loading layer [==================================================>] 5.862MB/
5.862MB
3fc750b41be7: Loading layer [==================================================>] 821.8kB/
821.8kB
20a7b70bdf2f: Loading layer [==================================================>] 59.53MB/
59.53MB
879c0d8666e3: Loading layer [==================================================>] 6.749MB/
6.749MB
ba4121fa2557: Loading layer [==================================================>] 3.451MB/
3.451MB
ee87976d1e1f: Loading layer [==================================================>] 470.4MB/
470.4MB
Loaded image: apstracli:release_4.2.0.11
admin@aos-server:~$
```

Start Apstra CLI

1. Start the Apstra CLI Docker container with the `docker run` command. In the example below, replace `4.2.0.11` with your Apstra CLI version, and replace `10.28.65.3` with the IP address of your Apstra server. The password is your Apstra GUI password (not the VM password).

```
admin@aos-server:~$ docker run --rm -ti -v ~:/mytmp apstracli:release_4.2.0.11 -s 10.28.65.3
Password [admin]:
Welcome to Juniper Apstra CLI! Press TAB for suggestions
Juniper Apstra CLI version: release-4.2.0.11
Juniper Apstra Server URL: https://10.28.65.3:443, Version: 4.2.0.11
apstra-cli>
```

2. Apstra CLI comes with a built-in feature that auto-completes commands. Press the TAB key, then the up and down arrow keys to explore this tool and its functionality. You can also type `--help` for descriptions of each function.

For examples of how to use apstra-cli, see "Apstra-CLI Commands" on page 1421 in the References section. For assistance with using Apstra CLI, contact "Juniper Support " on page 1374.

Apstra CLI Commands

SUMMARY

A few examples of apstra-cli commands. The complete list is available in apstra-cli.

IN THIS SECTION

scenario change-device-password

To comply with security requirements and best practices you may need to change root passwords and local user passwords on device system agents on a regular basis. You can change passwords on all devices in a blueprint by running a single command. Instead of entering a specific system ID you would enter `all`.

Use the following command to change all devices at once:

```
scenario change-device-password --blueprint <bp_id> --system all --old-password <old_password> --new-password
<new_password>
```

Use the following command to change a specific device:

```
scenario change-device-password --blueprint <bp_id> --system <sys_id> --old-password <old_password> --new-password
<new_password>
```

`scenario change-device-password` is a collection of the following eleven tasks:

- Check old password by ssh connection
- State creation of configlet for password
- Commit blueprint
- Check new password by ssh connection
- Change system agent password
- Check system agent status
- Update device pristine config
- State deletion of configlet used for password change
- Commit blueprint
- Check new password by ssh connection
- Check system agent status

scenario change-root-password

This command applies to Juniper, Arista and SONiC devices. Cisco devices are not supported. Use the following command to change all device root passwords at once:

```
scenario change-root-password --all --old-password <password> --new-password <password>
```

Use the following command to change a specific device root password:

```
scenario change-root-password --system <system> --old-password <password> --new-password <password>
```

config-syntax-check (Juniper only)

Command Syntax for Datacenter blueprints:

```
blueprint --blueprint <bp_id> config-syntax-check --system <sys_id> --username <device_username> --password
<device_password>
```

Command Syntax for Freeform blueprints:

```
blueprint --blueprint <bp_id> freeform-system config-syntax-check --system <sys_id> --username <device_username>
--password <device_password>
```

With the `config-syntax-check` command, you can verify configuration syntax on your Juniper devices before committing your blueprint. This check is useful when working with configlets in Datacenter blueprints and when working with config templates in Freeform blueprints.

This command works only with hierarchical configuration to verify whether configuration syntax is correct. It doesn't work for `set` commands.

RELATED DOCUMENTATION

Install Apstra CLI Utility | **1420**

PART 20

Guides

5-Stage Clos Architecture

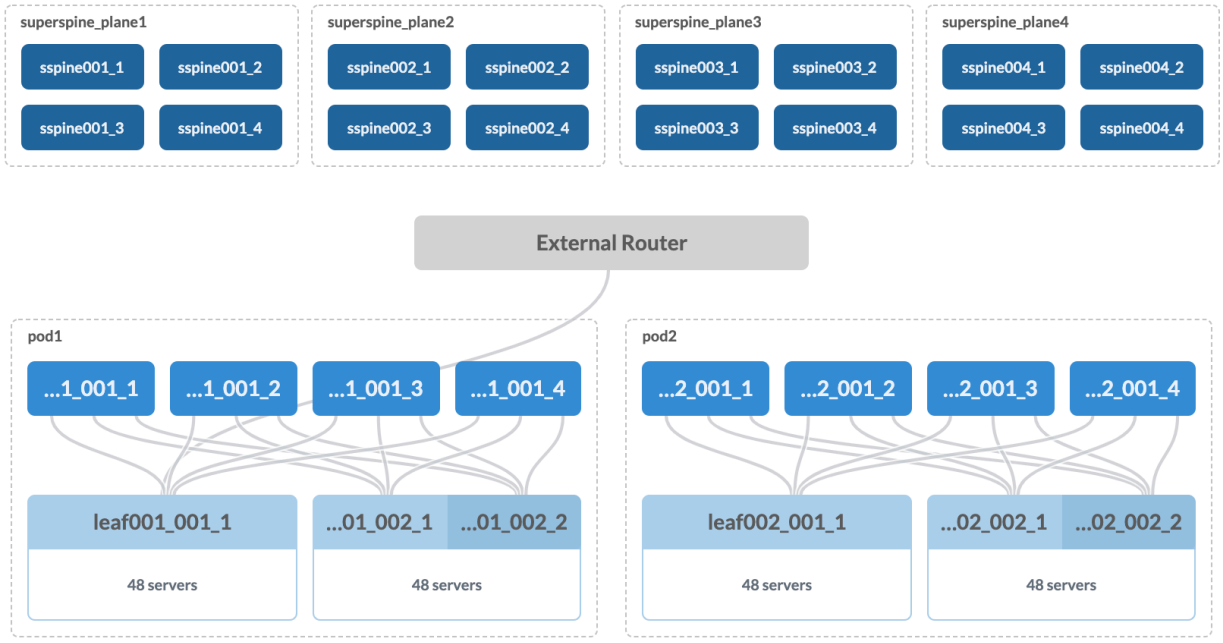
IN THIS SECTION

5-Stage Clos Overview

IN THIS SECTION

5-stage Clos architecture allows for large-scale topologies. With its additional aggregation layer, you can interconnect multiple **pods** into a single fabric. **Superspine** devices provide the additional layer that interconnects multiple pods. Planes are groups of superspine devices. Each 5-stage topology consists of

one or more planes. Each plane consists of one or more superspine devices. See below for an example.



Careful planning and consideration are required to build large 5-stage Clos networks. Refer to the limitations below when you're designing and validating your 5-stage topology. For assistance, contact "Juniper Support" on page 1374.

5-Stage Clos Limitations

- You cannot change a 3-stage topology to a 5-stage topology.
- You must use the same overlay control protocol (Pure IP Fabric or MP-EBGP-EVPN), specified during template creation) for all rack types in all pods.
- Root Cause Analysis is not supported.
- IPv6 / IPv4 support:
  - IPv6 support in the underlay depends on the NOS. See the"Apstra 5.0.0 Feature Matrix" on page 1481.
  - IPv6 applications are supported.
  - IPv6 virtual networks are supported on EVPN blueprints.
  - The entire fabric across all pods must be either all IPv4, all IPv6 or all dual-stack
- Unsupported external connectivity implementations:
  - One generic system connecting to multiple pods

  - EVPN with external generic systems on superspine devices
  - External generic systems on spine devices and leaf devices in the same pod
- Unsupported blueprint modifications:
  - Add or remove superspine planes

5-Stage Clos and EVPN

Extending EVPN networks across multiple pods within the same blueprint adds the following value:

- Scaling: provide any-to-any connectivity for applications distributed across multiple pods.
- Redistributing Workloads: To load-balance applications, you can migrate a group of applications from one pod to another pod while preserving application IP and MAC addresses.
- Performing pod maintenance: Migrate all applications from one pod to another, while preserving the application IP and MAC addresses.
- Active / Standby applications across sites / pods: Deploy A/S applications across multiple pods to provide high availability at pod level, or as part of application migration tasks.
- Facilitate external connectivity for a virtual network from a remote pod without external connectivity.

5-stage Clos networks support the Junos QFX series of switches. You can use the ESI redundancy protocol, create templates from them, and then use those templates as pods in 5-stage Clos networks. For more information about working with Juniper devices with EVPN, see "Juniper EVPN Support" on page 1429.

Just like in other Apstra-managed networks, required configuration is rendered to bring up multi-pod networks, and with proprietary *Intent-based Networking* technology the networks are validated to ensure they operate as designed.

The method for creating cross-pod "virtual networks" on page 251 is the same method as for 3-stage networks.

Create 5-Stage Clos Network

Creating a 5-stage Clos network follows the same workflow as for "3-stage Clos networks" on page 2, with the addition of creating a pod-based template and adhering to the 5-stage requirements described in the workflow below:

1. Confirm that the global catalog includes logical devices (Design > Logical Devices) that meet the 5-stage requirements below; create them if necessary:

- Make sure that devices have a sufficient number of ports and port groups; the exact number depends on your design.
- Spine logical devices require a leaf-facing port group, and if they will be facing a superspine device they also require a **Superspine** port role in that port group.
- Superspine logical devices require a **Spine** port role in the port group.

2. Confirm that the global catalog includes interface maps (Design > Interface Maps) that map the logical devices to the correct device profiles; create them if necessary. The required number of interface maps depends on your design; each device model used requires its own interface map. At a minimum, if you are using only one model, you need two interface maps as listed below:
   - Superspine logical device to device profile
   - Spine logical device to device profile
3. Create one or more rack-based templates, each including at least one link for **Superspine Connectivity**.
4. Create a pod-based template that uses as the pod the rack-based template(s) created in the previous step. Pod-based templates are essentially templates of templates where one or more rack-based templates are combined into a larger topology. (If you don't see the rack-based template that you created in the previous step in the pods drop-down list, it's probably because you didn't include a superspine-to-spine link.)
5. Create pools for resources ("ASNs" on page 928, "IPv4 addresses" on page 934, "IPv6 addresses" on page 937) needed in the network.
6. "Create a blueprint" on page 8 using the pod-based template that you created in the previous step.
7. Build the 5-stage Clos network in the same manner as for building a 3-stage Clos network.

SEE ALSO

What are Logical Devices | 845

What are Interface Maps | 853

What are Templates | 889

Modify 5-stage Clos Network

You can modify 5-stage blueprints in the same manner as for 3-stage networks, provided that you take into account the limitations described above. For information about rack changes, see Racks. For information about adding and removing pods, or changing pod names, see "Pods" on page 237, and for information about adding superspine devices to planes see Planes."Racks (Datacenter)" on page 229

Juniper EVPN Support

IN THIS SECTION

Overview

The Junos EVPN ESI multi-homing feature enables you to directly connect end servers to leaf devices and provide redundant connectivity via multi-homing. This feature is supported only on LAGs that span two leaf devices on the fabric. EVPN ESI also removes the need for "peer-link", and hence facilitates clean leaf-spine design.

Blueprints using the **MP-EBGP EVPN** Overlay Control Protocol can use Juniper Junos devices. Racks with leaf-pair redundancy can implement **EVPN ESI multi-homing**.

EVPN ESI multi-homing helps to maintain EVPN service and traffic forwarding to and from the multi-homed site in the event of the following types of network failures and avoid single point of failure as per the scenarios below:

- Link failure from one of the leaf devices to end server device
- Failure of one of the leaf devices
- Fast convergence on the local VTEP by changing next-hop adjacencies and maintaining end host reachability across multiple remote VTEPs

EVPN multi-homing Terminology and Concepts

The following terminology and concepts are used with EVPN multi-homing:

EVI - EVPN instance that spans between the leaf devices making up the EVPN. It's represented by the Virtual Network Identifier (VNI). EVI is mapped to VXLAN-type virtual networks (VN).

MAC-VRF - A virtual routing and forwarding (VRF) table to house MAC addresses on the VTEP leaf device (often called a "MAC table"). A unique route distinguisher and VRF target is configured per MAC-VRF.

Ethernet Segment (ES) - Ethernet links span from an end host to multiple ToR leaf devices and form ES. It constitutes a set of bundled links.

Ethernet Segment Identifier (ESI) - Represents each ES uniquely across the network. ESI is only supported on LAGs that span two leaf devices on the fabric.

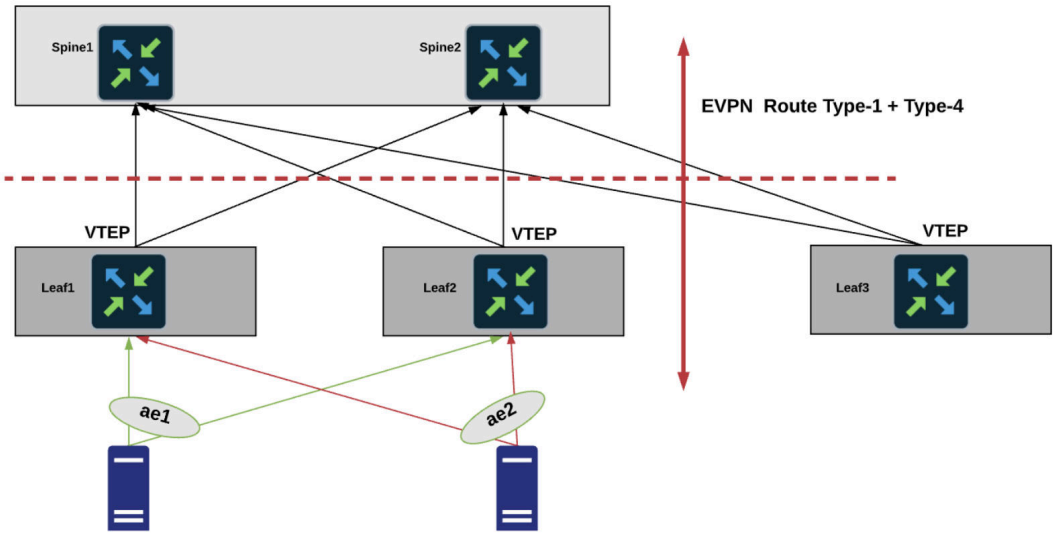
ESI helps with end host level redundancy in an EVPN VXLAN-based blueprint. Ethernet links from each Juniper ToR leaf connected to the server are bundled as an aggregated Ethernet interface. LACP is enabled for each aggregated Ethernet interface of the Juniper devices. Multi-homed interfaces into the ES are identified using the ESI.

ESI has certain restrictions and requirements as listed below:

- ESI based ToR leaf devices cannot have any L2/L3 peer links as EVPN multi-homing eliminates peer links used by MLAG/vPC.
- A bond of two physical interfaces towards a single leaf is not supported in the ESI implementation; make sure the server with LAG in that rack type spans two leaf devices.
- ESI and MLAG/vPC-based rack types cannot be mixed in a single blueprint.
- L2 External Connectivity Points (ECPs) with an ESI-based rack type is not supported. Only L3 ECPs are supported.
- Per-leaf VN assignment - having different VLAN sets among individual leaf devices for an ESI-based port channel is not supported.
- Connecting a single server to a single leaf using a bond of two physical interfaces cannot use an ESI.
- ESI is supported only on LAGs (port-channels) and not directly on physical interfaces. This has no functional impact, as leaf local port-channels for multi-home links are automatically generated.
- Only ESI **active-active redundancy** mode is supported. Active-standby mode is not supported.
- **active-active** redundancy mode is only supported for Juniper EVPN multi-homing where each Juniper ToR leaf attached to an ES is allowed to forward traffic to and from a given VLAN.
- More than two leaf devices in one ESI segment using ESI-based rack types is not supported.
- Switching from an ESI to MLAG rack type or vice versa is not supported under Flexible Fabric Expansion (FFE) operations.

Topology Specification

In the example below Leaf1 and Leaf2 are part of the same ES, and Leaf3 is the switch sending traffic towards the ES.



Juniper EVPN multi-homing uses five route types:

- Type 1 - Ethernet Auto-Discovery (EAD) Route
- Type 2 - MAC advertisement Route
- Type 3 - Inclusive Multicast Route
- Type 4 - Ethernet Segment Route
- Type 5 - IP Prefix Route

BGP EVPN running on Juniper devices use:

- Type 2 to advertise MAC and IP (host) information
- Type 3 to carry VTEP information
- Type 5 to advertise IP prefixes in a Network Layer Reachability Information (NLRI).

NOTE: In Junos MAC/IP Type 2 route type doesn't contain VNI and RT for the IP part of the route, it is derived from the accompanying Type 5 route type.

Type 1 routes are used for per-ES auto-discovery (A-D) to advertise EVPN multi-homing mode. Remote ToR leaf devices in the EVPN network use the EVPN Type 1 route type functionality to learn the EVPN Type 2 MAC routes from other leaf devices. In this route type ESI and the Ethernet Tag ID are considered to be part of the prefix in the NLRI. Upon a link failure between ToR leaf and end server VTEP withdraws Ethernet Auto-Discovery routes (Type 1) per ES. The Juniper EVPN multi-homing Ethernet Tag value is set to the VLAN ID for ES auto-discovery/ES route types.

Mass Withdrawal - Used for fast convergence during link failure scenarios between leaf devices to the end server using Type 1 EAD/ES routes.

DF Election - Used to prevent forwarding of the loops and the duplicates as only a single switch is allowed to decapsulate and forward the traffic for a given ES. Ethernet Segment Route is exported and imported when ESI is locally configured under the LAG. Type 4 NLRI is mainly used for designated forwarder(DF) elections and to apply Split Horizon Filtering.

Split Horizon - It is used to prevent forwarding of the loops and the duplicates for the Broadcast, Unknown-unicast and Multicast (BUM) traffic. Only the BUM traffic that originates from a remote site is allowed to be forwarded to a local site.

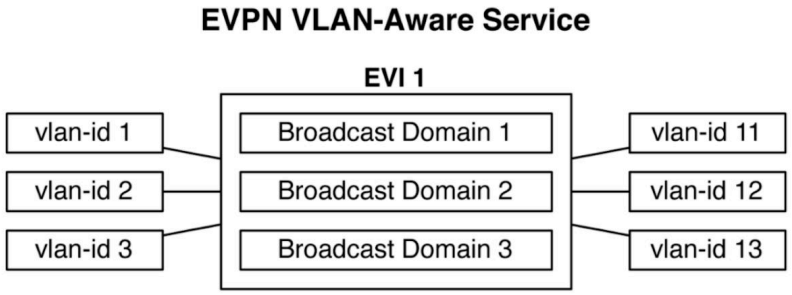
EVPN Services

IN THIS SECTION

EVPN VLAN-Aware

Junos can support three Ethernet Services: (1) VLAN-based, (2) VLAN Bundle, or (3) VLAN-Aware. Apstra's data center reference design natively leverages the VLAN-Aware model. With the EVPN VLAN-Aware Service each VLAN is mapped directly to its own EVPN instance (EVI). The mapping between VLAN, Bridge Domain (BD) and EVPN instance (EVI) is N:1:1. For example, N VLANs are mapped into a

single BD mapped into a single EVI. In this model all VLAN IDs share the same EVI as shown below:



VLAN-aware Ethernet Services in Junos have a separate Route target for each VLAN (which is Juniper internal optimization), so each VLAN has a label to mimic VLAN-based implementations.

From the control plane perspective EVPN MAC/IP routes (Type 2) for VLAN-aware services carry VLAN ID in the Ethernet Tag ID attribute that is used to disambiguate MAC routes received.

From the data plane perspective - every VLAN is tagged with its own VNI that is used during packet lookup to place it onto the right Bridge Domain(BD)/VLAN.

Create EVPN Network

Creating an EVPN network follows the same workflow as for other networks.

1. Create/Install "offbox device agents" on page 685 for all switches. (Onbox agents are not supported on Junos.)
2. Confirm that the global catalog includes logical devices (Design > Logical Devices) that meet Juniper device requirements; create them if necessary:
3. Confirm that the global catalog includes interface maps (Design > Interface Maps) that map the logical devices to the correct device profiles for the Juniper devices; create them if necessary.
4. Create a rack type.
   - For single leaf racks, specify redundancy protocol **None** in the **Leaf** section.
   - For dual leaf racks
     - Specify redundancy protocol **ESI** in the **Leaf** section.
     - When specifying the end server in the **Server** section, specify attachment type as **Dual-Homed** towards ESI-based ToR leaf devices. EVPNs using ESs have a link aggregation option. Select the LAG mode **LACP (Active)**

5. Create a rack-based template.
6. Create a generic system for an external router.
7. Create resource pools for "ASNs" on page 928, "IP addresses" on page 934, and "VNIs" on page 930.
8. "Create a blueprint" on page 8 based on the ESI-based template, then build the EVPN-based network topology for the Juniper devices by "assigning resources" on page 58, "device profiles" on page 62, and "device IDs" on page 64.

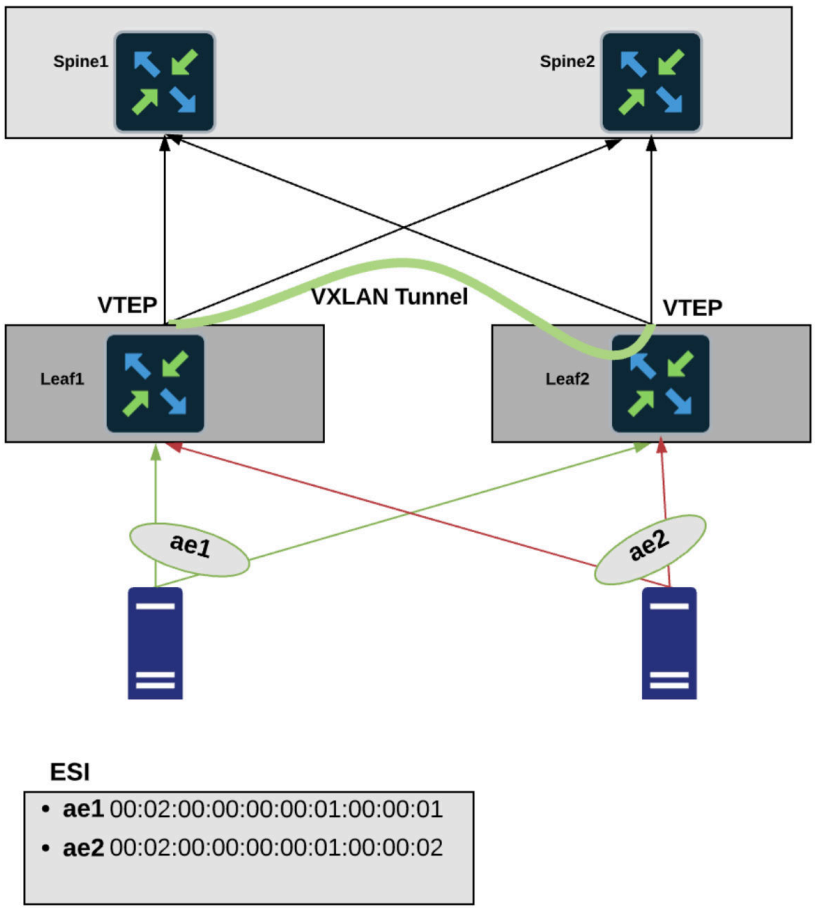
Configuration Rendering

IN THIS SECTION

Reference Design

- **Underlay** - The underlay in the data center fabric is Layer-3 configured using standard eBGP over the physical interfaces of Juniper devices.
- **Overlay** - Overlay is configured eBGP over `lo0.0` address. EVPN VXLAN is used as an overlay protocol. All the ToR devices are enabled with L2 VN. Each one of these L2 VNs can have its default gateway hosted on connected ToR leaf devices. For the inter-VN traffic VXLAN routing is done in the fabric using L3 VNIs on the border leaf devices as per standard design.
- **VXLAN VTEPs** - On Juniper leaf devices one IP address on `lo0.0` is rendered which is used as VTEP address. The VTEP IP address is used to establish the VXLAN tunnel.
- **EVPN multi-homing LAG** - **Unique ESI value** and **LACP system IDs** are used per EVPN LAG. The multi-homed links are configured with an ESI and a LACP system identifier is specified for each link. The ESI is used to identify LAG groups and loop prevention. To support Active/Active and multi-homing for Juniper leaf devices, they are configured with the same LACP parameter for a given ESI

so that they appear as a single system.



ESI MAC addresses are auto-generated internally. You can "configure the value of the most significant byte (msb)" on page 435 used in the generated MAC. A new facade API is added to update the MSB value. A new node is added to the rack based template that contains the MAC MSB value. The default value of this byte is 2 and you can change it to any even number up to 254. Updating this value results in regeneration of all ESI MACs in the blueprint. This is exposed to address DCI use cases where ESIs must be unique across multiple blueprints (IP Fabrics).

- **L3VNIs** - L3VNI is rendered as a routing zone per VRF. Multi-tenancy functionality is available to ensure that workloads remain logically separated within a VN (overlay) construct using routing zone.
- **Route Target (RT) for L2/L3 VNIs** - Auto-generated for L2/L3 VNIs in the format VNI:1. There is 1 (fabric-wide) RT per MAC-VRF (that is, L3VNI). The value must be the same across all switches participating in one EVI. You can find the RT in the blueprint by navigating to **Staged > Virtual > Virtual Networks** and clicking the VN name. RT is in the parameters section.

- **Route Distinguisher (RD) for L2/L3 VNIs** - For Junos VLAN-Aware based model, the RD is per EVI (switch). There is no RD for each l2 VNI. RD exists only for routing zone VRF in the format `{primary_loopback}:vlan_id`.

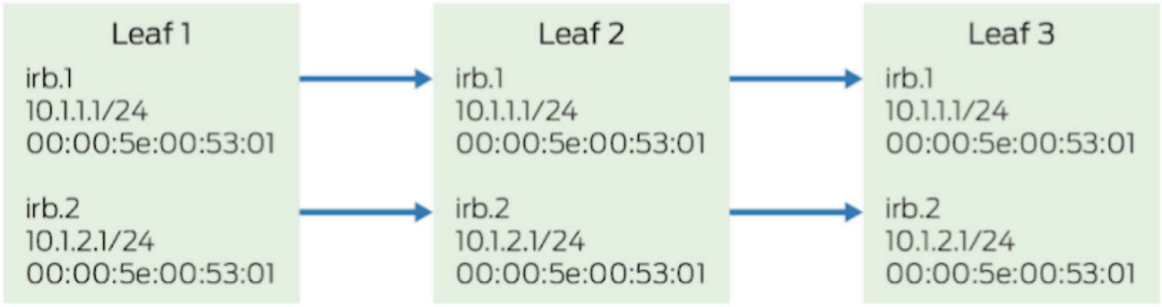
- **Virtual Switch Configuration** - Under the *switch-options* hierarchy for Juniper devices the *vtep-source-interface* parameter is rendered, then the VTEP IP address used to establish the VXLAN tunnel is specified. Reachability to loopback interface (for example, lo0.0) is provided by the underlay. The RD here defines the EVI specific RD carried by Type 1, Type 2, Type 3 routes. RD for the global switch options is provided in the format `{loopback_id}:65534`.

  The RT here defines the global RT inherited by EVPN routes. It is used by Type 1 routes. A default RT value is rendered for it (`100:100`) for global switch options across all switches.

- **MTU** - The MTU values that are rendered for Juniper Devices:

  - L2 ports: 9100
  - L3 ports: 9216
  - Integrated Routing and Bridging (IRB) Interfaces: 9000

- **Anycast Gateway** - The same IP on IRB interfaces of all the leaf devices is configured and no virtual gateway is set. Every IRB interface that participates in the stretched L2 service has the same IP/MAC configured as below:



In this model, all default gateway IRB interfaces in an overlay subnet are configured with the same IP and MAC address. A benefit of this model is that only a single IP address is required per subnet for default gateway IRB interface addressing, which simplifies gateway configuration on end systems.

Here MAC address of the IRB is auto generated.

Limitations

The following limitations apply to EVPN multi-homing topologies for Juniper devices:

- Only two-way multi-homing is supported. More than two Juniper leaf devices in a multi-homed group is not supported.

- Juniper EVPN with EVPN on other network vendors in the same blueprint is not supported.
- No Pure IP Fabric support.
- IPv6-based fabrics do not support Junos.
- In Juniper EVPN multi-homing, L3 External Connectivity Points (ECP) towards generic systems are supported; L2 ECP is not supported.
- BGP routing from Junos leaf devices to Apstra-managed Layer 3 servers is not supported.

SEE ALSO

Create Rack-based Template | **901**

Extensible Telemetry Guide

IN THIS SECTION

Extensible Telemetry Overview

Install Apstra device drivers and telemetry collectors to collect additional telemetry that can be used in "analytics probes" on page 29. The device drivers enable Apstra to connect to a NOS and collect telemetry. Apstra ships with drivers for EOS, NX-OS, Ubuntu, and CentOS. To add a driver for an operating system not listed here, contact "Juniper Support" on page 1374.

Telemetry collectors are Python modules that help collect extended telemetry. The following sections describe the pipeline for creating telemetry collectors and extending Apstra with new collectors. You need familiarity with Python to be able to develop collectors.

Set Up Development Environment

To get access to telemetry collectors (which are housed in the *aos_developer_sdk* repository) contact "Juniper Support" on page 1374. Contribute any new collectors that you develop to the repository.

To keep your system environment intact, we recommend that you use a virtual environment to isolate the required Python packages (for development and testing). You can download the base development environment, *aos_developer_sdk.run*, from https://support.juniper.net/support/downloads/?p=apstra/. To load the environment, execute:

```
aos_developer_sdk$ bash aos_development_sdk.run
4d8bbfb90ba8: Loading layer [==================================================>]  217.6kB/
217.6kB
7d54ea05a373: Loading layer [==================================================>]  4.096kB/
4.096kB
e2e40f457231: Loading layer [==================================================>]  1.771MB/
1.771MB
Loaded image: aos-developer-sdk:2.3.1-129

================================================================================
Loaded AOS Developer SDK Environment Container Image
aos-developer-sdk:2.3.1-129.

Container can be run by
    docker run -it \
        -v <path to aos developer_sdk cloned repo>:/aos_developer_sdk \
        --name <container name> \
        aos-developer-sdk:2.3.1-129

================================================================================
```

This command loads the *aos_developer_sdk* Docker image. After the image load is complete, the command to start the environment is printed. Start the container environment as specified by the command. To install the dependencies, execute:

```
root@f2ece48bb2f1:/# cd /aos_developer_sdk/
root@f2ece48bb2f1:/aos_developer_sdk# make setup_env
...
```

The environment is now set up for developing and testing the collectors. Apstra SDK packages, such as device drivers and REST client, are also installed in the environment.

Develop Collector

To develop a telemetry collector, specify the following *in order*.

1. **Service for which the collector is developed** - Identify what the service is. For example, the service could be to collect received and transmitted bytes from the switch interfaces. Identify a name for the service. Using service names that are reserved for built-in services (ARP, BGP, interface, hostname, route, MAC, XCVR, LAG, MLAG) is prohibited.
2. **The schema of the data provided to Apstra** - Identify how the collector output is to be structured. A collection of key-value pairs should be posted to Apstra. Identify what each item is, that is, what is the key/value syntactically and semantically. For the above mentioned example, key is a string that identifies the interface name. The value is a JSON string, with the JSON having two keys 'rx' and 'tx' both having an integer value.
3. **Network Operating System (NOS) for which the collector is developed** - The collector plugins are NOS-specific. Before writing a collector, identify the NOS(s) for which collector(s) are required.
4. **How the required data can be obtained from the device** - Identify the commands that can be used in the device to retrieve the required information. For example, 'show interfaces' command gives received and transmitted bytes from an Arista EOS device.
5. **Storage Schema Path** - The type of key and value in each item determines the storage schema path. The type of collector selected determines the storage schema for the application. The storage schema defines the high level structure of the data returned by the service. The storage schema path for your collector can be determined using the following table:

   Table 61: Determining Storage Schema Path

Key Type	Value Type	Storage Schema Path
String	String	aos.sdk.telemetry.schemas.generic

Table 61: Determining Storage Schema Path *(Continued)*

Key Type	Value Type	Storage Schema Path
String	Dict	aos.sdk.telemetry.schemas.generic
Dict	String	aos.sdk.telemetry.schemas.iba_string_data
Dict	Integer	aos.sdk.telemetry.schemas.iba_integer_data

6. **Application Schema** - Application schema defines the schema for each item posted to the framework. Application schema is expressed using draft 4 version of json schema. Each item is comprised of a key and value. The following table specifies two sample items.

Table 62: Sample item with its storage schema path

Storage Schema Path	Sample Item

aos.sdk.telemetry.schemas.generic	`{`
	`"identity": "eth0",`
	`"value": "up",`
	`}`

aos.sdk.telemetry.schemas.iba_string_data	`{`
	`"key": {`
	`"source_ip": "10.1.1.1",`
	`"dest_ip": "10.1.1.2",`
	`},`
	`"value": "up",`
	`}`

NOTE: * An item returned by collectors with generic storage schema should specify the key value using the key 'identity' and the value using the key 'value'.

* An item returned by collectors with IBA-based schemas should specify the key value using the key 'key' and the value using the key 'value'.

Using this information, you can write the JSON schema. The following table maps the sample item specified above to its corresponding JSON schema.

Table 63: Sample Application Schema

Sample Item	Application Schema
{ "identity": "eth0", "value": "up", }	{ "type": "object", "properties": { "identity": { "type": "string", }, "value": { "type": "string", } } } {
{ "key": { "source_ip": "10.1.1.1", "dest_ip": "10.1.1.2", }, "value": "up", }	{ "type": "object", "properties": { "key": { "type": "object", "properties": { "source_ip": { "type": "string", "format": "ipv4" }, "dest_ip": { "type": "string", "format": "ipv4" }, "required": ["source_ip", "dest_ip"], } }, "value": { "type": "string", } } } {

You can specify more complex schema using the constructs available in JSON schema. Update the schema in the file `aos_developer_sdk/aosstdcollectors/aosstdcollectors/json_schemas/<service_name>.json`

NOTE: As of Apstra version 4.0.1, you can "import the service schema" on page 941 via the GUI.

Write Collector

IN THIS SECTION

Collector is a class that must derive from *aos.sdk.system_agent.base_telemetry_collector.BaseTelemetryCollector*. Override the *collect* method of the collector with the logic to:

Collect Data from Device

The device driver instance inside the collector provides methods to execute commands against the devices. For example, most Apstra device drivers provide methods `get_json` and `get_text` to execute commands and return the output.

NOTE: The device drivers for aos_developer_sdk environment are preinstalled. You can explore the methods available to collect data. For example:

```
>>> from aos.sdk.driver.eos import Device
>>> device = Device('172.20.180.10', 'admin', 'admin')
>>> device.open()
>>> pprint.pprint(device.get_json('show version'))
{u'architecture': u'i386',
 u'bootupTimestamp': 1548302664.0,
 u'hardwareRevision': u'',
 u'internalBuildId': u'68f3ae78-65cb-4ed3-8675-0ff2219bf118',
 u'internalVersion': u'4.20.10M-10040268.42010M',
```

```
   u'isIntlVersion': False,
   u'memFree': 3003648,
   u'memTotal': 4011060,
   u'modelName': u'vEOS',
   u'serialNumber': u'',
   u'systemMacAddress': u'52:54:00:ce:87:37',
   u'uptime': 62620.55,
   u'version': u'4.20.10M'}
  >>> dir(device)
  ['AOS_VERSION_FILE', '__class__', '__delattr__', '__dict__', '__doc__',
  '__format__', '__getattribute__', '__hash__', '__init__', '__module__',
  '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
  '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'close',
  'device_info', 'driver', 'execute', 'get_aos_server_ip',
  'get_aos_version_related_info', 'get_device_aos_version',
  'get_device_aos_version_number', 'get_device_info', 'get_json',
  'get_text', 'ip_address', 'onbox', 'open', 'open_options', 'password',
  'probe', 'set_device_info', 'upload_file', 'username']
```

Parse Data

The collected data needs to be parsed and re-formatted per the Apstra framework and the service schema identified above. Collectors with generic storage schema follow the following structure:

```
{
    "items": [
        {
            "identity": <key goes here>,
            "value": <value goes here>,
        },
        {
            "identity": <key goes here>,
            "value": <value goes here>,
        },
        ...
    ]
}
```

Collectors with IBA-based schema follow the following structure:

```
[
    {
        "key": <key goes here>,
        "value": <value goes here>,
    },
    {
        "key": <key goes here>,
        "value": <value goes here>,
    },
    ...
]
```

In the structures above, the data posted has multiple items. Each item has a key and a value. For example, to post interface specific information, there would be an identity/key-value pair for each interface you want to post to the framework.

NOTE: In the case when you want to use a third party package to parse data obtained from a device, list the Python package and version in the path. `<aos_developer_sdk>/aosstdcollectors/requirements_<NOS>.txt`. The packages installed by the dependency do not conflict with packages that Apstra software uses. The Apstra-installed packages are available at `/etc/aos/python_dependency.txt` in the development environment.

Post Data to Framework

When data is collected and parsed as per the required schema, post the data to the framework. You can use the `post_data` method available in the collector. It accepts one argument, and that is the data that should be posted to the framework.

The folder `aos_developer_sdk/aosstdcollectors/aosstdcollectors` in the repository contains folders for each NOS. Add your collector to the folder that matches the NOS. Cumulus is no longer supported as of Apstra version 4.1.0, although this example remains for illustrative purposes. For example, to write a collector for Cumulus, add the collector to `aos_developer_sdk/aosstdcollectors/aosstdcollectors/cumulus`, and name the file after the service name. For example, if the service name is `interface_in_out_bytes`, then name the file `interface_in_out_bytes.py`.

In addition to defining the collector class, define the function `collector_plugin` in the collector file. The function takes one argument and returns the collector class that is implemented.

For example, a generic storage schema based collector looks like:

```
"""
    Service Name: interface_in_out_bytes
    Schema:
        Key: String, represents interface name.
        Value: Json String with two possible keys:
            rx: integer value, represents received bytes.
            tx: integer value, represents transmitted bytes.
    DOS: eos
    Data collected using command: 'show interfaces'
    Type of Collector: BaseTelemetryCollector
    Storage Schema Path: aos.sdk.telemetry.schemas.generic
    Application Schema: {
        'type': 'object',
        'properties': {
            'identity': {
                'type': 'string',
            },
            'value': {
                'type': 'object',
                'properties': {
                    'rx': {
                        'type': 'number',
                    },
                    'tx': {
                        'type': 'number',
                    }
                },
                'required': ['rx', 'tx'],
            }
        }
    }

"""
import json
from aos.sdk.system_agent.base_telemetry_collector import BaseTelemetryCollector


# Inheriting from BaseTelemetryCollector
class InterfaceRxTxCollector(BaseTelemetryCollector):
```

```
    # Overriding collect method
    def collect(self):

        # Obtaining the command output using the device instance.
        collected_data = self.device.get_json('show interfaces')

        # Data is in the format
        # "interfaces": {
        #     "<interface_name>": {
        #         ....
        #         "interfaceCounters": {
        #         ....
        #         "inOctets": int
        #         "outOctets": int
        #         ....
        #         }
        #     }
        #     ...
        # }

        # Parse the data as per the schema and structure required.
        parsed_data = json.dumps({
            'items': [
                {
                    'identity': intf_name,
                    'value': json.dumps({
                        'rx': intf_stats['interfaceCounters'].get('inOctets'),
                        'tx': intf_stats['interfaceCounters'].get('outOctets'),
                    })
                } for intf_name, intf_stats in collected_data['interfaces'].iteritems()
                if 'interfaceCounters' in intf_stats
            ]
        })

        # Post the data to the framework
        self.post_data(parsed_data)


# Define collector_plugin class to return the Collector
def collector_plugin(_device):
    return InterfaceRxTxCollector
```

An IBA storage schema based collector looks like:

```
"""
    Service Name: iba_bgp
    Schema:
        Key: JSON String, specifies local IP and peer IP.
        Value: String. ‘1’ if state is established ‘2’ otherwise
    DOS: eos
    Data collected using command: 'show ip bgp summary vrf all'
    Storage Schema Path: aos.sdk.telemetry.schemas.iba_string_data
    Application Schema: {
        'type': 'object',
        'properties': {
            key: {
                'type': 'object',
                'properties': {
                    'local_ip': {
                        'type': 'string',
                    },
                    'peer_ip': {
                        'type': 'string',
                    }
                },
                'required': ['local_ip', 'peer_ip'],
            },
            'value': {
                'type': 'string',
            }
        }
    }
"""

from aos.sdk.system_agent.base_telemetry_collector import IBATelemetryCollector

def parse_text_output(collected):
    result = [
        {'key': {'local_ip': str(vrf_info['routerId']), 'peer_ip': str(peer_ip)},
         'value': str(
             1 if session_info['peerState'] == 'Established' else 2)}
        for vrf_info in collected['vrfs'].itervalues()
        for peer_ip, session_info in vrf_info['peers'].iteritems()]
    return result
```

```
# Inheriting from BaseTelemetryCollector
class IbaBgpCollector(BaseTelemetryCollector):
    # Overriding collect method
    def collect(self):
        # Obtaining the command output using the device instance.
        collected_data = self.device.get_json('show ip bgp summary vrf all')
    # Parse the data as per the schema and structure required and
    # post to framework.
        self.post_data(parse_text_output(collected_data))

# Define collector_plugin class to return the Collector
def collector_plugin(device):
    return IbaBgpCollector
```

Unit Test Collector

The folder `aos_developer_sdk/aosstdcollectors/test` in the repository contains folders based on the NOS. Add your test to the folder that matches the NOS. For example, a test to a collector for Cumulus is added to `aos_developer_sdk/aosstdcollectors/test/cumulus`. We recommend that you name the unit test with the prefix `test_`.

The existing infrastructure implements a Pytest fixture `collector_factory` that is used to mock the device driver command response. The general flow for test development is as follows.

1. Use the collector factory to get a collector instance and mocked Apstra framework. The collector factory takes the collector class that you have written as input.
2. Mock the device response.
3. Invoke collect method.
4. Validate the data posted to the mocked Apstra framework.

For example, a test looks like:

```
import json
from aosstdcollectors.eos.interface_in_out_bytes import InterfaceRxTxCollector


# Test method with prefix 'test_'
```

```
def test_sanity(collector_factory):

    # Using collector factory to retrieve the collector instance and mocked
    # Apstra framework.
    collector, mock_framework = collector_factory(InterfaceRxTxCollector)

    command_response = {
        'interfaces': {
            'Ethernet1': {
                'interfaceCounters': {
                    'inOctets': 10,
                    'outOctets': 20,
                }
            },
            'Ethernet2': {
                'interfaceCounters': {
                    'inOctets': 30,
                    'outOctets': 40,
                }
            }
        }
    }
    # Set the device get_json method to retrieve the command response.
    collector.device.get_json.side_effect = lambda _: command_response

    # Invoke the collect method
    collector.collect()

    expected_data = [
        {
            'identity': 'Ethernet1',
            'value': json.dumps({
                'rx': 10,
                'tx': 20,
            }),
        },
        {
            'identity': 'Ethernet2',
            'value': json.dumps({
                'rx': 30,
                'tx': 40,
            })
        }
```

```
    ]
    # validate the data posted by the collector
    data_posted_by_collector = json.loads(mock_framework.post_data.call_args[0][0])
    assert sorted(expected_data) == sorted(data_posted_by_collector["items"])
```

To run the test, execute:

```
root@1df9bf89aeaf:/aos_developer_sdk# make test
root@1df9bf89aeaf:/aos_developer_sdk# make test
```

This command executes all the tests in the repository.

Package Collector

All the collectors are packaged based on the NOS. To generate all packages, execute make at `aos_develop_sdk`. You can find the build packages at `aos_developer_sdk/dist`. The packages build can be broadly classified as:

Package	Description
Built-In Collector Packages	These packages have the prefix *aosstdcollectors_builtin_*. To collect telemetry from a device per the reference design, Apstra requires services as listed in the `<deviceblah>` section. Built-In collector packages contain collectors for these services. The packages are generated on a per NOS basis.
Custom Collector Packages	These package have the prefix *aosstdcollectors_custom_* in their names. The packages are generated on a per NOS basis. The package named *aosstdcollectors_custom_ <NOS>-0.1.0-py2-none-any.whl* contains the developed collector.
Apstra SDK Device Driver Packages	These packages have a prefix *apstra_devicedriver_*. These packages are generated on a per NOS basis. Packages are generated for NOS that are not available by default in Apstra.

Upload Packages

If the built-in collector packages and the Apstra SDK Device Driver for your Device Operating System (NOS) were not provided with the Apstra software, you must upload them to the Apstra server.

If you are using an offbox solution and your NOS is not EOS, you must upload the built-in collector package.

Upload the package containing your collector(s) and assign them to a Device System Agent or System Agent Profile.

Use Telemetry Collector

IN THIS SECTION

Set up Telemetry Service Registry

The registry maps the service to its application schema and the storage schema path. You can manage the telemetry service registry with the REST endpoint `/api/telemetry-service-registry`. You can't enable the collector for a service without adding a registry entry for the particular service. The registry entry for a service cannot be modified while the service is in use.

NOTE: When executing `make`, all application schemas are packaged together to a tar file (json_schemas.tgz) in the dist folder. With apstra-cli, you have the option of importing all the schemas in the .tgz file.

Start Collector

To start a service, use the POST API `/api/systems/<system_id>/services` with the following three arguments:

Arguments	
Input_data	The data provided as input to the collector. Defaults to None.
Interval	Interval at which to run the service. Defaults to 120 seconds.
Name	Name of the service.

NOTE: You can also manage collectors via the apstra-cli utility.

Delete Collector

To delete a service, use the DELETE API `/api/systems/<system_id>/services/<service_name>`.

Get Collected Data

To retrieve collected data, use the GET API `/api/systems/<system_id>/services/<service_name>/data`. Only the data collected in the last iteration is saved. Data does not persist over Apstra restart.

List Running Collector Services

To retrieve the list of services enabled on a device, use the GET API `/api/systems/<system_id>/services`.

Intent-Based Analytics with apstra-cli Utility

IN THIS SECTION

IBA with apstra-cli Overview

You can work with Intent-based analytics (IBA) from the Apstra GUI, or for non-production environments you can use the experimental apstra-cli utility (formerly called aos-cli). For information about how to use IBA probes from the GUI, see "Probes Introduction" on page 29 in the Analytics section. This guide shows you how to use apstra-cli.

> **NOTE**: The apstra-cli utility is an experimental tool and has limited support. Do not use it in production environments unless advised by Juniper Support. Some versions of apstra-cli are not intended for certain Apstra releases. Some apstra-cli commands may or may not work between different Apstra releases. It's always best to test a version of apstra-cli with a specific Apstra release in a non-production environment, or contact "Juniper Support" on page 1374 for assistance.

The apstra-cli utility enables you to extract information from the Apstra server for analytics (and other functionalities). The workflow for IBA probes is as follows:

1. Install apstra-cli.
2. Install packages.
3. Create device agent profiles.
4. Install device agents.
5. Install IBA probes.

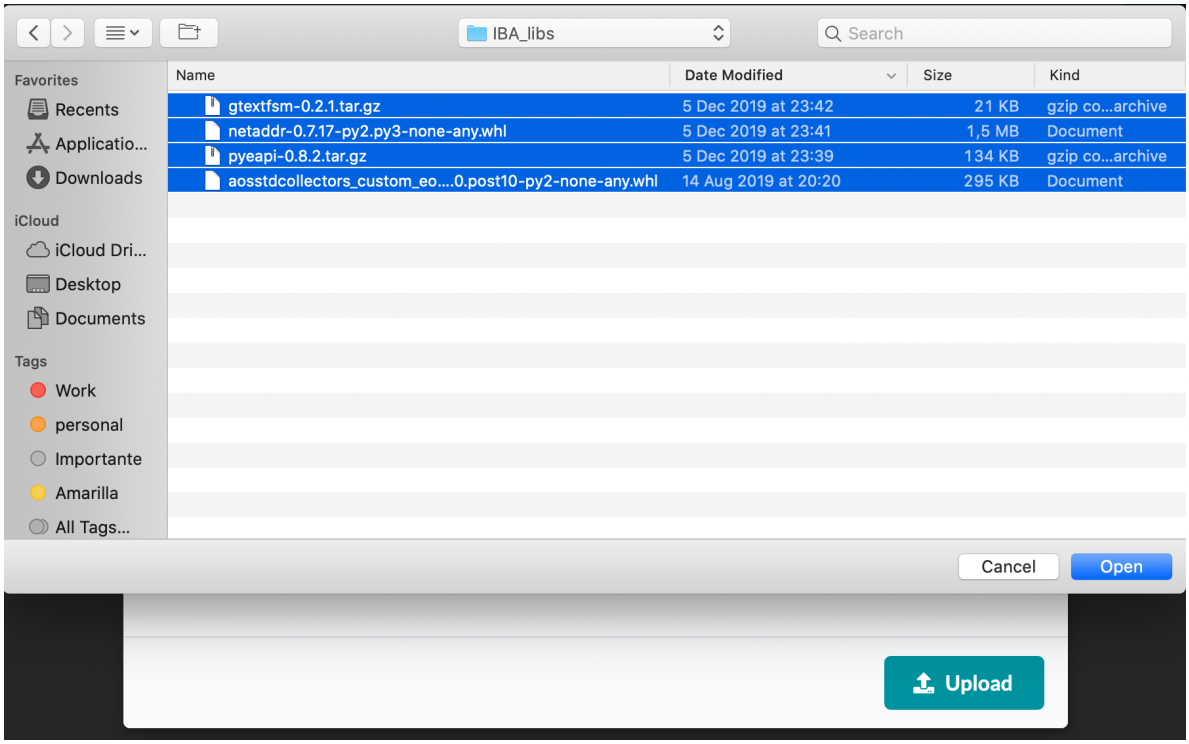
After probes are instantiated you can use "Syslog" on page 1290 to send messages to Syslog servers.

Install apstra-cli

"Install the apstra-cli utility" on page 1420.

Install Packages

1. Download the latest Apstra SDK package from Juniper Support Knowledge Base article KB37156.
2. Custom collector packages enable the collection of telemetry from devices. Extract the collector for your platform (for example, `aosstdcollectors_custom_eos-0.1.0.post10-py2-none-any.whl` where `eos` is the platform and `10` is the version).
3. Collectors require specific Python library packages. If the Apstra environment has Internet access, the files are automatically installed. If the environment doesn't have Internet access, download the following files from the official Python repository. Make sure to download the correct versions:
   - netaddr-0.7.17-py2.py3-none-any.whl
   - gtextfsm-0.2.1.tar.gz
   - pyeapi-0.8.2.tar.gz
4. From the left navigation menu in the Apstra GUI, navigate to **Devices > System Agents > Packages** and click **Upload Packages**.
5. Either click **Choose File** and navigate to the custom collector package (and if the Internet is inaccessible, the three (3) Python packages), or drag and drop the file(s) into the dialog window. See example below for Arista devices in an environment without Internet access:



Upload Packages

Drag and drop files here or click the button below.

Choose Files

aosstdcollectors_custom_eos-0.1.0.post10-py2-none-any.whl	295kB	✖
gtextfsm-0.2.1.tar.gz	21kB	✖
netaddr-0.7.17-py2.py3-none-any.whl	1.53MB	✖
pyeapi-0.8.2.tar.gz	134kB	✖

Upload

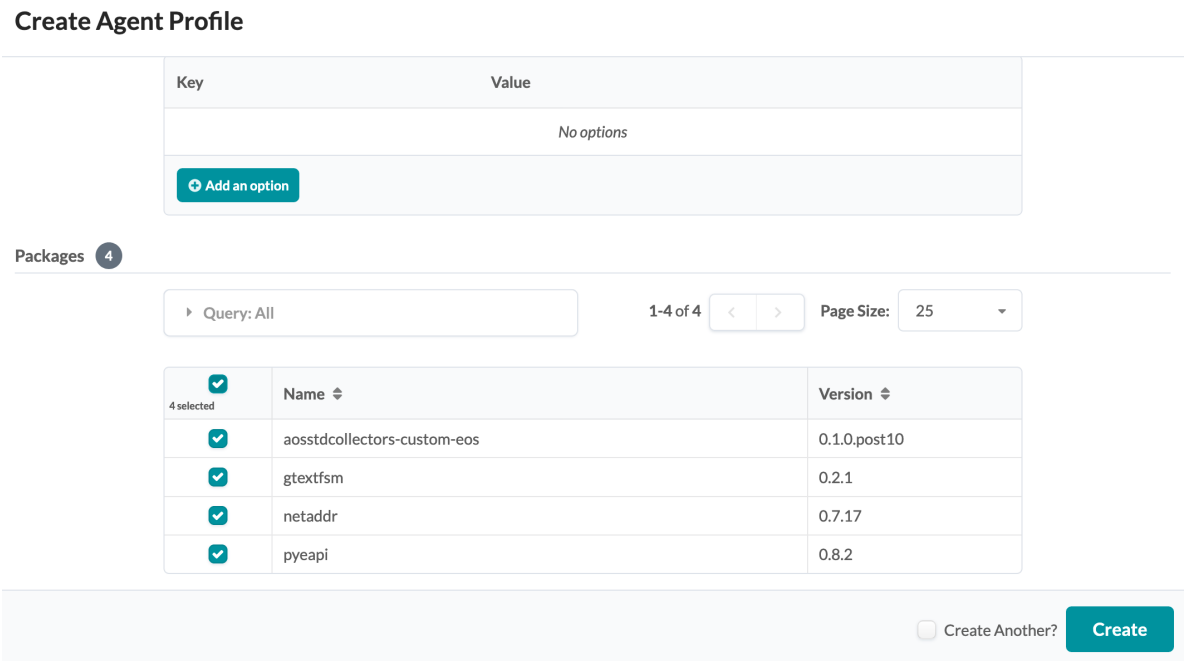
6. Click **Upload** to upload the packages to the Apstra server, then close the dialog to return to the summary table view.

Create Agent Profiles

With agent profiles you can specify packages once in the profile, then apply the profile to multiple agents at the same time. Let's create a profile that contains all four packages. (Remember, if your environment has Internet access, you only need to include the custom collector package.)

1. From the left navigation menu, navigate to **Devices > System Agents > Agent Profiles** and click **Create Agent Profile**.
2. For this example, select **EOS** from the platform drop-down list.

3. In the **Packages** section, select the four uploaded packages to associate them with the agent profile. (If your environment has Internet access, you only need to include the custom collector package.)



4. Click **Create** to create the agent profile and return to the summary table view.

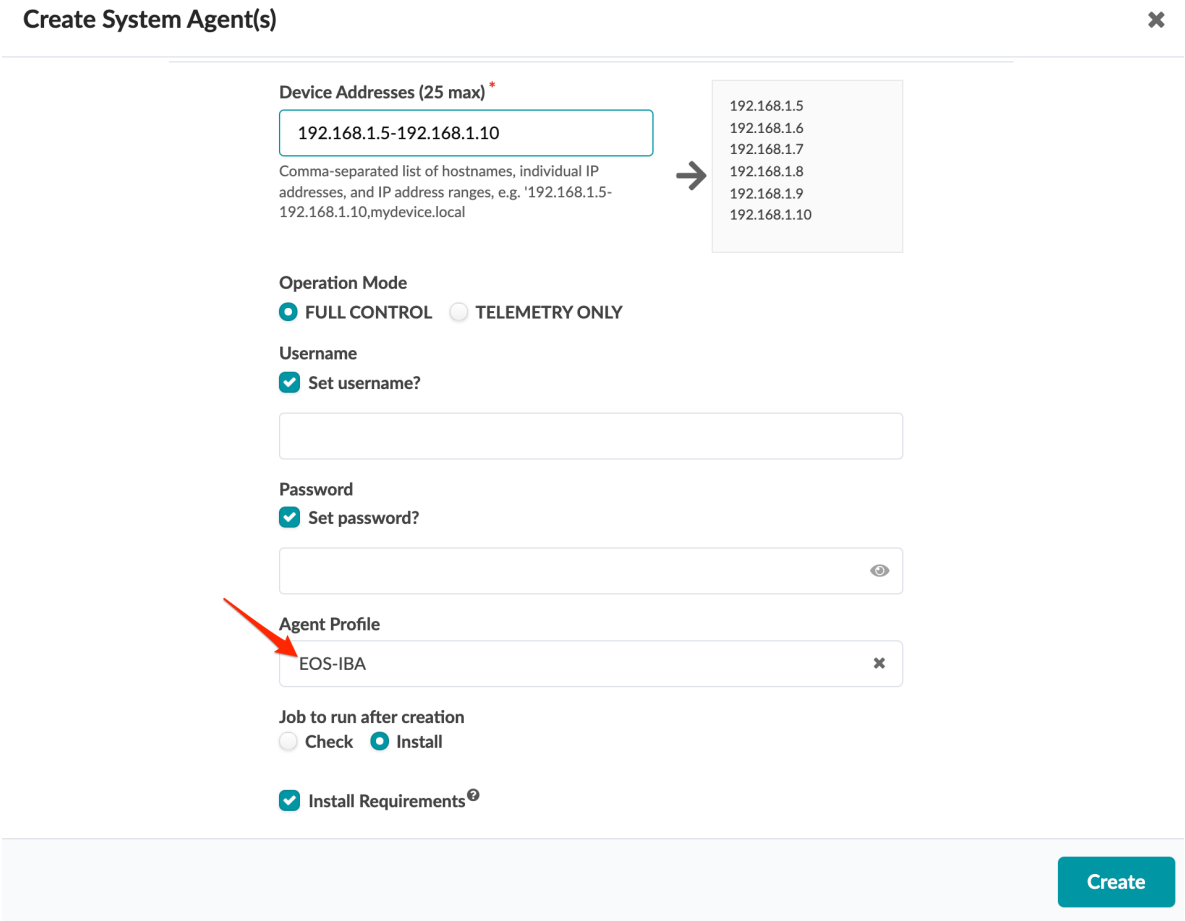
For more information about agent profiles, see "Agent Profiles" on page 696.

Create Agents

Now let's create agents for Arista devices and use the agent profile to associate the packages to them. We recommend that you use agent profiles to associate custom collector packages so you can bulk update agents later, as needed, with a single command.

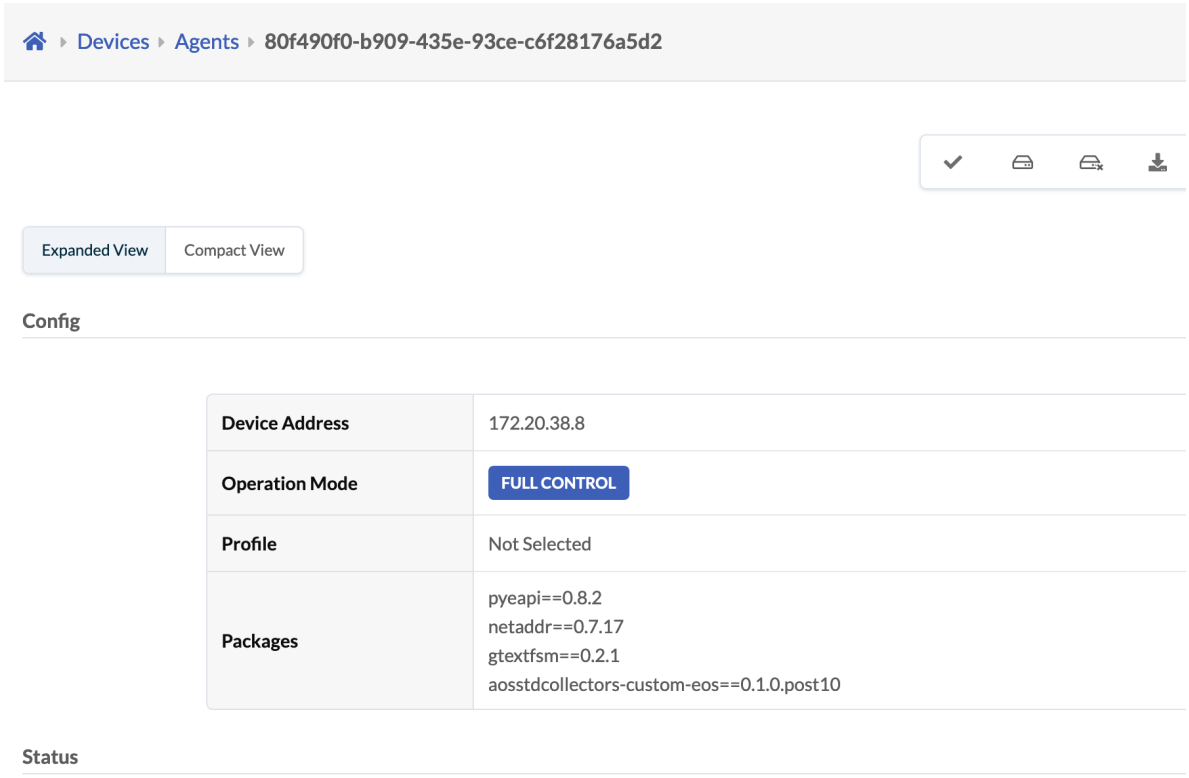
1. From the left navigation menu, navigate to **Devices > System Agents > Agents** and click **Create Onbox Agent(s)**.

2. Enter details for the agent and select the agent profile from the drop-down list as shown in the image below:



3. To verify that packages have been successfully installed on agents, from the left navigation menu, navigate to **Devices > Managed Devices** and click the management IP of the device. Click the **Agent** tab. The **Config** section lists any installed packages. If you manually uploaded the Python packages (netaddr, gtextfsm and pyeapi) they are listed. If the Apstra server has Internet access, they were automatically uploaded and won't be listed here. (To see all packages installed on the device, log in to

the device and check the `/tmp/plugins` folder.)



Update Agents from apstra-cli

As of apstra-cli build 423, you can update agents with a given agent profile, as needed, based on IP/ID or OS type (os_type) (for example, EOS).

To update agents by IP range with a specific agent profile, use the command `system-agents update-profile` as shown in the example below. When setting the `--profile` option, apstra-cli shows available agent profiles. To select, use the up and down arrow keys.

```
apstra-cli> system-agents update-profile --ip 172.20.120.6-11 --profile
                                                                EOS-IBA  EOS
```

For example.

```
apstra-cli> system-agents update-profile --ip 172.20.120.6-11 --profile 692bb0bb-c5e0-4d7e-a70c-
c24b0d5650a8
Successfully updated agent 172.20.120.9 with given profile
```

```
Successfully updated agent 172.20.120.6 with given profile
Successfully updated agent 172.20.120.11 with given profile
Successfully updated agent 172.20.120.7 with given profile
Successfully updated agent 172.20.120.10 with given profile
Successfully updated agent 172.20.120.8 with given profile
apstra-cli>
```

Install IBA Probes

You can install IBA probes using the Apstra GUI, or for non-production environments you can use apstra-cli. For information about how to create or instantiate predefined probes from the GUI, see "Probes Introduction" on page 29 in the Analytics section. This section shows you how to use the apstra-cli utility.

All probes described in this document are included in apstra-cli build 412 and later. Probe `.j2` files may be made available if the probe file is not built into the apstra-cli build.

Some of these probes require an updated service registry. Download the latest Apstra SDK and extract the `json-schemas.tar.gz` file. Copy the file to the `/home/admin` directory of the Apstra server so it is available in the apstra-cli `/mytmp` directory.

```
apstra-cli> service-registry import-from --file /mytmp/json-schemas.tar.gz
Successfully imported service registry entry for interface_details
Successfully imported service registry entry for route_count
Successfully imported service registry entry for multicast_groups
Successfully imported service registry entry for sfp
Successfully imported service registry entry for resource_usage
Successfully imported service registry entry for mlag_domain
Successfully imported service registry entry for stp
Successfully imported service registry entry for vtep_counters
Successfully imported service registry entry for vlan
Successfully imported service registry entry for evpn_type5
Successfully imported service registry entry for ping
Successfully imported service registry entry for vxlan_info
Successfully imported service registry entry for pim_neighbor_count
Successfully imported service registry entry for lldp_details
Successfully imported service registry entry for evpn_type3
Successfully imported service registry entry for multicast_info
Successfully imported service registry entry for bgp_vrf
Successfully imported service registry entry for traceroute
```

```
Successfully imported service registry entry for vrf
Successfully imported service registry entry for table_usage
Successfully imported service registry entry for vxlan_address_table
Successfully imported service registry entry for acl_stats
Successfully imported service registry entry for device_info
Successfully imported service registry entry for power_supply
Successfully imported service registry entry for interface_buffer
Successfully imported service registry entry for pim_rp
Successfully imported service registry entry for anycast_rp
Successfully imported service registry entry for bgp_iba
Successfully imported service registry entry for interface_iba
apstra-cli>
```

To create probes, use the `probe create` apstra-cli command. You'll be prompted for additional options.

```
apstra-cli> probe create
--blueprint           Id of the blueprint
--file                Filename of json file with probe data. Choose from dropdown or specify
custom path
--skip-service-check  [Optional] By default, required telemetry services are checked and enabled
on target
--check-status        [Optional] Wait for probe to become operational. Default: False
--service-interval    When skip-service-check is False and service is not alreadypresent, this
indicates
```

To select the blueprint ID, use `--blueprint` and tab-completion.

```
apstra-cli> probe create --blueprint 67cd936d-c2de-49f8-8708-df465f0cdc68
                            L2 Virtual  two_stage_l3clos
```

To list available probes supplied with apstra-cli, use `--file` and tab-completion. Scroll through the list with the up and down arrow keys.

```
apstra-cli> probe create --blueprint 67cd936d-c2de-49f8-8708-df465f0cdc68 --file
                                                                        evpn.j2
                                                                        sfp.j2

memory_usage_threshold_anomalies.j2

bandwidth_utilization_history.j2
```

```
                                                                power_supply_anomalies.j2

virtual_infra_vlan_mismatch.j2

hardware_vtep_counters_enabled.j2
```

Some probes need additional `Probe template variables`.

```
apstra-cli> probe create --blueprint 67cd936d-c2de-49f8-8708-df465f0cdc68 --file /usr/local/lib/
python2.7/site-packages/aos_cli/resources/probes/memory_usage_threshold_anomalies.j2
--skip-service-check  [Optional] By default, required telemetry services are checked and enabled
on target
--check-status        [Optional] Wait for probe to become operational. Default: False
--service-interval    When skip-service-check is False and service is not alreadypresent, this
indicates
--process             Probe template variable
--os_family           Probe template variable
```

To see installed IBA probes in the blueprint, navigate to **Analytics > Probes**.

Apstra IBA Probes Examples

IN THIS SECTION

The following section describes how to install some of the most interesting probes which are not available by default.

Packet Drops

Packet drop IBA probes detect an abnormal amount of packet drops on device interfaces that the Apstra software manages, based on interface telemetry that device agents collect.

Filename	Description
pkt_discard_anomalies.j2	Detect Fabric interfaces having sustained packet discards

To install the `pkt_discard_anomalies.j2` IBA Probe:

```
apstra-cli> probe create --blueprint 67cd936d-c2de-49f8-8708-df465f0cdc68 --file /usr/local/lib/
python2.7/site-packages/aos_cli/resources/probes/pkt_discard_anomalies.j2
Ensuring needed telemetry services for probe are enabled...
Successfully created probe f472ba21-d60f-44dc-9f5d-8318c8b9c07b in blueprint 67cd936d-
c2de-49f8-8708-df465f0cdc68
apstra-cli>
```

Switch Memory Leak (Arista EOS only)

Switch Memory Leak IBA probes detect abnormal memory leaks in specified processes on devices that the Apstra software manages, based on system telemetry that device agents collect. This probe requires device user credentials set in the device agent configuration that has login and access to the device BASH prompt.

Filename	Description
memory_usage_threshold_anomalies.j2	Detect memory leaks in specified process on all switches in the Fabric
system_memory_usage_threshold_anomalies.j2	Detect switches having potential memory leaks in the Fabric

The `memory_usage_threshold_anomalies.j2` IBA probe requires additional "Probe template variables" for `os_family` and `process`.

```
apstra-cli> probe create --blueprint 67cd936d-c2de-49f8-8708-df465f0cdc68 --file /usr/local/lib/
python2.7/site-packages/aos_cli/resources/probes/memory_usage_threshold_anomalies.j2
  --skip-service-check  [Optional] By default, required telemetry services are checked and
enabled on target
  --check-status        [Optional] Wait for probe to become operational. Default: False
  --service-interval    When skip-service-check is False and service is not alreadypresent, this
indicates
  --process             Probe template variable
  --os_family           Probe template variable
```

The only option for `os_family` is `eos` for Arista EOS. The (2) options for `process` are `edac-poller` and `fastcapi` or `configagent`.

```
apstra-cli> probe create --blueprint 67cd936d-c2de-49f8-8708-df465f0cdc68 --file /usr/local/lib/
python2.7/site-packages/aos_cli/resources/probes/memory_usage_threshold_anomalies.j2 --os_family
eos --process fastcapi
Ensuring needed telemetry services for probe are enabled...
Enabled service resource_usage on device l2-virtual-002-leaf1:172.20.60.11
Enabled service resource_usage on device l2-virtual-001-leaf1:172.20.60.9
Enabled service resource_usage on device spine2:172.20.60.8
Enabled service resource_usage on device spine1:172.20.60.6
Enabled service resource_usage on device l2-virtual-003-leaf1:172.20.60.10
Enabled service resource_usage on device l2-virtual-004-leaf1:172.20.60.7
Successfully created probe 6a258d83-1053-42ad-935c-0550cc500b7d in blueprint 67cd936d-
c2de-49f8-8708-df465f0cdc68
apstra-cli>
```

```
apstra-cli> probe create --blueprint rack-based-blueprint-10990707 --file /usr/local/lib/
python2.7/site-packages/aos_cli/resources/probes/memory_usage_threshold_anomalies.j2 --os_family
eos --process configagent
Ensuring needed telemetry services for probe are enabled...
Successfully created probe ed2c6be1-b4b1-4e1b-bd07-da431e89eeec in blueprint rack-based-
blueprint-10990707
apstra-cli>
```

NOTE: "FastCapi" as service process is valid only for EOS version 4.18. For the newer version of EOS, for example 4.20 and later only ConfigAgent is valid. Take extra care that service name is in lowercase during probe creation. So it should be `configagent` instead of `ConfigAgent`.

To install the IBA probe for a second process, repeat the `probe create` command for the other process.

You can edit the IBA probe name to include the process name.

To install the `system_memory_usage_threshold_anomalies.j2` IBA probe:

```
apstra-cli> probe create --blueprint 67cd936d-c2de-49f8-8708-df465f0cdc68 --file /usr/local/lib/
python2.7/site-packages/aos_cli/resources/probes/system_memory_usage_threshold_anomalies.j2
Ensuring needed telemetry services for probe are enabled...
Successfully created probe a669ccf8-cba7-414b-ad46-a7d4b4ca3928 in blueprint 67cd936d-
```

```
c2de-49f8-8708-df465f0cdc68
apstra-cli>
```

Fault Tolerance

These (2) probes require apstra-cli build 430 or later.

Filename	Description
spine_fault_tolerance.j2	Find out if failure of given number of spines in the fabric is going to be tolerated. Raise anomaly if total traffic on all spines is more than the available spine capacity, with the specified number of spine failures.
lag_link_fault_tolerance.j2	Find out if failure of one link in a server LAG is going to be tolerated. Monitors total traffic in each LAG against total available capacity of the bond, with one link failure. Raise anomaly for racks with more than 50% of such overused bonds, sustained for certain duration.

To install the `spine_fault_tolerance.j2` IBA Probe:

```
apstra-cli> probe create --blueprint bf7a322c-ee3a-4dcf-aa20-df0560f538da --file /usr/local/lib/
python2.7/site-packages/aos_cli/resources/probes/spine_fault_tolerance.j2 --
number_of_faulty_spines_to_be_tolerated 1
Successfully created probe 0f0e9bf7-d9b3-43d7-906e-a9f0675e68f2 in blueprint bf7a322c-ee3a-4dcf-
aa20-df0560f538da
apstra-cli>
```

NOTE: `number_of_faulty_spines_to_be_tolerated` must be specified.

To install the `lag_link_fault_tolerance.j2` IBA Probe:

```
apstra-cli> probe create --blueprint bf7a322c-ee3a-4dcf-aa20-df0560f538da --file /usr/local/lib/
python2.7/site-packages/aos_cli/resources/probes/lag_link_fault_tolerance.j2
Successfully created probe 45ce5fe8-555f-41a9-b0ae-267125669d3f in blueprint bf7a322c-ee3a-4dcf-
aa20-df0560f538da
apstra-cli>
```

AOSOM-Streaming Guide

IN THIS SECTION

AOSOM-Streaming Overview

IN THIS SECTION

NOTE: AOSOM streaming is demonstration software, not intended for production environments.

You can configure Apstra to generate Google Protocol Buffer (protobuf) streams for counter data (perfmon), alerts, and events. Each data type is sent to a streaming receiver over its own TCP socket. Even if all three data types are configured for the same streaming receiver, three connections are created between the Apstra server and the streaming receiver. This also allows for all three types to be sent to three different streaming receivers. You can choose from the many open-source projects, or develop your own solutions to capture, store and inspect the protobuf data. Apstra has developed a project available on GitHub called AOSOM-Streaming to demonstrate how this can be achieved using several open-source components. The AOSOM-Streaming project is meant to help you understand how you can consume the AOS protobuf stream. It is for demonstration purposes only, except for the Apstra

Telegraf input plugin. Apstra software fully supports this plug-in for use as part of your streaming telemetry solution.

The Aosom Streaming project provides a packaged solution to collect and visualize telemetry streaming information coming from an Apstra server. This provides a web interface experience and example queries to handle alerts, counters, and Apstra events. This open-source project officially lives on Github at https://github.com/Apstra/aosom-streaming.

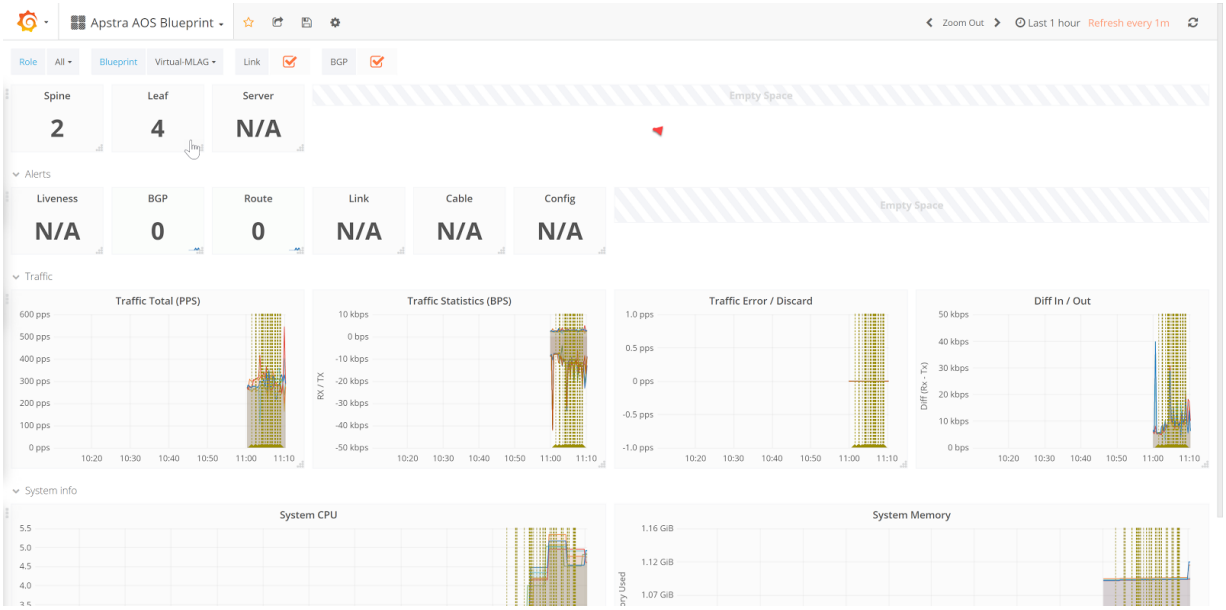
The packaged solution includes:

- A graphical Interface based on Grafana (port 3000)
- Prometheus for Counters and Alerts (port 9090)
- Influxdb for Events (port 8086)
- 2 Collectors, one for each database based on Telegraf.

Grafana

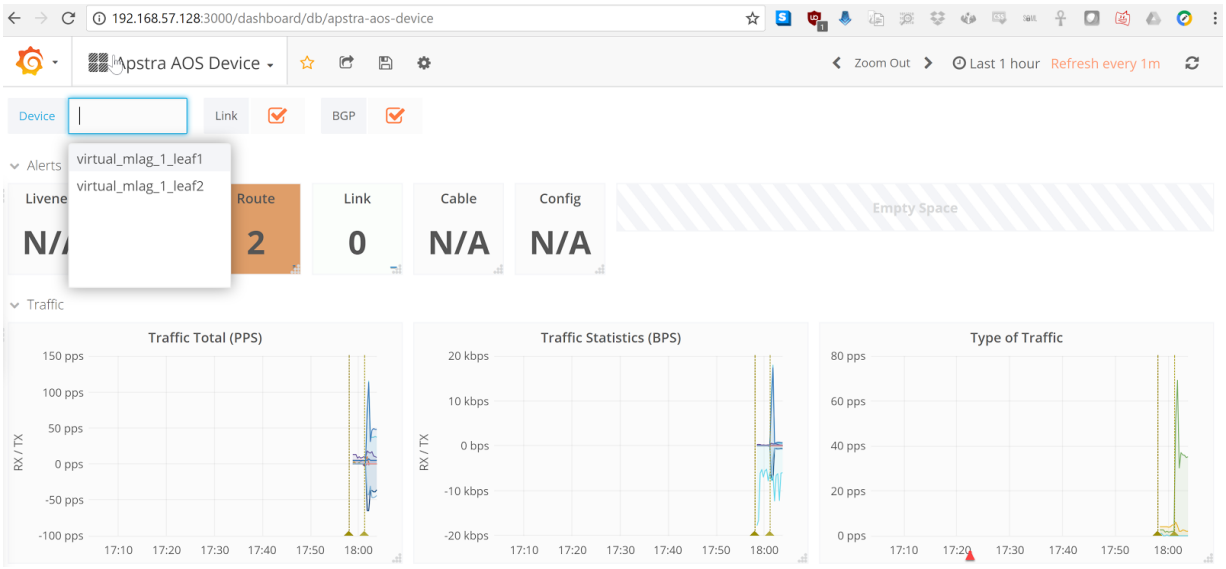
From a web browser enter the URL **http://<aosom-streaming>:3000** and enter username **admin** (default) and password **admin** (default).

The grafana GUI includes two main sections (top left). **Apstra AOS Blueprint** describes overall telemetry alerts and traffic throughput, as well as individual devices for interface telemetry. Blueprints are learned automatically using the Apstra 'telegraf' Docker container; no further configuration is necessary.



In the screenshot above, we can observe traffic in the demo Apstra environment, and aggregate CPU, traffic, and errors.

To filter telemetry events based on specific and individual devices, change the dashboard at the top to **Apstra AOS Device**. Here we can observe there are two active route anomalies in the blueprint, and Apstra has received telemetry for two leaf switches.



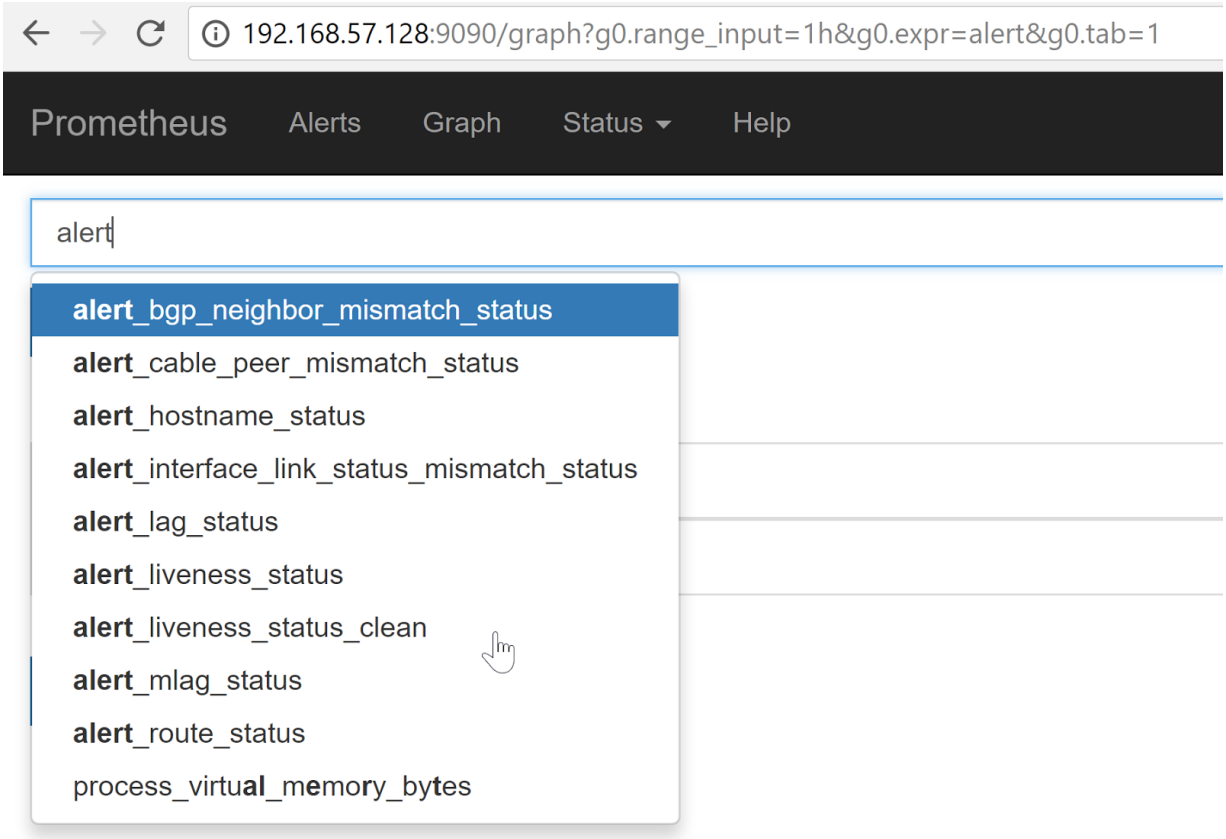
Scroll down to view device statistics such as CPU and Memory:

Prometheus

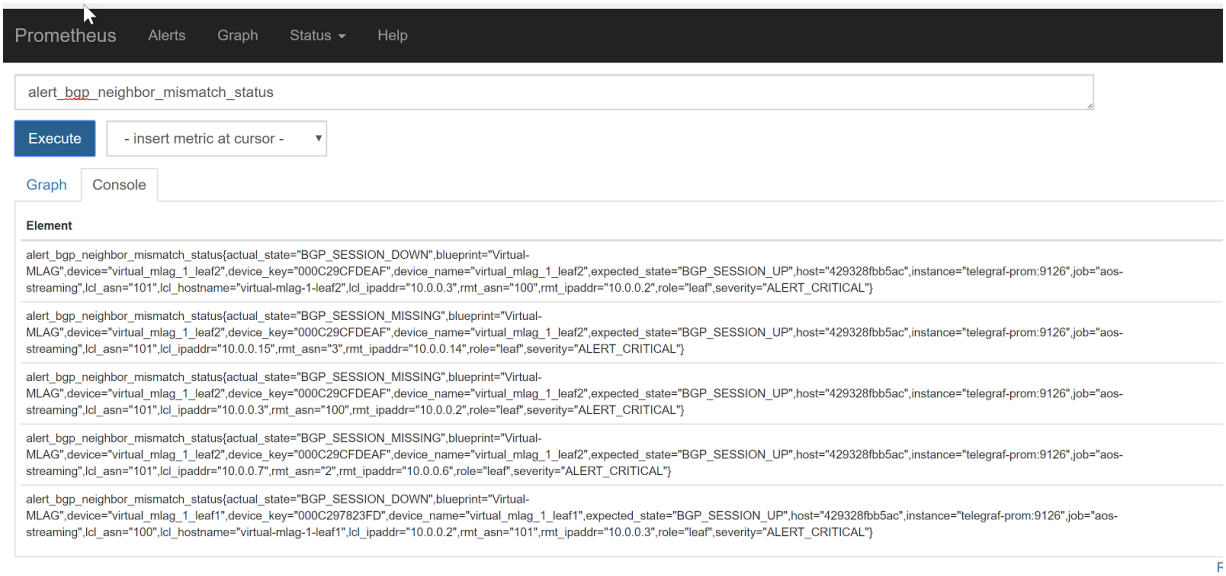
Prometheus is used for alerts and device telemetry counter storage in the Aosom-streaming appliance. From a web browser enter the URL **http://<aosom-streaming>:9090** to access the Prometheus GUI.

When incoming events appear, Apstra dynamically builds each of the queries. To see example query names, begin typing under ‘execute’. Starting with ‘alert’ it tab-completes available alerts that

prometheus has received from Apstra.



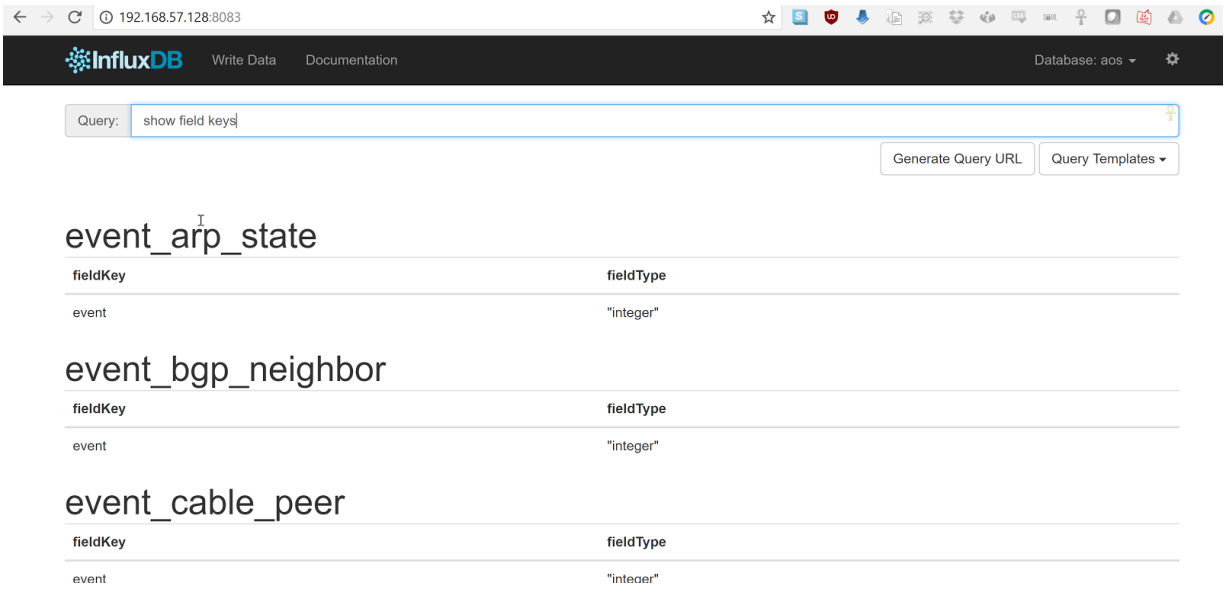
Here is an example of BGP Neighbors being offline.



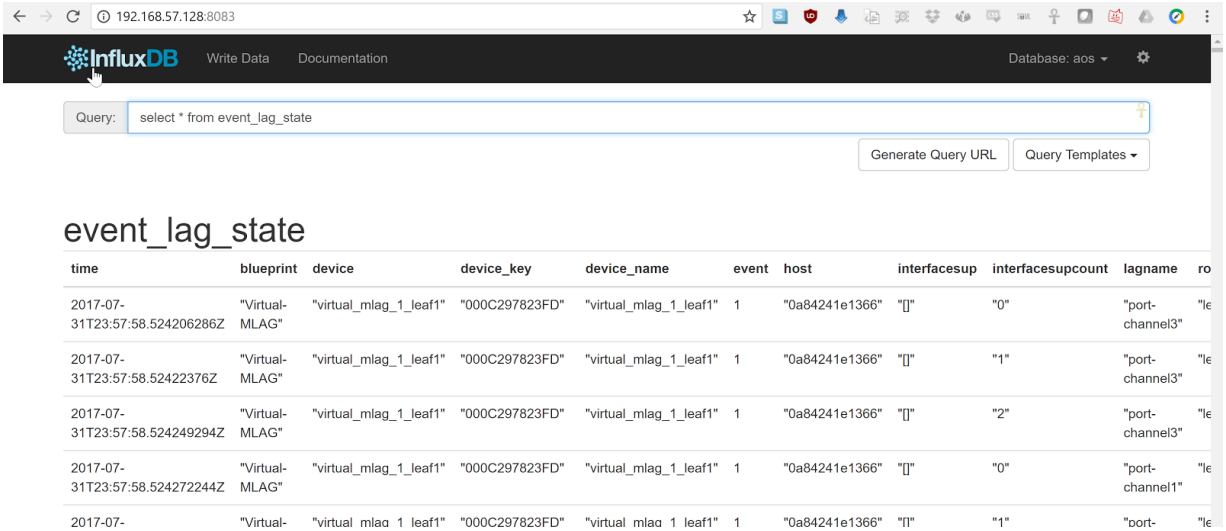
InfluxDB

InfluxDB is used to store Apstra events from telemetry streaming. From a web browser enter the URL **http://<aosom-streaming>:8083** to access InfluxDB.

We can show the available influxdb keys with queries, such as **show field keys** or **show measurements**.



Once we know a measurement, we can view the data and keys with **select * from <measurement>** -- In this case, we'll capture the LAG interface status.



NOTE: Developing an influx-db application is beyond the scope of this documentation.

Configure Aosom-Streaming

To configure telemetry streaming as part of this project, you'll edit `variables.env`, run the `make start` file and restart the containers. No Apstra server configuration is required. Documentation for starting, stopping, and clearing data is available at https://github.com/Apstra/aosom-streaming

The telegraf project connects to the Apstra API and posts an IP:Port that Apstra uses to stream realtime telemetry data back to.

1. Copy `variables.default` to `variables.env`:

```
aosom@ubuntu:~/aosom-streaming$ cp variables.default variables.env
```

2. Configure `variables.env`.

```
AOS_SERVER=192.168.57.250
LOCAL_IP=192.168.57.128

INPUT_PORT_INFLUX=4444
INPUT_PORT_PROM=6666
```

```
AOS_LOGIN=admin
AOS_PASSWORD=admin
AOS_PORT=443

GRAFANA_LOGIN=admin
GRAFANA_PASSWORD=admin
```

- `AOS_SERVER` - the IP address of the Apstra server that sends telemetry data to the aosom-streaming server.
- `LOCAL_IP` - the IP address assigned to ens33 (first ethernet interface). In this case, it is learned via DHCP on this VM. See `ip addr show dev ens33`. GRAFANA configuration options to specify the username and password for the grafana web interface.
- `AOS_LOGIN`, `AOS_PASSWORD`, `AOS_PORT` - You can customize username, port and password information.

3. Run the command `make start` to set up the project, or if you're making configuration changes, run `make update`.

```
aosom@ubuntu:~/aosom-streaming$ make start
-- Start all components --
Creating network "aosomstreaming_default" with the default driver
Creating volume "aosomstreaming_grafana_data_2" with default driver
Pulling telegraf-influx (apstra/telegraf:1.2)...
1.2: Pulling from apstra/telegraf
00d19003217b: Pull complete
72dd23d7de04: Pull complete
cf6581f43cce: Pull complete
Digest: sha256:1539d4b84618abb44bdffb1e0a27399a7272814be36535f4a7dfa04661d6e5f6
Status: Downloaded newer image for apstra/telegraf:1.2
Pulling prometheus (prom/prometheus:v1.5.2)...
v1.5.2: Pulling from prom/prometheus
557a0c95bfcd: Pull complete
a3ed95caeb02: Pull complete
caf4d0cf9832: Pull complete
ee054001e2db: Pull complete
b95bf6c4c81b: Pull complete
86503a6ba368: Pull complete
ff27c7b0b50e: Pull complete
534e30a17a42: Pull complete
475d41733562: Pull complete
Digest: sha256:e049c086e35c0426389cd2450ef193f6c18b3d0065b97e5f203fdb254716fa1c
Status: Downloaded newer image for prom/prometheus:v1.5.2
```

```
Pulling influxdb (influxdb:1.1.1-alpine)...
1.1.1-alpine: Pulling from library/influxdb
0a8490d0dfd3: Pull complete
5f0fd352f87d: Pull complete
873718bcf8aa: Pull complete
3fbaf3e4140e: Pull complete
Digest: sha256:e0184202151b2abb9ceee79e6523d9492fc3c632324eb6f7bf1a672dd130a3bb
Status: Downloaded newer image for influxdb:1.1.1-alpine
Pulling grafana (grafana/grafana:4.1.2)...
4.1.2: Pulling from grafana/grafana
43c265008fae: Pull complete
c2ab838d4052: Pull complete
e8a816c8f505: Pull complete
Digest: sha256:05d925bd64cd3f9d6f56a4353774ccec588586579ab738f933cd002b7f96aca3
Status: Downloaded newer image for grafana/grafana:4.1.2
Creating aosomstreaming_telegraf-influx_1
Creating aosomstreaming_prometheus_1
Creating aosomstreaming_telegraf-prom_1
Creating aosomstreaming_influxdb_1
Creating aosomstreaming_grafana_1
```

Reconfigure Aosom-streaming after Apstra Server Upgrade

After you upgrade the Apstra server you must reconfigure to ensure a proper streaming connection.

1. If you upgraded the Apstra server onto a different VM (or if the server IP address is different for any reason), update the `variables.env` file with the new Apstra IP address.
2. Run the `docker ps` command to verify that the current **Telegraf** container image matches the proper version for the new Apstra release.

   ```
   admin@aeon-ztps:~$ docker ps
   CONTAINER ID IMAGE
   4edf204e7be9 apstra/telegraf:latest
   ```

   You can check the different Telegraf versions in the Apstra Docker Hub.
3. If required, modify the `docker-compose.yml` file and point to the correct Docker image.
4. Run the command `docker-compose up -d` to restart the service.
5. Run the `docker ps` command to verify that the container is running with the new image.

NOTE: For assistance regarding which version to install or if you have any questions about the procedure, contact "Juniper Support" on page 1374.

Build Aosom-Streaming VM (Optional)

IN THIS SECTION

You can build your own Aosom-streaming VM, which is a Docker container. This steps show you how to set up a basic Docker server.

Install Ubuntu 16.04.2

Download the Ubuntu 16.04.2 ISO and provision a new VM. The default username is **aosom** and the password is **admin**.

For larger blueprints, we recommend changing RAM to at least 8GB and CPU to at least 2 vCPU. More disk space may also be required.

Resource	Quantity
RAM	8GB
CPU	2 vCPU
Network	1 vNIC

Install Packages

Install required packages, based on Ubuntu 16.04.2.

```
apt-get update
```

Update the system to ensure all packages are up to date.

```
apt-get install docker docker-compose git make curl openssh-server
```

```
aosom@ubuntu:~$ sudo apt-get install docker docker-compose git make curl openssh-server
[sudo] password for aosom:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  bridge-utils cgroupfs-mount containerd dns-root-data dnsmasq-base docker.io
  git-man liberror-perl libnetfilter-conntrack3 libperl5.22 libpython-stdlib
  libpython2.7-minimal libpython2.7-stdlib libyaml-0-2 patch perl
  perl-modules-5.22 python python-backports.ssl-match-hostname
  python-cached-property python-cffi-backend python-chardet
  python-cryptography python-docker python-dockerpty python-docopt
  python-enum34 python-funcsigs python-functools32 python-idna
  python-ipaddress python-jsonschema python-minimal python-mock
  python-ndg-httpsclient python-openssl python-pbr python-pkg-resources
  python-pyasn1 python-requests python-six python-texttable python-urllib3
  python-websocket python-yaml python2.7 python2.7-minimal rename runc
  ubuntu-fan xz-utils
Suggested packages:
  mountall aufs-tools btrfs-tools debootstrap docker-doc rinse zfs-fuse
  | zfsutils git-daemon-run | git-daemon-sysvinit git-doc git-el git-email
  git-gui gitk gitweb git-arch git-cvs git-mediawiki git-svn diffutils-doc
  perl-doc libterm-readline-gnu-perl | libterm-readline-perl-perl make
  python-doc python-tk python-cryptography-doc python-cryptography-vectors
  python-enum34-doc python-funcsigs-doc python-mock-doc python-openssl-doc
  python-openssl-dbg python-setuptools doc-base python-ntlm python2.7-doc
  binutils binfmt-support make
The following NEW packages will be installed:
  bridge-utils cgroupfs-mount containerd dns-root-data dnsmasq-base docker
  docker-compose docker.io git git-man liberror-perl libnetfilter-conntrack3
```

```
  libperl5.22 libpython-stdlib libpython2.7-minimal libpython2.7-stdlib
  libyaml-0-2 patch perl perl-modules-5.22 python
  python-backports.ssl-match-hostname python-cached-property
  python-cffi-backend python-chardet python-cryptography python-docker
  python-dockerpty python-docopt python-enum34 python-funcsigs
  python-functools32 python-idna python-ipaddress python-jsonschema
  python-minimal python-mock python-ndg-httpsclient python-openssl python-pbr
  python-pkg-resources python-pyasn1 python-requests python-six
  python-texttable python-urllib3 python-websocket python-yaml python2.7
  python2.7-minimal rename runc ubuntu-fan xz-utils make
0 upgraded, 54 newly installed, 0 to remove and 3 not upgraded.
Need to get 32.4 MB of archives.
After this operation, 174 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

Add the aosom user to the Docker group. This allows ‘aosom’ to make Docker configuration changes without having to escalate to sudo.

```
aosom@ubuntu:~/aosom-streaming$ sudo usermod -aG docker aosom
Log out and log back in again for ‘aosom’ user to be properly added to the group.
```

Copy the Aosom-streaming Docker containers over with ‘git clone’.

```
aosom@ubuntu:~$ git clone https://github.com/Apstra/aosom-streaming.git
Cloning into 'aosom-streaming'...
remote: Counting objects: 303, done.
remote: Total 303 (delta 0), reused 0 (delta 0), pack-reused 303
Receiving objects: 100% (303/303), 64.10 KiB | 0 bytes/s, done.
Resolving deltas: 100% (176/176), done.
Checking connectivity... done.
aosom@ubuntu:~$
```

Set Container Restart Policy

The AOSOM-Streaming package does not set the Docker restart policy; this is up to your orchestration toolchain. Open aosom-streaming/docker-compose.yml and add `restart: always` to each of the service directives. This ensures that Docker containers are online after a service reboot.

```
git diff docker-compose.yml
```

```
aosom@ubuntu:~/aosom-streaming$ git diff docker-compose.yml
diff --git a/docker-compose.yml b/docker-compose.yml
index 799d4c5..0d0fcc2 100644
--- a/docker-compose.yml
+++ b/docker-compose.yml
@@ -16,6 +16,7 @@ services:
       - prometheus
     ports:
       - "3000:3000"
+    restart: always

 # ------------------------------------------------------------------------
 # Prometheus -
@@ -30,6 +31,7 @@ services:
         - '-config.file=/etc/prometheus/prometheus.yml'
     ports:
         - '9090:9090'
+    restart: always

 # ------------------------------------------------------------------------
 # influxdb
@@ -43,6 +45,7 @@ services:
     ports:
      - "8083:8083"
      - "8086:8086"
+    restart: always

 # ------------------------------------------------------------------------
 # Telegraf - Prom
@@ -57,6 +60,7 @@ services:
       - /etc/localtime:/etc/localtime
     ports:
         - '6666:6666'
```

```
+    restart: always

 # ------------------------------------------------------------------------
 # Telegraf - Influx
@@ -71,3 +75,4 @@ services:
       - /etc/localtime:/etc/localtime
     ports:
         - '4444:4444'
+    restart: always
```

Set up `variables.env` and start container per Aosom-Streaming application setup section.

Change System Hostname

Modify `/etc/hostname` to aosom, and change the loopback IP in `/etc/hosts` to aosom from ubuntu.

Troubleshooting

IN THIS SECTION

While most troubleshooting information is included in the Github main page at https://github.com/Apstra/aosom-streaming, you can run some simple commands to make sure the environment is healthy.

Check for Logs from Apstra to Aosom-streaming

Run Docker logs aosomstreaming_telegraf-influx_1

You should see a blueprint ID, and some influxdb 'write' events when telemetry events occur on AOS - BGP, liveness, config deviation, etc.

```
GetBlueprints() - Id 0033cf3f-41ed-4ddc-91f5-ea68318fba9b
2017-07-31T23:59:13Z D! Finished to Refresh Data, will sleep for 20 sec
2017-07-31T23:59:15Z D! Output [influxdb] buffer fullness: 11 / 10000 metrics.
```

```
2017-07-31T23:59:15Z D! Output [influxdb] wrote batch of 11 metrics in 5.612057ms
2017-07-31T23:59:20Z D! Output [influxdb] buffer fullness: 4 / 10000 metrics.
2017-07-31T23:59:20Z D! Output [influxdb] wrote batch of 4 metrics in 5.349171ms
2017-07-31T23:59:25Z D! Output [influxdb] buffer fullness: 11 / 10000 metrics.
2017-07-31T23:59:25Z D! Output [influxdb] wrote batch of 11 metrics in 4.68295ms
2017-07-31T23:59:30Z D! Output [influxdb] buffer fullness: 4 / 10000 metrics.
2017-07-31T23:59:30Z D! Output [influxdb] wrote batch of 4 metrics in 5.007029ms
GetBlueprints() - Id 0033cf3f-41ed-4ddc-91f5-ea68318fba9b
2017-07-31T23:59:33Z D! Finished to Refresh Data, will sleep for 20 sec
```

Ensure Containers are Running

To see and ensure that all the expected containers are running, run `docker ps`:

```
aosom@ubuntu:~/aosom-streaming$ docker ps
CONTAINER ID        IMAGE                    COMMAND                  CREATED
STATUS              PORTS                                           NAMES
e03d003a2ef9        grafana/grafana:4.1.2    "/run.sh"                3 minutes ago       Up 3
minutes        0.0.0.0:3000->3000/tcp                           aosomstreaming_grafana_1
3042d45f1107        prom/prometheus:v1.5.2   "/bin/prometheus -con"   3 minutes ago       Up 3
minutes        0.0.0.0:9090->9090/tcp                           aosomstreaming_prometheus_1
429328fbb5ac        apstra/telegraf:1.2      "telegraf -debug"        3 minutes ago       Up 3
minutes        0.0.0.0:6666->6666/tcp                           aosomstreaming_telegraf-prom_1
0a84241e1366        apstra/telegraf:1.2      "telegraf -debug"        3 minutes ago       Up 3
minutes        0.0.0.0:4444->4444/tcp                           aosomstreaming_telegraf-influx_1
f4d2deb0e428        influxdb:1.1.1-alpine    "/entrypoint.sh influ"   3 minutes ago       Up 3
minutes        0.0.0.0:8083->8083/tcp, 0.0.0.0:8086->8086/tcp   aosomstreaming_influxdb_1
```

PART 21

References

CHAPTER 52

Feature Matrix

IN THIS CHAPTER

Apstra 5.0.0 Feature Matrix

IN THIS SECTION

Fabric Roles

Fabric Roles	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
Access Switch	No	No	No	Yes	No
Non-EVPN-VXLAN Leaf (IP forwarder only)	Yes	Yes	Yes	Yes	Yes
EVPN-VXLAN Leaf	Yes	Yes	Yes	Yes	Yes
Spine or Superspine	Yes	Yes	Yes	Yes	Yes

Fabric Connectivity

Fabric Connectivity	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
3-stage Clos	Yes	Yes	Yes	Yes	Yes
5-stage Clos	Yes	Yes	Yes	Yes	Yes
Collapsed Fabric	No	No	Tech Preview (See Note)	Yes	Yes
Freeform	No	No	No	Yes	Yes
IP only Fabric (non-EVPN/ VXLAN overlay)	Yes	Yes	Yes	Yes	Yes
EVPN-VXLAN fabric	Yes	Yes	Yes	Yes	Yes

(Continued)

Fabric Connectivity	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
IPv6 Fabric RFC-5549 (default VRF, non EVPN)	Yes	Yes	Yes	No	No
IPv4 Fabric (default VRF, non EVPN)	Yes	Yes	Yes	Yes	Yes
IPv4 Fabric + IPv4 Overlay (VTEP) + IPv4 and/or IPv6 Virtual Networks	Yes	Yes	Yes	Yes	Yes
IPv6 Overlay (VTEP)	No	No	No	No	No
IPv4 and IPv6 Dual Stack Fabric + IPv4 Overlay (VTEP) + IPv4 and/or IPv6 Virtual Networks	Yes	Yes	Yes	Yes	Yes

Device Management

Device Management	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
Onbox agent	Yes	Yes	Yes	Not Possible	Yes*
Offbox agent	Yes	Yes	No	Yes	Yes

(Continued)

Device Management	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
Custom Telemetry Collector (GUI-based)	No	No	No	Yes	Yes
Apstra ZTP GUI	Yes	Yes	Yes	Yes	Yes
Device OS upgrade	Yes	Yes	Yes	Yes	Yes
Traffic draining (maintenance mode) for spines/ superspines	Yes	Yes	Yes	Yes	Yes
Traffic draining (leaf devices)	Yes	Yes	Yes	Yes	Yes

* The minimum release version for Junos OS Evolved switches on onbox agents is 22.4.R3.

Connectivity (from Leaf Layer)

Connectivity (from Leaf Layer)	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
LAG	Yes	Yes	Yes	Yes	Yes
MLAG/vPC	Yes	Yes	Yes	Not possible	Not possible
EVPN ESI (with LACP) for VXLAN Virtual Networks only	No	No	Yes	Yes	Yes

(Continued)

Connectivity (from Leaf Layer)	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
VLANs Virtual Networks	Yes	Yes	Yes	Yes	Yes
Static VXLAN Virtual Networks	Yes	Yes	Not possible	No	No
EVPN VXLAN Virtual Networks	Yes	Yes	Yes	Yes	Yes
IPv4 DHCP relay	Yes	Yes	Yes	Yes	Yes
IPv6 DHCP relay	Yes	Yes	Yes	Yes	Yes
EVPN DCI: Over the TOP	Yes	Yes	Yes	Yes	Yes
EVPN DCI: Integrated Interconnect	No	No	Not possible	Yes	Yes
802.1x	Yes	No	No	No	No
Security Policies (L3 ACLs)	Yes	Yes	No	Yes	Yes

Connectivity (from Access Layer)

Connectivity (from Access Layer)	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
LAG	N/A	N/A	Tech Preview (See Note)	Yes	N/A
ESI LAG	N/A	N/A	Tech Preview (See Note)	Yes	N/A

NOTE: This has been classified as a Juniper Apstra Technology Preview feature. These features are "as is" and voluntary use. Juniper Support will attempt to resolve any issues that customers experience when using these features and create bug reports on behalf of support cases. However, Juniper may not provide comprehensive support services to Tech Preview features. For additional information, refer to the "Juniper Apstra Technology Previews" on page 1781 page or contact "Juniper Support" on page 1374 .

Routing Policies

Routing Policies	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
Import all routes or default route or extra routes only	Yes	Yes	Yes	Yes	Yes
Export loopback, link and VN IP. Export extra routes	Yes	Yes	Yes	Yes	Yes
Export aggregate prefixes	Yes	Yes	Yes	Yes	Yes

(Continued)

Routing Policies	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
Export L3 server link subnets	Yes	Yes	Yes	Yes	Yes
Route target import/export policies	Yes	Yes	Yes	Yes	Yes

Miscellaneous

Miscellaneous	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
Configlets	Yes	Yes	Yes	Yes	Yes
FFE: add racks/add links/ change speed, etc.	Yes	Yes	Yes	Yes	Yes
Mixed leaf/ spine link speed	Yes	Yes	Yes	Yes	Yes

Virtual Network CT Type

Virtual Network CT Type	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
Single Virtual Network	Yes	Yes	Yes	Yes	Yes
Multiple Virtual Network	Yes	Yes	Yes	Yes	Yes

(Continued)

Virtual Network CT Type	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
VLAN (default VRF, non-VXLAN)	Yes	Yes	Yes	Yes	Yes

IP Link CT Type

IP Link CT Type	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
L3 Sub-interface on non-LAG physical interface (untagged/vlan tagged, default/non-default RZ, IPv4)	Yes	Yes	Yes	Yes	Yes
L3 Sub-interface on non-LAG physical interface (untagged/vlan tagged, default/non-default RZ, IPv6)	Yes	Yes	Yes	Yes	Yes
L3 Sub-interface on LAG interface (untagged/vlan tagged, default/non-default RZ, IPv4)	Yes	Yes	Yes	Yes	Yes

(Continued)

IP Link CT Type	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
L3 Sub-interface on LAG interface (untagged/vlan tagged, default/non-default RZ, IPv6)	Yes	Yes	Yes	Yes	Yes
L3 Sub-interface on LAG interface (untagged/vlan tagged, default RZ, IPv4) - spine/sspine	Yes	Yes	Yes	Yes	Yes
L3 Sub-interface on LAG interface (untagged/vlan tagged, default RZ, IPv6) - spine/sspine	Yes	Yes	Yes	Yes	Yes

Static Route CT Type

Static Route CT Type	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
Static Route (IPv4) applied on L3 Sub-interface	Yes	Yes	Yes	Yes	Yes

(Continued)

Static Route CT Type	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
Static Route (IPv6) applied on L3 Sub-interface	Yes	Yes	Yes	Yes	Yes
Static Route (IPv4) applied on SVI	Yes	Yes	Yes	Yes	Yes
Static Route (IPv6) applied on SVI	Yes	Yes	Yes	Yes	Yes
Static Route with Share IP Endpoint Enabled (IPv4)	Yes	Yes	Yes	Yes	Yes
Static Route with Share IP Endpoint Enabled (IPv6)	Yes	Yes	Yes	Yes	Yes

Custom Static Route CT Type

Custom Static Route CT Type	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
Custom Static Route (IPv4, default/non-default RZ)	Yes	Yes	Yes	Yes	Yes
Custom Static Route (IPv6, default/non-default RZ)	Yes	Yes	Yes	Yes	Yes

BGP to Generic CT Type

BGP to Generic CT Type	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
BGP session on L3 Sub-interface towards generic (IPv4, default/ non-default RZ)	Yes	Yes	Yes	Yes	Yes
BGP session on L3 Sub-interface towards generic (IPv6, default/ non-default RZ)	Yes	Yes	Yes	Yes	Yes
BGP session on SVI towards generic (IPv4, default RZ)	Yes	Yes	Yes	Yes	Yes
BGP session on SVI towards generic (IPv4, non-default RZ)	Yes	Yes	Yes	Yes	Yes
BGP session on SVI towards generic (IPv6, non-default RZ)	Yes	Yes	Yes	Yes	Yes
BGP session on SVI towards generic (IPv6, default RZ)	Yes	Yes	Yes	Yes	Yes

(Continued)

BGP to Generic CT Type	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
BGP session on SVI (mlag/esi) towards dual-homed generic using secondary IPs (IPv4, default VRF)	Yes	Yes	Not possible	Not possible	Not possible
BGP session on SVI (mlag/esi) towards dual-homed generic using secondary IPs (IPv4, non-default VRF)	Yes	Yes	Not possible	Yes	Yes
BGP session on SVI (mlag/esi) towards dual-homed generic using secondary IPs (IPv6, default VRF)	Yes	Yes	Not possible	Not possible	Not possible
BGP session on SVI (mlag/esi) towards dual-homed generic using secondary IPs (IPv6, non-default VRF)	Yes	Yes	Not possible	Yes	Yes
BGP session to generic with Share IP Endpoint Enabled (IPv4)	Yes	Yes	Yes	Yes	Yes

(Continued)

BGP to Generic CT Type	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
BGP session to generic with Share IP Endpoint Enabled (IPv6)	Yes	Yes	Yes	Yes	Yes
BGP session to generic with **dynamic ASN (IPv4)**	No	No	No	No	No
BGP session to generic with **Static ASN (IPv4)**	Yes	Yes	Yes	Yes	Yes
BGP session to generic with **dynamic ASN (IPv6)**	No	No	No	No	No
BGP session to generic with **static ASN (IPv6)**	Yes	Yes	Yes	Yes	Yes
BGP Unnumbered session (link-local peering) on L3 Sub-interface (BP has IPv6 app enabled, default VRF)	Yes	Yes	Yes	No	No

(Continued)

BGP to Generic CT Type	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
BGP Unnumbered session (link-local peering) on L3 Sub-interface (BP has IPv6 app enabled, non-default VRF)	No	Yes	Yes	No	No
BGP Unnumbered session (link-local peering) on SVI (BP has IPv6 app enabled, default VRF)	Yes	Yes	Yes	No	No
BGP Unnumbered session (link-local peering) on SVI (BP has IPv6 app enabled, non-default VRF)	No	Yes	Yes	No	No
BGP Unnumbered session (link-local peering) on L3 Sub-interface (default VRF, BP has IPv6 app disabled)	Yes	Yes	Yes	No	No

(Continued)

BGP to Generic CT Type	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
BGP Unnumbered session (link-local peering) on L3 Sub-interface (non-default VRF, BP has IPv6 app disabled)	No	Yes	Yes	No	No
BGP Unnumbered session (link-local peering) on SVI (BP has IPv6 app disabled, default VRF only)	No	No	No	No	No
BGP Peering combinations (Int to Int, Lo to Int, Int to Lo, Lo to Lo)	Yes	Yes	Yes	Yes	Yes
BGP session (IPv6 addressed) with IPv4 SAFI (rfc5549) with **static ASN (BP has IPv6 app enabled)**	No	No	No	No	No
BGP session (IPv6 addressed) with IPv4 SAFI (rfc5549) with **dynamic ASN (BP has IPv6 app enabled)**	No	No	No	No	No

BGP to IP Endpoint CT Type

BGP to IP Endpoint CT Type	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
BGP session from L3 sub-interface to any IP endpoint in the network (IPv4, default/ non-default RZ)	Yes	Yes	Yes	Yes	Yes
BGP session from L3 sub-interface to any IP endpoint in the network (IPv6, default/ non-default RZ)	Yes	Yes	Yes	Yes	Yes
BGP session from SVI to any IP endpoint in the network (IPv4, default/ non-default RZ)	Yes	Yes	Yes	Yes	Yes
BGP session from SVI to any IP endpoint in the network (IPv6, non-default RZ)	Yes	Yes	Yes	Yes	Yes
BGP session from SVI to any IP endpoint in the network (IPv6, default RZ)	Yes	Yes	Yes	Yes	Yes

(Continued)

BGP to IP Endpoint CT Type	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
BGP session from Loopback to any IP endpoint in the network (IPv4, default/non-default RZ)	Yes	Yes	Yes	Yes	Yes
BGP session from Loopback to any IP endpoint in the network (IPv6, default/non-default RZ)	Yes	Yes	Yes	Yes	Yes
BGP session with specific peer IP and **Static ASN (IPv4)**	Yes	Yes	Yes	Yes	Yes
BGP session with specific peer IP and **Static ASN (IPv6)**	Yes	Yes	Yes	Yes	Yes
BGP session with specific peer IP and and **dynamic ASN (IPv4)**	No	No	No	No	No

(Continued)

BGP to IP Endpoint CT Type	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
BGP session with specific peer IP and **dynamic ASN (IPv6)**	No	No	No	No	No
BGP session (IPv6 addressed) with IPv4 SAFI (rfc5549) with **static ASN (BP has IPv6 app enabled)**	No	No	No	No	No
BGP session (IPv6 addressed) with IPv4 SAFI (rfc5549) with **dynamic ASN (BP has IPv6 app enabled)**	No	No	No	No	No

Dynamic BGP Peering CT Type

Dynamic BGP Peering CT Type	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
Dynamic BGP prefix peering on SVI (IPv4), default VRF	Yes	Yes	Yes	Yes	Yes

(Continued)

Dynamic BGP Peering CT Type	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
Dynamic BGP prefix peering on SVI (IPv4), non-default VRF	Yes	Yes	Yes	Yes	Yes
Dynamic BGP prefix peering on SVI (IPv6), default VRF	Yes	Yes	Yes	Yes	Yes
Dynamic BGP prefix peering on SVI (IPv6), non-default VRF	Yes	Yes	Yes	Yes	Yes
Dynamic BGP prefix peering on L3 sub-interface (IPv4), default VRF	Yes	Yes	Yes	Yes	Yes
Dynamic BGP prefix peering on L3 sub-interface (IPv4), non-default VRF	Yes	Yes	Yes	Yes	Yes
Dynamic BGP prefix peering on L3 sub-interface (IPv6), default VRF	Yes	Yes	Yes	Yes	Yes

(Continued)

Dynamic BGP Peering CT Type	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
Dynamic BGP prefix peering on L3 sub-interface (IPv6), non-default VRF	Yes	Yes	Yes	Yes	Yes
Dynamic prefix peering (link-local prefix peering, rfc5549), (BP has IPv6 app disabled)	Yes	No	No	No	No
Dynamic prefix peering (IPv6 peering, IPv4 AFI, rfc5549), (BP has IPv6 app enabled)	No	No	No	No	No

Routing Policy CT Type

Routing Policy CT Type	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
Routing Policy on a BGP session with import/export IPv4 prefixes	Yes	Yes	Yes	Yes	Yes

(Continued)

Routing Policy CT Type	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
Routing Policy on a BGP session with import/export IPv6 prefixes	Yes	Yes	Yes	Yes	Yes
Routing Policy on a BGP session with IPv4 aggregate prefixes	Yes	Yes	Yes	Yes	Yes
Routing Policy on a BGP session with IPv6 aggregate prefixes	Yes	Yes	Yes	Yes	Yes

BGP Attributes (common to all BGP CTs)

BGP Attributes (common to all BGP CTs)	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
BGP: enable Password/MD5 based authentication	Yes	Yes	Yes	Yes	Yes
BGP: Custom BGP timers (Keep Alive timer, Hold timer)	Yes	Yes	Yes	Yes	Yes
BGP: Custom TTL	Yes	Yes	Yes	Yes	Yes

(Continued)

BGP Attributes (common to all BGP CTs)	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
BGP: Enable Single-hop BFD	Yes	Yes	Yes	Yes	Yes

DCI Features

DCI Features	EOS	NX-OS	SONiC	Junos OS	Junos OS Evolved
Type 5 route filtering	No	Yes	No	Yes	Yes

CHAPTER 53

Devices

IN THIS CHAPTER

Qualified Devices and NOS Versions

IN THIS SECTION

Device Roles and Definitions

Table 64: Device Roles and Definitions

Device Role	Definition
Access	• Access role in an EVPN Fabric or IP Fabric • VLAN-based services and does not participate in the EVPN/VXLAN domain
IP Forwarder Only	• Spine or Superspine in EVPN Fabric or any role in IP Fabric • Does not establish or terminate VXLAN services
EVPN Leaf	• A device that participates in the EVPN domain and can establish and terminate VXLAN services • It is a superset of capabilities and devices that are EVPN leafs can perform any role in IP Fabric or EVPN Fabric

Apstra Release 5.0.0

IN THIS SECTION

Juniper - Apstra 5.0.0

Juniper Junos OS - Apstra 5.0.0

Device Role	Qualified NOS Version	Supported Devices (Series)
Access	• 21.4R3-S7 • 22.2R3 • 22.4R3 • 23.4R2	Refer to **IP Forwarder or EVPN Leaf** section below for devices. (Any Junos leaf device can be an Access.)
IP Forwarder Only	• 21.4R3-S7 • 22.2R3 • 22.4R3 • 23.4R2	• QFX5200 • QFX5210

(Continued)

Device Role	Qualified NOS Version	Supported Devices (Series)

IP Forwarder or EVPN Leaf	• 21.4R3-S7
	• 22.2R3
	• 22.4R3
	• 23.4R2
		• QFX5100 - Don't use as leaf with Layer 3 VNI
		• QFX5110 - Can't be used as a border leaf. It can't route between VXLAN IRB and L3 interface.
		• QFX5120
		• QFX10002/8/16 chassis and 4 line cards:
		  • QFX10000-30C
		  • QFX10000-30C-M
		  • QFX10000-36Q
		  • QFX10000-60S-6Q
		• EX4400-24MP
		• EX4400-24X
		• EX4400-48MP
		• EX4400-24T
		• EX4400-48T
		• EX4400-48F
		• EX4650-48Y
		NOTE: The EX4400-series switches are primarily designed for campus environments and have scalability limitations when used as leaf devices in data center deployments. This can negatively impact the entire fabric. However, they can be safely utilized in smaller data center

(Continued)

Device Role	Qualified NOS Version	Supported Devices (Series)
		deployments. Refer to our Juniper Validated Design (JVD) notes and tables for additional details and guidance.

Interconnect Gateway Leaf	• 22.4R3 (minimum)
	• 23.4R2
		• QFX5120
		• QFX10002/8/16 chassis and 4 line cards:
		  • QFX10000-30C
		  • QFX10000-30C-M
		  • QFX10000-36Q
		  • QFX10000-60S-6Q
		• ACX7100
		  Tech Preview - contact Juniper sales team.

Juniper Junos OS Evolved - Apstra 5.0.0

Device Role	Qualified NOS Versions	Supported Devices (Series)
IP Forwarder Only • EVPN - spine role • IP Fabric - any role	• 22.2R3-EVO • 22.4R3-EVO • 23.4R2-EVO	• QFX5220 • QFX5230-64CD • QFX5240 • QFX5240-64OD • QFX5240-64QD

(Continued)

Device Role	Qualified NOS Versions	Supported Devices (Series)

IP Forwarder or EVPN Leaf	• 22.2R3-EVO
	• 22.4R3-EVO
	• 23.4R2-EVO
		• QFX5130
		• QFX5130E-32CD
		• QFX5130-48C
		• QFX5130-48CM
		• QFX5700 chassis and 3 line cards:
		  • JNP-FPC-4CD
		  • JNP-FPC-16C
		  • JNP-FPC-16C
		• PTX10001-36MR
		• PTX10004/8/16 chassis and 2 line cards:
		  • PTX10K-LC1202-36MR
		  • PTX10K-LC1201-36CD
		• ACX7100-32C
		• ACX7100-48L
		• ACX7024

(Continued)

Device Role	Qualified NOS Versions	Supported Devices (Series)

Interconnect Gateway Leaf	• 22.4R3-EVO (minimum)
	• 23.4R2-EVO
		• ACX7100-32C
		Tech Preview - contact Juniper sales team.
		• ACX7100-48L
		Tech Preview - contact Juniper sales team.
		• QFX5130
		• QFX5700 chassis and 3 line cards:
		  • JNP-FPC-4CD
		  • JNP-FPC-16C
		  • JNP-FPC-16C

Line Cards for Modular Chassis - Apstra 5.0.0

Device Model	Line Cards
PTX10004/8/16	• PTX10K-LC1202-36MR • PTX10K-LC1201-36CD
QFX10002/8/16	• QFX10000-30C • QFX10000-30C-M • QFX10000-36Q • QFX10000-60S-6Q

(Continued)

Device Model	Line Cards
QFX5700	• JNP-FPC-4CD • JNP-FPC-16C • JNP-FPC-16C

SONiC - Apstra 5.0.0

Device Role	Qualified NOS Versions	Supported Device Series
Check vendor documentation for feature capabilities of the desired device	4.1.2 • Edge Standard 4.1.2 • Enterprise Standard 4.1.2 4.2.1 • Edge Standard 4.2.1 • Enterprise Standard 4.2.1	If you need a device not listed below, reach out to an Apstra Specialist or PLM. • Dell E3248PXE-ON • Dell Z9664F-ON • Dell Z9432F-ON • Dell Z9432-ON • Dell Z9332F-ON • Dell Z9264F-ON • Dell Z9100-ON • Dell N3248TE-ON • Dell N4248T • Dell S5448F-ON • Dell S5296F-ON • Dell S5248F-ON • Dell S5232F-ON • Dell S5212F-ON • Dell S6000-ON • Edgecore/Accton AS7816-64X • Edgecore/Accton AS7726-32X • Edgecore/Accton AS7712-32X • Edgecore/Accton AS7326-56X • Edgecore/Accton AS5712-54X

(Continued)

Device Role	Qualified NOS Versions	Supported Device Series
		• Edgecore/Accton AS5835-54T • Edgecore/Accton AS5835-54X

Cisco NX-OS - Apstra 5.0.0

Device Role	Qualified NOS Versions	Supported Device Series
Check vendor documentation for feature capabilities of the desired device	• 9.3(13) • 10.2(5) • 10.2(6) • 10.3(4a)	If you need a device not listed below, reach out to an Apstra Specialist or PLM. • C3132QV • C3164PQ • C3172PQ • C36180YC-R • C92348GC-X • C9236C • C93108TC-EX • C93108TC-FX • C93108TC-FX3P • C93180LC-EX • C93180YC-EX • C93180YC-FX • C93180YC-FX3 • C93180YC-FX3S • C93240YC-FX2 • C9332C • C9332PQ • C9336C-FX2 • C93360YC-FX2 • C9348GC-FXP

(Continued)

Device Role	Qualified NOS Versions	Supported Device Series
		• C93600CD-GX Tech Preview - contact Juniper sales team. • C9364C • C9364C-GX • C9372PX on 9.3(10) • C9372TX • C9396PX • C9504 • C9508

Arista EOS - Apstra 5.0.0

Device Role	Qualified NOS Versions	Supported Device Series
Check vendor documentation for feature capabilities of the desired device	• 4.24.5M • 4.28.7.1M • 4.30.3M	If you need a device not listed below, reach out to an Apstra Specialist or PLM. • CCS-720XP-48ZC2 • CCS-720XP-96ZC2 • DCS-7050CX3M-32S • DCS-7050QX-32 • DCS-7050QX-32S • DCS-7050SX2-72Q • DCS-7050SX3-48YC8 • DCS-7050SX3-48YC12 • DCS-7050SX3-96YC8 • DCS-7050SC064 • DCS-7050SC-128 • DCS-7050T-36 • DCS-7050TX3-48C8 • DCS-7050TX-48 • DCS-7050TX-64 • DCS-7050TX-72Q • DCS-7060CX2-32S • DCS-7060CX-32S • DCS-7150S-24 • DCS-7150S-52

(Continued)

Device Role	Qualified NOS Versions	Supported Device Series
		• DCS-7150S-64 • DCS-7160-32CQ • DCS-7160-48TC6 • DCS-7160-48YC6 • DCS-7250QX-64 • DCS-7260CX3-64 • DCS-7260CX3-64E • DCS-7260CX-64 • DCS-7280CR2-60 • DCS-7280CR2A-30 • DCS-7280CR3-32P4 • DCS-7280CR3-96 • DCS-7280CR3K-32D4 • DCS-7280QR-C36 • DCS-7280QRA-C36S • DCS-7280SE-64 • DCS-7280SE-68 • DCS-7280SE-72 • DCS-7280SR2-48YC6 • DCS-7280SR3-48YC8 • DCS-7280SR-48C6 • DCS-7280TR-48C6

(Continued)

Device Role	Qualified NOS Versions	Supported Device Series
		• DCS-7504N • DCS-7504R3 • DCS-7508N • DCS-7508R3 • DCS-7512R3

NOS Versions that are not Qualified

NOS versions that are not in the table above are also expected to work if they are in the same code train and contain only bug fixes. This is usually indicated with version numbers that differ only by the last digit; however, this is not strictly guaranteed by the NOS vendors.

If you plan to use a device or NOS version close to the qualified ones but not listed, we highly recommend that you review the NOS release notes to ensure no backward incompatible or breaking changes are listed. We strongly advise testing the new version thoroughly in a staging environment before deploying it to production.

To request consideration for qualification for a release train not listed, contact your Juniper Apstra Sales representative.

Examples of Only Bug Fix NOS versions:

- Junos and Junos Evolved
  - 20.2R2-S1 > 20.2R2-S3.5 (reason: only service release number change)
  - 20.2R2 > 20.2R3 (reason: R2 > R3 expected to contain only bugfixes)
- Arista EOS
  - 4.25.4M > 4.25.5M (reason: same code train, last digit change and M indicates Maintenance release)
- Cisco NXOS
  - 9.3.13 is the minimum version required to upgrade to 10.3.4a
  - 10.2.5 is the minimum version required to upgrade to 10.3.4a

- 10.2(9)M > 10.2(10)M (reason: same code train, last digit change and M indicates Maintenance release)

Examples of Non-Bug Fix Versions:

- Junos and Junos Evolved
  - 20.2R1 > 20.2R2 (reason: R1 > R2 can have new features + bug fixes)
  - 20.2R2 > 20.4R2 (reason: different release trains)
- Arista EOS
  - 4.25.4M > 4.26.5M (reason: different release trains)
- Cisco NXOS
  - 10.2(1)F > 10.2(3)F (reason: multiple last digit change, F indicates Feature release)

NOS Upgrade Paths

IN THIS SECTION

You can upgrade a network operating system (NOS) from a recommended NOS release in a previous Apstra release to a recommended NOS release in a newer Apstra release. In the same Apstra release, you can upgrade between NOS releases. See the sections below for supported paths. Upgrading to an unqualified version does not cause an error' If you upgrade to an unqualified version, be sure to perform due diligence.

For information about other upgrade paths that may be available, or to request support for a specific upgrade path, contact "Juniper Support" on page 1374.

Apstra Release 5.0.0

Juniper Junos OS & Apstra 5.0.0

Table 65: Juniper Junos OS & Apstra 5.0

From Version	To Version
20.4R3	22.2R3.15
21.2R3	• 22.4R3.25 • 23.4R2.13
21.4R3	• 22.2R3.15 • 23.4R2.13
22.2R2.10	22.4R3.25
22.2R3.15	• 21.4R3 • 22.4R3.25 • 23.4R2.13
22.4R3.25	• 21.2R3 • 21.4R3 • 23.4R2.13
23.4R2.13	• 21.4R3 • 22.2R3.15

Juniper Junos OS Evolved & Apstra 5.0.0

Table 66: Juniper Junos OS Evolved & Apstra 5.0.0

From Version	To Version
21.2R3-EVO	• 21.4R3-EVO • 22.4R3.23-EVO • 23.4R2.14-EVO
21.2R3.10-EVO	21.2R3-EVO
21.4R3-EVO	• 22.2R3.13-EVO • 23.4R2.14-EVO
22.2R2.12-EVO	• 22.4R3.23-EVO • 23.4R2.14-EVO
22.2R3.13-EVO	• 21.4R3-EVO • 22.4R3.23-EVO
22.4R2.11-EVO	22.4R3.23-EVO
22.4R3.23-EVO	• 21.2R3-EVO • 22.2R3.13-EVO • 22.4R2.11-EVO • 23.4R2.14-EVO
23.4R2.14-EVO	• 21.2R3-EVO • 22.4R3.23-EVO

Cisco NX-OS & Apstra 5.0.0

Table 67: Cisco NX-OS & Apstra 5.0.0

From Version	To Version
9.3(11)	9.3(13)
9.3(13)	• 9.3(11) • 10.2(6) • 10.3(4a)
10.1(2)	10.2(6)
10.2(5)	• 10.2(6) • 10.3(4a)
10.2(6)	• 9.3(11) • 9.3(13) • 10.2(5)
10.3(4a)	• 9.3(13) • 10.2(6)

Arista EOS & Apstra 5.0.0

Table 68: Arista EOS & Apstra 5.0.0

From Version	To Version
4.24.5M	• 4.27.6M • 4.28.7.1M • 4.30.3M
4.25.3.1M	• 4.24.5M • 4.27.6M
4.27.6M	• 4.25.3.1M • 4.28.7.1M • 4.30.3M
4.28.7.1M	4.24.5M
4.30.3M	4.27.6M

SONiC & Apstra 5.0.0

Table 69: SONiC & Apstra 5.0.0

From Version	To Version
4.0.5-GA-adv	• 4.1.2-GA-adv • 4.2.1-GA-adv
4.1.2-GA-adv	• 4.0.5-GA-adv • 4.2.1-GA-adv

Table 69: SONiC & Apstra 5.0.0 *(Continued)*

From Version	To Version
4.1.2-GA-edge	4.2.1-GA-edge
4.2.1-GA-adv	4.1.2-GA-adv
4.2.1-GA-edge	4.1.2-GA-edge

Agent Configuration File (Devices)

IN THIS SECTION

Controller Section

```
[controller]
# <metadb> provides directory service for AOS. It must be configured properly
# for a device to connect to AOS controller.
metadb = tbt://aos-server:29731
# Use <web> to specify AOS web server IP address or name. This is used by
# device to make REST API calls to AOS controller. It is assumed that AOS web
# server is running on the same host as metadb if this option is not specified
web =
# <interface> is used to specify the management interface.This is currently
# being used only on server devices and the AOS agent on the server device will
```

```
# not come up unless this is specified.
interface =
```

metadb

Agent Server Discovery is a client-server model. The Apstra Device agent registers directly to the Apstra server via the `metadb` connection. The Apstra server can be discovered from static IP or DNS.

Dynamic DNS - By default, Apstra device agents point to the DNS entry **aos-server**, relying on dhcp-provided DNS resolution and hostname resolution. On the Apstra server, if the *metadb* connection entry points to a DNS entry, then the Apstra agents must be able to resolve that DNS entry as well. DNS must be configured so aos-server resolves to an interface on the Apstra server itself, and so the agents are configured with `metadb = tbt://aos-server:29731`

Static DNS - We can add a static DNS entry pointing directly to the IP of aos-server. Add a static DNS entry, or use a DNS Nameserver configuration on the device.

Arista and Cisco Static Hostname

```
localhost(config)#ip host aos-server 192.168.25.250
```

Obtaining IP from Apstra Server

```
admin@aos-server:~# ip addr show dev eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 08:00:27:8a:39:05 brd ff:ff:ff:ff:ff:ff
    inet 192.168.59.250/24 brd 192.168.59.255 scope global eth0
    inet6 fe80::a00:27ff:fe8a:3905/64 scope link
    valid_lft forever preferred_lft forever
```

Then the agents will be configured with `metadb = tbt://aos-server:29731`.

web

In a future release, the Apstra REST API will be able to run on a separate server from the Apstra server itself. This feature is for Apstra internal usage only.

interface

The device agent source interface applies to Linux servers only (Ubuntu, CentOS). This source IP is the server interface that the device agent uses when registering with Apstra. For example, on a server, to bind the device agent to *eth1* instead of the default *eth0*, specify `interface = eth1`.

Service Section

```
[service]
# When managing device configuration AOS agent will restore backup config if it
# fails to connect to AOS controller in <backup_config_restoration_timeout>,
# specified as <hh:mm:ss>. Set it to 00:00:00 to disable backup restoration
backup_config_restoration_timeout = 00:00:00
```

The service section manages specific agent configuration related to configuration rendering and telemetry services.

backup_config_restoration_timeout

Configuration is not *stored* on the device. This prevents a device from booting up and immediately participating in fabric that may not be properly configured yet. The Apstra device agent is configured after the discovery phase completes.

`backup_restoration_timeout = 00:00:00` This disabled state (default) keeps the Apstra device agent from replacing the running configuration if it cannot contact the Apstra server. Any previous configuration state is not restored.

`backup_restoration_timeout = 00:15:00` Any value other than the default `00:00:00` enables the Apstra agent to boot and replace the running configuration with the most known previous state after the specified period of time (fifteen minutes in this example). Specifically, the files from `/.aos/rendered/` are restored to the system after the configuration restore period expires.

Logrotate Section

```
[logrotate]

# AOS has builtin log rotate functionality. You can disable it by setting
# <enable_log_rotate> to 0 if you want to use linux logrotate utility to manage
# your log files. AOS agent reopens log file on SIGHUP
enable_log_rotate = 1
# Log file will be rotated when its size exceeds <max_file_size>
max_file_size = 1M
```

```
# The most recent <max_kept_backups> rotated log files will be saved. Older
# ones will be removed. Specify 0 to not save rotated log files, i.e. the log
# file will be removed as soon as its size exceeds limit.
max_kept_backups = 5
# Interval, specified as <hh:mm:ss>, at which log files are checked for
# rotation.
check_interval = 1:00:00
```

Apstra logs to the `/var/log/aos` folder under a series of files. Apstra implements its own method of log rotation to prevent `/var/log/aos` from filling up. You can enable (2) or disable (1) log rotation. Each individual log file is rotated when it approaches the appropriate maximum size. Log rotation occurs by default every hour.

Device Info Section

```
[device_info]
# <model> is used to specify the device's hardware model to be reported to AOS
# device manager. This is only used by servers, so can be ignored for non-
# server devices such as switches. By default a server reports "Generic Model"
# which matches a particular HCL entry's selector::model value in AOS. Specify
# another model for the server to be classified as a different HCL entry.
model = Generic Model
```

model

The device info section is used to modify the default device model of servers as they register to Apstra. For example, `Server 2x10G` changes the server to a dual-attached L3 server. All valid options for `model` include:

- Generic Model
- Server 2x10G
- Server 1x25G
- Server 1x40G
- Server 4x10G

Device Profile Section

```
# <device_profile_id> is used to specify the device profile to be associated to
# the device. Selector in the specified device profile should match the
# reported device facts.
device_profile_id =
[credential]
username = admin
```

Juniper Telemetry Commands

This section assists network administrators in understanding why telemetry alarms exist, and how they are generated. This is a partial list of interface commands.

Apstra uses CLI to retrieve telemetry from Junos OS and Junos OS Evolved devices.

Table 70: Juniper Telemetry Commands

Service	Command
Interface Counters	`show interfaces extensive`
Interface Error Counters	`show interfaces extensive`
Interface Status	`show interfaces terse`
LLDP neighbors	`show lldp neighbors`
BGP sessions	`show bgp neighbor`
Hostname	`show system information`
ARP	`show arp no-resolve` Provides the ARP information. This is combined with `show configuration routing-instances` which provides the VRF membership for interfaces.

Table 70: Juniper Telemetry Commands *(Continued)*

Service	Command
MAC Table	Apstra has two collectors for retrieving MAC telemetry: `show ethernet-switching table extensive` is used with CLIs gRPC collectors use Xpaths. `/network-instances/network-instance/mac-table/entries/entry`
Routing Table	`show route table inet`
Port Channel	`show lacp interfaces`

Arista Telemetry Commands

This section assists network administrators in understanding why telemetry alarms exist, and how they are generated. This is not an exhaustive list of interface commands.

Arista EOS uses a few techniques from the EOS SDK API to directly subscribe to event notifications from the switch, for example 'interface down' or 'new route' notifications. When using an event-based notification, you do not have to continually render 'show' commands every few seconds. The EOS SDK gives you the information immediately as soon as the switch has the status.

CAUTION: Event-based subscription requires the EOSProxySDK agent.

When the Arista API does not provide information (LLDP statistics), Apstra runs CLI commands at a regular interval to derive telemetry expectations.

Table 71: Arista Telemetry Commands

Service	Command
Interface counters	`show interface counters`

Table 71: Arista Telemetry Commands *(Continued)*

Service	Command
Interface error counters	`show interfaces counters errors`
Interface status	`show interfaces status`
LLDP neighbors	`show lldp neighbors detail`
BGP Sessions	`show ip bgp summary`
Hostname	`show hostname`
ARP	ARP collection is done using an event-monitor for performance. `show event-monitor arp` and `show ip arp`
MAC Table	MAC address collection is done using an event-monitor for performance. `show event-monitor mac` and `show mac address-table`
Routing table	`show ip route`
Port-channel	`show port-channel summary`
MLAG	`show mlag` and `show mlag interfaces`

Cisco Telemetry Commands

This section assists network administrators in understanding why telemetry alarms exist, and how they are generated. This is a partial list of interface commands.

Cisco telemetry is derived from the NX-API with 'show' commands and embedded event manager applets that provide context data to the device agent while it is running. Most commands are run as their CLI version wrapped into JSON output.

Table 72: Cisco Telemetry Commands

Service	Command
Interface counters	`show interface counters | json`
Interface error counters	`show interface counters errors | json`
Interface status	`show interface status | json`
LLDP neighbors	`show lldp neighbors detail | json`
BGP Sessions	`show bgp session | json`
Hostname	`show hostname | json` and `show hosts | json`
ARP	`show ip arp vrf default | json`
MAC Table	`show mac address-table | json`
Routing table	`show ip route | json`
Port-channel	`show port-channel summary | json`
MLAG	`show vpc | json`

Linux Server Telemetry Command

Linux Servers use simple CLI commands and standard Linux sockets for most telemetry collection.

Table 73: Linux Server Telemetry Commands

Service	Command
Interface counters	`ethtool -m`
Interface error counters	`ethtool -m`

Table 73: Linux Server Telemetry Commands *(Continued)*

Service	Command
Interface status	Interface status is collected using the netlink api (`AF_INET`)
LLDP neighbors	`lldpctl -f xml`
BGP Sessions	`vtysh -c 'show ip bgp summary json'`
Hostname	`hostname`
ARP	`ip -4 neigh`
MAC Table	`brctl showmacs`
Routing table	`show ip route` and the `AF_INET` linux socket
Port-channel	`netshow bondmems --json`
MLAG	`clagctl -j`

CHAPTER 54

Blueprint Analytics

IN THIS CHAPTER

Predefined Dashboards

IN THIS SECTION

Dashboard: Device Environmental Health Summary

Goal	Show device environmental data
Trigger	Presence of at least one assigned system

Widgets / Probes	• Systems missing power supplies / Device Environmental Checks • Systems missing fans / Device Environmental Checks • Switch temperature alarm / Device Environmental Checks • Systems with inoperative power supplies / Device Environmental Checks • Systems with inoperative fans / Device Environmental Checks • Power supply temperature alarm / Device Environmental Checks • Systems with faulty power supply fans / Device Environmental Checks • Airflow direction mismatch / Device Environmental Checks

Dashboard: Device Health Summary

Ensure that the same metric is not collected twice from the same device.

Goal	Present utilization data for system CPU, system memory and maximum disk utilization of a partition on every system present
Trigger	Presence of at least one deployed system
Widgets / Probes	• Systems with high cpu utilization / Device System Health • Systems with high memory utilization / Device System Health • Systems with high disk utilization / Device System Health

Dashboard: Device Telemetry Health Summary

Goal	Present sustained service execution anomalies under the device telemetry health probe

Trigger	Presence of at least one deployed system
Widgets / Probes	• Systems with degraded waiting time per service / Device Telemetry Health • Systems that sustained telemetry timeouts per service / Device Telemetry Health • Systems that sustained telemetry failures per service / Device Telemetry Health • Systems that sustained telemetry underruns per service / Device Telemetry Health

Dashboard: Drain Validation

Goal	Ensure drained switches are indeed drained of traffic by ensuring total bandwidth is minimal
Trigger	Presence of at least one drained switch
Widgets / Probes	Drained Switches Excess Traffic / Drain Traffic Anomaly

Dashboard: Throughput Health MLAG

Goal	Find issues in physical infrastructure that affect the available throughput caused by issues such as imbalanced traffic over a group of L3 (ECMP) or L2 (LAG) links
Trigger	Created on blueprints with no redundancy groups or MLAG blueprint
Widgets / Probes	• LAG Imbalance / LAG Imbalance • MLAG Imbalance / MLAG Imbalance • Fabric ECMP Imbalance / ECMP Imbalance (Fabric Interfaces)

Dashboard: Traffic Trends

Goal	Visualize traffic trends for general insights into fabric usage
Trigger	Grouped Ingress Traffic last 1 hour / Bandwidth Utilization
Widgets / Probes	Grouped Egress Traffic last 1 hour / Bandwidth Utilization

Dashboard: Virtual Infra Fabric Health Check

Goal	Find problems in physical or virtual infrastructure that affect workload connectivity
Trigger	Presence of at least one virtual infra manager in the blueprint
Widgets / Probes	• Hypervisor VLANs missing in Fabric / Hypervisor & Fabric VLAN Config Mismatch • Hypervisor PNIC LAG Status / Hypervisor & Fabric LAG Config Mismatch • Hypervisor Low MTU anomalies / Hypervisor MTU Threshold Check • Critical Services affected by VLAN misconfig / VMs Without Fabric Configured VLANs • Hypervisor has inconsistent MTU / Hypervisor MTU Mismatch

Dashboard: Virtual Infra Redundancy Check

Goal	Find single points of failure in physical or virtual infrastructure that affect high availability and available bandwidth for workloads
Trigger	Presence of at least one virtual infra manager in the blueprint
Widgets / Probes	• Hypervisors without ToR switch redundancy / Hypervisor Redundancy Checks • Virtual Infra Networks without link redundancy / Hypervisor Redundancy Checks

Predefined Probes

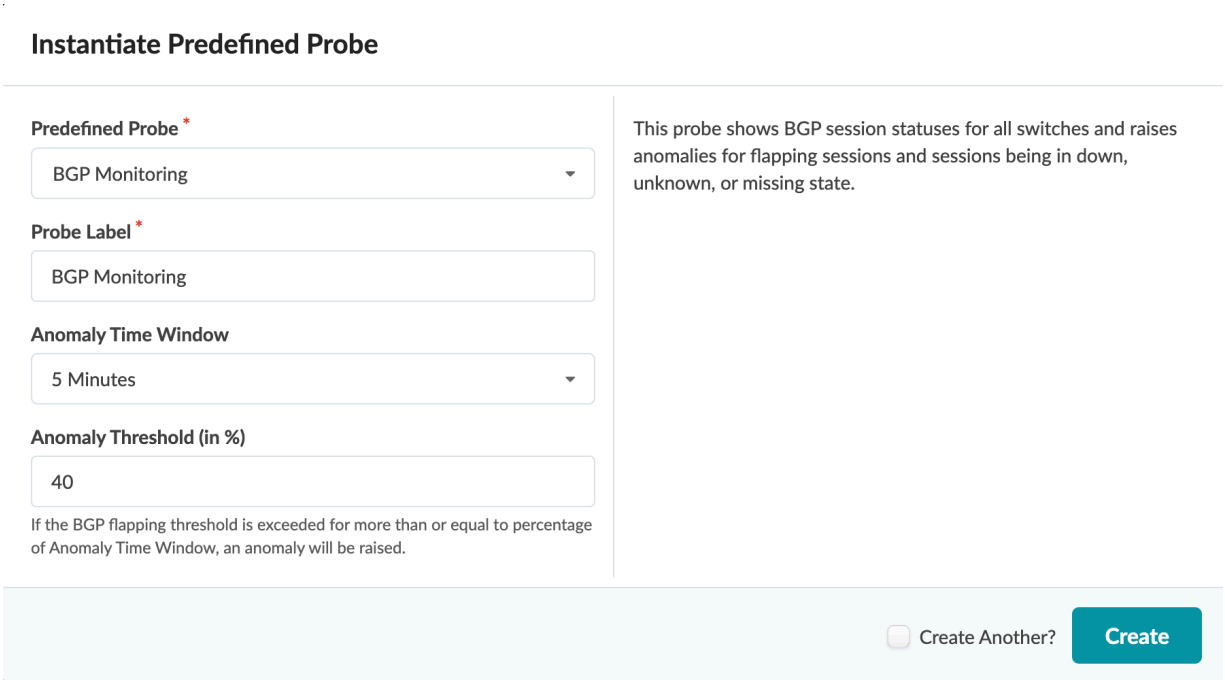
IN THIS SECTION

Apstra software ships with many predefined probes that you can instantiate (Analytics > Probes > Create Probe > Instantiate Predefined Probe).

Probe: BGP Monitoring

IN THIS SECTION

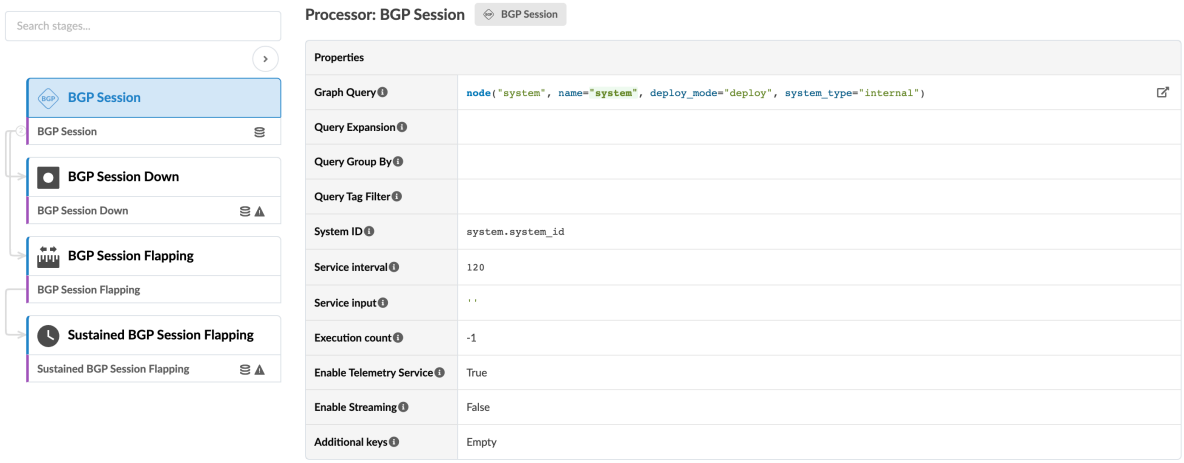
This probe is supported on Junos OS, Junos OS EVO, and Arista EOS (not Cisco or SONiC). The BGP Monitoring probe shows BGP session status for all switches and raises anomalies for flapping BGP sessions. In Freeform blueprints, the probe also monitors and raises anomalies when BGP sessions are down, missing or unknown. (In Datacenter blueprints, BGP session up and down state is included with built-in telemetry, so it's not required in this probe.)



The probe includes 4 processors and stages as shown below:

BGP Session

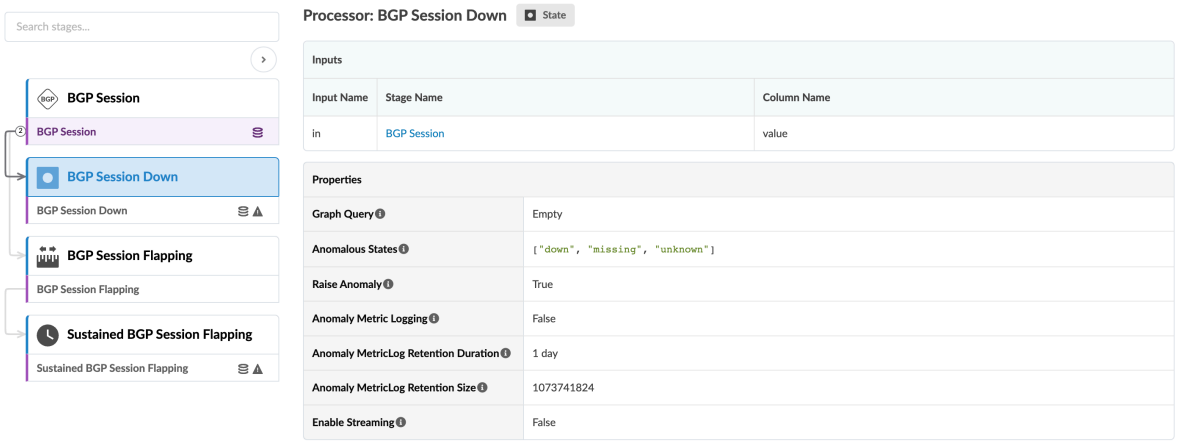
The **BGP Session** processor includes the parameters as shown in the screenshot below:



The **BGP Session** stage shows all BGP sessions for devices.

BGP Session Down

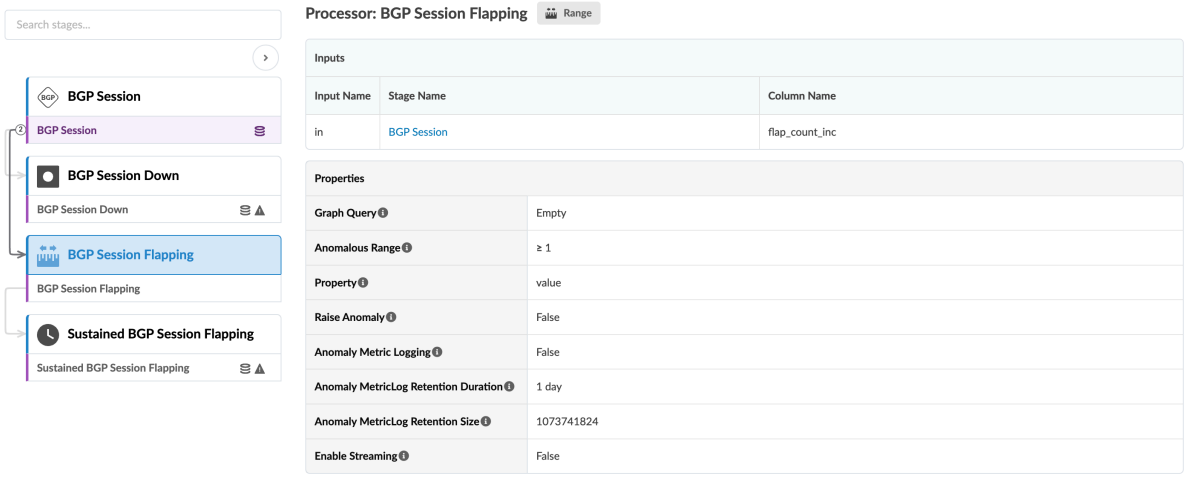
BGP Session Down is included only in Freeform blueprints. The processor includes the parameters as shown in the screenshot below:



The **BGP Session Down** stage determines if the BGP session is not "up" and raises an anomaly accordingly.

BGP Session Flapping

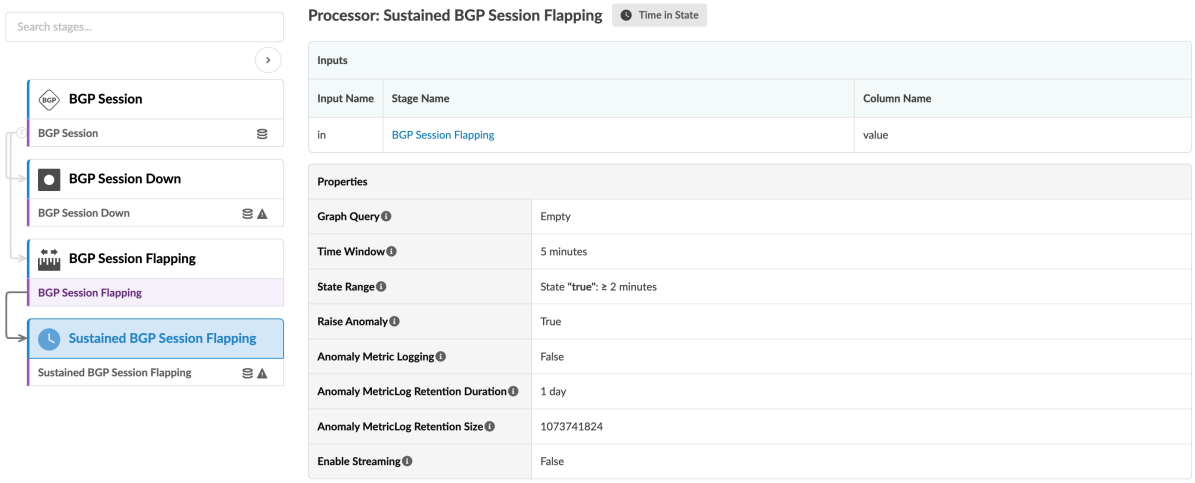
The **BGP Session Flapping** processor includes the parameters as shown in the screenshot below:



The **BGP Session Flapping** stage checks if the BGP session has new flaps for the last service interval period. (2 minutes by default).

Sustained BGP Session Flapping

The **Sustained BGP Session Flapping** processor includes the parameters as shown in the screenshot below:



The **Sustained BGP Session Flapping** stage checks if the BGP session has new flaps for the specified period of time. For example, assume there are BGP flaps between leaf1 and spine1 nodes. The fabric BGP session between these nodes generates new BGP flaps when the interface status is changed on spine1 that's connected to leaf1. When *shutdown* and *up* interface is performed seven times on spine 1, it creates seven flaps for fabric BGP sessions between leaf1 and spine1. The seven new flaps are added and two anomalies are raised.

For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see more details specific to the probe.

Probe: Bandwidth Utilization

The bandwidth utilization probe calculates bandwidth utilization. It captures history of bandwidth utilization trends at differing levels of aggregation.

Instantiate Predefined Probe

Predefined Probe *

Bandwidth Utilization

Probe Label *

Bandwidth Utilization

First summary average period

2 Minutes

First summary history duration

1 Hour

Second summary average period

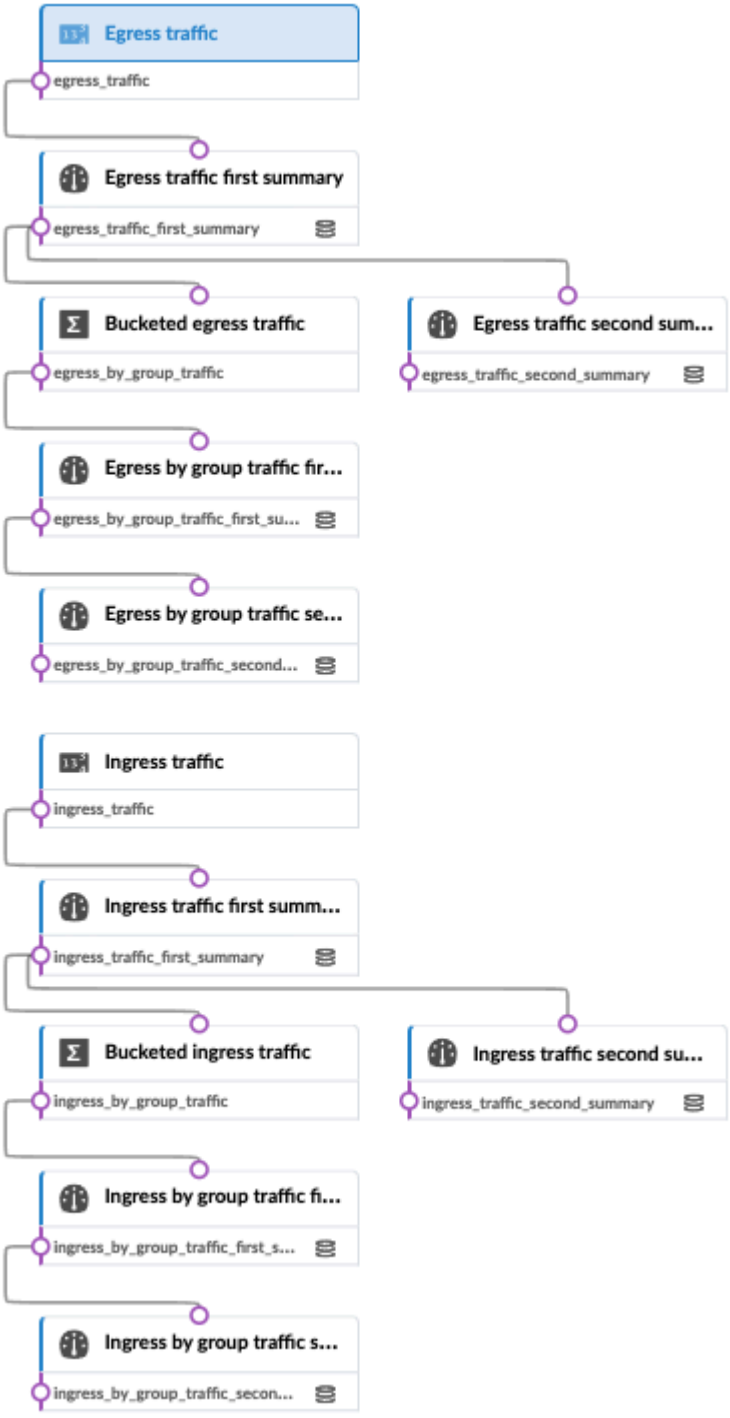
1 Hour

Second summary history duration

30 Days

Generate a probe to calcutale bandwidth utilization

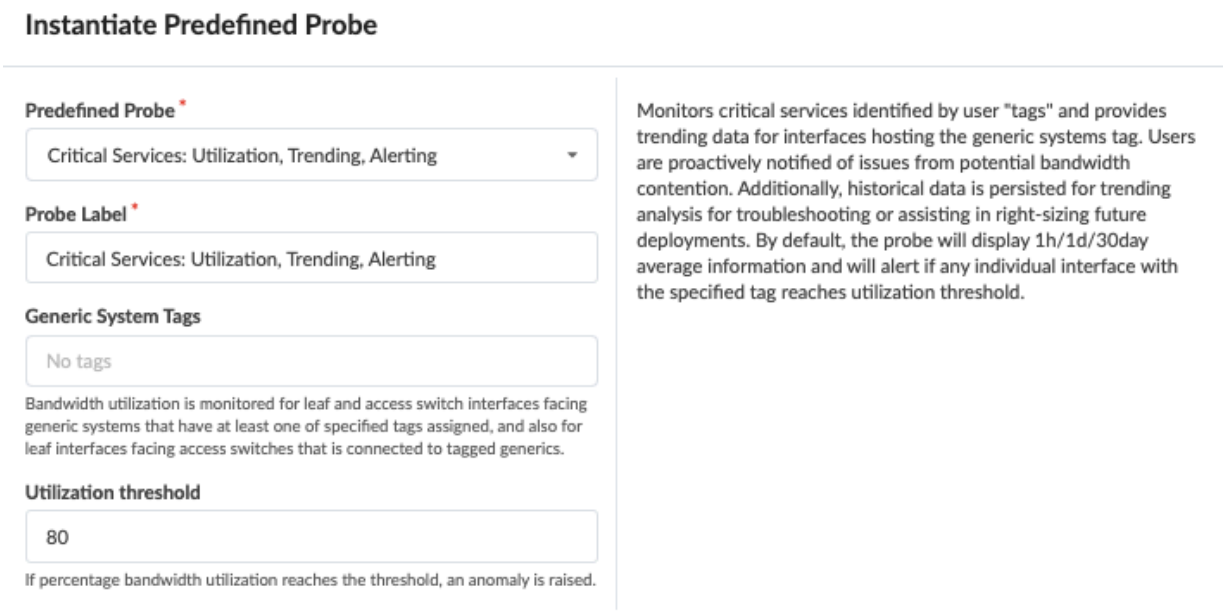
This probe captures history of bandwidth utilization trends at differing levels of aggregation.

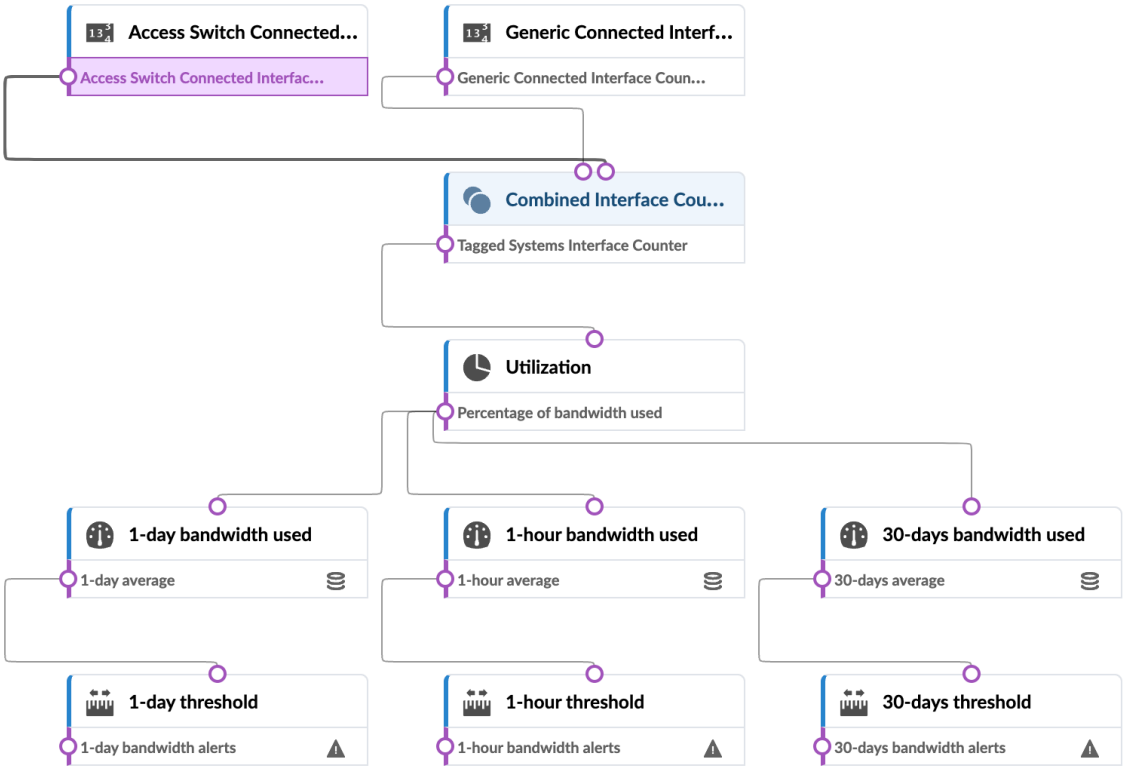


For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: Critical Services: Utilization, Trending, Alerting

The critical services probe monitors critical services identified by user *tags* and provides trending data for interfaces hosting the generic systems tag. Users are proactively notified of issues from potential bandwidth contention. Additionally, historical data is persisted for trending analysis for troubleshooting or assisting in right-sizing future deployments. By default, the probe displays 1h/1d/30day average information and alerts if any individual interface with the specified tag reaches utilization threshold.





For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: Device Environmental Checks

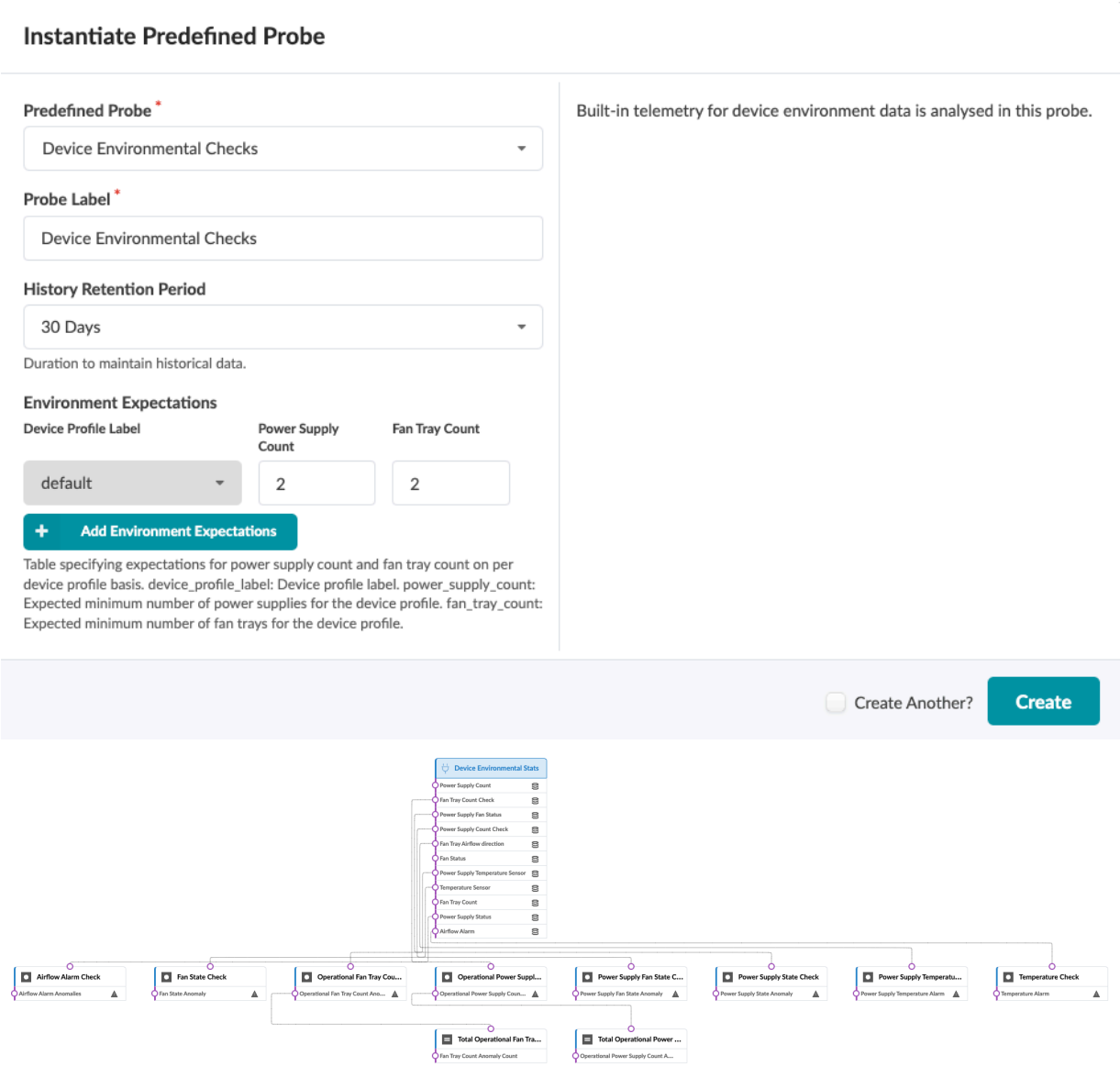
The device environmental checks probe monitors critical environmental metrics for managed switches including power supply, fan and temperature for real-time values of historical data retention over time.

When you instantiate this predefined probe, the instantiation menu displays a list of switch models in the blueprint. PSU count, fan count and airflow direction information provide intent for deploying the switches.

If you have multiple blueprints that use the same switch model, you can set one expectation for the switch in one blueprint and a different expectation for the switch in a different blueprint.

Within one blueprint, all switches of the same model must have the same expectations. For example, you can't differentiate between specific QFX5120-48Y switches.

NOTE: The device environmental checks probe is only enabled for Juniper devices.



For more information about this probe, from the blueprint, navigate to **Analytics Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

This probe is also used to generate a environmental data predefined report that shows the environmental metrics of all your devices. See the "Environmental Data Analytics Report " on page 42.

Probe: Device System Health

The device system health probe alerts if the system health parameters (CPU, memory and disk usage) exceed their specified thresholds for the specified duration.

Instantiate Predefined Probe

Predefined Probe *

Device System Health

Probe Label *

Device System Health

CPU utilization threshold

80

If percentage CPU utilization exceeds the threshold, an anomaly is raised

Memory utilization threshold

80

If percentage memory utilization exceeds the threshold, an anomaly is raised

Disk utilization threshold

80

If percentage disk utilization exceeds the threshold, an anomaly is raised

Duration

11 minutes

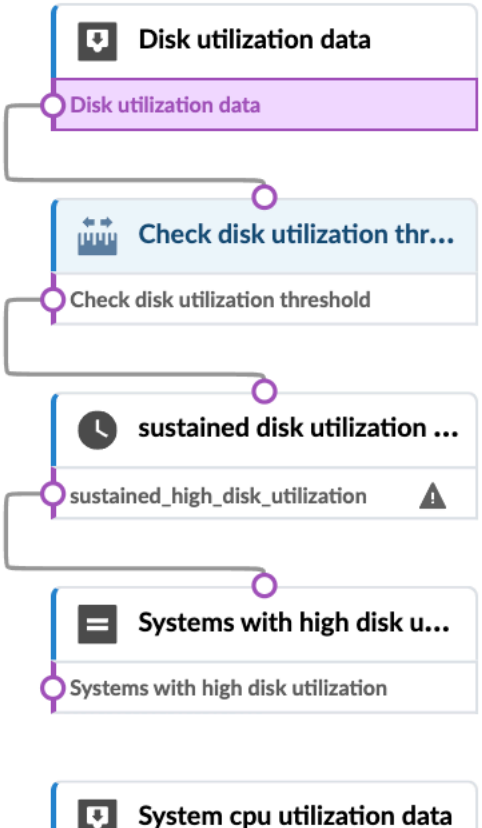
Time period in recent-history over which utilization data will be considered

Threshold Duration

6 minutes

Total amount of time in recent-history during which the utilization has to be high for anomaly to be raised

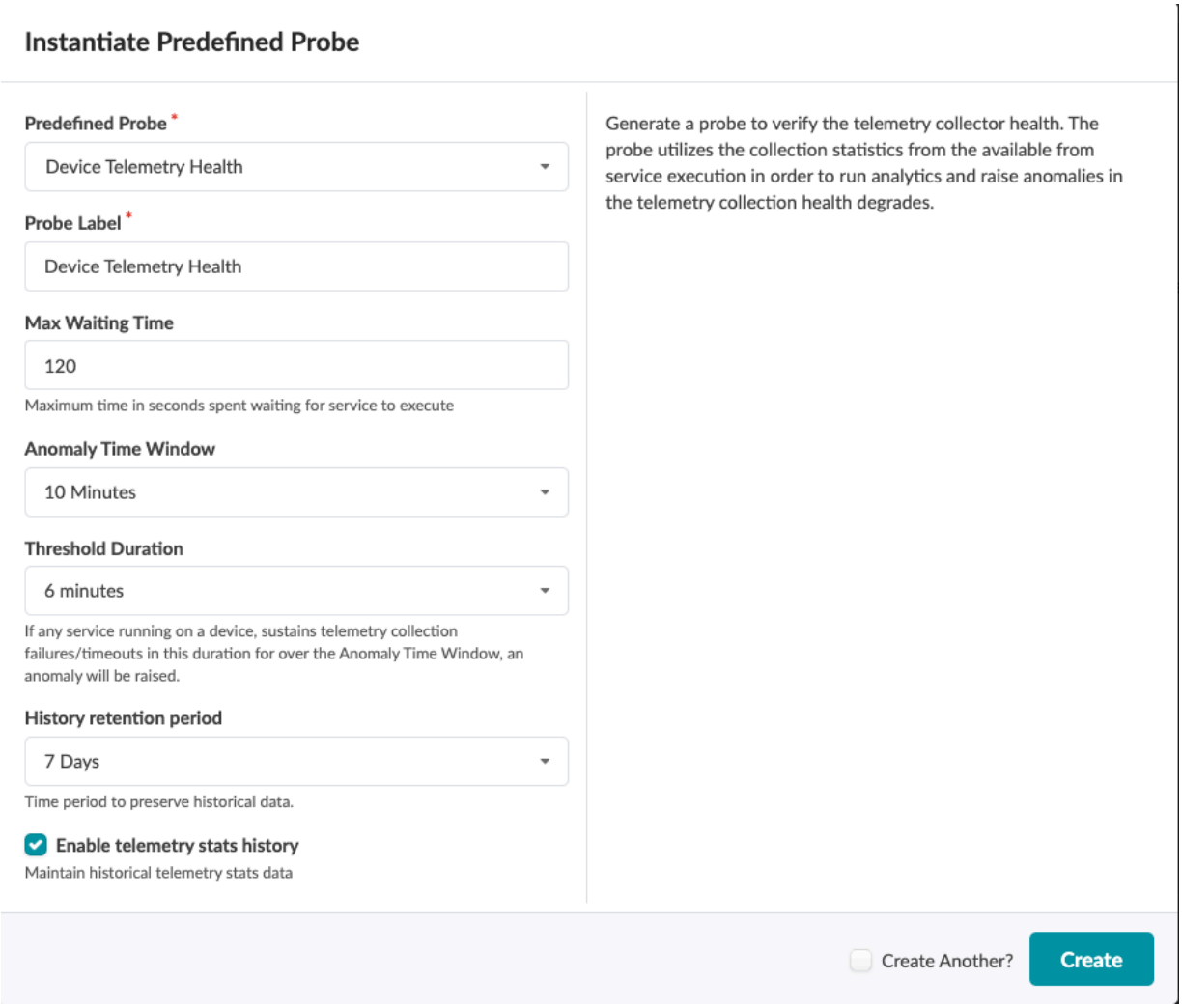
This probe alerts if the system health parameters (CPU, memory and disk usage) exceed their specified thresholds for the specified duration.

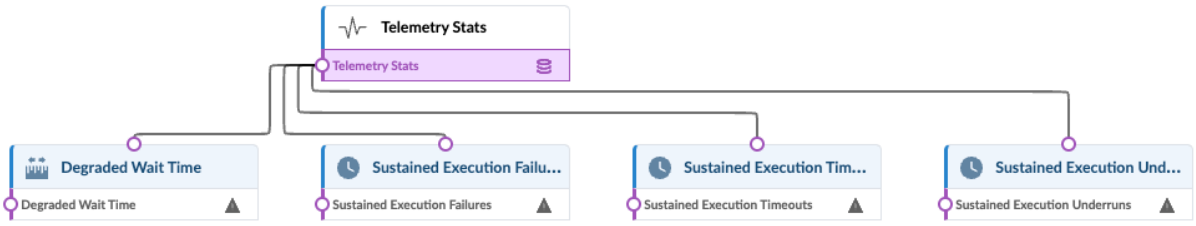


For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: Device Telemetry Health

The device telemetry health probe verifies telemetry collector health. It runs analytics on the collection statistics from available service execution and if the telemetry collection health degrades, anomalies are raised.





For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: Device Traffic

The device traffic probe (previously known as headroom probe) provides insights about link capacity between two points in the network. It provides multiple interface counters (rx, tx, discard, errors and so on) for all managed devices. It displays all interface counters available for the system, their utilization on a per-port and aggregated utilization per-system basis. If rules are violated, it raises anomalies.

Instantiate Predefined Probe

Predefined Probe *

Device Traffic

Probe Label *

Device Traffic

Interface counters average period

2 Minutes

The average period duration for interface counters

Enable interface counters history

Maintain historical interface counters data

Interface counters history retention period

30 Days

Duration to maintain historical interface counters data

Enable system counters history

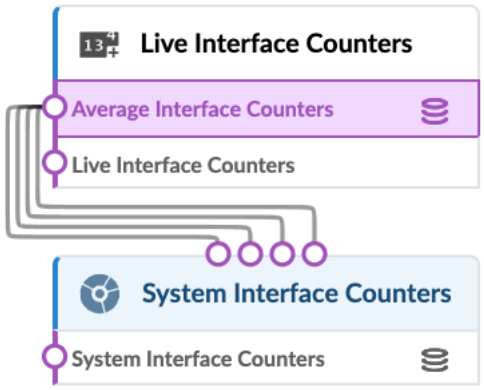
Maintain historical system interface counters data

System interface counters history retention period

30 Days

Duration to maintain historical system interface counters data

This probe displays the all the interface counters available for the system, their utilizations and utilizations aggregated on a per system basis.

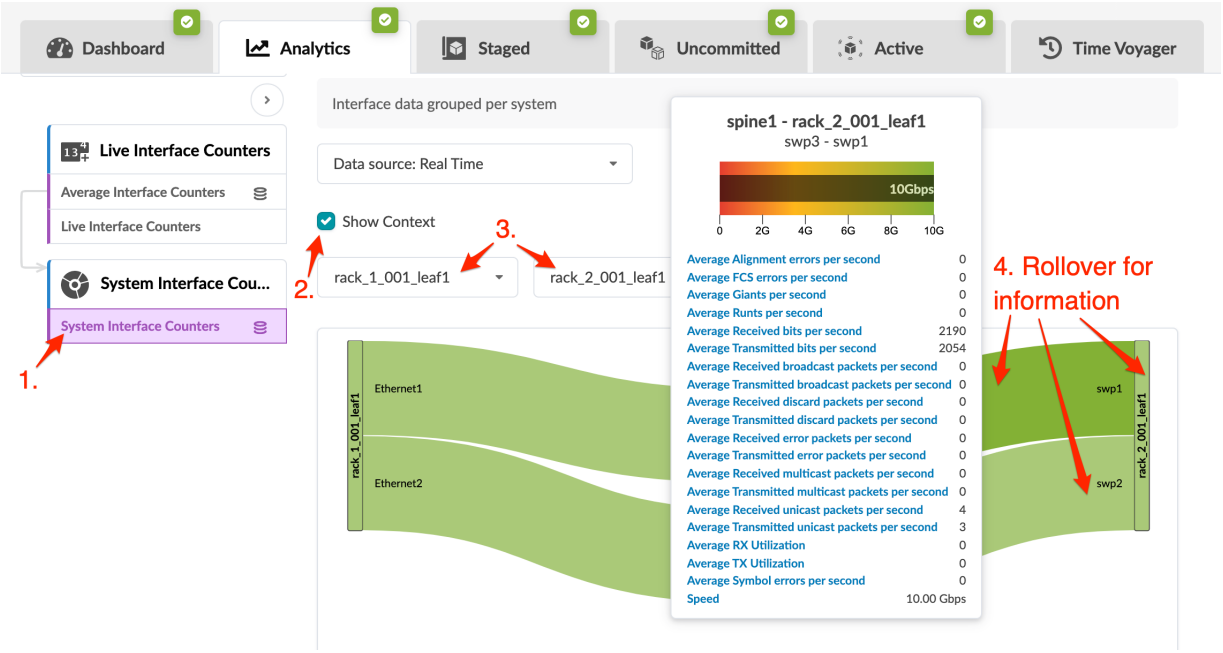


NOTE: You can change probe inputs, but if you change the probe processors then the probe is not a **predefined** probe anymore and the traffic layer view is not available in the active topology. For more information about the traffic layer view, see "Physical Blueprint" on page 71.

Source Processor	**Live Interface Counters** ("Traffic Monitor" on page 1728)	Purpose: Wires in Interface traffic counters every 5 seconds (by default) for all managed devices and keeps historical data based on retention period specified during probe creation.		
		Output Stages	**Average Interface Counters**	Set of interface counters samples, for each port of each managed device, based on specified average time with historical data.
			Live Interface Counters	Set of live interface counter samples for each port of each managed device
Additional Processor(s)	**System interface counters** ("System Utilization" on page 1717)	Purpose: This processor consumes in 'Average Interface Counters' for calculating interface counters per system with historical data. It uses properties rx_bps_average, rx_utilization_average, tx_bps_average, and tx_utilization_average to compute the system TX and RX utilization and to compute headroom between the specified source and destination systems. Input Stage: Average Interface counters		
		Output Stage: System Interface Counters		Set of system interface counters samples (for each device of managed devices) indicating Aggregated TX/RX, Aggregated TX/RX %, and Max interface TX/RX utilization %. The system level RX/TX calculation aggregates the Tx/RX of all the device interfaces that are "up". The max interface RX/TX calculation is the device interface with the highest Rx and the device interface with highest Tx.

To see traffic between a particular source and destination from the device traffic probe, click **System Interface Counters**, check the **Show Context** check box, then select a source and destination from the drop-down lists. Roll over different sections to display relevant information. Different colors represent link capacity, where green means plenty of capacity and red means that the link is running out of

capacity.



For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: Drain Traffic Anomaly

The drain traffic anomaly probe raises anomalies when excess traffic is on a node that is being drained.

Instantiate Predefined Probe

Predefined Probe *

Drain Traffic Anomaly

Probe Label *

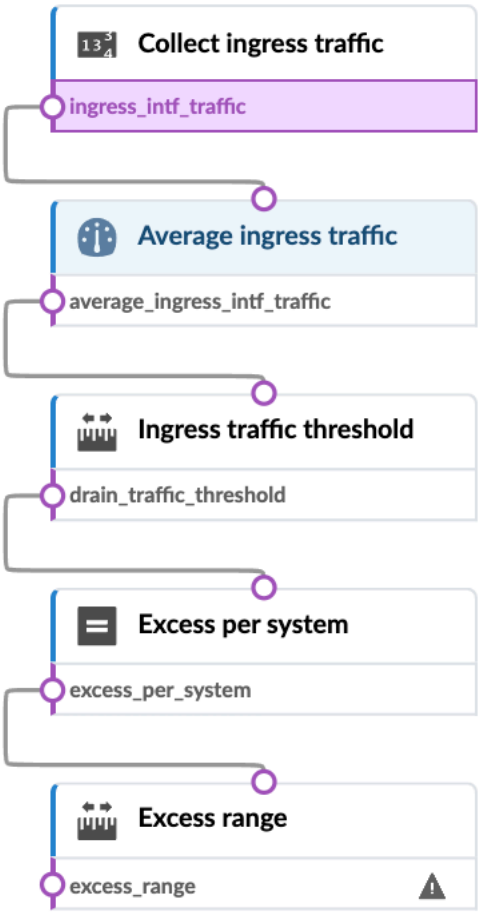
Drain Traffic Anomaly

Threshold

100000

Traffic threshold in bits per second. An anomaly will be raised if a traffic on some interface is in excess of this value.

Generate a probe to raise anomaly when there is excess traffic on a node that is being drained.



For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: ECMP Imbalance (External Interfaces)

Purpose

This probe calculates ECMP imbalance on generic system-facing ports. The set of external-facing links (keyed by common system_id) is determined to be imbalanced if the standard deviation of the tx_bytes counter (averaged periodically over the specified period) for the involved interfaces is above "Max Standard Deviation". If such imbalance is observed for more than "Threshold Duration" over the last "Duration" time period, an anomaly is raised. The last "Anomaly History Count" anomaly state changes are stored for observation. If more than "Max Imbalanced Systems" systems are imbalanced,an anomaly is raised. We maintain for inspection the number of imbalanced systems over the last "System Imbalance History Count" samples.

When instantiating this probe, external router tag(s) must be specified.

Source Processor

external interface traffic (Interface Counters)

Purpose: wires in interface traffic samples (measured in transmitted bytes per second) from each interface connected to the generic systems.

Output Stage: external_int_traffic

Additional Processor(s)

external interface traffic avg (Periodic Average)

Purpose: Calculate average traffic during period specified by average_period facade parameter. Unit is bytes per second.

Input Stage: external_int_traffic

Output Stage: external_int_traffic_avg

Set of traffic average values (for each generic system-facing interface). Each set member has the following keys to identify it: label (human-readable name of the system), system_id (id of the system, usually serial number), interface (name of the interface).

external interface std-dev (Standard Deviation)

Purpose: calculate standard deviation for a set consisting of traffic averages for each generic system-facing interface on a given system. Grouping per system is achieved using 'group_by' property set to 'system_id' and 'label'.

Input Stage: external_int_traffic_avg

Output Stage: ext_int_std_dev — Set of values, each indicating standard deviation (as a measure of ECMP imbalance) for traffic averages for each generic system-facing interface on a given system. Each set member has 'system_id' and 'label' key to identify system whose ECMP imbalance the value represents.

std-dev percentage (Ratio)

Input Stage: ext_int_std_dev

Output Stage: std_dev_percentage

live ecmp imbalance (Range)

Purpose: Evaluate if standard deviation between generic system-facing interfaces on each system is within acceptable range. In this case acceptable range is between 0 and std_max facade parameter (in bytes per second unit).

Input Stage: std_dev_percentage

Output Stage: live_ecmp_imbalance — Set of true/false values, each indicating if standard deviation (as a measure of ECMP imbalance) for traffic averages for each external router-facing interface on a given leaf is within acceptable range. Each set member has system_id key to identify system whose ECMP imbalance the value represents.

links imbalanced percentage (Match Percentage)

Input Stage: live_ecmp_imbalance

Output Stage: links_imbalanced_percentage

systems imbalanced (Range)

Input Stage: links_imbalanced_percentage

Output Stage: systems_imbalanced

sustained ecmp imbalance (Time in State)

Purpose: Evaluate if standard deviation between generic system-facing interfaces on each leaf has been outside acceptable range, (as defined by 'live ecmp imbalance' processor) for more than 'threshold_duration' seconds during last 'total_duration' seconds. These two parameters are part of facade specification.

Input Stage: systems_imbalanced

	Output Stage: sustained_ecmp_imbalance	Set of true/false values, each indicating if standard deviation (as a measure of ECMP imbalance) for traffic averages for each external router-facing interface on a given system has been outside acceptable range for more than specified period of time. Each set member has system_id key to identify system whose ECMP imbalance the value represents.
systems imbalanced count (Match Count)	Purpose: Count how many systems have external ecmp imbalance anomaly true at any instant in time. Input Stage: sustained_ecmp_imbalance	
	Output Stage: system_tx_imbalance_count	Number of systems with external ecmp imbalance.
live system imbalanced (Range)	Purpose: Evaluate if the number of imbalanced systems is within acceptable range, which in this instance means less than 'max_systems_imbalanced' value which is a facade parameter Input Stage: system_tx_imbalance_count	
	Output Stage: live_system_imbalance_count	Boolean indicating if the number of imbalanced systems is within accepted range, i.e. less than 'max_systems_imbalanced" which is a facade parameter

For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: ECMP Imbalance (Fabric Interfaces)

Purpose	This probe calculates ECMP imbalance on fabric ports. A given set of ECMP links (only calculated on leaf-to-spine links), identified by common system_id, is determined to be imbalanced if the standard-deviation of the tx_bytes counter (averaged periodically over the specified period) for the involved leaf-interfaces is above "Max Standard Deviation".

If such imbalance is observed for more-than "Threshold Duration" over the last"Duration" time period, we raise an anomaly.

The last "Anomaly History Count" anomaly state-changes are stored for observation.

If more-than "Max Imbalanced Systems" systems are imbalanced, we raise a distinct anomaly.

We maintain for inspection the number of imbalanced systems over the last "System Imbalance History Count" samples.

Source Processor

leaf fabric interface traffic (Interface Counters-)

Purpose: wires in interface traffic samples (measured in bytes per second) from each spine-facing interface on each leaf.

Output Stage: leaf_fabric_int_traffic

Set of traffic samples (for each spine-facing interface on each leaf). Each set member has the following keys to identify it: label (human-readable name of the leaf), system_id (id of the leaf system, usually serial number), interface (name of the interface).

Additional Processor(s)

leaf fabric interface traffic avg (Periodic Average)

Purpose: Calculate average traffic during period specified by average_period facade parameter. Unit is bytes per second.

Input Stage: leaf_fabric_int_traffic

Output Stage: leaf_fabric_int_tx_avg

Set of traffic average values (for each spine-facing interface on each leaf). Each set member has the following keys to identify it: label (human-readable name of the leaf), system_id (id of the leaf system, usually serial number), interface (name of the interface).

leaf fabric interface std-dev (Standard Deviation)

Purpose: calculate standard deviation for a set consisting of traffic averages for each spine-facing interface on a given leaf. Grouping per leaf is achieved using 'group_by' property set to 'system_id'.

Input Stage: leaf_fabric_int_tx_avg

Output Stage: leaf_fab_int_std_dev

Set of values, each indicating standard deviation (as a measure of ECMP imbalance) for traffic averages for each spine-facing interface on a given leaf. Each set member has

system_id key to identify leaf whose ECMP imbalance the value represents.

std-dev percentage (Ratio)

Input Stage: leaf_fab_int_std_dev

Output Stage: std_dev_percentage

live ecmp imbalance (Range)

Purpose: Evaluate if standard deviation between spine-facing interfaces on each leaf is within acceptable range. In this case acceptable range is between 0 and std_max facade parameter (in bytes per second unit).

Input Stage: std_dev_percentage

Output Stage: live_ecmp_imbalance

Set of true/false values, each indicating if standard deviation (as a measure of ECMP imbalance) for traffic averages for each spine-facing interface on a given leaf is within acceptable range. Each set member has system_id key to identify leaf whose ECMP imbalance the value represents.

sustained ecmp imbalance (Time in State)

Purpose: Evaluate if standard deviation between spine-facing interfaces on each leaf has been outside acceptable range, (as defined by 'live ecmp imbalance' processor) for more than 'threshold_duration' seconds during last 'total_duration' seconds. These two parameters are part of facade specification.

Input Stage: live_ecmp_imbalance

Output Stage: system_imbalance

systems imbalanced count (Match Count)

Purpose: Count how many systems have ecmp imbalance anomaly true at any instant in time.

Input Stage: system_imbalance

Output Stage: system_imbalance_count

Number of systems with ecmp imbalance.

imbalanced system count out of range (Range)

Purpose: Evaluate if the number of imbalanced systems is within acceptable range, which in this instance means less than 'max_systems_imbalanced' value which is a facade parameter.

Input Stage: system_imbalanced_count

Output Stage: imbalanced_system_count_out_of_range	Boolean indicating if the number of imbalanced systems is within accepted range, i.e. less than 'max_systems_imbalanced" which is a facade parameter.

For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: ECMP Imbalance (Spine to Superspine Interfaces)

The ECMP imbalance (spine to superspine interfaces) probe calculates ECMP imbalance on spine-to-superspine ports. A given set of ECMP links (only calculated on spine-to-superspine links), identified by common system_id, is determined to be imbalanced if the standard-deviation of the tx_bytes counter (averaged periodically over the specified period) for the involved spine interfaces is above "Max Standard Deviation". If such imbalance is observed for more-than "Threshold Duration" the last "Duration" period, we raise an anomaly. The last "Anomaly History Count" anomaly state-changes are stored for observation. If more-than "Max Imbalanced Systems" systems are imbalanced, we raise a distinct anomaly. We maintain for inspection the number of imbalanced systems over the last "System Imbalance History Count" samples.

Instantiate Predefined Probe

Predefined Probe *

ECMP Imbalance (Spine to Superspine Interfaces)

Probe Label *

ECMP Imbalance (Spine to Superspine Interfaces)

Max Standard Deviation

20

Maximum standard deviation in bps across a set of ECMP paths on a given system (in percents of link bandwidth). If this standard deviation is exceeded, we consider that system to be imbalanced

Average Period

30 seconds

Period over which to average input bps counter samples

Threshold Duration

2 minutes 10 seconds

Total amount of time in recent-history during which set of ECMP links must be unbalanced for anomaly to be raised

Duration

5 Minutes

Time period in recent-history over which we will consider ECMP imbalance

Max Imbalanced Systems

1

If this number of total imbalanced systems is exceeded, an anomaly is raised

Generate a probe to calculate ECMP imbalance on spine to superspine ports.

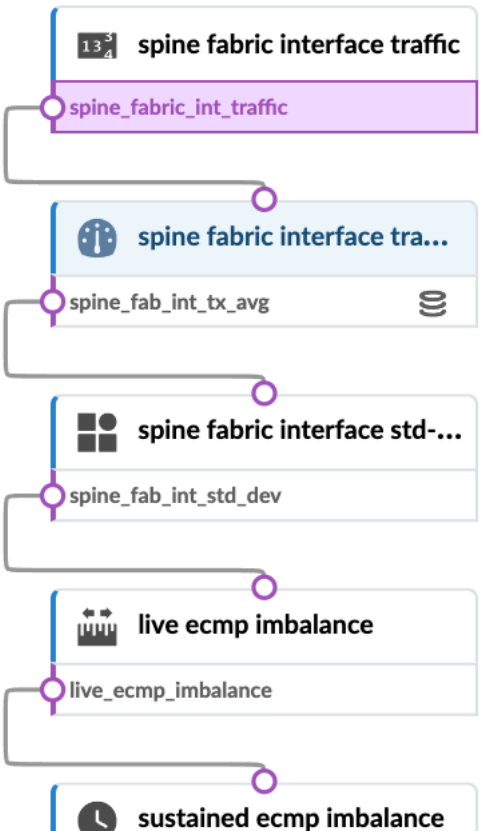
A given set of ECMP links (only calculated on spine to superspine links), identified by common system_id, is determined to be imbalanced if the standard-deviation of the tx_bytes counter (averaged periodically over the specified period) for the involved spine interfaces is above "Max Standard Deviation".

If such imbalance is observed for more-than "Threshold Duration" the last "Duration" period, we raise an anomaly.

The last "Anomaly History Count" anomaly state-changes are stored for observation.

If more-than "Max Imbalanced Systems" systems are imbalanced, we raise a distinct anomaly.

We maintain for inspection the number of imbalanced systems over the last "System Imbalance History Count" samples.



For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: ESI Imbalance

The ESI imbalance probe calculate ESI imbalance. It calculates the standard deviation across links for all ESIs in the network. If any are over the specified threshold in the last specified time period, an anomaly is raised. It also calculates percentage of ESIs in each rack in this state.

Instantiate Predefined Probe

Predefined Probe *

ESI Imbalance

Probe Label *

ESI Imbalance

Max Standard Deviation

20

Maximum standard deviation used for imbalance detection (in percents of link bandwidth).

Duration

1 Minute

Time period in recent-history over which average traffic will be considered

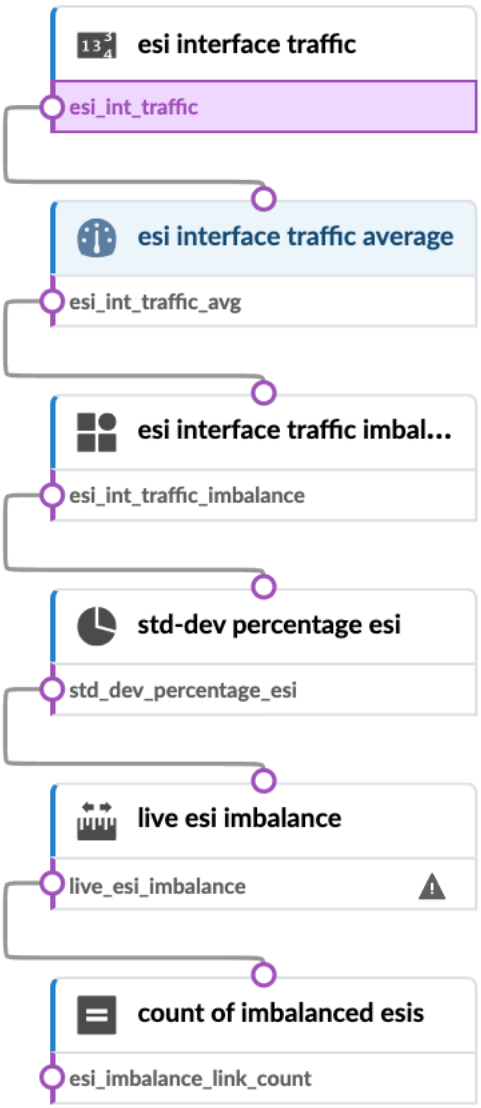
History duration

1 Hour

Time period during which the data of deviation will be retained

Generate a probe to calculate ESI imbalance

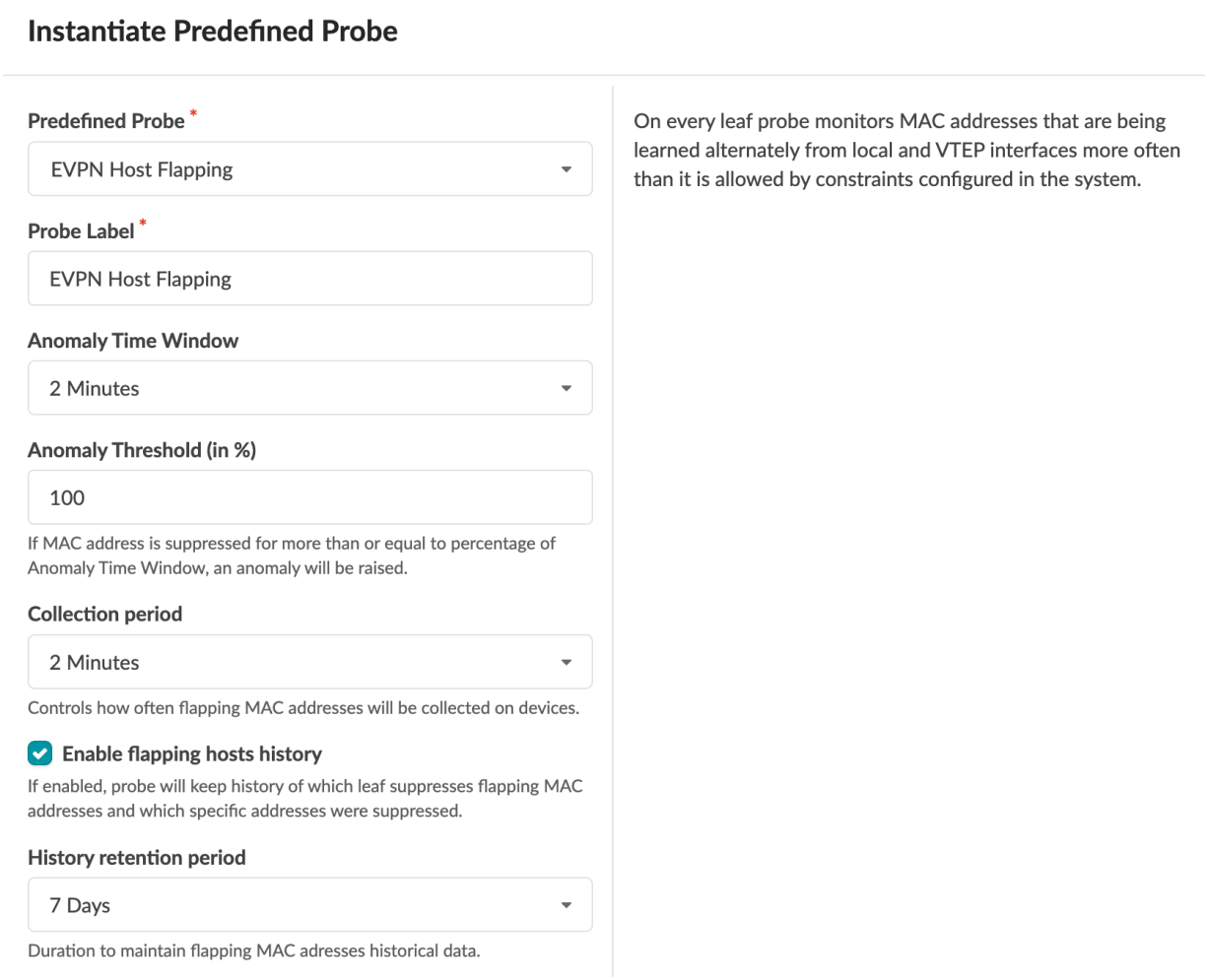
Calculates std deviation across links for all ESIs in the network. If any are over the specified threshold in the last specified time period, an anomaly is raised. Also calculates percentage of ESIs in each rack in this state.

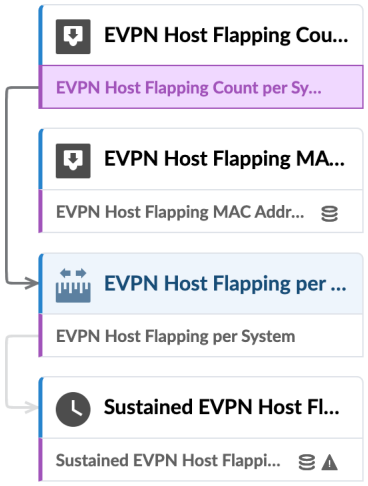


For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: EVPN Host Flapping

EVPN host flaps occur when an L2 loop is mistakenly created under the leaf devices by connecting a hub to two different leaf devices.





For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: EVPN VXLAN Type-3 Route Validation

The EVPN VXLAN Type-3 route validation probe validates EVPN Type-3 routes on every leaf in the network. It collects appropriate telemetry data, compares it to the set of Type-3 routes expected to be present and alerts if expected routes are missing on any device.

You can configure the following parameters:

- **Probe Label**: Name to identify the probe.
- **Anomaly Time Window** : Average period duration for interface counters.
- **Anomaly Threshold (in %)**: If routes are missing for more than or equal to percentage of Anomaly Time Window, an anomaly is raised. If Anomaly Time Window ATW, and Anomaly Threshold is AT. It calculates Z = (ATW * AT)/100 in seconds. E.g. If ATW = 20 seconds, AT = 5%, then Z = (20 * 5)/100 = 1 second. When the route is in Missing state for Z seconds from total ATW duration, anomaly is raised.
- **Collection period**: All these probes are polling-based so they have a polling period.
- **Monitored VNs**: Specify the virtual networks to be monitored. Either list of desired VN's e.g. "1-3,6,8,10-13" or " * " to monitor all virtual networks.

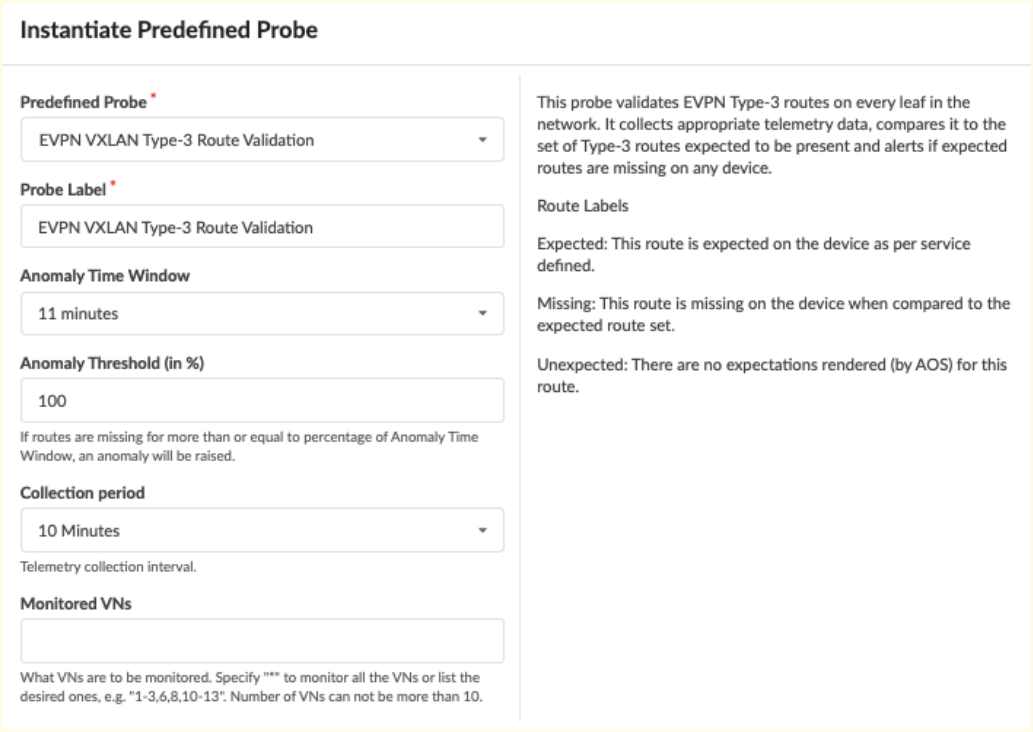
The route labels include the following:

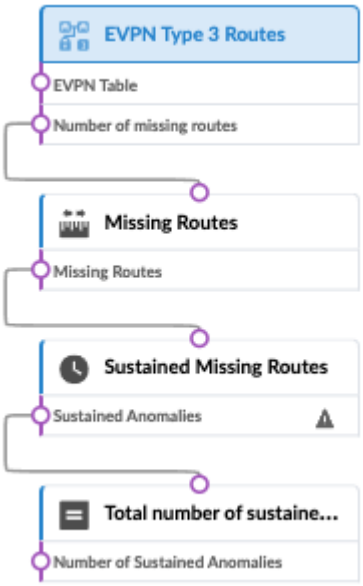
- **Expected**: This route is expected on the device as per service defined.
- **Missing**: This route is missing on the device when compared to the expected route set.

- **Unexpected**: There are no expectations rendered (by AOS) for this route.

This probe is created with an empty **Monitored VNs** (monitored_vn) list, which means that the probe does not monitor any virtual networks by default. When you instantiate this probe you must specify a list of virtual networks (up to ten) for which routes are collected, or you can specify " * " in which case all virtual networks are monitored.

CAUTION: Specifying " * " in the **Monitored VNs** field may result in high cpu/memory/ network I/O overhead associated with BGP routing table iteration on the device side.





NOTE: Auto-enabling the **EVPN VXLAN Route Summary** analytics dashboard enables the **EVPN VXLAN Type-3 Route Validation** and **EVPN Flood List Validation** probes automatically (but not the EVPN VXLAN Type-5 Route Validation probe). See `Configuring Auto-Enabled Dashboards<configure_dashboard>` for information about enabling the dashboard.

For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: EVPN VXLAN Type-5 Route Validation

The EVPN VXLAN Type-5 route validation probe validates the EVPN Type 5 routes on every leaf. The collected data is matched against the graph data to ascertain any missing routes on any system.

You can configure the following parameters:

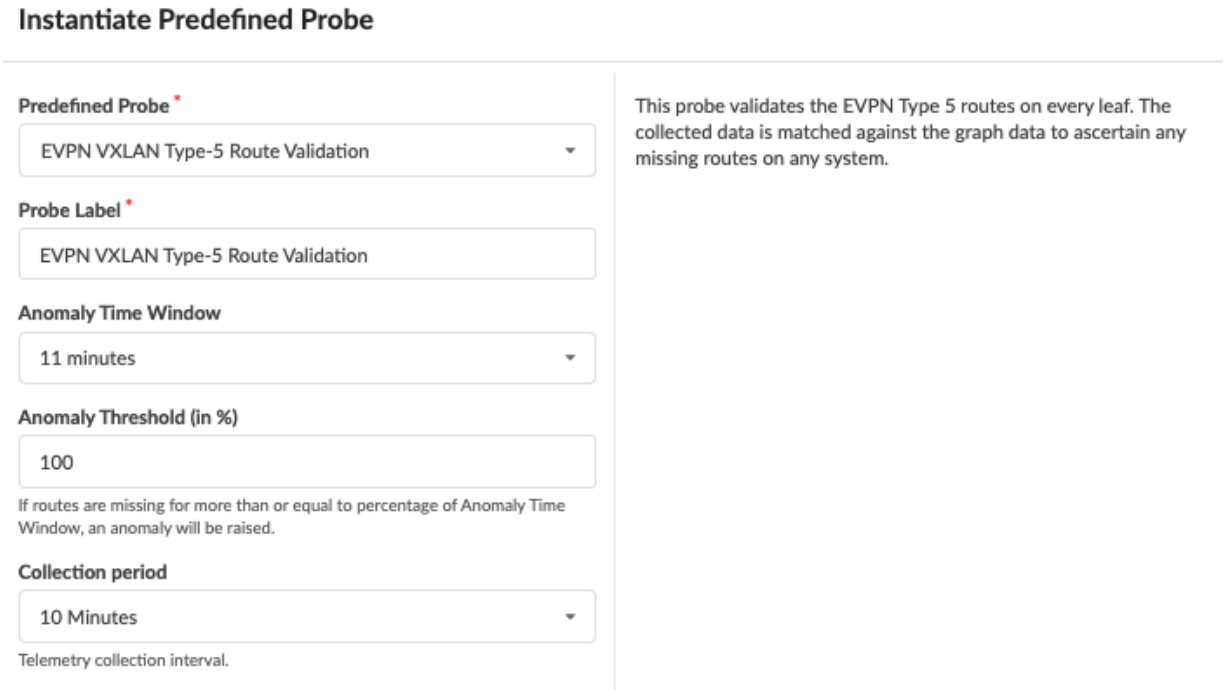
- **Probe Label**: Name to identify the probe.
- **Anomaly Time Window** : Average period duration for interface counters.
- **Anomaly Threshold (in %)**: If routes are missing for more than or equal to percentage of Anomaly Time Window, an anomaly is raised. If Anomaly Time Window ATW, and Anomaly Threshold is AT. It calculates Z = (ATW * AT)/100 in seconds. E.g. If ATW = 20 seconds, AT = 5%, then Z = (20 * 5)/100 = 1 second. When the route is in Missing state for Z seconds from total ATW duration, anomaly is raised.

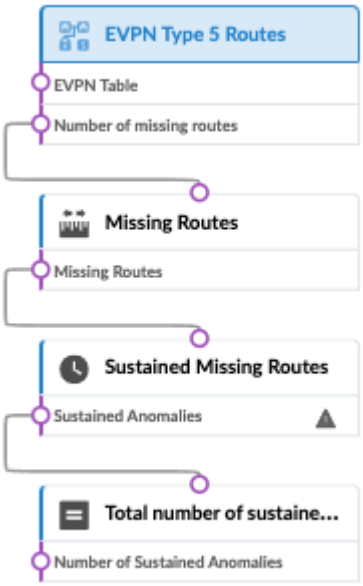
- **Collection period**: All these probes are polling-based so they have a polling period.

The route labels include the following:

- **Expected**: This route is expected on the device as per service defined.
- **Missing**: This route is missing on the device when compared to the expected route set.
- **Unexpected**: There are no expectations rendered (by AOS) for this route.

If this probe is enabled it monitors all virtual networks from all devices. It does not provide the “monitored VN list” configuration option like the VXLAN Type-3 probe does.





NOTE: Auto-enabling the **EVPN VXLAN Route Summary** analytics dashboard enables the **EVPN VXLAN Type-3 Route Validation** and **EVPN Flood List Validation** probes automatically (but not the EVPN VXLAN Type-5 Route Validation probe). See `Configuring Auto-Enabled Dashboards<configure_dashboard>` for information about enabling the dashboard.

For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: External Routes

Purpose The External Routes probe automatically activates the collection of received or advertised routes across all BGP sessions established with generic systems into a single stage output table (mixing received, used and advertised routes). This probe assists with troubleshooting external network connectivity problems.

Parameters The External Routes probe parameters below can be configured at time of creation or anytime afterwards.

AFI: Address Family Identifiers - IPv4 or IPv6

Type: advertised-routes or received-routes

Routing Zone (VRF): All or specific name

Prefix: Only routes matching the prefix

Filter options: exact or longer

More-specific prefixes mask: Match more-specific prefixes from a parent prefix, up until le_mask prefix length.

Less-specific prefixes mask: Match less-specific prefixes from a parent prefix, up from ge_mask to the prefix length of the route.

For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: Hot/Cold Interface Counters (Fabric Interfaces)

Purpose

This probe determines hot/cold interface counters. It determines if interface counters are hot (too high) or cold (too low). A given interface (considering only leaf fabric interfaces) is considered to be in a hot state if its average counter value is greater than "Max". A given interface (considering only leaf fabric interfaces) is considered to be in a cold state if its average counter value is less than "Min". If such undesired state is observed for more-than "Threshold Duration" over the last "Duration" period, an anomaly is raised. Distinct anomalies are raised for hot and cold states. If more than "Max Hot Interface Percentage" percent of interfaces on a given device are hot, we raise an anomaly. If more than "Max Cold Interface Percentage" percent of interfaces on a given device are cold, we raise an anomaly. Finally, the last "Anomaly History Count" anomaly state-changes are stored for observation.

Source Processor

leaf interface traffic (Interface Counters)

Purpose: wires in interface traffic samples (measured in bytes per second) from each spine facing interface on each leaf.

Output Stage: leaf_int_traffic

Set of traffic samples (for each spine-facing interface on each leaf). Each set member has the following keys to identify it: system_id (id of the leaf system, usually serial number), interface (name of the interface), role (role of the interface, such as 'fabric').

Additional Processor(s)

leaf interface tx avg (Periodic Average)

Purpose: Calculate average traffic during period specified by average_period facade parameter. Unit is bytes per second.

Input Stage: leaf_int_traffic

Output Stage: leaf_int_tx_avg

Set of traffic average values (for each spine-facing interface on each leaf). Each set member has the

following keys to identify it: system_id (id of the leaf system, usually serial number), interface (name of the interface), role (role of the interface, such as 'fabric').

interface sum per device (Sum)

Purpose: Sum average traffic for all interface under consideration per device.

Input Stage: leaf_int_tx_avg

Output Stage: if_counter_sum_per_device	Set of numbers, each indicating the total average traffic for all interface under consideration per device, expressed in bytes per second. Each set member has the following key to identify it: system_id (id of the leaf system, usually serial number).

interface sum per device per link role (Sum)

Purpose: Sum average traffic for all interface under consideration per device, per interface role.

Input Stage: leaf_int_tx_avg

Output Stage: if_counter_sum_per_device_role	Set of numbers, each indicating the total average traffic for all interface under consideration per device, expressed in bytes per second. Each set member has the following keys to identify it: system_id (id of the leaf system, usually serial number), role (role of the interface, such as 'fabric').

live leaf interface cold (Range)

Purpose: Evaluate if the average traffic on spine facing interfaces on each leaf is within acceptable range. In this case acceptable range means larger than min facade parameter (in bytes per second unit).

Input Stage: leaf_int_tx_avg

Output Stage: live_leaf_int_cold	Set of true/false values, each indicating if traffic averages for each spine-facing interface on each leaf is within acceptable range. Each set member has the following keys to identify it: system_id (id of the leaf system, usually serial number), interface (name of

the interface) role (role of the interface, such as 'fabric'). Samples unit is bytes per second.

live leaf interface hot (Range)

Purpose: Evaluate if the average traffic on spine-facing interfaces on each leaf is within acceptable range. In this case acceptable range is between 0 and max facade parameter (in bytes per second unit).

Input Stage: leaf_int_tx_avg

Output Stage: live_leaf_int_hot

Set of true/false values, each indicating if traffic averages for each spine-facing interface on each leaf is within acceptable range. Each set member has the following keys to identify it: system_id (id of the leaf system, usually serial number), interface (name of the interface) role (role of the interface, such as 'fabric'). Samples unit is bytes per second.

sustained cold leaf interface (Time in State)

Purpose: Evaluate if the average traffic spine facing interfaces on each leaf has been outside acceptable range, (as defined by 'live leaf interface cold' processor) for more than 'threshold_duration' seconds during the last 'total_duration' seconds. These two parameters are part of facade specification.

Input Stage: live_leaf_int_cold

Output Stage: cold_leaf_int

Set of true/false values, each indicating if the traffic average for each spine-facing interface on each leaf has been in 'cold' range for more than specified period of time. Each set member has the following keys to identify it: system_id (id of the leaf system, usually serial number), interface (name of the interface) role (role of the interface, such as 'fabric'). Samples unit is bytes per second.

sustained hot leaf interface (Time in State)

Evaluate if the average traffic spine facing interfaces on each leaf has been outside acceptable range, (as defined by 'live leaf interface hot' processor) for more than 'threshold_duration' seconds during the last 'total_duration' seconds. These two parameters are part of facade specification.

Input Stage: live_leaf_int_hot

	Output Stage: hot_leaf_int	Set of true/false values, each indicating if the traffic average for each spine-facing interface on each leaf has been in 'hot' range for more than specified period of time. Each set member has the following keys to identify it: system_id (id of the leaf system, usually serial number), interface (name of the interface) role (role of the interface, such as 'fabric'). Samples unit is bytes per second.

system percent cold (Match Percentage)

Purpose: Calculate percentage of interfaces that are cold on any given device under consideration.

Input Stage: cold_leaf_int

Output Stage: system_perc_cold	Set of numbers, each indicating the the percentage of cold interfaces on any given device under consideration. Each set member has the following key to identify it: system_id (id of the leaf system, usually serial number).

system percent hot (Match Percentage)

Purpose: Calculate percentage of interfaces that are hot on any given device under consideration.

Input Stage: hot_leaf_int

Output Stage: system_perc_hot	Set of numbers, each indicating the the percentage of hot interfaces on any given device under consideration. Each set member has the following key to identify it: system_id (id of the leaf system, usually serial number).

device cold (Range)

Purpose: Evaluate if the percentage of cold interfaces on a specific device is outside the acceptable range, where acceptable range in his case means less than 'max_cold_interface_percentage', which is a facade parameter.

Input Stage: system_perc_cold

Output Stage: device_cold_anomalous	Set of boolean values, each indicating if the the percentage of cold interfaces on any given device was out of acceptable range. Each set member has the following key to

	identify it: system_id (id of the leaf system, usually serial number).
device hot (Range)	Purpose: Evaluate if the percentage of hot interfaces on a specific device is outside the acceptable range, where acceptable range in his case means less than 'max_hot_interface_percentage', which is a facade parameter. Input Stage: system_perc_hot
Output Stage: device_hot_anomalous	Set of boolean values, each indicating if the the percentage of hot interfaces on any given device was out of acceptable range. Each set member has the following key to identify it: system_id (id of the leaf system, usually serial number).

For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: Hot/Cold Interface Counters (Specific Interfaces)

The hot/cold interface counters (specific interfaces) probe determines hot/cold specific interface counters. It determines if interface counters averaged over "Average Period" are hot (too high) or cold (too low). A given interface (out of the specified list) is considered to be in a hot state if its average counter value is greater than "Max". A given interface (out of the specified list) is considered to be in a cold state if its average counter value is less than "Min". If such undesired state is observed for more-than "Threshold Duration" over the last "Duration" time period, we raise an anomaly. Distinct anomalies are raised for hot and cold states. If more than "Max Hot Interface Percentage" percent of interfaces on a given device are hot, we raise an anomaly. If more than "Max Cold Interface Percentage" percent of interfaces on a given device are cold, we raise an anomaly. Finally, the last "Anomaly History Count" anomaly state-changes are stored for observation.

Instantiate Predefined Probe

Predefined Probe *

Hot/Cold Interface Counters (Specific Interfaces)

Probe Label *

Hot/Cold Interface Counters (Specific Interfaces)

Interfaces *

No interfaces specified.

+ Add Interface

Counter Type *

A type of an interface counter.

Min

0

Minimum level of counter

Max

10

Maximum level of counter

Max Cold Interface Percentage

30

Maximum percentage of cold interfaces on a device

Max Hot Interface Percentage

30

Maximum percentage of hot interfaces on a device

Average Period

1 Minute

Period over which to average input counter samples

Threshold Duration

10 seconds

Total amount of time in recent-history during which interface must be hot/cold for anomaly to be raised

Duration

1 Minute

Time period in recent-history over which interface counter hot/cold status will be considered

Generate a probe to determine hot/cold specific interface counters

This probe determines if interface counters averaged over "Average Period" are hot (too high) or cold (too low).

A given interface (out of the specified list) is considered to be in a hot state if its average counter value is greater than "Max"

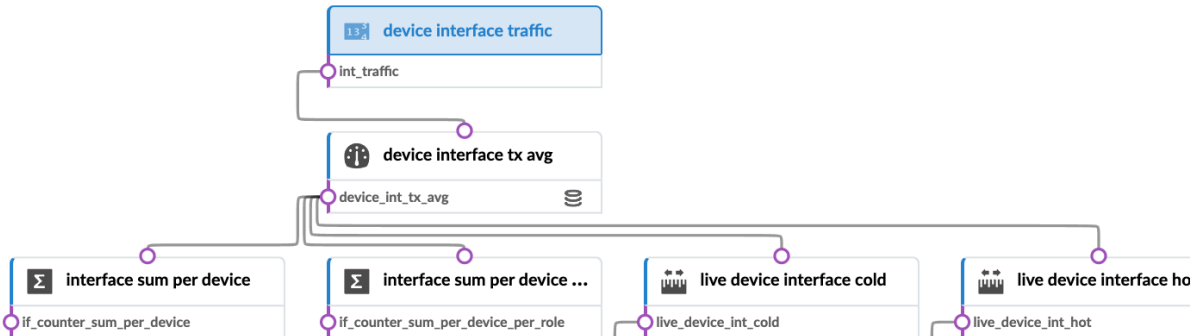
A given interface (out of the specified list) is considered to be in a cold state if its average counter value is less than "Min"

If such undesired state is observed for more-than "Threshold Duration" over the last "Duration" time period, we raise an anomaly. Distinct anomalies are raised for hot and cold states.

If more than "Max Hot Interface Percentage" percent of interfaces on a given device are hot, we raise an anomaly.

If more than "Max Cold Interface Percentage" percent of interfaces on a given device are cold, we raise an anomaly.

Finally, the last "Anomaly History Count" anomaly state-changes are stored for observation.



For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: Hot/Cold Interface Counters (Spine to Superspine Interfaces)

The hot/cold interface counters (spine-to-superspine interfaces) probe calculates ECMP imbalance on spine-to-superspine ports. A given set of ECMP links (only calculated on spine-to-superspine links), identified by common system_id, is determined to be imbalanced if the standard-deviation of the tx_bytes counter (averaged periodically over the specified period) for the involved spine interfaces is above "Max Standard Deviation". If such an imbalance is observed for more-than "Threshold Duration" the last "Duration" period, we raise an anomaly. The last "Anomaly History Count" anomaly state-changes are stored for observation. If more-than "Max Imbalanced Systems" systems are imbalanced, we raise a distinct anomaly. We maintain for inspection the number of imbalanced systems over the last "System Imbalance History Count" samples.

Instantiate Predefined Probe

Predefined Probe *

Hot/Cold Interface Counters (Spine to Superspine Interfaces)

Probe Label *

Hot/Cold Interface Counters (Spine to Superspine Interfaces)

Counter Type *

A type of an interface counter.

Min

0

Minimum level of counter

Max

10

Maximum level of counter

Max Cold Interface Percentage

30

Maximum percentage of cold interfaces on a device

Max Hot Interface Percentage

30

Maximum percentage of hot interfaces on a device

Average Period

1 Minute

Period over which to average input counter samples

Threshold Duration

10 seconds

Total amount of time in recent-history during which interface must be hot/cold for anomaly to be raised

Duration

1 Minute

Time period in recent-history over which interface counter hot/cold status will be considered

Generate a probe to determine hot/cold spine to superspine interface counters.

This probe determines if interface counters are hot (too high) or cold (too low).

A given interface (considering only spine to superspine interfaces) is considered to be in a hot state if its average counter value is greater than "Max"

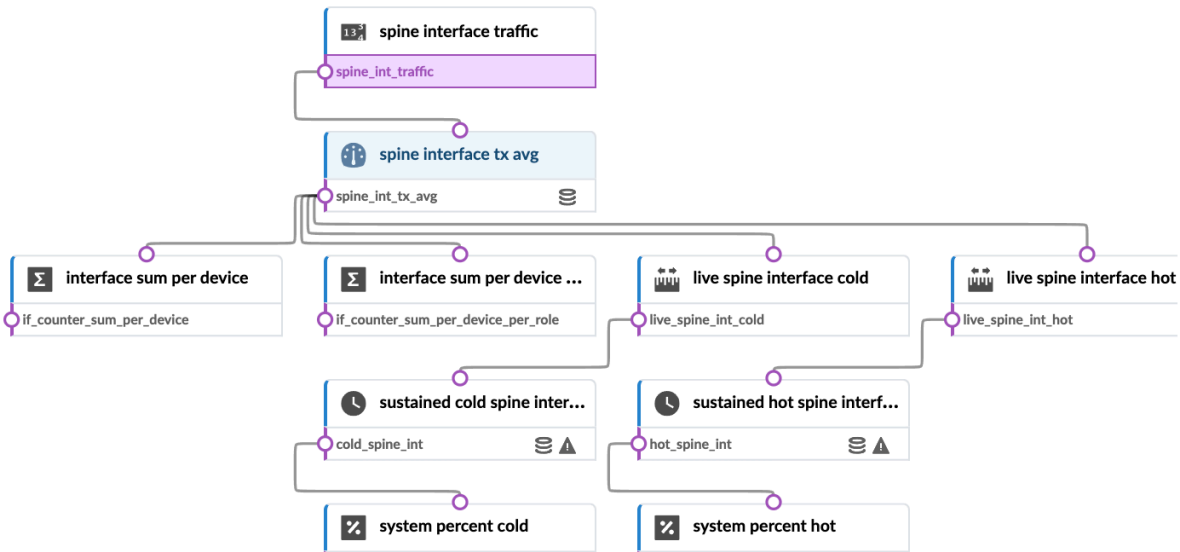
A given interface (considering only spine to superspine interfaces) is considered to be in a cold state if its average counter value is less than "Min"

If such undesired state is observed for more-than "Threshold Duration" over the last "Duration" time period, we raise an anomaly. Distinct anomalies are raised for hot and cold states.

If more than "Max Hot Interface Percentage" percent of interfaces on a given device are hot, we raise an anomaly.

If more than "Max Cold Interface Percentage" percent of interfaces on a given device are cold, we raise an anomaly.

Finally, the last "Anomaly History Count" anomaly state-changes are stored for observation.



For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: Hypervisor and Fabric LAG Config Mismatch Probe (Virtual Infra)

Purpose	Detect inconsistent LAG configs between fabric and virtual infra and calculate LAGs missing on hypervisors and managed leaf devices connected to hypervisors.	
Source Processor	**Hypervisor NICs with LAG (generic graph collector)**	output stage: Hypervisor NICs LAG Intent Status (discrete state set) (generated from graph)
Additional Processor(s)	**Hypervisor NIC LAG anomalies (state)**	input stage: Hypervisor NICs LAG Intent Status output stage: Hypervisor NIC LAG Mismatch Anomaly (discrete state set)

Example Usage

vSphere Integration - This probe detects inconsistent LAG configs between fabric LAG dual-leaf devices and ESXi hosts. LACP mode information is collected from the fabric LAG dual-leaf devices and also connects to vCenter API and collects LAG groups and members per hypervisor.

> **NOTE**: Current validation is done on vCenter virtual Distributed Switches only, not on virtual Standard Switches. LLDP must be enabled on vCenter vDS switches.

Anomalies are raised if any of the following occurs:

- LAG member ports on ToR are connected to non-LAG physical ports on ESXi.
- Non-LAG member ports on ToR are connected to LAG physical ports on ESXi.

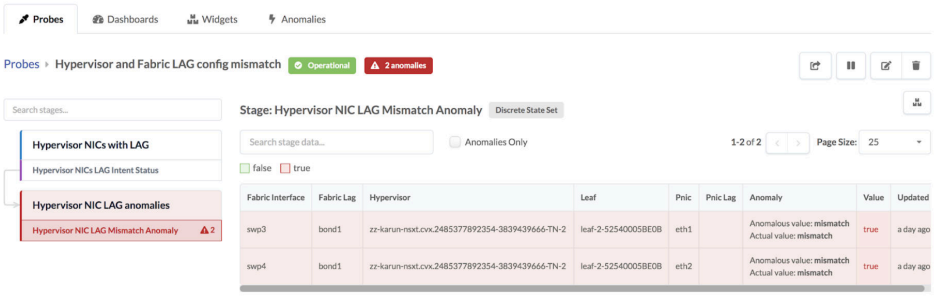
NSX Integration - Enabling this probe activates a continuous LAG validation between NSX-T transport nodes and data center fabric. It validate that LAGs are properly configured between fabric LAG dual-leaf devices and NSX-T transport nodes. The NSX-T uplink profile defines the network interface configuration facing the fabric in terms of LAG and LACP config. Network interface misconfiguration between the transport node and the ToR switch is validated and detected.

Anomalies are raised in the following circumstances:

- NSX-T transport nodes are not configured for LAG but ToR has LAG member ports in the fabric.
- ESXi hosts are dual-attached to ToR leaf devices but corresponding NSX-T transport nodes are “single-attached” or they are using “NIC-teaming” using active-standby or load-balanced config.

1. Add NSX-T API user as a Virtual Infra.
2. Add NSX-T Manager in the blueprint (External Systems > Virtual Infra Managers).
3. Enable this probe (Hypervisor and Fabric LAG config mismatch).

Let’s say in the NSX-T uplink profile, LAG is deleted but the fabric has LAG in terms of ToR leaf devices having LAG member ports. As a result in a blueprint after enabling this probe LAG mismatch anomalies are raised.



Since the LAG on the NSX-T transport nodes has been deleted, there is a mismatch between physical network adapter (pnic) on ESXi host LAG configuration and LAG configuration on ToR leaf devices.

For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: Hypervisor and Fabric VLAN Config Mismatch

IN THIS SECTION

Hypervisor & Fabric VLAN Config Mismatch Probe Overview

Purpose

Calculate VLAN mismatch between configured virtual networks on leaf devices and VLANs needed by VMs running on hypervisors attached to leaf devices. (Formerly known as Virtual Infra VLAN Match). Detects misconfiguration of hypervisor trunk logical switches when VLAN tag is configured inside a VM (not on the bridge itself).

Source Processors

Fabric configured VLAN configs (generic graph collector)

output stage: Fabric VLAN configs (number set) (generated from graph)

Hypervisor expected VLAN configs (generic graph)

output stage: Hypervisor VLAN configs (number set)

Additional Processor(s)

Hypervisor unique VLAN configs (set count)

input stage: Hypervisor VLAN configs

output stage: Hypervisor unique VLAN configs (number set)

Differences between Hypervisor and Fabric (set comparison)

input stages:

Hypervisor unique VLAN configs

Fabric VLAN configs

output stages:

Common in Fabric and Hypervisor (number set)

Fabric Only (number set)

Hypervisor Only (number set)

Fabric missing VLAN configs accumulator (accumulate)

input stage: Hypervisor Only

output stage: Hypervisor Only TimeSeries (number set time series)

Hypervisor missing VLAN configs accumulator (accumulate)

input stage: Fabric Only

output stage: Fabric Only TimeSeries (number set time series)

Check for Fabric missing VLAN configs (range)

input stage: Hypervisor Only TimeSeries

output stage: Fabric missing VLAN configs anomaly (discrete state set)

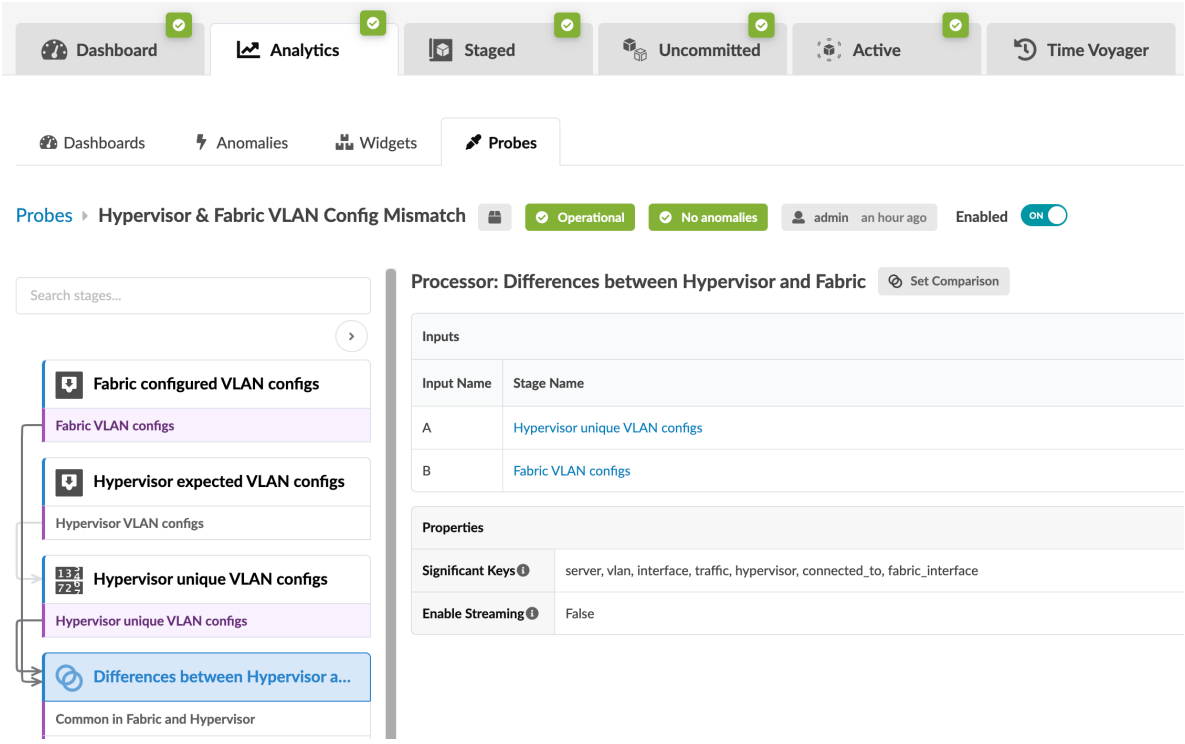
Check for Hypervisor missing VLAN configs (range)

input stage: Fabric Only TimeSeries

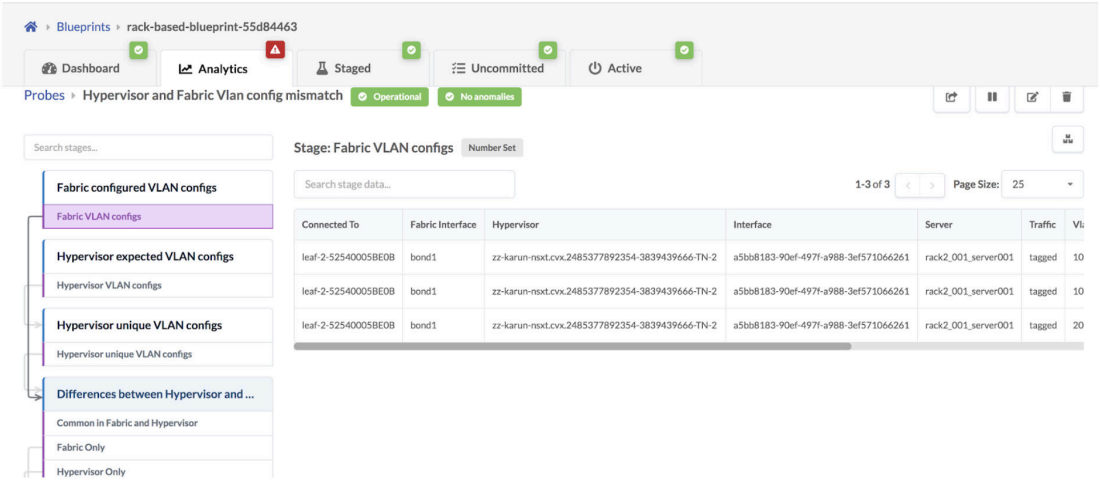
output stage: Hypervisor missing VLAN configs anomaly (discrete state set)

Usage with NSX-T Integration

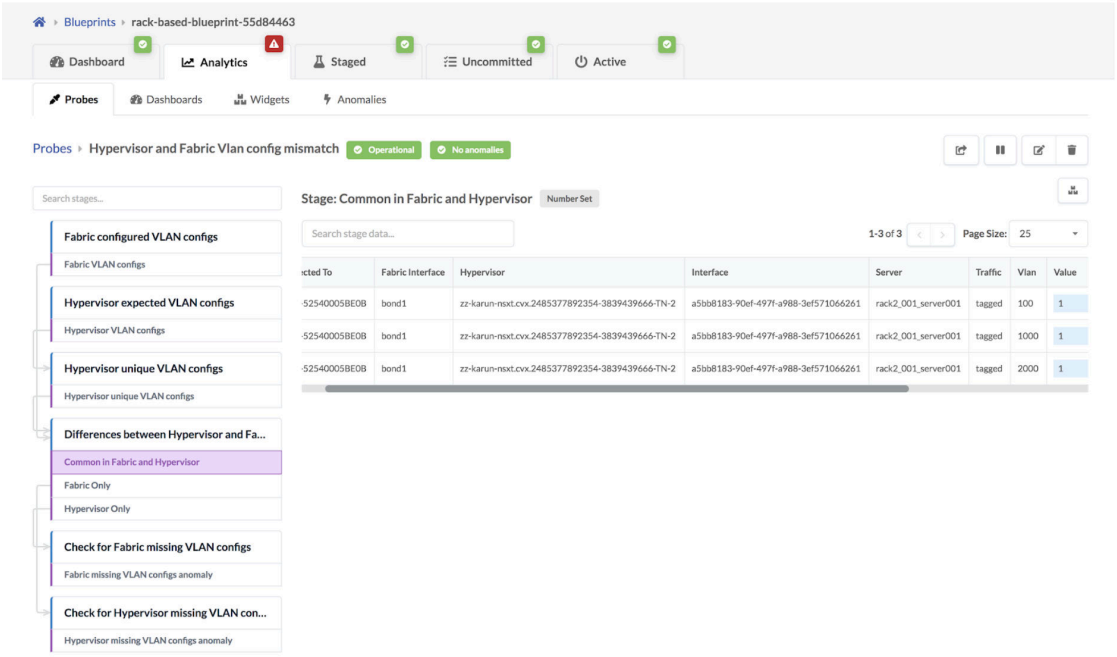
1. From the blueprint, navigate to **Analytics > Probes** and click **Hypervisor & Fabric VLAN Config Mismatch** in the probe name list to go to its details. When the VLANs between the data center fabric and the NSX-T transport nodes match, then the probe looks similar to the image below:



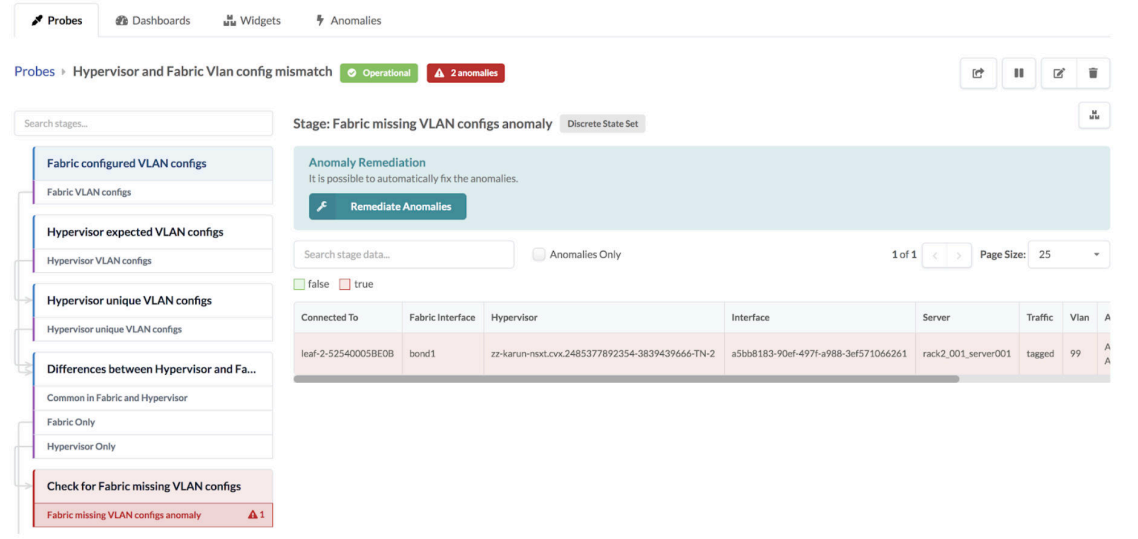
2. Click the **Fabric VLAN Configs** stage to show the VLANs tagged towards NSX-T transport nodes on fabric ToR leaf devices as shown below:



3. Click the **Common in Fabric and Hypervisor** stage to show that VLANs in the NSX-T transport nodes and the fabric match.



If the VLAN defined in the Uplink Transport Zone used for BGP peering is modified in the NSX-T Manager, then VLAN mismatch anomalies are raised.



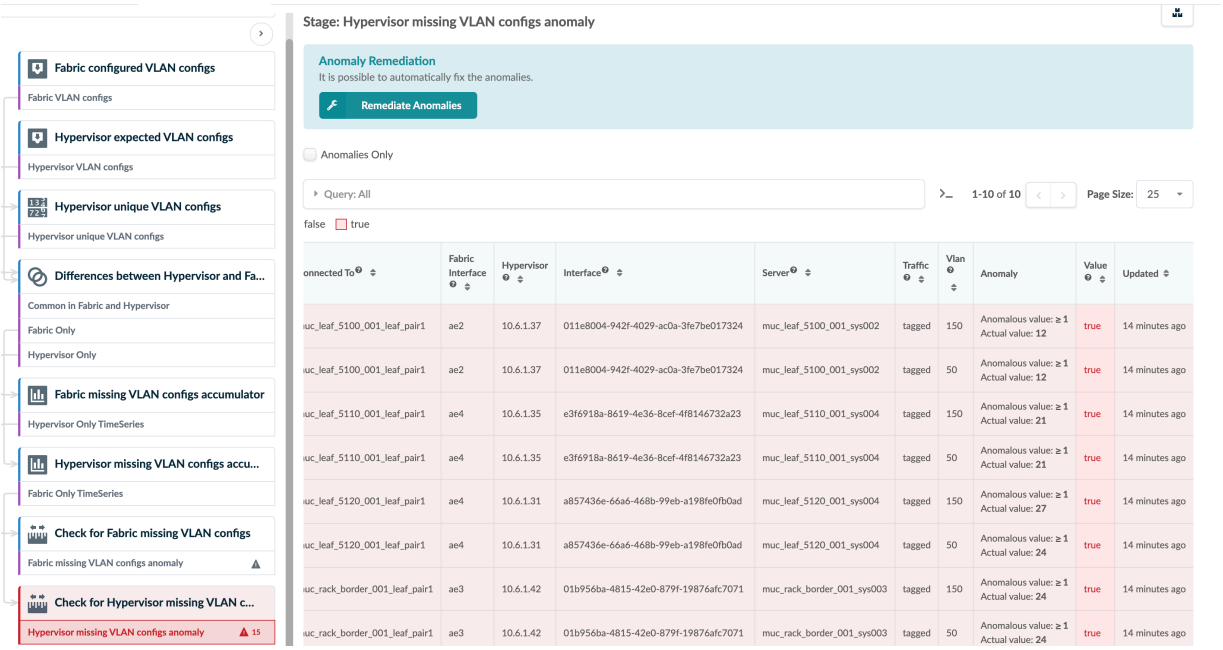
Some other reasons for mismatching include the following:

- If the configured VLAN NSX-T transport node is missing in the fabric.
- If the configured VLAN NSX-T transport node is in the fabric, but the end VMs or servers are not part of this virtual network or VLAN.
- If a segment is created in NSX-T for either an overlay or VLAN-based transport zone. It could be that the configured VLAN spanning the logical switch/segment on the transport node is missing on the fabric.
- If L2 bridging for VMs in different overlay logical segments is broken because one VM exists in one logical switch/segment and the other VM exists in a separate uplink logical switch/segment.

As an example, a VLAN is missing in NSX-T 3.0 Host Transport node on the Overlay segment connected to ToR leaf devices and respective VXLAN VN is present in Juniper Apstra Fabric and ports towards

Hypervisors are assigned in a **Virtual Network** based Connectivity Template as below:

A **Hypervisor missing VLAN Configs** anomaly is raised as shown below:

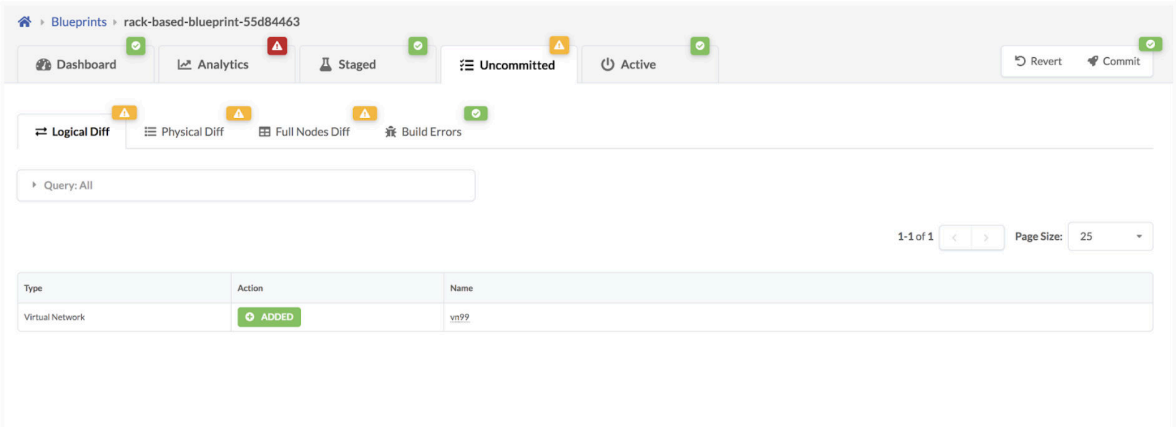


In some scenarios, a VLAN mismatch anomaly can be remediated. If so, the **Remediate Anomalies** button appears on the probe details page as shown in the screenshot above. Example scenarios include:

- NSX-T transport nodes use an uplink profile to define transport VLAN over which overlay tunnel comes up. Fabric could be missing the rack-local VN for transport VLAN on hypervisors. One-click remediation can be provided by creating a new rack-local virtual network with the proper VLAN ID in the fabric.

- A rack-local virtual network is defined with VLAN ID Y, however, the connected virtual infra nodes (i.e hypervisors) do not have the VLAN ID in the logical segment/switch. One-click remediation can be provided by removing the endpoint from the affected VLAN ID.

If the **Remediate Anomalies** button appears under the stage name, you can click it to automatically stage the changes required to remediate the anomaly. You can see the staged changes on the **Uncommitted** tab.



Review the staged configuration, add any necessary resources (such as IP subnet address, virtual gateway IP, as so on), then commit the configuration.

Usage with VCenter Integration

Some anomalies, that are raised because of a VLAN config mismatch between vCenter and the fabric, can automatically be remediated, such as the following.

- If the vCenter Distributed Virtual Switch (vDS) port group does not have a corresponding rack-local VN (VLAN) for VLAN ID X. With one-click remediation, a new rack-local virtual network (VLAN) with the proper VLAN ID is created.
- If endpoint X in a rack-local VN with VLAN ID Y, does not have a corresponding dVS port group. With one-click remediation, the endpoint is removed from the affected VLAN ID.

Note

vCenter vDS must be used with VLAN specific ID allocation on the port group for L2 network segmentation at the hypervisor level.

A VLAN-based rack-local virtual network is extending each VLAN segment defined on the vDS, across servers within the same rack. For example, vDS port group VLAN 10 = rack-local virtual network with VLAN 10.

For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: Hypervisor MTU Mismatch Probe (Virtual Infra - NSX-T Only)

Purpose	NSX-T Only - Detect maximum transmission unit (MTU) value deviations across hypervisor physical network adapters (pnics).	
Source Processor	**Interface MTU (generic graph collector)**	output stage: Interface MTU (number set) (generated from graph)
Additional Processor(s)	**Check MTU mismatch between hypervisors (standard deviation)**	input stage: Interface MTU output stage: Hypervisor MTU Deviation (number set)
	MTU Mismatch (range)	input stage: Hypervisor MTU Deviation (number set) output stage: MTU Mismatch (discrete state set)
Example Usage	**NSX Integration** - If validation fails between NSX-T nodes and the controller in terms of mismatch of minimum configured MTU to support Geneve encapsulation or if the VLANs defined on NSX-T nodes are not configured on ToR leaf interfaces connecting an NSX node to the fabric, then anomalies are raised.	

For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

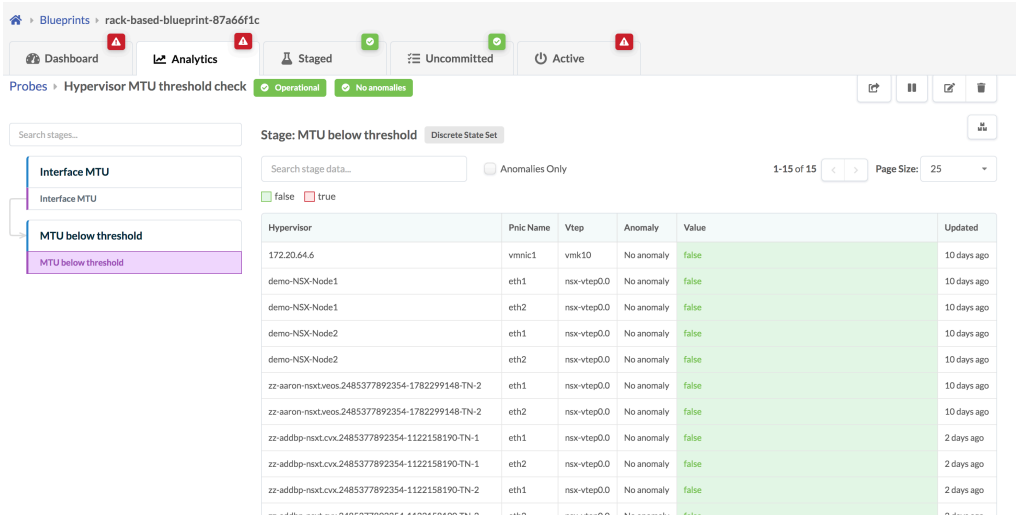
Probe: Hypervisor MTU Threshold Check Probe (Virtual Infra)

Purpose	Detect virtual infra interfaces with maximum transmission units (MTU) below a specified threshold (default: 1600).	
Source Processor	**Interface MTU (generic graph collector)**	output stage: Interface MTU (number set) (generated from graph)
Additional Processor(s)	**MTU below threshold (range)**	input stage: Interface MTU output stage: MTU below threshold (discrete state set)

Example Usage

NSX Integration - To carry VXLAN-encapsulated overlay traffic, an MTU greater than 1600 is recommended. NSX-T transport nodes connected to ToR leaf devices that are below the specified threshold are detected.

To support Geneve encapsulation, the MTU configuration on NSX-T nodes involved in an overlay transport zone must have a valid MTU setting on the ESXi host. The image (from a previous Apstra version) below shows hypervisors with the MTU above the threshold.



If any of the hypervisors were below the threshold, the expected value would change to **true** and an anomaly would be raised.

For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: Hypervisor Missing LLDP Config Probe (Virtual Infra)

Purpose

Detect virtual infra hosts that are not configured for LLDP. (Formerly known as Virtual Infra missing LLDP config).

Source Processor

Hypervisor NIC LLDP Config (generic graph)	output stage: Hypervisor NIC LLDP config (discrete state set) (generated from graph)

Additional Processor(s)

LLDP config by switch (match count)	input stage: Hypervisor NIC LLDP config output stage: LLDP config by switch (number set)
Switches missing LLDP config (range)	input stage: LLDP config by switch

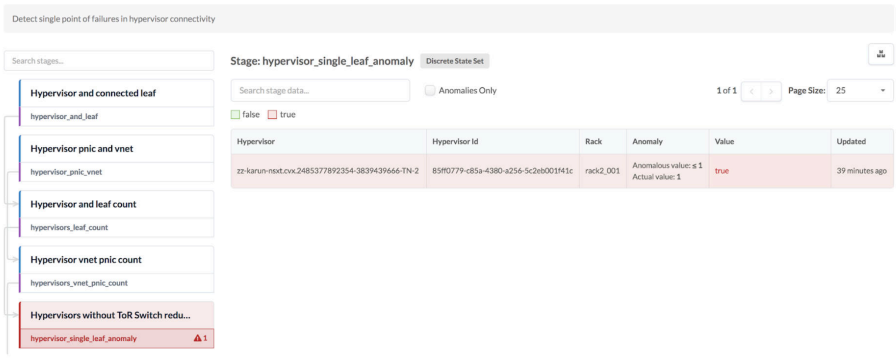
		output stage: Switches missing LLDP config anomaly (discrete state set)
Example Usage	**VMware Integration** - If LLDP information is missing on ToR connected to physical ports on ESXi, an anomaly is raised.	

For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: Hypervisor Redundancy Checks Probe (Virtual Infra)

Purpose	Detect hypervisor redundancy.	
Source Processors	**Hypervisor and connected leaf (generic graph)**	output stage: hypervisor_and_leaf (text set) (generated from graph)
	Hypervisor pnic and vnet (generic graph collector)	output stage: hypervisor_pnic_vnet (text set) (generated from graph)
Additional Processor(s)	**Hypervisor and leaf count (set count)**	input stage: hypervisor_and_leaf output stage: hypervisors_leaf_count (number set)
	Hypervisor vnet pnic count (set count)	input stage: hypervisors_pnic_vnet output stage: hypervisors_vnet_pnic_count (number set)
	Hypervisor without ToR Switch redundancy (range)	input stage: hypervisors_leaf_count output stage: hypervisor_single_leaf_anomaly (discrete state set)
	Networks without link redundancy (range)	input stage: hypervisors_vnet_pnic_count output stage: hypervisor_vnet_single_pnic_anomaly (discrete state set)
Example Usage	**NSX-T Integration** - an anomaly is raised in cases without redundancy or a single point of failure (SPOF) in hypervisor connectivity. Examples include: • NSX-T transport nodes with a single non-LAG uplink towards ToR leaf devices in the fabric can result in a single point of failure (SPOF) for overlay traffic.	

- NSX-T transport nodes with a single LAG uplink with both members going to a single ToR leaf can result in a single point of failure (SPOF).
- Lack of redundancy between fabric LAG dual-leaf devices and ESXi hosts.



For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: Interface Flapping (Fabric Interfaces)

Purpose

This probe determines if fabric interfaces are flapping. A given interface (considering only fabric interfaces) is considered to be flapping if it transitions state more than "Threshold" times over the last "Duration". Such flapping will cause an anomaly to be raised. If more-than "Max Flapping Interfaces Percentage" percent of interfaces on a given device are flapping, an anomaly will be raised for that device. Finally, the last "Anomaly History Count" anomaly state-changes are stored for observation.

Source Processor

leaf fab int status (Service Data Collector)

Purpose: wires in interface status telemetry for all fabric interfaces on the leaf devices.

Output Stage: leaf_if_status

Set of operational states ("up" or "down"). Each set member corresponds to a leaf fabric interface and has the following keys to identify it: system_id (id of the leaf system, usually serial number), interface (name of the interface).

Additional Processor(s)

leaf fabric interface status history (Accumulate)

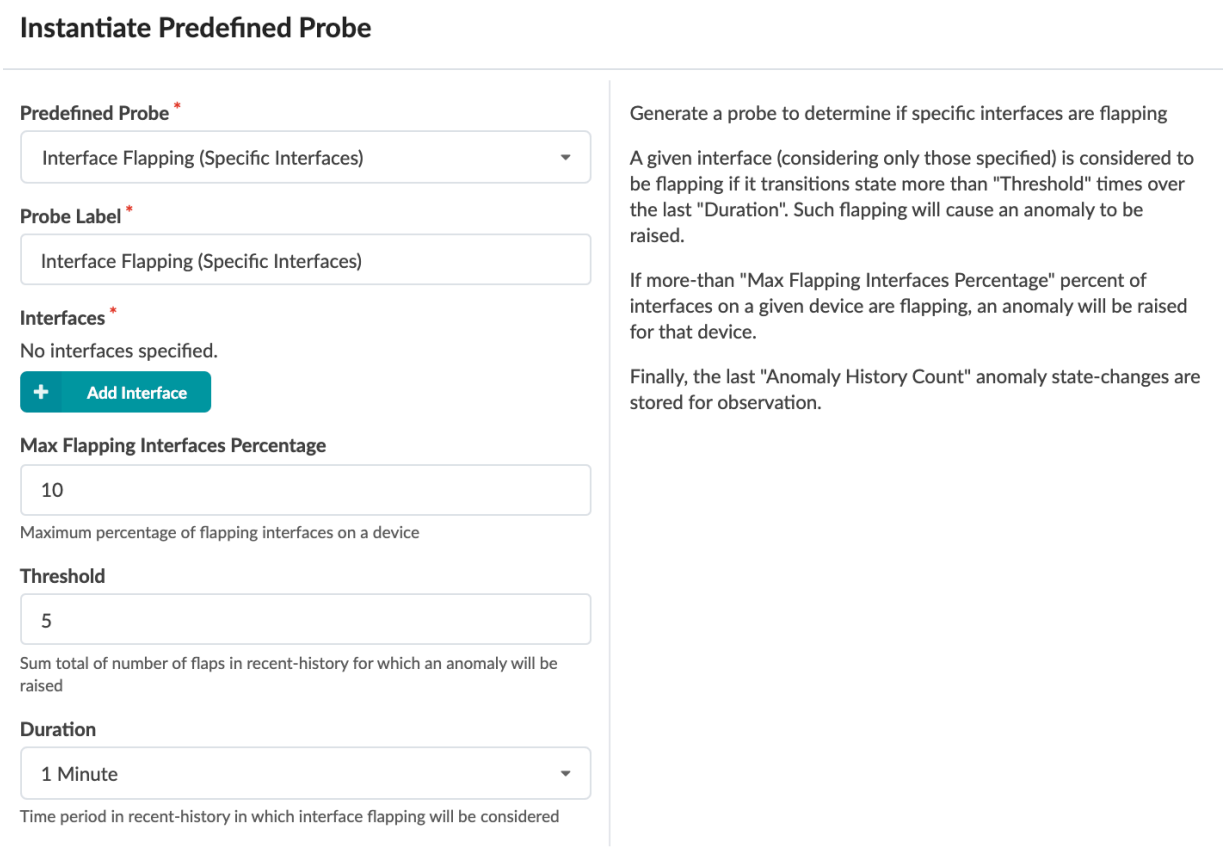
Purpose: create recent history time series for each interface status In terms of the number of samples, the time series will hold the smaller of: 1024 samples or samples collected during the last 'total_duration' seconds (facade parameter).

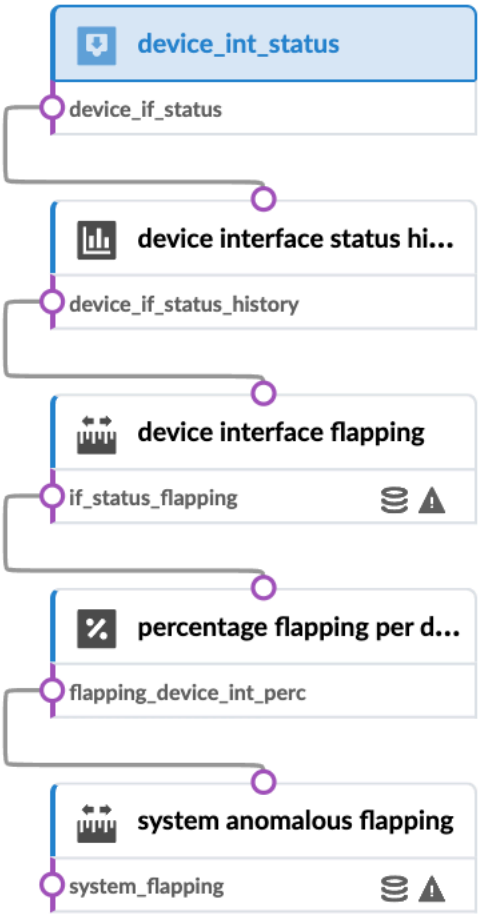
	Input Stage: leaf_if_status	
	Output Stage: leaf_fab_int_status_accumulate	Set of interface status time series (for each spine facing interface on each leaf). Each set member has the following keys to identify it: system_id (id of the leaf system, usually serial number), interface (name of the interface).
leaf fabric interface flapping (Range)	Purpose: Count the number of state changes in the leaf_fab_int_status_accumulate ("up" to "down" and "down" to "up"). If the count is higher than 'threshold' facade parameter return "true", otherwise "false".	
	Input Stage: leaf_fab_int_status_accumulate	
	Output Stage: if_status_flapping	Set of statuses (for each spine facing interface on each leaf), indicating if the interface has been flapping or not. Each set member has the following keys to identify it: system_id (id of the leaf system, usually serial number), interface (name of the interface).
percentage flapping per device interfaces (MatchPercentage)	Input Stage: if_status_flapping	
	Output Stage: flapping_fab_int_perc	
system anomalous flapping (Range)	Input Stage: flapping_fab_int_perc	
	Output Stage: system_flapping	Set of statuses for each leaf, indicting if the leaf has higher then acceptable percentage of flapping interfaces. Each set member has the following key to identify it: system_id (id of the leaf system, usually serial number).

For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: Interface Flapping (Specific Interfaces)

The interface flapping (specific interfaces) probe determines if specific interfaces are flapping. A given interface (considering only those specified) is considered to be flapping if it transitions state more than "Threshold" times over the last "Duration". Such flapping causes an anomaly to be raised. If more-than "Max Flapping Interfaces Percentage" percent of interfaces on a given device are flapping, an anomaly is raised for that device. Finally, the last "Anomaly History Count" anomaly state-changes are stored for observation.



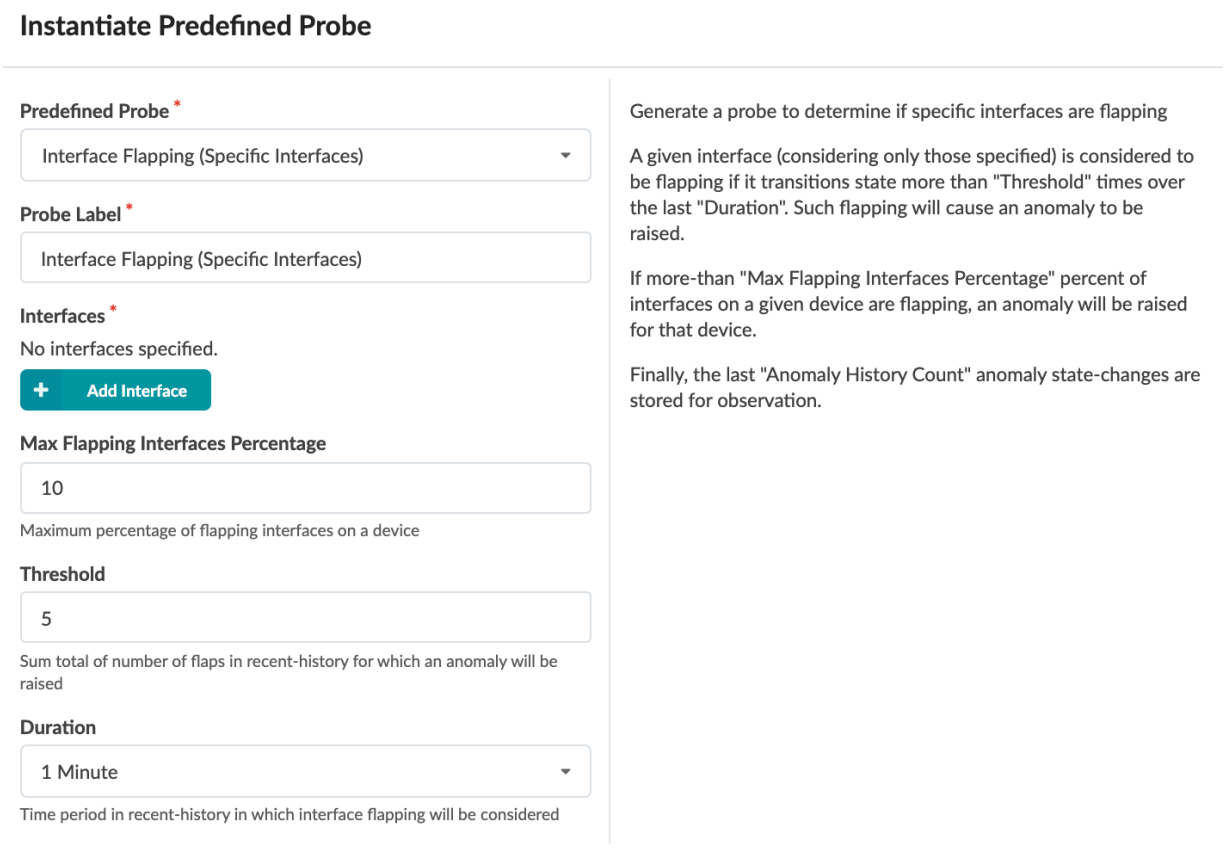


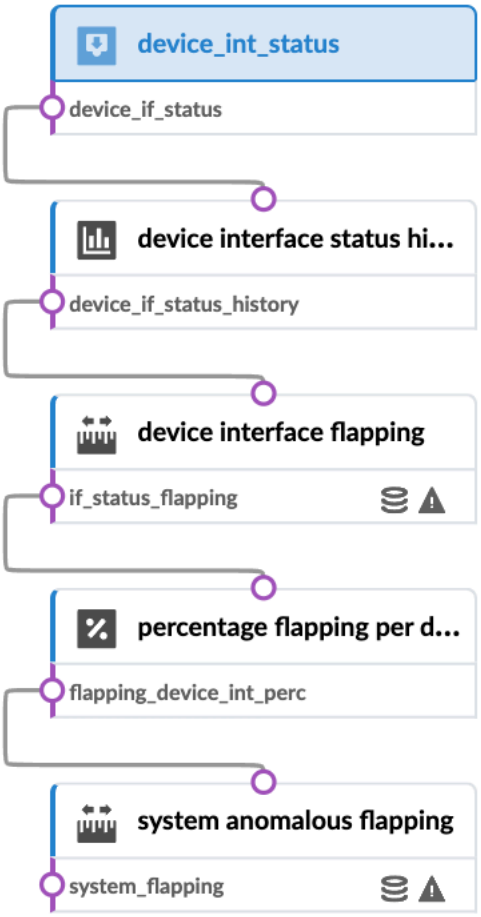
For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: Interface Flapping (Specific Interfaces)

The interface flapping (specific interfaces) probe determines if specific interfaces are flapping. A given interface (considering only those specified) is considered to be flapping if it transitions state more than "Threshold" times over the last "Duration". Such flapping causes an anomaly to be raised. If more-than "Max Flapping Interfaces Percentage" percent of interfaces on a given device are flapping, an anomaly is raised for that device. Finally, the last "Anomaly History Count" anomaly state-changes are stored for

observation.



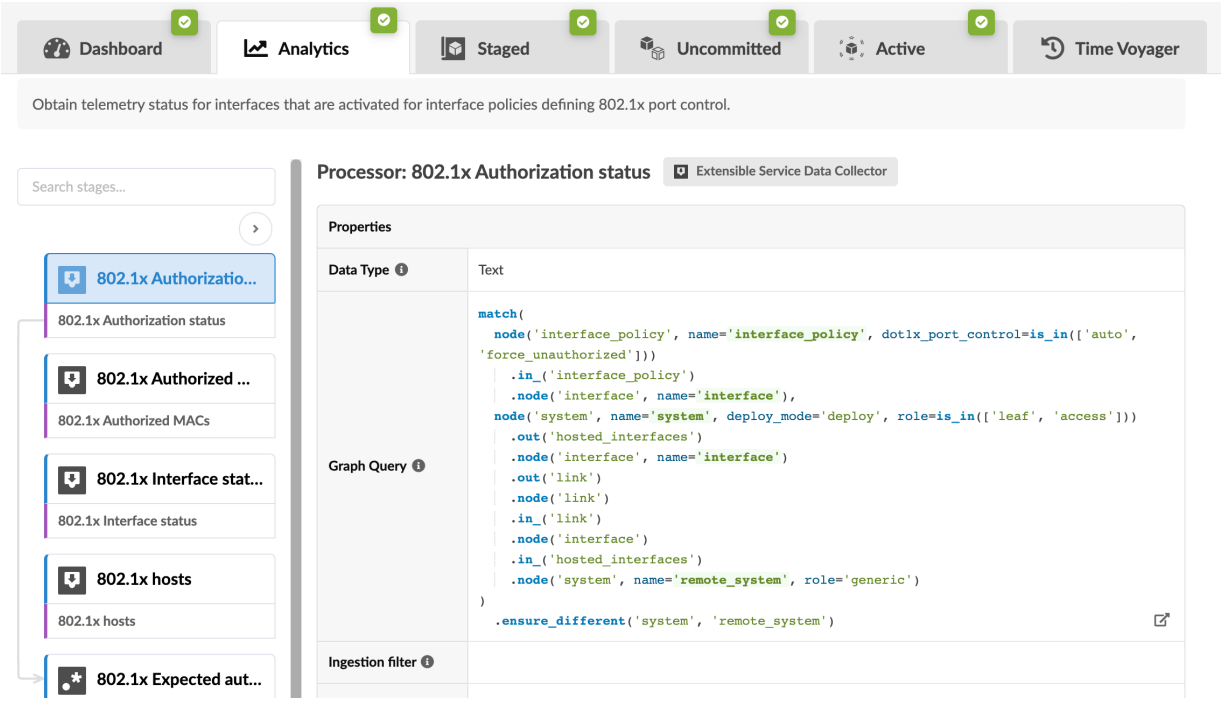


For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: Interface Policy 802.1x

The Interface Policy predefined probe is used to monitor 802.1X supplicants and interface authentication. You can instantiate this probe to maintain 802.1X networks. The 802.1X hosts probe gives a fast view of network 802.1X MAC addresses, authorization status, ports, and dynamic VLAN

information.



For more information about interface policies, see `Interface Policies <interface_policies>`.

For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: LAG Imbalance

The LAG imbalance probe calculates LAG imbalance. It calculates the standard deviation across physical links for all LAGs in the network.

Instantiate Predefined Probe

Predefined Probe *

LAG Imbalance

Probe Label *

LAG Imbalance

Max Standard Deviation

20

Maximum standard deviation used for imbalance detection (in percents of link bandwidth).

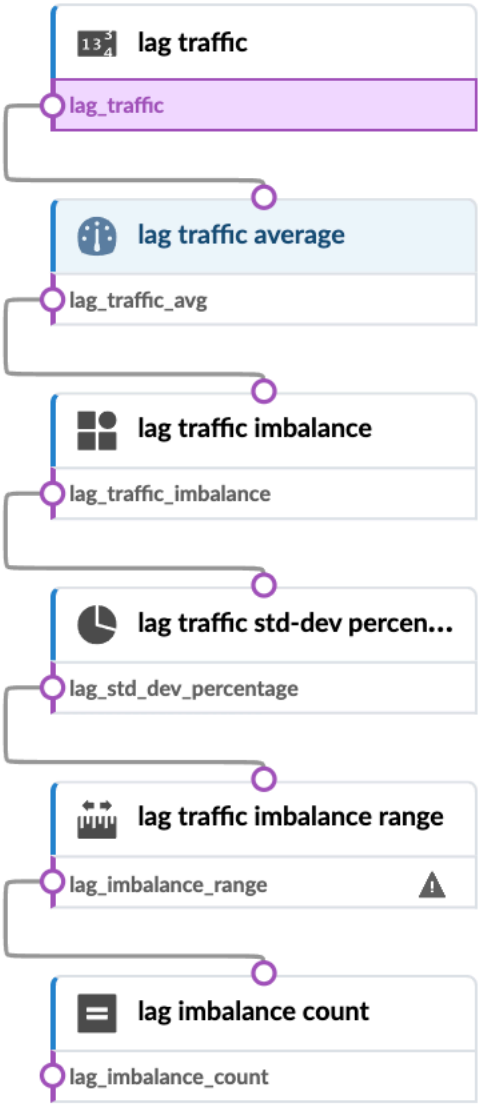
Duration

1 Minute

Time period in recent-history over which imbalance will be considered

Generate a probe to calculate LAG imbalance

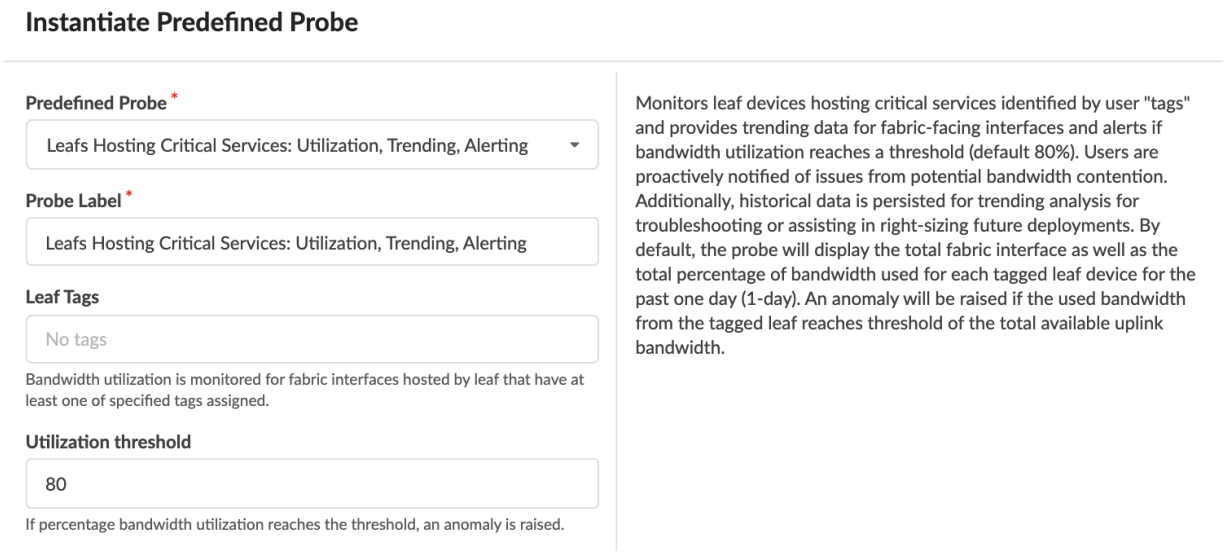
Calculates std deviation across physical links for all LAGs in the network.

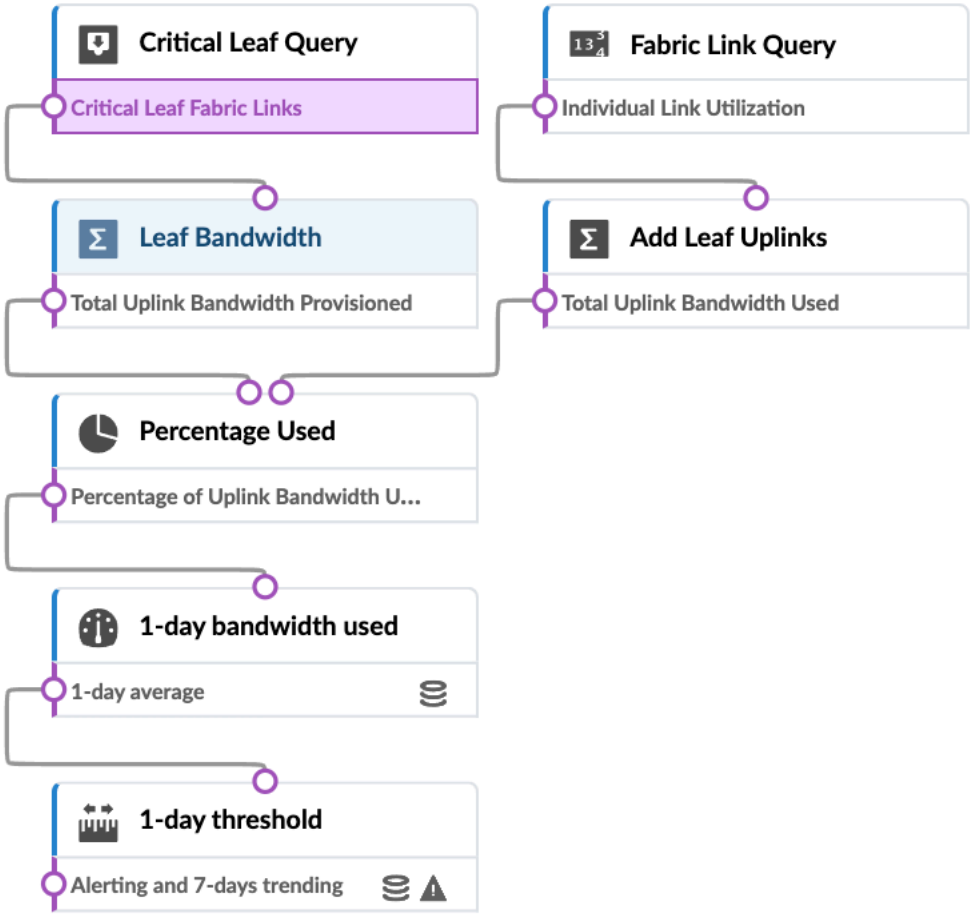


For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: Leafs Hosting Critical Services: Utilization, Trending, Alerting

Monitors leaf devices hosting critical services identified by user "tags" and provides trending data for fabric-facing interfaces and alerts if bandwidth utilization reaches a threshold (80%). Users are proactively notified of issues from potential bandwidth contention. Additionally, historical data is persisted for trending analysis for troubleshooting or assisting in right-sizing future deployments. By default, the probe will display the total fabric interface as well as the total percentage of bandwidth used for each tagged leaf device for the past one day (1-day). An anomaly will be raised if the used bandwidth from the tagged leaf reaches 80% of the total available uplink bandwidth.





For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: Link Fault Tolerance in Leaf and Access LAGs

The link fault tolerance in leaf and access LAG probe monitors LAG fault tolerance issues from a capacity viewpoint.

Instantiate Predefined Probe

Predefined Probe *

Link Fault Tolerance in Leaf and Access LAGs

Probe Label *

Link Fault Tolerance in Leaf and Access LAGs

History Duration

12 Hours

Time period of history to maintain

Duration

10 Minutes

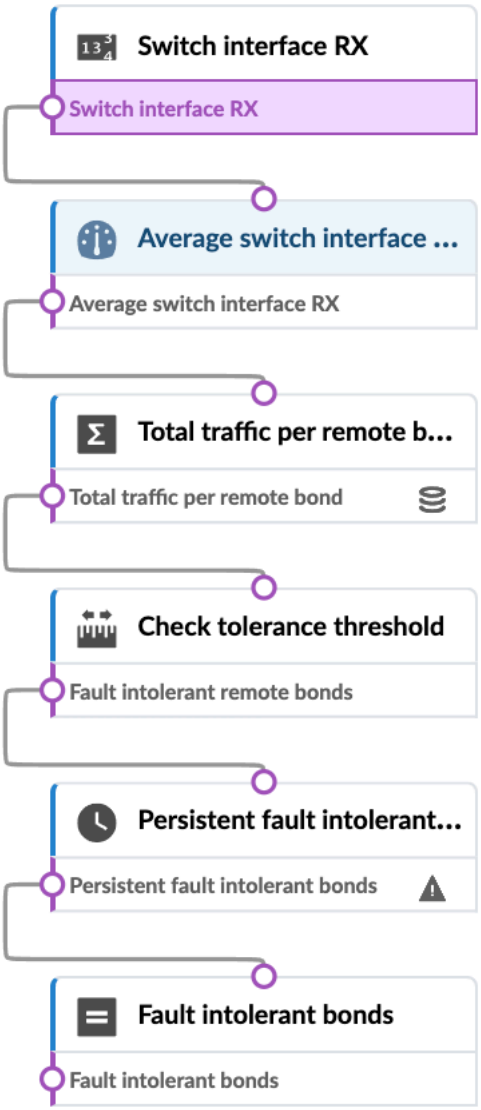
Time period in recent-history over which anomaly intolerant bonds will be considered

Threshold Duration

9 minutes

Total amount of time in recent-history during which bonds with traffic exceeding tolerance threshold is observed for anomaly to be raised

Generate a probe to monitor LAG fault tolerance issues from capacity viewpoint



For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: MAC Monitor

The MAC Monitor probe observes MAC address activity on switches and checks VTEP MAC Tables to ensure accurate EVPN type 2 MAC advertisement integration into the Forwarding Database (FDb). It identifies, validates, and flags missing MAC addresses across all VTEPs. This data is summarized by calculating MAC address discrepancies.

Instantiate Predefined Probe

Predefined Probe *

MAC Monitor

Probe Label *

MAC Monitor

Minimum number of missing MAC addresses per system

1

The minimum count of missing MAC addresses per system to qualify the system as being in bad health.

Raise Anomaly

Trigger an anomaly alert for virtual networks experiencing consistently high levels of missing MAC addresses.

The interval for monitoring high counts of missing MAC addresses.

11 minutes

This parameter should be understood in the context of the "Systems across VNIs experiencing persistent absence of MAC addresses processor" as follows: If the value of "Systems across VNIs missing MAC addresses" remains "true" for more than a specified threshold percentage of time within the last interval of minutes (using a sliding window approach), then set the value of "Systems across VNIs experiencing persistent absence of MAC addresses" to "true".

Threshold for High Missing MAC Address Count (as a percentage)

100

This parameter should be understood in the context of the "Systems across VNIs experiencing persistent absence of MAC addresses processor" as follows: If the value of "Systems across VNIs missing MAC addresses" remains "true" for more than a specified threshold percentage of time within the last interval of minutes (using a sliding window approach), then set the value of "Systems across VNIs experiencing persistent absence of MAC addresses" to "true".

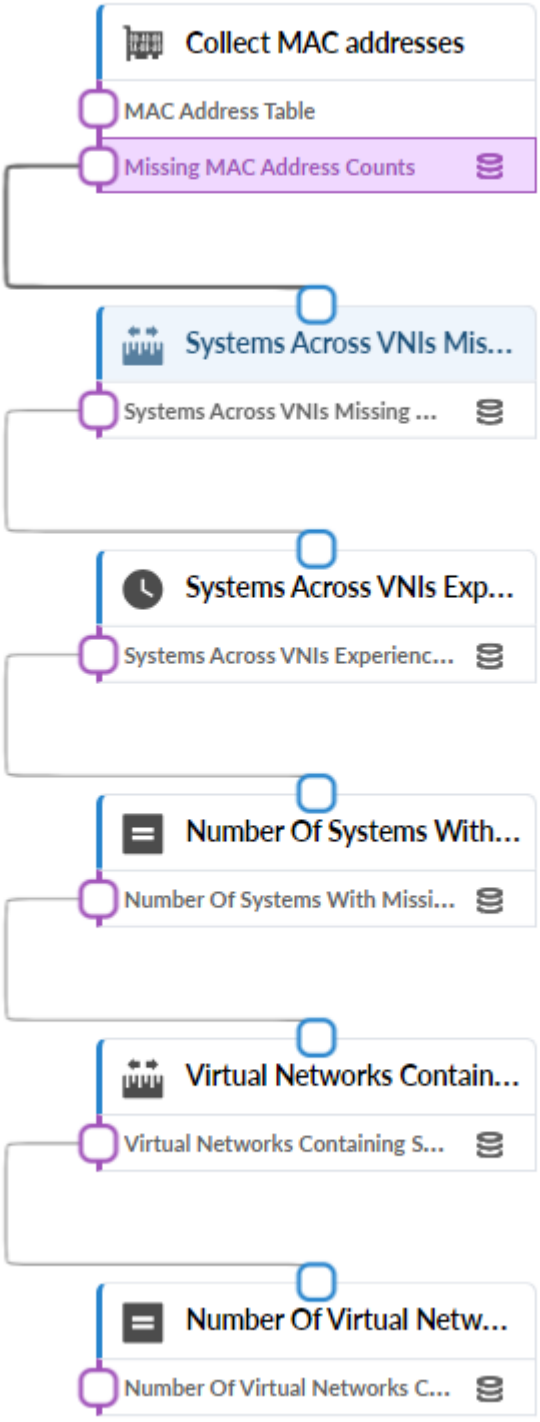
This probe observes MAC address activity in leaf and access switches (overlay traffic), conducting routine checks on VTEP Mac Tables to ensure accurate integration of EVPN type 2 MAC advertisements into the Forwarding Database (FDb).

It gathers data from the internal MAC telemetry service and identifies the complete set of MAC addresses associated with a specific VNI. The tool then validates the presence of these MAC addresses across all VTEPs. Any discrepancy in MAC address presence is recorded as 'Missing MAC Address Counts' per VTEP per VNI. Instances where the 'Missing MAC Address Counts' exceed a predefined threshold are flagged. If this elevated count persists for a specified 'interval for monitoring high counts of missing MAC' duration, the systems are categorized as 'Systems across VNIs experiencing persistent absence of MAC addresses'.

The data is summarized by calculating the total number of systems lacking MAC addresses per virtual network and the total count of virtual networks that include systems with missing MAC addresses within the data center (DC).

Metrics are retained for 30 days across all stages, except for the "MAC Address Table", which does not support logging of metrics.

Create Another? Create



For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: MLAG Imbalance

The MLAG Imbalance probe calculates MLAG imbalance. It calculates standard deviation across links for all MLAGs in the network. If any are over the specified threshold in the last specified time period, an anomaly is raised. It calculates the percentage of MLAGs in each rack in this state. It calculates standard deviation across port-channels for all port-channels in all MLAGs in the network. If any are over the specified threshold in the last specified time period, an anomaly is raised. It also calculates the percentage of MLAGs in each rack in this state. Finally, it calculates standard deviation of port-channels across their containing MLAGs. If the standard deviation for any of these MLAGs is over the specified threshold, an anomaly is raised. Finally, we calculate the percentage of port-channels in each rack in this state.

Source Processor	**mlag interface traffic (Interface Counters)**	Purpose: wires in interface traffic samples (measured in bytes per second) all leaf interfaces that are part of an MLAG. Unit is bytes per second. **Output stage: mlag_int_traffic** — Set of traffic samples (for each mlag interface on each leaf). Each set member has the following keys to identify it: mlag_id, server (label of the server node), leaf (label of the leaf node), rack (label of the rack), system_id (leaf serial number), interface (name of the interface).
Additional Processor(s)	**mlag interface traffic average (Periodic Average)**	Purpose: Calculate average traffic during period specified by average_period facade parameter. Unit is bytes per second. Input Stage: mlag_int_traffic **Output Stage: mlag_int_traffic_avg** — Set of traffic average values (for each spine-facing interface on each leaf). Each set member has the following keys to identify it: mlag_id, server (label of the server node), leaf (label of the leaf node), rack (label of the rack), system_id (leaf serial number), interface (name of the interface). Unit is bytes per second.
	mlag interface traffic imbalance (Standard Deviation)	Purpose: Calculate standard deviation between traffic averages on all interfaces belonging to a given MLAG. Unit is bytes per second. Input Stage: mlag_int_traffic_avg **Output Stage: mlag_int_traffic_imbalance** — Set of numbers, one for each mlag_id, each indicating standard deviation of the

		average traffic on each interface that is part of this MLAG. Each set member has the following keys to identify it: rack, mlag_id. Unit is bytes per second.
port-channel interface std-dev (Standard Deviation)	Purpose: Calculate standard deviation between traffic averages on all interfaces belonging to a port channel. Unit is bytes per second. Input Stage: mlag_int_traffic_avg	
	Output Stage: port_channel_int_std_dev	Set of numbers, one for each port channel identified by mlag_id, leaf pair. Each number each indicates standard deviation of the average traffic on each interface that is part of this port channel. Each set member has the following keys to identify it: rack, mlag_id, leaf. Unit is bytes per second.
port-channel total traffic (Sum)	Purpose: Calculate total traffic per port channel. Unit is byte per second. Input Stage: mlag_int_traffic_avg	
	Output Stage: mlag_port_channel_total	Set of numbers, each indicating total traffic for each port channel. Each set member has the following key to identify it: rack, mlag_id, leaf. Unit is byte per second.
mlag port-channel traffic std-dev (Standard Deviation)	Purpose: Calculate standard deviation between traffic averages on both port channels belonging to an MLAG. Unit is bytes per second. Input Stage: mlag_port_channel_total	
	Output Stage: mlag_port_channel_imbalance	Set of numbers, one for each MLAG identified by mlag_id, rack pair. Each number indicates standard deviation of the average traffic on each port channel that is part of this MLAG. Each set member has the following keys to identify it: rack, mlag_id. Unit is bytes per second.

std-dev percentage mlag (Ratio)

Input Stage: mlag_int_traffic_imbalance

Output Stage: std_dev_percentage_mlag

std-dev percentage port-channel (Ratio)

Input Stage: port_channel_int_std_dev

Output Stage: std_dev_percentage_pc

live mlag imbalance (Range)

Purpose: Evaluate if the MLAG imbalance as measured by standard deviation for the average traffic on each member interface is within acceptable range. In this case acceptable range is between 0 and std_max facade parameter (in bytes per second unit).

Input Stage: std_dev_percentage_mlag

Output Stage: live_mlag_imbalance	Set of true/false values, each indicating if MLAG imbalance for the average traffic on each member interface is within acceptable range for each mlag. Each set member has the following keys to identify it: rack, mlag_id.

live port-channel imbalance (Range)

Purpose: Evaluate if the port channel imbalance as measured by standard deviation for the average traffic on each member interface is within acceptable range. In this case acceptable range is between 0 and std_max facade parameter (in bytes per second unit).

Input Stage: std_dev_percentage_pc

Output Stage: live_port_channel_imbalance	Set of true/false values, each indicating if port channel imbalance for the average traffic on each member interface is within acceptable range for each mlag. Each set member has the following keys to identify it: rack, mlag_id, leaf.

std-dev percentage mlag port-channel (Ratio)

Input Stage: mlag_port_channel_imbalance

Output Stage: std_dev_percentage_mlag_pc

live mlag port-channel imbalance (Range)

Purpose: Evaluate if the mlag imbalance as measured by standard deviation for the average traffic on each member port channel is within acceptable range. In this case acceptable range is between 0 and std_max facade parameter (in bytes per second unit).

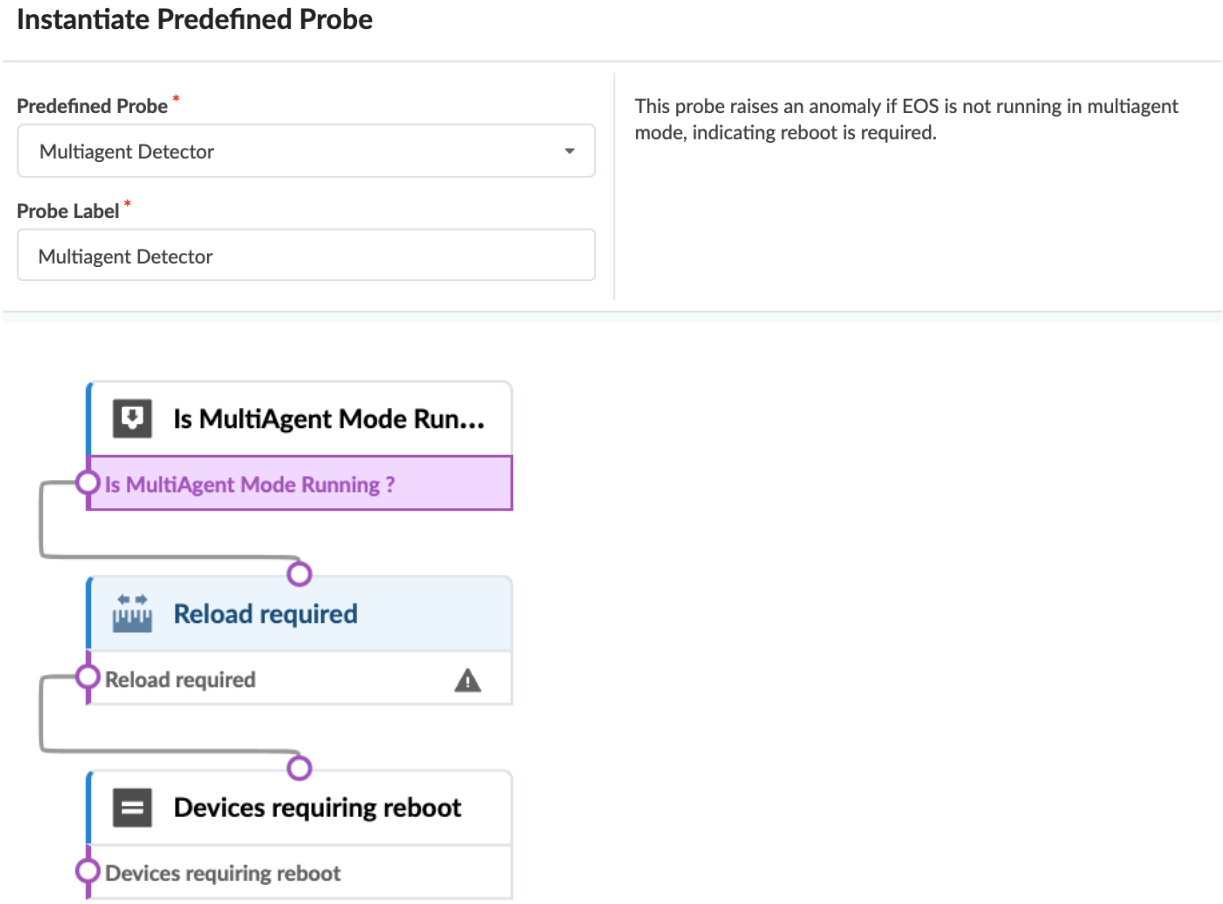
	Input Stage: std_dev_percentage_mlag_pc	
	Output Stage: mlag_port_channel_imbalance_out_of_range	Set of true/false values, each indicating if MLAG imbalance between the average traffic on each member port channel is within acceptable range for each mlag. Each set member has the following keys to identify it: rack, mlag_id.
mlag imbalance per link count (Match Count)	Input Stage: live_mlag_imbalance Output Stage: mlag_imbalance_link_count	
port-channel imbalance per rack (Match Percentage)	Purpose: Calculate percentage of port channels on a given rack that have imbalance anomaly. Input Stage: live_port_channel_imbalance	
	Output Stage: port_channel_imbalance_per_rack	Set of numbers, each indicating the percentage of port channels with imbalance on each rack. Each set member has the following key to identify it: rack, mlag_id, leaf.
mlag port-channel imbalance per rack (Match Percentage)	Purpose: Calculate percentage of MLAGs on a given rack that have port channel imbalance anomaly. Input Stage: mlag_port_channel_imbalance_out_of_range	
	Output Stage: mlag_port_channel_imbalance_anomaly_per_rack	Set of numbers, each indicating the percentage of port channels with imbalance on each rack. Each set member has the following key to

identify it: rack, mlag_id.

For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: Multiagent Detector

The multiagent detector probe raises an anomaly if EOS is not running in multiagent mode, indicating that a reboot is required.



For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

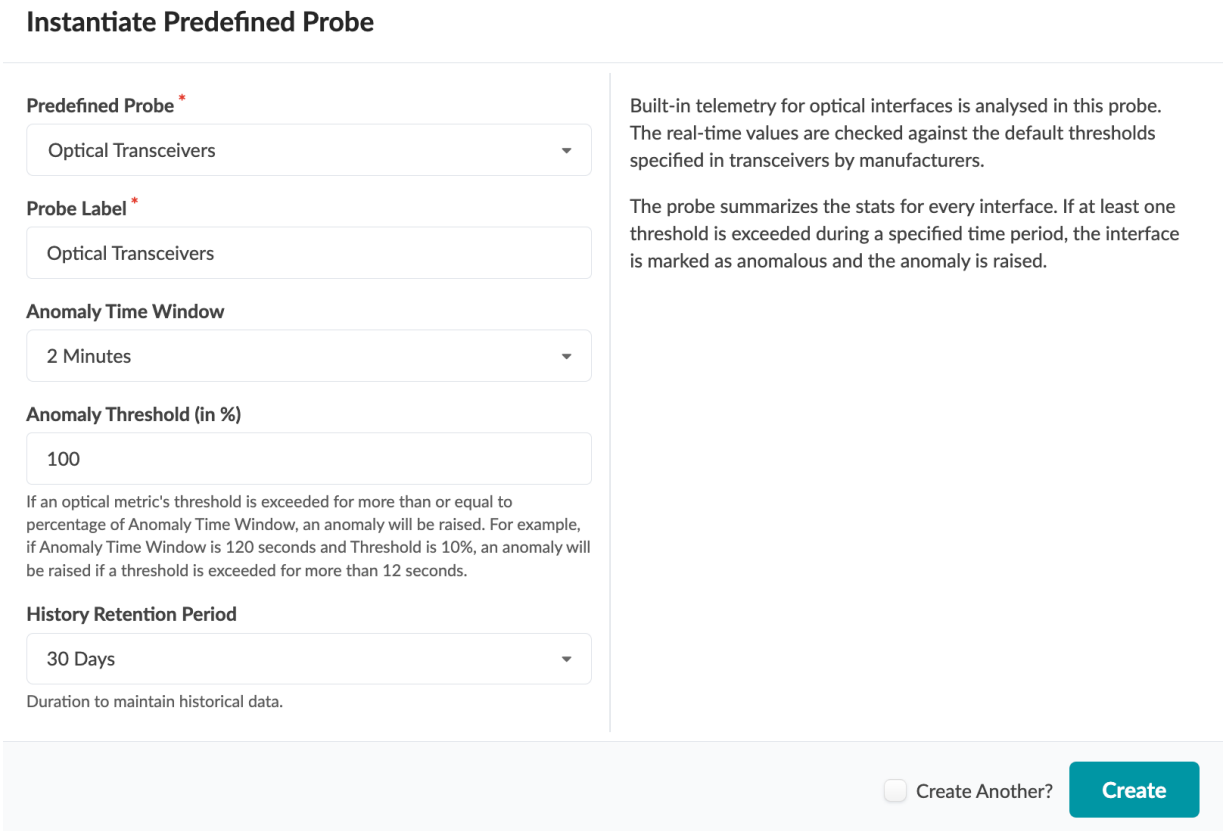
Probe: Optical Transceivers

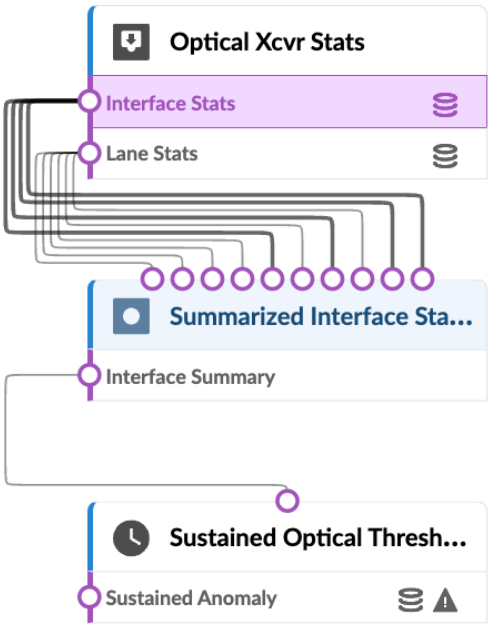
The Optical Transceivers probe monitors optical statistics based on the following telemetry data:

- Temperature C) of the physical port (interface stats)
- Voltage (V) of the physical port (interface stats)
- Transmit Power Level (dBm) of each optical lane (lane stats)
- Receive Power Level (dBm) of each optical lane (lane stats)
- Transmit Bias (mA) of each optical lane (lane stats)

If telemetry data falls outside the specified range for the specified amount of time, a warning or alarm is raised, as applicable.

Warnings and alarms specify whether the value causing the anomaly was too high or too low.





For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: Packet Discard Percentage

The packet discard percentage probe raises visibility into issues related to physical interfaces.

Instantiate Predefined Probe

Predefined Probe *

Packet Discard Percentage

Probe Label *

Packet Discard Percentage

Ingress History Duration

12 Hours

Time period in recent-history for discard ingress packets and total rx packets

Discard Percent Threshold

1

Discard percentage threshold. Consider the discard percentage is too high if it is greater than this threshold

Duration

1 minute 30 seconds

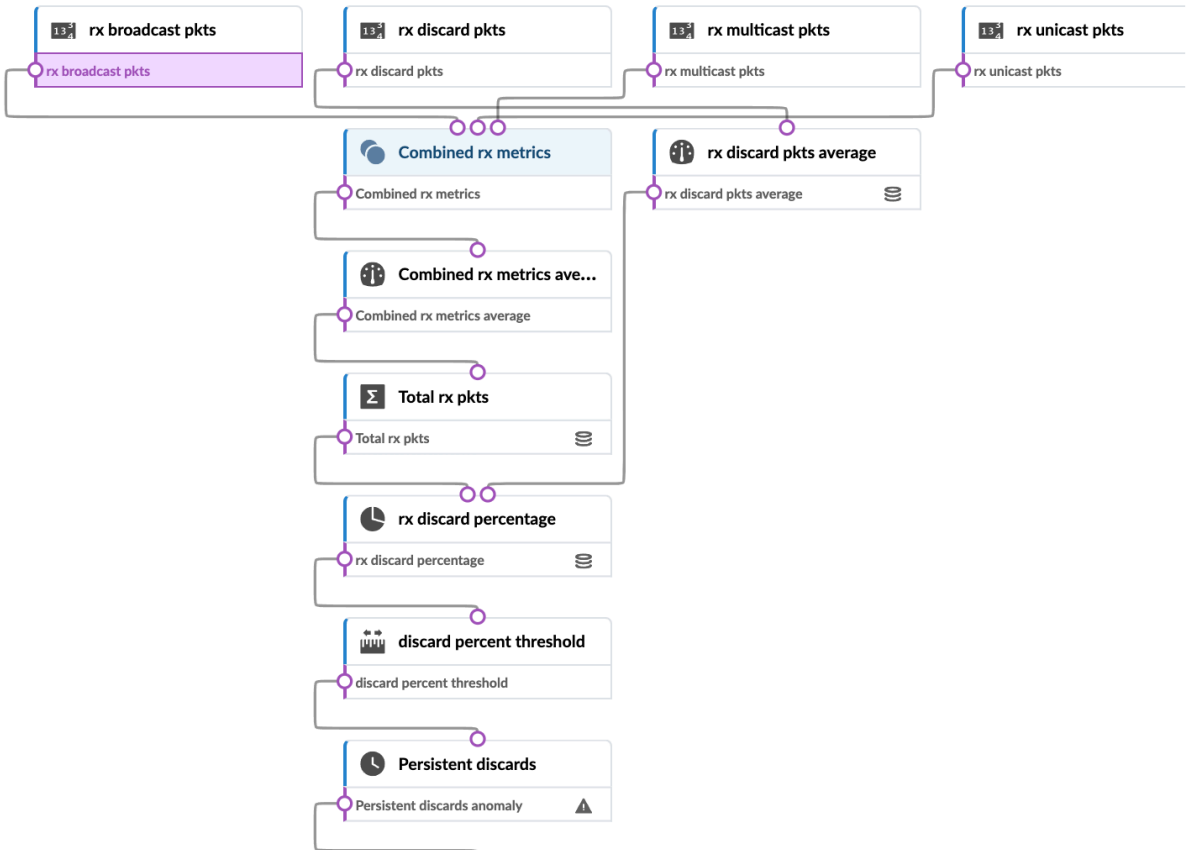
Time period in recent-history over which discard percentage data will be considered

Threshold Duration

20 seconds

Total amount of time in recent-history during which the discard percentage is higher than threshold for anomaly to be raised

Generate a probe to raise visibility into issues related to physical interfaces



For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: Spine Fault Tolerance

The spine fault tolerance probe monitors spine fault tolerance issues from a capacity viewpoint.

Instantiate Predefined Probe

Predefined Probe *
Spine Fault Tolerance

Probe Label *
Spine Fault Tolerance

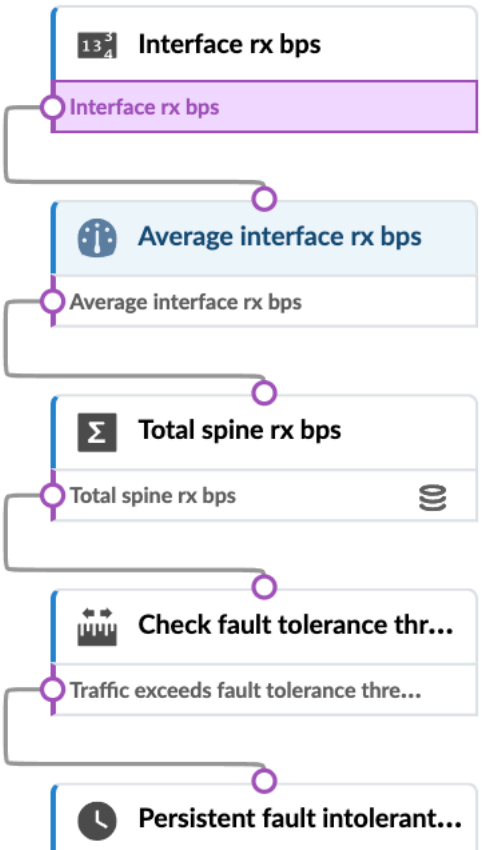
History Duration
12 Hours
Time period of history to maintain

Number of Faulty Spines
1
Number of faulty spine used to monitor link fault tolerance

Duration
10 Minutes
Time period in recent-history in which anomaly intolerant traffic will be considered

Threshold Duration
9 minutes
Total amount of time in recent-history during which total rx traffic of spines exceeding tolerance threshold is observed for anomaly to be raised

Generate a probe to monitor spine fault tolerance issues from capacity viewpoint



For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe: Total East/West Traffic

Purpose	The Total East/West Traffic probe calculates total east/west traffic. This probe takes the sum of all traffic to leaf devices from their directly-attached servers and subtracts from that the sum of all traffic to external routers (all traffic values in this calculation are averaged periodically over "Average Period"). The result of this is the total east/west traffic. Time series of length "History Sample Count" is maintained for the sum of server traffic, the sum of external traffic, and the total east/west traffic. When instantiating this probe, external router tag(s) must be specified (new in version 4.0).		
Source Processors	**external router south-north link traffic (Interface Counters)**	Purpose: wires in interface traffic samples (measured in bytes per second) for traffic sent to external routers Output Stage: ext_router_interface_traffic	
	leaf server traffic counters (Interface Counters)	Purpose: wires in interface traffic samples (measured in bytes per second) for traffic received on leaf devices from the servers	
		Output Stage: server_traffic_counters	Set of traffic samples (for each server-facing interface on each leaf) in the receive direction. Each set member has the following keys to identify it: system_id (id of the leaf system, usually serial number), interface (name of the interface).
Additional Processor(s)	**external router south-north links traffic average (Periodic Average)**	Purpose: Calculate average traffic for each interface-facing external router traffic during period specified by average_period facade parameter. Unit is bytes per second. Input Stage: ext_router_interface_traffic	
		Output Stage: ext_router_interface_traffic_avg	Set of traffic average values (for each external router-facing interface on each device). Each set member has the following keys to identify

it: system_id (id of the leaf system, usually serial number), interface (name of the interface).

server traffic average (Periodic Average)

Purpose: Calculate average server traffic during period specified by average_period facade parameter. Unit is bytes per second.

Input Stage: server_traffic_counters

Output Stage: server_traffic_avg — Set of traffic average values (for each server-facing interface on each leaf) in the receive direction. Each set member has the following keys to identify it: system_id (id of the leaf system, usually serial number), interface (name of the interface).

south-north traffic (Sum)

Purpose: Calculate total traffic by summing average traffic on each interface-facing external router. Unit is bytes per second.

Input Stage: ext_router_interface_traffic_avg

Output Stage: total_outgoing_traffic — Total south-north traffic average in bytes per second.

total server traffic (Sum)

Purpose: Calculate total server traffic by summing average traffic on each interface attached to servers in receive direction. Unit is bytes per second.

Input Stage: server_traffic_avg

Output Stage: total_server_traffic — Total server traffic average in bytes per second.

outgoing_traffic_average (Periodic Average)

Purpose: Calculate total south-north traffic over average_period seconds, which is a facade parameter. Unit is bytes per second.

Input Stage: total_outgoing_traffic

	Output Stage: total_outgoing_traffic_average	Total south-north traffic average in bytes per second.

server generated traffic average (Periodic Average)	Purpose: Purpose: Calculate total average server traffic over average_period seconds, which is a facade parameter. Unit is bytes per second. Input Stage: total_server_traffic	
	Output Stage: total_server_traffic_history	Time series showing total average server traffic over recent history. Unit is bytes per second.

east-west traffic (Subtract)	Purpose: create recent history time series showing how total average east-west traffic changed over time. In terms of the number of samples, the time series holds history_sample_count values (facade parameter). Unit is bytes per second. Input Stages: total_outgoing_traffic_average and total_server_traffic_history	
	Output Stage: eastwest_traffic_history	Time series showing how total average east-west traffic changed over recent history. Unit is bytes per second

Probe: VMs without Fabric Configured VLANs Probe (Virtual Infra)

Purpose	Calculate VMs missing a VLAN and calculate VMs not backed by VLANs on managed leaf devices connected to hypervisors.	
Source Processors	**VMs backed by Fabric VLANs (generic graph collector)**	output stage: VMs backed by Fabric VLANs (number set) (generated from graph)
	VMs on hypervisors connected to Fabric (generic)	output stage: VMs on hypervisors connected to Fabric (number set)

Additional Processor(s)

Differences between Fabric and Hypervisor (`set comparison<processor_set_comparison>`)

input stage(s): VMs backed by Fabric VLANs (number set)

VMs on hypervisors connected to Fabric (number set)

output stage: VMs not backed by Fabric VLANs (number set)

Affected VM Anomalies (`range <processor_range>`)

input stage: VMs not backed by Fabric VLANs

output stage: Affected VM Anomalies (discrete state set)

Example Usage

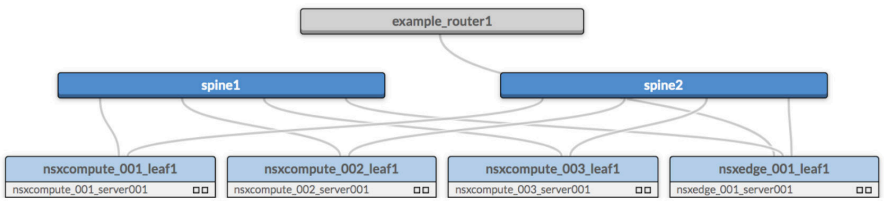
NSX-T Integration - VMs participating in a particular network are attached to an NSX logical switch. In NSX transport zone controls to which hypervisors or ESXi host an NSX logical switch can span. To have VXLAN connectivity for these VMs they need to be part of the same transport zone. This predefined anomaly helps validate that all VLAN backend interfaces defined for NSX-T nodes are also configured on the ToR interfaces connecting that node to the fabric.

VLAN probe anomaly checks for VLAN specification in case of NSX-T via one of the two methods below:

Method One: When you have VMs that are connected to the NSX-T overlay, you can configure a bridge-backed logical switch to provide layer 2 connectivity with other devices or VMs. So via VLAN specification on NSX-T layer 2 bridges and fabric if respective VXLAN VN is not there, then an anomaly is raised.

Method Two: Edge uplinks go out through VLAN logical switches. So let's say if the uplink VLAN logical switch has a particular VLAN ID and respective VLAN on ToR port connected to the hypervisor host is not configured then also this VLAN probe will raise anomalies and help detect such misconfiguration.

The following is a simple topology where nsxcompute_001_server_001 and nsxedge_001_server001 are ESXi hosting VMs that are connected to the NSX-T overlay network.



There is one VM on each ESXi host that needs a VXLAN VN endpoint on each leaf, i.e. nsxcompute_001_leaf1 and nsxedge_001_leaf1 to communicate on the overlay network.

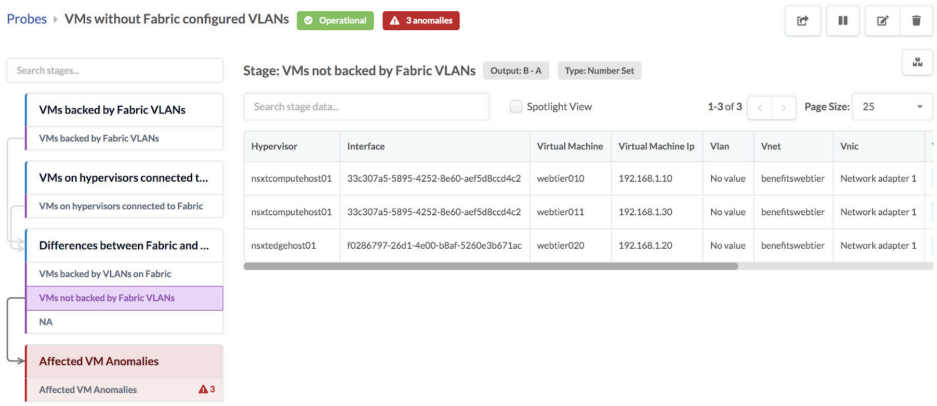
When VXLAN VNs assigned to ToR leaf devices are deleted, VLAN misconfig anomalies are raised as below under Fabric Health in the dashboard.

Critical services affected by VLAN misconfig

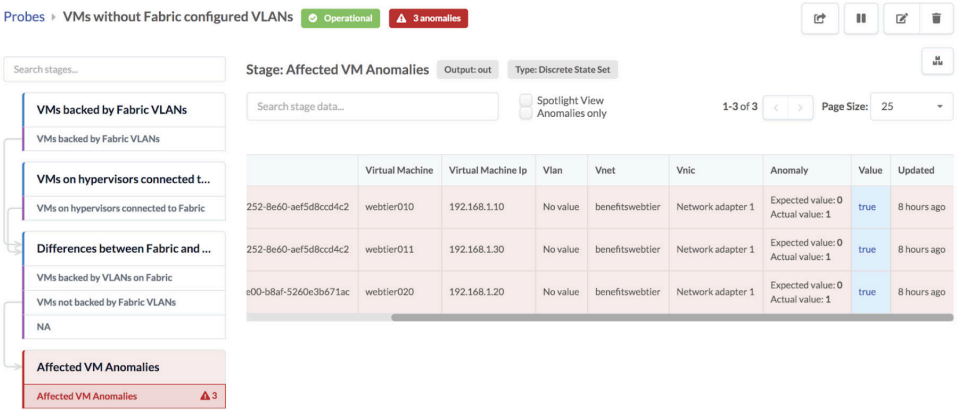
Hypervisor	Virtual Machine	Virtual Machine Ip
nsxtcomputehost01	webtier010	192.168.1.10
nsxtcomputehost01	webtier011	192.168.1.30
nsxtedgehost01	webtier020	192.168.1.20

View stage

VMs not backed by Fabric VLANs shows VMs with VLAN missing.



Affected VM Anomalies shows VLAN missing in the fabric.



Probe: VXLAN Flood List Validation

The VXLAN flood list validation probe validates the VXLAN flood list entries on every leaf in the network. It collects appropriate telemetry data, compares it to the set of flood list forwarding entries expected to be present and alerts if expected entries are missing on any device.

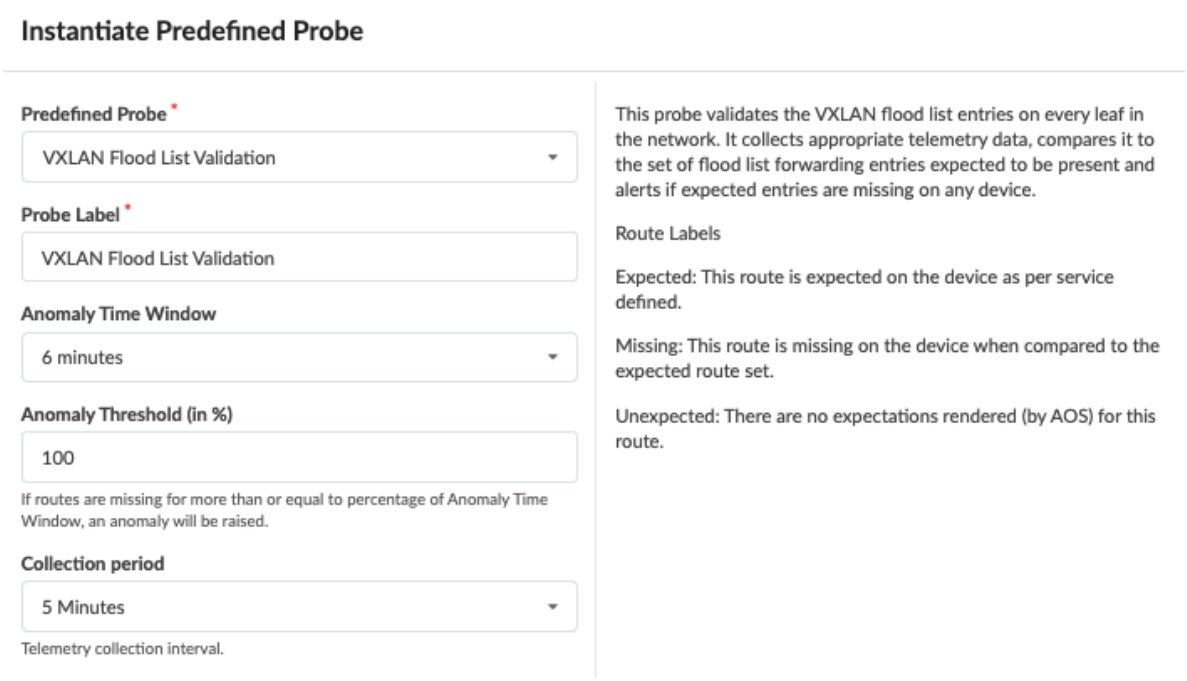
You can configure the following parameters:

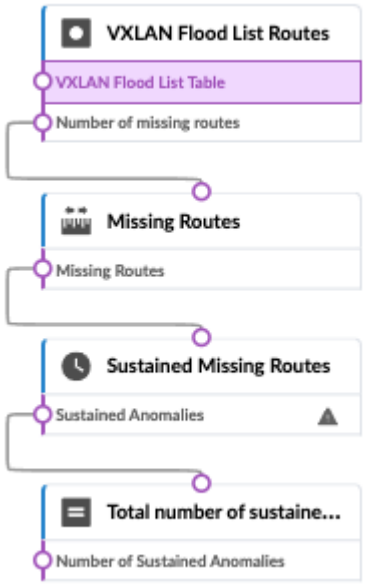
- **Probe Label**: Name to identify the probe.
- **Anomaly Time Window** : Average period duration for interface counters.
- **Anomaly Threshold (in %)**: If routes are missing for more than or equal to percentage of Anomaly Time Window, an anomaly is raised. If Anomaly Time Window ATW, and Anomaly Threshold is AT. It calculates Z = (ATW * AT)/100 in seconds. E.g. If ATW = 20 seconds, AT = 5%, then Z = (20 * 5)/100 = 1 second. When the route is in Missing state for Z seconds from total ATW duration, anomaly is raised.
- **Collection period**: All these probes are polling-based so they have a polling period.

The route labels include the following:

- **Expected**: This route is expected on the device as per service defined.
- **Missing**: This route is missing on the device when compared to the expected route set.

- **Unexpected**: There are no expectations rendered (by AOS) for this route.





NOTE: Auto-enabling the **EVPN VXLAN Route Summary** analytics dashboard enables the **EVPN VXLAN Type-3 Route Validation** and **EVPN Flood List Validation** probes automatically (but not the EVPN VXLAN Type-5 Route Validation probe). See `Configuring Auto-Enabled Dashboards<configure_dashboard>` for information about enabling the dashboard.

For more information about this probe, from the blueprint, navigate to **Analytics > Probes**, click **Create Probe**, then select **Instantiate Predefined Probe** from the drop-down list. Select the probe from the **Predefined Probe** drop-down list to see details specific to the probe.

Probe Processors

IN THIS SECTION

Processor: Accumulate

IN THIS SECTION

The Accumulate processor used in IBA probes creates one number or discrete state time-series on output for each input with the same properties; each time the input changes, it takes its timestamp and value and appends them to the corresponding output series. If total duration (total_duration) is set and the length of the output time series in time is greater than duration, it removes old samples from the time series until this is no longer the case. If max samples (max_samples) is set and the length of the output time series in terms of number of samples is greater than max_samples, it removes old samples from the time series until this is no longer the case.

Parameter	Description
Input Types	Table (number or discrete state)

(Continued)

Parameter	Description
Output Types	Table (number or discrete state, accumulate=True)
Max Samples (max_samples)	Limits the maximum number of samples or an expression that evaluates to number of samples (default:1024)
Total Duration (total_duration)	Limits the number of samples by their total duration. (in seconds) or an expression that evaluates to number of seconds (default:0)

(Continued)

Parameter	Description

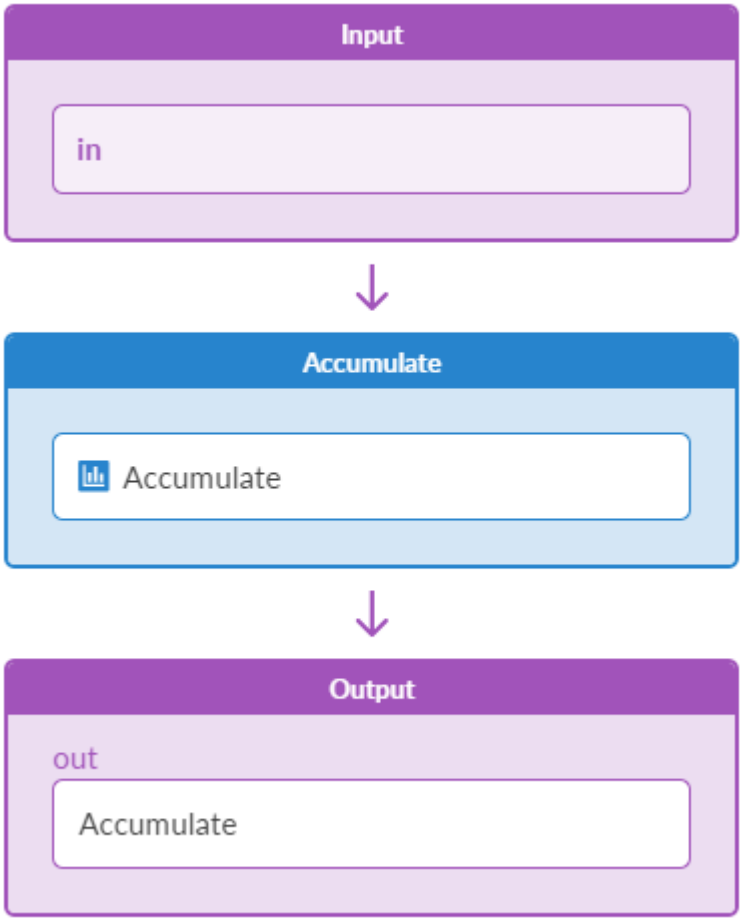
Graph Query (graph_query)	One or more queries on graph specified as strings, or a list of such queries. (String will be deprecated in a future release.) Multiple queries should provide all the named nodes referenced by the expression fields (including additional_properties). Graph query is executed on the "operation" graph. 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"]".
	
	In collector processors (`*_collector`, `if_counter`) it is used to choose a set of nodes for further processing (for example, all leaf devices, or all interfaces between leaf and spine devices)
	
	In other processors it is used for general parameterization and it is only supported as a list of queries.
	
	`graph_query: "node("system", role="leaf", name="system").`
	`out("hosted_interfaces").`
	`node("interface", name="iface").out("link").`
	`node("link", role="spine_leaf")"`
	
	`graph_query: ["node("system", role="leaf", name="system")",`
	`"node("system", role="spine", name="system")"]`
	
	Non-collector processors containing the `graph_query` configuration parameter, can be parameterized to use data from arbitrary nodes in the graph, such as property set nodes. Property sets allow you to parameterize macro level SLAs for individual business units. In the example below, `graph_query` matches a node of type `property_set` with label `probe_propset`. It's

(Continued)

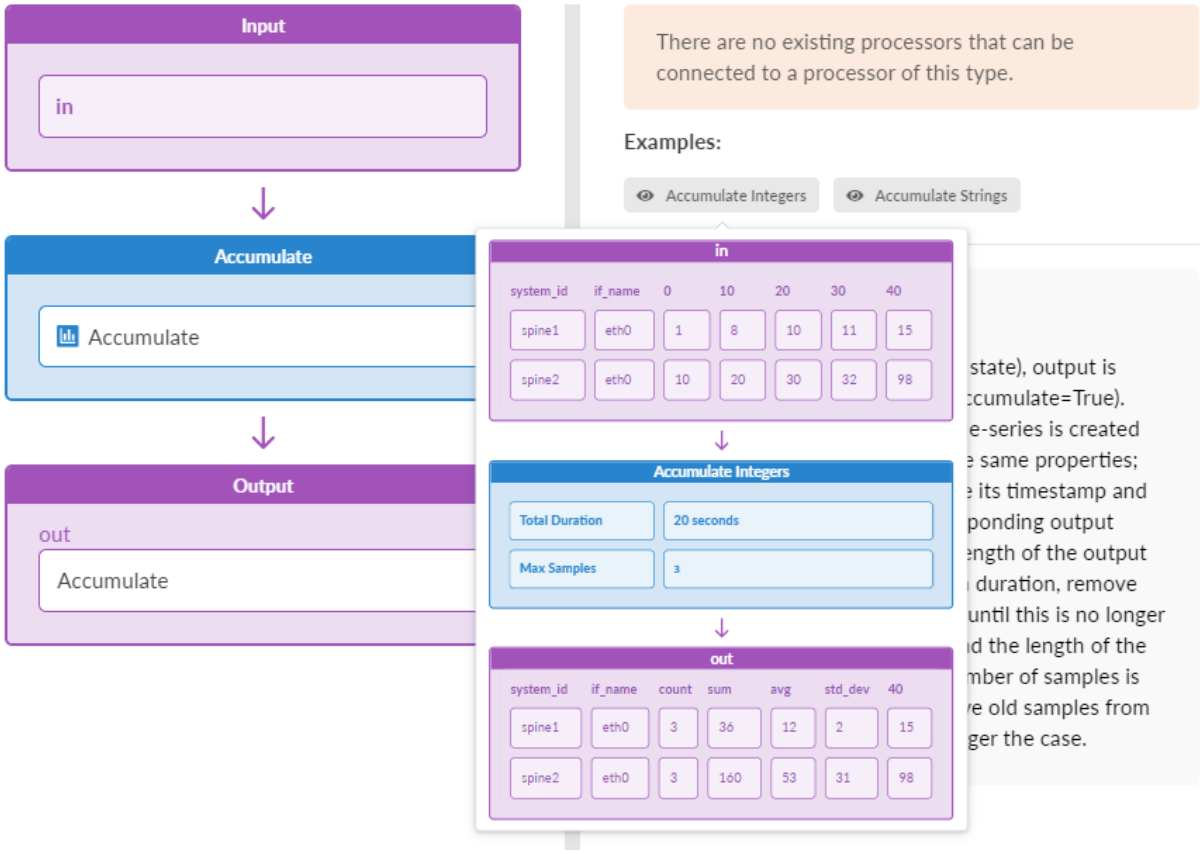
Parameter	Description

	accessed using the special `query_result` variable, where Index `0` means it's the first node in query results. If a query returned `N` nodes, they could be accessed using indices starting from `0` to `N-1`. `ps` is what the actual node is referred to in the query; the rest depends on the structure of the node. The `int()` casting is required because values of `property_set` nodes are strings. Here it's assumed that a property set node has the label `probe_propset` and that the value `accumulate_duration` was already created.
	
	`graph_query: [node("property_set", label="probe_propset", name="ps")]`
	`duration: int(query_result[0] ["ps"].values["accumulate_duration"])`
	
	Another example is a that probes can validate a compliance requirement; the compliance value may change over time and/or it can be used by more than one probe. Also, a probe can validate NOS versions on devices. In this case, property sets can be used to define the current NOS version requirement. If it changes tomorrow: change the property set value, instead of going under the probe stage.

Enable Streaming (enable_streaming)	Makes samples of output stages streamed if enabled. An optional boolean that defaults to False. If set to True, all output stages of this processor are streamed in the generic protobuf schema.



In the Add Processor window of the Apstra UI, hover over the "Accumulate Integers" or "Accumulate Strings" tooltips for visual examples of how the Accumulate Processor functions.



Example: Accumulate

Assume a configuration of

```
max_samples: 3
total_duration: 0
```

Assume the following input at time t=1

```
[if_name=eth0] : "up"
[if_name=eth1] : "down"
[if_name=eth3] : "up"
```

We have the following output at time t=1

```
[if_name=eth0] : [{"up", 1 second"}]
[if_name=eth1] : [{"down", 1 second"}]
[if_name=eth3] : [{"up", 1 second"}]
```

Assume the following input at time t=2

```
[if_name=eth0] : "down"
[if_name=eth1] : "down"
[if_name=eth3] : "up"
```

We have the following output at time t=2

```
[if_name=eth0] : [{"up", 1 second"}, {"down", 2 seconds"}]
[if_name=eth1] : [{"down", 1 second"}]
[if_name=eth3] : [{"up", 1 second"}]
```

Assume the following input at time t=3

```
[if_name=eth0] : "up"
[if_name=eth1] : "down"
[if_name=eth3] : "up"
```

We have the following output at time t=3

```
[if_name=eth0] : [{"up", 1 second"}, {"down", 2 seconds"}, {"up", 3 seconds"}]
[if_name=eth1] : [{"down", 1 second"}]
[if_name=eth3] : [{"up", 1 second"}]
```

Assume the following input at time t=4

```
[if_name=eth0] : "down"
[if_name=eth1] : "down"
[if_name=eth3] : "up"
```

We have the following output at time t=4

```
[if_name=eth0] : [{"down", 2 seconds"}, {"up", 3 seconds"}, {"down", 4 seconds"}]
[if_name=eth1] : [{"down", 1 second"}]
[if_name=eth3] : [{"up", 1 second"}]
```

If the expressions are used for max_samples or total_duration, then they are evaluated for each input item and the corresponding key is added for each output item.

```
max_samples: context.ref_max_samples * 2
total_duration: context.ref_duration * 2
```

Sample input:

```
[if_name=eth0, ref_max_samples=10, ref_duration=60] : "up"
[if_name=eth1, ref_max_samples=20, ref_duration=120] : "down"
```

Output

```
[if_name=eth0, max_samples=20, duration=120] : "up"
[if_name=eth1, max_samples=40, duration=240] : "down"
```

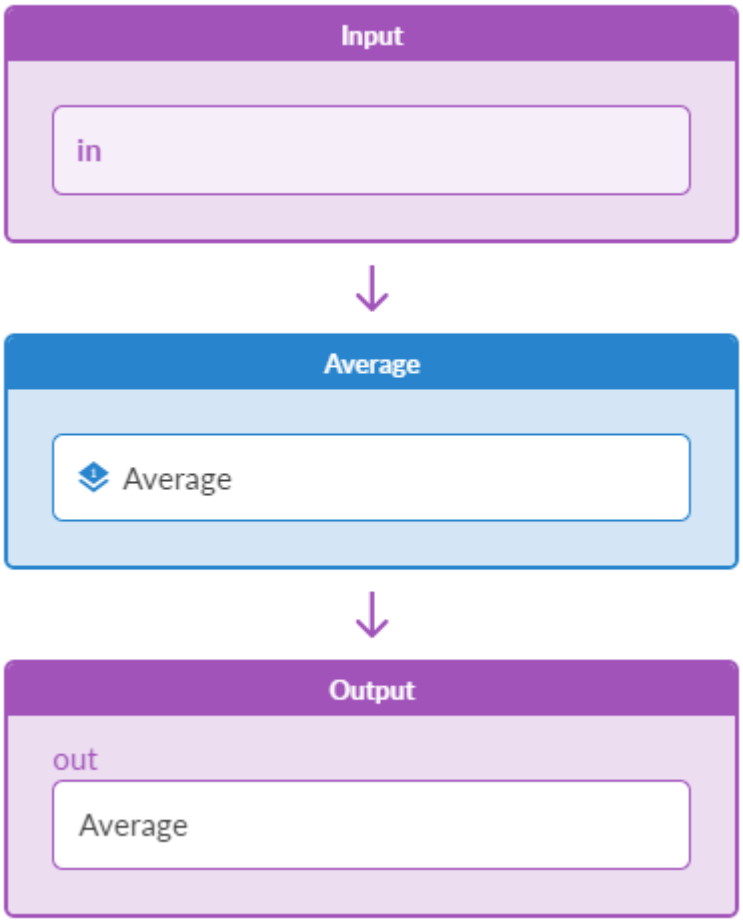
Processor: Average

The Average processor groups as described by **Group by**, then calculates averages and outputs one average for each group.

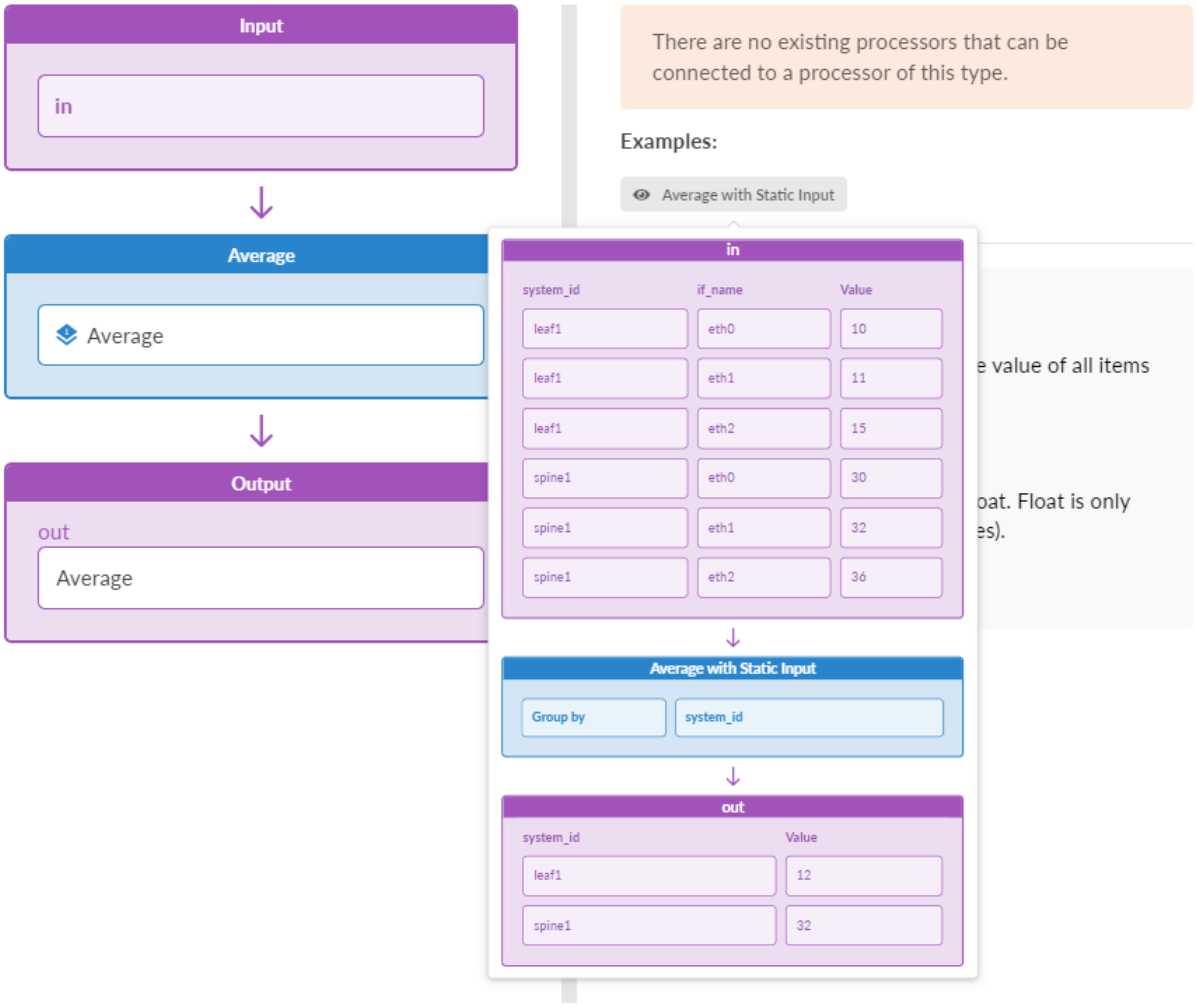
Parameter	Description
Input Types	Table (number), Table (number, accumulate=True)
Output Types	Table(number)

(Continued)

Parameter	Description
Group by (group_by)	Accepts a list of property names to group input items into output items, produces only one output group for the empty list.Most processors take input and produce output. Many of them produce one output per input (for example, if input is a DSS, output is a DSS of same size). However, some processors reduce the size of the output relative to the size of the input. Effectively, they partition the input into groups, run some calculation on each of the groups that produce a single value per each group, and use that as output. Clearly, the size of the output set depends on the grouping scheme. We call such processors **grouping processors** and they all take the **Group by** configuration parameter. In the case of an empty list, the input is considered to be a single group; thus, the output is of size 1 and either N, DS, or TS. If a list of property names is specified, for example ["system_id", "iface_role"], or a single property is specified, for example ["system_id"], we divide the input into groups such that for each group, every item in the group has the same values for the given list of property names. See the "standard deviation processor" on page 1703 example for how this works. The output type of a processor depends on a value of the group_by parameter; for an empty list, a processor produces a single value result, such as N, DS, or T, and for grouping by one or more properties it returns a set result, such as NS, DSS, or TS.
Enable Streaming (enable_streaming)	Makes samples of output stages streamed if enabled. An optional boolean that defaults to False. If set to True, all output stages of this processor are streamed in the generic protobuf schema.



In the Add Processor window of the Apstra UI, hover over the "Average with Static Input" tooltip for a visual example of how the Average processor functions.

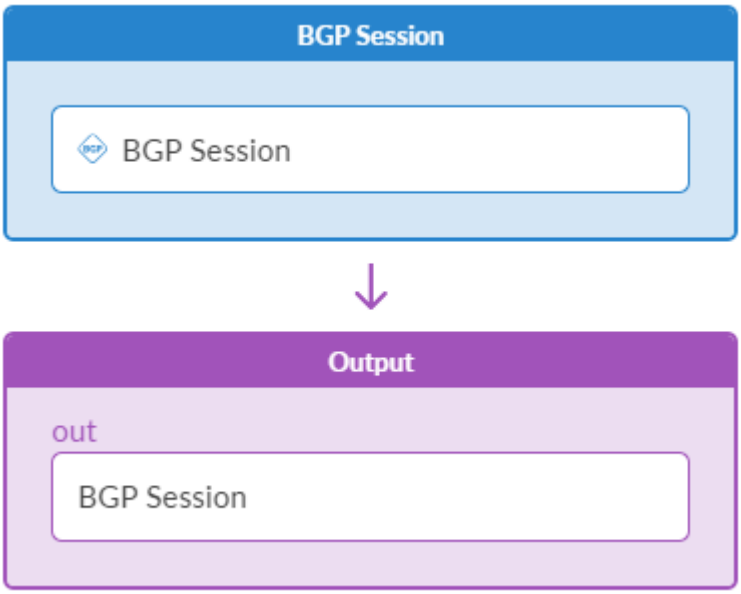


Example: Average

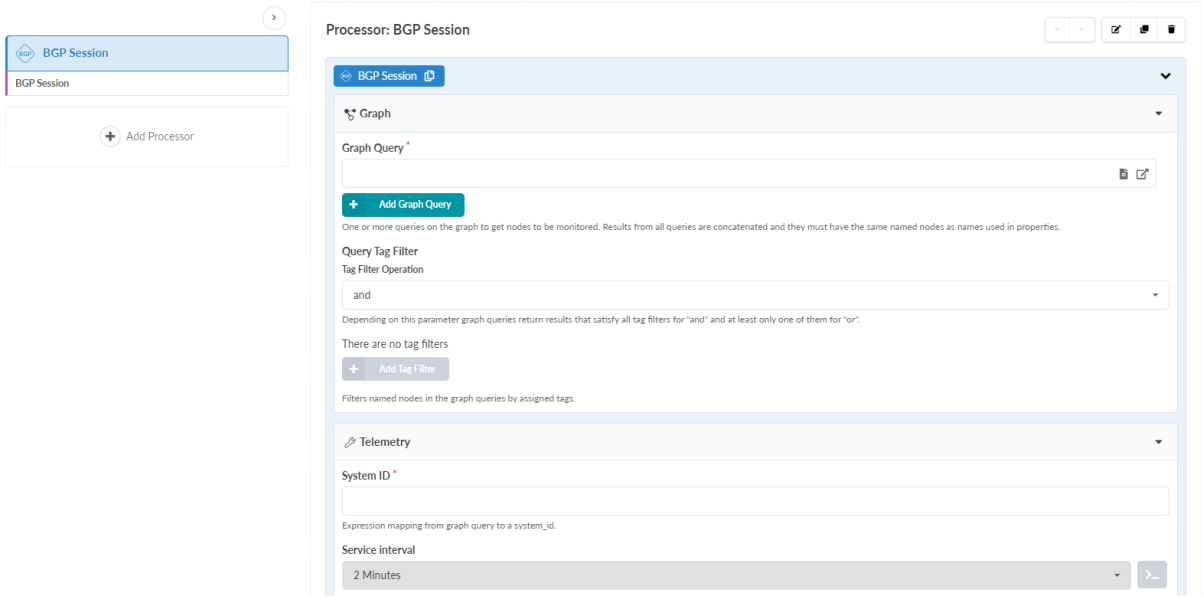
See "standard deviation" on page 1703 example. It's the same except we calculate average instead of standard deviation.

Processor: BGP Session

The BGP Session processor collects metrics and statistics on the BGP sessions across all VRFs and address-families. The reported data includes: Session state (Ex: "Established", "Connect",..) and flap counts.



Configuration options:

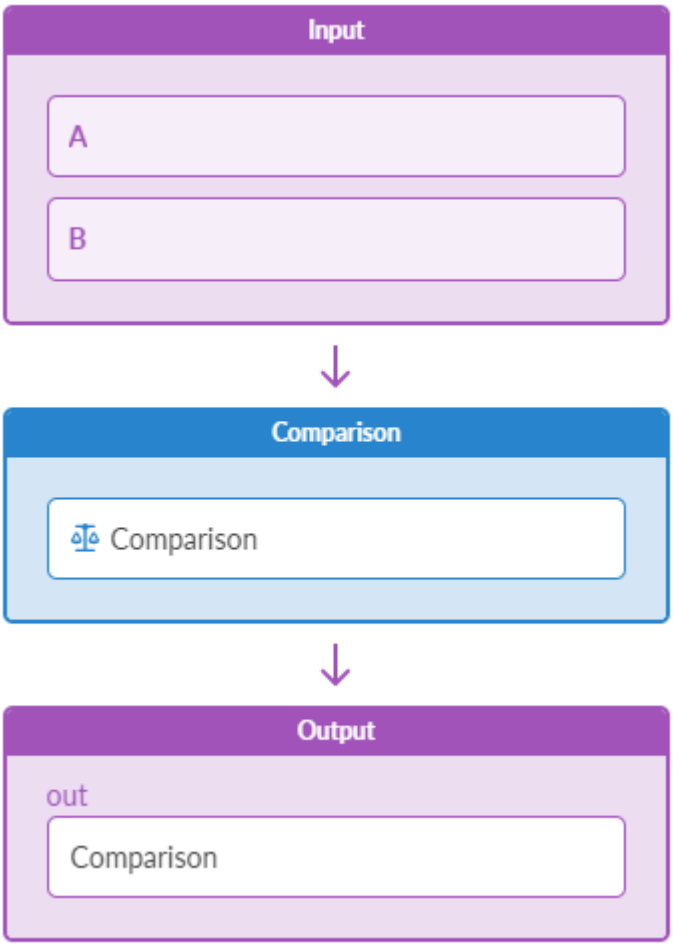


Processor: Comparison

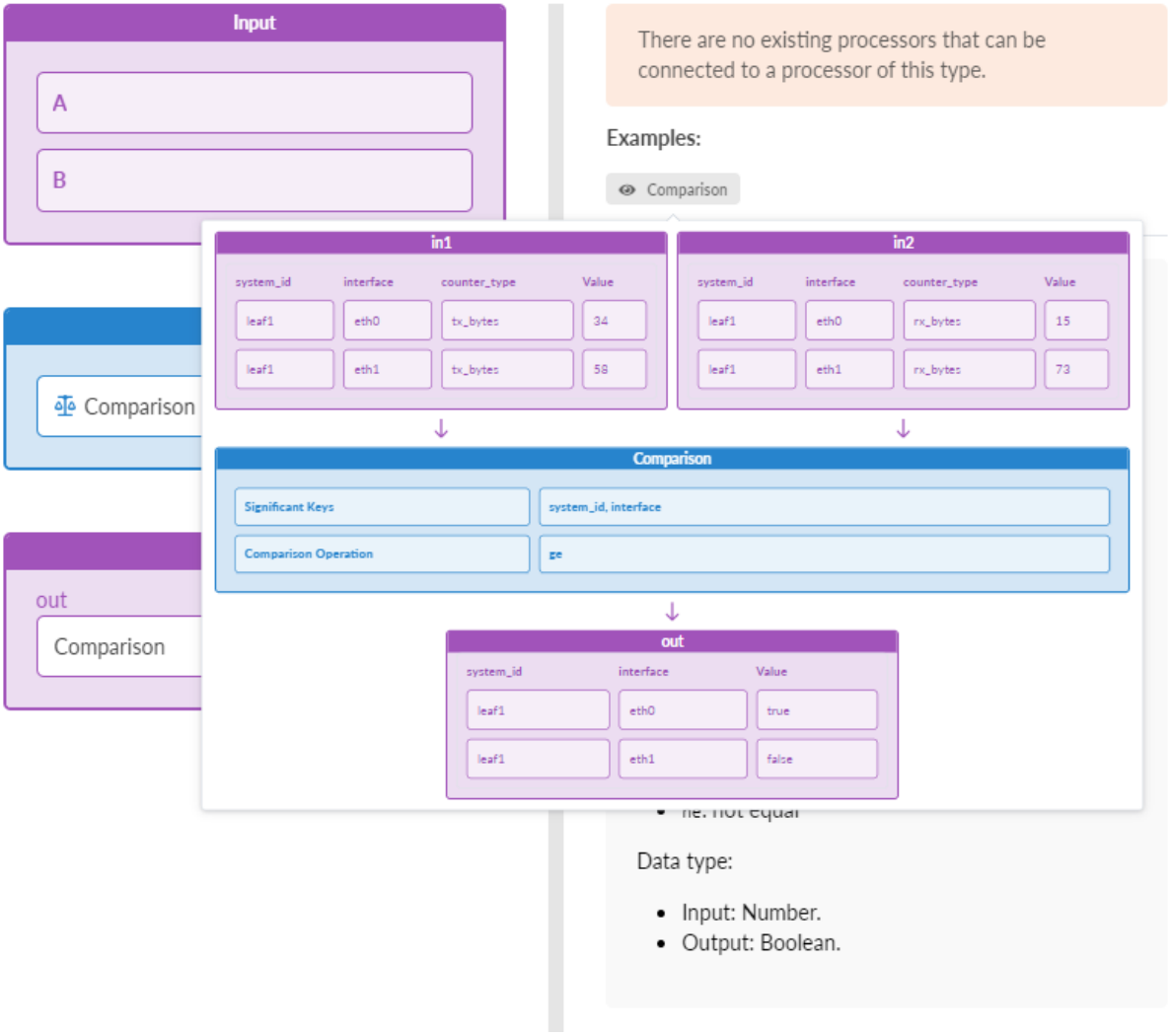
IN THIS SECTION

The Comparison processor takes two Table(number) inputs: 'A' and 'B'. It then matches corresponding items from the inputs by their keys, and performs a comparison operation defined by the 'operation' configuration property. If the inputs have different sets of keys, the 'significant_keys' configuration property should be set, which is a list of keys used to map items from the inputs. Otherwise, if the inputs set of keys are different, no items will be matched and an empty result is returned. Also, inputs and significant_keys (if specified) must allow only 1:1 item mapping from 'A' to 'B'. If it allows to match one item from 'A' to more than one item from 'B' and vice versa, the probe goes into error state.

Parameters	Description
Input Types	Tablev(number)
Output Types	Table (discrete state): true or false
Comparison Operation (operation)	Operation for comparing operands. le (less than or equal), ne (not equal), ge (greater than or equal), gt (greater than), lt (less than), eq (equal)
Significant Keys (significant_keys)	List of keys to map items from the inputs for applying the specified operation. It is typically used by processors that take multiple inputs and perform operations on them. When inputs have the same sets of keys it does not need to be specified. When inputs have different sets of keys, it must be specified and it must allow only 1:1 items mapping from the given inputs, otherwise the probe will go into error state.
Enable Streaming (enable_streaming)	Makes samples of output stages streamed if enabled. An optional boolean that defaults to False. If set to True, all output stages of this processor are streamed in the generic protobuf schema.



From the Add Processor window in the Apstra UI, hover over the "Comparison" tooltip for a visual example of how the Comparison Processor functions.



Example: Comparison

```
significant_keys: ["system_id", "interface"]
operation: "ge"
```

Input A:

```
[system_id=leaf1,interface=eth0,counter_type=tx_bytes]: 34
[system_id=leaf1,interface=eth1,counter_type=tx_bytes]: 58
```

Input B:

```
[system_id=leaf1,interface=eth0,counter_type=rx_bytes]: 15
[system_id=leaf1,interface=eth1,counter_type=rx_bytes]: 73
```

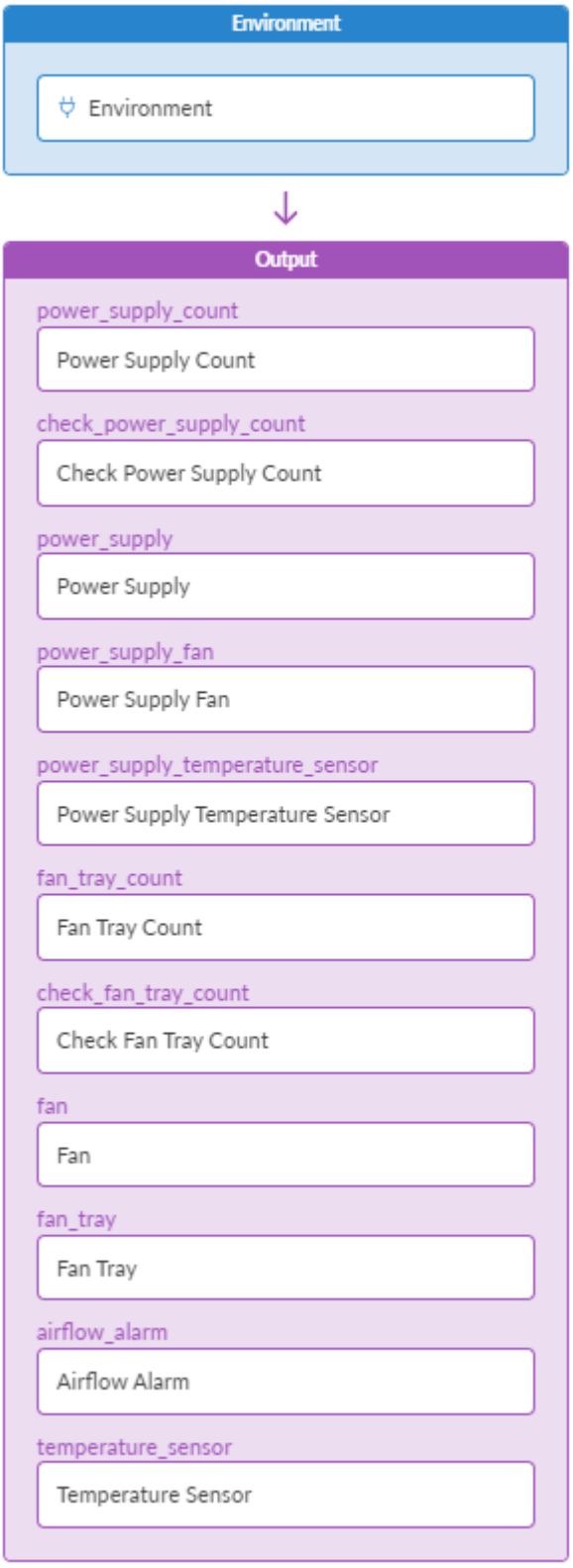
Output (Discrete-State-Set):

```
[system_id=leaf1,interface=eth0]: "true"
[system_id=leaf1,interface=eth1]: "false"
```

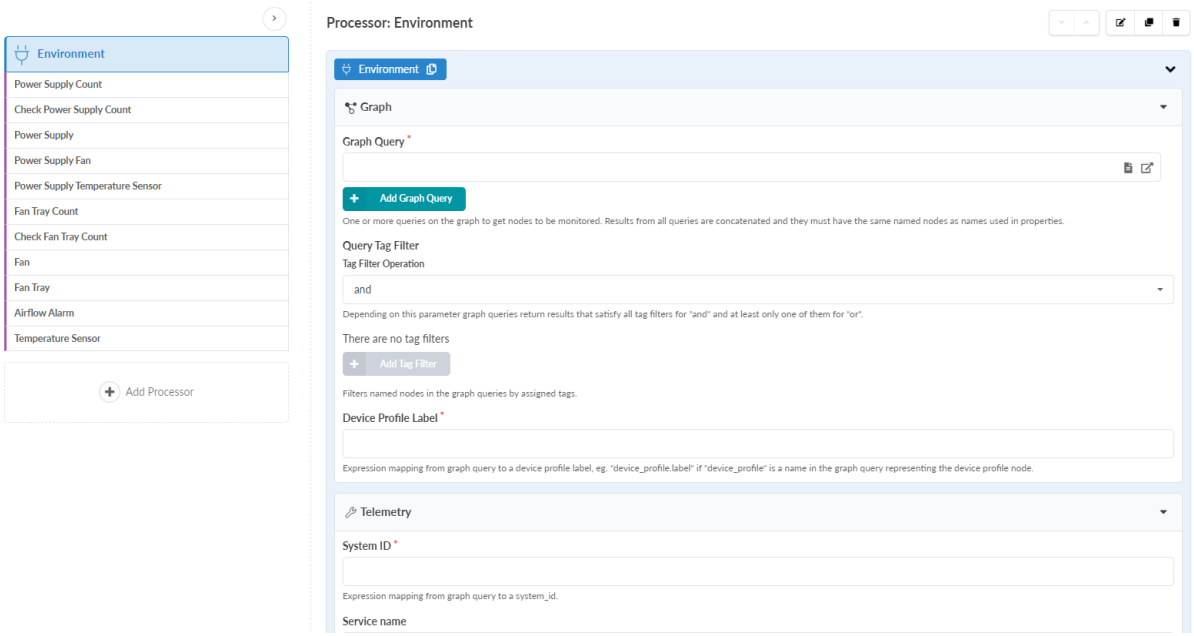
Processor: Environment

The Environment Processor collects metrics and statistics of the environmental elements including:

- Power Supply:
  - Total units count and functioning units count
  - Status of each unit
  - Status of each fan in each unit
- Fan:
  - Total fan tray count
  - Status and speed (RPM)
  - Airflow Direction
- Temperature:
  - Sensor status and temperature measurement



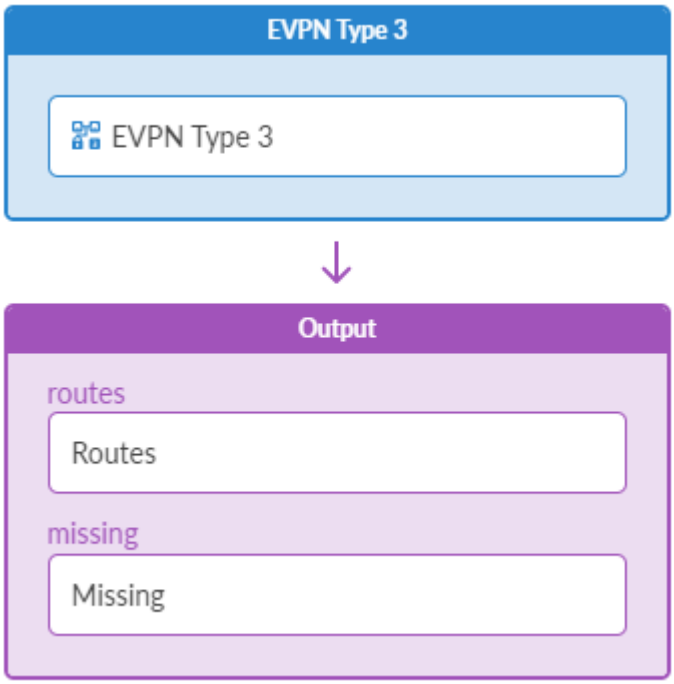
From the Add Processor window in the Apstra UI, clicking **Add** displays configuration options for the Environment Processor.



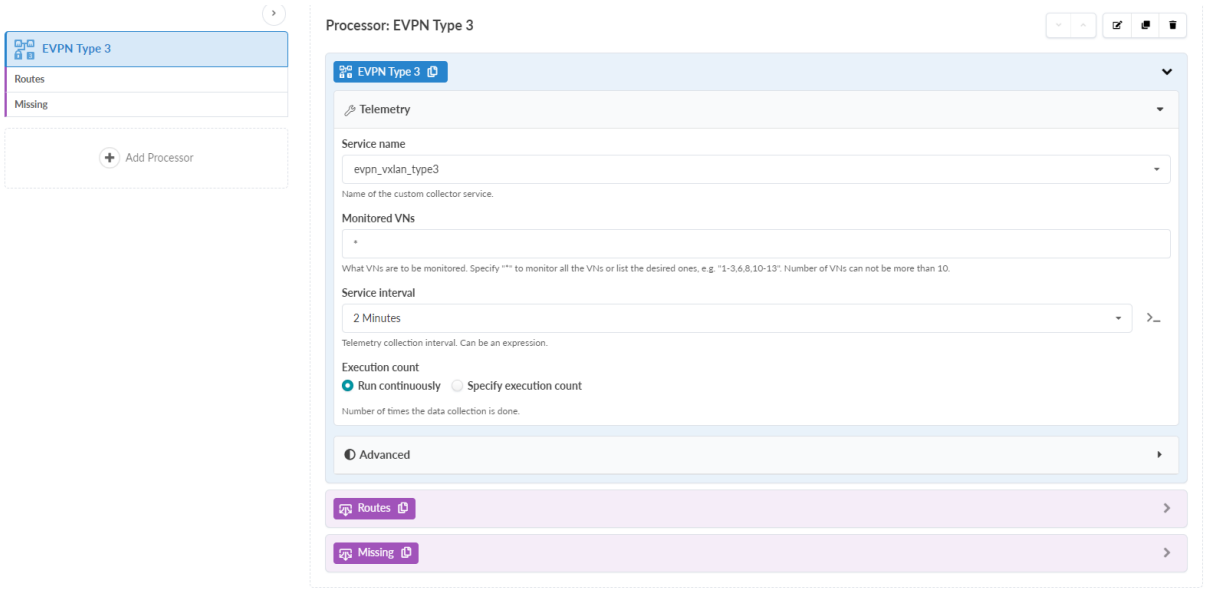
Processor: EVPN Type 3

The EVPN Type 3 processor generates a configuration containing expectations of EVPN type 3 routes.

Parameter	Description
Execution count	Number of times the data is collected
Monitored VNs	The VNs to be monitored. Specify * to monitor all the VNs or list the desired ones, e.g. "1-3,6,8,10-13".
Service Interval	Telemetry collection interval in seconds.
Service name	Name of the custom collector service.
Enable Streaming (enable_streaming)	Makes samples of output stages streamed if enabled. An optional boolean that defaults to False. If set to True, all output stages of this processor are streamed in the generic protobuf schema.



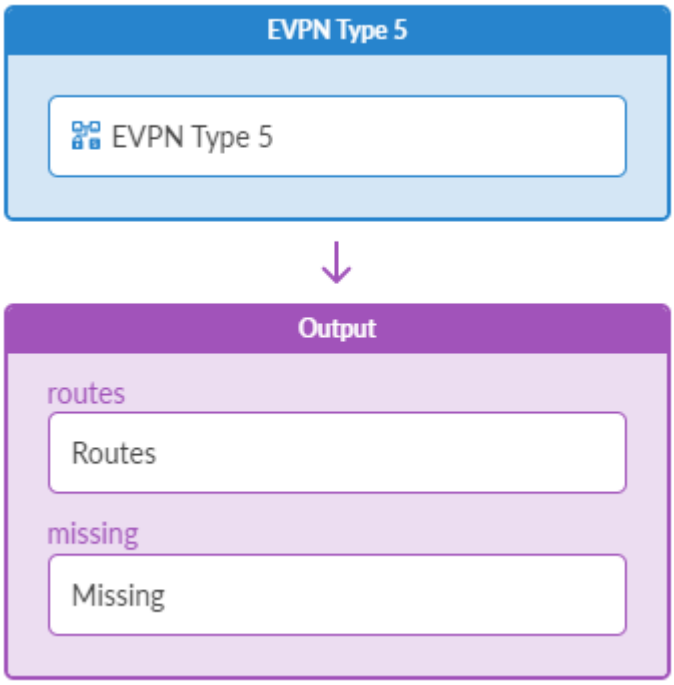
Configuration options:



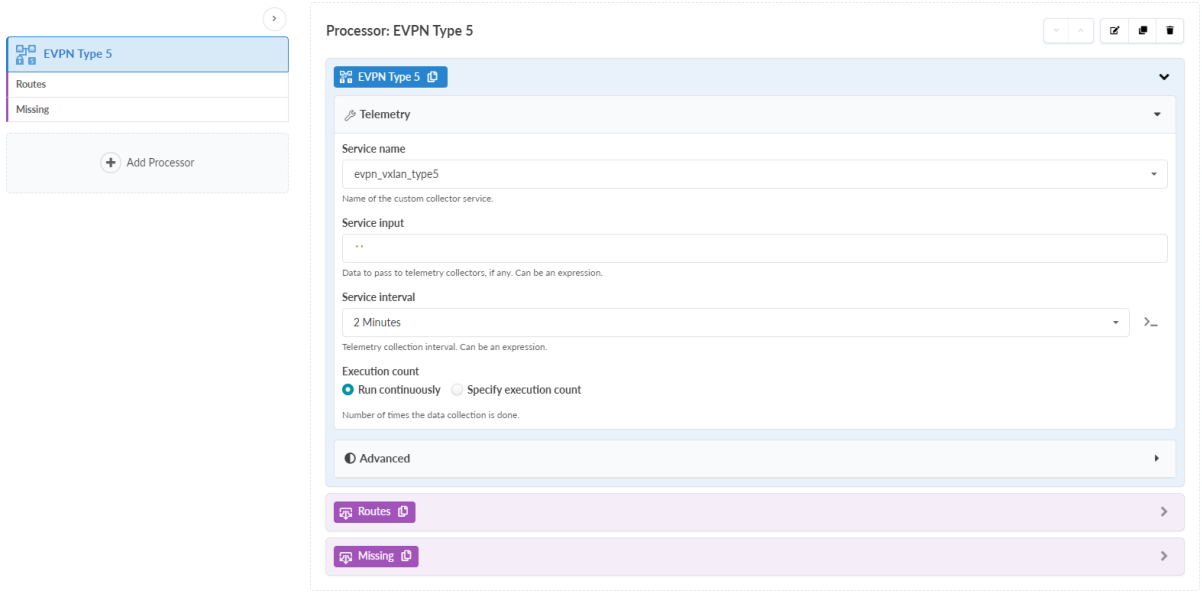
Processor: EVPN Type 5

The EVPN Type 5 processor generates a configuration containing expectations of EVPN type 5 routes.

Parameter	Description
Input Types	Number-Set (NS), Discrete-State-Set (DSS)
Output Types	NSTS, DSSTS
Execution count	Number of times the data collection is done.
Service input	Data to pass to telemetry collectors, if any.
Service Interval	Telemetry collection interval in seconds.
Service name	Name of the custom collector service.
Enable Streaming (enable_streaming)	Makes samples of output stages streamed if enabled. An optional boolean that defaults to False. If set to True, all output stages of this processor are streamed in the generic protobuf schema.



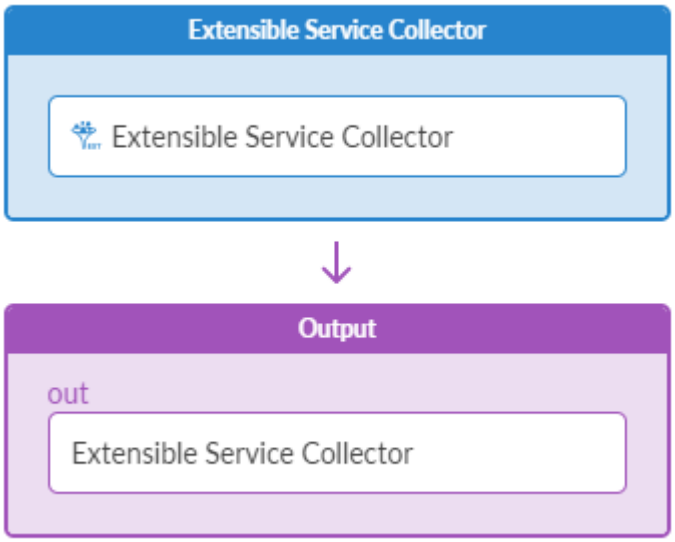
Configuration options:



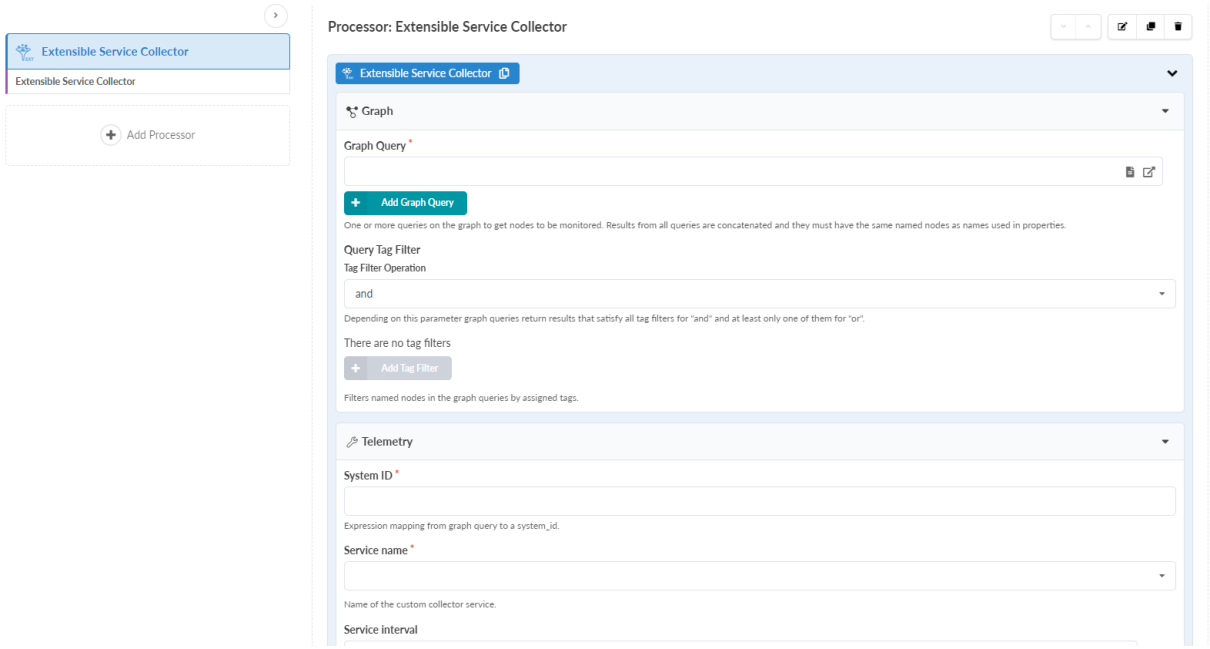
Processor: Extensible Service Collector

The Extensible Service Collector Processor ingests data produced by custom telemetry services of type "extensible". Use this processor for any service created with custom telemetry collectors.

Supports both static series (graph-driven telemetry collection) and dynamic series (collector-driven telemetry collection).



Configuration options:



Processor: Generic Graph Collector

IN THIS SECTION

The Generic Graph Collector Processor imports data from the graph into the output stage, depending on the configuration (a graph query).

'graph query' and 'additional properties' behave as in other source processors. Importantly, the expression in the 'value' field yields a value per each item. Thus, unique to this source processor, values come from the graph rather than from device telemetry.

Parameter	Description
Input Types	No inputs. This is a source processor.
Output Types	Table (discrete state or number or text)
Data Type	Type of data the service collects: numbers (ns) (such as device temperature), discrete states (dss) (such as device status), text or tables

(Continued)

Parameter	Description

Graph Query (graph_query)	One or more queries on graph specified as strings, or a list of such queries. (String will be deprecated in a future release.) Multiple queries should provide all the named nodes referenced by the expression fields (including additional_properties). Graph query is executed on the "operation" graph. 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"]".
	
	In collector processors (`*_collector`, `if_counter`) it is used to choose a set of nodes for further processing (for example, all leaf devices, or all interfaces between leaf and spine devices)
	
	In other processors it is used for general parameterization and it is only supported as a list of queries.
	
	`graph_query: "node("system", role="leaf", name="system").`
	`out("hosted_interfaces").`
	`node("interface", name="iface").out("link").`
	`node("link", role="spine_leaf")"`
	
	`graph_query: ["node("system", role="leaf", name="system")",`
	`"node("system", role="spine", name="system")"]`

Query Expansion	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.

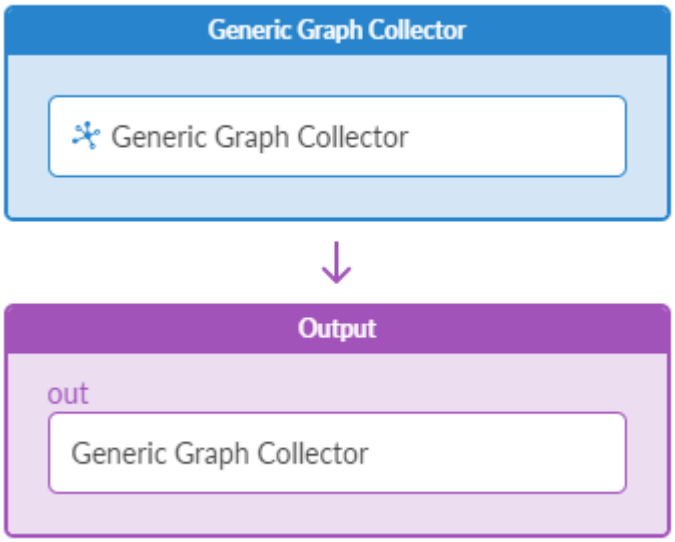
(Continued)

Parameter	Description
Query Group by (query_group_by)	List (of strings) of node and relationship names used in the graph query to group query results by. Each element in this list represents a named node or relationship matcher in the `graph_query` field.It is not an expression to be consistent with existing `group_by` field in grouping processors. Non-expression is simple and more intuitive. When grouping is active (`query_group_by` is not null), query results are d by the specified list of names, where one output item is created per each group. In this case, the expressions can only access matcher names specified in `query_group_by` and the query results for each group are accessed using a new `group_items` variable. The `group_items` variable is a list of query results, where each result has named nodes/relationships, not present in `query_group_by`. The following list describes the behavior for various values of this field: • Value of query_group_by field - Semantics • Omitted or provided as json null (ala None in Python) - No grouping is done. This is equivalent to current behavior of extensible_data_collector. Using 'group_items' in this case is not permitted and results in probe error state. • Empty list ([]) - Produces one group containing all the query results. • One or more matcher names - The query results are grouped by the specified nodes or relationships. If this list covers all available matchers in the query, the number of groups is equal to the number of query results.
Query Tag Filter (query_tag_filter)	Filters named nodes in the graph queries by assigned tags.

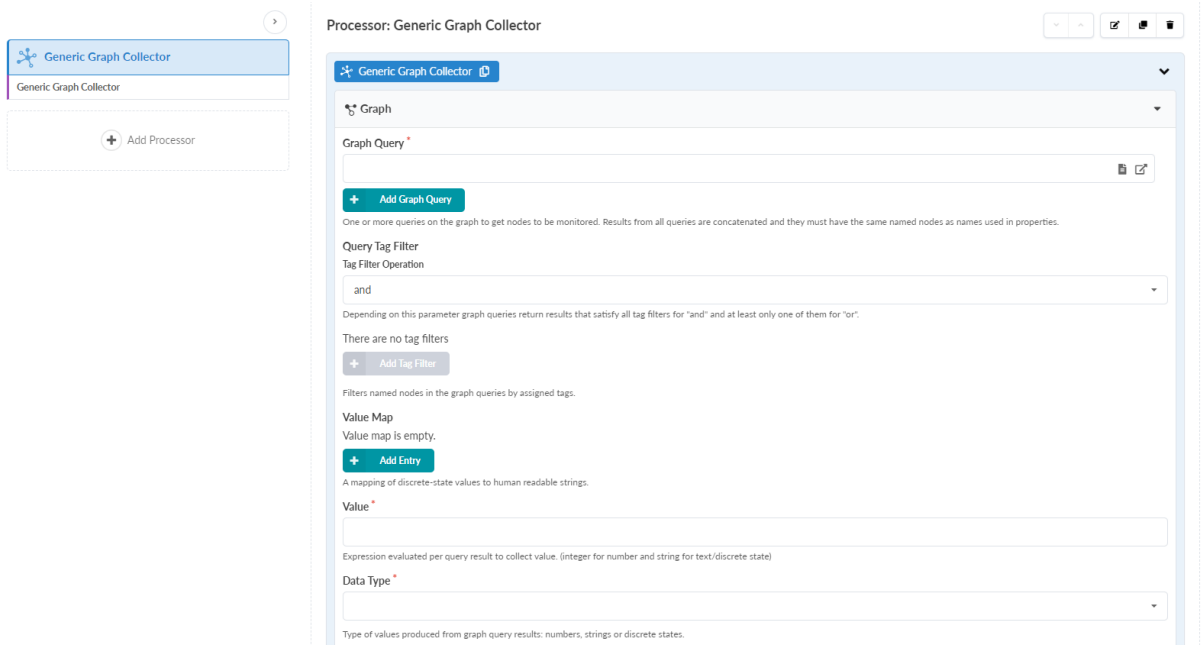
Value Map	A mapping of discrete-state values to human readable strings. A dictionary with all possible Discrete-State-Set states mapped to human-readable representation; applicable for Discrete-State-Set data (that is, when data_type is 'dss') only.
	
	`{`
	`"0": "unknown",`
	`"1": "down",`
	`"2": "up",`
	`"3": "missing"`
	`}`

(Continued)

Parameter	Description
Value (value)	Expression evaluated per query result to collect value. (integer for NS and string for TS/DSS)
Additional Keys	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. The value of this property is evaluated for each item to associate items with metrics provided by a corresponding collector service. The association is done by keys because each collector reports a set of metrics where each metric is identified by a key in a format that is specific for each collector.
Enable Streaming (enable_streaming)	Makes samples of output stages streamed if enabled. An optional boolean that defaults to False. If set to True, all output stages of this processor are streamed in the generic protobuf schema.



Configuration options:



Example: Generic Graph Collector

```
graph_query: "node("system", role="leaf", name="system").
              out("hosted_interfaces").
              node("interface", name="iface").out("link").
              node("link", role="spine_leaf")"
system_id: "system.system_id"
interface: "iface.if_name"
value: "iface.if_type"
data_type: "dss"
value_map: {0: "ip", 1: "loopback", ...}
```

Sample output (DSS):

```
[system_id=leaf1,interface=eth0]: "ip"
[system_id=leaf1,interface=eth1]: "ip"
```

Processor: Generic Service Data Collector

The Generic Service Data Collector processor collects data supplied by a custom service that is not 'lldp', 'bgp' or 'interface'. Service name is specified as 'service_name', service specific key is specified as 'key', 'data_type' to specifies if the collected data is numbers or discrete state values, and 'value_map' for the specific data could be specified as well.

Parameter	Description
Input Types	No inputs. This is a source processor.
Output Types	Discrete-State-Set (DSS), Number-Set (NS), TS (based on data_type)
Data Type	Type of data the service collects: numbers (ns) (such as device temperature), discrete states (dss) (such as device status), text or tables

Graph Query (graph_query)	One or more queries on graph specified as strings, or a list of such queries. (String will be deprecated in a future release.) Multiple queries should provide all the named nodes referenced by the expression fields (including additional_properties). Graph query is executed on the "operation" graph. 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"]".
	
	In collector processors (`*_collector`, `if_counter`) it is used to choose a set of nodes for further processing (for example, all leaf devices, or all interfaces between leaf and spine devices)
	
	In other processors it is used for general parameterization and it is only supported as a list of queries.
	
	`graph_query: "node("system", role="leaf", name="system").`
	`out("hosted_interfaces").`
	`node("interface", name="iface").out("link").`
	`node("link", role="spine_leaf")"`
	
	`graph_query: ["node("system", role="leaf", name="system")",`
	`"node("system", role="spine", name="system")"]`

Query Expansion	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.

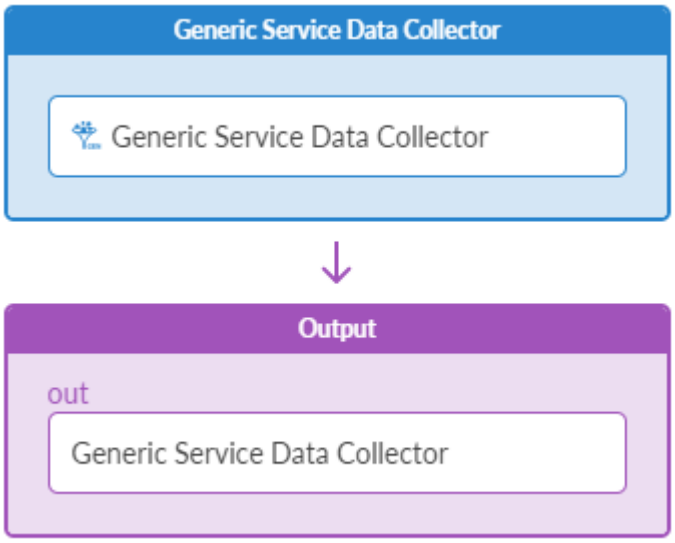
(Continued)

Parameter	Description
Query Group by (query_group_by)	List (of strings) of node and relationship names used in the graph query to group query results by. Each element in this list represents a named node or relationship matcher in the `graph_query` field.It is not an expression to be consistent with existing `group_by` field in grouping processors. Non-expression is simple and more intuitive. When grouping is active (`query_group_by` is not null), query results are d by the specified list of names, where one output item is created per each group. In this case, the expressions can only access matcher names specified in `query_group_by` and the query results for each group are accessed using a new `group_items` variable. The `group_items` variable is a list of query results, where each result has named nodes/ relationships, not present in `query_group_by`. The following list describes the behavior for various values of this field: • Value of query_group_by field - Semantics • Omitted or provided as json null (ala None in Python) - No grouping is done. This is equivalent to current behavior of extensible_data_collector. Using 'group_items' in this case is not permitted and results in probe error state. • Empty list ([]) - Produces one group containing all the query results. • One or more matcher names - The query results are grouped by the specified nodes or relationships. If this list covers all available matchers in the query, the number of groups is equal to the number of query results.
Query Tag Filter (query_tag_filter)	Filters named nodes in the graph queries by assigned tags.

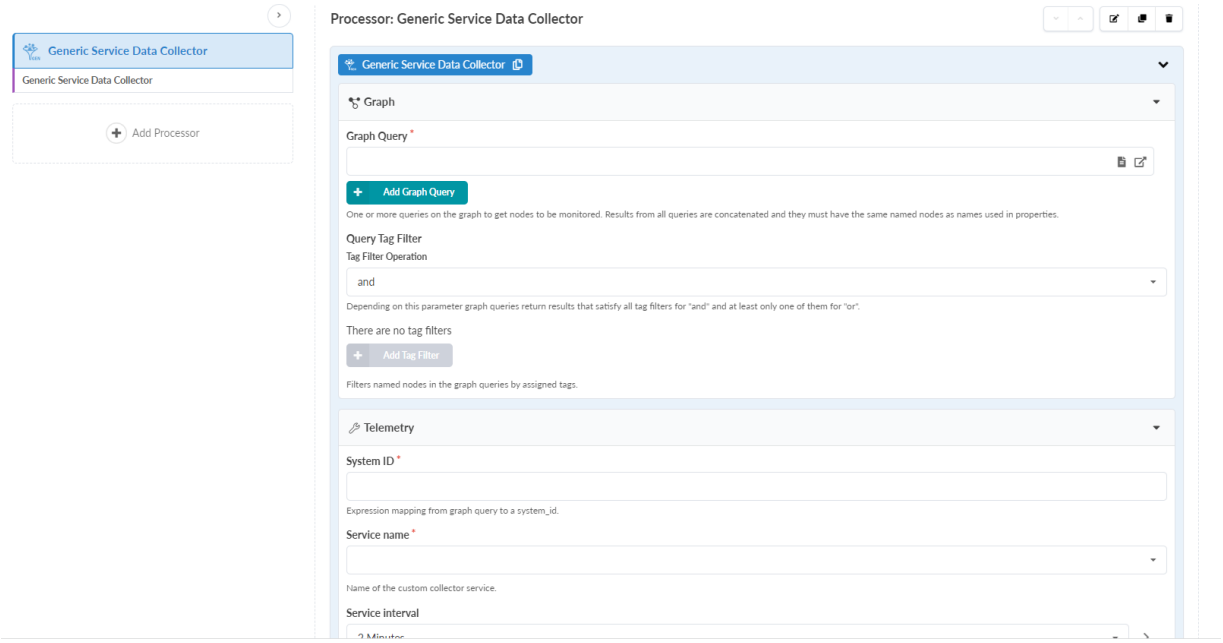
Value Map	A mapping of discrete-state values to human readable strings. A dictionary with all possible Discrete-State-Set states mapped to human-readable representation; applicable for Discrete-State-Set data (that is, when data_type is 'dss') only.
	
	`{`
	`"0": "unknown",`
	`"1": "down",`
	`"2": "up",`
	`"3": "missing"`
	`}`

(Continued)

Parameter	Description
Key (key)	Expression mapping from graph query to whatever key is necessary for the service.
Service name (service_name)	Name of the custom collector service.
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.
Execution count	Number of times the data collection is done.
Service input (service_input)	Data to pass to telemetry collectors, if any. Can be an expression.
Service interval (service_interval)	Telemetry collection interval in seconds. Can be an expression.
Additional Keys	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. The value of this property is evaluated for each item to associate items with metrics provided by a corresponding collector service. The association is done by keys because each collector reports a set of metrics where each metric is identified by a key in a format that is specific for each collector.
Enable Streaming (enable_streaming)	Makes samples of output stages streamed if enabled. An optional boolean that defaults to False. If set to True, all output stages of this processor are streamed in the generic protobuf schema.



Configuration options:



Processor: Interface Counters

IN THIS SECTION

- Example: Interface Counter | **1652**

The Interface Counters processor selects interfaces according to the configuration and outputs counter stats of the specified types (such as 'tx_bytes').

Parameter	Description
Input Types	No inputs. This is a source processor.
Output Types	Table(number)
Counter Type (counter_type)	A type of an interface counter. enum of: tx_unicast_packets, tx_broadcast_packets, tx_multicast_packets, tx_bytes, tx_error_packets, tx_discard_packets, rx_unicast_packets, rx_broadcast_packets, rx_multicast_packets, rx_bytes, rx_error_packets, rx_discard_packets.

Graph Query (graph_query)	One or more queries on graph specified as strings, or a list of such queries. (String will be deprecated in a future release.) Multiple queries should provide all the named nodes referenced by the expression fields (including additional_properties). Graph query is executed on the "operation" graph. 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"]".
	
	In collector processors (`*_collector`, `if_counter`) it is used to choose a set of nodes for further processing (for example, all leaf devices, or all interfaces between leaf and spine devices)
	
	In other processors it is used for general parameterization and it is only supported as a list of queries.
	
	`graph_query: "node("system", role="leaf", name="system").`
	`out("hosted_interfaces").`
	`node("interface", name="iface").out("link").`
	`node("link", role="spine_leaf")"`
	
	`graph_query: ["node("system", role="leaf", name="system")",`
	`"node("system", role="spine", name="system")"]`

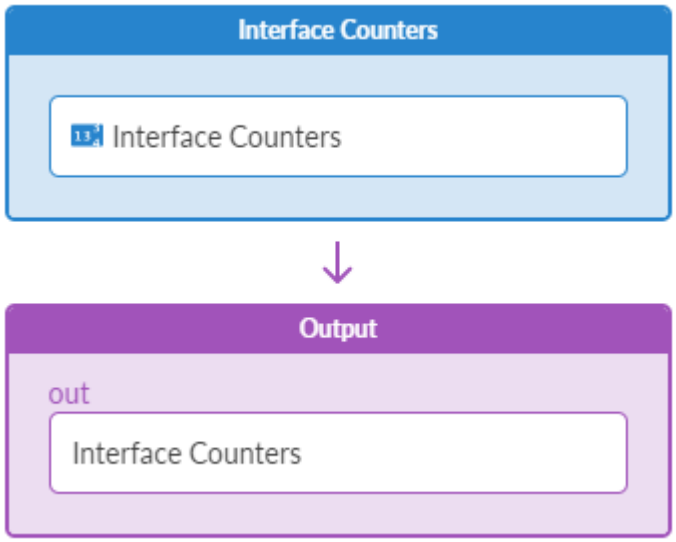
Query Expansion	For every path, originally returned by graph queries, passed to each generator the latter one produces a set of items and for each item it produces a new path extended by a corresponding property name which value is set of a value of the produced item.

(Continued)

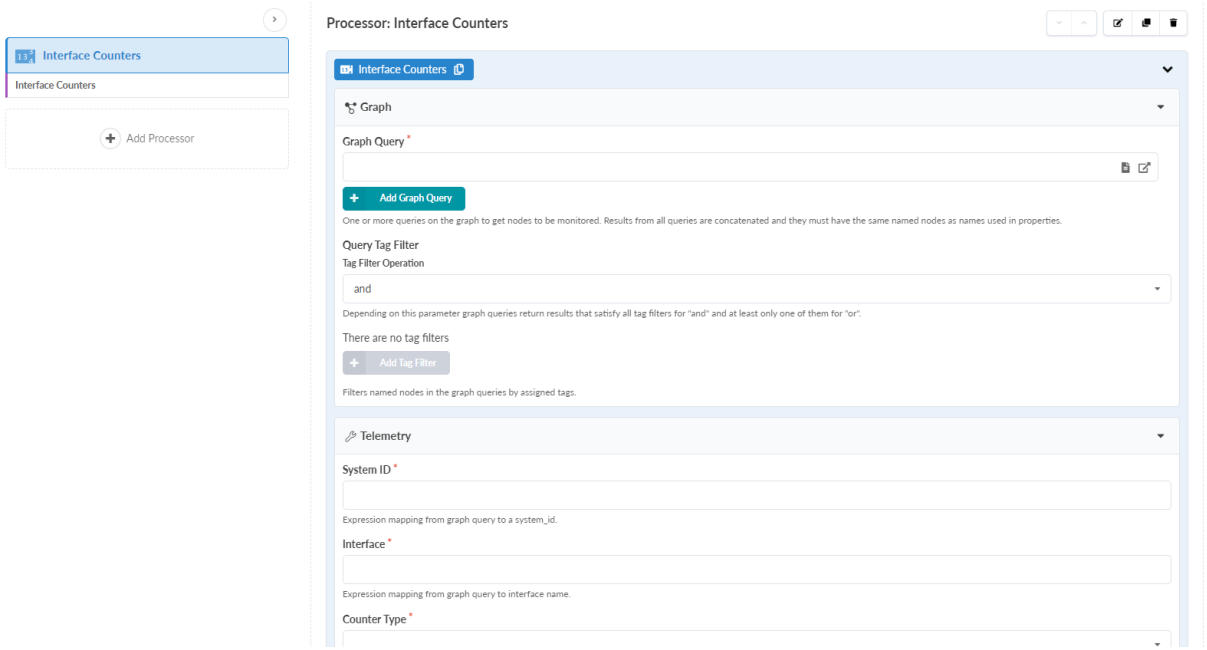
Parameter	Description
Query Group by (query_group_by)	List (of strings) of node and relationship names used in the graph query to group query results by. Each element in this list represents a named node or relationship matcher in the `graph_query` field.It is not an expression to be consistent with existing `group_by` field in grouping processors. Non-expression is simple and more intuitive. When grouping is active (`query_group_by` is not null), query results are d by the specified list of names, where one output item is created per each group. In this case, the expressions can only access matcher names specified in `query_group_by` and the query results for each group are accessed using a new `group_items` variable. The `group_items` variable is a list of query results, where each result has named nodes/relationships, not present in `query_group_by`. The following list describes the behavior for various values of this field: • Value of query_group_by field - Semantics • Omitted or provided as json null (ala None in Python) - No grouping is done. This is equivalent to current behavior of extensible_data_collector. Using 'group_items' in this case is not permitted and results in probe error state. • Empty list ([]) - Produces one group containing all the query results. • One or more matcher names - The query results are grouped by the specified nodes or relationships. If this list covers all available matchers in the query, the number of groups is equal to the number of query results.
Query Tag Filter (query_tag_filter)	Filters named nodes in the graph queries by assigned tags.
Interface (interface)	Expression mapping from graph query to interface name, e.g. "iface.if_name" if "iface" is a name in the graph query.
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.
Additional Keys	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. The value of this property is evaluated for each item to associate items with metrics provided by a corresponding collector service. The association is done by keys because each collector reports a set of metrics where each metric is identified by a key in a format that is specific for each collector.

(Continued)

Parameter	Description
Enable Streaming (enable_streaming)	Makes samples of output stages streamed if enabled. An optional boolean that defaults to False. If set to True, all output stages of this processor are streamed in the generic protobuf schema.



Configuration options:



Example: Interface Counter

```
graph_query: "node("system", name="system").out("hosted_interfaces").
              node("interface", name="iface").out("link").
              node("link", role="spine_leaf")"
counter_type: "rx_bytes"
system_id: "system.system_id"
interface: "interface.if_name"
role: "system.role"
```

In this example, we create a NSS that has an entry for rx_bytes (per second) per every interface in the system. Each entry is implicitly tagged by "system_id" and "interface". Furthermore, as we have specified an additional property, each entry is also tagged by role of the system.

```
[system_id=spine1,role=spine,key=eth0]: 10
[system_id=spine2,role=spine,key=eth1]: 11
[system_id=leaf0,role=leaf, key=swp1]: 12
```

Processor: Logical Operator

The Logical Operator processor calculates the logical operation, "AND" or "OR" of inputs. It takes two or more inputs that represent boolean values.

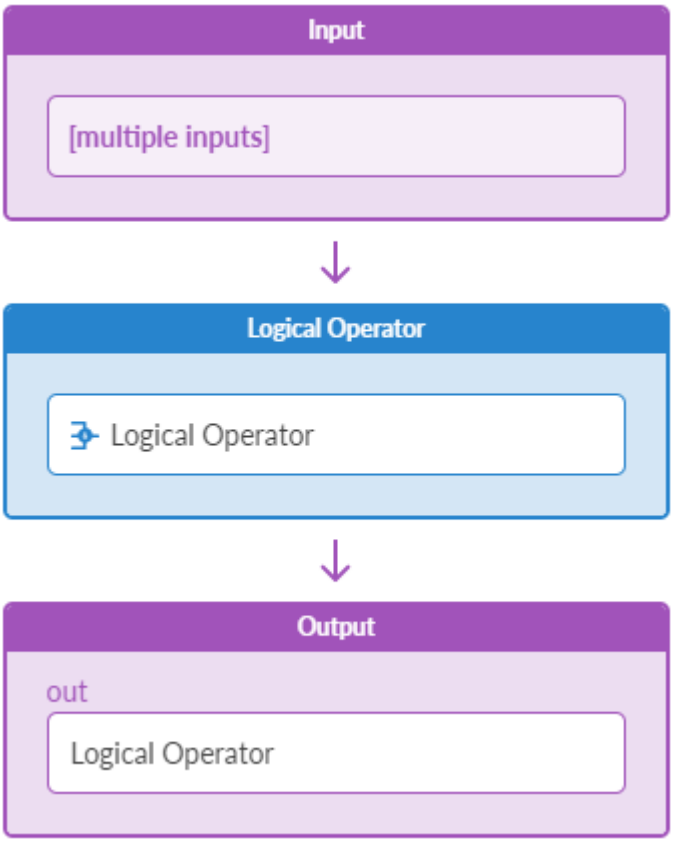
The property 'operation' specifies the logical operation. The property 'input_columns' specifies column names that input items should be taken from.

The series from the different input stages should match by their natural keys. If the list of keys does not allow a unique 1:1 mapping, or if the input stages have different sets of keys, use the "Significant Keys" property to specify the keys for the mapping.

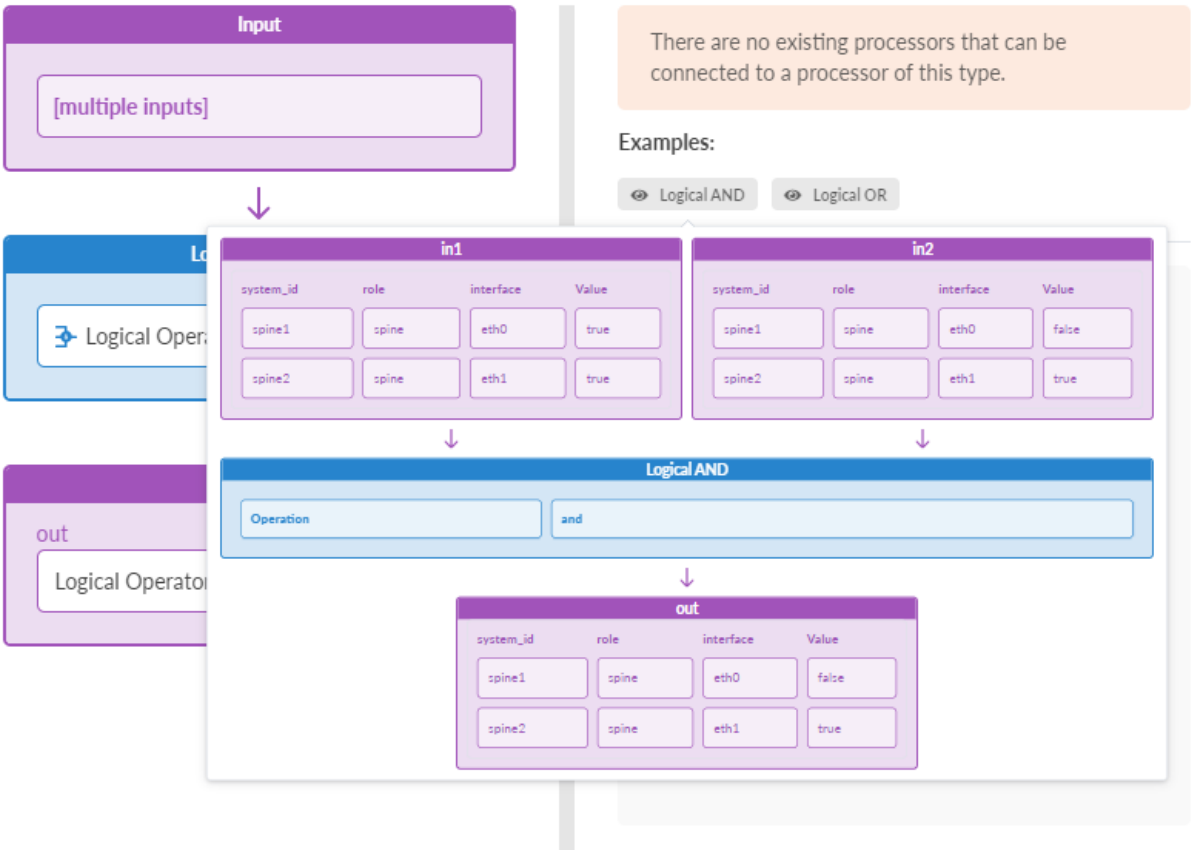
Parameter	Description
Input Types	Tables that contain discrete_state type column according to the 'input_columns' property or Table (discrete_state) if the 'input_columns' is not specified.
Output Types	Table (discrete state)
Operation	Logical operation type that is used for processing the input data

(Continued)

Parameter	Description
Significant Keys (significant_keys)	List of keys to map items from the inputs for applying the specified operation. It is typically used by processors that take multiple inputs and perform operations on them. When inputs have the same sets of keys it does not need to be specified. When inputs have different sets of keys, it must be specified and it must allow only 1:1 items mapping from the given inputs, otherwise the probe will go into error state.
Enable Streaming (enable_streaming)	Makes samples of output stages streamed if enabled. An optional boolean that defaults to False. If set to True, all output stages of this processor are streamed in the generic protobuf schema.



In the Apstra UI, hover over the "Logical AND" or "Logical OR" tooltips for visual examples of how the Logical Operator processor functions.

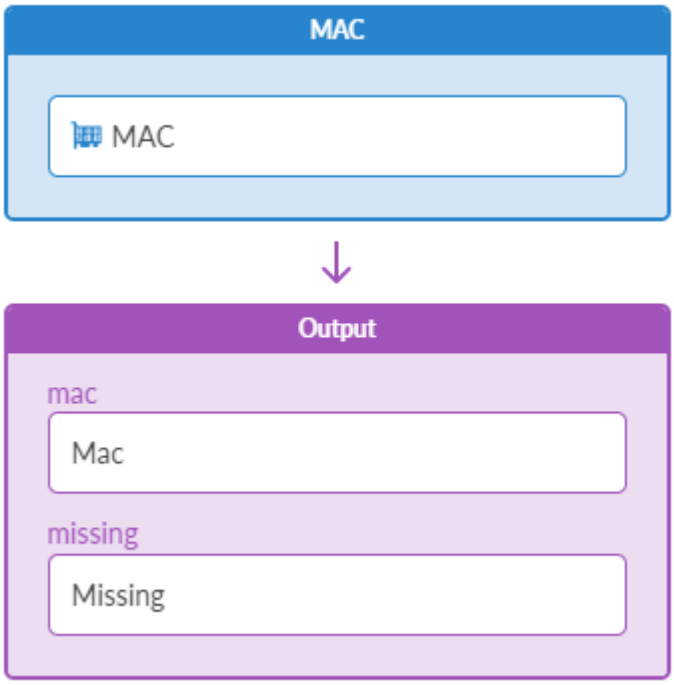


Processor: MAC

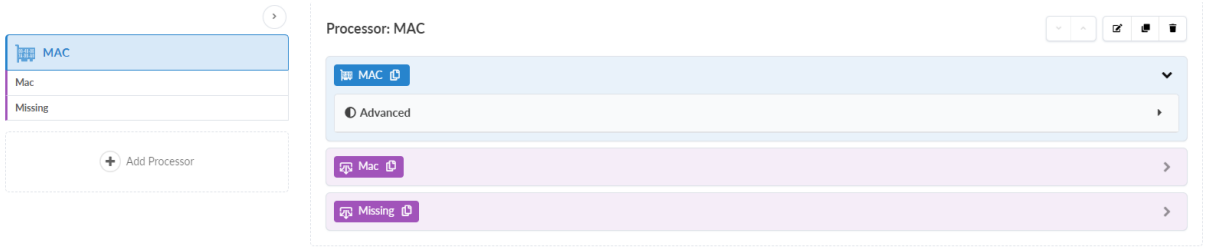
The MAC Monitor processor collect MAC address tables for all Leafs and Access-switches and compares to automatically generated expectations, derived from the Virtual Network intent.

The processor has two output stages:

- MAC Address Table: Dump of the system's MAC address table and reports for every MAC entry:
  - Interface name from which it has been learned
  - VLAN ID, VNI and Virtual Network label
  - Status, indicating if the entry is missing or not
- Missing MAC Counts: Provides on a per System per VNI basis the count of MAC entries missing according to the Virtual Network intent.



Configuration options:



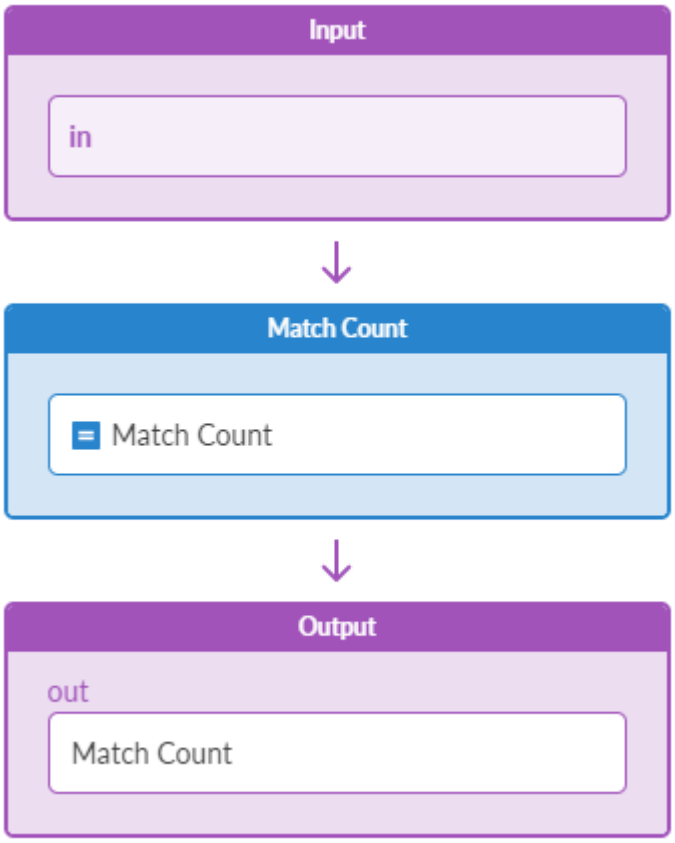
Processor: Match Count

IN THIS SECTION

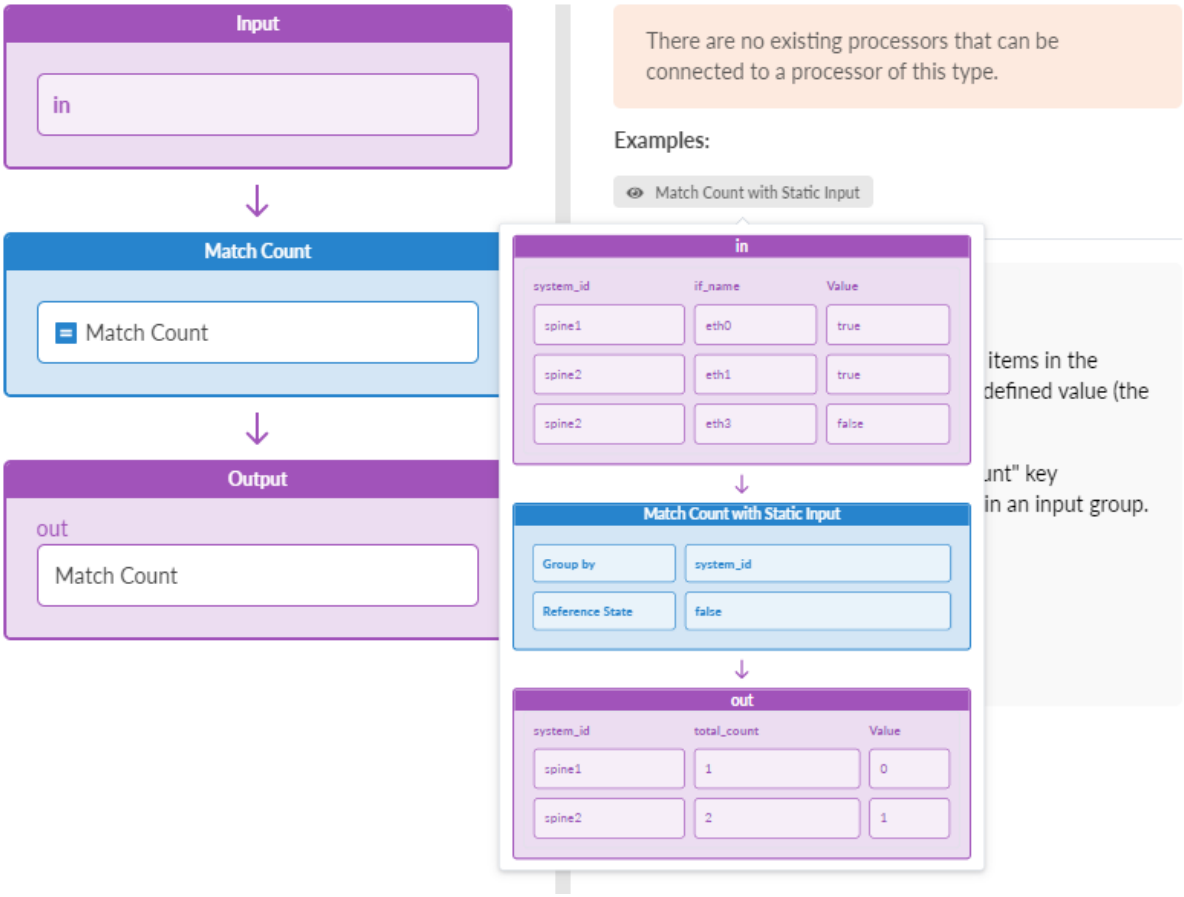
- Example: Match Count | **1658**

For each input group, the Match Count processor creates a single output that is the number of items in the input group that are equal to the reference. The 'total_count' key is added into output item keys where the value is a number of items in an input group.

Parameter	Description
Input Types	Table(text or discrete state)
Output Types	NS
Group by (group_by)	Accepts a list of property names to group input items into output items, produces only one output group for the empty list.Most processors take input and produce output. Many of them produce one output per input (for example, if input is a DSS, output is a DSS of same size). However, some processors reduce the size of the output relative to the size of the input. Effectively, they partition the input into groups, run some calculation on each of the groups that produce a single value per each group, and use that as output. Clearly, the size of the output set depends on the grouping scheme. We call such processors **grouping processors** and they all take the **Group by** configuration parameter. In the case of an empty list, the input is considered to be a single group; thus, the output is of size 1 and either N, DS, or TS. If a list of property names is specified, for example ["system_id", "iface_role"], or a single property is specified, for example ["system_id"], we divide the input into groups such that for each group, every item in the group has the same values for the given list of property names. See the `standard deviation processor<processor_standard_deviation>` example for how this works. The output type of a processor depends on a value of the group_by parameter; for an empty list, a processor produces a single value result, such as N, DS, or T, and for grouping by one or more properties it returns a set result, such as NS, DSS, or TS.
Reference State (reference_state)	DS or TS value which is used as a reference state to match input samples. discrete-state value
Enable Streaming (enable_streaming)	Makes samples of output stages streamed if enabled. An optional boolean that defaults to False. If set to True, all output stages of this processor are streamed in the generic protobuf schema.



In the Add Processor window of the Apstra UI, hover over the "Match Count with Static Input" tooltip for a visual example of how the Match Count processor functions.



Example: Match Count

Assume a configuration of:

```
reference_state: "false"
group_by: []
```

Sample Input:

```
[if_name=eth0] : "true"
[if_name=eth1] : "true"
[if_name=eth3] : "false"
```

Sample Output:

```
[] : 1
```

In the above example, we have 1 as the output because 1 element of the input group matches the reference value of "false".

Processor: Match Percentage

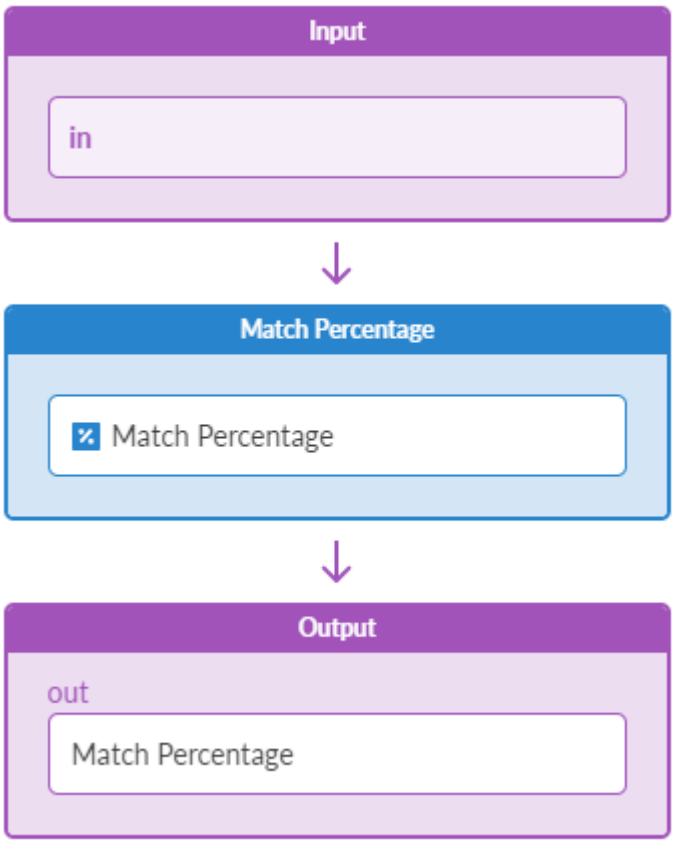
IN THIS SECTION

For each input group, the Match Percentage processor creates a single output that is the percentage of items in the input group that are equal to the reference.

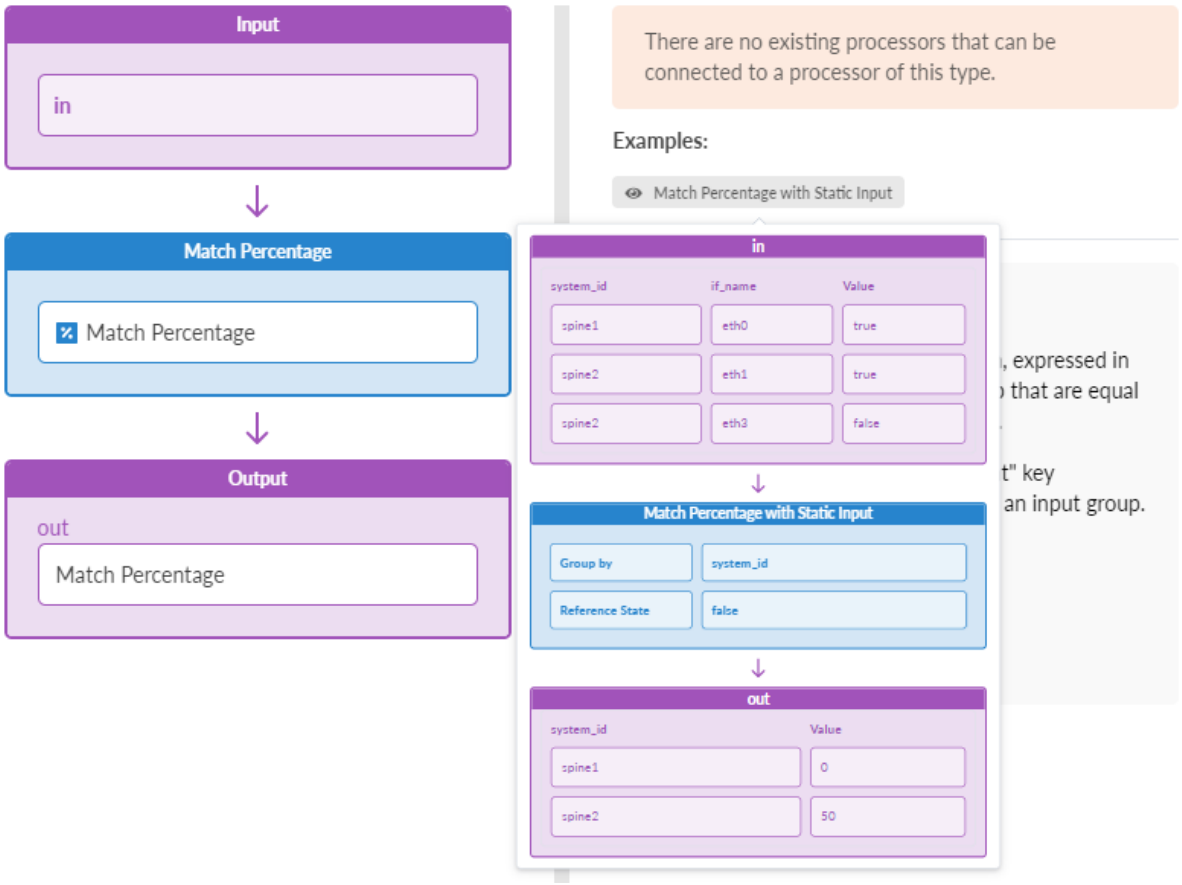
Parameter	Description
Input Types	Table(text or discrete state)
Output Types	Table(number)
Group by (group_by)	Accepts a list of property names to group input items into output items, produces only one output group for the empty list.Most processors take input and produce output. Many of them produce one output per input (for example, if input is a DSS, output is a DSS of same size). However, some processors reduce the size of the output relative to the size of the input. Effectively, they partition the input into groups, run some calculation on each of the groups that produce a single value per each group, and use that as output. Clearly, the size of the output set depends on the grouping scheme. We call such processors **grouping processors** and they all take the **Group by** configuration parameter. In the case of an empty list, the input is considered to be a single group; thus, the output is of size 1 and either N, DS, or TS. If a list of property names is specified, for example ["system_id", "iface_role"], or a single property is specified, for example ["system_id"], we divide the input into groups such that for each group, every item in the group has the same values for the given list of property names. See the `standard deviation processor <processor_standard_deviation>` example for how this works. The output type of a processor depends on a value of the group_by parameter; for an empty list, a processor produces a single value result, such as N, DS, or T, and for grouping by one or more properties it returns a set result, such as NS, DSS, or TS.
Reference State (reference_state)	DS or TS value which is used as a reference state to match input samples.

(Continued)

Parameter	Description
Enable Streaming (enable_streaming)	Makes samples of output stages streamed if enabled. An optional boolean that defaults to False. If set to True, all output stages of this processor are streamed in the generic protobuf schema.



In the Add Processor window of the Apstra UI, hover over the "Match Percentage with Static Input" tooltip for a visual example of how the Match Percentage processor functions.



Example: Match Percentage

Assume a configuration of:

```
reference_state: "false"
group_by: []
```

Sample Input:

```
[if_name=eth0] : "true"
[if_name=eth1] : "true"
[if_name=eth3] : "false"
```

Sample Output:

```
[] : 33
```

In the above example, we have 33% as the output because 33% of the input group match the reference value of "false".

Processor: Match String

IN THIS SECTION

The Match String processor checks that a string matches a regular expression. It accepts text series on input, for each series it configures a check that verifies if the input value matches the configured regular expression. Regular expression syntax is PCRE-compatible. Note that regexp matching is done in a partial mode, so if the full match is needed, regular expression needs to be specified accordingly. The output series contains anomaly values, such as 'false' and 'true'.

Parameter	Description
Input Types	Time-Series (TS), TSTS
Output Types	Table(discrete state)

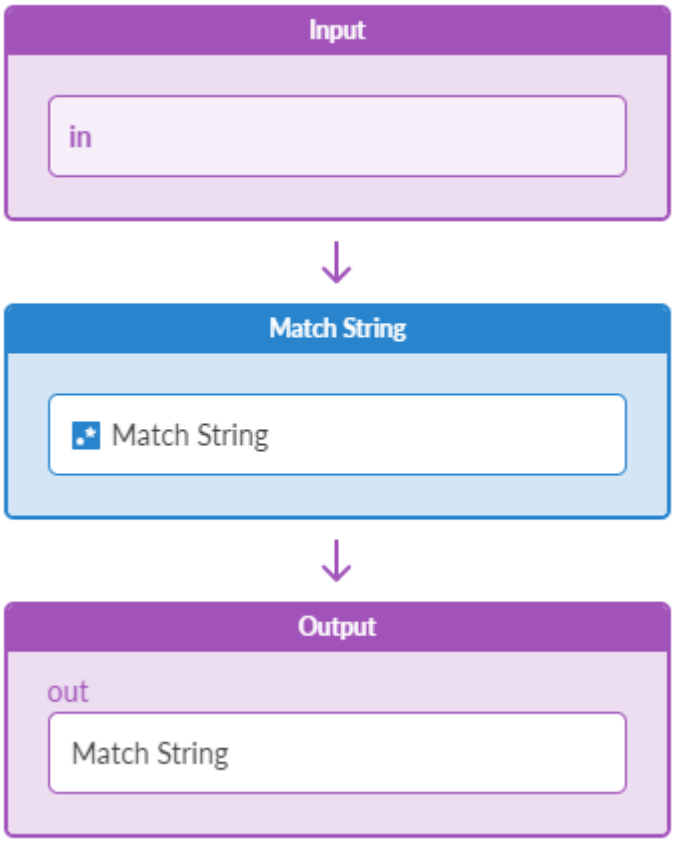
(Continued)

Parameter	Description

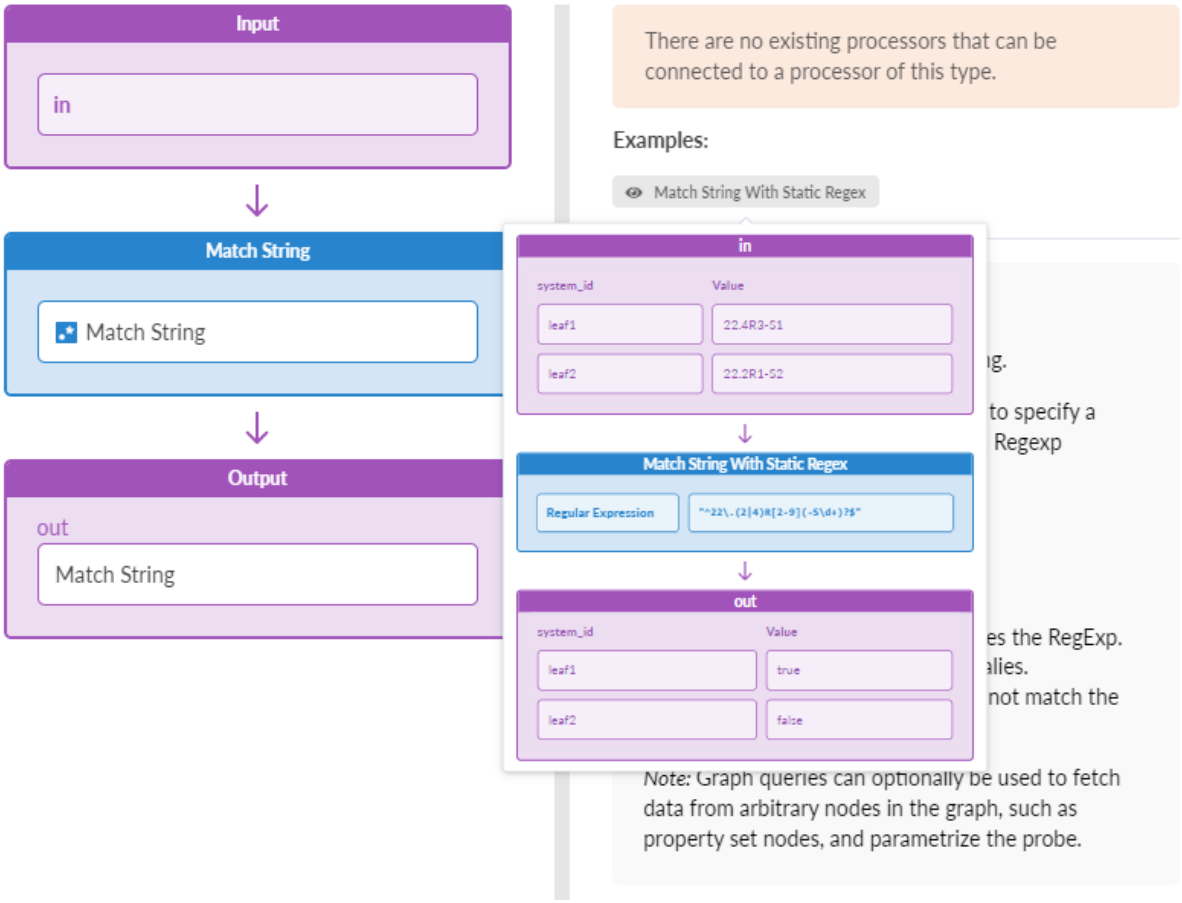
Graph Query (graph_query)	One or more queries on graph specified as strings, or a list of such queries. (String will be deprecated in a future release.) Multiple queries should provide all the named nodes referenced by the expression fields (including additional_properties). Graph query is executed on the "operation" graph. 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"]".
	
	In collector processors (`*_collector`, `if_counter`) it is used to choose a set of nodes for further processing (for example, all leaf devices, or all interfaces between leaf and spine devices)
	
	In other processors it is used for general parameterization and it is only supported as a list of queries.
	
	`graph_query: "node("system", role="leaf", name="system").`
	`out("hosted_interfaces").`
	`node("interface", name="iface").out("link").`
	`node("link", role="spine_leaf")"`
	
	`graph_query: ["node("system", role="leaf", name="system")",`
	`"node("system", role="spine", name="system")"]`
	
	Non-collector processors containing the `graph_query` configuration parameter, can be parameterized to use data from arbitrary nodes in the graph, such as property set nodes. Property sets allow you to parameterize macro level SLAs for individual business units. In the example below, `graph_query` matches a node of type `property_set` with label `probe_propset`. It's accessed using the special `query_result` variable, where Index `0` means it's the first node in query results. If a query returned `N` nodes, they could be accessed using indices starting from `0` to `N-1`. `ps` is what the actual node is referred to in the query; the rest depends on the structure of the node. The `int()` casting is required because values of `property_set` nodes are strings. Here it's assumed that a property set node has the label `probe_propset` and that the value `accumulate_duration` was already created.
	
	`graph_query: [node("property_set", label="probe_propset", name="ps")]`
	`duration: int(query_result[0]["ps"].values["accumulate_duration"])`

(Continued)

Parameter	Description
	Another example is a that probes can validate a compliance requirement; the compliance value may change over time and/or it can be used by more than one probe. Also, a probe can validate NOS versions on devices. In this case, property sets can be used to define the current NOS version requirement. If it changes tomorrow: change the property set value, instead of going under the probe stage.
Regular Expression (regexp)	Expression that evaluates to a PCRE-compatible regular expression.
Anomaly MetricLog Retention Duration	Retain anomaly metric data in MetricDb for specified duration in seconds
Anomaly MetricLog Retention Size	Maximum allowed size, in bytes of anomaly metric data to store in MetricDB
Anomaly Metric Logging	Enable metric logging for anomalies
Enable Streaming (enable_streaming)	Makes samples of output stages streamed if enabled. An optional boolean that defaults to False. If set to True, all output stages of this processor are streamed in the generic protobuf schema.
Raise Anomaly (raise_anomaly)	Outputs “true” and “false” values, “true” meaning an appropriate item is anomalous, and "false" meaning the item is not anomalous. When Raise Anomaly is set to True, an actual anomaly is generated in addition to a sample in the output.



In the Add Processor window of the Apstra UI, hover over the "Match String With Static Regex" tooltip for a visual example of how the Match String processor functions.



Example: Match String

```
regexp: "os_version_pattern"
```

Sample Input (TS)

```
[device=leaf1,os_version_pattern=^4.[7-9].[0-9]+$] : 4.1
[device=leaf2,os_version_pattern=^4.[7-9].[0-9]+$] : 4.7
```

Sample Output (DSS):

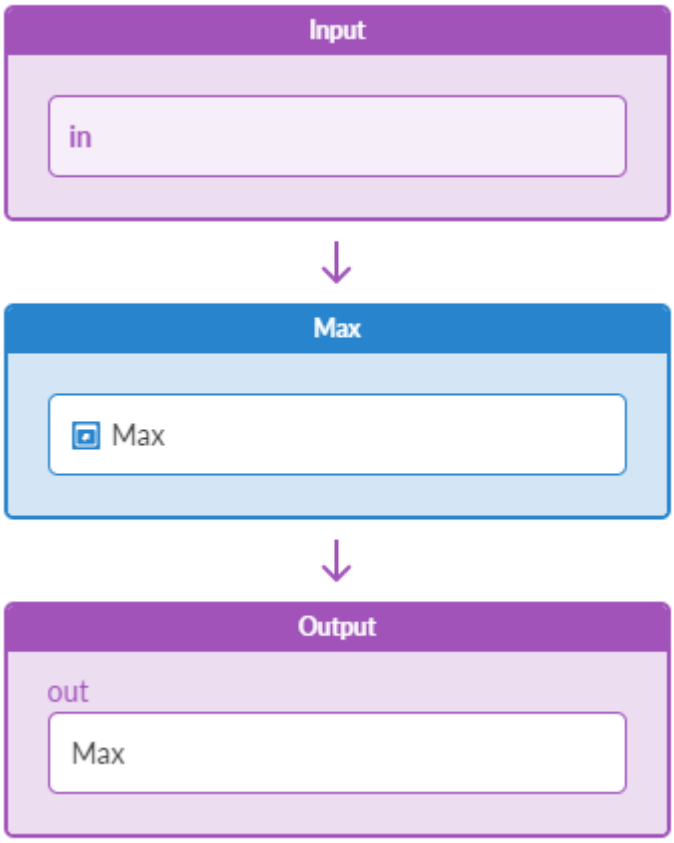
```
[device=leaf1,os_version_pattern=^4.[7-9].[0-9]+$,regex=^4.[7-9].[0-9]+$] : "true"
[device=leaf2,os_version_pattern=^4.[7-9].[0-9]+$,regex=^4.[7-9].[0-9]+$] : "false"
```

Processor: Max

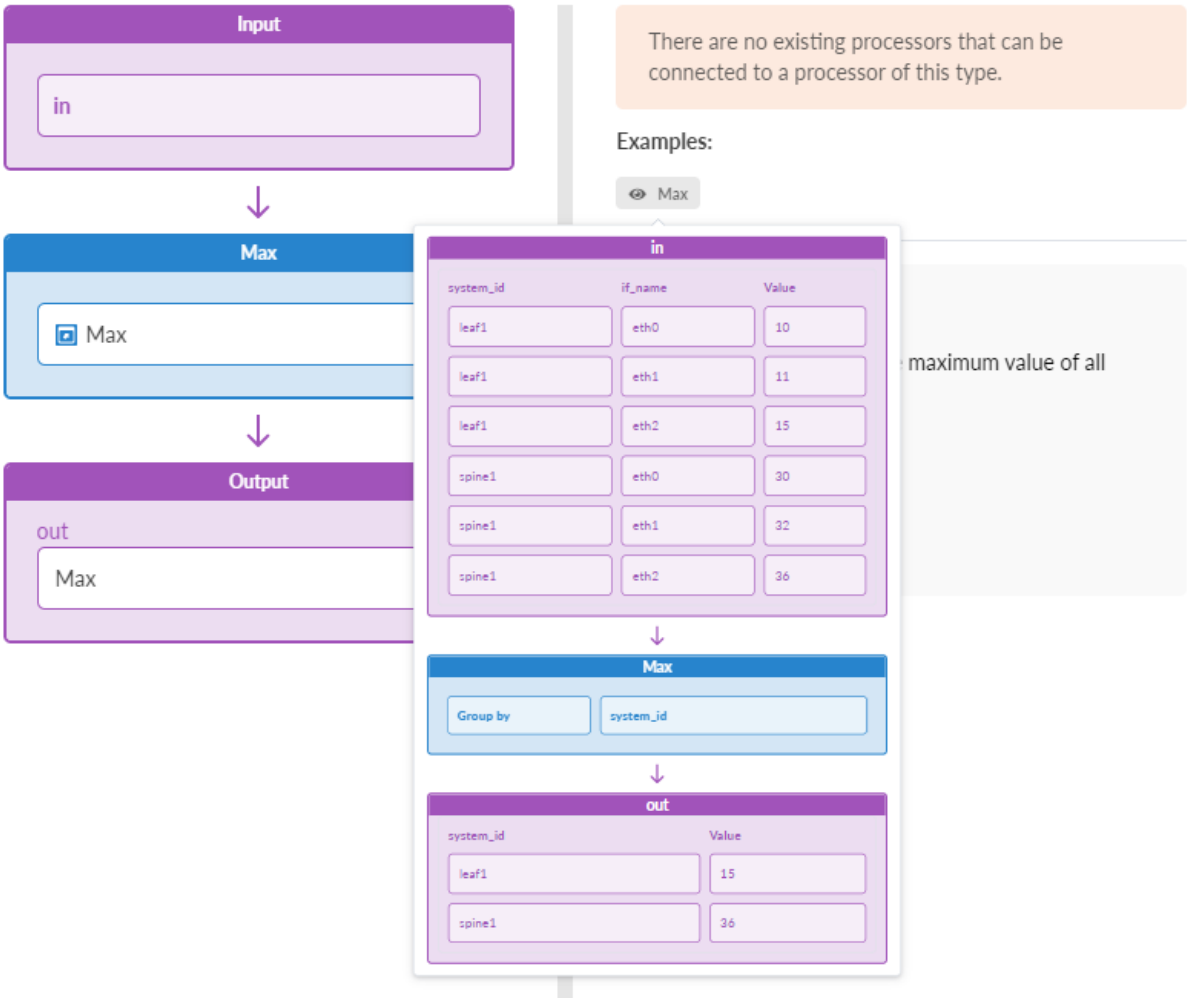
IN THIS SECTION

The Max processor groups as described by **Group by**, then finds the maximum value and outputs it for each group.

Parameter	Description
Input Types	Table (number), Table (number, accumulate=True)
Output Types	Table (number)
Group by (group_by)	Accepts a list of property names to group input items into output items, produces only one output group for the empty list.Most processors take input and produce output. Many of them produce one output per input (for example, if input is a DSS, output is a DSS of same size). However, some processors reduce the size of the output relative to the size of the input. Effectively, they partition the input into groups, run some calculation on each of the groups that produce a single value per each group, and use that as output. Clearly, the size of the output set depends on the grouping scheme. We call such processors **grouping processors** and they all take the **Group by** configuration parameter. In the case of an empty list, the input is considered to be a single group; thus, the output is of size 1 and either N, DS, or TS. If a list of property names is specified, for example ["system_id", "iface_role"], or a single property is specified, for example ["system_id"], we divide the input into groups such that for each group, every item in the group has the same values for the given list of property names. See the "standard deviation processor" on page 1703 example for how this works. The output type of a processor depends on a value of the group_by parameter; for an empty list, a processor produces a single value result, such as N, DS, or T, and for grouping by one or more properties it returns a set result, such as NS, DSS, or TS.
Enable Streaming (enable_streaming)	Makes samples of output stages streamed if enabled. An optional boolean that defaults to False. If set to True, all output stages of this processor are streamed in the generic protobuf schema.



In the Add Processor window of the Apstra UI, hover over the "Max" tooltip for a visual example of how the Max processor functions.



Example: Max

Assume a configuration of:

```
group_by: ["system_id"]
```

Sample Input:

```
[system_id=leaf0,if_name=swp40] : 10
[system_id=leaf0,if_name=swp41] : 11
[system_id=leaf0,if_name=swp42] : 15
[system_id=spine0,if_name=eth15] : 32
[system_id=spine0,if_name=eth16] : 30
[system_id=spine0,if_name=eth17] : 36
```

Output "out":

```
[system_id=leaf0] : 15
[system_id=spine0] : 36
```

Processor: Min

IN THIS SECTION

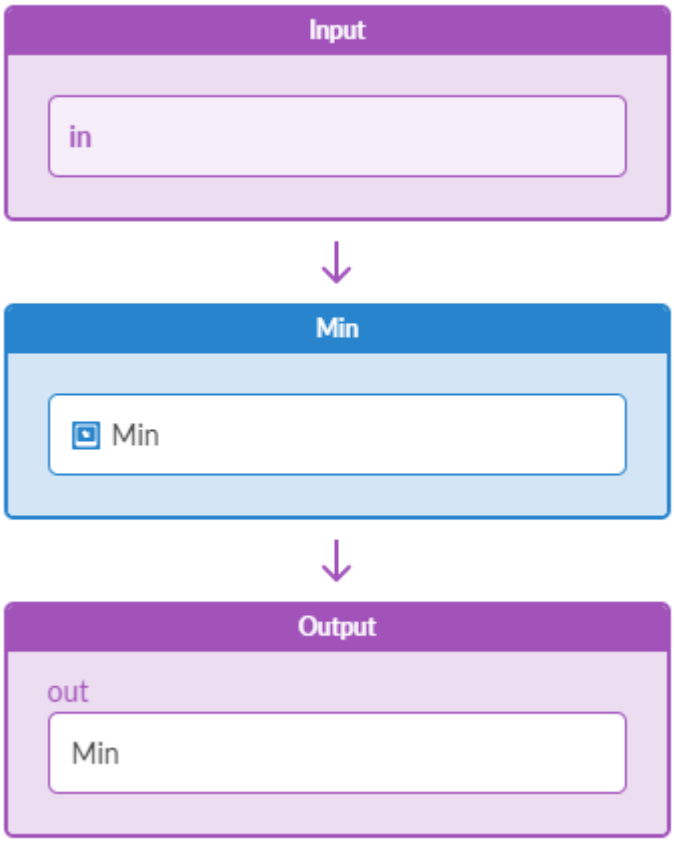
- Example: Min | **1673**

The Min processor groups as described in **Group by**, then finds the minimum value and outputs it for each group.

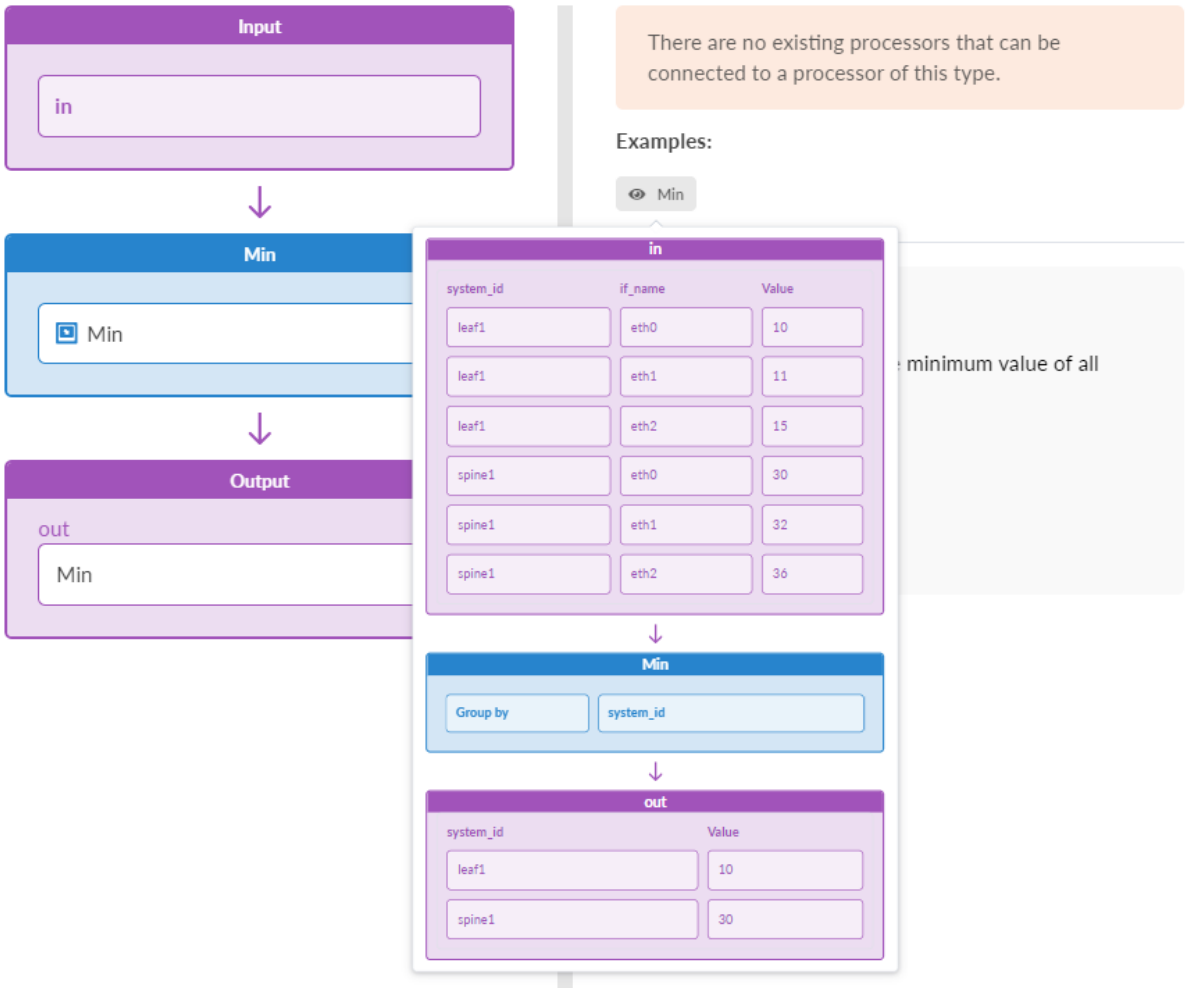
Parameter	Description
Input Types	Table (number), Table (number, accumulate=True)
Output Types	Table (number)

(Continued)

Parameter	Description
Group by (group_by)	Accepts a list of property names to group input items into output items, produces only one output group for the empty list.Most processors take input and produce output. Many of them produce one output per input (for example, if input is a DSS, output is a DSS of same size). However, some processors reduce the size of the output relative to the size of the input. Effectively, they partition the input into groups, run some calculation on each of the groups that produce a single value per each group, and use that as output. Clearly, the size of the output set depends on the grouping scheme. We call such processors **grouping processors** and they all take the **Group by** configuration parameter. In the case of an empty list, the input is considered to be a single group; thus, the output is of size 1 and either N, DS, or TS. If a list of property names is specified, for example ["system_id", "iface_role"], or a single property is specified, for example ["system_id"], we divide the input into groups such that for each group, every item in the group has the same values for the given list of property names. See the "standard deviation processor" on page 1703 example for how this works. The output type of a processor depends on a value of the group_by parameter; for an empty list, a processor produces a single value result, such as N, DS, or T, and for grouping by one or more properties it returns a set result, such as NS, DSS, or TS.
Enable Streaming (enable_streaming)	Makes samples of output stages streamed if enabled. An optional boolean that defaults to False. If set to True, all output stages of this processor are streamed in the generic protobuf schema.



In the Add Processor window in the Apstra UI, hover over the "Min" tooltip for a visual example of how the Min processor functions.



Example: Min

Assume a configuration of:

```
group_by: ["system_id"]
```

Sample Input:

```
[system_id=leaf0,if_name=swp40] : 10
[system_id=leaf0,if_name=swp41] : 11
[system_id=leaf0,if_name=swp42] : 15
[system_id=spine0,if_name=eth15] : 32
[system_id=spine0,if_name=eth16] : 30
[system_id=spine0,if_name=eth17] : 36
```

Output "out":

```
[system_id=leaf0] : 10
[system_id=spine0] : 30
```

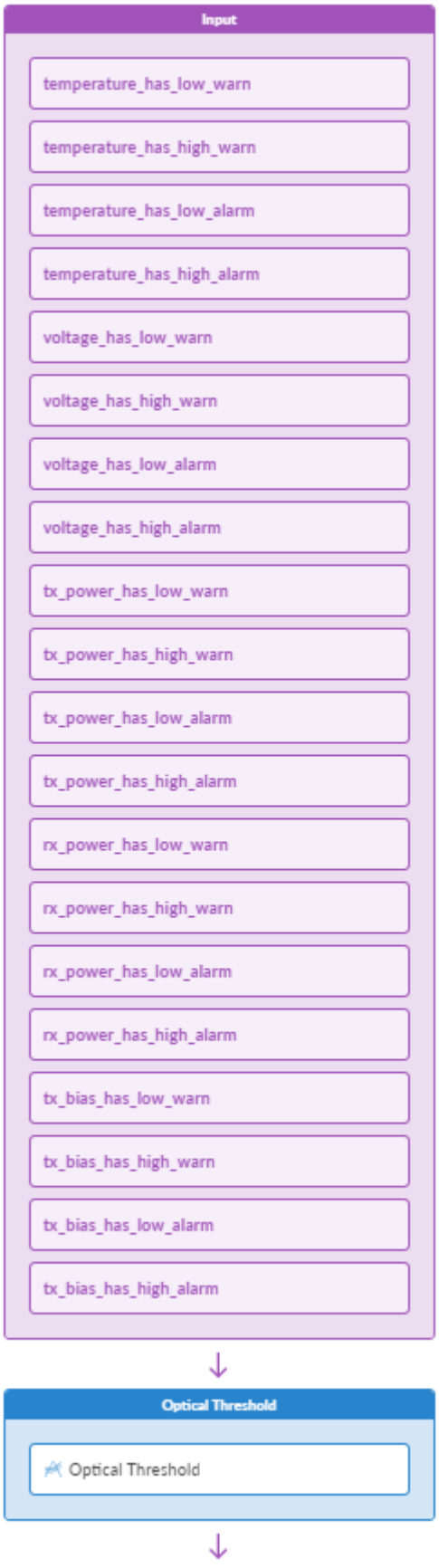
Processor: Optical Threshold

The Optical Threshold processor is responsible for aggregating the data from the "Optical Xcvr" processor. It takes two stages:

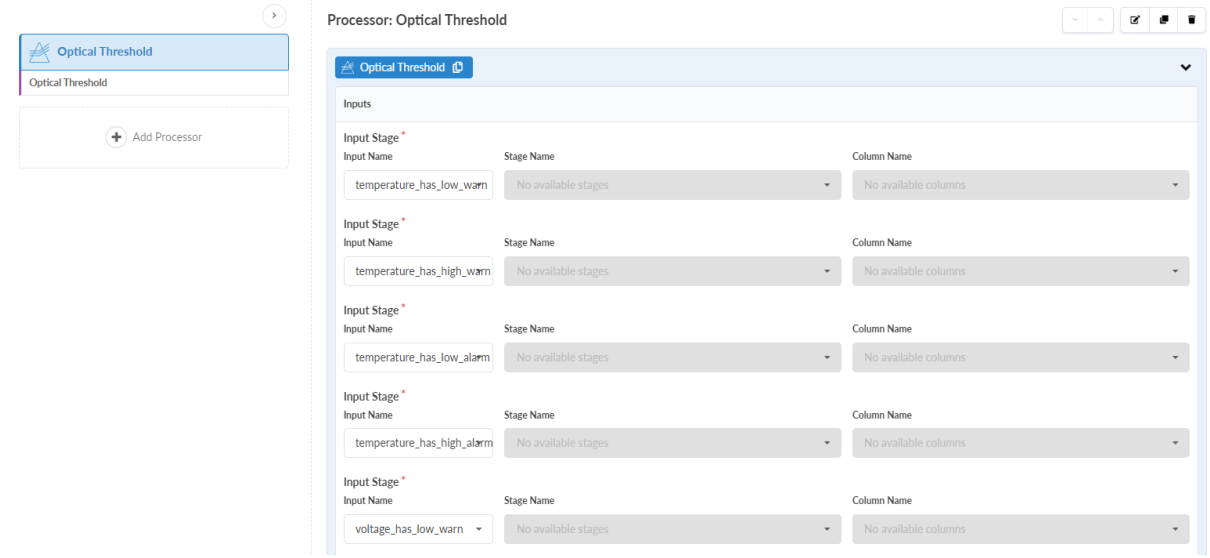
- Interface-related metrics
- Lane-related metrics

Every stage contains certain false/true values like `tx_power_has_low_warn`, `voltage_has_high_alarm`, etc. `true` value means the threshold has been exceeded. `false` - the opposite.

The processor produces for every input interface just one false/true value. It's `true` if there is at least one input value equal to `true` in its input corresponding to this interface (including both interface and lanes).


Input
temperature_has_low_warn
temperature_has_high_warn
temperature_has_low_alarm
temperature_has_high_alarm
voltage_has_low_warn
voltage_has_high_warn
voltage_has_low_alarm
voltage_has_high_alarm
tx_power_has_low_warn
tx_power_has_high_warn
tx_power_has_low_alarm
tx_power_has_high_alarm
rx_power_has_low_warn
rx_power_has_high_warn
rx_power_has_low_alarm
rx_power_has_high_alarm
tx_bias_has_low_warn
tx_bias_has_high_warn
tx_bias_has_low_alarm
tx_bias_has_high_alarm
Optical Threshold
Optical Threshold

Configuration options:



Processor: Optical Xcvr

The Optical Xcvr processor collects metrics and statistics of the optical transceivers. The processor contains two output stages:

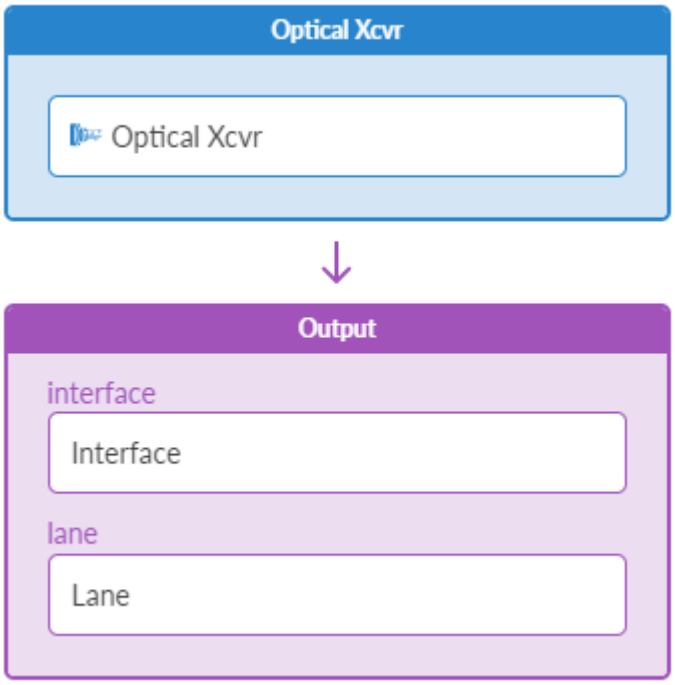
- "Interface Stats": Reports two metrics, Temperature (Celsius) and Voltage (Volt)
- "Lane Stats": Reports three metrics, Tx Power (dBm), Rx Power (dBm), Tx Bias (mA)

For each reported metric, the processor provides the actual measurement as well as the vendor's threshold limits with a True/False flag indicating if any metric has been crossing any of its thresholds. These thresholds are:

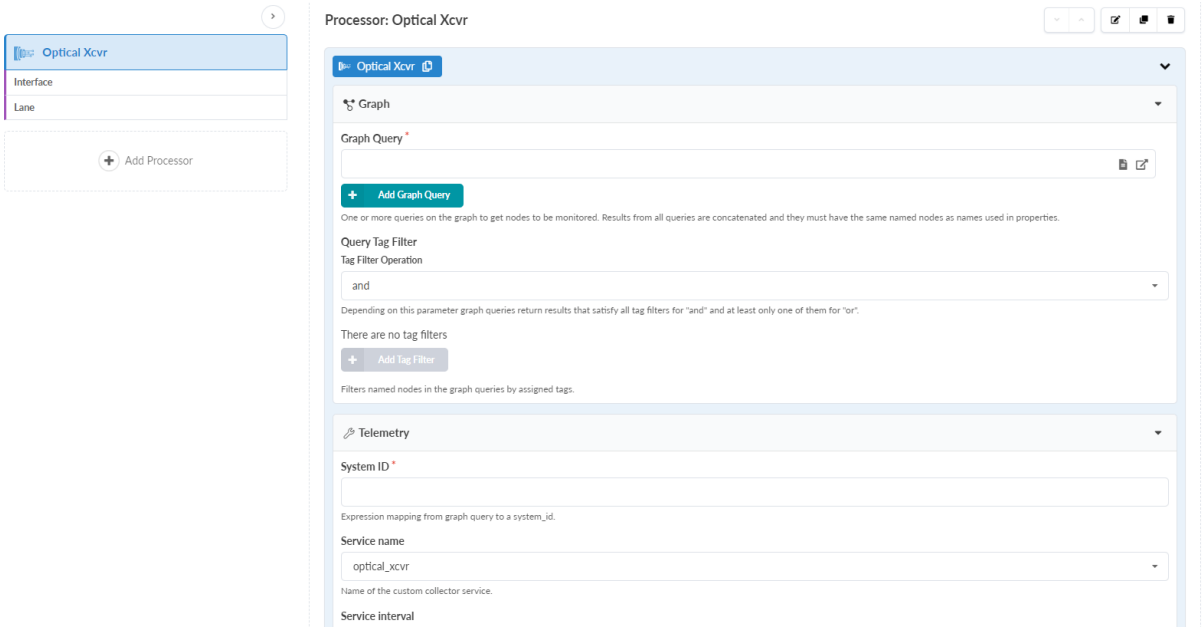
- High Alarm, High Warning, Low Warning and Low Alarm

For every monitored transceiver the processor will provide meta-data information:

- Vendor Name, Vendor Part Number, Vendor Serial Number, Media Type and Fiber Type



Configuration options:



Processor: Periodic Average

IN THIS SECTION

One number is created on output for each input. Each <period>, the output is set to the average of the input over the last <period>. This is not a weighted average.

Graph queries can optionally be used to fetch data from arbitrary nodes in the graph, such as property set nodes, and parametrize the probe.

Parameter	Description
Input Types	Table (number)
Output Types	Table (number)
Period	Size of the averaging period. (time in seconds, integer, or an expression that evaluates to time in seconds integer value)

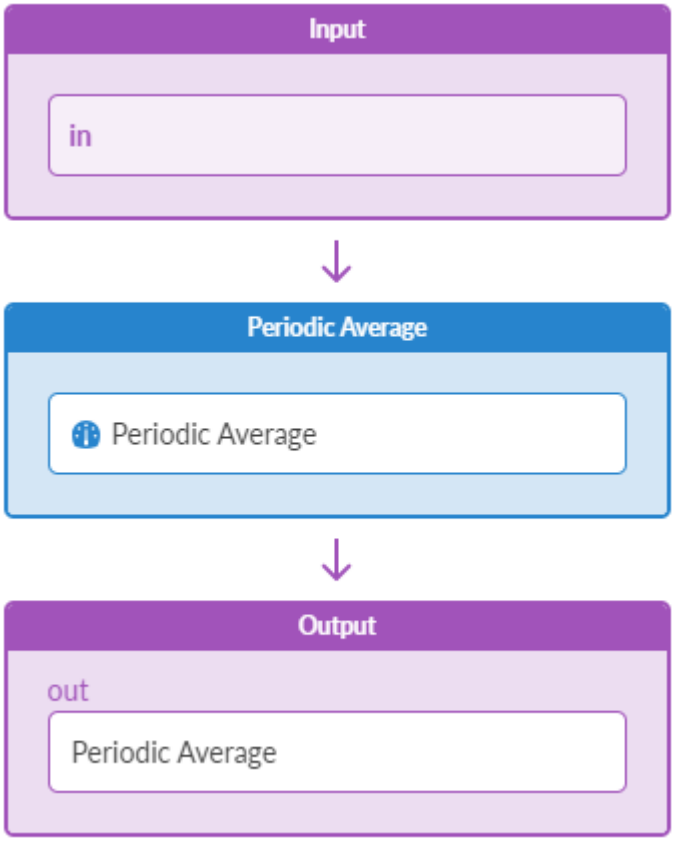
(Continued)

Parameter	Description

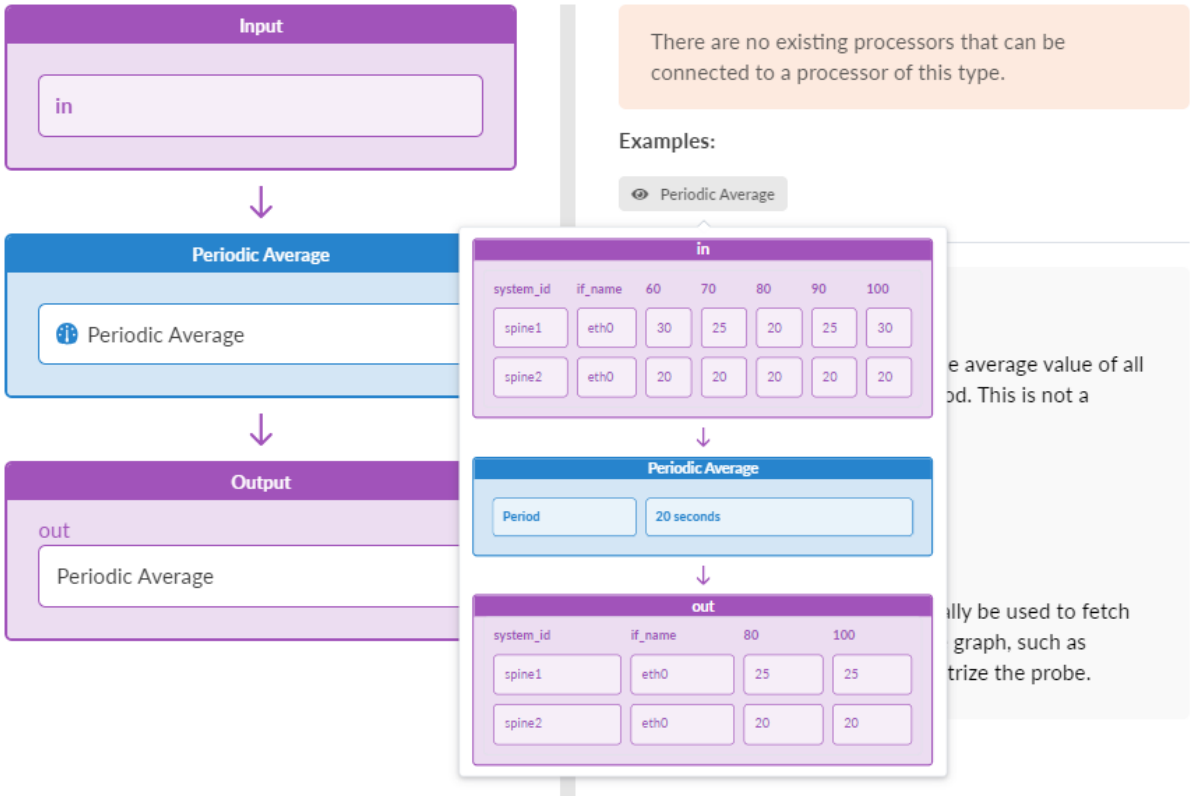
Graph Query (graph_query)	One or more queries on graph specified as strings, or a list of such queries. (String will be deprecated in a future release.) Multiple queries should provide all the named nodes referenced by the expression fields (including additional_properties). Graph query is executed on the "operation" graph. 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"]".
	
	In collector processors (`*_collector`, `if_counter`) it is used to choose a set of nodes for further processing (for example, all leaf devices, or all interfaces between leaf and spine devices)
	
	In other processors it is used for general parameterization and it is only supported as a list of queries.
	
	`graph_query: "node("system", role="leaf", name="system").`
	`out("hosted_interfaces").`
	`node("interface", name="iface").out("link").`
	`node("link", role="spine_leaf")"`
	
	`graph_query: ["node("system", role="leaf", name="system")",`
	`"node("system", role="spine", name="system")"]`
	
	Non-collector processors containing the `graph_query` configuration parameter, can be parameterized to use data from arbitrary nodes in the graph, such as property set nodes. Property sets allow you to parameterize macro level SLAs for individual business units. In the example below, `graph_query` matches a node of type `property_set` with label `probe_propset`. It's accessed using the special `query_result` variable, where Index `0` means it's the first node in query results. If a query returned `N` nodes, they could be accessed using indices starting from `0` to `N-1`. `ps` is what the actual node is referred to in the query; the rest depends on the structure of the node. The `int()` casting is required because values of `property_set` nodes are strings. Here it's assumed that a property set node has the label `probe_propset` and that the value `accumulate_duration` was already created.
	
	`graph_query: [node("property_set", label="probe_propset", name="ps")]`
	`duration: int(query_result[0]["ps"].values["accumulate_duration"])`

(Continued)

Parameter	Description
	Another example is a that probes can validate a compliance requirement; the compliance value may change over time and/or it can be used by more than one probe. Also, a probe can validate NOS versions on devices. In this case, property sets can be used to define the current NOS version requirement. If it changes tomorrow: change the property set value, instead of going under the probe stage.
Enable Streaming (enable_streaming)	Makes samples of output stages streamed if enabled. An optional boolean that defaults to False. If set to True, all output stages of this processor are streamed in the generic protobuf schema.



In the Add Processor window of the Apstra UI, hover over the "Periodic Average" tooltip for a visual example of how the Periodic Average processor functions.



Example: Periodic Average

```
period: 2
```

Assume the following input at time t=1

```
[if_name=eth0] : 10
[if_name=eth1] : 20
[if_name=eth3] : 30
```

And following input at time t=1.5

```
[if_name=eth0] : 20
[if_name=eth1] : 30
[if_name=eth3] : 40
```

And the following at time t=2.1

```
[if_name=eth0] : 40
[if_name=eth1] : 50
[if_name=eth3] : 60
```

We would now have the following output:

```
[if_name=eth0] : 15
[if_name=eth1] : 25
[if_name=eth3] : 35
```

This output is the average over the last discrete period of 2 seconds (time=0 to time=2). Notice that the average is not weighted by time; frequently-occuring closely-spaced samples will bias the average.

The next time the output would be updated would be at time t=4, in which case it would contain the average of the input over the range [t=2, t=4], a period of the configured two seconds.

Processor: Periodic Change

For each input value, outputs the change in absolute value over a defined period. The input values must change monotonically, for example, not decrease over time.

Graph queries can optionally be used to fetch data from arbitrary nodes in the graph, such as property set nodes, and parametrize the probe.

Parameter	Description
Input Types	Table (number)
Output Types	Table (number)
Period	Size of the averaging period. (time in seconds, integer, or an expression that evaluates to time in seconds integer value)

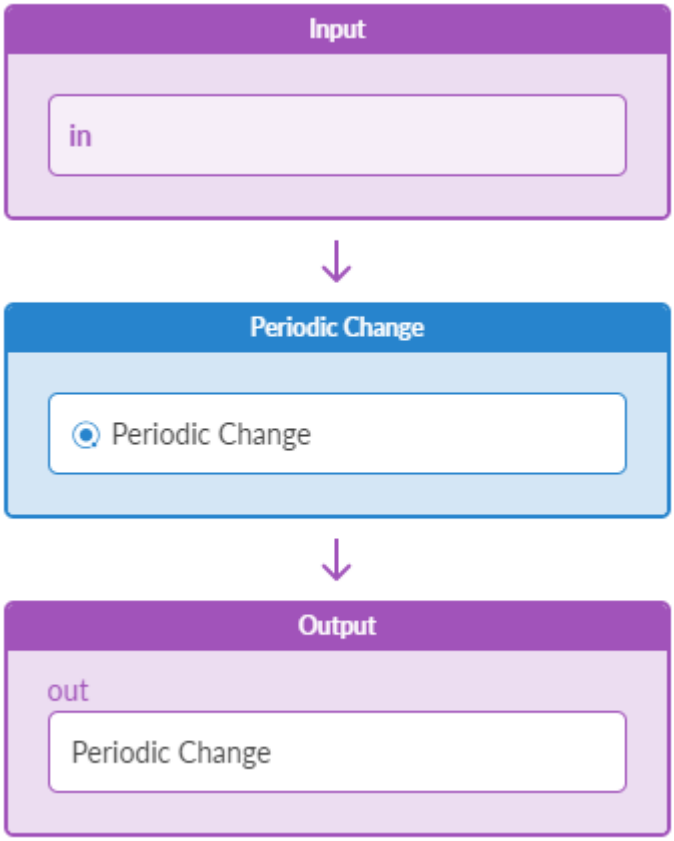
(Continued)

Parameter	Description

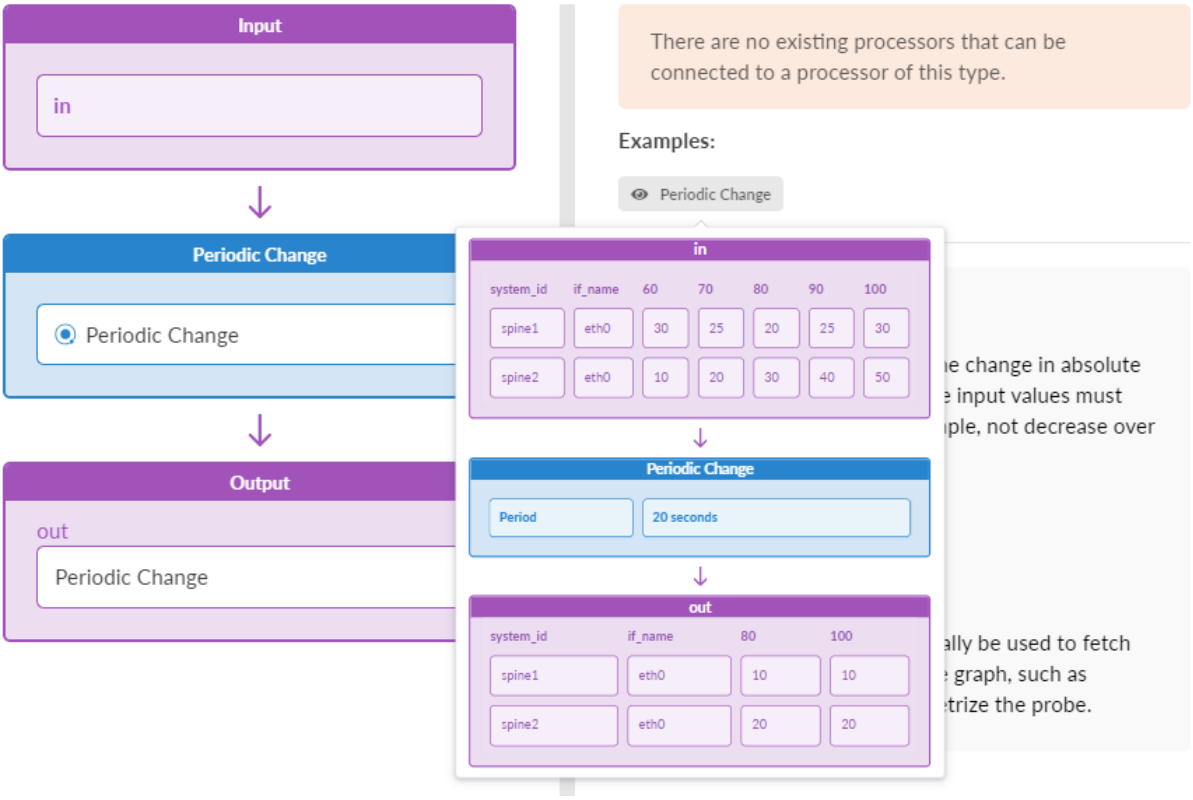
Graph Query (graph_query)	One or more queries on graph specified as strings, or a list of such queries. (String will be deprecated in a future release.) Multiple queries should provide all the named nodes referenced by the expression fields (including additional_properties). Graph query is executed on the "operation" graph. 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"]".
	
	In collector processors (`*_collector`, `if_counter`) it is used to choose a set of nodes for further processing (for example, all leaf devices, or all interfaces between leaf and spine devices)
	
	In other processors it is used for general parameterization and it is only supported as a list of queries.
	
	`graph_query: "node("system", role="leaf", name="system").`
	`out("hosted_interfaces").`
	`node("interface", name="iface").out("link").`
	`node("link", role="spine_leaf")"`
	
	`graph_query: ["node("system", role="leaf", name="system")",`
	`"node("system", role="spine", name="system")"]`
	
	Non-collector processors containing the `graph_query` configuration parameter, can be parameterized to use data from arbitrary nodes in the graph, such as property set nodes. Property sets allow you to parameterize macro level SLAs for individual business units. In the example below, `graph_query` matches a node of type `property_set` with label `probe_propset`. It's accessed using the special `query_result` variable, where Index `0` means it's the first node in query results. If a query returned `N` nodes, they could be accessed using indices starting from `0` to `N-1`. `ps` is what the actual node is referred to in the query; the rest depends on the structure of the node. The `int()` casting is required because values of `property_set` nodes are strings. Here it's assumed that a property set node has the label `probe_propset` and that the value `accumulate_duration` was already created.
	
	`graph_query: [node("property_set", label="probe_propset", name="ps")]`
	`duration: int(query_result[0]["ps"].values["accumulate_duration"])`

(Continued)

Parameter	Description
	Another example is a that probes can validate a compliance requirement; the compliance value may change over time and/or it can be used by more than one probe. Also, a probe can validate NOS versions on devices. In this case, property sets can be used to define the current NOS version requirement. If it changes tomorrow: change the property set value, instead of going under the probe stage.
Enable Streaming (enable_streaming)	Makes samples of output stages streamed if enabled. An optional boolean that defaults to False. If set to True, all output stages of this processor are streamed in the generic protobuf schema.



In the Add Processor window in the Apstra UI, hover over the "Periodic Change" tooltip for a visual example of how the Periodic Change processor functions.



Processor: Range

IN THIS SECTION

- Example: Range | **1690**

The Range processor checks that a value is in a range. According to the specified range, it 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'. (Previously called 'not_in_range' and 'range_check'.) The range processor generates the output of True when the input matches the specified criteria.

Parameter	Description
Input Types	Table (number), Table (number, accumulate=True)

(Continued)

Parameter	Description
Output Types	Table (discrete state)
Property	A property of input items which is used to check against the range. Enum of either value, sample_count, sum, avg
Anomalous Range (range)	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.

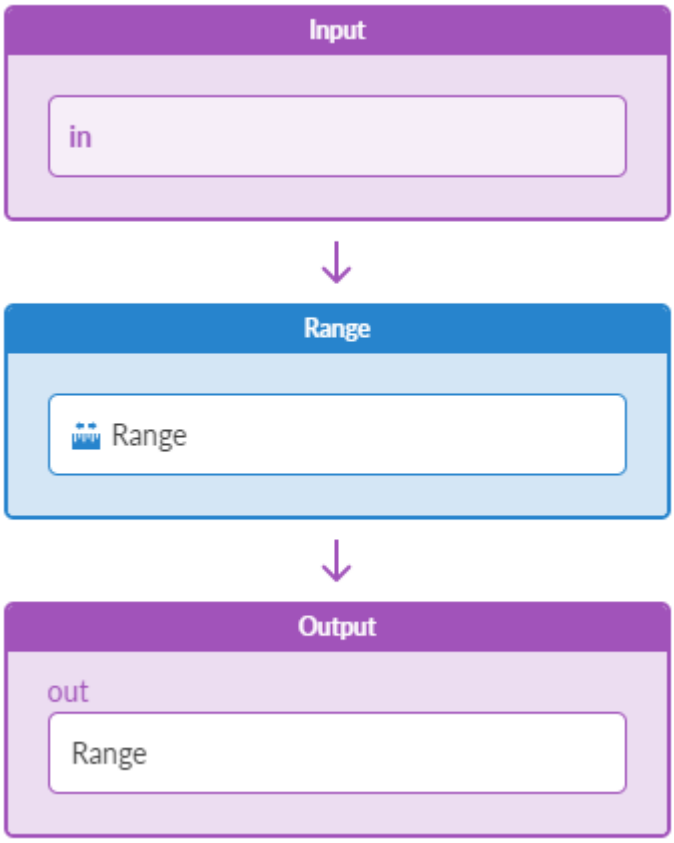
(Continued)

Parameter	Description

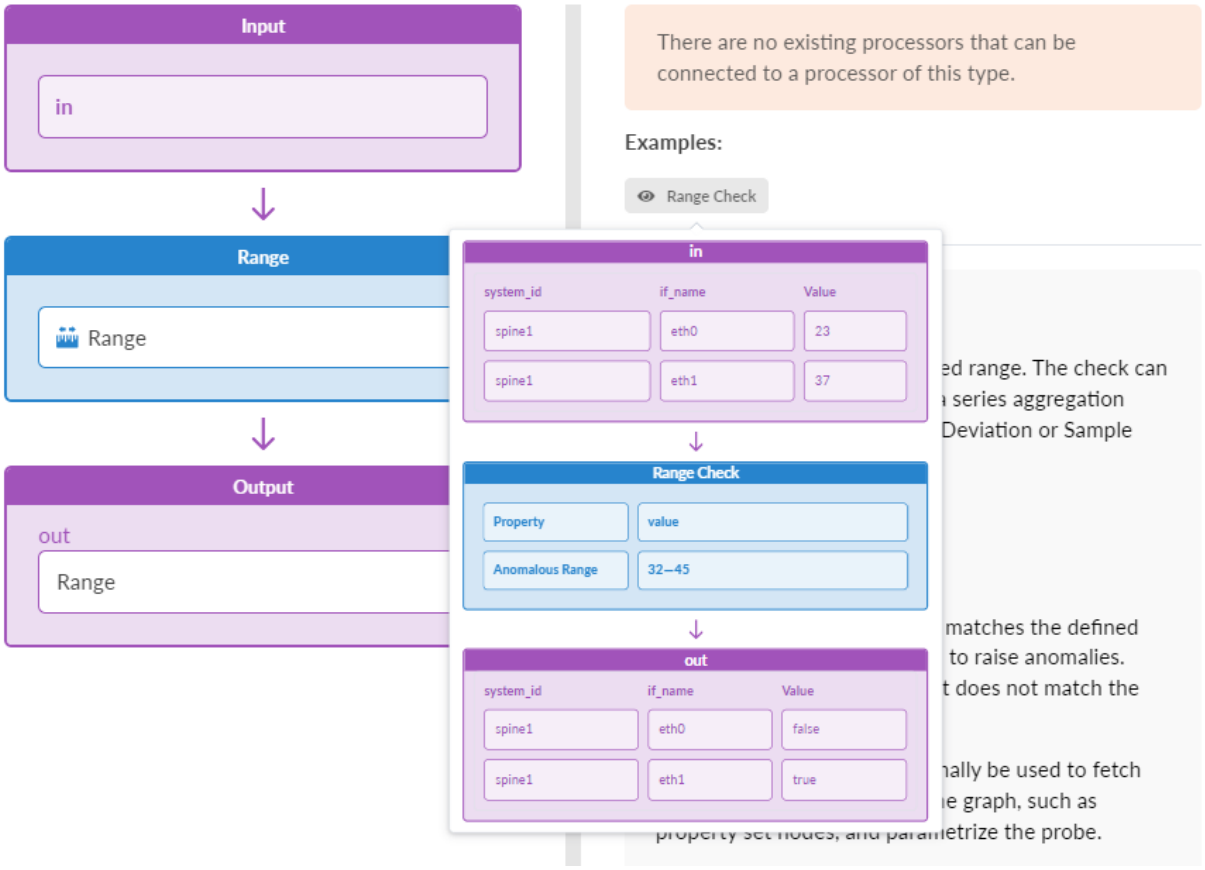
Graph Query (graph_query)	One or more queries on graph specified as strings, or a list of such queries. (String will be deprecated in a future release.) Multiple queries should provide all the named nodes referenced by the expression fields (including additional_properties). Graph query is executed on the "operation" graph. 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"]".
	
	In collector processors (`*_collector`, `if_counter`) it is used to choose a set of nodes for further processing (for example, all leaf devices, or all interfaces between leaf and spine devices)
	
	In other processors it is used for general parameterization and it is only supported as a list of queries.
	
	`graph_query: "node("system", role="leaf", name="system").`
	`out("hosted_interfaces").`
	`node("interface", name="iface").out("link").`
	`node("link", role="spine_leaf")"`
	
	`graph_query: ["node("system", role="leaf", name="system")",`
	`"node("system", role="spine", name="system")"]`
	
	Non-collector processors containing the `graph_query` configuration parameter, can be parameterized to use data from arbitrary nodes in the graph, such as property set nodes. Property sets allow you to parameterize macro level SLAs for individual business units. In the example below, `graph_query` matches a node of type `property_set` with label `probe_propset`. It's accessed using the special `query_result` variable, where Index `0` means it's the first node in query results. If a query returned `N` nodes, they could be accessed using indices starting from `0` to `N-1`. `ps` is what the actual node is referred to in the query; the rest depends on the structure of the node. The `int()` casting is required because values of `property_set` nodes are strings. Here it's assumed that a property set node has the label `probe_propset` and that the value `accumulate_duration` was already created.
	
	`graph_query: [node("property_set", label="probe_propset", name="ps")]`
	`duration: int(query_result[0]["ps"].values["accumulate_duration"])`

(Continued)

Parameter	Description
	Another example is a that probes can validate a compliance requirement; the compliance value may change over time and/or it can be used by more than one probe. Also, a probe can validate NOS versions on devices. In this case, property sets can be used to define the current NOS version requirement. If it changes tomorrow: change the property set value, instead of going under the probe stage.
Anomaly MetricLog Retention Duration	Retain anomaly metric data in MetricDb for specified duration in seconds
Anomaly MetricLog Retention Size	Maximum allowed size, in bytes of anomaly metric data to store in MetricDB
Anomaly Metric Logging	Enable metric logging for anomalies
Enable Streaming (enable_streaming)	Makes samples of output stages streamed if enabled. An optional boolean that defaults to False. If set to True, all output stages of this processor are streamed in the generic protobuf schema.
Raise Anomaly (raise_anomaly)	Outputs “true” and “false” values, “true” meaning an appropriate item is anomalous, and "false" meaning the item is not anomalous. When Raise Anomaly is set to True, an actual anomaly is generated in addition to a sample in the output.



In the Add Processor window in the Apstra UI, hover over the "Range Check" tooltip for a visual example of how the Range processor functions.



Example: Range

```
range: {"min": 35, "max": 45}
property: "value"
```

Sample Input (NS)

```
[if_name=eth0] : 23
[if_name=eth1] : 55
[if_name=eth3] : 37
```

Sample Output (DSS)

```
[if_name=eth0] : "false"
[if_name=eth1] : "false"
[if_name=eth3] : "true"
```

If expressions are used for min or max fields of the range property, then they are evaluated for each input item which results into item-specific thresholds. Properties of the respective output item are extended by range_min or range_max properties with calculated values.

```
range: {"max": "speed * 0.7"}
property: "value"
```

Sample Input (NS)

```
[if_name=eth0,speed=10000000000] : 800000000
[if_name=eth1,speed=1000000000] : 800000000
```

Sample Output (DSS)

```
if_name=eth0,speed=10000000000,range_max=7000000000] : "false"
[if_name=eth1,speed=1000000000,range_max=700000000] : "true"
```

Processor: Ratio

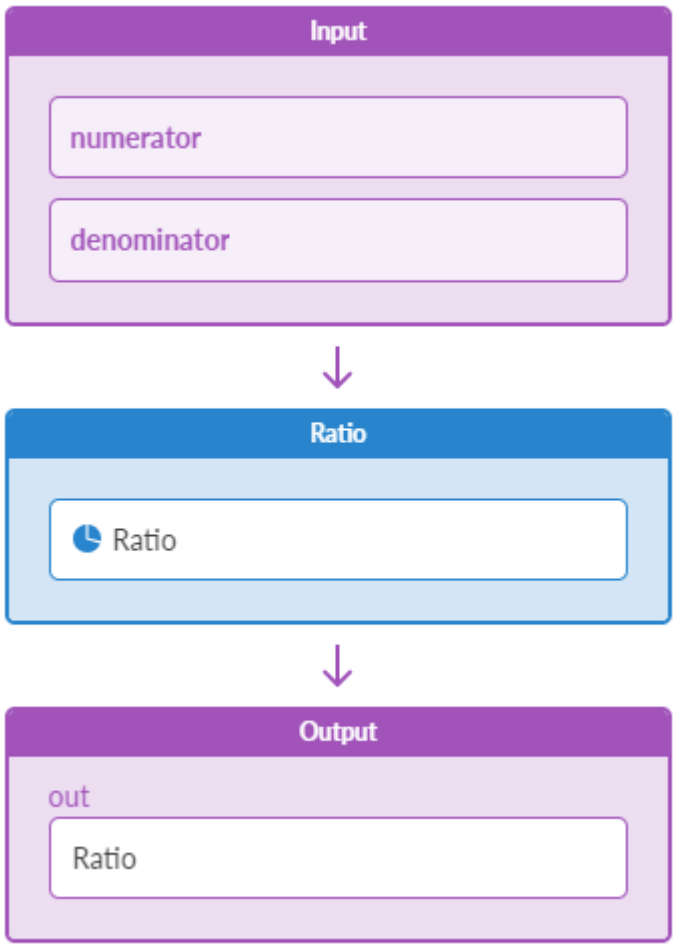
IN THIS SECTION

The Ratio processor calculates the ratio of inputs. It takes two inputs: numerator and denominator. Denominator is optional and could be specified as 'denominator' configuration property instead. It could be either an integer or an expression that evaluates to an integer. It should not be '0'.

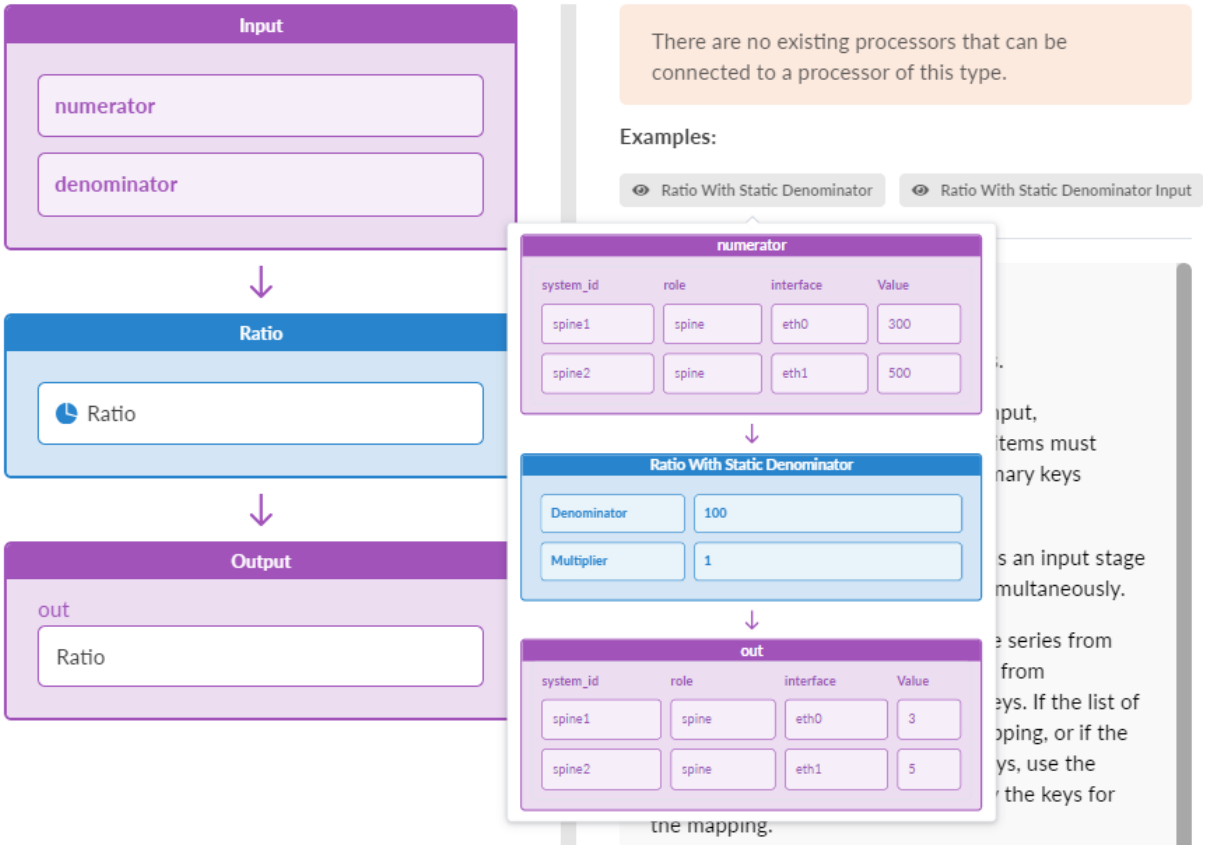
When 'denominator' is specified as an input, 'numerator' and 'denominator' input items must allow only 1:1 mapping. If that is not the case, 'significant_keys' configuration property should be specified to list keys that will allow such mapping.

It also supports 'multiplier' configuration property, which is an integer value greater than one to multiply numerator by before calculating ratio. This allows it to overcome limitations of dealing with integers. Default value is 100.

Parameter	Description
Input Types	Table (number)
Output Types	Table (number)
Denominator	Integer or an expression that evaluates to integer that is used as denominator. Optional denominator value if it's not specified as input; should be non-zero integer or an expression that evaluates to non-zero integer.
Significant Keys (significant_keys)	List of keys to map items from the inputs for applying the specified operation. It is typically used by processors that take multiple inputs and perform operations on them. When inputs have the same sets of keys it does not need to be specified. When inputs have different sets of keys, it must be specified and it must allow only 1:1 items mapping from the given inputs, otherwise the probe will go into error state.
Multiplier	Multiply numerator by a given value before calculating ratio. Optional. Default is 100.
Enable Streaming (enable_streaming)	Makes samples of output stages streamed if enabled. An optional boolean that defaults to False. If set to True, all output stages of this processor are streamed in the generic protobuf schema.



In the Add Processor window of the Apstra UI, hover over the "Ratio With Static Denominator" or "Ratio With Static Denominator Input" tooltips for visual examples of how the Ratio processor functions.



Example: Ratio Output

Simple scenario with a static denominator.

```
denominator: 100
multiplier: 1
```

Input 'numerator':

```
[system_id=spine1,role=spine,interface=eth0]: 300
[system_id=spine2,role=spine,interface=eth1]: 500
```

Output:

```
[system_id=spine1,role=spine,interface=eth0]: 3
[system_id=spine1,role=spine,interface=eth1]: 5
```

Configuration where numerator and denominator are coming from inputs, and 'multiplier' value is the default 100:

```
significant_keys: ['system_id', 'interface']
```

Input 'numerator':

```
[system_id=spine1,role=spine,interface=eth0]: 300
[system_id=spine2,role=spine,interface=eth1]: 750
```

Input 'denominator':

```
[system_id=spine1,role=spine,interface=eth0]: 150
[system_id=spine2,role=spine,interface=eth1]: 250
```

Output:

```
[system_id=spine1,interface=eth0]: 200
[system_id=spine1,interface=eth1]: 300
```

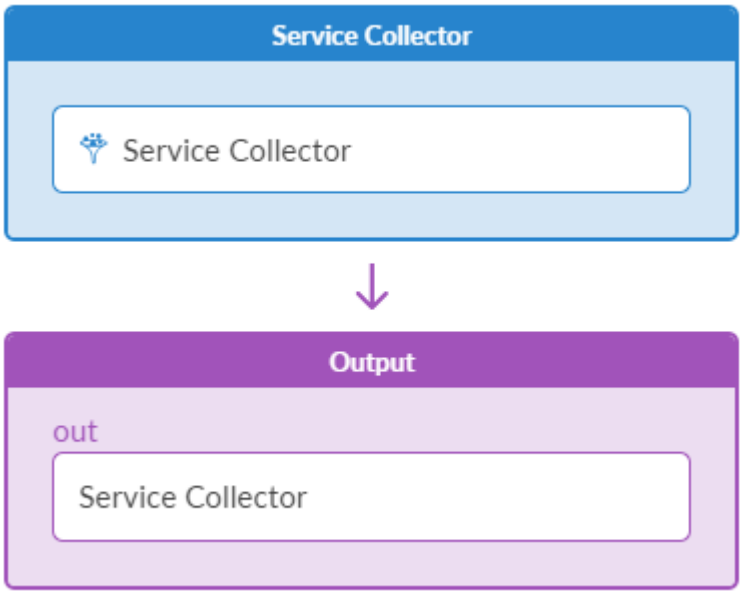
Processor: Service Collector

The Service Collector processor ingests telemetry data from the following built-in services. Each service requires the following keys to be defined in the following order:

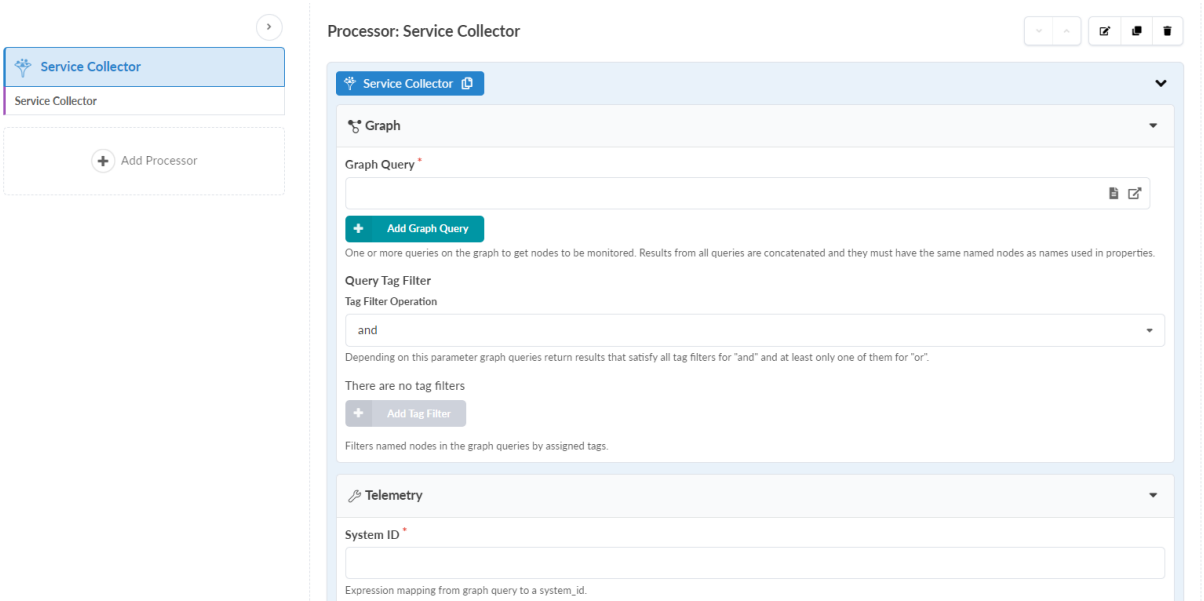
- bgp
  - `source_ip` - String
  - `source_asn` - String
  - `dest_ip` - String
  - `dest_asn` - String
  - `vrf_name` - String
  - `address_family` - String with one of passible values: "ipv4", "ipv6", or "evpn"
- lldp
  - `interface` - String

- hostname
  - `hostname_type` - String with one of possible values: "hostname", "domain", or "fqdn"
- interface
  - `interface` - String

Supports static series only (graph-driven telemetry collection).



Configuration options:



Processor: Set Comparison

IN THIS SECTION

The Set Comparison processor does a set-comparison of input stages.

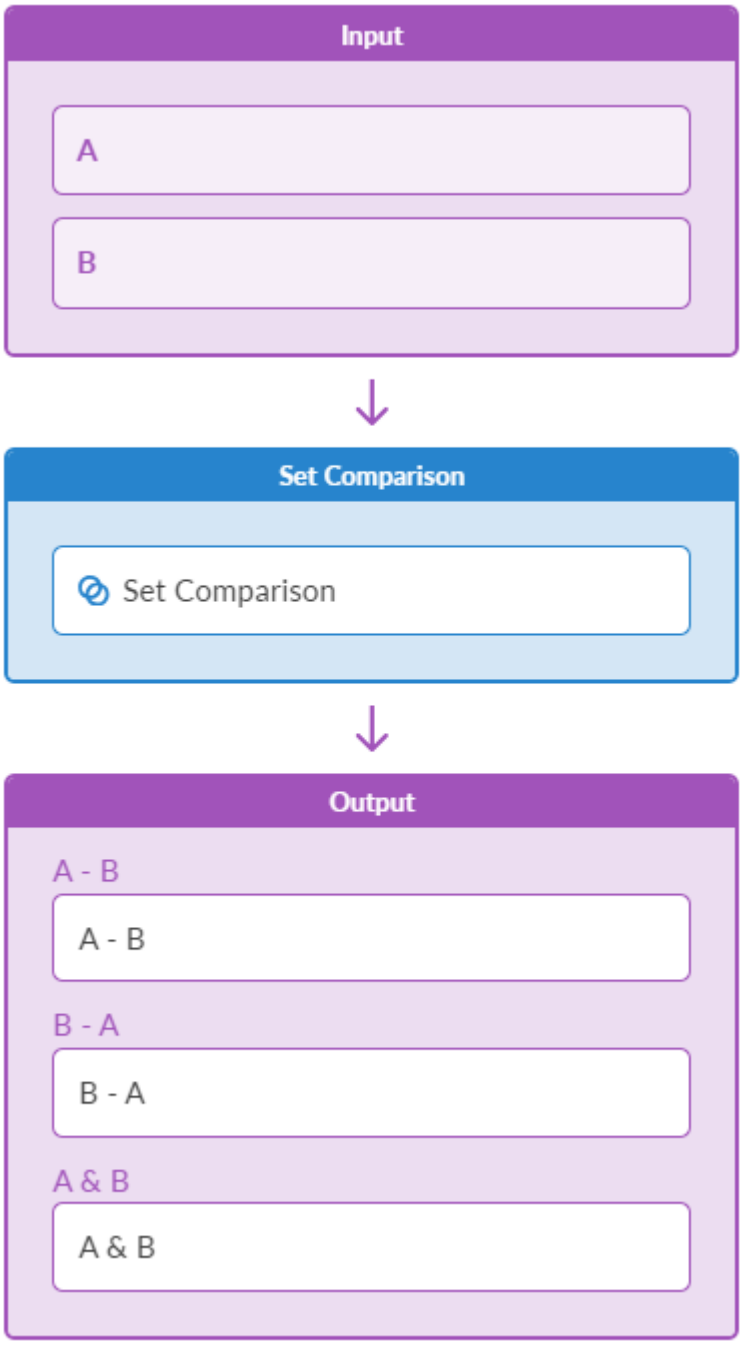
Accept two DS or NS inputs, called "A" and "B". There are three outputs: A stage "A - B" that contains the items that are only in stage "A," a stage "B - A" that contains the items that are only in stage "B," and a stage "A & B" that contains the items that are in both stage "A" and stage "B."

When conducting the above operations, we first normalize all items in each stage by dropping all the keys that are not in "significant_keys." It is an error if a key in "significant_keys" is not present in either stage "A" or "B."

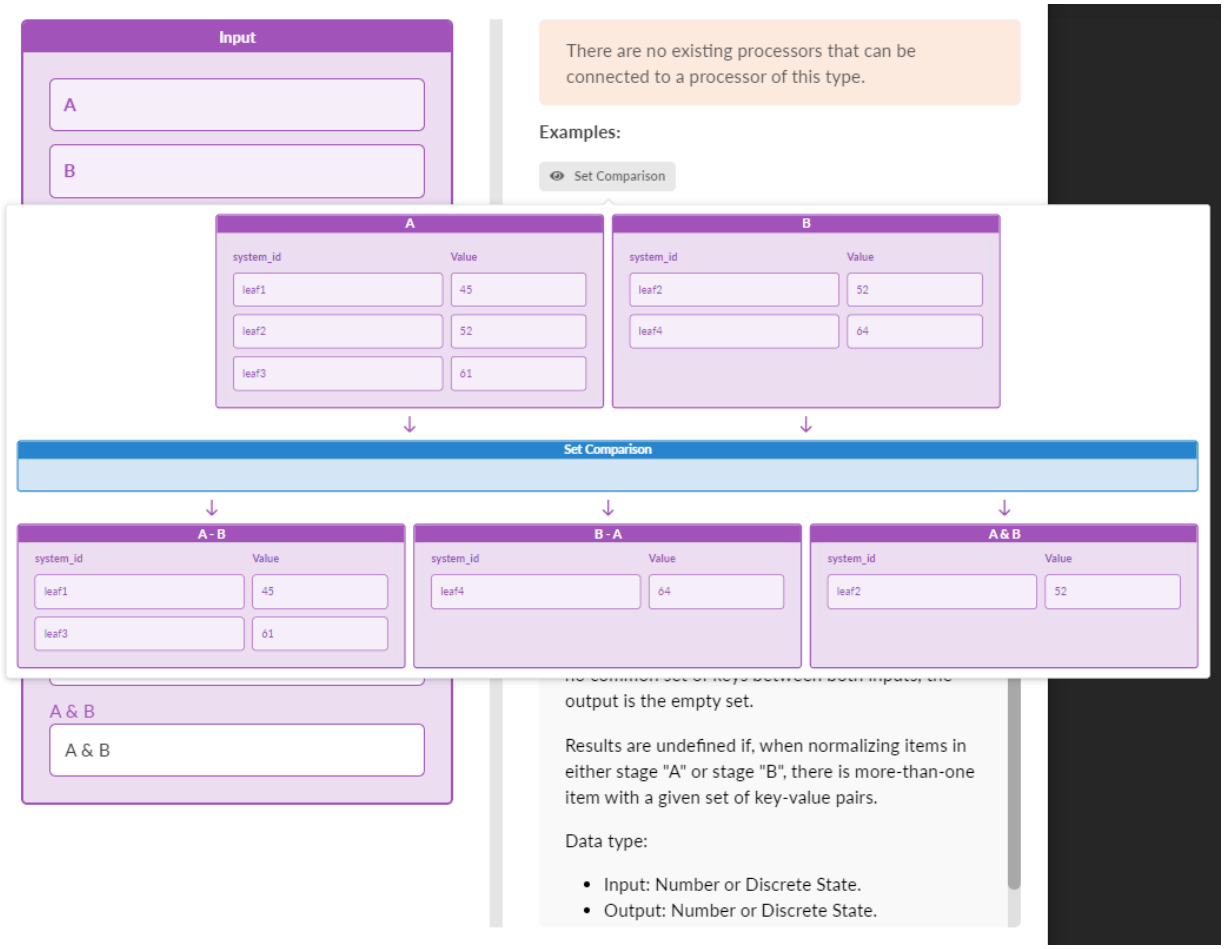
Furthermore, only the keys of each normalized item are considered; values are preserved (and kept from stage "A" in the intersection output), but not considered in the comparison operations.

Results are undefined if, when normalizing items in either stage_A or stage_B, there is more-than-one item with a given set of key-value pairs.

Parameter	Description
Input Types	Table (number or discrete state)
Significant Keys (significant_keys)	List of keys to map items from the inputs for applying the specified operation. It is typically used by processors that take multiple inputs and perform operations on them. When inputs have the same sets of keys it does not need to be specified. When inputs have different sets of keys, it must be specified and it must allow only 1:1 items mapping from the given inputs, otherwise the probe will go into error state.
Enable Streaming (enable_streaming)	Makes samples of output stages streamed if enabled. An optional boolean that defaults to False. If set to True, all output stages of this processor are streamed in the generic protobuf schema.



In the Add Processor window of the Apstra UI, hover over the "Set Comparison" tooltip for a visual example of how the Set Comparison processor functions.



Example: Set Comparison

Consider we have inputs with device temperature information.

Input A:

```
[system_id=leaf1]: 45
[system_id=leaf2]: 52
[system_id=leaf3]: 61
```

Input B:

```
[system_id=leaf2]: 52
[system_id=leaf4]: 64
```

Outputs will be the following.

A - B:

```
[system_id=leaf1]: 45
[system_id=leaf3]: 61
```

B - A:

```
[system_id=leaf4]: 64
```

A & B:

```
[system_id=leaf2]: 52
```

Processor: Set Count

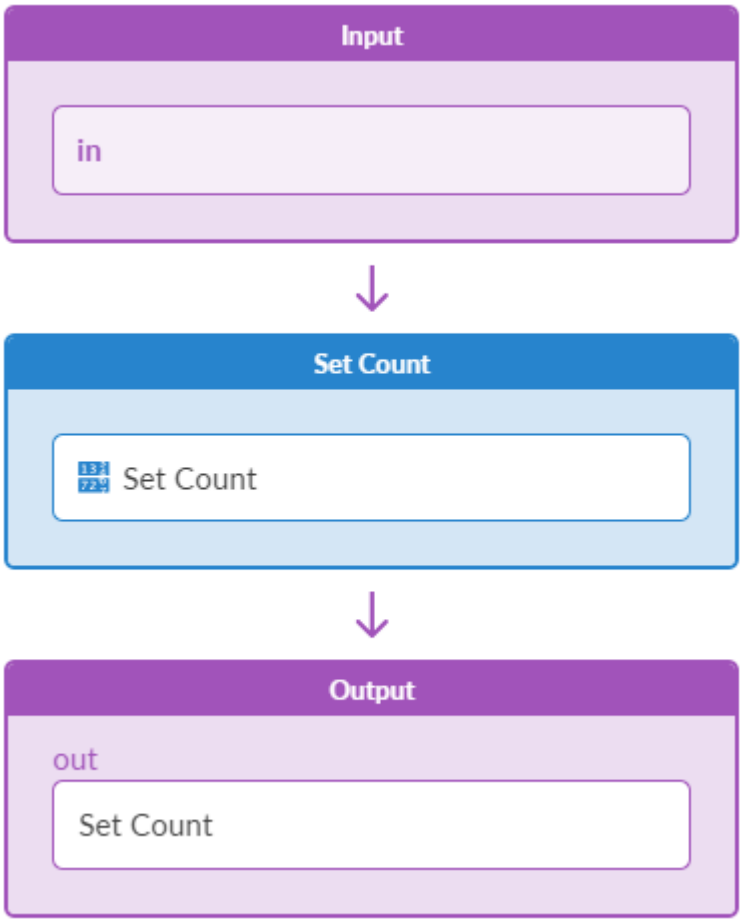
IN THIS SECTION

The Set Count processor groups as described in **Group by**, then calculates the number of items in each group.

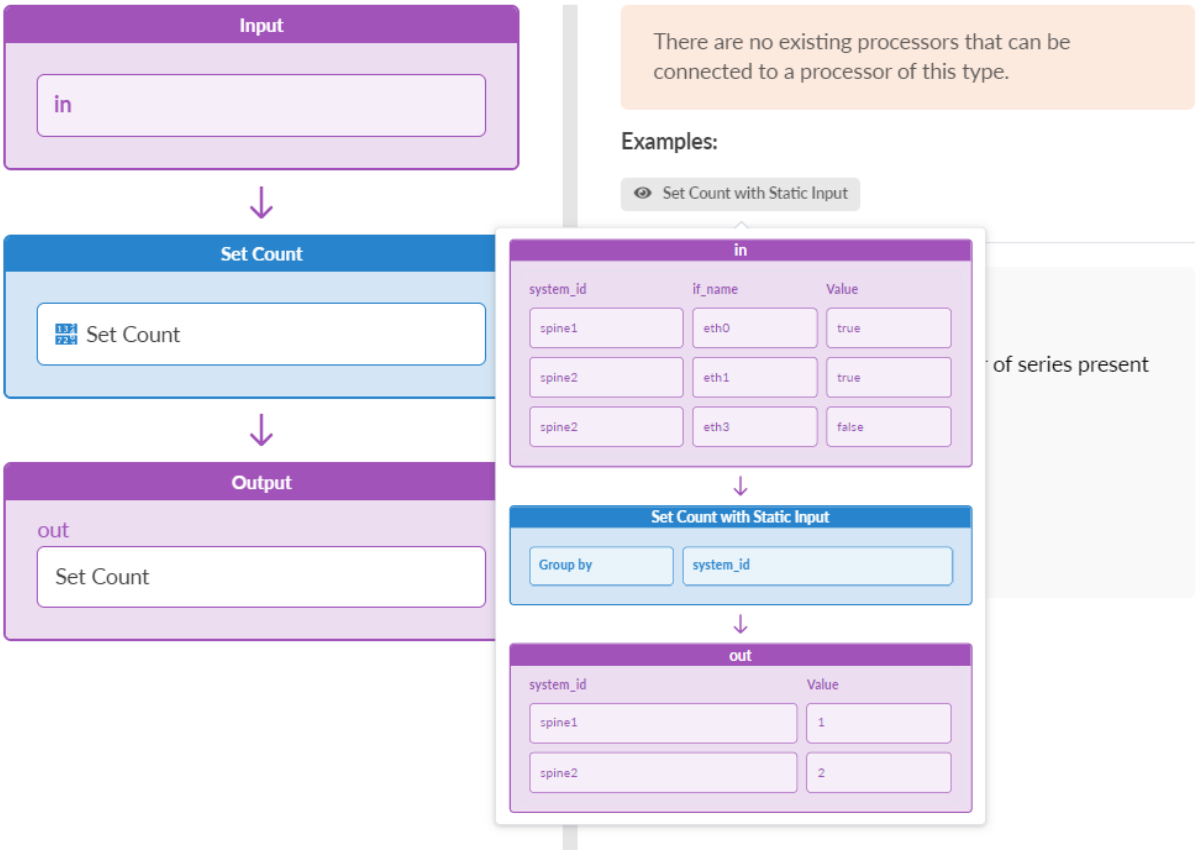
Parameter	Descripton
Input Types	Table (number or text or discrete state)
Output Types	Table (number)

(Continued)

Parameter	Descripton
Group by (group_by)	Accepts a list of property names to group input items into output items, produces only one output group for the empty list.Most processors take input and produce output. Many of them produce one output per input (for example, if input is a DSS, output is a DSS of same size). However, some processors reduce the size of the output relative to the size of the input. Effectively, they partition the input into groups, run some calculation on each of the groups that produce a single value per each group, and use that as output. Clearly, the size of the output set depends on the grouping scheme. We call such processors **grouping processors** and they all take the **Group by** configuration parameter. In the case of an empty list, the input is considered to be a single group; thus, the output is of size 1 and either N, DS, or TS. If a list of property names is specified, for example ["system_id", "iface_role"], or a single property is specified, for example ["system_id"], we divide the input into groups such that for each group, every item in the group has the same values for the given list of property names. See the "standard deviation processor" on page 1703 example for how this works. The output type of a processor depends on a value of the group_by parameter; for an empty list, a processor produces a single value result, such as N, DS, or T, and for grouping by one or more properties it returns a set result, such as NS, DSS, or TS.
Enable Streaming (enable_streaming)	Makes samples of output stages streamed if enabled. An optional boolean that defaults to False. If set to True, all output stages of this processor are streamed in the generic protobuf schema.



In the Add Processor window of the Apstra UI, hover over the "Set Count With Static Input" tooltip for a visual example of how the Set Count processor functions.



Example: Set Count

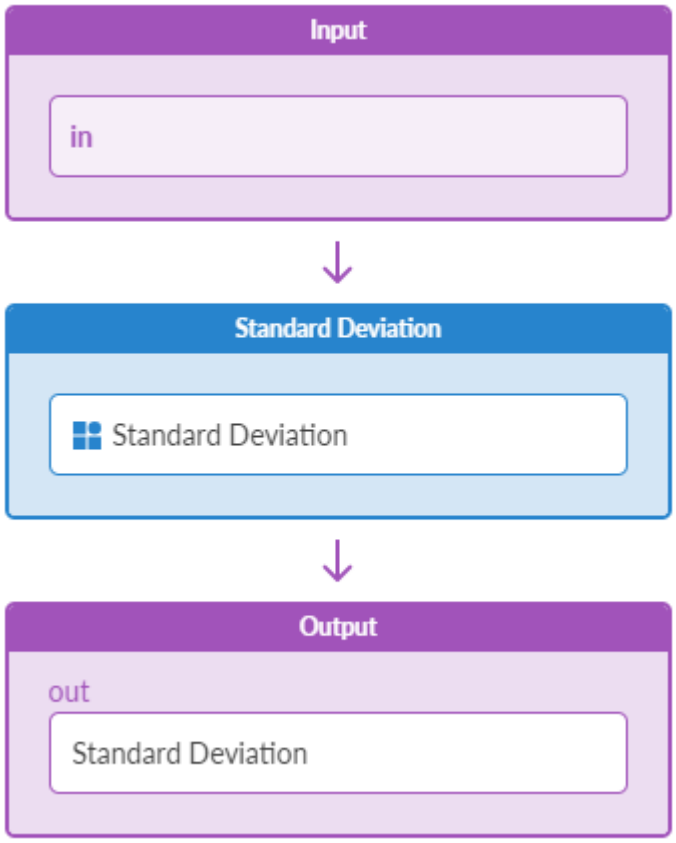
See "standard deviation" on page 1703 example. It's the same except we calculate the number of stage items.

Processor: Standard Deviation

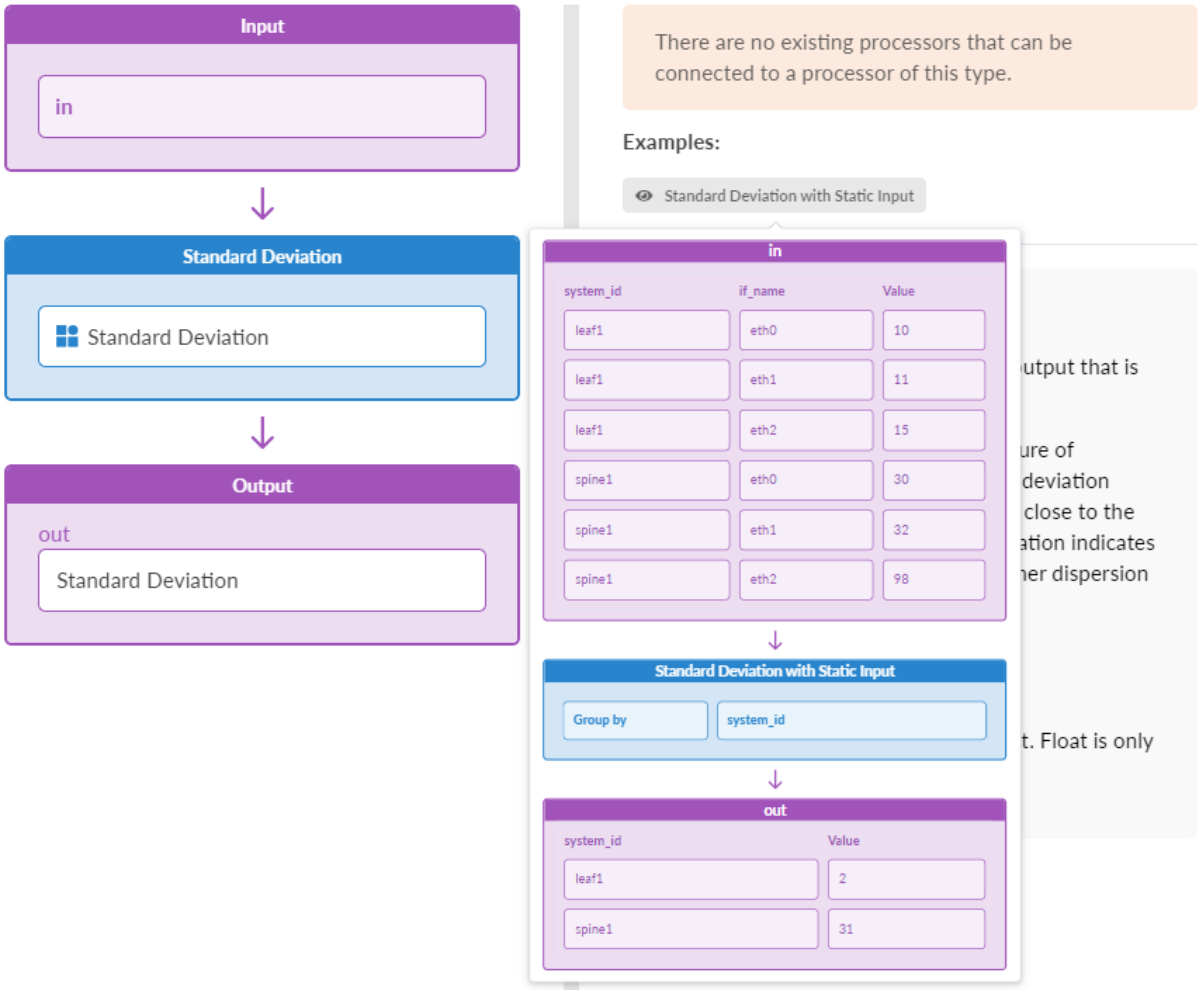
IN THIS SECTION

The Standard Deviation processor groups as described by **Group by**, calculates the standard deviation, then outputs one standard deviation per group.

Parameter	Description
Input Types	Table (number), Table (number, accumluate=True)
Output Types	Table (number)
Group by (group_by)	Accepts a list of property names to group input items into output items, produces only one output group for the empty list.Most processors take input and produce output. Many of them produce one output per input (for example, if input is a DSS, output is a DSS of same size). However, some processors reduce the size of the output relative to the size of the input. Effectively, they partition the input into groups, run some calculation on each of the groups that produce a single value per each group, and use that as output. Clearly, the size of the output set depends on the grouping scheme. We call such processors **grouping processors** and they all take the **Group by** configuration parameter. In the case of an empty list, the input is considered to be a single group; thus, the output is of size 1 and either N, DS, or TS. If a list of property names is specified, for example ["system_id", "iface_role"], or a single property is specified, for example ["system_id"], we divide the input into groups such that for each group, every item in the group has the same values for the given list of property names. See the "standard deviation processor" on page 1703 example for how this works. The output type of a processor depends on a value of the group_by parameter; for an empty list, a processor produces a single value result, such as N, DS, or T, and for grouping by one or more properties it returns a set result, such as NS, DSS, or TS.
DDoF (ddof)	Delta Degrees of Freedom, standard deviation correction value, is used to correct divisor (N - DDoF) in calculations, e.g. DDoF=0 - uncorrected sample standard deviation, DDoF=1 - corrected sample standard deviation.
Enable Streaming (enable_streaming)	Makes samples of output stages streamed if enabled. An optional boolean that defaults to False. If set to True, all output stages of this processor are streamed in the generic protobuf schema.



In the Add Processor window of the Apstra UI, hover over the "Standard Deviation with Static Input" tooltip for a visual example of how the Standard Deviation processor functions.



Example: Standard Deviation

```
group_by: ["role", "system_id"]
ddof: 1
```

Also assume an NS input of

```
[role:fabric, system_id:spine1, if_name=eth0] :10
[role:fabric, system_id:spine1, if_name=eth1] :11
[role:server, system_id:spine1, if_name=eth3] :12
[role:server, system_id:spine1, if_name=eth4] :13
[role:fabric, system_id:spine2, if_name=eth0] :14
[role:fabric, system_id:spine2, if_name=eth1] :15
```

```
[role:server, system_id:spine2, if_name=eth3] :16
[role:server, system_id:spine2, if_name=eth4] :17
```

Given the above, the output would be a number-set of

```
[role:fabric, system_id:spine1] : stddev([10, 11])
[role:fabric, system_id:spine2] : stddev([14, 15])
[role:server, system_id:spine1] : stddev([12, 13])
[role:server, system_id:spine2] : stddev([16, 17])
```

Processor: State

IN THIS SECTION

The State processor checks that a value is one of the specified anomalous states. It outputs DSS with anomaly values, such as 'true' if the value is in the specified states, and otherwise, it returns 'false'. (previously called 'state_check' and 'in_state'). The State processor supports multiple reference states and output is 'true' when input is in any of the specified states.

Parameter	Description
Input Types	Table(discrete state, accumulate=True or False)
Output Types	Table (discrete state)

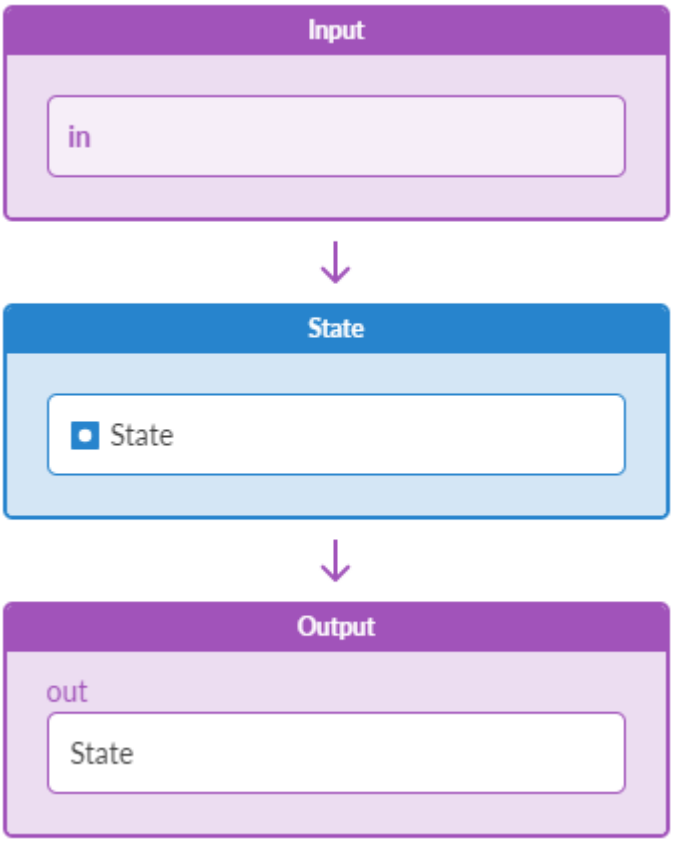
(Continued)

Parameter	Description

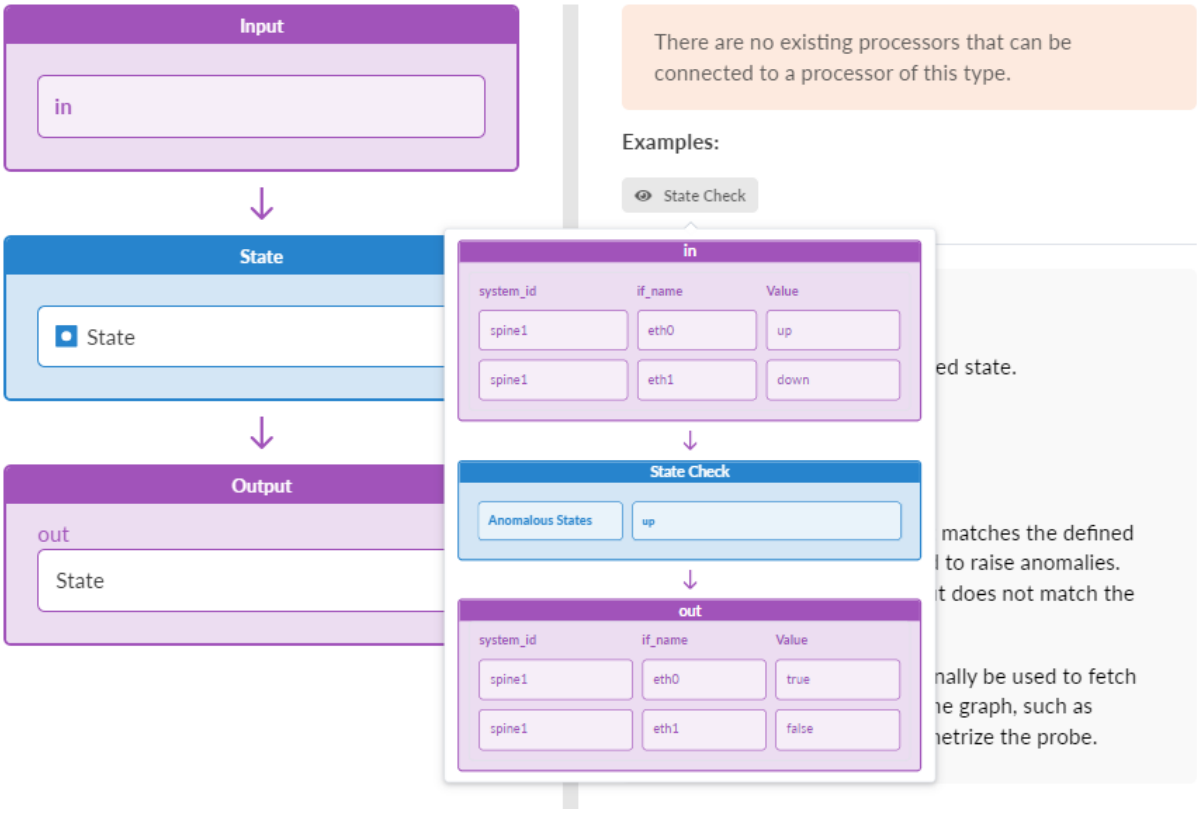
Graph Query (graph_query)	One or more queries on graph specified as strings, or a list of such queries. (String will be deprecated in a future release.) Multiple queries should provide all the named nodes referenced by the expression fields (including additional_properties). Graph query is executed on the "operation" graph. 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"]".
	
	In collector processors (`*_collector`, `if_counter`) it is used to choose a set of nodes for further processing (for example, all leaf devices, or all interfaces between leaf and spine devices)
	
	In other processors it is used for general parameterization and it is only supported as a list of queries.
	
	`graph_query: "node("system", role="leaf", name="system").`
	`out("hosted_interfaces").`
	`node("interface", name="iface").out("link").`
	`node("link", role="spine_leaf")"`
	
	`graph_query: ["node("system", role="leaf", name="system")",`
	`"node("system", role="spine", name="system")"]`
	
	Non-collector processors containing the `graph_query` configuration parameter, can be parameterized to use data from arbitrary nodes in the graph, such as property set nodes. Property sets allow you to parameterize macro level SLAs for individual business units. In the example below, `graph_query` matches a node of type `property_set` with label `probe_propset`. It's accessed using the special `query_result` variable, where Index `0` means it's the first node in query results. If a query returned `N` nodes, they could be accessed using indices starting from `0` to `N-1`. `ps` is what the actual node is referred to in the query; the rest depends on the structure of the node. The `int()` casting is required because values of `property_set` nodes are strings. Here it's assumed that a property set node has the label `probe_propset` and that the value `accumulate_duration` was already created.
	
	`graph_query: [node("property_set", label="probe_propset", name="ps")]`
	`duration: int(query_result[0]["ps"].values["accumulate_duration"])`

(Continued)

Parameter	Description
	Another example is a that probes can validate a compliance requirement; the compliance value may change over time and/or it can be used by more than one probe. Also, a probe can validate NOS versions on devices. In this case, property sets can be used to define the current NOS version requirement. If it changes tomorrow: change the property set value, instead of going under the probe stage.
Anomalous States	Expression that evaluates to DS value or list of DS values which is used for the check. For example, it can be: "'true'" (expression evaluating to a string) or "['missing', 'unknown', 'down']" (expression evaluating to a list of strings).
Anomaly MetricLog Retention Duration	Retain anomaly metric data in MetricDb for specified duration in seconds
Anomaly MetricLog Retention Size	Maximum allowed size, in bytes of anomaly metric data to store in MetricDB
Anomaly Metric Logging	Enable metric logging for anomalies
Enable Streaming (enable_streaming)	Makes samples of output stages streamed if enabled. An optional boolean that defaults to False. If set to True, all output stages of this processor are streamed in the generic protobuf schema.
Raise Anomaly (raise_anomaly)	Outputs “true” and “false” values, “true” meaning an appropriate item is anomalous, and "false" meaning the item is not anomalous. When Raise Anomaly is set to True, an actual anomaly is generated in addition to a sample in the output.



In the Add Processor window of the Apstra UI, hover over the "State Check" tooltip for a visual example of how the State processor functions.



Example: State

```
state: '"up"'
```

Sample Input (DS)

```
[if_name=eth0] : "up"
[if_name=eth1] : "down"
[if_name=eth3] : "up"
```

Sample Output (DSS)

```
[if_name=eth0] : "false"
[if_name=eth1] : "true"
[if_name=eth3] : "false"
```

If expression is used for the state field, then it's evaluated for each input item, and it results into item-specific state value. Properties of the respective output item are extended by the state property with value of the evaluated expression.

```
state: expected_if_state
```

Sample Input (DS):

```
[if_name=eth0,expected_if_state=up] : "up"
[if_name=eth1,expected_if_state=down] : "down"
[if_name=eth3,expected_if_state=up] : "down"
```

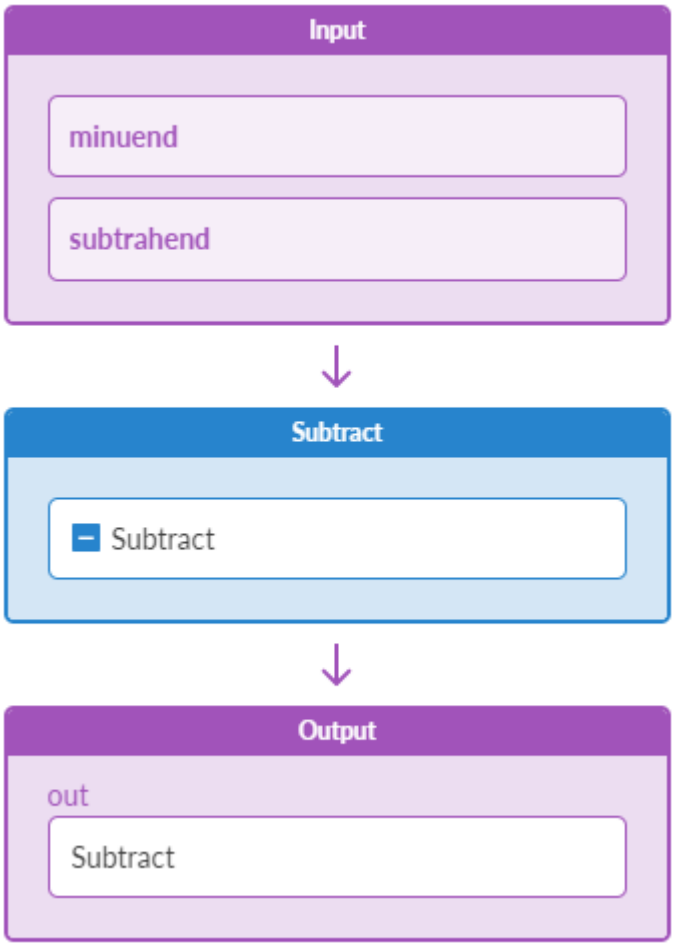
Sample Output (DSS)

```
[if_name=eth0,state=up] : "false"
[if_name=eth1,state=down] : "false"
[if_name=eth3,state=up] : "true"
```

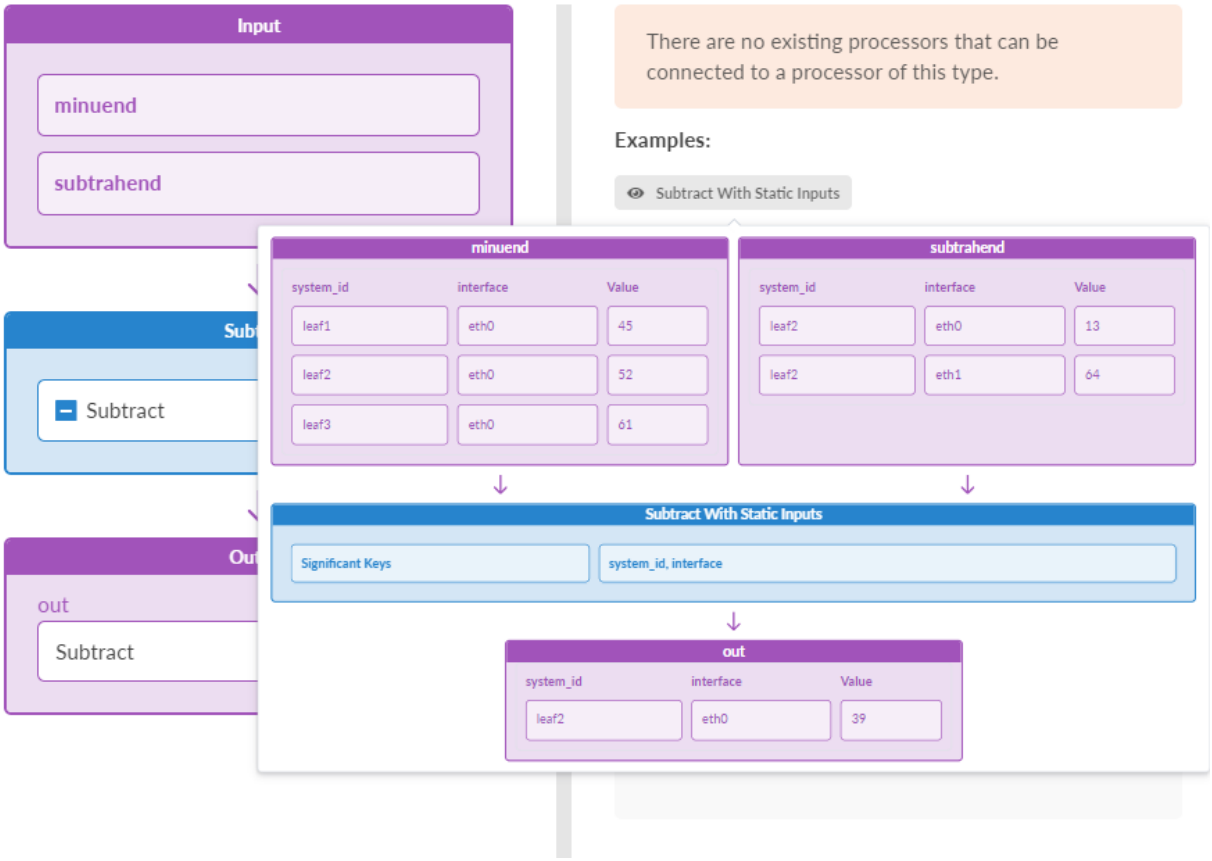
Processor: Subtract

One number is created on output for each number with the same properties in both inputs. For each input item the processor leaves only significant keys, drops the others and puts the result. If there is no common set of properties between both inputs, the output is the empty set.

Parameter	Description
Input Types	Table (number)
Output Types	Table (number)
Significant Keys (significant_keys)	List of keys to map items from the inputs for applying the specified operation. It is typically used by processors that take multiple inputs and perform operations on them. When inputs have the same sets of keys it does not need to be specified. When inputs have different sets of keys, it must be specified and it must allow only 1:1 items mapping from the given inputs, otherwise the probe will go into error state.
Enable Streaming (enable_streaming)	Makes samples of output stages streamed if enabled. An optional boolean that defaults to False. If set to True, all output stages of this processor are streamed in the generic protobuf schema.



In the Add Processor window of the Apstra UI, hover over the "Subtract With Static Inputs" tooltip for a visual example of how the Subtract processor functions.



Processor: Sum

IN THIS SECTION

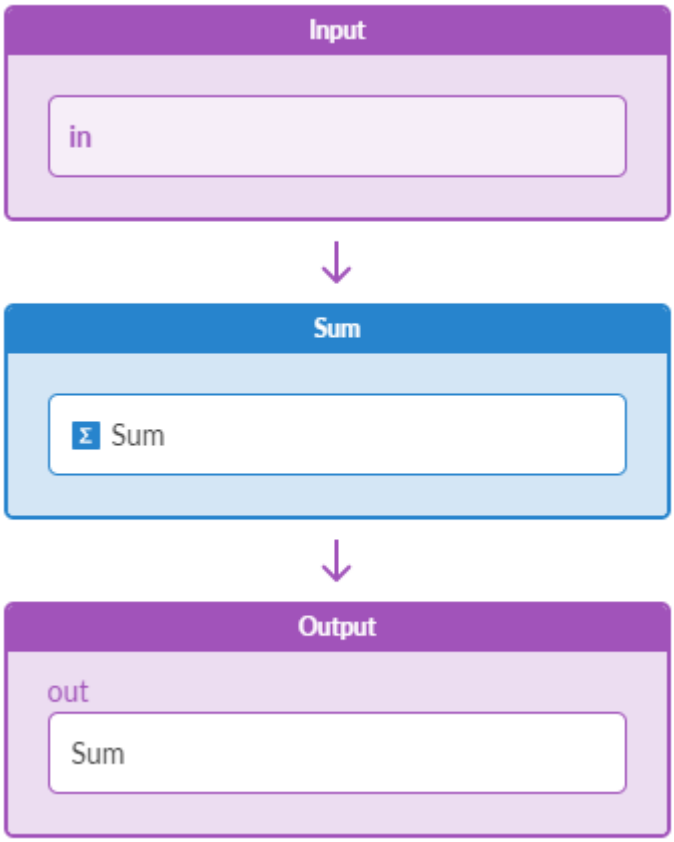
- Example: Sum Output | **1717**

The Sum processor groups as described by **Group by** property, then calculates sum and outputs one for each group.

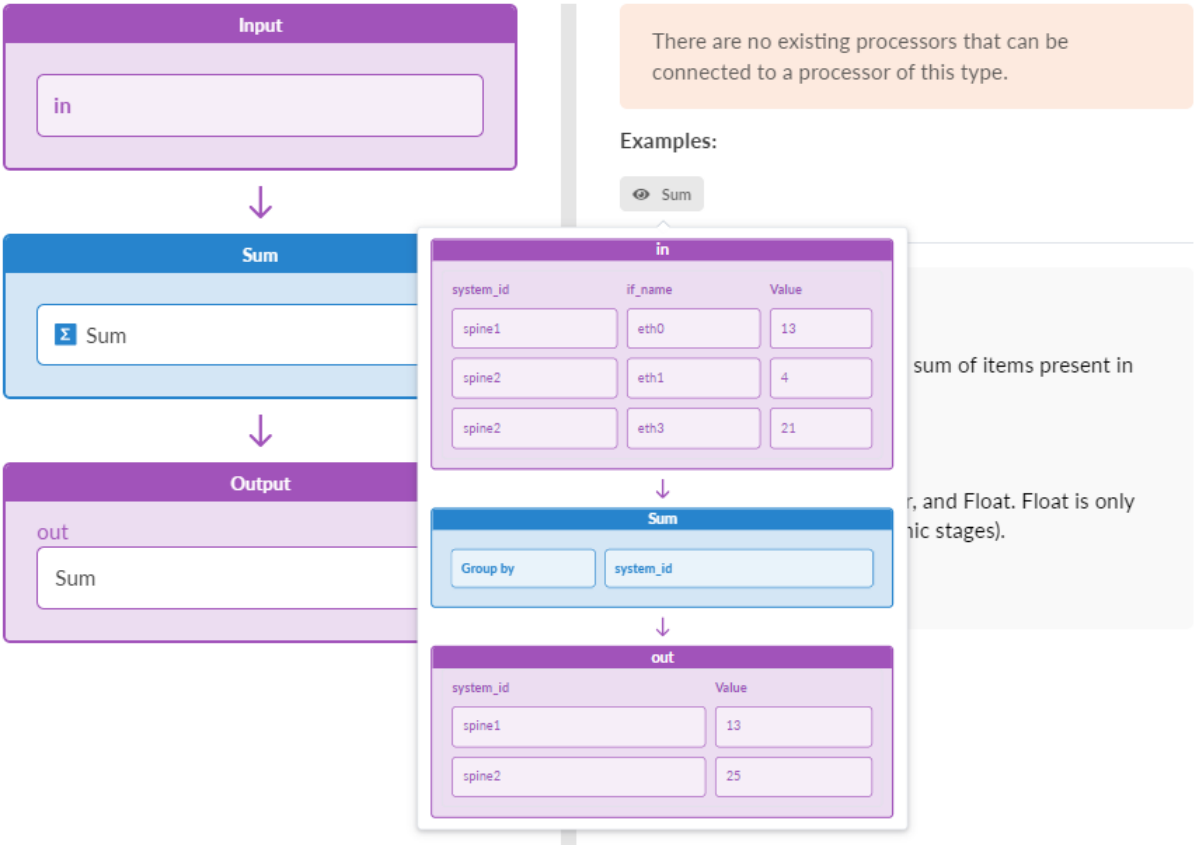
Parameter	Description
Input Types	Table (number), Table (number, accumulate=True)
Output Types	Table (number)

(Continued)

Parameter	Description
Group by (group_by)	Accepts a list of property names to group input items into output items, produces only one output group for the empty list.Most processors take input and produce output. Many of them produce one output per input (for example, if input is a DSS, output is a DSS of same size). However, some processors reduce the size of the output relative to the size of the input. Effectively, they partition the input into groups, run some calculation on each of the groups that produce a single value per each group, and use that as output. Clearly, the size of the output set depends on the grouping scheme. We call such processors **grouping processors** and they all take the **Group by** configuration parameter. In the case of an empty list, the input is considered to be a single group; thus, the output is of size 1 and either N, DS, or TS. If a list of property names is specified, for example ["system_id", "iface_role"], or a single property is specified, for example ["system_id"], we divide the input into groups such that for each group, every item in the group has the same values for the given list of property names. See the "standard deviation processor" on page 1703 example for how this works. The output type of a processor depends on a value of the group_by parameter; for an empty list, a processor produces a single value result, such as N, DS, or T, and for grouping by one or more properties it returns a set result, such as NS, DSS, or TS.
Enable Streaming (enable_streaming)	Makes samples of output stages streamed if enabled. An optional boolean that defaults to False. If set to True, all output stages of this processor are streamed in the generic protobuf schema.



In the Add Processor window of the Apstra UI, hover over the "Sum" tooltip for a visual example of how the Sum processor functions.



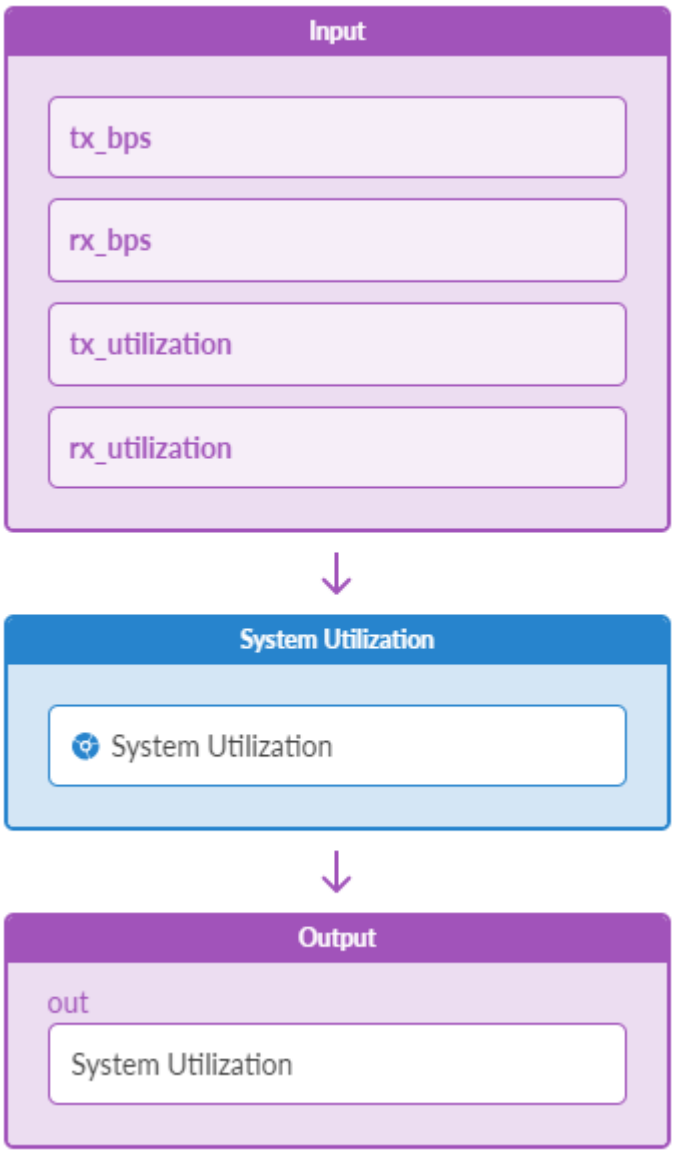
Example: Sum Output

See "standard deviation" on page 1703 example. It's the same except we calculate sum instead of std deviation.

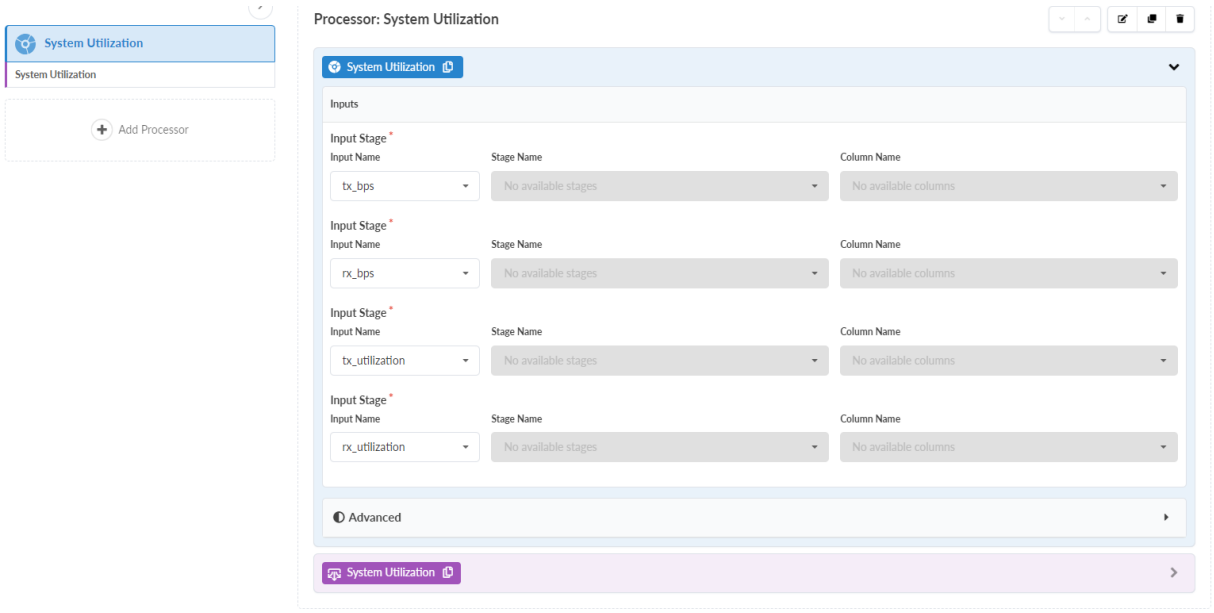
Processor: System Utilization

Interface Counters Utilization Per System processor groups detailed interface counter data by system ID and then calculates aggregate TX and RX bits, their aggregate utilization and identifies the highest TX and RX utilizations among the interfaces.

Parameter	Description
Enable Streaming (enable_streaming)	Makes samples of output stages streamed if enabled. An optional boolean that defaults to False. If set to True, all output stages of this processor are streamed in the generic protobuf schema.



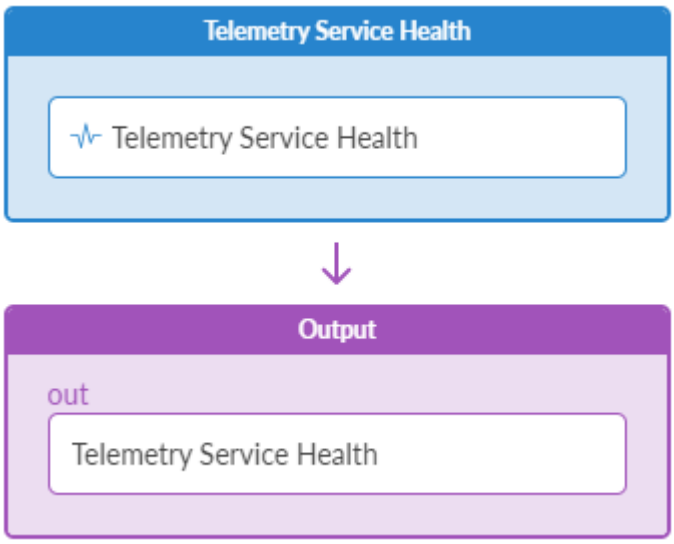
Configuration options:



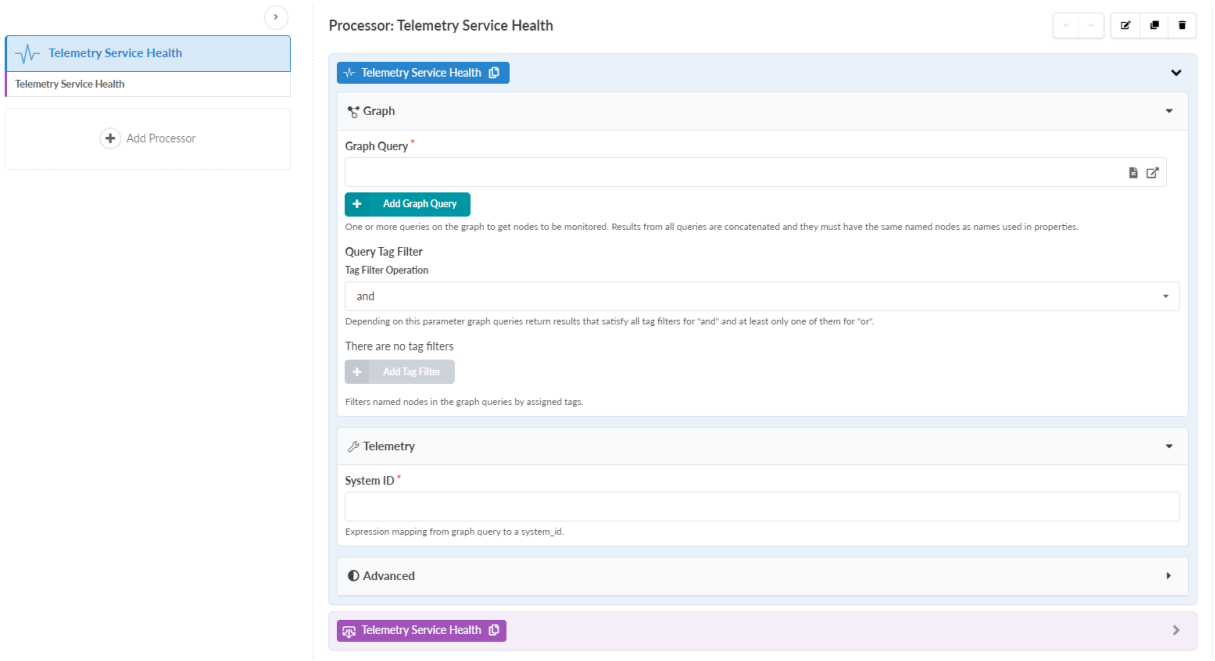
Processor: Telemetry Service Health

The Telemetry Service Health processor collects metrics and statistics on any telemetry service running on any system, including built-in and custom services.

The reported metrics includes, but not limited to, the Run count, Success count, Failure count, Timeout count and Underrun count. It will also reports the execution time of the service as well as raise True/False flags if the last execution had any failure, timeout or underrun.



Configuration options:



Processor: Time in State

IN THIS SECTION

The Time in State processor measures time when a value is in the range. For each input DS, monitor it over the last time_window seconds. If at any moment, for the state in state_range, the amount of time we have been in that state over the last time_window seconds falls into a range specified in the corresponding state_range entry, we set the corresponding output DS to 'true'. Otherwise, the output DS for a given input DS is nominally 'false'. (previously called 'time_in_state_check')

Parameter	Description
Input Types	Discrete-State (DS)
Output Types	Discrete-State (DS)
Time Window (time_window)	How long to monitor state. (seconds or an expression that evaluates to integer)

(Continued)

Parameter	Description
State Range (state_range)	Map state value to its allowed time range in seconds. dict mapping from a single possible state to a single range of time during the most recent time_window seconds that the value from input state is allowed to be in that state. At least one of the range object's two fields must be specified. The omitted field is regarded as "infinity". The fields are numbers (integers or floats) or expressions evaluated into numbers. State is a string or an expression that evaluates to string.

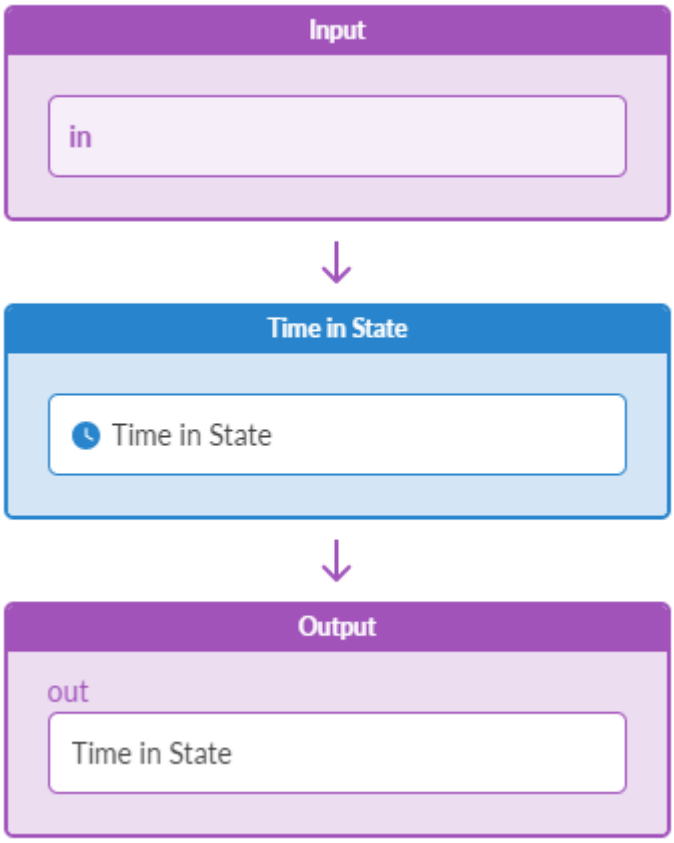
(Continued)

Parameter	Description

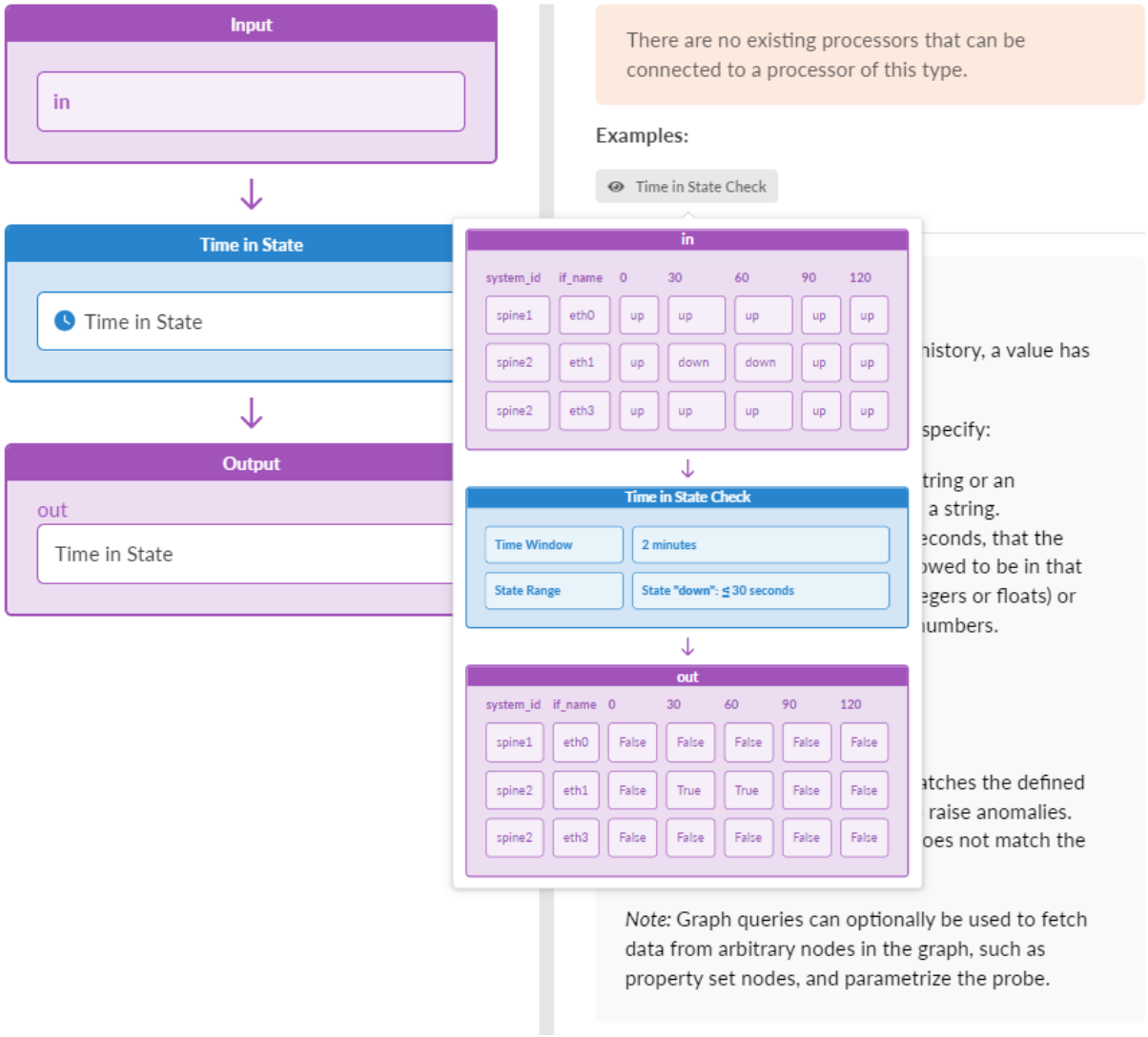
Graph Query (graph_query)	One or more queries on graph specified as strings, or a list of such queries. (String will be deprecated in a future release.) Multiple queries should provide all the named nodes referenced by the expression fields (including additional_properties). Graph query is executed on the "operation" graph. 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"]".
	
	In collector processors (`*_collector`, `if_counter`) it is used to choose a set of nodes for further processing (for example, all leaf devices, or all interfaces between leaf and spine devices)
	
	In other processors it is used for general parameterization and it is only supported as a list of queries.
	
	`graph_query: "node("system", role="leaf", name="system").`
	`out("hosted_interfaces").`
	`node("interface", name="iface").out("link").`
	`node("link", role="spine_leaf")"`
	
	`graph_query: ["node("system", role="leaf", name="system")",`
	`"node("system", role="spine", name="system")"]`
	
	Non-collector processors containing the `graph_query` configuration parameter, can be parameterized to use data from arbitrary nodes in the graph, such as property set nodes. Property sets allow you to parameterize macro level SLAs for individual business units. In the example below, `graph_query` matches a node of type `property_set` with label `probe_propset`. It's accessed using the special `query_result` variable, where Index `0` means it's the first node in query results. If a query returned `N` nodes, they could be accessed using indices starting from `0` to `N-1`. `ps` is what the actual node is referred to in the query; the rest depends on the structure of the node. The `int()` casting is required because values of `property_set` nodes are strings. Here it's assumed that a property set node has the label `probe_propset` and that the value `accumulate_duration` was already created.
	
	`graph_query: [node("property_set", label="probe_propset", name="ps")]`
	`duration: int(query_result[0]["ps"].values["accumulate_duration"])`

(Continued)

Parameter	Description
	Another example is a that probes can validate a compliance requirement; the compliance value may change over time and/or it can be used by more than one probe. Also, a probe can validate NOS versions on devices. In this case, property sets can be used to define the current NOS version requirement. If it changes tomorrow: change the property set value, instead of going under the probe stage.
Anomaly MetricLog Retention Duration	Retain anomaly metric data in MetricDb for specified time period
Anomaly MetricLog Retention Size	Maximum allowed size, in bytes of anomaly metric data to store in MetricDB
Anomaly Metric Logging	Enable metric logging for anomalies
Enable Streaming (enable_streaming)	Makes samples of output stages streamed if enabled. An optional boolean that defaults to False. If set to True, all output stages of this processor are streamed in the generic protobuf schema.
Raise Anomaly (raise_anomaly)	Outputs “true” and “false” values, “true” meaning an appropriate item is anomalous, and "false" meaning the item is not anomalous. When Raise Anomaly is set to True, an actual anomaly is generated in addition to a sample in the output.



In the Add Processor window of the Apstra UI, hover over the "Time in State Check" tooltip for a visual example of how the Time in State processor functions.



Example: Time in State

Config is set to:

```
time_window : 2 seconds
state_range: { "down" : [{"max": 1},] }
```

The above configuration means that for the input DS, we will set output to True and optionally raise an anomaly if the input is in the "down" state for more-than one second out of the last two seconds.

In the sample below, certain values are capitalized to indicate what has changed from the previous time.

Sample Input at time t=0

```
[if_name=eth0] : "up"
[if_name=eth1] : "up"
[if_name=eth3] : "up"
```

Sample Output at time t=0

```
[if_name=eth0] : "false"
[if_name=eth1] : "false"
[if_name=eth3] : "false"
```

Sample Input at time t=1:

```
[if_name=eth0] : "up"
[if_name=eth1] : "down"
[if_name=eth3] : "up"
```

Sample Output at time t=1

```
[if_name=eth0] : "false"
[if_name=eth1] : "false"
[if_name=eth3] : "false"
```

Sample Input at time t=2:

```
[if_name=eth0] : "up"
[if_name=eth1] : "down"
[if_name=eth3] : "up"
```

Sample Output at time t=2

```
[if_name=eth0] : "false"
[if_name=eth1] : "true"
[if_name=eth3] : "false"
```

Sample Input at time t=3:

```
[if_name=eth0] : "up"
[if_name=eth1] : "up"
[if_name=eth3] : "up"
```

Sample Output at time t=3

```
[if_name=eth0] : "false"
[if_name=eth1] : "True"
[if_name=eth3] : "false"
```

Sample Input at time t=4:

```
[if_name=eth0] : "up"
[if_name=eth1] : "up"
[if_name=eth3] : "up"
```

Sample Output at time t=4

```
[if_name=eth0] : "false"
[if_name=eth1] : "false"
[if_name=eth3] : "false"
```

If expressions are used for min or max fields for states specified in the state property, then they are evaluated for each input item which results into item-specific thresholds. Properties of the respective output items are extended by range_min or range_max keys with calculated values.

If state key is an expression, output items are extended with state key. The same applies for time_window property.

Configuration:

```
time_window : int(100/context.severity)
state_range: { context.ref_state : [{"max": "int(20*(context.severity/5.0))"}] }
```

Sample Input at times t=0..6:

```
[if_name=eth0,severity=1,ref_state=down] : "down"
[if_name=eth1,severity=2,ref_state=down] : "down"
```

Sample Output at time t=6:

```
[if_name=eth0,range_max=4,time_window=100,state=down] : "true"
[if_name=eth1,range_max=8,time_window=50,state=down] : "false"
```

Processor: Traffic Monitor

The Traffic Monitor processor selects interfaces according to the configuration and outputs all available interface-related counters (e.g tx_bits, rx_bits etc) and interface utilization.

Parameter	Description
Input Types	No inputs. This is a source processor

(Continued)

Parameter	Description

Graph Query (graph_query)	One or more queries on graph specified as strings, or a list of such queries. (String will be deprecated in a future release.) Multiple queries should provide all the named nodes referenced by the expression fields (including additional_properties). Graph query is executed on the "operation" graph. 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"]".
	
	In collector processors (`*_collector`, `if_counter`) it is used to choose a set of nodes for further processing (for example, all leaf devices, or all interfaces between leaf and spine devices)
	
	In other processors it is used for general parameterization and it is only supported as a list of queries.
	
	`graph_query: "node("system", role="leaf", name="system").`
	`out("hosted_interfaces").`
	`node("interface", name="iface").out("link").`
	`node("link", role="spine_leaf")"`
	
	`graph_query: ["node("system", role="leaf", name="system")",`
	`"node("system", role="spine", name="system")"]`

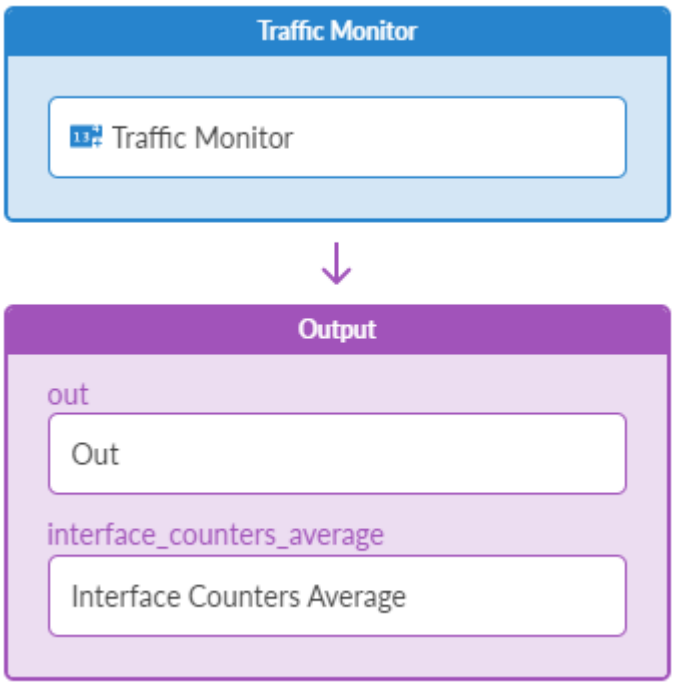
Query Expansion	For every path, originally returned by graph queries, passed to each generator the latter one produces a set of items and for each item it produces a new path extended by a corresponding property name which value is set of a value of the produced item.

(Continued)

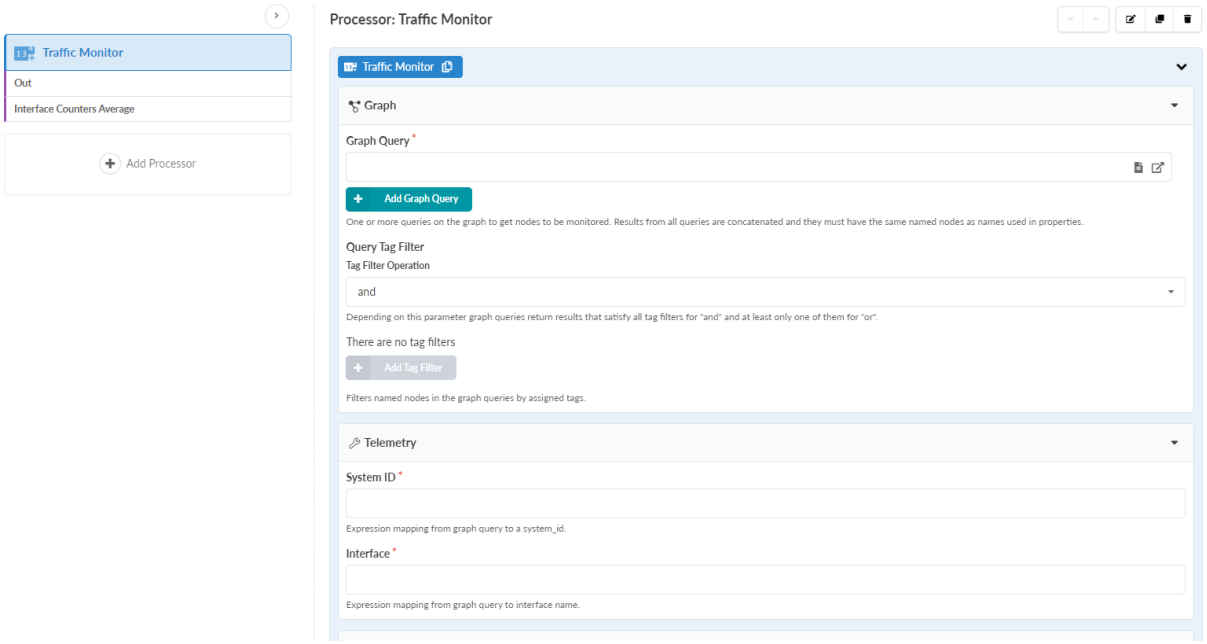
Parameter	Description
Query Group by (query_group_by)	List (of strings) of node and relationship names used in the graph query to group query results by. Each element in this list represents a named node or relationship matcher in the `graph_query` field.It is not an expression to be consistent with existing `group_by` field in grouping processors. Non-expression is simple and more intuitive. When grouping is active (`query_group_by` is not null), query results are d by the specified list of names, where one output item is created per each group. In this case, the expressions can only access matcher names specified in `query_group_by` and the query results for each group are accessed using a new `group_items` variable. The `group_items` variable is a list of query results, where each result has named nodes/relationships, not present in `query_group_by`. The following list describes the behavior for various values of this field: • Value of query_group_by field - Semantics • Omitted or provided as json null (ala None in Python) - No grouping is done. This is equivalent to current behavior of extensible_data_collector. Using 'group_items' in this case is not permitted and results in probe error state. • Empty list ([]) - Produces one group containing all the query results. • One or more matcher names - The query results are grouped by the specified nodes or relationships. If this list covers all available matchers in the query, the number of groups is equal to the number of query results.
Query Tag Filter (query_tag_filter)	Filters named nodes in the graph queries by assigned tags.
Interface	Expression mapping from graph query to interface name, e.g. "iface.if_name" if "iface" is a name in the graph query.
Port Speed	Expression mapping from graph query to link speed in bits per second, e.g. "functions.speed_to_bits(link.speed)" if "link" is a name in the graph query.
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.
Period	Duration of the averaging period

(Continued)

Parameter	Description
Additional Keys	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. The value of this property is evaluated for each item to associate items with metrics provided by a corresponding collector service. The association is done by keys because each collector reports a set of metrics where each metric is identified by a key in a format that is specific for each collector.
Enable Streaming (enable_streaming)	Makes samples of output stages streamed if enabled. An optional boolean that defaults to False. If set to True, all output stages of this processor are streamed in the generic protobuf schema.



Configuration options:



Processor: Union

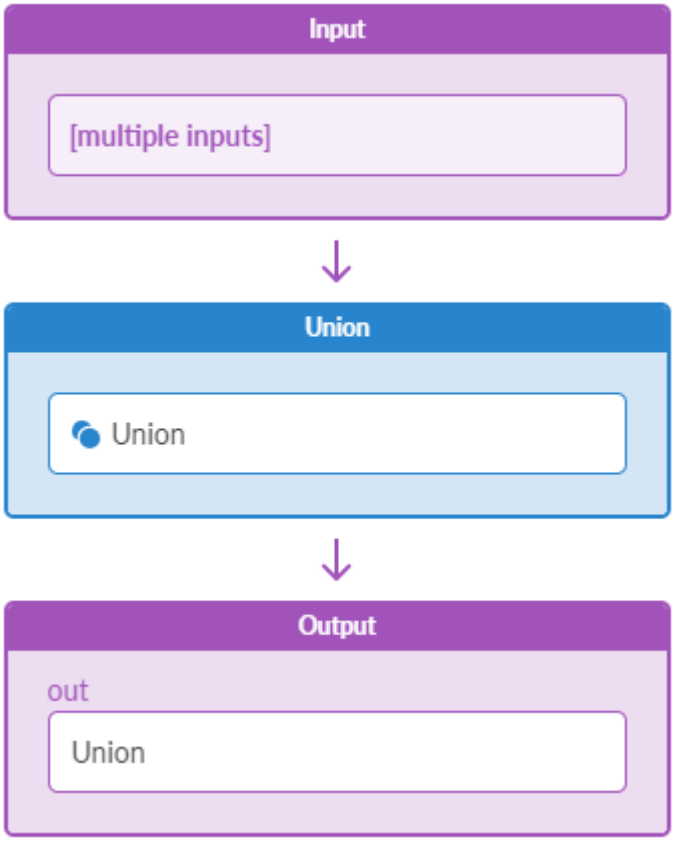
IN THIS SECTION

The Union processor merges all input items into one set of items. For each input item the processor leaves only signification keys, drops the others and puts the result.

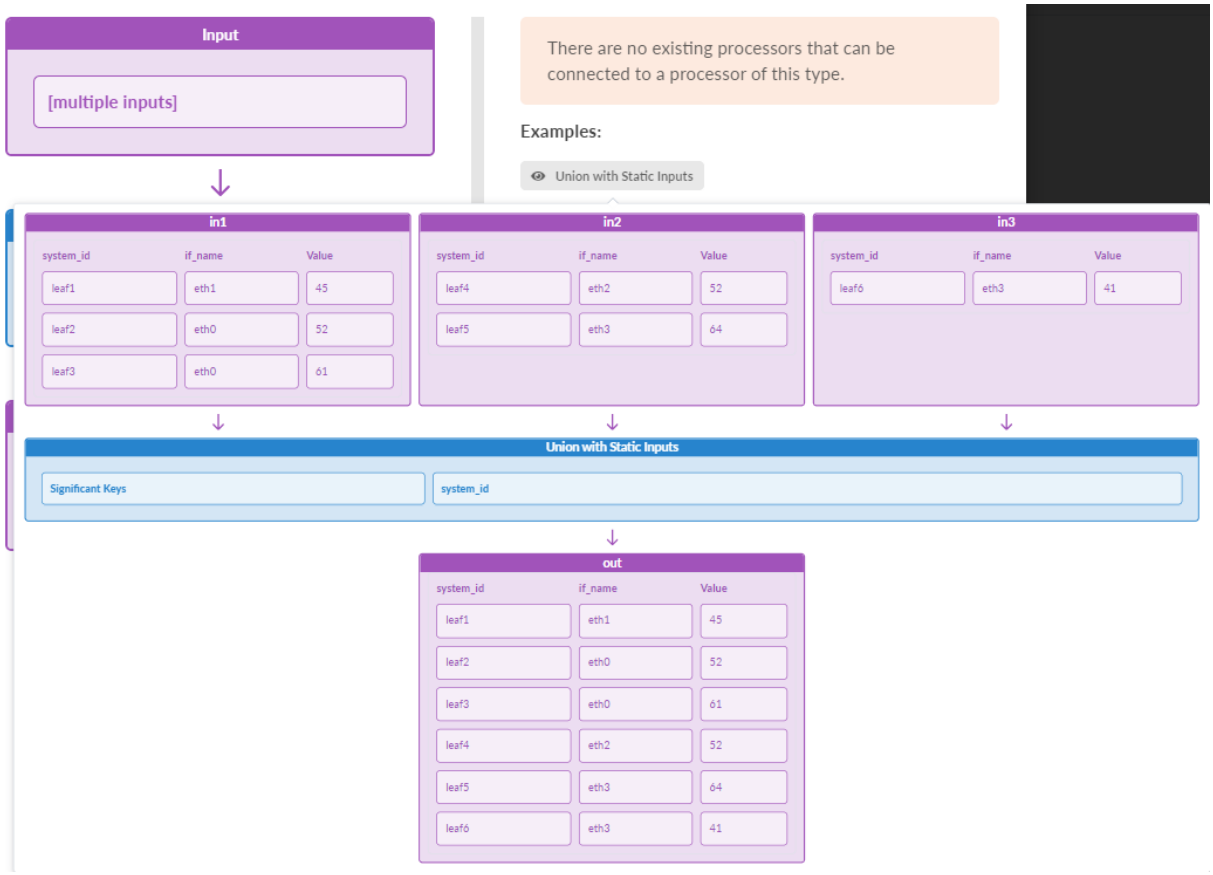
Parameter	Description
Input Types	Table (number or text or discrete state)
Output Types	Table (number or text or discrete state)
Significant Keys (significant_keys)	List of keys to map items from the inputs for applying the specified operation. It is typically used by processors that take multiple inputs and perform operations on them. When inputs have the same sets of keys it does not need to be specified. When inputs have different sets of keys, it must be specified and it must allow only 1:1 items mapping from the given inputs, otherwise the probe will go into error state.

(Continued)

Parameter	Description
Enable Streaming (enable_streaming)	Makes samples of output stages streamed if enabled. An optional boolean that defaults to False. If set to True, all output stages of this processor are streamed in the generic protobuf schema.



In the Add Processor window of the Apstra UI, hover over the "Union with Static Inputs" tooltip for a visual example of how the Union processor functions.



Example: Union

Config is set to:

```
significant_keys: ["system_id"]
```

Consider we have inputs with device temperature information.

Input "in_1":

```
[system_id=leaf1,interface=eth1]: 45
[system_id=leaf2,interface=eth0]: 52
[system_id=leaf3,interface=eth0]: 61
```

Input "in_2":

```
[system_id=leaf4,interface=eth2]: 52
[system_id=leaf5,interface=eth3]: 64
```

Input "in_3":

```
[system_id=leaf6,interface=eth3]: 41
```

Output will be the following.

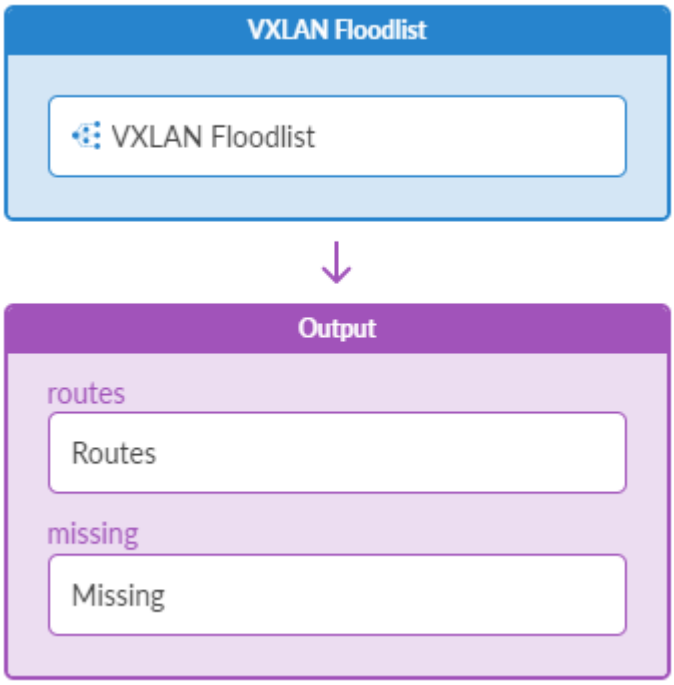
Output "out":

```
[system_id=leaf1]: 45
[system_id=leaf2]: 52
[system_id=leaf3]: 61
[system_id=leaf4]: 52
[system_id=leaf5]: 64
[system_id=leaf6]: 41
```

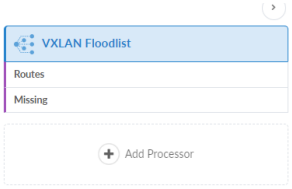
Processor: VXLAN Floodlist

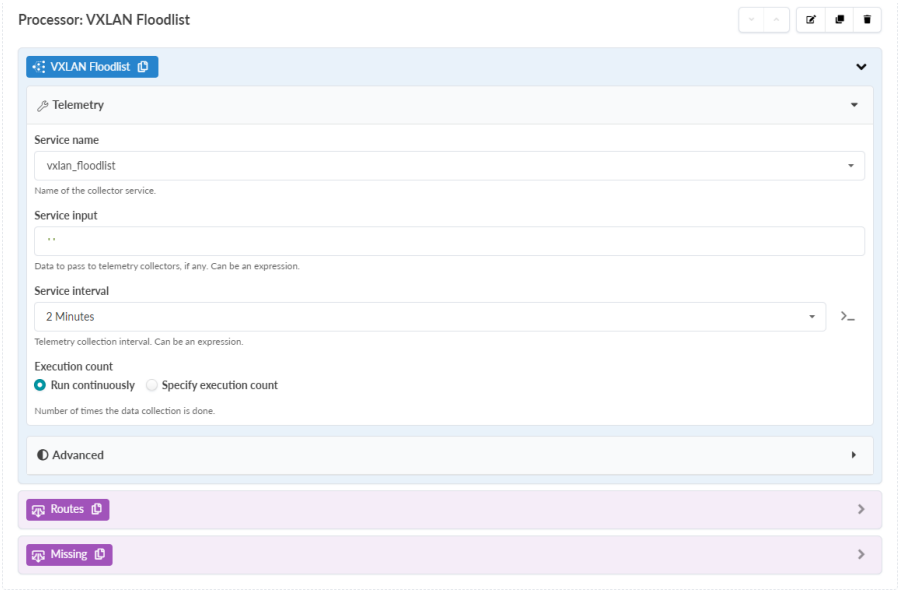
The VXLAN Floodlist processor generates a configuration containing expectations of vxlan floodlist routes.

Parameter	Description
Execution count	Number of times the data is collected
Service input (service_input)	Data to pass to telemetry collectors, if any. Can be an expression.
Service interval (service_interval)	Telemetry collection interval in seconds. Can be an expression.
Service name (service_name)	Name of the custom collector service.
Enable Streaming (enable_streaming)	Makes samples of output stages streamed if enabled. An optional boolean that defaults to False. If set to True, all output stages of this processor are streamed in the generic protobuf schema.



Configuration options:





Configlet Examples (Design)

IN THIS SECTION

Juniper Junos Configlet Interface-Level Example on 4.0.2: gigether-options

When you're creating an interface-level configlet during the design phase, you won't know interface names. It's not until you're working in the blueprint that you'll have that information. Interface-level configlets for Junos are designed for you to enter details without including the `set interface` command. For example, to change Junos interface "gigether-options", you can use a interface-level hierarchical or set configlet.

```
gigether-options no-auto-negotiation
gigether-options fec none
```

```
gigether-options {
    no-auto-negotiation;
```

```
    fec none;
}
```

When you import the configlet into your blueprint, you'll specify interfaces such as `xe-0/0/0`. For a Junos Interface-Level set configlet Apstra software will prepend the set commands:

```
set interfaces xe-0/0/0 gigether-options no-auto-negotiation
set interfaces xe-0/0/0 gigether-options fec none
```

For a Junos Interface-Level hierarchical configlet Apstra software will load Junos structured configuration:

```
interfaces {
    xe-0/0/0 {
        gigether-options {
            no-auto-negotiation;
            fec none;
        }
    }
}
```

Juniper Junos Configlet Example on 4.0.2: MTU (section Interface-Level: Delete)

If you want to use a Junos interface-level configlet to remove an existing configuration, you can use an interface level delete configlet. Like the interface level set configlet, when you are creating the configlet during the design phase, you won't know interface names. It's not until you're working in the blueprint that you'll have that information. Interface-level delete configlets for Junos are designed for you to enter details without including the `delete interface` command. For example, to remove the Junos interface "mtu" configuration.

```
mtu
```

When you import the configlet into your blueprint, you'll specify interfaces such as `xe-0/0/0`. For a Junos Interface-Level delete configlet Apstra software will prepend the delete commands:

```
delete interfaces xe-0/0/0 mtu
```

Juniper Junos Configlet Example on 4.0.2 Example: SNMP (multiple sections)

You can create a configlet with a generator at the Top-Level to enable SNMP. To avoid SNMP alarms on server-facing interfaces, for example, you can create a second generator at the Interface-Level to set up `no-traps`.

Top-Level template text is validated to begin with ‘set’ or ‘delete’. See below for example text.

```
set snmp community public authorization read-only
set snmp description “this is configlet test” set snmp location “Apstra DC”
set snmp contact “june at juniper dot net”
set snmp trap-group authentication-traps targets 10.0.10.1
set snmp trap-group authentication-traps targets 192.168.15.27
set snmp trap-group authentication-traps categories authentication
```

Interface-Level template text is not validated because it's not a complete CLI command. See below for example text.

```
no-traps
```

When you import the configlet into your blueprint, you'll specify interfaces such as `ex-0/0/0` and Apstra software will prepend the set command as .

```
set interface xe-0/0/0 no-traps
```

Juniper Junos Configlet Example on 4.0.1 and 4.0.0: NTP (section SYSTEM)

Sample text for configuring NTP servers on Junos devices. (On Apstra version 4.0.2 SYSTEM is called Top-Level/Hierarchical.)

```
system {
    ntp {
        boot-server 10.1.4.1;
        server 10.1.4.2;
    }
}
```

Cisco NX-OS Configlet Example: Syslog (section SYSTEM)

Sample text for configuring Syslog on NX-OS devices.

```
logging server 192.168.0.30
logging facility local3
logging trap warning
```

```
no logging server 192.168.0.30
no logging facility local3
no logging trap warning
```

Arista EOS Configlet Example: NTP (section SYSTEM)

Sample text for configuring NTP servers on EOS devices. This configlet uses property sets for the NTP server IP addresses.

```
ntp server {{NTP_SERVER_1}}
ntp server {{NTP_SERVER_2}}
```

```
no ntp server {{NTP_SERVER_1}}
no ntp server {{NTP_SERVER_2}}
```

Arista EOS Configlet Example: Interface Speed (section INTERFACE)

Sample text for applying 'speed auto' to an interface. (You specify devices and interfaces when you import the configlet into a blueprint.)

```
speed auto
```

```
no speed auto
```

Enterprise SONiC Configlet Example: NTP (section SYSTEM)

Sample text for using the `config` command to set up an NTP server to use mgmt VRF on SONiC devices.

```
sonic-db-cli CONFIG_DB hset 'NTP |global' vrf mgmt
config ntp add {{ntp_server}}
```

```
config ntp del {{ntp_server}}
```

Enterprise SONiC Configlet Example: SNMP (section SYSTEM)

Sample text for using the `config` command to set up an SNMP snmptrap to use mgmt VRF on SONiC devices.

```
config snmptrap modify 2 {{SNMP_SERVER}} -v mgmt -c mypass
```

```
config snmptrap del 2
```

Enterprise SONiC Configlet Example: Syslog (section SYSTEM)

Sample text for using the `config` command to set the Syslog server for SONiC devices.

```
config syslog add {{syslog_host}}
```

```
config syslog del {{syslog_host}}
```

Enterprise SONiC Configlet Example: Static Route (section FRR)

Sample text for adding a static route

```
ip route 4.2.2.2/32 {{static_route_next_hop}}
ip route 4.2.2.3/32 {{static_route_next_hop}}
```

Enterprise SONiC Configlet Example: sonic-cli Commands (section SYSTEM)

Sample text for using the `sonic-cli` command to set up the `delay-restore` option for SONiC `mclag`. You must use `sudo -u admin` at the beginning, `and surround terms that contain spaces with single quotes` in each sonic-cli command, and `< /dev/console` at the end.

```
sudo -u admin sonic-cli -c config -c 'mclag domain 1' -c 'delay-restore 600' < /dev/console
```

```
sudo -u admin sonic-cli -c config -c 'mclag domain 1' -c 'no delay-restore' < /dev/console
```

Apstra EVPN Support Addendum

IN THIS SECTION

When deploying EVPN on Apstra-supported devices and NOSs, be aware of several caveats and limitations. Even though EVPN is a standard, vendors implement protocols in very different manners. Also, different ASICs support varying feature sets that impact EVPN BGP VXLAN implementations (Routing In and Out of Tunnels (RIOT) for example). The following sections describe supported EVPN deployment implementations.

Qualified Vendor and NOS

Apstra software supports EVPN on the following hardware. For recommended NOS versions, see "Qualified Devices and NOS Versions" on page 1503.

Hardware ASIC Support

Apstra supports EVPN on the following hardware ASICs:

- Arista DCS 7280SE with Arad chipset
- Cisco Cloudscale
- Mellanox Spectrum A1
- Trident Trident2 (see below)
- Trident Trident2+ (see below)
- Trident Trident3 (see below)
- Trident Tomahawk (see below)
- Juniper Q5

Table 74: Apstra EVPN ASIC Support

ASIC	Example Switches	Notes
Arista Trident2	Arista DCS-7050	Can use as Spine, Leaf, or Border Leaf. Must set up EOS Recirculation interface(s) to use as a Layer3 Leaf (see Arista VXLAN documentation for more information).
Arista Trident3	DCS-7050CX3	Can use as Spine, Leaf, or Border Leaf.
Arista XP80	Arista DCS-7160	Ca use as Spine, Leaf, or Border Leaf.
Arista Jericho	DCS-7280R	Can use as Spine, Leaf, or Border Leaf.
Cisco Cloudscale	Cisco 93180YC-EX	Can use as Spine, Leaf, or Border Leaf
Cisco Trident2 with ALE	Cisco 9396PX, 9372PX, 9332PQ, 9504	Can use as Spine, Leaf, or Border Leaf (see TCAM Carving in NXOS section).

Table 74: Apstra EVPN ASIC Support *(Continued)*

ASIC	Example Switches	Notes
Cisco Trident2+	Cisco 3132Q-V	Can't use as Border Leaf
Juniper Q5	Juniper QFX10002	Can use as Spine, Leaf, or Border Leaf
Juniper Trident2	Juniper QFX5100	Can use as Spine or Layer2 Leaf
Juniper Trident2+	Juniper QFX5110	Can use as Spine, Leaf, or Border Leaf
Juniper Trident3	Juniper QFX5120	Can use as Spine, Leaf, or Border Leaf

For recommended NOS versions, refer to Qualified Devices and NOS <device_support>.

Limitations

IN THIS SECTION

EVPN Layer2 Limitations

- VLAN (Rack-local) Virtual networks must be in the default routing zone.
- VxLAN (Inter-rack) Virtual networks can't be part of the default routing zone.

EVPN Layer3 Limitations

- Generic systems with BGP peering to non-default routing zones must connect to leaf devices.
- Generic systems with BGP peering only to the default routing zone can connect to leaf devices, spine devices or superspine devices.
- Multi-zone security segmentations only support up to 16 routing zones (VRFs) on Arista (HW Limitation)

- Inter routing zone (VRF) routing must be handled on a generic system (EVPN type 5 route leaking)
- All BGP sessions and loopback addresses are part of the default routing zone.

TCAM Carving in NX-OS

To successfully deploy EVPN on Cisco Nexus devices other then Cisco Cloudscale, you must first configure Cisco NXOS TCAM carving. These other devices may include Cisco NXOSv, or Cisco Nexus "Trident2" devices such as 9396PX, 9372PX, 9332PQ, or 9504. On Cisco NXOS the ARP Suppression feature is used in order to minimize ARP flooding.

For details, see Juniper Support Knowledge Base article KB36733

Before installing the device agent, we recommend that you apply TCAM Carving during device management setup or during Cisco Power-on Auto Provisioning (POAP). TCAM Carving requires a device reboot.

Alternatively, you can apply TCAM Carving with configlets when you deploy the blueprint. You must manually reboot devices.

Use `show hardware access-list tcam region` to show and verify TCAM allocation on Cisco NX-OS.

Cisco NXOSv TCAM Carving

```
hardware access-list tcam region vacl 0
hardware access-list tcam region racl 0
hardware access-list tcam region arp-ether 256
```

```
no hardware access-list tcam region arp-ether 256
no hardware access-list tcam region racl 0
no hardware access-list tcam region vacl 0
```

Cisco Trident2 TCAM Carving

```
hardware access-list tcam region l3qos 0
hardware access-list tcam region arp-ether 256 double-wide
```

```
no hardware access-list tcam region l3qos 0
no hardware access-list tcam region arp-ether 256 double-wide
```

Arista EOS VxLAN Routing

IN THIS SECTION

Recirculation Interface for Arista Trident2 Devices

VxLAN Routing for Trident2 devices (for example, 7050QX-32) is supported but requires assigning EOS recirculation interfaces to unused physical interfaces on the device. You can use configlets to deploy this to all devices that require this configuration.

```
interface Recirc-Channel501
 switchport recirculation features vxlan
interface Ethernet35
 traffic-loopback source system device mac
 channel-group recirculation 501
interface Ethernet36
```

```
 traffic-loopback source system device mac
 channel-group recirculation 501
```

```
interface Ethernet35
 no traffic-loopback source system device mac
 no channel-group recirculation 501
interface Ethernet36
 no traffic-loopback source system device mac
 no channel-group recirculation 501
no interface Recirc-Channel501
```

VxLAN Routing System Profile for Arista Jericho Devices

We recommend when using VxLAN Routing for Jericho devices (for example, 7280SR-48C6) that you assign EOS VxLAN Routing System Profile on the device.

Before installing the device agent, we recommend that you apply the Arista TCAM system profile during the device management setup or during Arista Zero-Touch Provisioning (ZTP). TCAM system profile requires a device reboot.

Alternatively, you can use configlets to deploy this to all devices requiring this configuration and manually reboot the devices.

```
hardware tcam
   system profile vxlan-routing
```

```
hardware tcam
   no system profile vxlan-routing
```

VxLAN Routing Profile for Arista Arad Devices

We recommend when using VxLAN Routing for Arista Arad devices (for example, on 7280SE platform) that you assign EOS VxLAN Routing Profile on the device.

Before installing the device agent, we recommend that you apply the Arista TCAM system profile during the device management setup or during Arista Zero-Touch Provisioning (ZTP). TCAM system profile requires a device reboot.

Alternatively, you can use configlets to deploy this to all devices requiring this configuration and manually reboot the devices.

```
hardware tcam
   profile vxlan-routing
```

Graph Node VTEP Types

IN THIS SECTION

Unicast VTEPs

Unicast VTEPs do not apply to Arista.

Cisco Unicast VTEPs - Vendor Definition: Anycast VTEP

Apstra IP Allocation

Unique per leaf in MLAG pair

Not allocated to singleton switches

MLAG Configuration

```
interface loopback1
   IP address 10.0.0.1/32
   IP address 10.0.0.3/32 secondary
```

```
interface nve1
   source-interface loopback1
```

```
interface loopback1
   IP address 10.0.0.2/32
   IP address 10.0.0.3/32 secondary
interface nve1
   source-interface loopback1
```

Single Switch Configuration

```
interface loopback1
   IP address 10.0.0.1/32
interface nve1
   source-interface loopback1
```

Logical VTEPs

Arista Logical VTEPs

Apstra IP Allocation

Logical VTEP configured as primary IP on loopback1 interface for both MLAG and singleton switches

All top of rack nodes share same logical VTEP IP:

- MLAG leaf devices share same logical VTEP IP
- Singleton leaf device gets its own VTEP IP

MLAG Configuration

```
interface loopback1
   IP address: 10.0.0.1/32
   IP address: 10.0.0.4/32 secondary
```

```
interface vxlan1
   vxlan source-interface loopback1
```

```
interface loopback1
   IP address: 10.0.0.1/32
   IP address: 10.0.0.4/32 secondary
interface vxlan1
   vxlan source-interface loopback1
```

Single Switch Configuration

```
interface loopback1
   IP address: 10.0.0.5/32
   IP address 10.0.0.4/32 secondary
interface vxlan1
   vxlan source-interface loopback1
```

Anycast VTEP

Anycast VTEPs do not apply to Cisco.

Arista Anycast VTEPs

Apstra IP Allocation

One anycast VTEP for entire blueprint, shared between all Arista leaf devices

Configured as secondary IP on loopback1 interface

MLAG Configuration

```
interface loopback1
   IP address 10.0.0.1/32
   IP address 10.0.0.5/32 secondary
interface vxlan1
   vxlan source-interface loopback1
```

```
interface loopback1
   IP address 10.0.0.1/32
```

```
   IP address 10.0.0.5/32 secondary
interface vxlan1
   vxlan source-interface loopback1
```

Single Switch Configuration

```
interface loopback1
   IP address 10.0.0.5/32
   IP address 10.0.0.4/32 secondary
interface vxlan1
   vxlan source-interface loopback1
```

Apstra Server Configuration File

IN THIS SECTION

/etc/aos/aos.conf

Controller

```
admin@aos-server:/etc/aos$ cat aos.conf
[controller]
metadb=eth0

# Role for the controller. Set the option to "slave" in order to setup AOS as a
# slave AOS. The options "metadb" and "node_id" should be also set while
# setting "role" to "slave"
role = controller
# Id of the slave node. Empty in case the server is the controller. The ID is
# generated by the controller.
node_id =
```

Security

```
[security]

# ***EXPERIMENTAL FEATURE*** This feature should not be enabled without Apstra
# engineering assistance. Enable secure connections for AOS system agents.
enable_secure_sysdb_connection = 0
# This encrypts sensitive data when sending configuration to device. This also
# enables aos agents to use appropriate credentials to access and/or configure
# device.   Default behavior to configure or run commands using device root
```

```
# Note:  Manual agent installation will not work if this is enabled.
enable_encryption_to_device = 0
```

Log Rotate

```
[logrotate]

# AOS has builtin log rotate functionality. You can disable it by setting
# <enable_log_rotate> to 0 if you want to use linux logrotate utility to manage
# your log files. AOS agent reopens log file on SIGHUP
enable_log_rotate = 1
# Log file will be rotated when its size exceeds <max_file_size>
max_file_size = 1M
# The most recent <max_kept_backups> rotated log files will be saved. Older
# ones will be removed. Specify 0 to not save rotated log files, i.e. the log
# file will be removed as soon as its size exceeds limit.
max_kept_backups = 5
# Interval, specified as <hh:mm:ss>, at which log files are checked for
# rotation.
check_interval = 1:00:00
# Maximum number of recent invalid persistence group kept
max_kept_invalid_persistence_groups = 3
```

Auth Sysdb Log Rotator

```
[auth_sysdb_log_rotator]

# AOS has builtin auth sysdb persistence file rotation functionality. Default
# value is 1 which means sysdb retention policy is enabled. You can disable it
# by setting it to 0 and you also can enable it again by setting it to 1. All
# retention policy parameters will be reloaded by restarting AOS service, or
# sending SIGHUP signal to SysdbResourceManager agent via "sudo kill -s 1
# $(pgrep -f SysdbResourceManager)"
enable_auth_sysdb_rotate = 1
# Maximum number of backup copies of valid auth sysdb persistence file groups
# in /var/lib/aos/db. AOS will remove all the older groups. Default value is 5,
```

```
# which means AOS will keep the latest 5 groups. Min value is 3. It should be
# specified as a positive number or empty. Leaving it empty means no groups
# number limitation. It will be set to default value if it is configured in
# invalid format. It will be set to minimum value if it is configured to a
# smaller value.
max_kept_backups = 5
# Maximum total size of valid auth sysdb persistence file groups in
# /var/lib/aos/db. Default value is empty, which means no size limitation. It
# should be specified as empty or a positive number ending with k/m/g (case
# insensitive) or no suffix. Otherwise, it will be set to default value. AOS
# will keep at least 3 valid groups no matter how <max_total_files_size> being
# configured.
max_total_files_size =
# Interval, specified as <hh:mm:ss>, at which auth sysdb persistence files are
# checked for rotation. Default value is 1:00:00. It will be set to default
# value is it is configured in invalid format. Min value is 00:01:00. It will
# be set to min value if it is configured to a smaller value. AOS also update
# all the retention policy parameters per <check_interval> when it is enabled.
check_interval = 1:00:00
```

Main Sysdb Log Rotator

Four parameters for configuring the main graph datastore retention policy.

```
[main_sysdb_log_rotator]

# AOS has builtin main sysdb persistence file rotation functionality. Default
# value is 1 which means sysdb retention policy is enabled. You can disable it
# by setting it to 0 and you also can enable it again by setting it to 1. All
# retention policy parameters will be reloaded by restarting AOS service, or
# sending SIGHUP signal to SysdbResourceManager agent via "sudo kill -s 1
# $(pgrep -f SysdbResourceManager)"
enable_main_sysdb_rotate = 1
# Maximum number of backup copies of valid main sysdb persistence file groups
# in /var/lib/aos/db. AOS will remove all the older groups. Default value is 5,
# which means AOS will keep the latest 5 groups. Min value is 3. It should be
# specified as a positive number or empty. Leaving it empty means no groups
# number limitation. It will be set to default value if it is configured in
# invalid format. It will be set to minimum value if it is configured to a
```

```
# smaller value.
max_kept_backups = 5
# Maximum total size of valid main sysdb persistence file groups in
# /var/lib/aos/db. Default value is empty, which means no size limitation. It
# should be specified as empty or a positive number ending with k/m/g (case
# insensitive) or no suffix. Otherwise, it will be set to default value. AOS
# will keep at least 3 valid groups no matter how <max_total_files_size> being
# configured.
max_total_files_size =
# Interval, specified as <hh:mm:ss>, at which main sysdb persistence files are
# checked for rotation. Default value is 1:00:00. It will be set to default
# value is it is configured in invalid format. Min value is 00:01:00. It will
# be set to min value if it is configured to a smaller value. AOS also update
# all the retention policy parameters per <check_interval> when it is enabled.
check_interval = 1:00:00
```

`enable_main_sysdb_rotate = 1` enables and disables the policy.

- Set to **1** to enable the retention policy (default). If you enable the policy after it has been disabled, you must restart the Apstra server for it to be enabled again.
- Set to **0** to disable the retention policy and keep all backups. AOS VM file disk utilization issues may occur. The policy will be disabled during the next retention check (`check_interval`). There is no need to restart the Apstra server unless you want to disable the policy immediately.

`max_kept_backups = 5` maximum number of backups to store in `/var/lib/aos/db`.

- Leave default of **5** to keep the latest five backups.
- Set to an empty string to keep an unlimited number of backups.
- Setting to an invalid number results in the default value of **5**.
- Setting to a number smaller than **3** (the minimum) results in the minimum value of **3**.

`max_total_files_size =` maximum file group size to store in `/var/lib/aos/db`

- Leave default of an empty string for no size limitation.
- Set to a number ending in k, m, or g (case-sensitve) or without a suffix.

The effect of `max_kept_backups` and `max_total_files_size` is cumulative. For security, Apstra keeps a minimum of three groups of valid Main Graph Datastore persistence files.

`check_interval = 1:00:00` time between retention checks and parameter updates (if file has been updated) (format: `<hh:mm:ss>`).

- Leave default of **1:00:00** to check every hour.
- Setting to an invalid number results in the default value of **1:00:00**.
- Setting to a number smaller than **00:01:00** (the minimum) results in the minimum value of **1:00:00**.

Anomaly Sysdb Log Rotator

```
[anomaly_sysdb_log_rotator]

# AOS has builtin anomaly sysdb persistence file rotation functionality.
# Default value is 1 which means sysdb retention policy is enabled. You can
# disable it by setting it to 0 and you also can enable it again by setting it
# to 1. All retention policy parameters will be reloaded by restarting AOS
# service, or sending SIGHUP signal to SysdbResourceManager agent via "sudo
# kill -s 1 $(pgrep -f SysdbResourceManager)"
enable_anomaly_sysdb_rotate = 1
# Maximum number of backup copies of valid anomaly sysdb persistence file
# groups in /var/lib/aos/db. AOS will remove all the older groups. Default
# value is 5, which means AOS will keep the latest 5 groups. Min value is 3. It
# should be specified as a positive number or empty. Leaving it empty means no
# groups number limitation. It will be set to default value if it is configured
# in invalid format. It will be set to minimum value if it is configured to a
# smaller value.
max_kept_backups = 5
# Maximum total size of valid anomaly sysdb persistence file groups in
# /var/lib/aos/db. Default value is empty, which means no size limitation. It
# should be specified as empty or a positive number ending with k/m/g (case
# insensitive) or no suffix. Otherwise, it will be set to default value. AOS
# will keep at least 3 valid groups no matter how <max_total_files_size> being
# configured.
max_total_files_size =
# Interval, specified as <hh:mm:ss>, at which anomaly sysdb persistence files
# are checked for rotation. Default value is 1:00:00. It will be set to default
# value is it is configured in invalid format. Min value is 00:01:00. It will
# be set to min value if it is configured to a smaller value. AOS also update
# all the retention policy parameters per <check_interval> when it is enabled.
check_interval = 1:00:00
```

Device Image Management

```
[device_image_management]

# Enable version compatibility check. By default version compatibility check is
# enabled. A device will not connect to AOS if its version of AOS device agent
# is not compatible with AOS controller
enable_version_check = 1
# Enable AOS device agent image auto upgrade. By default auto image upgrade is
# disabled. With this option enabled a device can download an image from the
# controller and upgrade itself if needed.
enable_auto_upgrade = 0
# A device will retry in specified timeout (in seconds) if it fails version
# compatibility check or to download/install new image.
retry_timeout = 600
```

Authentication

```
[authentication]

# Enable authentication/authorization check. By default
# authentication/authorization is enabled. You can disable it by setting enable
# to 0
enable = 1
# Set token expiration time (in seconds). By default token will be expired
# after 24 hours (86400 seconds).
token_expiration = 86400
# Enable ratelimiting. This mechanism protects against password bruteforce. By
# default ratelimiting is enabled. You can disable it by setting
# enable_ratelimit to 0
enable_ratelimit = 1
```

Device Config Management

```
[device_config_management]

# Setting to push quarantine config to unacknowledged devices. By default it is
# disabled as it causes traffic disruptions.Set the value to 1 to enable
# pushing quarantine config, which shuts down all interfaces on the device.
enable_push_quarantine_config = 0
```

Telemetry Init

```
[telemetry_init]

# Number of initial BGP telemetry update rounds before anomaly detection is
# started.
bgp = 4
# Number of initial interface telemetry update rounds before anomaly detection
# is started.
interface = 4
# Number of initial LAG telemetry update rounds before anomaly detection is
# started.
lag = 4
# Number of initial LLDP telemetry update rounds before anomaly detection is
# started.
lldp = 4
# Number of initial route telemetry update rounds before anomaly detection is
# started.
route = 4
# Number of initial MLAG telemetry update rounds before anomaly detection is
# started.
mlag = 4
```

Telemetry Global Config

```
[telemetry_global_config]

# Python multithreading enable/disable knob for telemetry  collection
multithreading_config = 1
# Execution timeout for extensible telemetry collectors
command_timeout = 120
```

Task API

```
[task_api]

# Default maximum time in seconds a task can stay in its current state.
default_timeout = 600.0
# Time in seconds a blueprint.create task can stay in its current state.Format:
# "timeout_<task_type>"
timeout_blueprint.create = 360.0
# Time in seconds a blueprint.deploy task can stay in its current state.Format:
# "timeout_<task_type>"
timeout_blueprint.deploy = 300.0
# Time in seconds blueprint.facade.* tasks can stay in their current state.
# Specific facade task overrides prevail over this one.Format:
# "timeout_<task_type>"
timeout_blueprint.facade = 600.0
# Maximum number of tasks, which allowed in the queue. When number of tasks
# becomes higher this value, task rotation will be started.
max_tasks_in_queue = 100
# Maximum number of Bytes in data field which does not require compression. If
# data size is greater than threshold data will be compressed before storing it
# in sysdb.
max_uncompressed_data_size = 1000
```

Statistics

```
[statistics]

# Enable or disable full validation for pod statistics. Disable if Racks and/or
# Pods tabs load times are excessive
pod_full_validation = enabled
```

Enterprise

```
[enterprise]

# Enable or disable Enterprise related features
enable = 0
```

Syslog

```
[syslog]

# Interval, specified as <hh:mm:ss>, at which collector will recollect hostname
hostname_check_interval = 00:00:10
```

Builtin Telemetry Disable

```
[builtin_telemetry_disable]

# Disable telemetry service lldp for the specified set of system IDs. System
# IDs can be provided as a comma seperated list(eg: a, b, c, d). In order to
# disable the service for all devices, specify the value "all".
lldp_disable_devices =
# Disable telemetry service arp for the specified set of system IDs. System IDs
```

```
# can be provided as a comma seperated list(eg: a, b, c, d). In order to
# disable the service for all devices, specify the value "all".
arp_disable_devices =
# Disable telemetry service hostname for the specified set of system IDs.
# System IDs can be provided as a comma seperated list(eg: a, b, c, d). In
# order to disable the service for all devices, specify the value "all".
hostname_disable_devices =
# Disable telemetry service mac for the specified set of system IDs. System IDs
# can be provided as a comma seperated list(eg: a, b, c, d). In order to
# disable the service for all devices, specify the value "all".
mac_disable_devices =
# Disable telemetry service xcvr for the specified set of system IDs. System
# IDs can be provided as a comma seperated list(eg: a, b, c, d). In order to
# disable the service for all devices, specify the value "all".
xcvr_disable_devices =
# Disable telemetry service interface for the specified set of system IDs.
# System IDs can be provided as a comma seperated list(eg: a, b, c, d). In
# order to disable the service for all devices, specify the value "all".
interface_disable_devices =
# Disable telemetry service interface_counters for the specified set of system
# IDs. System IDs can be provided as a comma seperated list(eg: a, b, c, d). In
# order to disable the service for all devices, specify the value "all".
interface_counters_disable_devices =
# Disable telemetry service bgp for the specified set of system IDs. System IDs
# can be provided as a comma seperated list(eg: a, b, c, d). In order to
# disable the service for all devices, specify the value "all".
bgp_disable_devices =
# Disable telemetry service mlag for the specified set of system IDs. System
# IDs can be provided as a comma seperated list(eg: a, b, c, d). In order to
# disable the service for all devices, specify the value "all".
mlag_disable_devices =
# Disable telemetry service route for the specified set of system IDs. System
# IDs can be provided as a comma seperated list(eg: a, b, c, d). In order to
# disable the service for all devices, specify the value "all".
route_disable_devices =
# Disable telemetry service lag for the specified set of system IDs. System IDs
# can be provided as a comma seperated list(eg: a, b, c, d). In order to
# disable the service for all devices, specify the value "all".
lag_disable_devices =
```

Agent Management

```
[agent_management]

# Override the default heartbeat timeout for agents spawned dynamically by
# AgentManager. The value must be a non-negative number. The unit is seconds.
# The value 0 is used to turn off heartbeat-based agent timeouts and restarts.
# The minimum non-0 value allowed is 60. If not provided, then the default
# timeout value (600 seconds) is used.
heartbeat_period =
```

Show Tech

```
[show_tech]

# Minimum free space in the file system for /var/lib/aos/show_tech needed to
# initiate controller show tech collection via the Apstra API (in MBytes,
# default: 4096, min: 4096)
min_free_disk_space = 4096
# The directory /var/lib/aos/show_tech must be smaller than this size to
# initiate controller show tech collection via the Apstra API (in MBytes,
# default: 10240, min: 4096)
max_directory_size = 10240
# Maximum controller show tech collection duration before job times out (in
# seconds, default: 1200, min: 1200)
controller_timeout = 1200.0
```

System Operation Filesystem Thresholds

```
[system_operation_filesystem_thresholds]

# Default operation thresholds for filesystem utilization, used unless an
# option for a specific filesystem is specified in the section. Two thresholds
# are specified - warning and critical. When resource utilization passes each
# threshold, an operation anomaly is raised at the corresponding level. When a
```

```
# critical threshold is crossed the APIs are automatically transitioned into
# read-only mode. Numbers here are utilization levels, between 0.0 and 1.0.
# Note: Both 0.0 and 1.0 utilization levels are not allowed.
default = warning:0.8 critical:0.9
```

System Operation Memory Thresholds

```
[system_operation_memory_thresholds]

# Operation thresholds for memory utilization of the controller VM. Two
# thresholds are specified - warning and critical. When resource utilization
# passes each threshold, an operation anomaly is raised at the corresponding
# level. When a critical threshold is crossed the APIs are automatically
# transitioned into read-only mode. Numbers here are utilization levels,
# between 0.0 and 1.0. Note: Both 0.0 and 1.0 utilization levels are not
# allowed.
default = warning:0.8 critical:0.9
```

Graph

IN THIS SECTION

Graph Overview

Apstra uses the Graph model to represent a single source of truth regarding infrastructure, policies, constraints etc. This Graph model is subject to constant change and we can query it for various reasons. It is represented as a graph. All information about the network is modeled as nodes and relationships between them.

Every object in a graph has a unique ID. Nodes have a type (a string) and a set of additional properties based on a particular type. For example, all switches in our system are represented by nodes of type system and can have a property role which determines which role in the network it is assigned (spine/leaf/server). Physical and logical switch ports are represented by an interface node, which also has a property called if_type.

Relationships between different nodes are represented as graph edges which we call relationships. Relationships are directed, meaning each relationship has a source node and a target node. Relationships also have a type which determines which additional properties particular relationship can have. E.g. system nodes have relationships of type hosted_interfaces towards interface nodes.

A set of possible node and relationship types is determined by a graph schema. The schema defines which properties nodes and relationships of particular type can have along with types of those properties (string/integer/boolean/etc) and constraints. We use and maintain an open source schema library, Lollipop, that allows flexible customization of value types.

Going back to the graph representing a single source of truth, one of the most challenging aspects was how to reason about it in the presence of change, coming from both the operator and the managed system. In order to support this we developed what we call Live Query mechanism which has three essential components:

- Query Specification
- Change Notification
- Notification Processing

Having modeled our domain model as a graph, you can run searches on the graph specified by graph queries to find particular patterns (subgraphs) in a graph. The language to express the query is conceptually based on Gremlin, an open source graph traversal language. We also have parsers for queries expressed in another language - Cypher, which is a query language used by popular graph database neo4j.

Query Specification

You start with a node() and then keep chaining method calls, alternating between matching relationships and nodes:

```
node('system', name='system').out().node('interface', name='interface').out().node('link',
name='link')
```

The query above translated in english reads something like: starting from a node of type system, traverse any outgoing relationship that reaches node of type interface, and from that node traverse all outgoing relationship that lead to node of type `link.

At any point you can add extra constraints:

```
node('system', role='spine', name='system').out().node('interface', if_type='ip',
name='interface')
```

Notice role=`spine` argument, it will select only system nodes that have role property set to spine.

Same with if_type property for interface nodes.

```
node('system', role=is_in(['spine', 'leaf']), name='system')
.out()
.node('interface', if_type=ne('ip'), name='interface')
```

That query will select all system nodes that have role either spine or leaf and interface nodes that have if_type anything but ip (ne means not equal).

You can also add cross-object conditions which can be arbitrary Python functions:

```
node('system', name='system')
.out().node('interface', name='if1')
.out().node('link')
.in_().node('interface', name='if2')
.in_().node('system', name='remote_system')
.where(lambda if1, if2: if1.if_type != if2.if_type)
```

Name objects to refer to them and use those names as argument names for your constraint function (of course you can override that but it makes a convenient default behavior). So, in example above it will take two interface nodes named if1 and if2, pass them into given where function and filter out those

paths, for which function returns False. Don't worry about where you place your constraint: it will be applied during search as soon as all objects referenced by constraint are available.

Now, you have a single path, you can use it to do searches. However, sometimes you might want to have a query more complex than a single path. To support that, query DSL allows you to define multiple paths in the same query, separated by comma(s):

```
match(
    node('a').out().node('b', name='b').out().node('c'),
    node(name='b').out().node('d'),
)
```

This `match()` function creates a grouping of paths. All objects that share the same name in different paths will actually be referring to the same object. Also, `match()` allows adding more constraints on objects with `where()`. You can do a distinct search on particular objects and it will ensure that each combination of values is seen only once in results:

```
match(
    node('a', name='a').out().node('b').out().node('c', name='c')
).distinct(['a', 'c'])
```

This matches a chain of a -> b -> c nodes. If two nodes a and c are connected through more than one node of type b, the result will still contain only one (a, c) pair.

There is another convenient pattern to use when writing queries: you separate your structure from your criteria:

```
match(
    node('a', name='a').out().node('b').out().node('c', name='c'),
    node('a', foo='bar'),
    node('c', bar=123),
)
```

Query engine will optimize that query into:

```
match(
    node('a', name='a', foo='bar')
    .out().node('b')
    .out().node('c', name='c', bar=123)
)
```

No cartesian product, no unnecessary steps.

Change Notification

Ok, now you have a graph query defined. What does a notification result look like? Each result will be a dictionary mapping a name that you have defined for a query object to object found. E.g. for following query

```
node('a', name='a').out().node('b').out().node('c', name='c')
```

results will look like `{'a': <node type='a'>, 'c': <node type='c'>}`. Notice, only named objects are present (there is no `<node type='b'>` in results, although that node is present in query because it does not have a name).

You register a query to be monitored and a callback to execute if something will change. Later, if someone will modify the graph being monitored, it will detect that new graph updates caused new query results to appear, or old results to disappear or update. The response executes the callback that is associated with the query. The callback receives the whole path from the query as a response, and a specific action (added/updated/removed) to execute.

Notification Processing

When the result is passed to the processing (callback) function, from there you can specify reasoning logic. This could really be anything, from generating logs, errors, to rendering configurations, or running semantic validations. You could also modify the graph itself, using graph APIs and some other piece of logic may react to changes you made. This way, you can enforce the graph as a single source of truth while it also serves as a logical communication channel between pieces of your application logic. The Graph API consists of three parts:

Graph management - methods to add/update/remove stuff in a graph. `add_node()`, `set_node()`, `del_node()`, `get_node()add_relationship()`, `set_relationship()`, `del_relationship()`, `get_relationship()`, `commit()` Query `get_nodes()get_relationships()` Observable interface `add_observer()`,`remove_observer()`

Graph management APIs are self-explanatory. `add_node()` creates new node, `set_node()` updates properties of existing node, and `del_node()` deletes a node.

`commit()` is used to signal that all updates to the graph are complete and they can be propagated to all listeners.

Relationships have similar API.

The observable interface allows you to add/remove observers - objects that implement notification a callback interface. Notification callback consists of three methods:

- `on_node()` - called when any node/relationship is added, removed or updated
- `on_relationship()` - called when any node/relationship is added, removed or updated
- `on_graph()` - called when the graph is committed

The Query API is the heart of our graph API and is what powers all searching. Both `get_nodes()` and `get_relationships()` allow you to search for corresponding objects in a graph. Arguments to those functions are constraints on searched objects.

E.g. `get_nodes()` returns you all nodes in a graph, `get_nodes(type='system')` returns you all system nodes, `get_nodes(type='system', role='spine')` allows you to constrain returned nodes to those having particular property values. Values for each argument could be either a plain value or a special property matcher object. If the value is a plain value, the corresponding result object should have its property equal to the given plain value. Property matchers allow you to express a more complex criterias, e.g. not equal, less than, one of given values and so on:

NOTE: The example below is for directly using Graph python. For demonstration purposes, you can replace graph.get_nodes with node in the Graph explorer. This specific example will not work on the Apstra GUI.

```
graph.get_nodes(
    type='system',
    role=is_in(['spine', 'leaf']),
     system_id=not_none(),
)
```

In your graph schema you can define custom indexes for particular node/relationship types and the methods `get_nodes()` and `get_relationships()` pick the best index for each particular combination of constraints passed to minimize search time.

Results of `get_nodes()`/`get_relationships()` are special iterator objects. You can iterate over them and they will yield all found graph objects. You can also use APIs that those iterators provide to navigate those result sets. E.g. `get_nodes()` returns you a NodeIterator object which has methods `out()` and `in_()`. You can use those to get an iterator over all outgoing or incoming relationship from each node in the original result set. Then, you can use those to get nodes on the other end of those relationships and continue

from them. You can also pass property constraints to those methods the same way you can do for `get_nodes()` and `get_relationships()`.

```
graph.get_nodes('system', role='spine') \
    .out('interface').node('interface', if_type='loopback')
```

The code in the example above finds all nodes with type system and role spine and then finds all their loopback interfaces.

Putting It All Together

Thequery below is an example of an internal rule that Apstra can use to derive telemetry expectations -- for example, link and interface status. The @rule will insert a callback to process_spine_leaf_link, in which case we write to telemetry expectations.

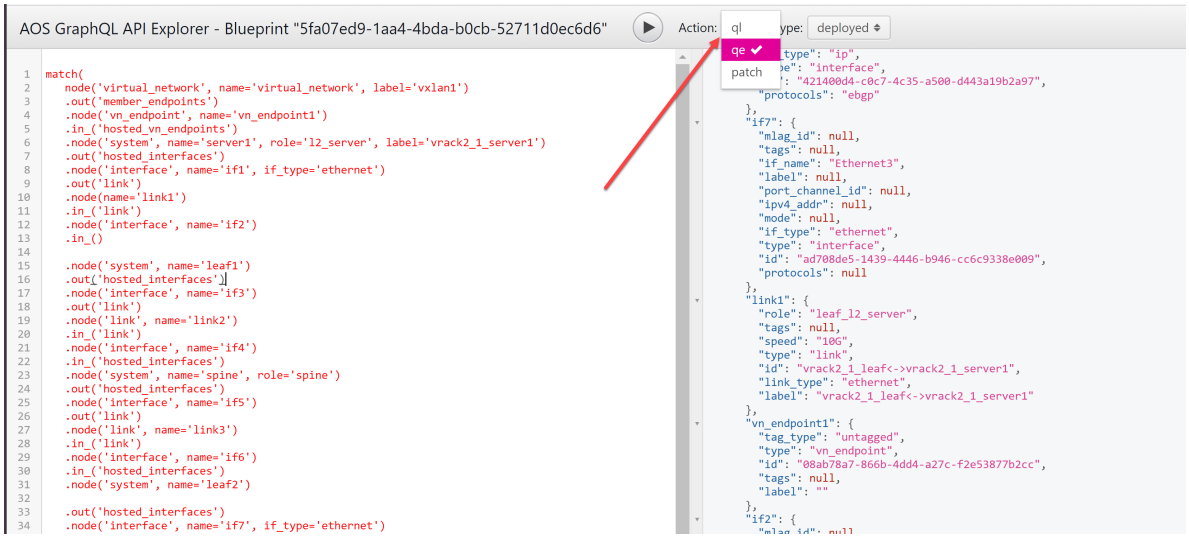
```
@rule(match(
    node('system', name='spine_device', role='spine')
    .out('hosted_interfaces')
    .node('interface', name='spine_if')
    .out('link')
    .node('link', name='link')
    .in_('link')
    .node('interface', name='leaf_if')
    .in_('hosted_interfaces')
    .node('system', name='leaf_device', role='leaf')
))
def process_spine_leaf_link(self, path, action):
    """
    Process link between spine and leaf

    """
    spine = path['spine_device']
    leaf = path['leaf_device']
    if action in ['added', 'updated']:
        # do something with added/updated link
        pass
    else:
```

```
        # do something about removed link
        pass
```

Convenience Functions

To avoid creating complex `where()` clauses when building a graph query, use convenience functions, available from the Apstra GUI.

1. From the blueprint navigate to the **Staged** view or **Active** view, then click the **GraphQL API Explorer** button (top-right **>_**). The graph explorer opens in a new tab.
2. Type a graph query on the left. See function descriptions below.
3. From the **Action** drop-down list, select **qe**.
4. Click the **Execute Query** button (looks like a play button) to see results.



Functions

The Query Engine describes a number of helpful functions:

match(*path_queries)

This function returns a `QueryBuilder` object containing each result of a matched query. This is generally a useful shortcut for grouping multiple match queries together.

These two queries are not a 'path' together (no intended relationship). Notice the comma to separate out arguments. This query will return all of the leaf devices and spine devices together.

```
match(
  node('system', name='leaf', role='leaf'),
  node('system', name='spine', role='spine'),
)
```

node(self, type=None, name=None, id=None, **properties)

- **Parameters**
  - **type** (str or None) - Type of node to search for
  - **name** (str or None) - Sets the name of the property matcher in the results
  - **id** (str or None) - Matches a specific node by node ID in the graph
  - **properties** (dict or None) - Any additional keyword arguments or additional property matcher convenience functions to be used
- **Returns** - Query builder object for chaining queries
- **Return type** - QueryBuilder

While both a function, this is an alias for the PathQueryBuilder nodes -- see below.

iterate()

- Returns - generator
- Return type: generator

Iterate gives you a generator function that you can use to iterate on individual path queries as if it were a list. For example:

```
def find_router_facing_systems_and_intfs(graph):
    return q.iterate(graph, q.match(
        q.node('link', role='to_external_router')
        .in_('link')
        .node('interface', name='interface')
        .in_('hosted_interfaces')
        .node('system', name='system')
    ))
```

PathQueryBuilder Nodes

node(self, type=None, name=None, id=None, **properties)

This function describes specific graph node, but is also a shortcut for beginning a path query from a specific node. The result of a `node()` call returns a path query object. When querying a path, you usually want to specify a node `type`: for example `node('system')` would return a system node.

- **Parameters**
  - **type** (str or None) - Type of node to search for
  - **name** (str or None) - Sets the name of the property matcher in the results
  - **id** (str or None) - Matches a specific node by node ID in the graph
  - **properties** (dict or None) - Any additional keyword arguments or additional property matcher convenience functions to be used
- **Returns** - Query builder object for chaining queries
- **Return type** - QueryBuilder

If you want to use the node in your query results, you need to name it --`node('system', name='device')`. Furthermore, if you want to match specific kwarg properties, you can directly specify the match requirements -

`node('system', name='device', role='leaf')`

```
node('system', name='device', role='leaf')
```

out(type=None, id=None, name=None, **properties)

Traverses a relationship in the 'out' direction according to a graph schema. Acceptable parameters are the type of relationship (for example, interfaces), the specific name of a relationship, the id of a relationship, or other property matches that must match exactly given as keyword arguments.

- **Parameters**
  - **type** (str or None) - Type of node relationship to search for
  - **id** (str or None) - Matches a specific relationship by relationship ID in the graph
  - **name** (str or None) - Matches a specific relationship by named relationship

For example:

```
node('system', name='system') \
 .out('hosted_interfaces')
```

in_(type=None, id=None, name=None, **properties)

Traverses a relationship in the 'in' direction. Sets current node to relationship source node. Acceptable parameters are the type of relationship (for example, interfaces), the specific name of a relationship, the id of a relationship, or other property matches that must match exactly given as keyword arguments.

- Parameters
  - **type** (str or None) - Type of node relationship to search for
  - **id** (str or None) - Matches a specific relationship by relationship ID in the graph
  - **name** (str or None) - Matches a specific relationship by named relationship
  - **properties** (dict or None) - Matches relationships by any further kwargs or functions

```
node('interface', name='interface') \
 .in_('hosted_interfaces')
```

where(predicate, names=None)

Allows you to specify a callback function against the graph results as a filter or constraint. The predicate is a callback (usually lambda function) run against the entire query result. `where()` can be used directly on an a path query result.

- Parameters
  - predicate (callback) - Callback function to run against all nodes in graph
  - names (str or None) - If names are given they are passed to callback function for match

```
node('system', name='system') \
 .where(lambda system: system.role in ('leaf', 'spine'))
```

enure_different(*names)

Allows a user to ensure two different named nodes in the graph are not the same. This is helpful for relationships that may be bidirectional and could match on their own source nodes. Consider the query:

- Parameters

- names (tuple or list) - A list of names to ensure return different nodes or relationships from the graph

```
match(node('system', name='system', role='leaf') \
  .out('hosted_interfaces') \
  .node('interface', name='interface', ipv4_addr=not_none()) \
  .out('link') \
  .node('link', name='link') \
  .in_('link') \
  .node('interface', name='remote_interface', ipv4_addr=not_none())) \
  .ensure_different('interface', 'remote_interface')
```

The last line could be functionally equivalent to the `where()` function with a lambda callback function

```
match(node('system', name='system', role='leaf') \
  .out('hosted_interfaces') \
  .node('interface', name='interface', ipv4_addr=not_none()) \
  .out('link') \
  .node('link', name='link') \
  .in_('link') \
  .node('interface', name='remote_interface', ipv4_addr=not_none())) \
  .where(lambda interface, remote_interface: interface != remote_interface)
```

Property matchers

Property matches can be run on graph query objects directly - usually used within a `node()` function. Property matches allow for a few functions.

eq(value)

Ensures the property value of the node matches exactly the results of the `eq(value)` function.

- Parameters
  - value - Property to match for equality

```
node('system', name='system', role=eq('leaf'))
```

Which is similar to simply setting a value as a kwarg on a node object:

```
node('system', name='system', role='leaf')
```

```
node('system', name='system').where(lambda system: system.role == 'leaf')
```

Returns:


```
{
  "count": 4,
  "items": [
    {
      "system": {
        "tags": null,
        "hostname": "l2-virtual-mlag-2-leaf1",
        "label": "l2_virtual_mlag_2_leaf1",
        "system_id": "000C29EE8EBE",
        "system_type": "switch",
        "deploy_mode": "deploy",
        "position": null,
        "role": "leaf",
        "type": "system",
        "id": "391598de-c2c7-4cd7-acdd-7611cb097b5e"
      }
    },
    {
      "system": {
        "tags": null,
        "hostname": "l2-virtual-mlag-2-leaf2",
        "label": "l2_virtual_mlag_2_leaf2",
        "system_id": "000C29D62A69",
        "system_type": "switch",
        "deploy_mode": "deploy",
        "position": null,
        "role": "leaf",
        "type": "system",
        "id": "7f286634-fbd1-43b3-9aed-159f1e0e6abb"
      }
    },
    {
```

```
      "system": {
        "tags": null,
        "hostname": "l2-virtual-mlag-1-leaf2",
        "label": "l2_virtual_mlag_1_leaf2",
        "system_id": "000C29CFDEAF",
        "system_type": "switch",
        "deploy_mode": "deploy",
        "position": null,
        "role": "leaf",
        "type": "system",
        "id": "b9ad6921-6ce3-4d05-a5c7-c31d96785045"
      }
    },
    {
      "system": {
        "tags": null,
        "hostname": "l2-virtual-mlag-1-leaf1",
        "label": "l2_virtual_mlag_1_leaf1",
        "system_id": "000C297823FD",
        "system_type": "switch",
        "deploy_mode": "deploy",
        "position": null,
        "role": "leaf",
        "type": "system",
        "id": "71bbd11c-ed0f-4a38-842f-341781c01c24"
      }
    }
  ]
}
```


ne(value)

Not-equals. Ensures the property value of the node does NOT match results of `ne(value)` function

- Parameters
  - value - Value to ensure for inequality condition

```
node('system', name='system', role=ne('spine'))
```

Similar to:

```
node('system', name='system').where(lambda system: system != 'spine')
```

gt(value)

Greater-than. Ensures the property of the node is greater than the results of `gt(value)` function.

- Parameters
  - value - Ensure property function is greater than this value

```
node('vn_instance', name='vlan', vlan_id=gt(200))
```

ge(value)

Greater-than or Equal To. Ensures the property of the node is greater than or equal to results of `ge()`.

- Parameters: value - Ensure property function is greater than or equal to this value

```
node('vn_instance', name='vlan', vlan_id=ge(200))
```

lt(value)

Less-than. Ensures the property of the node is less than the results of `lt(value)`.

- Parameters
  - value - Ensure property function is less than this value

```
node('vn_instance', name='vlan', vlan_id=lt(200))
```

Similar to:

```
node('vn_instance', name='vlan').where(lambda vlan: vlan.vlan_id <= 200)
```

le(value)

Less-than or Equal to. Ensures the property is less than, or equal to the results of `le(value)` function.

- Parameters

- value - Ensures given value is less than or equal to property function

```
node('vn_instance', name='vlan', vlan_id=le(200))
```

Similar to:

```
node('vn_instance', name='vlan').where(lambda vlan: vlan.vlan_id < 200)
```

is_in(value)

Is in (list). Check if the property is in a given list or set containing items `is_in(value)`.

- Parameters
  - value (list) - Ensure given property is in this list

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

Similar to:

```
node('system', name='system').where(lambda system: system.role in ['leaf', 'spine'])
```

not_in(value)

Is not in (list). Check if the property is NOT in a given list or set containing items `not_in(value)`.

- Parameters
  - value (list) - List Value to ensure property matcher is not in

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

Similar to:

```
node('system', name='system').where(lambda system: system.role not in ['leaf', 'spine'])
```

is_none()

A query that expects is_none expects this particular attribute to be specifically `None`.

```
node('interface', name='interface', ipv4_addr=is_none()
```

Similar to:

```
node('interface', name='interface').where(lambda interface: interface.ipv4_addr is None)
```

not_none()

A matcher that expects this attribute to have a value.

```
node('interface', name='interface', ipv4_addr=not_none()
```

Similar to:

```
node('interface', name='interface').where(lambda interface: interface.ipv4_addr is not None)
```

Apstra Graph Datastore

The Apstra graph datastore is an in-memory graph database. The log file size is checked periodically, and when a blueprint change is committed. If the graph datastore reaches 100MB or more, a new graph datastore checkpoint file is generated. The database itself does not remove any graph datastore persistence logs or checkpoint files. Apstra provides clean-up tools for the main graph datastore.

Valid graph datastore persistence file groups contain four files: `log`, `log-valid`, `checkpoint`, and `checkpoint-valid`. Valid files are the effective indicators for log and checkpoint files. The name of each persistence file has three parts: basename, id, and extension.

```
# regex for sysdb persistence files.
# e.g.
#      _Main-0000000059ba612e-00017938-checkpoint-valid
#       \--/ \----------------------/ \--------------/
#     basename           id                extension
```

- **basename** - derived from the main graph datastore partition name.

- **id** - a unix timestamp obtained from gettimeofday. Seconds and microseconds in the timestamp are separated by a "-". A persistence file group can be identified by id. The timestamp can also help to determine the generated time sequence of persistence file groups.
- **extension** - `log`, `log-valid`, `checkpoint`, or `checkpoint-valid`.

Juniper Apstra Tech Previews

IN THIS SECTION

Tech Previews

Tech Previews give you the ability to test functionality and provide feedback during the development process of innovations that are not final production features. The goal of a Tech Preview is for the feature to gain wider exposure and potential full support in a future release. Customers are encouraged to provide feedback and functionality suggestions for a Technology Preview feature before it becomes fully supported.

Tech Previews may not be functionally complete, may have functional alterations in future releases, or may get dropped under changing markets or unexpected conditions, at Juniper's sole discretion. Juniper recommends that you use Tech Preview features in non-production environments only.

Juniper considers feedback to add and improve future iterations of the general availability of the innovations. Your feedback does not assert any intellectual property claim, and Juniper may implement your feedback without violating your or any other party's rights.

These features are "as is" and voluntary use. Juniper Support will attempt to resolve any issues that customers experience when using these features and create bug reports on behalf of support cases. However, Juniper may not provide comprehensive support services to Tech Preview features. Certain features may have reduced or modified security, accessibility, availability, and reliability standards relative to General Availability software. Tech Preview is not supported under existing service agreements, SLAs, or support service.

Here's a list of the Tech Previews:

- ESI LAG

ESI LAG on page 1486

- LAG

  LAG on page 1486

- ACX7100

  "ACX7100" on page 1507

- ACX7100-32C

  "ACX7100-32C" on page 1507

- ACX7100-48L

  "ACX7100-48L" on page 1507

- C93600CD-GX

  "C93600CD-GX" on page 1514

- Interface Policies

  "Interface Policies" on page 388

- SONiC in Collapsed Fabric

  SONic in Collapsed Fabric on page 1482

- SONiC in Connectivity (from Access Layer)

  "SONiC in Connectivity (from Access Layer)" on page 1486

- Exploratory Analytics

  "QBA Exporatory Interface" on page 1232

- Interfaces and Visualization of History of Anomalies

  "Blueprint Anomaly History" on page 620

For additional details, please contact "Juniper Support " on page 1374or your local account team.